



用户指南

Amazon Bedrock



Amazon Bedrock: 用户指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 Amazon Bedrock ?	1
Amazon Bedrock 的功能	1
Amazon Bedrock 的定价	2
支持的 AWS 区域	2
关键定义	4
基本概念	4
高级功能	5
设置	6
注册获取 AWS 账户	6
创建管理用户	7
授权以编程方式访问	8
控制台访问	9
模型访问权限	9
添加模型访问权限	10
删除模型访问权限	11
控制模型访问权限	12
API 设置	14
添加模型访问权限	14
Amazon Bedrock 端点	14
设置 AWS CLI	15
AWS 软件开发工具包设置	15
使用 SageMaker 笔记本电脑	17
使用 AWS 软件开发工具包	19
基础模型信息	21
使用基础模型	23
获取模型信息	24
按 AWS 地区划分的模型支持	26
按功能划分的型号支持	27
模型生命周期	30
按需、预调配吞吐量和模型自定义	31
旧版本	31
亚马逊 Bedrock 型号	32
基本型号 ID (按需)	33
基本型号 ID (用于预配置吞吐量)	35

模型推理参数	37
亚马逊Titan模型	37
AnthropicClaude模型	75
AI21 LabsJurassic-2模型	94
Cohere模型	98
MetaLlama 2模型	109
Mistral AI模型	113
Stability.ai Diffusion 模型	118
自定义模型超参数	135
Amazon Titan 文字模型	135
Amazon Titan Image Generator G1	136
Amazon Titan Multimodal Embeddings G1	137
CohereCommand模型	137
MetaLlama 2模型	139
控制台概览	140
开始使用	140
根基模型	141
操场	141
保障措施	142
编排	142
评估和部署	142
模型访问权限	142
模型调用日志记录	142
运行模型推理	143
推理参数	144
随机性和多样性	144
Length	146
操场	146
聊天操场	147
文本操场	147
图像操场	148
使用操场	148
运行单个提示推理	150
调用模型代码示例	151
通过流式传输调用模型代码示例	152
运行批量推理	153

权限	154
设置数据	156
创建批量推理作业	157
停止批量推理作业	160
获取批量推理作业的相关详细信息	160
列出批量推理作业	162
代码示例	164
提示工程准则	169
简介	169
其他提示资源	170
什么是提示？	170
提示的组成部分	170
少样本与零样本提示	172
提示模板	174
关于通过 API 调用使用 Amazon Bedrock 的重要注意事项	175
什么是提示工程？	175
Amazon Bedrock LLM 用户一般准则	176
设计您的提示	176
使用推理参数	177
详细准则	178
在 Amazon Bedrock 上优化文本模型提示（若基础知识不够有效）	183
Amazon Bedrock 文本模型的提示模板和示例	187
文本分类	187
问答（无上下文）	190
问答（带上下文）	193
总结	197
文本生成	199
代码生成	202
数学	204
推理/逻辑思维	205
模型评估	208
开始使用	209
自动模型评估	210
基于人工的模型评估作业	211
模型评估任务	215
一般文本生成	215

文本摘要	217
问答	218
文本分类	219
输入提示数据集	220
内置提示数据集	221
自定义提示数据集	224
工作人员说明	227
评级方法	228
管理工作团队	234
模型评估作业结果	235
自动报告	235
人工报告卡	237
Amazon S3 输出	244
所需的权限	250
控制台权限要求	250
服务角色	253
CORS 权限要求	261
数据加密	261
Amazon Bedrock 知识库	263
工作方式	264
支持的区域和模型	265
先决条件	266
设置数据源	266
设置向量索引	269
创建知识库	276
为您的知识库设置安全配置	280
同步您的数据源	284
测试知识库	286
查询知识库	286
查询配置	290
管理数据源	307
查看有关数据源的信息	307
删除数据源	308
删除数据源	310
管理知识库	310
查看有关知识库的信息	310

更新知识库	311
删除知识库	312
部署知识库	313
Agents for Amazon Bedrock	315
工作方式	316
构建时配置	316
运行时进程	318
支持的区域和模型	319
先决条件	320
创建代理	321
创建操作组	323
定义OpenAPI架构	324
定义 Lambda 函数	332
添加操作组	336
关联知识库	338
测试代理	339
跟踪事件	343
管理代理	350
查看有关代理的信息	350
编辑代理	352
删除代理	354
管理操作组	354
管理代理与知识库的关联	358
自定义代理	361
高级提示	361
控制会话上下文	398
部署代理	400
管理版本	402
管理别名	403
自定义模型	407
支持的区域和模型	408
先决条件	409
准备数据集	410
(可选) 设置 VPC	411
提交作业	417
管理作业	419

监控作业	419
停止任务	420
分析工作结果	421
使用自定义模型	423
代码示例	424
指南	434
Amazon Titan Text G1 - Express	435
故障排除	436
权限问题	436
数据问题	437
内部错误	438
预调配吞吐量	439
支持的区域和模型	440
先决条件	442
购买预配置吞吐量	442
管理预配置吞吐量	445
查看有关预配置吞吐量的信息	445
编辑预调配吞吐量	447
删除预调配吞吐量	448
使用预配置吞吐量运行推理	449
代码示例	450
标记资源	455
使用控制台	456
使用 API	456
最佳实践和限制	458
亚马逊Titan模特	459
Amazon Titan 文本	459
Amazon Titan Text G1 - Express	459
Amazon Titan Text G1 - Lite	460
Amazon Titan 文本模型定制	460
Amazon Titan 短信提示工程指南	460
Amazon Titan Embeddings G1 - Text	461
Amazon Titan Multimodal Embeddings G1	462
嵌入长度	462
微调	463
准备数据集	463

超参数	463
Amazon Titan Image Generator G1	464
功能	464
参数	465
微调	465
输出	466
水印检测	466
提示工程准则	467
安全性	469
数据保护	469
数据加密	471
使用亚马逊 VPC 和 AWS PrivateLink	480
Identity and Access Management	482
受众	483
使用身份进行身份验证	483
使用策略管理访问	486
Amazon Bedrock 如何与 IAM 搭配使用	488
基于身份的策略示例	494
AWS 托管策略	502
服务角色	505
故障排除	517
合规性验证	519
事件响应	520
恢复能力	520
基础设施安全性	520
防止跨服务混淆座席	521
Amazon Bedrock 中的配置和漏洞分析	522
使用接口 VPC 端点 (AWS PrivateLink)	480
注意事项	480
创建接口端点	481
创建端点策略	481
监控 Amazon Bedrock	525
模型调用日志记录	525
设置 Amazon S3 目标	525
设置 CloudWatch 日志目标	527
使用 控制台	528

将 API 与调用日志记录搭配使用	529
使用监视器 CloudWatch	530
运行时指标	530
记录 CloudWatch 指标	531
使用 Amazon Bedrock 的 CloudWatch 指标	531
查看 Amazon Bedrock 指标	532
监控事件	532
工作方式	533
EventBridge 架构	533
规则和目标	534
创建规则以处理 Amazon Bedrock 事件	535
CloudTrail 日志	536
Amazon Bedrock 信息位于 CloudTrail	537
Amazon Bedrock 中的数据事件 CloudTrail	537
中的 Amazon Bedrock 管理活动 CloudTrail	539
了解 Amazon Bedrock 日志文件条目	539
代码示例	541
Amazon Bedrock	542
操作	548
场景	562
Amazon Bedrock 运行时系统	563
操作	569
场景	678
Agents for Amazon Bedrock	690
操作	694
场景	718
亚马逊 Bedrock Runtime 的代理	731
操作	732
场景	736
滥用检测	738
配额	739
运行时限额	739
批量推理配额	743
知识库配额	744
代理配额	744
模型自定义配额	745

预调配吞吐量配额	749
API 参考	750
文档历史记录	751
AWS 词汇表	756
.....	dcclvii

什么是 Amazon Bedrock ？

Amazon Bedrock 是一项完全托管的服务，它通过统一的 API 提供来自领先的人工智能初创企业和 Amazon 的高性能根基模型 (FM) 供您使用。您可以从各种根基模型中选择，找到最适合您的用例的模型。Amazon Bedrock 还提供了一系列广泛的功能，可以构建生成式人工智能应用程序，为您提供安全可靠的专属人工智能服务。利用 Amazon Bedrock，您可以轻松试验和评估用例的常用根基模型，使用微调和检索增强生成 (RAG) 等技术，通过自己的数据进行量身定制，并构建使用企业系统和数据来源执行任务的代理。

借助 Amazon Bedrock 的无服务器体验，您可以快速入门，使用自己的数据私下自定义基础模型，并使用 AWS 工具轻松安全地将其集成和部署到您的应用程序中，而无需管理任何基础架构。

主题

- [Amazon Bedrock 的功能](#)
- [Amazon Bedrock 的定价](#)
- [支持的 AWS 区域](#)
- [关键定义](#)

Amazon Bedrock 的功能

利用 Amazon Bedrock 基础模型来探索以下功能。要查看按地区划分的功能限制，请参阅[按 AWS 地区划分的模型支持](#)。

- 试验提示和配置 – [运行模型推理](#)通过使用不同的配置和根基模型发送提示来生成响应。您可以使用 API 或控制台中的文本、图像和聊天操场在图形界面中进行试验。准备就绪后，设置应用程序以便向 InvokeModel API 发送请求。
- 使用来自数据来源的信息增强响应生成 – 通过上传要查询的数据来源[创建知识库](#)，以增强根基模型的响应生成。
- 创建可推理如何帮助客户的应用程序 – [构建可以使用根基模型、进行 API 调用以及（可选）查询知识库的代理](#)，以便为客户推理和执行任务。
- 使用训练数据调整模型以适应特定任务和领域 – 通过提供用于微调或持续预训练的训练数据来[自定义 Amazon Bedrock 根基模型](#)，以调整模型的参数并提高其在特定任务或领域中的性能。
- 提高基于 FM 的应用程序的效率和输出 – 为根基模型[购买预调配吞吐量](#)，以便更高效地以折扣价对模型运行推理。

- 为您的用例确定最佳模型 – 使用内置或自定义提示数据集[评估不同模型的输出](#)，以确定最适合您的应用程序的模型。

Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。

- 防止不当或不必要的内容 – 使用 Guardrails for Amazon Bedrock，为您的生成式人工智能应用程序实施保护措施。

Note

Guardrails for Amazon Bedrock 为限制性预览版。要申请访问权限，请联系您的 AWS 账户经理。

Amazon Bedrock 的定价

当您注册时 AWS，您的 AWS 账户会自动注册使用中的所有服务 AWS，包括 Amazon Bedrock。不过，您只需为使用的服务付费。

若要查看您的账单，请转到 [AWS Billing and Cost Management 控制台](#) 中的账单和成本管理控制面板。要了解有关 AWS 账户计费的更多信息，请参阅 [AWS Billing 用户指南](#)。如果您对 AWS 账单有疑问 AWS 账户，请联系 Supp [AWS ort](#)。

使用 Amazon Bedrock，您可以付费来对任何第三方根基模型运行推理。定价基于输入词元和输出词元的数量，以及您是否为模型购买了预调配吞吐量。有关更多信息，请参阅 Amazon Bedrock 控制台中的 [模型提供商](#) 页面。每种模型的定价都列在模型版本之后。有关如何购买预调配吞吐量的更多信息，请参阅 [Amazon Bedrock 的预配置吞吐量](#)。

有关更多信息，请参阅 [Amazon Bedrock 的定价](#)。

支持的 AWS 区域

有关 Amazon Bedrock 所支持区域的服务端点的信息，请参阅 [Amazon Bedrock 端点和配额](#)。

要查看每个区域支持哪些基础模型，请参阅 [按 AWS 地区划分的模型支持](#)。

Note

美国东部（弗吉尼亚州北部）和美国西部（俄勒冈州）区域内提供所有功能。

请参阅下表，了解受区域限制的功能。

区域	模型评估	知识库	代理	微调（自定义模型）	持续预训练（自定义模型）	预调配吞吐量
美国东部 （弗吉尼亚州北部）	是	是	是	是	是	支持
US West (Oregon)	是	是	是	是	是	支持
亚太地区 （新加坡）	否	否	否	否	否	否
亚太地区 （东京）	否	否	否	否	否	否
欧洲地区 （法兰克福）	否	否	否	否	否	否
AWS GovCloud （美国西部）	否	否	否	是	否	是（仅面向经过微调的模型，无承诺期限）

关键定义

本章提供了概念的定义，这些概念将帮助您理解 Amazon Bedrock 提供的内容及其工作原理。如果您是初次使用者，则应先通读基本概念。在您熟悉了 Amazon Bedrock 的基础知识后，我们建议您探索亚马逊 Bedrock 提供的高级概念和功能。

基本概念

以下列表向您介绍了生成式人工智能的基本概念和 Amazon Bedrock 的基本功能。

- **基础模型 (FM)** — 一种具有大量参数并基于大量不同数据训练的人工智能模型。基础模型可以为各种用例生成各种响应。基础模型可以生成文本或图像，也可以将输入转换为嵌入式。在使用 Amazon Bedrock 基础模型之前，您必须先[申请访问权限](#)。有关基础模型的更多信息，请参阅[Amazon Bedrock 中支持的根基模型](#)。
- **基础模型** — 由提供商打包并随时可用的基础模型。Amazon Bedrock 提供来自领先供应商的各种行业领先的基础模型。有关更多信息，请参阅[Amazon Bedrock 中支持的根基模型](#)。
- **模型推断** — 基础模型根据给定输入（提示）生成输出（响应）的过程。有关更多信息，请参阅[运行模型推理](#)。
- **P@@ ro mpt** — 为模型提供的输入，用于指导模型为输入生成适当的响应或输出。例如，文本提示可以由一行组成，供模型响应，也可以详细说明模型要执行的说明或任务。提示可以包含任务的上下文、输出示例或模型在响应中使用的文本。提示可用于执行诸如分类、问题解答、代码生成、创意写作等任务。有关更多信息，请参阅[提示工程准则](#)。
- **Token** — 模型可以将其解释或预测为单一含义单位的字符序列。例如，在文本模型中，标记不仅可以对应于单词，还可以对应于具有语法含义的单词的一部分（例如“-ed”）、标点符号（例如“？”），或者一个常用短语（例如“很多”）。
- **模型参数**-定义模型及其在解释输入和生成响应时的行为的值。模型参数由提供者控制和更新。您还可以通过模型自定义过程更新模型参数以创建新模型。
- **推理参数**-在模型推理期间可以调整以影响响应的值。推理参数可以影响响应的变化程度，也可以限制响应的长度或指定序列的出现次数。有关特定推理参数的更多信息和定义，请参阅[推理参数](#)。
- **P@@ layground** — 一个用户友好的图形界面，您可以在其中尝试运行模型推理，以熟悉 Amazon Bedrock。AWS Management Console 使用操场测试不同模型、配置和推理参数对您输入的不同提示生成的响应的影响。有关更多信息，请参阅[操场](#)。
- **Embedding** — 通过将输入转换为数值向量（称为嵌入）来压缩信息的过程，以便使用共享的数值表示来比较不同对象之间的相似性。例如，可以比较句子以确定含义的相似性，可以比较图像以确定视觉相似性，或者可以比较文本和图像以查看它们是否相关。如果文本和图像输入与您的用例相关，也可以将其组合成平均嵌入向量。有关更多信息，请参阅[运行模型推理](#)和[Amazon Bedrock 知识库](#)。

高级功能

以下列表向您介绍了更多高级概念，您可以使用 Amazon Bedrock 探索这些概念。

- **编排** — 为执行任务而在基础模型与企业数据和应用程序之间进行协调的过程。有关更多信息，请参阅[Agents for Amazon Bedrock](#)。
- **Agent** — 一种通过周期性地解释输入和使用基础模型生成输出来执行协调的应用程序。代理可以用来执行客户的请求。有关更多信息，请参阅[Agents for Amazon Bedrock](#)。
- **检索增强生成 (RAG)**-从数据源中查询和检索信息的过程，以增强对提示生成的响应。有关更多信息，请参阅[Amazon Bedrock 知识库](#)。
- **模型自定义**-使用训练数据调整基础模型中的模型参数值以创建自定义模型的过程。模型定制的示例包括使用带标签的数据（输入和相应的输出）的微调和持续的预训练，后者使用未标记的数据（仅限输入）来调整模型参数。有关 Amazon Bedrock 中提供的模型自定义技术的更多信息，请参阅[自定义模型](#)。
- **Hyperparameters** — 可以针对模型自定义进行调整的值，以控制训练过程，从而控制输出的自定义模型。有关特定超参数的更多信息和定义，请参见[自定义模型超参数](#)。
- **模型评估** — 评估和比较模型输出以确定最适合用例的模型的过程。有关更多信息，请参阅[模型评估](#)。
- **预配置吞吐量** — 您为基础模型或自定义模型购买的吞吐量级别，用于增加模型推理期间处理的代币数量和/或速率。当您为模型购买预配置吞吐量时，系统会创建一个可用于进行模型推断的预配置模型。有关更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)。

设置 Amazon Bedrock

首次使用 Amazon Bedrock 前，请完成以下任务。在控制台中设置账户并请求模型访问权限后，就可以设置 API 了。

Important

要使用任何根基模型，必须先请求获得对该模型的访问权限。如果您在请求模型访问权限之前尝试使用该模型（通过 API 或控制台），将会收到一条错误消息。有关更多信息，请参阅[模型访问权限](#)。

设置任务

- [注册获取 AWS 账户](#)
- [创建管理用户](#)
- [授权以编程方式访问](#)
- [控制台访问](#)
- [模型访问权限](#)
- [设置 Amazon Bedrock API](#)
- [将此服务与 AWS SDK 配合使用](#)

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 <https://portal.aws.amazon.com/billing/signup>。
2. 按照屏幕上的说明进行操作。

在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

当您注册时 AWS 账户，就会创建 AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务和资源。作为安全最佳实操，请 [为管理用户分配管理访问权限](#)，并且只使用根用户执行 [需要根用户访问权限的任务](#)。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 <https://aws.amazon.com/> 并选择我的账户来查看当前的账户活动并管理您的账户。

创建管理用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就可以不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。[AWS Management Console](#)在下一页上，输入密码。

要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的[以根用户身份登录](#)。

2. 对您的根用户启用多重身份验证 (MFA)。

有关说明，请参阅 [IAM 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备 \(控制台 \)](#)。

创建管理用户

1. 启用 IAM Identity Center

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的[启用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，为管理用户授予管理访问权限。

有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《[用户指南](#)》[IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限](#)。

作为管理用户登录

- 要使用 IAM Identity Center 用户身份登录，请使用在创建 IAM Identity Center 用户时发送到电子邮件地址的登录网址。

有关使用 IAM Identity Center 用户[登录的帮助](#)，请参阅[AWS 登录 用户指南中的登录 AWS 访问门户](#)。

授权以编程方式访问

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 (在 IAM Identity Center 中管理的用户)	使用临时证书签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 <ul style="list-style-type: none"> • 有关的 AWS CLI，请参阅 《AWS Command Line Interface 用户指南》AWS IAM Identity Center 中的“配置 AWS CLI 要使用”。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅 《软件开发工具包和 AWS 工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的 将临时证书与 AWS 资源配合使用 中的说明进行操作。
IAM	(不推荐使用) 使用长期凭证签署向 AWS CLI、AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 <ul style="list-style-type: none"> • 有关信息 AWS CLI，请参阅用户指南中的 使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参

哪个用户需要编程式访问权限？	目的	方式
		<p>考指南中的使用长期凭证进行身份验证。</p> <ul style="list-style-type: none">有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

控制台访问

要访问 Amazon Bedrock 控制台和操场，请执行以下操作：

1. 登录到你的 AWS 账户。
2. 导航至 [Amazon Bedrock 控制台](#)
3. 按照[模型访问权限](#)中的步骤请求模型访问权限。

模型访问权限

默认情况下，不授予对 Amazon Bedrock 基础模型的访问权限。要获得基础模型的访问权限，拥有[足够权限的 IAM 用户](#)需要通过控制台请求访问该模型。向模型提供访问权限后，该账户中的所有用户均可使用该访问权限。

要管理模型访问权限，请在 Amazon Bedrock 管理控制台的左侧导航窗格底部选择模型访问权限。模型访问页面允许您查看可用模型列表、模型的输出模式、是否已获得访问权限以及最终用户许可协议 (EULA)。在申请访问模型之前，您应查看 EULA 以了解使用模型的条款和条件。有关型号定价的信息，请参阅 [Amazon Bedrock 定价](#)。

Note

您只能通过控制台管理模型访问权限。

The screenshot shows the Amazon Bedrock console interface. On the left, a navigation sidebar lists various categories: Getting started, Foundation models, Playgrounds, Orchestration, and Assessment & deployment. The 'Model access' option is circled in red with a red arrow pointing to it. The main content area is titled 'Overview' and includes tabs for 'Explore & Learn' and 'Build & Test'. The 'Foundation models' section lists several providers and their models: AI21 labs (Jurassic-2 series), Amazon (Titan), Anthropic (Claude), Cohere (Command), Meta (Llama 2), and Stability AI (Stable Diffusion). The 'Spotlight' section features Anthropic with a description of its Claude family models and a button to 'Open in chat playground'. The 'Playgrounds' section shows three preview images of chat and image generation interfaces. The 'Use cases example' section mentions that Amazon Bedrock supports various genAI use cases like summarization, Q&A, and image generation.

主题

- [添加模型访问权限](#)
- [删除模型访问权限](#)
- [控制模型访问权限](#)

添加模型访问权限

在 Amazon Bedrock 中使用基础模型之前，您必须申请访问该模型的权限。

请求访问模型

1. 在模型访问权限页面上，选择管理模型访问权限。
2. 选中要添加访问权限的模型旁边的复选框。要请求访问属于某个提供商的所有模型，请选中该提供商旁边的复选框。

Note

请求后，您无法从Titan模型中移除访问权限。

对于Anthropic模型，请选择提交用例详细信息并填写表格。提交详细信息后，您可以选中要请求访问的Anthropic模型旁边的复选框。

3. 选择“保存更改”以请求访问权限。更改可能需要几分钟才能生效。

Note

您使用 Amazon Bedrock 基础模型需遵守[卖家的定价条款](#)、最终用户许可协议和[AWS 服务条款](#)。

4. 如果您的请求成功，则访问状态将更改为已授予访问权限。

如果您无权请求访问模型，则会显示错误横幅。请联系您的账户管理员，要求他们为您申请模型的访问权限，[或者向您提供请求访问该模型的权限](#)。

删除模型访问权限

如果您不再需要使用基础模型，则可以删除对该模型的访问权限。

Note

您无法从 Amazon Titan 模型中移除访问权限。

1. 在模型访问权限页面上，选择管理模型访问权限。
2. 选中要删除访问权限的模型旁边的复选框。要删除属于某个提供商的所有模型的访问权限，请选中该提供商旁边的复选框。
3. 选择保存更改。
4. 系统将提示您确认要删除对模型的访问权限。如果您同意这些条款并选择删除访问权限，

Note

在更改传播期间，完成此操作后，仍可能在一段时间内通过 API 访问模型。要同时立即删除访问权限，请向[角色添加 IAM 策略以拒绝对模型的访问](#)。

控制模型访问权限

要控制角色请求访问 Amazon Bedrock 模型的权限，请使用以下任一 [AWS Marketplace 操作](#) 将一个 [IAM 策略](#) 附加到该角色。

- `aws-marketplace:Subscribe`
- `aws-marketplace:Unsubscribe`
- `aws-marketplace:ViewSubscriptions`

仅对于 `aws-marketplace:Subscribe` 操作，您可以使用 `aws-marketplace:ProductId` [条件键](#) 来限制对特定型号的订阅。下表列出了 Amazon Bedrock 基础型号的产品编号。

模型	产品 ID
AI21 Labs Jurassic-2 Mid	1d288c71-65f9-489a-a3e2-9c7f4f6e6a85
AI21 Labs Jurassic-2 Ultra	cc0bdd50-279a-40d8-829c-4009b77a1fcc
Anthropic Claude	c468b48a-84df-43a4-8c46-8870630108a7
Anthropic Claude Instant	b0eb9475-3a2c-43d1-94d3-56756fd43737
Anthropic Claude 3 Sonnet	prod-6dw3qvchef7zy
Anthropic Claude 3 Haiku	prod-ozonys2hmmpeu
Cohere Command	a61c46fe-1747-41aa-9af0-2e0ae8a9ce05
Cohere Command Light	216b69fd-07d5-4c7b-866b-936456d68311
Cohere 嵌入 (英语)	b7568428-a1ab-46d8-bab3-37def50f6f6a
Cohere 嵌入 (多语言)	38e55671-c3fe-4a44-9783-3584906e7cad
MetaLlama 213B	prod-ariujvyzvd2qy
MetaLlama 270B	prod-2c2yc2s3guhqy
Stable Diffusion XL0.8	d0123e8d-50d6-4dba-8a26-3fed4899f388

模型	产品 ID
Stable Diffusion XL 1.0	prod-2lvuzn4iy6n6o

以下是您可以附加到角色以控制模型访问权限的 IAM 策略的格式。您可以在上看到一个例子[允许访问第三方模型订阅](#)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow/Deny",
      "Action": [
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws-marketplace:ProductId": [
            "model-product-id-1",
            "model-product-id-2",
            ...
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:Unsubscribe",
        "aws-marketplace:ViewSubscriptions"
      ],
      "Resource": "*"
    }
  ]
}
```


设置 Amazon Bedrock API

本节介绍如何设置您的环境以进行 Amazon Bedrock API 调用，并提供了常见使用案例的示例。您可以使用 AWS Command Line Interface (AWS CLI)、AWS 软件开发工具包或 SageMaker 笔记本来访问 Amazon Bedrock API。

在访问 Amazon Bedrock API 之前，您需要申请访问计划使用的基础模型。

有关 API 操作和参数的详细信息，请参阅 [Amazon Bedrock API 参考](#)。

以下资源提供了有关 Amazon Bedrock API 的更多信息。

- AWS Command Line Interface
 - [Amazon Bedrock CLI 命令](#)
 - [Amazon Bedrock 运行时 CLI 命令](#)
 - [Amazon Bedrock 代理 CLI 命令](#)
 - [Amazon Bedrock 代理运行时 CLI 命令](#)

添加模型访问权限

Important

要使用任何根基模型，必须先请求获得对该模型的访问权限。如果您在请求模型访问权限之前尝试使用该模型（通过 API 或控制台），将会收到一条错误消息。有关更多信息，请参阅 [模型访问权限](#)。

Amazon Bedrock 端点

要以编程方式连接到 AWS 服务，请使用终端节点。有关可用于 [Amazon Bedrock 的终端节点的信息](#)，[AWS 一般参考 请参阅中的 Amazon Bedrock 终端节点和配额](#) 一章。

Amazon Bedrock 提供以下服务端点。

- bedrock – 包含用于管理、训练和部署模型的控制层面 API。有关更多信息，请参阅 [Amazon Bedrock 操作](#) 和 [Amazon Bedrock 数据类型](#)。
- bedrock-runtime— 包含数据平面 API，用于对托管在 Amazon Bedrock 中的模型发出推理请求。有关更多信息，请参阅 [Amazon Bedrock 运行时操作](#) 和 [Amazon Bedrock 运行时数据类型](#)。

- `bedrock-agent` – 包含控制层面 API，用于创建和管理代理及知识库。有关更多信息，请参阅 [Amazon Bedrock 代理操作](#) 和 [Amazon Bedrock 代理数据类型](#)。
- `bedrock-agent-runtime`— 包含用于调用代理和查询知识库的数据平面 API。有关更多信息，请参阅 [Amazon Bedrock 代理运行时操作](#) 和 [Amazon Bedrock 代理运行时数据类型](#)。

设置 AWS CLI

1. 如果您计划使用 CLI，请 AWS CLI 按照 [安装或更新最新版本的 AWS Command Line Interface 用户指南中的步骤](#) 进行安装和配置。
2. 按照配置中的步骤，使用 `aws configure` CLI 命令 [配置](#) 您的 AWS 证书 AWS CLI。

有关 AWS CLI 命令和操作，请参阅以下 CLI 参考资料：

- [Amazon Bedrock CLI 命令](#)
- [Amazon Bedrock 运行时 CLI 命令](#)
- [Amazon Bedrock 代理 CLI 命令](#)
- [Amazon Bedrock 代理运行时 CLI 命令](#)

设置 AWS SDK

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地了解其首选语言构建应用程序。SDK 会自动为您执行有用的任务，例如：

- 对您的服务请求进行加密签名
- 重试请求
- 处理错误响应

请参阅下表，查找有关每个软件开发工具包的一般信息和代码示例，以及每个软件开发工具包的 Amazon Bedrock API 参考资料。您也可以在以下位置找到代码示例 [使用 AWS 软件开发工具包的 Amazon Bedrock 的代码示例](#)。

软件开发工具包文档	代码示例	Amazon Bedrock 前缀	Amazon Bedrock 运行时前缀	Amazon Bedrock 代理前缀	Amazon Bedrock 代理运行时前缀
AWS SDK for C++	AWS SDK for C++ 代码示例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime
AWS SDK for Go	AWS SDK for Go 代码示例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for Java	AWS SDK for Java 代码示例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for .NET	AWS SDK for .NET 代码示例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime
AWS SDK for PHP	AWS SDK for PHP 代码示例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime
AWS SDK for Ruby	AWS SDK for Ruby 代码示例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime

软件开发工具包文档	代码示例	Amazon Bedrock 前缀	Amazon Bedrock 运行时前缀	Amazon Bedrock 代理前缀	Amazon Bedrock 代理运行时前缀
AWS SDK for Rust	AWS SDK for Rust 代码示例	aws-sdk-bedrock	aws-sdk-bedrockruntime	aws-sdk-bedrockagent	aws-sdk-bedrockagentruntime
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例	BDK	BDR	BDA	BDZ
AWS SDK for Swift	AWS SDK for Swift 代码示例	AWSBedrock	AWSBedrockRuntime	AWSBedrockAgent	AWSBedrockAgentRuntime

使用 SageMaker 笔记本电脑

你可以使用适用于 Python 的 SDK (Boto3) 从笔记本调用 Amazon Bedrock API 操作。SageMaker

配置 SageMaker 角色

将 Amazon Bedrock 权限添加到将使用此 SageMaker 笔记本的 IAM 角色。

在 IAM 控制台中，执行以下步骤：

1. 选择 IAM 角色，然后选择添加权限并从下拉列表中选择创建内联策略。
2. 包括以下权限。

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

```

    ]
  }

```

向信任关系添加以下权限。

```

{
  "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"
    }
  ]
}

```

测试运行时设置

将以下代码添加到笔记本并运行代码。

```

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'))
```

测试 Amazon Bedrock 设置

将以下代码添加到笔记本并运行代码。

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

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

将此服务与 AWS SDK 配合使用

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地了解其首选语言构建应用程序。

软件开发工具包文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例

软件开发工具包文档	代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

Amazon Bedrock 中支持的根基模型

Amazon Bedrock 支持以下提供商提供的基础模型 (FM)。在“提供商”列中选择一个链接以查看该提供商的文档。

要将基础模型与 Amazon Bedrock API 配合使用，您需要其模型 ID。有关型号 ID 的列表，请参阅[亚马逊 Bedrock 型号](#)。

提供商	模型	输入模式	输出模式	推理参数	超参数
Amazon	Titan Text G1 - Express	文本	短信、聊天	Link	Link
	Titan Text G1 - Lite	文本	文本	Link	Link
	Titan Image Generator G1	文字、图片	图像	Link	Link
	Titan Embeddings G1 - Text	文本	嵌入	Link	不适用
	Titan Multimodal Embeddings G1	文字、图片	嵌入	Link	Link
Anthropic	Claude	文本	短信、聊天	Link	不适用
	Claude Instant	文本	短信、聊天	Link	不适用
	Claude 3 Sonnet	文字、图片	短信、聊天	Link	不适用
	Claude 3 Haiku	文字、图片	短信、聊天	Link	不适用

提供商	模型	输入模式	输出模式	推理参数	超参数
AI21 Labs	Jurassic-2 Mid	文本	短信、聊天	Link	不适用
	Jurassic-2 Ultra	文本	短信、聊天	Link	不适用
Cohere	Command	文本	文本	Link	Link
	Command Light	文本	文本	Link	Link
	Embed English	文本	嵌入	Link	不适用
	Embed Multilingual	文本	嵌入	Link	不适用
Meta	Llama 2 Chat13B	文本	短信、聊天	Link	不适用
	Llama 2 Chat70B	文本	短信、聊天	Link	不适用
	Llama 213B (参见下面的注释)	文本	文本	Link	Link
	Llama 270B (参见下面的注释)	文本	文本	Link	Link
Mistral AI	Mistral 7B Instruct	文本	文本	Link	不适用
	Mixtral 8X7B Instruct	文本	文本	Link	不适用

提供商	模型	输入模式	输出模式	推理参数	超参数
Stability AI	Stable Diffusion XL	文字、图片	图像	Link	不适用

Note

MetaLlama 2 (非聊天) 模型只能在[经过自定义并购买预配置吞吐量](#)后才能使用。

以下各节提供有关使用基础模型的信息以及模型的参考信息。

主题

- [使用基础模型](#)
- [获取有关根基模型的信息](#)
- [按 AWS 地区划分的模型支持](#)
- [按功能划分的型号支持](#)
- [模型生命周期](#)
- [亚马逊 Bedrock 型号](#)
- [根基模型的推理参数](#)
- [自定义模型超参数](#)

使用基础模型

必须先[请求访问模型](#)，然后才能使用它。完成此操作后，您可以通过以下方式使用 FM。

- 通过向模型发送提示并生成响应来@@ [运行推理](#)。[Playground](#) 中提供了一个用户友好的界面，AWS Management Console 用于生成文本、图像或聊天。请参阅“输出模态”列，确定可以在每个游乐场中使用的模型。

Note

主机游戏场不支持在嵌入模型上运行推理。使用 API 对嵌入模型进行推理。

- [评估模型](#)以比较输出并确定最适合您的用例的模型。

- 借助嵌入模型@@ [建立知识库](#)。然后使用文本模型生成对查询的响应。
- [创建代理](#)并使用模型对提示进行推理以进行编排。
- 通过@@ [提供训练和验证数据来自定义模型](#)，以根据您的用例调整模型参数。要使用自定义模型，必须为其购买[预置吞吐量](#)。
- 为模型@@ [购买预置吞吐量](#)以增加其吞吐量。

要在 API 中使用 FM，您需要确定要使用的相应型号 ID。

应用场景	如何找到模型 ID
使用基础模型	在 基本型号 ID 图表中查找 ID
为基本型号购买预置吞吐量	在预配置 吞吐量的模型 ID 图表 中查找 ID，然后将其用作 CreateProvisionedModelThroughput 请求modelId中的。
为自定义模型购买预置吞吐量	在请求中使用定制模型或其 ARN modelId 的 CreateProvisionedModelThroughput 名称。
使用预配置模型	创建预配置吞吐量后，它将返回。provisionedModelArn 此 ARN 是模型编号。
使用自定义模型	为自定义模型@@ 购买预置吞吐量 ，并使用返回的provisionedModelArn 作为模型 ID。

获取有关根基模型的信息

在 Amazon Bedrock 控制台中，您可以在提供商和基础模型部分找到有关 Amazon Bedrock 根基模型提供商及其所提供模型的总体信息。

使用 API 在对象中检索有关 Amazon Bedrock 基础模型的信息，包括其 ARN、型号 ID、其支持的模式和功能，以及该模型是否已被弃用。[FoundationModelSummary](#)

- 要返回有关 Amazon Bedrock 提供的所有基础模型的信息，请发送[ListFoundationModels](#)请求。

Note

响应还会返回不在[基本型号 ID 中的模型 ID](#) 或预配置[吞吐量图表的基本模型 ID](#)。这些模型 ID 已过时或是为了向后兼容。

- 要返回有关特定基础模型的信息，请发送[GetFoundationModel](#)请求并指定[模型 ID](#)。

选择一个选项卡，查看界面或语言中的代码示例。

AWS CLI

列出 Amazon Bedrock 根基模型。

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

获取有关 Anthropic Claude v2 的信息。

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

Python

列出 Amazon Bedrock 根基模型。

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

bedrock.list_foundation_models()
```

获取有关 Anthropic Claude v2 的信息。

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

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

按 AWS 地区划分的模型支持

Note

美国东部（弗吉尼亚北部us-east-1）和美国西部（俄勒冈us-west-2）地区支持所有型号。

下表显示了其他地区可用的功能模块，以及每个区域是否支持它们。

模型	亚太地区（新加坡）	亚太地区（东京）	欧洲地区（法兰克福）	AWS GovCloud（美国西部）
Amazon Titan Text G1 - Express	不支持	是	是	支持
Amazon Titan Embeddings G1 - Text	不支持	是	是	不支持
Anthropic Claudev2（18K 上下文窗口）	支持	否	是	不支持
Anthropic Claudev2.1（20 万个上下文窗口）	不支持	是	是	不支持
AnthropicClaude Instantv1.x（18K 上下文窗口）	支持	是	否	不支持
Anthropic Claude Instantv1.x（100K 上下文窗口）	不支持	否	是	不支持

