Amazon Bedrock

User Guide

Amazon Bedrock: User Guide

Table of Contents

What is Amazon Bedrock?

Amazon Bedrock is a fully managed service that makes base models from Amazon and third-party model providers accessible through an API.

Topics

Access the Amazon Bedrock models

Important
You must request access to a model before you can use it. If you try to use the model (with the API or console) before you have requested access to it, you receive an error message. For more information, see Model access (p. 11).

Features of Amazon Bedrock

With Amazon Bedrock, you can explore the following capabilities:

- **Text playground** – A hands-on text generation application in the AWS Management Console.
- **Image playground** – A hands-on image generation application in the console.
- **Chat playground** – A hands-on conversation generation application in the console.
- **Examples library** – Example use cases to load.
- **Amazon Bedrock API** – Explore with the AWS CLI, or use the API to access the base models.
- **Embeddings** – Use the API to generate embeddings from the Titan Embeddings G1 - Text model.
- **Provisioned throughput** – Purchase throughput to run inference on models at discounted rates.

  Note
  Provisioned throughput is currently available for the following models.

Model name	Model ID for provisioned throughput
Titan Text G1 - Express 8K	amazon.titan-text-express-v1:0:8k
Titan Embeddings G1 - Text	amazon.titan-embed-text-v1:2:8k
Anthropic Claude V2 18K	anthropic.claude-v2:0:18k
Anthropic Claude V2 100K	anthropic.claude-v2:0:100k
Anthropic Claude Instant V1 100K	anthropic.claude-instant-v1:2:100K
Stable Diffusion XL 1.0	stability.stable-diffusion-xl-v0

- **Fine-tuning** – Create a training dataset and fine-tune an Amazon Bedrock model.

Note
To use fine-tuning, you must have access to the Amazon Titan Text G1 - Express model. The Amazon Titan Text G1 - Express model is in limited preview. To request access, contact your AWS account manager.

- **Model invocation logging** – Collect invocation logs, model input data, and model output data for all invocations in your AWS account used in Amazon Bedrock.

  Note
  Model invocation logging is in preview release for Amazon Bedrock and is subject to change.

The following capabilities are in limited preview release. To request access, contact your AWS account manager.

- **Agents for Amazon Bedrock** – Build agents to perform orchestration and carry out tasks for your customers.
- **Knowledge base for Amazon Bedrock** – Draw from data sources to help your agent find information for your customers.

Supported models in Amazon Bedrock

For details about the Amazon Bedrock model providers and their models and model IDs see the **Base models** pages in the Amazon Bedrock console. You can also use the ListFoundationModels (p. 23) API operation to retrieve information about the current list of models.

Amazon Bedrock supports the following models:

- AI21 Labs
  - Jurassic-2 Ultra
  - Jurassic-2 Mid
- Amazon
  - Titan Text G1 - Express
  - Titan Embeddings G1 - Text

    Note
    The Amazon Titan Text G1 - Express model is in limited preview release. Access will be granted on an ongoing basis.
- Anthropic
  - Claude v1.x
  - Claude v2.x
  - Claude Instant v1.x
- Cohere
  - Command
- Stability.ai
  - Stable Diffusion XL 0.x
  - Stable Diffusion XL 1.x

    Note
    The Stability.ai models are in limited preview release. To request access, contact your AWS account manager. Stable Diffusion XL 1.x is only available with provisioned throughput. For more information, see Provisioned throughput (p. 55).

For additional information about these models, see the following links:

- Anthropic documentation.
- AI21 Studio documentation.
- Cohere documentation.
- Stability.ai documentation.

Supported Regions

Amazon Bedrock is available in the following AWS Regions:

- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)

Amazon Bedrock pricing

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including Amazon Bedrock. However, you are charged only for the services that you use.

To see your bill, go to the Billing and Cost Management Dashboard in the AWS Billing and Cost Management console. To learn more about AWS account billing, see the *AWS Billing User Guide*. If you have questions concerning AWS billing and AWS accounts, contact AWS Support.

With Amazon Bedrock, you pay to run inference on any of the third-party foundation models. Pricing is based on the volume of input tokens and output tokens, and on whether you have purchased provisioned throughput for the model. For more information, see the Model providers page in the Amazon Bedrock console. For each model, pricing is listed following the model version. For more information about purchasing provisioned throughput, see Provisioned throughput (p. 55).

For more information, see Amazon Bedrock Pricing.

Set up Amazon Bedrock

Before you use Amazon Bedrock for the first time, complete the following tasks.

Setup tasks

Add model access

Important
You must request access to a model before you can use it. If you try to use the model (with the API or console) before you have requested access to it, you receive an error message. For more information, see Model access (p. 11).

Console access

To access the Amazon Bedrock console and playground:

1. Sign in to your AWS account.
2. Navigate to: Amazon Bedrock console

   You can also access the Amazon Bedrock console in US East (N. Virginia), US West (Oregon) and Asia Pacific (Singapore).

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to an administrative user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center.

  For instructions, see Getting started in the *AWS IAM Identity Center User Guide*.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Set up the AWS CLI

You don't need the AWS Command Line Interface (AWS CLI) to use Amazon Bedrock. If you prefer, you can skip this step and set up the AWS CLI later.

To install and configure the AWS CLI

1. Install the AWS CLI. For instructions, see Installing or updating the latest version of the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.
2. Configure the AWS CLI. For instructions, see Configuring the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Settings

You can manage your account level settings for Amazon Bedrock in the **Settings** page. The settings include data logs, data permissions, and model access.

To access settings, go to the bottom of the left-side navigation pane in Amazon Bedrock, and select **Settings**.

Topics

Model invocation logging

Model invocation logging can be used to collect invocation logs, model input data, and model output data for all invocations in your AWS account used in Amazon Bedrock. By default, logging is disabled.

With invocation logging, you can collect the full request data, response data, and metadata associated with all calls performed in your account. Logging can be configured to provide the destination resources where the log data will be published. Supported destinations include Amazon CloudWatch Logs and Amazon Simple Storage Service (Amazon S3). Only destinations from the same account and region are supported.

Before you can enable invocation logging, you need to set up an Amazon S3 or CloudWatch Logs destination. You can enable invocation logging through either the console or the API.

Topics

Set up an Amazon S3 destination

You can set up an S3 destination for logging in Amazon Bedrock with these steps:

1. Create an S3 bucket where the logs will be delivered.
2. Add a bucket policy to it like the one below (Replace values for *`accountId`*, *`region`*, *`bucketName`*, and optionally *`prefix`*):

   Note
   A bucket policy is automatically attached to the bucket on your behalf when you configure logging with the permissions `S3:GetBucketPolicy` and `S3:PutBucketPolicy`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AmazonBedrockLogsWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "bedrock.amazonaws.com"
      },
```

```
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucketName/prefix/AWSLogs/accountId/BedrockModelInvocationLogs/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "accountId"
        },
        "ArnLike": {
           "aws:SourceArn": "arn:aws:bedrock:region:accountId:*"
        }
      }
    }
  ]
}
```

3. (Optional) If configuring SSE-KMS on the bucket, add the below policy on the KMS key:

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "bedrock.amazonaws.com"
    },
    "Action": "kms:GenerateDataKey",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "accountId"
        },
        "ArnLike": {
           "aws:SourceArn": "arn:aws:bedrock:region:accountId:*"
        }
    }
}
```

For more information on S3 SSE-KMS configurations, see Specifying KMS Encryption.

Note
The bucket ACL must be disabled in order for the bucket policy to take effect. For more information, see Disabling ACLs for all new buckets and enforcing Object Ownership.

Set up CloudWatch Logs destination

You can set up a Amazon CloudWatch Logs destination for logging in Amazon Bedrock with the following steps:

1. Create a CloudWatch log group where the logs will be published.
2. Create an IAM role with the following permissions for CloudWatch Logs.

   Trusted entity:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "bedrock.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "accountId"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:bedrock:region:accountId:*"
        }
      }
    }
  ]
}
```

Role policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:region:accountId:log-group:logGroupName:log-
stream:aws/bedrock/modelinvocations"
         }
    ]
}
```

For more information on setting up SSE for CloudWatch Logs, see Encrypt log data in CloudWatch Logs using AWS Key Management Service.

Using the console

To enable model invocation logging, drag the slider button next to the **Logging** toggle switch in the **Settings** page. Additional configuration settings for logging will appear on the panel.

Choose which data requests and responses you want to publish to the logs. You can choose any combination of the following output options:

- Text
- Image
- Embedding

Choose where to publish the logs:

- Amazon S3 only
- CloudWatch Logs only
- Both Amazon S3 and CloudWatch Logs

Amazon S3 and CloudWatch Logs destinations are supported for invocation logs, and small input and output data. For large input and output data or binary image outputs, only Amazon S3 is supported. The following details summarize how the data will be represented in the target location.

- **S3 destination** — Gzipped JSON files, each containing a batch of invocation log records, are delivered to the specified S3 bucket. Similar to a CloudWatch Logs event, each record will contain the invocation metadata, and input and output JSON bodies of up to 100 KB in size. Binary data or JSON bodies larger than 100 KB will be uploaded as individual objects in the specified Amazon S3 bucket under the data prefix. The data can be queried using Amazon S3 Select and Amazon Athena, and can be catalogued for ETL using AWS Glue. The data can be loaded into OpenSearch service, or be processed by any Amazon EventBridge targets.
- **CloudWatch Logs destination** — JSON invocation log events are delivered to a specified log group in CloudWatch Logs. The log event contains the invocation metadata, and input and output JSON bodies of up to 100 KB in size. If an Amazon S3 location for large data delivery is provided, binary data or JSON bodies larger than 100 KB will be uploaded to the Amazon S3 bucket under the data prefix instead. data can be queried using CloudWatch Logs Insights, and can be further streamed to various services in real-time using CloudWatch Logs.

Using APIs with invocation logging

Model invocation logging can be configured using the following APIs:

- `PutModelInvocationLoggingConfiguration`
- `GetModelInvocationLoggingConfiguration`
- `DeleteModelInvocationLoggingConfiguration`

For more information on how to use APIs with invocation logging, see the Bedrock API Guide.

Model access

Amazon Bedrock users need to request access to models before they are available for use. If you want to add additional models for text, chat, and image generation, you need to request access to models in Amazon Bedrock. To request access to additional models, select the **Model access** link in the left side navigation panel in the Amazon Bedrock console.

Topics

Manage model access

The account does not have access to models by default. Admin users with IAM access permissions can add access to specific models using the model access page. After the admin adds access to models, those models are available for all users of the account.

Charges are accrued when the models are used in Amazon Bedrock. You can review the **End User License Agreement** (EULA) by selecting the link for each model.

To add or remove model access, select **Manage model access**.

Edit model access

You can request access to models to use them in Amazon Bedrock. This page lists base models of LLMs to text, image, and embedding models. You can review your status to see which models you have access to. The modality lists whether the model is used for text, embedding, or images.

Add model access

You can add access to a model in Amazon Bedrock with the following steps:

1. Open the Amazon Bedrock console at Amazon Bedrock console
2. Go to the **Model access** link in the left side navigation panel in Amazon Bedrock, or go to the **Edit model access** page.
3. Select the check box next to the model you want to add access to. For Anthropic models, you must also request access when you click the **Request access** button. Models are not available as a default setting in Amazon Bedrock.
4. Select **Confirm** to add access to any third party models through Amazon Marketplace. Note: Your use of Amazon Bedrock and it's models is subject to the seller's pricing terms, EULA and the Amazon Bedrock service terms.
5. Select the **Save Changes** button in the lower right corner of the page. It may take several minutes to save changes to the **Model access** page.
6. Models will show as **Available** on the **Model access** page under **Access** status, if access is granted.

Some users may not have IAM permissions to add access to models. A banner message will appear if you try to add access to models and you are a non-admin user on your account. You will need to contact your account administrator to request that they add access to the models before you can use them in Amazon Bedrock.

When you have access to the model, you can select it using the **View model access** button or the **Edit model access** page. Accounts do not have access to Amazon models by default.

Remove model access

When you are using Amazon Bedrock, you may decide to use only certain models for your work, and to remove access to models you are no longer using. You can remove access to a model in Amazon Bedrock with the following steps:

1. Open the Amazon Bedrock console at https://us-east-1.console.aws.amazon.com/bedrock/home?region=us-east-1#/.
2. Go to the **Model access** link in the left side navigation panel in Amazon Bedrock, or go to the **Edit model access** page.
3. Deselect the check box next to the model you want to remove access.
4. Select the **Save changes** button at the bottom right corner of the page.
5. You will be prompted to confirm you want to remove access to models. You cannot remove access to models if there are resources using those models.
6. Users must remove the associations to the model listed in the information window in order to remove access. You can select the link next to the resource name to go to the location, and remove access to the model for that resource. Repeat this step for each item listed as a dependent resource.
7. Once you have removed access all the resources listed, select **Okay** to complete the remove access step.
8. Once completed, you will see a banner message that confirms the action was completed successfully.

Use the Amazon Bedrock console

The Amazon Bedrock console provides the following capabilities and features:

- From the **Base models** page under **Foundation models**, you can view the available models and group them by various attributes. You can also filter the model view, search for models, and view information about the model providers.
- After you select a model, you can open the model to experiment with it in the **Text playground**, **Image playground**, or **Chat playground**.
- Amazon Bedrock provides example prompts for each of the supported models. The **Examples** page displays up to 20 examples for each model provider. You can filter the list of examples using one or more attributes.

> **Important**
> You must request access to a model before you can use it. If you try to use the model (with the API or console) before you have requested access to it, you receive an error message. For more information, see Model access (p. 11).

Topics

Base models

Amazon Bedrock supports base models from Amazon and third-party model providers.

From the Amazon Bedrock console, you can view the available models and group them by various attributes. You can also filter the model view, search for models, and view information about the model providers.

After you select a model, you can choose **Open in playground** to experiment with the model.

Topics

View and group models

You can list models and view model details using the Amazon Bedrock console.

1. From the Amazon Bedrock console, choose **Base models** under **Foundation models**.
2. You can view the models in a list view or a card view and group them by model family, by model provider, or by model modality (text, image, or embedding).

Filter and search models

You can filter models by model name, provider, by modality, or by model attributes.

You can set multiple filters. You can also start by entering text in the text box. The console displays all the possible filter values that match the text.

1. From the Amazon Bedrock console, choose **Base Models** under **Foundation models**.
2. To set a filter, choose the **Find resource** text box to display the filter properties.
3. Select the desired property, the operator for the filter, then enter the value to filter by.

View model provider details

To view model provider details using the Amazon Bedrock console, choose **Model providers** under **Foundation models**.

As an alternative, from the **Base models** page, select **View provider details** under a provider name.

Choose one of the tabs near the top of the **Model providers** page to view details about that model provider. The page displays the following information about the provider:

- **Provider overview** – An overview description about the provider.
- **Models** – A tabbed list of available models. When you choose one of the models, you can open the model in the playground.
- **Content limitations** – Details about the content policy for the provider.

Request access to base models

Important
You must request access to a model before you can use it. If you try to use the model (with the API or console) before you have requested access to it, you receive an error message. For more information, see Model access (p. 11).

Text playground

From the Amazon Bedrock console, choose **Playgrounds** and then **Text** to view the text playground.

You can also navigate directly to the playground when you choose a model from a model details page or the examples page.

Topics

- Adjust the model parameters (p. 16)
- Select a different model (p. 16)

Submit a prompt to a text model

The following procedure shows how to use the text playground to submit a prompt to a text model.

1. Open the Amazon Bedrock console.
2. From the left-side menu, choose **Text** under **Playgrounds**.
3. Set up the text playground.

   a. You can toggle on the following options in the top-right corner:

      - **Streaming** – Generate text in real-time. This option is only available for models that support streaming.

   b. From the dropdown menu above the text panel, select a model provider. After you select a provider, select a model from the model dropdown menu. The dropdown menu lists the available models from this provider. You can also select a model that you have customized (for more information, see Use a fine-tuned model for inference (p. 43). If you select a customized model, you must have set up provisioned throughput for it beforehand.

   c. (Optional) Adjust the **Inference configurations**. The default settings are optimal for most prompts, but might not work well for your use case. You can restore the default settings by choosing the **Reset** button. For more information, see Inference parameters for foundation models (p. 30).
4. Enter your own prompt into the text field. A prompt is a natural language command, such as **`write a blog post about computers`**.
5. Under the input panel, choose **Run** to generate a text response.

   Note
   If the response violates the content moderation policy, Amazon Bedrock does not display it. If you have turned on streaming, Amazon Bedrock clears the entire response if it generates content that violates the policy. For more details, navigate to the Amazon Bedrock console, select **Providers**, and read the text under the **Content limitations** section.

Interact with the prompt and generated text

The generated text appears under your prompt in green text. Amazon Bedrock returns common markdown and tables in a rich text view. If you turn on streaming, you can't interact with the text until Bedrock finishes generating it. Interact with the prompt that you provided and the generated response in the following ways:

1. If the response is not to your satisfaction, you can edit the text directly. To interact with tables, code, and rich text in the response, use the icons at the upper-right of these objects:

   - **Edit** – Select the pencil icon to revert the table or code to plain text format for editing.
   - **Copy** – Select the copy icon to copy the object in plain text format.
2. Use the icons next to the **Run** button to undo and redo edits you have made.
3. Use the icons under the generated text to carry out the following actions:

   - **Copy** – Select the copy icon to copy the response as plain text.
   - **Delete** – Select the trash can icon to delete the latest model response.
4. After you experiment with the models, choose **View API request** to view the equivalent API request. You can copy this code into your application.

Re-run a query

You can modify the prompt and then re-run the query. Edit the prompt or response text, and then choose **Run**. After you edit the response text, the console changes the text color to black. The console displays the new response as green text.

Use advanced prompts

A prompt can consist of a simple command without any additional context, such as the example **`write a blog post about computers`**. These prompts are known as *zero-shot prompts*.

To improve the precision of the output, you can provide a few examples of the type of output you desire in the prompt. This is known as *few-shot prompting* (or *few-shot learning*).

In the following example, the first two sequences include the input text and the desired response. The last sequence is asking the model to answer in a similar format as the first two sequences.

```
Input: I love going to the mall
Sentiment: Happy
Input: I don't like going to the dentist.
Sentiment: Sad
Input: I enjoy the park
Sentiment:
```

In general, large language models work to predict the next logical word in a given sequence. How you prompt the model influences the model to produce a specific style of result.

Adjust the model parameters

When you choose a provider and model in the playground, the console loads the inference configuration parameters that apply to the selected model and sets their default values.

When you adjust the inference parameter settings, the model generates text that can differ in context, style, relevance, and length. Changing the parameter settings usually involves some trial and error to get the desired results.

For information about the inference parameters that each model supports, see Inference parameters for foundation models (p. 30).

Select a different model

You can select a different model provider and model from the dropdown menus at the top of the page. The console sets the inference configuration parameters to the default values for the model that you select.

Chat playground

From the Amazon Bedrock console, choose **Playgrounds** and then **Chat** to view the chat playground.

You can also navigate directly to the playground when you choose a model from a model details page or the examples page.

Topics

Submit a prompt to a chat model

The following procedure shows how to use the chat playground to submit a prompt to a chat model:

1. Open the Amazon Bedrock console.
2. From the left-side menu, choose **Chat** under **Playgrounds**.
3. Set up the chat playground.

    a. From the dropdown menu above the text panel, select a model provider. After you select a provider, select a model from the model dropdown menu. The dropdown menu lists the available models from this provider. You can also select a model that you have customized (for more information, Use a fine-tuned model for inference (p. 43)). If you select a customized model, you must have set up provisioned throughput for it beforehand.

    b. You can toggle on the following options in the top-right corner:

    - **Streaming** – Generate chat in real-time. This option isn't available if the model selected doesn't support streaming.

4. Select **Add instructions** to , then choose **Update**. In the **Response** panel, the console displays the response from the model.
5. Enter your own prompt into the text field. A prompt is a natural language phrase or command, such as **`Tell me about the best restaurants to visit in Seattle.`**

    Note
    If the response violates the content moderation policy, Amazon Bedrock doesn't display it. If you have turned on streaming, Amazon Bedrock clears the entire response if it generates content that violates the policy. For more details, navigate to the Amazon Bedrock console, select **Providers**, and read the text under the **Content limitations** section.

Update the chat instructions

You can modify the instructions and then update the chat model with the new instructions. Choose **Update** to edit the instructions to the model, and then choose **Confirm** after you make changes.

For example, you can add instructions to guide the personality and tone of the model responses. At any time, you can adjust the instructions to the model and re-run your prompt. This is useful when designing the conversation experience that users will have with the chat model.

Use advanced prompts

A prompt can consist of a simple command without any additional context, such as the example **`Tell me about the best car to buy.`** These prompts are known as *zero-shot prompts*.

To improve your prompt, you can provide a few examples of the type of output you desire. This is known as *few-shot prompting* (or *few-shot learning*).

For example, you can use a more complex prompt such as **`Tell me about the best car to buy. Give me a numbered list with the top 5 options. Include information on pricing, fuel efficiency, consumer ratings, and comfort of design. List the sources used`**

```
for each item in the list. hybrid or electric technology used. Order the list
so that the most desirable option is listed first.
```

Adjust the inference configuration

When you choose a provider and model in the playground, the console loads the inference configuration parameters that apply to the selected model and sets their default values.

When you adjust the inference parameter settings, the model generates text that can differ in context, style, relevance, and length. Changing the parameter settings usually involves some trial and error to get the desired results. To adjust the inference parameters, select the **Update** link in the bottom right of the chat window, and then choose **Confirm** after you make adjustments.

For information about the inference parameters that each model supports, see Inference parameters for foundation models (p. 30).

Select a different model

You can select a different model provider and model from the dropdown menus at the top of the page. The console sets the inference configuration parameters to the default values for the model that you select.

Image playground

From the Amazon Bedrock console, choose **Playgrounds** and then **Image** to view the image playground.

You can also navigate directly to the image playground after you choose a model from a model details page or the examples page.

Topics

Submit a prompt to a model

The following procedure shows how to use the image playground to submit a prompt to a text-to-image model.

1. Open the Amazon Bedrock console.
2. From the left-side menu, choose **Image** under **Playgrounds**.
3. From the dropdown menu above the text panel, select a model provider. After you select a provider, select a model from the model dropdown menu. The dropdown menu lists the available models from this provider.
4. Enter text into the **Prompt** field, or use one of the example prompts. A prompt is a natural language command, such as `Draw a picture of a computer`.
5. (Optional) Adjust the **Inference configurations**. The default settings are generally optimal but might not work for your use case. You can restore the default settings by choosing the **Reset** button.
6. Under the **Response** field, choose **Run** to generate the image. The generated image appears in the **Response** field, under your prompt.
7. (Optional) Choose **Download image** to save the image to your local machine.
8. (Optional) Choose **Save** to save the current prompt as a favorite in your history items.

