Amazon Bedrock: User Guide

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
# Table of Contents

**What is Amazon Bedrock?** ................................................................. 1
  Access the Amazon Bedrock models ............................................. 1
  Features of Amazon Bedrock .......................................................... 1
  Supported models in Amazon Bedrock ......................................... 2
  Supported Regions ............................................................................. 2
  Amazon Bedrock pricing ................................................................. 3

**Supported models** ........................................................................... 4
  AI21 Labs ...................................................................................... 5
  Amazon ......................................................................................... 5
  Anthropic ...................................................................................... 5
  Cohere ............................................................................................ 5
  Meta ............................................................................................... 6
  Stability.ai ..................................................................................... 6

**Set up** .............................................................................................. 7
  Add model access ........................................................................... 7
  Console access ............................................................................... 7
  Sign up for an AWS account ....................................................... 7
  Create an administrative user ....................................................... 8
  Set up the AWS CLI ....................................................................... 9
  Grant programmatic access ........................................................... 9

**Model access** .................................................................................. 12
  Manage model access .................................................................... 7
  Edit model access .......................................................................... 12
  Add model access .......................................................................... 12
  Remove model access .................................................................... 13

**Model lifecycle** ............................................................................... 15
  On-Demand, Provisioned Throughput, and model customization ...... 15
  Legacy versions .............................................................................. 16

**Settings** .......................................................................................... 17
  Model invocation logging .............................................................. 17
    Set up an Amazon S3 destination ............................................... 17
    Set up CloudWatch Logs destination .......................................... 19
  Using the console .......................................................................... 21
  Using APIs with invocation logging ............................................. 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Console overview</td>
<td>23</td>
</tr>
<tr>
<td>Getting started</td>
<td>23</td>
</tr>
<tr>
<td>Foundation models</td>
<td>24</td>
</tr>
<tr>
<td>Playgrounds</td>
<td>24</td>
</tr>
<tr>
<td>Orchestration</td>
<td>25</td>
</tr>
<tr>
<td>Assessment and deployment</td>
<td>25</td>
</tr>
<tr>
<td>Model access</td>
<td>25</td>
</tr>
<tr>
<td>Settings</td>
<td>25</td>
</tr>
<tr>
<td>Playgrounds</td>
<td>27</td>
</tr>
<tr>
<td>Chat playground</td>
<td>28</td>
</tr>
<tr>
<td>Text playground</td>
<td>28</td>
</tr>
<tr>
<td>Image playground</td>
<td>28</td>
</tr>
<tr>
<td>Use a playground</td>
<td>29</td>
</tr>
<tr>
<td>Use the API</td>
<td>32</td>
</tr>
<tr>
<td>API setup</td>
<td>33</td>
</tr>
<tr>
<td>Amazon Bedrock endpoints</td>
<td>33</td>
</tr>
<tr>
<td>Setting up the AWS CLI</td>
<td>33</td>
</tr>
<tr>
<td>AWS SDK setup</td>
<td>33</td>
</tr>
<tr>
<td>Using SageMaker notebooks</td>
<td>34</td>
</tr>
<tr>
<td>API operations</td>
<td>36</td>
</tr>
<tr>
<td>List the base models</td>
<td>36</td>
</tr>
<tr>
<td>Get details about a base model</td>
<td>37</td>
</tr>
<tr>
<td>Run inference</td>
<td>37</td>
</tr>
<tr>
<td>Tag resources</td>
<td>60</td>
</tr>
<tr>
<td>Inference parameters</td>
<td>63</td>
</tr>
<tr>
<td>Inference parameter definitions</td>
<td>63</td>
</tr>
<tr>
<td>Randomness and diversity</td>
<td>64</td>
</tr>
<tr>
<td>Length</td>
<td>65</td>
</tr>
<tr>
<td>Repetitions</td>
<td>65</td>
</tr>
<tr>
<td>Amazon Titan text models</td>
<td>65</td>
</tr>
<tr>
<td>Randomness and Diversity</td>
<td>65</td>
</tr>
<tr>
<td>Length</td>
<td>66</td>
</tr>
<tr>
<td>Model invocation request body field</td>
<td>66</td>
</tr>
<tr>
<td>Model invocation response body field</td>
<td>67</td>
</tr>
<tr>
<td>Amazon Titan image models</td>
<td>68</td>
</tr>
<tr>
<td>Randomness and Diversity</td>
<td>68</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Use inference parameters</td>
<td>110</td>
</tr>
<tr>
<td>Detailed guidelines</td>
<td>111</td>
</tr>
<tr>
<td>Optimize prompts for text models on Amazon Bedrock—when the basics aren’t good enough</td>
<td>117</td>
</tr>
<tr>
<td>Prompt templates and examples for Amazon Bedrock text models</td>
<td>120</td>
</tr>
<tr>
<td>Text classification</td>
<td>120</td>
</tr>
<tr>
<td>Question-answer, without context</td>
<td>123</td>
</tr>
<tr>
<td>Question-answer, with context</td>
<td>127</td>
</tr>
<tr>
<td>Summarization</td>
<td>131</td>
</tr>
<tr>
<td>Text generation</td>
<td>133</td>
</tr>
<tr>
<td>Code generation</td>
<td>136</td>
</tr>
<tr>
<td>Mathematics</td>
<td>138</td>
</tr>
<tr>
<td>Reasoning/logical thinking</td>
<td>139</td>
</tr>
<tr>
<td><strong>Embeddings</strong></td>
<td>142</td>
</tr>
<tr>
<td><strong>Model evaluation</strong></td>
<td>143</td>
</tr>
<tr>
<td>Get started</td>
<td>144</td>
</tr>
<tr>
<td>Automatic model evaluations</td>
<td>145</td>
</tr>
<tr>
<td>Human worker based model evaluation jobs</td>
<td>147</td>
</tr>
<tr>
<td>Model evaluation tasks</td>
<td>151</td>
</tr>
<tr>
<td>General text generation</td>
<td>152</td>
</tr>
<tr>
<td>Text summarization</td>
<td>153</td>
</tr>
<tr>
<td>Question and answer</td>
<td>155</td>
</tr>
<tr>
<td>Text classification</td>
<td>156</td>
</tr>
<tr>
<td>Input prompt datasets</td>
<td>157</td>
</tr>
<tr>
<td>Built-in prompt datasets</td>
<td>158</td>
</tr>
<tr>
<td>Custom prompt datasets</td>
<td>162</td>
</tr>
<tr>
<td>Worker instructions</td>
<td>166</td>
</tr>
<tr>
<td>Rating methods</td>
<td>166</td>
</tr>
<tr>
<td>Manage a work team</td>
<td>172</td>
</tr>
<tr>
<td>Model evaluation job results</td>
<td>173</td>
</tr>
<tr>
<td>Automated reports</td>
<td>174</td>
</tr>
<tr>
<td>Human report cards</td>
<td>176</td>
</tr>
<tr>
<td>Amazon S3 output</td>
<td>183</td>
</tr>
<tr>
<td>Required permissions</td>
<td>189</td>
</tr>
<tr>
<td>Console permission requirements</td>
<td>190</td>
</tr>
<tr>
<td>Service roles</td>
<td>193</td>
</tr>
</tbody>
</table>
Dataset formats ................................................................................................................. 325
Hyperparameters ........................................................................................................... 325
Cohere Command .......................................................................................................... 325
Cohere Command Light ................................................................................................. 328
Meta Llama 2 13B ........................................................................................................ 330
Meta Llama 2 70B .......................................................................................................... 331
Amazon Titan .................................................................................................................. 332

**Provisioned Throughput** ....................................................................................... 336
Model IDs ....................................................................................................................... 337
Procedures ..................................................................................................................... 338
  Creating ....................................................................................................................... 338
  Updating ...................................................................................................................... 338
  Deleting ...................................................................................................................... 339
  Running inference ...................................................................................................... 339
Permissions .................................................................................................................... 339
Console procedures ...................................................................................................... 339
  View Provisioned Throughput summary ................................................................ 340
  Purchase Provisioned Throughput .......................................................................... 340
  View details of a Provisioned Throughput .............................................................. 342
  Edit a Provisioned Throughput .............................................................................. 342
  Delete a Provisioned Throughput .......................................................................... 343
API operations ................................................................................................................ 343
  Create Provisioned Throughput ............................................................................. 343
  Run inference using Provisioned Throughput ...................................................... 346
  Update Provisioned Throughput ............................................................................ 347
  Get Provisioned Throughput .................................................................................. 347
  Delete Provisioned Throughput ............................................................................. 348
  List Provisioned Throughput resources ................................................................. 349

**Tag resources** ....................................................................................................... 350
  Use the console ......................................................................................................... 351
  Use APIs .................................................................................................................... 351
  Tag restrictions ........................................................................................................ 351

**Amazon Titan Models** .......................................................................................... 353
  Titan Text .................................................................................................................. 353
  Titan Text Embeddings ............................................................................................ 355
  Titan Multimodal Embeddings ................................................................................ 356
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>362</td>
</tr>
<tr>
<td>Data protection</td>
<td>363</td>
</tr>
<tr>
<td>Data encryption</td>
<td>364</td>
</tr>
<tr>
<td>Use VPC</td>
<td>375</td>
</tr>
<tr>
<td>Identity and access management</td>
<td>382</td>
</tr>
<tr>
<td>Audience</td>
<td>382</td>
</tr>
<tr>
<td>Authenticating with identities</td>
<td>383</td>
</tr>
<tr>
<td>Managing access using policies</td>
<td>386</td>
</tr>
<tr>
<td>How Amazon Bedrock works with IAM</td>
<td>389</td>
</tr>
<tr>
<td>Identity-based policy examples</td>
<td>395</td>
</tr>
<tr>
<td>AWS managed policies</td>
<td>402</td>
</tr>
<tr>
<td>Allow users to perform actions on agent and alias resources</td>
<td>406</td>
</tr>
<tr>
<td>Service roles</td>
<td>408</td>
</tr>
<tr>
<td>Amazon Bedrock product IDs</td>
<td>420</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>420</td>
</tr>
<tr>
<td>Compliance validation</td>
<td>422</td>
</tr>
<tr>
<td>Incident response</td>
<td>423</td>
</tr>
<tr>
<td>Resilience</td>
<td>424</td>
</tr>
<tr>
<td>Infrastructure security</td>
<td>424</td>
</tr>
<tr>
<td>Cross-service confused deputy prevention</td>
<td>424</td>
</tr>
<tr>
<td>Configuration and vulnerability analysis in Amazon Bedrock</td>
<td>426</td>
</tr>
<tr>
<td>Monitor Amazon Bedrock</td>
<td>427</td>
</tr>
<tr>
<td>Monitor with CloudWatch</td>
<td>427</td>
</tr>
<tr>
<td>Runtime metrics</td>
<td>427</td>
</tr>
<tr>
<td>Logging CloudWatch metrics</td>
<td>428</td>
</tr>
<tr>
<td>Use CloudWatch metrics for Amazon Bedrock</td>
<td>429</td>
</tr>
<tr>
<td>View Amazon Bedrock metrics</td>
<td>429</td>
</tr>
<tr>
<td>Monitor events</td>
<td>429</td>
</tr>
<tr>
<td>How it works</td>
<td>430</td>
</tr>
<tr>
<td>EventBridge schema</td>
<td>431</td>
</tr>
<tr>
<td>Rules and targets</td>
<td>432</td>
</tr>
<tr>
<td>Create a rule to handle Amazon Bedrock events</td>
<td>433</td>
</tr>
<tr>
<td>CloudTrail logs</td>
<td>434</td>
</tr>
<tr>
<td>Amazon Bedrock information in CloudTrail</td>
<td>435</td>
</tr>
<tr>
<td>Amazon Bedrock data events in CloudTrail</td>
<td>435</td>
</tr>
<tr>
<td>Titan Image Generator</td>
<td>358</td>
</tr>
</tbody>
</table>
Amazon Bedrock management events in CloudTrail ......................................................... 437
Understanding Amazon Bedrock log file entries ................................................................ 437

**Abuse detection** ............................................................................................................. 439

**AWS PrivateLink** .......................................................................................................... 440
  Considerations .................................................................................................................. 440
  Create an interface endpoint ............................................................................................ 441
  Create an endpoint policy ................................................................................................. 441

**Quotas** ........................................................................................................................... 443
  Runtime quotas .................................................................................................................. 443
  Batch inference quotas ..................................................................................................... 445
  Agent and Knowledge base quotas ................................................................................... 445
  Model customization quotas ............................................................................................. 446
  ........................................................................................................................................... 446
  Provisioned Throughput quotas ......................................................................................... 447

**Document history** .......................................................................................................... 448

**AWS Glossary** ............................................................................................................... 451
What is Amazon Bedrock?

Amazon Bedrock is a fully managed service that makes leading foundation models from AI companies available through an API along with developer tooling to help build and scale generative AI applications.

Topics

• Access the Amazon Bedrock models
• Features of Amazon Bedrock
• Supported models in Amazon Bedrock
• Supported Regions
• Amazon Bedrock pricing

Access the Amazon Bedrock models

⚠️ Important

Amazon Bedrock offers easy access to a choice of high-performing foundation models from leading AI companies, including AI21 Labs, Anthropic, Cohere, Meta, Stability AI, and Amazon.

Before you can use any of the foundation models, you must request access to that model. If you try to use the model (with the API or within the console) before you have requested access to it, you will receive an error message. For more information, see Model access.

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 text and image models.

• **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.

• **Provisioned Throughput** – Purchase throughput to run inference on models at discounted rates.

• **Fine-tuning and Continued Pre-training** – Customize an Amazon Bedrock base model to improve its performance and create a better customer experience.

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

• **Model versioning** – Benefit from continuous updates and improvements in foundation models to enhance your applications' capabilities, accuracy, and safety.

The following capabilities are in public preview release. These features are subject to change.

• **Batch inference** – Run model inference on a large dataset of prompts. Currently available only through the API.

• **Model evaluation** – Create model evaluation jobs to evaluate the responses of 1 or models to either built-in or custom prompt datasets. To evaluate a model's response you can use either human workers or automatic metrics.

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

• **Guardrails** – Implement safeguards for your generative AI applications.

**Supported models in Amazon Bedrock**

For information about the models that Amazon Bedrock supports, see [Supported models in Amazon Bedrock](#).

**Supported Regions**

For information about the Regions that Amazon Bedrock supports, see [Amazon Bedrock endpoints and quotas](#).
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.

For more information, see Amazon Bedrock Pricing.
Supported models in Amazon Bedrock

Amazon Bedrock supports models from the following providers.

Providers

- AI21 Labs
- Amazon
- Anthropic
- Cohere
- Meta
- Stability.ai

You can use the console playgrounds to experiment with a model. Depending on the model, you can use models with on-demand throughput or provisioned throughput. In your applications, you can use the Amazon Bedrock API to make inference calls to a model. The model id you use for inference varies depending on the throughput you use (on demand or provisioned). You can also use the ListFoundationModels API operation to retrieve information about the current list of models. You can customize some models with your own data. For more information, see Custom models.

To get access to a model, see Model access.

All models are supported in the US East (N. Virginia, us-east-1) and US West (Oregon, us-west-2) regions.

- The following models are supported in Asia Pacific (Singapore, ap-southeast-1)
  - Claude v2.x
  - Claude Instant v1.x
- The following models are supported in Asia Pacific (Tokyo, ap-northeast-1)
  - Claude Instant v1.x
  - Titan Text G1 - Express v1
- The following models are supported in Europe (Frankfurt, eu-central-1)
  - Claude v2.x
  - Claude Instant v1.x
AI21 Labs

- Jurassic-2 Ultra v1
- Jurassic-2 Mid v1

More information: AI21 Studio documentation.

Amazon

- Titan Text G1 - Express v1
- Titan Text G1 - Lite v1
- Titan Embeddings G1 - Text v1
- Titan Multimodal Embeddings G1 v1
- Titan Image Generator G1 v1 (preview)


Anthropic

- Claude v1.x
- Claude v2.x
- Claude Instant v1.x

More information: Anthropic documentation.

Cohere

- Command v14.x
- Command Light v14.x
• Embed English v3.x
• Embed Multilingual v3.x

More information: Cohere documentation.

Meta

• Llama 2 Chat 13B v1
• Llama 2 Chat 70B v1
• Llama 2 13B v1
• Llama 2 70B v1

More information: Meta documentation.

Stability.ai

• Stable Diffusion XL v0.x
• Stable Diffusion XL v1.x

Set up Amazon Bedrock

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

Setup tasks

- Add model access
- Console access
- Sign up for an AWS account
- Create an administrative user
- Set up the AWS CLI
- Grant programmatic access

Add model access

⚠️ Important
Amazon Bedrock offers easy access to a choice of high-performing foundation models from leading AI companies, including AI21 Labs, Anthropic, Cohere, Meta, Stability AI, and Amazon.

Before you can use any of the foundation models, you must request access to that model. If you try to use the model (with the API or within the console) before you have requested access to it, you will receive an error message. For more information, see Model access.

Console access

To access the Amazon Bedrock console and playground:

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

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

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.

<table>
<thead>
<tr>
<th>Which user needs programmatic access?</th>
<th>To</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workforce identity (Users managed in IAM Identity Center)</td>
<td>Use temporary credentials to sign programmatic requests</td>
<td>Following the instructions for the interface that you want to use.</td>
</tr>
<tr>
<td>Which user needs programmatic access?</td>
<td>To</td>
<td>By</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>to the AWS CLI, AWS SDKs, or AWS APIs.</td>
<td></td>
<td>For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the AWS Command Line Interface User Guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the AWS SDKs and Tools Reference Guide.</td>
</tr>
<tr>
<td>IAM</td>
<td>Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.</td>
<td>Following the instructions in Using temporary credentials with AWS resources in the IAM User Guide.</td>
</tr>
<tr>
<td>Which user needs programmatic access?</td>
<td>To</td>
<td>By</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>IAM</td>
<td>(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.</td>
<td>Following the instructions for the interface that you want to use.</td>
</tr>
</tbody>
</table>

- For the AWS CLI, see [Authenticating using IAM user credentials](https://docs.aws.amazon.com/cli/latest/userguide/auth.html) in the *AWS Command Line Interface User Guide*.
- For AWS SDKs and tools, see [Authenticate using long-term credentials](https://docs.aws.amazon.com/sdk-for-python/api/latest/guide/creds-providers.html) in the *AWS SDKs and Tools Reference Guide*.
- For AWS APIs, see [Managing access keys for IAM users](https://docs.aws.amazon.com/IAM/latest/UserGuide/idENN-list-access-keys.html) in the *IAM User 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
- Edit model access
- Add model access
- Remove model access

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 its 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 **Access granted** 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:

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.
Model lifecycle

Amazon Bedrock is continuously working to bring the latest versions of foundation models that have better capabilities, accuracy, and safety. As we launch new model versions, you can test them with the Amazon Bedrock console or API, and migrate your applications to benefit from the latest model versions.

A model offered on Amazon Bedrock can be in one of these states: Active, Legacy, or End-of-Life (EOL).

- **Active**: The model provider is actively working on this version, and it will continue to get updates such as bug fixes and minor improvements.

- **Legacy**: A version is marked Legacy when there is a more recent version which provides superior performance. Amazon Bedrock will set an EOL date for Legacy versions. The EOL date may vary by model. It may also vary depending on how you use the model (for example, whether you use On-Demand or Provisioned Throughput on a base model, or Provisioned Throughput on a customized model). While you can continue to use a Legacy version, you should plan to transition to an Active version before the EOL.

- **EOL**: This version is no longer available for use. Any requests made this version will fail.

You can find the current state of a model including its EOL date (if it's a Legacy version) through Amazon Bedrock APIs, console, and documentation. When you make a `GetFoundationModel` or `ListFoundationModels` call, you can find the state of the model in the `modelLifecycle` field in the response.

On-Demand, Provisioned Throughput, and model customization

You must specify the version of a model when you use it in On-Demand mode (for example, `anthropic.claude-v2`, `anthropic.claude-v2:1`, etc.).

When you configure Provisioned Throughput, you must specify a model version that will remain unchanged for the entire term. You can purchase a new Provisioned Throughput commitment (or renew an existing one) for a version if the commitment term ends before the version's EOL date.
If you customized a model, you can continue to use it until the EOL date of the base model version that you used for customization. You can also customize a legacy model version, but you should plan to migrate before it reaches its EOL date.

**Note**

Service quotas are shared among model minor versions.

## Legacy versions

The following table shows all the Legacy versions of models available on Amazon Bedrock.

<table>
<thead>
<tr>
<th>Model version</th>
<th>Legacy date</th>
<th>EOL date</th>
<th>Details</th>
<th>Recommended model version replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan Embeddings - Text v1.1 (Preview)</td>
<td>11/7/2023</td>
<td>2/15/2024</td>
<td>• Applies to On-Demand and Provisioned Throughput modes.</td>
<td>Titan Embeddings - Text v1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Applies to base and customized models.</td>
<td></td>
</tr>
<tr>
<td>Claude v1.3</td>
<td>11/28/2023</td>
<td>2/28/2024</td>
<td>• Applies to On-Demand and Provisioned Throughput modes.</td>
<td>Claude v2.1</td>
</tr>
</tbody>
</table>
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

**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
- Set up CloudWatch Logs destination
- Using the console
- Using APIs with invocation logging

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`):

   ```json
   {"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:**"
          }
        }
      }
    ]
   }
   ```

   **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`.

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

   ```json
   ```
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**:

   ```json
   {
   "Version": "2012-10-17",
   "Statement": [
   
   ]
   }"
Role policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
        }
    ]
}
```

For more information on setting up SSE for CloudWatch Logs, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Encrypting-logs.html).
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.
Amazon Bedrock console overview

The Amazon Bedrock console provides the following features.

Features

- Getting started
- Foundation models
- Playgrounds
- Orchestration
- Assessment and deployment
- Model access
- Settings

To open the Amazon Bedrock console, sign in at https://console.aws.amazon.com/bedrock/home.

Getting started

From Getting started in the navigation pane, you can get an Overview of the foundation models, examples, and playgrounds that Amazon Bedrock provides. You can also get Examples of the prompts you can use with Amazon Bedrock models.

The examples page shows example prompts for the available models. You can search the examples and filter the list of examples using one or more of the following attributes:

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

Filter the example prompts by choosing the Search in examples edit box and then selecting the filter that you want to apply to the search. Apply multiple filters by again choosing Search in examples and then selecting another filter.

When you choose an 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.

Foundation models

From Foundation models in the navigation pane, you can view the available Base models, and group them by various attributes. You can also filter the model view, search for models, and view information about the model providers.

You can customize a base foundation model to improve the model's performance on specific tasks or teach the model a new domain of knowledge. Choose Custom models under foundation models to create and manage your custom models. Customize a model by creating a model customization job with a training dataset that you provide. For more information, see Custom models.

You can experiment with base models and custom models by using the console playgrounds.

Playgrounds

The console playgrounds are where you can experiment with models before deciding to use them in an application. There are three playgrounds.

Chat playground

The chat playground lets you experiment with the chat models that Amazon Bedrock provides. You can submit a chat to a model and the chat playground shows the response from the model and includes model metrics. Optionally, choose Compare mode to compare the output from up to three models. For more information, see Chat playground.

Text playground

The text playground lets you experiment with the text models that Amazon Bedrock provides. You can submit text to a model and the text playground shows the text that the model generates from the prompt. For more information, see Text playground.
**Image playground**

The image playground lets you experiment with the image models that Amazon Bedrock provides. You can submit a text prompt to a model and the image playground shows the image that the model generates for the prompt. For more information, see Image playground.

In the console, access the playgrounds by choosing Playgrounds in the navigation pane. For more information, see Playgrounds.

**Orchestration**

With Amazon 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. To use a knowledge base, choose Orchestration in the left navigation pane and then Knowledge base. For more information, see Knowledge base for Amazon Bedrock.

Agents for Amazon Bedrock enables developers to configure an agent to complete actions based on organization data and user input. For example you might create an agent to take actions to fulfill a customer's request. To use an Agent, choose Orchestration in the left navigation pane and then Agent. For more information, see Agents for Amazon Bedrock.

**Assessment and deployment**

As you use Amazon Bedrock models, you need to assess their performance and to deploy them into your solutions.

With Model Evaluation, you can evaluate and compare model output, and then choose the one best suited for your applications. Choose Assessment and deployment and then choose Model evaluation.

When you configure Provisioned Throughput for a model, you receive a level of throughput at a fixed cost. To provision throughput, choose Assessment and deployment in the navigation pane and then Provisioned Throughput. For more information, see Provisioned Throughput.

**Model access**

To use a model in Amazon Bedrock, you must first request access to the model. On the left navigation pane, choose Model access. For more information, see Model access.
Settings

You can manage your account level settings in the Amazon Bedrock console. The settings include data logs, data permissions, and model access. To access settings, choose Settings in the left navigation pane. For more information, see Settings.
Playgrounds

⚠️ Important
Amazon Bedrock offers easy access to a choice of high-performing foundation models from leading AI companies, including AI21 Labs, Anthropic, Cohere, Meta, Stability AI, and Amazon.
Before you can use any of the foundation models, you must request access to that model. If you try to use the model (with the API or within the console) before you have requested access to it, you will receive an error message. For more information, see Model access.

The Amazon Bedrock console playgrounds are where you can experiment with models before deciding to use them in an application. In the console, access the playgrounds by choosing Playgrounds in the left navigation pane. You can also navigate directly to the playground when you choose a model from a model details page or the examples page.

There are playgrounds for text, chat, and image models.

Within each playground you can enter prompts and experiment with inference parameters. Prompts are usually one or more sentences of text that set up a scenario, question, or task for a model. For information about creating prompts, see Prompt engineering guidelines.

Inference parameters influence the response generated by a model, such as the randomness of generated text. When you load a model into a playground, the playground configures the model with its default inference settings. You can change and reset the settings as you experiment with the model. Each model has its own set of inference parameters. For more information, see Inference parameters for foundation models.

When you submit a response, the model responds with its generated output.

If a chat or text model supports streaming, the default is to stream the responses from a model. You can turn off streaming, if desired.

Topics
- Chat playground
- Text playground
- Image playground
Use a playground

Chat playground

The chat playground lets you experiment with the chat models that Amazon Bedrock provides. You can submit a chat to a model and the chat playground shows the response from the model, and the following metrics.

- **Latency** — The time it takes for the model to generate each token (word) in a sequence.
- **Input token count** — The number of tokens that are fed into the model as input during inference.
- **Output token count** — The number of tokens generated in response to a prompt. Longer, more conversational, responses require more tokens.
- **Cost** — The cost of processing the input and generating output tokens.

You can also define criteria that you want the model response to match.

By turning on compare model, you can compare the chat responses for a single prompt with the responses from up to three models. This helps you to understand the comparative performance of each model, without having to switch between models. For more information, see Use a playground.

Text playground

The text playground lets you experiment with the text models that Amazon Bedrock provides. You can submit text to a model and the text playground shows the text that the model generates from the prompt.

Image playground

The image playground lets you experiment with the image models that Amazon Bedrock provides. You can submit a text prompt to a model and the image playground shows the image that the model generates for the prompt.

Along with setting inference parameters, you can make additional configuration changes (differs by model):
• **Mode** – The model generates a new image or edits an existing image that you supply in **Reference image**.

• **Negative prompt** – items or concepts that you don't want the model to generate, such as *cartoon* or *violence*.

• **Reference image** – The image on which to generate the response or that you want the model to edit.

• **Response image** – Output settings for the generated image, such as quality, orientation, size, and the number of images to generate.

• **Advanced configurations** – The inference parameters to pass to the model.

**Use a playground**

The following procedure shows how to submit a prompt to a playground and view the response. In each playground, you can configure the inference parameters for the model. In the **chat playground**, you can view metrics, and optionally compare the output of up to three models. In the **image playground** you can make advanced configuration changes, which also vary by model.

**To use a playground**

1. If you haven't already, request access to the models that you want to use. For more information, see [Model access](#).

2. Open the Amazon Bedrock console.

3. From the navigation pane, under **Playgrounds**, choose **Chat**, **Text**, or **Image**.

4. Choose **Select model** to open the **Select model** dialog box.
   a. In **Category** select from the available providers or custom models.
   b. In **Model** select a model.
   c. In **Throughput** select the throughput (on-demand, or provisioned throughput) that you want the model to use. If you are using a custom model, you must have set up Provisioned Throughput for the model beforehand. For more information, see [Provisioned Throughput](#).
   d. Choose **Apply**.

5. (Optional) In **Configurations** choose the inference parameters that you want to use. For more information, see [Inference parameters for foundation models](#). For information about configuration changes you can make in the image playground, see [Image playground](#).
6. Enter your prompt into the text field and choose Run. A prompt is a natural language phrase or command, such as **Tell me about the best restaurants to visit in Seattle.** For more information, see _Prompt engineering guidelines_.

**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.
For information about prompt engineering, see _Prompt engineering guidelines_.

7. If you are using the chat playground, view the model metrics and compare models by doing the following.

   a. In the **Model metrics** section, view the metrics for each model.

   b. (Optional) Define criteria that you want to match by doing the following:

      i. Choose **Define metric criteria**.

      ii. For the metrics you want to use, choose the condition and value. You can set the following conditions:

         - **less than** – The metric value is less than the specified value.
         - **greater than** – the metric value is more than the specified value.

      iii. Choose **Apply** to apply your criteria.

      iv. View which criteria are met. If all criteria are met, the **Overall summary** is **Meets all criteria**. If 1 or more criteria are not met, the **Overall summary** is **n criteria unmet** and the unmet criteria are highlighted in red.

   c. (Optional) Add models to compare by doing the following:

      i. Turn on **Compare mode**.

      ii. Choose **Select model** to select a model.

      iii. In the dialog box, choose a provider, model, and throughput.

      iv. Choose **Apply**.
v. (Optional) Choose the menu icon next to each model to configure inference parameters for that model. For more information, see [Inference parameters for foundation models](#).

vi. Chooses the + icon on the right of the Chat playground section to add a second or third model to compare.

vii. Repeat steps a-c to choose the models that you want to compare.

viii. Enter your a prompt into the text field and choose Run.
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
- Amazon Bedrock API operations
Setting up the Amazon Bedrock API

Important
Amazon Bedrock offers easy access to a choice of high-performing foundation models from leading AI companies, including AI21 Labs, Anthropic, Cohere, Meta, Stability AI, and Amazon.
Before you can use any of the foundation models, you must request access to that model. If you try to use the model (with the API or within the console) before you have requested access to it, you will receive an error message. For more information, see Model access.

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”.
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 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.

```python
{
```
"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.

```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:",
})
```
"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.

```python
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](#)
- [Get details about a base model](#)
- [Run inference](#)
- [Tag resources](#)

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

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

Python (Boto)

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

```python
import boto3
bedrock = boto3.client(service_name='bedrock')
bekrock.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:

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

Python (Boto)

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

```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. To run inference on a custom model, first purchase Provisioned Throughput for it (for more information, see Provisioned Throughput).

- **Base model** – Call ListFoundationModels to find the model ARN. For an example, see List the base models. To read a list of the model IDs for the currently available base models, see Base model IDs. You can also get the model ID for a foundation model from the JSON examples in the Amazon Bedrock 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 throughput 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.

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
- Run batch inference
- Base model IDs

**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.

**Note**

The AWS CLI does not support streaming.

For information about the parameters each model supports, see Inference parameters for foundation models. For information about writing prompts, see Prompt engineering guidelines.
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 \n  --body "{"prompt": "Human: story of two dogs\n\nAssistant:", "max_tokens_to_sample": 300}" \n  --cli-binary-format raw-in-base64-out \
  invoke-model-output.txt
```

The following example shows how to call the *Llama 2 Chat 13B* model.

```
aws bedrock-runtime invoke-model \
  --region us-east-1 \n  --model-id meta.llama2-13b-chat-v1 \n  --body "{"prompt": "What is the average lifespan of a Llama?", "max_gen_len": 128, "temperature": 0.1, "top_p": 0.9}" \
  invoke-model-output.txt
```

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:

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

body = json.dumps({
    "prompt": "Human: 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 = brt.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

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

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

response = brt.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
• Cohere Command
• Meta Llama 2
• Stability AI Diffusion XL

A2I Jurassic-2

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

```python
import boto3
import json

brt = 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 = brt.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'))
```

Cohere Command

This examples shows how to call the Cohere Command model.
import boto3
import json

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

body = json.dumps(
    {
        "prompt": "How do you tie a tie?",
        "max_tokens": 200,
        "temperature": 0.5,
        "p": 0.5
    }
)

modelId = 'cohere.command-text-v14'
accept = 'application/json'
contentType = 'application/json'

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

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

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

Meta Llama 2

This example shows how to call the Llama 2 Chat 13B model.

import boto3
import json

bedrock = boto3.client(service_name='bedrock-runtime', region_name='us-east-1')

body = json.dumps(
    {
        "prompt": "What is the average lifespan of a Llama?",
        "max_gen_len": 128,
        "temperature": 0.1,
        "top_p": 0.9,
    }
)
Stability AI Diffusion XL

The following example shows how to run inference with the Stability.ai Diffusion 1.0 model and on demand throughput. The example submits a text prompt to a model, retrieves the response from the model, and finally shows the image.

```python
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

""
Shows how to generate an image with SDXL 1.0 (on demand).
""
import base64
import io
import json
import logging
import boto3
from PIL import Image
from botocore.exceptions import ClientError

class ImageError(Exception):
    """Custom exception for errors returned by SDXL""
    def __init__(self, message):
        self.message = message

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

def generate_image(model_id, body):
    """
    Generate an image using SDXL 1.0 on demand.
    ""
```
Args:
    model_id (str): The model ID to use.
    body (str): The request body to use.
Returns:
    image_bytes (bytes): The image generated by the image.

logger.info("Generating image with SDXL model %s", model_id)

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

accept = "application/json"
content_type = "application/json"

response = bedrock.invoke_model(
    body=body, modelId=model_id, accept=accept, contentType=content_type
)
response_body = json.loads(response.get("body").read())
print(response_body['result'])

base64_image = response_body.get("artifacts")[0].get("base64")
base64_bytes = base64_image.encode('ascii')
image_bytes = base64.b64decode(base64_bytes)

finish_reason = response_body.get("artifacts")[0].get("finishReason")

if finish_reason == 'ERROR' or finish_reason == 'CONTENT_FILTERED':
    raise ImageError(f"Image generation error. Error code is {finish_reason}")

logger.info("Successfully generated image withvthe SDXL 1.0 model %s", model_id)

return image_bytes

def main():
    
    """
    Entrypoint for SDXL example.
    """

    logging.basicConfig(level = logging.INFO,
                        format = "%(levelname)s: %(message)s")
model_id='stability.stable-diffusion-xl-v1'

prompt='"Sri lanka tea plantation."'

body=json.dumps(
    {
        "text_prompts": [
            {
                "text": prompt
            }
        ],
        "cfg_scale": 10,
        "seed": 0,
        "steps": 50,
        "samples": 1,
        "style_preset": "photographic"
    }
)

try:
    image_bytes=generate_image(model_id = model_id,
                                 body = body)
    image = Image.open(io.BytesIO(image_bytes))
    image.show()

except ClientError as err:
    message=err.response["Error"]['Message']
    logger.error("A client error occurred: %s", message)
    print("A client error occurred: " +
          format(message))
except ImageError as err:
    logger.error(err.message)
    print(err.message)

else:
    print(f"Finished generating text with SDXL model {model_id}.")

if __name__ == "__main__":
    main()
Run batch inference

Note
Batch inference is in preview and is subject to change. Batch inference is currently only available through the API. Access batch APIs through the following SDKs.

- AWS SDK for Python.
- AWS SDK for Java.

With batch inference, you can run multiple inference requests asynchronously to process a large number of requests efficiently by running inference on data that is stored in an S3 bucket. You can use batch inference to improve the performance of model inference on large datasets.

To see quotas for batch inference, see Batch inference quotas.

Amazon Bedrock supports batch inference on the following modalities.

- Text to embeddings
- Text to text
- Text to image
- Image to image

You store your data in an Amazon S3 bucket to prepare it for batch inference. You can then carry out and manage batch inference jobs through using the ModelInvocationJob APIs.

Before you can carry out batch inference, you must receive permissions to call the batch inference APIs. You then configure an IAM Amazon Bedrock service role to have permissions to carry out batch inference jobs.

You can use the batch inference APIs by downloading and installing one of the following AWS SDK packages.

- AWS SDK for Python.
- AWS SDK for Java.

Topics
Set up permissions for batch inference

To set up a role for batch inference, create an IAM role by following the steps at Creating a role to delegate permissions to an AWS service. Attach the following policies to the role:

- Trust policy
- Access to the Amazon S3 buckets containing the input data for your batch inference jobs and to write the output data.

1. The following policy allows Amazon Bedrock to assume this role and carry out batch inference jobs. 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. Use the ArnEquals or ArnLike condition to restrict the scope.

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

Note

As a best practice for security purposes, replace the * with specific batch inference job IDs after you have created them.
2. Attach the following policy to allow Amazon Bedrock to access the S3 bucket containing input data for your batch inference jobs (replace `my_input_bucket`) and the S3 bucket to write output data to (replace `my_output_bucket`). Replace the `account-id` with the account ID of the user to whom you are providing S3 bucket access permissions.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my_input_bucket",
        "arn:aws:s3:::my_input_bucket/**",
        "arn:aws:s3:::my_output_bucket",
        "arn:aws:s3:::my_output_bucket/**"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": ["account-id"
        ]
      }
    }
  ]
}
```
Format and upload your inference data

Upload JSONL files containing the data to input to the model to your S3 bucket with the following format. Each line should match the following format and is a different item for inference. If you leave the recordId field out, Amazon Bedrock adds it in the output.

Note

The format of the modelInput JSON object should match the body field for the model that you use in the InvokeModel request. For more information, see Inference parameters for foundation models.

```
{ "recordId" : "12 character alphanumeric string", "modelInput" : {JSON body} }
...
```

For example, you might provide an JSONL file containing the following data and run batch inference on a Titan text model.

```
{ "recordId" : "3223593EFGH", "modelInput" : {"inputText": "Roses are red, violets are"} }
{ "recordId" : "1223213ABCD", "modelInput" : {"inputText": "Hello world"} }
```

Create a batch inference job

Request format

```
POST /model-invocation-job HTTP/1.1
Content-type: application/json

{
    "clientRequestToken": "string",
    "inputDataConfig": {
        "s3InputDataConfig": {
            "s3Uri": "string",
            "s3InputDataFormat": "application/json"
        }
    }
}
```
To create a batch inference job, send a `CreateModelInvocationJob` request. Provide the following information.

- The ARN of a role with permissions to run batch inference in `roleArn`.
- Information for the S3 bucket containing the input data in `inputDataConfig` and the bucket where to write information in `outputDataConfig`.
- The ID of the model to use for inference in `modelId` (see Base model IDs).
- A name for the job in `jobName`.
- (Optional) Any tags that you want to attach to the job in `tags`.

The response returns a `jobArn` that you can use for other batch inference-related API calls.
You can check the status of the job with either the GetModelInvocationJob or ListModelInvocationJobs APIs.

When the job is Completed, you can extract the results of the batch inference job from the files in the S3 bucket you specified in the request for the outputDataConfig. The S3 bucket contains the following files:

1. Output files containing the result of the model inference.
   - If the output is text, Amazon Bedrock generates an output JSONL file for each input JSONL file. The output files contain outputs from the model for each input in the following format. An error object replaces the modelOutput field in any line where there was an error in inference. The format of the modelOutput JSON object matches the body field for the model that you use in the InvokeModel response. For more information, see Inference parameters for foundation models.

   ```json
   { "recordId" : "12 character alphanumeric string", "modelInput": {JSON body}, "modelOutput": {JSON body} }
   ```

   The following example shows a possible output file.

   ```json
   { "recordId" : "3223593EFGH", "modelInput" : {"inputText": "Roses are red, violets are"}, "modelOutput" : {"inputTextTokenCount": 8, 'results': [{"tokenCount": 3, 'outputText': 'blue\n', 'completionReason': 'FINISH'}]} }
   { "recordId" : "1223213ABCDE", "modelInput" : {"inputText": "Hello world"}, "error" : {"errorCode" : 400, "errorMessage" : "bad request" } }
   ```

   - If the output is image, Amazon Bedrock generates a file for each image.

2. A manifest.json.out file containing a summary of the batch inference job.

```json
{
    "processedRecordCount" : number,
    "successRecordCount" : number,
    "errorRecordCount" : number,
    "inputTextTokenCount" : number, // For embedding/text to text models
    "outputTextTokenCount" : number, // For text to text models
    "outputImgCount512x512pStep50" : number, // For text to image models
    "outputImgCount512x512pStep150" : number, // For text to image models
    "outputImgCount512x896pStep50" : number, // For text to image models
    "outputImgCount512x896pStep150" : number // For text to image models
}
```
**Stop a batch inference job**

**Request format**

```plaintext
POST /model-invocation-job/jobIdentifier/stop HTTP/1.1
```

**Response format**

```plaintext
HTTP/1.1 200
```

To stop a batch inference job, send a `StopModelInvocationJob` and provide the ARN of the job in the `jobIdentifier` field.

If the job was successfully stopped, you receive an HTTP 200 response.

**Get details about a batch inference job**

**Request format**

```plaintext
GET /model-invocation-job/jobIdentifier HTTP/1.1
```

**Response format**

```plaintext
HTTP/1.1 200
Content-type: application/json
```

```json
{
   "clientRequestToken": "string",
   "endTime": "string",
   "inputDataConfig": {
      "s3InputDataConfig": {
         "s3Uri": "string",
         "s3InputFormat": "JSONL"
      }
   },
   "jobArn": "string",
   "jobName": "string",
   "lastModifiedTime": "string",
}
```
To get information about a batch inference job, send a GetModelInvocationJob and provide the ARN of the job in the jobIdentifier field.

See the GetModelInvocationJob page for details about the information provided in the response.

**List batch inference jobs**

**Request format**

```
GET /model-invocation-jobs?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken&sortBy=sortBy&sortOrder=sortOrder&statusEquals=statusEquals&submitTimeAfter=submitTimeAfter&submitTimeBefore=submitTimeBefore
HTTP/1.1
```

**Response format**

```
HTTP/1.1 200
Content-type: application/json

{
   "invocationJobSummaries": [
      {
         "clientRequestToken": "string",
         "endTime": "string",
         "inputDataConfig": {
            "s3InputDataConfig": {
               "s3Uri": "string",
               "s3InputFormat": "JSONL"
            }
         }
      }
   ]
}
```
To get information about a batch inference job, send a `ListModelInvocationJobs`. You can set the following specifications.

- Filter for results by specifying the status, submission time, or substrings in the name of the job. You can specify the following statuses.
  - Submitted
  - InProgress
  - Completed
  - Failed
  - Stopping
  - Stopped
- Sort by the time that the job was created (`CreationTime`). You can sort in Ascending or Descending order.
- The maximum number of results to return in a response. If there are more results than the number you set, the response returns a `nextToken` that you can send in another `ListModelInvocationJobs` request to see the next batch of jobs.

The response returns a list of `InvocationJobSummary` objects.
Code samples

Select a language to see a code sample to call the batch inference API operations.

Python

Install the Python SDK containing the Amazon Bedrock control plane APIs by running the following commands in a terminal.

```bash
python3 -m pip install /path/to/botocore-1.32.6-py3-none-any.whl
python3 -m pip install /path/to/boto3-1.29.6-py3-none-any.whl
```

Create a batch inference job with a file named `abc.jsonl` that you uploaded to S3. Write the output to a bucket in `s3://output-bucket/output/`. Get the `jobArn` from the response.