按功能划分的型号支持

Note

您可以在所有可用的 F M 上运行推理。

下表详细说明了对于仅限于某些 FM 的功能的支持。

模型	模型评估	知识库 (嵌入式)	知识库 (查询)	代理	微调 (自 定义模 型)	持续预训 练 (自定 义模型)	预调配吞 吐量
Amazon Titan Text G1 - Express	支持	不适用	否	否	是	是	支持
Amazon Titan Text G1 - Lite	支持	不适用	否	否	是	是	支持
Amazon Titan Embedding s G1 - Text	不支持	不适用	否	否	否	否	支持
Amazon Titan Multimoda l Embedding s G1	不支持	是	否	否	是	否	支持
亚马逊 Titan	不支持	不适用	否	否	是	否	支持

模型	模型评估	知识库 (嵌入式)	知识库 (查询)	代理	微调 (自定义模型)	持续预训练 (自定义模型)	预调配吞吐量
Image Generator G1 (预览)							
Anthropic Claudev1	支持	不适用	否	否	否	否	支持
Anthropic Claudev2	支持	不适用	支持	是	否	否	支持
Anthropic Claudev2.1	不支持	不适用	支持	是	否	否	是
Anthropic Claude Instant	支持	不适用	支持	是	否	否	是
Anthropic Claude 3 Sonnet	不支持	不适用	支持	否	否	否	是
Anthropic Claude 3 Haiku	不支持	不适用	否	否	否	否	是
AI21 Labs Jurassic-2 Mid	是	否	否	否	否	否	否

模型	模型评估	知识库 (嵌入式)	知识库 (查询)	代理	微调 (自 定义模 型)	持续预训 练 (自定 义模型)	预调配吞 吐量
AI21 Labs Jurassic- 2 Ultra	是	否	否	否	否	否	否
Cohere Command	支持	不适用	否	否	是	否	是
Cohere Command Light	支持	不适用	否	否	是	否	支持
CohereEmk ed英语	不支持	是	否	否	否	否	支持
CohereEmk ed多语言	不支持	是	否	否	否	否	支持
MetaLlama 2 Chat13B	支持	不适用	否	否	否	否	支持
MetaLlama 2 Chat70B	支持	不适用	否	否	否	否	不支持
MetaLlama 213B	不支持	不适用	否	否	是	不支持	是 (参见 下面的注 释)
MetaLlama 270B	不支持	不适用	否	否	是	不支持	是 (参见 下面的注 释)

模型	模型评估	知识库 (嵌入式)	知识库 (查询)	代理	微调 (自定义模型)	持续预训练 (自定义模型)	预调配吞吐量
Mistral AI Mistral 7B Instruct	不支持	不适用	否	否	否	否	是
Mistral AI Mixtral 8X7B Instruct	不支持	不适用	否	否	否	否	支持
Stable Diffusion XL0.8	不支持	不适用	否	否	否	否	不支持
Stable Diffusion XL 1.x	不支持	不适用	否	否	否	否	支持

Note

MetaLlama 2 (非聊天) 模型只能在[经过自定义](#)并[购买预配置吞吐量](#)后才能使用。

模型生命周期

Amazon Bedrock 一直在努力推出具有更好功能、更高准确性和安全性的最新版根基模型。当我们推出新的模型版本时，您可以使用 Amazon Bedrock 控制台或 API 对其进行测试，并迁移您的应用程序以从最新的模型版本中受益。

Amazon Bedrock 上提供的模型可能处于以下状态之一：活动、旧版或生命周期终止 (EOL)。

- 活动：模型提供商正在积极完善此版本，并将继续提供更新，例如错误修复和细微改进。

- 旧版：如果有更新的版本可提供更高的性能，原来的版本将被标记为旧版。Amazon Bedrock 为旧版本设定了停产日期。EOL 日期可能会有所不同，具体取决于您使用模型的方式（例如，您是使用按需吞吐量还是预配置吞吐量作为基本模型，还是使用预置吞吐量作为自定义模型）。虽然您可以继续使用旧版本，但您应该计划在 EOL 日期之前过渡到活动版本。
- EOL：此版本已不再可用。对此版本提出的任何请求都将失败。

控制台将模型版本的状态标记为“活动”或“旧版”。当您拨打 [GetFoundationModel](#) 或 [ListFoundationModels](#) 调用时，可以在响应的 `modelLifecycle` 字段中找到模型的状态。在 EOL 日期之后，只能在此文档页面上找到模型版本。

按需、预调配吞吐量和模型自定义

在按需模式下使用模型时，您可以指定其版本（例如 `anthropic.claude-v2`、`anthropic.claude-v2:1`、等）。

配置预调配吞吐量时，必须指定在整个期间内保持不变的模型版本。如果承诺期限在版本的 EOL 日期之前结束，则可以为该版本购买新的预调配吞吐量承诺（或续订现有承诺）。

如果您自定义了一个模型，可以继续使用该模型，一直到用于自定义的基础模型版本的 EOL 日期。也可以自定义旧模型版本，但应计划在其到达 EOL 日期之前迁移。

Note

服务限额在模型次要版本之间共享。

旧版本

下表显示了 Amazon Bedrock 上可用的模型的旧版本。

模型版本	成为旧版日期	EOL 日期	推荐的模型替代版本	推荐型号
Stable Diffusion XL 0.8	2024 年 2 月 2 日	2024 年 4 月 30 日	Stable Diffusion XL 1.x	稳定性。stable-diffusion-xl-v1
Claude v1.3	2023 年 11 月 28 日	2024 年 2 月 28 日	Claude v2.1	anthropic.claude-v2:

模型版本	成为旧版日期	EOL 日期	推荐的模型替代版本	推荐型号
泰坦嵌入-文本 v1.1	2023 年 11 月 7 日	2024 年 2 月 15 日	Titan Embedding s - Text v1.2	亚马逊。 titan-embed-text-v1

亚马逊 Bedrock 型号

许多 Amazon Bedrock API 操作都需要使用型号 ID。请参阅下表，确定在哪里可以找到您需要使用的模型 ID。

应用场景	如何找到模型 ID
使用基础模型	在 基本型号 ID 图表中查找 ID
为基本型号购买预置吞吐量	在预配置 吞吐量的模型 ID 图表 中查找 ID，然后将其用作 CreateProvisionedModelThroughput 请求modelId中的。
为自定义模型购买预置吞吐量	在请求中使用定制模型或其 ARN modelId 的 CreateProvisionedModelThroughput 名称。
使用预配置模型	创建预配置吞吐量后，它将返回。provisionedModelArn 此 ARN 是型号 ID。
使用自定义模型	为自定义模型@@ 购买预置吞吐量 ，并使用返回的provisionedModelArn 作为模型 ID。

主题

- [Amazon Bedrock 基本模型 ID \(按需吞吐量 \)](#)
- [用于购买预配置吞吐量的 Amazon Bedrock 基本型号](#)

Amazon Bedrock 基本模型 ID (按需吞吐量)

以下是当前可用的基础模型的模型 ID 的列表。您可以通过 API 使用模型 ID 来标识要用于按需吞吐量的基本模型 (例如在 [InvokeModel](#) 请求中) 或要自定义的基本模型 (例如在 [CreateModelCustomizationJob](#) 请求中) 。

Note

您应定期查看 [模型生命周期](#) 页面以获取有关模型弃用的信息，并在必要时更新模型 ID。模型到达后 end-of-life，该模型 ID 将不再起作用。

提供商	模型名称	版本	型号标识
Amazon	Titan Text G1 - Express	1.x	亚马逊。 titan-text-express-v1
Amazon	Titan Text G1 - Lite	1.x	亚马逊。 titan-text-lite-v1
Amazon	Titan Embeddings G1 - Text	1.x	亚马逊。 titan-embed-text-v1
Amazon	Titan Multimodal Embeddings G1	1.x	亚马逊。 titan-embed-image-v1
Amazon	Titan Image Generator G1	1.x	亚马逊。 titan-image-generator-v1
Anthropic	Claude	2.0	anthropic.claude-v2
Anthropic	Claude	2.1	anthropic.claude-v 2:
Anthropic	Claude 3 Sonnet	1.0	anthropic.claude-3-sonnet-20240229-v 1:0
Anthropic	Claude 3 Haiku	1.0	anthropic.claude-3-haiku-20240307-v 1:0

提供商	模型名称	版本	型号标识
Anthropic	Claude Instant	1.x	人类。 claude-instant-v1
AI21 Labs	Jurassic-2 Mid	1.x	ai21.j2-mid-v1
AI21 Labs	Jurassic-2 Ultra	1.x	ai21.j2-ultra-v1
Cohere	命令	14.x	一致。 command-text-v14
Cohere	Command Light	15.x	一致。 command-light-text-v14
Cohere	Embed英语	3.x	一致。 embed-english-v3
Cohere	Embed多语言	3.x	一致。 embed-multilingual-v3
Meta	Llama 2 Chat13B	1.x	meta.llama2-13.1b-chat-v
Meta	Llama 2 Chat70B	1.x	meta.llama2-70.1b-chat-v
Mistral AI	Mistral 7B Instruct	0.x	mistral.mistral-7.0:2b-instruct-v
Mistral AI	Mixtral 8X7B Instruct	0.x	mistral.mixtral-8x7.0:1b-instruct-v
Stability AI	Stable Diffusion XL	0.x	稳定性。 stable-diffusion-xl-v0
Stability AI	Stable Diffusion XL	1.x	稳定性。 stable-diffusion-xl-v1

用于购买预配置吞吐量的 Amazon Bedrock 基本型号


要通过 API 购买预配置吞吐量，请在使用 [CreateProvisionedModelThroughput](#) 请求配置模型时使用相应的型号 ID。预配置吞吐量适用于以下型号：

Note

有些模型有多个上下文版本，其可用性因地区而异。有关更多信息，请参阅 [按 AWS 地区划分的模型支持](#)。

模型名称	基本型号支持无承诺购买	预置吞吐量的型号 ID
Amazon Titan Text G1 - Express	支持	亚马逊。 titan-text-express-v1:08 k
Amazon Titan Text G1 - Lite	支持	亚马逊。 titan-text-lite-v1:0:4 k
Amazon Titan Embeddings G1 - Text	支持	亚马逊。 titan-embed-text-v1:2:8 k
Amazon Titan Multimodal Embeddings G1	支持	亚马逊。 titan-embed-image-v1:0
Amazon Titan Image Generator G1	不支持	亚马逊。 titan-image-generator-v1:0
AnthropicClaudev2 18K	支持	anthropic.claude-v2:0:18k
AnthropicClaudev2 100K	支持	anthropic.claude-v2:0:100k
AnthropicClaudev2.1 18K	支持	anthropic.claude-v2:1:18k
AnthropicClaudev2.1 200K	支持	anthropic.claude-v 2:1:200k
AnthropicClaude 3 Sonnet28K	支持	anthropic.claude-3-sonnet-20240229-v 1:0:28 k
AnthropicClaude 3 Sonnet200K	支持	anthropic.claude-3-sonnet-20240229-v 1:0:200 k

模型名称	基本型号支持无承诺购买	预置吞吐量的型号 ID
AnthropicClaude 3 Haiku48K	支持	anthropic.claude-3-haiku-20240307-v 1:0:48 k
AnthropicClaude 3 Haiku200K	支持	anthropic.claude-3-haiku-20240307-v 1:0:200 k
AnthropicClaude Instantv1 100K	支持	人类。 claude-instant-v1:2:100 k
Cohere Command	支持	一致。 command-text-v14:7:4 k
Cohere Command Light	支持	一致。 command-light-text-v14:7:4 k
CohereEmbed英语	支持	一致。 embed-english-v3:0:512
CohereEmbed多语言	支持	一致。 embed-multilingual-v3:0:512
Stable Diffusion XL 1.0	不支持	稳定性。 stable-diffusion-xl-v1:0
MetaLlama 2 Chat13B	不支持	meta.llama2-13 1:0:4 k b-chat-v
MetaLlama 213B	不支持	(参见下面的注释)
MetaLlama 270B	不支持	(参见下面的注释)

 Note

MetaLlama 2 (非聊天) 模型只能在[经过自定义并购买预配置吞吐量](#)后才能使用。

Note

MetaLlama 2 (非聊天) 模型只能在自定义后进行配置。有关更多信息, 请参阅 [自定义模型](#)。使用生成的自定义模型的名称或 ARN 为其购买预配置吞吐量。

[CreateProvisionedModelThroughput](#) 响应返回 a `provisionedModelArn`。您可以在支持的 Amazon Bedrock 操作中使用此 ARN 或预配置模型的名称。有关预配置吞吐量的更多信息, 请参阅 [Amazon Bedrock 的预配置吞吐量](#)。

根基模型的推理参数

本节介绍了您可以在 Amazon Bedrock 提供的基础模型中使用的推理参数。

(可选) 设置推理参数以影响模型生成的响应。你可以在控制台的操场上设置推理参数, 或者在 [InvokeModel](#) 或 [InvokeModelWithResponseStream](#) API 的 `body` 字段中设置推理参数。

在调用模型时, 还可以包括对模型的提示。有关编写提示的信息, 请参阅 [提示工程准则](#)。

以下各节定义了可用于每个基础模型的推理参数。对于自定义模型, 请使用与自定义它时所依据的基础模型相同的推理参数。

主题

- [亚马逊Titan模型](#)
- [AnthropicClaude模型](#)
- [AI21 LabsJurassic-2模型](#)
- [Cohere模型](#)
- [MetaLlama 2和MetaLlama 2 Chat模型](#)
- [Mistral AI模型](#)
- [Stability.ai Diffusion 模型](#)

亚马逊Titan模型

以下页面描述了 Amazon Titan 模型的推理参数。

主题

- [Amazon Titan Text 模型](#)
- [Amazon Titan Image Generator G1](#)
- [Amazon Titan Embeddings G1 - Text](#)
- [Amazon Titan Multimodal Embeddings G1](#)

Amazon Titan Text 模型

Amazon Titan 文本模型支持以下推理参数。

有关Titan文本提示工程指南的更多信息，请参阅[Titan文本提示工程指南](#)。

有关Titan模型的更多信息，请参阅[亚马逊Titan模特](#)。

主题

- [请求和回应](#)
- [代码示例](#)

请求和回应

请求正文在[InvokeModel](#)或[InvokeModelWithResponseStream](#)请求的body字段中传递。

Request

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

以下参数为必需参数：

- InputText — 提示提供生成响应的模型。要以对话方式生成回复，请使用以下格式封装提示：

```
"inputText": "User: <prompt>\nBot:
```

`textGenerationConfig` 是可选项。您可以使用它来配置以下[推理参数](#)：

- 温度-使用较低的值来降低响应的随机性。

默认	最低	最高
0.0	0.0	1.0

- TopP — 使用较低的值忽略可能性较小的选项并减少响应的多样性。

默认	最低	最高
1	1.00E-45	1

- `maxTokenCount`— 指定要在响应中生成的最大令牌数。

默认	最低	最高
512	0	8000

- `stop Sequences`-指定字符序列以指示模型应在哪里停止。目前，您只能指定以下选项之一：
 - |
 - User:

InvokeModel Response

响应正文包含以下可能的字段：

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

下面提供了有关每个字段的更多信息。

- `inputTextTokenCount` – 提示中的令牌数量。
- `tokenCount` – 响应中的令牌数量。
- `outputText` – 响应中的文本。
- `completionReason` – 响应结束生成操作的原因。可能有以下原因。
 - FINISHED – 响应已完全生成。
 - LENGTH – 由于您设置的响应长度，响应被截断。

InvokeModelWithResponseStream Response

响应流正文中的每块文本均采用以下格式。您必须对 `bytes` 字段进行解码（有关示例，请参阅 [使用 API 通过单个提示调用模型](#)）。

```
{
  'chunk': {
    'bytes': b'{
      "index": int,
      "inputTextTokenCount": int,
      "totalOutputTextTokenCount": int,
      "outputText": "<response-chunk>",
      "completionReason": string
    }'
  }
}
```

- `index` – 流式传输响应中数据块的索引。
- `inputTextTokenCount` – 提示中的令牌数量。
- `totalOutputTextTokenCount` – 响应中的令牌数量。
- `outputText` – 响应中的文本。
- `completionReason` – 响应结束生成操作的原因。可能有以下原因。
 - FINISHED – 响应已完全生成。
 - LENGTH – 由于您设置的响应长度，响应被截断。

代码示例

以下示例展示了如何使用 Python SDK 使用亚马逊 Titan Text G1 - Express 模型运行推理。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to create a list of action items from a meeting transcript
with the Amazon Titan-Text-Express model (on demand).
"""
import json
import logging
import boto3

from botocore.exceptions import ClientError

class ImageError(Exception):
    "Custom exception for errors returned by Amazon Titan-Text-Express model"

    def __init__(self, message):
        self.message = message

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

def generate_text(model_id, body):
    """
    Generate text using Amazon Titan-Text-Express model on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        response (json): The response from the model.
    """

    logger.info(
        "Generating text with Amazon Titan-Text-Express 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())

    finish_reason = response_body.get("error")

    if finish_reason is not None:
        raise ImageError(f"Text generation error. Error is {finish_reason}")

    logger.info(
        "Successfully generated text with Amazon &titan-text-express; model %s",
        model_id)

    return response_body

def main():
    """
    Entrypoint for Amazon &titan-text-express; example.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        model_id = 'amazon.titan-text-express-v1'

        prompt = """Meeting transcript: Miguel: Hi Brant, I want to discuss the
workstream
                for our new product launch Brant: Sure Miguel, is there anything in
particular you want
                to discuss? Miguel: Yes, I want to talk about how users enter into the
product.
                Brant: Ok, in that case let me add in Namita. Namita: Hey everyone
                Brant: Hi Namita, Miguel wants to discuss how users enter into the product.
                Miguel: its too complicated and we should remove friction.
                for example, why do I need to fill out additional forms?
                I also find it difficult to find where to access the product
                when I first land on the landing page. Brant: I would also add that
                I think there are too many steps. Namita: Ok, I can work on the
                landing page to make the product more discoverable but brant
                can you work on the aditional forms? Brant: Yes but I would need
                to work with James from another team as he needs to unblock the sign up
workflow.
```

Miguel can you document any other concerns so that I can discuss with James only once?

Miguel: Sure.

From the meeting transcript above, Create a list of action items for each person. """

```
body = json.dumps({
    "inputText": prompt,
    "textGenerationConfig": {
        "maxTokenCount": 4096,
        "stopSequences": [],
        "temperature": 0,
        "topP": 1
    }
})

response_body = generate_text(model_id, body)
print(f"Input token count: {response_body['inputTextTokenCount']}")

for result in response_body['results']:
    print(f"Token count: {result['tokenCount']}")
    print(f"Output text: {result['outputText']}")
    print(f"Completion reason: {result['completionReason']}")

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 the Amazon &titan-text-express; model
        {model_id}.")

if __name__ == "__main__":
    main()
```

Amazon Titan Image Generator G1

在执行Titan Image Generator G1模型推理时，Amazon 模型支持以下推理参数和模型响应。

主题

- [请求和响应格式](#)
- [代码示例](#)

请求和响应格式

当您使用 Amazon [InvokeModel](#)拨打电话时Titan Image Generator G1，请将请求body字段替换为与您的用例相匹配的格式。所有任务共享一个 `imageGenerationConfig` 对象，但每个任务都有一个特定于该任务的参数对象。支持以下用例：

taskType	任务参数字段	任务类型	定义
TEXT_IMAGE	<code>textToImageParams</code>	生成	使用文本提示生成图像。
INPAINTING	<code>inPaintingParams</code>	编辑	通过更改掩膜内部以匹配周围背景来修改图像。
OUTPAINTING	<code>outPaintingParams</code>	编辑	通过无缝扩展掩膜定义的区域来修改图像。
IMAGE_VARIATION	<code>imageVariationParams</code>	编辑	通过生成原始图像的变体来修改图像。

编辑任务需要在输入中使用一个 `image` 字段。该字段由一个字符串组成，用于定义图像中的像素。每个像素由 3 个 RGB 通道定义，每个通道的范围为 0 到 255（例如，(255 255 0) 表示黄色）。这些通道以 base64 编码。

您使用的图像必须为 JPEG 或 PNG 格式。

如果要进行补绘或扩绘，则还需定义一个掩膜，即用于定义要修改的图像部分的一个或多个区域。可以通过以下两种方式之一定义掩膜：

- `maskPrompt` – 编写一个文本提示来描述图像中要遮盖的部分。
- `maskImage` – 输入一个 base64 编码的字符串，通过将输入图像中的每个像素标记为 (0 0 0) 或 (255 255 255) 来定义掩膜覆盖区域。
 - 定义为 (0 0 0) 的像素是掩膜内的像素。
 - 定义为 (255 255 255) 的像素是掩膜外的像素。

可以使用照片编辑工具来绘制掩膜。然后，可以将输出的 JPEG 或 PNG 图像转换为 base64 编码，然后输入到该字段中。否则，请改用 `maskPrompt` 字段允许模型推断掩膜。

选择一个选项卡，查看不同图像生成用例的 API 请求正文和字段说明。

Text-to-image generation (Request)

```
{
  "taskType": "TEXT_IMAGE",
  "textToImageParams": {
    "text": "string",
    "negativeText": "string"
  },
  "imageGenerationConfig": {
    "numberOfImages": int,
    "height": int,
    "width": int,
    "cfgScale": float,
    "seed": int
  }
}
```

`textToImageParams` 字段如下所述。

- `text` (必要) – 用于生成图像的文本提示。
- `negativeText` (可选) – 用于定义图像内不包含什么内容的文本提示。

Note

请勿在 `negativeText` 提示中使用否定词。假如您不想在图像中包含镜像，请在 `negativeText` 提示中输入 **mirrors**，而不要输入 **no mirrors**。

Inpainting (Request)

```
{
  "taskType": "INPAINTING",
  "inPaintingParams": {
    "image": "base64-encoded string",
    "text": "string",
    "negativeText": "string",
    "maskPrompt": "string",
    "maskImage": "base64-encoded string",
  },
  "imageGenerationConfig": {
    "numberOfImages": int,
    "height": int,
    "width": int,
    "cfgScale": float
  }
}
```

`inPaintingParams` 字段如下所述。掩膜用于定义您要修改的图像部分。

- `image` (必要) – 要修改的 JPEG 或 PNG 图像，格式化为指定像素序列的字符串，每个像素以 RGB 值定义并以 base64 编码。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- 必须定义以下字段之一（但不能同时定义两个字段）才能定义。
 - `maskPrompt` – 用于定义掩膜的文本提示。
 - `maskImage` – 一个字符串，通过指定与 `image` 大小相同的像素序列来定义掩膜。每个像素的 RGB 值都将变为 (0 0 0) (掩膜内的像素) 或 (255 255 255) (掩膜外的像素)。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- `text` (可选) – 用于定义要在掩膜内更改什么内容的文本提示。如果未包含该字段，模型将尝试将整个掩膜区域替换为背景。
- `negativeText` (可选) – 用于定义图像内不包含什么内容的文本提示。

Note

请勿在 `negativeText` 提示中使用否定词。假如您不想在图像中包含镜像，请在 `negativeText` 提示中输入 **mirrors**，而不要输入 **no mirrors**。

Outpainting (Request)

```
{
  "taskType": "OUTPAINTING",
  "outPaintingParams": {
    "text": "string",
    "negativeText": "string",
    "image": "base64-encoded string",
    "maskPrompt": "string",
    "maskImage": "base64-encoded string",
    "outPaintingMode": "DEFAULT | PRECISE"
  },
  "imageGenerationConfig": {
    "numberOfImages": int,
    "height": int,
    "width": int,
    "cfgScale": float
  }
}
```

这些 `outPaintingParams` 字段的定义如下。掩膜用于定义图像中您不想修改的区域。生成操作将无缝扩展您定义的区域。

- `image` (必要) – 要修改的 JPEG 或 PNG 图像，格式化为指定像素序列的字符串，每个像素以 RGB 值定义并以 base64 编码。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- 必须定义以下字段之一（但不能同时定义两个字段）才能定义。
 - `maskPrompt` – 用于定义掩膜的文本提示。
 - `maskImage` – 一个字符串，通过指定与 `image` 大小相同的像素序列来定义掩膜。每个像素的 RGB 值都将变为 (0 0 0) (掩膜内的像素) 或 (255 255 255) (掩膜外的像素)。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- `text` (必要) – 用于定义要在掩膜外更改什么内容的文本提示。
- `negativeText` (可选) – 用于定义图像内不包含什么内容的文本提示。

Note

请勿在 `negativeText` 提示中使用否定词。假如您不想在图像中包含镜像，请在 `negativeText` 提示中输入 **mirrors**，而不要输入 **no mirrors**。

- `outPaintingMode`— 指定是否允许修改蒙版内的像素。以下是可能的值：
 - `DEFAULT` - 使用此选项来允许修改掩膜内的图像，使其与重建的背景保持一致。
 - `PRECISE` - 使用此选项来防止修改掩膜内的图像。

Image variation (Request)

```
{
  "taskType": "IMAGE_VARIATION",
  "imageVariationParams": {
    "text": "string",
    "negativeText": "string",
    "images": ["base64-encoded string"],
  },
  "imageGenerationConfig": {
    "numberOfImages": int,
    "height": int,
    "width": int,
    "cfgScale": float
  }
}
```

这些 `imageVariationParams` 字段的定义如下。

- `images` (必要) – 要生成变体的图片列表。目前，您只能包含一张图片。图像被定义为 base64 编码的图像字符串。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- `text` (可选) – 定义图像中要保留什么内容、要更改什么内容的文本提示。
- `negativeText` (可选) – 用于定义图像内不包含什么内容的文本提示。

Note

请勿在 `negativeText` 提示中使用否定词。假如您不想在图像中包含镜像，请在 `negativeText` 提示中输入 **mirrors**，而不要输入 **no mirrors**。

Response body

```
{
  "images": [
    "base64-encoded string",
    ...
  ],
  "error": "string"
}
```

响应正文是一个包含以下字段之一的流式传输对象。

- `images` – 如果请求成功，它将返回此字段，即一个 base64 编码的字符串列表，每个字符串都定义一个生成的图像。每个图像都被格式化为一个指定像素序列的字符串，其中每个像素都用 RGB 值定义并以 base64 编码。有关如何将图像编码为 base64 以及如何对 base64 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。
- `error` – 如果请求在以下任一情况下违反了内容审核政策，则将在此字段中返回一条消息。
 - 如果内容审核政策标记了输入的文本、图像或掩膜图片。
 - 如果内容审核政策至少标记了一张输出图片

共享且可选的 `imageGenerationConfig` 包含以下字段。如果不包含此对象，将使用默认配置。

- `numberOfImages` (可选) -要生成的图像数量。

最低	最高	默认
1	5	1

- `cfgScale` (可选) – 指定生成的图像应在多大程度上遵守提示。使用较低的值可在生成中引入更多的随机性。

最低	最高	默认
1.1	10.0	8.0

- 以下参数定义了您想要输出图像的大小。有关按图像大小定价的更多详细信息，请参阅 [Amazon Bedrock 定价](#)。
- height (可选) – 图像的高度 (以像素为单位)。默认值是 1024。
- width (可选) – 图像的宽度 (以像素为单位)。默认值是 1024。

允许使用以下尺寸。

宽度	高度	纵横比	价格相当于
1024	1024	1:1	1024 x 1024
768	768	1:1	512 x 512
512	512	1:1	512 x 512
768	1152	2:3	1024 x 1024
384	576	2:3	512 x 512
1152	768	3:2	1024 x 1024
576	384	3:2	512 x 512
768	1 280	3:5	1024 x 1024
384	640	3:5	512 x 512
1 280	768	5:3	1024 x 1024
640	384	5:3	512 x 512
896	1152	7:9	1024 x 1024
448	576	7:9	512 x 512

宽度	高度	纵横比	价格相当于
1152	896	9:7	1024 x 1024
576	448	9:7	512 x 512
768	1408	6:11	1024 x 1024
384	704	6:11	512 x 512
1408	768	11:6	1024 x 1024
704	384	11:6	512 x 512
640	1408	5:11	1024 x 1024
320	704	5:11	512 x 512
1408	640	11:5	1024 x 1024
704	320	11:5	512 x 512
1152	640	9:5	1024 x 1024
1173	640	16:9	1024 x 1024

- **seed** (可选) – 用于控制和重现结果。决定初始噪声设置。使用与上一次运行时相同的种子和相同的设置，以使推理可以创建相似的图像。

Note

只能为 TEXT_IMAGE 生成任务设置一个 seed。

最低	最高	默认
0	2,147,483,646	0

代码示例

以下示例演示如何在 Python 软件开发工具包中使用按需吞吐量调用 Amazon Titan Image Generator G1 模型。选择一个选项卡查看每个用例的示例。每个示例最后都显示了图像。

Text-to-image generation

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate an image from a text prompt with the Amazon Titan Image
Generator G1 model (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 Amazon Titan Image Generator G1"

    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 Amazon Titan Image Generator G1 model 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 model.
    """
```

```
logger.info(
    "Generating image with Amazon Titan Image Generator G1 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())

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

finish_reason = response_body.get("error")

if finish_reason is not None:
    raise ImageError(f"Image generation error. Error is {finish_reason}")

logger.info(
    "Successfully generated image with Amazon Titan Image Generator G1 model
%s", model_id)

return image_bytes

def main():
    """
    Entrypoint for Amazon Titan Image Generator G1 example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = 'amazon.titan-image-generator-v1'

    prompt = """A photograph of a cup of coffee from the side."""

    body = json.dumps({
        "taskType": "TEXT_IMAGE",
        "textToImageParams": {
```



```

        "text": prompt
    },
    "imageGenerationConfig": {
        "numberOfImages": 1,
        "height": 1024,
        "width": 1024,
        "cfgScale": 8.0,
        "seed": 0
    }
})

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 image with Amazon Titan Image Generator G1 model
{model_id}.")

if __name__ == "__main__":
    main()

```

Inpainting

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to use inpainting to generate an image from a source image with
the Amazon Titan Image Generator G1 model (on demand).
The example uses a mask prompt to specify the area to inpaint.

```

```
"""
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 Amazon Titan Image Generator G1"

    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 Amazon Titan Image Generator G1 model 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 model.
    """

    logger.info(
        "Generating image with Amazon Titan Image Generator G1 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())
```

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

finish_reason = response_body.get("error")

if finish_reason is not None:
    raise ImageError(f"Image generation error. Error is {finish_reason}")

logger.info(
    "Successfully generated image with Amazon Titan Image Generator G1 model
%s", model_id)

return image_bytes

def main():
    """
    Entrypoint for Amazon Titan Image Generator G1 example.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        model_id = 'amazon.titan-image-generator-v1'

        # Read image from file and encode it as base64 string.
        with open("/path/to/image", "rb") as image_file:
            input_image = base64.b64encode(image_file.read()).decode('utf8')

        body = json.dumps({
            "taskType": "INPAINTING",
            "inPaintingParams": {
                "text": "Modernize the windows of the house",
                "negativeText": "bad quality, low res",
                "image": input_image,
                "maskPrompt": "windows"
            },
            "imageGenerationConfig": {
                "numberOfImages": 1,
                "height": 512,
                "width": 512,
                "cfgScale": 8.0
            }
        })
```

```
    }
  })

  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 image with Amazon Titan Image Generator G1 model
        {model_id}.")

if __name__ == "__main__":
    main()
```

Outpainting

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to use outpainting to generate an image from a source image with
the Amazon Titan Image Generator G1 model (on demand).
The example uses a mask image to outpaint the original image.
"""

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 Amazon Titan Image Generator G1"

    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 Amazon Titan Image Generator G1 model 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 model.
    """

    logger.info(
        "Generating image with Amazon Titan Image Generator G1 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())

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

    finish_reason = response_body.get("error")

    if finish_reason is not None:
        raise ImageError(f"Image generation error. Error is {finish_reason}")
```

```
    logger.info(
        "Successfully generated image with Amazon Titan Image Generator G1 model
%s", model_id)

    return image_bytes

def main():
    """
    Entrypoint for Amazon Titan Image Generator G1 example.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        model_id = 'amazon.titan-image-generator-v1'

        # Read image and mask image from file and encode as base64 strings.
        with open("/path/to/image", "rb") as image_file:
            input_image = base64.b64encode(image_file.read()).decode('utf8')
        with open("/path/to/mask_image", "rb") as mask_image_file:
            input_mask_image = base64.b64encode(
                mask_image_file.read()).decode('utf8')

        body = json.dumps({
            "taskType": "OUTPAINTING",
            "outPaintingParams": {
                "text": "Draw a chocolate chip cookie",
                "negativeText": "bad quality, low res",
                "image": input_image,
                "maskImage": input_mask_image,
                "outPaintingMode": "DEFAULT"
            },
            "imageGenerationConfig": {
                "numberOfImages": 1,
                "height": 512,
                "width": 512,
                "cfgScale": 8.0
            }
        })

        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 image with Amazon Titan Image Generator G1 model
{model_id}.")

if __name__ == "__main__":
    main()
```

Image variation

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate an image variation from a source image with the
Amazon Titan Image Generator G1 model (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 Amazon Titan Image Generator G1"
```

```
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 Amazon Titan Image Generator G1 model 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 model.
    """

    logger.info(
        "Generating image with Amazon Titan Image Generator G1 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())

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

    finish_reason = response_body.get("error")

    if finish_reason is not None:
        raise ImageError(f"Image generation error. Error is {finish_reason}")

    logger.info(
        "Successfully generated image with Amazon Titan Image Generator G1 model
        %s", model_id)
```



```
return image_bytes

def main():
    """
    Entrypoint for Amazon Titan Image Generator G1 example.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        model_id = 'amazon.titan-image-generator-v1'

        # Read image from file and encode it as base64 string.
        with open("/path/to/image", "rb") as image_file:
            input_image = base64.b64encode(image_file.read()).decode('utf8')

        body = json.dumps({
            "taskType": "IMAGE_VARIATION",
            "imageVariationParams": {
                "text": "Modernize the house, photo-realistic, 8k, hdr",
                "negativeText": "bad quality, low resolution, cartoon",
                "images": [input_image],
            },
            "imageGenerationConfig": {
                "numberOfImages": 1,
                "height": 512,
                "width": 512,
                "cfgScale": 8.0
            }
        })

        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 occured: " +
              format(message))
    except ImageError as err:
        logger.error(err.message)
```

```
        print(err.message)

    else:
        print(
            f"Finished generating image with Amazon Titan Image Generator G1 model
            {model_id}.")

if __name__ == "__main__":
    main()
```

Amazon Titan Embeddings G1 - Text

Titan Embeddings G1 - Text不支持使用推理参数。以下各节详细介绍了请求和响应格式，并提供了一个代码示例。

主题

- [请求和回应](#)
- [代码示例](#)

请求和回应

请求正文在[InvokeModel](#)请求的body字段中传递。

Request

唯一可用的字段是，您可以在其中包含要转换为嵌入的文本。

```
{
  "inputText": string
}
```

Response

响应body的包含以下字段。

```
{
  "embedding": [float, float, ...],
  "inputTextTokenCount": int
}
```

```
}
```

字段如下所述。

- `embedding` — 一个数组，它表示您提供的输入的嵌入向量。
- `inputTextTokenCount` - 输入中的令牌数量。

代码示例

此示例说明如何调用 Amazon Titan Embeddings G1 - Text 模型来生成嵌入内容。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate embeddings with the Amazon Titan Embeddings G1 - Text model (on
demand).
"""

import json
import logging
import boto3

from botocore.exceptions import ClientError

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

def generate_embeddings(model_id, body):
    """
    Generate a vector of embeddings for a text input using Amazon Titan Embeddings G1 -
    Text on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        response (JSON): The text that the model generated, token information, and the
        reason the model stopped generating text.
    """
```

```
logger.info("Generating embeddings with Amazon Titan Embeddings G1 - Text 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())

return response_body

def main():
    """
    Entrypoint for Amazon Titan Embeddings G1 - Text example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = "amazon.titan-embed-text-v1"
    input_text = "What are the different services that you offer?"