9. After you experiment with the model, choose **View API request** to view the equivalent API request. You can copy this code into your application.

Adjust the model parameters

When you choose a provider and model in the playground, the console loads the inference configuration parameters that apply to the selected model and sets their default values.

When you adjust the inference parameter settings, the model generates a new image that can differ in style. Changing the parameter settings usually involves some trial and error to get the desired results.

For information about the inference parameters that each model supports, see Inference parameters for foundation models (p. 30).

Examples library

Amazon Bedrock provides example prompts for each of the supported models. From the console, choose **Examples** to view the available examples.

The Amazon Bedrock console displays up to 20 examples for each model provider. You can filter the list of examples using one or more of the following attributes:

- Modality (text, image, or embedding)
- Provider
- Model name
- Category

You can also filter by entering text in the text box. The console displays all the possible filter values that match the text.

When you select a specific example, the Amazon Bedrock console displays the following information about the example:

- A description of what the example accomplishes.
- The model name (and model provider) where the example runs.
- The example prompt and the expected response.
- The inference configuration parameter settings for the example.
- The API request that runs the example.

To run the example, choose **Open in playground**.

Use the Amazon Bedrock API

This section describes how to set up your environment to make Amazon Bedrock API calls and provides examples of common use-cases. You can access the Amazon Bedrock API using the AWS Command Line Interface (AWS CLI), an AWS SDK, or a SageMaker Notebook.

You can make API calls to Amazon Bedrock through the following SDKs:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for .NET
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

For details about the API operations and parameters, see the Amazon Bedrock API Reference.

The following resources provide additional information about the Amazon Bedrock API.

- *AWS General Reference*
  - Amazon Bedrock endpoints and quotas
- *AWS Command Line Interface*
  - Amazon Bedrock CLI commands
  - Amazon Bedrock Runtime CLI commands

Topics

Setting up the Amazon Bedrock API

Important
You must request access to a model before you can use it. If you try to use the model (with the API or console) before you have requested access to it, you receive an error message. For more information, see Model access (p. 11).

You can access the Amazon Bedrock API using the AWS CLI, an AWS SDK, or a SageMaker Notebook.

Amazon Bedrock endpoints

To connect programmatically to an AWS service, you use an endpoint. Refer to the Amazon Bedrock endpoints and quotas chapter in the AWS General Reference for information about the endpoints that you can use for Amazon Bedrock.

Setting up the AWS CLI

1. If you plan to use the CLI, install and configure the AWS CLI. See the section called "Set up the AWS CLI" (p. 5).
2. Configure your AWS credentials using the `aws configure` CLI command (see Configure the AWS CLI).

Refer to the following references for AWS CLI commands and operations:

- Amazon Bedrock CLI commands
- Amazon Bedrock Runtime CLI commands

Setting up an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language. Currently, you can make Amazon Bedrock API calls through the following SDKs.

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for .NET
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

For code examples, find a method in Amazon Bedrock API operations (p. 23) and select the tab that corresponds to the programming language of your choice.

Using SageMaker notebooks

You can use the SDK for Python (Boto3) to invoke Amazon Bedrock API operations from a SageMaker notebook.

Prerequisites

Note the following prerequisites:

1. Request Amazon Bedrock access for the AWS account that hosts the notebook.
2. Use the console to accept the Amazon Bedrock terms and conditions.

Configure the SageMaker role

Add Amazon Bedrock permissions to the IAM role for this SageMaker notebook.

From the IAM console, perform these steps:

1. Choose the IAM role, then choose **Add Permissions** and select **Create Inline Policies** from the dropdown list.

2. Include the following permission.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "bedrock:*",
            "Resource": "*"
        }
    ]
}
```

Add the following permissions to the trust relationships.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Test the Runtime setup

Add the following code to your notebook and run the code.

```
import boto3
import json
bedrock = boto3.client(service_name='bedrock-runtime')

body = json.dumps({
    "prompt": "\n\nHuman:explain black holes to 8th graders\n\nAssistant:",
    "max_tokens_to_sample": 300,
    "temperature": 0.1,
    "top_p": 0.9,
})

modelId = 'anthropic.claude-v2'
accept = 'application/json'
contentType = 'application/json'

response = bedrock.invoke_model(body=body, modelId=modelId, accept=accept,
 contentType=contentType)

response_body = json.loads(response.get('body').read())
# text
```

```
print(response_body.get('completion'))
```

Test the Amazon Bedrock setup

Add the following code to your notebook and run the code.

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.get_foundation_model(modelIdentifier='anthropic.claude-v2')
```

Amazon Bedrock API operations

Topics

List the base models

Use the ListFoundationModels operation to retrieve information about the base models, such as the model ID that you need to perform inference with the InvokeMethod operation. See the following code examples:

AWS CLI

List the base models using the following command in the AWS CLI:

```
aws bedrock list-foundation-models
```

Python (Boto)

The following example demonstrates how to list the base models using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.list_foundation_models()
```

Get details about a base model

Use the GetFoundationModel operation to retrieve detailed information about the specified base model. See the following code examples:

AWS CLI

Get information about a base model using the following command in the AWS CLI:

```
aws bedrock get-foundation-model --model-identifier anthropic.claude-v2
```

Python (Boto)

The following example demonstrates how to list the base models using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.get_foundation_model(modelIdentifier='anthropic.claude-v2')
```

Run inference

Use the InvokeModel operation to run inference on models. You use the`modelId` field to specify the model that you want to use. The method for finding the `modelId` depends on the type of model you use.

- **Base model** – Call ListFoundationModels to find the model ARN. For an example, see List the base models (p. 23). To read a list of the model IDs for the currently available base models, see Base model IDs (p. 27). You can also get the model ID for a foundation model from the JSON examples in the Amazon Bedrock console.
- **Custom model** – Call ListCustomModels and find the `modelArn` in the response. For more information, see Retrieve information about your customized models (p. 51). You can also find the model ARN in the **Model details** when you select a model in the **Fine-tuned models** section in the console.
- **Model with provisioned throughput** – If you have created a provisioned throughput for a foundation or custom model, call ListProvisionedModelThroughputs and find the `modelArn` in the response. You can also find the model ARN in the **Model details** when you select a model in the **Provisioned throuhput** section in the console.

Each base model has its own parameters that you set in the `body` field. For more information, see Inference parameters for foundation models (p. 30).

To run inference with streaming, use the InvokeModelWithResponseStream operation. Pick a base model that supports streaming or a model that you created from a base model that supports streaming.

Topics

- Running inference on a model (p. 24)
- Base model IDs (p. 27)

Running inference on a model

The following examples show how to run inference on a model with InvokeModel and, with Python, run inference with streaming with the InvokeModelWithResponseStream operation.

AWS CLI

The following example shows how to generate text with the AWS CLI using the prompt *"story of two dogs"* and the *Anthropic Claude V2* model. The example returns up to *300* tokens in the response and saves the response to the file *output.txt*:

```
aws bedrock-runtime invoke-model \
```

```
    --model-id anthropic.claude-v2 \
    --body "{\"prompt\": \"\n\nHuman:story of two dogs\n\nAssistant:\",
 \"max_tokens_to_sample\" : 300}" \
    --cli-binary-format raw-in-base64-out \
    invoke-model-output.txt
```

Note
The AWS CLI does not support streaming.

Python (Boto)

The following example shows how to generate text with Python using the prompt *"explain black holes to 8th graders"* and the *Anthropic Claude V2* model:

```
import boto3
import json
bedrock = boto3.client(service_name='bedrock-runtime')

body = json.dumps({
    "prompt": "\n\nHuman:explain black holes to 8th graders\n\nAssistant:",
    "max_tokens_to_sample": 300,
    "temperature": 0.1,
    "top_p": 0.9,
})

modelId = 'anthropic.claude-v2'
accept = 'application/json'
contentType = 'application/json'

response = bedrock.invoke_model(body=body, modelId=modelId, accept=accept,
 contentType=contentType)

response_body = json.loads(response.get('body').read())

# text
print(response_body.get('completion'))
```

The following example shows how to generate streaming text with Python using the prompt *"write an essay for living on mars in 1000 words"* and the *Anthropic Claude V2* model:

```
import boto3
import json

bedrock = boto3.client(service_name='bedrock-runtime')

body = json.dumps({
    'prompt': '\n\nHuman:write an essay for living on mars in 1000 words\n
\nAssistant:',
    'max_tokens_to_sample': 100
})

response = bedrock.invoke_model_with_response_stream(
    modelId='anthropic.claude-v2',
    body=body
)

stream = response.get('body')
if stream:
    for event in stream:
        chunk = event.get('chunk')
        if chunk:
            print(json.loads(chunk.get('bytes').decode()))
```

Base model inference examples

The following Python (Boto) examples show how you can perform inference with the InvokeModel operation on different Amazon Bedrock base models.

Topics

A2I Jurassic-2

This examples shows how to call the *A2I Jurassic-2 Mid* model.

```
import boto3
import json

bedrock = boto3.client(service_name='bedrock-runtime')

body = json.dumps({"prompt": "Translate to spanish: 'Amazon Bedrock is the easiest way to
 build and scale generative AI applications with base models (FMs)'.", "maxTokens": 200,
    "temperature": 0.5,"topP": 0.5})

modelId = 'ai21.j2-mid-v1'
accept = 'application/json'
contentType = 'application/json'

response = bedrock.invoke_model(body=body, modelId=modelId, accept=accept,
 contentType=contentType)

response_body = json.loads(response.get('body').read())

# text
print(response_body.get("completions")[0].get("data").get("text"))
```

Stability AI Diffusion XL

This example shows how to call the *Stability AI Stability Diffusion XL* model.

```
import boto3
import json

bedrock = boto3.client(service_name='bedrock-runtime')

prompt_data = "A photograph of an dog on the top of a mountain covered in snow."
body = json.dumps({
  "text_prompts": [
    {
      "text": prompt_data
    }
  ],
  "cfg_scale":10,
  "seed":20,
  "steps":50
})
modelId = "stability.stable-diffusion-xl-v0"
accept = "application/json"
contentType = "application/json"

response = bedrock.invoke_model(
    body=body, modelId=modelId, accept=accept, contentType=contentType
)
```

```
response_body = json.loads(response.get("body").read())
print(response_body['result'])
print(f'{response_body.get("artifacts")[0].get("base64")[0:80]}...')
```

Base model IDs

The following is a list of model IDs for the currently available base models. You use a model ID to identify the base model that you want to use in a call to InvokeModel or InvokeModelWithResponseStream.

Provider	Model name	Version	Model Id
AI21 Labs	Jurassic-2 Mid	1.x	ai21.j2-mid-v1
AI21 Labs	Jurassic-2 Ultra	1.x	ai21.j2-ultra-v1
Amazon	Titan Text G1 - Lite	1.x	amazon.titan-text-lite-v1
Amazon	Titan Embeddings G1 - Text	1.x	amazon.titan-embed-text-v1
Amazon	Titan Text G1 - Express	1.x	amazon.titan-text-express-v1
Amazon	Titan Text G1 - Agile	1.x	amazon.titan-text-agile-v1
Anthropic	Claude	1.x	anthropic.claude-v1
Anthropic	Claude	2.x	anthropic.claude-v2
Anthropic	Claude Instant	1.x	anthropic.claude-instant-v1
Cohere	Command	14.x	cohere.command-text-v14
Stability AI	Stable Diffusion XL	0.x	stability.stable-diffusion-xl-v0

Note
The Amazon Titan Text G1 - Express model is in limited preview release. Access will be granted on an ongoing basis.

Note
The Stability.ai models are in limited preview release. To request access, contact your AWS account manager. Stable Diffusion XL 1.x is only available with provisioned throughput. For more information, see Provisioned throughput (p. 55).

Tag resources

Use the TagResource and UntagResource operations to tag and untag resources. You need the ARN of the resource to tag/untag.

The Amazon Bedrock resources that you can tag are:

- Custom models

- Model customization jobs
- Provisioned models

For more information about restrictions on tagging, see Tag restrictions (p. 100).

To list the tags for a resource, use the ListTagsForResource operation.

AWS CLI

The following example demonstrates how to add two tags to a resource using the AWS CLI. Separate key/value pairs with a space:

```
aws bedrock tag-resource \
    --resource-arn "arn:aws:resource-arn" \
    --tags key=key1,value=value1 key=key2,value=value2
```

The following example demonstrates how to remove the tags with the keys *key1* and *key2* from a resource. Separate keys with a space:

```
aws bedrock untag-resource \
    --resource-arn "arn:aws:resource-arn" \
    --tag-keys key=key1 key2
```

The following example demonstrates how to list tags for a resource:

```
aws bedrock list-tags-for-resource \
    --resource-arn "arn:aws:iam::resource-arn"
```

Python (Boto)

The following example demonstrates how to add tags to a resource using Python:

```
import boto3

bedrock = boto3.client(service_name='bedrock')

tags = [
    {
        'key': 'key1',
        'value': 'value1'
    },
    {
        'key': 'key2',
        'value': 'value2'
    }
]

bedrock.tag_resource(resourceARN='arn:aws:bedrock:resource-arn', tags=tags)
```

The following example demonstrates how to remove the tags with the keys *key1* and *key2* from a resource using Python

```
import boto3

bedrock = boto3.client(service_name='bedrock')

bedrock.untag_resource(resourceARN='arn:aws:bedrock:resource-arn', tagKeys=['key1',
 'key2'])
```

The following example demonstrates how to list the tags for a resource using Python:

```
import boto3

bedrock = boto3.client(service_name='bedrock')

bedrock.list_tags_for_resource(resourceARN='arn:aws:bedrock:resource-arn')
```

Inference parameters for foundation models

Run inference using any of the foundation models in Bedrock. Optionally, set inference parameters to influence the response generated by the model. The following sections define the inference parameters available for each base model. For a custom model, use the same inference parameters as the base model from which it was customized.

Topics

Inference parameter definitions

Typically, foundation models support the following types of inference parameters.

Topics

Randomness and diversity

Foundation models typically support the following parameters to control randomness and diversity in the response.

- **Temperature**– Large language models use probability to construct the words in a sequence. For any given sequence, there is a probability distribution of options for the next word in the sequence. When you set the temperature closer to zero, the model tends to select the higher-probability words. When you set the temperature further away from zero, the model may select a lower-probability word.

  In technical terms, the temperature modulates the probability density function for the next tokens, implementing the temperature sampling technique. This parameter can deepen or flatten the density function curve. A lower value results in a steeper curve with more deterministic responses, and a higher value results in a flatter curve with more random responses.
- **Top K** – Temperature defines the probability distribution of potential words, and Top K defines the cutoff where the model no longer selects the words. For example, if K=50, the model selects from 50 of the most probable words that could be next in a given sequence. When you lower the Top K value, it reduces the probability that an unusual word gets selected next in a sequence.

In technical terms, Top K is the number of the highest-probability vocabulary tokens to keep for Top-K-filtering.

- **Top P** – Top P defines a cut off based on the sum of probabilities of the potential choices. If you set Top P below 1.0, the model considers the most probable options and ignores less probable ones. Top P is similar to Top K, but instead of capping the number of choices, it caps choices based on the sum of their probabilities.

  For the example prompt "I hear the hoof beats of ," you might want the model to provide "horses," "zebras," or "unicorns" as the next word. If you set the temperature to its maximum, without capping Top K or Top P, you increase the probability of getting unusual results such as "unicorns." If you set the temperature to 0, you increase the probability of "horses." If you set a high temperature and reduce the value of Top K or Top P, you increase the probability of "horses" or "zebras," and decrease the probability of "unicorns."

Length

Foundation models typically support the following parameters control the length of the generated response.

- **Response length** – Configures the minimum and maximum number of tokens to use in the generated response.
- **Length penalty** – Length penalty optimizes the model to be more concise in its output by penalizing longer responses.

  In technical terms, the length penalty penalizes the model exponentially for lengthy responses. 0.0 means no penalty. To generate longer sequences, set a value less than 0.0 for the model. To generate shorter sequences, set a value greater than 0.0.
- **Stop sequences** – A stop sequence is a sequence of characters. If the model encounters a stop sequence, it stops generating further tokens. Different models support different types of characters in a stop sequence, different maximum sequence lengths, and may support the definition of multiple stop sequences.

Repetitions

Foundation models typically support the following parameters help control repetition in the generated response.

- **Repetition penalty (presence penalty)** – Prevents repetitions of the same words (tokens) in responses. 1.0 means no penalty. Greater than 1.0 decreases repetition.

Amazon Titan models

The Amazon Titan models support the following inference parameters.

Randomness and Diversity

The Amazon Titan models support the following parameters to control randomness and diversity in the response.

- **Temperature** (`temperature`)– Use a lower value to decrease randomness in the response.
- **Top P** (`topP`) – Use a lower value to ignore less probable options.

Length

The Amazon Titan models support the following parameters to control the length of the generated response.

- **Response length** (`maxTokenCount`) – Specify the maximum number of tokens in the generated response.
- **Stop sequences** (`stopSequences`) – Specify character sequences to indicate where the model should stop. Use the | (pipe) character to separate different sequences (maximum 20 characters).

When you make an InvokeModel or InvokeModelWithResponseStream call using a Titan model, fill the body field with a JSON object that conforms to the one below. Enter the prompt in the `inputText` field.

```
{
    "inputText": string,
    "textGenerationConfig": {
        "temperature": float,
        "topP": float,
        "maxTokenCount": int,
        "stopSequences": [string]
    }
}
```

The following table shows the minimum, maximum, and default values for the numerical parameters.

Category	Parameter	JSON field format	Minimum	Maximum	Default
Randomness and diversity	Temperature	temperature	0	1	0
	Top P	topP	0	1	1
Length	Response length	maxTokenCount	0	8,000	512

Anthropic Claude models

The Anthropic Claude models support the following types of controls.

Randomness and diversity

The Anthropic Claude models support the following parameters to control randomness and diversity in the response.

- **Temperature** (`temperature`)– Use a lower value to decrease randomness in the response.
- **Top P** (`topP`) – Use a lower value to ignore less probable options.
- **Top K** (`topK`) – Specify the number of token choices the model uses to generate the next token.

Length

The Anthropic Claude models support the following parameters to control the length of the generated response.

- **Maximum length** (`max_tokens_to_sample`) – Specify the maximum number of tokens to use in the generated response. We recommend a limit of 4,000 tokens for optimal performance.
- **Stop sequences** (`stop_sequences`) – Configure up to four sequences that the model recognizes. After a stop sequence, the model stops generating further tokens. The returned text doesn't contain the stop sequence.

When you make an InvokeModel or InvokeModelWithResponseStream call using an Anthropic model, fill the `body` field with a JSON object that conforms to the one below. Copy the format in the `prompt` field, replacing *`prompt`* with your prompt.

```
{
    "prompt": "\n\nHuman:<prompt>\n\nAssistant:",
    "temperature": float,
    "top_p": float,
    "top_k": int,
    "max_tokens_to_sample": int,
    "stop_sequences": ["\n\nHuman:"]
}
```

The following table shows the minimum, maximum, and default values for the numerical parameters.

Category	Parameter	JSON object format	Minimum	Maximum	Default
Randomness and diversity	Temperature	temperature	0	1	0.5
	Top P	top_p	0	1	1
	Top K	top_k	0	500	250
Length	Max tokens to sample	max_tokens_to_sample	0	8,000	200

AI21 Labs Jurassic-2 models

The AI21 Jurassic-2 models support the following types of controls.

Randomness and Diversity

The AI21 Jurassic-2 models support the following parameters to control randomness and diversity in the response.

- **Temperature** (`temperature`)– Use a lower value to decrease randomness in the response.
- **Top P** (`topP`) – Use a lower value to ignore less probable options.

Length

The AI21 Jurassic-2 models support the following parameters to control the length of the generated response.

- **Max completion length** (`maxTokens`) – Specify the maximum number of tokens to use in the generated response.

- **Stop sequences** (`stopSequences`) – Configure stop sequences that the model recognizes and after which it stops generating further tokens. Press the Enter key to insert a newline character in a stop sequence. Use the Tab key to finish inserting a stop sequence.

Repetitions

The AI21 Jurassic-2 models support the following parameters to control repetition in the generated response.

- **Presence penalty** (`presencePenalty`) – Use a higher value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion.
- **Count penalty** (`countPenalty`) – Use a higher value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion. Proportional to the number of appearances.
- **Frequency penalty** (`frequencyPenalty`) – Use a high value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion. The value is proportional to the frequency of the token appearances (normalized to text length).
- **Penalize special tokens** – Reduce the probability of repetition of special characters. The default values are `true`.
  - **Whitespaces** (`applyToWhitespaces`) – A `true` value applies the penalty to whitespaces and new lines.
  - **Punctuations** (`applyToPunctuation`) – A `true` value applies the penalty to punctuation.
  - **Numbers** (`applyToNumbers`) – A `true` value applies the penalty to numbers.
  - **Stop words** (`applyToStopwords`) – A `true` value applies the penalty to stop words.
  - **Emojis** (`applyToEmojis`) – A `true` value excludes emojis from the penalty.

When you make an InvokeModel or InvokeModelWithResponseStream call using an AI21 model, fill the body field with a JSON object that conforms to the one below. Enter the prompt in the `prompt` field.

```
{
    "prompt": string,
    "temperature": float,
    "topP": float,
    "maxTokens": int,
    "stopSequences": [string],
    "countPenalty": {
        "scale": int
    },
    "presencePenalty": {
        "scale": float
    },
    "frequencyPenalty": {
        "scale": int
    }
}
```

To penalize special tokens, add those fields to any of the penalty objects. For example, you can modify the `countPenalty` field as follows.

```
{
    "countPenalty": {
        "scale": int,
        "applyToWhitespaces": boolean,
        "applyToPunctuations": boolean,
```

```
        "applyToNumbers": boolean,
        "applyToStopwords": boolean,
        "applyToEmojis": boolean
    }
}
```

The following table shows the minimum, maximum, and default values for the numerical parameters.

Category	Parameter	JSON object format	Minimum	Maximum	Default
Randomness and diversity	Temperature	temperature	0	1	0.5
	Top P	topP	0	1	0.5
Length	Max tokens (mid, ultra, and large models)	maxTokens	0	8,191	200
	Max tokens (other models)		0	2,048	200
Repetitions	Presence penalty	presencePenalty	0	5	0
	Count penalty	countPenalty	0	1	0
	Frequency penalty	frequencyPenalty	0	500	0

Cohere Command model

The Cohere Command model supports the following controls.