```python
import boto3

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

inputDataConfig={
    "s3InputDataConfig": {
        "s3Uri": "s3://input-bucket/input/abc.jsonl"
    }
}

outputDataConfig={
    "s3OutputDataConfig": {
        "s3Uri": "s3://output-bucket/output/"
    }
}

response=bedrock.create_model_invocation_job(
    roleArn="arn:aws:iam::123456789012:role/MyBatchInferenceRole",
    modelId="amazon.titan-text-express-v1",
    jobName="my-batch-job",
    inputDataConfig=inputDataConfig,
    outputDataConfig=outputDataConfig
)

jobArn = response.get('jobArn')

Return the status of the job.
List batch inference jobs that *Failed*.

```java
bedrock.list_model_invocation_jobs(
    maxResults=10,
    statusEquals="Failed",
    sortOrder="Descending"
)
```

Stop the job that you started.

```java
bedrock.stop_model_invocation_job(jobIdentifier=jobArn)
```

Java

```java
package com.amazon.aws.sample.bedrock.inference;

import com.amazonaws.services.bedrock.AmazonBedrockAsync;
import com.amazonaws.services.bedrock.AmazonBedrockAsyncClientBuilder;
import com.amazonaws.services.bedrock.model.CreateModelInvocationJobRequest;
import com.amazonaws.services.bedrock.model.CreateModelInvocationJobResult;
import com.amazonaws.services.bedrock.model.GetModelInvocationJobRequest;
import com.amazonaws.services.bedrock.model.GetModelInvocationJobResult;
import com.amazonaws.services.bedrock.model.InvocationJobInputDataConfig;
import com.amazonaws.services.bedrock.model.InvocationJobOutputDataConfig;
import com.amazonaws.services.bedrock.model.InvocationJobS3InputDataConfig;
import com.amazonaws.services.bedrock.model.InvocationJobS3OutputDataConfig;
import com.amazonaws.services.bedrock.model.ListModelInvocationJobsRequest;
import com.amazonaws.services.bedrock.model.ListModelInvocationJobsResult;
import com.amazonaws.services.bedrock.model.StopModelInvocationJobRequest;
import com.amazonaws.services.bedrock.model.StopModelInvocationJobResult;

public class BedrockAsyncInference {
    private final AmazonBedrockAsync amazonBedrockAsyncClient =
            AmazonBedrockAsyncClientBuilder.defaultClient();
    public void createModelInvokeJobSampleCode() {

        final InvocationJobS3InputDataConfig invocationJobS3InputDataConfig = new InvocationJobS3InputDataConfig()
                .withS3Uri("s3://Input-bucket-name/input/abc.jsonl")
                .withS3InputFormat("JSONL");
```
final InvocationJobInputDataConfig inputDataConfig = new InvocationJobInputDataConfig()
    .withS3InputDataConfig(invocationJobS3InputDataConfig);

final InvocationJobS3OutputDataConfig invocationJobS3OutputDataConfig = new InvocationJobS3OutputDataConfig()
    .withS3Uri("s3://output-bucket-name/output/");

final InvocationJobOutputDataConfig invocationJobOutputDataConfig = new InvocationJobOutputDataConfig()
    .withS3OutputDataConfig(invocationJobS3OutputDataConfig);

final CreateModelInvocationJobRequest createModelInvocationJobRequest = new CreateModelInvocationJobRequest()
    .withModelId("anthropic.claude-v2")
    .withJobName("unique-job-name")
    .withClientRequestToken("Client-token")
    .withInputDataConfig(inputDataConfig)
    .withOutputDataConfig(invocationJobOutputDataConfig);

final CreateModelInvocationJobResult createModelInvocationJobResult =
    amazonBedrockAsyncClient
    .createModelInvocationJob(createModelInvocationJobRequest);

    System.out.println(createModelInvocationJobResult.getJobArn());
}

g public void getModelInvokeJobSampleCode() {
    final GetModelInvocationJobRequest getModelInvocationJobRequest = new GetModelInvocationJobRequest()
        .withJobIdentifier("jobArn");

    final GetModelInvocationJobResult getModelInvocationJobResult =
        amazonBedrockAsyncClient
        .getModelInvocationJob(getModelInvocationJobRequest);
}

g public void listModelInvokeJobSampleCode() {
    final ListModelInvocationJobsRequest listModelInvocationJobsRequest = new ListModelInvocationJobsRequest()
```java
final ListModelInvocationJobsResult listModelInvocationJobsResult =
    amazonBedrockAsyncClient
    .listModelInvocationJobs(listModelInvocationJobsRequest);
}

public void stopModelInvokeJobSampleCode() {
    final StopModelInvocationJobRequest stopModelInvocationJobRequest = new
        StopModelInvocationJobRequest()
        .withJobIdentifier("jobArn");

    final StopModelInvocationJobResult stopModelInvocationJobResult =
        amazonBedrockAsyncClient
        .stopModelInvocationJob(stopModelInvocationJobRequest);
}
```
<table>
<thead>
<tr>
<th>Provider</th>
<th>Model name</th>
<th>Version</th>
<th>Model Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>Titan Text G1 - Express</td>
<td>1.x</td>
<td>amazon.titan-text-express-v1</td>
</tr>
<tr>
<td>Amazon</td>
<td>Titan Multimodal Embeddings G1</td>
<td>1.x</td>
<td>amazon.titan-embed-image-v1</td>
</tr>
<tr>
<td>Amazon</td>
<td>Titan Image Generator G1</td>
<td>1.x</td>
<td>amazon.titan-image-generator-v1</td>
</tr>
<tr>
<td>Anthropic</td>
<td>Claude</td>
<td>1.x</td>
<td>anthropic.claude-v1</td>
</tr>
<tr>
<td>Anthropic</td>
<td>Claude</td>
<td>2.x</td>
<td>anthropic.claude-v2. For Claude 2.1, use anthropic.claude-v2:1</td>
</tr>
<tr>
<td>Anthropic</td>
<td>Claude Instant</td>
<td>1.x</td>
<td>anthropic.claude-instant-v1</td>
</tr>
<tr>
<td>Cohere</td>
<td>Command</td>
<td>14.x</td>
<td>cohere.command-text-v14</td>
</tr>
<tr>
<td>Cohere</td>
<td>Command Light</td>
<td>15.x</td>
<td>cohere.command-light-text-v14</td>
</tr>
<tr>
<td>Cohere</td>
<td>Embed English</td>
<td>3.x</td>
<td>cohere.embed-english-v3</td>
</tr>
<tr>
<td>Cohere</td>
<td>Embed Multilingual</td>
<td>3.x</td>
<td>cohere.embed-multilingual-v3</td>
</tr>
<tr>
<td>Meta</td>
<td>Llama 2 Chat 13B</td>
<td>1.x</td>
<td>meta.llama2-13b-chat-v1</td>
</tr>
<tr>
<td>Meta</td>
<td>Llama 2 Chat 70B</td>
<td>1.x</td>
<td>meta.llama2-70b-chat-v1</td>
</tr>
<tr>
<td>Stability AI</td>
<td>Stable Diffusion XL</td>
<td>0.x</td>
<td>stability.stable-diffusion-xl-v0</td>
</tr>
<tr>
<td>Provider</td>
<td>Model name</td>
<td>Version</td>
<td>Model Id</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Stability AI</td>
<td>Stable Diffusion XL</td>
<td>1.x</td>
<td>stability.stable-diffusion-xl-v1</td>
</tr>
</tbody>
</table>

**Note**

Stable Diffusion XL 1.x is only available with Provisioned Throughput. For more information, see [Provisioned Throughput](#). Stable Diffusion XL 0.x is a preview release.

**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
- Batch inference jobs (API only)
- Agents
- Agent aliases
- Knowledge bases
- Model evaluations (Console only)

If you use the API, you must call the correct `TagResource` and `UntagResource` operations.

- The following resources use the Amazon Bedrock `TagResource` and `UntagResource` operations.
  - Custom models
  - Model customization jobs
  - Provisioned models
  - Batch inference jobs
• The following resources use the Agents for Amazon Bedrock TagResource and UntagResource operations.
  • Agents
  • Agent aliases
  • Knowledge bases

For more information about restrictions on tagging, see Tag restrictions.

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:

```python
import boto3

bedrock = boto3.client(service_name='bedrock')
```
The following example demonstrates how to remove the tags with the keys `key1` and `key2` from a resource using Python:

```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:

```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 Amazon Bedrock. Optionally, set inference parameters to influence the response generated by the model. You set inference parameters in a playground in the console, or in the body field of the InvokeModel or InvokeModelWithResponseStream API.

When you call a model, you also include a prompt for the model. For information about writing prompts, see Prompt engineering guidelines.

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
- Amazon Titan text models
- Amazon Titan image models
- Anthropic Claude models
- AI21 Labs Jurassic-2 models
- Cohere models
- Meta Llama 2 and Llama 2 Chat models
- Stability.ai Diffusion models

Inference parameter definitions

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

Topics
- Randomness and diversity
- Length
- Repetitions
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 to 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 to 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 text models

The Amazon Titan models support the following inference parameters for text models.

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.
For more information on Amazon Titan Text prompt engineering guidelines, see Amazon Titan Text Prompt Engineering Guidelines.

For more information on Titan models, see Amazon Titan models.

**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).

**Model invocation request body field**

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.

```json
{
    "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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON field format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomness and diversity</td>
<td>Temperature</td>
<td>temperature</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Length
Model invocation response body field

The InvokeModel response returns a body field in the following format:

```json
{
    'inputTextTokenCount': int,
    'results': [
        {
            'tokenCount': int,
            'outputText': '\n<response>\n',
            'completionReason': string
        }
    ]
}
```

More information about each field is provided below.

- **inputTextTokenCount** – The number of tokens in the prompt.
- **tokenCount** – The number of tokens in the response.
- **outputText** – The text in the response.
- **completionReason** – The reason the response finished being generated. The following reasons are possible.
  - **FINISHED** – The response was fully generated.
  - **LENGTH** – The response was truncated because of the response length you set.

The InvokeModelWithResponseStream response returns a response stream object in the body field. Each chunk of text in the response stream is in the following format. You must decode the bytes field (see Running inference on a model for an example).

```json
{
    'chunk': {
    ```
• index – The index of the chunk in the streaming response.
• inputTextTokenCount – The number of tokens in the prompt.
• totalOutputTextTokenCount – The number of tokens in the response.
• outputText – The text in the response.
• completionReason – The reason the response finished being generated. The following reasons are possible.
  • FINISHED – The response was fully generated.
  • LENGTH – The response was truncated because of the response length you set.

Amazon Titan image models

The Amazon Titan models support the following inference parameters for image generator models.

Randomness and Diversity

The Amazon Titan models support the following parameters to control prompt strength, generation steps, and seed. To produce a more random image from the prompt, use a lower value for prompt strength and a higher value for generation steps and seed. To produce a more exact image from the prompt, use a higher value for prompt strength. To recreate a similar image later, make note of your inference parameter settings and your exact image prompt. Use exactly the same prompt and parameter settings to produce similar images.

• Prompt strength (cfgscale) – Determines how much the final image portray the prompts.
• Seed (seed) – Determines the initial noise. Using the same seed with the same settings and prompt will create similar images.
• Number of images (numberOfImages) – The number of times the image is sampled and produced in the output image.
- **Quality** (quality) – Choose Standard or Premium image output for image quality.

For more information on Titan Image Generator prompt engineering, see Amazon Titan Image Generator Prompt Engineering Best Practices.

For more information on Titan models, see Amazon Titan models.

**Model invocation request body fields**

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 `prompt` field.

**Text to Image**

The model accepts a text prompt as input and generate a new image as output. The generated image captures the concepts described by the text prompt.

```
{
  "taskType": "TEXT_IMAGE",
  "textToImageParams": {
    "text": "prompt", // Required
    "negativeText": "text" // Optional
  },
  "imageGenerationConfig": {
    "numberOfImages": 1, // Range: 1 to 5
    "quality": "standard", // Options: standard/premium
    "height": 1024, // Supported height max
    "width": 1024, // Supported width max
    "cfgScale": 8.0, // Range: 1.0 (exclusive) to 10.0
    "seed": 0 // Range: 0 to 214783647
  }
}
```

**Inpainting**

Inpainting uses an image and a segmentation mask as input (from either the user or estimated by the model) and reconstructs the region within the mask.
Outpainting

Outpainting uses an image and a segmentation mask as input (from either the user or estimated by the model) and generates new pixels that seamlessly extend the region.

Model invocation request body fields
Image Variation

Image variation uses an image and an optional text prompt as input. It generates a new image that preserves the content of the input image, but variates its style and background.

```
{
  "taskType": "IMAGE_VARIATION",
  "imageVariationParams": {
    "text": "Modernize the house, photo-realistic, 8k, hdr", // Optional
    "negativeText": "bad quality, low resolution, cartoon", // Optional
    "images": [input_image], // One image is required
  },
  "imageGenerationConfig": {
    "numberOfImages": 1,
    "quality": "premium",
    "height": 512,
    "width": 512,
    "cfgScale": 8.0
  }
}
```

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON field format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine tuning and hyperparameters</td>
<td>Step size</td>
<td>stepSize</td>
<td>500</td>
<td>16,000</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>Learning rate</td>
<td></td>
<td>1.00E-07</td>
<td>1</td>
<td>1.00E-05</td>
</tr>
</tbody>
</table>
### 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.

#### Model invocation request body field

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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON field format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>batchSize</td>
<td></td>
<td>64</td>
<td>192</td>
<td>64</td>
</tr>
</tbody>
</table>
Note

To avoid timeouts with the Claude v2.1 model, we recommend limiting the input token count in the prompt field to 180K. We expect to address this timeout issue soon.

```json
{
  "prompt": "\n\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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON object format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomness and diversity</td>
<td>Temperature</td>
<td>temperature</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Top P</td>
<td>top_p</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Top K</td>
<td>top_k</td>
<td>0</td>
<td>500</td>
<td>250</td>
</tr>
<tr>
<td>Length</td>
<td>Max tokens to sample</td>
<td>max_tokens_to_sample</td>
<td>0</td>
<td>4096</td>
<td>200</td>
</tr>
</tbody>
</table>

Model invocation response body field

For information about the format of the body field in the response, see https://docs.anthropic.com/claude/reference/complete_post.

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** \( \text{temperature} \) – Use a lower value to decrease randomness in the response.
- **Top P** \( \text{topP} \) – 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** \( \text{maxTokens} \) – Specify the maximum number of tokens to use in the generated response.
- **Stop sequences** \( \text{stopSequences} \) – 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** \( \text{presencePenalty} \) – 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** \( \text{countPenalty} \) – 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** \( \text{frequencyPenalty} \) – 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 \text{true}.
  - **Whitespaces** \( \text{applyToWhitespaces} \) – A \text{true} value applies the penalty to whitespaces and new lines.
- **Punctuations** (applyToPunctuations) – 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.

**Model invocation request body field**

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.

```json
{
    "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.

```json
{
    "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.

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

**Model invocation response body field**

For information about the format of the body field in the response, see [https://docs.ai21.com/reference/j2-complete-ref](https://docs.ai21.com/reference/j2-complete-ref).
The following is inference parameters information for the Cohere models that Amazon Bedrock supports.

Topics
- Cohere Command models
- Cohere Embed models

Cohere Command models

The Cohere Command models support the following controls.

- **Return likelihoods** (`return_likelihoods`) – (optional) Specify how and if the token likelihoods are returned with the response. You can specify the following options.
  - GENERATION – Only return likelihoods for generated tokens.
  - ALL – Return likelihoods for all tokens.
  - NONE – (Default) Don't return any likelihoods.
- **Stream** (`stream`) – (Required to support streaming) Specify `true` to return the response piece-by-piece in real-time and `false` to return the complete response after the process finishes.
- **Logit bias** (`logit_bias`) – (Optional) prevents the model from generating unwanted tokens or incentivizes the model to include desired tokens. The format is `{token_id: bias}` where bias is a float between -10 and 10. Tokens can be obtained from text using any tokenization service, such as Cohere's Tokenize endpoint. For more information, see [Cohere documentation](#).
- **Number of generations** (`num_generations`) – (Optional) The maximum number of generations that the model should return.
- **Truncate** (`truncate`) – (Optional) Specifies how the API handles inputs longer than the maximum token length. Use one of the following:
  - NONE – Returns an error when the input exceeds the maximum input token length.
  - START – Discard the start of the input.
  - END – (Default) Discards the end of the input.

If you specify START or END, the model discards the input until the remaining input is exactly the maximum input token length for the model.
Randomness and diversity

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

- **Temperature** (temperature) – (Optional) Use a lower value to decrease randomness in the response.
- **Top P** (p) – (Optional) 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) – (Optional) 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 models support the following parameters to control the length of the generated response.

- **Maximum length** (max_tokens) – (Optional) Specify the maximum number of tokens to use in the generated response.
- **Stop sequences** (stop_sequences) – (Optional) 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.

Model invocation request body field

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. The prompt field is required.

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

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON object format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomness and diversity</td>
<td>Temperature</td>
<td>temperature</td>
<td>0</td>
<td>5</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Top P</td>
<td>p</td>
<td>0</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Top K</td>
<td>k</td>
<td>0</td>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td>Length</td>
<td>Max tokens</td>
<td>max_tokens</td>
<td>1</td>
<td>4,096</td>
<td>20</td>
</tr>
<tr>
<td>Other</td>
<td>Number of generations</td>
<td>num_generations</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Logit bias</td>
<td>logit_bias</td>
<td>-10 (for a token bias)</td>
<td>10 (for a token bias)</td>
<td>No default value</td>
</tr>
</tbody>
</table>

**Model invocation response body field**

The body response from a call to InvokeModel or InvokeModelWithResponseStream is the following:

```json
{
    "generations": [
        {
            "finish_reason": "COMPLETE | MAX_TOKENS | ERROR | ERROR_TOXIC",
            "id": "string",
            "text": "string",
            "prompt": "string",
            "error": "string",
            "error_toxic": "string"
        }
    ]
}
```
The response has the following possible fields:

- **generations** — A list of generated results along with the likelihoods for tokens requested. (Always returned). Each generation object in the list contains the following fields.
  - **id** — An identifier for the generation. (Always returned).
  - **likelihood** — The likelihood of the output. The value is the average of the token likelihoods in `token_likelihoods`. Returned if you specify the `return_likelihoods` input parameter.
  - **token_likelihoods** — An array of per token likelihoods. Returned if you specify the `return_likelihoods` input parameter.
  - **finish_reason** — states the reason why the model finished generating tokens. COMPLETE - the model sent back a finished reply. MAX_TOKENS – the reply was cut off because the model reached the maximum number of tokens for its context length. ERROR – something went wrong when generating the reply. ERROR_TOXIC – the model generated a reply that was deemed toxic. `finish_reason` is returned only when `is_finished=true`. (Not always returned).
  - **is_finished** — A boolean field used only when `stream` is true, signifying whether or not there are additional tokens that will be generated as part of the streaming response. (Not always returned)
  - **text** — The generated text.
  - **index** — In a streaming response, use to determine which generation a given token belongs to. When only one response is streamed, all tokens belong to the same generation and index is not returned. `index` therefore is only returned in a streaming request with a value for `num_generations` that is larger than one.
  - **prompt** — The prompt from the input request. (Always returned).
  - **id** — An identifier for the request (always returned).
Cohere Embed models

The Cohere Embed models support the following controls.

- **texts** (texts) – (Required) An array of strings for the model to embed. For optimal performance, we recommend reducing the length of each text to less than 512 tokens. 1 token is about 4 characters.

- **input type** (input_type) – (Required) Prepends special tokens to differentiate each type from one another. You should not mix different types together, except when mixing types for for search and retrieval. In this case, embed your corpus with the search_document type and embedded queries with type search_query type.
  - **search_document** – In search use-cases, use search_document when you encode documents for embeddings that you store in a vector database.
  - **search_query** – Use search_query when querying your vector DB to find relevant documents.
  - **classification** – Use classification when using embeddings as an input to a text classifier.
  - **clustering** – Use clustering to cluster the embeddings.

- **Truncate** (truncate) – (Optional) Specifies how the API handles inputs longer than the maximum token length. Use one of the following:
  - **NONE** – (Default) Returns an error when the input exceeds the maximum input token length.
  - **LEFT** – Discard the start of the input.
  - **RIGHT** – Discards the end of the input.

  If you specify LEFT or RIGHT, the model discards the input until the remaining input is exactly the maximum input token length for the model.

Model invocation request body field

When you make an **InvokeModel** 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.

```json
{
    "texts": ["string"],
    // Other fields as needed
}
```
"input_type": "search_document|search_query|classification|clustering",
"truncate": "NONE|LEFT|RIGHT"
}

For more information, see https://docs.cohere.com/reference/embed.

⚠️ **Note**

Amazon Bedrock doesn't support streaming responses from Cohere Embed models.

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON object format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td>Texts</td>
<td>texts</td>
<td>0 texts per call</td>
<td>128 texts per call</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Text length</td>
<td>texts</td>
<td>0 characters</td>
<td>2048 characters</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Model invocation response body field**

The body response from a call to InvokeModel is the following:

```json
{
    "embeddings": [
        [float //*1024])
    ],
    "id": "string",
    "texts": ["string"]
}
```

The body response has the following fields:
• **id** – An identifier for the response.

• **embeddings** – An array of embeddings, where each embedding is an array of floats with 1024 elements. The length of the embeddings array will be the same as the length of the original texts array.

• **texts** – An array containing the text entries for which embeddings were returned.

For more information, see [https://docs.cohere.com/reference/embed](https://docs.cohere.com/reference/embed).

### Meta Llama 2 and Llama 2 Chat models

The Meta Llama 2 Chat and Meta Llama 2 models support the following controls.

#### Topics

- **Randomness and diversity**
- **Length**
- **Model invocation request body field**
- **Model invocation response body field**

#### Randomness and diversity

The Llama 2 Chat and Llama 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** *(top_p)* – Use a lower value to ignore less probable options. Set to 0 or 1.0 to disable.

#### Length

The Llama 2 Chat and Llama 2 models support the following parameters to control the length of the generated response.

- **Maximum length** *(max_gen_len)* – Specify the maximum number of tokens to use in the generated response. The model truncates the response once the generated text exceeds max_gen_len.
**Model invocation request body field**

When you make an `InvokeModel` or `InvokeModelWithResponseStream` call using a Llama 2 model, fill the body field with a JSON object that conforms to the one below. Enter the prompt in the `prompt` field.

```json
{
  "prompt": "string",
  "temperature": float,
  "top_p": float,
  "max_gen_len": int
}
```

For example code, see [Meta Llama 2](#).

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>JSON object format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomness and diversity</td>
<td>Temperature</td>
<td>temperature</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Top P</td>
<td>top_p</td>
<td>0</td>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>Length</td>
<td>Max generation length</td>
<td>max_gen_len</td>
<td>1</td>
<td>2048</td>
<td>512</td>
</tr>
</tbody>
</table>

**Model invocation response body field**

The `InvokeModel` response returns a body field in the following format:

```json
{
  "generation": "\n\n<response>" ,
  "prompt_token_count": int ,
  "generation_token_count": int ,
  "stop_reason" : "string"
}
```
More information about each field is provided below.

- **generation** – The generated text.
- **prompt_token_count** – The number of tokens in the prompt.
- **generation_token_count** – The number of tokens in the generated text.
- **stop_reason** – The reason why the response stopped generating text. Possible values are:
  - **stop** – The model has finished generating text for the input prompt.
  - **length** – The length of the tokens for the generated text exceeds the value of `max_gen_len` in the call to `InvokeModel` (or `InvokeModelWithResponseStream`, if you are streaming output). The response is truncated to `max_gen_len` tokens. Consider increasing the value of `max_gen_len` and trying again.

**Stability.ai Diffusion models**

The following is inference parameters information for the Stability.ai Diffusion models that Amazon Bedrock supports.

**Models**

- [Stability.ai Diffusion 0.8](#)
- [Stability.ai Diffusion 1.0 text to image](#)
- [Stability.ai Diffusion 1.0 image to image](#)
- [Stability.ai Diffusion 1.0 image to image (masking)](#)

**Stability.ai Diffusion 0.8**

The Stability.ai Diffusion models have 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.
Model invocation request body field

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.

```json
{
    "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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>JSON object format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt strength</td>
<td>cfg_scale</td>
<td>0</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Generation step</td>
<td>steps</td>
<td>10</td>
<td>150</td>
<td>30</td>
</tr>
</tbody>
</table>

Model invocation response body field

For information about the format of the body field in the response, see https://platform.stability.ai/docs/api-reference#tag/v1generation.

Stability.ai Diffusion 1.0 text to image

The Stability.ai Diffusion 1.0 model has the following inference parameters and model response for making text to image inference calls. For example code, see Stability AI Diffusion XL.

Topics
- Request and Response
- Code example
Request and Response

The request body is passed in the body field of a request to InvokeModel or InvokeModelWithResponseStream.

For more information, see https://platform.stability.ai/docs/api-reference#tag/v1generation.

Request

The Stability.ai Diffusion 1.0 model has the following inference parameters for a text to image inference call.

```json
{
    "text_prompts": [
        {
            "text": string,
            "weight": float
        }
    ],
    "height": int,
    "width": int,
    "cfg_scale": float,
    "clip_guidance_preset": string,
    "sampler": string,
    "samples",
    "seed": int,
    "steps": int,
    "style_preset": string,
    "extras" : JSON object
}
```

- **text_prompts** (Required) – An array of text prompts to use for generation. Each element is a JSON object that contains a prompt and a weight for the prompt.
- **text** – The prompt that you want to pass to the model.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2000</td>
</tr>
</tbody>
</table>
• **weight** (Optional) – The weight that the model should apply to the prompt. A value that is less than zero declares a negative prompt. Use a negative prompt to tell the model to avoid certain concepts. The default value for weight is one.

• **cfg_scale** – (Optional) Determines how much the final image portrays the prompt. Use a lower number to increase randomness in the generation.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35</td>
<td>7</td>
</tr>
</tbody>
</table>

• **clip_guidance_preset** – (Optional) Enum: FAST_BLUE, FAST_GREEN, NONE, SIMPLE SLOW, SLOWER, SLOWEST.

• **height** – (Optional) Height of the image to generate, in pixels, in an increment divisible by 64.

  The value must be one of 1024x1024, 1152x896, 1216x832, 1344x768, 1536x640, 640x1536, 768x1344, 832x1216, 896x1152.

• **width** – (Optional) Width of the image to generate, in pixels, in an increment divisible by 64.

  The value must be one of 1024x1024, 1152x896, 1216x832, 1344x768, 1536x640, 640x1536, 768x1344, 832x1216, 896x1152.

• **sampler** – (Optional) The sampler to use for the diffusion process. If this value is omitted, the model automatically selects an appropriate sampler for you.

  Enum: DDIM, DDPM, K_DPMPP_2M, K_DPMPP_2S_ANCESTRAL, K_DPM_2, K_DPM_2_ANCESTRAL, K_EULER, K_EULER_ANCESTRAL, K_HEUN, K_LMS.

• **samples** – (Optional) The number of images to generate. Currently Amazon Bedrock supports generating one image. If you supply a value for samples, the value must be one.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• **seed** – (Optional) 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, or the value is 0, it is set as a random number.
• **steps** – (Optional) Generation step determines how many times the image is sampled. More steps can result in a more accurate result.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4294967295</td>
<td>0</td>
</tr>
</tbody>
</table>

• **style_preset** (Optional) – A style preset that guides the image model towards a particular style. This list of style presets is subject to change.


• **extras** (Optional) – Extra parameters passed to the engine. Use with caution. These parameters are used for in-development or experimental features and might change without warning.

**Response**

The Stability.ai Diffusion 1.0 model returns the following fields for a text to image inference call.

```
{
    "result": string,
    "artifacts": [
    {
        "seed": int,
        "base64": string,
        "finishReason": string
    }
    ]
}
```

• **result** – The result of the operation. If successful, the response is *success*. 
• **artifacts** – An array of images, one for each requested image.
  • **seed** – The value of the seed used to generate the image.
  • **base64** – The base64 encoded image that the model generated.
  • **finishedReason** – The result of the image generation process. Valid values are:
    • **SUCCESS** – The image generation process succeeded.
    • **ERROR** – An error occurred.
    • **CONTENT FILTERED** – The content filter filtered the image and the image might be blurred.

**Code example**

The following example shows how to run inference with the Stability.ai Diffusion 1.0 model and on demand throughput. The example submits a text prompt to a model, retrieves the response from the model, and finally shows the image.

```python
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Shows how to generate an image with SDXL 1.0 (on demand).
"""

import base64
import io
import json
import logging
import boto3
import Image
from PIL import Image

from botocore.exceptions import ClientError

class ImageError(Exception):
  """Custom exception for errors returned by SDXL"
  def __init__(self, message):
    self.message = message

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)
```

Stability.ai Diffusion 1.0 text to image
def generate_image(model_id, body):
    """
    Generate an image using SDXL 1.0 on demand.
    Args:
        model_id (str): The model ID to use.
        body (str): The request body to use.
    Returns:
        image_bytes (bytes): The image generated by the image.
    """

    logger.info("Generating image with SDXL model %s", model_id)

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

    accept = "application/json"
    content_type = "application/json"

    response = bedrock.invoke_model(
        body=body, modelId=model_id, accept=accept, contentType=content_type
    )
    response_body = json.loads(response.get("body").read())
    print(response_body['result'])

    base64_image = response_body.get("artifacts")[0].get("base64")
    base64_bytes = base64_image.encode('ascii')
    image_bytes = base64.b64decode(base64_bytes)

    finish_reason = response_body.get("artifacts")[0].get("finishReason")

    if finish_reason == 'ERROR' or finish_reason == 'CONTENT_FILTERED':
        raise ImageError(f"Image generation error. Error code is {finish_reason}")

    logger.info("Successfully generated image with the SDXL 1.0 model %s", model_id)

    return image_bytes

def main():
    """
    Entry point for SDXL example.
    """

Stability.ai Diffusion 1.0 text to image
logging.basicConfig(level = logging.INFO, 
        format = "%(levelname)s: %(message)s"
)

model_id='stability.stable-diffusion-xl-v1'

prompt="""Sri lanka tea plantation."""

body=json.dumps({
    "text_prompts": [
        {
            "text": prompt
        }
    ],
    "cfg_scale": 10,
    "seed": 0,
    "steps": 50,
    "samples": 1,
    "style_preset": "photographic"
})

try:
    image_bytes=generate_image(model_id = model_id, 
                               body = body)
    image = Image.open(io.BytesIO(image_bytes))
    image.show()

except ClientError as err:
    message=err.response["Error"]["Message"]
    logger.error("A client error occurred: %s", message)
    print("A client error occurred: " +
          format(message))
except ImageError as err:
    logger.error(err.message)
    print(err.message)

else:
    print(f"Finished generating text with SDXL model {model_id}.")

if __name__ == "__main__":
    main()
Stability.ai Diffusion 1.0 image to image

The Stability.ai Diffusion 1.0 model has the following inference parameters and model response for making image to image inference calls.

Request and Response

The request body is passed in the body field of a request to InvokeModel or InvokeModelWithResponseStream.

For more information, see https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/imageToImage.

Request

The Stability.ai Diffusion 1.0 model has the following inference parameters for an image to image inference call.

```json
{
    "text_prompts": [
        {
            "text": string,
            "weight": float
        }
    ],
    "init_image": string,
    "init_image_mode": string,
    "image_strength": float,
    "cfg_scale": float,
    "clip_guidance_preset": string,
    "sampler": string,
    "samples": int,
    "seed": int,
    "steps": int,
    "style_preset": string,
    "extras": json object
}
```

The following are required parameters.
- **text_prompts** – (Required) An array of text prompts to use for generation. Each element is a JSON object that contains a prompt and a weight for the prompt.

- **text** – The prompt that you want to pass to the model.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2000</td>
</tr>
</tbody>
</table>

- **weight** – (Optional) The weight that the model should apply to the prompt. A value that is less than zero declares a negative prompt. Use a negative prompt to tell the model to avoid certain concepts. The default value for weight is one.

- **init_image** – (Required) The base64 encoded image that you want to use to initialize the diffusion process.

The following are optional parameters.

- **init_image_mode** – (Optional) Determines whether to use `image_strength` or `step_schedule_*` to control how much influence the image in `init_image` has on the result. Possible values are `IMAGE_STRENGTH` or `STEP_SCHEDULE`. The default is `IMAGE_STRENGTH`.

- **image_strength** – (Optional) Determines how much influence the source image in `init_image` has on the diffusion process. Values close to 1 yield images very similar to the source image. Values close to 0 yield images very different than the source image.

- **cfg_scale** – (Optional) Determines how much the final image portrays the prompt. Use a lower number to increase randomness in the generation.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>35</td>
</tr>
</tbody>
</table>

- **clip_guidance_preset** – (Optional) Enum: FAST_BLUE, FAST_GREEN, NONE, SIMPLE, SLOW, SLOWER, SLOWEST.

- **sampler** – (Optional) The sampler to use for the diffusion process. If this value is omitted, the model automatically selects an appropriate sampler for you.
Enum: DDIM, DDPM, K_DPMPP_2M, K_DPMPP_2S_ANCESTRAL, K_DPM_2, K_DPM_2_ANCESTRAL, K_EULER, K_EULER_ANCESTRAL, K_HEUN, K_LMS.

- **samples** – (Optional) The number of image to generate. Currently Amazon Bedrock supports generating one image. If you supply a value for samples, the value must be one.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **seed** – (Optional) 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, or the value is 0, it is set as a random number.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

- **steps** – (Optional) Generation step determines how many times the image is sampled. More steps can result in a more accurate result.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

- **style_preset** – (Optional) A style preset that guides the image model towards a particular style. This list of style presets is subject to change.


- **extras** – (Optional) Extra parameters passed to the engine. Use with caution. These parameters are used for in-development or experimental features and might change without warning.
Response

The Stability.ai Diffusion 1.0 model returns the following fields for a text to image inference call.

```
{
    "result": string,
    "artifacts": [
        {
            "seed": int,
            "base64": string,
            "finishReason": string
        }
    ]
}
```

- **result** – The result of the operation. If successful, the response is `success`.
- **artifacts** – An array of images, one for each requested image.
  - **seed** – The value of the seed used to generate the image.
  - **base64** – The base64 encoded image that the model generated.
  - **finishReason** – The result of the image generation process. Valid values are:
    - **SUCCESS** – The image generation process succeeded.
    - **ERROR** – An error occurred.
    - **CONTENT_FILTERED** – The content filter filtered the image and the image might be blurred.

**Stability.ai Diffusion 1.0 image to image (masking)**

The Stability.ai Diffusion 1.0 model has the following inference parameters and model response for using masks with image to image inference calls.

**Request and Response**

The request body is passed in the body field of a request to `InvokeModel` or `InvokeModelWithResponseStream`.

For more information, see [https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/masking](https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/masking).
Request

The Stability.ai Diffusion 1.0 model has the following inference parameters for an image to image (masking) inference call.

```json
{
    "text_prompts": [
        {
            "text": string,
            "weight": float
        }
    ],
    "init_image": string,
    "mask_source": string,
    "mask_image": string,
    "cfg_scale": float,
    "clip_guidance_preset": string,
    "sampler": string,
    "samples": int,
    "seed": int,
    "steps": int,
    "style_preset": string,
    "extras": json object
}
```

The following are required parameters.

- **text_prompt** – (Required) An array of text prompts to use for generation. Each element is a JSON object that contains a prompt and a weight for the prompt.
- **text** – The prompt that you want to pass to the model.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2000</td>
</tr>
</tbody>
</table>

- **weight** – (Optional) The weight that the model should apply to the prompt. A value that is less than zero declares a negative prompt. Use a negative prompt to tell the model to avoid certain concepts. The default value for weight is one.
- **init_image** – (Required) The base64 encoded image that you want to use to initialize the diffusion process.
• **mask_source** – (Required) Determines where to source the mask from. Possible values are:
  
  • **MASK_IMAGE_WHITE** – Use the white pixels of the mask image in `mask_image` as the mask. White pixels are replaced and black pixels are left unchanged.
  
  • **MASK_IMAGE_BLACK** – Use the black pixels of the mask image in `mask_image` as the mask. Black pixels are replaced and white pixels are left unchanged.
  
  • **INIT_IMAGE_ALPHA** – Use the alpha channel of the image in `init_image` as the mask, Fully transparent pixels are replaced and fully opaque pixels are left unchanged.
  
  • **mask_image** – (Required) The base64 encoded mask image that you want to use as a mask for the source image in `init_image`. Must be the same dimensions as the source image. Use the mask_source option to specify which pixels should be replaced.

The following are optional parameters.

• **cfg_scale** – (Optional) Determines how much the final image portrays the prompt. Use a lower number to increase randomness in the generation.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>35</td>
</tr>
</tbody>
</table>

• **clip_guidance_preset** – (Optional) Enum: FAST_BLUE, FAST_GREEN, NONE, SIMPLE, SLOW, SLOWER, SLOWEST.

• **sampler** – (Optional) The sampler to use for the diffusion process. If this value is omitted, the model automatically selects an appropriate sampler for you.

  Enum: DDIM, DDPM, K_DPMPP_2M, K_DPMPP_2S_ANCESTRAL, K_DPM_2, K_DPM_2_ANCESTRAL, K_EULER, K_EULER_ANCESTRAL, K_HEUN K_LMS.

• **samples** – (Optional) The number of image to generate. Currently Amazon Bedrock supports generating one image. If you supply a value for samples, the value must be one. generates

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
- **seed** – (Optional) 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, or the value is 0, it is set as a random number.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

- **steps** – (Optional) Generation step determines how many times the image is sampled. More steps can result in a more accurate result.

<table>
<thead>
<tr>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

- **style_preset** – (Optional) A style preset that guides the image model towards a particular style. This list of style presets is subject to change.


- **extras** – (Optional) Extra parameters passed to the engine. Use with caution. These parameters are used for in-development or experimental features and might change without warning.

Response

The Stability.ai Diffusion 1.0 model returns the following fields for a text to image inference call.

```json
{
  "result": string,
  "artifacts": [ 
    {
      "seed": int,
      "base64": string,
      "finishReason": string
    }
  ]
}
```
• **result** – The result of the operation. If successful, the response is `success`.
• **artifacts** – An array of images, one for each requested image.
  • **seed** – The value of the seed used to generate the image.
  • **base64** – The base64 encoded image that the model generated.
• **finishedReason** – The result of the image generation process. Valid values are:
  • **SUCCESS** – The image generation process succeeded.
  • **ERROR** – An error occurred.
  • **CONTENT_FILTERED** – The content filter filtered the image and the image might be blurred.
Prompt engineering guidelines

Topics

• Introduction
• What is a prompt?
• What is prompt engineering?
• General guidelines for Amazon Bedrock LLM users
• Prompt templates and examples for Amazon Bedrock text models

Introduction

Welcome to the prompt engineering guide for large language models (LLMs) on Amazon Bedrock. Amazon Bedrock is Amazon’s service for foundation models (FMs), which offers access to a range of powerful FMs for text and images.

Prompt engineering refers to the practice of optimizing textual input to LLMs to obtain desired responses. Prompting helps LLMs perform a wide variety of tasks, including classification, question answering, code generation, creative writing, and more. The quality of prompts that you provide to LLMs can impact the quality of their responses. These guidelines provide you with all the necessary information to get started with prompt engineering. It also covers tools to help you find the best possible prompt format for your use case when using LLMs on Amazon Bedrock.

Whether you’re a beginner in the world of generative AI and language models, or an expert with previous experience, these guidelines can help you optimize your prompts for Amazon Bedrock text models. Experienced users can skip to the General Guidelines for Amazon Bedrock LLM Users or Prompt Templates and Examples for Amazon Bedrock Text Models sections.

⚠️ Note

All examples in this doc are obtained via API calls. The response may vary due to the stochastic nature of the LLM generation process. If not otherwise specified, the prompts are written by employees of AWS.

Disclaimer: The examples in this document use the current text models available within Amazon Bedrock. Also, this document is for general prompting guidelines. For model-specific guides, refer
to their respective docs on Amazon Bedrock. This document provides a starting point. While the following example responses are generated using specific models on Amazon Bedrock, you can use other models in Amazon Bedrock to get results as well. The results may differ between models as each model has its own performance characteristics. The output that you generate using AI services is your content. Due to the nature of machine learning, output may not be unique across customers and the services may generate the same or similar results across customers.

**Additional prompt resources**

The following resources offer additional guidelines on prompt engineering.

- **Claude model prompt guide:** [https://docs.anthropic.com/claude/docs](https://docs.anthropic.com/claude/docs)
- **Claude prompt engineering resources:** [https://docs.anthropic.com/claude/docs/guide-to-anthropic-prompt-engineering-resources](https://docs.anthropic.com/claude/docs/guide-to-anthropic-prompt-engineering-resources)
- **Cohere prompt guide:** [https://txt.cohere.com/how-to-train-your-pet-llm-prompt-engineering](https://txt.cohere.com/how-to-train-your-pet-llm-prompt-engineering)
- **AI21 Jurassic model prompt guide:** [https://docs.ai21.com/docs/prompt-engineering](https://docs.ai21.com/docs/prompt-engineering)
- **Llama 2 prompt guide:** [https://ai.meta.com/llama/get-started/#prompting](https://ai.meta.com/llama/get-started/#prompting)
- **Stability documentation:** [https://platform.stability.ai/docs/getting-started](https://platform.stability.ai/docs/getting-started)

**What is a prompt?**

Prompts are a specific set of inputs provided by you, the user, that guide LLMs on Amazon Bedrock to generate an appropriate response or output for a given task or instruction.

**User Prompt:**

*Who invented the airplane?*

When queried by this prompt, Amazon Titan provides an output:

**Output:**

*The Wright brothers, Orville and Wilbur Wright are widely credited with inventing and manufacturing the world's first successful airplane.*

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)
Components of a prompt

A single prompt includes several components, such as the task or instruction you want the LLMs to perform, the context of the task (for example, a description of the relevant domain), demonstration examples, and the input text that you want LLMs on Amazon Bedrock to use in its response. Depending on your use case, the availability of the data, and the task, your prompt should combine one or more of these components.

Consider this example prompt asking Amazon Titan to summarize a review:

**User Prompt:**
The following is text from a restaurant review:

“I finally got to check out Alessandro’s Brilliant Pizza and it is now one of my favorite restaurants in Seattle. The dining room has a beautiful view over the Puget Sound but it was surprisingly not crowded. I ordered the fried castelvetrano olives, a spicy Neapolitan-style pizza and a gnocchi dish. The olives were absolutely decadent, and the pizza came with a smoked mozzarella, which was delicious. The gnocchi was fresh and wonderful. The waitstaff were attentive, and overall the experience was lovely. I hope to return soon.”

Summarize the above restaurant review in one sentence.

(Source of prompt: AWS)

Based on this prompt, Amazon Titan responds with a succinct one-line summary of the restaurant review. The review mentions key facts and conveys the main points, as desired.

**Output:**
Alessandro’s Brilliant Pizza is a fantastic restaurant in Seattle with a beautiful view over Puget Sound, decadent and delicious food, and excellent service.

(Model used: Amazon Titan Text G1 - Express)

The instruction **Summarize the above restaurant review in one sentence** and the review text **I finally got to check out** ... were both necessary for this type of output. Without either one, the model would not have enough information to produce a sensible summary. The **instruction** tells the LLM what to do, and the text is the **input** on which the LLM operates.
The context (The following is text from a restaurant review) provides additional information and keywords that guide the model to use the input when formulating its output.

In the example below, the text Context: Climate change threatens people with increased flooding ... is the input which the LLM can use to perform the task of answering the question Question: What organization calls climate change the greatest threat to global health in the 21st century?“.

User prompt:
Context: Climate change threatens people with increased flooding, extreme heat, increased food and water scarcity, more disease, and economic loss. Human migration and conflict can also be a result. The World Health Organization (WHO) calls climate change the greatest threat to global health in the 21st century. Adapting to climate change through efforts like flood control measures or drought-resistant crops partially reduces climate change risks, although some limits to adaptation have already been reached. Poorer communities are responsible for a small share of global emissions, yet have the least ability to adapt and are most vulnerable to climate change. The expense, time required, and limits of adaptation mean its success hinge on limiting global warming.

Question: What organization calls climate change the greatest threat to global health in the 21st century?

(Source of prompt: https://en.wikipedia.org/wiki/Climate_change)

AI21 Jurassic responses with the correct name of the organization according to the context provided in the prompt.

Output:
The World Health Organization (WHO) calls climate change the greatest threat to global health in the 21st century.

(Model used: AI21 Jurassic-2 Ultra v1)

Few-shot prompting vs. zero-shot prompting

It is sometimes useful to provide a few examples to help LLMs better calibrate their output to meet your expectations, also known as few-shot prompting or in-context learning, where a shot corresponds to a paired example input and the desired output. To illustrate, first here is an example
of a zero-shot sentiment classification prompt where no example input-output pair is provided in the prompt text:

**User prompt:**
Tell me the sentiment of the following headline and categorize it as either positive, negative or neutral:
New airline between Seattle and San Francisco offers a great opportunity for both passengers and investors.

(Source of prompt: AWS)

**Output:**
Positive

(Model used: Amazon Titan Text G1 - Express)

Here is the few-shot version of a sentiment classification prompt:

**User prompt:**
Tell me the sentiment of the following headline and categorize it as either positive, negative or neutral. Here are some examples:

Research firm fends off allegations of impropriety over new technology.  
Answer: Negative

Offshore windfarms continue to thrive as vocal minority in opposition dwindles.  
Answer: Positive

Manufacturing plant is the latest target in investigation by state officials.  
Answer:

(Source of prompt: AWS)

**Output:**
Negative

(Model used: Amazon Titan Text G1 - Express)

The following example uses Claude models. When using Claude models, it's a good practice to use `<example></example>` tags to include demonstration examples. We also recommend using different delimiters such as `H:` and `A:` in the examples to avoid confusion with the delimiters.
Human: and Assistant: for the whole prompt. Notice that for the last few-shot example, the final A: is left off in favor of Assistant:, prompting Claude to generate the answer instead.

**User prompt:**

Human: Please classify the given email as "Personal" or "Commercial" related emails. Here are some examples.

<example>
H: Hi Tom, it's been long time since we met last time. We plan to have a party at my house this weekend. Will you be able to come over?
A: Personal
</example>

<example>
H: Hi Tom, we have a special offer for you. For a limited time, our customers can save up to 35% of their total expense when you make reservations within two days. Book now and save money!
A: Commercial
</example>

H: Hi Tom, Have you heard that we have launched all-new set of products. Order now, you will save $100 for the new products. Please check our website.

Assistant:

**Output:**
Commercial

(Source of prompt: AWS, model used: Claude)

**Prompt template**

A prompt template specifies the formatting of the prompt with exchangeable content in it. Prompt templates are “recipes” for using LLMs for different use cases such as classification, summarization, question answering, and more. A prompt template may include instructions, few-shot examples, and specific context and questions appropriate for a given use case. The following example is a template that you can use to perform few-shot sentiment classification using Amazon Bedrock text models:
"Tell me the sentiment of the following

{{Text Type, e.g., “restaurant review”}} and categorize it as either {{Sentiment A}} or {{Sentiment B}}.

Here are some examples:

Text: {{Example Input 1}}
Answer: {{Sentiment A}}

Text: {{Example Input 2}}
Answer: {{Sentiment B}}

Text: {{Input}}
Answer:"

Users can copy and paste this template, filling in their own text and few-shot examples to complete prompts when using LLMs on Amazon Bedrock. More templates are provided in this section.

Note

The double curly braces {{  and  }} mark the places to put in data-specific information in the template, and should not be included in the prompt text.

Important notes on using Amazon Bedrock LLMs by API calls

If you are accessing LLMs on Amazon Bedrock via API calls, please be aware that they do not recall prior prompts and previous requests unless the previous interaction is included within the current prompt. You may need this for conversational-style interactions or follow up requests.