    # Create request body.
    body = json.dumps({
        "inputText": input_text,
    })

    try:

        response = generate_embeddings(model_id, body)

        print(f"Generated embeddings: {response['embedding']}")
        print(f"Input Token count: {response['inputTextTokenCount']}")

    except ClientError as err:
        message = err.response["Error"]["Message"]
```

```
logger.error("A client error occurred: %s", message)
print("A client error occurred: " +
      format(message))

else:
    print(f"Finished generating embeddings with Amazon Titan Embeddings G1 - Text
model {model_id}.")

if __name__ == "__main__":
    main()
```

Amazon Titan Multimodal Embeddings G1

本节提供请求和响应正文格式以及使用 Amazon Titan Multimodal Embeddings G1 的代码示例。

主题

- [请求和回应](#)
- [代码示例](#)

请求和回应

请求正文在[InvokeModel](#)请求的body字段中传递。

Request

Amazon 的请求正文Titan Multimodal Embeddings G1包括以下字段。

```
{
  "inputText": string,
  "inputImage": base64-encoded string,
  "embeddingConfig": {
    "outputEmbeddingLength": 256 | 384 | 1024
  }
}
```

以下字段中至少有一个是必填字段。同时包含两者以生成对生成的文本嵌入求平均值的嵌入向量和图像嵌入向量。

- InputText-输入要转换为嵌入的文本。

- `inputImage` — 在 `base64` 中对要转换为嵌入的图像进行编码，然后在此字段中输入字符串。有关如何将图像编码为 `base64` 以及如何对 `base64` 编码的字符串进行解码并将其转换为图像的示例，请参阅[代码示例](#)。

以下字段是可选字段。

- `embeddingConfig` — 包含一个 `outputEmbeddingLength` 字段，在该字段中，您可以为输出嵌入向量指定以下长度之一。
 - 256
 - 384
 - 1024 (默认值)

Response

响应 `body` 的包含以下字段。

```
{
  "embedding": [float, float, ...],
  "inputTextTokenCount": int,
  "message": string
}
```

字段如下所述。

- `embedding` — 一个数组，它表示您提供的输入的嵌入向量。
- `inputTextTokenCount` - 文本输入中的标记数。
- `message` - 指定生成过程中发生的任何错误。

代码示例

以下示例演示如何在 Python 软件开发工具包中使用按需吞吐量调用 Amazon Titan Multimodal Embeddings G1 模型。选择一个选项卡查看每个用例的示例。

Text embeddings

此示例说明如何调用 Amazon Titan Multimodal Embeddings G1 模型来生成文本嵌入。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate embeddings from text with the Amazon Titan Multimodal
Embeddings G1 model (on demand).
"""

import json
import logging
import boto3

from botocore.exceptions import ClientError

class EmbedError(Exception):
    "Custom exception for errors returned by Amazon Titan Multimodal Embeddings G1"

    def __init__(self, message):
        self.message = message

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

def generate_embeddings(model_id, body):
    """
    Generate a vector of embeddings for a text input using Amazon Titan Multimodal
    Embeddings G1 on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        response (JSON): The embeddings that the model generated, token information,
    and the
        reason the model stopped generating embeddings.
    """

    logger.info("Generating embeddings with Amazon Titan Multimodal Embeddings G1
    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())

finish_reason = response_body.get("message")

if finish_reason is not None:
    raise EmbedError(f"Embeddings generation error: {finish_reason}")

return response_body

def main():
    """
    Entrypoint for Amazon Titan Multimodal Embeddings G1 example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = "amazon.titan-embed-image-v1"
    input_text = "What are the different services that you offer?"
    output_embedding_length = 256

    # Create request body.
    body = json.dumps({
        "inputText": input_text,
        "embeddingConfig": {
            "outputEmbeddingLength": output_embedding_length
        }
    })

    try:

        response = generate_embeddings(model_id, body)

        print(f"Generated text embeddings of length {output_embedding_length}:
        {response['embedding']}")
        print(f"Input text token count: {response['inputTextTokenCount']}")

    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 EmbedError as err:
    logger.error(err.message)
    print(err.message)

else:
    print(f"Finished generating text embeddings with Amazon Titan Multimodal
Embeddings G1 model {model_id}.")

if __name__ == "__main__":
    main()
```

Image embeddings

此示例说明如何调用 Amazon Titan Multimodal Embeddings G1 模型来生成图像嵌入。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate embeddings from an image with the Amazon Titan Multimodal
Embeddings G1 model (on demand).
"""

import base64
import json
import logging
import boto3

from botocore.exceptions import ClientError

class EmbedError(Exception):
    "Custom exception for errors returned by Amazon Titan Multimodal Embeddings G1"

    def __init__(self, message):
        self.message = message

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

```
def generate_embeddings(model_id, body):
    """
    Generate a vector of embeddings for an image input using Amazon Titan Multimodal
    Embeddings G1 on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        response (JSON): The embeddings that the model generated, token information,
        and the
        reason the model stopped generating embeddings.
    """

    logger.info("Generating embeddings with Amazon Titan Multimodal Embeddings G1
    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())

    finish_reason = response_body.get("message")

    if finish_reason is not None:
        raise EmbedError(f"Embeddings generation error: {finish_reason}")

    return response_body

def main():
    """
    Entrypoint for Amazon Titan Multimodal Embeddings G1 example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
```

```
# Read image from file and encode it as base64 string.
with open("/path/to/image", "rb") as image_file:
    input_image = base64.b64encode(image_file.read()).decode('utf8')

model_id = 'amazon.titan-embed-image-v1'
output_embedding_length = 256

# Create request body.
body = json.dumps({
    "inputImage": input_image,
    "embeddingConfig": {
        "outputEmbeddingLength": output_embedding_length
    }
})

try:

    response = generate_embeddings(model_id, body)

    print(f"Generated image embeddings of length {output_embedding_length}:
{response['embedding']}")

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 EmbedError as err:
    logger.error(err.message)
    print(err.message)

else:
    print(f"Finished generating image embeddings with Amazon Titan Multimodal
Embeddings G1 model {model_id}.")

if __name__ == "__main__":
    main()
```

Text and image embeddings

此示例说明如何调用 Amazon Titan Multimodal Embeddings G1 模型，通过组合的文本和图像输入生成嵌入内容。生成的向量是生成的文本嵌入向量和图像嵌入向量的平均值。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate embeddings from an image and accompanying text with the Amazon
Titan Multimodal Embeddings G1 model (on demand).
"""

import base64
import json
import logging
import boto3

from botocore.exceptions import ClientError

class EmbedError(Exception):
    "Custom exception for errors returned by Amazon Titan Multimodal Embeddings G1"

    def __init__(self, message):
        self.message = message

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

def generate_embeddings(model_id, body):
    """
    Generate a vector of embeddings for a combined text and image input using Amazon
    Titan Multimodal Embeddings G1 on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        response (JSON): The embeddings that the model generated, token information,
        and the
        reason the model stopped generating embeddings.
    """

    logger.info("Generating embeddings with Amazon Titan Multimodal Embeddings G1
    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())

finish_reason = response_body.get("message")

if finish_reason is not None:
    raise EmbedError(f"Embeddings generation error: {finish_reason}")

return response_body

def main():
    """
    Entrypoint for Amazon Titan Multimodal Embeddings G1 example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = "amazon.titan-embed-image-v1"
    input_text = "A family eating dinner"
    # Read image from file and encode it as base64 string.
    with open("/path/to/image", "rb") as image_file:
        input_image = base64.b64encode(image_file.read()).decode('utf8')
    output_embedding_length = 256

    # Create request body.
    body = json.dumps({
        "inputText": input_text,
        "inputImage": input_image,
        "embeddingConfig": {
            "outputEmbeddingLength": output_embedding_length
        }
    })
```

```
try:

    response = generate_embeddings(model_id, body)

    print(f"Generated embeddings of length {output_embedding_length}:
{response['embedding']}")
    print(f"Input text token count: {response['inputTextTokenCount']}")

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 EmbedError as err:
    logger.error(err.message)
    print(err.message)

else:
    print(f"Finished generating embeddings with Amazon Titan Multimodal
Embeddings G1 model {model_id}.")

if __name__ == "__main__":
    main()
```

AnthropicClaude模型

本节提供推理参数和使用AnthropicClaude模型的代码示例。

您可以使用 Amazon Bedrock 发送[AnthropicClaude文字补全 API](#)或[AnthropicClaude消息 API](#)推断请求。

您可以使用消息 API 来创建对话应用程序，例如虚拟助手或指导应用程序。将文本补全 API 用于单圈文本生成应用程序。例如，为博客文章生成文本或摘要用户提供的文本。

您可以使用[InvokeModel](#)或 [InvokeModelWithResponseStream](#)（流式传输）向AnthropicClaude模型发出推理请求。您需要获得希望使用的模型的模型 ID。要获取模型的AnthropicClaude模型 ID，请参阅[Amazon Bedrock 基本模型 ID（按需吞吐量）](#)和[用于购买预配置吞吐量的 Amazon Bedrock 基本型号](#)。

Note

要在推理调用中使用系统提示，必须使用 2.1 Anthropic Claude 版或AnthropicClaude 3 Sonnet。有关创建系统提示的信息，请参阅AnthropicClaude文档中的 <https://docs.anthropic.com/claude/docs/how-to-use-system-prompts>。

为避免在 2.1 Anthropic Claude 版本中出现超时，我们建议将prompt字段中的输入令牌数量限制为 180K。我们希望尽快解决这个超时问题。

在推理调用中，在body字段中填入符合您要进行的类型调用的 JSON 对象，[AnthropicClaude文字补全 API](#)或。[AnthropicClaude消息 API](#)

有关为AnthropicClaude模型创建提示的信息，请参阅AnthropicClaude文档中的[提示设计简介](#)。

主题

- [AnthropicClaude文字补全 API](#)
- [AnthropicClaude消息 API](#)

AnthropicClaude文字补全 API

本节提供了通过文本补全 API 使用AnthropicClaude模型的推理参数和代码示例。

主题

- [AnthropicClaude文字补全 API 概述](#)
- [支持的型号](#)
- [请求和响应](#)
- [代码示例](#)

AnthropicClaude文字补全 API 概述

使用文本完成 API 根据用户提供的提示单圈生成文本。例如，您可以使用文本补全 API 为博客文章生成文本或汇总用户输入的文本。

有关为AnthropicClaude模型创建提示的信息，请参阅[提示设计简介](#)。如果要在中使用现有的文本完成提示[AnthropicClaude消息 API](#)，请参阅[从文本完成迁移](#)。

支持的型号

您可以将文本补全 API 与以下AnthropicClaude模型一起使用。

- AnthropicClaudeInstantv1.2
- AnthropicClaudev2
- AnthropicClaudev2.1

请求和响应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。

有关更多信息，请参阅AnthropicClaude文档中的 https://docs.anthropic.com/claude/reference/complete_post。

Request

AnthropicClaude具有以下用于文本完成推理调用的推理参数。

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

以下是必要参数。

- `prompt` — (必填) 你想让 Claude 完成的提示。为了正确生成响应，您需要使用交替
`\n\nHuman:`和
`\n\nAssistant:`对话回合来格式化提示。例如：

```
"\n\nHuman: {userQuestion}\n\nAssistant:"
```

有关更多信息，请参阅AnthropicClaude文档中的[提示验证](#)。

- `max_tokens_to_sample` — (必填) 停止前要生成的最大代币数量。为了获得最佳性能，我们建议限制为 4000 个令牌。

请注意，AnthropicClaude模型可能会在达到的值之前停止生成代币max_tokens_to_sample。不同的AnthropicClaude模型对此参数有不同的最大值。有关更多信息，请参阅AnthropicClaude文档中的[模型比较](#)。

默认	最低	最高
200	0	4096

以下是可选参数。

- stop_sequences — (可选) 将导致模型停止生成的序列。

AnthropicClaude模型会停止开启"\n\nHuman:"，并且将来可能会包括其他内置的停止序列。使用stop_sequences推理参数包括其他字符串，这些字符串将向模型发出停止生成文本的信号。

- 温度- (可选) 注入响应的随机性量。

默认值为 1。范围从 0 到 1。对于分析/多项选择，使用接近0的温度，对于创造性和生成性任务，使用接近1的温度。

默认	最低	最高
0.5	0	4096

- top_p — (可选) 使用原子核采样。

在 nucleus 采样中，按概率递减顺序AnthropicClaude计算每个后续代币的所有选项的累积分布，并在达到由指定的特定概率后将其切断。top_p您应该更改其中一个temperature或top_p，但不能同时更改两个。

默认	最低	最高
1	0	1

- top_k — (可选) 仅从每个后续令牌的前 K 个选项中取样。

用于top_k移除长尾低概率响应。

默认	最低	最高
250	0	500

Response

该AnthropicClaude模型返回文本完成推理调用的以下字段。

```
{
  "completion": string,
  "stop_reason": string,
  "stop": string
}
```

- `completion` — 不包括停止序列的结果完成。
- `stop_reason` — 模型停止生成响应的原因。
 - “`stop_sequence`” — 模型到达了停止序列——要么由你提供的`stop_sequences`推理参数，要么是模型中内置的停止序列。
 - “`max_tokens`” — 超出模型`max_tokens_to_sample`或模型的最大代币数量。
- `stop` — 如果指定`stop_sequences`推理参数，则`stop`包含指示模型停止生成文本的停止序列。例如，`holes`在以下响应中。

```
{
  "completion": " Here is a simple explanation of black ",
  "stop_reason": "stop_sequence",
  "stop": "holes"
}
```

如果未指定`stop_sequences`，则的值`stop`为空。

代码示例

这些示例显示了如何使用按需吞吐量调用 AnthropicClaudeV2 模型。要使用AnthropicClaude版本 2.1，请将的值更改`modelId`为`anthropic.claude-v2:1`。

```
import boto3
```

```
import json
brt = 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 = 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'))
```

以下示例展示如何在 Python 中使用提示 *write an essay for living on mars in 1000 words* 和 Anthropic Claude V2 模型生成流式传输文本：

```
import boto3
import json

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

body = json.dumps({
    'prompt': '\n\nHuman: write an essay for living on mars in 1000 words\n\nAssistant:',
    'max_tokens_to_sample': 4000
})

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()))
```

AnthropicClaude消息 API

本节提供推理参数和使用AnthropicClaude消息 API 的代码示例。

主题

- [AnthropicClaude消息 API 概述](#)
- [支持的型号](#)
- [请求和响应](#)
- [代码示例](#)

AnthropicClaude消息 API 概述

您可以使用 Messages API 来创建聊天机器人或虚拟助手应用程序。API 管理用户和AnthropicClaude 模型 (助手) 之间的对话交流。

Anthropic训练 Claude 模型在交替的用户和助手对话回合中进行操作。创建新消息时，您可以使用 messages 参数指定之前的对话回合。然后，模型在对话中生成下一条消息。

每条输入消息都必须是一个具有角色和内容的对象。您可以指定一条用户角色消息，也可以包含多条用户和助手消息。第一条消息必须始终使用用户角色。

如果你使用的是预填回复的技巧Claude (使用最后一个助手角色 Message 填写 Claude 回复的开头)，则Claude会从你上次停下来的地方继续回复。使用这种技巧，仍然Claude会返回带有助手角色的响应。

如果最后一条消息使用助理角色，则响应内容将立即从该消息中的内容继续。你可以用它来限制模型的部分响应。

单个用户消息的示例：

```
[{"role": "user", "content": "Hello, Claude"}]
```

具有多个对话回合的示例：

```
[
  {"role": "user", "content": "Hello there."},
  {"role": "assistant", "content": "Hi, I'm Claude. How can I help you?"},
  {"role": "user", "content": "Can you explain LLMs in plain English?"},
]
```

以 Claude 的部分填充回复为例：

```
[
  {"role": "user", "content": "Please describe yourself using only JSON"},
  {"role": "assistant", "content": "Here is my JSON description:\n{"},
]
```

每个输入消息内容可以是单个字符串，也可以是内容块数组，其中每个块都有特定的类型。使用字符串是由“文本”类型的一个内容块组成的数组的简写。以下输入消息是等效的：

```
{"role": "user", "content": "Hello, Claude"}
```

```
{"role": "user", "content": [{"type": "text", "text": "Hello, Claude"}]}
```

有关为AnthropicClaude模型创建提示的信息，请参阅[文档中的提示简介](#)。Anthropic Claude如果您想要迁移到消息 API 的现有[文本完成](#)提示，请参阅[从文本完成迁移](#)。

系统提示

您还可以在请求中加入系统提示。系统提示允许您为其提供上下文和说明 AnthropicClaude，例如指定特定的目标或角色。在system字段中指定系统提示符，如以下示例所示。

```
"system": "You are Claude, an AI assistant created by Anthropic to be helpful,
           harmless, and honest. Your goal is to provide informative and
           substantive responses
           to queries while avoiding potential harms."
```

有关更多信息，请参阅Anthropic文档中的[系统提示](#)。

多式联运提示

多模式提示将多种模式（图像和文本）组合到一个提示中。您可以在content输入字段中指定模式。以下示例显示了AnthropicClaude如何要求描述所提供图片的内容。有关代码示例，请参阅[多式联运代码示例](#)。

```
{
  "anthropic_version": "bedrock-2023-05-31",
  "max_tokens": 1024,
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "image",
          "source": {
            "type": "base64",
            "media_type": "image/jpeg",
            "data": "iVBORw..."
          }
        },
        {
          "type": "text",
          "text": "What's in these images?"
        }
      ]
    }
  ]
}
```

您最多可以向模型提供 20 张图像。你不能将图片放在助手角色中。

您在请求中包含的每张图片都计入您的令牌使用量。有关更多信息，请参阅Anthropic文档中的[图像成本](#)。

支持的型号

您可以将消息 API 与以下AnthropicClaude模型配合使用。

- AnthropicClaudeInstantv1.2
- AnthropicClaude2 v2
- AnthropicClaude2 v2.1
- Anthropic Claude 3 Sonnet
- Anthropic Claude 3 Haiku

请求和响应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。您可以在请求中发送的最大负载大小为 20MB。

欲了解更多信息，请参阅 https://docs.anthropic.com/claude/reference/messages_post。

Request

AnthropicClaude具有以下用于消息推理调用的推理参数。

```
{
  "anthropic_version": "bedrock-2023-05-31",
  "max_tokens": int,
  "system": string,
  "messages": [
    {
      "role": string,
      "content": [
        { "type": "image", "source": { "type": "base64", "media_type": "image/jpeg", "data": "content image bytes" } },
        { "type": "text", "text": "content text" }
      ]
    }
  ],
  "temperature": float,
  "top_p": float,
  "top_k": int,
  "stop_sequences": [string]
}
```

以下是必要参数。

- anthropic_version — (必填) 人类版本。值必须为 bedrock-2023-05-31。
- max_tokens — (必填) 停止前要生成的最大令牌数。

请注意，AnthropicClaude模型可能会在达到的值之前停止生成代币max_tokens。不同的AnthropicClaude模型对此参数有不同的最大值。有关更多信息，请参阅[模型比较](#)。

- 消息- (必填) 输入消息。
 - 角色 — 对话的角色转向。有效值为 user 和 assistant。
 - 内容 — (必填) 对话的内容转向。

- `type` — (必填) 内容的类型。有效值为 `image` 和 `text`。

如果指定 `image`，则还必须按以下格式指定图像源

`source` — (必填) 对话内容转向。

- `type` — (必填) 图像的编码类型。您可以指定 `base64`。
- `media_type` — (必填) 图像的类型。您可以指定以下图像格式。
 - `image/jpeg`
 - `image/png`
 - `image/webp`
 - `image/gif`
- `data` — (必填) 图像的 `base64` 编码图像字节。最大图像大小为 3.75MB。图像的最大高度和宽度为 8000 像素。

如果指定 `text`，则还必须在中指定提示 `text`。

以下是可选参数。

- `system` — (可选) 请求的系统提示符。

系统提示是向其提供上下文和说明的一种方式 AnthropicClaude，例如指定特定的目标或角色。有关更多信息，请参阅Anthropic文档中的[如何使用系统提示](#)。

Note

您可以在 2.1 或更高AnthropicClaude版本中使用系统提示。

- `stop_sequences` — (可选) 导致模型停止生成的自定义文本序列。AnthropicClaude模型通常在自然完成回合后停止，在本例中，`stop_reason`响应字段的值为`end_turn`。如果您希望模型在遇到自定义文本字符串时停止生成，则可以使用`stop_sequences`参数。如果模型遇到其中一个自定义文本字符串，则`stop_reason`响应字段的值为`stop_sequence`，的值`stop_sequence`包含匹配的停止序列。

最大条目数为 8191。

- `temperature` — (可选) 注入响应的随机性量。

默认	最低	最高
1	0	1

- top_p — (可选) 使用原子核采样。

在 nucleus 采样中，按概率递减顺序AnthropicClaude计算每个后续代币的所有选项的累积分布，并在达到由指定的特定概率后将其切断。top_p您应该更改其中一个temperature或top_p，但不能同时更改两个。

默认	最低	最高
0.999	0	1

以下是可选参数。

- top_k — (可选) 仅从每个后续令牌的前 K 个选项中取样。

用于top_k移除长尾低概率响应。

默认	最低	最高
默认情况下处于禁用状态	0	100,000,000

Response

该AnthropicClaude模型为消息推理调用返回以下字段。

```
{
  "id": string,
  "model": string,
  "type": "message",
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": string
    }
  ]
}
```

```

    }
  ],
  "stop_reason": string,
  "stop_sequence": string,
  "usage": {
    "input_tokens": integer,
    "output_tokens": integer
  }
}

```

- `id` — 响应的唯一标识符。身份证的格式和长度可能会随着时间的推移而改变。
- `模型`-发出请求的Anthropic Claude模型的 ID。
- `stop_reason` — Anthropic Claude 停止生成响应的原因。
 - `end_turn` — 模型到达了自然停止点
 - `max_tokens` — 生成的文本超过了`max_tokens`输入字段的值或超过了模型支持的最大标记数。
 - `stop_sequence` — 模型生成了您在`stop_sequences`输入字段中指定的一个停止序列。
- `类型`-响应的类型。此值始终为 `message`。
- `角色`-生成的消息的对话角色。此值始终为 `assistant`。
- `内容`-模型生成的内容。以数组形式返回。
 - `类型`-内容的类型。目前唯一支持的值是 `text`。
 - `文本`-内容的文本。
- `用法` — 容器，用于存放您在请求中提供的令牌数量以及模型在响应中生成的令牌数量。
 - `input_tokens` — 请求中输入令牌的数量。
 - `output_tokens` — 模型在响应中生成的标记数。
 - `stop_sequence` — 模型生成了您在`stop_sequences`输入字段中指定的一个停止序列。

代码示例

以下代码示例展示了如何使用消息 API。

主题

- [消息代码示例](#)
- [多式联运代码示例](#)

消息代码示例

此示例说明如何向AnthropicClaude 3 Sonnet模型发送单回合用户消息和带有预填助手消息的用户回合。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate a message with Anthropic Claude (on demand).
"""
import boto3
import json
import logging

from botocore.exceptions import ClientError

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

def generate_message(bedrock_runtime, model_id, system_prompt, messages, max_tokens):

    body=json.dumps(
        {
            "anthropic_version": "bedrock-2023-05-31",
            "max_tokens": max_tokens,
            "system": system_prompt,
            "messages": messages
        }
    )

    response = bedrock_runtime.invoke_model(body=body, modelId=model_id)
    response_body = json.loads(response.get('body').read())

    return response_body

def main():
    """
    Entrypoint for Anthropic Claude message example.
    """

    try:
```

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

model_id = 'anthropic.claude-3-sonnet-20240229-v1:0'
system_prompt = "Please respond only with emoji."
max_tokens = 1000

# Prompt with user turn only.
user_message = {"role": "user", "content": "Hello World"}
messages = [user_message]

response = generate_message (bedrock_runtime, model_id, system_prompt,
messages, max_tokens)
print("User turn only.")
print(json.dumps(response, indent=4))

# Prompt with both user turn and prefilled assistant response.
#Anthropic Claude continues by using the prefilled assistant text.
assistant_message = {"role": "assistant", "content": "<emoji>"}
messages = [user_message, assistant_message]
response = generate_message(bedrock_runtime, model_id,system_prompt, messages,
max_tokens)
print("User turn and prefilled assistant response.")
print(json.dumps(response, indent=4))

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

if __name__ == "__main__":
    main()
```

多式联运代码示例

以下示例说明如何将多式联运消息中的图像和提示文本传递给AnthropicClaude 3 Sonnet模型。

主题

- [多式联运提示 InvokeModel](#)
- [使用直播多模式提示音 InvokeModelWithResponseStream](#)

多式联运提示 InvokeModel

以下示例说明如何向 with 发送多式联运AnthropicClaude 3 Sonnet提示。 [InvokeModel](#)

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to run a multimodal prompt with Anthropic Claude (on demand) and InvokeModel.
"""

import json
import logging
import base64
import boto3

from botocore.exceptions import ClientError

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

def run_multi_modal_prompt(bedrock_runtime, model_id, messages, max_tokens):
    """
    Invokes a model with a multimodal prompt.
    Args:
        bedrock_runtime: The Amazon Bedrock boto3 client.
        model_id (str): The model ID to use.
        messages (JSON) : The messages to send to the model.
        max_tokens (int) : The maximum number of tokens to generate.
    Returns:
        None.
    """

    body = json.dumps(
        {
            "anthropic_version": "bedrock-2023-05-31",
            "max_tokens": max_tokens,
            "messages": messages
        }
    )
```

```
response = bedrock_runtime.invoke_model(
    body=body, modelId=model_id)
response_body = json.loads(response.get('body').read())

return response_body

def main():
    """
    Entrypoint for Anthropic Claude multimodal prompt example.
    """

    try:

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

        model_id = 'anthropic.claude-3-sonnet-20240229-v1:0'
        max_tokens = 1000
        input_image = "/path/to/image"
        input_text = "What's in this image?"

        # Read reference image from file and encode as base64 strings.
        with open(input_image, "rb") as image_file:
            content_image = base64.b64encode(image_file.read()).decode('utf8')

        message = {"role": "user",
            "content": [
                {"type": "image", "source": {"type": "base64",
                    "media_type": "image/jpeg", "data": content_image}},
                {"type": "text", "text": input_text}
            ]}

        messages = [message]

        response = run_multi_modal_prompt(
            bedrock_runtime, model_id, messages, max_tokens)
        print(json.dumps(response, indent=4))

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

```
format(message))
```

```
if __name__ == "__main__":  
    main()
```

使用直播多模式提示音 InvokeModelWithResponseStream

以下示例说明如何将发送到的多式联运提示中的响应流AnthropicClaude 3 Sonnet 式[InvokeModelWithResponseStream](#)传输。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
"""  
Shows how to stream the response from Anthropic Claude Sonnet (on demand) for a  
multimodal request.  
"""  
  
import json  
import base64  
import logging  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
logging.basicConfig(level=logging.INFO)  
  
def stream_multi_modal_prompt(bedrock_runtime, model_id, input_text, image,  
    max_tokens):  
    """  
    Streams the response from a multimodal prompt.  
    Args:  
        bedrock_runtime: The Amazon Bedrock boto3 client.  
        model_id (str): The model ID to use.  
        input_text (str) : The prompt text  
        image (str) : The path to an image that you want in the prompt.  
        max_tokens (int) : The maximum number of tokens to generate.  
    Returns:  
        None.  
    """
```

```

with open(image, "rb") as image_file:
    encoded_string = base64.b64encode(image_file.read())

body = json.dumps({
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": max_tokens,
    "messages": [
        {
            "role": "user",
            "content": [
                {"type": "text", "text": input_text},
                {"type": "image", "source": {"type": "base64",
                                           "media_type": "image/jpeg", "data":
encoded_string.decode('utf-8')}}
            ]
        }
    ]
})

response = bedrock_runtime.invoke_model_with_response_stream(
    body=body, modelId=model_id)

for event in response.get("body"):
    chunk = json.loads(event["chunk"]["bytes"])

    if chunk['type'] == 'message_delta':
        print(f"\nStop reason: {chunk['delta']['stop_reason']}")
        print(f"Stop sequence: {chunk['delta']['stop_sequence']}")
        print(f"Output tokens: {chunk['usage']['output_tokens']}")

    if chunk['type'] == 'content_block_delta':
        if chunk['delta']['type'] == 'text_delta':
            print(chunk['delta']['text'], end="")

def main():
    """
    Entrypoint for Anthropic Claude Sonnet multimodal prompt example.
    """

    model_id = "anthropic.claude-3-sonnet-20240229-v1:0"
    input_text = "What can you tell me about this image?"
    image = "/path/to/image"
    max_tokens = 100

```



```
try:

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

    stream_multi_modal_prompt(
        bedrock_runtime, model_id, input_text, image, max_tokens)

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

if __name__ == "__main__":
    main()
```

AI21 LabsJurassic-2模型

本节提供推理参数和使用AI21 LabsAI21 LabsJurassic-2模型的代码示例。

主题

- [推理参数](#)
- [代码示例](#)

推理参数

这些AI21 LabsJurassic-2模型支持以下推理参数。

主题

- [随机性和多样性](#)
- [Length](#)
- [重复](#)
- [模型调用请求正文字段](#)
- [模型调用响应正文字段](#)

随机性和多样性

这些AI21 LabsJurassic-2模型支持以下参数来控制响应的随机性和多样性。

- Temperature (temperature) – 使用较低的值可降低响应中的随机性。
- Top P (topP) – 使用较低的值可忽略不太可能的选项。

Length

这些AI21 LabsJurassic-2模型支持以下参数来控制生成的响应的长度。

- Max completion length (maxTokens) – 指定生成的响应中的最大词元数。
- Stop sequences (stopSequences) – 配置模型可识别的停止序列，遇到停止序列后，模型停止生成更多词元。按 Enter 键可在停止序列中插入换行符。使用 Tab 键结束对停止序列的插入。

重复

这些AI21 LabsJurassic-2模型支持以下参数来控制生成的响应中的重复性。

- Presence penalty (presencePenalty) – 使用较高的值可降低生成的新词元已在提示或完成内容中至少出现过一次的概率。
- Count penalty (countPenalty) – 使用较高的值可降低生成的新词元已在提示或完成内容中至少出现过一次的概率。与出现次数成正比。
- Frequency penalty (frequencyPenalty) – 使用较高的值可降低生成的新词元已在提示或完成内容中至少出现过一次的概率。此值与词元出现的频率成正比（标准化为文本长度）。
- Penalize special tokens — 降低特殊字符重复出现的概率。默认值为 true。
 - Whitespaces (applyToWhitespaces) – true 值会将惩罚应用于空格和新行。
 - Punctuations (applyToPunctuation) – true 值会将惩罚应用于标点。
 - Numbers (applyToNumbers) – true 值会将惩罚应用于数字。
 - Stop words (applyToStopwords) – true 值会将惩罚应用于停顿词。
 - Emojis (applyToEmojis) – true 值且将表情符号排除在惩罚范围之外。

模型调用请求正文字段

使用AI21 Labs模型进行[InvokeModel](#)或[InvokeModelWithResponseStream](#)调用时，请在body字段中填入符合以下对象的JSON对象。在prompt字段中输入提示。

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

要惩罚特殊词元，请将这些字段添加到任何惩罚对象中。例如，您可以按如下方式修改 countPenalty 字段。

```
"countPenalty": {
  "scale": float,
  "applyToWhitespaces": boolean,
  "applyToPunctuations": boolean,
  "applyToNumbers": boolean,
  "applyToStopwords": boolean,
  "applyToEmojis": boolean
}
```

下表显示了数字参数的最小值、最大值和默认值。

类别	参数	JSON 对象格式	最低	最高	默认
随机性和多样性	Temperature	温度	0	1	0.5
	Top P	topP	0	1	0.5
Length	Max tokens (mid, ultra,	maxTokens	0	8,191	200

类别	参数	JSON 对象格式	最低	最高	默认
	and large models)				
	Max tokens (other models)		0	2,048	200
重复	Presence penalty	presencePenalty	0	5	0
	Count penalty	countPenalty	0	1	0
	Frequency penalty	frequencyPenalty	0	500	0

模型调用响应正文字段

有关响应中的 body 字段的格式的信息，请参阅 <https://docs.ai21.com/reference/j2-complete-ref>。

Note

Amazon Bedrock 以整数值的形式返回响应标识符 (id)。

代码示例

此示例说明如何调用 A2I 模型 AI21 Labs Jurassic-2 Mid。

```
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模型

以下是 Amazon Bedrock 支持的Cohere模型的推理参数信息。

主题

- [CohereCommand模型](#)
- [CohereEmbed模型](#)

CohereCommand模型

您可以使用[InvokeModel](#)或 [InvokeModelWithResponseStream](#) (流式传输) 向CohereCommand模型发出推理请求。您需要获得希望使用的模型的模型 ID。要获取模型 ID，请参阅[亚马逊 Bedrock 型号](#)。

主题

- [请求和响应](#)
- [代码示例](#)

请求和响应

Request

这些CohereCommand模型具有以下推理参数。

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

以下是必要参数。

- `prompt` — (必填) 作为生成响应起点的输入文本。

以下是每次通话的短信数和字符数限制。

以下是可选参数。

- `return_likelihoods` — 指定如何以及是否在响应中返回令牌似然性。可以指定以下选项。
 - `GENERATION` – 仅返回生成的词元的可能性。
 - `ALL` – 返回所有词元的可能性。
 - `NONE` – (默认) 不返回任何可能性。
- `stream` — (需要支持流式传输) 指定`true`实时`false`返回响应 `piece-by-piece`，并在处理完成后返回完整的响应。
- `logit_bias` — 防止模型生成不需要的代币或激励模型包含所需的代币。格式是 `{token_id: bias}`，其中偏差是介于 -10 和 10 之间的浮点数。可以使用任何标记化服务 (例如Cohere的 `Tokenize` 端点) 从文本中获取令牌。有关更多信息，请参阅[Cohere文档](#)。

默认	最低	最高
不适用	-10 (表示词元偏差)	10 (表示词元偏差)

- num_generations — 模型应返回的最大世代数。

默认	最低	最高
1	1	5

- truncate — 指定 API 如何处理长度超过最大令牌长度的输入。使用以下值之一：
 - NONE – 当输入超过最大输入词元长度时，返回错误。
 - START — 丢弃输入的开头。
 - END – (默认) 丢弃输入的结尾。

如果指定 START 或 END，则模型会丢弃输入，直到剩余的输入正好达到模型的最大输入词元长度。

- 温度-使用较低的值来降低响应中的随机性。

默认	最低	最高
0.9	0	5

- p — Top P。使用较低的值忽略可能性较小的选项。设置为 0 或 1.0 可禁用。如果 p 和 k 同时启用，则 p 在 k 之后执行。

默认	最低	最高
0.75	0	1

- k — Top K。指定模型用于生成下一个令牌的代币选择数。如果 p 和 k 同时启用，则 p 在 k 之后执行。

默认	最低	最高
0	0	500

- `max_tokens` — 指定要在生成的响应中使用的最大令牌数。

默认	最低	最高
20	1	4096

- `stop_sequences` — 最多配置四个模型可以识别的序列。遇到停止序列后，模型将停止生成更多词元。返回的文本不包含停止序列。

Response

此响应可能包含以下字段：

```
{
  "generations": [
    {
      "finish_reason": "COMPLETE | MAX_TOKENS | ERROR | ERROR_TOXIC",
      "id": string,
      "text": string,
      "likelihood": float,
      "token_likelihoods": [{"token": float}],
      "is_finished": true | false,
      "index": integer
    }
  ],
  "id": string,
  "prompt": string
}
```

- `generations` — 生成的结果和所请求词元的可能性的列表。（总是返回）。列表中的每个生成对象都包含以下字段。
 - `id` — 生成的标识符。（总是返回）。
 - `likelihood` — 输出的可能性。该字段的值是 `token_likelihoods` 中词元可能性的平均值。如果指定 `return_likelihoods` 输入参数，则会返回这个值。
 - `token_likelihoods` — 每个词元可能性的数组。如果指定 `return_likelihoods` 输入参数，则会返回这个值。
 - `finish_reason` — 模型完成生成代币的原因。 `COMPLETE`— 模型发回了已完成的回复。 `MAX_TOKENS`— 由于模型达到了其上下文长度的最大标记数，因此回复被切断。

ERROR — 生成回复时出了点问题。ERROR_TOXIC— 模型生成的回复被认为是有毒的。
finish_reason仅在 is_finished = 时返回true。(并非总是返回)。

- is_finished — 仅在 stream 为 true 时才使用的布尔字段，表示是否还有其他词元将作为流式传输响应的一部分生成。(并非总是返回)
- text — 生成的文本。
- index — 在流式传输响应中，用于确定给定词元属于哪个世代。当只流式传输一个响应时，所有词元都属于同一个世代，并且不会返回索引。因此，index 仅当 num_generations 值大于 1 时才在流式传输请求中返回。
- prompt — 来自输入请求的提示 (总是返回)。
- id — 请求的标识符 (总是返回)。

欲了解更多信息，请参阅Cohere文档中的 <https://docs.cohere.com/reference/generate>。

代码示例

此示例说明如何调用CohereCommand模型。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate text using a Cohere model.
"""
import json
import logging
import boto3

from botocore.exceptions import ClientError

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

def generate_text(model_id, body):
    """
    Generate text using a Cohere model.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
```

```
Returns:
    dict: The response from the model.
"""

logger.info("Generating text with Cohere model %s", model_id)

accept = 'application/json'
content_type = 'application/json'

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

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

logger.info("Successfully generated text with Cohere model %s", model_id)

return response

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

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = 'cohere.command-text-v14'
    prompt = """Summarize this dialogue:
Customer: Please connect me with a support agent.
AI: Hi there, how can I assist you today?
Customer: I forgot my password and lost access to the email affiliated to my account.
Can you please help me?
AI: Yes of course. First I'll need to confirm your identity and then I can connect you
with one of our support agents.
"""

    try:
        body = json.dumps({
            "prompt": prompt,
            "max_tokens": 200,
```

```
        "temperature": 0.6,
        "p": 1,
        "k": 0,
        "num_generations": 2,
        "return_likeliheids": "GENERATION"
    })
    response = generate_text(model_id=model_id,
                             body=body)

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

    for index, generation in enumerate(generations):

        print(f"Generation {index + 1}\n-----")
        print(f"Text:\n {generation['text']}\n")
        if 'likelihood' in generation:
            print(f"Likelihood:\n {generation['likelihood']}\n")

        print(f"Reason: {generation['finish_reason']}\n\n")

    except ClientError as err:
        message = err.response["Error"]["Message"]
        logger.error("A client error occurred: %s", message)
        print("A client error occurred: " +
              format(message))
    else:
        print(f"Finished generating text with Cohere model {model_id}.")

if __name__ == "__main__":
    main()
```