- **Return likelihoods** (`return_likelihoods`) – Specify how and if the token likelihoods are returned with the response. You can specify the following options. The default option is `NONE`.
  - `GENERATION` – Only return likelihoods for generated tokens.
  - `ALL` – Return likelihoods for all tokens.
  - `NONE` – Don't return any likelihoods.
- **Stream** (`stream`) – Specify `true` to return the response piece-by-piece in real-time and `false` to return the complete response after the process finishes.

Randomness and diversity

The Cohere Command model supports the following parameters to control randomness and diversity in the response.

- **Temperature** (`temperature`) – Use a lower value to decrease randomness in the response.
- **Top P** (p) – Use a lower value to ignore less probable options. Set to 0 or 1.0 to disable. If both p and k are enabled, p acts after k.
- **Top K** (k) – Specify the number of token choices the model uses to generate the next token. If both p and k are enabled, p acts after k.

Length

The Cohere Command model supports the following parameters to control the length of the generated response.

- **Maximum length** (`max_tokens`) – Specify the maximum number of tokens to use in the generated response.
- **Stop sequences** (`stop_sequences`) – Configure up to four sequences that the model recognizes. After a stop sequence, the model stops generating further tokens. The returned text doesn't contain the stop sequence.

When you make an InvokeModel or InvokeModelWithResponseStream call using a Cohere model, fill the `body` field with a JSON object that conforms to the one below. Enter the prompt in the `prompt` field.

```
{
    "prompt": "string",
    "temperature": float,
    "p": float,
    "k": float,
    "max_tokens": int,
    "stop_sequences": ["string"],
    "return_likelihoods": "GENERATION|ALL|NONE",
    "stream": boolean,
    "num_generations": int
}
```

The following table shows the minimum, maximum, and default values for the numerical parameters.

Category	Parameter	JSON object format	Minimum	Maximum	Default
Randomness and diversity	Temperature	temperature	0	5	0.9
	Top P	p	0	1	0.75
	Top K	k	0	500	0
Length	Max tokens	max_tokens	1	4,096	20
Other	Number of generations	num_generations	1	5	1

Stability.ai Diffusion models

The Stability.ai Diffusion models support the following controls.

- **Prompt strength** (`cfg_scale`) – Determines how much the final image portrays the prompt. Use a lower number to increase randomness in the generation.
- **Generation step** (`steps`) – Generation step determines how many times the image is sampled. More steps can result in a more accurate result.
- **Seed** (`seed`) – The seed determines the initial noise setting. Use the same seed and the same settings as a previous run to allow inference to create a similar image. If you don't set this value, it is set as a random number.

When you make an InvokeModel or InvokeModelWithResponseStream call using a Stability.ai model, fill the body field with a JSON object that conforms to the one below. Enter the prompt in the `text` field in the `text_prompts` object.

```
{
    "text_prompts": [
        {"text": "string"}
    ],
    "cfg_scale": float,
    "steps": int,
    "seed": int
}
```

The following table shows the minimum, maximum, and default values for the numerical parameters.

Parameter	JSON object format	Minimum	Maximum	Default
Prompt strength	cfg_scale	0	30	10
Generation step	step	10	150	30

Embeddings

Text embeddings represent meaningful vector representations of unstructured text such as documents, paragraphs, and sentences. You input a body of text and the output is a (1 x n) vector. You can use embedding vectors for a wide variety of applications.

Bedrock supports one model for text embeddings, the Titan Embeddings G1 - Text model (`amazon.titan-embed-text-v1`). This model supports text retrieval, semantic similarity, and clustering. The maximum input text is 8K tokens and the maximum output vector length is 1536.

To use a text embeddings model, use the InvokeModel API operation with `amazon.titan-embed-text-v1` as the `modelId` and retrieve the `embedding` object in the response.

To see Jupyter notebook examples:

1. Sign in to the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/home.
2. From the left-side menu, choose **Base models**.
3. Scroll down and select the **Titan Embeddings G1 - Text** model.
4. In the **Titan Embeddings G1 - Text** tab, select **View example notebook** to see example notebooks for embeddings.

Custom models

> **Note**
> To use fine-tuning, you must have access to the Amazon Titan Text G1 - Express model. The Amazon Titan Text G1 - Express model is in limited preview. To request access, contact your AWS account manager.

You can customize a Amazon Bedrock model to improve its performance and create a better customer experience. Amazon Bedrock currently offers the ability to fine-tune a model by providing your own labeled training data to help retain or improve its accuracy while employing smaller datasets and therefore reducing the training time.

After you complete a model customization job, you can purchase provisioned throughput (see Provisioned throughput (p. 55)) for the customized model so that you can use the model for inference using the InvokeModel or InvokeModelWithResponseStream API operations or the text playground (see Text playground (p. 14)).

For information about quotas for model customization, see Model customization quotas (p. 139).

Topics

Fine-tuning

Fine-tune a Amazon Bedrock model by providing your own labeled training dataset in order to improve the model's performance on specific tasks. By providing some labeled examples related to a specific task, you help the model learn the task it's supposed to carry out. Through this process, known as *few-shot fine-tuning*, you can create a new model that improves upon the performance and efficiency of the original model for a given task.

To fine-tune a model, you upload a training dataset and, optionally, a validation dataset to Amazon S3 and provide the Amazon S3 bucket path to the Bedrock fine-tuning job. You can also adjust the hyperparameters for fine-tuning. To fine-tune a model in the console, follow the steps at Using the console (p. 40). To fine-tune through the API, follow the steps at Using the API (p. 43).

Prepare the datasets

Before you upload your training and validation data to Amazon S3, you need to pre-process the format of your data. Model customization only supports JSONL format. Use 6 characters per token as an approximation for the number of tokens.

If you provide a validation dataset (which is optional), Amazon Bedrock returns validation loss metrics at the end of the model customization job.

Note
The following formats pertain to fine-tuning a Titan Text G1 - Express model.

After you pre-process your training and validation data, upload the datasets to Amazon S3 and provide permissions for Amazon Bedrock to access the data by attaching an IAM policy similar to the example shown in Grant custom jobs access to your training data (p. 122) to your Amazon Bedrock service role.

For both the training and optional validation sets, each line contains both an `input` and `output` field. The format is as follows. For quotas, see Quotas for Amazon Bedrock (p. 139).

```
{"input": "<prompt text>", "output": "<expected generated text>"}
{"input": "<prompt text>", "output": "<expected generated text>"}
{"input": "<prompt text>", "output": "<expected generated text>"}
```

The following is an example item for a question-answer task:

```
{"input": "what is AWS", "output": "it's amazon web services"}
```

Using the console

The following topics describe the major steps required to fine-tune a model:

Topics

Submit a job

Note
Before you begin these steps, be sure to grant Amazon Bedrock permissions to access the training and validation data and to write the output data by attaching an IAM policy similar to the example shown in Grant custom jobs access to your training data (p. 122) to your Amazon Bedrock service role.

To submit a fine-tuning job

1. Sign in to the AWS Management Console and open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/home.
2. From the left menu, choose **Custom models**.
3. Select **Fine-tune model**.
4. Configure the settings for the fine-tuning job:
   a. In the **Model details** section, choose the **Source model** that you want to fine-tune with your own data and give your resulting fine-tuned model a name in **Fine-tuned model name**.
      Note
      Currently, Titan Text G1 - Express is the only supported model for fine-tuning.
   b. (Optional) You can choose **Model encryption** to choose a different KMS key from the default key.

For information about configuring KMS keys, see Data encryption (p. 102).

c. (Optional) You can expand the **Tags** section and select **Add new tag** add tags to associate with the model.

d. In the **Job configuration** section, enter a name for the fine-tuning training job in **Job name** and add any tags to associate with the job.

e. (Optional) In the **VPC settings** section, choose the **VPC** that contains the input data and output data Amazon S3 locations.

   i. For **Subnet(s)**, add the VPC subnets.

   ii. For **Security group(s)**, add security groups to control access to the data in your VPC.

   For information about configuring the VPC, see Protect jobs using a VPC (p. 105).

f. Provide the link to the training dataset file and, optionally, the validation dataset file in your Amazon S3 bucket in the **Input data** section.

g. In the **Hyperparameters** section, input the values for the following hyperparameters to use in training. See Guidelines for model customization (p. 51) for guidelines on choosing values.

   - **Epochs** – The number of times to pass the training dataset to the model.
   - **Batch size** – The number of records to send to the model in each batch.
   - **Learning rate** – The step size for parameter updates in each iteration.
   - **Learning rate warmup steps** – Affects the speed at which the algorithm converges to the optimal weights.

h. Provide a link to the Amazon S3 folder in which to save the output of the fine-tuning job in the **Output data** section. The training loss metrics and validation loss metrics for each epoch will be stored in separate files in the location that you specify.

i. You use a service role to provide permissions for Amazon Bedrock to write to Amazon S3 on your behalf.

   In the **Service access** section, select one of the following:

   - **Use an existing service role** – Select a service role from the drop-down list.
   - **Create and use a new service role** – Enter a name for the service role.

   Note
   If your job includes VPC configuration, the console cannot create a new service role for the job. Create the service role using the example described in Configure your model customization job to use VPC (p. 109).

5. Select **Fine-tune model** to begin the job.

Monitor the job

The fine-tuning job can take several hours. The duration of the job depends on the size of the training data (number of records, input tokens, and output tokens), number of epochs, and batch size.

To monitor the status of the fine-tuning job

1. Open the Amazon Bedrock console.
2. From the left menu, choose **Custom models**.
3. The **Training jobs** tab displays the fine-tuning jobs that you have initiated. Look at the **Status** column to monitor the progress of the job.

4. Select a job to view the details you input for training.

Stop a job

You can stop a Bedrock fine-tuning job while it's in progress. You can't resume a stopped job.

To stop a fine-tuning job

1. Open the Amazon Bedrock console.
2. From the left menu, chose **Custom models**.
3. From the **Training Jobs** tab, choose the radio button next to the job to stop.
4. Select the **Stop job** button.
5. A modal appears to warn you that you can't resume the training job if you stop it. Select **Stop job** to confirm.

   Note
   Amazon Bedrock charges for the tokens that it used to train the model before you stopped the job. Amazon Bedrock doesn't create an intermediate custom model for a stopped job.

Analyze the job results

After the fine-tuning job completes, you can see the following information in your S3 output folder:

- Completion status
- Training and validation loss metrics

Amazon Bedrock stores your customized models in AWS-managed storage scoped to your account. You can see your customized model in the **Models** table of the model customization dashboard page. Choose a model to view details associated with that custom model. Output models aren't downloadable.

The S3 output for a fine-tuning job contains the following output files in your S3 folder:

```
- model-customization-job-training_job_id/
    - training_artifacts/
        - step_wise_training_metrics.csv
    - validation_artifacts/
        - post_fine_tuning_validation/
            - validation_metrics.csv
```

Use the `step_wise_training_metrics.csv` and the `validation_metrics.csv` files to analyze the model customization job and to help you adjust the model as necessary.

The structure of the `step_wise_training_metrics.csv` file is shown in the following example:

step_number	epoch_number	training_loss	perplexity
1	1	0.2	25
2	1	0.18	22
...	...	...	18

The structure of the `validation_metrics.csv` file is shown in the following example:

step_number	epoch_number	validation_loss	perplexity
1	1	0.12	20
2	1	0.09	17
...	...	...	15

Use a fine-tuned model for inference

Before you can use a customized model for inference, you need to purchase provisioned throughput for it. You can then use it for inference in the Text or Chat playground.

To purchase provisioned throughput for a custom model.

1. Open the Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left side navigation pane, select **Custom models**.
3. Choose the radio button next to the model, select **Actions**, and choose **Purchase provisioned throughput**. For more information, see Provisioned throughput (p. 55).

To use a fine-tuned model for inference

1. From the left side navigation pane, select **Custom models**.
2. In the **Models** tab, choose the model that you want to use in the Text or Chat Playground and select **Open in playground**.

Using the API

This section demonstrates how to fine-tune your models using API operations. We provide examples with the AWS Command Line Interface and the AWS SDK for Python (Boto3).

Topics

Set up an IAM role for model customization

Create an IAM role or use an existing role that grants the required permissions for your customization job to access the data in your Amazon S3 buckets. Use the IAM CreateRole API operation to allow Amazon Bedrock to assume a role and the IAM CreatePolicy API operation to create a policy to allow the role to access the Amazon S3 buckets containing your training, validation, and output data. Attach the policy with the AttachRolePolicy API operation. Skip this step if you are using an existing role with the correct IAM permissions. If you submit a job using the console (see Submit a job (p. 40)), the console can create a role with the correct permissions for you.

Preparing the role and access policies:

1. Create the following policy document called *BedrockAssumeRolePolicy.json* (or another name of your choice) to allow Amazon Bedrock to assume a role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
                }
            }
        }
    ]
}
```

2. Determine the Amazon S3 locations where you uploaded your training and validation data and the location for Amazon Bedrock to upload your training metrics data.
3. Create the following policy document called *BedrockAccessTrainingValidationS3Policy.json* (or another name of your choice) to allow access to these Amazon S3 locations, replacing the values in the `Resource` list with actual Amazon S3 ARNs:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:ListObjects"
            ],
            "Resource": [
                "arn:aws:s3:::my_training_data_bucket/myfolder",
                "arn:aws:s3:::my_training_data_bucket/myfolder/*",
                "arn:aws:s3:::my_validation_data_bucket/myfolder",
                "arn:aws:s3:::my_validation_data_bucket/myfolder/*"
            ]
        }
    ]
}
```

4. Create the following policy called *BedrockAccessOutputS3Policy.json* to allow both access and writing to these Amazon S3 locations, replacing the values in the `Resource` list with actual Amazon S3 ARNs:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:ListObjects"
            ],
            "Resource": [
                "arn:aws:s3:::my_output_bucket/myfolder",
                "arn:aws:s3:::my_output_bucket/myfolder/*"
            ]
        }
    ]
}
```

AWS CLI

1. Create an IAM role, replacing the `role-name` with a name of your choice and using the file name for the assume role policy you created:

```
aws iam create-role \
    --role-name "MyBedrockModelCustomizationRole" \
    --assume-role-policy-document file://path/to/BedrockAssumeRolePolicy.json
```

2. Create the training and validation data access policy, replacing the `policy-name` with a name of your choice:

```
aws iam create-policy \
    --policy-name "MyBedrockTrainingValidationS3BucketPolicy" \
    --policy-document file://path/to/BedrockAccessTrainingValidationS3Policy.json
```

3. Create the output data access policy, replacing the `policy-name` with a name of your choice:

```
aws iam create-policy \
    --policy-name "MyBedrockOutputS3BucketPolicy" \
    --policy-document file://path/to/BedrockAccessOutputS3Policy.json
```

4. Attach the IAM role training and data access policy, replacing the `role-name` with the role name you created in step 1 and the `policy-arn` with the ARN that the response in step 2 returned to you:

```
aws iam attach-role-policy \
    --role-name "MyBedrockModelCustomizationRole" \
    --policy-arn "PolicyARNFromCreateTrainingValidationPolicyStep"
```

5. Attach the IAM role output data access policy, replacing the `role-name` with the role name you created in step 1 and the `policy-arn` with the ARN that the response in step 3 returned to you:

```
aws iam attach-role-policy \
    --role-name "MyBedrockModelCustomizationRole" \
    --policy-arn "PolicyARNFromCreateOutputPolicyStep"
```

6. Use the ARN returned to you in step 1 as the `roleArn` when submitting a model customization job.

Python

1. Create an IAM role by following the code sample in the **Create a role** section in IAM examples using SDK for Python (Boto3), replacing the `role_name` with a name of your choice and replacing the `trust_policy` with the contents from the *`BedrockAssumeRolePolicy.json`* you created.
2. Create two access policies by following the code sample in the **Create a policy** section in IAM examples using SDK for Python (Boto3), replacing the `PolicyName` with two different names of your choice and replacing the `policy_doc` with the contents from the *`BedrockAccessTrainingValidationS3Policy.json`* and the *`BedrockAccessOutputS3Policy.json`* that you created.
3. Attach the IAM role policies, by following the code sample in the **Attach a policy to a role** section in IAM examples using SDK for Python (Boto3), replacing replacing the `role_name` with the role name you created in step 1 and the `policy_arn` with the ARNs that the responses in step 2 returned to you.
4. Use the ARN returned to you in step 1 as the `roleArn` when submitting a model customization job.

Submit a job

Use the Amazon Bedrock CreateModelCustomizationJob API operation to submit a model customization job. Minimally, you must provide the following fields in the CreateModelCustomizationJob request:

- `customizationType` – To fine-tune a model, use the value **`FINE_TUNING`**.
- `baseModelIdentifier` – The ARN of the model to customize.
- `customModelName` – The name to give the newly customized model.
- `hyperParameters` – Parameters related to tuning the model. For the Titan Text G1 - Express model, the following fields can be specified in this object. See Guidelines for model customization (p. 51) for guidelines on choosing values.
  - **Epochs** – The number of times to pass the training dataset to the model.
  - **Batch size** – The number of records to send to the model in each batch.
  - **Learning rate** – The step size for parameter updates in each iteration.
  - **Learning rate warmup steps** – Affects the speed at which the algorithm converges to the optimal weights.
- `jobName` – The name to give the training job.
- `roleArn` – The ARN of the service role.
- `trainingDataConfig` – An object containing the URI of the Amazon S3 location of the training data.
- `validationDataConfig` – An object containing the URI of the Amazon S3 location of the validation data.
- `outputDataConfig` – An object containing the URI of the Amazon S3 location to write the output data to.

The response returns a `jobArn` that you can use to monitor or stop the model customization job.

AWS CLI

The following example demonstrates how to submit a model customization job using the AWS CLI:

First create a text file named *`FineTuningData.json`*. Copy the JSON code from below into the text file, replacing the bucket, path, and file names with the correct paths to your training, validation, and output data:

```
{
    "trainingDataConfig": {
        "s3Uri": "s3://bucket/path/to/train.jsonl"
    },
    "validationDataConfig": {
        "validators": [{
            "s3Uri": "s3://bucket/path/to/validation.jsonl"
        }]
    },
    "outputDataConfig": {
        "s3Uri": "s3://bucket/path/to/output-folder"
    }
}
```

Run the following command in the command line

Note
Currently, Titan Text G1 - Express is the only supported model for fine-tuning.

```
aws bedrock create-model-customization-job \
    --customization-type "FINE_TUNING" \
    --base-model-identifier "arn:aws:bedrock:us-east-1::foundation-model/foundation-
model-id" \
    --role-arn "arn:aws:iam::arn-for-MyBedrockModelCustomizationRole" \
    --job-name "job-name" \
    --custom-model-name "custom-model-name" \
    --hyper-parameters
 epochCount="1",batchSize="1",learningRate="0.005",learningRateWarmupSteps="0" \
    --cli-input-json file://path/to/FineTuningData.json
```

To add a VPC configuration, add the following argument to the above command to specify the security group and subnets:

```
   --vpc-config '{securityGroupIds": ["xx"], "subnetIds": ["subnet-yy", "subnet-zz"]}'
```

To encrypt your model with a KMS key, add the following argument to the above command, replacing the values to specify the key with which you want to encrypt your model.

```
   --customModelKmsKeyId 'arn:aws:kms:region:account-id:key/key-id'
```

To add tags, add the following argument to the above command, replacing the keys and values with the tags you want to attach to the job and/or output model and making sure to separate key/value pairs with a space:

```
   --tags key=key1,value=value1 key=key2,value=value2
```

Use the `jobArn` that the operation returns to check the status of the job or to analyze or stop the job.

Python

The following example demonstrates how to submit a model customization job using Python. Uncomment the relevant sections to add optional tags to the job and/or resulting model:

```
import boto3
import json
bedrock = boto3.client(service_name='bedrock')

# Set parameters
customizationType = "FINE_TUNING"
baseModelIdentifier = "arn:aws:bedrock:us-east-1::foundation-model/foundation-model-id"
roleArn = "arn:aws:iam::arn-for-MyBedrockModelCustomizationRole"
jobName = "job-name"
customModelName = "custom-model-name"
hyperParameters = {
        "epochCount": "1",
        "batchSize": "1",
        "learningRate": "0.005",
        "learningRateWarmupSteps": "0"
    }
trainingDataConfig = {"s3Uri": "s3://bucket/path/to/train.jsonl"}
validationDataConfig = {
        "validators": [{
            "name": "validation",
            "s3Uri": "s3://bucket/path/to/validation.jsonl"
        }]
    }
outputDataConfig = {"s3Uri": "s3://bucket/path/to/"}

# # Uncomment to add optional tags
# jobTags = [
#     {
#         "key": "key1",
#         "value": "value1"
#     }
# ]
# customModelTags = [
#     {
#         "key": "key1",
#         "value": "value1"
#     }
# ]

# Create job
bedrock.create_model_customization_job(
    jobName=jobName,
    customModelName=customModelName,
    roleArn=roleArn,
    baseModelIdentifier=baseModelIdentifier,
    hyperParameters=hyperParameters,
    #    # Uncomment to add optional tags
    #    jobTags=jobTags,
    #    customModelTags=customModelTags,
    trainingDataConfig=trainingDataConfig,
    validationDataConfig=validationDataConfig,
    outputDataConfig=outputDataConfig
)
```

Use the `jobArn` that the code returns to check the status of the job or to analyze or stop the job.

Monitor a job

To list all your model customization jobs, send an Amazon Bedrock ListModelCustomizationJobs request. To monitor the status of a model customization job, send an Amazon Bedrock GetModelCustomizationJob request by entering the `jobArn`. You can find it in one of the following ways:

1. In the Amazon Bedrock console, select **Fine-tune** and choose the job from the **Training jobs** table. Look for the **Model customization job ARN** in the **Model configuration** section.
2. Look in the `jobArn` field in the response returned from the `CreateModelCustomizationJob` call that created the job.

Monitor the progress of the job by looking in the `status` field of the response.

AWS CLI

List your model customization jobs using the AWS CLI with the following command:

```
aws bedrock list-model-customization-jobs
```

The following example demonstrates how to monitor a model customization job using the AWS CLI:

```
aws bedrock get-model-customization-job \
    --job-identifier "arn:aws:bedrock:job-arn-from-create-model-customization"
```

Find the value of the `status` field in the response.

Python

The following demonstrates how to list your model customization jobs using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.list_model_customization_jobs()
```

The following example demonstrates how to monitor a model customization job:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

fine_tune_job = bedrock.get_model_customization_job(jobIdentifier='arn:aws:bedrock:job-arn-from-create-model-customization')

print(fine_tune_job['status'])
```

Stop a job

To use the API to stop a model customization job, follow these steps:

1. If the job `status` in the GetModelCustomizationJob response is `IN_PROGRESS`, send a StopModelCustomizationJob request with the `jobArn` of the training job. The system marks the job for termination and sets the state to `STOPPING`.
2. The system stops the job and sets the state to `STOPPED`.

   If the job completes before the system stops it, the system sets the state to `COMPLETED`.

   Note
   Amazon Bedrock charges for the tokens that it used to train the model before you stopped the job. Amazon Bedrock doesn't create an intermediate custom model for a stopped job.

AWS CLI

The following example demonstrates how to stop a model customization job using the AWS CLI:

```
aws bedrock stop-model-customization-job \
    --job-identifier "arn:aws:bedrock:job-arn-from-create-model-customization"
```

Python

The following example demonstrates how to stop a model customization job using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.stop_model_customization_job(jobIdentifier='arn:aws:bedrock:job-arn-from-create-model-customization')
```

Analyze a job

To analyze a model customization job, send an Amazon Bedrock GetCustomModelb request by entering the `jobArn`. You can find it in one of the following ways:

1. In the Amazon Bedrock console, select **Fine-tune** and choose the job from the **Training jobs** table. Look for the **Model customization job ARN** in the **Model configuration** section.
2. Look in the `jobArn` field in the response returned from the `CreateModelCustomizationJob` call that created the job.