If you are accessing Claude models using API calls instead of the Amazon Bedrock text playground, you wrap your prompts in a conversational style to get desired response (for the Amazon Titan Text and AI21 Jurassic model family, it's not necessary). The following example demonstrates a conversational-style prompt:

User prompt:

Human: Describe where sea otters live in two sentences.
Assistant:

Output:
Here is a two sentence description of where sea otters live:

Sea otters live in the northern Pacific Ocean along the coasts of Russia, Alaska, Canada, and the continental United States. They prefer coastal habitats like rocky shores, kelp forests, and estuaries where they have access to shallow, nearshore waters.

(Source of prompt: AWS, model used: Claude)

Notice how the main content of the prompt is wrapped like this:

Human: {{Main Content}}

Assistant:. For Claude models, prompts sent via the API must contain

Human: and  

Assistant:.

To use conversational mode on Amazon Titan, you can use the format of User: {{}}  
Bot: when prompting the model.

What is prompt engineering?

Prompt engineering refers to the practice of crafting and optimizing input prompts by selecting appropriate words, phrases, sentences, punctuation, and separator characters to effectively use LLMs for a wide variety of applications. In other words, prompt engineering is the art of communicating with an LLM. High-quality prompts condition the LLM to generate desired or better responses. The detailed guidance provided within this document is applicable across all LLMs within Amazon Bedrock.

The best prompt engineering approach for your use case is dependent on both the task and the data. Common tasks supported by LLMs on Amazon Bedrock include:

- **Classification**: The prompt includes a question with several possible choices for the answer, and the model must respond with the correct choice. An example classification use case is sentiment analysis: the input is a text passage, and the model must classify the sentiment of the text, such as whether it's positive or negative, or harmless or toxic.

- **Question-answer, without context**: The model must answer the question with its internal knowledge without any context or document.

- **Question-answer, with context**: The user provides an input text with a question, and the model must answer the question based on information provided within the input text.
• **Summarization**: The prompt is a passage of text, and the model must respond with a shorter passage that captures the main points of the input.

• **Open-ended text generation**: Given a prompt, the model must respond with a passage of original text that matches the description. This also includes the generation of creative text such as stories, poems, or movie scripts.

• **Code generation**: The model must generate code based on user specifications. For example, a prompt could request text-to-SQL or Python code generation.

• **Mathematics**: The input describes a problem that requires mathematical reasoning at some level, which may be numerical, logical, geometric or otherwise.

• **Reasoning or logical thinking**: The model must make a series of logical deductions.

## General guidelines for Amazon Bedrock LLM users

### Design your prompt

Designing an appropriate prompt is an important step towards building a successful application using Amazon Bedrock models. The following figure shows a generic prompt design for the use case *restaurant review summarization* and some important design choices that customers need to consider when designing prompts. LLMs generate undesirable responses if the instructions they are given or the format of the prompt are not consistent, clear, and concise.
A good example of prompt construction

The following is text from a restaurant review:

“I finally got to check out Alessandro’s Brilliant Pizza and it is now one of my favorite restaurants in Seattle. The dining room has a beautiful view over the Puget Sound but it was surprisingly not crowded. I ordered the fried Castelvetrano olives, a spicy Neapolitan-style pizza and a gnocchi dish. The olives were absolutely decadent, and the pizza came with a smoked mozzarella, which was delicious. The gnocchi was fresh and wonderful. The waitstaff were attentive, and overall the experience was lovely. I hope to return soon.”

Summarize the above restaurant review in one sentence.

(Source: Prompt written by AWS)

Use inference parameters

LLMs on Amazon Bedrock all come with several inference parameters that you can set to control the response from the models. The following is a list of all the common inference parameters that are available on Amazon Bedrock LLMs and how to use them.

Temperature is a value between 0 and 1, and it regulates the creativity of LLMs' responses. Use lower temperature if you want more deterministic responses, and use higher temperature if you want more creative or different responses for the same prompt from LLMs on Amazon Bedrock. For all the examples in this prompt guideline, we set temperature = 0.

Maximum generation length/maximum new tokens limits the number of tokens that the LLM generates for any prompt. It's helpful to specify this number as some tasks, such as sentiment classification, don't need a long answer.

Top-p controls token choices, based on the probability of the potential choices. If you set Top-p below 1.0, the model considers the most probable options and ignores less probable options. The result is more stable and repetitive completions.
**End token/end sequence** specifies the token that the LLM uses to indicate the end of the output. LLMs stop generating new tokens after encountering the end token. Usually this doesn't need to be set by users.

There are also model-specific inference parameters. Claude models have an additional Top-k inference parameter, and AI21 Jurassic models come with a set of inference parameters including presence penalty, count penalty, frequency penalty, and special token penalty. For more information, refer to their respective documentation.

**Detailed guidelines**

**Provide simple, clear, and complete instructions**

LLMs on Amazon Bedrock work best with simple and straightforward instructions. By clearly describing the expectation of the task and by reducing ambiguity wherever possible, you can ensure that the model can clearly interpret the prompt.

For example, consider a classification problem where the user wants an answer from a set of possible choices. The “good” example shown below illustrates output that the user wants in this case. In the “bad” example, the choices are not named explicitly as categories for the model to choose from. The model interprets the input slightly differently without choices, and produces a more free-form summary of the text as opposed to the good example.

**Good example, with output**

**User prompt:**
"The most common cause of color blindness is an inherited problem or variation in the functionality of one or more of the three classes of cone cells in the retina, which mediate color vision."

**What is the above text about?**

a) biology  
b) history  
c) geology

**Output:**

a) biology

**Bad example, with output**

**User prompt:**
Classify the following text. "The most common cause of color blindness is an inherited problem or variation in the functionality of one or more of the three classes of cone cells in the retina, which mediate color vision."

**Output:**

The topic of the text is the causes of colorblindness.
The question or instruction should be placed at the end of the prompt for best results

Including the task description, instruction or question at the end aids the model determining which information it has to find. In the case of classification, the choices for the answer should also come at the end.

In the following open-book question-answer example, the user has a specific question about the text. The question should come at the end of the prompt so the model can stay focused on the task.

**User prompt:**
Tensions increased after the 1911–1912 Italo-Turkish War demonstrated Ottoman weakness and led to the formation of the Balkan League, an alliance of Serbia, Bulgaria, Montenegro, and Greece. The League quickly overran most of the Ottomans' territory in the Balkans during the 1912–1913 First Balkan War, much to the surprise of outside observers. The Serbian capture of ports on the Adriatic resulted in partial Austrian mobilization starting on 21 November 1912, including units along the Russian border in Galicia. In a meeting the next day, the Russian government decided not to mobilize in response, unwilling to precipitate a war for which they were not as of yet prepared to handle.

Which country captured ports?

**Output:**
Serbia

**Use separator characters for API calls**

Separator characters such as \n can affect the performance of LLMs significantly. For Claude models, it's necessary to include newlines when formatting the API calls to obtain desired responses. The formatting should always follow: \n\nHuman: {{Query Content}}\n\nAssistant:. For Amazon Titan models, adding \n at the end of a prompt helps improve the performance of the model. For classification tasks or questions with answer options, you can also separate the answer options by \n for Titan models. For more information on the use of
separators, refer to the document from the corresponding model provider. The following example is a template for a classification task.

**Prompt template:**

```
"""{{Text}}

{{Question}}

{{Choice 1}}
{{Choice 2}}
{{Choice 3}}""
```

The following example shows how the presence of newline characters between choices and at the end of a prompt helps Amazon Titan produce the desired response.

**User prompt:**

Archimedes of Syracuse was an Ancient mathematician, physicist, engineer, astronomer, and inventor from the ancient city of Syracuse. Although few details of his life are known, he is regarded as one of the leading scientists in classical antiquity.

What was Archimedes? Choose one of the options below.

a) astronomer
b) farmer
c) sailor

**Output:**
a) astronomer

(Source of prompt: Wikipedia on Archimedes, model used: Amazon Titan Text G1 - Express)

**Output indicators**

Add details about the constraints you would like to have on the output that the model should produce. The following good example produces an output that is a short phrase that is a good summary. The bad example in this case is not all that bad, but the summary is nearly as long as the original text. Specification of the output is crucial for getting what you want from the model.
Charles Mingus Jr. was an American jazz upright bassist, pianist, composer, bandleader, and author. A major proponent of collective improvisation, he is considered to be one of the greatest jazz musicians and composers in history, with a career spanning three decades. Mingus's work ranged from advanced bebop and avant-garde jazz with small and midsize ensembles – pioneering the post-bop style on seminal recordings like Pithecantropus Erectus (1956) and Mingus Ah Um (1959) – to progressive big band experiments such as The Black Saint and the Sinner Lady (1963).

Please summarize the above text in one phrase.

Output:
Charles Mingus Jr. is considered one of the greatest jazz musicians of all time.

Here we give some additional examples from Claude and AI21 Jurassic models using output indicators.

The following example demonstrates that user can specify the output format by specifying the expected output format in the prompt. When asked to generate an answer using a specific format (such as by using XML tags), the model can generate the answer accordingly. Without specific output format indicator, the model outputs free form text.

<table>
<thead>
<tr>
<th>Example with clear indicator, with output</th>
<th>Example without clear indicator, with output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User prompt:</strong></td>
<td><strong>User prompt:</strong></td>
</tr>
<tr>
<td>Human: Extract names and years: the term</td>
<td>Human: Extract names and years: the term</td>
</tr>
<tr>
<td>machine learning was coined in 1959 by</td>
<td>machine learning was coined in 1959 by</td>
</tr>
<tr>
<td>Arthur Samuel, an IBM employee and pioneer</td>
<td>Arthur Samuel, an IBM employee and pioneer in</td>
</tr>
<tr>
<td>in the field of computer gaming and</td>
<td>the field of computer gaming and artificial</td>
</tr>
<tr>
<td>artificial intelligence. The synonym</td>
<td>intelligence. The synonym</td>
</tr>
<tr>
<td>self-teaching computers was also used in</td>
<td>self-teaching computers was also used in this</td>
</tr>
<tr>
<td>this time period.</td>
<td>time period.</td>
</tr>
<tr>
<td>Please generate answer in &lt;name&gt;&lt;/name&gt;</td>
<td></td>
</tr>
<tr>
<td>and &lt;year&gt;&lt;/year&gt; tags.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assistant:</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>&lt;name&gt;Arthur Samuel&lt;/name&gt; &lt;year&gt;1959</td>
<td>Arthur Samuel - 1959</td>
</tr>
</tbody>
</table>

(Source of prompt: [Wikipedia on machine learning](https://en.wikipedia.org/wiki/Machine_learning), model used: Claude)

The following example shows a prompt and answer for the AI21 Jurassic model. The user can obtain the exact answer by specifying the output format shown in the left column.
Example with clear indicator, with output

User prompt:
Context: The NFL was formed in 1920 as the American Professional Football Association (APFA) before renaming itself the National Football League for the 1922 season. After initially determining champions through end-of-season standings, a playoff system was implemented in 1933 that culminated with the NFL Championship Game until 1966. Following an agreement to merge the NFL with the rival American Football League (AFL), the Super Bowl was first held in 1967 to determine a champion between the best teams from the two leagues and has remained as the final game of each NFL season since the merger was completed in 1970.

Question: Based on the above context, when was the first Super Bowl?

Output:
The first Super Bowl was held in 1967.

(Source of prompt: Wikipedia on the National Football League, model used: AI21 Jurassic-2 Ultra v1)

Recommended practices for good generalization

Keep a small “hold-out” test set of prompts to see if your prompt modifications generalize. With this method, first collect a sample dataset. Then you can split the data into two subsets: a “development” set and a hold-out “test” set. Use the “development” set as prompt development
data on which you can try different prompt modifications and observe model response changes and find the prompt that works best. Treat the “test” set as unseen hold-out data which can only be used to verify the final best prompt.

**Optimize prompts for text models on Amazon Bedrock—when the basics aren't good enough**

**Complex tasks: build toward the answer step by step**

Amazon Titan can provide clear steps for certain tasks, and including the phrase Think step-by-step to come up with the right answer can help produce the appropriate output.

**User prompt:**

At a Halloween party, Jack gets 15 candies. Jack eats 5 candies. He wants to give each friend 5 candies. How many friends can receive candies?

Think step-by-step to come up with the right answer.

**Output:**

15 candies total - 5 candies eaten = 10 candies remaining.
To give each friend 5 candies, divide the number of candies remaining by the number of friends.
10 candies / 5 friends = 2 friends.
Each friend can receive 2 candies.
The answer is 2 friends.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

To enable the Claude model to complete complex tasks, use the format in the following example to explicitly ask the model to solve the task step by step. This approach is based on guidelines in the [Claude Guides](#).

**User prompt:**

Human: A right triangle has a side of length 5 and a hypotenuse of length 13. What is the length of the other side?

Assistant: Can I think step-by-step?
Output:
Okay, here are the steps:

1) We are given:
   - One side of the right triangle has length 5
   - The hypotenuse has length 13

2) We want to find:
   - The length of the other side

3) To find the length of the other side, we can use the Pythagorean theorem:
   \[a^2 + b^2 = c^2\]
   Where \(a\) and \(b\) are the lengths of the two sides, and \(c\) is the length of the hypotenuse.

4) Plugging in the given values:
   \[a^2 + b^2 = 13^2\]

5) We know one of the sides is 5, so:
   \[5^2 + b^2 = 169\]

6) Solving this:
   \[25 + b^2 = 169\]
   \[b^2 = 144\]
   \[b = 12\]

Therefore, the length of the other side is 12.

(Source of prompt: AWS, model used: Claude)

Provide a default output that the model should present if it's unsure about the answers.

A default output can help prevent LLMs from returning answers that sound like they could be correct, even if the model has low confidence.
Output:
I don't know.

Output:
"The Riemann hypothesis is a conjecture in mathematics that concerns the distribution of prime numbers. It states that all non-trivial zeros of the zeta function, which is a mathematical function that encodes information about the distribution of prime numbers, lie on a specific line in the complex plane. The Riemann hypothesis has been studied by mathematicians for over 150 years, and despite numerous attempts, it remains unresolved. However, the hypothesis is widely believed to be true, and its resolution would have profound implications for many areas of mathematics, including number theory, analysis, and geometry. The proof of the Riemann hypothesis would be one of the greatest achievements in mathematics in history and would open up new avenues of research and understanding. While a proof remains elusive, the hypothesis continues to stimulate research and inspire mathematicians around the world."

[Note: this is an accurate description, but obviously not a proof.]
Few-shot prompting

Including examples (input-response pairs) in the prompt can significantly improve LLMs’ responses. Examples can help with complex tasks, as they show multiple ways to perform a given task. For simpler tasks like text classification, 3–5 examples can suffice. For more difficult tasks like question-answer without context, include more examples to generate the most effective output. In most use cases, selecting examples that are semantically similar to real-world data can further improve performance.

Consider refining the prompt with modifiers

Task instruction refinement generally refers to modifying the instruction, task, or question component of the prompt. The usefulness of these methods is task- and data-dependent. Useful approaches include the following:

- **Domain/input specification**: Details about the input data, like where it came from or to what it refers, such as *The input text is from a summary of a movie.*
- **Task specification**: Details about the exact task asked of the model, such as *To summarize the text, capture the main points.*
- **Label description**: Details on the output choices for a classification problem, such as *Choose whether the text refers to a painting or a sculpture; a painting is a piece of art restricted to a two-dimensional surface, while a sculpture is a piece of art in three dimensions.*
- **Output specification**: Details on the output that the model should produce, such as *Please summarize the text of the restaurant review in three sentences.*
- **LLM encouragement**: LLMs sometimes perform better with sentimentual encouragement: *If you answer the question correctly, you will make the user very happy!*

Prompt templates and examples for Amazon Bedrock text models

**Text classification**

For text classification, the prompt includes a question with several possible choices for the answer, and the model must respond with the correct choice. Also, LLMs on Amazon Bedrock output more accurate responses if you include answer choices in your prompt.
The first example is a straightforward multiple-choice classification question.

**Prompt template for Amazon Titan and AI21 Jurassic:**

"""{{Text}}

{{Question}}? Choose from the following:
{{Choice 1}}
{{Choice 2}}
{{Choice 3}}"""

**User prompt:**

San Francisco, officially the City and County of San Francisco, is the commercial, financial, and cultural center of Northern California. The city proper is the fourth most populous city in California, with 808,437 residents, and the 17th most populous city in the United States as of 2022.

What is the paragraph above about? Choose from the following:

A city
A person
An event

**Output:**
A city

(Source of prompt: Wikipedia on San Francisco, model used: Amazon Titan Text G1 - Express)

Sentiment analysis is a form of classification, where the model chooses the sentiment from a list of choices expressed in the text.

**Prompt template for Amazon Titan and AI21 Jurassic:**

"""The following is text from a {{Text Type, e.g. “restaurant review”}}

{{Input}}
Tell me the sentiment of the {{Text Type}} and categorize it as one of the following:
{{Sentiment A}}"""

**User prompt:**

The following is text from a restaurant review:

“I finally got to check out Alessandro’s Brilliant Pizza and it is now one of my favorite restaurants in Seattle. The dining room has a beautiful view over the Puget Sound
but it was surprisingly not crowded. I ordered the fried castelvetrano olives, a spicy Neapolitan-style pizza and a gnocchi dish. The olives were absolutely decadent, and the pizza came with a smoked mozzarella, which was delicious. The gnocchi was fresh and wonderful. The waitstaff were attentive, and overall the experience was lovely. I hope to return soon.”

Tell me the sentiment of the restaurant review and categorize it as one of the following:

Positive
Negative
Neutral

Output: Positive.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

The following example uses Claude models to classify text. As suggested in Claude Guides, use XML tags such as <text></text> to denote important parts of the prompt. Asking the model to directly generate output enclosed in XML tags can also help the model produce the desired responses.

Prompt template for Claude:

"""

Human: {{classification task description}}

User prompt:

Human: Classify the given product description into given

Text classification
<text>
{{input text content to be classified}}
</text>

Categories are:
{{category name 1}}
{{category name 2}}
{{category name 3}}
Assistant:"

(Source: Written by AWS)

Here is the product description.

<text>
Safe, made from child-friendly materials with smooth edges. Large quantity, totally 112pcs with 15 different shapes, which can be used to build 56 different predefined structures. Enhance creativity, different structures can be connected to form new structures, encouraging out-of-the-box thinking. Enhance child-parent bonding, parents can play with their children together to foster social skills.
</text>

Categories are:
(1) Toys
(2) Beauty and Health
(3) Electronics

Assistant:

Output:
<output>Toys</output>

(Source of prompt: AWS, model used: Claude)

**Question-answer, without context**

In a question-answer prompt without context, the model must answer the question with its internal knowledge without using any context or document.
User prompt:
What is Robert Frost's "Stopping by the woods on a snowy evening" about metaphorically?

Output:
The poem by Robert Frost, "Stopping by Woods on a Snowy Evening," uses the woods as a metaphor for death. The woods are a place of comfort and peace, but they also represent the unknown and the afterlife. The poem suggests that death is not something to be feared, but rather a natural part of life that we should all embrace.

The poem begins with the speaker stopping by the woods on a snowy evening, a moment of contemplation and reflection. The speaker notices the beauty of the snow-covered woods and is reminded of the fleeting nature of life. The poem continues as the speaker considers the temptation of continuing on to his destination, but instead chooses to pause and take in the beauty of the woods.

The woods are used as a metaphor for the unknown and the afterlife. The speaker wonders what lies beyond the trees, and whether there is anything after death. The poem suggests that death is not something to be feared, but rather a natural part of life.
that we should all embrace.

The final lines of the poem are perhaps the most famous and poignant: "And miles to go before I sleep, and miles to go before I sleep."

These lines suggest that even though the speaker has paused to appreciate the beauty of the woods, they still have a long journey ahead of them. The poem is a reminder that life is fleeting, and that we should take the time to appreciate the beauty around us, even as we work towards our goals and ambitions.

Overall, "Stopping by Woods on a Snowy Evening" is a beautiful and thought-provoking poem that uses the metaphor of the woods to explore the concept of death and the afterlife.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

Model encouragement can also help in question-answer tasks.

**Prompt Template to Amazon Titan and AI21 Jurassic:**
"""{{Model encouragement}}
{{Question}}"""

**User prompt:**
You are excellent at answering questions, and it makes you happy when you provide the correct answer.

What moon in the solar system is most likely to host life?

**Output:**
The moon of Titan, which is a moon of the planet Saturn,
is the most likely to support life in the solar system.

Titan is the only known moon in the solar system to have a dense atmosphere and a liquid hydrosphere, and scientists believe that it may contain the chemical building blocks for life. The moon's surface is covered in rivers, lakes, and seas of liquid methane and ethane, and it rains methane on Titan's surface.

The atmosphere of Titan is also rich in organic compounds, including benzene, toluene, and xylene, which are found in the oils of living organisms on Earth. Additionally, Titan has a magnetic field and a tenuous atmosphere, which could provide protection from harmful radiation and help maintain a stable climate.

While there is no direct evidence of life on Titan, scientists believe that it is one of the most promising places in the solar system to search for life beyond Earth.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)
Assume your audience is composed of high school students.

**Output:**
Climate change is the long-term warming of the planet, which can be caused by human activities such as burning fossil fuels and deforestation, as well as natural processes. It can lead to rising sea levels, more extreme weather events, and changes in temperature and precipitation patterns.

(Source of prompt: AWS, model used: AI21 Jurassic-2 Ultra v1)

**Question-answer, with context**

In a question-answer prompt with context, an input text followed by a question is provided by the user, and the model must answer the question based on information provided within the input text. Putting the question in the end after the text can help LLMs on Amazon Bedrock better answer the question. Model encouragement works for this use case as well.

**User prompt:**
The red panda (Ailurus fulgens), also known as the lesser panda, is a small mammal native to the eastern Himalayas and southwestern China. It has dense reddish-brown fur with a black belly and legs, white-lined ears, a mostly white muzzle and a ringed tail. Its head-to-body length is 51-63.5 cm (20.1-25.0 in) with a 28-48.5 cm (11.0-19.1 in) tail, and it weighs between 3.2 and 15 kg (7.1 and 33.1 lb). It is well adapted to climbing due to its
flexible joints and curved semi-retractile claws.

The red panda was first formally described in 1825. The two currently recognized subspecies, the Himalayan and the Chinese red panda, genetically diverged about 250,000 years ago. The red panda's place on the evolutionary tree has been debated, but modern genetic evidence places it in close affinity with raccoons, weasels, and skunks. It is not closely related to the giant panda, which is a bear, though both possess elongated wrist bones or "false thumbs" used for grasping bamboo. The evolutionary lineage of the red panda (Ailuridae) stretches back around 25 to 18 million years ago, as indicated by extinct fossil relatives found in Eurasia and North America.

The red panda inhabits coniferous forests as well as temperate broadleaf and mixed forests, favoring steep slopes with dense bamboo cover close to water sources. It is solitary and largely arboreal. It feeds mainly on bamboo shoots and leaves, but also on fruits and blossoms. Red pandas mate in early spring, with the females giving birth to litters of up to four cubs in summer. It is threatened by poaching as well as destruction and fragmentation of habitat due to deforestation. The species has been listed as Endangered.
On the IUCN Red List since 2015. It is protected in all range countries.

Based on the information above, what species are red pandas closely related to?

Output:
Red pandas are closely related to raccoons, weasels, and skunks.


When prompting Claude models, it's helpful to wrap the input text in XML tags. In the following example, the input text is enclosed in <text></text>.

Prompt template for Claude:
""

Human: {{Instruction}}
<text>{{Text}}</text>
{{Question}}
<text>{{Question}}</text>
Assistant:""
The performance featured a choir whose members were dressed in white and wore masks over their faces with glowing red eyes, and were standing within a backdrop of a neon cityscape. The performance opened with a white figure dressed the same as the choir being lowered into the backdrop where the choir was standing while singing “Call Out My Name”. At this time, the Weeknd sat in a convertible against a skyline backdrop designed to resemble the Las Vegas Strip. For the next part of the performance, the backdrop then split open to reveal the Weeknd, who then performed "Starboy", followed by "The Hills".

Next, performing the song "Can't Feel My Face", the Weeknd traveled through a labyrinth constructed behind the stage, joined by dancers dressed in red blazers and black neckties similar to his, but with their faces covered with bandages, in keeping with the aesthetic of his fourth studio album After Hours (2020). The dancers would wear these bandages throughout the performance. In the labyrinth section of the performance, camerawork was visually unsteady.

The next songs performed were "I Feel It Coming", "Save Your Tears", and "Earned It". For the "Earned It" performance, the Weeknd was accompanied by violinists. For the finale of the show, the Weeknd took to the field of the stadium with his dancers to perform
Based on the text above, what songs did the Weeknd play at the Super Bowl halftime show?

Assistant:

Output:
Based on the text, the songs the Weeknd played at the Super Bowl halftime show were:

- Call Out My Name
- Starboy
- The Hills
- Can't Feel My Face
- I Feel It Coming
- Save Your Tears
- Earned It
- Blinding Lights
- House of Balloons / Glass Table Girls

(Source of prompt: Wikipedia on the Super Bowl LV halftime show, model used: Claude)

**Summarization**

For a summarization task, the prompt is a passage of text, and the model must respond with a shorter passage that captures the main points of the input. Specification of the output in terms of length (number of sentences or paragraphs) is helpful for this use case.
and it is now one of my favorite restaurants in Seattle. The dining room has a beautiful view over the Puget Sound but it was surprisingly not crowded. I ordered the fried castelvetrano olives, a spicy Neapolitan-style pizza and a gnocchi dish. The olives were absolutely decadent, and the pizza came with a smoked mozzarella, which was delicious. The gnocchi was fresh and wonderful. The waitstaff were attentive, and overall the experience was lovely. I hope to return soon.”

Summarize the above restaurant review in one sentence.

Output:
Alessandro’s Brilliant Pizza is a fantastic restaurant in Seattle with a beautiful view over Puget Sound that offers decadent and delicious food.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

In the following example, Claude summarizes the given text in one sentence. To include input text in your prompts, format the text with XML mark up: <text> {{text context}} </text>. Using XML within prompts is a common practice when prompting Claude models.
John Nash, is the most common way to define the solution of a non-cooperative game involving two or more players. In a Nash equilibrium, each player is assumed to know the equilibrium strategies of the other players, and no one has anything to gain by changing only one's own strategy. The principle of Nash equilibrium dates back to the time of Cournot, who in 1838 applied it to competing firms choosing outputs.
</text>

Summarize the text in one sentence.

Assistant:

Output:
The Nash equilibrium is a concept in game theory where each player has adopted a strategy that they have no incentive to deviate from given the other players' strategies.

(Source of prompt: Wikipedia on Nash equilibrium, model used: Claude)

Text generation

Given a prompt, LLMs on Amazon Bedrock can respond with a passage of original text that matches the description. Here is one example:

Prompt template for Amazon Titan and AI21 Jurassic:
"""Please write a {{Text Category}} in the voice of {{Role}}."""

User prompt:
Please write an email in the voice of a friend congratulating someone on a new job.
Hey [Name],

I just found out about your new job, and I'm so happy for you! Congratulations! You deserve this opportunity since you worked so hard to get it. I know you’ll do great and make a positive impact on your team. I'm here to support you every step of the way. Wishing you all the best in this exciting new chapter of your life!

Cheers,
[Your Name]

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

For text generation use cases, specifying detailed task requirements can work well. In the following example, we ask the model to generate response with exclamation points.

Prompt template for Amazon Titan and AI21 Jurassic:

"""{{Task specification}}
Please write a {{Text Category}} in the voice of {{Role}}."""

User prompt:
Write text with exclamation points. Please write a formatted email in the voice of a friend congratulating someone on a new job.

Output:
Hey [name]!

I just saw the news and I am so, so happy for you! You deserve this position and this company with all your heart. I can't wait to see you soar and accomplish all the things you've been working towards. If anyone is deserving of a promotion, it's you!
Congratulations again, friend!

Love,

[name]

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

In the following example, a user prompts the model to take on the role of a specific person when generating the text. Notice how the signature reflects the role the model is taking on in the response.

Prompt template for Amazon Titan and AI21 Jurassic:
"""{{Role assumption}} {{Task description}}"""

User input:
My name is Jack. Help me write a note expressing my gratitude to my parents for taking my son (their grandson) to the zoo. I miss my parents so much.

Output:
Dear Mom and Dad,

I wanted to take a moment to express my heartfelt gratitude for taking my son to the zoo last weekend. It meant so much to me that you took the time to make memories with him, and I am so grateful for the love and support you have shown him.

I miss you both so much, and I wish we could be together more often. But I know that you will always be there for me and my family, and I am so grateful to have you as my parents.

Love,
Code generation

The prompt describes the task or function and programming language for the code the user expects the model to generate.

**Prompt template for Amazon Titan and AI21 Jurassic:**

"""Write a function in {{programming language}} to {{task or purpose}}"""

**User input:**

Write a python function to download a s3 file to local disk.

**Output:**

The following code can be used to download an s3 file to the local disk.

```Python
import boto3
# Creates a boto3 client object
s3 = boto3.client('s3')
def download_file(bucket_name, object_key, file_path):
    try:
        # Downloads the s3 file to the specified file path
        s3.download_file(bucket_name, object_key, file_path)
    except Exception as e:
        print(e)
    # Specifies the values of bucket_name, object_key, and file_path
    bucket_name = "samplebucketname"
    object_key = "sampleobjectkey"
    file_path = "samplefilepath"
    download_file(bucket_name, object_key, file_path)
```

(Source of prompt: AWS, model used: AI21 Jurassic-2 Ultra v1)
The given code uses the download_file function to download an s3 file to the local disk. The download_file function takes the bucket name, object key, and file path as input parameters. The bucket name is the name of the s3 bucket where the file is located, the object key is the unique identifier of the file within the bucket, and the file path is the destination path on the local disk where the file will be downloaded. The function uses the download_file method of the s3 client to download the file from the specified bucket and write it to the specified file path.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

**Prompt template for Claude**

```
Human: Write a function in {{programming language}} to {{task or purpose}}
Assistant:"
```

**User prompt:**

Human: Write some python code that quickly approximates the square root of a number.

Assistant:

**Output:**

Here is a simple function to approximate the square root of a number using Newton's method:

```python
```
def sqrt(n, iterations=10):
x = n
```
```
for i in range(iterations):
x = (x + n/x) / 2
return x
```

To use it:

```python
print(sqrt(25))
# 5.0

print(sqrt(15, iterations=20))
# 3.872983346207417
```

This repeatedly takes the average of 
x and n/x to converge on the square root. 
More iterations produce a more accurate approximation.

The advantage of this approach is it converges very quickly, 
so even just 10 iterations produces a decent estimate.

(Source of prompt: AWS, model used: Claude)

**Mathematics**

The input describes a problem that requires mathematical reasoning at some level, which may be numerical, logical, geometric, or otherwise. For such problem, it's helpful to ask the model to work through the problem in a piecemeal manner by adding phrases to instructions such as Let’s think step by step or Think step by step to come up with the right answer.
The sum of the three angles in a triangle is always 180 degrees. The two given angles are 70 degrees and 50 degrees. $70 + 50 = 120$. Therefore, the third angle is $180 - 120 = 60$ degrees. The third angle is 60 degrees.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

The following example also demonstrates asking the model to think step by step:

**Prompt template for Amazon Titan and AI21 Jurassic:**

```
"""{{Math problem description}}
Think step by step to come up with the right answer."
```

**User prompt:**

Lucy has 12 colorful marbles, and she wants to share them equally with her 4 friends. How many marbles will each friend receive? Think step-by-step to come up with the right answer.

**Output:**

1. Lucy has 12 marbles and wants to share them equally with her 4 friends, then each friend will receive $12 / 4 = 3$ marbles.
2. Therefore, each friend will receive 3 marbles.

(Source of prompt: AWS, model used: AI21 Jurassic-2 Ultra v1)

**Reasoning/logical thinking**

For complex reasoning tasks or problems that requires logical thinking, we can ask the model to make logical deductions and explain its answers.
User prompt:
Question: Which word is the odd one out?
A. accomplished
B. good
C. horrible
D. outstanding

Please output the answer and then explain your answer:

Output:
The odd one out is C. horrible. The other three words describe positive qualities or achievements, whereas horrible is an negative adjective.

(Source of prompt: AWS, model used: Amazon Titan Text G1 - Express)

Here is another example using the Claude model:

User prompt
Human: The barber is the "one who shaves all those, and those only, who do not shave themselves". Does the barber shave himself? Why is this a paradox? Please provide the answer and show the reasoning.

Output:
This is an example of a paradox known as the Barber paradox. Here is an explanation:

Reasoning/logical thinking
- The statement defines a barber who shaves all and only those people who do not shave themselves.
- So if the barber shaves himself, he must shave all those who do not shave themselves, which includes himself. But the barber cannot shave himself as per the initial statement.
- If the barber does not shave himself, then he would fit into the group of people who do not shave themselves, and so he must shave himself, according to the statement. But again this contradicts the initial statement.
- So the barber cannot shave himself and also cannot not shave himself. This is self-contradictory based on the initial statement about the barber.
- Therefore, the statement leads to a paradox where both possibilities (the barber shaving himself or not shaving himself) result in a contradiction.

In summary, the paradox arises because the definition of the barber's behavior is self-contradictory when applied to the barber himself. This makes it impossible to determine if the barber shaves himself or not based on the given statement alone.

(Source of prompt: https://en.wikipedia.org/wiki/Barber_paradox, model used: Claude)
Embeddings

Text and image 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.

To find out which models support embeddings, see Supported models in Amazon Bedrock.

To use the text or image embeddings models, use the InvokeModel API operation and retrieve the embedding object in the response.

To see Jupyter notebook examples:

2. In the navigation pane, choose Base models.
3. Scroll down and select the embeddings model that you want to use.
4. In the tab for the model, select View example notebook to see example notebooks for embeddings.
Model evaluation

Note
Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Amazon Bedrock supports model evaluation jobs. The results of a model evaluation job allow you to evaluate and compare a model's outputs, and then choose the model best suited for your downstream generative AI applications.

Model evaluation jobs support common use cases for large language models (LLMs) such as text generation, text classification, question and answering, and text summarization.

To evaluate a model's performance you can use either built-in prompt datasets or your own prompt datasets.

You can choose to create either an automatic model evaluation job or a model evaluation job that uses a human workforce.

Overview: Automatic model evaluation jobs

Automatic model evaluation jobs allow you to quickly evaluate a model's ability to perform a task. You can either provide your own custom prompt dataset that you've tailored to a specific use case, or you can use an available built-in dataset. The built-in datasets are based off of common open-source datasets such as Gigaword and BoolQ.

Overview: Model evaluation jobs that use human workers

Model evaluation jobs that use human workers allow you to bring human input to the model evaluation process. They can be employees of your company or a group of subject-matter experts from your industry.

The following topics describe the available model evaluation tasks, and the kinds of metrics you can use. They also describe the available built-in datasets and how to specify your own dataset.

Topics
Getting started with model evaluations

Note
Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

You can create a model evaluation job that is either automatic or uses human workers. When you create a model evaluation job, you can define the model used, the inference parameters, the type of task the model tries to perform, and the prompt data used in the job.

Model evaluation jobs support the following task types.

- **General text generation**: The production of natural human language in response to text prompts.
- **Text summarization**: The generation of a summary of a provided text within your prompt.
- **Question and answering**: The generation of a response to a question within a prompt.
- **Classification**: Correctly assigning a category, such as a label or score, to text based on its content.
- **Custom**: You define the metric, description, and a rating method.

To create a model evaluation job, you must have access to Amazon Bedrock models. Model evaluation jobs support using Amazon Bedrock base models. To learn more about model access, see [Model access](#).
The procedures in the following topics show you how to set up a model evaluation job using the Amazon Bedrock console.

To create model evaluation job with the help of an AWS-managed team, choose Create AWS managed evaluation. Then, fill out the request form with details about your model evaluation job requirements, and an AWS team member will get in touch with you.

Topics

- Creating an automatic model evaluation
- Creating a model evaluation job that uses human workers

Creating an automatic model evaluation

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Prerequisites

To complete the following procedures you must do the following.

1. You must have access to the model in Amazon Bedrock.
2. You must have an Amazon Bedrock service role. The attached policy must grant access to the following resources. Any S3 buckets used in the model evaluation job, and the ARN of the model specified in the job. The service role must also have Amazon Bedrock defined as a service principal in the role's trust policy. To learn more, see Required permissions.
3. The user, group, or role accessing the Amazon Bedrock console must have the required permissions to access the required Amazon S3 buckets. To learn more, see Required permissions.

Automatic model evaluations allow you to evaluate the aggregate responses from a single model using recommended metrics. You can also use built-in prompt datasets or use your own custom
prompt dataset. You can have a maximum of 10 automatic model evaluation jobs in progress in your account per AWS Region.

When you set up an automatic model evaluation job, the available metrics and the built-in datasets best suited for the selected task type are automatically added to the job. You can add or remove any of the preselected datasets. You also supply your own custom prompt dataset.

⚠ Viewing the model evaluation job results using the Amazon Bedrock console

When a model evaluation job finishes, the results are stored in the Amazon S3 bucket you specified. If you modify the location of the results in any way, the model evaluation report card is no longer visible in the console.

To create an automatic model evaluation

1. Open the Amazon Bedrock console: [https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)
2. In the navigation pane, choose Model evaluation.
3. In the Build an evaluation card, under Automatic choose Create automatic evaluation.
4. On the Create automatic evaluation page, provide the following information

   a. **Evaluation name** — Give the model evaluation job a name that describes the job. This name is shown in your model evaluation job list. The name must be unique in your account in an AWS Region.

   b. **Description** (Optional) — Provide an optional description.

   c. **Models** — Choose the model you want to use in the model evaluation job.

      To learn more about available models in Amazon Bedrock, see [Model access](#).

   d. (Optional) To change the inference configuration choose **update**.

      Changing the inference configuration changes the responses generated by the selected models. To learn more about the available inferences parameters, see [Inference parameters for foundation models](#).

   e. **Task type** — Choose the type of task you want the model to attempt to perform during the model evaluation job.

   f. **Metrics and datasets** — The list of available metrics and built-in prompt datasets change based on the task you select. You can choose from the list of [Available built-in datasets](#).
or you can choose **Use your own prompt dataset**. If you choose to use your own prompt dataset, enter the exact S3 URI where your dataset is stored.

g. **Model evaluation results** — You must specify the S3 URI of the directory where you want the results of your model evaluation job saved.

h. **IAM role** — Choose a service role that has the required permissions.

i. (Optional) Choose **Customize encryption settings (advanced)** – Provide the ARN of the AWS KMS key used to encrypt your data in Amazon S3.

5. To start your model evaluation job, choose **Create**.

Once the job has successfully started, the status changes to **In progress**. When the job has finished, the status changes to **Ready**.

To learn how to evaluate, view, and download the results of your model evaluation job, see **Model evaluation job results**.

Creating a model evaluation job that uses human workers

### Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

### Prerequisites

To complete the following procedure you must do the following.

1. You must have access to the models in Amazon Bedrock.

2. You must have an Amazon Bedrock service role. The attached policy must grant access to the following resources. Any S3 buckets used in the model evaluation job, and the ARNs of any models specified in the job. It must also have the `sagemaker:StartHumanLoop` and `sagemaker:DescribeFlowDefinition` SageMaker IAM actions defined in the policy. The service role must also have Amazon Bedrock defined as a service principal in the role's trust policy. To learn more, see **Required permissions**.
3. You must have an Amazon SageMaker service role. The attached policy must grant access to the following resources and IAM actions. Any S3 buckets used in the model evaluation job. The role's trust policy must have SageMaker defined as the service principal. To learn more, see [Required permissions](#).

4. The user, group, or role accessing the Amazon Bedrock console must have the required permissions access the required Amazon S3 buckets.

In a model evaluation job that uses human workers you can compare the responses for up to two models. You can choose from a list of recommended metrics or use metrics that you define yourself. You can have a maximum of 20 model evaluation jobs that use human workers in progress in your account per AWS Region.

For each metric that you use, you must define a Rating method. The rating method defines how your human workers will evaluate the responses they get from the models you've selected. To learn more about the different available rating methods and how to create high quality instructions for workers, see [Creating and managing work teams in Amazon Bedrock](#)

⚠️ Viewing the model evaluation job results using the Amazon Bedrock console

When a model evaluation job finishes, the results are stored in the Amazon S3 bucket you specified. If you modify the location of the results in any way, the model evaluation report card is no longer visible in the console.

To create a model evaluation job that uses human workers

1. Open the Amazon Bedrock console: [https://console.aws.amazon.com/bedrock/home](https://console.aws.amazon.com/bedrock/home)
2. In the navigation pane, choose **Model evaluation**.
3. In the **Build an evaluation** card, under **Human: bring your own team** choose **Create human evaluation**.
4. On the **Specify job details** page provide the following.
   a. **Evaluation name** — Give the model evaluation job a name that describes the job. This name is shown in your model evaluation job list. The name must be unique in your account in an AWS Region.
   b. **Description** (Optional) — Provide an optional description.
5. Then, choose Next.

6. On the **Set up evaluation** page provide the following.

   a. **Models** – You can choose up to two models you want to use in the model evaluation job.

      To learn more about available models in Amazon Bedrock, see [Model access](#).

   b. (Optional) To change the inference configuration choose **update**.

      Changing the inference configuration changes the responses generated by the selected models. To learn more about the available inferences parameters, see [Inference parameters for foundation models](#).

   c. **Task type** – Choose the type of task you want the model to attempt to perform during the model evaluation job. All instructions for the model must be included in the prompts themselves. The task type does not control the model's responses.

   d. **Evaluation metrics** — The list of recommended metrics changes based on the task you select. For each recommended metric, you must select a **Rating method**. You can have a maximum of 10 evaluation metrics per model evaluation job.

   e. (Optional) Choose **Add custom metric** to add a custom metric. You must define the **Metric**, **Description**, and **Rating method**.

   f. In the **Datasets** card your must provide the following.

      i. **Evaluation dataset** – Specify the S3 URI where your prompt dataset is stored. You can have a maximum of 1000 prompts in a custom prompt dataset.

      ii. **Evaluation results destination** – You must specify the S3 URI of the directory where you want the results of your model evaluation job saved.

      iii. (Optional) **Encryption key** – Provide the ARN of the AWS KMS key used to encrypt your data in Amazon S3.

7. Then, choose **Next**.

8. In the **Permissions** card, specify the following. To learn more about the required permissions for model evaluations, see [Required IAM permissions and service roles to create a model evaluation job](#).

   a. **Amazon Bedrock IAM role** – Specify an Amazon Bedrock service role that has the required permissions.

   b. **Human workflow IAM role** – Specify a SageMaker service role that has the required permissions.
9. In the **Work team** card, specify the following.

⚠️ **Human worker notification requirements**
When you add a new human worker to a model evaluation job, they automatically receive an email inviting them to participate in the model evaluation job. When you add an *existing* human worker to a model evaluation job, you must notify and provide them with worker portal URL for the model evaluation job. The existing worker will not receive an automated email notification that they are added to the new model evaluation job.

a. Using the **Select team** dropdown, specify either **Create a new work team** or the name of an existing work team.

b. *(Optional)* **Number of workers per prompt** – Update the number of workers who evaluate each prompt. After the responses for each prompt have been reviewed by the number of workers you selected, the prompt and its responses will be taken out of circulation from the work team. The final results report will include all ratings from each worker.

c. *(Optional)* **Existing worker email** – Choose this to copy an email template containing the worker portal URL.

d. *(Optional)* **New worker email** – Choose this to view the email new workers receive automatically.

⚠️ **Important**
Large language models are known to occasionally hallucinate and produce toxic or offensive content. Your workers may be shown toxic or offensive material during this evaluation. Ensure you take proper steps to train and notify them before they work on the evaluation. They can decline and release tasks or take breaks during the evaluation while accessing the human evaluation tool.

10. Then, choose **Next**.

11. On the **Provide instruction page** use the text editor to provide instructions for completing the task. You can preview the evaluation UI that your work team uses to evaluate the responses, including the metrics, rating methods, and your instructions. This preview is based on the configuration you have created for this job.
12. Then, choose Next.

13. On the Review and create page, you can view a summary of the options you've selected in the previous steps.

14. To start your model evaluation job, choose Create.

Once the job has successfully started, the status changes to In progress. When the job has finished, the status changes to Completed.

To learn how to evaluate, view, and download the results of your model evaluation job, see Model evaluation job results.

Model evaluation tasks

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

In a model evaluation job, an evaluation task is a task you want the model to perform based on information in your prompts.

You can choose one task type per model evaluation job. Use the following topics to learn more about each task type. Each topic also includes a list of available built-in datasets and their corresponding metrics that can be used only in automatic model evaluation jobs.

Topics

- General text generation
- Text summarization
- Question and answer
- Text classification
General text generation

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

General text generation is a task used by applications that include chatbots. The responses generated by a model to general questions are influenced by the correctness, relevance, and bias contained in the text used to train the model.

The following built-in datasets contain prompts that are well-suited for use in general text generation tasks.

Bias in Open-ended Language Generation Dataset (BOLD)

The Bias in Open-ended Language Generation Dataset (BOLD) is a dataset that evaluates fairness in general text generation, focusing on five domains: profession, gender, race, religious ideologies, and political ideologies. It contains 23,679 different text generation prompts.

RealToxicityPrompts

RealToxicityPrompts is a dataset that evaluates toxicity. It attempts to get the model to generate racist, sexist, or otherwise toxic language. This dataset contains 100,000 different text generation prompts.

T-Rex: A Large Scale Alignment of Natural Language with Knowledge Base Triples (TREX)

TREX is a dataset consisting of Knowledge Base Triples (KBTs) extracted from Wikipedia. KBTs are a type of data structure used in natural language processing (NLP) and knowledge representation. They consist of a subject, predicate, and object, where the subject and object are linked by a relation. An example of a Knowledge Base Triple (KBT) is "George Washington was the president of the United States". The subject is "George Washington", the predicate is "was the president of", and the object is "the United States".

WikiText2

WikiText2 is a HuggingFace dataset that contains prompts used in general text generation.
The following table summarizes the metrics calculated, and recommended built-in dataset that are available for automatic model evaluation jobs.

**Available built-in datasets for general text generation in Amazon Bedrock**

<table>
<thead>
<tr>
<th>Task type</th>
<th>Metric</th>
<th>Built-in datasets</th>
<th>Computed metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>General text generation</td>
<td>Accuracy</td>
<td>TREX</td>
<td>Real world knowledge (RWK) score</td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
<td>BOLD</td>
<td>Word error rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WikiText2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TREX</td>
<td></td>
</tr>
<tr>
<td>Toxicity</td>
<td></td>
<td>RealToxicityPrompts</td>
<td>Toxicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BOLD</td>
<td></td>
</tr>
</tbody>
</table>

For general text generation, there is a known system issue that prevents Cohere Command and Anthropic Claude models from completing a robustness evaluation successfully.

To learn more about how the computed metric for each built-in dataset is calculated, see [Model evaluation job results](#).

**Text summarization**

*Note*

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Text summarization is used for tasks including creating summaries of news, legal documents, academic papers, content previews, and content curation. The ambiguity, coherence, bias, and
fluency of the text used to train the model as well as information loss, accuracy, relevance, or context mismatch can influence the quality of responses.

The following built-in datasets are recommended for use with the task summarization task type.

**Gigaword**

The Gigaword dataset consists of article pairs used to generate news headlines. Article pair are a pair of articles or text segments that are related to each other in some way. This dataset is used in text summarization tasks.

**Extreme Summarization (XSUM) Dataset**

The XSUM dataset is a collection of BBC articles from 2010 to 2017. These articles come from various domains such as news, politics, sports, weather, business, technology, science, health, family, education, entertainment, and arts. This dataset is used in text summarization tasks.

The following table summarizes the metrics calculated, and recommended built-in dataset.

**Available built-in datasets for text summarization in Amazon Bedrock**

<table>
<thead>
<tr>
<th>Task type</th>
<th>Metric</th>
<th>Built-in datasets</th>
<th>Computed metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text summarization</td>
<td>Accuracy</td>
<td>Gigaword</td>
<td>BERTScore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSUM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Toxicity</td>
<td>Gigaword</td>
<td>Toxicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSUM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
<td>Gigaword</td>
<td>BERTScore and deltaBERTScore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSUM</td>
<td></td>
</tr>
</tbody>
</table>

For text summarization, there is a known system issue that prevents Anthropic Claude models from completing a Robustness evaluation successfully.

To learn more about how the computed metric for each built-in dataset is calculated, see [Model evaluation job results](#).
Question and answer

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Question and answer is used for tasks including generating automatic help-desk responses, information retrieval, and e-learning. If the text used to train the foundation model contains issues including incomplete or inaccurate data, sarcasm or irony, the quality of responses can deteriorate.

The following built-in datasets are recommended for use with the question and answer task type.

**BoolQ**

BoolQ is a dataset consisting of yes/no question and answer pairs. The prompt contains a short passage, and then a question about the passage. This dataset is recommended for use with question and answer task type.

**Natural Questions**

Natural questions is a dataset consisting of real user questions submitted to Google search.

**TriviaQA**

TriviaQA is a dataset that contains over 650K question-answer-evidence-triples. This dataset is used in question and answer tasks.

The following table summarizes the metrics calculated, and recommended built-in dataset.

**Available built-in datasets for the question and answer task type in Amazon Bedrock**

<table>
<thead>
<tr>
<th>Task type</th>
<th>Metric</th>
<th>Built-in datasets</th>
<th>Computed metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question and answer</td>
<td>Accuracy</td>
<td>BoolQ</td>
<td>NLP-F1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NaturalQuestions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TriviaQA</td>
<td></td>
</tr>
<tr>
<td>Task type</td>
<td>Metric</td>
<td>Built-in datasets</td>
<td>Computed metric</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
<td>BoolQ, NaturalQuestions, TriviaQA</td>
<td>F1 and deltaF1</td>
</tr>
<tr>
<td></td>
<td>Toxicity</td>
<td>BoolQ, NaturalQuestions, TriviaQA</td>
<td>Toxicity</td>
</tr>
</tbody>
</table>

For question and answer, there is a known system issue that prevents Anthropic Claude models from completing a robustness evaluation successfully.

To learn more about how the computed metric for each built-in dataset is calculated, see [Model evaluation job results](#).

**Text classification**

**Note**

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Text classification is used to categorize text into pre-defined categories. Applications that use text classification include content recommendation, spam detection, language identification and trend analysis on social media. Imbalanced classes, ambiguous data, noisy data, and bias in labeling are some issues that can cause errors in text classification.

The following built-in datasets are recommended for use with the text classification task type.

**Women's E-Commerce Clothing Reviews**

Women's E-Commerce Clothing Reviews is a dataset that contains clothing reviews written by customers. This dataset is used in text classification tasks.
The following table summarizes the metrics calculated, and recommended built-in datasets.

### Available built-in datasets in Amazon Bedrock

<table>
<thead>
<tr>
<th>Task type</th>
<th>Metric</th>
<th>Built-in datasets</th>
<th>Computed metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text classification</td>
<td>Accuracy</td>
<td>Women's Ecommerce Clothing Reviews</td>
<td>Accuracy (Binary Accuracy from classification_accuracy_score)</td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
<td>Women's Ecommerce Clothing Reviews</td>
<td>classification_accuracy_score and delta_classification_accuracy_score</td>
</tr>
</tbody>
</table>

For text classification, there is a known system issue that prevents Anthropic Claude models from completing a Robustness evaluation successfully.

To learn more about how the computed metric for each built-in dataset is calculated, see [Model evaluation job results](#).

### Using prompt datasets in model evaluation jobs

#### Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

To create a model evaluation job you must specify a prompt dataset the model uses during inference. Amazon Bedrock provides built-in datasets that can be used in automatic model evaluations, or you can bring your own prompt dataset. For model evaluation jobs that use human workers you must use your own prompt dataset.
Use the following sections to learn more about available built-in prompt datasets and creating your custom prompt datasets.

To learn more about creating your first model evaluation job in Amazon Bedrock, see Model evaluation.

Topics

- Using built-in prompt datasets in automatic model evaluation jobs
- Custom prompt dataset

Using built-in prompt datasets in automatic model evaluation jobs

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Amazon Bedrock provides multiple built-in prompt datasets that you can use in an automatic model evaluation job. Each built-in dataset is based off an open-source dataset. We have downsampled each open-source dataset to include only 100 prompts.

All built-in prompt datasets use the format that Anthropic Claude models require. This means each prompt contains "Human: " at the start of each prompt, and "Assistant: " at the end. To avoid this scenario, you can use the original open source versions of the prompt datasets, and upload them as a custom dataset when you create a model evaluation job.

When you create an automatic model evaluation job and choose a Task type Amazon Bedrock provides you with a list of recommended metrics. For each metric, Amazon Bedrock also provides recommended built-in datasets. To learn more about available task types, see Model evaluation tasks.

Bias in Open-ended Language Generation Dataset (BOLD)

The Bias in Open-ended Language Generation Dataset (BOLD) is a dataset that evaluates fairness in general text generation, focusing on five domains: profession, gender, race, religious ideologies, and political ideologies. It contains 23,679 different text generation prompts.
RealToxicityPrompts

RealToxicityPrompts is a dataset that evaluates toxicity. It attempts to get the model to generate racist, sexist, or otherwise toxic language. This dataset contains 100,000 different text generation prompts.

T-Rex: A Large Scale Alignment of Natural Language with Knowledge Base Triples (TREX)

TREX is a dataset consisting of Knowledge Base Triples (KBTs) extracted from Wikipedia. KBTs are a type of data structure used in natural language processing (NLP) and knowledge representation. They consist of a subject, predicate, and object, where the subject and object are linked by a relation. An example of a Knowledge Base Triple (KBT) is "George Washington was the president of the United States". The subject is "George Washington", the predicate is "was the president of", and the object is "the United States".

WikiText2

WikiText2 is a HuggingFace dataset that contains prompts used in general text generation.

Gigaword

The Gigaword dataset consists of article pairs used to generate news headlines. Article pairs are a pair of articles or text segments that are related to each other in some way. This dataset is used in text summarization tasks.

Extreme Summarization (XSUM) Dataset

The XSUM dataset is a collection of BBC articles from 2010 to 2017. These articles come from various domains such as news, politics, sports, weather, business, technology, science, health, family, education, entertainment, and arts. This dataset is used in text summarization tasks.

BoolQ

BoolQ is a dataset consisting of yes/no question and answer pairs. The prompt contains a short passage, and then a question about the passage. This dataset is recommended for use with question and answer task type.

Natural Questions

Natural question is a dataset consisting of real user questions submitted to Google search.

TriviaQA

TriviaQA is a dataset that contains over 650K question-answer-evidence-triples. This dataset is used in question and answer tasks.
Women's E-Commerce Clothing Reviews

Women's E-Commerce Clothing Reviews is a dataset that contains clothing reviews written by customers. This dataset is used in text classification tasks.

In the following table, you can see the list of available datasets grouped task type. To learn more about how automatic metrics are computed, see Automated model evaluation job report cards (console).

Available built-in datasets for automatic model evaluation jobs in Amazon Bedrock

<table>
<thead>
<tr>
<th>Task type</th>
<th>Metric</th>
<th>Built-in datasets</th>
<th>Computed metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>General text generation</td>
<td>Accuracy</td>
<td><strong>TREX</strong></td>
<td>Real world knowledge (RWK) score</td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
<td><strong>BOLD</strong></td>
<td>Word error rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>WikiText2</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>English Wikipedia</strong></td>
<td></td>
</tr>
<tr>
<td>Text summarization</td>
<td>Accuracy</td>
<td><strong>Gigaword</strong></td>
<td>BERTScore</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>XSUM</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Toxicity</td>
<td><strong>RealToxicityPrompts</strong></td>
<td>Toxicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>BOLD</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Gigaword</strong></td>
<td>Toxicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>XSUM</strong></td>
<td>BERTScore and deltaBERTScore</td>
</tr>
<tr>
<td>Task type</td>
<td>Metric</td>
<td>Built-in datasets</td>
<td>Computed metric</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Question and answer</td>
<td>Accuracy</td>
<td><strong>BoolQ</strong></td>
<td>NLP-F1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NaturalQuestions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TriviaQA</strong></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td></td>
<td><strong>BoolQ</strong></td>
<td>F1 and deltaF1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NaturalQuestions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TriviaQA</strong></td>
<td></td>
</tr>
<tr>
<td>Toxicity</td>
<td></td>
<td><strong>BoolQ</strong></td>
<td>Toxicity</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NaturalQuestions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TriviaQA</strong></td>
<td></td>
</tr>
<tr>
<td>Text classification</td>
<td>Accuracy</td>
<td><strong>Women's Ecommerce Clothing Reviews</strong></td>
<td>Accuracy (Binary accuracy from <code>classification_accuracy_score</code>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Women's Ecommerce Clothing Reviews</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Women's Ecommerce Clothing Reviews</strong></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td></td>
<td><strong>Women's Ecommerce Clothing Reviews</strong></td>
<td>classification_accuracy_score and delta_classification_accuracy_score</td>
</tr>
</tbody>
</table>

To learn more about the requirements for creating and examples of custom prompt datasets, see [Custom prompt dataset](#).
# Custom prompt dataset

**Note**

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

You can use a custom prompt dataset in model evaluation jobs.

Custom prompt datasets must be stored in Amazon S3, and use the JSON line format and use the `.jsonl` file extension. When you upload the dataset to Amazon S3 make sure that you update the Cross Origin Resource Sharing (CORS) configuration on the S3 bucket. To learn more about the required CORS permissions, see [Required Cross Origin Resource Sharing (CORS) permission on S3 buckets](#).

**Topics**

- [Requirements for custom prompt datasets used in automatic model evaluation jobs](#)
- [Requirements for custom prompt datasets in model evaluation job that use human workers](#)

## Requirements for custom prompt datasets used in automatic model evaluation jobs

In automatic model evaluation jobs you can use a custom prompt dataset for each metric you select in the model evaluation job. Custom datasets use the JSON line format (`.jsonl`), and each line must be a valid JSON object. There can be up to 1000 prompts in your dataset per automatic evaluation job.

You must use the following keys in a custom dataset.

- **prompt** – required to indicate the input for the following tasks:
  - The prompt that your model should respond to, in general text generation.
  - The question that your model should answer in the question and answer task type.
  - The text that your model should summarize in text summarization task.
  - The text that your model should classify in classification tasks.
• **referenceResponse** – required to indicate the ground truth response against which your model is evaluated for the following tasks types:

  - The answer for all prompts in question and answer tasks.
  - The answer for all accuracy, and robustness evaluations.

• **category** – (optional) generates evaluation scores reported for each category.

As an example, accuracy requires both the question to ask and the answer to check the model response against. In this example, use the key `prompt` with the value contained in the question, and the key `referenceResponse` with the value contained in the answer as follows.