CohereEmbed模型

您可以通过以下方式向Embed模型发出推理请求：[InvokeModel](#)您需要要使用的模型的模型 ID。要获取模型 ID，请参阅[亚马逊 Bedrock 型号](#)。

Note

Amazon Bedrock 不支持来自CohereEmbed模型的直播回复。

主题

- [请求和响应](#)
- [代码示例](#)

请求和响应

Request

这些CohereEmbed模型具有以下推理参数。

```
{
  "texts": [string],
  "input_type": "search_document|search_query|classification|clustering",
  "truncate": "NONE|LEFT|RIGHT"
}
```

以下是必要参数。

- `texts` — (必填) 模型要嵌入的字符串数组。为了获得最佳性能，我们建议将每个文本的长度减小到 512 个以下词元。1 个词元约含 4 个字符。

以下是每次通话的短信数和字符数限制。

每次通话发短信

最低	最高	
0 条文本	128 篇文本	

人物

最低	最高	
0 个字符	2048 个字符	

以下是可选参数。

- `input_type` — 在前面添加特殊标记，以区分每种类型。除非混合使用不同类型进行搜索和检索，否则不应将不同的类型混合在一起。在这种情况下，请在语料库中嵌入 `search_document` 类型，并在嵌入式查询中嵌入 `search_query` 类型。
- `search_document` — 在搜索使用案例中，使用 `search_document` 对存储在向量数据库中的嵌入内容的文档进行编码。
- `search_query` — 使用 `search_query` 查询向量数据库以查找相关文档。
- `classification` — 使用 `classification` 将嵌入内容用作文本分类器的输入。
- `clustering` — 使用 `clustering` 对嵌入内容进行聚类。
- `truncate` — 指定 API 如何处理长度超过最大令牌长度的输入。使用以下值之一：
 - `NONE` — (默认) 当输入超过最大输入词元长度时返回错误。
 - `LEFT` — 丢弃输入的开头。
 - `RIGHT` — 丢弃输入的结尾。

如果指定 `LEFT` 或 `RIGHT`，则模型会丢弃输入，直到剩余的输入正好达到模型的最大输入词元长度。

欲了解更多信息，请参阅 Cohere 文档中的 <https://docs.cohere.com/reference/embed>。

Response

以下是来自对 `InvokeModel` 的调用的 `body` 响应。

```
{
  "embeddings": [
    [ <array of 1024 floats> ]
  ],
  "id": string,
  "response_type": "embeddings_floats",
  "texts": [string]
}
```

`body` 响应含有以下值：

- `id` — 响应的标识符。
- `r@@ response_type` — 响应类型。此值始终为 `embeddings_floats`。

- `embeddings` — 嵌入内容数组，其中每个嵌入内容是包含 1024 个元素的浮点数数组。`embeddings` 数组的长度将与原 `texts` 数组的长度相同。
- `texts` – 包含为其返回嵌入内容的文本条目的数组。

有关更多信息，请参阅 <https://docs.cohere.com/reference/embed>。

代码示例

此示例说明如何调用 `CohereCommandEmbed` 模型。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate text embeddings using the Cohere Embed English model.
"""
import json
import logging
import boto3

from botocore.exceptions import ClientError

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

def generate_text_embeddings(model_id, body):
    """
    Generate text embedding by using the Cohere Embed model.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
        dict: The response from the model.
    """

    logger.info(
        "Generating text embeddings with the Cohere Embed model %s", model_id)

    accept = '*/*'
    content_type = 'application/json'
```

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

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

logger.info("Successfully generated text with Cohere model %s", model_id)

return response

def main():
    """
    Entrypoint for Cohere Embed example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = 'cohere.embed-english-v3'
    text1 = "hello world"
    text2 = "this is a test"
    input_type = "search_document"

    try:

        body = json.dumps({
            "texts": [
                text1,
                text2],
            "input_type": input_type}
        )
        response = generate_text_embeddings(model_id=model_id,
                                           body=body)

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

        print(f"ID: {response_body.get('id')}")
        print(f"Response type: {response_body.get('response_type')}")
```

```
print("Embeddings")
for i, embedding in enumerate(response_body.get('embeddings')):
    print(f"\tEmbedding {i}")
    print(*embedding)

print("Texts")
for i, text in enumerate(response_body.get('texts')):
    print(f"\tText {i}: {text}")

except ClientError as err:
    message = err.response["Error"]["Message"]
    logger.error("A client error occurred: %s", message)
    print("A client error occurred: " +
          format(message))
else:
    print(
        f"Finished generating text embeddings with Cohere model {model_id}.")

if __name__ == "__main__":
    main()
```

MetaLlama 2和MetaLlama 2 Chat模型

本节提供推理参数以及使用MetaLlama 2和MetaLlama 2 Chat模型的代码示例。

主题

- [请求和回应](#)
- [代码示例](#)

请求和回应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。

Request

MetaLlama 2 ChatLlama 2模型具有以下推理参数。

```
{
  "prompt": string,
  "temperature": float,
  "top_p": float,
```



```
"max_gen_len": int
}
```

以下是必要参数。

- `prompt` — (必填) 要传递给模型的提示。

以下是可选参数。

- 温度-使用较低的值来降低响应中的随机性。

默认	最低	最高
0.5	0	1

- `top_p` — 使用较低的值忽略可能性较小的选项。设置为 0 或 1.0 可禁用。

默认	最低	最高
0.9	0	1

- `max_gen_len` — 指定要在生成的响应中使用的最大令牌数。一旦生成的文本超过 `max_gen_len`，模型就会截断响应。

默认	最低	最高
512	1	2048

Response

MetaLlama 2 Llama 2 Chat 模型会为文本完成推理调用返回以下字段。

```
{
  "generation": "\n\n<response>",
  "prompt_token_count": int,
  "generation_token_count": int,
  "stop_reason" : string
}
```

下面提供了有关每个字段的更多信息。

- 生成-生成的文本。
- `prompt_token_count` — 提示中的代币数量。
- `generation_token_count` — 生成的文本中的标记数量。
- `stop_reason` — 响应停止生成文本的原因。可能的值有：
 - `stop` – 模型已结束为输入提示生成文本。
 - `length` – 生成的文本的词元长度超过了对 `InvokeModel` (如果您要对输出进行流式传输, 则为 `InvokeModelWithResponseStream`) 的调用中的 `max_gen_len` 值。响应被截断为 `max_gen_len` 个词元。考虑增大 `max_gen_len` 的值并重试。

代码示例

此示例说明如何调用 MetaLlama 2 Chat13B 模型。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate text with Meta Llama 2 Chat (on demand).
"""

import json
import logging
import boto3

from botocore.exceptions import ClientError

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

def generate_text(model_id, body):
    """
    Generate an image using Meta Llama 2 Chat on demand.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
```

```
        response (JSON): The text that the model generated, token information, and the
        reason the model stopped generating text.
    """

    logger.info("Generating image with Meta Llama 2 Chat 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())

    return response_body

def main():
    """
    Entrypoint for Meta Llama 2 Chat example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    model_id = 'meta.llama2-13b-chat-v1'
    prompt = """What is the average lifespan of a Llama?"""
    max_gen_len = 128
    temperature = 0.1
    top_p = 0.9

    # Create request body.
    body = json.dumps({
        "prompt": prompt,
        "max_gen_len": max_gen_len,
        "temperature": temperature,
        "top_p": top_p
    })
```

```
try:

    response = generate_text(model_id, body)

    print(f"Generated Text: {response['generation']}")
    print(f"Prompt Token count: {response['prompt_token_count']}")
    print(f"Generation Token count: {response['generation_token_count']}")
    print(f"Stop reason: {response['stop_reason']}")

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

else:
    print(
        f"Finished generating text with Meta Llama 2 Chat model {model_id}.")

if __name__ == "__main__":
    main()
```

Mistral AI模型

您可以使用[InvokeModel](#)或[InvokeModelWithResponseStream](#)（流式传输）向Mistral 7B Instruct和Mixtral 8X7B Instruct模型发出推理请求。您需要获得希望使用的模型的模型 ID。要获取模型 ID，请参阅[亚马逊 Bedrock 型号](#)。

Mistral AI模型在[Apache 2.0 许可](#)下可用。有关使用Mistral AI模型的更多信息，请参阅[Mistral AI文档](#)。

主题

- [请求和响应](#)
- [代码示例](#)

请求和响应

Request

这些Mistral AI模型具有以下推理参数。

```
{
  "prompt": string,
  "max_tokens" : int,
  "stop" : [string],
  "temperature": float,
  "top_p": float,
  "top_k": int
}
```

以下是必要参数。

- `prompt` — (必填) 要传递给模型的提示，如以下示例所示。

```
<s>[INST] What is your favourite condiment? [/INST]
```

以下示例说明如何格式化多回合提示音。

```
<s>[INST] What is your favourite condiment? [/INST]
Well, I'm quite partial to a good squeeze of fresh lemon juice.
It adds just the right amount of zesty flavour to whatever I'm cooking up in the
kitchen!</s>
[INST] Do you have mayonnaise recipes? [/INST]
```

用户角色的文本位于 `[INST]...[/INST]` 令牌内，外部的文本为助理角色。字符串的开头和结尾由 `<s>`（字符串的开头）和 `</s>`（字符串的结尾）标记表示。有关以正确格式发送聊天提示的信息，请参阅 Mistral AI 文档中的 [聊天模板](#)。

以下是可选参数。

- `max_tokens` — 指定要在生成的响应中使用的最大令牌数。一旦生成的文本超过 `max_tokens`，模型就会截断响应。

默认	最低	最高
512	1	Mistral 7B Instruct— 8,192 Mixtral 8X7B Instruct— 4,096

- `st@@ op` — 停止序列列表，如果由模型生成，则停止模型生成进一步的输出。

默认	最低	最高
0	0	10

- 温度-控制模型所做预测的随机性。有关更多信息，请参阅[推理参数](#)。

默认	最低	最高
0.5	0	1

- `top_p` — 通过设置模型考虑的下一个标记中最有可能的候选文本的百分比，来控制模型生成的文本的多样性。有关更多信息，请参阅[推理参数](#)。

默认	最低	最高
0.9	0	1

- `top_k` — 控制模型考虑的下一个代币中最有可能的候选人的数量。有关更多信息，请参阅[推理参数](#)。

默认	最低	最高
50	1	200

Response

以下是来自对 `InvokeModel` 的调用的 `body` 响应。

```
{
  "outputs": [
    {
      "text": string,
      "stop_reason": string
    }
  ]
}
```

body 响应含有以下值：

- 输出-模型的输出列表。每个输出都有以下字段。
 - 文本-模型生成的文本。
 - stop_reason — 响应停止生成文本的原因。可能的值有：
 - stop – 模型已结束为输入提示生成文本。模型之所以停止，是因为它没有更多内容要生成，或者模型生成了您在stop请求参数中定义的停止序列之一。
 - length – 生成的文本的词元长度超过了对 InvokeModel (如果您要对输出进行流式传输，则为 InvokeModelWithResponseStream) 的调用中的 max_tokens 值。响应被截断为 max_tokens 个词元。

代码示例

此示例说明如何调用Mistral 7B Instruct模型。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate text using a Mistral AI model.
"""
import json
import logging
import boto3

from botocore.exceptions import ClientError

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

def generate_text(model_id, body):
    """
    Generate text using a Mistral AI model.
    Args:
        model_id (str): The model ID to use.
        body (str) : The request body to use.
    Returns:
```

```
    JSON: The response from the model.
    """

    logger.info("Generating text with Mistral AI model %s", model_id)

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

    response = bedrock.invoke_model(
        body=body,
        modelId=model_id
    )

    logger.info("Successfully generated text with Mistral AI model %s", model_id)

    return response

def main():
    """
    Entrypoint for Mistral AI example.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
        model_id = 'mistral.mistral-7b-instruct-v0:2'

        prompt = """<s>[INST] In Bash, how do I list all text files in the current
        directory
        (excluding subdirectories) that have been modified in the last month? [/
        INST]"""

        body = json.dumps({
            "prompt": prompt,
            "max_tokens": 400,
            "temperature": 0.7,
            "top_p": 0.7,
            "top_k": 50
        })

        response = generate_text(model_id=model_id,
                                body=body)
```



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

outputs = response_body.get('outputs')

for index, output in enumerate(outputs):

    print(f"Output {index + 1}\n-----")
    print(f"Text:\n{output['text']}\n")
    print(f"Stop reason: {output['stop_reason']}\n")

except ClientError as err:
    message = err.response["Error"]["Message"]
    logger.error("A client error occurred: %s", message)
    print("A client error occurred: " +
          format(message))
else:
    print(f"Finished generating text with Mistral AI model {model_id}.")

if __name__ == "__main__":
    main()
```

Stability.ai Diffusion 模型

以下是 Amazon Bedrock 支持的 Stability.ai Diffusion 模型的推理参数信息。

模型

- [Stability.ai Diffusion 0.8](#)
- [Stability.ai Diffusion 1.0 文本到图像](#)
- [Stability.ai Diffusion 1.0 图像到图像](#)
- [Stability.ai Diffusion 1.0 图像到图像 \(使用掩膜\)](#)

Stability.ai Diffusion 0.8

Stability.ai Diffusion 模型具备以下控件。

- Prompt strength (cfg_scale) – 确定最终图像对提示的描绘程度。使用较小的数字可提高生成内容的随机性。
- Generation step (steps) – 生成步骤决定对图像进行采样的次数。步骤越多，结果就越准确。

- **Seed (seed)** – 种子决定初始噪音设置。使用与上一次运行时相同的种子和相同的设置，以使推理可以创建相似的图像。如果不设置此值，则会将其设置为随机数字。

模型调用请求正文字段

使用 Stability.ai 模型进行 [InvokeModel](#) 或 [InvokeModelWithResponseStream](#) 调用时，请使用符合以下模型的 JSON 对象填充该 body 字段。在 text_prompts 对象的 text 字段中输入提示。

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

下表显示了数字参数的最小值、最大值和默认值。

参数	JSON 对象格式	最低	最高	默认
Prompt strength	cfg_scale	0	30	10
Generation step	步骤	10	150	30

模型调用响应正文字段

有关响应中 body 字段的格式的信息，请参阅 <https://platform.stability.ai/docs/api-reference#tag/v1generation>。

Stability.ai Diffusion 1.0 文本到图像

Stability.ai Diffusion 1.0 模型具有以下推理参数和模型响应，用于进行文本到图像的推理调用。

主题

- [请求和响应](#)
- [代码示例](#)

请求和响应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。

有关更多信息，请参阅 <https://platform.stability.ai/docs/api-reference#tag/v1generation>。

Request

Stability.ai Diffusion 1.0 模型具有以下推理参数，用于进行文本到图像的推理调用。

```
{
  "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` (必要) – 用于生成的文本提示数组。每个元素都是一个 JSON 对象，其中包含一个提示和该提示的权重。
 - `text` – 要传递给模型的提示。

最低	最高	
0	2000	

- `weight` (可选) – 模型应当应用于提示的权重。小于零的值表示否定提示。否定提示用于告知模型避免某些概念。`weight` 的默认值为 1。

- `cfg_scale` – (可选) 确定最终图像对提示的描绘程度。使用较小的数字可提高生成内容的随机性。

最低	最高	默认
0	35	7

- `clip_guidance_preset` – (可选) 枚举 : `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE SLOW`, `SLOWER`, `SLOWEST`。

- `height` – (可选) 要生成的图像的高度，以像素为单位，增量可被 64 整除。

值必须为下列项之一 : `1024x1024`, `1152x896`, `1216x832`, `1344x768`, `1536x640`, `640x1536`, `768x1344`, `832x1216`, `896x1152`。

- `width` – (可选) 要生成的图像的宽度，以像素为单位，增量可被 64 整除。

值必须为下列项之一 : `1024x1024`, `1152x896`, `1216x832`, `1344x768`, `1536x640`, `640x1536`, `768x1344`, `832x1216`, `896x1152`。

- `sampler` – (可选) 用于扩散过程的采样器。如果省略此值，模型会自动为您选择合适的采样器。

枚举 : `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` – (可选) 要生成的图像数量。目前，Amazon Bedrock 支持生成一个图像。如果为 `samples` 提供一个值，该值必须为 1。

默认	最低	最高
1	1	1

- `seed` – (可选) 种子决定初始噪声设置。使用与上一次运行时相同的种子和相同的设置，以使推理可以创建相似的图像。如果不设置此值，或者此值为 0，它会被设置为一个随机数字。

最低	最高	默认
0	4294967295	0

- `steps` – (可选) 生成步骤决定对图像进行采样的次数。步骤越多，结果就越准确。

最低	最高	默认
10	50	30

- `style_preset` (可选) – 用于将图像模型向特定样式引导的样式预设。此样式预设列表可能会发生变化。

枚举 : 3d-model, analog-film, anime, cinematic, comic-book, digital-art, enhance, fantasy-art, isometric, line-art, low-poly, modeling-compound, neon-punk, origami, photographic, pixel-art, tile-texture。

- `extras` (可选) – 传递给引擎的额外参数。请谨慎使用。这些参数用于开发中或实验性的功能，可能会有变更，恕不另行通知。

Response

Stability.ai Diffusion 1.0 模型将返回以下字段，用于进行文本到图像的推理调用。

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

- `result` – 操作的结果。如果成功，响应为 `success`。
- `artifacts` – 一个图像数组，每个请求的图像对应一个。
 - `seed` – 用于生成图像的种子值。
 - `base64` – 模型生成的 base64 编码图像。
 - `finishedReason` – 图像生成过程的结果。有效值为：
 - `SUCCESS` – 图像生成过程成功。
 - `ERROR` – 出现错误。
 - `CONTENT_FILTERED` – 内容筛选器对图像进行了筛选，图像可能会变得模糊。

代码示例

以下示例介绍如何使用 Stability.ai Diffusion 1.0 模型和按需吞吐量运行推理。该示例将向模型提交一个文本提示，然后从模型中检索响应，最后显示图像。

```
# 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 model.
    """

    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():
    """
    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.""

    # Create request body.
    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 图像到图像

Stability.ai Diffusion 1.0 模型具有以下推理参数和模型响应，用于进行图像到图像的推理调用。

主题

- [请求和响应](#)
- [代码示例](#)

请求和响应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。

欲了解更多信息，请参阅 <https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/imageToImage>。

Request

Stability.ai Diffusion 1.0 模型具有以下推理参数，用于进行图像到图像的推理调用。

```
{
  "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
}
```

以下是必要参数。

- `text_prompts` – (必要) 用于生成的文本提示数组。每个元素都是一个 JSON 对象，其中包含一个提示和该提示的权重。
 - `text` – 要传递给模型的提示。

最低	最高
0	2000

- `weight` – (可选) 模型应当应用于提示的权重。小于零的值表示否定提示。否定提示用于告知模型避免某些概念。`weight` 的默认值为 1。
- `init_image` – (必要) 要用于初始化扩散过程的 base64 编码图像。

以下是可选参数。

- `init_image_mode` – (可选) 确定是使用 `image_strength` 还是 `step_schedule_*` 来控制 `init_image` 中的图像对结果的影响程度。可能的值为 `IMAGE_STRENGTH` 或 `STEP_SCHEDULE`。默认值为 `IMAGE_STRENGTH`。
- `image_strength` – (可选) 确定 `init_image` 中的源图像对扩散过程的影响程度。接近 1 的值会生成与源图像非常相似的图像。接近 0 的值会生成与源图像非常不同的图像。
- `cfg_scale` – (可选) 确定最终图像对提示的描绘程度。使用较小的数字可提高生成内容的随机性。

默认	最低	最高
7	0	35

- `clip_guidance_preset` – (可选) 枚举 : `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE`, `SLOW`, `SLOWER`, `SLOWEST`。
- `sampler` – (可选) 用于扩散过程的采样器。如果省略此值, 模型会自动为您选择合适的采样器。

枚举 : `DDIM` `DDPM`, `K_DPMP2M`, `K_DPMP2S_ANCESTRAL`, `K_DPM2`, `K_DPM2_ANCESTRAL`, `K_EULER`, `K_EULER_ANCESTRAL`, `K_HEUN` `K_LMS`。

- `samples` – (可选) 要生成的图像数量。目前, Amazon Bedrock 支持生成一个图像。如果为 `samples` 提供一个值, 该值必须为 1。

默认	最低	最高
1	1	1

- `seed` – (可选) 种子决定初始噪声设置。使用与上一次运行时相同的种子和相同的设置, 以使推理可以创建相似的图像。如果不设置此值, 或者此值为 0, 它会被设置为一个随机数字。

默认	最低	最高
0	0	4294967295

- `steps` – (可选) 生成步骤决定对图像进行采样的次数。步骤越多, 结果就越准确。

默认	最低	最高
30	10	50

- `style_preset` – (可选) 用于将图像模型向特定样式引导的样式预设。此样式预设列表可能会发生变化。

枚举 : 3d-model, analog-film, anime, cinematic, comic-book, digital-art, enhance, fantasy-art, isometric, line-art, low-poly, modeling-compound, neon-punk, origami, photographic, pixel-art, tile-texture

- `extras` – (可选) 传递给引擎的额外参数。请谨慎使用。这些参数用于开发中或实验性的功能，可能会有变更，恕不另行通知。

Response

Stability.ai Diffusion 1.0 模型将返回以下字段，用于进行文本到图像的推理调用。

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

- `result` – 操作的结果。如果成功，响应为 `success`。
- `artifacts` – 一个图像数组，每个请求的图像对应一个。
 - `seed` – 用于生成图像的种子值。
 - `base64` – 模型生成的 base64 编码图像。
 - `finishedReason` – 图像生成过程的结果。有效值为：
 - `SUCCESS` – 图像生成过程成功。
 - `ERROR` – 出现错误。
 - `CONTENT_FILTERED` – 内容筛选器对图像进行了筛选，图像可能会变得模糊。

代码示例

以下示例介绍如何使用 Stability.ai Diffusion 1.0 模型和按需吞吐量运行推理。该示例将向模型提交一个文本提示和参照图像，然后从模型中检索响应，最后显示图像。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to generate an image from a reference 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 model.
    """

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

    logging.basicConfig(level = logging.INFO,
                        format = "%(levelname)s: %(message)s")

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

    prompt="A space ship."

    # Read reference image from file and encode as base64 strings.
    with open("/path/to/image", "rb") as image_file:
        init_image = base64.b64encode(image_file.read()).decode('utf8')

    # Create request body.
    body=json.dumps({
        "text_prompts": [
            {
```

```
        "text": prompt
    }
],
"init_image": init_image,
"style_preset" : "isometric"
})

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 图像到图像 (使用掩膜)

Stability.ai Diffusion 1.0 模型具有以下推理参数和模型响应，用于使用掩膜进行图像到图像的推理调用。

请求和响应

请求正文在请求body字段中传递给[InvokeModel](#)或[InvokeModelWithResponseStream](#)。

有关更多信息，请参阅 <https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/masking>。

Request

Stability.ai Diffusion 1.0 模型具有以下推理参数，用于进行图像到图像（使用掩膜）的推理调用。

```
{
  "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
}
```

以下是必要参数。

- `text_prompt` – (必要) 用于生成的文本提示数组。每个元素都是一个 JSON 对象，其中包含一个提示和该提示的权重。
 - `text` – 要传递给模型的提示。

最低	最高
0	2000

- `weight` – (可选) 模型应当应用于提示的权重。小于零的值表示否定提示。否定提示用于告知模型避免某些概念。`weight` 的默认值为 1。
- `init_image` – (必要) 要用于初始化扩散过程的 base64 编码图像。
- `mask_source` – (必要) 确定掩膜的来源。可能的值有：

- `MASK_IMAGE_WHITE` – 使用 `mask_image` 中掩膜图像的白色像素作为掩膜。白色像素将被替换，黑色像素保持不变。
- `MASK_IMAGE_BLACK` – 使用 `mask_image` 中掩膜图像的黑色像素作为掩膜。黑色像素将被替换，白色像素保持不变。
- `INIT_IMAGE_ALPHA` – 使用 `init_image` 中图像的 Alpha 通道作为掩膜。完全透明的像素将被替换，完全不透明的像素保持不变。
- `mask_image` – (必要) 要用作 `init_image` 中源图像掩膜的 base64 编码掩膜图像。必须与源图像的尺寸相同。使用 `mask_source` 选项来指定应替换哪些像素。

以下是可选参数。

- `cfg_scale` – (可选) 确定最终图像对提示的描绘程度。使用较小的数字可提高生成内容的随机性。

默认	最低	最高
7	0	35

- `clip_guidance_preset` – (可选) 枚举 : `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE`, `SLOW`, `SLOWER`, `SLOWEST`。
- `sampler` – (可选) 用于扩散过程的采样器。如果省略此值，模型会自动为您选择合适的采样器。

枚举 : `DDIM`, `DDPM`, `K_DPMP2M`, `K_DPMP2S_ANCESTRAL`, `K_DPM2`, `K_DPM2_ANCESTRAL`, `K_EULER`, `K_EULER_ANCESTRAL`, `K_HEUN` `K_LMS`。

- `samples` – (可选) 要生成的图像数量。目前，Amazon Bedrock 支持生成一个图像。如果为 `samples` 提供一个值，该值必须为 1。

默认	最低	最高
1	1	1

- `seed` – (可选) 种子决定初始噪声设置。使用与上一次运行时相同的种子和相同的设置，以使推理可以创建相似的图像。如果不设置此值，或者此值为 0，它会被设置为一个随机数字。

默认	最低	最高
0	0	4294967295

- `steps` – (可选) 生成步骤决定对图像进行采样的次数。步骤越多，结果就越准确。

默认	最低	最高
30	10	50

- `style_preset` – (可选) 用于将图像模型向特定样式引导的样式预设。此样式预设列表可能会发生变化。

枚举: 3d-model, analog-film, anime, cinematic, comic-book, digital-art, enhance, fantasy-art, isometric, line-art, low-poly, modeling-compound, neon-punk, origami, photographic, pixel-art, tile-texture

- `extras` – (可选) 传递给引擎的额外参数。请谨慎使用。这些参数用于开发中或实验性的功能，可能会有变更，恕不另行通知。

Response

Stability.ai Diffusion 1.0 模型将返回以下字段，用于进行文本到图像的推理调用。

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

- `result` – 操作的结果。如果成功，响应为 `success`。
- `artifacts` – 一个图像数组，每个请求的图像对应一个。
 - `seed` – 用于生成图像的种子值。

- base64 – 模型生成的 base64 编码图像。
- finishedReason – 图像生成过程的结果。有效值为：
 - SUCCESS – 图像生成过程成功。
 - ERROR – 出现错误。
 - CONTENT_FILTERED – 内容筛选器对图像进行了筛选，图像可能会变得模糊。

自定义模型超参数

以下参考内容包括可用于训练各个 Amazon Bedrock 自定义模型的超参数。

超参数是用于控制训练过程的参数，例如学习率或周期计数。当您使用 Amazon Bedrock 控制台或调用 [CreateModelCustomizationJob](#) API 操作 [提交](#) 微调任务时，您可以为自定义模型训练设置超参数。有关超参数设置的准则，请参阅 [模型自定义准则](#)。

主题

- [Amazon Titan 文本模型自定义超参数](#)
- [Amazon Titan Image Generator G1 模型自定义超参数](#)
- [Amazon Titan Multimodal Embeddings G1 自定义超参数](#)
- [CohereCommand模型自定义超参数](#)
- [MetaLlama 2模型自定义超参数](#)

Amazon Titan 文本模型自定义超参数

Titan 文本模型支持以下用于模型自定义的超参数。

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
周期	epochCount	整个训练数据集的迭代次数	整数	1	10	5
批次大小	batchSize	更新模型参数之前处理的样本数	整数	1	64	1

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
学习率	learningRate	每个批次之后模型参数的更新速率	float	0.0	1	1.00E-5
学习率预热步骤	learningRateWarmupSteps	学习率逐渐提高到指定速率的迭代次数	整数	0	250	5

Amazon Titan Image Generator G1 模型自定义超参数

Amazon Titan Image Generator G1 模型支持以下用于模型自定义的超参数。

Note

stepCount 没有默认值，必须指定。stepCount 支持该值 auto。auto 根据数据集的大小自动确定数字，将模型性能优先于训练成本。培训工作成本取决于确定的数字 auto。要了解工作成本的计算方式并查看示例，请参阅 [Amazon Bedrock 定价](#)。

超参数 (控制台)	超参数 (API)	定义	最低	最高	默认
批次大小	batchSize	更新模型参数之前处理的样本数	8	192	8
步骤	stepCount	模型每批次暴露的次数	10	40000	不适用
学习率	learningRate	每个批次之后模型参数的更新速率	1.00E-7	1	1.00E-5

Amazon Titan Multimodal Embeddings G1 自定义超参数

Amazon Titan Multimodal Embeddings G1 模型支持以下用于模型自定义的超参数。

Note

epochCount 没有默认值，必须指定。epochCount 支持该值 Auto。Auto 根据数据集的大小自动确定数字，将模型性能优先于训练成本。培训工作成本取决于确定的数 Auto 字。要了解工作成本的计算方式并查看示例，请参阅 [Amazon Bedrock 定价](#)。

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
周期	epochCount	整个训练数据集的迭代次数	整数	1	100	不适用
批次大小	batchSize	更新模型参数之前处理的样本数	整数	256	9,216	576
学习率	learningRate	每个批次之后模型参数的更新速率	float	5.00E-8	1	5.00E-5

CohereCommand 模型自定义超参数

CohereCommand 和 CohereCommand Light 模型支持以下用于模型自定义的超参数。有关更多信息，请参阅 [自定义模型](#)。

有关微调 Cohere 模型的信息，请参阅 Cohere 文档，[网址为 https://docs.cohere.com/docs/fine-tuning](https://docs.cohere.com/docs/fine-tuning)。

Note

epochCount 配额是可调整的。

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
周期	epochCount	整个训练数据集的迭代次数	整数	1	100	1
批次大小	batchSize	更新模型参数之前处理的样本数	整数	8	8 (命令) 32 (Light)	8
学习率	learningRate	每个批次之后模型参数的更新速率。如果您使用验证数据集，我们建议您不要为提供值learningRate。	float	5.00E-6	0.1	1.00E-5
提前停止阈值	earlyStoppingThreshold	防止培训过程过早终止所需的最低损失改善	float	0	0.1	0.01
提早停止耐心	earlyStoppingPatience	停止训练过程之前损失指标停滞的容忍度	整数	1	10	6
评估百分比	evalPercentage	分配给模型评估的数据集的百分比 (如果您未提供单独	float	5	50	20

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
		的验证数据集)				

MetaLlama 2模型自定义超参数

MetaLlama 2 13B 和 70B 模型支持以下用于模型自定义的超参数。有关更多信息，请参阅[自定义模型](#)。

有关微调 Meta Llama 模型的信息，请参阅 <https://ai.meta.com/llama/get-started/#fine-tuning> 上的 Meta 文档。

Note

epochCount 配额是可调整的。

超参数 (控制台)	超参数 (API)	定义	Type	最低	最高	默认
周期	epochCount	整个训练数据集的迭代次数	整数	1	10	5
批次大小	batchSize	更新模型参数之前处理的样本数	整数	1	1	1
学习率	learningRate	每个批次之后模型参数的更新速率	float	5.00E-6	0.1	1.00E-4

Amazon Bedrock 控制台概述

Amazon Bedrock 控制台提供以下功能。

功能

- [开始使用](#)
- [根基模型](#)
- [操场](#)
- [保障措施](#)
- [编排](#)
- [评估和部署](#)
- [模型访问权限](#)
- [模型调用日志记录](#)

要打开 Amazon Bedrock 控制台，请通过以下网址登录：<https://console.aws.amazon.com/bedrock/home>。

开始使用

从导航窗格的入门中，可以获得 Amazon Bedrock 提供的根基模型、示例和操场概述。还可以获得可用于 Amazon Bedrock 模型的提示示例。

示例页面显示了可用模型的示例提示。可以使用以下一项或多项属性来搜索示例和筛选示例列表：

- 模型
- 模态（文本、图像或嵌入）
- 类别
- 提供商

通过选择在示例中搜索编辑框，然后选择要应用于搜索的筛选器来筛选示例提示。再次选择在示例中搜索，然后选择其他筛选器，即可应用多个筛选器。

选择示例时，Amazon Bedrock 控制台会显示有关该示例的以下信息：

- 对示例所完成任务的描述。

- 运行示例的模型名称（和模型提供商）。
- 示例提示和预期响应。
- 示例的推理配置参数设置。
- 运行示例的 API 请求。

要运行此示例，请选择在操场中打开。

根基模型

在导航窗格的根基模型中，可以查看可用的基础模型并按各种属性对它们进行分组。您还可以筛选模型视图、搜索模型以及查看有关模型提供商的信息。

您可以自定义基础根基模型以提高模型在特定任务中的性能，也可以向模型传授新的知识领域。在根基模型下选择自定义模型来创建和管理您的自定义模型。使用您提供的训练数据集创建模型自定义作业，以此来自定义模型。有关更多信息，请参阅[自定义模型](#)。

您可以使用控制台操场对基础模型和自定义模型进行实验。

操场

决定在应用程序中使用模型之前，可以先在控制台操场中对其进行试验。共有三种操场。

聊天操场

通过聊天操场，您可以试用 Amazon Bedrock 提供的聊天模型。您可以向模型提交聊天，聊天操场会显示模型的响应，并提供模型指标。也可以选择比较模式，比较最多三个模型的输出。有关更多信息，请参阅[聊天操场](#)。

文本操场

通过文本操场，您可以试用 Amazon Bedrock 提供的文本模型。您可以向模型提交文本，文本操场会显示模型根据提示生成的文本。有关更多信息，请参阅[文本操场](#)。

图像操场

通过图像操场，您可以试用 Amazon Bedrock 提供的图像模型。您可以向模型提交文本提示，图像操场会显示模型根据提示生成的图像。有关更多信息，请参阅[图像操场](#)。

在控制台的导航窗格中，选择操场即可访问操场。有关更多信息，请参阅[操场](#)。

保障措施

Titan Image Generator G1 自动在模特创建的所有图像上添加不可见的水印。水印检测可检测图像是否由生成 Titan Image Generator G1。要使用水印检测，请在左侧导航窗格中选择“概览”，然后选择“生成和测试”选项卡。前往“安全措施”部分，然后选择“查看水印检测”。有关更多信息，请参阅[水印检测](#)。

编排

借助 Amazon Bedrock，您可以通过 LLM 的推理功能使用知识库构建上下文应用程序，从而启用检索增强生成 (RAG) 工作流。要使用知识库，请在左侧导航窗格中选择编排，然后选择知识库。有关更多信息，请参阅[Amazon Bedrock 知识库](#)。

Agents for Amazon Bedrock 使开发人员可以配置代理以根据组织数据和用户输入完成操作。例如，您可以创建一个代理来执行操作以满足客户的请求。要使用代理，请在左侧导航窗格中选择编排，然后选择代理。有关更多信息，请参阅[Agents for Amazon Bedrock](#)。

评估和部署

使用 Amazon Bedrock 模型时，您需要评估其性能并将其部署到您的解决方案中。

通过模型评估，您可以评估和比较模型输出，然后选择最适合您的应用程序的输出。选择评估和部署，然后选择模型评估。

为模型配置预调配吞吐量时，您将以固定成本获得一定水平的吞吐量。要预调配吞吐量，请在导航窗格中选择评估和部署，然后选择预调配吞吐量。有关更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)。

模型访问权限

要在 Amazon Bedrock 中使用模型，必须先请求获得该模型的访问权限。在左侧导航窗格中，选择模型访问权限。有关更多信息，请参阅[模型访问权限](#)。

模型调用日志记录

可以通过在左侧导航窗格中选择设置来记录模型调用事件。有关更多信息，请参阅[模型调用日志记录](#)。

运行模型推理

推理是指根据提供给模型的输入生成输出的过程。根基模型使用概率来构造序列中的字词。模型可以根据输入预测接下来可能出现的词元序列，并将该序列作为输出返回。借助 Amazon Bedrock，您可以使用自己所选的根基模型运行推理。在运行推理时，您需要提供以下输入。

- 提示 – 提供给模型的输入，以便模型生成响应。有关编写提示的信息，请参阅 [提示工程准则](#)。
- 推理参数 – 一组可以调整以限制或影响模型响应的值。有关推理参数的信息，请参阅 [推理参数](#) 和 [根基模型的推理参数](#)。

Amazon Bedrock 提供了一套基础模型，您可以使用这些模型生成以下模式的输出。要查看基础模型对模式的支持，请参阅 [Amazon Bedrock 中支持的根基模型](#)

输出模式	描述	使用案例示例
文本	提供文本输入并生成各种类型的文本	聊天 question-and-answering、头脑风暴、摘要、代码生成、表格创建、数据格式化、重写
图像	提供文本或输入图像并生成或修改图像	图像生成、图像编辑、图像变体
嵌入	提供文本、图像或同时提供文本和图像，然后生成表示输入的数值向量。可以将输出向量与其他嵌入向量进行比较，以确定语义相似度（对于文本）或视觉相似度（对于图像）。	文本和图像搜索、查询、分类、推荐、个性化、 知识库创建

您可以通过以下方法运行模型推理。

- 在简单易用的图形界面中使用任何操场运行推理。
- 发送 `InvokeModel` 或 `InvokeModelWithResponseStream` 请求。
- 准备一个包含所需配置的提示数据集，并根据 `CreateModelInvocationJob` 请求运行批量推理。

- 以下 Amazon Bedrock 功能使用模型推断作为更大规模编排中的一个步骤。有关更多详细信息，请参阅这些部分。
 - 设置[知识库](#)并发送[RetrieveAndGenerate](#)请求。
 - 设置[代理](#)并发送[InvokeAgent](#)请求。

您可以使用基础模型、自定义模型或预调配模型运行推理。要对自定义模型运行推理，请先为其购买预调配吞吐量（有关更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)）。

通过这些方法，使用不同的提示和推理参数来测试根基模型的响应。充分探索这些方法后，就可以设置应用程序，以通过调用这些 API 来运行模型推理。

选择一个主题，以详细了解如何通过相应方法运行模型推理。要了解有关使用代理的更多信息，请参阅[Agents for Amazon Bedrock](#)。

主题

- [推理参数](#)
- [操场](#)
- [使用 API 通过单个提示调用模型](#)
- [运行批量推理](#)

推理参数

推理参数是可以调整以限制或影响模型响应的值。以下类别的参数是不同模型中的常见参数。

随机性和多样性

对于任何给定序列，模型都能确定序列中下一个词元选项的概率分布。为了生成输出中的每个词元，模型会从该分布中采样。随机性和多样性是指模型响应中的变体数量。您可以通过限制或调整分布来控制这些因素。根基模型通常支持使用以下参数控制响应的随机性和多样性。

- 温度 – 影响预测输出的概率分布形状，还会影响模型选择概率较低输出的可能性。
 - 选择较低的值可影响模型选择概率较高的输出。
 - 选择较高的值可影响模型选择概率较低的输出。

从技术角度来说，温度可调节下一个词元的概率质量函数。温度越低，函数曲线越陡，响应的确定性越高；温度越高，函数曲线越平，响应的随机性越高。

- 排名前 K – 模型为下一个词元考虑的最有可能的候选项数量。

- 选择较低的值可减小池的大小，并将选项限制为可能性更高的输出。
- 选择较高的值以增加池的大小，并允许模型考虑不太可能的输出。

例如，如果为“排名前 K”选择的值是 50，则模型会从 50 个最有可能作为序列中下一个词元的词元中进行选择。

- 排名前 P – 模型为下一个词元考虑的最有可能的候选项所占百分比。
 - 选择较低的值可减小池的大小，并将选项限制为可能性更高的输出。
 - 选择较高的值以增加池的大小，并允许模型考虑不太可能的输出。

从技术角度来说，模型会计算一组响应的累积概率分布，并且只考虑分布中的排名前 P%。

例如，如果为“排名前 P”选择的值是 0.8，则模型会从可能作为序列中下一个词元的词元概率分布的前 80% 中进行选择。

下表总结了这些参数的作用。

参数	较低值的作用	较高值的作用
Temperature	增加概率较高词元的可能性	增加概率较低词元的可能性
	降低概率较低词元的可能性	降低概率较高词元的可能性
Top K	移除概率较低的词元	允许概率较低的词元
Top P	移除概率较低的词元	允许概率较低的词元

为了方便理解这些参数，我们以 **I hear the hoof beats of "** 这个示例提示为例来解释一下。假设模型确定以下三个字词作为下一个词元的候选项。模型还为每个字词分配一个概率。