You can analyze metrics by using the `trainingLoss` value in the `trainingMetrics` object and the `validatorLoss` values in the `validationMetrics` object to analyze the model customization job or by reading the data from the S3 output file.

AWS CLI

The following example demonstrates how to analyze a model customization job using the AWS CLI:

```
aws bedrock get-model-customization-job \
    --job-identifier "arn:aws:bedrock:job-arn-from-create-model-customization"
```

Look in the `trainingMetrics` and `validatorLoss` fields to analyze the model customization job.

Python

The following example demonstrates how to analyze a model customization job using Python:

```
import boto3
import json
import pandas as pd

bedrock = boto3.client(service_name='bedrock')

fine_tune_job = bedrock.get_model_customization_job(jobIdentifier='arn:aws:bedrock:job-arn-from-create-model-customization')


# Get S3 URI
fine_tune_job['outputDataConfig']['s3Uri']
```

Use the S3 URI to download the output data. For more information, see Downloading objects. Open the folder and navigate to the metrics files.

Retrieve information about your customized models

Model customization creates a custom model upon successful completion of the job. To list your custom models, send an Amazon Bedrock ListCustomModels request. To get information about a specific model that you have fine-tuned, send an Amazon Bedrock GetCustomModel request, providing the ARN of the model as the `modelIdentifier` in the request. You can find the ARN of the model in one of the following ways:

1. In the Amazon Bedrock console, select **Fine-tune** and choose the model from the **Models** table. Look for the **Fine-tuned model ARN** in the **Model details** section.
2. Look in the `outputModelArn` field in the response returned from the `GetModelCustomizationJob` call that created the job. This field only appears in the response after the job has finished.

AWS CLI

Use the following command to list your fine-tuned models in the AWS CLI:

```
aws bedrock list-custom-models
```

The following example demonstrates how to get information about a fine-tuned model in the AWS CLI

```
aws bedrock get-custom-model \
    --model-identifier "arn:aws:bedrock:customized-model-arn"
```

Python

Use the following command to list your fine-tuned models in Python:

```
import boto3
import json
bedrock = boto3.client(service_name='bedrock')

bedrock.list_custom_models()
```

The following example demonstrates how to get information about a fine-tuned model in the AWS CLI

```
import boto3
import json
bedrock = boto3.client(service_name='bedrock')

bedrock.get_custom_model(modelIdentifier='arn:aws:bedrock:customized-model-arn')
```

Guidelines for model customization

In this section, we provide guidelines and recommended values as a baseline for customization of the Titan Text G1 - Express model. However, you should experiment with values to determine which parameters work best for your specific case. The ideal parameters depend on the dataset and the task for which the model is intended.

Use the training and validation metrics from the output files generated when you submit a fine-tuning job to help you adjust your parameters. Find these files in the Amazon S3 bucket to which you wrote the output, or use the GetCustomModel operation.

Size of the input training dataset

In general, the larger the training dataset, the better the performance for a specific task. However, if there are too many samples, the model might perform worse on a different task (for example, if the training dataset for a summarization task contains 100,000 samples, the model might perform worse on a classification task).

Large language models perform better than small supervised models when there is limited training data (for example, 1 to 100 samples).

Model size

In general, the larger the model, the better the task performs given limited training data.

If you are using the model for a *classification* task, you might see relatively small gains for few-shot fine-tuning (less than 100 samples), especially if the number of classes is relatively small (less than 100).

Epochs

We recommend using the following metrics to determine the number of epochs to set:

1. **Validation output accuracy** – Set the number of epochs to one that yields a high accuracy.
2. **Training and validation loss** – Determine the number of epochs after which the training and validation loss becomes stable. This corresponds to when the model converges. Find the training loss values in the `step_wise_training_metrics.csv` and `validation_metrics.csv` files.

Batch size

When you change the batch size, we recommend that you change the learning rate using the following formula:

```
newLearningRate = oldLearningRate x newBatchSize / oldBatchSize
```

Learning rate

In general, use smaller learning rates for larger models. We recommend using a learning rate in the range of 1.00E-06 to 1.00E-05. This parameter doesn't appear to play a big role in question-answer and classification tasks. However, it might dramatically impact performance in summarization tasks.

The following table shows recommended learning rate values for few-shot fine-tuning:

Task	Minimum learning rate	Default learning rate	Max learning rate
Summarization	1.00E-06	3.00E-06	5.00E-05
Classification	5.00E-06	5.00E-05	5.00E-05
Question-answer	5.00E-06	5.00E-06	5.00E-05

Learning warmup steps

We recommend the default value of 0.

Troubleshooting

This section summarizes errors that you might run into in fine-tuning and what to check if you come across them.

Permissions issues

If you encounter an issue with permissions to access an Amazon S3 bucket, check that the following are true:

1. If the Amazon S3 bucket uses a CM-KMS key for Server Side encryption, ensure that the IAM role passed to Amazon Bedrock has `kms:Decrypt` permissions for the AWS KMS key. For example, see Allow a user to enccrypt and decrypt with any AWS KMS key in a specific AWS account.
2. The Amazon S3 bucket is in the same region as the Amazon Bedrock model customization job.
3. The IAM role trust policy includes the service SP (`bedrock.amazonaws.com`).

The following messages indicate issues with permissions to access training or validation data in an Amazon S3 bucket:

```
Could not validate GetObject permissions to access Amazon S3 bucket: training-data-bucket
 at key train.jsonl
Could not validate GetObject permissions to access Amazon S3 bucket: validation-data-bucket
 at key validation.jsonl
```

If you encounter one of the above errors, check that the IAM role passed to the service has `s3:GetObject` and `s3:ListBucket` permissions for the training and validation dataset Amazon S3 URIs. For example, see Submit a job (p. 46).

The following message indicates issues with permissions to write the output data in an Amazon S3 bucket:

```
Amazon S3 perms missing (PutObject): Could not validate PutObject permissions to access S3
 bucket: bedrock-output-bucket at key output/.write_access_check_file.tmp
```

If you encounter the above error, check that the IAM role passed to the service has `s3:PutObject` permissions for the output data Amazon S3 URI. For example, see Submit a job (p. 46).

Data issues

The following errors are related to issues with the training, validation, or output data files:

Invalid file format

```
Unable to parse Amazon S3 file: fileName.jsonl. Data files must conform to JSONL format.
```

If you encounter the above error, check that the following are true:

1. Each line is in JSON.
2. Each JSON has two keys, an *input* and an *output*, and each key is a string. For example:

```
{
    "input": "this is my input",
    "output": "this is my output"
}
```

3. There are no additional new lines or empty lines.

Character quota exceeded

```
Input size exceeded in file fileName.jsonl for record starting with...
```

If you encounter an error beginning with the text above, ensure that the number of characters conforms to the character quota in Fine-tuning quotas (p. 140).

Token count exceeded

```
Maximum input token count 4097 exceeds limit of 4096
Maximum output token count 4097 exceeds limit of 4096
Max sum of input and output token length 4097 exceeds total limit of 4096
```

If you encounter the above error, ensure that the number of tokens conforms to the token quota in Fine-tuning quotas (p. 140).

Internal error

```
Encountered an unexpected error when processing the request, please try again
```

If you encounter the above error, there might be an issue with the service. Try the job again. If the issue persists, contact AWS Support.

Provisioned throughput

When you configure provisioned throughput for a model, you receive a level of throughput at a fixed cost.

You can use provisioned throughput with Amazon and third-party base models, and with customized models.

Provisioned throughput pricing varies depending on the model that you use and the level of commitment you choose. You receive a discounted rate when you commit to a longer period of time. For details about pricing for each model, see the Model providers page in the Amazon Bedrock console.

Your options for throughput for a model differ depending on whether you run inference on a base model or a custom model.

Pricing option	Base model	Custom model
Provisioned throughput, no commitment (hourly pricing)	Not available	Available (maximum 2 provisioned throughputs per account)
Provisioned throughput, 1 month commitment	Available	Available
Provisioned throughput, 6 month commitment	Available	Available

You specify provisioned throughput in Model Units (MU). A model unit delivers a specific throughput level for the specified model. The throughput level of a MU for a given Text model specifies the following:

- **The total number of input tokens per minute** – The number of input tokens that an MU can process across all requests within a span of one minute.
- **The total number of output tokens per minute** – The number of output tokens that an MU can generate across all requests within a span of one minute.

Model unit quotas depend on the level of commitment you specify for the provisioned throughput.

- For custom models with no commitment, a quota of one model unit is available for each provisioned throughput. You can create up to two provisioned throughputs per account.
- For base or custom models with commitment, there is a default quota of 0 model units. To request an increase, use the limit increase form.

Note
Provisioned throughput is currently available for the following models.

Model name	Model ID for provisioned throughput
Titan Text G1 - Express 8K	amazon.titan-text-express-v1:0:8k

Model name	Model ID for provisioned throughput
Titan Embeddings G1 - Text	amazon.titan-embed-text-v1:2:8k
Anthropic Claude V2 18K	anthropic.claude-v2:0:18k
Anthropic Claude V2 100K	anthropic.claude-v2:0:100k
Anthropic Claude Instant V1 100K	anthropic.claude-instant-v1:2:100K
Stable Diffusion XL 1.0	stability.stable-diffusion-xl-v0

Topics

Procedures

Provisioned throughput supports the following procedures.

Creating

When you create a provisioned throughput, the provisioned throughput starts in an interim state (`Creating`) while Bedrock creates the provisioned throughput resources. If the creation is successful, the provisioned throughput transitions to `InService` state, and you can start using it for inference.

If the creation fails, the provisioned throughput transitions to `Failed` state.

Updating

You can perform the following types of updates to a provisioned throughput:

- Change the name of the provisioned throughput.
- Specify a new custom model that uses the same base model as the current custom model.
- Specify a base model. It must be the base model of the current custom model.

When you update a provisioned throughput, it transitions to the `Updating` state while Bedrock performs the update. If the update is successful, the provisioned throughput transitions to `InService` state. During the update, you can run inference using the provisioned throughput without disrupting the on-going traffic from your end customers. If the update includes a new model, you may receive output from the old model until the update is fully deployed.

If the update fails, the provisioned throughput transitions to `Failed` state.

Deleting

Deleting a provisioned throughput is a synchronous operation. The provisioned throughput deletion takes effect immediately.

Running inference

You can run inference using a provisioned throughput that is in `InService` state.

If you request more throughput than is configured for the provisioned throughput, the request is throttled (you receive the throughput defined by the provisioned throughput).

Permissions

To add provisioned throughput to a base model or a model that was customized from a base model, you must have first requested access to the base model by following the steps at Model access (p. 11).

To add provisioned throughput to a custom model that is encrypted, your role must include permissions to carry out the `kms:Decrypt` action on the model. For an example, see Use a customer managed key during inference (p. 105).

Provisioned throughput console procedures

This section describes the console procedures for provisioned throughput.

Topics

View provisioned throughput summary

Use the summary page to review the status of each provisioned throughput. The **Overview** panel displays the number of provisioned throughput resources in each state. You can update or delete an active provisioned throughput, and you can create a new provisioned throughput.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, choose **Provisioned throughput** under **Deployments**.
3. From the **Provisioned throughput** table, you can review summary information about each provisioned throughput.

Purchase provisioned throughput

You can purchase provisioned throughput for a foundation or custom model.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left menu, choose **Provisioned throughput** under **Deployments**. Then select **Purchase provisioned throughput**. The console opens the **Purchase provisioned throughput** page.
3. Under **Provisioned throughput details**:

    a. Enter a name for the provisioned throughput.
    b. Select the model category and model for the provisioned throughput.

Note
Provisioned throughput is currently available for the following models.

Model name	Model ID for provisioned throughput
Titan Text G1 - Express 8K	amazon.titan-text-express-v1:0:8k
Titan Embeddings G1 - Text	amazon.titan-embed-text-v1:2:8k
Anthropic Claude V2 18K	anthropic.claude-v2:0:18k
Anthropic Claude V2 100K	anthropic.claude-v2:0:100k
Anthropic Claude Instant V1 100K	anthropic.claude-instant-v1:2:100K
Stable Diffusion XL 1.0	stability.stable-diffusion-xl-v0

c. (Optional) Under **Tags**, you can associate one or more tags with this provisioned throughput.

4. Under **Model units & commitment term**:

   a. Enter the desired number of model units.

   b. Choose the amount of time for which you want to commit to using the provisioned throughput. To opt out of commitment for a custom model, you must set the number of model units to 1 and then select **No commitment**. With this option, you pay an hourly amount for the allocated throughput until you delete it.

   Note
   To see limitations on model units and commitment term, see Provisioned throughput (p. 55).

5. Under **Estimated purchase summary**, review the estimated cost.
6. Choose **Purchase provisioned throughput**.
7. Review the note that appears and acknowledge the commitment duration and price by selecting the checkbox. Then choose **Confirm purchase**.

The console displays the overview page. For the provisioned throughput that you just purchased, the console displays it in the table, with the status set to **Creating**.

View details of a provisioned throughput

From the summary page, you can view the details for any of your provisioned throughput resources.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left menu, choose **Provisioned throughput**.
3. From the **Provisioned throughput** table, choose a provisioned throughput. The console opens the details page.
4. Under **Provisioned throughput overview**:
5. Under **Tags**, the console displays the tags that are associated with this provisioned throughput. Choose **Manage tags** to add or remove tags for this provisioned throughput.

Edit a provisioned throughput

You can edit only a few fields of a provisioned throughput.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left menu, choose **Provisioned throughput**.
3. From the **Provisioned throughput** table, select the provisioned throughput to edit.
4. The console displays the provisioned throughput fields that you can edit.
5. Choose **Save** to start the update.

Delete a provisioned throughput

Note
You can't delete a provisioned throughput with commitment before the commitment term is complete.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left menu, choose **Provisioned throughput**.
3. From the **Provisioned throughput** table, select the provisioned throughput to delete.
4. The console displays a modal form to warn you that delete is permanent. Choose **Confirm** to proceed.

Using the provisioned throughput API

Amazon Bedrock provides API operations to create and manage your provisioned throughput resources.

Topics

Create provisioned throughput

Use the CreateProvisionedModelThroughput operation to create a provisioned throughput for a base or custom model.

When you send a `CreateProvisionedModelThroughput` operation for a base model, Bedrock validates your access permissions for the requested model and checks for capacity availability. If the validations are successful, Bedrock creates the provisioned throughput and returns the ARN of the provisioned throughput.

When you send a `CreateProvisionedModelThroughput` operation for a custom model, Bedrock validates your access permissions for the requested custom model. and checks for capacity availability. If the custom model is encrypted using a customer-managed KMS key, your permissions must include `kms:Decrypt` permission for this model.

Note
To see quotas for your options for `commitmentDuration` and `modelUnits`, see Provisioned throughput (p. 55).

To create a provisioned throughput for an Amazon Titan foundation model, use the following IDs instead of the default ones.

Note
Provisioned throughput is currently available for the following models.

Model name	Model ID for provisioned throughput
Titan Text G1 - Express 8K	amazon.titan-text-express-v1:0:8k
Titan Embeddings G1 - Text	amazon.titan-embed-text-v1:2:8k
Anthropic Claude V2 18K	anthropic.claude-v2:0:18k
Anthropic Claude V2 100K	anthropic.claude-v2:0:100k
Anthropic Claude Instant V1 100K	anthropic.claude-instant-v1:2:100K
Stable Diffusion XL 1.0	stability.stable-diffusion-xl-v0

The following code examples demonstrate how to create a provisioned throughput using the AWS CLI and the Python SDK.

AWS CLI

Create the provisioned throughput using the following command in the AWS CLI:

```
aws bedrock create-provisioned-model-throughput
   --model-units 1
   --commitment-duration SixMonths
   --provisioned-model-name test-model
   --model-id arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-v2
```

Python (Boto)

The following example demonstrates how to create the provisioned throughput using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')
bedrock.create_provisioned_model_throughput(
    modelUnits=1,
    commitmentDuration='SixMonths',
    provisionedModelId='test-model',
    modelId='arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-v2'
   )
```

Run inference using provisioned throughput

Use the InvokeModel or InvokeModelWithResponseStream operation to run inference using provisioned throughput. Specify the provisioned model ARN as the `modelId` parameter.

The following code examples show how to run inference for an Anthropic Claude model with a provisioned throughput using the AWS CLI and Python. For information about the prompt format and inference parameters, see Inference parameters for foundation models (p. 30).

AWS CLI

The following example demonstrates how to run inference for an Anthropic Claude model with a provisioned throughput in the AWS CLI.

```
aws bedrock-runtime invoke-model
    --model-id provisioned-model-arn \
    --body "{\"prompt\": \"\n\nHuman:story of two dogs\n\nAssistant:\"}" invoke-model-
output.txt
```

Python (Boto)

The following example demonstrates how to run inference for an Anthropic Claude model with a provisioned throughput using Python:

```
import boto3
import json

bedrock = boto3.client(service_name='bedrock-runtime')

body = json.dumps({"prompt": "\n\nHuman:explain black holes to 8th graders\n
\nAssistant:"})
modelId = 'provisioned-model-arn'
accept = 'application/json'
contentType = 'application/json'

response = bedrock.invoke_model(body=body, modelId=modelId, accept=accept,
 contentType=contentType)
response_body = json.loads(response.get('body').read())

# text
print(response_body.get('results')[0].get('outputText'))

# embedding
print(response_body.get('embedding'))
```

Update provisioned throughput

Use the UpdateProvisionedModelThroughput operation to update the specified provisioned throughput.

The following code examples demonstrate how to update a provisioned throughput using the AWS CLI and Python.

AWS CLI

Update the provisioned throughput using the following command in the AWS CLI:

```
aws bedrock update-provisioned-model-throughput
--provisioned-model-id provisioned-model-arn | provisioned-model-name
--model-arn custom-model-arn | foundation-model-arn
```

Python (Boto)

The following example demonstrates how to update the provisioned throughput using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.update_provisioned_model_throughput(
    provisionedModelId='provisioned-model-arn | provisioned-model-name'
    modelArn='custom-model-arn | foundation-model-arn'
)
```

Get provisioned throughput

Use the GetProvisionedModelThroughput operation to retrieve informationabout the specified provisioned throughput.

The following code examples demonstrates how to retrieve information using the AWS CLI and Python.

AWS CLI

Retrieve information about the provisioned throughput using the following command in the AWS CLI:

```
aws bedrock get-provisioned-model-throughput
--provisioned-model-id provisioned-model-arn | provisioned-model-name
```

Python (Boto)

The following example demonstrates how to retrieve information about the provisioned throughput using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.get_provisioned_model_throughput(
    provisionedModelId='my-provisioned-model-arn | provisioned model name'
)
```

Delete provisioned throughput

Use the DeleteProvisionedModelThroughput operation to delete the specified provisioned throughput.

The following code examples demonstrate how to delete a provisioned throughput using the AWS CLI and Python.

AWS CLI

Delete the provisioned throughput using the following command in the AWS CLI:

```
aws bedrock delete-provisioned-model-throughput
  --provisioned-model-id provisioned-model-arn | provisioned-model-name
```

Python (Boto)

The following example demonstrates how to delete the provisioned throughput using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.delete_provisioned_model_throughput(
    provisionedModelId='my-provisioned-model-arn | provisioned model name'
)
```

List provisioned throughput resources

Use the ListProvisionedModelThroughputs operation to list the provisioned throughput resources that you have defined.

The following code examples demonstrate how to list the provisioned throughput resources using the AWS CLI and Python.

AWS CLI

List the provisioned throughput resources using the following command in the AWS CLI:

```
aws bedrock list-provisioned-model-throughputs
```

Python (Boto)

The following example demonstrates how to list the provisioned throughput resources using Python:

```
import boto3
bedrock = boto3.client(service_name='bedrock')

bedrock.list_provisioned_model_throughputs()
```

Agents for Amazon Bedrock

Note
Agents for Amazon Bedrock is in limited preview release. To request access, contact your AWS account manager.

Agents for Amazon Bedrock enables developers to configure an agent to complete actions based on organization data and user input. Agents orchestrate interactions between foundation models, data sources, software applications, and user conversations, and automatically call APIs to take actions. Developers can easily integrate the agents and accelerate delivery of generative AI applications saving weeks of development effort.

With Agents for Amazon Bedrock, you can automate tasks for your customers. For example, you can create an agent that helps customers process insurance claims or one that helps customers make travel reservations and answer questions related to these tasks. You don't have to worry about provisioning, managing infrastructure, or writing custom code. Agents for Amazon Bedrock manages monitoring, encryption, user permissions, and API invocation.

Agents for Amazon Bedrock can carry out the following tasks:

- Extend foundation models to understand user requests and break down the tasks it needs to perform into smaller steps.
- Collect additional information from a user through natural conversation.
- Take actions to fulfill a customer's request.
- Make API calls to your company systems to carry out actions.
- Augment performance and accuracy by using data sources that you provide to facilitate Retrieval-Augmented Generation (RAG).
- Carry out source attribution.

To take advantage of Agents for Amazon Bedrock, you carry out the following steps:

1. (Optional) Set up a vector database and then create a knowledge base to store your private data in that database. For more information, see Building a knowledge base (p. 65).
2. Create an agent for your use-case, add actions that it can carry out, and attach the knowledge base you created to augment its performance. For more information, see Building an agent (p. 77).
3. Test your agent in the console or through API calls and modify the configurations as necessary. For more information, see Test your agent (p. 84).
4. When you have sufficiently modified your agent and it is ready to be deployed to your application, create an alias to point to a version of your agent. For more information, see Deploying an agent: versioning and aliases (p. 87).
5. Set up your application to make API calls to your agent alias.