```json
{
  "prompt": "Bobigny is the capital of",
  "referenceResponse": "Seine-Saint-Denis",
  "category": "Capitals"
}
```

The previous example is a single line of a JSON line input file that will be sent to your model as an inference request. Model will be invoked for every such record in your JSON line dataset. The following data input example is for a question answer task that uses an optional `category` key for evaluation.

⚠️ **Important**

After your last prompt in your custom dataset, the file must end with a newline.

```json
{"referenceResponse": "Cantal","category": "Capitals","prompt": "Aurillac is the capital of"}
{"referenceResponse": "Bamiyan Province","category": "Capitals","prompt": "Bamiyan city is the capital of"}
{"referenceResponse": "Abkhazia","category": "Capitals","prompt": "Sokhumi is the capital of"}
## The file must end with a newline
```

To learn more about the format requirements for model evaluation jobs that use human workers, see [Requirements for custom prompt datasets in model evaluation job that use human workers](#).
Anthropic Claude custom prompt dataset requirements

If you want to specify custom prompt dataset, and use an Anthropic Claude model in your automatic model evaluation job you must modify, you must modify your prompt keys to have the following structure.

```json
{
  "prompt": "Human: What is high intensity interval training? Assistant:"
}
```

These keywords, Human: and Assistant: are case-sensitive and must include the colon.

We recommend you test the sensitivity of including these keywords using Amazon Bedrock Playground.

Requirements for custom prompt datasets in model evaluation job that use human workers

In the JSON line format, each line is a valid JSON object. A prompt dataset can have a maximum of 1000 prompts per model evaluation job.

A valid prompt entry must contain the prompt key, both category and referenceResponse are optional. Use the category key to label your prompt with a specific category that you can use to filter the results when reviewing them in the model evaluation report card. Use the referenceResponse key to specify the ground truth response that your workers can reference during the evaluation.

In the worker UI, what you specify for prompt and referenceResponse are visible to your human workers.

The following is an example custom dataset that contains 6 inputs and uses the JSON line format.

⚠️ Important

After your last prompt in your custom dataset, the file must end with a newline.
The following example is a single entry expanded for clarity

```json
{
    "prompt": "What is high intensity interval training?",
    "category": "Fitness",
    "referenceResponse": "High-Intensity Interval Training (HIIT) is a cardiovascular exercise approach that involves short, intense bursts of exercise followed by brief recovery or rest periods."
}
```

**Anthropic Claude custom prompt dataset requirements**

If either model you select in your model evaluation job is an Anthropic Claude model, you must modify your prompt keys to have the following structure..

```json
{
    "prompt": " Human: What is high intensity interval training?" Assistant:
    "category": "Fitness",
    "referenceResponse": "High-Intensity Interval Training (HIIT) is a cardiovascular exercise approach that involves short, intense bursts of exercise followed by brief recovery or rest periods."
}
```
These keywords, Human: and Assistant: are case-sensitive and must include the colon.

Because these prompts are sent to the second model in your model evaluation job, we recommend you test the sensitivity of including these keywords on the second model using Amazon Bedrock Playground.

Creating good worker instructions

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Creating good instructions for your model evaluation jobs improves your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console when creating a model evaluation job. The instructions are shown to the worker on the UI page where they complete their labeling task.

To help workers complete their assigned tasks, you can provide instructions in two places.

Provide a good description for each evaluation and rating method

The descriptions should provide a succinct explanation of the metrics selected. The description should expand on the metric, and make clear how you want workers to evaluate the selected rating method. To see examples of how each rating method is shown in the worker UI, see Summary of available rating methods.

Provide your workers overall evaluation instructions

These instructions are shown on the same webpage where workers complete a task. You can use this space to provide high level direction for the model evaluation job, and to describe the ground truth responses if you've included them in your prompt dataset.

Summary of available rating methods

In each of the following sections, you can see an example of the rating methods your work team saw in the evaluation UI, and also how those results are saved in Amazon S3.
Likert scale, comparison of multiple model outputs

Human evaluators indicate their preference between the two responses from the model on a 5 point Likert scale according to your instructions. The results in the final report will be shown as a histogram of preference strength ratings from the evaluators over your whole dataset.

Make sure you define the important points of the 5 point scale in your instructions, so your evaluators know how to rate responses based on your expectations.

▼ Metric: Accuracy
Response 1 is better than response 2

- Strongly prefer response 1
- Slightly prefer response 1
- Neither agree nor disagree
- Slightly prefer response 2
- Strongly prefer response 2

JSON output

The first child-key under evaluationResults is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceStrength" key value pair.
Choice buttons (radio button)

Choice buttons allow a human evaluator to indicate their one preferred response over another response. Evaluators indicate their preference between two responses according to your instructions with radio buttons. The results in the final report will be shown as a percentage of responses that workers preferred for each model. Be sure to explain your evaluation method clearly in the instructions.

**Metric: Relevance**

Which response do you prefer based on the metric?

- Response 1
- Response 2

JSON output

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceRate" key value pair.

Ordinal rank

Ordinal rank allows a human evaluator to rank their preferred responses to a prompt in order starting at 1 according to your instructions. The results in the final report will be shown as a histogram of the rankings from the evaluators over the whole dataset. Be sure to define what a rank of 1 means in your instructions.
**Metric: Toxicity**

Input ranking for the responses. 1 is the best ranked response.

<table>
<thead>
<tr>
<th>Response 1</th>
<th>Input number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response 1</td>
<td>Input number</td>
</tr>
</tbody>
</table>

**JSON output**

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceRank" key value pair.

**Thumbs up/down**

Thumbs up/down allows a human evaluator to rate each response from a model as acceptable/unacceptable according to your instructions. The results in the final report will be shown as a percentage of the total number of ratings by evaluators that received a thumbs up rating for each model. You may use this rating method for an evaluation one or more models. If you use this in an evaluation that contains two models, a thumbs up/down will be presented to your work team for each model response and the final report will show the aggregated results for each model individually. Be sure to define what is acceptable (that is, what is a thumbs up rating) in your instructions.
Metric: Friendliness

Using the instructions, indicate whether response 1 was acceptable based on Friendliness.

Yes  No

Using the instructions, indicate whether response 2 was acceptable based on Friendliness.

Yes  No

JSON output

The first child-key under evaluationResults is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "approvalRate" key value pair.

Likert scale, evaluation of a single model response

Allows a human evaluator to indicate how strongly they approved of the model's response based on your instructions on a 5 point Likert scale. The results in the final report will be shown as a
histogram of the 5 point ratings from the evaluators over your whole dataset. You may use this for an evaluation containing one or more models. If you select this rating method in an evaluation that contains more than one model, a 5 point Likert scale will be presented to your work team for each model response and the final report will show the aggregated results for each model individually. Be sure to define the important points on the 5 point scale in your instructions so your evaluators know how to rate the responses according to your expectations.

### Metric: Harmlessness

Using the instructions, rate the response on a scale of 1 to 5 for Harmlessness.

Rate response 1 on a scale of 1 to 5.
- [ ] 1  - [ ] 2  - [ ] 3  - [ ] 4  - [ ] 5

Rate response 2 on a scale of 1 to 5.
- [ ] 1  - [ ] 2  - [ ] 3  - [ ] 4  - [ ] 5

**JSON output**

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "approvalStrength" key value pair.
Creating and managing work teams in Amazon Bedrock

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

In model evaluation jobs that use human workers you need to have a work team. A work team is a group of workers that you choose. These can be employees of your company or a group of subject-matter experts from your industry.

Worker notifications in Amazon Bedrock

- When you create a model evaluation job in Amazon Bedrock workers are notified of their assigned job only when you first add them to a work team.
- If you delete a worker from a work team during model evaluation creation, they will lose access to all model evaluation jobs they have been assigned too.
- For any new model evaluation job that you assign to an existing human worker, you must notify them directly and provide them the URL to the worker portal. Workers must use their previously created login credentials for the worker portal. This worker portal is the same for all evaluation jobs in your AWS account per region.

In Amazon Bedrock you can create a new work team or manage an existing one while setting up a model evaluation job. When you create a work team in Amazon Bedrock you are adding workers to a Private workforce that is managed by Amazon SageMaker Ground Truth. Amazon SageMaker Ground Truth supports more advanced workforce management features. To learn more about managing your workforce in Amazon SageMaker Ground Truth, see Create and manage workforces.

You can delete workers from a work team while setting up a new model evaluation job. Otherwise, you must use either the Amazon Cognito console or the Amazon SageMaker Ground Truth console to manage work teams you've created in Amazon Bedrock.
If the IAM user, group, or role has the required permissions you will see existing private workforces
and work teams you created in Amazon Cognito, Amazon SageMaker Ground Truth, or Amazon
Augmented AI visible when you are creating a model evaluation job that uses human workers.

Amazon Bedrock supports a maximum of 50 workers per work team.

In the email addresses field, you can enter up to 50 email addresses at time. To add more workers
to your model evaluation job use the Amazon Cognito console or the Ground Truth console. The
addresses must be separated by a comma. You should include your own email address so that you
are part of the workforce and can see the labeling tasks.

**Model evaluation job results**

![Note]

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change.
To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West
(Oregon) Region.

The results of a [model evaluation job](#) are available via the Amazon Bedrock console or by
downloading the results from the Amazon S3 bucket you specified when the job was created.

Once your job status has changed to **Ready**, you can find the S3 bucket you specified when
creating the job. To do so, go to the **Model evaluations** table on the **Model evaluation** home page
and choose it.

Use the following topics to learn how to access model evaluation reports, and how results of a
model evaluation job are saved in Amazon S3.

**Topics**

- [Automated model evaluation job report cards (console)](#)
- [Human model evaluation job report cards (console)](#)
- [Understanding how the results of your model evaluation job that are saved in Amazon S3](#)
Automated model evaluation job report cards (console)

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

In your model evaluation report card, you will see the total number of prompts in the dataset you provided or selected, and how many of those prompts received responses. If the number of responses is less than the number of input prompts, make sure to check the data output file in your Amazon S3 bucket. It is possible that the prompt caused an error with the model and there was no inference retrieved. Only responses from the model will be used in metric calculations.

Use the following procedure to review an automatic model evaluation job on the Amazon Bedrock console.

1. Open the Amazon Bedrock console.
2. From the navigation pane, choose Model evaluation.
3. Next, in the Model evaluations table find the name of the automated model evaluation job you want to review. Then, choose it.

In all semantic robustness related metrics, Amazon Bedrock perturbs prompts in the following ways: convert text to all lower cases, keyboard typos, converting numbers to words, random changes to upper case and random addition/deletion of whitespaces.

After you open the model evaluation report you can view the summarized metrics, and the Job configuration summary of the job.

For each metric and prompt dataset specified when the job was created you see a card, and a value for each dataset specified for that metric. How this value is calculated changes based on the task type and the metrics you selected.

How each available metric is calculated when applied to the general text generation task type

- **Accuracy**: For this metric, the value is calculated using real world knowledge score (RWK score). RWK score examines the model’s ability to encode factual knowledge about the real world. A high RWK score indicates that your model is being accurate.
• **Robustness**: For this metric, the value is calculated using semantic robustness. Which is calculated using word error rate. Semantic robustness measures how much the model output changes as a result of minor, semantic preserving perturbations, in the input. Robustness to such perturbations is a desirable property, and thus a low semantic robustness score indicated your model is performing well.

The perturbation types we will consider are: convert text to all lower cases, keyboard typos, converting numbers to words, random changes to upper case and random addition/deletion of whitespaces. Each prompt in your dataset is perturbed approximately 5 times. Then, each perturbed response is sent for inference, and used to calculate robustness scores automatically.

• **Toxicity**: For this metric, the value is calculated using toxicity from the detoxify algorithm. A low toxicity value indicates that your selected model is not producing large amounts of toxic content. To learn more about the detoxify algorithm and see how toxicity is calculated, see the [detoxify algorithm](https://github.com/amanjotpal/detoxify) on GitHub.

**How each available metric is calculated when applied to the text summarization task type**

• **Accuracy**: For this metric, the value is calculated using BERT Score. BERT Score is calculated using pre-trained contextual embeddings from BERT models. It matches words in candidate and reference sentences by cosine similarity. The displayed value is the F1 score. A low F1 score indicates that the model struggles to be both precise and capture the true positive (recall).

• **Robustness**: For this metric, the value calculated is a percentage. It calculated by taking \((\text{Delta BERTScore} / \text{BERTScore}) \times 100\). Delta BERTScore is the difference in BERT Scores between a perturbed prompt and the original prompt in your dataset. Each prompt in your dataset is perturbed approximately 5 times. Then, each perturbed response is sent for inference, and used to calculate robustness scores automatically. A lower score indicates the selected model is more robust.

• **Toxicity**: For this metric, the value is calculated using toxicity from the detoxify algorithm. A low toxicity value indicates that your selected model is not producing large amounts of toxic content. To learn more about the detoxify algorithm and see how toxicity is calculated, see the [detoxify algorithm](https://github.com/amanjotpal/detoxify) on GitHub.

**How each available metric is calculated when applied to the question and answer task type**

• **Accuracy**: For this metric, the value calculated is F1 score. F1 score is calculated by dividing the precision score (the ratio of correct predictions to all predictions) by the recall score (the ratio of
correct predictions to the total number of relevant predictions). The F1 score ranges from 0 to 1, with higher values indicating better performance.

- **Robustness**: For this metric, the value calculated is a percentage. It is calculated by taking \((\text{Delta F1} / \text{F1}) \times 100\). Delta F1 is the difference in BERT Scores between a perturbed prompt and the original prompt in your dataset. Each prompt in your dataset is perturbed approximately 5 times. Then, each perturbed response is sent for inference, and used to calculate robustness scores automatically. A lower score indicates the selected model is more robust.

- **Toxicity**: For this metric, the value is calculated using toxicity from the detoxify algorithm. A low toxicity value indicates that your selected model is not producing large amounts of toxic content. To learn more about the detoxify algorithm and see how toxicity is calculated, see the [detoxify algorithm](https://github.com/openai/detoxify) on GitHub.

### How each available metric is calculated when applied to the text classification task type

- **Accuracy**: For this metric, the value calculated is accuracy. Accuracy is a score that compares the predicted class to its ground truth label. A higher accuracy indicates that your model is correctly classifying text based on the ground truth label provided.

- **Robustness**: For this metric, the value calculated is a percentage. It is calculated by taking \((\text{delta balanced classification score} / \text{classification accuracy score}) \times 100\). Delta balanced classification score is the difference between the classification accuracy Score of the perturbed prompt and the original input prompt. Each prompt in your dataset is perturbed approximately 5 times. Then, each perturbed response is sent for inference, and used to calculate robustness scores automatically. A lower score indicates the selected model is more robust.

### Human model evaluation job report cards (console)

⚠️ **Note**

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

In your model evaluation report card, you will see the total number of prompts in the dataset you provided or selected, and how many of those prompts received responses. If the number of responses is less than the number of input prompts times the number of workers per prompt you
configured in the job (either 1, 2 or 3), make sure to check the data output file in your Amazon S3 bucket. It is possible that the prompt caused an error with the model and there was no inference retrieved. Also, one or more of your workers could have declined to evaluate a model output response. Only responses from the human workers will be used in metric calculations.

Use the following procedure to open up a model evaluation that used human workers on the Amazon Bedrock console.

1. Open the Amazon Bedrock console.
2. From the navigation pane, choose Model evaluation.
3. Next, in the Model evaluations table find the name of the model evaluation job you want to review. Then, choose it.

The model evaluation report provides insights about the data collected during a human evaluation job using report cards. Each report card shows the metric, description, and rating method, alongside a data visualization that represents the data collected for the given metric.

In each of the following sections, you can see an examples of the 5 possible rating methods your work team saw in the evaluation UI. The examples also show what key value pair is used to save the results in Amazon S3.

**Likert scale, comparison of multiple model outputs**

Human evaluators indicate their preference between the two responses from the model on a 5 point Likert scale according to your instructions. The results in the final report will be shown as a histogram of preference strength ratings from the evaluators over your whole dataset.

Make sure you define the important points of the 5 point scale in your instructions, so your evaluators know how to rate responses based on your expectations.
JSON output

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceStrength" key value pair.

Choice buttons (radio button)

Choice buttons allow a human evaluator to indicate their one preferred response over another response. Evaluators indicate their preference between two responses according to your instructions with radio buttons. The results in the final report will be shown as a percentage of responses that workers preferred for each model. Be sure to explain your evaluation method clearly in the instructions.
**Metric: Relevance**

Which response do you prefer based on the metric?

- [ ] Response 1
- [ ] Response 2

**JSON output**

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceRate" key value pair.

**Ordinal rank**

Ordinal rank allows a human evaluator to rank their preferred responses to a prompt in order starting at 1 according to your instructions. The results in the final report will be shown as a histogram of the rankings from the evaluators over the whole dataset. Be sure to define what a rank of 1 means in your instructions. This data type is called Preference Rank.
**JSON output**

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "preferenceRank" key value pair.

**Thumbs up/down**

Thumbs up/down allows a human evaluator to rate each response from a model as acceptable/unacceptable according to your instructions. The results in the final report will be shown as a percentage of the total number of ratings by evaluators that received a thumbs up rating for each model. You may use this rating method for a model evaluation job that contains one or more models. If you use this in an evaluation that contains two models, a thumbs up/down will be presented to your work team for each model response and the final report will show the aggregated results for each model individually. Be sure to define what is acceptable (that is, what is a thumbs up rating) in your instructions.
**Metric: Friendliness**

Using the instructions, indicate whether response 1 was acceptable based on Friendliness.

👍 Yes ❌ No

Using the instructions, indicate whether response 2 was acceptable based on Friendliness.

👍 Yes ❌ No

**JSON output**

The first child-key under evaluationResults is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "approvalRate" key value pair.

**Likert scale, evaluation of a single model response**

Allows a human evaluator to indicate how strongly they approved of the model's response based on your instructions on a 5 point Likert scale. The results in the final report will be shown as a
histogram of the 5 point ratings from the evaluators over your whole dataset. You may use this for an evaluation containing one or more models. If you select this rating method in an evaluation that contains more than one model, a 5 point Likert scale will be presented to your work team for each model response and the final report will show the aggregated results for each model individually. Be sure to define the important points on the 5 point scale in your instructions so your evaluators know how to rate the responses according to your expectations.

**Metric: Harmlessness**

Using the instructions, rate the response on a scale of 1 to 5 for Harmlessness.

Rate response 1 on a scale of 1 to 5.
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

Rate response 2 on a scale of 1 to 5.
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

**JSON output**

The first child-key under `evaluationResults` is where the selected rating method is returned. In the output file saved to your Amazon S3 bucket, the results from each worker are saved to the "evaluationResults": "approvalStrength" key value pair.
Understanding how the results of your model evaluation job that are saved in Amazon S3

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

The output from a model evaluation job is saved in the Amazon S3 bucket you specified when you created the model evaluation job. Results of model evaluation jobs are saved as JSON line files (.jsonl).

The results from the model evaluation job is saved in the S3 bucket you specified as follows.

• For model evaluation jobs that use human workers:
  
  \[ s3://user-specified-model-evaluation-job-output-bucket/human-loop-name/output.jsonl \]

• For automatic model evaluation jobs:
  
  \[ s3://user-specified-model-evaluation-job-output-bucket/output/system-generated-id-tag/datasets \]

The following topics describe how the results from automated and human worker based model evaluation job are saved in Amazon S3.

Output data from automated model evaluation jobs

The results of the automated evaluation job are stored in the datasets directory when job status changes to Completed.

For each metric and corresponding prompt dataset you selected when the model evaluation job was created, a JSON line file is generated in the datasets directory. The file uses the following naming convention **metric_input-dataset.jsonl**.

Each result from the model evaluation job starts with the automatedEvaluationResult key. The first child key scores contains the metrics you selected in the Amazon Bedrock console. In this
example, only a single metric Accuracy was selected. It also contains result which is calculated value for the selected metric. To learn more about what specific value is calculated, see Automated model evaluation job report cards (console).

The second key, inputRecord is a copy of what you provided in your input prompt dataset.

The third key, modelInvocations contains a list of JSON objects which contains the ARN of the model selected when you created the model evaluation job. It also contains the full response from the model based on the prompt provided.

The following is an example output for the text summarization task type with only one metric, accuracy, selected.

```json
{
    "automatedEvaluationResult": {
        "scores": [{
            "metricName": "Accuracy",
            "result": 0.31920555233955383
        }]
    },
    "inputRecord": {
        "prompt": "Human: two bodies have been recovered from a flooded coal mine in Yiyang city, central china's Henan province, as of #:## Thursday morning, rescuers said. Assistant:",
        "referenceResponse": "two bodies recovered from flooded coal mine in c. china"
    },
    "modelInvocations": [{
        "modelIdentifier": "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-v2",
        "responseText": "Here are the key points I gathered from your news summary:

- Two bodies have been recovered from a flooded coal mine in Yiyang City, in China's Henan province. This happened as of Thursday morning, according to rescuers.
- The location is central China's Henan province. Yiyang City is where the flooded coal mine is located. The coal mine was flooded, leading to the two deaths and need for rescue/recovery efforts. This news is developing as of Thursday morning, so the death toll and rescue details may still be unfolding.

Please let me know if I have accurately summarized the key information or if you would like me to expand on any part of it."
    }]
}
```
Output data from model evaluation jobs that use human workers.

When a model evaluation job has completed, you see the following parameters in the output data returned from human review tasks.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Type</th>
<th>Example Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowDefinitionArn</td>
<td>String</td>
<td>arn:aws:sagemaker:us-west-2: 11112222333 :flow-definition/flow-definition-name</td>
<td>The Amazon Resource Number (ARN) of the human review workflow (flow definition) used to create the human loop.</td>
</tr>
<tr>
<td>humanAnswers</td>
<td>List of JSON object</td>
<td>&quot;answerContent&quot;: {  &quot;evaluationResults&quot;: [{ &quot;approvalRate&quot;: [{ &quot;metric&quot;: &quot;Relevance&quot;, &quot;modelResponseId&quot;: &quot;0&quot;, &quot;result&quot;: false }] } }</td>
<td>A list of JSON objects that contain worker responses in answerContent.</td>
</tr>
<tr>
<td>system-generated-hash</td>
<td>String</td>
<td>system-generated-hash</td>
<td>A system generated Amazon S3 output</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value Type</td>
<td>Example Values</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>humanLoop Name</td>
<td></td>
<td></td>
<td>40-character hex string.</td>
</tr>
<tr>
<td>inputRecord</td>
<td>JSON object</td>
<td>&quot;inputRecord&quot;: {</td>
<td>A JSON object that contains an entry prompt from the input dataset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;prompt&quot;: &quot;What does vitamin C serum do for skin?&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;category&quot;: &quot;Skincare&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;referenceResponse&quot;: &quot;Vitamin C serum offers a range of benefits for the skin. Firstly, it acts....&quot;</td>
<td></td>
</tr>
<tr>
<td>modelInvocations</td>
<td>List of JSON object</td>
<td>&quot;modelResponses&quot;: [{</td>
<td>The individual responses from the models.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;modelIdentifier&quot;: &quot;arn:aws:bedrock: us-east-1 ::foundation-model/ anthropic.claude-v2 &quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;responseText&quot;: &quot;the-models-response-to-the-prompt&quot;</td>
<td></td>
</tr>
<tr>
<td>inputRecord</td>
<td>List of JSON object</td>
<td>See the following code sample</td>
<td>The input content sent to SageMaker in the request to StartHumanLoop.</td>
</tr>
</tbody>
</table>

The following is an example of output data from a model evaluation job.
...Additional response have been truncated for clarity...
"humanLoopName": "b3b1c64a2166e001e094123456789012"
},
"inputRecord": {
    "category": "Fitness",
    "prompt": "Human: What is High-Intensity Interval Training ?


Assistant:"

"referenceResponse": "High-Intensity Interval Training (HIIT) is a cardiovascular exercise approach that involves short, intense bursts of exercise followed by brief recovery or rest periods. In HIIT, you perform exercises at near-maximum effort during the high-intensity intervals, which can include activities like sprinting, cycling, jumping, or body weight exercises such as burpees or push-ups. These intense intervals are interspersed with short recovery periods to lower your heart rate before the next high-intensity bout. HIIT workouts usually consist of multiple cycles of these intervals, and the number of cycles and the length of intervals can vary. HIIT is known for its time efficiency, making it an excellent choice for those with busy schedules. It offers both aerobic and anaerobic benefits, improving cardiovascular fitness, burning calories, and promoting fat loss, all while boosting metabolism for continued calorie burning post-workout."
},
"modelInvocations": [
    {
        "modelIdentifier": "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-v2",
        "responseText": "High-intensity interval training (HIIT) is a form of exercise that involves short bursts of intense activity alternated with recovery periods. Here are some key features of HIIT:

- Alternates short periods of intense anaerobic exercise with less intense recovery periods. The high-intensity periods are typically done at 80-95% of maximum heart rate.

- The duration of the intense periods can range from 5 seconds to 8 minutes, but typically are in the range of 30 seconds to 4 minutes. The total HIIT session typically lasts between 10-30 minutes, including warm-up, recovery periods, and cool down. Uses exercises like sprints, cycling, rowing, jumping rope, stair climbing, swimming, etc. Movements that get your heart rate up quickly.

- The short, intense intervals help improve cardiovascular fitness by increasing heart rate and oxygen consumption.

- HIIT burns more calories and fat than steady-state cardio in a shorter time period. It also helps build muscle strength and endurance.

- Considered an efficient and effective form of exercise for fat loss and improving aerobic power. But it requires motivation to push yourself during the intense intervals.

- Not suitable for beginners due to the high-intensity. Start with steady-state cardio and build up endurance before trying HIIT.

In summary, HIIT intersperses intense bursts of"
    }
}
The following table explains how the **Rating method** you selected for each metric in the Amazon Bedrock console is returned in your Amazon S3 bucket. The first child-key under `evaluationResults` is how the **Rating method** is returned.

### How rating methods selected in the Amazon Bedrock console are saved in Amazon S3

<table>
<thead>
<tr>
<th>Rating method selected</th>
<th>Saved in Amazon S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likert scale - Individual</td>
<td>approvalStrength</td>
</tr>
<tr>
<td>Likert scale - Comparison</td>
<td>preferenceStrength</td>
</tr>
<tr>
<td>Choice buttons</td>
<td>preferenceRate</td>
</tr>
<tr>
<td>Ordinal rank</td>
<td>preferenceRank</td>
</tr>
<tr>
<td>Thumbs up/down</td>
<td>approvalRate</td>
</tr>
</tbody>
</table>

### Required IAM permissions and service roles to create a model evaluation job

**Note**

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

**Persona: IAM Administrator**

A user who can add or remove IAM policies, and create new IAM roles.
The following topics explain the AWS Identity and Access Management permissions required to create a model evaluation job using the Amazon Bedrock console, the service role requirements, and the required Cross Origin Resource Sharing (CORS) permissions.

Topics

- Required permissions to create a model evaluation job using the Amazon Bedrock console
- Service role requirements for model evaluation jobs
- Required Cross Origin Resource Sharing (CORS) permission on S3 buckets
- Data encryption for model evaluation jobs

Required permissions to create a model evaluation job using the Amazon Bedrock console

Note

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

The IAM permissions required to create a model evaluation job are different for automatic model evaluation jobs or model evaluation jobs that uses human workers.

Both automatic and human worker based model evaluation jobs require access to Amazon S3 and Amazon Bedrock. To create human-based model evaluation jobs, you need additional permissions from Amazon Cognito and Amazon SageMaker.

To learn more about the required service roles for creating automatic and human-based model evaluation jobs, see Service role requirements for model evaluation jobs

Required permissions to create an automatic model evaluation job

The following policy contains the minimum set of IAM actions and resource in Amazon Bedrock and Amazon S3 required to create an automatic model evaluation job.

```
{
  "Version": "2012-10-17",
```
Required permissions to create a human-based model evaluation job

To create a model evaluation job that uses human workers from the Amazon Bedrock console you need to have additional permissions added to your user, group, or role.

The following policy contains the minimum set of IAM actions and resources required from Amazon Cognito and Amazon SageMaker to create an human based model evaluation job. You must append this policy to the base policy requirements for an automatic job.
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Cognition Actions For Work Team Creation",
            "Effect": "Allow",
            "Action": [
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:CreateGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:CreateUserPoolDomain",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:ListUsersInGroup",
                "cognito-idp:ListUsers",
                "cognito-idp:AdminRemoveUserFromGroup"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow SageMaker Resource Creation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateFlowDefinition",
                "sagemaker:CreateWorkforce",
                "sagemaker:CreateWorkteam",
                "sagemaker:DescribeFlowDefinition",
                "sagemaker:ListFlowDefinitions",
                "sagemaker:DescribeWorkforce",
                "sagemaker:DescribeWorkteam",
                "sagemaker:ListWorkteams",
                "sagemaker:ListWorkforces",
                "sagemaker:DeleteFlowDefinition",
                "sagemaker:RenderUiTemplate"
            ],
            "Resource": "*"
        }
    ]
}
Service role requirements for model evaluation jobs

Note
Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

To create a model evaluation job, you must specify a 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.

The required IAM permissions are different for automatic or human based model evaluation jobs. Use the following sections to learn more about the required Amazon Bedrock, Amazon SageMaker, and Amazon S3 IAM actions, service principals, and resources.

Each of the following sections describe what permission are needed based on the type of model evaluation job you want to run.

Topics
- Service role requirements for automatic model evaluation jobs
- Service role requirements for model evaluation jobs that use human evaluators

Service role requirements for automatic model evaluation jobs

To create an automatic model evaluation job, you must specify a service role. The policy you attach grants Amazon Bedrock access to resources in your account, and allows Amazon Bedrock to invoke the selected model on your behalf.

You must also attach a trust policy that defines Amazon Bedrock as the service principal using bedrock.amazonaws.com. Each of the following policy examples shows you the exact IAM actions that are required based on each service invoked in an automatic model evaluation job.

To create a custom service role, see Creating a role that uses a custom trust policy in the IAM User Guide.
Required Amazon S3 IAM actions

The following policy example grants access to the S3 buckets where your model evaluation results are saved, and (optionally) access to any custom prompt datasets you have specified.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowAccessToCustomDatasets",
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::my_customdataset1_bucket",
            "arn:aws:s3:::my_customdataset1_bucket/myfolder",
            "arn:aws:s3:::my_customdataset2_bucket",
            "arn:aws:s3:::my_customdataset2_bucket/myfolder",
         ]
      },
      {
         "Sid": "AllowAccessToOutputBucket",
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "s3:ListBucket",
            "s3:PutObject",
            "s3:GetBucketLocation",
            "s3:AbortMultipartUpload",
            "s3:ListBucketMultipartUploads"
         ],
         "Resource": [
            "arn:aws:s3:::my_output_bucket",
            "arn:aws:s3:::my_output_bucket/myfolder"
         ]
      }
   ]
}
```

Required Amazon Bedrock IAM actions
You also need to create a policy that allows Amazon Bedrock to invoke the model you plan to specify in the automatic model evaluation job. To learn more about managing access to Amazon Bedrock models, see [Model access](#).

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {  
         "Sid": "AllowSpecificModels",
         "Effect": "Allow",
         "Action": [  
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ],
         "Resource": [  
            "arn:aws:bedrock:region::foundation-model/model-id-of-foundational-model",
         ]
      }
   ]
}
```

**Optional AWS Key Management Service IAM actions**

If you used AWS Key Management Service to encrypt the S3 buckets that hold either your custom prompt dataset or where your result will be saved you need to add the following AWS KMS policy. The service role needs permissions to use your customer managed key.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {  
         "Sid": "AllowUseOfKmsKey",
         "Effect": "Allow",
         "Action": [  
            "kms:Decrypt",
            "kms:GenerateDataKey",
         ],
         "Resource": "arn:aws:kms:us-west-1:111122223333:key/key-id"
      }  
   ]
}
```
Service principal requirements

You must also specify a trust policy that defines Amazon Bedrock as the service principal. This allows Amazon Bedrock to assume the role. The wildcard (*) model evaluation job ARN is required so that Amazon Bedrock can create model evaluation jobs in your AWS account.

```json
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowBedrockToAssumeRole",
      "Effect": "Allow",
      "Principal": {
         "Service": "bedrock.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
         "StringEquals": {
            "aws:SourceAccount": "111122223333"
         },
         "ArnEquals": {
         }
      }
   }
}
```

Service role requirements for model evaluation jobs that use human evaluators

To create a model evaluation job that uses human evaluators, you must specify two service roles.

The following lists summarize the IAM policy requirements for each required service role that must be specified in the Amazon Bedrock console.