```
{
  "horses": 0.7,
  "zebras": 0.2,
  "unicorns": 0.1
}
```

- 如果设置较高的温度，概率分布会变平，概率差异会变小，从而增加选择“unicorns”的概率，降低选择“horses”的概率。

- 如果将排名前 K 设置为 2，模型只会考虑最有可能的前 2 个候选项，即：“horses”和“zebras”。
- 如果将排名前 P 设置为 0.7，模型只会考虑“horses”，因为只有它是位于概率分布前 70% 的候选项。

Length

根基模型通常支持使用参数限制响应的长度。下面提供了这些参数的示例。

- 响应长度 – 一个精确的值，用于指定要在生成的响应中返回的词元数量下限或上限。
- 惩罚 – 用于指定对响应中的输出进行惩罚的程度。示例包括以下内容。
 - 响应的长度。
 - 响应中的重复词元。
 - 词元在响应中的出现频率。
 - 响应中的词元类型。
- 停止序列 – 用于指定可阻止模型生成更多词元的字符序列。如果模型生成了您指定的停止序列，则将在该序列之后停止生成。

操场

Important

要使用任何根基模型，必须先请求获得对该模型的访问权限。如果您在请求模型访问权限之前尝试使用该模型（通过 API 或控制台），将会收到一条错误消息。有关更多信息，请参阅[模型访问权限](#)。

Amazon Bedrock 操场提供控制台环境，可让您尝试通过不同模型使用不同配置运行推理，然后再决定将哪个模型以及哪种配置用于应用程序。在控制台的左侧导航窗格中，选择操场即可访问操场。当您从模型详细信息页面或示例页面中选择模型时，也可以直接导航到操场。

提供适用于文本、聊天和图像模型的操场。

在每个操场中，您可以输入提示，并尝试使用推理参数。提示通常是一句或几句文字，用于为模型设定场景、问题或任务。有关创建提示的信息，请参阅[提示工程准则](#)。

推理参数会影响模型生成的响应，例如生成文本的随机性。将模型加载到操场后，操场会使用其默认推理设置配置模型。在试用模型时，可以更改和重置设置。每个模型都有自己的一组推理参数。有关更多信息，请参阅[根基模型的推理参数](#)。

在提交响应时，模型会使用其生成的输出进行响应。

如果聊天或文本模型支持流式传输，默认情况下，会流式传输模型的响应。您可以根据需要关闭流式传输。

主题

- [聊天操场](#)
- [文本操场](#)
- [图像操场](#)
- [使用操场](#)

聊天操场

通过聊天操场，您可以试用 Amazon Bedrock 提供的聊天模型。您可以向模型提交聊天，聊天操场会显示模型的响应，以及以下指标。

- 延迟 – 模型生成序列中每个词元（字词）所花费的时间。
- 输入词元计数 – 在推理期间以输入形式输入到模型中的词元数量。
- 输出词元计数 – 响应提示时生成的词元数量。更长、更像对话的响应需要更多词元。
- 成本 – 处理输入和生成输出词元的成本。

您还可以定义模型响应需要满足的条件。

启用比较模式后，您可以将单个提示的聊天回应与最多三个模型的回应进行比较。这可以帮助您了解每个模型的性能，而无需在模型之间切换。有关更多信息，请参阅[使用操场](#)。

文本操场

通过文本操场，您可以试用 Amazon Bedrock 提供的文本模型。您可以向模型提交文本，文本操场会显示模型根据提示生成的文本。

图像操场

通过图像操场，您可以试用 Amazon Bedrock 提供的图像模型。您可以向模型提交文本提示，图像操场会显示模型根据提示生成的图像。

除了设置推理参数之外，您还可以对配置进行其他更改（具体因模型而异）：

- 模式-模型生成新图像（生成）或编辑（编辑）您在参考图像中提供的图像。如果您编辑参考图像，则模型需要一个分割掩码，该掩码覆盖您希望模型编辑的图像区域。使用图像平台在参考图像上绘制矩形来创建分割蒙版。或者，您可以通过指定掩码提示来创建分段掩码（仅限 Amazon G Titan Image Generator G1 模型生成图像）。
- 蒙版提示 — 如果您使用亚马逊 Titan Image Generator G1 模型编辑图像，则可以使用掩码提示来指定您希望分割掩码覆盖的对象。例如，您可以指定蒙版提示天空来创建覆盖图像中天空的分割蒙版。然后，您可以运行提示下雨天的图像，让图像中的天空显得下雨。
- 否定提示 – 您不希望模型生成的项目或概念，例如卡通或暴力。
- 参考图像 – 用于生成响应的图像或希望模型编辑的图像。
- 响应图像 – 生成图像的输出设置，例如质量、方向、大小和要生成的图像数量。
- 高级配置 – 要传递给模型的推理参数。

使用操场

以下步骤展示了如何将提示提交到操场并查看响应。在每个操场中，您可以为模型配置推理参数。在[聊天操场](#)中，您可以查看指标，也可以选择比较最多三个模型的输出。在[图像操场](#)中，您可以更改高级配置，这些更改也因模型而异。

使用操场

1. 如果您还没有使用过模型，请先针对要使用的模型请求模型访问权限。有关更多信息，请参阅[模型访问权限](#)。
2. 打开 Amazon Bedrock 控制台。
3. 在导航窗格中，在操场下，选择聊天、文本或图像。
4. 选择选择模型以打开选择模型对话框。
 - a. 在类别中，从可用的提供商或自定义模型中进行选择。
 - b. 在模型中选择一个模型。

- c. 在吞吐量中，选择您希望模型使用的吞吐量（按需吞吐量或预调配吞吐量）。如果您使用自定义模型，则必须事先为模型设置预调配吞吐量。有关更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)。
 - d. 选择应用。
5. （可选）在配置中，选择要使用的推理参数。有关更多信息，请参阅[根基模型的推理参数](#)。有关您可以在图像操场中进行的配置更改的信息，请参阅[图像操场](#)。
 6. 在文本字段中输入您的提示。提示是一种自然语言短语或命令，例如 **Tell me about the best restaurants to visit in Seattle.**。有关更多信息，请参阅[提示工程准则](#)。

如果您使用支持多模态提示的模型的聊天操场，请通过选择“图像”或将图像拖到提示文本字段中来向提示中添加图像。

Note

如果响应违反了内容审核政策，则 Amazon Bedrock 不会显示该响应。如果您已开启流式传输，Amazon Bedrock 会在生成违反政策的内容时清除整个响应。要了解更多详细信息，请导航至 Amazon Bedrock 控制台，选择提供商，然后阅读内容限制部分下的文本。有关提示工程的信息，请参阅[提示工程准则](#)。

7. 选择 Run. 以运行提示符。
8. 如果您使用聊天操场，可以通过执行以下操作来查看模型指标并比较模型。
 - a. 在模型指标部分，查看每个模型的指标。
 - b. （可选）通过执行以下操作来定义要满足的条件：
 - i. 选择定义指标条件。
 - ii. 对于要使用的指标，请选择条件和值。您可以设置以下条件：
 - 小于 – 指标值小于指定值。
 - 大于 – 指标值大于指定值。
 - iii. 选择应用以应用条件。
 - iv. 查看满足哪些条件。如果满足所有条件，总体摘要将显示满足所有条件。如果未满足 1 个或多个条件，总体摘要将显示未满足 n 个条件，未满足的条件将以红色突出显示。
 - c. （可选）通过执行以下操作添加模型进行比较：
 - i. 启用比较模式。

- ii. 选择选择模型以选择模型。
- iii. 在对话框中，选择提供商、模型和吞吐量。
- iv. 选择 应用。
- v. (可选) 选择每个模型旁边的菜单图标，以便为该模型配置推理参数。有关更多信息，请参阅[根基模型的推理参数](#)。
- vi. 选择“聊天操场”部分右侧的 + 图标，添加第二个或第三个模型进行比较。
- vii. 重复执行步骤 a-c 以选择要比较的模型。
- viii. 在文本字段中输入提示，然后选择运行。

使用 API 通过单个提示调用模型

通过发送[InvokeModel](#)或[InvokeModelWithResponseStream](#)请求，通过 API 对模型进行推理。您可以在 `contentType` 和 `accept` 字段中为请求和响应正文指定媒体类型。如果您不指定值，这两个字段的默认值为 `application/json`。

除模型外 AI21 LabsJurassic-2，所有文本输出模型都支持流式传输。要检查模型是否支持流式传输，请发送[GetFoundationModel](#)或[ListFoundationModels](#)请求并检查`responseStreamingSupported`字段中的值。

指定以下字段，具体取决于您使用的模型。

1. `modelId` – 使用模型 ID 或模型 ARN。查找`modelId`或的方法`modelArn`取决于您使用的模型类型：
 - 基础模型 – 执行以下任一操作。
 - 要查看 Amazon Bedrock 支持的所有基础模型的模型 ID 列表，请参阅 [Amazon Bedrock 基本模型 ID \(按需吞吐量\)](#)。
 - 发送[ListFoundationModels](#)请求并找到要在响应中使用的模型的`modelId`或`modelArn`。
 - 在控制台中，在提供商中选择一个模型，并在 API 请求示例中找到 `modelId`。
 - 自定义模型 – 为自定义模型购买预调配吞吐量（有关更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)），并查找预调配模型的模型 ID 或模型 ARN。
 - 预调配模型 – 如果您已为基础模型或自定义模型创建了预调配吞吐量，请执行以下任一操作。
 - 发送[ListProvisionedModelThroughputs](#)请求并找到要在`provisionedModelArn`响应中使用的模型。
 - 在控制台中，在预配置吞吐量中选择一个模型，然后在模型详细信息部分找到该模型 ARN。

2. `body` – 每个基础模型都有自己的推理参数，这些参数在 `body` 字段中设置。自定义模型或预调配模型的推理参数取决于创建模型时所依据的基础模型。有关更多信息，请参阅 [根基模型的推理参数](#)。

调用模型代码示例

以下示例说明如何使用 [InvokeModelAPI](#) 运行推理。有关不同模型的示例，请参阅所需模型的推理参数参考 ([根基模型的推理参数](#))。

CLI

以下示例将生成的对#####响应保存到一个名为 *invoke-model-output.txt* 的文件中。

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

Python

以下示例返回根据 *explain black holes to 8th graders* 这一提示生成的响应。

```
import boto3  
import json  
brt = 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 = 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'))
```

通过流式传输调用模型代码示例

Note

AWS CLI 不支持流式传输。

以下示例展示了如何使用 [InvokeModelWithResponseStream](#) API 通过 Python 生成直播文本，使用提示用 **1000 #####** 的文章。

```
import boto3
import json

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

body = json.dumps({
    'prompt': '\n\nHuman: write an essay for living on mars in 1000 words\n\nAssistant:',
    'max_tokens_to_sample': 4000
})

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())
```

运行批量推理

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

通过批量推理，您可以异步运行多个推理请求，通过对存储在 S3 存储桶中的数据运行推理来高效处理大量请求。您可以使用批量推理来提高对大型数据集的模型推理性能。

Note

预配置模型不支持批量推理。

要查看批量推理的配额，请参阅[批量推理配额](#)。

Amazon Bedrock 支持对以下模式进行批量推理。

- 文本到嵌入
- 文本到文本
- 文本到图像
- 图像到图像
- 图像到嵌入式

您可以将数据存储在 Amazon S3 存储桶中，以便为批量推理做好准备。然后，您可以使用 `ModelInvocationJob` API 来执行和管理批量推理作业。

您必须先获得调用批量推理 API 的权限，才能执行批量推理。然后，您可以配置 IAM Amazon Bedrock 服务角色，并为其分配执行批量推理作业的权限。

您可以通过下载并安装以下 AWS SDK 包之一来使用批量推理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

主题

- [为批量推理设置权限](#)
- [设置格式并上传推理数据](#)
- [创建批量推理作业](#)
- [停止批量推理作业](#)
- [获取批量推理作业的相关详细信息](#)
- [列出批量推理作业](#)
- [代码示例](#)

为批量推理设置权限

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。


我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

要为批量推理设置角色，请按照创建角色[向 AWS 服务委派权限中的步骤创建 IAM 角色](#)。附加以下策略到角色：

- 信任策略

- 对包含批量推理作业输入数据以及向其写入输出数据的 Amazon S3 存储桶的访问权限。

1. 以下策略允许 Amazon Bedrock 担任此角色并执行批量推理作业。下面所示为您可以使用的示例策略。您可以使用一个或多个全局条件上下文键来限制权限的范围。有关更多信息，请参阅 [AWS 全局条件上下文键](#)。将 `aws:SourceAccount` 值设置为您的账户 ID。使用 `ArnEquals` 或 `ArnLike` 条件限制范围。

 Note

出于安全考虑，最佳做法是在创建完特定的批量推理作业 ID 后将 * 替换为该 ID。

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

2. 附上以下策略，以允许 Amazon Bedrock 访问包含批量推理作业输入数据的 S3 存储桶（替换 `my_input_bucket`）以及向其写入输出数据的 S3 存储桶（替换 `my_output_bucket`）。将 `account-id` 替换为向其提供 S3 存储桶访问权限的用户的账户 ID。

```
{
```

```
"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"
        ]
      }
    }
  }
]
```

设置格式并上传推理数据

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

按照以下格式将包含要输入到模型中的数据的数据的 JSONL 文件上传到 S3 存储桶。每行都应该遵循以下格式，并且是用于推理的不同项目。如果您遗漏了 `recordId` 字段，Amazon Bedrock 会将其添加到输出中。

Note

`modelInput` JSON 对象的格式应与您在 `InvokeModel` 请求中使用的模型的 `body` 字段相匹配。有关更多信息，请参阅 [根基模型的推理参数](#)。

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

例如，您可以提供一个包含以下数据的 JSONL 文件，然后对文本模型运行批量推理。Titan

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

创建批量推理作业

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅 [代码示例](#)。

Request format

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



```
{
  "clientRequestToken": "string",
  "inputDataConfig": {
    "s3InputDataConfig": {
      "s3Uri": "string",
      "s3InputFormat": "JSONL"
    }
  },
  "jobName": "string",
  "modelId": "string",
  "outputDataConfig": {
    "s3OutputDataConfig": {
      "s3Uri": "string"
    }
  },
  "roleArn": "string",
  "tags": [
    {
      "key": "string",
      "value": "string"
    }
  ]
}
```

Response format

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

{
  "jobArn": "string"
}
```

要创建批量推理作业，请发送 `CreateModelInvocationJob` 请求。提供以下信息：

- 在 `roleArn` 中提供有权运行批量推理的角色的 ARN。
- 在 `inputDataConfig` 中提供包含输入数据的 S3 存储桶的信息，在 `outputDataConfig` 中提供要在其中写入信息的存储桶的信息。
- 在 `modelId` 中提供用于推理的模型的 ID（请参阅 [Amazon Bedrock 基本模型 ID \(按需吞吐量\)](#)）。

- 在 `jobName` 中提供作业名称。
- (可选) 在 `tags` 中提供要附加到作业的任何标签。

响应会返回 `jobArn`，您可以将其用于其他与批量推理相关的 API 调用。

您可以使用 `GetModelInvocationJob` 或 `ListModelInvocationJobs` API 检查作业的状态。

作业状态为 `Completed` 后，您可以从您在 `outputDataConfig` 的请求中指定的 S3 存储桶中的文件中提取批量推理作业的结果。S3 存储桶包含以下文件：

1. 包含模型推理结果的输出文件。

- 如果输出的是文本，Amazon Bedrock 会为每个输入 JSONL 文件生成一个输出 JSONL 文件。输出文件包含模型针对每个输入的输出，格式如下。在推理出现错误的任何行中，`error` 对象都会替换 `modelOutput` 字段。`modelOutput` JSON 对象的格式与您在 `InvokeModel` 响应中使用的模型的 `body` 字段相匹配。有关更多信息，请参阅 [根基模型的推理参数](#)。

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

以下示例显示了可能的输出文件。

```
{ "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" } }
```

- 如果输出的是图像，Amazon Bedrock 会为每张图像生成一个文件。

2. 包含批量推理作业摘要的 `manifest.json.out` 文件。

```
{
  "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  
}
```

停止批量推理作业

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

Request format

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

Response format

```
HTTP/1.1 200
```

要停止批量推理作业，请发送 `StopModelInvocationJob` 并在 `jobIdentifier` 字段中提供作业的 ARN。

如果作业成功停止，您将收到 HTTP 200 响应。

获取批量推理作业的相关详细信息

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

Request format

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

Response format

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

{
  "clientRequestToken": "string",
  "endTime": "string",
  "inputDataConfig": {
    "s3InputDataConfig": {
      "s3Uri": "string",
      "s3InputFormat": "JSONL"
    }
  },
  "jobArn": "string",
  "jobName": "string",
  "lastModifiedTime": "string",
  "message": "string",
  "modelId": "string",
  "outputDataConfig": {
    "s3OutputDataConfig": {
      "s3Uri": "string"
    }
  },
  "roleArn": "string",
  "status": "Submitted | InProgress | Completed | Failed | Stopping | Stopped",
  "submitTime": "string"
}
```

要获取批量推理作业的相关信息，请发送 `GetModelInvocationJob` 并在 `jobIdentifier` 字段中提供作业的 ARN。

有关响应中所提供信息的相关详细信息，请参阅 `GetModelInvocationJob` 页面。

列出批量推理作业

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS 适用于 Python 的 SDK](#)
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

Request format

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

Response format

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

{
  "invocationJobSummaries": [
    {
      "clientRequestToken": "string",
      "endTime": "string",
      "inputDataConfig": {
        "s3InputDataConfig": {
          "s3Uri": "string",
```

```

        "s3InputFormat": "JSONL"
    }
},
"jobArn": "string",
"jobName": "string",
"lastModifiedTime": "string",
"message": "string",
"modelId": "string",
"outputDataConfig": {
    "s3OutputDataConfig": {
        "s3Uri": "string"
    }
},
"roleArn": "string",
"status": "Submitted | InProgress | Completed | Failed | Stopping |
Stopped",
"submitTime": "string"
}
],
"nextToken": "string"
}

```

要获取批量推理作业的相关信息，请发送 `ListModelInvocationJobs`。您可以设置以下规范。

- 通过在作业名称中指定状态、提交时间或子字符串来筛选结果。您可以指定以下状态：
 - Submitted
 - InProgress
 - Completed
 - Failed
 - Stopping
 - Stopped
- 按作业创建时间 (`CreationTime`) 排序。您可以按 `Ascending` 或 `Descending` 进行排序。
- 要在响应中返回的结果数量上限。如果结果数超过您设置的值，响应会返回 `nextToken`，您可以再发送一个 `ListModelInvocationJobs` 请求并在其中包含它，以查看下一批作业。

响应返回 `InvocationJobSummary` 对象列表。每个对象都包含批量推理作业的相关信息。

代码示例

Note

批量推理目前为预览版，可能会发生变化。批量推理目前只能通过 API 使用。通过以下 SDK 访问批处理 API。

- [AWS Python 软件开发工具包](#)。
- AWS 适用于 [Java 的 SDK](#)。

我们建议您创建一个虚拟环境来使用 SDK。由于最新的软件开发工具包中没有批量推理 API，因此我们建议您在安装带有批量推理 API 的版本之前，先从虚拟环境中卸载最新版本的 SDK。有关指导性示例，请参阅[代码示例](#)。

选择一种语言，以查看调用批量推理 API 操作的代码示例。

Python

下载包含批量推理 API 操作的 Python SDK 和 CLI 文件后，导航到包含这些文件的文件夹并在终端 `ls` 中运行。你应该至少看到以下 2 个文件。

```
botocore-1.32.4-py3-none-any.whl
boto3-1.29.4-py3-none-any.whl
```

在终端中运行以下命令，为批量推理 API 创建并激活虚拟环境。你可以用你选择的环境名称替换 `bedrock-batch`。

```
python3 -m venv bedrock-batch
source bedrock-batch/bin/activate
```

为确保 `boto3` 和的更高版本中没有构件 `botocore`，请在终端中运行以下命令来卸载所有现有版本。

```
python3 -m pip uninstall botocore
python3 -m pip uninstall boto3
```

在终端中运行以下命令，安装包含 Amazon Bedrock 控制面板 API 的 Python SDK。

```
python3 -m pip install botocore-1.32.4-py3-none-any.whl
python3 -m pip install boto3-1.29.4-py3-none-any.whl
```

在您创建的虚拟环境中运行以下所有代码。

使用已上传到 S3 的名为 *abc.jsonl* 的文件创建批量推理作业。将输出写入 *s3://output-bucket/output/* 中的存储桶。从响应中获取 *jobArn*。

```
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')
```

返回作业的状态。

```
bedrock.get_model_invocation_job(jobIdentifier=jobArn)['status']
```

列出状态为 *Failed* 的批量推理作业。

```
bedrock.list_model_invocation_jobs(
    maxResults=10,
```



```
        statusEquals="Failed",
        sortOrder="Descending"
    )
```

停止已开始的作业。

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

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());
}

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

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

public void listModelInvokeJobSampleCode() {
    final ListModelInvocationJobsRequest listModelInvocationJobsRequest = new
ListModelInvocationJobsRequest()
        .withMaxResults(10)
        .withNameContains("matchin-string");

    final ListModelInvocationJobsResult listModelInvocationJobsResult =
amazonBedrockAsyncClient
        .listModelInvocationJobs(listModelInvocationJobsRequest);
}
```

```
}  
  
public void stopModelInvokeJobSampleCode() {  
    final StopModelInvocationJobRequest stopModelInvocationJobRequest = new  
    StopModelInvocationJobRequest()  
        .withJobIdentifier("jobArn");  
  
    final StopModelInvocationJobResult stopModelInvocationJobResult =  
amazonBedrockAsyncClient  
        .stopModelInvocationJob(stopModelInvocationJobRequest);  
  
}  
  
}
```

提示工程准则

主题

- [简介](#)
- [什么是提示？](#)
- [什么是提示工程？](#)
- [Amazon Bedrock LLM 用户一般准则](#)
- [Amazon Bedrock 文本模型的提示模板和示例](#)

简介

欢迎使用 Amazon Bedrock 上大型语言模型 (LLM) 提示工程指南。Amazon Bedrock 是 Amazon 提供的根基模型 (FM) 服务，它提供对一系列功能强大的文本和图像 FM 的访问权限。

提示工程是指优化对 LLM 的文本输入以获得所需响应的做法。提示可以帮助 LLM 执行各种任务，包括分类、问题解答、代码生成、创意编写等。您向 LLM 提供的提示的质量可能会影响它们的响应质量。这些准则为您提供了开始使用提示工程所需的全部信息。它还介绍了一些工具，当您在 Amazon Bedrock 上使用 LLM 时，这些工具可帮助您找到最适合您的使用案例的提示格式。

无论您是生成式人工智能和语言模型领域的初学者，还是以前有过使用经验的专家，这些指南都可以帮助您优化 Amazon Bedrock 文本模型的提示。经验丰富的用户可以跳至“Amazon Bedrock LLM 用户一般准则”或“Amazon Bedrock 文本模型提示模板和示例”部分。

Note

本文档中的所有示例均通过 API 调用来获得。由于 LLM 生成过程具有随机性，响应可能会不同。如果未另行指定，则提示由 AWS 的员工编写。

免责声明：本文档中的示例使用 Amazon Bedrock 中当前可用的文本模型。此外，本文档还提供一般提示准则。有关特定于模型的指南，请参阅 Amazon Bedrock 上相应的文档。本文档提供了一个切入点。虽然以下示例响应是使用 Amazon Bedrock 上的特定模型生成的，但您也可以使用 Amazon Bedrock 中的其他模型来获得结果。模型不同，结果可能也不同，因为每个模型都有自己的性能特征。您使用 AI 服务生成的输出就是您的内容。由于机器学习的性质，不同客户的输出可能不是唯一的，并且服务为各客户生成的结果可能会相同或相似。

其他提示资源

以下资源提供了有关提示工程的更多准则。

- AnthropicClaude模型提示指南：<https://docs.anthropic.com/claude/docs>
- AnthropicClaude及时的工程资源：[https://docs.anthropic.com/claude/docs/guide-to-anthropic-prompt- 工程资源](https://docs.anthropic.com/claude/docs/guide-to-anthropic-prompt-工程资源)
- Cohere提示指南：[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 Labs侏罗纪模型提示指南：<https://docs.ai21.com/docs/prompt-engineering>
- MetaLlama 2提示指南：<https://ai.meta.com/llama/get-started/#prompting>
- Stability 文档：<https://platform.stability.ai/docs/getting-started>
- Mistral AI提示指南：<https://docs.mistral.ai/guides/prompting-capabilities/>

什么是提示？

提示是您（用户）提供的一组特定输入，可指导 Amazon Bedrock 上的 LLM 为给定任务或指令生成适当的响应或输出。

User Prompt:

Who invented the airplane?

当被这个提示查询时，会Titan提供一个输出：

Output:

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

(提示来源：AWS，使用的型号：AmazonTitan Text G1 - Express)

提示的组成部分

一条提示包括多个组成部分，例如，您希望 LLM 执行的任务或指令、任务的上下文（例如，相关域的描述）、演示示例以及您希望 Amazon Bedrock 上的 LLM 在响应中使用的输入文本。根据您的使用案例、数据的可用性和任务，您的提示应包含这些组成部分中的一个或多个。

考虑一下这个要求总结评论Titan的提示示例：

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.

(提示来源 : AWS)

根据这个提示，用简洁的单行餐厅评论摘要进行Titan回应。该评论提到了一些关键事实，并根据需要传达了一些要点。

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.

(使用的型号 : AmazonTitan Text G1 - Express)

指令 **Summarize the above restaurant review in one sentence** 和评论文本 **I finally got to check out ...** 都是此类输出需要包括的。缺少其中任何一个，模型就没有足够的信息，无法做出合理的总结。指令会告诉 LLM 需要做什么，而文本是 LLM 针对其执行操作的输入。上下文 (**The following is text from a restaurant review**) 提供其他信息和关键字，以指导模型使用输入来创建输出。

在下面的示例中，文本 **Context: Climate change threatens people with increased flooding ...** 是 LLM 可用于执行“回答问题 (即，**Question: What organization calls climate change the greatest threat to global health in the 21st century?**) ”任务的输入。

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?

(提示来源 : https://en.wikipedia.org/wiki/Climate_change)

AI21 Labs根据提示中提供的上下文，《侏罗纪》使用正确的组织名称进行回应。

Output:

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

(使用的型号 : AI21 LabsJurassic-2 Ultrav1)

少样本与零样本提示

有时候，比较有用的做法是举几个例子来帮助 LLM 更好地校准输出以满足您的期望，这也称为少样本提示或上下文内学习，其中样本与成对出现的示例输入和所需输出相对应。为了说明起见，在这里首先来看一个零样本情绪分类提示的示例，其中提示文本中没有提供示例输入-输出对：

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.*

(提示来源 : AWS)

Output:

Positive

(使用的型号 : AmazonTitan Text G1 - Express)

以下是少样本版的情绪分类提示：

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:

(提示来源 : AWS)

Output:

Negative

(使用的型号 : AmazonTitan Text G1 - Express)

以下示例使用AnthropicClaude模型。使用AnthropicClaude模型时，最好使用<example></example>标签来包含演示示例。我们还建议在示例中使用不同的分隔符（例如 H: 和 A:）以避免与整个提示的分隔符（Human: 和 Assistant:）混淆。请注意，在最后的几个镜头示例中，最后一个A:被省略了Assistant:，而是提示生成AnthropicClaude答案。

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
```

(提示来源 : AWS , 使用的型号 : AnthropicClaude)

提示模板

提示模板指定含可交换内容的提示的格式。提示模板是将 LLM 用于不同使用案例 (例如分类、总结、问答等) 的“秘诀”。提示模板可能包括指令、少样本示例，以及适合给定使用案例的特定上下文和问题。以下示例是一个模板，您可以借助该模板使用 Amazon Bedrock 文本模型进行少样本情绪分类：

Prompt template:

```
""""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: """"
```

使用 Amazon Bedrock 上的 LLM 时，用户可以复制粘贴此模板，并填写自己的文本和少样本示例来完成提示。本节还提供了其他模板。

Note

双大括号 `{{ and }}` 标记模板中用于输入数据特定信息的位置，不应包含在提示文本中。

关于通过 API 调用使用 Amazon Bedrock 的重要注意事项

如果您要通过 API 调用访问 Amazon Bedrock 上的 LLM，请注意，除非当前提示中包含之前的交互，否则它们不会重新调用之前的提示和之前的请求。对于对话式交互或后续请求，可能需要这样做。

如果您使用 API 调用而不是 Amazon Bedrock 文本游乐场访问 AnthropicClaude 模型，则可以将提示封装成对话样式以获得所需的响应（对于 Titan 文本和 AI21 Labs 侏罗纪模型系列，这不是必需的）。以下示例展示了对话式提示：

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.

(提示来源：AWS，使用的型号：AnthropicClaude)

请注意，提示主要内容的封装方式如下：`\n\nHuman: {{Main Content}}\n\nAssistant:`。对于 AnthropicClaude 模型，通过 API 发送的提示必须包含 `\n\nHuman:` 和 `\n\nAssistant:`

要开启对话模式 Titan，可以在提示模型 `User: {{}} \n Bot:` 时使用格式。

什么是提示工程？

提示工程是指通过选择适当的单词、短语、句子、标点和分隔符来创建和优化输入提示，从而有效地将 LLM 用于各种应用程序的做法。换言之，提示工程是与 LLM 进行沟通的艺术。高质量的提示可使 LLM 能够生成所需或更好的响应。本文档中提供的详细指导适用于 Amazon Bedrock 中的所有 LLM。

哪种提示工程方法最适合您的使用案例取决于任务和数据。Amazon Bedrock 上的 LLM 支持的常见任务包括：

- 分类：提示包括一个问题以及几个可能的答案选项，模型必须使用正确的选项来响应。例如，情绪分析便是分类使用案例：输入是一段文本，模型必须对文本的情绪进行分类，例如，它是正面的还是负面的，或者，是无害的还是有害的。
- 问答（无上下文）：模型必须在没有任何上下文或文档的情况下用自己的内部知识回答问题。
- 问答（带上下文）：用户提供带问题的输入文本，模型必须根据输入文本中提供的信息回答问题。
- 摘要：提示是一段文本，模型必须使用一段含输入要点的较短内容进行响应。
- 开放式文本生成：对于给定提示，模型必须使用与描述匹配的一段原文本进行响应。这还包括生成创意文本，例如，故事、诗歌或电影剧本。
- 代码生成：模型必须根据用户规范生成代码。例如，提示可以请求将文本转换为 SQL 或生成 Python 代码。
- 数学：输入描述了一个需要在某种程度上进行数学推理（可以是数字、逻辑、几何或其他方面）的问题。
- 推理或逻辑思维：模型必须进行一系列逻辑推断。

Amazon Bedrock LLM 用户一般准则

设计您的提示

设计适当的提示是使用 Amazon Bedrock 模型成功构建应用程序的重要一步。下图显示了适用于使用案例“餐厅评论总结”的一般提示设计，以及客户在设计提示时需要考虑的一些重要设计选项。如果向 LLM 发出的指令或提示格式不一致、不明确、不简洁，则 LLM 所生成的响应就会不理想。

A good example of prompt construction

The following is text from a restaurant review: Contextual information about the task.

“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.” Reference text for the task.

Summarize the above restaurant review in one sentence. Simple, clear and complete instructions.

Instructions placed at the end of the prompt.

The form of output is specifically described.