Topics

- Deploying an agent: versioning and aliases (p. 87)
- Using the API (p. 88)
- How Bedrock Agent works with IAM (p. 90)

Building a knowledge base

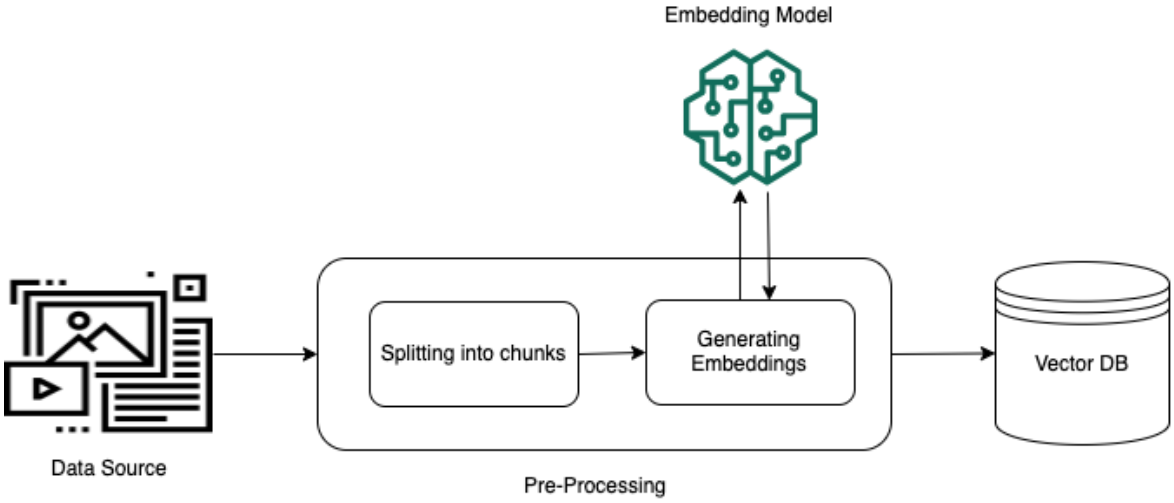
Note
Knowledge base is in limited preview release. To request access, contact your AWS account manager.

With Bedrock, you can enable a Retrieval-Augmented Generation (RAG) workflow by using knowledge bases to build contextual applications by using the reasoning capabilities of LLMs. RAG is a popular technique that combines the use of private data with Large Language Models (LLMs). The combination of Bedrock with knowledge bases enables a faster time to market by automating the RAG solution and reducing the build time for your agent. Adding a knowledge base also increases cost-effectiveness by removing the need to continually train your model to be able to leverage your private data.

RAG starts with an initial step to retrieve relevant documents from a data store (most commonly a vector index) based on the user's query. It then employs a language model to generate a response by considering both the retrieved documents and the original query. The following steps in the setup and implementation of RAG are automated for you by the knowledge base service.

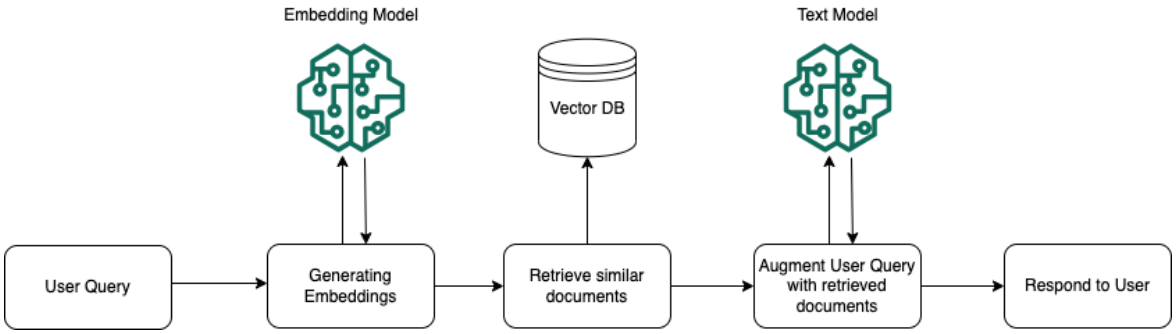
Pre-processing data

To enable effective retrieval from private data, a common practice is to first split the documents into manageable chunks. The following step is to convert the chunks to vectors and then to write them to a vector index while maintaining a mapping to the original document, in order to generate the embeddings. The following image illustrates pre-processing of data for the vector database.



Runtime execution

At runtime, an embedding model is used to convert the user's query to a vector. The vector index is then queried to find documents similar to the user's query by comparing document vectors to the user query vector. In the final step, semantically similar documents retrieved from the vector index are added as context for the original user query. When generating a response for the user, the semantically similar documents are prompted in the text model. The following image illustrates how RAG operates at runtime to augment responses to user queries.



You can build a RAG-based application in Bedrock by creating a knowledge base and associating it to an agent to augment its generative capabilities with your own data. You use a knowledge base to load your private data into a vector index. A knowledge base reads data from your Amazon S3 bucket, splits it into smaller chunks, generates vector embeddings, and stores the embeddings in a vector index that you provide. You can associate a knowledge base with multiple agents. After you configure an agent with a knowledge base, it can use the information stored in the corresponding vector index to augment its responses to user queries.

Topics

Create a service role and configure IAM permissions

Knowledge bases use service roles to access AWS resources (for more information, see Creating a role to delegate permissions to an AWS service).

Before you can create a knowledge base, you need to create a service role and attach a trust policy for the role you create.

To create the service role and attach a trust policy

1. Create an IAM role with the prefix `AmazonBedrockExecutionRoleForKnowledgeBase_`. For more information about creating a role, see Creating a role to delegate permissions to an AWS service.
2. Create a trust policy for the role you create. The following shows an example policy you can use. You can restrict the scope of the permission by using one or more global condition context keys. For more information, see AWS global condition context keys. Set the `aws:SourceAccount` value to your account ID. You can use the `ArnEquals` or `ArnLike` condition to restrict the scope to specific knowledge bases.

   Note
   As a best practice for security purposes, replace the * with specific knowledge base IDs after you have created them.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
```

```
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:knowledge-base/*"
            }
        }
    }]
}
```

3. Attach the trust policy to the role.

For a knowledge base, you need to grant Bedrock the following permissions:

- Access to Amazon Bedrock embedding models
- Access to the Amazon S3 object containing your data sources
- (If you encrypted your Amazon S3 data) Permissions to decrypt and encrypt your customer-managed AWS KMS key for your data sources
- (If you create a vector database in Amazon OpenSearch Service) Access to your OpenSearch Service collection
- (If you create a vector database in Pinecone or Redis Enterprise Cloud) Permissions for AWS Secrets Manager to authenticate your Pinecone or Redis Enterprise Cloud account

Topics

Permissions to access Amazon Bedrock models

To allow Bedrock to access Amazon Bedrock models to embed your source data, attach the following policy to your Bedrock service role. Replace *region* with your region your foundation model is in and *foundation-model-id* with the ID of the foundation model. Currently, only the Titan Embeddings G1 - Text Model *amazon.titan-embed-text-v1* is supported.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "bedrock:ListFoundationModels",
                "bedrock:ListCustomModels"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:region::foundation-model/foundation-model-id"
            ]
        }
    ]
}
```

Permissions to access your data sources in Amazon S3

To allow Bedrock to access your Amazon S3 data, attach the following policy attached to your Bedrock service role. Replace *bucket/path/to/folder* with the path to the object containing all the data source files for your knowledge base and *account-id* with the account that the Amazon S3 object belongs to.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::bucket/path/to/folder",
            "arn:aws:s3:::bucket/path/to/folder/*"
        ],
        "Condition": {
            "StringEquals": {
                "aws:PrincipalAccount": "account-id"
            }
        }
    }]
}
```

(Optional) Permissions to decrypt your AWS KMS key for your data sources in Amazon S3

If you encrypted your data sources in Amazon S3 with a AWS KMS key, attach the following policy to your Bedrock service role to allow Bedrock to decrypt your key. Replace *region* and *account-id* with the region and account ID to which the key belongs. Replace *key-id* with the ID of your AWS KMS key.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "KMS:Decrypt",
        ],
        "Resource": [
            "arn:aws:kms:region:account-id:key/key-id"
        ],
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.region.amazonaws.com"
                ]
            }
```

```
        }
    }]
}
```

(Optional) Permissions to access your vector database in Amazon OpenSearch Service

If you created a vector database in Amazon OpenSearch Service for your knowledge base, attach the following policy to your Bedrock service role to allow access to the collection. Replace *region* and *account-id* with the region and account ID to which the database belongs. Input the ID of your Amazon OpenSearch Service collection in *collection-id*. You can allow access to multiple collections by adding them to the `Resources` list.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "aoss:APIAccessAll"
        ],
        "Resource": [
            "arn:aws:aoss:region:account-id:collection/collection-id"
        ]
    }]
}
```

(Optional) Permissions to access your vector database in Pinecone or Redis Enterprise Cloud

If you created a vector database in Pinecone or Redis Enterprise Cloud for your knowledge base, attach the following policy to your Bedrock service role to allow AWS Secrets Manager to authenticate your account to access the database. Replace *region* and *account-id* with the region and account ID to which the database belongs. Replace *secret-id* with the ID of your secret.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue"
        ],
        "Resource": [
            "arn:aws:secretsmanager:region:account-id:secret:secret-id"
        ]
    }]
}
```

If your secret is encrypted with a AWS KMS key, attach the following policy to your Bedrock service role to allow it to decrypt your key. Replace *region* and *account-id* with the region and account ID to which the key belongs. Replace *key-id* with the ID of your AWS KMS key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
```

```
            ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/keyId"
            ]
        }
    ]
}
```

(Optional) Permissions for AWS to manage a AWS KMS key for transient data storage during data ingestion

To allow the creation of a AWS KMS key for transient data storage in the process of ingesting your data source, attach the following policy to your Bedrock service role. Replace the *region*, *account-id*, and *key-id* with the appropriate values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "kms:GenerateDataKey",
              "kms:Decrypt"
          ],
          "Resource": [
              "arn:aws:kms:region:account-id:key/key-id"
          ]
        }
    ]
}
```

Set up your data for ingestion

Before you can create a knowledge base, you need to carry out the following actions:

1. Configure your source data and upload them to an Amazon S3 bucket.
2. Create a vector database and configure fields for Amazon Bedrock to write data to and to access the index.

Configure your source data

By default, knowledge base automatically splits your source data into chunks, such that each chunk contains, at most, 200 tokens. If a document contains less than 200 tokens, then it is not split any further. To split your documents into chunks containing less than 200 tokens, pre-process the documents by splitting them into smaller chunks.

After you pre-process your source data, upload it to an Amazon S3 bucket. To set up an Amazon S3 bucket, see Getting started with Amazon S3. Knowledge base supports the following file formats:

- Plain text (.txt)
- Markdown (.md)
- HyperText Markup Language (.html)
- Microsoft Word document (.doc/.docx)
- Comma-separated values (.csv)
- Microsoft Excel spreadsheet (.xls/.xlsx)
- Portable Document Format (.pdf)

Create a vector database

Create a vector index in one of the following supported options.

- Amazon OpenSearch Service
- Pinecone
- Redis Enterprise Cloud

After processing your data, Amazon Bedrock writes the following information to the index you created.

- Text extracted from your documents.
- The vectors, corresponding to your text, that were generated by the embeddings model.
- The Amazon S3 path of the source file where the text was extracted from.

Topics

Create a vector index in Amazon OpenSearch Service

1. Log into Amazon OpenSearch Service and create a collection. Take note of the **Collection ARN**, which you will fill out when you create a knowledge base.
2. Once the collection is created, select it and create a vector index.

For detailed documentation on setting up a vector index in Amazon OpenSearch Service, see Working with vector search collections.

While you set up the vector index, take note of the **Collection ARN**, which you will fill out when you create a knowledge base.

There are additional configurations that you must provide when creating a vector index:

- **Vector index name** – The name of the vector index. Choose any valid name of your choice. Later, when you create your knowledge base, enter the name you choose in the **Vector index name** field.
- **Vector field** – The field where the vector embeddings will be stored. Choose any valid name of your choice. Later, when you create your knowledge base, enter the name you choose in the **Vector field** field.
- **Dimensions** – The number of dimensions in the vector. Choose 1536 if you use the Titan Embeddings Model. Later, when you create your knowledge base, enter this number for the **Dimensions** field.
- **Distance metric** – The metric used to measure the similarity between vectors.. We recommend that you experiment with different metrics for your use-case. If you use the Titan Embeddings Model, you can start with **cosine similarity**.
- **Metadata management** – Expand this field and configure the vector index to store additional metadata that a knowledge base can retrieve with vectors. The fields you need to configure are as follows:
  - **Text field** – Amazon Bedrock chunks the raw text in your data and stores the chunks in this field.
    - **Mapping field** – The field where the text will be stored. Choose any valid name of your choice. Later, when you create your knowledge base, enter the name you choose for the **Text field name** field.

    - **Data type** – Select `String`.
    - **Filterable** – Select `False`.
- **Bedrock-managed metadata field** – Amazon Bedrock stores metadata related to the data in this field. The metadata includes the following:
    - **Mapping field** – Choose any valid name of your choice. Later, when you create your knowledge base, enter the name you choose for the **Bedrock-managed metadata field name** field.
    - **Data type** – Select `String`.
    - **Filterable** – Select `False`.

Security configurations

After you create the knowledge base, you must return to Amazon OpenSearch Service and set up security configurations in your vector database by adjusting the **Network access** and **Data access** settings of your collection. For more information, see Create a knowledge base (p. 73).

Create a vector index in Pinecone

> **Note**
> If you use Pinecone, you agree to authorize AWS to access the designated third-party source on your behalf in order to provide the Bedrock service to you. You're responsible for complying with any third-party terms applicable to use and and transfer of data from the third-party service.

For detailed documentation on setting up a vector index in Pinecone, see Manage indexes.

While you set up the vector index, take note of the following information, which you will fill out when you create a knowledge base:

- The endpoint URL for your index management page.
- (Optional) The namespace to be used to write new data to your database. For more information, see Using namespaces.

There are additional configurations that you must provide when creating a Pinecone index:

- **Name** – The name of the vector index. Choose any valid name of your choice. Later, when you create your knowledge base, enter the name you choose in the **Vector index name** field.
- **Dimensions** – The number of dimensions in the vector. Choose 1536 if you use the Titan Embeddings Model. Later, when you create your knowledge base, enter this number in the **Dimensions** field.
- **Distance metric** – The metric used to measure the similarity between vectors. We recommend that you experiment with different metrics for your use-case. If you use the Titan Embeddings Model, you can start with **cosine similarity**.

Configure the Secrets Manager

To access your Pinecone index, you must provide your Pinecone API key to Amazon Bedrock through the AWS Secrets Manager.

To set up a secret for your Pinecone configuration

1. Follow the steps at Create an AWS Secrets Manager secret, setting the key as `apiKey` and the value as the API key to access your Pinecone index.
2. To find your API key, open your Pinecone console and select **API Keys**.
3. After you create the secret, take note of its ARN. Later, when you create your knowledge base, enter the ARN in the **Credentials secret ARN** field.

Create a vector index in Redis Enterprise Cloud

Note
If you use Redis Enterprise Cloud, you agree to authorize AWS to access the designated third-party source on your behalf in order to provide the Bedrock service to you. You're responsible for complying with any third-party terms applicable to use and and transfer of data from the third-party service.

For detailed documentation on setting up a vector index in Redis Enterprise Cloud, see Integrating Redis Enterprise Cloud with Amazon Bedrock.

While you set up the vector index, take note of the following information, which you will fill out when you create a knowledge base:

- The public endpoint URL for your database.
- The name of the vector index for your database.

Configure the Secrets Manager

To access your Redis Enterprise Cloud cluster, you must provide your Redis Enterprise Cloud security configuration to Amazon Bedrock through the AWS Secrets Manager.

To set up a secret for your Redis Enterprise Cloud configuration

1. Enable TLS to use your database with Amazon Bedrock by following the steps at Transport Layer Security (TLS).
2. Follow the steps at Create an AWS Secrets Manager secret. Set up the following keys with the appropriate values from your Redis Enterprise Cloud configuration in the secret:
   - `username` – The username to access your Redis Enterprise Cloud database. To find your username, look under the **Security** section of your database in the Redis Console.
   - `password` – The password to access your Redis Enterprise Cloud database. To find your password, look under the **Security** section of your database in the Redis Console.
   - `serverCertificate` – The content of the certificate from the Redis Cloud Certificate authority. Download the server certificate from the Redis Admin Console by following the steps at Download certificates.
   - `clientPrivateKey` – The private key of the certificate from the Redis Cloud Certificate authority. Download the server certificate from the Redis Admin Console by following the steps at Download certificates.
   - `clientCertificate` – The public key of the certificate from the Redis Cloud Certificate authority. Download the server certificate from the Redis Admin Console by following the steps at Download certificates.
3. After you create the secret, take note of its ARN. Later, when you create your knowledge base, enter the ARN in the **Credentials secret ARN** field.

Create a knowledge base

To create a knowledge base

1. Open the Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select **Knowledge base**.
3. In the **Knowledge base** section, select **Create knowledge base**.
4. On the **Provide knowledge base details** page, carry out the following actions:

a. In the **Knowledge base details** section, enter a name for the knowledge base and provide an optional description for it.

b. In the **IAM permissions** section, choose to let Amazon Bedrock create a new service role or to use an existing service role that allows Amazon Bedrock to access other services on your behalf. For more information, see Create a service role and configure IAM permissions (p. 66).

c. If you want to attach any tags to the knowledge base, select **Add new tag** In the **tags** section and add the tags as key-value pairs.

d. Select **Next**.

5. On the **Set up data source** page, you provide the information for the data source to add to the knowledge base by carrying out the following actions:

   a. In the **Data source** section, carry out the following actions:

      i. Provide a name for the data source and the URI of the Amazon S3 object.

      ii. If you encrypted your Amazon S3 data, provide the AWS KMS key in the **Customer-managed AWS KMS key for Amazon S3 data** to allow Amazon Bedrock to decrypt it.

      iii. While converting your data into embeddings, Amazon Bedrock encrypts your transient data with a key that AWS owns and manages, by default. You can select the checkbox labeled **Customize encryption settings (advanced)** under **AWS KMS key for transient data storage**.

   b. In the **Embeddings model** section, choose an embeddings model to convert the knowledge base from your data into an embedding. Currently, only the Amazon Bedrock Titan embeddings model is available.

   c. In the **Vector database** section, select the service that contains a vector database that you have already created. Check that your database is already configured with the required fields (for more information, see Set up your data for ingestion (p. 70)). Fill in the fields to allow Amazon Bedrock to map information from the knowledge base to your database, so that it can store, update, and manage embeddings.

      Note
      If you use a database in Amazon OpenSearch Service, you need to have configured the fields under **Metadata field mapping** beforehand. If you use a database in Pinecone or Redis Enterprise Cloud, you can provide names for these fields here and Amazon Bedrock will dynamically create them in the vector index for you.

   d. Select **Next**.

6. On the **Review and create** page, check the configuration and details of your knowledge base. Select **Edit** in any section that you need to modify. When you are satisfied, select **Create knowledge base**.

7. The knowledge base creation process begins and the **Status** of the source becomes **In progress**. The time it takes to create the knowledge base depends on the amount of data you provided. When the knowledge base is finished being created, a green success banner appears and the **Status** of the knowledge base changes to **Ready**.

   Note
   If you chose to store your embeddings in an Amazon OpenSearch Service vector database, remember to set up your security configurations in OpenSearch Service for the knowledge base after it has been created. For more information, see Security configurations (p. 72).

Set up security configurations for your newly created knowledge base. Follow the steps in the tab corresponding to the database that you set up.

OpenSearch Service

To create a data access policy

1. In the OpenSearch Service console, navigate to your collection.
2. Select **Manage data access**.
3. Select **Create access policy** and give the policy a name and an optional description.
4. Choose **JSON** as the policy definition method and paste the following JSON object into the editor, replacing *collection-name* with the name of your collection and *service-role-arn* with the role ARN that you passed when creating your knowledge base.

```
{[
    {
        "Rules": [
          {
            "Resource": [
              "index/collection_name/*"
            ],
            "Permission": [
              "aoss:DescribeIndex",
              "aoss:ReadDocument",
              "aoss:WriteDocument"
            ],
            "ResourceType": "index"
          }
        ],
        "Principal": [
          "service-role-arn"
        ],
        "Description": "Data access policy"
    }
]
```

5. Select **Create** to create the policy.

Pinecone or Redis Enterprise Cloud

To integrate the third-party knowledge base, attach the following policy to your Bedrock service role, replacing *knowledge-base-arn*.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "bedrock:AssociateThirdPartyKnowledgeBase"
        ],
        "Resource": [
            "knowledge-base-arn"
        ],
        "Condition": {
            "StringEquals": {
                "bedrock:ThirdPartyKnowledgeBaseCredentialsSecretArn": "secret-arn"
            }
        }
    }]
}
```

Manage a knowledge base

To manage a knowledge base

1. Open the Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select **Knowledge base**.
3. To view details for a knowledge base, either select the **Name** of the source or choose the radio button next to the source and select **Edit**.
4. On the details page, you can carry out the following actions:
   - To change the details of the knowledge base, select **Edit** in the **Knowledge base overview** section.
   - To update the tags attached to the knowledge base, select **Manage tags** in the **Tags** section.
   - If you update the data source from which the knowledge base was created and need to sync the changes, select **Sync** in the **Data source** section.
   - To view the details of a data source, select a **Data source name**. Within the details, you can choose the radio button next to a sync event in the **Sync history** section and select **View warnings** to see why files in the data ingestion job failed to sync.
   - To manage the embeddings model used for the knowledge base, select **Edit provisioned throughput**.
   - Select **Save changes** when you are finished editing.

To delete a knowledge base

1. Open the Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select **Knowledge base**.
3. To delete a source, either choose the radio button next to the source and select **Delete** or choose the **Name** of the source and then select **Delete** in the top right corner of the details page.
4. Review the warnings for deleting a knowledge base. If you accept these conditions, enter **`delete`** in the input box and select **Delete** to confirm.
5. Make sure to delete the knowledge base from any agents that it was added to. To do this, carry out the following steps:
   a. From the left navigation pane, select **Agents**.
   b. Choose the **Name** of the agent that you want to delete the knowledge base from.
   c. A red banner appears to warn you to delete the reference to the knowledge base, which no longer exists, from the agent.
   d. Select the radio button next to the knowledge base that you want to remove. Select **More** and then choose **Delete**.

Add a knowledge base to an agent

To add a knowledge base to an agent

1. Open the Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select **Agents**.
3. Choose the **Name** of the agent that you want to add knowledge bases to.
4. In the **Knowledge base** section, select **Add**.
5. Choose the knowledge base from the dropdown list under **Select knowledge base** and specify the instructions for the agent regarding the knowledge base.

Building an agent

Note
Bedrock is in limited preview release. To request access, contact your AWS account manager.

To build an agent, you set up the following components:

- The configuration of the agent itself, which defines the purpose of the agent.
- Action groups that define what actions the agent is designed to carry out.
- (Optional) A knowledge base of data sources to augment the generative capabilities of the agent.

  Note
  If you plan to attach a knowledge base to your agent, first set up your knowledge base by following the steps at Building a knowledge base (p. 65).

You carry out the following steps to set up an agent to interact with your customers:

1. Choose a foundational model that the agent can use for orchestration.
2. Give instructions to the agent, describe the actions it can perform, and provide an API schema for the actions.
3. Add private data to the agent by setting up your data sources in a knowledge base and associating the knowledge base with your agent.
4. Test the agent and iterate on the working draft.
5. After the agent is working as expected, create an alias so that you can integrate the agent with your application.
6. Configure your application to make API calls to the agent alias.
7. Update the working draft of the agent and create new aliases as necessary.

Topics

Create a service role and configure IAM permissions

Before you can create an agent, you need to create a service role and attach a trust policy for the role you create.

To create the service role and attach a trust policy

1. Create an IAM role with the prefix `AmazonBedrockExecutionRoleForAgents_`. For more information about creating a role, see Creating a role to delegate permissions to an AWS service.
2. Create a trust policy for the role you create. The following shows an example policy you can use. You can restrict the scope of the permission by using one or more global condition context keys. For more information, see AWS global condition context keys. Set the `aws:SourceAccount` value to your account ID. You can use the `ArnEquals` or `ArnLike` condition to restrict the scope to specific agents.

   Note
   As a best practice for security purposes, replace the * with specific agent IDs after you have created them.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:agent/*"
            }
        }
    }]
}
```

3. Attach the trust policy to the role.

Grant Bedrock permissions to access the following resources on your behalf through configuring role-based permissions:

- The Amazon Bedrock base models
- The Amazon S3 objects containing the OpenAPI schemas for the action groups in your agents

You also need to provide permissions for Bedrock to access the AWS Lambda functions for the action groups in your agents through configuring resource-based policies for your Lambda functions.