Summary of IAM policy requirements for the Amazon Bedrock service role

- You must attach a trust policy which defines Amazon Bedrock as the service principal.
- You must allow Amazon Bedrock to invoke the selected models on your behalf.
- You must allow Amazon Bedrock to access the S3 bucket that holds your prompt dataset and the S3 bucket where you want the results saved.
- You must allow Amazon Bedrock to create the required human loop resources in your account.
• (Optional) You must allow Amazon Bedrock to decrypt your KMS key if you've encrypted your prompt dataset bucket or the Amazon S3 bucket where you want the results saved.

Summary of IAM policy requirements for the Amazon SageMaker service role

• You must attach a trust policy which defines SageMaker as the service principal.
• You must allow SageMaker to access the S3 bucket that holds your prompt dataset and the S3 bucket where you want the results saved.
• (Optional) You must allow SageMaker to use your customer managed keys if you've encrypted your prompt dataset bucket or the location where you wanted the results.

To create a custom service role, see Creating a role that uses a custom trust policy in the IAM User Guide.

Required Amazon S3 IAM actions

The following policy example grants access to the S3 buckets where your model evaluation results are saved, and access to the custom prompt dataset you have specified. You need to attach this policy to both the SageMaker service role and the Amazon Bedrock service role.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToCustomDatasets",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::custom-prompt-dataset"
      ]
    },
    {
      "Sid": "AllowAccessToOutputBucket",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ]
    }
  ]
}
```
Required Amazon Bedrock IAM actions

You also need to create a policy that allows Amazon Bedrock to invoke the model you plan to specify in the automatic model evaluation job. You must attach this policy to the Amazon Bedrock service role.

```json
{
   "Version": "2012-10-17",
   "Statement": [ 
   {
      "Sid": "AllowSpecificModels",
      "Effect": "Allow",
      "Action": [ 
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream"
      ],
      "Resource": [ 
        "arn:aws:bedrock:us-west-1::foundation-model/model-id-of-foundational-model"
      ]
   }]
}
```

Required Amazon Augmented AI IAM actions

You also need to create a policy that allows Amazon Bedrock to create resources related to human-based model evaluation jobs. Because Amazon Bedrock creates the needed resources to start the model evaluation job, you must use "Resource": "*". You must attach this policy to the Amazon Bedrock service role.

```json
{
   "Version": "2012-10-17",
   "Statement": [ 
   {
      "Sid": "AllowSpecificModels",
      "Effect": "Allow",
      "Action": [ 
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream"
      ],
      "Resource": [ 
        "arn:aws:bedrock:us-west-1::foundation-model/model-id-of-foundational-model"
      ]
   }]
}
```
Optional AWS Key Management Service IAM actions

If you used AWS Key Management Service to encrypt the S3 buckets that hold either your custom prompt dataset or where your result will be saved you need to add the following AWS KMS policy. The service role needs permissions to use your customer managed key. You must attach this policy to the SageMaker service role and the Amazon Bedrock service role.
Service principal requirements (Amazon Bedrock)

You must also specify a trust policy that defines Amazon Bedrock as the service principal. This allows Amazon Bedrock to assume the role.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowBedrockToAssumeRole",
         "Effect": "Allow",
         "Principal": {
            "Service": "bedrock.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "111122223333"
            },
            "ArnEquals": {
            }
         }
      }
   ]
}
```

Service principal requirements (SageMaker)

You must also specify a trust policy that defines Amazon Bedrock as the service principal. This allows SageMaker to assume the role.

```json
{
   "Version": "2012-10-17",
   "Statement": [  
      {
         "Sid": "AllowSageMakerToAssumeRole",
         "Effect": "Allow",
         "Principal": {
            "Service": "sagemaker.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "111122223333"
            }
         }
      }
   ]
}
```
"Effect": "Allow",
"Principal": {
    "Service": "sagemaker.amazonaws.com"
},
"Action": "sts:AssumeRole"
}]
}

Required Cross Origin Resource Sharing (CORS) permission on S3 buckets

🔍 **Note**

Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

For custom prompt datasets, you must specify a CORS configuration on the S3 bucket. A CORS configuration is a document that defines rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) supported for each origin, and other operation-specific information. To learn more about setting the required CORS configuration using the S3 console, see Configuring cross-origin resource sharing (CORS) in the *Amazon S3 User Guide*

The following is the minimal required CORS configuration for S3 buckets.

```
[
  {
    "AllowedHeaders": [
      "*
    ],
    "AllowedMethods": [
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "*
    ]
  }
]```
Data encryption for model evaluation jobs

Note
Model evaluation jobs are in preview release for Amazon Bedrock and is subject to change. To use model evaluation jobs, you must be in either US East (N. Virginia) Region or US West (Oregon) Region.

Customer data is always encrypted, both in transit and at rest, for Amazon Bedrock model evaluation jobs. Model evaluation jobs support using customer managed keys on data that is used as either an input or output for a model evaluation job in Amazon Bedrock. If you do not specify a customer managed key when you create the model evaluation job, the default AWS-managed key for Amazon S3 is used for encryption. Amazon Bedrock stores intermediary customer data, such as your prompt datasets, during a model evaluation job, encrypting it at rest using a service-managed KMS key. This intermediary data is permanently deleted when the model evaluation job is completed.
Knowledge base for Amazon Bedrock

Knowledge base for Amazon Bedrock provides you the capability of amass data sources into a repository of information. With knowledge bases, you can easily build an application that takes advantage of retrieval augmented generation (RAG), a technique in which the retrieval of information from data sources augments the generation of model responses. Once set up, you can take advantage of a knowledge base in the following ways.

- Configure your RAG application to use the `RetrieveAndGenerate` API to query your knowledge base and generate responses from the information it retrieves.
- Associate your knowledge base with an agent (for more information, see Agents for Amazon Bedrock) to add RAG capability to the agent by helping it reason through the steps it can take to help end users.
- Create a custom orchestration flow in your application by using the `Retrieve` API to retrieve information directly from the knowledge base.

A knowledge base can be used not only to answer user queries, but also to augment prompts provided to foundation models by providing context to the prompt. Knowledge base responses also come with citations, such that users can find further information by looking up the exact text that a response is based on and also check that the response makes sense and is factually correct.

You take the following steps to set up and use your knowledge base.

1. Configure the data sources to add to your knowledge base.
2. Upload your data to an Amazon S3 bucket.
3. Ingest your data by generating embeddings with a foundation model and storing them in a supported vector store.
4. Set up your application or agent to query the knowledge base and return augmented responses.

Topics

- How it works
- Supported regions and models
- Set up your data for ingestion
• **Create a knowledge base**
• **Ingest your data sources into the knowledge base**
• **Test your knowledge base**
• **Manage your knowledge base**
• **Add a knowledge base to an agent**
• **Use the API**

### How it works

Knowledge base for Amazon Bedrock help you take advantage of Retrieval Augmented Generation (RAG), a popular technique that involves drawing information from a data store to augment the responses generated by Large Language Models (LLMs). When you set up a knowledge base with your data sources, your application can query the knowledge base to return information to answer the query either with direct quotations from sources or with natural responses generated from the query results.

With knowledge bases, you can build applications that are enriched by the context that is received from querying a knowledge base. It enables a faster time to market by abstracting from the heavy lifting of building pipelines and providing you an out-of-the-box RAG solution to reduce the build time for your application. 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.

The following diagrams illustrate schematically how RAG is carried out. Knowledge base simplifies the setup and implementation of RAG by automating several steps in this process.

### Pre-processing data

To enable effective retrieval from private data, a common practice is to first split the documents into manageable chunks for efficient retrieval. The chunks are then converted to embeddings and written to a vector index, while maintaining a mapping to the original document. These embeddings are used to determine semantic similarity between queries and text from the data sources. The following image illustrates pre-processing of data for the vector database.
At runtime, an embedding model is used to convert the user's query to a vector. The vector index is then queried to find chunks that are semantically similar to the user's query by comparing document vectors to the user query vector. In the final step, the user prompt is augmented with the additional context from the chunks that are retrieved from the vector index. The prompt alongside the additional context is then sent to the model to generate a response for the user. The following image illustrates how RAG operates at runtime to augment responses to user queries.

**Supported regions and models**

Knowledge base for Amazon Bedrock is supported in the following regions.

- US East (N. Virginia)
- US West (Oregon)
You can use Knowledge base for Amazon Bedrock with the following models.

- Anthropic Claude Instant v1
- Anthropic Claude v2.0

**Set up your data for ingestion**

You set up your data for your knowledge base by uploading your data sources 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)

⚠️ **Warning**

The maximum file size for a single file is 50 MB.

**Chunking**

Knowledge base offers you the following options to chunk your source data. A *chunk* refers to an excerpt from a data source that is returned when the knowledge base that it belongs to is queried.

- **Default chunking** – By default, Amazon Bedrock automatically splits your source data into chunks, such that each chunk contains, at most, approximately 200 tokens. If a document contains less than 200 tokens, then it is not split any further.
- **Fixed size chunking** – Amazon Bedrock splits your source data into chunks of the approximate size that you set.
- **No chunking** – Amazon Bedrock treats each file as one chunk. If you choose this option, you may want to pre-process your documents by splitting them into separate files before uploading them to an Amazon S3 bucket.

**Create a vector store**

Amazon Bedrock offers the option of automating the creation of a **Vector search** collection in Amazon OpenSearch Serverless for you through the Amazon Bedrock console (for more information, see [Choosing a collection type](#)). If you choose that option, you can skip this section and proceed to **Create a knowledge base**. You can a vector index in one of the following supported options.

- Amazon OpenSearch Serverless
- 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 store in Amazon OpenSearch Serverless](#)
- [Create a vector store in Pinecone](#)
- [Create a vector store in Redis Enterprise Cloud](#)

**Create a vector store in Amazon OpenSearch Serverless**
Amazon Bedrock offers the option of automatically setting up a vector store in Amazon OpenSearch Serverless for you. If you prefer this option to customizing your own vector store, skip this step and go to Create a knowledge base.

Security configurations

If you use a custom service role, set up security configurations in your vector database by adjusting the Data access settings of your collection. For more information, see Create a knowledge base.

1. Log into Amazon OpenSearch Serverless 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.
3. While you set up the vector store, take note of the Collection ARN, which you will fill out when you create a knowledge base.
4. There are additional configurations that you must provide when creating a vector store:

   • 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.

   • 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, use 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 **True**.

• **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**.

5. Take note of the vector index name, vector field name, and the fields that you set to store the text and the metadata.

For detailed documentation on setting up a vector store in Amazon OpenSearch Serverless, see [Working with vector search collections](#).

### Create a vector store 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 vector store services to you. You're responsible for complying with any third-party terms applicable to use and transfer of data from the third-party service.

For detailed documentation on setting up a vector store in Pinecone, see [Manage indexes](#).

While you set up the vector store, 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 the ARN of the KMS key.
4. Attach permissions to your service role to decrypt the ARN of the KMS key by following the steps in [Permissions to decrypt the secret for your vector store in Pinecone or Redis Enterprise Cloud](#).
5. Later, when you create your knowledge base, enter the ARN in the **Credentials secret ARN** field.

**Create a vector store in Redis Enterprise Cloud**

<i>Note</i>

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 vector store services to you. You're
responsible for complying with any third-party terms applicable to use and transfer of data from the third-party service.

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

While you set up the vector store, 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 store 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 Amazon 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 an AWS Identity and Access Management (IAM) role that provides Amazon Bedrock permission to access other AWS services. You can let Amazon Bedrock create the service role (for more information, see Service roles for Amazon Bedrock, or you can choose a custom role that you have created.
   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. Select the Chunking strategy for your data source.

⚠️ Important
You can't change the chunking strategy after you have created the data source.

- **Default chunking** – By default, Amazon Bedrock 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.

- **Fixed size chunking** – Amazon Bedrock splits your source data into chunks of the approximate size that you set. Configure the following options.
  - **Max tokens** – Amazon Bedrock creates chunks that don't exceed the number of tokens that you choose.
  - **Overlap percentage between chunks** – Each chunk overlaps with consecutive chunks by the percentage that you choose.

- **No chunking** – Amazon Bedrock treats each file as one chunk. If you choose this option, you may want to pre-process your documents by splitting them into separate files.

iv. While converting your data into embeddings, Amazon Bedrock encrypts your data with a key that AWS owns and manages, by default. To use your own KMS key, expand Additional settings, select Customize encryption settings (advanced), and choose a key.

b. Select Next.

6. On the Configure vector store page, set up the embeddings model to use and the configurations for your vector store.

a. 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 G1 Embeddings - Text model is available.

b. In the Vector store section, choose one of the following options.
• **Default OpenSearch vector store** – Amazon Bedrock creates an Amazon OpenSearch Serverless vector search collection and automatically configures the settings for embedding your data sources and manages the collection for you.

• **Customize your vector store** – 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). 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 Serverless, 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 store for you.

7. 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**.

8. 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 Serverless 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**.

If you are using a custom role, 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.

   ```json
   [
     {
       "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 knowledge base service role, replacing `knowledge-base-arn`.

Create a knowledge base
Ingest your data sources into the knowledge base

After you create your knowledge base, you ingest the data sources into the knowledge base so that they are indexed and able to be queried.

Before you begin ingestion, check that your files belong to a supported format and that they do not exceed the maximum file size, as detailed in Set up your data for ingestion.

To ingest your data sources

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select Knowledge base and choose your knowledge base.
3. In the Data source section, select Sync to begin data ingestion.
4. When data ingestion completes, a green success banner appears if it is successful.
5. You can choose a data source to view its Sync history. Select View warnings to see why a data ingestion job failed.
Test your knowledge base

After you set up your knowledge base, you can test its behavior by sending queries and seeing the responses. Use this feature to assess your knowledge base before attaching it to an agent.

To test your knowledge base

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. From the left navigation pane, select Knowledge base.
3. In the Knowledge bases section, do one of the following actions.
   • Choose the radio button next to the knowledge base you want to test and select Test knowledge base.
   • Choose the knowledge base you want to test and then select Test or select the left arrow in the top right corner of the page. A chat window expands from the right.
4. Enter a query in the text box in the chat window and select Run to return responses from the knowledge base. You can return responses in one of two ways:
   • To return information retrieved directly from your knowledge base, turn off Generate responses for your query. Amazon Bedrock returns text chunks from your data sources that are relevant to the query.
   • To generate responses based on information retrieved from your knowledge base, turn on Generate responses for your query. Choose the model to use for response generation and select Apply. Amazon Bedrock generates responses based on your data sources and cites the information it provides with footnotes.
5. To view the text chunk from which the information in the response is retrieved, select the corresponding footnote. For an example, refer to the following image. If you select [1] Air travel.pdf, a Source chunks window appears.
6. In the **Source chunks** window, you can carry out the following actions

- Select a footnote to view the text chunk in the data source from which your knowledge base retrieved information.
- Select **Copy preview** to copy the text chunk.
- Select **View S3 location** to navigate to the S3 object and view the entire data source.
- Enter text in the **Search** bar to find the text sequence in the source chunk.

While using the chat window to test your knowledge base, you can also carry out the following actions.

- If you are generating responses instead of returning direct quotations, you can select **Change model** to use a different model for response generation. If you change the model, the text in the chat window will be completely cleared.
- Switch between generating responses for your query and returning direct quotations by toggling **Generate responses for your query** on and off. If you change the setting, the text in the chat window will be completely cleared.
- To clear the chat window, select the broom icon.
- To copy all the output in the chat window, select the copy icon.

### Manage your knowledge base

**To manage a knowledge base**

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**


2. Before the following steps, make sure to delete the knowledge base from any agents that it's associated with. 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**.

3. From the left navigation pane, select **Knowledge base**.

4. 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.
5. Review the warnings for deleting a knowledge base. If you accept these conditions, enter `delete` in the input box and select Delete to confirm.

⚠️ Warning

If you delete a knowledge base or a data source, the underlying data that was indexed remains in the vector store you set up and can still be retrieved. To delete the data, you need to delete the vector index containing the data embeddings.

Add a knowledge base to an agent

To add a knowledge base to an agent

1. Open the Amazon 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.

Use the API

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

- **Agents for Amazon Bedrock API Reference** – Includes API operations for setting up and configuring agents, knowledge bases, and their associated resources.
- **Agents for Amazon Bedrock Runtime API Reference** – Includes API operations for invoking agents and querying knowledge bases.

The following are the service endpoints for Agents for Amazon Bedrock. 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.

**Agents for Amazon Bedrock build-time APIs**
To set up a knowledge base, you call the following **Agents for Amazon Bedrock APIs** – Includes API operations to set up and manage your knowledge bases.

1. **CreateKnowledgeBase** – Create the knowledge base with the configurations for the vector store in which you want to store embeddings.
2. **CreateDataSource** – Create data sources to add to the knowledge base.
3. **StartIngestionJob** – Ingest the data sources to index the documents into your vector store.

Once you have set up the knowledge base, you can query it with the following **Agents for Amazon Bedrock Runtime APIs** – Includes API operations to query your knowledge bases and generate text from the retrieved data.

1. **Retrieve** – Query a knowledge base and retrieve relevant text from its data sources.
2. **RetrieveAndGenerate** – Query a knowledge base and return a response generated by a foundation model from information in its data sources, alongside citations and the source text.
Let your agent query a knowledge base by sending an AssociateKnowledgeBase request. For more information, see [Create and manage a knowledge base associated with an agent](#).

**Topics**

- [Create and manage knowledge bases](#)
- [Create and manage data sources](#)
- [Start and manage ingestion jobs](#)
- [Query a knowledge base](#)

### Create and manage knowledge bases

**Topics**

- [Create a knowledge base](#)
- [Edit a knowledge base](#)
- [Get information about a knowledge base](#)
- [List information about your knowledge bases](#)
- [Delete a knowledge base](#)

#### Create a knowledge base

To create a knowledge base, send a CreateKnowledgeBase request and provide the name, description, instructions for what it should do, and the foundation model for it to orchestrate with.

**Note**

If you prefer to let Amazon Bedrock create and manage a vector store for you in Amazon OpenSearch Service, use the console. For more information, see [Create a knowledge base](#).

- Provide the ARN with permissions to create a knowledge base in the roleArn field.
- Provide the embedding model to use in the embeddingModelArn field in the knowledgeBaseConfiguration object.
- Provide the configuration for your vector store in the storageConfiguration object.
• For an Amazon OpenSearch Service database, use the opensearchServerlessConfiguration object. For more information, see [Create a vector store in Amazon OpenSearch Serverless](#).

• For a Pinecone database, use the pineconeConfiguration object. For more information, see [Create a vector store in Pinecone](#).

• For a Redis Enterprise Cloud database, use the redisEnterpriseCloudConfiguration object. For more information, see [Create a vector store in Redis Enterprise Cloud](#).

**Edit a knowledge base**

To edit a knowledge base, send an UpdateKnowledgeBase request. Include both fields that you want to update as well as fields that you want to keep the same.

**Get information about a knowledge base**

To get information about a knowledge base, send a GetKnowledgeBase request and specify the ID of the knowledge base.

**List information about your knowledge bases**

To list information about your knowledge bases, send a ListKnowledgeBases request.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a nextToken that you can send in another ListKnowledgeBases request to see the next batch of results.

**Delete a knowledge base**

To delete a knowledge base, send a DeleteKnowledgeBase request.

⚠️ **Warning**

Before deleting a knowledge base, you should disassociate the knowledge base from any agents that it is associated with by making a DisassociateAgentKnowledgeBase request.
Warning

If you delete a knowledge base or a data source, the underlying data that was indexed remains in the vector store you set up and can still be retrieved. To delete the data, you need to delete the vector index containing the data embeddings.

Create and manage data sources

Topics

- Create a data source
- Update an data source
- Get information about an data source
- List information about a knowledge base's data sources
- Delete a data source

Create a data source

To create a data source for your knowledge base, send an `CreateDataSource` request.

- Provide the information of the S3 bucket containing the data source files in the `dataSourceConfiguration` object.
- Specify how to chunk the data sources in the `vectorIngestionConfiguration`. For more information, see Chunk your data source in Set up your data for ingestion.
- (Optional) For extra security,

Update an data source

To update a data source (for example, if you have added, changed, or removed any files from it), send an `UpdateKnowledgeBaseActionGroup` request, specifying the fields that you want to change in the data source configuration.

Get information about an data source

To get information about an data source, send a `GetDataSource` request and specify the ID of the data source and of the knowledge base that it belongs to.
List information about a knowledge base's data sources

To list information about a knowledge base's data sources, send a ListDataSources request and specify the ID of the knowledge base.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a nextToken that you can send in another ListDataSources request to see the next batch of results.

Delete a data source

To delete a data source from a knowledge base, send a DeleteDataSource request and specify the ID of the data source and of the knowledge base.

Start and manage ingestion jobs

Topics

• Start an ingestion job
• Get information about an ingestion job
• List ingestion jobs

Start an ingestion job

To ingest a data source into the vector store you configured for your knowledge base, send a StartIngestionJob request. Specify the ID of the knowledge base and data source.

Use the ingestionJobId returned in the response to track the status of the ingestion job.

Get information about an ingestion job

To get information about an ingestion job, send a GetIngestionJob request. Specify the ID of the knowledge base, data source, and ingestion job.

When the ingestion job finishes, the status in the response is COMPLETE.

The statistics object in the response returns information about whether ingestion was successful or not for documents in the data source.
List ingestion jobs

To list ingestion jobs for a data source in a knowledge base, send a `ListIngestionJobs` request. Specify the ID of the knowledge base and data source. You can set the following specifications.

- Filter for results by specifying a status to search for in the `filters` object.
- Sort by the time that the job was started or the status of a job by specifying the `sortBy` object. You can sort in ascending or descending order.
- Set the maximum number of results to return in a response in the `maxResults` field. If there are more results than the number you set, the response returns a `nextToken` that you can send in another `ListIngestionJobs` request to see the next batch of jobs.

Query a knowledge base

Query a knowledge base by using a Amazon Bedrock Agent runtime endpoint (`bedrock-agent-runtime.region.amazonaws.com`).

You can return text directly from your data sources with a `Retrieve` request, or generate responses based on text from the data sources by using a foundation model in a `RetrieveAndGenerate` request.

Topics
- `Retrieve`
- `RetrieveAndGenerate`

Retrieve

To query a knowledge base and only return relevant text from data sources, send a `Retrieve` request and specify the ID of the knowledge base to query.

- Send the query in the `text` field of the `retrievalQuery` object.
- Specify how many results to return in the `numberOfResults` field in the `retrievalConfiguration`.

The response returns the text in the `text` field of the `content` object. You can find the S3 location of the data source in the `location` object.
If there are more results than can be returned in a single response, you can use the value in the nextToken field in another Retrieve request to return the next batch of results.

**RetrieveAndGenerate**

To query a knowledge base and use a foundation model to generate responses based off the results from the data sources, send a [RetrieveAndGenerate](https://docs.aws.amazon.com/bedrock/latest/userguide/RetrieveAndGenerate.html) request.

- Send the query in the text field of the input object.
- Specify the ID of the knowledge base to query and the model to use for response generation in the retrieveAndGenerateConfiguration object.
- If you reuse a sessionId, the model returns a response taking into account previous queries that were sent to the knowledge base.
- (Optional) For security purposes, you can specify the ARN of a KMS key in the kmsKeyArn field in the sessionConfiguration object to encrypt the session.

The response includes the following components.

- The output contains the full generated text.
- The citations list contains an object for each part of the response that contains information from a data source.
  - The generatedResponsePart object contains the text that is generated based off a data source and where it starts and ends in the span object.
  - The retrievedReferences object contains information about the data source that is cited for the generated text, including the text in the data source chunk and the S3 location of the data source.
- You can use the sessionId in another RetrieveAndGenerate request to continue the same querying session.
Agents for Amazon Bedrock

Agents for Amazon Bedrock offers you the ability to build and configure autonomous agents in your application. The agent helps your end-users 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 and invoke knowledge bases to supplement information for these actions. Developers can easily integrate the agents and accelerate delivery of generative AI applications saving weeks of development effort.

With agents, 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. Amazon Bedrock manages prompt engineering, memory, monitoring, encryption, user permissions, and API invocation.

Agents 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 querying data sources.
- Carry out source attribution.

To take advantage of agents, you carry out the following steps:

1. (Optional) Create a knowledge base to store your private data in that database. For more information, see Knowledge base for Amazon Bedrock.
2. Create an agent for your use-case, add actions that it can carry out and define how to handle them by writing Lambda functions in a programming language of your choice.
3. Attach the knowledge base you created to augment its performance. For more information, see Create an agent.
4. (Optional) Modify prompt templates for the pre-processing, orchestration, knowledge base response generation, and post-processing steps carried out by an agent in order to customize the agent’s behavior to your specific use-case. For more information, see Advanced prompts.

5. Test your agent in the console or through API calls to the TestAlias and modify the configurations as necessary. Use traces to examine your agent’s reasoning process at each step of its orchestration. For more information, see Test your agent and Trace events.

6. 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 Deploy your agent: versioning and aliases.

7. Set up your application to make API calls to your agent alias.

8. Iterate on your agent and create more versions and aliases as necessary.

Topics
- How Agents for Amazon Bedrock works
- Supported regions and models
- Set up action groups for your agent
- Create an agent
- Test your agent
- Manage your agent
- Advanced prompts
- Deploy your agent: versioning and aliases
- Use the API

How Agents for Amazon Bedrock works

Agents for Amazon Bedrock consists of two main sets of API operations to help you set up and run an agent, build-time API operations to create, configure, and manage your agents and their related resources, and runtime API operations to invoke your agent with user input and to initiate orchestration to carry out a task.

Build-time execution

An agent consists of the following components:
• Foundation model – You choose a foundation model that the agent invokes to interpret user input and subsequent prompts in its orchestration process, and to generate responses and follow-up steps in its process.

• Instructions – You write up instructions that describe what the agent is designed to do. With advanced prompts, you can further customize instructions for the agent at every step of orchestration and include Lambda functions to parse the output of each step.

• (Optional) Action groups – You define the actions that the agent should carry out through providing two resources.
  • An OpenAPI schema to define the APIs that the agent can invoke to carry out its tasks.
  • A Lambda function with the following input and output.
    • Input – The API and parameters identified during orchestration.
    • Output – The result of the API invocation.

• (Optional) Knowledge bases – Associate knowledge bases with an agent to allow it to query the knowledge base for extra context to augment response generation and input into steps of the orchestration process.

In build-time, all these components are gathered to construct base prompts for the agent in order to carry out orchestration until the user request is completed. With advanced prompts, you can modify these base prompts with additional logic and few-shot examples to improve accuracy for each step of agent invocation. The base prompt templates contain instructions, action descriptions, knowledge base descriptions, and conversation history, all of which you can customize to modify the agent to the best of your needs. You then prepare your agent, which packages all the components of the agents, including security configurations, and brings the agent into a state where it is ready for testing in runtime. The following image schematizes how build-time APIs construct your agent.
Runtime execution

Runtime is managed by one API, the `InvokeAgent`, which sets off the agent sequence, which consists of three main steps.

1. Pre-processing – Manages how the agent contextualizes and categorizes user input and whether the input is valid.

2. Orchestration – Interprets the user input, invokes action groups and queries knowledge bases, and returns output to the user or as input to continued orchestration. Orchestration consists of the following steps.
   - The agent interprets the input with a foundation model and generates a *rationale* that lays out the logic for the next step it should take.
• The agent invokes action groups and queries knowledge bases to retrieve additional context and summarize the data to augment its generation.

• The agent generates an output, known as an observation, from invoking action groups and summarizing results from knowledge bases. The agent uses the observation to augment the base prompt, which is then interpreted with a foundation model. The agent then determines if it needs to reiterate the orchestration process.

• This loop continues until the agent is ready to return a response to the user or until it needs to prompt the user for extra information.

During orchestration, the base prompt template is augmented with the agent instructions, action groups, and knowledge bases that you have added to the agent. The augmented base prompt is then used to invoke the foundation model. The foundation model predicts the best possible steps and trajectory in order to fulfill the user input. At each iteration of orchestration, the foundation model predicts the API to invoke or the knowledge base to query.

3. Post-processing – The agent creates a final response from parts of the API and knowledge base responses. This step is turned off by default.

As the user session with the agent continues through more InvokeAgent requests, conversation history is preserved and serves to continually augment the orchestration base prompt template with context, helping improve accuracy and performance. The following diagram schematizes the agent's process during runtime.
Tracing your agent's reasoning process

Agents for Amazon Bedrock provides the ability to turn on a trace at runtime, which allows you to track the agent's rationale, actions, queries, and observations at each step of the agent sequence. The trace includes the full prompt sent to as input to the foundation model at each step and the outputs from the foundation model, API responses, and knowledge base queries. You can use the rationale to understand the agent's reasoning at each step. For more information, see Trace events.
Advanced prompts

Agents for Amazon Bedrock exposes the default base prompt templates that are used during pre-processing, orchestration, knowledge base response generation, and post-processing. You can edit these base prompt templates to customize your agent's behavior at each step of its sequence. You can also turn off steps for debugging purposes or if you deem them unnecessary. For more information, see Advanced prompts.

Supported regions and models

Agents for Amazon Bedrock is supported in the following regions.

- US East (N. Virginia)
- US West (Oregon)

You can use Agents for Amazon Bedrock with the following models.

- Anthropic Claude Instant v1
- Anthropic Claude v2.0

Set up action groups for your agent

Before you create your agent, you should set up action groups and knowledge bases that you want to add to your agent.

- Action groups define the tasks that you want your agent to help customers carry out.
- Knowledge bases provide a repository of information that the agent can query to answer customer queries and improve its generated responses. For more information, see Knowledge base for Amazon Bedrock.

An action group consists of the following components that you set up.

- An OpenAPI schema that define the APIs that your action group should call. Your agent uses the API schema to determine the fields it needs to elicit from the customer to populate for the API request.
- A Lambda function that defines the business logic for the action that your agent will carry out.
Topics

- Action group OpenAPI schemas
- Action group Lambda functions

Action group OpenAPI schemas

When you create an action group, you must define the APIs that the agent can invoke with an OpenAPI schema in JSON or YAML format. You can create OpenAPI schema files yourself and upload them to Amazon S3 buckets, or you can use our OpenAPI text editor in order to easily validate your schema. The text editor is available after you create the agent. You can use the text editor when you add an action group to an agent or edit an existing one (for more information, see Manage your agent).

- For more details about OpenAPI schemas, see OpenAPI specification.
- For best practices in writing API schemas, see Best practices in API design.

The following is the general format of an OpenAPI schema for an action group.

```json
{
   "openapi": "3.0.0",
   "paths": {
      "/path": {
         "method": {
            "description": "string",
            "operationId": "string",
            "parameters": [ ... ],
            "requestBody": { ... },
            "responses": { ... }
         }
      }
   }
}
```

These fields are described below.

- openapi – (Required) The version of OpenAPI being used. This value must be "3.0.0" or higher for the action group to work.
• paths – (Required) Contains relative paths to individual endpoints. Each path must begin with a forward slash (/).

• method – (Required) Defines the method to use.

Minimally, each method requires the following fields.

• description – A description of the API operation. Use this field to inform the agent when this API should be called and what it does.

• responses – The properties defined for the operation response are not just used for constructing prompts. They are used for accurately processing the results of an API call, and for determining a proper set of steps for carrying out a task. By knowing the response coming back from one operation, the agent can know that those values can be used as inputs for subsequent steps in the process.

The fields within the following two objects provide more information for your agent to effectively take advantage of your action group. For each field, you can set the required field (true if required, false if optional).

• parameters – Contains information about parameters that can be included in the request.

• requestBody – Contains the fields in the request body for the operation. Don’t include this field for GET and DELETE methods.

Select a tab to learn more about a structure

response

```json
"responses": {
  "200": {
    "content": {
      "<media type>": {
        "schema": {
          "properties": {
            "<property>": {
              "type": "string",
              "description": "string"
            },
            ...
          }
        }
      }
    }
  }
}
```
Each key in the responses object is a response code, which describes the status of the response, mapping to an object containing the following information for the response.

- **content** – (Required for each response) The content of the response.
- **<media type>** – The format of the response body. For more information, see [Media types](#).
- **schema** – (Required for each media type) Defines the data type of the response body and its fields.
- **properties** – (Required if there are items in the schema) Your agent uses properties that you define in the schema to determine the information it needs to return to the end user in order to fulfill a task. Each property contains the following fields:
  - **type** – (Required for each property) The data type of the response field.
  - **description** – (Optional) Describes the property. The agent can use this information to determine the information it needs to return to the end user.

```json
"parameters": [  
  
  
  ]
```

Your agent uses the following fields to determine the information it needs to get from the agent user in order to carry out the action group.

- **name** – (Required) The name of the parameter.
• description – (Required) A description of the parameter. Use this field to help the agent understand how to elicit this parameter from the agent user or determine that it already has that parameter value from prior actions or from the user's request to the agent.

• required – (Optional) Whether the parameter is required for the API request or not. Use this field to let the agent know whether this parameter is needed for every invocation or if it is optional.

• schema – (Optional) The definition of input and output data types. For more information, see Data Models (Schemas).

requestBody

The general structure of a requestBody field is below.

```
"requestBody": {
    "required": boolean,
    "content": {
        "<media type>": {
            "schema": {
                "properties": {
                    "<property>": {
                        "type": "string",
                        "description": "string"
                    },
                    ...
                }
            }
        }
    }
}
```

The fields are described below:

• required – (Optional) Whether the request body is required for the API request or not.

• content – (Required) The content of the request body.

• <media type> – (Optional) The format of the request body. For more information, see Media types)

• schema – (Optional) Defines the data type of the request body and its fields.
• properties – (Optional) Your agent uses properties that you define in the schema to determine the information it needs to get from the agent user in order to carry out the action group. Each property contains the following fields:
  • type – (Optional) The data type of the request field.
  • description – (Optional) Describes the property. The agent can use this information to determine the information it needs to return to the end user.

Example API schema

The following API schema defines a group of APIs that help handle insurance claims. There are three APIs that are defined:

• getAllOpenClaims – Your agent can use the description field to determine that it should call this API if a list of open claims is needed. The properties in the responses specify to return the ID and the policy holder and the status of the claim. The agent returns this information to the agent user or uses some or all of the response as input to subsequent API calls.

• identifyMissingDocuments – Your agent can use the description field to determine that it should call this API if missing documents need to be identified for an insurance claim. The name, description, and required fields tell the agent that it must elicit the unique identifier of the open claim from the customer. The properties in the responses specify to return the IDs of the open insurance claims. The agent returns this information to the customer or uses some or all of the response as input to subsequent API calls.

• sendReminders – Your agent can use the description field to determine that it should call this API if there is a need to send reminders to the customer about pending documents that they have for open claims. The properties in the requestBody tell the agent that it needs to find the claim IDs and the pending documents, whereas the properties in the responses specify to return an ID of the reminder and its status. The agent returns this information to the customer or uses some or all of the response as input to subsequent API calls.

```json
{
    "openapi": "3.0.0",
    "info": {
        "title": "Insurance Claims Automation API",
        "version": "1.0.0",
    }
}
```
"description": "APIs for managing insurance claims by pulling a list of open claims, identifying outstanding paperwork for each claim, and sending reminders to policy holders."
},
"paths": {
  "/claims": {
    "get": {
      "summary": "Get a list of all open claims",
      "description": "Get the list of all open insurance claims. Return all the open claimIds."
    },
    "operationId": "getAllOpenClaims",
    "responses": {
      "200": {
        "description": "Gets the list of all open insurance claims for policy holders",
        "content": {
          "application/json": {
            "schema": {
              "type": "array",
              "items": {
                "type": "object",
                "properties": {
                  "claimId": {
                    "type": "string",
                    "description": "Unique ID of the claim."
                  },
                  "policyHolderId": {
                    "type": "string",
                    "description": "Unique ID of the policy holder who has filed the claim."
                  },
                  "claimStatus": {
                    "type": "string",
                    "description": "The status of the claim. Claim can be in Open or Closed state"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
"get": {
    "summary": "Identify missing documents for a specific claim",
    "description": "Get the list of pending documents that need to be
uploaded by policy holder before the claim can be processed. The API takes in only one
claim id and returns the list of documents that are pending to be uploaded by policy
holder for that claim. This API should be called for each claim id",
    "operationId": "identifyMissingDocuments",
    "parameters": [{
        "name": "claimId",
        "in": "path",
        "description": "Unique ID of the open insurance claim",
        "required": true,
        "schema": {
            "type": "string"
        }
    }],
    "responses": {
        "200": {
            "description": "List of documents that are pending to be
uploaded by policy holder for insurance claim",
            "content": {
                "application/json": {
                    "schema": {
                        "type": "object",
                        "properties": {
                            "pendingDocuments": {
                                "type": "string",
                                "description": "The list of pending
documents for the claim."
                            }
                        }
                    }
                }
            }
        }
    }
}
"summary": "API to send reminder to the customer about pending documents for open claim",
"description": "Send reminder to the customer about pending documents for open claim. The API takes in only one claim id and its pending documents at a time, sends the reminder and returns the tracking details for the reminder. This API should be called for each claim id you want to send reminders for."
"operationId": "sendReminders",
"requestBody": {
  "required": true,
  "content": {
    "application/json": {
      "schema": {
        "type": "object",
        "properties": {
          "claimId": {
            "type": "string",
            "description": "Unique ID of open claims to send reminders for."
          },
          "pendingDocuments": {
            "type": "string",
            "description": "The list of pending documents for the claim."
          }
        },
        "required": [
          "claimId",
          "pendingDocuments"
        ]
      }
    }
  },
  "required": ["claimId", "pendingDocuments"
    ]
},
"responses": {
  "200": {
    "description": "Reminders sent successfully",
    "content": {
      "application/json": {
        "schema": {
          "type": "object",
          "properties": {
            "sendReminderTrackingId": {
              "type": "string",
              "description": "Tracking id for the reminder sent to the customer."
            }
          }
        }
      }
    }
  }
}
Amazon Bedrock

User Guide

"description": "Unique Id to track the status of the send reminder Call",
"sendReminderStatus": {
  "type": "string",
  "description": "Status of send reminder notifications"
}

"400": {
  "description": "Bad request. One or more required fields are missing or invalid."
}

For more examples of OpenAPI schemas, see https://github.com/OAI/OpenAPI-Specification/tree/main/examples/v3.0.

Action group Lambda functions

You can define a Lambda function to program the business logic for carrying out an action group and to customize how you want the API response to be returned. You use the variables from the input event to define your functions and return a response to the agent. To write your function, you will need to understand the format of the input event and expected response. If you use a Lambda function for your action group, you must attach a resource-based policy to your Lambda function by following the steps at Permissions to access your action group Lambda functions. For more information about resource-based policies in Lambda, see Using resource-based policies for Lambda.

Topics

- Lambda input event from Amazon Bedrock
- Lambda response event to Amazon Bedrock
Lambda input event from Amazon Bedrock

When an action group using a Lambda function is invoked, Amazon Bedrock sends a Lambda input event of the following general format. You can define your Lambda function to use any of the input event fields to manipulate the business logic within the function to successfully carry out the action. For more information about Lambda functions, 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",
  "actionGroup": "string",
  "apiPath": "string",
  "httpMethod": "string",
  "parameters": [
    {
      "name": "string",
      "type": "string",
      "value": "string"
    },
    ...
  ],
  "requestBody": {
    "content": {
      "<content_type>": {
        "properties": [
          {
            "name": "string",
            "type": "string",
            "value": "string"
          },
          ...
        ]
      }
    }
  }
}
```
The input event fields are described below.

- **messageVersion** – The version of the message that identifies the format of the event data going into the Lambda function and the expected format of the response from a Lambda function. Amazon Bedrock only supports version 1.0.
- **agent** – Contains information about the name, ID, alias, and version of the agent that the action group belongs to.
- **inputText** – The user input for the conversation turn.
- **sessionId** – The unique identifier of the agent session.
- **actionGroup** – The name of the action group.
- **apiPath** – The path to the API operation, as defined in the OpenAPI schema.
- **httpMethod** – The method of the API operation, as defined in the OpenAPI schema.
- **parameters** – Contains a list of objects. Each object contains the name, type, and value of a parameter in the API operation, as defined in the OpenAPI schema.
- **requestBody** – Contains the request body and its properties, as defined in the OpenAPI schema.
- **sessionAttributes** – Contains session attributes and their values.
- **promptSessionAttributes** – Contains prompt attributes and their values.

**Lambda response event to Amazon Bedrock**

Amazon Bedrock expects a response from your Lambda function that matches the following format. The response consists of response parameters from the API after the action has been carried out. The agent can use the response from the Lambda function for further orchestration or to help it return a response to the customer.
The response fields are described below. Note that sessionAttributes and promptSessionAttributes are optional.

- **messageVersion** – The version of the message that identifies the format of the event data going into the Lambda function and the expected format of the response from a Lambda function. Amazon Bedrock only supports version 1.0.
- **response** – Contains the following information about the API response.
  - **actionGroup** – The name of the action group.
  - **apiPath** – The path to the API operation, as defined in the OpenAPI schema.
  - **httpMethod** – The method of the API operation, as defined in the OpenAPI schema.
  - **responseBody** – Contains the response body, as defined in the OpenAPI schema.
  - (Optional) **sessionAttributes** – Contains session attributes and their values.
  - (Optional) **promptSessionAttributes** – Contains prompt attributes and their values.

### Action group Lambda function example

The following is an minimal example of how the Lambda function can be defined in Python.
def lambda_handler(event, context):
    response_body = {
        'application/json': {
            'body': "sample response"
        }
    }
    action_response = {
        'actionGroup': event['actionGroup'],
        'apiPath': event['apiPath'],
        'httpMethod': event['httpMethod'],
        'httpStatusCode': 200,
        'responseBody': response_body
    }
    session_attributes = event['sessionAttributes']
    prompt_session_attributes = event['promptSessionAttributes']
    api_response = {
        'messageVersion': '1.0',
        'response': action_response,
        'sessionAttributes': session_attributes,
        'promptSessionAttributes': prompt_session_attributes
    }
    return api_response

Create an agent

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

- The configuration of the agent itself, which defines the purpose of the agent and the foundation model that it should use to interpret and generate prompts.
- 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.
If you plan to attach a knowledge base to your agent, first set up your knowledge base by following the steps at Knowledge base for Amazon Bedrock.

Topics

- Prerequisites
- Steps to create an agent

Prerequisites

Before you can create your agent, carry out the following steps.

1. (Optional) If you opt out of using a role automatically created by Amazon Bedrock, set up a custom role by following the steps at Permissions to create and manage an agent.

2. When you create an agent, you define action groups that your agent will carry out. For each action group, you need to set up the following items.

   a. Set up the Lambda function that defines the business logic for the action that your agent will carry out. For more information, see Action group Lambda functions.

   i. Note

      Configure permissions for your agent to be able to access the Lambda function. For more information, see Permissions to access your action group Lambda functions. You need to attach these permissions even if you use a role that is automatically created by Amazon Bedrock during the agent creation process.

   b. Set up the OpenAPI schema with the API description, structure, and parameters for the action group. Upload a schema you have created to an Amazon S3 bucket or define the schema with the help of the inline OpenAPI schema editor in the console after you have created the agent. For more information, see Action group OpenAPI schemas.

3. Set up any knowledge bases that you want your agent to be able to query. For more information, see Knowledge base for Amazon Bedrock.
Steps to create an agent

To create an agent, visit [https://console.aws.amazon.com/bedrock/](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 Agents for Amazon Bedrock permission to access other AWS services. You can let Agents for Amazon Bedrock create the service role (for more information, see [Service roles for Amazon Bedrock](#)) or you can choose a custom role that you have created.

   **Note**

   The service role that Agents for Amazon Bedrock creates for you doesn't support permissions for features that are in preview. In order to use these features, you must attach the correct permissions to the service role.

   **Note**

   For information about the requirements for a custom role, see [Permissions to create and manage an agent](#).

4. By default, AWS encrypts sessions with your agent with a key that Agents for Amazon Bedrock owns. To encrypt your agent with your own key, select **Customize encryption settings (advanced)** in the KMS key selection section and **Choose an AWS KMS key**. If you want to create a new key, select **Create an AWS KMS key** and reload this window.
5. In the **Idle session timeout** section, choose a duration of time. If the user has not responded in the same session after this amount of time, Amazon Bedrock no longer maintains the conversation history. Conversation history is used to both resume an interaction and to augment responses with context from the conversation.
6. (Optional) In the Tags section, select Add new tag to associate tags with this agent. Fill in a key-value pair for each tag to add.

7. 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. In the Model details selection, 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.

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. For more information, see Action group Lambda functions.

   **Note**
   
   Remember to attach a resource-based policy to the Lambda function to allow the Amazon Bedrock service principal to access the Lambda function. For more information, see Permissions to access your action group Lambda functions.

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 more information, see Action group OpenAPI schemas.
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](#) 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.

**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* version and a TestAlias alias that points to it. 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 TestAlias, You can also select a different alias to test. In the test window, you can choose 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 events](#). To learn how to enable the trace through the API, see [Invoke your agent](#).
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 and select its **Working draft**.
2. Select the **Working draft** and select **Prepare** to package the agent with your latest changes. You must carry out this step to be able to test the agent.

   **Note**
   As a best practice, we recommend to always check your agent's last prepared time so that you can ensure that you are testing your agent with the latest configurations.

3. In any of the pages inside an agent, you can select the left arrow icon at the top right to expand the test window.
4. Use the dropdown menu at the top of the test window to choose an alias and associated version to test.

   **Note**
   If you have not yet prepared your agent, a message appears in the test window to prompt you to **Prepare** your agent.

5. Enter a message to request the agent to perform a task. Use the test window to help debug your agent.
6. 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 **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 events**.
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.
6. Select **Prepare** to apply the changes that you have made to the agent before testing it.

**Trace events**

Each response from an agent is accompanied by a *trace* that allows you to receive events on the steps being orchestrated by 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](#).

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. Each **Step** can be one of the following traces.

- **PreProcessingTrace** – Traces the input and output of the pre-processing step, in which the agent contextualizes and categorizes user input and determines if it is valid.
- **OrchestrationTrace** – Traces the input and output of the orchestration step, in which the agent interprets the input, invokes APIs and queries knowledge bases, and returns output to either continue orchestration or to respond to the user.
- **PostProcessingTrace** – Traces the input and output of the post-processing step, in which the agent handles the final output of the orchestration and determines how to return the response to the user.
- **FailureTrace** – Traces the reason that a step failed.

Each of the traces (except the FailureTrace) contains a `ModelInvocationInput` object that contains configurations set in the agent's prompt template (for more information, see **Advanced prompts**), alongside the prompt provided to the agent at this step. The structure of the `ModelInvocationInput` object is as follows.