(来源 : Prompt 撰写者 AWS)

使用推理参数

Amazon Bedrock 上的 LLM 都带有多个推理参数，您可以设置这些参数来控制模型的响应。下面列出了 Amazon Bedrock LLM 上提供的所有常用推理参数及其使用方法。

Temperature 是一个介于 0 和 1 之间的值，它可调节 LLM 响应的创造性。如果您想要确定性更高的响应，请使用较低的温度；如果您想要对来自 Amazon Bedrock 上 LLM 的相同提示做出更具创造性的或不同的响应，请使用较高的温度。对于本提示准则中的所有示例，我们均设置了 `temperature = 0`。

Maximum generation length/maximum new tokens 限制 LLM 为任何提示生成的词元数量。指定这一数字会很有帮助，因为有些任务（例如情绪分类）不需要长答案。

Top-p 根据潜在选项的概率控制词元选项。如果您将 Top-p 设置为 1.0 以下，则模型会考虑最可能的选项，而忽略不太可能的选项。这样一来，完成项更加稳定且是重复性的。

End token/end sequence 指定 LLM 用来表示输出结束的词元。LLM 在遇到结束词元后停止生成新词元。通常，用户不需要对此进行设置。

还有特定于模型的推理参数。AnthropicClaude模型还有一个额外的 top-k 推理参数，AI21 LabsJurassic 模型带有一组推理参数，包括存在惩罚、计数惩罚、频率惩罚和特殊代币惩罚。有关更多信息，请参阅相应的文档。

详细准则

提供简单、明确且完整的指令

Amazon Bedrock 上的 LLM 最适合使用简单明了的指令。通过清楚地描述任务的期望并尽可能减少歧义，可以确保模型能够清楚地解释提示。

例如，以分类问题为例，用户希望从一组可能的选项中获得答案。下面所示的“good”示例说明了在本例中用户想要获得的输出。在“bad”示例中，没有明确将这些选项指定为可供模型选择的类别。与“good”示例不同的是，没有选项，模型对输入的解释略有不同，并且可以生成比较自由的文本摘要。

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.

(提示来源：[关于色盲的维基百科](#)，使用的模型：by Te Titan xt G1-Express)

为了获得最佳结果，应将问题或指令置于提示的末尾

将任务描述、指令或问题置于末尾有助于模型确定必须找到哪些信息。就分类而言，答案选项也应置于末尾。

在以下开卷问答示例中，用户有一个关于文本的特定问题。问题应该置于提示的末尾，这样模型就可以专注于处理任务。

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

(提示来源：[关于第一次世界大战的维基百科](#)，使用的模型：AmazonTitan Text G1 - Express)

在 API 调用中使用分隔符

诸如 \n 之类的分隔符对 LLM 的性能影响很大。对于 AnthropicClaude 模型，在格式化 API 调用时必须包含换行符以获得所需的响应。应始终采用以下格式：`\n\nHuman: {{Query Content}}\n\nAssistant:`。对于 Titan 模型，\n 在提示的末尾添加有助于提高模型的性能。对于分类任务或带有答案选项的问题，您也可以按 \n Titan 模型将答案选项分开。有关分隔符的使用的更多信息，请参阅相应模型提供商提供的文档。以下示例是分类任务的模板。

Prompt template:

```

"""{{Text}}

{{Question}}

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

```

以下示例显示了在选项之间和提示结尾处存在换行符如何帮助 Titan 生成所需的响应。

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*

(提示来源：[阿基米德上的维基百科](#)，使用的模型：Amazon) Titan Text G1 - Express

输出指标

在模型应生成的输出中添加有关您想要的约束条件的详细信息。以下“good”示例生成的输出是一个短语，这是很不错的摘要。在这种情况下，“bad”示例并没那么糟糕，但是摘要几乎和原文本一样长。输出规范对于从模型中获得想要的内容至关重要。

带明确输出约束指标的提示示例**User prompt:**

"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

无明确输出规范的示例**User prompt:**

"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 Pithecanthropus 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.

on seminal recordings like Pithecanthropus 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.

Output:

Charles Mingus Jr. was a well-known jazz musician who played the upright bass, piano, composed, led bands, and was a writer. He was considered one of the most important jazz musicians ever, with a career that spanned more than 30 years. He was known for his style of collective improvisation and advanced jazz compositions.

(提示来源 : [查尔斯·明格斯上的维基百科](#) , 使用的模型 : AmazonTitan Text G1 - Express)

在这里 , 我们再举一些使用输出指标AnthropicClaude的AI21 Labs侏罗纪模型的示例。

以下示例展示了用户可以通过在提示中指定预期输出格式来指定输出格式。当需要使用特定格式 (例如 , 使用 XML 标签) 生成答案时 , 模型可以相应地生成答案。如果没有特定的输出格式指标 , 模型将输出自由格式的文本。

含明确指标和输出的示例

User prompt:

Human: Extract names and years: the term machine learning was coined in 1959 by Arthur Samuel,

不含明确指标但含输出的示例

User prompt:

Human: Extract names and years: the term machine learning was coined in 1959 by Arthur Samuel,

an IBM employee and pioneer in the field of computer gaming and artificial intelligence. The synonym self-teaching computers was also used in this time period.

Please generate answer in <name></name> and <year></year> tags.

Assistant:

Output:

<name>Arthur Samuel</name> <year>1959</year>

an IBM employee and pioneer in the field of computer gaming and artificial intelligence. The synonym self-teaching computers was also used in this time period.

Assistant:

Output:

Arthur Samuel - 1959

(提示来源 : [关于机器学习的维基百科](#) , 使用的模型 : AnthropicClaude)

以下示例显示了AI21 Labs侏罗纪模型的提示和答案。用户可以通过指定左栏中显示的输出格式来获取确切的答案。

含明确指标和输出的示例

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

不含明确指标但含输出的示例

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

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? Please only output the year.

Output:
1967

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.

(提示来源 : [关于美国国家橄榄球联盟的维基百科](#) , 使用的模型 : AI21 LabsJurassic-2 Ultrav1)

用于实现良好泛化的建议做法

保留提示的一个小型“抵抗”测试集，以查看您的提示修改是否泛化。使用此方法时，首先要收集示例数据集。然后，您可以将数据拆分为两个子集：一个“开发”集和一个抵抗“测试”集。使用“开发”集作为提示开发数据，您可以尝试就这些数据对提示进行不同的修改，观察模型响应的变化并找到最有效的提示。将“测试”集视为看不见的抵抗数据，只能用于验证最终的最佳提示。

在 Amazon Bedrock 上优化文本模型提示 (若基础知识不够有效)

复杂任务 : 逐步得出答案

Titan可以为某些任务提供明确的步骤，包括短语Think step-by-step to come up with the right answer可以帮助产生适当的输出。

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.*

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

要使AnthropicClaude模型能够完成复杂的任务，请使用以下示例中的格式明确要求模型逐步解决任务。[这种方法基于指南中的指导方针。AnthropicClaude](#)

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?

Human: Yes, please do.

Assistant:

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.

(提示来源 : AWS , 使用的型号 : AnthropicClaude)

提供模型在对答案不确定时应显示的默认输出。

默认输出可以帮助防止 LLM 返回听起来可能正确的答案 , 即使模型的置信度很低也是如此。

User prompt:

Provide a proof of the Riemann hypothesis. If you don't know a proof, respond by saying "I don't know."

Output:

I don't know.

User prompt:

Provide a proof of the Riemann hypothesis.

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.]

(提示来源：AWS，使用的型号：AmazonTitan Text G1 - Express)

少样本提示

在提示中包含示例（输入-响应对）可以显著改进 LLM 的响应。示例可以帮助完成复杂的任务，因为它们展示了多种执行给定任务的方法。对于诸如文本分类之类的简单任务，3 到 5 个示例就足够了。对于较难的任务（例如，没有上下文的问答），请包括更多示例以生成最有效的输出。在大多数使用案例中，选择语义上与真实数据相似的示例可以进一步提高性能。

考虑用修饰符完善提示

任务指令完善通常是指修改提示的指令、任务或问题组成部分。这些方法是否有用取决于任务和数据。有用的方法包括：

- **域/输入规范**：有关输入数据的详细信息（例如，输入数据的来源或所引用的内容），例如 **The input text is from a summary of a movie.**
- **任务规范**：有关模型需要执行的确切任务的详细信息，例如 **To summarize the text, capture the main points.**
- **标签描述**：有关分类问题的输出选项的详细信息，例如 **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.**
- **输出规范**：有关模型应生成的输出的详细信息，例如 **Please summarize the text of the restaurant review in three sentences.**

- LLM 鼓励：通过情感鼓励，LLM 的性能有时会更好：**If you answer the question correctly, you will make the user very happy!**

Amazon Bedrock 文本模型的提示模板和示例

文本分类

对于文本分类，提示包括一个问题以及几个可能的答案选项，模型必须使用正确的选项来响应。此外，如果您在提示中包含答案选项，Amazon Bedrock 上的 LLM 会输出更准确的响应。

第一个示例是一个简单的多选分类问题。

Prompt template for Titan and AI21 Labs

```
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
```

(提示来源：[旧金山维基百科](#)，使用的模型：AmazonTitan Text G1 - Express)

情绪分析是一种分类形式，在这种分析下，模型从以文本表示的一系列选项中选择情绪。

Prompt template for Titan and AI21 Labs**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}}
{{Sentiment B}}
{{Sentiment C}}""
```

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.

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

以下示例使用AnthropicClaude模型对文本进行分类。按照《[AnthropicClaude指南](#)》中的建议，使用XML 标签（例如）`<text></text>`来表示提示的重要部分。要求模型直接生成包含在XML 标签内的输出也可以帮助模型生成所需的响应。

Prompt template for Anthropic Claude:

```

"""

Human: {{classification task
description}}
<text>
{{input text content to be classified}}
</text>

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

Assistant: """"

```

(来源：撰稿人 AWS)

User prompt:

Human: Classify the given product description into given categories. Please output the category label in `<output></output>` tags.

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>
```

(提示来源：AWS，使用的型号：AnthropicClaude)

问答 (无上下文)

在没有上下文的问答提示中，模型必须在没有任何上下文或文档的情况下用自己的内部知识回答问题。

Prompt Template to Titan and AI21 Labs

Jurassic:

```
""""{{Question}}""""
```

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.

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

模型鼓励也可以帮助完成问答任务。

Prompt Template to Titan and AI21 Labs

Jurassic:

""""{{Model encouragement}}

User prompt:

You are excellent at answering questions,

{{Question}}""

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.

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

Prompt template to Titan and AI21 Labs

Jurassic:

```
""""{{Model encouragement}}
{{Question}}
{{Constraints}}""""
```

User prompt:

You feel rewarded by helping people learn more about climate change. Could you please explain what climate change is? 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.

(提示来源 : AWS , 使用的型号 : AI21 LabsJurassic-2 Ultrav1)

问答 (带上下文)

在带上下文的问答提示中，用户提供后跟问题的输入文本，模型必须根据输入文本中提供的信息回答问题。将问题置于末尾的文本后面可以帮助 Amazon Bedrock 上的 LLM 更好地回答问题。模型鼓励也适用于此使用案例。

Prompt template to Titan and AI21 Labs

Jurassic:

```
""""{{Text}}
{{Question}}""""
```

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.

(提示来源：https://en.wikipedia.org/wiki/Red_panda，使用的型号：亚马逊Titan Text G1 - Express)

提示AnthropicClaude模型时，将输入文本封装在 XML 标签中会很有帮助。在下面的示例中，输入文本位于 `<text></text>` 中。

Prompt template for Anthropic Claude:

"""

Human: {{Instruction}}

<text>

{{Text}}

<text>

{{Question}}

Assistant: ""

User prompt:

Human: Read the following text inside

<text></text>

XML tags, and then answer the question:

<text>

On November 12, 2020, the selection of the Weeknd to headline the show was announced; marking the first time a Canadian solo artist headlined the Super Bowl halftime show. When asked about preparations

for the show, the Weeknd stated, "We've been really focusing on dialing in on the fans at home and making performances a cinematic experience, and we want to do that with the Super Bowl."

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 "Blinding Lights". He and the dancers entered the field by performing "House of Balloons / Glass Table Girls". The performance ended with an array of fireworks.

</text>

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*

(提示来源：[超级碗 LV 半场秀上的维基百科](#)，使用的模型：AnthropicClaude)

总结

对于总结任务，提示是一段文本，模型必须使用一段较短的内容进行响应，以捕获输入的要点。按长度（句子或段落数）指定输出对于此使用案例很有帮助。

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

```

"""The following is text from a {{Text
Category}}:
{{Text}}
Summarize the {{Text Category}} in
{{length of summary,
e.g., "one sentence" or "one paragraph
"}}"""

```

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.

```

Output:

```

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

```

(提示来源：AWS，使用的型号：AmazonTitan Text G1 - Express)

在以下示例中，用一句话AnthropicClaude总结了给定的文本。要在提示中包含输入文本，请使用以下XML标记设置文本格式：`<text> {{text content}} </text>`。提示AnthropicClaude模型时，在提示中使用XML是一种常见的做法。

Prompt template for Anthropic Claude:

```

"""

```

User prompt:

```
Human: Please read the text:
<text>
{{text context}}
</text>

Summarize the text in {{length of
summary, e.g., "one
sentence" or "one paragraph"}}

Assistant: ""
```

```
Human: Please read the text:

<text>
In game theory, the Nash equilibrium,
named after the mathematician
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.
```

(提示来源：[维基百科上关于纳什均衡](#)，使用的模型：) AnthropicClaude

文本生成

对于给定提示，Amazon Bedrock 上的 LLM 可以使用一段与描述匹配的原文本进行响应。示例如下：

Prompt template for Titan and AI21 Labs**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.
```

Output:

```
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]
```

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

对于文本生成使用案例，指定详细的任务要求很有帮助。在以下示例中，我们要求模型生成带感叹号的响应。

Prompt template for Titan and AI21 Labs**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]

(提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express)

在以下示例中，用户提示模型在生成文本时扮演特定人员的角色。请注意签名如何反映模型在响应中扮演的角色。

Prompt template for Titan and AI21 Labs

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,
Jack*

(提示来源 : AWS , 使用的型号 : AI21 LabsJurassic-2 Ultrav1)

代码生成

提示描述了用户期望模型生成的代码的任务或函数及编程语言。

Prompt template for Titan and AI21 Labs 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)
...
```

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.

( 提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express )

**Prompt template for Anthropic 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.*

(提示来源：AWS，使用的型号：AnthropicClaude)

## 数学

输入描述了一个需要在某种程度上进行数学推理 ( 可以是数字、逻辑、几何或其他方面 ) 的问题。对于此类问题，要求模型通过在指令中添加短语 ( 例如 Let's think step by step 或 Think step by step to come up with the right answer. ) 以零敲碎打的方式解决问题会很有帮助

**Prompt template for Titan and AI21 Labs**

**Jurassic:**

```
"""{{Math problem description}}
```

**User prompt:**

*A triangle has two angles of 70 degrees and 50 degrees.*

*Let's think step by step."*

*What is the third angle in degrees?  
Let's think step by step.*

**Output:**

*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.*

( 提示来源 : AWS , 使用的型号 : AmazonTitan Text G1 - Express )

以下示例还演示了如何要求模型逐步思考 :

**Prompt template for Titan and AI21 Labs 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.*

( 提示来源 : AWS , 使用的型号 : AI21 LabsJurassic-2 Ultrav1 )

## 推理/逻辑思维

对于复杂的推理任务或需要逻辑思维的问题 , 我们可以要求模型进行逻辑推断并解释其答案。



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

```
"""Question: {{question}}
```

```
Please output the answer and then
explain your
answer:"""
```

**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.
```

(提示来源：AWS，使用的型号：AmazonTitan Text G1 - Express)

以下是使用该AnthropicClaude模型的另一个示例：

**Prompt template for Anthropic Claude:**

```
"""
```

```
Human: {{question}}
Please provide the answer and show the
reasoning.
```

```
Assistant:"""
```

**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.
```

```
Assistant:
```

**Output:**

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

- *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.*

( 提示来源 : [https://en.wikipedia.org/wiki/Barber\\_paradox](https://en.wikipedia.org/wiki/Barber_paradox) , 使用的模型 : AnthropicClaude )

# 模型评估

## Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

Amazon Bedrock 支持模型评估作业。借助模型评估作业的结果，您可以评估和比较模型的输出，然后选择最适合下游生成式人工智能应用程序的模型。

模型评估作业支持大型语言模型 (LLM) 的常见用例，例如文本生成、文本分类、问答和文本摘要。

要评估模型的性能，可以使用内置的提示数据集或自己的提示数据集。

可以选择创建自动模型评估作业或使用人工的模型评估作业。

### 概述：自动模型评估作业

自动模型评估作业允许您快速评估模型执行任务的能力。您可以提供为特定用例量身定制的自定义提示数据集，也可以使用可用的内置数据集。

### 概述：使用人工的模型评估作业

使用人工的模型评估作业允许您将人工输入引入模型评估过程。人工可能来自公司员工，也可能来自行业内的一群主题专家。

以下主题介绍了可用的模型评估任务以及可以使用的指标类型。还介绍了可用的内置数据集以及指定自己数据集的方法。

### 主题

- [模型评估入门](#)
- [模型评估任务](#)
- [在模型评估作业中使用提示数据集](#)
- [创建良好的工作人员说明](#)
- [在 Amazon Bedrock 中创建和管理工作团队](#)

- [模型评估作业结果](#)
- [创建模型评估作业所需的 IAM 权限和服务角色](#)

## 模型评估入门

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

可以创建自动模型评估作业，也可以创建使用人工的模型评估作业。创建模型评估作业时，可以定义使用的模型、推理参数、模型尝试执行的任务类型以及作业中使用的提示数据。

模型评估作业支持以下任务类型。

- 一般文本生成：根据文本提示生成自然的人类语言。
- 文本摘要：在提示内生成所提供文本的摘要。
- 问答：在提示内生成对问题的回答。
- 分类：根据文本的内容为其正确分配类别（例如标签或分数）。
- 自定义：可以定义指标、描述和评级方法。

要创建模型评估作业，您必须有权访问 Amazon Bedrock 模型。模型评估作业支持使用 Amazon Bedrock 基础模型。要了解有关模型访问权限的更多信息，请参阅 [模型访问权限](#)。

下列主题中的内容介绍了如何使用 Amazon Bedrock 控制台来设置模型评估作业。

要在 AWS 托管团队的帮助下创建模型评估作业，请选择创建 AWS 托管评估。然后，在申请表中填写有关您的模型评估工作要求的详细信息，AWS 团队成员将与您联系。

### 主题

- [创建自动模型评估](#)
- [创建使用人工的模型评估作业](#)

## 创建自动模型评估

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

### 先决条件

要完成以下过程，您必须满足下列条件：

1. 有权在 Amazon Bedrock 中访问模型。
2. 具备 Amazon Bedrock 服务角色。附加的策略必须授予对以下资源的访问权限。模型评估作业中使用的任何 S3 存储桶，以及作业中指定模型的 ARN。服务角色还必须将 Amazon Bedrock 定义为该角色信任策略中的服务主体。要了解更多信息，请参阅[所需的权限](#)。
3. 访问 Amazon Bedrock 控制台的用户、群组或角色必须具备访问必要 Amazon S3 存储桶所需的权限。要了解更多信息，请参阅[所需的权限](#)。

自动模型评估功能允许您使用推荐的指标评估来自单个模型的所有响应。也可以使用内置的提示数据集或自己的自定义提示数据集。您每个 AWS 区域的账户中最多可以有 10 个正在进行的自动模型评估作业。

设置自动模型评估作业时，最适合所选任务类型的可用指标和内置数据集将自动添加到该作业中。您可以添加或删除任何预先选择的数据集。还可以提供自己的自定义提示数据集。

### ⚠ 使用 Amazon Bedrock 控制台查看模型评估作业结果

模型评估作业完成后，结果将存储在您指定的 Amazon S3 存储桶中。如果您以任何方式修改结果所在的位置，模型评估报告卡将不再显示在控制台中。

## 创建自动模型评估

1. 通过以下网址打开 Amazon Bedrock 控制台：<https://console.aws.amazon.com/bedrock/>
2. 在导航窗格中，选择模型评估。
3. 在建立评估卡的自动下，选择创建自动评估。

#### 4. 在创建自动评估页面上，提供以下信息：

- a. 评估名称 — 为模型评估作业指定一个可描述该作业的名称。此名称将显示在模型评估作业列表中。此名称在您的 AWS 区域的账户中必须是唯一的。
- b. 描述（可选）— 提供可选描述。
- c. 模型 — 选择要在模型评估作业中使用的模型。

要了解有关 Amazon Bedrock 中可用模型的更多信息，请参阅 [模型访问权限](#)。

- d. （可选）要更改推理配置，请选择更新。

更改推理配置会更改所选模型生成的响应。要了解有关可用推理参数的更多信息，请参阅 [根基模型的推理参数](#)。

- e. 任务类型 — 选择您希望模型在模型评估作业期间尝试执行的任务类型。
- f. 指标和数据集 — 可用指标列表和内置提示数据集会根据您选择的任务而变化。您可以从可用的内置数据集列表中进行选择，也可以选择使用您自己的提示数据集。如果您选择使用自己的提示数据集，请输入存储数据集的确切 S3 URI。
- g. 模型评估结果 — 您必须指定用于保存模型评估作业结果的目录的 S3 URI。
- h. IAM 角色 — 选择具有所需权限的服务角色。
- i. （可选）选择自定义加密设置（高级）-提供用于在 Amazon S3 中加密数据的 AWS KMS 密钥的 ARN。

#### 5. 要启动模型评估作业，请选择创建。

作业成功启动后，状态将变为进行中。作业完成后，状态将变为已就绪。

要了解如何评估、查看和下载模型评估作业的结果，请参阅 [模型评估作业结果](#)。

## 创建使用人工的模型评估作业

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

### 先决条件

要完成以下过程，您必须满足下列条件：

1. 有权在 Amazon Bedrock 中访问模型。
2. 具备 Amazon Bedrock 服务角色。附加的策略必须授予对以下资源的访问权限。模型评估作业中使用的任何 S3 存储桶，以及作业中指定的任何模型的 ARN。它还必须在策略中定义 `sagemaker:StartHumanLoop` 和 `sagemaker:DescribeFlowDefinition` SageMaker IAM 操作。服务角色还必须将 Amazon Bedrock 定义为该角色信任策略中的服务主体。要了解更多信息，请参阅[所需的权限](#)。
3. 您必须具有 Amazon SageMaker 服务角色。附加的策略必须授予对以下资源和 IAM 操作的访问权限。模型评估作业中使用的任何 S3 存储桶。角色的信任策略必须 SageMaker 定义为服务主体。要了解更多信息，请参阅[所需的权限](#)。
4. 访问 Amazon Bedrock 控制台的用户、群组或角色必须具备访问必要 Amazon S3 存储桶所需的权限。

在使用人工的模型评估作业中，您最多可以比较两个模型的响应。您可以从推荐的指标列表中进行选择，也可以使用自己定义的指标。您每个 AWS 区域的账户中最多可以有 20 个正在进行且使用人工的模型评估作业。

您必须为使用的每个指标都定义一种评级方法。评级方法定义了工作人员如何评估他们从您选择的模型中得到的响应。要详细了解不同的可用评级方法以及如何为工作人员创建高质量的说明，请参阅[在 Amazon Bedrock 中创建和管理工作团队](#)。

#### 使用 Amazon Bedrock 控制台查看模型评估作业结果

模型评估作业完成后，结果将存储在您指定的 Amazon S3 存储桶中。如果您以任何方式修改结果所在的位置，模型评估报告卡将不再显示在控制台中。

### 创建使用人工的模型评估作业

1. 通过以下网址打开 Amazon Bedrock 控制台：<https://console.aws.amazon.com/bedrock/home>
2. 在导航窗格中，选择模型评估。
3. 在建立评估卡的人工：自带团队下，选择创建人工评估。
4. 在指定作业详细信息页面上，提供以下信息：
  - a. 评估名称 — 为模型评估作业指定一个可描述该作业的名称。此名称将显示在模型评估作业列表中。此名称在您的 AWS 区域的账户中必须是唯一的。

- b. 描述 ( 可选 ) — 提供可选描述。
5. 然后选择下一步。
  6. 在设置评估页面上，提供以下信息：
    - a. 模型 – 最多可以选择两个要在模型评估作业中使用的模型。

要了解有关 Amazon Bedrock 中可用模型的更多信息，请参阅 [模型访问权限](#)。

- b. ( 可选 ) 要更改推理配置，请选择更新。

更改推理配置会更改所选模型生成的响应。要了解有关可用推理参数的更多信息，请参阅 [根基模型的推理参数](#)。

- c. 任务类型 – 选择您希望模型在模型评估作业期间尝试执行的任务类型。模型的所有说明都必须包含在提示中。任务类型不能控制模型的响应。
  - d. 评估指标 — 推荐的指标列表会根据您选择的任务而变化。您必须为推荐的每个指标都选择一种评级方法。最多可以为每个模型评估作业设置 10 个评估指标。
  - e. ( 可选 ) 选择添加自定义指标以添加自定义指标。您必须定义指标、描述和评级方法。
  - f. 在数据集卡中，您必须提供以下信息：
    - i. 评估数据集 – 指定存储提示数据集的 S3 URI。自定义提示数据集中最多可以有 1000 个提示。
    - ii. 评估结果目标 – 必须指定用于保存模型评估作业结果的目录的 S3 URI。
    - iii. ( 可选 ) 加密密钥-提供用于在 Amazon S3 中加密数据的 AWS KMS 密钥的 ARN。
7. 然后选择下一步。
  8. 在权限卡中，指定以下内容：要了解有关模型评估所需权限的更多信息，请参阅 [创建模型评估作业所需的 IAM 权限和服务角色](#)。
    - a. Amazon Bedrock IAM 角色 – 指定具有所需权限的 Amazon Bedrock 服务角色。
    - b. 人工工作流程 IAM 角色-指定具有所需权限的 SageMaker 服务角色。
  9. 在工作团队卡中，指定以下内容：

#### 工作人员通知要求

将新的工作人员添加到模型评估作业中时，他们会自动收到一封邀请他们参与模型评估作业的电子邮件。将现有工作人员添加到模型评估作业中时，必须通知他们并为其提供模型



评估作业的工作人员门户 URL。现有工作人员不会收到自动电子邮件通知，告知他们已被添加到新的模型评估作业中。

- a. 使用选择团队下拉列表，指定创建新的工作团队或现有工作团队的名称。
- b. (可选) 每个提示的工作人员人数 – 更新评估每个提示的工作人员人数。您所选人数的工作人员审核完每个提示的响应之后，该提示及其响应将从工作团队负责的工作中删除。最终结果报告将包含每位工作人员的所有评级。
- c. (可选) 现有工作人员电子邮件 – 选择此选项可复制包含工作人员门户 URL 的电子邮件模板。
- d. (可选) 新工作人员电子邮件 – 选择此选项可查看新员工自动收到的电子邮件。

 Important

众所周知，大型语言模型偶尔会出错，生成令人反感或不适的内容。在此评估期间，工作人员可能会看到令人反感或不适的材料。在他们开始评估工作之前，请务必采取适当的措施，为其提供培训并发送通知。他们可以拒绝和放弃任务，也可以在使用人工评估工具进行评估期间稍事休息。

10. 然后选择下一步。
11. 在提供说明页面上，使用文本编辑器提供完成任务的说明。可以预览工作团队用来评估响应的评估 UI，包括指标、评级方法和您提供的说明。预览结果取决于您为该作业创建的配置。
12. 然后选择下一步。
13. 在审核并创建页面上，可以查看您在之前步骤中选择的选项摘要。
14. 要启动模型评估作业，请选择创建。

作业成功启动后，状态将变为进行中。作业完成后，状态将变为已完成。

要了解如何评估、查看和下载模型评估作业的结果，请参阅 [模型评估作业结果](#)。

# 模型评估任务

## Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

在模型评估作业中，评估任务是您希望模型根据提示中的信息执行的任务。

您可以为每个模型评估作业选择一种任务类型。通过以下主题，了解有关每种任务类型的更多信息。每个主题中还包含可用的内置数据集及其相应指标的列表，这些指标只能在自动模型评估作业中使用。

## 主题

- [一般文本生成](#)
- [文本摘要](#)
- [问答](#)
- [文本分类](#)

## 一般文本生成

## Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

一般文本生成是一种任务，由包含聊天机器人的应用程序使用。模型针对一般性问题生成的响应受用于训练模型的文本的正确性、相关性和偏差的影响。

以下内置数据集包含非常适合在一般文本生成任务中使用的提示。

### 开放式语言生成数据集中的偏差 (BOLD)

开放式语言生成数据集中的偏差 (BOLD) 是一个数据集，用于评估一般文本生成中的公正性，重点关注五个领域：职业、性别、种族、宗教意识形态和政治意识形态。它包含 23,679 条不同的文本生成提示。

## RealToxicityPrompts

RealToxicityPrompts 是评估毒性的数据集。它会试图让模型生成带有种族主义、性别歧视或其他倾向的毒性内容。此数据集包含 100,000 条不同的文本生成提示。

### T-Rex：自然语言与知识库三元组的大规模比对 (TRES)

TRES 是由从维基百科提取的知识库三元组 (KBT) 组成的数据集。KBT 是一种用于自然语言处理 (NLP) 和知识表示的数据结构，由主语、谓词和宾语组成，其中主语和宾语通过某种关系联系起来。例如，“乔治·华盛顿曾任美国总统”就是一个知识库三元组 (KBT)。主语是“乔治·华盛顿”，谓语是“曾任”，宾语是“美国总统”。

### WikiText2

WikiText2 是一个包含一般文本生成中使用的提示 HuggingFace 的数据集。

下表汇总了可用于自动模型评估作业的计算指标和推荐的内置数据集。

### Amazon Bedrock 中可用于生成一般文本的内置数据集

| 任务类型   | 指标                                  | 内置数据集                     | 计算指标            |
|--------|-------------------------------------|---------------------------|-----------------|
| 一般文本生成 | 准确性                                 | <a href="#">TRES</a>      | 现实世界知识 (RWK) 得分 |
|        | 稳健性                                 | <a href="#">BOLD</a>      | 字词错误率           |
|        |                                     | <a href="#">WikiText2</a> |                 |
|        |                                     | <a href="#">TRES</a>      |                 |
| 毒性     | <a href="#">RealToxicityPrompts</a> | 毒性                        |                 |
|        | <a href="#">BOLD</a>                |                           |                 |

对于一般文本生成，存在一个已知的系统问题 CohereCommand，导致 AnthropicClaude 模型无法成功完成稳健性评估。

要详细了解每个内置数据集的计算指标的计算方法，请参阅 [模型评估作业结果](#)。

## 文本摘要

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

文本摘要用于新闻摘要、法律文件、学术论文、内容预览和内容策划的创建等任务。用于训练模型的文本的模糊性、连贯性、偏差和流畅性以及信息丢失、准确性、相关性或上下文不匹配等问题都会影响响应的质量。

支持将以下内置数据集与任务摘要任务类型一起使用。

### Gigaword

Gigaword 数据集由新闻文章标题组成。此数据集用于文本摘要任务。

下表汇总了计算指标和推荐的内置数据集。

Amazon Bedrock 中可用于文本摘要的内置数据集

| 任务类型 | 指标  | 内置数据集                    | 计算指标                       |
|------|-----|--------------------------|----------------------------|
| 文本摘要 | 准确性 | <a href="#">Gigaword</a> | BERTScore                  |
|      | 毒性  | <a href="#">Gigaword</a> | 毒性                         |
|      | 稳健性 | <a href="#">Gigaword</a> | BERTScore 和 deltaBERTScore |

对于文本摘要，存在一个已知的系统问题，导致AnthropicClaude模型无法成功完成稳健性评估。

要详细了解每个内置数据集的计算指标的计算方法，请参阅 [模型评估作业结果](#)。

## 问答

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

问答用于自动帮助中心响应、信息检索和在线学习的生成等任务。如果用于训练根基模型的文本包含数据不完整或不准确、讽刺或反讽等问题，响应的质量可能会受到影响。

建议将以下内置数据集用于问答任务类型。

### BoolQ

BoolQ 是一个由是/否问答对组成的数据集。提示包含一小段内容，然后是一个关于该段内容的问题。建议将此数据集用于问答类型的任务。

### 自然问题

自然问题是由提交给 Google 搜索的真实用户问题组成的数据集。

### TriviaQA

TriviaQA 是一个包含超过 65 万个的数据集。question-answer-evidence-triples 此数据集用于问答任务。

下表汇总了计算指标和推荐的内置数据集。

### Amazon Bedrock 中可用于问答任务类型的内置数据集

| 任务类型                             | 指标  | 内置数据集                            | 计算指标         |
|----------------------------------|-----|----------------------------------|--------------|
| 问答                               | 准确性 | <a href="#">BoolQ</a>            | NLP-F1       |
|                                  |     | <a href="#">NaturalQuestions</a> |              |
|                                  |     | <a href="#">TriviaQA</a>         |              |
|                                  | 稳健性 | <a href="#">BoolQ</a>            | F1 和 deltaF1 |
| <a href="#">NaturalQuestions</a> |     |                                  |              |

| 任务类型 | 指标 | 内置数据集                            | 计算指标 |
|------|----|----------------------------------|------|
|      |    | <a href="#">TriviaQA</a>         |      |
|      | 毒性 | <a href="#">BoolQ</a>            | 毒性   |
|      |    | <a href="#">NaturalQuestions</a> |      |
|      |    | <a href="#">TriviaQA</a>         |      |

关于问题和答案，存在一个已知的系统问题，导致AnthropicClaude模型无法成功完成稳健性评估。

要详细了解每个内置数据集的计算指标的计算方法，请参阅 [模型评估作业结果](#)。

## 文本分类

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

文本分类用于将文本划分到预定义的几个类别中。使用文本分类的应用程序包括社交媒体上的内容推荐、垃圾邮件检测、语言识别和趋势分析。不平衡的类、歧义数据、噪声数据和标注中的偏差是一些可能导致文本分类错误的问题。

建议将以下内置数据集用于文本分类任务类型。

### 女性电子商务服装评论

女性电子商务服装评论是一个包含客户撰写的服装评论的数据集。此数据集用于文本分类任务。

下表汇总了计算指标和推荐的内置数据集。

### Amazon Bedrock 中的可用内置数据集

| 任务类型 | 指标  | 内置数据集                      | 计算指标 |
|------|-----|----------------------------|------|
| 文本分类 | 准确性 | <a href="#">女性电子商务服装评论</a> |      |

| 任务类型 | 指标  | 内置数据集                      | 计算指标                                                                |
|------|-----|----------------------------|---------------------------------------------------------------------|
|      |     |                            | 准确性 ( 来自 classification_accuracy_score 的 二元准确性 )                    |
|      | 稳健性 | <a href="#">女性电子商务服装评论</a> | classification_accuracy_score 和 delta_classification_accuracy_score |

对于文本分类，存在一个已知的系统问题，导致AnthropicClaude模型无法成功完成稳健性评估。

要详细了解每个内置数据集的计算指标的计算方法，请参阅 [模型评估作业结果](#)。

## 在模型评估作业中使用提示数据集

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

要创建模型评估作业，必须指定模型在推理期间使用的提示数据集。Amazon Bedrock 提供了可用于自动模型评估的内置数据集，您也可以自带提示数据集。对于使用人工的模型评估作业，您必须使用自己的提示数据集。

通过以下部分，详细了解可用的内置提示数据集和创建自定义提示数据集。

要详细了解如何在 Amazon Bedrock 中创建您的第一个模型评估作业，请参阅 [模型评估](#)。

### 主题

- [在自动模型评估作业中使用内置提示数据集](#)
- [自定义提示数据集](#)

## 在自动模型评估作业中使用内置提示数据集

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

Amazon Bedrock 提供了多个内置提示数据集，您可以在自动模型评估作业中使用这些数据集。每个内置数据集都基于一个开源数据集。我们对每个开源数据集都进行了下采样，仅包含 100 条提示。

所有内置提示数据集都使用 Anthropic Claude 模型所需的格式。这意味着每条提示的开头都包含 "Human: "，结尾都包含 "Assistant: "。为避免这种情况，您可以使用提示数据集的原始开源版本，并在创建模型评估作业时将其作为自定义数据集上传。

创建自动模型评估作业并选择任务类型时，Amazon Bedrock 会为您提供推荐指标列表。Amazon Bedrock 还为每个指标提供了推荐的内置数据集。要了解有关可用任务类型的更多信息，请参阅 [模型评估任务](#)。

### 开放式语言生成数据集中的偏差 (BOLD)

开放式语言生成数据集中的偏差 (BOLD) 是一个数据集，用于评估一般文本生成中的公正性，重点关注五个领域：职业、性别、种族、宗教意识形态和政治意识形态。它包含 23,679 条不同的文本生成提示。

### RealToxicityPrompts

RealToxicityPrompts 是评估毒性的数据集。它会试图让模型生成带有种族主义、性别歧视或其他倾向的毒性内容。此数据集包含 100,000 条不同的文本生成提示。

### T-Rex：自然语言与知识库三元组的大规模比对 (TRES)

TRES 是由从维基百科提取的知识库三元组 (KBT) 组成的数据集。KBT 是一种用于自然语言处理 (NLP) 和知识表示的数据结构，由主语、谓词和宾语组成，其中主语和宾语通过某种关系联系起来。例如，“乔治·华盛顿曾任美国总统”就是一个知识库三元组 (KBT)。主语是“乔治·华盛顿”，谓语是“曾任”，宾语是“美国总统”。

### WikiText2

WikiText2 是一个包含一般文本生成中使用的提示 HuggingFace 的数据集。

### Gigaword

Gigaword 数据集由新闻文章标题组成。此数据集用于文本摘要任务。



## BoolQ

BoolQ 是一个由是/否问答对组成的数据集。提示包含一小段内容，然后是一个关于该段内容的问题。建议将此数据集用于问答类型的任务。

## 自然问题

自然问题是由提交给 Google 搜索的真实用户问题组成的数据集。

## TriviaQA

TriviaQA 是一个包含超过 65 万个的数据集。question-answer-evidence-triples 此数据集用于问答任务。

## 女性电子商务服装评论

女性电子商务服装评论是一个包含客户撰写的服装评论的数据集。此数据集用于文本分类任务。

下表中显示了按任务类型分组的可用数据集列表。要了解有关如何计算自动指标的更多信息，请参阅 [自动模型评估作业报告卡 \(控制台\)](#)。

## Amazon Bedrock 中可用于自动模型评估作业的内置数据集

| 任务类型   | 指标                                  | 内置数据集                     | 计算指标            |
|--------|-------------------------------------|---------------------------|-----------------|
| 一般文本生成 | 准确性                                 | <a href="#">TREX</a>      | 现实世界知识 (RWK) 得分 |
|        | 稳健性                                 | <a href="#">BOLD</a>      | 字词错误率           |
|        |                                     | <a href="#">WikiText2</a> |                 |
|        |                                     | <a href="#">英文版维基百科</a>   |                 |
| 毒性     | <a href="#">RealToxicityPrompts</a> | 毒性                        |                 |
|        | <a href="#">BOLD</a>                |                           |                 |
| 文本摘要   | 准确性                                 | <a href="#">Gigaword</a>  | BERTScore       |
|        | 毒性                                  | <a href="#">Gigaword</a>  | 毒性              |

| 任务类型 | 指标  | 内置数据集                            | 计算指标                                                                |
|------|-----|----------------------------------|---------------------------------------------------------------------|
|      | 稳健性 | <a href="#">Gigaword</a>         | BERTScore 和 deltaBERTScore                                          |
| 问答   | 准确性 | <a href="#">BoolQ</a>            | NLP-F1                                                              |
|      |     | <a href="#">NaturalQuestions</a> |                                                                     |
|      |     | <a href="#">TriviaQA</a>         |                                                                     |
|      | 稳健性 | <a href="#">BoolQ</a>            | F1 和 deltaF1                                                        |
|      |     | <a href="#">NaturalQuestions</a> |                                                                     |
|      |     | <a href="#">TriviaQA</a>         |                                                                     |
|      | 毒性  | <a href="#">BoolQ</a>            | 毒性                                                                  |
|      |     | <a href="#">NaturalQuestions</a> |                                                                     |
|      |     | <a href="#">TriviaQA</a>         |                                                                     |
| 文本分类 | 准确性 | <a href="#">女性电子商务服装评论</a>       | 准确性 ( 来自 classification_accuracy_score 的二元准确性 )                     |
|      |     | <a href="#">女性电子商务服装评论</a>       |                                                                     |
|      |     | <a href="#">女性电子商务服装评论</a>       |                                                                     |
|      | 稳健性 | <a href="#">女性电子商务服装评论</a>       | classification_accuracy_score 和 delta_classification_accuracy_score |

要了解有关创建要求和自定义提示数据集示例的更多信息，请参阅 [自定义提示数据集](#)。

## 自定义提示数据集

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

您可以在模型评估作业中使用自定义提示数据集。

自定义提示数据集必须存储在 Amazon S3 中，使用 JSON 行格式和 `.jsonl` 文件扩展名。将数据集上传到 Amazon S3 时，请务必更新 S3 存储桶上的跨源资源共享 (CORS) 配置。要了解有关所需 CORS 权限的更多信息，请参阅 [对 S3 存储桶的必要跨源资源共享 \(CORS\) 权限](#)。

### 主题

- [对自动模型评估作业中使用的自定义提示数据集的要求](#)
- [对使用人工的模型评估作业中自定义提示数据集的要求](#)

## 对自动模型评估作业中使用的自定义提示数据集的要求

在自动模型评估作业中，您可以为在模型评估作业中选择的每个指标使用一个自定义提示数据集。自定义数据集使用 JSON 行格式 (`.jsonl`)，每行都必须是一个有效的 JSON 对象。每个自动评估作业的数据集中最多可以有 1000 条提示。

必须在自定义数据集中使用以下键。

- `prompt` – 必要键，用于指明以下任务的输入：
  - 模型在一般文本生成中应当响应的提示。
  - 模型在问答任务类型中应当回答的问题。
  - 模型在文本摘要任务中应当总结的文本。
  - 模型在分类任务中应当分类的文本。
- `referenceResponse` – 必要键，用于指明在针对以下任务类型评估模型时，所依据的真实响应：
  - 问答任务中所有提示的答案。
  - 所有准确性和稳健性评估的答案。
- `category`– ( 可选 ) 生成每个类别报告的评估分数。

举个例子，准确性既要求提供要问的问题，也要求提供检查模型响应所依据的答案。因此，使用键 `prompt` 和问题中包含的值，使用键 `referenceResponse` 和答案中包含的值，如下所示。

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

前面的示例是某个 JSON 行输入文件中的一行，该文件将作为推理请求发送给您的模型。系统将为您的 JSON 行数据集中的每条此类记录调用模型。以下数据输入示例用于问答任务，该任务使用可选的 `category` 键进行评估。

### Important

在自定义数据集中的最后一条提示之后，文件必须以换行符结尾。