If you are going to attach a knowledge base to your agent, you also need to provide permissions for Amazon Bedrock to query the knowledge base.

Topics

Permissions to access Amazon Bedrock base models

To provide permissions for Bedrock to use the Anthropic Claude v1 model for orchestration, attach the following policy to an IAM role.

```
{
    "Version": "2012-10-17",
    "Statement": [{
       "Effect": "Allow",
       "Action": [
        "bedrock:InvokeModel"
       ],
    "Resource": [
            "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-v1",

        ]
    }]
```

```
}
```

Permissions to access your action group API schemas in Amazon S3

Provide permissions for Bedrock to access the Amazon S3 URIs of the API schemas for your agent's action groups. Attach the following policy to an IAM role. In the `Resource` field, you can provide an Amazon S3 object containing the schemas or you can add the URI of each schema to the list.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::path/to/schema"
        ],
        "Condition": {
            "StringEquals": {
                "aws:PrincipalAccount": "account-id"
            }
        }
    }]
}
```

Permissions to access your action group Lambda functions

Provide permissions for Bedrock to access the Lambda functions for your agent's action groups. Attach the following resource-based policy to a Lambda function. The `Condition` specifies the ARN of the user account and the ARN of the agent. The `Condition` field is optional, but is recommended for security purposes. For more information, see Using resource-based policies for Lambda.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "statement-name",
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": [
            "lambda:InvokeFunction",
        ],
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "account-id"
            },
            "ArnLike": {
                "AWS:SourceArn": "arn:aws:bedrock:region:account-id:agent/agent-id"
            }
        }
    }]
}
```

(Optional) Permissions to access your knowledge bases

To provide permissions for Bedrock to access a knowledge base you have set up, attach the following policy to an IAM role. Replace *account-id* and *knowledge-base-id* with the appropriate values and *statement-name* with the statement name of your choice.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement-name",
            "Effect": "Allow",
            "Action": [
                "bedrock:QueryKnowledgeBase"
            ],
            "Resource": [
                "arn:aws:bedrock:*:account-id:knowledge-base/knowledge-base-id"
            ]
        }
    ]
}
```

Create an agent

To create an agent, https://console.aws.amazon.com/bedrock/ Choose **Agents** in the left navigation pane. Then select **Create** at the top right corner of the **Agents** section.

Provide agent details

1. In the **Agent name** section, give the agent a name and an optional description.
2. In the **User input** section, select whether you want the agent to request the user for additional information when trying to complete a task. If you select **No**, the agent doesn't request the user for additional details and informs the user that it doesn't have enough information to complete the task.
3. In the **IAM permissions** section, choose an AWS Identity and Access Management (IAM) role that provides Amazon Bedrock permission to access other AWS services. See Create a service role and configure IAM permissions (p. 77) for more information.
4. In the **Idle session timeout** section, choose the duration that Amazon Bedrock keeps a session with a user open. Amazon Bedrock maintains prompt variables for the duration of the session so that your agent can resume an interaction with the same variables.
5. Select **Next** when you are done setting up the agent configuration.

Select model

1. For the agent to perform orchestration, you need to choose a model. Choose a model provider and then choose a model in the dropdown menu to train your agent.
2. In **Instructions for the Agent**, provide details to tell the agent what it should do and how it should interact with users. Refer to the following sample instructions:

```
You are an office assistant in an insurance agency. You are friendly and polite. You
 help with managing insurance claims and coordinating pending paperwork.
```

Add Action groups

1. Provide a name for your action group and describe what the action does in the **Description for action groups**. For example, you can use the following text for an action group called *SetupDNSResources*:

```
Use this Action whenever you want to setup DNS resources for setting up a website.
```

2. In **Select Lambda function**, choose a Lambda function that you created in AWS Lambda. The Lambda function provides the business logic that is carried out upon invoking the action. Choose the version of the function to use.

   Note
   Remember to configure a policy to allow the Amazon Bedrock service principal to access the Lambda function. For more information, see Create a service role and configure IAM permissions (p. 77) for more information.. You can optionally provide the ARN of the agent as the `SourceArn` when configuring the policy.

   The following is the general format of the Lambda input event. Use the input event fields to create your Lambda function. For more information, see Event-driven invocation in the AWS Lambda documentation.

```
{
    "messageVersion": "1.0",
    "agent": {
        "name": "string",
        "id": "string",
        "alias": "string",
        "version": "string"
    },
    "inputText": "string",
    "sessionId": "string",
    "actionGroups": [
        {
            "actionGroup": "string",
            "apiPath": "string",
            "httpMethod": "string",
            "parameters": [
                {
                    "name": "string",
                    "type": "string",
                    "value": "string"
                },
                ...
            ],
            "requestBody": {
                "content": {
                    "<content_type>": {
                        "properties": [
                            {
                                "name": "string",
                                "type": "string",
                                "value": "string"
                            },
                            ...
                        ]
                    }
                }
            }
        },
        ...
    ],
    "sessionAttributes": {
```

```
        "string": "string",
    }
}
```

Amazon Bedrock expects a response from your Lambda function in the following format.

```
{
    "messageVersion": "1.0",
    "response": [
        {
            "actionGroup": "string",
            "apiPath": "string",
            "httpMethod": "string",
            "httpStatusCode": number,
            "responseBody": {
                "<contentType>": {
                    "body": "string"
                }
            }
        }
    ]
}
```

3. In **Select API schema**, provide a link to the Amazon S3 URI of the schema with the API description, structure, and parameters for the action group. For examples of the schema, see https://github.com/OAI/OpenAPI-Specification/tree/main/examples/v3.0.

   Note
   Remember to assign the IAM role permissions to access the Amazon S3 URI of the schema. For more information, see See Create a service role and configure IAM permissions (p. 77) for more information.

4. Select **Add another Action group** to set up another action group for your agent. When you are done adding action groups, select **Next**.

Add knowledge bases

1. If you have not yet created any knowledge bases, select **Knowledge base** in the left navigation pane and follow the instructions at Create a knowledge base (p. 73) to create one. Otherwise, select a knowledge base from the dropdown menu.
2. Write a prompt in **Knowledge base instructions for the agent** to describe how the agent should use the knowledge base. For example, you can use the following text for a knowledge base called *Domain name system details*:

```
Use this knowledge base whenever you are creating a DNS record
```

3. Select **Add another knowledge base** to set up another knowledge base for your agent. When you are done adding knowledge bases, select **Next**.

Review the configuration of your agent and choose **Edit** for any sections you want to change. Select **Create** when you are ready to create the agent. When the process finishes, a green banner appears at the top to inform you that the agent was successfully created.

Edit your agent

After you create an agent, you can update its configuration as required. The configuration applies to the working draft.

To edit the agent configuration

1. Choose an agent in the **Agents** section.
2. Select **Edit** in the **Agent overview** section.
3. Edit the existing fields as necessary.
4. Select **Save changes**. A green success banner appears at the top.

You might want to try different foundation models for your agent or change the instructions for the agent. These changes apply only to the working draft.

To change the foundation model that your agent uses or the instructions to the agent.

1. Choose an agent in the **Agents** section.
2. Select **View** in the **Working draft** section or choose the working draft.
3. In the **Model details** section, select **Edit**
4. Edit the fields as necessary.
5. Select **Save** to remain in the same window. A green success banner appears at the top.
6. Test the updated agent in the right panel and make changes as necessary.
7. Select **Save and exit** to return to the working draft page.

Manage the action groups of an agent

After creating an agent, you can add more action groups or edit them. Adding and editing take place within the working draft. To carry out these operations, choose an agent from the **Agents** section and then choose the **Working draft** in the **Working Draft** section.

To add an action group

1. Select **Add** in the **Action groups** section.
2. Fill out the action group details
3. Select **Add**. A green success banner appears at the top.

To edit an action group

1. Do one of the following:
   - Choose the radio button next to the action group to edit and select **Edit**.
   - Select an action group to see its details. Choose **Edit** at the top.
2. Edit the existing fields as necessary.
3. Select **Save** to remain in the same window. A green success banner appears if there are no issues and a red banner appears if there are errors in the edit.
4. Test the updated agent in the right panel and make changes as necessary.
5. Select **Save and exit** to return to the working draft page.

Manage the knowledge bases of an agent

After creating an agent, you can add more knowledge bases or edit them. Adding and editing take place within the working draft. To carry out these operations, choose an agent from the **Agents** section and then choose the **Working draft** in the **Working Draft** section.

To add a knowledge base

1. Select **Add** in the **Knowledge bases** section.
2. Choose a knowledge base that you have created and provide instructions for how the agent should interact with it.
3. Select **Add**. A green success banner appears at the top.

To edit a knowledge base

1. Select a knowledge base.
2. In the knowledge base details page, select **Edit**
3. Edit the existing fields as necessary.
4. Select **Save** to remain in the same window. A green success banner appears if there are no issues and a red banner appears if there are errors in the edit.
5. Test the updated agent in the right panel and make changes as necessary.
6. Select **Save and exit** to return to the working draft page.

To delete a knowledge base

1. Select the radio button next to the knowledge base to delete.
2. Select **Delete**.

Test your agent

Once you have created an agent, you can find it in the **Agents** section. When you first create an agent, you will have a *working draft*. The working draft is a version of the agent that you can use to iteratively build the agent. By default, you can interact with the working draft with the `AgentTestAlias`, You can also select a different alias to test. In the test window, you can opt to show the trace for each response. The trace shows the agent's reasoning process, step-by-step, and is a useful tool for debugging your agent. To learn more about the trace, see Trace enablement (p. 85). To learn how to enable the trace through the API, see Invoke your agent (p. 89).

You have access to a test window to interact with your agent. You can use this function to debug your agent and make the necessary changes for it to work as expected.

To test the agent

1. Choose an agent in the **Agents** section.
2. In any of the pages inside an agent, you can select the left arrow icon at the top right to expand the test window.
3. Use the dropdown menu at the top of the test window to choose an alias and associated version to test.
4. Enter a message to request the agent to perform a task. Use the test window to help debug your agent.
5. When viewing a response, you have the following options.

   - If the agent pulls information from a source, the response contains footnotes. Select a footnote to view the citation for that part of the response.
   - Under the response from the bot, select **Show trace** to view the agent's reasoning process and usage of the attached knowledge bases and action groups, in addition to the **Inference configurations** used. The trace breaks down how the response was formed, step-by-step. Select

an arrow next to a step to expand or collapse the trace for that step. For more information, see Trace enablement (p. 85).

You have the option of turning action groups and knowledge bases on and off. Use this feature to debug your agent and to isolate which actions or knowledge bases need to be updated by assessing its behavior with different settings.

To turn an action group or knowledge base on or off

1. Choose an agent in the **Agents** section.
2. Select the working draft in the **Working draft**.
3. In the **Action groups** or **Knowledge base** section, hover over the **State** of the action group whose state you want to change.
4. An edit button appears. Select it and then choose from the dropdown menu whether the action group or knowledge base is **Enabled** or **Disabled**.
5. If an action group is **Disabled**, the agent doesn't try to carry it out. If a knowledge base is **Disabled**, the agent doesn't use it in orchestration. Turn action groups on and off and use the test window to debug your agent.

Trace enablement

Each response from an agent is accompanied by a *trace* that you can view when you test the agent. The trace helps you follow the agent's reasoning process that leads it to the response it gives at that point in the conversation.

In the console test window, the trace is enabled by default. When you use an API, the trace is disabled by default unless you set the `enableTrace` to `true`. For more information about enabling the trace through the API, see Invoke your agent (p. 89).

Use the trace to track the agent's path from the user input to the response it returns. With the trace, you can find information about the inputs to the action groups that the agent invokes and the knowledge bases that it looks up to help it respond to the user, in addition to the outputs that the action groups and knowledge bases return. You can view the reasoning that the agent uses to determine the action that it takes or the query that it makes to a knowledge base. If a step in the trace fails, the trace returns a reason for the failure. Use the detailed information in the trace to debug your agent by identifying steps at which it has trouble or at which it yields unexpected behavior and ways in which you can improve the behavior.

When you show the trace in the test window in the console, a window appears showing a trace for each **Step** in the reasoning process. The **Step** consists of the following four types of traces. Each **Step** contains either a `failureTrace` or any combination of a `rationaleTrace`, `invocationInputTrace`, or `observationTrace`.

- `rationaleTrace` – Contains the reasoning, based on the user input, that the agent uses to justify carrying out an action group or getting information from a knowledge base.

```
{
    "traceId": "string",
    "text": "string"
}
```

The fields are described below.

- `traceId` – The unique identifier of the trace step. Any `rationaleTrace`, `invocationInput`, or `observationTrace` that is in the same step has the same `traceId`.

- `text` – The reasoning or thought process of the agent, based on the user input.

- `invocationInputTrace` – Contains information pertaining to the action group or knowledge base that is being invoked.

```
{
    "traceId": "string",
    "invocationType": "ACTION_GROUP | KNOWLEDGE_BASE",
    "actionGroupInvocationInput": {
        "actionGroupName": "string",
        "apiPath": "string",
        "parameters": [
            {
                "name": "string",
                "type": "string",
                "value": "string",
            },
            ...
        ],
        "requestBody": {
            "content": {
                "string": [
                    {
                        "name": "string",
                        "type": "string",
                        "value": "string",
                    },
                    ...
                ]
            }
        },
        "verb": "string"
    },
    "knowledgeBaseLookupInput": {
        "knowledgeBaseId": "string",
        "text": "string"
    }
}
```

  The fields are described below.

  - `traceId` – The unique identifier of the trace.
  - `invocationType` – Specifies whether the agent is invoking an action group or a knowledge base.
  - `actionGoupInvocationInput` – Contains the following metadata about the action group being invoked.
    - `actionGroupName` – The name of the action group that the agent is invoking.
    - `apiPath` – The path to the API to call, based off the action group.
    - `parameters` – The parameters in the Lambda input event.
    - `requestBody` – The paremeters in the request body for the Lambda input event.
    - `verb` – The API method being used, based off the action group.
  - `knowledgeBaseLookupInput` – Contains the following information about the knowledge base and the search query for the knowledge base.
    - `knowledgeBaseId` – The unique identifier of the knowledge base that the agent is looking up.
    - `text` – The query being made to the knowledge base.

- `observationTrace` – Contains the result or output of an action group or knowledge base, or the response to the user.

```
{
    "traceId": "string",
```

```
    "invocationType": "ACTION_GROUP | KNOWLEDGE_BASE | FINISH",
    "actionGroupInvocationOutput": {
        "text": "string"
    },
    "knowledgeBaseLookupOutput": {
        "sourceReferences": {
            "textSourceReferences": [
                {
                    "referenceText": "string",
                    "sourceLocation": {
                        "s3SourceLocation": {
                            "s3Uri": "string"
                        }
                    }
                },
                ...
            ]
        }
    },
    "finalResponse": {
        "text": "string"
    }
}
```

The fields are described below.

- `traceId` – The unique identifier of the trace.
- `invocationType` – Specifies whether the agent is invoking an action group, a knowledge base, or returning a response to the user.
- `actionGoupInvocationOutput` – Contains the JSON-formatted string returned by the API invoked by the action group.
- `knowledgeBaseLookupOutput` – Contains the text in the knowledge base and the S3 location of the data source. Each object in the list of `testSourceReferences` contains the following fields.
  - `referenceText` – The text from the knowledge base that is returned from the knowledge base query.
  - `sourceLocation` – Contains the S3 URI of the data source from which the returned text was found.
- `failureTrace` – Contains one field, a `failureReason`, which describes the error that occurred.

```
{
    "failureReason": "string"
}
```

Deploying an agent: versioning and aliases

After you have sufficiently iterated on your working draft and are satisfied with the behavior of your agent, you can set it up for deployment and integration into your application by creating *aliases* of your agent.

To deploy your agent, you create an *alias*. During alias creation, Amazon Bedrock automatically creates a version of your agent. The alias points to this newly created version. You can point the alias to previously created version if necessary. You then configure your application to make API calls to that alias.

The version is like a snapshot that preserves the resource as it exists at the time it was created. You can keep modifying the working draft and create new aliases (and consequently, versions) of your agent as necessary. In Amazon Bedrock, you create a new version of your agent by creating an alias that points

to the new version by default. Amazon Bedrock creates versions in numerical order, starting from 1. Because a version acts as a snapshot of your agent at the time you created it, it is immutable.

Aliases let you efficiently switch between different versions of your agent without requiring the application to keep track of the version. For example, you can change an alias to point to a previous version of your agent if there are changes that you need to quickly revert.

The working draft version is `DRAFT` and the alias that points to it is the `AgentTestAlias`.

To manage versions and aliases of an agent, select **Agents** from the left navigation pane and choose the agent from the **Agents** section.

To create a new alias (and optionally a new version)

1. Select **Create alias** at the top right corner. Alternatively, select the **Deploy** tab and choose **Create** in the **Aliases** section.
2. Entire a unique name for the alias and provide an optional description.
3. Choose one of the following options
   - Create a new version and to associate the alias with it
   - Associate the alias with an existing version. From the dropdown menu, choose the version you want to associate the alias to.
4. Select **Create alias**. A green success banner appears at the top.

To manage versions of the agent, check that you are in the **Build** tab and that the **Versions** section displays underneath.

To view the details of a version

1. Select the version to view from the **Versions** section.
2. You can't modify any part of a version, but you can view details about the model, action groups, or Lambda function by choosing the name of the information you want to view.

To manage aliases for the agent, select the **Deploy** tab and check that the **Aliases** section displays underneath.

To associate an alias to a different version

1. Select the radio button next to the alias you want to edit.
2. Select the **Edit** button.
3. Choose one of the following options.
   - Create a new version and to associate the alias with it
   - Associate the alias with an existing version. From the dropdown menu, choose the version you want to associate the alias to.

Using the API

The following are the service endpoints for the Bedrock service. To connect programmatically to an AWS service, you use an endpoint. For information about endpoints for other AWS services, see AWS service endpoints in the AWS General Reference.

The following table provides a list of Region-specific endpoints that Bedrock supports.

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	invoke-agent-bedrock.us-east-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	invoke-agent-bedrock.us-west-2.amazonaws.com	HTTPS

Currently, the only API available for Bedrock is `InvokeAgent`.

Invoke your agent

To interact with your agent, send an `InvokeAgent` request. Use `TSTALIASID` as the `agentAliasId` to invoke the draft version of your agent. You must use the Bedrock runtime endpoint to call this operation: `invoke-agent-bedrock.`*`region`*`.amazonaws.com`

The following shows a sample request.