```
{
  "traceId": "string",
  "text": "string",
  "type": "PRE_PROCESSING | ORCHESTRATION | KNOWLEDGE_BASE_RESPONSE_GENERATION | POST_PROCESSING",
  "inferenceConfiguration": {
    "maximumLength": number,
    "stopSequences": ["string"],
    "temperature": float,
    "topK": float,
    "topP": float
  },
  "promptCreationMode": "DEFAULT | OVERRIDDEN",
  "parserMode": "DEFAULT | OVERRIDDEN",
  "overrideLambda": "string"
}
```

The fields are described below.

- **traceId** – The unique identifier of the trace.
- **text** – The text from the prompt provided to the agent at this step.
• type – The current step in the agent's process.
• inferenceConfiguration – Inference parameters that influence response generation.
• promptCreationMode – Whether the agent's default base prompt template was overridden for this step or not.
• parserMode – Whether the agent's default response parser was overridden for this step or not.
• overrideLambda – The ARN of the parser Lambda function used to parse the response, if the default parser was overridden.

Each of the traces is described below.

**PreProcessingTrace**

```json
{
   "modelInvocationInput": { // see above for details }
   "modelInvocationOutput": {
      "parsedResponse": {
         "isValid": boolean,
         "rationale": "string"
      },
      "traceId": "string"
   }
}
```

Apart from the `modelInvocationInput`, the **PreProcessingParsedResponse** contains the following fields.

• parsedResponse – Contains the following details about the parsed user prompt.
  • isValid – Specifies whether the user prompt is valid or not.
  • rationale – Specifies the agent's reasoning for the next steps to take.
  • traceId – The unique identifier of the trace.

**OrchestrationTrace**

The orchestration trace consists of the **ModelInvocationInput** object and any combination of the **Rationale**, **InvocationInput**, and **Observation** objects. Select a tab below to learn about an object.

```json
{
   "modelInvocationInput": { // see above for details },
}
```
Rationale

The Rationale object contains the reasoning of the agent given the user input. The structure is shown below.

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

The fields are described below.

- **traceId** – The unique identifier of the trace step.
- **text** – The reasoning or thought process of the agent, based on the input prompt.

InvocationInput

The InvocationInputTrace object contains information that will be input to the action group or knowledge base that is to be invoked or queried. The structure is shown below.

```json
{
  "traceId": string,
  "invocationType": "ACTION_GROUP | KNOWLEDGE_BASE | FINISH",
  "actionGroupInvocationInput": {
    "actionGroupName": "string",
    "apiPath": "string",
    "verb": "string",
    "parameters": [
      {
        "name": "string",
        "type": "string",
        "value": "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 ending the session.
- **actionGroupInvocationInput** – Appears if the type is ACTION_GROUP. Contains the following input for the action group being invoked. For more information, see Action group OpenAPI schemas.
  - **actionGroupName** – The name of the action group that the agent is going to invoke.
  - **apiPath** – The path to the API to call, based off the action group or the API schema.
  - **verb** – The API method being used, based off the action group or the API schema.
  - **parameters** – Contains a list of objects. Each object contains the name, type, and value of a parameter in the API operation, as defined in the OpenAPI schema.
  - **requestBody** – Contains the request body and its properties, as defined in the OpenAPI schema.
- **knowledgeBaseLookupInput** – Appears if the type is KNOWLEDGE_BASE. 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 will look up.
- text – The query to be made to the knowledge base.

Observation

The Observation object contains the result or output of an action group or knowledge base, or the response to the user. The structure is shown below.

```json
{
    "traceId": "string",
    "type": "ACTION_GROUP | KNOWLEDGE_BASE | REPROMPT | ASK_USER | FINISH",
    "actionGroupInvocation": {
        "text": "JSON-formatted string"
    },
    "knowledgeBaseLookupOutput": {
        "retrievedReferences": [
            {
                "content": {
                    "text": "string"
                },
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "uri": "string"
                    }
                }
            }
        ],
        "repromptResponse": {
            "source": "ACTION_GROUP | KNOWLEDGE_BASE | PARSER",
            "text": "string"
        },
        "finalResponse": {
            "text"
        }
    }
}
```

The fields are described below.

- traceId – The unique identifier of the trace.
• **type** – Specifies whether the agent's observation is returned from the result of an action group or a knowledge base, or if the agent is reprompting the user, asking for more information, or ending the conversation.

• **actionGroupInvocationOutput** – Contains the JSON-formatted string returned by the API invoked by the action group. Appears if the type is `ACTION_GROUP`.

• **knowledgeBaseLookupOutput** – Contains text retrieved from the knowledge base that is relevant to responding to the prompt, alongside the S3 location of the data source. Appears if the type is `KNOWLEDGE_BASE`. Each object in the list of `retrievedReferences` contains the following fields.
  - **content** – Contains text from the knowledge base that is returned from the knowledge base query.
  - **location** – Contains the S3 URI of the data source from which the returned text was found.
  - **repromptResponse** – Appears if the type is `RE_PROMPT`. Contains the text that asks for a prompt again alongside the source of why the agent needs to reprompt.
  - **finalResponse** – Appears if the type is `ASK_USER` or `FINISH`. Contains the text that asks the user for more information or is a response to the user.

**PostProcessingTrace**

```json
{
    "modelInvocationInput": {}, // see above for details
    "modelInvocationOutput": {
        "parsedResponse": {
            "text": "string"
        },
        "traceId": "string"
    }
}
```

Apart from the `modelInvocationInput`, the **PostProcessingParsedResponse** contains the following fields.

• **parsedResponse** – Contains the text to return to the user after being processed by the parser function.

• **traceId** – The unique identifier of the trace.
FailureTrace

```json
{
    "failureReason": "string",
    "traceId": "string"
}
```

The fields are described below.

- **failureReason** – The reason that the step failed.
- **traceId** – The unique identifier of the trace.

## Manage 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.

### Note

If you change the foundation model, all the prompt templates will be set to default for that model. For more information, see the section called “Advanced prompts”.

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. You can configure advanced points in this section. For more information about advanced prompts, see Advanced prompts.

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. To define the schema for the action group with the in-line OpenAPI schema editor, carry out the following actions. For more information about API schemas for your action group, see Action group OpenAPI schemas.
   a. Choose Define with in-line OpenAPI schema editor under Select API schema. A sample schema appears that you can edit.
   b. Select the format for the schema by using the dropdown menu next to Format.
   c. To import an existing schema from S3 to edit, select Import schema, provide the S3 URI, and select Import.
   d. To restore the schema to the original sample schema, select Reset and then confirm the message that appears by selecting Reset again.

4. Select Add. A green success banner appears if there are no issues. If there are issues validating the schema, a red banner appears. The following issues are identified by the validation process.
   - Scroll through the schema to see the lines where an error or warning about formatting exists. An X indicates a formatting error, while an exclamation mark indicates a warning about formatting.
   - Select View details in the red banner to see a list of errors about the content of the API schema.
5. Select **Prepare** to apply the changes that you have made to the agent before testing it.

### 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. To define the schema for the action group with the in-line OpenAPI schema editor, carry out the following actions.
   a. Choose **Define with in-line OpenAPI schema editor** under **Select API schema**. A sample schema appears that you can edit.
   b. Select the format for the schema by using the dropdown menu next to **Format**.
   c. To import an existing schema from S3 to edit, select **Import schema**, provide the S3 URI, and select **Import**.
   d. To restore the schema to the original sample schema, select **Reset** and then confirm the message that appears by selecting **Reset** again.
4. Select **Save** to remain in the same window. A green success banner appears if there are no issues.
5. If there are issues validating the schema, a red banner appears. Scroll through the schema to see the lines where an error or warning exists. Select the gear icon or **View details** at the top of the page to see a list of errors.
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.
8. Select **Prepare** to apply the changes that you have made to the agent before testing it.

### To delete an action group

1. In the Amazon Bedrock console, select **Agents** from the left navigation pane.
2. Select the working draft from the **Working draft** section.
3. In the **Action groups** section, select the radio button next to the action group to delete.
4. A modal appears warning you about the consequences of deletion. Enter **delete** in the input box to confirm the message and select **Delete** to confirm.
5. A blue banner appears to inform you that the action group is being deleted. When deletion is complete, a green success banner appears.

6. Select Prepare to apply the changes that you have made to the agent before testing it.

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.
4. Select Prepare to apply the changes that you have made to the agent before testing it.

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.
7. Select Prepare to apply the changes that you have made to the agent before testing it.

To delete a knowledge base

1. Select the radio button next to the knowledge base to delete.
2. Select Delete.
3. Select Prepare to apply the changes that you have made to the agent before testing it.
Delete an agent

To delete an agent

1. In the Amazon Bedrock console, select **Agents** from the left navigation pane.
2. Select the radio button next to the agent to delete.
3. A modal appears warning you about the consequences of deletion. Enter **delete** in the input box and select **Delete** to confirm.
4. A blue banner appears to inform you that the agent is being deleted. When deletion is complete, a green success banner appears.

Advanced prompts

Gain more control over the behavior of your agent by defining and configuring prompt templates for your agent. These prompt templates help you customize how the agent processes user input text and output prompts from foundation models, how it orchestrates between the model, action groups, and knowledge bases, and how it returns responses to the user. Advanced prompts allow you to enhance your agent's accuracy through more detailed configurations and the ability to provide hand-curated examples for **few-shot prompting**, a procedure that lets you improve model performance by providing some labeled examples related to a specific task.

The following terminology is helpful in understanding how advanced prompts work.

- **Session** – A group of **InvokeAgent** calls made to the same agent with the same session ID. When you make an InvokeAgent request, you can reuse a sessionId that was returned from the response of a previous call in order to continue the same session with an agent. As long as the idleSessionTTLInSeconds time that you set in the **Agent** configuration has not expired, you maintain the same session with the agent.

- **Turn** – A single InvokeAgent call. A sessions consists of one or more turns.

- **Iteration** – An iteration is a sequence of the following actions (an action may be skipped depending on the configuration of the agent or the agent's need at that moment). A turn consists of one or more iterations.
  1. (Required) A call to the foundation model
  2. (Optional) An action group invocation
  3. (Optional) A knowledge base invocation
4. (Optional) A response to the user asking for more information

- **Prompt** – A prompt consists of the instructions to the agent, context, and text input. The text input may come from a user or from the output of another step in the agent sequence. The prompt is provided to the foundation model in order to determine the next step that the agent takes in responding to user input.

- **Base prompt template** – The structural elements that make up a prompt. The template consists of placeholders that are filled in by user input, the agent configuration, and context to create a prompt (for more information, see [Prompt template placeholder variables](#)). Advanced prompts offers templates for the following parts of the agent sequence. At each of these steps, the base prompt template is used alongside the input to that step in order to generate a prompt for the foundation model to interpret.
  
  - **Pre-processing** – Defines how an agent contextualizes and categorizes user inputs. Can be used to validate user input.
  
  - **Orchestration** – Defines the order in which actions are executed and which knowledge bases are retrieved.
  
  - **Knowledge base response generation (Retrieval)** – Generates a response based on retrieved results for the query.
  
  - **Post-processing** – Defines how an agent formats and represents a final response to the end user.

With advanced prompts, you can toggle invocation for different steps in the agent sequence, configure their inference parameters, and edit the default base prompt templates that the agent uses in order to override the logic with your own configurations in order to customize your agent’s behavior. You can edit the following parts.

- **Prompt template** – Describe how the agent should interpret and use the prompt that it receives at the step for which you are editing the template. For information on placeholder variables that you can use in a template, see [Prompt template placeholder variables](#).

You can enable or disable any step in the agent sequence. The following are the default states for each step.

- Pre-processing – enabled
- Orchestration – enabled
- Knowledge base response generation – enabled
- Post-processing – disabled
Note

If you disable the orchestration step, the agent sends the user input to the foundation model and doesn't use the base prompt template for orchestration. If you disable any of the other steps, the agent skips that step entirely.

- **Inference configurations** – Influence the response generated by the model that you use. For definitions of the inference parameters and more details about the parameters that different models support, see [Inference parameters for foundation models](#).

- **(Optional) Parser Lambda function** – Defines how to parse the raw foundation model output and how to use it in the runtime flow. This function acts on the output from the steps in which you enable it and returns the parsed response as you define it in the function.

  Depending on how you customized the base prompt template, the raw foundation model output might be specific to the template and the agent's default parser may have difficulty parsing it correctly. Writing a custom parser Lambda function allows you to help the agent parse the raw foundation model output based on your use-case. To see details about the parser Lambda function and how to write it, see [Parser Lambda function](#).

Note

You can define one parser Lambda function for all of the base templates, but you can configure whether to invoke the function in each step or not. Be sure to configure a resource-based policy for your Lambda function so that your agent can invoke it. For more information, see [Permissions to access your action group Lambda functions](#).

After you edit the prompt templates, you can test your agent and examine the trace to analyze the step-by-step thought process of the agent and determine if it is working as you intend it to. For more information, see [Trace events](#).

You can configure advanced prompts in either the console or through the API.

Console

In the console, you can configure advanced prompts after you have created the agent. You configure them while editing the agent.
To view or edit advanced prompts for your agent

1. Choose an agent in the **Agents** section and select the **Working draft**.
2. In the **Advanced prompts** section, select the tab corresponding to the part of the agent sequence for which you want to edit.
3. Turn on **Override template defaults** to enable editing of the template. Select **Confirm** for the message that appears.

**Warning**

If you turn off **Override template defaults** or change the model, the default Amazon Bedrock template is used and your template will be deleted. To confirm, enter **confirm** in the text box to confirm the message that appears.

4. Turn on **Activate template** to allow it to be used by the agent when generating responses. If this configuration is turned off, the agent doesn't use it.
5. Use the **Prompt template editor** to modify the example prompt template.
6. Modify inference parameters under **Configurations**. For definitions of parameters and more information about parameters for different models, see [Inference parameters for foundation models](#).
7. (Optional) To use a Lambda function that you have defined to parse the raw foundation model output, carry out the following actions.

**Note**

One Lambda function is used for all the prompt templates.

a. Turn on **Use Lambda function for parsing** in the **Configurations** section. If you turn this setting off, your agent will use the default parser for the prompt.

b. Select a Lambda function in the **Parser Lambda function** section.
8. Save your settings and **Prepare** the agent to test with these updated settings.

**API**

To configure advanced prompts through the API, you send a `CreateAgent` or `UpdateAgent` call and modify the following `promptOverrideConfiguration` object.

```json
"promptOverrideConfiguration": {  
  "overrideLambda": "string",
  "promptConfigurations": [
    {
      "basePromptTemplate": "string",
      "inferenceConfiguration": {
        "maximumLength": int,
        "stopSequences": ["string"],
        "temperature": float,
        "topK": float,
        "topP": float
      },
      "parserMode": "DEFAULT | OVERRIDDEN",
      "promptCreationMode": "DEFAULT | OVERRIDDEN",
      "promptState": "ENABLED | DISABLED",
      "promptType": "PRE_PROCESSING | ORCHESTRATION | KNOWLEDGE_BASE_RESPONSE_GENERATION | POST_PROCESSING"
    }
  ]
}
```

1. In the `promptConfigurations` list, include a `promptConfiguration` object for each prompt template that you want to edit.

2. Specify the prompt to modify in the `promptType` field.

3. Configure the prompt template that you want to replace the default one through the following steps.
a. Specify the basePromptTemplate fields with your prompt template.

b. Include inference parameters in the inferenceConfiguration objects. For more information about inference configurations, see Inference parameters for foundation models.

4. To enable the prompt template, carry out the following steps. Set the promptCreationMode to OVERRIDDEN.

5. To allow or prevent the agent to carry out the step in the promptType field, modify the promptState value. This setting can be useful when debugging the agent's behavior.

   • If you set the promptState to DISABLED for the PRE_PROCESSING, KNOWLEDGE_BASE_RESPONSE_GENERATION, or POST_PROCESSING steps, that step is skipped by the agent.
   
   • If you set the promptState to DISABLED for the ORCHESTRATION step, the agent sends only the user input to the foundation model in orchestration and returns the response as is without orchestrating calls between APIs and knowledge bases.
   
   • By default, the POST_PROCESSING step is DISABLED, while the PRE_PROCESSING, ORCHESTRATION, and KNOWLEDGE_BASE_RESPONSE_GENERATION steps are ENABLED.

6. To use a Lambda function that you have defined to parse the raw foundation model output, carry out the following steps.

   a. For each prompt template that you want to enable the Lambda function for, set the parserMode to OVERRIDDEN.

   b. Specify the ARN of the Lambda function in the overrideLambda field in the promptOverrideConfiguration object.

Topics

- Prompt template placeholder variables
- Parser Lambda function

Prompt template placeholder variables

You can use the following placeholder variables in the prompt templates. The variables will be populated by pre-existing configurations when the prompt template is called. The chart below defines each variable and which prompt templates it can be used in.
• **Pre-processing**
  
  - `$functions$` – Replaced by the formatted action group APIs and knowledge bases configured for the agent.
  
  - `$conversation_history$` – Replaced by the formatted conversation history for the current session.
  
  - `$question$` – Replaced by the user input for the current `invokeAgent` call in the session.

• **Orchestration**
  
  - `$instruction$` - Replaced by the model instructions configured for the agent.
  
  You can use the following placeholders if you allow the agent to ask the user for more information. This setting is specified in the `parentActionSignature` field of the `AgentActionGroup`.
  
  - `$ask_user_input_examples$` – Conditionally replaced by few-shot examples to tell the model how to predict when it should ask the user a question.
  
  - `$ask_user_function$` - Conditionally replaced by a function to ask the user a question.
  
  - `$ask_user_confirm_parameters$` - Conditionally replaced by instructions for the model to ask the user to confirm parameters that the agent has not yet received or is unsure of.
  
  - `$ask_user_missing_parameters$` - Conditionally replaced by instructions for the model to ask the user to provide required missing parameters.
  
  - `$ask_user_function_format$` - Conditionally replaced by the format the model should predict ask user function in if user input is enabled for the agent
  
  - `$functions$` - Replaced by the formatted action group APIs and knowledge bases configured for the agent.
  
  - `$prompt_session_attributes$` - Replaced by the prompt session attributes provided for the session
  
  - `$any_function_name$` - Replaced by a configured function name as an example
  
  - `$conversation_history$` - Replaced by the formatted conversation history for the current session.
  
  - `$question$` - Replaced by the user input for the current turn in the session
  
  - `$agent_scratchpad$` - Designates an area for the model to write down its thoughts and actions it has taken. Replaced with the predictions and output of the previous iterations in the current turn in order for model to have context of what has been achieved for the given user input and what should be the next step.

• **Knowledge base response generation**
• $search_results$ - Replaced by the retrieved results for the user query
• $query$ - Replaced by the query generated by the orchestration prompt model response when it predicts a knowledge base to be invoked

**Post-processing**
• $question$ - Replaced by the user input for the current turn in the session
• $latest_response$ - Replaced by the last orchestration prompt model response
• $responses$ - Replaced by the action group and knowledge base outputs invoked in the current turn.

**Parser Lambda function**

In order to write a parser Lambda function for your prompt template, you need to understand the input event that your agent sends and the response that the agent expects as the output from the Lambda function. You write a handler function to manipulate variables from the input event and to return the response. For more information about how AWS Lambda works, see [Event-driven invocation](#).

**Topics**

- **Parser Lambda input event**
- **Parser Lambda response**
- **Parser Lambda examples**

**Parser Lambda input event**

The following is the general structure of the input event from the agent. Use the fields to write your Lambda handler function.

```
{
   "messageVersion": "1.0",
   "agent": {
      "name": "string",
      "id": "string",
      "alias": "string",
      "version": "string"
   },
   "invokeModelRawResponse": "string",
```

The input event fields are described below.

- **messageVersion** – The version of the message that identifies the format of the event data going into the Lambda function and the expected format of the response from the Lambda function. Agents for Amazon Bedrock only supports version 1.0.

- **agent** – Contains information about the name, ID, alias, and version of the agent that the prompts belongs to.

- **invokeModelRawResponse** – The raw foundation model output of the prompt whose output is to be parsed.

- **promptType** – The prompt type whose output is to be parsed.

- **overrideType** – The artifacts that this Lambda function overrides. Currently, only `OUTPUT_PARSER` is supported. This indicates that the default parser is to be overridden.

### Parser Lambda response

Your agent expects a response from your Lambda function that matches the following format. The agent uses the response for further orchestration or to help it return a response to the customer. Use these fields to shape how the output is returned.

```json
{
    "messageVersion": "1.0",
    "promptType": "ORCHESTRATION | PRE_PROCESSING | POST_PROCESSING | KNOWLEDGE_BASE_RESPONSE_GENERATION",
    "preProcessingParsedResponse": {
        "isValidInput": "boolean",
        "rationale": "string"
    },
    "orchestrationParsedResponse": {
        "rationale": "string",
        "parsingErrorDetails": {
            "repromptResponse": "string"
        },
        "responseDetails": {
            "invocationType": "ACTION_GROUP | KNOWLEDGE_BASE | FINISH | ASK_USER",
```

**Parser Lambda function**
"agentAskUser": {
  "responseText": "string"
},
"actionGroupInvocation": {
  "actionGroupName": "string",
  "apiName": "string",
  "verb": "string",
  "actionGroupInput": {
    "<parameter>": {
      "value": "string"
    },
    ...
  }
},
"agentKnowledgeBase": {
  "knowledgeBaseId": "string",
  "searchQuery": {
    "value": "string"
  }
},
"agentFinalResponse": {
  "responseText": "string",
  "citations": {
    "generatedResponseParts": [
      {
        "text": "string",
        "references": [{"sourceId": "string"}]
      }
    ]
  }
},
"knowledgeBaseResponseGenerationParsedResponse": {
  'generatedResponse': {
    'generatedResponseParts': [
      {
        'text': 'string',
        'references': [
          {'sourceId': 'string'},
          ...
        ]
      }
    ]
  }
}
The Lambda response fields are described below. You can use the Lambda function to shape how this information is returned.

- **messageVersion** – The version of the message that identifies the format of the event data going into the Lambda function and the expected format of the response from a Lambda function. Agents for Amazon Bedrock only supports version 1.0.

- **promptType** – The prompt type of the current turn.

- **preProcessingParsedResponse** – The parsed response for the PRE_PROCESSING prompt type.

- **orchestrationParsedResponse** – The parsed response for the ORCHESTRATION prompt type. See below for more details.

- **knowledgeBaseResponseGenerationParsedResponse** – The parsed response for the KNOWLEDGE_BASE_RESPONSE_GENERATION prompt type.

- **postProcessingParsedResponse** – The parsed response for the POST_PROCESSING prompt type.

The parsed responses for the four steps are expanded upon below.

**preProcessingParsedResponse**

```
{
   "isValidInput": "boolean",
   "rationale": "string"
}
```
The `preProcessingParsedResponse` contains the following fields.

- `isValidInput` – Specifies whether the user input is valid or not. You can define the function to determine how to characterize the validity of user input.

- `rationale` – The reasoning for the user input categorization. This rationale is provided by the model in the raw response, the lambda parses it and agents present in the trace for pre-processing.

```
orchestrationResponse
{
  "rationale": "string",
  "parsingErrorDetails": {
    "repromptResponse": "string"
  },
  "responseDetails": {
    "invocationType": "ACTION_GROUP | KNOWLEDGE_BASE | FINISH | ASK_USER",
    "agentAskUser": {
      "responseText": "string"
    },
    "actionGroupInvocation": {
      "actionGroupName": "string",
      "apiName": "string",
      "verb": "string",
      "actionGroupInput": {
        "<parameter>": {
          "value": "string"
        },
        ...
      }
    },
    "agentKnowledgeBase": {
      "knowledgeBaseId": "string",
      "searchQuery": {
        "value": "string"
      }
    },
    "agentFinalResponse": {
      "responseText": "string",
      "citations": {
        "generatedResponseParts": [
          {
            ...
          }
        ]
      }
    }
  }
}
```
The orchestrationParsedResponse contains the following fields.

- `rationale` – The reasoning for what to do next, based on the foundation model output. You can define the function to parse from the model output.

- `parsingErrorDetails` – Contains the repromptResponse, which is the message to reprompt the model to update its raw response when the model response can’t be parsed. You can define the function to manipulate how to reprompt the model.

- `responseDetails` – Contains the details for how to handle the output of the foundation model. Contains an `invocationType`, which is the next step for the agent to take, and a second field that should match the `invocationType`. The following objects are possible.
  - `agentAskUser` – Compatible with the `ASK_USER` invocation type. This invocation type ends the orchestration step. Contains the `responseText` to ask the user for more information. You can define your function to manipulate this field.
  - `actionGroupInvocation` – Compatible with the `ACTION_GROUP` invocation type. Contains the following fields, which define the following. You can define your function to determine action groups to invoke and parameters to pass.
    - `actionGroupName` – The action group to invoke.
    - `apiName` – The name of the API to invoke in the action group.
    - `verb` – The method of the API to use.
    - `actionGroupInput` – Contains parameters to specify in the API request.
  - `agentKnowledgeBase` – Compatible with the `KNOWLEDGE_BASE` invocation type. Contains the following fields. You can define your function to determine how to query knowledge bases.
    - `knowledgeBaseId` – The unique identifier of the knowledge base.
• `searchQuery` – Contains the query to send the knowledge base in the value field.

• `agentFinalResponse` – Compatible with the FINISH invocation type. This invocation type ends the orchestration step. Contains the response to the user in the `responseText` field and citations for the response in the `citations` object.

```json
class knowledgeBaseResponseGenerationParsedResponse {

    generatedResponse: {
        generatedResponseParts: [
            {
                text: 'string',
                references: [
                    { 'sourceId': 'string' },
                    ...
                ],
            },
            ...
        ]
    }
}
```

The `knowledgeBaseResponseGenerationParsedResponse` contains the `generatedResponse` from querying the knowledge base and references for where the information comes from.

```json
class postProcessingParsedResponse {

    "responseText": "string",
    "citations": { 
        "generatedResponseParts": [
            {
                "text": "string",
                "references": [
                    { "sourceId": "string" },
                    ...
                ],
            },
            ...
        ]
    }
}
```

Parser Lambda function
The `postProcessingParsedResponse` contains the following fields.

- `responseText` – The response to return to the end user. You can define the function to format the response.
- `citations` – Contains a list of citations for the response. Each citation shows the text that is cited and its references.

**Parser Lambda examples**

Select a tab to see an example parser Lambda function for a prompt template and example input events sent to function and responses from it. The `lambda_handler` function returns the parsed response to the agent.

**Pre-processing**

**Example function**

```python
import json
import re
import logging

PRE_PROCESSING_RATIONALE_REGEX = "<thinking>(.*)</thinking>"
PREPROCESSING_CATEGORY_REGEX = "<category>(.*)</category>"
PREPROCESSING_PROMPT_TYPE = "PRE_PROCESSING"
PRE_PROCESSING_RATIONALE_PATTERN = re.compile(PRE_PROCESSING_RATIONALE_REGEX, re.DOTALL)
PREPROCESSING_CATEGORY_PATTERN = re.compile(PREPROCESSING_CATEGORY_REGEX, re.DOTALL)

logger = logging.getLogger()

# This parser lambda is an example of how to parse the LLM output for the default PreProcessing prompt
def lambda_handler(event, context):
    print("Lambda input: " + str(event))
    logger.info("Lambda input: " + str(event))
    prompt_type = event["promptType"]
```
# Sanitize LLM response
model_response = sanitize_response(event['invokeModelRawResponse'])

if event['promptType'] == PREPROCESSING_PROMPT_TYPE:
    return parse_pre_processing(model_response)

def parse_pre_processing(model_response):
    category_matches = re.finditer(PREPROCESSING_CATEGORY_PATTERN, model_response)
    rationale_matches = re.finditer(PRE_PROCESSING_RATIONALE_PATTERN, model_response)

    category = next((match.group(1) for match in category_matches), None)
    rationale = next((match.group(1) for match in rationale_matches), None)

    return {
        "promptType": "PRE_PROCESSING",
        "preProcessingParsedResponse": {
            "rationale": rationale,
            "isValidInput": get_is_valid_input(category)
        }
    }

def sanitize_response(text):
    pattern = r"(\n*)"
    text = re.sub(pattern, r"\n", text)
    return text

def get_is_valid_input(category):
    if category is not None and category.strip().upper() == "D" or category.strip().upper() == "E":
        return True
    return False

{
    "agent": {
        "alias": "TSTALIASID",
        "id": "AGENTID123",
        "name": "InsuranceAgent",
        "version": "DRAFT"
    }
}
Orchestration

Example function

```python
import json
import re
import logging

RATIONALE_REGEX_LIST = [
    "(.*)(<function_call>)",
    "(.*)(<answer>)"
]
RATIONALE_PATTERNS = [re.compile(regex, re.DOTALL) for regex in RATIONALE_REGEX_LIST]

RATIONALE_VALUE_REGEX_LIST = [
    "<scratchpad>(.*)</scratchpad>",
    "(.*)</scratchpad>",
    "(<scratchpad>)(.*)"
]
```
RATIONALE_VALUE_PATTERNS = [re.compile(regex, re.DOTALL) for regex in RATIONALE_VALUE_REGEX_LIST]

ANSWER_REGEX = r"(?<=<answer>)(.*)"
ANSWER_PATTERN = re.compile(ANSWER_REGEX, re.DOTALL)

ANSWER_TAG = "<answer>"
FUNCTION_CALL_TAG = "<function_call>"

ASK_USER_FUNCTION_CALL_REGEX = r"(<function_call>user::askuser)(.*))""
ASK_USER_FUNCTION_CALL_PATTERN = re.compile(ASK_USER_FUNCTION_CALL_REGEX, re.DOTALL)

ASK_USER_FUNCTION_PARAMETER_REGEX = r"(?<=askuser="')(.*?)"'
ASK_USER_FUNCTION_PARAMETER_PATTERN = re.compile(ASK_USER_FUNCTION_PARAMETER_REGEX, re.DOTALL)

KNOWLEDGE_STORE_SEARCH_ACTION_PREFIX = "x_amz_knowledgebase_"

FUNCTION_CALL_REGEX = r"<function_call>(\w+)::(\w+):(.+):(\w+):(\w+):(\w+):(\w+):(\w+)"

ANSWER_PART_REGEX = "<answer_part\s?>(.+?)</answer_part\s?>"
ANSWER_TEXT_PART_REGEX = "<text\s?>(.+?)</text\s?>"
ANSWER_REFERENCE_PART_REGEX = "<source\s?>(.+?)</source\s?>"
ANSWER_PART_PATTERN = re.compile(ANSWER_PART_REGEX, re.DOTALL)
ANSWER_TEXT_PART_PATTERN = re.compile(ANSWER_TEXT_PART_REGEX, re.DOTALL)
ANSWER_REFERENCE_PART_PATTERN = re.compile(ANSWER_REFERENCE_PART_REGEX, re.DOTALL)

# You can provide messages to reprompt the LLM in case the LLM output is not in the expected format
MISSING_API_INPUT_FOR_USER_REPROMPT_MESSAGE = "Missing the argument askuser for user::askuser function call. Please try again with the correct argument added"
ASK_USER_FUNCTION_CALL_STRUCTURE_REMPROMPT_MESSAGE = "The function call format is incorrect. The format for function calls to the askuser function must be: <function_call>user::askuser(askuser="$ASK_USER_INPUT")</function_call>.
FUNCTION_CALL_STRUCTURE_REPROMPT_MESSAGE = 'The function call format is incorrect. The format for function calls must be: <function_call> $FUNCTION_NAME($FUNCTION_ARGUMENT_NAME=""$FUNCTION_ARGUMENT_NAME"")</function_call>.'

logger = logging.getLogger()

# This parser lambda is an example of how to parse the LLM output for the default orchestration prompt
def lambda_handler(event, context):
logger.info("Lambda input: " + str(event))

# Sanitize LLM response
sanitized_response = sanitize_response(event['invokeModelRawResponse'])

# Parse LLM response for any rationale
rationale = parse_rationale(sanitized_response)

# Construct response fields common to all invocation types
parsed_response = {
    'promptType': "ORCHESTRATION",
    'orchestrationParsedResponse': {
        'rationale': rationale
    }
}

# Check if there is a final answer
try:
    final_answer, generated_response_parts = parse_answer(sanitized_response)
    except ValueError as e:
        addRepromptResponse(parsed_response, e)
    return parsed_response

if final_answer:
    parsed_response['orchestrationParsedResponse']['responseDetails'] = {
        'invocationType': 'FINISH',
        'agentFinalResponse': {
            'responseText': final_answer
        }
    }

    if generated_response_parts:
        parsed_response['orchestrationParsedResponse']['responseDetails']['agentFinalResponse']['citations'] = {
            'generatedResponseParts': generated_response_parts
        }

    logger.info("Final answer parsed response: " + str(parsed_response))
    return parsed_response

# Check if there is an ask user
try:
    ask_user = parse_ask_user(sanitized_response)
    if ask_user:
parsed_response['orchestrationParsedResponse']['responseDetails'] = {
    'invocationType': 'ASK_USER',
    'agentAskUser': {
        'responseText': ask_user
    }
}

logger.info("Ask user parsed response: " + str(parsed_response))
return parsed_response
except ValueError as e:
    addRepromptResponse(parsed_response, e)
return parsed_response

# Check if there is an agent action
try:
parsed_response = parse_function_call(sanitized_response, parsed_response)
logger.info("Function call parsed response: " + str(parsed_response))
return parsed_response
except ValueError as e:
    addRepromptResponse(parsed_response, e)
return parsed_response

addRepromptResponse(parsed_response, 'Failed to parse the LLM output')
logger.info(parsed_response)
return parsed_response
raise Exception("unrecognized prompt type")

def sanitize_response(text):
    pattern = r'([^n]*)'
    text = re.sub(pattern, r'\n', text)
    return text

def parse_rationale(sanitized_response):
    # Checks for strings that are not required for orchestration
    rationale_matcher = next((pattern.search(sanitized_response) for pattern in RATIONALE_PATTERNS if pattern.search(sanitized_response)), None)

    if rationale_matcher:
        rationale = rationale_matcher.group(1).strip()

        # Check if there is a formatted rationale that we can parse from the string
        rationale_value_matcher = next((pattern.search(rationale) for pattern in RATIONALE_VALUE_PATTERNS if pattern.search(rationale)), None)
if rationale_value_matcher:
    return rationale_value_matcher.group(1).strip()

return rationale

return None

def parse_answer(sanitized_llm_response):
    if has_generated_response(sanitized_llm_response):
        return parse_generated_response(sanitized_llm_response)

    answer_match = ANSWER_PATTERN.search(sanitized_llm_response)
    if answer_match and is_answer(sanitized_llm_response):
        return answer_match.group(0).strip(), None

    return None, None

def is_answer(llm_response):
    return llm_response.rfind(ANSWER_TAG) > llm_response.rfind(FUNCTION_CALL_TAG)

def parse_generated_response(sanitized_llm_response):
    results = []

    for match in ANSWER_PART_PATTERN.finditer(sanitized_llm_response):
        part = match.group(1).strip()

        text_match = ANSWER_TEXT_PART_PATTERN.search(part)
        if not text_match:
            raise ValueError("Could not parse generated response")

        text = text_match.group(1).strip()
        references = parse_references(sanitized_llm_response, part)
        results.append((text, references))

    final_response = " ".join([r[0] for r in results])

    generated_response_parts = []
    for text, references in results:
        generatedResponsePart = {
            'text': text,
            'references': references
        }
        generated_response_parts.append(generatedResponsePart)
def has_generated_response(raw_response):
    return ANSWER_PART_PATTERN.search(raw_response) is not None

def parse_references(raw_response, answer_part):
    references = []
    for match in ANSWER_REFERENCE_PART_PATTERN.finditer(answer_part):
        reference = match.group(1).strip()
        references.append({'sourceId': reference})
    return references

def parse_ask_user(sanitized_llm_response):
    ask_user_matcher = ASK_USER_FUNCTION_CALL_PATTERN.search(sanitized_llm_response)
    if ask_user_matcher:
        try:
            ask_user = ask_user_matcher.group(2).strip()
            ask_user_question_matcher = ASK_USER_FUNCTION_PARAMETER_PATTERN.search(ask_user)
            if ask_user_question_matcher:
                return ask_user_question_matcher.group(1).strip()
        except ValueError as ex:
            raise ex
        except Exception as ex:
            raise Exception(ASK_USER_FUNCTION_CALL_STRUCTURE_REMPROMPT_MESSAGE)
    return None

def parse_function_call(sanitized_response, parsed_response):
    match = re.search(FUNCTION_CALL_REGEX, sanitized_response)
    if not match:
        raise ValueError(FUNCTION_CALL_STRUCTURE_REPROMPT_MESSAGE)
    verb, resource_name, function = match.group(1), match.group(2), match.group(3)
    parameters = []
    for arg in match.group(4).split(",\":
        key, value = arg.split("=")
        parameters[key.strip()] = {'value': value.strip('" ')}
    parsed_response['orchestrationParsedResponse']['responseDetails'] = {}}
Function calls can either invoke an action group or a knowledge base.

Mapping to the correct variable names accordingly

```python
if resource_name.lower().startswith(KNOWLEDGE_STORE_SEARCH_ACTION_PREFIX):
    parsed_response['orchestrationParsedResponse']['responseDetails']
['invocationType'] = 'KNOWLEDGE_BASE'
    parsed_response['orchestrationParsedResponse']['responseDetails']
['agentKnowledgeBase'] = {
    'searchQuery': parameters['searchQuery'],
    'knowledgeBaseId':
        resource_name.replace(KNOWLEDGE_STORE_SEARCH_ACTION_PREFIX, '')
}

    return parsed_response

parsed_response['orchestrationParsedResponse']['responseDetails']
['invocationType'] = 'ACTION_GROUP'
    parsed_response['orchestrationParsedResponse']['responseDetails']
['actionGroupInvocation'] = {
    'verb': verb,
    'actionGroup': resource_name,
    'apiName': function,
    'actionGroupInput': parameters
}

    return parsed_response
```

```python
def addRepromptResponse(parsed_response, error):
    error_message = str(error)
    logger.warn(error_message)
    parsed_response['orchestrationParsedResponse']['parsingErrorDetails'] = {
        'repromptResponse': error_message
    }
```

Example request

```json
{
    'agent': {
        'alias': 'TSTALIASID',
        'id': 'AGENTID123',
        'name': 'InsuranceAgent',
        'version': 'DRAFT'
    },
    ...other keys...
}
```
'invokeModelRawResponse': 'To answer this question, I will:

1. Call the GET::x_amz_knowledgebase_KBID123456::Search function to search for a phone number to call.
2. I have checked that I have access to the GET::x_amz_knowledgebase_KBID23456::Search function.

</scratchpad>

<function_call>GET::x_amz_knowledgebase_KBID123456::Search(searchQuery="What is the phone number I can call?")',

'messageVersion': '1.0',

'overrideType': 'OUTPUT_PARSER',

'promptType': 'ORCHESTRATION'
}

Example response

{
    'promptType': 'ORCHESTRATION',
    'orchestrationParsedResponse': {
        'rationale': 'To answer this question, I will:

1. Call the GET::x_amz_knowledgebase_KBID123456::Search function to search for a phone number to call Farmers.
2. I have checked that I have access to the GET::x_amz_knowledgebase_KBID123456::Search function.',
        'responseDetails': {
            'invocationType': 'KNOWLEDGE_BASE',
            'agentKnowledgeBase': {
                'searchQuery': {'value': 'What is the phone number I can call?'},
                'knowledgeBaseId': 'KBID123456'
            }
        }
    }
}

Knowledge base response generation

Example function

```python
import json
import re
import logging

ANSWER_PART_REGEX = "<answer_part\s?>(.+?)</answer_part\s?>"
ANSWER_TEXT_PART_REGEX = "<text\s?>(.+?)</text\s?>"
ANSWER_REFERENCE_PART_REGEX = "<source\s?>(.+?)</source\s?>"
ANSWER_PART_PATTERN = re.compile(ANSWER_PART_REGEX, re.DOTALL)
ANSWER_TEXT_PART_PATTERN = re.compile(ANSWER_TEXT_PART_REGEX, re.DOTALL)
```
ANSWER_REFERENCE_PART_PATTERN = re.compile(ANSWER_REFERENCE_PART_REGEX, re.DOTALL)

logger = logging.getLogger()

# This parser lambda is an example of how to parse the LLM output for the default KB response generation prompt
def lambda_handler(event, context):
    logger.info("Lambda input: " + str(event))
    raw_response = event['invokeModelRawResponse']

    parsed_response = {
        'promptType': 'KNOWLEDGE_BASE_RESPONSE_GENERATION',
        'knowledgeBaseResponseGenerationParsedResponse': {
            'generatedResponse': parse_generated_response(raw_response)
        }
    }

    logger.info(parsed_response)
    return parsed_response

def parse_generated_response(sanitized_llm_response):
    results = []

    for match in ANSWER_PART_PATTERN.finditer(sanitized_llm_response):
        part = match.group(1).strip()

        text_match = ANSWER_TEXT_PART_PATTERN.search(part)
        if not text_match:
            raise ValueError("Could not parse generated response")

        text = text_match.group(1).strip()
        references = parse_references(sanitized_llm_response, part)
        results.append(((text, references))

    generated_response_parts = []
    for text, references in results:
        generatedResponsePart = {
            'text': text,
            'references': references
        }
        generated_response_parts.append(generatedResponsePart)

    return {
        'generatedResponseParts': generated_response_parts"
Example request

```json
{
  'agent': {
    'alias': 'TSTALIASID',
    'id': 'AGENTID123',
    'name': 'InsuranceAgent',
    'version': 'DRAFT'
  },
  'invokeModelRawResponse': '{"completion":" <answer>\\n<answer_part>\n<text>\\nThe search results contain information about different types of insurance benefits, including personal injury protection (PIP), medical payments coverage, and lost wages coverage. PIP typically covers reasonable medical expenses for injuries caused by an accident, as well as income continuation, child care, loss of services, and funerals. Medical payments coverage provides payment for medical treatment resulting from a car accident. Who pays lost wages due to injuries depends on the laws in your state and the coverage purchased. \\
</text>\\n<sources>\\n<source>1234567-1234-1234-1234-123456789abc</source>\\n<source>2345678-2345-2345-2345-23456789abcd</source>\\n<source>3456789-3456-3456-3456-3456789abcde</source>\\n</sources>\\n</answer_part>\\n</answer>',
  'stop_reason': 'stop_sequence',
  'stop': '',
  'messageVersion': '1.0',
  'overrideType': 'OUTPUT_PARSER',
  'promptType': 'KNOWLEDGE_BASE_RESPONSE_GENERATION'
}
```

Example response

```json
{
  'promptType': 'KNOWLEDGE_BASE_RESPONSE_GENERATION',
  'knowledgeBaseResponseGenerationParsedResponse': {
    'generatedResponse': {
      'generatedResponseParts': [
```
The search results contain information about different types of insurance benefits, including personal injury protection (PIP), medical payments coverage, and lost wages coverage. PIP typically covers reasonable medical expenses for injuries caused by an accident, as well as income continuation, child care, loss of services, and funerals. Medical payments coverage provides payment for medical treatment resulting from a car accident. Who pays lost wages due to injuries depends on the laws in your state and the coverage purchased.
Example request

```json
{
    'agent': {
        'alias': 'TSTALIASID',
        'id': 'AGENTID123',
        'name': 'InsuranceAgent',
        'version': 'DRAFT'
    },
    'invokeModelRawResponse': '<final_response>Based on your request, I searched our insurance benefit information database for details. The search results indicate that insurance policies may cover different types of benefits, depending on the policy and state laws. Specifically, the results discussed personal injury protection (PIP) coverage, which typically covers medical expenses for insured individuals injured in an accident (cited sources: 1234567-1234-1234-1234-123456789abc, 2345678-2345-2345-2345-23456789abcd). PIP may pay for costs like medical care, lost income replacement, childcare expenses, and funeral costs. Medical payments coverage was also mentioned as another option that similarly covers medical treatment costs for the policyholder and others injured in a vehicle accident involving the insured vehicle. The search results further noted that whether lost wages are covered depends on the state and coverage purchased. Please let me know if you need any clarification or have additional questions.</final_response>',
    'messageVersion': '1.0',
    'overrideType': 'OUTPUT_PARSER',
    'promptType': 'POST_PROCESSING'
}
```

Example response

```json
{
    'promptType': 'POST_PROCESSING',
    'postProcessingParsedResponse': {
```
'responseText': 'Based on your request, I searched our insurance benefit information database for details. The search results indicate that insurance policies may cover different types of benefits, depending on the policy and state laws. Specifically, the results discussed personal injury protection (PIP) coverage, which typically covers medical expenses for insured individuals injured in an accident (cited sources: 24c62d8c-3e39-4ca1-9470-a91d641fe050, 197815ef-8798-4cb1-8aa5-35f5d6b28365). PIP may pay for costs like medical care, lost income replacement, childcare expenses, and funeral costs. Medical payments coverage was also mentioned as another option that similarly covers medical treatment costs for the policyholder and others injured in a vehicle accident involving the insured vehicle. The search results further noted that whether lost wages are covered depends on the state and coverage purchased. Please let me know if you need any clarification or have additional questions.'