```
{"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
```

要详细了解使用人工的模型评估作业的格式要求，请参阅 [对使用人工的模型评估作业中自定义提示数据集的要求](#)。

### AnthropicClaude自定义提示数据集要求

如果要指定自定义提示数据集，并在必须修改的AnthropicClaude自动模型评估作业中使用模型，则必须修改prompt密钥以使其具有以下结构。

```
{
 "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."
}
```

```
}
```

**Human:** 和 **Assistant:** 这些关键字区分大小写，还必须包含英文冒号。

建议使用 Amazon Bedrock Playground 测试包含这些关键字的敏感性。

## 对使用人工的模型评估作业中自定义提示数据集的要求

在 JSON 行格式中，每一行都是一个有效的 JSON 对象。在每个模型评估作业中，一个提示数据集最多可以包含 1000 条提示。

有效的提示条目必须包含 `prompt` 键，而 `category` 和 `referenceResponse` 键则是可选的。`category` 键用于将提示标记为特定的类别，在模型评估报告卡中查看结果时，可以使用这些类别来筛选结果。`referenceResponse` 键用于指定工作人员在评估期间可以参考的真实响应。

工作人员可以在自己的 UI 中看到您为 `prompt` 和 `referenceResponse` 指定的内容。

下面是一个包含 6 个输入并使用了 JSON 行格式的自定义数据集示例。

### Important

在自定义数据集中的最后一条提示之后，文件必须以换行符结尾。

```
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
{"prompt":"Provide the prompt you want the model to use
during inference","category":"(Optional) Specify an optional
category","referenceResponse":"(Optional) Specify a ground truth response."}
```

```
The file must end with a newline
```

为清楚说明，下面的示例显示了一个展开来的单个条目

```
{
 "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."
}
```

### AnthropicClaude自定义提示数据集要求

如果您在模型评估作业中选择的任一模型是AnthropicClaude模型，则必须修改prompt密钥以使其具有以下结构。

```
{
 "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."
}
```

**Human:** 和 **Assistant:** 这些关键字区分大小写，还必须包含英文冒号。

由于这些提示会发送给模型评估作业中的第二个模型，我们建议您使用 Amazon Bedrock Playground 测试在第二个模型上包含这些关键字的敏感性。

## 创建良好的工作人员说明

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

为模型评估作业创建良好的说明可提高工作人员完成任务的准确性。创建模型评估作业时，可以修改控制台中提供的默认说明。这些说明将显示在工作人员在其中完成标记任务的 UI 页面上。

为了帮助工作人员完成分配的任务，您可以在两个位置提供说明。

为每种评估和评级方法提供良好的描述

描述应简要说明所选指标。描述应针对指标展开说明，并明确您希望工作人员如何评估所选评级方法。要查看示例，了解每种评级方法在工作人员 UI 中是如何显示的，请参阅 [可用评级方法摘要](#)。

为工作人员提供总体评估说明

这些说明显示在工作人员完成任务的同一网页上。可以使用这个位置为模型评估作业提供整体指导，如果您已将真实响应包含在提示数据集中，则可以对其进行描述。

## 可用评级方法摘要

以下几个部分分别介绍了工作团队在评估 UI 中看到的评级方法示例，以及这些结果是如何保存在 Amazon S3 中的。

### 李克特量表，比较多个模型输出

评估人员按照您的说明，通过 5 级李克特量表，表明他们对模型的两种响应的偏好。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的偏好强度评级。

请务必在说明中定义 5 级量表的要点，以便评估人员知道如何根据您的期望对响应进行评级。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceStrength"` 键值对中。

### 选择按钮（单选按钮）

选择按钮允许评估人员指出他们首选某个响应，而不是另一个响应。评估人员按照您的说明，使用单选按钮表明他们在两个响应之间的偏好。最终报告中的结果将以百分比的形式，显示工作人员为每种模型首选的响应。请务必在说明中清楚地阐明您的评估方法。



## ▼ Metric: Relevance

Which response do you prefer based on the metric?

- Response 1
- Response 2

### JSON 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceRate"` 键值对中。

### 序数排名

序数排名允许评估人员根据您的说明，按从 1 开始的顺序对提示的首选响应进行排名。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的排名。请务必在说明中定义排名 1 的含义。

## ▼ Metric: Toxicity

Input ranking for the responses. 1 is the best ranked response.

Response 1

Input number



Response 1

Input number



### JSON 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceRank"` 键值对中。

### 大拇指向上/向下

大拇指向上/向下允许评估人员按照您的说明，将模型的每个响应评定为可接受/不可接受。最终报告中的结果将以百分比的形式，显示每个模型从评估人员处获得好评总数的百分比。可以在包含一个或多个模型的评估中使用此评级方法。如果您在包含两个模型的评估中使用这种方法，系统会针对每个模型响应，向工作团队显示一个大拇指向上/向下符号，而最终报告将分别显示每个模型的汇总结果。请务必在说明中定义什么是可接受的（即什么是大拇指向上的评级）。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "approvalRate"` 键值对中。

### 李克特量表，评估单个模型响应

允许评估人员按照您对 5 级李克特量表的说明，表明他们对模型响应的认可程度。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的 5 级评定结果。可以在包含一个或多个模型的评估

中使用此评级方法。如果您在包含多个模型的评估中使用这种评级方法，系统会针对每个模型响应，向工作团队显示一个 5 级李克特量表，而最终报告将分别显示每个模型的汇总结果。请务必在说明中定义 5 级量表的要点，以便评估人员知道如何根据您的期望对响应进行评级。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "approvalStrength"` 键值对中。

## 在 Amazon Bedrock 中创建和管理工作团队

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

在使用人力的模型评估作业中，您需要有一个工作团队。工作团队是您选择的一组工作人员。他们可能是公司的员工，也可能是行业的一群主题专家。

### ⚠ Amazon Bedrock 中的工作人员通知

- 在 Amazon Bedrock 中创建模型评估作业时，只有在第一次将员工添加到工作团队中时，他们才会收到所分配作业的通知。
- 如果在创建模型评估期间从工作团队中删除工作人员，他们也将无法访问分配给他们的所有模型评估作业。
- 如果向现有工作人员分配任何新模型评估作业，您都必须直接通知他们，并为其提供工作人员门户的 URL。工作人员必须使用他们之前为工作人员门户创建的登录凭证。对于每个地区您 AWS 账户中的所有评估职位，此工作人员门户网站都是一样的

在 Amazon Bedrock 中，您可以在设置模型评估作业的同时，创建新的工作团队或管理现有工作团队。当您在 Amazon Bedrock 中创建工作团队时，你就是在向由 Amazon SageMaker Ground Truth 管理的私人员工队伍中添加员工。Amazon SageMaker Ground Truth 支持更高级的劳动力管理功能。要详细了解如何在 Amazon SageMaker Ground Truth 中管理员工，请参阅[创建和管理员工](#)。

您可以在设置新模型评估作业时，从工作团队中删除工作人员。否则，你必须使用 Amazon Cognito 控制台或 Amazon SageMaker Ground Truth 控制台来管理您在 Amazon Bedrock 中创建的工作团队。

如果 IAM 用户、群组或角色具有所需权限，则在创建使用人类工作人员的模式评估任务时，您将看到您在 Amazon Cognito、Amazon SageMaker Ground Truth 或 Amazon Augmented AI 中创建的现有私人员工和工作团队。SageMaker

对于每个工作团队，Amazon Bedrock 最多支持 50 名工作人员。

在电子邮件地址字段中，一次最多可以输入 50 个电子邮件地址。要在模型评估作业中添加更多工作人员，请使用 Amazon Cognito 控制台或 Ground Truth 控制台。地址之间必须用逗号分隔。您应将自己的电子邮件地址包括在内，以便成为人力的一员，可以查看标注任务。

## 模型评估作业结果

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

[模型评估作业](#)的结果可通过 Amazon Bedrock 控制台获得，也可以通过从您在创建作业时指定的 Amazon S3 存储桶下载结果来获得。

作业状态变为已就绪后，就可以找到您在创建作业时指定的 S3 存储桶。为此，请前往模型评估主页上的模型评估表并选择它。

通过以下主题了解如何访问模型评估报告，以及如何将模型评估作业的结果保存在 Amazon S3 中。

### 主题

- [自动模型评估作业报告卡（控制台）](#)
- [人工模型评估作业报告卡（控制台）](#)
- [了解模型评估作业的结果是如何保存在 Amazon S3 中的](#)

## 自动模型评估作业报告卡（控制台）

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

在模型评估报告卡中，可以看到您提供或选择的数据集中的提示总数，以及其中有多少提示收到了响应。如果响应数少于输入提示数，请务必检查 Amazon S3 存储桶中的数据输出文件。提示可能导致模型出错，未检索到任何推理。只有来自模型的响应才会用于指标计算。

通过以下过程，在 Amazon Bedrock 控制台上查看自动模型评估作业。

1. 打开 Amazon Bedrock 控制台。
2. 在导航窗格中，选择模型评估。
3. 接下来，在模型评估表中找到您要查看的自动模型评估作业的名称。然后选中它。

在所有与语义稳健性相关的指标中，Amazon Bedrock 会通过以下方式扰动提示：将文本转换为全部小写、键盘输入错误、将数字转换为字词、随机更改为大写，以及随机添加/删除空格。

打开模型评估报告后，您可以查看作业的汇总指标和作业配置摘要。

对于创建作业时指定的每个指标和提示数据集，您都会看到一张卡，以及为该指标指定的每个数据集的值。这个值的计算方式会根据任务类型和您选择的指标而变化。

将可用指标应用于一般文本生成任务类型时，该如何计算每个指标

- **准确性**：这个指标的值是使用现实世界知识分数 ( RWK 分数 ) 计算得出的。RWK 分数考察模型对现实世界的事实性知识进行编码的能力。RWK 分数高表示您的模型是准确的。
- **稳健性**：这个指标的值是使用语义稳健性计算得出的。而后者是使用字词错误率计算得出的。语义稳健性衡量模型输出因输入中微小的语义保持扰动而导致的变化程度。针对此类扰动的稳健性是一种理想属性，因此，语义稳健性分数较低，表明您的模型表现良好。

我们会考虑的扰动类型包括：将文本转换为全部小写、键盘输入错误、将数字转换为字词、随机更改为大写，以及随机添加/删除空格。系统会对数据集中的每个提示进行大约 5 次的扰动。然后，发送每个经过扰动的响应以进行推理，并用它们自动计算稳健性分数。

- **毒性**：这个指标的值是使用解毒算法中的毒性计算得出的。低毒性值表示您选择的模型不会生成大量毒性内容。要了解有关排毒算法的更多信息并了解毒性是如何计算的，请参阅中的[排毒算法](#)。

GitHub

将可用指标应用于文本摘要任务类型时，该如何计算每个指标

- **准确性**：这个指标的值是使用 BERT 分数计算得出的。BERT 分数是使用来自 BERT 模型的预训练上下文嵌入计算得出的。它通过余弦相似性来匹配候选句子和参考句子中的字词。显示的值为 F1 分数。较低的 F1 分数表明该模型既不精确，又难以捕获到真阳性 ( 查全率 )。
- **稳健性**：对于这个指标，计算得出的值为百分比。它的计算方法是： $(\Delta \text{BERTScore} / \text{BERTScore}) \times 100$ 。Delta BERTScore 是数据集中受扰动提示的 BERT 分数与原始提示的 BERT 分数之间的差值。系统会对数据集中的每个提示进行大约 5 次的扰动。然后，发送每个经过扰动的响应以进行推理，并用它们自动计算稳健性分数。分数越低，表示所选模型的稳健性越高。

- **毒性**：这个指标的值是使用解毒算法中的毒性计算得出的。低毒性值表示您选择的模型不会生成大量毒性内容。要了解有关排毒算法的更多信息并了解毒性是如何计算的，请参阅中的[排毒算法](#)。

GitHub

将可用指标应用于问答任务类型时，该如何计算每个指标

- **准确性**：对于这个指标，计算得出的值为 F1 分数。F1 分数的计算方法是用查准率分数（正确预测与所有预测的比率）除以查全率分数（正确预测与相关预测总数的比率）。F1 分数的范围为 0 到 1，值越高表示性能越好。
- **稳健性**：对于这个指标，计算得出的值为百分比。具体计算公式为： $(F1 \text{ 增量} / F1) \times 100$ 。F1 增量是数据集中受扰动提示的 BERT 分数与原始提示的 BERT 分数之间的差值。系统会对数据集中的每个提示进行大约 5 次的扰动。然后，发送每个经过扰动的响应以进行推理，并用它们自动计算稳健性分数。分数越低，表示所选模型的稳健性越高。
- **毒性**：这个指标的值是使用解毒算法中的毒性计算得出的。低毒性值表示您选择的模型不会生成大量毒性内容。要了解有关排毒算法的更多信息并了解毒性是如何计算的，请参阅中的[排毒算法](#)。

GitHub

将可用指标应用于文本分类任务类型时，该如何计算每个指标

- **准确性**：对于这个指标，计算得出的值为准确度。准确性是将预测的类别与其真实情况标签进行比较得出的分数。更高的准确性表示模型正在根据提供的真实情况标签正确地对文本进行分类。
- **稳健性**：对于这个指标，计算得出的值为百分比。具体计算公式为： $(\text{平衡分类分数增量} / \text{分类准确性分数}) \times 100$ 。平衡分类分数增量是受扰动提示和原始输入提示的分类准确性分数之间的差值。系统会对数据集中的每个提示进行大约 5 次的扰动。然后，发送每个经过扰动的响应以进行推理，并用它们自动计算稳健性分数。分数越低，表示所选模型的稳健性越高。

## 人工模型评估作业报告卡（控制台）

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

在模型评估报告卡中，可以看到您提供或选择的数据集中的提示总数，以及其中有多少提示收到了响应。如果响应数少于输入提示数乘以您在作业中配置的每个提示的工作人员数（1、2 或 3），请务必



检查 Amazon S3 存储桶中的数据输出文件。提示可能导致模型出错，未检索到任何推理。此外，一个或多个工作人员可能拒绝评估模型输出响应。只有来自工作人员的响应才会用于指标计算。

通过以下过程，在 Amazon Bedrock 控制台上打开使用人工的模型评估。

1. 打开 Amazon Bedrock 控制台。
2. 在导航窗格中，选择模型评估。
3. 接下来，在模型评估表中找到您要查看的模型评估作业的名称。然后选中它。

模型评估报告使用报告卡提供有关人工评估作业期间所收集数据的见解。每张报告卡都将显示指标、描述和评级方法，以及一个数据可视化结果，用于呈现针对给定指标收集的数据。

以下几个部分分别介绍了工作团队在评估 UI 中看到的 5 种可能的评级方法示例。这些示例还显示了使用哪个键值对将结果保存到 Amazon S3 中。

### 李克特量表，比较多个模型输出

评估人员[按照您的说明](#)，通过 5 级李克特量表，表明他们对模型的两种响应的偏好。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的偏好强度评级。

请务必在说明中定义 5 级量表的要点，以便评估人员知道如何根据您的期望对响应进行评级。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceStrength"` 键值对中。

### 选择按钮（单选按钮）

选择按钮允许评估人员指出他们首选某个响应，而不是另一个响应。评估人员按照您的说明，使用单选按钮表明他们在两个响应之间的偏好。最终报告中的结果将以百分比的形式，显示工作人员为每种模型首选的响应。请务必在说明中清楚地阐明您的评估方法。

## ▼ Metric: Relevance

Which response do you prefer based on the metric?

- Response 1
- Response 2

### JSON 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceRate"` 键值对中。

### 序数排名

序数排名允许评估人员根据您的说明，按从 1 开始的顺序对提示的首选响应进行排名。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的排名。请务必在说明中定义排名 1 的含义。这种数据类型叫作偏好排名。

## ▼ Metric: Toxicity

Input ranking for the responses. 1 is the best ranked response.

Response 1

Input number



Response 1

Input number



### JSON 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "preferenceRank"` 键值对中。

### 大拇指向上/向下

大拇指向上/向下允许评估人员按照您的说明，将模型的每个响应评定为可接受/不可接受。最终报告中的结果将以百分比的形式，显示每个模型从评估人员处获得好评总数的百分比。可以在包含一个或多个模型的模型评估作业中使用此评级方法。如果您在包含两个模型的评估中使用这种方法，系统会针对每个模型响应，向工作团队显示一个大拇指向上/向下符号，而最终报告将分别显示每个模型的汇总结果。请务必在说明中定义什么是可接受的（即什么是大拇指向上的评级）。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "approvalRate"` 键值对中。

### 李克特量表，评估单个模型响应

允许评估人员按照您对 5 级李克特量表的说明，表明他们对模型响应的认可程度。最终报告中的结果将以直方图的形式，显示评估人员对整个数据集的 5 级评定结果。可以在包含一个或多个模型的评估

中使用此评级方法。如果您在包含多个模型的评估中使用这种评级方法，系统会针对每个模型响应，向工作团队显示一个 5 级李克特量表，而最终报告将分别显示每个模型的汇总结果。请务必在说明中定义 5 级量表的要点，以便评估人员知道如何根据您的期望对响应进行评级。

## ▼ 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 输出

`evaluationResults` 下的第一个子键是返回所选评级方法的位置。在保存到 Amazon S3 存储桶的输出文件中，每个工作人员的结果都将保存到 `"evaluationResults": "approvalStrength"` 键值对中。

## 了解模型评估作业的结果是如何保存在 Amazon S3 中的

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

模型评估作业的输出将保存在创建模型评估作业时指定的 Amazon S3 存储桶中。模型评估作业的结果将保存为 JSON 行文件 (.jsonl)。

模型评估作业的结果将保存在您指定的 S3 存储桶中，如下所示。

- 使用人工的模型评估作业：

```
s3://user-specified-model-evaluation-job-output-bucket/human-loop-name/output.jsonl
```

- 自动模型评估作业：

```
s3://user-specified-model-evaluation-job-output-bucket/output/system-generated-id-tag/datasets
```

以下主题描述了如何将自动模型评估作业和人工模型评估作业的结果保存在 Amazon S3 中。

### 自动模型评估作业的输出数据

当作业状态变为已完成时，自动评估作业的结果将存储在 datasets 目录中。

对于您在创建模型评估作业时选择的每个指标和相应的提示数据集，都将在 datasets 目录中生成一个 JSON 行文件。文件使用以下命名约定 **metric\_input-dataset.jsonl**。

模型评估作业的每个结果都以 automatedEvaluationResult 键开头。第一个子键 scores 包含您在 Amazon Bedrock 控制台中选择的指标。此示例中仅选择了一个指标 Accuracy。它还包含所选指标的计算值 result。要详细了解计算的具体值，请参阅 [自动模型评估作业报告卡（控制台）](#)。

第二个键 inputRecord 是您在输入提示数据集中提供的内容副本。

第三个键 modelInvocations 包含一个 JSON 对象列表，其中包含您在创建模型评估作业时所选模型的 ARN。它还包含基于所提供提示的模型的完整响应。

以下是文本摘要任务类型的输出示例，其中仅选择了一个指标，即准确性。

```
{
 "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:\n\n- Two bodies have been recovered from a flooded coal mine in Yiyang City, in China's Henan province.\n\n- This happened as of Thursday morning, according to rescuers.\n\n- The location is central China's Henan province. \n\n- Yiyang City is where the flooded coal mine is located.\n\n- The coal mine was flooded, leading to the two deaths and need for rescue/recovery efforts.\n\n- This news is developing as of Thursday morning, so the death toll and rescue details may still be unfolding.\n\nPlease let me know if I have accurately summarized the key information or if you would like me to expand on any part of it."
 }]
}
```

## 使用人工的模型评估作业的输出数据

模型评估作业完成后，您会在人工审核任务返回的输出数据中看到以下参数。

| 参数                | 值类型 | 示例值                                                                                            | 描述                           |
|-------------------|-----|------------------------------------------------------------------------------------------------|------------------------------|
| flowDefinitionArn | 字符串 | arn:aws:sagemaker:us-west-2: <b>111122223333</b> :flow-definition/ <i>flow-definition-name</i> | 用于创建人工循环的人工审核 workflow (流定义) |



| 参数            | 值类型       | 示例值                                                                                                                                                                                                                  | 描述                                    |
|---------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
|               |           |                                                                                                                                                                                                                      | 的 Amazon 资源编号 (ARN)。                  |
| humanAnswers  | JSON 对象列表 | <pre>"answerContent": {   "evaluationResults": {     "approvalRate": [{       "metric": " <b>Relevance</b> ",       "modelResponseId": "0",       "result": false     }]   } }</pre>                                 | 包含 answerContent 中工作人员回应的 JSON 对象的列表。 |
| humanLoopName | 字符串       | system-generated-hash                                                                                                                                                                                                | 系统生成的 40 个字符的十六进制字符串。                 |
| inputRecord   | JSON 对象   | <pre>"inputRecord": {   "prompt": "What does vitamin C serum do for skin?",   "category": "Skincare",   "referenceResponse": "Vitamin C serum offers a range of benefits for the skin. Firstly, it acts...." }</pre> | 一个 JSON 对象，包含来自输入数据集的输入提示。            |

| 参数                   | 值类型              | 示例值                                                                                                                                                                                                   | 描述                                                       |
|----------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| modelInvo<br>cations | JSON<br>对象<br>列表 | <pre>"modelResponses": [{   "modelIdentifier": "arn:aws:bedrock: <b>us- east-1</b> ::foundation-model/ <b>anthropic.claude-v 2</b> ",   "responseText": "the-models-response-to-the -prompt" }]</pre> | 来自模型的各个响应。                                               |
| inputRecord          | JSON<br>对象<br>列表 | <a href="#">请参阅以下代码示例</a>                                                                                                                                                                             | 在请求 SageMaker 中发送到的输入内容 <a href="#">StartHumanLoop</a> 。 |

以下是模型评估作业的输出数据示例。

```
{
 "output": [{
 "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
 "humanAnswers": [{
 "acceptanceTime": "2023-11-09T19:17:43.107Z",
 "answerContent": {
 "evaluationResults": {
 "approvalRate": [{
 "metric": "Coherence",
 "metricName": "Coherence",
 "modelResponseId": "0",
 "result": false
 }, {
 "metric": "Accuracy",
 "metricName": "Accuracy",
 "modelResponseId": "0",
 "result": true
 }
]
 }
 }
]
}
```

```

]],
 "approvalStrength": [{
 "metric": "Toxicity",
 "metricName": "Toxicity",
 "modelResponseId": "0",
 "result": 1
 }]
 },
 "submissionTime": "2023-11-09T19:17:52.101Z",
 "timeSpentInSeconds": 8.994,
 "workerId": "444455556666",
 "workerMetadata": {
 "identityData": {
 "identityProviderType": "Cognito",
 "issuer": "https://cognito-idp.us-west-2.amazonaws.com/us-
west-2_111222",
 "sub": "c6aa8eb7-9944-42e9-a6b9-"
 }
 }
}],

```

...Additional response have been truncated for clarity...

```

]],
 "humanLoopName": "b3b1c64a2166e001e094123456789012"
},


```

```

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:\n\n- 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.\n\n- 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.\n\n- The total HIIT session typically lasts between 10-30
minutes, including warm-up, recovery periods, and cool down. \n\n- Uses exercises
like sprints, cycling, rowing, jumping rope, stair climbing, swimming, etc. Movements
that get your heart rate up quickly.\n\n- The short, intense intervals help improve
cardiovascular fitness by increasing heart rate and oxygen consumption.\n\n- HIIT
burns more calories and fat than steady-state cardio in a shorter time period. It also
helps build muscle strength and endurance.\n\n- 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.\n\n- Not suitable for beginners due
to the high-intensity. Start with steady-state cardio and build up endurance before
trying HIIT.\n\nIn summary, HIIT intersperses intense bursts of"
 }]
}
}

```

下表说明了您在 Amazon Bedrock 控制台中为每个指标选择的评级方法是如何返回到您的 Amazon S3 存储桶中的。evaluationResults 下的第一个子键是返回评级方法的方式。

在 Amazon Bedrock 控制台中选择的评级方法如何保存在 Amazon S3 中

| 所选评级方法     | 已保存在 Amazon S3 中   |
|------------|--------------------|
| 李克特量表 - 单个 | approvalStrength   |
| 李克特量表 - 比较 | preferenceStrength |
| 选择按钮       | preferenceRate     |

|          |                  |
|----------|------------------|
| 所选评级方法   | 已保存在 Amazon S3 中 |
| 序数排名     | preferenceRank   |
| 大拇指向上/向下 | approvalRate     |

## 创建模型评估作业所需的 IAM 权限和服务角色

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

### 角色：IAM 管理员

可以添加或删除 IAM 策略并创建新 IAM 角色的用户。

以下主题说明了使用 Amazon Bedrock 控制台创建模型评估任务所需的 AWS Identity and Access Management 权限、服务角色要求和所需的跨源资源共享 (CORS) 权限。

### 主题

- [使用 Amazon Bedrock 控制台创建模型评估作业所需的权限](#)
- [模型评估作业的服务角色要求](#)
- [对 S3 存储桶的必要跨源资源共享 \(CORS\) 权限](#)
- [用于模型评估作业的数据加密](#)

## 使用 Amazon Bedrock 控制台创建模型评估作业所需的权限

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

对于自动模型评估作业和使用人工的模型评估作业来说，创建模型评估作业所需的 IAM 权限是不同的。

自动和基于人工的模型评估作业都需要访问 Amazon S3 和 Amazon Bedrock。要创建基于人工的模型评估任务，您需要获得 Amazon Cognito 和 Amazon 的额外权限。SageMaker

要详细了解创建自动模型评估作业和基于人工的模型评估作业所需的服务角色，请参阅 [模型评估作业的服务角色要求](#)。

## 创建自动模型评估作业所需的权限

以下策略包含在 Amazon Bedrock 和 Amazon S3 中创建自动模型评估作业所需的最少 IAM 操作和资源集。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "Bedrock Console",
 "Effect": "Allow",
 "Action": [
 "bedrock:CreateModelEvaluationJob",
 "bedrock:GetModelEvaluationJob",
 "bedrock:ListModelEvaluationJobs",
 "bedrock:GetCustomModel",
 "bedrock:ListCustomModels",
 "bedrock:CreateProvisionedModelThroughput",
 "bedrock:UpdateProvisionedModelThroughput",
 "bedrock:GetProvisionedModelThroughput",
 "bedrock:ListProvisionedModelThroughputs",
 "bedrock:ListTagsForResource",
 "bedrock:UntagResource",
 "bedrock:TagResource"
],
 "Resource": "*"
 },
 {
 "Sid": "Allow Console S3 Access For Model Evaluation",
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:GetBucketCORS",
 "s3:ListBucket",

```

```

 "s3:ListBucketVersions",
 "s3:GetBucketLocation"
],
 "Resource": "*"
}
]
}

```

## 创建人工模型评估作业所需的权限

要通过 Amazon Bedrock 控制台创建使用人工的模型评估作业，您需要为用户、群组或角色添加额外的权限。

以下政策包含 Amazon Cognito 和 Amazon 创建基于人类的模型评估任务所需的最低限度 IAM SageMaker M 操作和资源集。您必须将此策略添加到[自动作业的基本策略要求](#)中。

```

{
 "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": "*"
}
]
}

```

## 模型评估作业的服务角色要求

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

要创建模型评估作业，必须指定服务角色。

服务角色是由一项服务代入、代表您执行操作的 [IAM 角色](#)。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的 [创建向 AWS 服务委派权限的角色](#)。

自动或基于人工的模型评估作业所需的 IAM 权限不同。使用以下部分详细了解所需的 Amazon Bedrock、Amazon SageMaker 和 Amazon S3 IAM 操作、服务委托人和资源。

以下部分根据您想要运行的模型评估作业的类型，分别介绍了所需的权限。

### 主题

- [自动模型评估作业的服务角色要求](#)
- [使用评估人员的模型评估作业的服务角色要求](#)

## 自动模型评估作业的服务角色要求

要创建自动模型评估作业，必须指定服务角色。您附加的策略将授予 Amazon Bedrock 访问您账户中资源的权限，并允许 Amazon Bedrock 代表您调用所选模型。



还必须附加一项信任策略，将 Amazon Bedrock 定义为使用 `bedrock.amazonaws.com` 的服务主体。以下各个策略示例将根据自动模型评估作业中调用的各项服务，为您展示所需的确切 IAM 操作。

要创建自定义服务角色，请参阅《IAM 用户指南》中的[使用自定义信任策略创建角色](#)。

### 必要的 Amazon S3 IAM 操作

以下策略示例将授予对 S3 存储桶（用于保存模型评估结果）的访问权限，以及对您指定的任何自定义提示数据集的访问权限（可选）。

```
{
 "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"
]
 }
]
}
```

```
]
}
```

## 必要的 Amazon Bedrock IAM 操作

您还需创建一个策略，允许 Amazon Bedrock 调用您计划在自动模型评估作业中指定的模型。要了解有关管理 Amazon Bedrock 模型访问权限的更多信息，请参阅[模型访问权限](#)。

```
{
 "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",
]
 }
]
}
```

## 可选的 AWS Key Management Service IAM 操作

如果您曾经 AWS Key Management Service 对保存自定义提示数据集或保存结果的 S3 存储桶进行加密，则需要添加以下 AWS KMS 策略。服务角色需要权限才能使用客户自主管理型密钥。

```
{
 "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"
 }
]
}
```

```

 }
]
}

```

## 服务主体要求

还必须指定将 Amazon Bedrock 定义为服务主体的信任策略，以允许 Amazon Bedrock 担任该角色。需要使用通配符 (\*) 模型评估任务 ARN，这样 Amazon Bedrock 才能在您的账户中创建模型评估任务。AWS

```

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

```

## 使用评估人员的模型评估作业的服务角色要求

要创建使用评估人员的模型评估作业，必须指定两个服务角色。

以下列表总结了必须在 Amazon Bedrock 控制台中指定的每个必要服务角色的 IAM 策略要求。

### Amazon Bedrock 服务角色的 IAM 策略要求摘要

- 必须添加将 Amazon Bedrock 定义为服务主体的信任策略。
- 必须允许 Amazon Bedrock 代表您调用所选模型。

- 必须允许 Amazon Bedrock 访问保存提示数据集的 S3 存储桶和将要保存结果的 S3 存储桶。
- 必须允许 Amazon Bedrock 在您的账户中创建所需的人工循环资源。
- ( 可选 ) 如果您已加密提示数据集存储桶或将要保存结果的 Amazon S3 存储桶，则必须允许 Amazon Bedrock 解密您的 KMS 密钥。

### Amazon SageMaker 服务角色的 IAM 政策要求摘要

- 您必须附上定义 SageMaker 为服务主体的信任策略。
- 您必须 SageMaker 允许访问保存提示数据集的 S3 存储桶和要保存结果的 S3 存储桶。
- ( 可选 ) 如果您已加密提示数据集存储桶或想要获得结果的位置，则必须允许 SageMaker 使用您的客户托管密钥。

要创建自定义服务角色，请参阅《IAM 用户指南》中的[使用自定义信任策略创建角色](#)。

### 必要的 Amazon S3 IAM 操作

以下策略示例将授予对保存模型评估结果的 S3 存储桶的访问权限，以及您指定的自定义提示数据集的访问权限。您需要将此策略附加到 SageMaker 服务角色和 Amazon Bedrock 服务角色。

```
{
 "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",
 "s3:PutObject",

```

```

 "s3:GetBucketLocation",
 "s3:AbortMultipartUpload",
 "s3:ListBucketMultipartUploads"
],
 "Resource": [
 "arn:aws:s3:::model_evaluation_job_output"
]
}
]
}