```
POST /agents/agentId/agentAliases/agentAliasId/sessions/sessionId/text HTTP/1.1
Content-type: application/json

{
   "inputText": "string",
   "endSession": boolean,
   "enableTrace": boolean,
   "sessionState": {
      "sessionAttributes": {
         "string" : "string"
      }
   }
}
```

The fields in the request are described below.

- `agentId` – The unique identifier for your agent. You can find the ID in your agent's details page.
- `agentAliasId` – The unique identifier for your agent's alias. You can find the ID in the details page for your agent's alias. To invoke the draft version of your agent, use `TSTALIASID`.
- `sessionId` – The unique identifier for the session. If you reuse this value, you continue an existing session with the agent if the value you set for the idle session timeout hasn't been exceeded.
- `inputText` – The prompt to provide the agent.
- `endSession` – Specifies whether to end the session with the agent or not.
- `enableTrace` – Specifies whether to return the trace or not in the response. By default, the trace is disabled. For more information, see Trace enablement (p. 85).
- `sessionState` – Contains session attributes for the session with the agent.

The response returns the following objects. If there is an error, the response returns an exception.

- chunk – The `bytes` field contains the response to the user as a base-64 encoded binary object. The `attribution` object contains a list of `citations` accompanying the response.
- `trace` – Contains the `agentId`, `agentAliasId`, and `sessionId`, alongside the `trace` object. For information about the fields in the trace object, see Trace enablement (p. 85). The trace is only returned if `enableTrace` was set as `true` in the request.

How Bedrock Agent works with IAM

Before you use IAM to manage access to Bedrock Agent, learn what IAM features are available to use with Bedrock Agent.

IAM features you can use with Bedrock Agent

IAM feature	Bedrock Agent support
Identity-based policies (p. 90)	Yes
Resource-based policies (p. 91)	No
Policy actions (p. 91)	Yes
Policy resources (p. 92)	Yes
Policy condition keys (p. 92)	No
ACLs (p. 92)	No
ABAC (tags in policies) (p. 93)	No
Temporary credentials (p. 93)	Yes
Principal permissions (p. 93)	Yes
Service roles (p. 94)	Yes
Service-linked roles (p. 94)	No

To get a high-level view of how Bedrock Agent and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Bedrock Agent

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Bedrock Agent

To view examples of Bedrock Agent identity-based policies, see Identity-based policy examples for Bedrock Agent (p. 94).

Resource-based policies within Bedrock Agent

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Bedrock Agent

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

For a list of Bedrock Agent actions, see Actions Defined by Bedrock Agent in the *Service Authorization Reference*.

Policy actions in Bedrock Agent use the following prefix before the action:

```
bedrock
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
        "bedrock:action1",
        "bedrock:action2"
        ]
```

To view examples of Bedrock Agent identity-based policies, see Identity-based policy examples for Bedrock Agent (p. 94).

Policy resources for Bedrock Agent

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To view examples of Bedrock Agent identity-based policies, see Identity-based policy examples for Bedrock Agent (p. 94).

Policy condition keys for Bedrock Agent

Supports service-specific policy condition keys	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

All Bedrock Agent actions support condition keys using agents or aliases as the resource.

To view examples of Bedrock Agent identity-based policies, see Identity-based policy examples for Bedrock Agent (p. 94).

ACLs in Bedrock Agent

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Bedrock Agent

Supports ABAC (tags in policies)	No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Bedrock Agent

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Bedrock Agent

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that

then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Bedrock Agent in the *Service Authorization Reference*.

Service roles for Bedrock Agent

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
> Changing the permissions for a service role might break Bedrock Agent functionality. Edit service roles only when Bedrock Agent provides guidance to do so.

When you create an agent resource, you must create or choose a role to allow Bedrock Agent to access resources on your behalf. If you have previously created a service role or service-linked role, Bedrock Agent provides you with a list of roles to choose from. It's important to choose a role that allows access to the required actions.

Service-linked roles for Bedrock Agent

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

Identity-based policy examples for Bedrock Agent

By default, users and roles don't have permission to create or modify Bedrock Agent resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by Bedrock Agent, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Bedrock Agent in the *Service Authorization Reference*.

> **Note**
> The Bedrock Agent service is available as a limited preview release, so its information is not included in the Service Authorization Reference.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Bedrock Agent resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Bedrock Agent console

To access the Bedrock Agent console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Bedrock Agent resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To provide access to the Bedrock Agent console, attach the following policy to the roles or entities that need access.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "BedrockConsole",
            "Effect": "Allow",
            "Action": [
               "bedrock:CreateAgent",
               "bedrock:UpdateAgent",
               "bedrock:GetAgent",
               "bedrock:ListAgents",
               "bedrock:CreateActionGroup",
               "bedrock:UpdateActionGroup",
               "bedrock:GetActionGroup",
               "bedrock:ListActionGroups",
               "bedrock:CreateAgentDraftSnapshot",
               "bedrock:GetAgentVersion",
               "bedrock:ListAgentVersions",
               "bedrock:CreateAgentAlias",
               "bedrock:UpdateAgentAlias",
               "bedrock:GetAgentAlias,
               "bedrock:ListAgentAliases"
               "bedrock:InvokeAgent"
            ],
            "Resource": "*"
        }
    ]
```

For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Allow users to perform actions on agent and alias resources

You can provision identities with permissions to perform actions on agent and alias resources. The ARNs of these resources are formatted as follows.

- Agents `arn:aws:bedrock:`*`region`*`:`*`account-id`*`:agent/`*`AGENTID`*
- Aliases `arn:aws:bedrock:`*`region`*`:`*`account-id`*`:agent-alias/`*`AGENTID`*`/`*`ALIASID`*

A role can call API operations on specific resources. For example, the `InvokeAgent` operation can only be used on alias resources and the `UpdateAgent` operation can only be used on agent resources, If you specify an operation in a policy that can't be used on the resource specified in the policy, Bedrock Agent returns an error. For a list of operations and the resources that they can be used with, see the following table. `CreateAgent` and `ListAgents` are not performed on a specific resource.

Operation	Resource
CreateAgent	N/A
UpdateAgent	Agent
GetAgent	Agent
ListAgents	N/A
CreateActionGroup	Agent
UpdateActionGroup	Agent
GetActionGroup	Agent
ListActionGroups	Agent
CreateAgentDraftSnapshot	Agent
GetAgentVersion	Agent
ListAgentVersions	Agent
CreateAgentAlias	Agent
UpdateAgentAlias	Alias
GetAgentAlias	Alias
ListAgentAliases	Agent
InvokeAgent	Alias

The following is a sample policy that you can attach to an IAM role to allow it to call Bedrock Agent API operations to get information about an agent, update an agent alias, and interact with an agent. Replace the *`sid`* with a policy identifier of your choice, the *`account-id`* with the account ID to which the agent belongs, the `AGENTID` with the ID of the agent, and the `ALIASID` with the alias of the agent. You can find agent and alias IDs with the `ListAgents` API operation or in the agent and alias details pages in the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "sid",
            "Effect": "Allow",
            "Action": "bedrock:GetAgent",
            "Resource": "arn:aws:bedrock:region:account-id:agent/AGENTID"
        },
        {
            "Sid": "sid",
            "Effect": "Allow",
            "Action": [
                "bedrock:UpdateAgentAlias",
                "bedrock:InvokeAgent"
            ],
            "Resource": [
                "arn:aws:bedrock:region:account-id:agent-alias/AGENTID/ALIASID"
            ]
        },
    ]
}
```

Tag resources

To help you manage your Amazon Bedrock resources, you can assign metadata to each resource as tags. A tag is a label that you assign to an AWS resource. Each tag consists of a key and a value.

The Amazon Bedrock resources that you can tag are:

- Custom models
- Model customization jobs
- Provisioned models

Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or application. Tags help you to do the following:

- Identify and organize your AWS resources. Many AWS resources support tagging, so you can assign the same tag to resources in different services to indicate that the resources are the same.
- Allocate costs. You activate tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use cost allocation tags in the *AWS Billing and Cost Management User Guide*.
- Control access to your resources. You can use tags with Amazon Bedrock to create policies to control access to Amazon Bedrock resources. These policies can be attached to an IAM role or user to enable tag-based access control.

Topics

Use the console

Tagging is supported for custom models and fine-tuning jobs. You can add and modify tags at any time in the fine-tuning process.

To add tags when submitting a new fine-tuning job:

1. Select **Fine-tune** from the left sidebar.
2. In the **Models** section, choose **Fine-tune model**.
3. In the **Model configuration** or **Job configuration** section, expand the **Tags** section.
4. Select **Add new tag** and fill in the **Key** and **Value** for the tag. You can add up to 50 tags in total.
5. Select **Remove tag** next to a key-value pair to remove that tag.

To add tags to an existing custom model or fine-tuning job:

1. Select **Fine-tune** from the left sidebar.
2. In the **Models** or **Training jobs** section, choose the model or job that you want to add tags to.
3. In the **Tags** section at the bottom, select **Manage Tags**.
4. Select **Add new tag** and fill in the **Key** and **Value** for the tag. You can add up to 50 tags in total.

5. Select **Remove tag** next to a key-value pair to remove that tag.

Use APIs

To carry out tagging operations, you need the Amazon Resource Name (ARN) of the resource on which you want to carry out a tagging operation. Use the TagResource and UnTagResource operations to tag and untag resources. To list the tags for a resource, use the ListTagsForResource operation. For tagging examples, see Tag resources (p. 27).

Tag restrictions

The following basic restrictions apply to tags on Amazon Bedrock resources:

- Maximum number of tags per resource – 50.
- Maximum number of keys – 50.
- Maximum key length – 128 characters.
- Maximum value length – 256 characters.
- Valid characters for key and value – a-z, A-Z, 0-9, space, and the following characters: _.:/=+- and @
- Keys and values are case-sensitive.
- Don't use `aws:` as a prefix for keys. This is reserved for AWS use.

Security in Amazon Bedrock

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Bedrock, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Bedrock. The following topics show you how to configure Bedrock to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Bedrock resources.

Topics

Data protection in Amazon Bedrock

The AWS shared responsibility model applies to data protection in Amazon Bedrock. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.

- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Bedrock or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Amazon Bedrock doesn't use your prompts and continuations to train any AWS models or distribute them to third parties. Your training data isn't used to train the base Amazon Titan models or distributed to third parties. Other usage data, such as usage timestamps, logged account IDs, and other information logged by the service, is also not used to train the models.

Amazon Bedrock uses the fine tuning data you provide only for fine tuning an Amazon Titan model. Amazon Bedrock doesn't use fine tuning data for any other purpose, such as training base foundation models.

Each model provider has an escrow account that they upload their models to. The Amazon Bedrock inference account has permissions to call these models, but the escrow accounts themselves don't have outbound permissions to Amazon Bedrock accounts. Additionally, model providers don't have access to Amazon Bedrock logs or access to customer prompts and continuations.

Amazon Bedrock redacts your natural language data from its service logs. None of your data is stored in Amazon Bedrock accounts. None of data you provide for fine tuning is stored in Amazon Bedrock accounts. Once training and evaluation data is used to fine tune a custom model, the training and evaluation data remains only in your AWS account. During training, your data exists in Amazon SageMaker instance memory, but is encrypted on these machines using an XTS-AES-256 cipher that is implemented on a hardware module, on the instance itself. Once a fine tuned custom model is complete, the model contains the model weight obtained by training the model on your training data. The custom model doesn't retain any of your training data.

Custom model metadata (name and Amazon Resource Name) and a provisioned model's metadata is stored in an Amazon DynamoDB table that is encrypted with a key that the Amazon Bedrock service owns.

Topics

Data encryption

Amazon Bedrock uses encryption to protect data at rest and data in transit.

Topics

Encryption in transit

Within AWS, all inter-network data in transit supports TLS 1.2 encryption.

Requests to the Amazon Bedrock API and console are made over a secure (SSL) connection. You pass AWS Identity and Access Management (IAM) roles to Amazon Bedrock to provide permissions to access resources on your behalf for training and deployment.

Encryption at rest

Amazon Bedrock provides Encryption of model customization jobs (p. 103) at rest.

Key management

You can use AWS Key Management Service (KMS) to manage the keys that you use with model customization jobs. For more information, see AWS Key Management Service concepts.

Encryption of model customization jobs

Amazon Bedrock encrypts the model artifacts from your model customization jobs. By default, Amazon Bedrock encrypts this data using an AWS managed key. Optionally, you can encrypt the model artifacts using a customer managed key.

For more information about AWS KMS keys, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

Topics

Model customization inputs

When you use Amazon Bedrock to run a model customization job, you store the input documents (training/validation data) in your Amazon S3 bucket. To encrypt these documents at rest, you can use the Amazon S3 SSE-S3 server-side encryption option. With this option, objects are encrypted with service keys managed by the Amazon S3 service.

For more information, see Protecting data using server-side encryption with Amazon S3-managed encryption keys (SSE-S3) in the *Amazon Simple Storage Service User Guide*.

Model customization outputs

When Amazon Bedrock completes a model customization job, it stores the job metrics in the Amazon S3 location that you specified when you created the job. To encrypt the metrics, you can use the Amazon S3 SSE-S3 server-side encryption option described in the previous section.

Amazon Bedrock stores the custom model artifacts in an Amazon S3 bucket controlled by AWS. By default, Amazon Bedrock encrypts this data using an *AWS managed key*. This type of KMS key is created by AWS, so you don't manage this KMS key yourself. AWS manages the key and uses it on your behalf.

Optionally, you can choose to encrypt the custom model artifacts with a *customer managed key*. This is a KMS key that you create, own, and manage in your AWS account.

Before you can use your own KMS key, configure the policies and permissions as described in the following sections.

Create a customer managed key

Any user with `CreateKey` permissions can create customer managed keys using either the AWS Key Management Service (AWS KMS) console or the CreateKey API operation. Make sure to create a symmetric encryption key.

Create a key policy and add the following policy statements to grant permissions to custom model builders and users.

```
{
    "Version": "2012-10-17",
    "Id": "KMS Key Policy",
    "Statement": [
        {
            "Sid": "Permissions for custom model users",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:user/CustomModelCaller"
            },
            "Action": "kms:Decrypt",
            "Resource": "*"
        },
        {
            "Sid": "Permissions for custom model builders",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:user/CustomModelBuilder"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:DescribeKey",
                "kms:CreateGrant"
            ],
            "Resource": "*"
        }
}
```

Use a customer managed key to run customization jobs

For users to create a model customization job, the user or role needs the following permissions to use the customer managed key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Bedrock custom model builder policy",
      "Effect": "Allow",
      "Action": [
           "kms:Decrypt",
           "kms:DescribeKey",
           "kms:GenerateDataKey"
           "kms:CreateGrant"
        ],
      "Resource": "arn:aws:kms:region:111122223333:key/key id"
    }
  ]
}
```

Use a customer managed key during inference

For users to run inference on a custom model, the user or role needs the following permissions to use the customer managed key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Bedrock Customer Invocation Policy",
      "Effect": "Allow",
      "Action": [
           "kms:Decrypt"
        ],
      "Resource": "arn:aws:kms:region:111122223333:key/key id"
    }
  ]
}
```

When you invoke a model to run inference (for a custom model encrypted with a customer managed key), if you don't have `kms:Decrypt` permissions for that key, the request fails with the following error message:

You don't have sufficient access to the model's KMS key. Ensure that kms:Decrypt permissions are correctly configured

Protect jobs using a VPC

When you run a model customization job, the job accesses your Amazon S3 bucket to download the input data and to upload job metrics.

To control access to your data, we recommend that you create a virtual private cloud (VPC). Configure it so that your training data is not accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the *Amazon VPC User Guide*.

Using a VPC protects your data and lets you monitor all network traffic in and out of the AWS job containers by using VPC Flow Logs. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

If you configure your VPC with no internet access, you must create a VPC endpoint that allows the customization job to access these S3 buckets.

When you configure your customization job to use VPC, the job creates an elastic network interface (ENI) that uses your VPC endpoint to access your S3 buckets. For information about ENIs, see Elastic Network Interfaces in the *Amazon VPC User Guide*.

Topics

Configure a VPC for Amazon Bedrock

When you configure the VPC for an Amazon Bedrock model customization job, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the *Amazon VPC User Guide*.

Topics

- Use a custom endpoint policy to restrict access to Amazon S3 (p. 106)
- VPC permissions for customization job role (p. 107)
- Configure route tables (p. 108)

Create an Amazon S3 VPC Endpoint

If you configure your VPC with no internet access, you need create a VPC endpoint. The endpoint allows access to the S3 buckets that contain your training data and the training loss metrics data that's stored by the job.

By creating a VPC endpoint, you allow your model customization jobs to access the buckets where you store your data and model artifacts.

We recommend that you also create a custom policy that allows only requests from your private VPC to access your S3 buckets. For more information, see Use interface VPC endpoints (AWS PrivateLink) (p. 137).

To create an Amazon S3 VPC endpoint

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**.
3. For **Service Name**, search for `com.amazonaws.region.s3`. Replace *region* with the name of the Region where your VPC resides.
4. Choose the **Gateway** type.
5. For **VPC**, choose the VPC you want to use for this endpoint.
6. For **Configure route tables**, select the route tables to be used by the endpoint. Amazon VPC automatically adds a route to each selected route table that points any Amazon S3 traffic to the new endpoint.
7. For **Policy**, choose **Full Access** to allow full access to Amazon S3 by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see Use a custom endpoint policy to restrict access to Amazon S3 (p. 106).

Use a custom endpoint policy to restrict access to Amazon S3

The default endpoint policy allows full access to Amazon S3 for any user or service in your VPC. To further restrict access to Amazon S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your VPC.

The following policy allows access to S3 buckets. Edit this policy to allow access to only the resources that your job needs.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
        "Sid": "Allow access to output location",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::my_output_bucket/myfolder"
        ]
      },
```

```
      {
        "Sid": "Allow access to input data",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::my_training_data_bucket/myfolder",
            "arn:aws:s3:::my_validation_data_bucket/myfolder"
        ]
      }
   ]
}
```

For information, see Using Amazon S3 Bucket Policies.

VPC permissions for customization job role

When you use a VPC with your model customization job, the data access role that you provide in the `CreateModelCustomizationJob` request must include the following permissions. Edit this policy to allow access to only the VPC resources that your job needs.

```
{
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcs",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
                ],
        "Resource": "*"
    }, {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
       ]
       "Resource":[
           "arn:aws:ec2:{region}:{AccountId}:network-interface/*"
       ],
       "Condition": {
           "StringEquals": {
               "aws:RequestTag/BedrockManaged": ["true"]
            },
            "ArnEquals": {
               "aws:RequestTag/BedrockModelCustomizationJobArn": ["arn:aws:bedrock:
{region}:{AccountId}:model-customization-job/*"]
            }
       }
    }, {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
       ]
       "Resource":[
           "arn:aws:ec2:{region}:{AccountId}:subnet/{subnet-Id}",
           "arn:aws:ec2:{region}:{AccountId}:subnet/{subnet-Id2}",
           "arn:aws:ec2:{region}:{AccountId}:security-group/{sg-Id}",
           "arn:aws:ec2:{region}:{AccountId}:security-group/{sg-Id2}"
       ]
    }, {
        "Effect": "Allow",
```

```
        "Action": [
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DeleteNetworkInterfacePermission",
        ]
        "Resource": "*",
        "Condition": {
           "ArnEquals": {
               "ec2:Subnet": [
                   "arn:aws:ec2:{region}:{AccountId}:subnet/{subnet-Id}",
                   "arn:aws:ec2:{region}:{AccountId}:subnet/{subnet-Id2}"
               ],
               "ec2:ResourceTag/BedrockModelCustomizationJobArn": ["arn:aws:bedrock:
{region}:{AccountId}:model-customization-job/*"]
           },
           "StringEquals": {
               "ec2:ResourceTag/BedrockManaged": "true",
           }
        }
    }, {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags"
        ],
        "Resource": "arn:aws:ec2:{region}:{AccountId}:network-interface/*",
        "Condition": {
            "StringEquals": {
                "ec2:CreateAction": [
                    "CreateNetworkInterface"
                ]
            },
            "ForAllValues:StringEquals": {
                "aws:TagKeys": [
                    "BedrockManaged",
                    "BedrockModelCustomizationJobArn"
                ]
            }
      }
    }
```

For more information about configuring the data access role for model customization jobs, see Set up an IAM role for model customization (p. 43).

Configure route tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs for the locations of the data in your training jobs resolve. Do this by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the *Amazon VPC User Guide*.

Configure your model customization job to use VPC

After you configure the VPC and the required roles and permissions as described in the previous sections, you can create a model customization job that uses this VPC.

When you specify the VPN subnets and security groups for a job, Amazon Bedrock creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow the Amazon Bedrock job to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the *Amazon VPC User Guide*. Amazon Bedrock tags ENIs that it creates with `BedrockManaged` and `BedrockModelCusomizationJobArn` tags.

We recommend that you provide at least one subnet in each Availability Zone.

You can use security groups to establish rules for controlling Amazon Bedrock access to your VPC resources.

Use the API

For the Amazon Bedrock API, you specify VPC subnets and security groups in the `VpcConfig` request parameter. The following is an example of the `VpcConfig` parameter that you include in your API call:

```
"VpcConfig": {
"SecurityGroupIds": [
   "sg-0123456789abcdef0"
    ],
    "Subnets": [
       "subnet-0123456789abcdef0",
       "subnet-0123456789abcdef1",
       "subnet-0123456789abcdef2"
    ]
}
```

For more information about configuring jobs using the API, see Submit a job (API) (p. 46).

Use the console

For the Amazon Bedrock console, you specify VPC subnets and security groups in the optional **VPC settings** section when you create the model customization job. For more information about configuring jobs using the console, see Submit a job (console) (p. 40).