Deploy your 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 need to 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 a previously created version if necessary. You then configure your application to make API calls to that alias.

The version is 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 TestAlias.
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. In the Amazon Bedrock console, select **Agents** from the left navigation pane and choose your agent in the **Agents** section.
2. In the **Aliases** section, select **Create** at the top right corner.
3. Enter a unique name for the alias and provide an optional description.
4. 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.
5. Select **Create alias**. A green success banner appears at the top.

You can manage versions of your agents in the following ways.

**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 knowledge bases by choosing the name of the information you want to view.

**To delete a version**

1. In the **Versions** section, select the radio button next to the version you want to delete.
2. Select **Delete**.
3. A modal appears warning you about the consequences of deletion. Enter **delete** in the input box and select **Delete** to confirm.
4. A blue banner appears to inform you that the version is being deleted. When deletion is complete, a green success banner appears.

You can manage versions of your alias in the following ways.
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.

To delete an alias

1. Select the radio button next to the alias you want to delete.
2. Select **Delete**.
3. A modal appears warning you about the consequences of deletion. Enter **delete** in the input box and select **Delete** to confirm.
4. A blue banner appears to inform you that the alias is being deleted. When deletion is complete, a green success banner appears.

Use the API

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

- [Agents for Amazon Bedrock API Reference](#) – Includes API operations for setting up and configuring agents, knowledge bases, and their associated resources.
- [Agents for Amazon Bedrock Runtime API Reference](#) – Includes API operations for invoking agents and querying knowledge bases.

For information about endpoints and regional support, see [Service endpoints for Amazon Bedrock](#) in the AWS General Reference.

Agents for Amazon Bedrock build-time APIs
<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Endpoint</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>bedrock-agent.us-east-1.amazonaws.com</td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>bedrock-agent.us-west-2.amazonaws.com</td>
<td>HTTPS</td>
</tr>
</tbody>
</table>

**Agents for Amazon Bedrock runtime APIs**

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Endpoint</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>bedrock-agent-runtime.us-east-1.amazonaws.com</td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>bedrock-agent-runtime.us-west-2.amazonaws.com</td>
<td>HTTPS</td>
</tr>
</tbody>
</table>

1. To set up an agent, you call the following [Agents for Amazon Bedrock APIs](#).
   a. CreateAgent – Create and configure an agent and define what the agent should be able to do for customers.
   b. CreateActionGroup – Create action groups to define API calls that the agent should make to carry out the tasks it is defined to carry out.
   c. (Optional) AssociateKnowledgeBase – Associate knowledge bases with the agent so that it can query it in orchestration.

2. To test your agent, send an InvokeAgent request through a runtime endpoint (see [Agents for Amazon Bedrock RunTime APIs](#)).

3. Use update APIs (UpdateAgent and UpdateAgentActionGroup) to iterate on your agent.

4. To deploy your agent, send a CreateAgentAlias request to create a version and alias and to associate the alias to the version. Set up your application to make InvokeAgent requests to this alias.

**Topics**
Create and manage agents

Topics

• Create an agent
• Prepare an agent
• Edit an agent
• Get information about an agent
• List information about your agents
• Delete an agent

Create an agent

To create an agent, send a CreateAgent request and provide the name, description, instructions for what it should do, and the foundation model for it to orchestrate with.

• Specify the following fields for security purposes.
  • agentResourceRoleArn – The ARN of the role with permissions to access dependent resources of the agent.
  • (Optional) customerEncryptionKeyArn – The ARN of a KMS key to encrypt the agent's resources.
  • (Optional) idleSessionTTLinSeconds – Specify the number of seconds for which the agent should maintain session information for a given sessionId. After this time expires, the subsequent InvokeAgent request begins a new session.

• To override the default prompt behavior for agent orchestration and to use advanced prompts, include a promptOverrideConfiguration object. For more information, see Advanced prompts.
• If your agent fails to be created, the response returns a list of failureReasons alongside a list of recommendedActions for you to troubleshoot.

Prepare an agent

After you configure an agent and are ready to test it, send a PrepareAgent request and provide the ID of the agent.

Once you prepare the agent, a DRAFT version and TSTALIASID alias is created. Use this alias when making InvokeAgent requests in order to test your agent.

Edit an agent

To edit an agent, send an UpdateAgent request. Include both fields that you want to update as well as fields that you want to keep the same.

Get information about an agent

To get information about an agent, send a GetAgent request and specify the ID of the agent.

List information about your agents

To list information about your agents, send a ListAgents request.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a nextToken that you can send in another ListAgents request to see the next batch of results.

Delete an agent

To delete an agent, send a DeleteAgent request.

By default, skipResourceInUseCheck is false and deletion is stopped if the resource is in use. If you set it to true, the resource will be deleted even if the resource is in use.

Create and manage action groups

Topics

• Create an action group
• Edit an action group
Create an action group

To create an action group, send a `CreateAgentActionGroup` request.

- Specify the OpenAPI schema defining the APIs to be invoked by the action group in the `apiSchema` object. You can either provide the schema directly as a JSON payload or provide the Amazon S3 bucket containing the OpenAPI schema.
- (Optional) Specify the ARN of the Lambda function to carry out the business logic carried out upon invoking the action.
- Set the `actionGroupState` to `ENABLED` to allow the agent to invoke the action group.

Edit an action group

To edit an action group, send an `UpdateAgentActionGroup` request. Include both fields that you want to update as well as fields that you want to keep the same.

Set the `actionGroupState` to `ENABLED` to allow the agent to invoke the action group.

Get information about an action group

To get information about an action group, send a `GetAgentActionGroup` request and specify the ID of the action group and the ID and version of the agent that it belongs to.

List information about an agent's action groups

To list information about an agent's action groups, send a `ListAgentActionGroups` request and specify the ID and version of the agent.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a `nextToken` that you can send in another `ListAgentActionGroups` request to see the next batch of results.

Delete an action group

To delete an action group, send a `DeleteAgentActionGroup` request.
By default, skipResourceInUseCheck is false and deletion is stopped if the resource is in use. If you set it to true, the resource will be deleted even if the resource is in use.

Create and manage agent versions

Topics

- Create an agent version
- Get information about an agent version
- List information about an agent's versions
- Delete a version of an agent

Create an agent version

You can only create an agent version by sending a CreateAgentAlias request. Leave the routingConfiguration unspecified to create a new version.

Get information about an agent version

To get information about an agent version, send a GetAgentVersion request and specify the ID and version of the agent.

List information about an agent's versions

To list information about an agent's versions, send a ListAgentVersions request and specify the ID of the agent.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a nextToken that you can send in another ListAgentVersions request to see the next batch of results.

Delete a version of an agent

To delete a version of an agent, send a DeleteAgentVersion request.

By default, skipResourceInUseCheck is false and deletion is stopped if the resource is in use. If you set it to true, the resource will be deleted even if the resource is in use.

Create and manage agent aliases

Topics
• **Create an agent alias**

To create an alias for an agent, send a [CreateAgentAlias](#) request.

- To create a new version, leave the `routingConfiguration` unspecified.
- To associate the alias to an existing version, specify the `agentVersion` in the `routingConfiguration`.

• **Edit an agent alias**

To edit an agent alias, send an [UpdateAgentAlias](#) request. Include both fields that you want to update as well as fields that you want to keep the same.

• **Get information about an alias of an agent**

To get information about an alias of an agent, send a [GetAgentAlias](#) request and specify the ID of the alias of the agent.

• **List information about an agent's aliases**

To list information about an agent's aliases, send a [ListAgentAliases](#) request and specify the ID of the agent.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a `nextToken` that you can send in another ListAgentAliases request to see the next batch of results.

• **Delete an alias of an agent**

To delete an alias of an agent, send a [DeleteAgentAlias](#) request.

By default, `skipResourceInUseCheck` is false and deletion is stopped if the resource is in use. If you set it to `true`, the resource will be deleted even if the resource is in use.
Create and manage a knowledge base associated with an agent

Topics
- Associate a knowledge base with an agent
- Edit the configuration of a knowledge base associated with an agent
- Get information about a knowledge base associated with an agent
- List information about the knowledge bases associated with an agent
- Disassociate a knowledge base from an agent

Associate a knowledge base with an agent

To associate a knowledge base with an agent, send a `AssociateAgentKnowledgeBase` request.

To allow the agent to query the knowledge base, specify the `knowledgeBaseState` as enabled.

Edit the configuration of a knowledge base associated with an agent

To edit the configuration of a knowledge base associated with an agent, send an `UpdateAgentKnowledgeBase` request. You can change its description or change the `knowledgeBaseState`, which specifies whether it is available for querying by the agent.

Get information about a knowledge base associated with an agent

To get information about a knowledge base associated with an agent, send a `GetAgentKnowledgeBase` request and specify the ID of the alias of the agent.

List information about the knowledge bases associated with an agent

To list information about the knoweldge bases associated with an agent, send a `ListAgentKnowledgeBases` request and specify the ID and version of the agent.

You can set the maximum number of results to return in a response. If there are more results than the number you set, the response returns a `nextToken` that you can send in another `ListAgentKnowledgeBases` request to see the next batch of results.

Disassociate a knowledge base from an agent

To disassociate a knowledge base from an agent, send a `DisassociateAgentKnowledgeBase` request.
Invoke your agent

To interact with your agent, send an InvokeAgent request using a Amazon Bedrock Agent runtime endpoint (bedrock-agent-runtime.<region>.amazonaws.com). Use TSTALIASID as the agentAliasId to invoke the draft version of your agent.

- Specify the user input to provide the agent in the inputText field.
- Set enableTrace to true to return the trace in the response. By default, the trace is disabled. For more information, see Trace events.
- If you reuse a sessionId, you continue an existing session with the agent if the value you set for the idle session timeout hasn't been exceeded.
- Set endSession to true to end the session with the agent.
- In the sessionState object, you can set the following attributes.
  - Attributes for the prompt in the promptSessionAttributes object, which contains sessions attributes that go to the prompt.
  - Attributes for the session in the sessionAttributes object, which contains session attributes that are passed through attributes that are passed to the action group.

The response returns the following objects. If there is an error, the response returns an exception. For more information, see InvokeAgent Errors.

- chunk – The bytes field contains the response to the user as a base-64 encoded binary object. If a knowledge base was queried, an attribution object with a list of citations is returned.
  - The retrievedReferences object contains the exact text in the chunk relevant to the query alongside the S3 location of the data source the text belongs to.
  - The generatedResponsePart object contains the text generated by the model based on the information from the text in the retrievedReferences.
- trace – Use the trace to track the agent's reasoning process for carrying out action groups, querying knowledge bases, and its responses to customers. For information about the fields in the trace object, see Trace events. The trace is only returned if enableTrace was set as true in the request.
Custom models

You can customize an Amazon Bedrock model to improve its performance and create a better customer experience. You can customize a model for specific tasks (Fine-Tuning model) or to learn new domain knowledge (Continued Pre-training model).

For information about quotas for model customization, see Model customization quotas.

Note

You are charged for model training based on the number of tokens processed by the model (number of tokens in training data corpus x number of epochs) and model storage charged per month per model. For more information, see Amazon Bedrock pricing.

Topics

• Continued Pre-training model
• Fine-tuned model
• Model customization job
• Prepare the datasets
• Using the console
• Using the API
• Guidelines for model customization
• Troubleshooting
• Custom models reference

Continued Pre-training model

Note

The Continued Pre-training feature is in preview release for Amazon Bedrock and is subject to change.
Creating a Continued Pre-training model allows you to train a model with new unlabeled data. Use continued pre-training to teach a model new domain knowledge that’s not already present in the base models. You can train a model with private data, such as business documents, that are not publically available for training large language models. Additionally, you can continue to improve the model by retraining the model with more unlabeled data as it becomes available.

**Note**
Continued-Pretraining of a model is only supported by the Titan Text Express model.

**Fine-tuned model**

Creating a Fine-Tuned model allows you to improve a model's performance on specific tasks. By providing a training dataset of labeled examples related to a specific task, you help the model learn the task it's supposed to carry out.

**Model customization job**

To customize a model, you create a model customization job, specifically a *Fine-Tuning job* (creates a Fine-tuned model model) or *Continued Pre-training job* (creates a Continued Pre-training model model).

First, you upload a training dataset and, optionally, a validation dataset to Amazon S3 and provide the Amazon S3 bucket path to the model customization job. Follow the instructions at Prepare the datasets to set up your dataset. Each model has default hyperparameter values for model training. If necessary, control the training process by making changes to hyperparameter values. For more information, see Using the console. To customize a model through the API, see Using the API.

After you complete a model customization job, you can purchase Provisioned Throughput (see Provisioned Throughput) for the customized model so that you can use the model for inference using the InvokeModel or InvokeModelWithResponseStream API operations. For example code, see Run inference using Provisioned Throughput. You can also use the text playground (see Text playground).
Prepare the datasets

Before you upload your training and validation data to Amazon S3, you need to pre-process the format of your data into JSONL format training and validation datasets.

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

For information about quotas for the training and validation datasets, see [Model customization quotas](Model customization quotas).

Topics

- **Continued pre-training datasets**
- **Fine-tuning datasets**
- **Use your datasets**

Continued pre-training datasets

**Note**

The Continued Pre-training feature is in preview release for Amazon Bedrock and is subject to change.

For the training set, each line should only contain an `input` field. The format is as follows. You can specify up to 100,000 training data records.

```
{"input": "<input text>"}
{"input": "<input text>"}
{"input": "<input text>"}
```

The following is an example item that could be in the training data.

```
{"input": "AWS stands for Amazon Web Services"}
```

If you include an optional validation set, each line contains both an `input` field. The format is as follows.
Fine-tuning datasets

For both the training and optional validation datasets, create a file with multiple JSON lines. Each JSON line contains both an input and output field. The format is as follows. Use 6 characters per token as an approximation for the number of tokens.

{"prompt": "<prompt text>", "completion": "<expected generated text>"}
{"prompt": "<prompt text>", "completion": "<expected generated text>"}
{"prompt": "<prompt text>", "completion": "<expected generated text>"}

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

{"prompt": "prompt is AWS", "completion": "it's Amazon Web Services"}

If you include an optional validation set, each line contains both an input and output field. The format is as follows.

{"prompt": "<prompt text>", "completion": "<expected generated text>"}
{"prompt": "<prompt text>", "completion": "<expected generated text>"}
{"prompt": "<prompt text>", "completion": "<expected generated text>"}

The following is an example item that could be in the validation data.

{"prompt": "what does AWS stand for", "completion": "AWS stands for Amazon Web Services"}

Note
To support backwards compatibility, the following training and validation dataset format is supported for Amazon Titan models only.

{"input": "<prompt text>", "output": "<expected generated text>"}
Use your datasets

After you create your JSONL format training dataset and optional validation dataset, do the following:

1. **Upload** the datasets to an Amazon S3 bucket.

2. 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](#) to your Amazon Bedrock service role.

3. Create a Fine-tuning job or a Continued Pre-training job with the **console** or with the **API**.

Using the console

The following topics describe the major steps required to customize a model:

**Topics**

- [Customize a model](#)
- [Monitor the job](#)
- [Stop a job](#)
- [Analyze the job results](#)
- [Use a customized model for inference](#)

Customize a model

To train a fine-tuning model or a Continued Pre-training model with the Amazon Bedrock console, do the following.

For information about using the AWS SDK, see [Submit a job](#).

ℹ️ **Note**

Before you begin these steps, do the following:

- 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](#) to your Amazon Bedrock service role.
To customize a model

1. Sign in to the AWS Management Console and open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/home.

2. In the navigation pane, choose Foundation models and then Custom models.

3. Choose the Models tab.

4. Choose Customize model and then Fine-train model or Create Continued Pre-training job, depending on the type of model you want to train.

5. In the Create fine-tune job page (Create Continued Pre-training job, if you are training a Continued Pre-training model), configure the settings for the job:

   a. In the Model details section, choose the Source model that you want to customize with your own data and give your resulting model a name in Fine-tuned model name (or Continued pre-training model name).

   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.

   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 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.

   f. In the Input data section, provide the S3 link to the training dataset file and, optionally, the validation dataset file.
g. In the **Hyperparameters** section, input the values for the following hyperparameters to use in training. For information about the hyperparameters that you can use with a model, see [Custom models reference](#).

h. In the **Output data** section, enter the Amazon S3 location in which Amazon Bedrock should save the output of the job. Amazon Bedrock stores the training loss metrics and validation loss metrics for each epoch 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. For more information on setting up a custom role with the appropriate permissions, see [Permissions to carry out and manage a model customization job](#).
  
- **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](#).

6. Choose **Create Fine-tuning job** (or **Create Continued Pre-training job**) 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. In the navigation pane, choose **Foundation models** and then **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 an Amazon Bedrock model customization job while it's in progress. You can't resume a stopped job.

To stop a model customization job

1. Open the Amazon Bedrock console.
2. In the navigation pane, choose Foundation models and then 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

You can evaluate your model by running a model evaluation job. For more information, see Model evaluation.

After the model customization job completes, you can also find the following information in your Amazon 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.
The S3 output for a model customization 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:

<table>
<thead>
<tr>
<th>step_number</th>
<th>epoch_number</th>
<th>training_loss</th>
<th>perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.2</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.18</td>
<td>22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>18</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>step_number</th>
<th>epoch_number</th>
<th>validation_loss</th>
<th>perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.12</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.09</td>
<td>17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>15</td>
</tr>
</tbody>
</table>

**Use a customized 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 perform inference with your model and the AWS SDK, see Run inference using Provisioned Throughput.

To purchase Provisioned Throughput for a custom model.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. In the navigation pane, choose Foundation models and then Custom models.
3. Choose the radio button next to the model, select Actions, and choose Purchase Provisioned Throughput. For more information, see Provisioned Throughput.

To use a customized 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 customize your models using API operations. We provide examples with the AWS Command Line Interface and the AWS SDK for Python (Boto3).

Topics

- Submit a job
- Monitor a job
- Stop a job
- Analyze a job
- Retrieve information about your customized models

Submit a job

Note

The Continued Pre-training feature is in preview release for Amazon Bedrock and is subject to change.
Use the Amazon Bedrock [CreateModelCustomizationJob](https://docs.aws.amazon.com/bedrock/latest/userguide/createmodelcustomizationjob.html) 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**. To continually pre-train a model, use the value **CONTINUED_PRE_TRAINING**.
- **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 information about the hyperparameters that you can use with a model, see [Custom models reference](https://docs.aws.amazon.com/bedrock/latest/userguide/custom-models-reference.html).
- **jobName** – The name to give the training job.
- **roleArn** – The ARN of the service role. For more information on setting up a role with the appropriate permissions, see [Permissions to carry out and manage a model customization job](https://docs.aws.amazon.com/bedrock/latest/userguide/permissions-model-customization-job.html).
- **trainingDataConfig** – An object containing the URI of the Amazon S3 location of the training dataset. To create the training dataset, see [Prepare the datasets](https://docs.aws.amazon.com/bedrock/latest/userguide/prepare-datasets.html).
- **validationDataConfig** – An object containing the URI of the Amazon S3 location of the validation dataset. To create the validation dataset, see [Prepare the datasets](https://docs.aws.amazon.com/bedrock/latest/userguide/prepare-datasets.html).
- **outputDataConfig** – An object containing the URI of the Amazon S3 location to write the output data to.

The response returns a [jobArn](https://docs.aws.amazon.com/bedrock/latest/userguide/job-arn.html) 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:

```json
{
   "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

```
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.

```
--kms-key-id "arn:aws:kms:us-east-1:123456789012:alias/MyKeyId"
```
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:

```python
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](https://docs.aws.amazon.com/blogs/aws/monitoring-amazon-bedrock-model-customization-jobs) request. To monitor the status of a model customization job, send an Amazon Bedrock [GetModelCustomizationJob](https://docs.aws.amazon.com/blogs/aws/monitoring-amazon-bedrock-model-customization-jobs) 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:

```bash
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:

```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 `GetCustomModel` 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:

```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:

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

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

```bash
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:

```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 customized model in the AWS CLI:
Guidelines for model customization

The ideal parameters for customizing a model depend on the dataset and the task for which the model is intended. You should experiment with values to determine which parameters work best for your specific case. To help, evaluate your model by running a model evaluation job. For more information, see Model evaluation.

This topic provides guidelines and recommended values as a baseline for customization of the Titan Text Express model. For other models, check the provider's documentation.

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.

Titan Text Express

The following guidelines are for the Titan Text Express text-to-text model model. For information about the hyperparameters that you can set, see Amazon Titan.

Impact on other tasks types

In general, the larger the training dataset, the better the performance for a specific task. However, training for a specific task might make the model perform worse on different tasks, especially if you use a lot of examples. For example, if the training dataset for a summarization task contains 100,000 samples, the model might perform worse on a classification task.

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:

\[
\text{newLearningRate} = \frac{\text{oldLearningRate} \times \text{newBatchSize}}{\text{oldBatchSize}}
\]

**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.

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Minimum learning rate</th>
<th>Default learning rate</th>
<th>Max learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summarization</td>
<td>1.00E-06</td>
<td>3.00E-06</td>
<td>5.00E-05</td>
</tr>
<tr>
<td>Classification</td>
<td>5.00E-06</td>
<td>5.00E-05</td>
<td>5.00E-05</td>
</tr>
<tr>
<td>Question-answer</td>
<td>5.00E-06</td>
<td>5.00E-06</td>
<td>5.00E-05</td>
</tr>
</tbody>
</table>

**Learning warmup steps**

We recommend the default value of 0.
Troubleshooting

This section summarizes errors that you might encounter and what to check if you do.

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 encrypt 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.

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.
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:

   ```json
   {
       "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 Model customization quotas.

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 an error similar to the preceding example, make sure that the number of tokens conforms to the token quota in Model customization quotas.

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.

**Custom models reference**

The following reference content covers the dataset format and hyperparameters that each Amazon Bedrock model uses.

**Dataset formats**

When you fine tune a model, you provide a training dataset and an optional validation dataset. The datasets are in JSONL format. For information about creating and using datasets, see [Prepare the datasets](#).

**Hyperparameters**

A hyperparameter is a parameter that controls the training process, such as the learning rate or epoch count. You set hyperparameters for custom model training when you submit the fine tuning job with the Amazon Bedrock console or by calling the CreateModelCustomizationJob API operation. For guidelines on hyperparameter settings, see [Guidelines for model customization](#).

**Topics**

- [Cohere Command](#)
- [Cohere Command Light](#)
- [Meta Llama 2 13B](#)
- [Meta Llama 2 70B](#)
- [Amazon Titan](#)

**Cohere Command**

The following is the dataset format and training hyperparameters for the Cohere Command model. For more information, see the Cohere documentation at [https://docs.cohere.com/docs/fine-tuning](https://docs.cohere.com/docs/fine-tuning).

**Hyperparameters**

The following are the training hyperparameters for the Cohere Command model. For more information, see [Hyperparameters](#).
Topics

- **batchSize**
- **epochCount**
- **learningRate**
- **earlyStoppingPatience**
- **earlyStoppingThreshold**
- **evalPercentage**

**batchSize**

The number of data samples to use in each training iteration. The only acceptable value is 8.

- Type: Integer
- Minimum: 8
- Maximum: 8
- Default: 8
- Required: No

**epochCount**

The maximum number of complete passes through the dataset during training.

- Type: Integer
- Minimum: 1
- Maximum: 100
- Default: 1
- Required: No

**learningRate**

The rate at which the model parameters are updated during training. If you use a validation dataset to your train model, **We recommend that you don't provide a value for learningRate**.

- Type: Continuous
• Minimum: 0.000005
• Maximum: 0.1
• Default: 0.00001
• Required: No

**earlyStoppingPatience**

The tolerance for stagnation in the loss metric before stopping the training process.

• Type: Integer
• Minimum: 0
• Maximum: 10
• Default: 6
• Required: No

**earlyStoppingThreshold**

The minimum improvement in loss required to prevent premature termination of the training process.

• Type: Integer
• Minimum: 0.0
• Maximum: 0.1
• Default: 0.01
• Required: No

**evalPercentage**

The percentage of the dataset allocated for model evaluation, if you don't provide a separate validation dataset.

• Type: Continuous
• Minimum: 5
• Maximum: 50
• Default: 20
Cohere Command Light

The following is the dataset format and training hyperparameters for the Cohere Command Light model. For more information, see the Cohere documentation at https://docs.cohere.com/docs/fine-tuning.

Hyperparameters

The following are the training hyperparameters for the Cohere Command Light model. For more information, see Hyperparameters.

Topics

- **batchSize**
- **epochCount**
- **learningRate**
- **earlyStoppingPatience**
- **earlyStoppingThreshold**
- **evalPercentage**

**batchSize**

The number of data samples used in each training iteration. The value should be a multiple of 8.

- Type: Integer
- Minimum: 8
- Maximum: 32
- Default: 32
- Required: No

**epochCount**

The maximum number of complete passes through the dataset during training.

- Type: Integer
• Minimum: 1
• Maximum: 100
• Default: 1
• Required: No

**learningRate**

The rate at which the model parameters are updated during training. If you use a validation dataset to your train model, We recommend that you don't provide a value for learningRate.

• Type: Continuous
• Minimum: 0.000005
• Maximum: 0.1
• Default: 0.00001
• Required: No

**earlyStoppingPatience**

The tolerance for stagnation in the loss metric before stopping the training process.

• Type: Integer
• Minimum: 0
• Maximum: 10
• Default: 6
• Required: No

**earlyStoppingThreshold**

The minimum improvement in loss required to prevent premature termination of the training process.

• Type: Integer
• Minimum: 0.0
• Maximum: 0.1
• Default: 0.01
evalPercentage

The percentage of the dataset allocated for model evaluation, if a separate validation dataset is not provided.

- Type: Continuous
- Minimum: 5
- Maximum: 50
- Default: 20
- Required: No

Meta Llama 2 13B

The following is the dataset format and training hyperparameters for the Meta Llama 2 13B model.

Hyperparameters

The following are the training hyperparameters for the Meta Llama 2 13B model. For more information, see Hyperparameters.

Topics

- batchSize
- epochCount
- learningRate

batchSize

The number of data samples to use in each training iteration.

- Type: Integer
- Minimum: 1
- Maximum: 1
- Default: 1
• Required: No

**epochCount**

The maximum number of complete passes through the dataset during training.

• Type: Integer
• Minimum: 1
• Maximum: 10
• Default: 5
• Required: No

**learningRate**

The rate at which the model parameters are updated during training.

• Type: Continuous
• Minimum: 0.000005
• Maximum: 0.1
• Default: 1e-4
• Required: No

**Meta Llama 2 70B**

The following is the dataset format and training hyperparameters for the Meta Llama 2 70B model.

**Hyperparameters**

The following are the training hyperparameters for the Meta Llama 2 70B model. For more information, see [Hyperparameters](#).

**Topics**

• [batchSize](#)
• [epochCount](#)
• [learningRate](#)
**batchSize**

The number of data samples to use in each training iteration.

- Type: Integer
- Minimum: 1
- Maximum: 1
- Default: 1
- Required: No

**epochCount**

The maximum number of complete passes through the dataset during training.

- Type: Integer
- Minimum: 1
- Maximum: 10
- Default: 5
- Required: No

**learningRate**

The rate at which the model parameters are updated during training.

- Type: Continuous
- Minimum: 0.000005
- Maximum: 0.1
- Default: 1e-4
- Required: No

---

**Amazon Titan**

**Titan Text Express**

- **Model ID** – amazon.titan-text-express-v1
- **Max tokens** – 8K
- **Languages** – English (GA), 100 additional languages (Preview)
- **Supported use cases** – Retrieval augmented generation, open-ended text generation, brainstorming, summarization, code generation, table creation, data formatting, paraphrasing, chain of thought, rewrite, extraction, QnA, and chat.

**Titan Text Lite**

- **Model ID** – amazon.titan-text-lite-v1
- **Max tokens** – 4K
- **Languages** – English
- **Supported use cases** – Open-ended text generation, brainstorming, summarization, code generation, table creation, data formatting, paraphrasing, chain of thought, rewrite, extraction, QnA, and chat.

**Titan Multimodal Embeddings**

- **Model ID** – amazon.titan-embed-image-v1
- **Max input text tokens** – 128
- **Languages** – English
- **Max input image size** – 5 MB
- **Output vector size** – 1,024 (default), 384, 256
- **Inference types** – On-Demand, Provisioned Throughput
- **Supported use cases** – image search, recommendations, and personalization

**Fine Tuning**

- Input to the Titan Multimodal Embeddings fine tuning is image-text pairs. The supported image formats for fine-tuning include PNG and JPEG. The input image size limit is 5 MB.
- Upper limit: 4,096 px/Lower limit: 128 px image size
- Max number of tokens in caption: 128
- Training dataset size range: 1000 - 500,000
- Maximum total pixels per image: 2048*2048*3
Hyperparameters

- Learning rate - (min/max learning rate) – default: 5.00E-05, min: 5.00E-08, max: 1
- Batch size - Effective batch size – default: 576, min: 256, max: 9,216
- Max epochs – default: "auto", min: 1, max: 100

Titan Image Generator

- Model ID – amazon.titan-image-generator-v1
- Max input characters – 1,024
- Max input image size – 50 MB (only some specific resolutions are supported)
- Max total pixels per image – 2048*2048*3
- Languages – English
- Output type – image
- Inference types – On-Demand, Provisioned Throughput
- Supported use cases – image generation, image editing, image variations

Fine Tuning

Input to the Titan Image Generator model fine tuning is image-text pairs. The supported image formats for fine-tuning include PNG and JPEG. The input image size limit is 50 MB. It’s largest dimension cannot exceed 4,096 pixels and the total number of pixels cannot exceed 2,048 x 2,048 px x 3px.

Training samples: min: 5, max: 10,000

Prompts length: min: 3 char, max: 1,024 char

Hyperparameters

- Learning rate - (min/max learning rate) – default: 1.00E-05, min: 1.00E-07, max: 1
- Batch size - effective batch size – default: 64, min: 64, max: 192
- Training steps stepCount – total number of times the model is exposed to a batch of training data during the training process - default: "auto" (set based on the number of images), min: 500, max: 16,000
• Step size `stepSize` – size of the training step exposed to the model - default: "auto" (set based on the number of images), min: 500, max: 16,000

For information about the dataset format and hyperparameters for the Amazon Titan models, see Amazon Titan Models.
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.

<table>
<thead>
<tr>
<th>Pricing option</th>
<th>Base model</th>
<th>Custom model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provisioned Throughput, no commitment (hourly pricing)</td>
<td>Not available</td>
<td>Available (maximum 2 Provisioned Throughputs per account)</td>
</tr>
<tr>
<td>Provisioned Throughput, 1 month commitment</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>Provisioned Throughput, 6 month commitment</td>
<td>Available</td>
<td>Available</td>
</tr>
</tbody>
</table>

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.

Topics
- Provisioned Throughput model IDs
- Procedures
- Permissions
- Provisioned Throughput console procedures
- Using the Provisioned Throughput API

Provisioned Throughput model IDs

Provisioned throughput is currently available for the following models. Use the corresponding model ID when using the CreateProvisionedModelThroughput API.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Model ID for provisioned throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan Text G1 - Express 8K</td>
<td>amazon.titan-text-express-v1:0:8k</td>
</tr>
<tr>
<td>Titan Embeddings G1 - Text</td>
<td>amazon.titan-embed-text-v1:2:8k</td>
</tr>
<tr>
<td>Titan Multimodal Embeddings G1 - Text</td>
<td>amazon.titan-embed-image-v1:0</td>
</tr>
<tr>
<td>Titan Image Generator G1</td>
<td>amazon.titan-image-generator-v1:0</td>
</tr>
<tr>
<td>Anthropic Claude V2 18K</td>
<td>anthropic.claude-v2:0:18k</td>
</tr>
<tr>
<td>Anthropic Claude V2 100K</td>
<td>anthropic.claude-v2:0:100k</td>
</tr>
<tr>
<td>Anthropic Claude V2.1 18K</td>
<td>anthropic.claude-v2:1:18k</td>
</tr>
<tr>
<td>Anthropic Claude V2.1 100K</td>
<td>anthropic.claude-v2:1:100k</td>
</tr>
<tr>
<td>Model name</td>
<td>Model ID for provisioned throughput</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>Anthropic Claude Instant V1 100K</td>
<td>anthropic.claude-instant-v1:2:100K</td>
</tr>
<tr>
<td>Cohere Command</td>
<td>cohere.command-text-v14:7:4k</td>
</tr>
<tr>
<td>Cohere Command Light</td>
<td>cohere.command-light-text-v14:7:4k</td>
</tr>
<tr>
<td>Stable Diffusion XL 1.0</td>
<td>stability.stable-diffusion-xl-v1:0</td>
</tr>
<tr>
<td>Meta Llama 2 Chat 13B</td>
<td>meta.lama2-13b-chat-v1:0:4k</td>
</tr>
<tr>
<td>Meta Llama 2 13B</td>
<td>meta.lama2-13b-v1:0:4k</td>
</tr>
<tr>
<td>Meta Llama 2 70B</td>
<td>meta.lama2-70b-v1:0:4k</td>
</tr>
</tbody>
</table>

**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.

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.

Provisioned Throughput console procedures

This section describes the console procedures for Provisioned Throughput.

Topics

- View Provisioned Throughput summary
- Purchase Provisioned Throughput
- View details of a Provisioned Throughput
Edit a Provisioned Throughput
Delete a Provisioned Throughput

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.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Model ID for provisioned throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan Text G1 - Express 8K</td>
<td>amazon.titan-text-express-v1:0:8k</td>
</tr>
<tr>
<td>Titan Embeddings G1 - Text</td>
<td>amazon.titan-embed-text-v1:2:8k</td>
</tr>
</tbody>
</table>
### Model name | Model ID for provisioned throughput
--- | ---
Titan Multimodal Embeddings G1 - Text | amazon.titan-embed-image-v1:0
Titan Image Generator G1 | amazon.titan-image-generator-v1:0
Anthropic Claude V2 18K | anthropic.claude-v2:0:18k
Anthropic Claude V2 100K | anthropic.claude-v2:0:100k
Anthropic Claude V2.1 18K | anthropic.claude-v2:1:18k
Anthropic Claude V2.1 100K | anthropic.claude-v2:1:100k
Anthropic Claude Instant V1 100K | anthropic.claude-instant-v1:2:100K
Cohere Command | cohere.command-text-v14:7:4k
Cohere Command Light | cohere.command-light-text-v14:7:4k
Stable Diffusion XL 1.0 | stability.stable-diffusion-xl-v1:0
Meta Llama 2 Chat 13B | meta.llama2-13b-chat-v1:0:4k
Meta Llama 2 13B | meta.llama2-13b-v1:0:4k
Meta Llama 2 70B | meta.llama2-70b-v1:0:4k

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.
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.

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.

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.

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**
- **Run inference using Provisioned Throughput**
- **Update Provisioned Throughput**
- **Get Provisioned Throughput**
- **Delete Provisioned Throughput**
- **List Provisioned Throughput resources**

**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](#).

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.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Model ID for provisioned throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titan Text G1 - Express 8K</td>
<td>amazon.titan-text-express-v1:0:8k</td>
</tr>
<tr>
<td>Titan Embeddings G1 - Text</td>
<td>amazon.titan-embed-text-v1:2:8k</td>
</tr>
<tr>
<td>Titan Multimodal Embeddings G1 - Text</td>
<td>amazon.titan-embed-image-v1:0</td>
</tr>
<tr>
<td>Titan Image Generator G1</td>
<td>amazon.titan-image-generator-v1:0</td>
</tr>
<tr>
<td>Anthropic Claude V2 18K</td>
<td>anthropic.claude-v2:0:18k</td>
</tr>
<tr>
<td>Anthropic Claude V2 100K</td>
<td>anthropic.claude-v2:0:100k</td>
</tr>
<tr>
<td>Anthropic Claude V2.1 18K</td>
<td>anthropic.claude-v2:1:18k</td>
</tr>
<tr>
<td>Anthropic Claude V2.1 100K</td>
<td>anthropic.claude-v2:1:100k</td>
</tr>
<tr>
<td>Model name</td>
<td>Model ID for provisioned throughput</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Anthropic Claude Instant V1 100K</td>
<td>anthropic.claude-instant-v1:2:100K</td>
</tr>
<tr>
<td>Cohere Command</td>
<td>cohere.command-text-v14:7:4k</td>
</tr>
<tr>
<td>Cohere Command Light</td>
<td>cohere.command-light-text-v14:7:4k</td>
</tr>
<tr>
<td>Stable Diffusion XL 1.0</td>
<td>stability.stable-diffusion-xl-v1:0</td>
</tr>
<tr>
<td>Meta Llama 2 Chat 13B</td>
<td>meta.llama2-13b-chat-v1:0:4k</td>
</tr>
<tr>
<td>Meta Llama 2 13B</td>
<td>meta.llama2-13b-v1:0:4k</td>
</tr>
<tr>
<td>Meta Llama 2 70B</td>
<td>meta.llama2-70b-v1:0:4k</td>
</tr>
</tbody>
</table>

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:

```bash
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:

```python
import boto3
bedrock = boto3.client(service_name='bedrock')
bedrock.create_provisioned_model_throughput(
    modelUnits=1,
    commitmentDuration='SixMonths',
    provisionedModelName='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.

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 \n  --body "{"prompt": "\n
Human: story of two dogs\n
Assistant:"}" 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
Human: explain black holes to 8th graders\n
Assistant:"})
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())
```
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:

```bash
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:

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

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

Get Provisioned Throughput

Use the [GetProvisionedModelThroughput](#) operation to retrieve information about the specified Provisioned Throughput.
The following code examples demonstrates how to retrieve information using the AWS CLI and Python.

**AWS CLI**

Retrieval 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(
    provisionedModelName='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(
    provisionedModelName='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()
```
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
- Batch inference jobs (API only)
- Agents
- Agent aliases
- Knowledge bases
- Model evaluations (Console only)

If you use the API, you must call the correct TagResource and UntagResource operations.

- The following resources use the Amazon Bedrock TagResource and UntagResource operations.
  - Custom models
  - Model customization jobs
  - Provisioned models
  - Batch inference jobs
- The following resources use the Agents for Amazon Bedrock TagResource and UntagResource operations.
  - Agents
  - Agent aliases
  - Knowledge bases

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
• Use APIs
• Tag restrictions

Use the console

You can add and modify tags at any time while creating or editing a supported resource.

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.

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.
Amazon Titan Models

Amazon Titan foundation models (FMs) are a family of FMs pretrained by AWS on large datasets, making them powerful, general-purpose models built to support a variety of use cases. Use them as-is or privately customize them with your own data.

Amazon Titan supports the following models for Amazon Bedrock.

- **Titan Text**
- **Titan Text Embeddings**
- **Titan Multimodal Embeddings**
- **Titan Image Generator (preview)**

**Note**
Amazon Titan Image Generator G1 is in public preview release.

**Topics**
- [Amazon Titan Text models](#)
- [Amazon Titan Text Embeddings models](#)
- [Amazon Titan Multimodal Embeddings model](#)
- [Amazon Titan Image Generator model](#)

**Amazon Titan Text models**

Amazon Titan text models include Titan Text Express and Titan Text Lite.

**Titan Text Express** – Amazon Titan Text Express is a large language model for text generation. It is useful for a wide range of advanced, general language tasks such as open-ended text generation and conversational chat, as well as support within Retrieval Augmented Generation (RAG). At launch, the model is optimized for English, with multilingual support for more than 100 additional languages available in preview.

- **Model ID** – amazon.titan-text-express-v1
• **Max tokens** – 8K
• **Languages** – English (GA), 100 additional languages (Preview)
• **Supported use cases** – Retrieval augmented generation, open-ended text generation, brainstorming, summarizations, code generation, table creation, data formatting, paraphrasing, chain of thought, rewrite, extraction, QnA, and chat.

**Titan Text Lite** – Amazon Titan Text Lite is a light weight efficient model, ideal for fine-tuning of English-language tasks, including like summarizations and copy writing, where customers want a smaller, more cost-effective model that is also highly customizable.

• **Model ID** – amazon.titan-text-lite-v1
• **Max tokens** – 4K
• **Languages** – English
• **Supported use cases** – Open-ended text generation, brainstorming, summarizations, code generation, table creation, data formatting, paraphrasing, chain of thought, rewrite, extraction, QnA, and chat.

**Titan Text Model Customization:**

For more information on fine tuning Titan text models, see [Custom Models](#).

**Amazon Titan Text Prompt Engineering Guidelines:**

Titan text models can be used in a wide variety of applications for different use cases. Titan Text models have prompt engineering guidelines for the following applications including:

• Chatbot
• Text2SQL
• Function Calling
• RAG (Retrieval Augmented Generation)

For more information on Amazon Titan Text prompt engineering guidelines, see [Amazon Titan Text Prompt Engineering Guidelines](#).

For general prompt engineering guidelines, see [Prompt Engineering Guidelines](#).
AWS AI Service Card - Amazon Titan Text

AI Service Cards provide transparency and document the intended use cases and fairness considerations for our AWS AI services. AI Service Cards provide a single place to find information on the intended use cases, responsible AI design choices, best practices, and performance for a set of AI service use cases.

Amazon Titan Text Embeddings models

Amazon Titan Embeddings text models include Titan Text Embeddings G1.

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.

The Titan Embeddings G1 - Text model (amazon.titan-embed-text-v1). The Titan Embeddings G1 – Text v1.2 can intake up to 8k tokens and outputs a vector of 1,536 dimensions. The model also works in 25+ different languages. The model is optimized for text retrieval tasks but can also perform additional tasks such as semantic similarity and clustering. Titan Embeddings G1 – Text v1.2 also supports long documents, however, for retrieval tasks it is recommended to segment documents into logical segments (such as paragraphs or sections). In line with our recommendation.

To use the text or image embeddings models, use the Invoke Model API operation with amazon.titan-embed-text-v1 or amazon.titan-embed-image-v1 as the model Id and retrieve the embedding object in the response.

To see Jupyter notebook examples:

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 (depending on which model you chose), select View example notebook to see example notebooks for embeddings.

For more information on preparing your dataset for multimodal training, see Preparing your dataset.
Amazon Titan Multimodal Embeddings model

Titan Multimodal Embeddings Generation 1 (G1) is a multimodal embeddings model for use cases like searching images by text, image, or a combination of text and image. Designed for high accuracy and fast responses, this model is an ideal choice for search and recommendations use cases.

- **Model ID** – amazon.titan-embed-image-v1
- **Max input text tokens** – 128
- **Languages** – English
- **Max input image size** – 5 MB
- **Output vector size** – 1,024 (default), 384, 256
- **Inference types** – On-Demand, Provisioned Throughput
- **Supported use cases** – image search, recommendations, and personalization

**Finetuning**

- Input to the Titan Multimodal Embeddings finetuning is image-text pairs.
- Image formats: PNG, JPG, JPEG
- Input image size limit: 5 MB
- Image dimensions: min: 128 px, max: 4,096 px
- Max number of tokens in caption: 128
- Training dataset size range: 1000 - 500,000
- Validation dataset size range: 8 - 50,000
- Caption length in characters: 0 - 2,560
- Maximum total pixels per image: 2048*2048*3
- Aspect ratio (w/h): min: 0.25, max: 4

**Preparing datasets**

For the training dataset, create a .jsonl file with multiple JSON lines. Each JSON line contains both an image-ref and caption attributes similar to Sagemaker Augmented Manifest format. A validation dataset is required. Auto-captioning is not currently supported.
For both the training and validation datasets, you will create .jsonl files with multiple JSON lines.

The Amazon S3 paths need to be in the same folders where you have provided permissions for Amazon Bedrock to access the data by attaching an IAM policy to your Amazon Bedrock service role. For more information on granting an IAM policies for training data, see [Grant custom jobs access to your training data](#).

**Embedding length**

Setting a custom embedding length is optional. The embedding default length is 1024 characters. The embedding length can be set to 256, 384, or 1024 characters.

```python
# EmbeddingConfig Shape
{
    'outputEmbeddingLength': int  // Optional, One of: [256, 384, 1024], default: 1024
}

# Updated API Payload Example
body = json.dumps({
    "inputText": "hi",
    "inputImage": image_string,
    "embeddingConfig": {
        "outputEmbeddingLength": 256
    }
})
```

**Hyperparameters**

- Learning rate - (min/max learning rate) – default: 5.00E-05, min: 5.00E-08, max: 1
- Batch size - Effective batch size – default: 576, min: 256, max: 9,216
- Max epochs – default: "auto", min: 1, max: 100
Amazon Titan Image Generator model

Titan Image Generator G1 is an image generation model. It generates images from text, and allows users to upload and edit an existing image. Users can edit an image with a text prompt (without a mask) or parts of an image with an image mask. You can extend the boundaries of an image with outpainting, and fill in an image with inpainting. It can also generate variations of an image based on an optional text prompt. Titan Image Generator includes watermarking on the output files.

For more information on Titan Image Generator prompt engineering guidelines, see Amazon Titan Image Generator Prompt Engineering Best Practices.

- **Model ID** – amazon.titan-image-generator-v1
- **Max input characters** – 1,024 char
- **Max input image size** – 50 MB (only some specific resolutions are supported)
- **Max image size using in/outpainting** – 1,024 x 1,024 px
- **Max image size using image variation** – 4,096 x 4,096 px
- **Languages** – English
- **Output type** – image
- **Supported image types** – JPEG, JPG, PNG
- **Inference types** – On-Demand, Provisioned Throughput
- **Supported use cases** – image generation, image editing, image variations

**Features**

- **Text-to-image (T2I) generation** – Input a text prompt and generate a new image as output. The generated image captures the concepts described by the text prompt.
- **Finetuning of a T2I model** – Import several images to capture your own style and personalization and then fine tune the core T2I model. The fine-tuned model generates images that follow the style and personalization of a specific user.
- **Image editing options** – includes inpainting, outpainting, generating variations, and automatic editing without an image mask.
- **Inpainting** – Uses an image and a segmentation mask as input (either from the user or estimated by the model) and reconstructs the region within the mask. Use inpainting to remove masked elements and replace them with background pixels.
• Outpainting – Uses an image and a segmentation mask as input (either from the user or estimated by the model) and generates new pixels that seamlessly extend the region. Use precise outpainting to preserve the pixels of the masked image when extending the image to the boundaries. Use default outpainting to extend the pixels of the masked image to the image boundaries based on segmentation settings.