Note
For a job that includes VPC configuration, the console cannot create a new service role for the job. Create the service role using the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
                }
            }
        }
    ]
}
```

Identity and access management for Amazon Bedrock

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and

authorized (have permissions) to use Bedrock resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Bedrock.

Service user – If you use the Bedrock service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Bedrock features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Bedrock, see Troubleshooting Amazon Bedrock identity and access (p. 123).

Service administrator – If you're in charge of Bedrock resources at your company, you probably have full access to Bedrock. It's your job to determine which Bedrock features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Bedrock, see How Amazon Bedrock works with IAM (p. 114).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Bedrock. To view example Bedrock identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Bedrock (p. 118).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Bedrock in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all

features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Bedrock works with IAM

Before you use IAM to manage access to Bedrock, learn what IAM features are available to use with Bedrock.

IAM features you can use with Amazon Bedrock

IAM feature	Bedrock support
Identity-based policies (p. 114)	Yes
Resource-based policies (p. 115)	No
Policy actions (p. 115)	Yes
Policy resources (p. 116)	Yes
Policy condition keys (p. 116)	Yes
ACLs (p. 117)	No
ABAC (tags in policies) (p. 117)	Partial
Temporary credentials (p. 117)	Yes
Principal permissions (p. 118)	Yes
Service roles (p. 118)	Yes
Service-linked roles (p. 118)	No

To get a high-level view of how Bedrock and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Bedrock

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Bedrock

To view examples of Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock (p. 118).

Resource-based policies within Bedrock

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Bedrock

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Bedrock actions, see Actions Defined by Amazon Bedrock in the *Service Authorization Reference*.

Policy actions in Bedrock use the following prefix before the action:

```
bedrock
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "bedrock:action1",
      "bedrock:action2"
         ]
```

To view examples of Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock (p. 118).

Policy resources for Bedrock

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To view examples of Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock (p. 118).

Policy condition keys for Bedrock

Supports service-specific policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

All Amazon Bedrock actions support condition keys using Amazon Bedrock models as the resource.

To view examples of Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock (p. 118).

ACLs in Bedrock

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Bedrock

Supports ABAC (tags in policies)	Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Bedrock

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically

create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Bedrock

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Bedrock in the *Service Authorization Reference*.

Service roles for Bedrock

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
> Changing the permissions for a service role might break Bedrock functionality. Edit service roles only when Bedrock provides guidance to do so.

Service-linked roles for Bedrock

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

Identity-based policy examples for Amazon Bedrock

By default, users and roles don't have permission to create or modify Bedrock resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by Bedrock, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Bedrock in the *Service Authorization Reference*.

Note
The Amazon Bedrock service is available as a limited preview release, so its information is not included in the Service Authorization Reference.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Bedrock resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Use the Bedrock console

To access the Amazon Bedrock console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Bedrock resources in your AWS account. If you create an

identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To provide access to the Amazon Bedrock console, attach the following policy to the roles or entities that need access. A few of the actions in the policy are for use by the Amazon Bedrock console only, so IAM might raise `Invalid Action` errors. You can ignore these errors and choose **Next** to complete the task.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BedrockConsole",
            "Effect": "Allow",
            "Action": [
               "bedrock:ListFoundationModels",
               "bedrock:GetFoundationModel",
               "bedrock:InvokeModel",
               "bedrock:InvokeModelWithResponseStream",
               "bedrock:CreateModelCustomizationJob",
               "bedrock:GetModelCustomizationJob",
               "bedrock:GetFoundationModelAvailability",
               "bedrock:ListModelCustomizationJobs",
               "bedrock:StopModelCustomizationJob",
               "bedrock:GetCustomModel",
               "bedrock:ListCustomModels",
               "bedrock:DeleteCustomModel",
               "bedrock:CreateProvisionedModelThroughput",
               "bedrock:UpdateProvisionedModelThroughput",
               "bedrock:GetProvisionedModelThroughput",
               "bedrock:DeleteProvisionedModelThroughput",
               "bedrock:ListProvisionedModelThroughputs",
               "bedrock:ListTagsForResource",
               "bedrock:UntagResource",
               "bedrock:TagResource",
               "bedrock:PutFoundationModelEntitlement",
               "bedrock:GetModelInvocationLoggingConfiguration",
               "bedrock:PutModelInvocationLoggingConfiguration",
               "bedrock:CreateFoundationModelAgreement",
               "bedrock:DeleteFoundationModelAgreement",
               "bedrock:ListFoundationModelAgreementOffers",
               "bedrock:GetUseCaseForModelAccess",
               "bedrock:PutUseCaseForModelAccess"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow access to third-party model subscriptions

To access the Amazon Bedrock models for the first time, you use the Amazon Bedrock console to subscribe to third-party models. Your IAM user or role that the console user assumes requires permission to access the subscription API operations.

The following example shows an identity-based policy to allow access to the subscription API operations. The example includes a condition key that limits the scope of the policy to the Amazon Bedrock products in the Marketplace.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:Subscribe"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws-marketplace:ProductId": [
                        "c468b48a-84df-43a4-8c46-8870630108a7",
                        "99d90be8-b43e-49b7-91e4-752f3866c8c7",
                        "b0eb9475-3a2c-43d1-94d3-56756fd43737",
                        "1d288c71-65f9-489a-a3e2-9c7f4f6e6a85",
                        "cc0bdd50-279a-40d8-829c-4009b77a1fcc",
                        "d0123e8d-50d6-4dba-8a26-3fed4899f388",
                        "a61c46fe-1747-41aa-9af0-2e0ae8a9ce05"
                    ]
                }
            }
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:Unsubscribe",
                "aws-marketplace:ViewSubscriptions"
            ],
            "Resource": "*"
        }
    ]
}
```

Deny access for inference on specific models

The following example shows an identity-based policy that denies access to running inference on a specific model.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "DenyInference",
        "Effect": "Deny",
        "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ]
        "Resource": "arn:aws:bedrock:*::foundation-model/model-id-of-model-to-deny"
    }
}
```

Grant custom jobs access to your training data

The following example grants access to the Amazon S3 locations of the input and output data.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:ListObjects"
            ],
            "Resource": [
                "arn:aws:s3:::my_training_data_bucket/myfolder",
                "arn:aws:s3:::my_validation_bucket/myfolder",
                "arn:aws:s3:::my_output_bucket/myfolder"
            ]
        }
    ]
}
```

The following example creates a trust policy to allow model-customization jobs to assume a role, if the job was created in the same account.

```
{
 "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
                }
            }
        }
        }
    ]
}
```

Permissions for using KMS keys with model customization

Provide the appropriate AWS KMS permissions for users and roles that create or use custom models. For details, see Use a customer managed key to run customization jobs (p. 104).

Service role

You can set up a service role that lets Amazon Bedrock run on your behalf.

1. Create an AmazonBedrock-ExecutionRole (RolePolicy) with the permissions needed for training (S3, KMS) and a Trusted Entity policy which allows Amazon Bedrock to assume this role (Reference role). Note the Amazon Resouce Name (ARN) of the role policy.
2. In the role policy, to set the context keys for `aws:SourceArn` and `aws:SourceAccount` to the Amazon Bedrock job Amazon Resouce Name (ARN) and your AWS Account ID when Amazon Bedrock tries to assume these credentials.
3. Create an IAM entity with permissions to access Amazon Bedrock and pass role permissions to the `RoleArn` you noted in step 1.

You can then run a fine tuning job with the console, or with the `CreateFineTuningJob` operation, passing the `>RoleArn` and the Amazon S3 URI of your training data.

Amazon Bedrock performs `PassRole` checks in the CreateFineTuningJob operation to prevent confused deputy scenarios. These PassRole checks validate that you are passing in a role that has permissions to pass that role into an AWS service to act on their behalf. Amazon Bedrock then uses the provided Role ARN to generate credentials that it uses to download the training and validation data stored in your account, as well as upload training metrics to your account.

Troubleshooting Amazon Bedrock identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Bedrock and IAM.

Topics

I am not authorized to perform an action in Bedrock

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `bedrock:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 bedrock:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `bedrock:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Bedrock.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Bedrock. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Bedrock resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Bedrock supports these features, see How Amazon Bedrock works with IAM (p. 114).

- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Compliance validation for Amazon Bedrock

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  Note
  Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Incident response in Amazon Bedrock

Incident response for Amazon Bedrock is an AWS responsibility. AWS has a formal, documented policy and program that governs incident response.

AWS operational issues with broad impact are posted on the AWS Service Health Dashboard.

Operational issues are also posted to individual accounts via the AWS Health Dashboard. For information on how to use the AWS Health Dashboard, see the AWS Health User Guide.

Resilience in Amazon Bedrock

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Amazon Bedrock

As a managed service, Amazon Bedrock is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Bedrock through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon Bedrock gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:bedrock:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be ResourceDescription.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Bedrock to prevent the confused deputy problem.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
                }
            }
        }
    ]
}
```

Configuration and vulnerability analysis in Amazon Bedrock

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Monitor

You can monitor Amazon Bedrock with Amazon CloudWatch and with Amazon EventBridge.

Topics

Monitor Amazon Bedrock with Amazon CloudWatch

You can monitor Amazon Bedrock using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. You can graph the metrics using the CloudWatch console. You can also set alarms that watch for certain thresholds, and send notifications or take actions when values exceed those thresholds.

For more information, see What is Amazon CloudWatch in the *Amazon CloudWatch User Guide*.

Topics

Runtime metrics

The following table describes runtime metrics provided by Amazon Bedrock.

Metric name	Unit	Description
Invocations	SampleCount	Number of requests to the InvokeModel or InvokeModelWithResponseStream API operations.
InvocationLatency	MilliSeconds	Latency of the invocations.
InvocationClientErrors	SampleCount	Number of invocations that result in client-side errors.
InvocationServerErrors	SampleCount	Number of invocations that result in AWS server-side errors.

Metric name	Unit	Description
InvocationThrottles	SampleCount	Number of invocations that the system throttled.
InputTokenCount	SampleCount	Number of tokens of text input.
OutputTokenCount	SampleCount	Number of tokens of text output.
ContentFilteredCount	SampleCount	Number of times the text output content was filtered.
OutputImageCount	SampleCount	Number of output images.

Logging CloudWatch metrics

For each delivery success or failure attempt, the following Amazon CloudWatch metrics are emitted under the namespace `AWS/Bedrock`, and `Across all model IDs` dimension:

- `ModelInvocationLogsCloudWatchDeliverySuccess`
- `ModelInvocationLogsCloudWatchDeliveryFailure`
- `ModelInvocationLogsS3DeliverySuccess`
- `ModelInvocationLogsS3DeliveryFailure`
- `ModelInvocationLargeDataS3DeliverySuccess`
- `ModelInvocationLargeDataS3DeliveryFailure`

If logs fail to deliver due to permission misconfiguration or transient failures, the delivery is retried periodically for up to 24 hours.

Use CloudWatch metrics for Amazon Bedrock

To retrieve metrics for your Amazon Bedrock operations, you specify the following information:

- The metric dimension. A *dimension* is a set of name-value pairs that you use to identify a metric. Amazon Bedrock supports the following dimensions:
  - `ModelId` – all metrics
  - `ModelId + ImageSize + BucketedStepSize` – OutputImageCount
- The metric name, such as `InvocationClientErrors`.

You can get metrics for Amazon Bedrock with the AWS Management Console, the AWS CLI, or the CloudWatch API. You can use the CloudWatch API through one of the AWS Software Development Kits (SDKs) or the CloudWatch API tools.

You must have the appropriate CloudWatch permissions to monitor Amazon Bedrock with CloudWatch For more information, see Authentication and Access Control for Amazon CloudWatch in the *Amazon CloudWatch User Guide*.

View Amazon Bedrock metrics

View Amazon Bedrock metrics in the CloudWatch console.

To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, choose **All Metrics**, and then search for **ModelId**.

Monitor Amazon Bedrock events in Amazon EventBridge

You can use Amazon EventBridge to monitor status change events in Amazon Bedrock. With Amazon EventBridge, you can configure Amazon SageMaker to respond automatically to a model customization job status change in Amazon Bedrock. Events from Amazon Bedrock are delivered to Amazon EventBridge in near real time. You can write simple rules to automate actions when an event matches a rule. If you use Amazon EventBridge with Amazon Bedrock, you can:

- Publish notifications whenever there is a state change event in the model customization you have triggered, whether you add new asynchronous workflows in the future. The event published should give you enough information to react to events in downstream workflows.
- Deliver job status updates without invoking the GetModelCustomizationJob API, which can mean handling API rate limit issues, API updates, and reduction in additional compute resources.

There is no cost to receive AWS events from Amazon EventBridge. For more information about, Amazon EventBridge, see Amazon EventBridge

Note

- Amazon Bedrock emits events on a best-effort basis. Events are delivered to Amazon EventBridge in near real time. With Amazon EventBridge, you can create rules that trigger programmatic actions in response to an event. For example, you can configure a rule that invokes an SNS topic to send an email notification or invokes a function to take some action. For more information, see the *Amazon EventBridge User Guide*.
- AWS Bedrock creates a new event every time there is a state change in a model customization job that you trigger and make best-effort delivery of such event.

Topics

How it works

To receive events from Amazon Bedrock, you need to create rules and targets to match, receive, and handle state change data through Amazon EventBridge. Amazon EventBridge is a serverless event bus that ingests change state events from AWS services, SaaS partners, and customer applications. It processes events based on rules or patterns that you create, and routes these events to one or more “targets” that you choose, such as AWS Lambda, Amazon Simple Queue Service, and Amazon Simple Notification Service.

Amazon Bedrock publishes your events via Amazon EventBridge whenever there is a change in the state of a model customization job. In each case, a new event is created and sent to Amazon EventBridge,

which then sends the event to your default event-bus. The event shows which customization job's state has changed, and the current state of the job. When Amazon EventBridge receives an event that matches a rule that you created, Amazon EventBridge routes it to the target that you specified. When you create a rule, you can configure these targets as well as downstream workflows based on the contents of the event.

EventBridge schema

The following event fields in the EventBridge event schema are specific to Amazon Bedrock.

- `jobArn` — The ARN of the model customization job.
- `outputModelArn` — The ARN of the output model. Published when the training job has completed.
- `jobStatus` — The current status of the job.
- `FailureMessage` — A failure message. Published when the training job has failed.

Event example

The folllowing is example event JSON for a failed model customization job.

```
{
    "version": "0",
    "id": "UUID",
    "detail-type": "Model Customization Job State Change",
    "source": "aws.bedrock",
    "account": "123412341234",
    "time": "2023-08-11T12:34:56Z",
    "region": "us-east-1",
    "resources": [ "arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghwxyz" ],
    "detail": {
        "version": "0.0",
        "jobName": "abcd-wxyz",
        "jobArn": "arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghwxyz",
        "outputModelName": "dummy-output-model-name",
        "outputModelArn": "arn:aws:bedrock:us-east-1:123412341234:dummy-output-model-name",
        "roleArn": "arn:aws:iam::123412341234:role/JobExecutionRole",
        "jobStatus": "Failed",
        "failureMessage": "Failure Message here.",
        "creationTime": "2023-08-11T10:11:12Z",
        "lastModifiedTime": "2023-08-11T12:34:56Z",
        "endTime": "2023-08-11T12:34:56Z",
        "baseModelArn": "arn:aws:bedrock:us-east-1:123412341234:base-model-name",
        "hyperParameters": {
            "batchSize" : "batchSizeNumberUsed",
            "epochCount": "epochCountNumberUsed",
            "learningRate": "learningRateUsed",
            "learningRateWarmupSteps": "learningRateWarmupStepsUsed"
            },
        "trainingDataConfig": {
            "s3Uri": "s3://bucket/key",
            },
        "validationDataConfig": {
            "s3Uri": "s3://bucket/key",
            },
        "outputDataConfig": {
            "s3Uri": "s3://bucket/key",
            }
    }
```

```
}
```

Rules and targets

When an incoming event matches a rule that you created, the event is routed to the target that you specified for that rule, and the target processes these events. Targets support JSON format and can include AWS services such as Amazon EC2 instances, Lambda functions, Kinesis streams, Amazon ECS tasks, Step Functions, Amazon SNS topics, and Amazon SQS. To receive and process events correctly, you need to create rules and targets for matching, receiving, and correctly handling event data. You can create these rules and targets either through the Amazon EventBridge console, or through the AWS CLI.

Example rule

This rule matches an event pattern emitted by: `source ["aws.bedrock"]`. The rule captures all events sent by Amazon EventBridge that have source "aws.bedrock" to your default event bus.

```
{
    "source": ["aws.bedrock"]
}
```

Target

When creating a rule in Amazon EventBridge, you need to specify a target where EventBridge sends the event that matches your rule pattern. These targets can be a SageMaker pipeline, a Lambda function, an SNS topic, an SQS queue or any of the other targets that EventBridge currently supports. You can refer to the *Amazon EventBridge* documentation to learn how to set targets for events. For a procedure that shows how to use Amazon Simple Notification Service as a target, see Create a rule to handle AWS Bedrock events (p. 132).

Create a rule to handle AWS Bedrock events

Complete the following procedures in order to receive email notifications about your AWS Bedrock events.

Create an Amazon Simple Notification Service topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. In the navigation pane, choose **Topics**.
3. Choose **Create topic**.
4. For **Type**, choose **Standard**.
5. For **Name**, enter a name for your topic.
6. Choose **Create topic**.
7. Choose **Create subscription**.
8. For **Protocol**, choose **Email**.
9. For **Endpoint**, enter the email address that receives the notifications.
10. Choose **Create subscription**.
11. You'll receive an email message with the following subject line: `AWS Notification - Subscription Confirmation`. Follow the directions to confirm your subscription.

Use the following procedure to create a rule to handle your AWS Bedrock events.

To create a rule to handle AWS Bedrock events

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. Choose **Create rule**.
3. For **Name**, enter a name for your rule.
4. For **Rule type**, choose **Rule with an event pattern**.
5. Choose **Next**.
6. For Event pattern, do the following:
    a. For **Event source**, choose **AWS services**.
    b. For **AWS service**, choose **Amazon Bedrock**.
    c. For **Event type**, choose **Model Customization Job State Change**.
    d. By default, we send notifications for every event. If you prefer, you can create an event pattern that filters events for a specific job state.
    e. Choose **Next**.
7. Specify a target as follows:
    a. For **Target types**, choose **AWS service**.
    b. For **Select a target**, choose **SNS topic**.
    c. For **Topic**, choose the SNS topic that you created for notifications.
    d. Choose **Next**.
8. (Optional) Add tags to your rule.
9. Choose **Next**.
10. Choose **Create rule**.

Log Amazon Bedrock API calls using AWS CloudTrail

Amazon Bedrock is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Bedrock. CloudTrail captures all API calls for Bedrock as events. The calls captured include calls from the Bedrock console and code calls to the Bedrock API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Bedrock. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Bedrock, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Bedrock information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Bedrock, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Bedrock, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure

other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Bedrock actions are logged by CloudTrail and are documented in the Amazon Bedrock API Reference. For example, calls to the `InvokeModel`, `ListFoundationModels` and `StopModelCustomizationJob` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Bedrock log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `InvokeModel` action.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAICFHPEXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/userxyz",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "userxyz"
    },
    "eventTime": "2023-10-11T21:58:59Z",
    "eventSource": "bedrock.amazonaws.com",
    "eventName": "InvokeModel",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Boto3/1.28.62 md/Botocore#1.31.62 ua/2.0 os/macos#22.6.0 md/arch#arm64
 lang/python#3.9.6 md/pyimpl#CPython cfg/retry-mode#legacy Botocore/1.31.62",
    "requestParameters": {
        "modelId": "stability.stable-diffusion-xl-v0"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "readOnly": false,
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "cipher suite",
        "clientProvidedHostHeader": "bedrock-runtime.us-west-2.amazonaws.com"
    }
}
```

Amazon Bedrock abuse detection

AWS is committed to the responsible use of AI. To help prevent potential misuse, Amazon Bedrock implements automated abuse detection mechanisms to identify and mitigate potential violations of AWS's Acceptable Use Policy (AUP) and Responsible AI Policy or a third-party model provider's AUP.

Our abuse detection mechanisms are fully automated, so there is no human review of, or access to, user inputs or model outputs.

Automated abuse detection includes several components:

- **Categorize content** — We use classifiers to detect harmful content (such as content that incites violence) in user inputs and model outputs. A classifier is an algorithm that processes model inputs and outputs, and assigns type of harm and level of confidence. We may run these classifiers on both Amazon Titan and third-party model usage. The classification process is automated and does not involve human review of user inputs or model outputs.
- **Identify patterns** — We use classifier metrics to identify potential violations and recurring behavior. We may compile and share aggregated and anonymized classifier metrics with third-party model providers. Amazon Bedrock does not store user input or model output and does not share these with third-party model providers.
- **Contact customers** — We may request information about customers' use of Bedrock and compliance with our policies. In the event that a customer continues to use the service in a manner that may violate AWS's policies or a third-party model provider's AUP, AWS may suspend access to Bedrock, taking into account severity, recurrence of the activity, customer cooperation, and whether customers have mechanisms in place to prevent misuse of the service.

Contact AWS Support if you have additional questions. For more information, see the Amazon Bedrock FAQs.

Use interface VPC endpoints (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon Bedrock. You can access Bedrock as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access Bedrock.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Bedrock.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Bedrock VPC endpoints

Before you set up an interface endpoint for Bedrock, review Considerations in the *AWS PrivateLink Guide*.

Bedrock supports making calls to all of its API actions through the VPC endpoint.

Bedrock endpoints are not available in all Availability Zones in a Region. When you create the endpoint, use the following command to list the Availability Zones.

```
aws ec2 describe-vpc-endpoint-services \
  --service-names com.amazonaws.region.bedrock-runtime --region region
```

Important
The service name `com.amazonaws.region.bedrock` is deprecated.

Create an interface endpoint for Bedrock

You can create an interface endpoint for Bedrock using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for Bedrock using the following service name:

```
com.amazonaws.region.bedrock-runtime
```

If you enable private DNS for the interface endpoint, you can make API requests to Bedrock using its default Regional DNS name. For example, `bedrock-runtime.us-east-1.amazonaws.com`.

Important
The private DNS `bedrock.region.amazonaws.com` is deprecated.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Bedrock through the interface endpoint. To control the access allowed to Bedrock from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for Bedrock actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed Bedrock actions for all principals on all resources.

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "bedrock:InvokeModel"
         ],
         "Resource":"*"
      }
   ]
}
```

Quotas for Amazon Bedrock

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

Some service quotas can be adjusted or increased. Refer to the **Adjustable** column in the following tables to see whether a quota can be adjusted. To request a quota increase, use the limit increase form.

Your AWS account has the following quotas related to Bedrock.

Runtime quotas

The following quotas are enforced when you use a model for inference.

Model	Requests processed per minute	Tokens processed per minute	Adjustable
Amazon Titan Express	400	300,000	No
Amazon Titan Text Embeddings	2,000	300,000	No
Anthropic Claude Instant	400	300,000	No
Anthropic Claude V2	100	200,000	No
AI21 Labs Jurassic-2 Mid	400	300,000	No
AI21 Labs Jurassic-2 Ultra	100	300,000	No
Cohere Command	400	300,000	No
Stability.ai Diffusion XL 1.0	60	N/A	No

Model customization quotas

The following quotas apply to model customization.

Model quotas

The following quotas are enforced for Titan Express.

Description	Maximum value	Adjustable
Sum of input and output tokens when batch size is 1	4096	No
Sum of input and output tokens when batch size is between 2 and 4	2048	No
Character quota	Token quota x 6	No

Fine-tuning quotas

The following quotas are enforced for fine-tuning.

Description	Default	Adjustable
Number of training records in a dataset	10,000	Yes
Number of validation records in a dataset	1,000	Yes
Training dataset file size	1 GB	Yes
Validation dataset file size	100 MB	Yes

Training quotas

The following quotas are enforced for model training.

Description	Default	Adjustable
Number of submitted model customization jobs at one time per account	2	Yes
Number of customized models	100	Yes
Number of tags attached to a resource	See AWS Resource Groups and Tagging endpoints and quotas	Yes

Provisioned throughput quotas

The following quotas are enforced for provisioned throughput.

Description	Default	Adjustable
Number of model units for a provisioned throughput for a foundation model	0	Yes

Description	Default	Adjustable
Number of model units for a provisioned throughput for a custom model	2	Yes

Document history for the Bedrock User Guide

- **Latest documentation update:** October 19, 2023

The following table describes important changes in each release of Amazon Bedrock. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Region expansion	Amazon Bedrock is now available in Europe (Frankfurt) (eu-central-1). For information on endpoints, see Amazon Bedrock endpoints and quotas.	October 19, 2023
Region expansion	Amazon Bedrock is now available in Asia Pacific (Tokyo) (ap-northeast-1). For information on endpoints, see Amazon Bedrock endpoints and quotas.	October 3, 2023
Gated general release	Gated general release of the Bedrock service. For more information, see Amazon Bedrock.	September 28, 2023
Added managed policy resource	Added AWS managed policy for BedrockThirdPartyModelAccess to include access to Marketplace models in Bedrock. For more information, see AWS managed policies.	September 28, 2023

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.