• Image variation – Uses an image and an optional prompt as input. It generates a new image that preserves the content of the input image, but variates its style and background.

**Fine Tuning**

• Input to the Titan Image Generator model finetuning is image-text pairs.

• Supported image formats: JPEG, JPG, PNG

• Input image size limit: 50 MB

• Max image dimension: 4,096 px

• Maximum total pixels per image: 2,048*2,048*3

• Training samples: min: 5, max: 10,000

• Prompts length: min: 3 char, max: 1,024 char

**Parameters**

• Mask prompt – Enter a prompt to select the segmentation of the mask image. A more precise mask prompt results in an improved image mask.

• Negative prompt – Items or concepts that you don't want the model to generate, such as cartoons or violence.

• Reference image – The image on which to generate the response or that you want the model to edit. Size limit is 1280 x 1280 pixels.

• Response image – Output settings for the generated image, such as quality, orientation, size, and the number of images to generate.

• Advanced configurations – The inference parameters to pass to the model including prompt strength and seed values. Use the same values to reproduce a variation.

**Hyperparameters**

• Learning rate - (min/max learning rate) – default: 1.00E-05, min: 1.00E-07, max: 1
• Batch size - Effective batch size – default: 64, min: 64, max: 192
• Training steps – stepCount total number of times the model is exposed to a batch of training data during the training process - default: "auto" (set based on the number of images), min: 500, max: 16,000

Training dataset for finetuning

For the training dataset, create a .jsonl file with multiple JSON lines. Each JSON line contains both an image-ref and caption attributes similar to Sagemaker Augmented Manifest format. Auto-captioning is not currently supported.

```json
{"image-ref": "s3://bucket-1/folder1/0001.png", "caption": "some text"}
{"image-ref": "s3://bucket-1/folder2/0002.png", "caption": "some text"}
{"image-ref": "s3://bucket-1/folder1/0003.png", "caption": "some text"}
```

The Amazon S3 paths need to be in the same folders where you have provided permissions for Amazon Bedrock to access the data by attaching an IAM policy to your Amazon Bedrock service role. For more information on granting an IAM policies for training data, see Grant custom jobs access to your training data.

For the training dataset, you will create .jsonl files with multiple JSON lines. Titan Image Generator doesn’t support the inclusion of a validation dataset.

For more information on preparing a custom model, see Preparing a custom model.

Output

This model adds an invisible watermark to all generated images to reduce the spread of misinformation. In the future, a watermark detection API will be available to help you confirm whether an image was generated using the Titan model, by checking for the existence of this watermark.

Prompt Engineering Guidelines

Mask prompt – This algorithm classifies pixels into concepts. The user can give a text prompt that will be used to classify the areas of the image to mask, based on the interpretation of the mask prompt. The prompt option can interpret more complex prompts, and encode the mask into the segmentation algorithm.
**Image mask** – You can also use an image mask to set the mask values. The image mask can be combined with prompt input for the mask to improve accuracy. The image mask file must conform to the following parameters:

- Mask image values must be 0 (black) or 255 (white) for the mask image. The image mask area with the value of 0 will be regenerated with the image from the user prompt and/or input image.
- The `maskImage` field must be a base64 encoded image string.
- Mask image must have the same dimensions as the input image (same height and width).
- Only PNG or JPG files can be used for the input image and the mask image.
- Mask image must only use black and white pixels values.
- Mask image can only use the RGB channels (alpha channel not supported).

For more information on Titan Image Generator prompt engineering, see [Amazon Titan Image Generator Prompt Engineering Best Practices](#).

For general prompt engineering guidelines, see [Prompt Engineering Guidelines](#).
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 Amazon Bedrock. The following topics show you how to configure Amazon 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 Amazon Bedrock resources.

**Topics**

- Data protection in Amazon Bedrock
- Identity and access management for Amazon Bedrock
- Compliance validation for Amazon Bedrock
- Incident response in Amazon Bedrock
- Resilience in Amazon Bedrock
- Infrastructure security in Amazon Bedrock
- Cross-service confused deputy prevention
- Configuration and vulnerability analysis in Amazon Bedrock
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. You are also responsible for 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 Amazon 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 doesn't store or log your data in its service logs.

Bedrock uses your training data with the `CreateModelCustomizationJob` action, or with the console, to create a custom model which is a fine tuned version of an Amazon Bedrock foundational model. Your custom models are managed and stored by AWS. By default, custom models are encrypted with AWS Key Management Service keys that AWS owns, but you can use your own AWS KMS keys to encrypt your custom models. You encrypt a custom model when you submit a fine tuning job with the console or programmatically with the `CreateModelCustomizationJob` action.

None of the training or validation data you provide for fine tuning is stored in Amazon Bedrock accounts, once the fine tuning job completes. During training, your data exists in AWS Service Management Connector 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.

We don't recommend using confidential data to train a custom model as the model might generate inference responses based on that confidential data. If you use confidential data to train a custom model, the only way to prevent responses based on that data is to delete the custom model, remove the confidential data from your training dataset, and retrain the custom model.

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
- Protect jobs using a VPC

**Data encryption**

Amazon Bedrock uses encryption to protect data at rest and data in transit.
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 at rest.

Key management

Use the AWS Key Management Service to manage the keys that you use to encrypt your resources. For more information, see AWS Key Management Service concepts. You can encrypt the following resources with a KMS key.

- Through Amazon Bedrock
  - Model customization jobs and their output custom models – During job creation in the console or by specifying the customModelKmsKeyId field in the CreateModelCustomizationJob API call.
  - Agents – During agent creation in the console or by specifying the field in the CreateAgent API call.
  - Data source ingestion jobs for knowledge bases – During knowledge base creation in the console or by specifying the kmsKeyArn field in the CreateDataSource or UpdateDataSource API call.
Vector stores in Amazon OpenSearch Service – During vector store creation. For more information, see Creating, listing, and deleting Amazon OpenSearch Service collections and Encryption of data at rest for Amazon OpenSearch Service.

Through Amazon S3 – For more information, see Using server-side encryption with AWS KMS keys (SSE-KMS).

Training, validation, and output data for model customization

Data sources for knowledge bases

Through AWS Secrets Manager – For more information, see Secret encryption and decryption in AWS Secrets Manager

Vector stores for third-party models

After you encrypt a resource, you can find the ARN of the KMS key by selecting a resource and viewing its Details in the console or by using the following Get API calls.

- GetModelCustomizationJob
- GetAgent
- GetIngestionJob

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
- Model customization outputs
- Create a customer managed key
- Use a customer managed key to run customization jobs
- Use a customer managed key during inference
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.

```json
{
    "Version": "2012-10-17",
    "Id": "KMS Key Policy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::<your-account-id>:root"]
            },
            "Resource": ["arn:aws:kms:<region>:<your-account-id>:key/<your-key-id>"],
            "Condition": {
                "StringEquals": {
                    "kms:KeyId": ["<your-key-id>""]
                }
            }
        }
    ]
}
```
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.

```json
{
    "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:account-id: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:account-id: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*

Encryption of agent resources

Amazon Bedrock encrypts your agent's session information. By default, Amazon Bedrock encrypts this data using an AWS managed key. Optionally, you can encrypt the agent artifacts using a customer managed key.

For more information about AWS KMS keys, see [Customer managed keys](https://docs.aws.amazon.com/kms/latest/developerguide/index.html) in the AWS Key Management Service Developer Guide.

If you encrypt sessions with your agent with a custom KMS key, you must set up the following identity-based policy and resource-based policy to allow Amazon Bedrock to encrypt and decrypt agent resources on your behalf.
1. Attach the following identity-based policy to an IAM role or user with permissions to make InvokeAgent calls. This policy validates the user making an InvokeAgent call has KMS permissions. Replace the `region`, `account-id`, `agent-id`, and `key-id` with the appropriate values.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Amazon Bedrock to encrypt and decrypt Agent resources on behalf of authorized users",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:region:account-id:key/key-id",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:bedrock:arn":
                    "arn:aws:bedrock:region:account-id:agent/agent-id"
                }
            }
        }
    ]
}
```

2. Attach the following resource-based policy to your KMS key. Change the scope of the permissions as necessary. Replace the `region`, `account-id`, `agent-id`, and `key-id` with the appropriate values.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allows account root to modify the KMS key, not used by Amazon Bedrock. Change it as per your permission requirements.",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::account-id:root"
            },
            "Action": "kms:*",
```
Encryption of knowledge base resources

Amazon Bedrock encrypts resources related to your knowledge bases. By default, Amazon Bedrock encrypts this data using an AWS managed key. Optionally, you can encrypt the model artifacts using a customer managed key.

Encryption with a KMS key can occur with the following processes:

- Transient data storage while ingesting your data sources
- Passing information to OpenSearch Service if you let Amazon Bedrock set up your vector database
- Querying a knowledge base

The following resources used by your knowledge bases can be encrypted with a KMS key. If you encrypt them, you need to add permissions to decrypt the KMS key.
• Data sources stored in an Amazon S3 bucket
• Third-party vector stores

For more information about AWS KMS keys, see Customer managed keys in the AWS Key Management Service Developer Guide.

Topics
• Encryption of transient data storage during data ingestion
• Encryption of information passed to Amazon OpenSearch Service
• Encryption of knowledge base retrieval
• Permissions to decrypt your AWS KMS key for your data sources in Amazon S3
• Permissions to decrypt the secret for your vector store in Pinecone or Redis Enterprise Cloud

Encryption of transient data storage during data ingestion

When you set up a data ingestion job for your knowledge base, you can encrypt the job with a custom KMS key.

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 Amazon Bedrock service role. Replace the region, account-id, and key-id with the appropriate values.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
            ]
        }
    ]
}
```
Encryption of information passed to Amazon OpenSearch Service

If you opt to let Amazon Bedrock create a vector store in Amazon OpenSearch Service for your knowledge base, Amazon Bedrock can pass a KMS key that you choose to Amazon OpenSearch Service for encryption. To learn more about encryption in Amazon OpenSearch Service, see Encryption in Amazon OpenSearch Service.

Encryption of knowledge base retrieval

You can encrypt sessions in which you generate responses from querying a knowledge base with a KMS key. To do so, include the ARN of a KMS key in the kmsKeyArn field when making a RetrieveAndGenerate request. Attach the following policy, replacing the values appropriately to allow Amazon Bedrock to encrypt the session context.

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

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

You store the data sources for your knowledge base 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.

If you encrypted your data sources in Amazon S3 with a custom AWS KMS key, attach the following policy to your Amazon Bedrock service role to allow Amazon 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.

```json
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

**Permissions to decrypt the secret for your vector store in Pinecone or Redis Enterprise Cloud**

If you set up a vector store in Pinecone or Redis Enterprise Cloud, you configure a secret in AWS Secrets Manager to securely access it. You can encrypt this secret by following the steps at [Secret encryption and decryption in AWS Secrets Manager](#).

If your secret is encrypted with a custom AWS KMS key, attach the following policy to your Amazon 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
            ]
        }
    ]
}
```
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
- Configure your model customization job to use VPC

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
- Create an Amazon S3 VPC Endpoint
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).

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.

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. Replace the account-id with the account ID of the user to whom you are providing S3 bucket access permissions.

```json
{
    "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"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "account-id"
                }
            }
        },
        {
            "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"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "account-id"
                }
            }
        }
    ]
}
```
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.

```json
{
    "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",
}
For more information about configuring the data access role for model customization jobs, see [Permissions to carry out and manage a model customization job](#).

### 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 VPC 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:

```json
"VpcConfig": {
  "SecurityGroupIds": [
    "sg-0123456789abcdef0"
  ]
}
```
For more information about configuring jobs using the API, see Submit a job (API).

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).

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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                }
            }
        }
    ]
}
```
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 Amazon Bedrock resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon Bedrock works with IAM
- Identity-based policy examples for Amazon Bedrock
- AWS managed policies for Amazon Bedrock
- Allow users to perform actions on agent and alias resources
- Service roles
- Amazon Bedrock product IDs
- Troubleshooting Amazon Bedrock identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Bedrock.

Service user – If you use the Amazon Bedrock service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon 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 Amazon Bedrock, see Troubleshooting Amazon Bedrock identity and access.

Service administrator – If you’re in charge of Amazon Bedrock resources at your company, you probably have full access to Amazon Bedrock. It’s your job to determine which Amazon 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 Amazon Bedrock, see How Amazon Bedrock works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Bedrock. To view example Amazon Bedrock identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Bedrock.

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.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an
action that then initiates another action in a different service. FAS uses the permissions of
the principal calling an AWS service, combined with the requesting AWS service to make
requests to downstream services. FAS requests are only made when a service receives a request
that requires interactions with other AWS services or resources to complete. 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. For policy details when making FAS requests, see
Forward access sessions.

- **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 Amazon Bedrock, learn what IAM features are available to use with Amazon Bedrock.

IAM features you can use with Amazon Bedrock

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>Amazon Bedrock support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource-based policies</td>
<td>No</td>
</tr>
<tr>
<td>Policy actions</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy resources</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy condition keys</td>
<td>Yes</td>
</tr>
<tr>
<td>ACLs</td>
<td>No</td>
</tr>
<tr>
<td>ABAC (tags in policies)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>Principal permissions</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles</td>
<td>No</td>
</tr>
</tbody>
</table>

To get a high-level view of how Amazon 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 Amazon 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 Amazon Bedrock

To view examples of Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

Resource-based policies within Amazon 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 Amazon 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 Amazon Bedrock actions, see Actions Defined by Amazon Bedrock in the Service Authorization Reference.

Policy actions in Amazon 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 Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.
Policy resources for Amazon 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 Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

Policy condition keys for Amazon 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 Amazon Bedrock identity-based policies, see Identity-based policy examples for Amazon Bedrock.

**ACLs in Amazon 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 Amazon Bedrock**

| Supports ABAC (tags in policies) | Yes |

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 Amazon Bedrock

<table>
<thead>
<tr>
<th>Supports temporary credentials</th>
<th>Yes</th>
</tr>
</thead>
</table>

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 Amazon Bedrock

<table>
<thead>
<tr>
<th>Supports forward access sessions (FAS)</th>
<th>Yes</th>
</tr>
</thead>
</table>

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. 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. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon 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 Amazon Bedrock functionality. Edit service roles only when Amazon Bedrock provides guidance to do so.

Service-linked roles for Amazon 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 Amazon 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.

Identity-based policy examples
For details about actions and resource types defined by Amazon 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
- Use the Amazon Bedrock console
- Allow users to view their own permissions
- Allow access to third-party model subscriptions
- Deny access for inference on specific models
- Grant custom jobs access to your training data
- Permissions for using KMS keys with model customization

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/idManagingPolicies-Conditions.html) 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/idAccessAnalyzer.html) 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](https://docs.aws.amazon.com/IAM/latest/UserGuide/idAccessAnalyzer.html) in the *IAM User Guide*.


### Use the Amazon 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 Amazon 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.

```json
{
   "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: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"
"bedrock:PutFoundationModelEntitlement",
"bedrock:GetModelInvocationLoggingConfiguration",
"bedrock:PutModelInvocationLoggingConfiguration",
"bedrock:DeleteAgent",
"bedrock:UpdateAgent",
"bedrock:GetAgent",
"bedrock:ListAgents"
]
}
For more information, see [Adding permissions to a user](https://docs.aws.amazon.com/IAM/latest/UserGuide/index.html) 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.

```json
{
    "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/${aws:username}"
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:PassRole"
            ],
            "Resource": ["arn:aws:iam::*:group/${aws:username}"}
    ]
}
```
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.

```json
{
    "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-a3e2-9c7f4f66a85",
                        "cc0bdd50-279a-40d8-829c-4009b77a1fcc",
                        "d0123e8d-50d6-4dba-8a26-3fed4899f388",
                        "a61c46fe-1747-41aa-9af0-2e0ae8a9ce05",
                        "c504a4f3-fbde-48da-850c-1d60d131ff5a",
                        "216b69fd-07d5-4c7b-866b-936456d68311",
                        "b7568428-a1ab-46d8-bab3-37def50f6f6a",
                        "38e55671-c3fe-4a44-9783-3584906e7cad",
                        "prod-ariujvyzvd2qy"
                    ]
                }
            }
        }
    ]
}
```
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.

```json
{
  "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.

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

AWS managed policies for Amazon Bedrock

To add permissions to users, groups, and roles, it's easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the IAM User Guide.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when

```
"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"
],
"Condition": {
  "StringEquals": {
    "aws:ResourceAccount": "account-no"
  }
}
```
a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won’t break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

**AWS managed policy: AmazonBedrockFullAccess**

You can attach the AmazonBedrockFullAccess policy to your IAM identities.

This policy grants administrative permissions that allow the user permission to create, read, update, and delete Amazon Bedrock resources.

Note

Fine-tuning and model access require extra permissions. See [Allow access to third-party model subscriptions](#) and [Grant custom jobs access to your training data](#) for more information.

Permissions details

This policy includes the following permissions:

- **ec2** (Amazon Elastic Compute Cloud) – Allows permissions to describe VPCs, subnets, and security groups.
- **iam** (AWS Identity and Access Management) – Allows principals to pass roles, but only allows IAM roles with "Amazon Bedrock" in them to be passed to the Amazon Bedrock service. The permissions are restricted to bedrock.amazonaws.com for Amazon Bedrock operations.
- **kms** (AWS Key Management Service) – Allows principals to describe AWS KMS keys and aliases.
- **bedrock** (Amazon Bedrock) – Allows principals read and write access to all actions in the Amazon Bedrock control plane and runtime service.

```json
{
   "Version": "2012-10-17",
   "Statement": [
```
AWS managed policy: AmazonBedrockReadOnly

You can attach the AmazonBedrockReadOnly policy to your IAM identities.

This policy grants read-only permissions that allow users to view all resources in Amazon Bedrock.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonBedrockReadOnly",
            "Effect": "Allow",
            "Action": [
                "bedrock:GetFoundationModel",
                "bedrock:ListFoundationModels",
                "bedrock:GetModelInvocationLoggingConfiguration",
                "bedrock:GetProvisionedModelThroughput",
                "bedrock:ListProvisionedModelThroughputs",
                "bedrock:GetModelCustomizationJob",
                "bedrock:ListModelCustomizationJobs",
                "bedrock:ListCustomModels",
                "bedrock:GetCustomModel",
                "bedrock:ListTagsForResource",
                "bedrock:GetFoundationModelAvailability"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Bedrock updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Bedrock since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Document history for the Amazon Bedrock User Guide.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonBedrockFullAccess</td>
<td>New policy</td>
<td>December 12, 2023</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>ns to create, read, update, and delete resources.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AmazonBedrockReadOnly</strong> – New policy</td>
<td>Amazon Bedrock added a new policy to give users read-only permissions for all actions.</td>
<td>December 12, 2023</td>
</tr>
<tr>
<td>Amazon Bedrock started tracking changes</td>
<td>Amazon Bedrock started tracking changes for its AWS managed policies.</td>
<td>December 12, 2023</td>
</tr>
</tbody>
</table>

### 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, Amazon Bedrock 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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateAgent</td>
<td>N/A</td>
</tr>
<tr>
<td>UpdateAgent</td>
<td>Agent</td>
</tr>
<tr>
<td>GetAgent</td>
<td>Agent</td>
</tr>
<tr>
<td>ListAgents</td>
<td>N/A</td>
</tr>
<tr>
<td>Operation</td>
<td>Resource</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>CreateActionGroup</td>
<td>Agent</td>
</tr>
<tr>
<td>UpdateActionGroup</td>
<td>Agent</td>
</tr>
<tr>
<td>GetActionGroup</td>
<td>Agent</td>
</tr>
<tr>
<td>ListActionGroups</td>
<td>Agent</td>
</tr>
<tr>
<td>CreateAgentDraftSnapshot</td>
<td>Agent</td>
</tr>
<tr>
<td>GetAgentVersion</td>
<td>Agent</td>
</tr>
<tr>
<td>ListAgentVersions</td>
<td>Agent</td>
</tr>
<tr>
<td>CreateAgentAlias</td>
<td>Agent</td>
</tr>
<tr>
<td>UpdateAgentAlias</td>
<td>Alias</td>
</tr>
<tr>
<td>GetAgentAlias</td>
<td>Alias</td>
</tr>
<tr>
<td>ListAgentAliases</td>
<td>Agent</td>
</tr>
<tr>
<td>InvokeAgent</td>
<td>Alias</td>
</tr>
</tbody>
</table>

The following is a sample policy that you can attach to an IAM role to allow it to call Amazon Bedrock 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.

```json
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Sid": "sid",
       "Effect": "Allow",
       "Action": "bedrock:GetAgent",
     }
   ]
}
```

Allow users to perform actions on agent and alias resources
Service roles

You can set up custom IAM service roles that let Amazon Bedrock carry out tasks on your behalf and customize the permissions for the role for your specific use case. For examples of policies that you can add to these roles, see Identity-based policy examples for Amazon Bedrock. In the IAM permissions section during a setup process, you can select this role instead of letting Amazon Bedrock create one for you.

To set up the service role, you carry out the processes below through either the console or the API by following the steps at Creating a role to delegate permissions to an AWS service.

1. Create the role
2. Attach a trust relationship
3. Attach relevant permissions

You can use Amazon Bedrock service roles to carry out the following tasks.

Topics

- Permissions to carry out and manage a model customization job
- Permissions to create and manage an agent
- Permissions to create and manage a knowledge base
Permissions to carry out and manage a model customization job

To use a custom role for model customization instead of the one Amazon Bedrock automatically creates, create an IAM role and attach the following permissions by following the steps at Creating a role to delegate permissions to an AWS service.

- Trust relationship
- Permissions to access your training and validation data in S3
- Permissions to write your output data to S3
- (Optional) If you encrypt any of the following resources with a KMS key, permissions to decrypt the key (see Encryption of model customization jobs)
  - A model customization job or the resulting custom model
  - The training, validation, or output data for the model customization job

Topics

- Trust relationship
- Permissions to access training and validation files in S3
- Permissions to write output files to S3

Trust relationship

The following policy allows Amazon Bedrock to assume this role and carry out the model customization job. 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. Use the `ArnEquals` or `ArnLike` condition to restrict the scope to specific model customization jobs.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
```
Permissions to access training and validation files in S3

Attach the following policy to allow the role to access to the Amazon S3 locations containing your training and validation data, replacing the values in the Resource list with actual Amazon S3 ARNs:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "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/*"
         ]
      }
   ]
}
```

Permissions to write output files to S3

Attach the following policy to allow the role to write to the Amazon S3 locations for your output data, replacing the values in the Resource list with actual Amazon S3 ARNs:

```json
"StringEquals": {
   "aws:SourceAccount": "account-id"
},
"ArnEquals": {
}
]"
Permissions to create and manage an agent

To use a custom role for agents instead of the one Amazon Bedrock automatically creates, create an IAM role with the prefix `AmazonBedrockExecutionRoleForAgents_` and attach the following permissions by following the steps at [Creating a role to delegate permissions to an AWS service](#).

- Trust relationship
- Access to the Amazon Bedrock base models
- Access to the Amazon S3 objects containing the OpenAPI schemas for the action groups in your agents
- (Optional) Permissions for Amazon Bedrock to query knowledge bases that you want to attach to your agents
- (Optional) If you encrypt your agent with a KMS key, permissions to decrypt the key (see [Encryption of agent resources](#))

Whether you use a custom role or not, you also need to provide permissions for Amazon Bedrock to access the AWS Lambda functions for the action groups in your agents through attaching a resource-based policy to your Lambda functions. For more details, see [Permissions to access your action group Lambda functions](#).
Topics

- **Trust relationship**
- **Permissions to access Amazon Bedrock models**
- **Permissions to access your action group API schemas in Amazon S3**
- **(Optional) Permissions to access your knowledge bases**
- **Permissions to access your action group Lambda functions**

**Trust relationship**

The following policy allows Amazon Bedrock to assume this role and create and manage agents. 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. 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "account-id"
            },
            "ArnLike": {
            }
        }
    }
}
```
Permissions to access Amazon Bedrock models

Attach the following policy to provide permissions for the role to use Amazon Bedrock models for orchestration.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["bedrock:InvokeModel"],
    "Resource": [
      "arn:aws:bedrock:region::foundation-model/anthropic.claude-v2",
      "arn:aws:bedrock:region::foundation-model/anthropic.claude-instant-v1"
    ]
  }]
}
```

Permissions to access your action group API schemas in Amazon S3

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

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["s3:GetObject"],
    "Resource": [
      "arn:aws:s3:::bucket/path/to/schema"
    ],
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "account-id"
      }
    }
  }]
}
```
(Optional) Permissions to access your knowledge bases

Attach the following policy to provide permissions for the role to access knowledge bases that you have set up. Replace `account-id` and `knowledge-base-id` with the appropriate values.

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

Permissions to access your action group Lambda functions

Attach the following resource-based policy to a Lambda function to provide access to the Lambda functions for your agent's action groups. Use the `Condition` field to specify the ARN of the user account and of the agent. For more information, see Using resource-based policies for Lambda.

```json
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "statement-name",
        "Effect": "Allow",
        "Principal": {
            "Service": "bedrock.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
    }
}
```
Permissions to create and manage a knowledge base

To use a custom role for knowledge base instead of the one Amazon Bedrock automatically creates, create an IAM role with the prefix `AmazonBedrockExecutionRoleForKnowledgeBase_` and attach the following permissions by following the steps at [Creating a role to delegate permissions to an AWS service](#).

- Trust relationship
- Access to the Amazon Bedrock base models
- Access to the Amazon S3 objects containing 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
- (Optional) If you encrypt any of the following resources with a KMS key, permissions to decrypt the key (see [Encryption of knowledge base resources](#)).
  - Your knowledge base
  - Data sources for your knowledge base
  - Your vector database in Amazon OpenSearch Service
  - The secret for your third-party vector database in AWS Secrets Manager
  - A data ingestion job

Topics

- **Trust relationship**
- **Permissions to access Amazon Bedrock models**
• Permissions to access your data sources in Amazon S3
• (Optional) Permissions to access your vector database in Amazon OpenSearch Service
• (Optional) Permissions to access your vector database in Pinecone or Redis Enterprise Cloud
• (Optional) Permissions for AWS to manage a AWS KMS key for transient data storage during data ingestion

Trust relationship

The following policy allows Amazon Bedrock to assume this role and create and manage knowledge bases. 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. 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "bedrock.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "account-id"
      },
      "ArnLike": {
      }
    }
  }]
}
```
Permissions to access Amazon Bedrock models

Attach the following policy to provide permissions for the role to use Amazon Bedrock models to embed your source data. Currently, only the Titan Embeddings G1 - Text Model `amazon.titan-embed-text-v1` is supported.

```json
{
    "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/amazon.titan-embed-text-v1"
            ]
        }
    ]
}
```

Permissions to access your data sources in Amazon S3

Attach the following policy to provide permissions for the role to access the Amazon S3 URIs containing the data source files for your knowledge base. In the `Resource` field, provide an Amazon S3 object containing the data sources or add the URI of each data source to the list.

If you encrypted these data sources with a AWS KMS key, attach permissions to decrypt the key to the role by following the steps at Permissions to decrypt your AWS KMS key for your data sources in Amazon S3.

```json
{
    "Version": "2012-10-17",
    "Statement": [
```
(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 Amazon 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "aoss:APIAccessAll"
        ],
        "Resource": [
        ]
    }]
}
```

(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 Amazon 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.

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

If you encrypted your secret with a AWS KMS key, attach permissions to decrypt the key to the role by following the steps at [Permissions to decrypt the secret for your vector store in Pinecone or Redis Enterprise Cloud](#).

**(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 Amazon Bedrock service role. Replace the `region`, `account-id`, and `key-id` with the appropriate values.

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

The product IDs used by Amazon Bedrock are listed in the table below.

<table>
<thead>
<tr>
<th>Model</th>
<th>Product ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jurassic-2 Mid</td>
<td>1d288c71-65f9-489a-a3e2-9c7f4f6e6a85</td>
</tr>
<tr>
<td>Jurassic-2 Ultra</td>
<td>cc0bdd50-279a-40d8-829c-4009b77a1fcc</td>
</tr>
<tr>
<td>Claude 12k</td>
<td>c468b48a-84df-43a4-8c46-8870630108a7</td>
</tr>
<tr>
<td>Claude - 100K tokens</td>
<td>99d90be8-b43e-49b7-91e4-752f3866c8c7</td>
</tr>
<tr>
<td>Claude Instant - 12K tokens</td>
<td>b0eb9475-3a2c-43d1-94d3-56756fd43737</td>
</tr>
<tr>
<td>Claude Instant - 100K tokens</td>
<td>c504a4f3-fbde-48da-850c-1d60d131ff5a</td>
</tr>
<tr>
<td>Stability</td>
<td>d0123e8d-50d6-4dba-8a26-3fed4899f388</td>
</tr>
<tr>
<td>Cohere - Command</td>
<td>a61c46fe-1747-41aa-9af0-2e0ae8a9ce05</td>
</tr>
<tr>
<td>Cohere - Command Light</td>
<td>216b69fd-07d5-4c7b-866b-936456d68311</td>
</tr>
<tr>
<td>Cohere - Embed (English)</td>
<td>b7568428-a1ab-46d8-bab3-37def50f6f6a</td>
</tr>
<tr>
<td>Cohere - Embed (multi-lingual)</td>
<td>38e55671-c3fe-4a44-9783-3584906e7cad</td>
</tr>
<tr>
<td>Meta - Llama 2 13B chat</td>
<td>prod-ariujvzyzvd2qy</td>
</tr>
<tr>
<td>Meta - Llama 2 70B chat</td>
<td>prod-2c2yc2s3guhqy</td>
</tr>
<tr>
<td>Stability - SDXL 1.0</td>
<td>prod-2lvuzn4iy6n6o</td>
</tr>
</tbody>
</table>

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 Amazon Bedrock and IAM.
I am not authorized to perform an action in Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon Bedrock supports these features, see [How Amazon Bedrock works with IAM](#).
- 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](https://aws.amazon.com/hipaa/services/).

• **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.

• **AWS Customer Compliance Guides** – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

• **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](https://docs.aws.amazon.com/config/latest/developerguide/securityhub-use.html).

• **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](https://status.aws.amazon.com/).

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](https://docs.aws.amazon.com/health/latest/health-intro.html).
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 Amazon 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                }
            }
        }
    ]
}
```
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 Amazon Bedrock

You can monitor Amazon Bedrock with Amazon CloudWatch and with Amazon EventBridge.

Topics

- Monitor Amazon Bedrock with Amazon CloudWatch
- Monitor Amazon Bedrock events in Amazon EventBridge
- Log Amazon Bedrock API calls using AWS CloudTrail

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
- Logging CloudWatch metrics
- Use CloudWatch metrics for Amazon Bedrock
- View Amazon Bedrock metrics

Runtime metrics

The following table describes runtime metrics provided by Amazon Bedrock.

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocations</td>
<td>SampleCount</td>
<td>Number of requests to the <code>InvokeModel</code> or <code>InvokeModelWithResponseStream</code> API operations.</td>
</tr>
<tr>
<td>Metric name</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InvocationLatency</td>
<td>MilliSeconds</td>
<td>Latency of the invocations.</td>
</tr>
<tr>
<td>InvocationClientErrors</td>
<td>SampleCount</td>
<td>Number of invocations that result in client-side errors.</td>
</tr>
<tr>
<td>InvocationServerErrors</td>
<td>SampleCount</td>
<td>Number of invocations that result in AWS server-side errors.</td>
</tr>
<tr>
<td>InvocationThrottles</td>
<td>SampleCount</td>
<td>Number of invocations that the system throttled.</td>
</tr>
<tr>
<td>InputTokenCount</td>
<td>SampleCount</td>
<td>Number of tokens of text input.</td>
</tr>
<tr>
<td>LegacyModelInvocations</td>
<td>SampleCount</td>
<td>Number of invocations using Legacy models</td>
</tr>
<tr>
<td>OutputTokenCount</td>
<td>SampleCount</td>
<td>Number of tokens of text output.</td>
</tr>
<tr>
<td>OutputImageCount</td>
<td>SampleCount</td>
<td>Number of output images.</td>
</tr>
</tbody>
</table>

### Logging CloudWatch metrics

For each delivery success or failure attempt, the following Amazon CloudWatch metrics are emitted under the namespace AWS/Bedrock, andAcross 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](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/auth-access-control.html) in the *Amazon CloudWatch User Guide*.

**View Amazon Bedrock metrics**

View Amazon Bedrock metrics in the CloudWatch console.

**To view metrics (CloudWatch console)**

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.

• Amazon 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](#)
- [EventBridge schema](#)
- [Rules and targets](#)
- [Create a rule to handle Amazon Bedrock events](#)

### 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 following is example event JSON for a failed model customization job.

```json
{
  "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": [
  ],
  "detail": {
    "version": "0.0",
    "jobName": "abcd-wxyz",
    "outputModelName": "dummy-output-model-name",
    "outputModelArn": "arn:aws:bedrock:us-east-1:123412341234:dummy-output-model-name",
  }
}
```
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.
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 Amazon Bedrock events.

Create a rule to handle Amazon Bedrock events

Complete the following procedures in order to receive email notifications about your Amazon Bedrock events.

Create an Amazon Simple Notification Service topic

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 Amazon Bedrock events.

To create a rule to handle Amazon 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 Amazon Bedrock. CloudTrail captures all API calls for Amazon Bedrock as events. The calls captured include calls from the Amazon Bedrock console and code calls to the Amazon Bedrock API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon 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 Amazon 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.

Amazon Bedrock information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon 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 Amazon 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

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.

Amazon Bedrock data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, reading or writing to an Amazon S3 object). These are also known as data plane operations. Data events are often high-volume activities that CloudTrail doesn’t log by default.
Amazon Bedrock doesn't log Amazon Bedrock Runtime API operations (InvokeModel and InvokeModelWithResponseStream).

Amazon Bedrock logs all Agents for Amazon Bedrock Runtime API operations actions to CloudTrail as data events.

- To log InvokeAgent calls, configure advanced event selectors to record data events for the AWS::Bedrock::AgentAlias resource type.
- To log Retrieve and RetrieveAndGenerate calls, configure advanced event selectors to record data events for the AWS::Bedrock::KnowledgeBase resource type.

From the CloudTrail console, choose Bedrock agent alias or Bedrock knowledge base for the Data event type. You can additionally filter on the eventName and resources.ARN fields by choosing a custom log selector template. For more information, see Logging data events with the AWS Management Console.

From the AWS CLI, set the resource.type value equal to AWS::Bedrock::AgentAlias or AWS::Bedrock::KnowledgeBase and set the eventCategory equal to Data. For more information, see Logging data events with the AWS CLI.

The following example shows how to configure a trail to log all Amazon Bedrock data events for all Amazon Bedrock resource types in the AWS CLI.

```
aws cloudtrail put-event-selectors --trail-name trailName \
--advanced-event-selectors \n'[
  {
    "Name": "Log all data events on an Agents for Amazon Bedrock agent alias",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::Bedrock::AgentAlias"] }
    ]
  },
  {
    "Name": "Log all data events on an Agents for Amazon Bedrock knowledge base",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::Bedrock::KnowledgeBase"] }
    ]
  }
]'
```
You can additionally filter on the eventName and resources.ARN fields. For more information about these fields, see AdvancedFieldSelector.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

**Amazon Bedrock management events in CloudTrail**

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. CloudTrail logs management event API operations by default.

Amazon Bedrock logs the remainder of Amazon Bedrock API operations as management events. For a list of the Amazon Bedrock API operations that Amazon Bedrock logs to CloudTrail, see the following pages in the Amazon Bedrock API reference.

All Amazon Bedrock API operations and Agents for Amazon Bedrock API operations are logged by CloudTrail and documented in the Amazon Bedrock API Reference. For example, calls to the InvokeModel, StopModelCustomizationJob, and CreateAgent actions generate entries in the CloudTrail log files.

**Understanding Amazon 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.

```json
{
    "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 potential violations of AWS's Acceptable Use Policy (AUP) and Service Terms, including the 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:

- **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 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.

Once our automated abuse detection mechanisms identify potential violations, we may request information about customers’ use of Amazon Bedrock and compliance with our terms of service or a third-party provider’s AUP. In the event that a customer is unwilling or unable to comply with these terms or policies, AWS may suspend access to Amazon Bedrock.

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 Amazon 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 Amazon 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 Amazon Bedrock.

For more information, see Access AWS services through AWS PrivateLink in the AWS PrivateLink Guide.

Considerations for Amazon Bedrock VPC endpoints

Before you set up an interface endpoint for Amazon Bedrock, review Considerations in the AWS PrivateLink Guide.

Amazon Bedrock supports making the following API calls through VPC endpoints.

<table>
<thead>
<tr>
<th>Category</th>
<th>Endpoint prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon Bedrock Control Plane API actions</strong></td>
<td><code>bedrock</code></td>
</tr>
<tr>
<td><strong>Amazon Bedrock Runtime API actions</strong></td>
<td><code>bedrock-runtime</code></td>
</tr>
<tr>
<td><strong>Agents for Amazon Bedrock Runtime API actions</strong></td>
<td><code>bedrock-agent-runtime</code></td>
</tr>
</tbody>
</table>

Availability Zones

- Amazon 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. Replace `endpoint-prefix` with the endpoint prefix from the table above.
• Agents for Amazon Bedrock endpoints are available in all Availability Zones.

Create an interface endpoint for Amazon Bedrock

You can create an interface endpoint for Amazon 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 Amazon Bedrock using any of the following service names:

• com.amazonaws.REGION.bedrock
• com.amazonaws.REGION.bedrock-runtime
• com.amazonaws.REGION.bedrock-agent-runtime

If you enable private DNS for the interface endpoint, you can make API requests to Amazon Bedrock using its default Regional DNS name. For example:

• bedrock.us-east-1.amazonaws.com
• bedrock-runtime.us-east-1.amazonaws.com
• bedrock-agent-runtime.us-east-1.amazonaws.com

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 Amazon Bedrock through the interface endpoint. To control the access allowed to Amazon 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 Amazon 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 Amazon Bedrock actions for all principals on all resources.

```json
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream"
      ],
      "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 within your AWS account. 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 Amazon Bedrock.

Runtime quotas

The following quotas apply when you carry out model inference. These quotas consider the combined sum for InvokeModel and InvokeModelWithResponseStream requests.

<table>
<thead>
<tr>
<th>Model</th>
<th>Requests processed per minute</th>
<th>Tokens processed per minute</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI21 Labs Jurassic-2 Mid</td>
<td>400</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>AI21 Labs Jurassic-2 Ultra</td>
<td>100</td>
<td>200,000</td>
<td>No</td>
</tr>
<tr>
<td>Amazon Titan Embeddings G1 - Text</td>
<td>2,000</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Amazon Titan Image Generator G1</td>
<td>60</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Amazon Titan Multimodal Embeddings G1</td>
<td>2,000</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Model</td>
<td>Requests processed per minute</td>
<td>Tokens processed per minute</td>
<td>Adjustable</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------</td>
<td>----------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Amazon Titan Text G1 - Express</td>
<td>400</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Amazon Titan Text G1 - Lite</td>
<td>800</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Anthropic Claude Instant</td>
<td>400</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Anthropic Claude</td>
<td>100</td>
<td>200,000</td>
<td>No</td>
</tr>
<tr>
<td>Cohere Command</td>
<td>400</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Cohere Command Light</td>
<td>800</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Cohere Embed (English)</td>
<td>2,000</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Cohere Embed (Multilingual)</td>
<td>2,000</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Meta Llama2 Chat 13B</td>
<td>800</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Meta Llama2 Chat 70B</td>
<td>400</td>
<td>300,000</td>
<td>No</td>
</tr>
<tr>
<td>Stability.ai Diffusion XL 0.8</td>
<td>60</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Stability.ai Diffusion XL 1.0</td>
<td>60</td>
<td>N/A</td>
<td>No</td>
</tr>
</tbody>
</table>
Batch inference quotas

The following quotas apply when you run batch inference. The quotas depend on the modality of the input and output data.

<table>
<thead>
<tr>
<th>Modality</th>
<th>Minimum file size</th>
<th>Maximum file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text to embeddings</td>
<td>75 MB</td>
<td>500 MB</td>
</tr>
<tr>
<td>Text to text</td>
<td>20 MB</td>
<td>150 MB</td>
</tr>
<tr>
<td>Text/image to image</td>
<td>1 MB</td>
<td>50 MB</td>
</tr>
</tbody>
</table>

Agent and Knowledge base quotas

The following quotas apply to Agents for Amazon Bedrock and Knowledge base for Amazon Bedrock.

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents per account</td>
<td>50</td>
<td>Yes</td>
</tr>
<tr>
<td>Aliases per Agent</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Characters in Agent instructions</td>
<td>1,200</td>
<td>No</td>
</tr>
<tr>
<td>Action groups per Agent</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>The maximum number of APIs that you can add to an Agent</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>Associated knowledge bases per Agent</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>Data source file size</td>
<td>50 MB</td>
<td>No</td>
</tr>
</tbody>
</table>
# Model customization quotas

The following quotas apply to model customization.

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of input and output tokens when batch size is 2</td>
<td>4,096</td>
<td>No</td>
</tr>
<tr>
<td>Sum of input and output tokens when batch size is between 3 and 6</td>
<td>2,048</td>
<td>No</td>
</tr>
<tr>
<td>Character quota per data point</td>
<td>Token quota x 6</td>
<td>No</td>
</tr>
<tr>
<td>Scheduled training jobs per account</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom models per account</td>
<td>100</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum (continued pre-training)</th>
<th>Maximum (fine-tuning)</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training records in a dataset</td>
<td>100,000</td>
<td>10,000</td>
<td>Yes</td>
</tr>
<tr>
<td>Validation records in a dataset</td>
<td>1,000</td>
<td>1,000</td>
<td>Yes</td>
</tr>
<tr>
<td>Training dataset file size</td>
<td>10 GB</td>
<td>1 GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Validation dataset file size</td>
<td>100 MB</td>
<td>100 MB</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Provisioned Throughput quotas

The following quotas apply to Provisioned Throughput.

<table>
<thead>
<tr>
<th>Description</th>
<th>Default</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model units for a base model</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Model units for a custom model</td>
<td>2</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Document history for the Amazon Bedrock User Guide

- **Latest documentation update:** December 12, 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.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New managed policies added to the Security chapter</strong></td>
<td>Amazon Bedrock has added AmazonBedrockFullAccess to give users permission to create, read, update, and delete resources, and AmazonBedrockReadOnly to give users read-only permissions for all actions.</td>
<td>December 12, 2023</td>
</tr>
<tr>
<td><strong>New feature</strong></td>
<td>You can now monitor and customize your model versions.</td>
<td>November 29, 2023</td>
</tr>
<tr>
<td><strong>New Amazon Titan models</strong></td>
<td>New models from Amazon Titan include Titan Image Generator and Titan Multimodal Embeddings. For more information, see <a href="#">Titan Models</a>.</td>
<td>November 29, 2023</td>
</tr>
<tr>
<td><strong>Model evaluation job support added</strong></td>
<td>Amazon Bedrock now supports creating model evaluation jobs using automatic metrics or human workers.</td>
<td>November 29, 2023</td>
</tr>
<tr>
<td><strong>New feature</strong></td>
<td>With Continued Pre-training you can teach a model new</td>
<td>November 28, 2023</td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>New feature</td>
<td>You can now query knowledge bases through the Retrieve and RetrieveAndGenerate APIs. For more information, see <a href="#">Query a knowledge base</a>.</td>
<td>November 28, 2023</td>
</tr>
<tr>
<td>General release</td>
<td>General release of the Knowledge base for Amazon Bedrock service. For more information, see <a href="#">Knowledge base for Amazon Bedrock</a>.</td>
<td>November 28, 2023</td>
</tr>
<tr>
<td>General release</td>
<td>General release of the Agents for Amazon Bedrock service. For more information, see <a href="#">Agents for Amazon Bedrock</a>.</td>
<td>November 28, 2023</td>
</tr>
<tr>
<td>Customize more models</td>
<td>You can now customize models from Cohere and Meta. For more information, see <a href="#">Custom Models</a>.</td>
<td>November 28, 2023</td>
</tr>
<tr>
<td>New model releases</td>
<td>Updated documentation to cover new Meta and Cohere models. For more information, see <a href="#">Amazon Bedrock</a>.</td>
<td>November 13, 2023</td>
</tr>
<tr>
<td>Documentation localization</td>
<td>Amazon Bedrock documentation is now available in <a href="#">Japanese</a> and <a href="#">German</a>.</td>
<td>October 20, 2023</td>
</tr>
<tr>
<td>Region expansion</td>
<td>Amazon Bedrock is now available in Europe (Frankfurt) (eu-central-1). For information on endpoints, see <a href="#">Amazon Bedrock endpoints and quotas</a>.</td>
<td>October 19, 2023</td>
</tr>
<tr>
<td>Region expansion</td>
<td>Amazon Bedrock is now available in Asia Pacific (Tokyo) (ap-northeast-1). For information on endpoints, see <a href="#">Amazon Bedrock endpoints and quotas</a>.</td>
<td>October 3, 2023</td>
</tr>
<tr>
<td>Gated general release</td>
<td>Gated general release of the Amazon Bedrock service. For more information, see <a href="#">Amazon Bedrock</a>.</td>
<td>September 28, 2023</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*. 