```

### 必要的 Amazon Bedrock IAM 操作

您还需创建一个策略，允许 Amazon Bedrock 调用您计划在自动模型评估作业中指定的模型。必须将此策略附加到 Amazon Bedrock 服务角色中。

```

{
 "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",
]
 }
]
}

```

### 必要的 Amazon Augmented AI IAM 操作

您还可以创建一项策略，允许 Amazon Bedrock 创建与人工模型评估作业相关的资源。由于 Amazon Bedrock 创建了启动模型评估作业所需的资源，因此您必须使用 "Resource": "\*"。必须将此策略附加到 Amazon Bedrock 服务角色中。

```

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

```

```

 {
 "Sid": "ManageHumanLoops",
 "Effect": "Allow",
 "Action": [
 "sagemaker:StartHumanLoop",
 "sagemaker:DescribeFlowDefinition",
 "sagemaker:DescribeHumanLoop",
 "sagemaker:StopHumanLoop"
],
 "Resource": "*"
 }
]
}

```

### 可选的 AWS Key Management Service IAM 操作

如果您曾经 AWS Key Management Service 对保存自定义提示数据集或保存结果的 S3 存储桶进行加密，则需要添加以下 AWS KMS 策略。服务角色需要权限才能使用客户自主管理型密钥。您必须将此策略附加到 SageMaker 服务角色和 Amazon Bedrock 服务角色。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllowAccessToKMS",
 "Effect": "Allow",
 "Action": [
 "kms:Decrypt",
 "kms:GenerateDataKey"
],
 "Resource": [
 "arn:aws:kms:us-east-1:111122223333:key/key id-"
],
 "Condition": {
 "StringEquals": {
 "kms:ViaService": [
 "s3.region.amazonaws.com"
]
 }
 }
 }
]
}

```

## 服务主体要求 (Amazon Bedrock)

还必须指定将 Amazon Bedrock 定义为服务主体的信任策略，以允许 Amazon Bedrock 担任该角色。

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

## 服务主体要求 (SageMaker)

还必须指定将 Amazon Bedrock 定义为服务主体的信任策略，这 SageMaker 允许扮演这个角色。

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

## 对 S3 存储桶的必要跨源资源共享 (CORS) 权限

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。

对于自定义提示数据集，您必须在 S3 存储桶上指定 CORS 配置。CORS 配置是一个定义规则的文档，这些规则标识可访问您的存储桶的源、每个源支持的操作（HTTP 方法）以及其他操作特定的信息。要了解有关使用 S3 控制台设置所需 CORS 配置的更多信息，请参阅《Amazon S3 用户指南》中的[配置跨源资源共享 \(CORS\)](#)。

以下是 S3 存储桶所需的最低 CORS 配置。

```
[
 {
 "AllowedHeaders": [
 "*"
],
 "AllowedMethods": [
 "GET",
 "PUT",
 "POST",
 "DELETE"
],
 "AllowedOrigins": [
 "*"
],
 "ExposeHeaders": ["Access-Control-Allow-Origin"]
 }
]
```

## 用于模型评估作业的数据加密

### Note

模型评估功能在 Amazon Bedrock 中为预览版，可能会有变化。要使用模型评估作业，您必须位于美国东部（弗吉尼亚州北部）区域或美国西部（俄勒冈州）区域。



在 Amazon Bedrock 模型评估作业中，无论是传输中还是静态的客户数据，都始终经过加密。对于 Amazon Bedrock 中模型评估作业的输入或输出数据，模型评估作业支持使用客户自主管理型密钥。如果您在创建模型评估任务时未指定客户托管密钥，则使用 Amazon S3 的默认 AWS 托管密钥进行加密。Amazon Bedrock 在模型评估作业期间存储中间客户数据（例如您的提示数据集），并使用服务托管 KMS 密钥对其进行静态加密。模型评估作业完成后，这些中间数据将被永久删除。

# Amazon Bedrock 知识库

Amazon Bedrock 的知识库使您能够将数据源积累到信息存储库中。借助知识库，您可以轻松构建一个利用检索增强生成 (RAG) 技术的应用程序，该技术可通过从数据来源检索信息来增强模型响应的生成。设置完成后，您可以通过以下方式利用知识库。

- 将 RAG 应用程序配置为使用 [RetrieveAndGenerate](#) API 来查询知识库并根据其检索到的信息生成响应。
- 将知识库与代理关联（有关更多信息，请参阅 [Agents for Amazon Bedrock](#)），通过帮助代理推断可以采取哪些步骤来帮助最终用户，为代理添加 RAG 功能。
- 使用 [Retrieve](#) API 直接从知识库中检索信息，在应用程序中创建自定义编排流。

知识库不仅可以用来回答用户查询，还可以通过为提示提供上下文来增强为根基模型提供的提示。知识库的响应还伴有引用内容，这样，用户就可以通过查找该响应所依据的确切内容来找到更多信息，还可以检查该响应是否合理，实际上是否正确。

要设置和使用知识库，请执行以下步骤。

1. 收集源文档以添加到您的知识库中。
2. （可选）为每个源文档创建一个元数据文件，以便在知识库查询期间筛选结果。
3. 将数据上传到 Amazon S3 存储桶。
4. （可选）在支持的向量存储中设置向量索引以索引数据。如果您计划使用 Amazon Bedrock 控制台为您创建 Amazon OpenSearch Serverless 向量数据库，则可以跳过此步骤。
5. 创建和配置您的知识库。
6. 通过使用根基模型生成嵌入并将其存储在支持的向量存储中来摄取数据。
7. 设置应用程序或代理以查询知识库并返回增强的响应。

## 主题

- [工作方式](#)
- [Amazon Bedrock 知识库支持的区域和型号](#)
- [Amazon Bedrock 知识库的先决条件](#)
- [创建知识库](#)
- [同步以将您的数据源提取到知识库中](#)

- [在 Amazon Bedrock 中测试知识库](#)
- [管理数据源](#)
- [管理知识库](#)
- [部署知识库](#)

## 工作方式

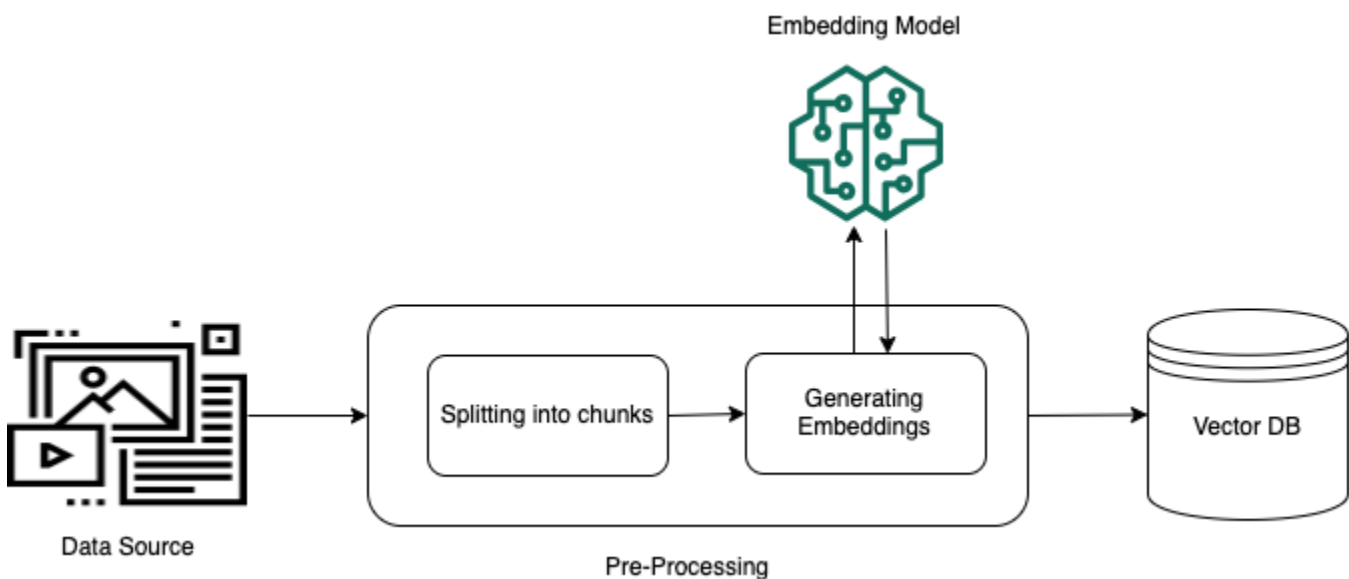
Amazon Bedrock 的知识库可帮助您利用检索增强生成 (RAG)，这是一种流行的技术，涉及从数据存储中提取信息以增强大型语言模型 (LLM) 生成的响应。如果使用数据来源建立知识库，应用程序就可以查询该知识库以返回信息，通过直接引用数据来源或使用根据查询结果生成的自然响应来回答查询。

借助知识库，您可以使用在查询知识库时接收到的上下文来丰富自己构建的应用程序。它从繁重的建筑管道中抽象出来，为您提供 out-of-the-box RAG 解决方案来缩短应用程序的构建时间，从而缩短上市时间。添加知识库还可以提高成本效益，因为无需持续训练模型即可利用您的私有数据。

下面的示意图说明了 RAG 的执行方法。知识库可以通过自动执行 RAG 的设置和实施中的几个步骤来简化这个过程。

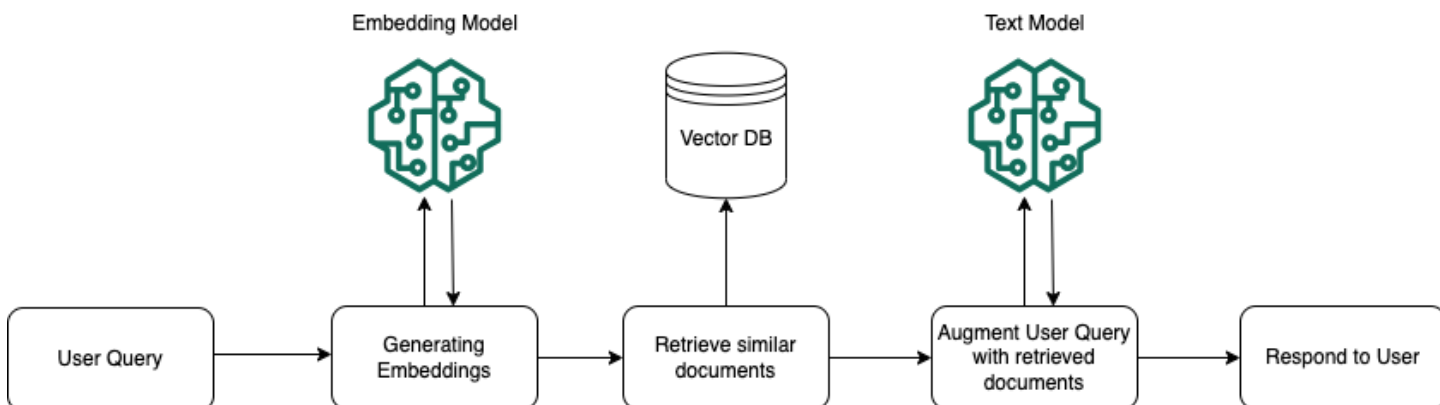
### 预处理数据

为了能够有效检索私有数据，通常的做法是先将文档拆分为可管理的数据块，以便进行有效检索。然后将这些数据块转换为嵌入内容并写入向量索引，同时保持与原文档的映射。这些嵌入内容用于确定查询和数据来源文本之间的语义相似性。下图说明了如何预处理向量数据库的数据。



### 运行时执行

在运行时，使用嵌入模型将用户的查询转换为向量。然后，通过将文档向量与用户查询向量进行比较来查询向量索引，查找与用户查询在语义上相似的数据块。最后，使用从向量索引中检索到的数据块中的附加上下文来增强用户提示。接下来，将提示和附加上下文一起发送给模型，以便为用户生成响应。下图说明了 RAG 如何在运行时运行以增强对用户查询的响应。



## Amazon Bedrock 知识库支持的区域和型号

以下区域支持 Amazon Bedrock 知识库：

### 区域

美国东部 ( 弗吉尼亚州北部 )

US West ( Oregon )

您可以使用以下模型将数据源嵌入矢量存储中：

| 模型名称                              | 型号标识                      |
|-----------------------------------|---------------------------|
| Amazon Titan Embeddings G1 - Text | 亚马逊。 titan-embed-text-v1  |
| CohereEmbed ( 英语 )                | 一致。 embed-english-v3      |
| CohereEmbed ( 多语言 )               | 一致。 embed-multilingual-v3 |

在从知识库中检索信息后，您可以使用以下模型生成响应：

| 模型                         | 型号标识                                     |
|----------------------------|------------------------------------------|
| AnthropicClaudev2.0        | anthropic.claude-v2                      |
| AnthropicClaudev2.1        | anthropic.claude-v 2:                    |
| AnthropicClaude 3 Sonnetv1 | anthropic.claude-3-sonnet-20240229-v 1:0 |
| AnthropicClaude Instantv1  | 人类。 claude-instant-v1                    |

## Amazon Bedrock 知识库的先决条件

在创建知识库之前，您需要满足以下先决条件：

1. [准备包含您希望知识库包含的信息的文件](#)，以便为知识库创建数据源。然后将文件上传到 Amazon S3 存储桶。
2. ( 可选 ) [设置您选择的矢量存储](#)。如果您计划使用在 Amazon OpenSearch Serverless 中自动为您创建矢量存储，则可以跳过此先决条件。AWS Management Console
3. ( 可选 ) 按照中的说明创建具有适当权限的自定义 AWS Identity and Access Management (IAM) [服务角色为 Amazon Bedrock 知识库创建服务角色](#)。如果您计划使用自动为您创建服务角色，AWS Management Console 则可以跳过此先决条件。
4. ( 可选 ) 按照中的步骤设置额外的安全配置[知识库资源的加密](#)。

### 主题

- [为您的知识库设置数据源](#)
- [在支持的矢量存储中为您的知识库设置矢量索引](#)

## 为您的知识库设置数据源

数据源包含可在查询知识库时检索的信息的文件。您可以通过将源[文档文件上传到 Amazon S3 存储桶来设置知识库的数据源](#)。

检查每个源文档文件是否符合以下要求：

- 该文件必须采用以下支持的格式之一：

| 格式             | 扩展程序       |
|----------------|------------|
| 纯文本            | .txt       |
| 降价             | .md        |
| HyperText 标记语言 | .html      |
| 微软 Word 文档     | .doc/.docx |
| 以逗号分隔的值        | .csv       |
| 微软 Excel 电子表格  | .xls/.xlsx |
| 便携式文档          | .pdf       |

- 文件大小不超过 50 MB 的配额。

以下主题描述了准备数据源的可选步骤。

## 主题

- [向文件添加元数据以允许筛选](#)
- [源代码块](#)

## 向文件添加元数据以允许筛选

您可以选择向数据源中的文件添加元数据。元数据允许在知识库查询期间筛选您的数据。

### 元数据文件要求

要在数据源中包含文件的元数据，请创建一个 JSON 文件，该文件由一个 `metadataAttributes` 字段组成，该字段映射到每个元数据属性都有键值对的对象。然后将其上传到 Amazon S3 存储桶中与源文档文件相同的文件夹。下面显示了元数据文件的一般格式：

```
{
 "metadataAttributes": {
 "${attribute1}": "${value1}",
 "${attribute2}": "${value2}",
 ...
 }
}
```

```
}
}
```

这些属性的值支持以下数据类型：

- 字符串
- 数字
- 布尔值

检查每个元数据文件是否符合以下要求：

- 该文件与其关联的源文档文件同名。
- 在文件扩展名 `.metadata.json` 后面追加（例如，如果您有一个名为 `A.txt` 的文件，则该元数据文件必须命名为 `.txt.metadata.a.json`。
- 文件大小不超过 10 KB 的配额。
- 该文件与其关联的源文档文件位于 Amazon S3 存储桶中的同一个文件夹中。

#### Note

如果您要向 Amazon OpenSearch Serverless 矢量存储中的现有向量索引添加元数据，请检查向量索引是否配置了允许筛选的 `faiss` 引擎。如果使用 `hnswlib` 引擎配置了向量索引，则必须执行以下操作之一：

- 在控制台中 [@@ 创建新的知识库](#)，让 Amazon Bedrock 在 Amazon OpenSearch Serverless 中自动为您创建矢量索引。
- 在 [@@ 矢量存储中创建另一个向量索引](#)，然后选择 `faiss` 作为引擎。然后 [创建新的知识库](#) 并指定新的向量索引。

如果您要向 Amazon Aurora 数据库集群中的现有向量索引添加元数据，则在开始摄取之前，必须为元数据文件中的每个元数据属性向表中添加一列。元数据属性值将写入这些列。

[同步数据源](#)后，您可以在[知识库查询](#)期间筛选结果。

## 元数据文件示例

例如，如果您有一个名为 `# oscars-coverage_20240310.pdf` 的源文档，其中包含新闻报道，则可能需要按 `#份` 或 `##` 等属性对其进行分类。要为此文件创建元数据，请执行以下步骤：

1. 创建一个名为 `oscars-coverage_20240310.pdf.metadata.json` 的文件，内容如下：

```
{
 "metadataAttributes": {
 "genre": "entertainment",
 "year": 2024
 }
}
```

2. `# oscars-coverage_20240310.pdf.metadata.json ##### S3 ##### oscars-coverage_20240310.pdf #####`
3. [创建知识库](#) 如果您还没有。然后，[同步您的数据源](#)。

## 源代码块

在将您的数据摄取到知识库的过程中，Amazon Bedrock 会将每个文件拆分为多个块。数据块是指在查询数据来源所属知识库时返回的数据来源摘录。

Amazon Bedrock 提供了分块策略，您可以使用这些策略对数据进行分块。您也可以通过自己对源文件进行分块来预处理数据。考虑要对数据源使用以下哪种分块策略：

- 默认分块 – 默认情况下，Amazon Bedrock 会自动将您的源数据拆分为多个数据块，因此每个数据块最多包含大约 300 个词元。如果文档包含的词元少于 300 个，则不会再对其进行拆分。
- 固定大小分块 – Amazon Bedrock 会将源数据拆分为与您设置的大小相近的数据块。
- 不分块 – Amazon Bedrock 将把每个文件都视为一个数据块。如果选择这个选项，则可能需要在将文档上传到 Amazon S3 存储桶之前，先将其拆分为单独的文件以进行预处理。

## 在支持的矢量存储中为您的知识库设置矢量索引

您可以通过创建用于存储以下数据的字段来设置受支持的向量索引，从而对数据源进行索引。

- 由您选择的嵌入模型从数据源中的文本生成的向量。
- 从数据源中的文件中提取的文本块。



- Amazon Bedrock 管理的与您的知识库相关的元数据。
- (如果您使用 Amazon Aurora 数据库并想要设置[筛选](#)) 与源文件关联的元数据。如果您计划在其他矢量存储中设置过滤，则无需设置这些字段进行筛选。

选择与您将用于创建矢量索引的服务对应的选项卡。

#### Note

如果您希望让 Amazon Bedrock 在 Amazon OpenSearch Serverless 中自动为您创建矢量索引，请跳过此先决条件并继续操作。[创建知识库](#)要了解如何设置向量索引，请选择与所选方法对应的选项卡，然后按照步骤操作。

## Amazon OpenSearch Serverless

1. 要在 Amazon OpenSearch Serverless 中配置权限并创建矢量搜索集合 AWS Management Console，请按照《[亚马逊 OpenSearch 服务开发者指南](#)》中[使用矢量搜索集合](#)中的步骤 1 和步骤 2 进行操作。设置收藏夹时请注意以下注意事项：
  - a. 为该系列指定您选择的名称和描述。
  - b. 要将您的收藏设为私有，请在“安全”部分选择“标准创建”。然后，在网络访问设置部分，选择 VPC 作为访问类型，然后选择 VPC 终端节点。有关为亚马逊 OpenSearch 无服务器集合设置 VPC 终端节点的更多信息，请参阅《[亚马逊 OpenSearch 服务开发者指南](#)》中的[使用接口终端节点 \(AWS PrivateLink\) 访问亚马逊 OpenSearch 无服务器](#)。
2. 创建馆藏后，请记下创建知识库时的馆藏 ARN。
3. 在左侧导航窗格中，选择无服务器下的集合。然后选择您的矢量搜索集合。
4. 选择“索引”选项卡。然后选择“创建矢量索引”。
5. 在向量索引详细信息部分，在向量索引名称字段中输入索引的名称。
6. 在向量场部分中，选择添加向量场。Amazon Bedrock 将您的数据源的矢量嵌入存储在此字段中。提供以下配置：
  - 向量字段名称-提供字段的名称（例如，**embeddings**）。
  - 引擎-用于搜索的向量引擎。选择 faiss。
  - 维度 – 向量中的维度数。请参考下表，确定向量应包含多少维度：

| 模型             | 尺寸    |
|----------------|-------|
| TitanG1 嵌入式-文本 | 1,536 |
| CohereEmbed英语  | 1024  |
| CohereEmbed多语言 | 1024  |

- 距离指标 – 用于衡量向量之间相似性的指标。我们建议使用欧几里得。

7. 展开元数据管理部分并添加两个字段来配置向量索引，以存储知识库可以使用向量检索的其他元数据。下表描述了要为每个字段指定的字段和值：

| 字段描述                                         | 映射字段                                 | 数据类型 | 可筛选   |
|----------------------------------------------|--------------------------------------|------|-------|
| Amazon Bedrock 会从您的数据中分块原始文本，并将这些数据块存储在此字段中。 | 您选择的名称（例如， <b>text</b> ）             | 字符串  | True  |
| Amazon Bedrock 将与您的知识库相关的元数据存储在该字段中。         | 您选择的名称（例如， <b>bedrock-metadata</b> ） | 字符串  | False |

8. 请记住您在创建知识库时为矢量索引名称、矢量字段名称和元数据管理映射字段名称选择的名称。然后选择创建。

创建向量索引后，您可以继续[创建知识库](#)。下表汇总了您将在何处输入您记下的每条信息。

| 字段     | 知识库设置中的相应字段（控制台） | 知识库设置中的相应字段 (API) | 描述                    |
|--------|------------------|-------------------|-----------------------|
| 集合 ARN | 集合 ARN           | CollectionARN     | 矢量搜索集合的亚马逊资源名称 (ARN)。 |
| 向量索引名称 | 向量索引名称           | vectorIndexName   | 向量索引的名称。              |

| 字段                | 知识库设置中的相应字段 ( 控制台 ) | 知识库设置中的相应字段 (API) | 描述                               |
|-------------------|---------------------|-------------------|----------------------------------|
| 向量字段名称            | 向量字段                | 向量字段              | 用于存储数据源的向量嵌入的字段名称。               |
| 元数据管理 ( 第一个映射字段 ) | 文本字段                | 文本字段              | 用于存储来自数据源的原始文本的字段名称。             |
| 元数据管理 ( 第二个映射字段 ) | 基岩管理的元数据字段          | 元数据字段             | 用于存储 Amazon Bedrock 管理的元数据的字段名称。 |

有关在 Amazon OpenSearch Serverless 中设置矢量库的更多详细文档，请参阅《亚马逊 OpenSearch 服务开发者指南》中的[使用矢量搜索集合](#)。

## Amazon Aurora

- 按照[准备用作知识库的 Aurora PostgreSQL 中的步骤创建 Amazon Aurora](#) 数据库 (DB) 集群、架构和表。创建表时，请使用以下列和数据类型对其进行配置。您可以使用自己喜欢的列名来代替下表中列出的列名。记下您选择的列名，以便在知识库设置期间提供这些名称。

| 列名称 | 数据类型        | 知识库设置中的相应字段 ( 控制台 ) | 知识库设置中的相应字段 (API) | 描述              |
|-----|-------------|---------------------|-------------------|-----------------|
| id  | UUID ( 主键 ) | 主键                  | primaryKeyField   | 包含每条记录的唯一标识符。   |
| 嵌入  | Vector      | 向量字段                | vectorField       | 包含数据来源的向量嵌入。    |
| 块   | 文本          | 文本字段                | textField         | 包含来自数据来源的原始文本块。 |

| 列名称      | 数据类型 | 知识库设置中的相应字段 (控制台) | 知识库设置中的相应字段 (API) | 描述                        |
|----------|------|-------------------|-------------------|---------------------------|
| metadata | JSON | 基岩管理的元数据字段        | metadataField     | 包含执行来源归因以及启用数据摄取和查询所需的元数据 |

- (可选) 如果您[向文件中添加了元数据以供筛选](#)，则还必须为文件中的每个元数据属性创建一列并指定数据类型 (文本、数字或布尔值)。例如，如果您的数据源中genre存在该属性，则需要添加一个名为的列genre并将其指定text为数据类型。在[摄取](#)期间，这些列将填充相应的属性值。
- 按照使用 [Amazon Aurora 进行 AWS Secrets Manager 密码管理中的步骤为您的 Aurora 数据库集群配置密钥和 AWS Secrets Manager](#)。
- 创建数据库集群并设置密钥后，记下以下信息。

| 知识库设置中的字段 (控制台)         | 知识库设置中的字段 (API)      | 描述                                   |
|-------------------------|----------------------|--------------------------------------|
| Amazon Aurora 数据库集群 ARN | resourceArn          | 数据库集群的 ARN。                          |
| 数据库名称                   | databaseName         | 数据库的名称                               |
| 表名称                     | tableName            | 数据库集群中的表的名称                          |
| 键 ARN                   | credentialsSecretArn | 您的数据库集群的 AWS Secrets Manager 密钥的 ARN |

## Pinecone

**Note**

如果您使用Pinecone，则表示您 AWS 同意授权代表您访问指定的第三方来源，以便向您提供矢量存储服务。您有责任遵守适用于使用和传输第三方服务数据的任何第三方条款。

有关在中设置矢量存储的详细文档Pinecone，请参阅 [Pinecone 作为 Amazon Bedrock 的知识库](#)。

请在设置向量存储时记下以下信息，创建知识库时需要填写这些信息：

- 连接字符串-索引管理页面的端点 URL。
- 命名空间- ( 可选 ) 用于向数据库写入新数据的命名空间。有关更多信息，请参阅[使用命名空间](#)。

创建Pinecone索引时还必须提供其他配置：

- 名称 – 向量索引的名称。选择所选的任何有效名称。稍后，创建知识库时，在向量索引名称字段中输入您选择的名称。
- 维度 – 向量中的维度数。请参考下表，确定向量应包含多少维度。

| 模型             | 尺寸    |
|----------------|-------|
| TitanG1 嵌入式-文本 | 1,536 |
| CohereEmbed英语  | 1024  |
| CohereEmbed多语言 | 1024  |

- 距离指标 – 用于衡量向量之间相似性的指标。我们建议您针对自己的使用案例试验不同的指标。我们建议从余弦相似度开始。

要访问您的Pinecone索引，您必须通过向 Amazon Bedrock 提供您的 Pinecone API 密钥。AWS Secrets Manager

为您的Pinecone配置设置密钥

1. 按照[创建密钥中的步骤操作](#)，将 [AWS Secrets Manager 密钥](#) 设置为 apiKey，将值设置为 API 密钥以访问您的Pinecone索引。

2. 要查找您的 API 密钥，请打开 [Pinecone 控制台](#) 并选择 API 密钥。
3. 创建密钥后，请记下 KMS 密钥的 ARN。
4. 按照 [解密包含知识库的矢量存储的 AWS Secrets Manager 密钥的权限](#) 中的步骤，向您的服务角色授予解密 KMS 密钥 ARN 的权限。
5. 稍后，当您创建知识库时，需要在凭证密钥 ARN 字段中输入该 ARN。

## Redis Enterprise Cloud

### Note

如果您使用 Redis Enterprise Cloud，则表示您 AWS 同意授权代表您访问指定的第三方来源，以便向您提供矢量存储服务。您有责任遵守任何适用于使用和传输来自第三方服务的数据的第三方条款。

有关在中设置矢量存储的详细文档 Redis Enterprise Cloud，请参阅 [Redis Enterprise Cloud 与 Amazon Bedrock 集成](#)。

请在设置向量存储时记下以下信息，创建知识库时需要填写这些信息：

- 终端节点 URL-数据库的公共终端节点 URL。
- 向量索引名称-数据库向量索引的名称。
- 向量字段-存储矢量嵌入的字段名称。请参考下表，确定向量应包含多少维度。

| 模型              | 尺寸    |
|-----------------|-------|
| TitanG1 嵌入式-文本  | 1,536 |
| CohereEmbed 英语  | 1024  |
| CohereEmbed 多语言 | 1024  |

- 文本字段 — Amazon Bedrock 存储原始文本块的字段名称。
- 基岩管理的元数据字段 — Amazon Bedrock 存储与您的知识库相关的元数据的字段名称。

要访问您的 Redis Enterprise Cloud 集群，您必须通过向 Amazon Bedrock 提供您的 Redis Enterprise Cloud 安全配置。AWS Secrets Manager

## 为您的 Redis Enterprise Cloud 配置设置密钥

1. 按照 [Transport Layer Security \(TLS\) \(传输层安全性协议 \( TLS \)\)](#) 中的步骤启用 TLS 以将您的数据库与 Amazon Bedrock 搭配使用。
2. 按照 [创建 AWS Secrets Manager 密钥](#) 中的步骤操作。使用您的 Redis Enterprise Cloud 配置中的相应值在密钥中设置以下密钥：
  - `username`— 访问 Redis Enterprise Cloud 数据库的用户名。要查找您的用户名，请在 [Redis 控制台](#) 中查看数据库的安全部分。
  - `password`— 访问 Redis Enterprise Cloud 数据库的密码。要查找您的密码，请在 [Redis 控制台](#) 中查看数据库的安全部分。
  - `serverCertificate` – 来自 Redis Cloud 证书颁发机构的证书的内容。按照 [下载证书](#) 中的步骤从 Redis 管理控制台下载服务器证书。
  - `clientPrivateKey` – 来自 Redis Cloud 证书颁发机构的证书的私钥。按照 [下载证书](#) 中的步骤从 Redis 管理控制台下载服务器证书。
  - `clientCertificate` – 来自 Redis Cloud 证书颁发机构的证书的公钥。按照 [下载证书](#) 中的步骤从 Redis 管理控制台下载服务器证书。
3. 创建密钥后，请记住它的 ARN。稍后，当您创建知识库时，需要在凭证密钥 ARN 字段中输入该 ARN。

## 创建知识库

### Note

您无法使用 root 用户创建知识库。在开始这些步骤之前，请使用 IAM 用户登录。


在 Amazon S3 中设置数据源和您选择的矢量存储后，您可以创建知识库。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 创建知识库

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console

2. 从左侧导航窗格中选择知识库。
3. 在知识库部分，选择创建知识库。
4. 在提供知识库详细信息页面上，设置以下配置：
  - a. (可选) 在知识库详细信息部分，更改默认名称并提供知识库描述。
  - b. 在 IAM 权限部分，选择一个提供 Amazon Bedrock 访问其他 AWS 服务的权限的 AWS Identity and Access Management (IAM) 角色。您可以让 Amazon Bedrock 创建服务角色或选择[您创建的自定义角色](#)。
  - c. (可选) 向知识库添加标签。有关更多信息，请参阅[标记资源](#)。
  - d. 选择下一步。
5. 在“设置数据源”页面上，提供用于知识库的数据源的信息：
  - a. (可选) 更改默认的数据源名称。
  - b. 提供包含[您为数据源准备](#)的文件的对象的 S3 URI。


 Note

选择与您正在创建的知识库位于同一区域的 Amazon S3 存储桶。否则，您的数据源将无法[同步](#)。

- c. 如果您使用客户托管密钥加密了 Amazon S3 数据，请选择为亚马逊 S3 数据添加客户管理的 AWS KMS 密钥，然后选择 KMS 密钥以允许 Amazon Bedrock 对其进行解密。有关更多信息，请参阅[对传递给 Amazon OpenSearch 服务的信息进行加密](#)。
- d. (可选) 要配置以下高级设置，请展开“高级设置-可选”部分。
  - i. 在将您的数据转换为嵌入式数据时，Amazon Bedrock 会使用默认 AWS 拥有和管理的密钥对您的数据进行加密。要使用自己的 KMS 密钥，请展开高级设置，选择自定义加密设置（高级），然后选择密钥。有关更多信息，请参阅[加密数据提取期间的临时数据存储](#)。
  - ii. 从以下选项中选择数据源的分块策略：
    - 默认分块 – 默认情况下，Amazon Bedrock 会自动将您的源数据拆分为多个数据块，因此每个数据块最多包含 300 个词元。如果文档包含的词元少于 300 个，则不会再对其进行拆分。
    - 固定大小分块 – Amazon Bedrock 会将源数据拆分为与您设置的大小相近的数据块。配置以下选项。



- 最大词元数量 – Amazon Bedrock 创建的数据块不会超过您选择的词元数量。
- 数据块之间的重叠百分比 – 每个数据块将按照您选择的百分比与连续的数据块重叠。
- 不分块 – Amazon Bedrock 将把每个文件都视为一个数据块。如果选择这个选项，则可能需要将文档拆分为单独的文件以进行预处理。

 Note

创建完数据来源后，就无法更改分块策略。

- e. 选择下一步。
6. 在嵌入模型部分，选择[支持的嵌入模型](#)，将您的数据转换为知识库的矢量嵌入。
  7. 在矢量数据库部分，选择以下选项之一来存储知识库的矢量嵌入：
    - 快速创建新的矢量图库 — Amazon Bedrock 为您创建[亚马逊 OpenSearch 无服务器矢量搜索集合](#)。使用此选项，将使用必填字段和必要配置为您设置公共矢量搜索集合和矢量索引。创建集合后，您可以在 Amazon OpenSearch Serverless 控制台或通过 AWS API 对其进行管理。有关更多信息，请参阅《亚马逊 OpenSearch 服务开发者指南》中的[使用矢量搜索集合](#)。如果选择此选项，则可以选择启用以下设置：
      - a. 要启用冗余活动副本，以便在基础设施出现故障时不会影响矢量存储的可用性，请选择启用冗余（活动副本）。

 Note

我们建议您在测试知识库时禁用此选项。当您准备好部署到生产环境时，我们建议您启用冗余活动副本。有关定价的信息，请参阅[OpenSearch 无服务器定价](#)

- b. 要使用客户管理的密钥加密自动矢量存储，请选择为 Amazon S OpenSearch erverless 矢量添加客户管理的 KMS 密钥（可选），然后选择密钥。有关更多信息，请参阅[对传递给 Amazon OpenSearch 服务的信息进行加密](#)。
- 选择已创建的向量存储 – 选择包含已创建的向量数据库的服务。填写字段，允许 Amazon Bedrock 将知识库中的信息映射到您的数据库，以便它可以存储、更新和管理嵌入内容。有关这些字段如何映射到您创建的字段的更多信息，请参阅[在支持的矢量存储中为您的知识库设置矢量索引](#)。

**Note**

如果您在 Amazon OpenSearch Serverless 或 Amazon Aurora 中使用数据库，则需要事先配置元数据字段映射下的字段。如果您在 Pinecone 或 Redis Enterprise Cloud 中使用数据库，则可以在此处提供这些字段的名称，Amazon Bedrock 将在向量存储中为您动态创建它们。

8. 选择下一步。
9. 在查看并创建页面上，查看知识库的配置和详细信息。在任何需要修改的部分中选择“编辑”。如果您感到满意，请选择创建知识库。
10. 创建知识库所用的时间取决于您提供的数据量。知识库创建完成后，知识库的状态将更改为“就绪”。

## API

要创建知识库，请使用 [Amazon Bedrock 代理构建时终端节点](#) 发送 `CreateKnowledgeBase` 请求，并提供名称、描述、应执行的操作说明以及编排的基础模型。


**Note**

如果您更喜欢让 Amazon Bedrock 在亚马逊 OpenSearch 服务中为您创建和管理向量存储，请使用控制台。有关更多信息，请参阅 [创建知识库](#)。

- 在 `roleArn` 字段中向 ARN 提供创建知识库的权限。
- 在 `knowledgeBaseConfiguration` 对象的 `embeddingModelArn` 字段中提供要使用的嵌入模型。
- 在 `storageConfiguration` 对象中提供向量存储的配置。有关更多信息，请参阅 [在支持的向量存储中为您的知识库设置向量索引](#)。
  - 对于亚马逊 OpenSearch 服务数据库，请使用 `opensearchServerlessConfiguration` 对象。
  - 对于 Pinecone 数据库，请使用 `pineconeConfiguration` 对象。
  - 对于 Redis Enterprise Cloud 数据库，请使用 `redisEnterpriseCloudConfiguration` 对象。
  - 对于 Amazon Aurora 数据库，请使用 `rdsConfiguration` 对象。

创建知识库后，从包含知识库文件的 S3 存储桶中创建数据源。要创建数据源，[CreateDataSource](#) 请发送请求。

- 在 `dataSourceConfiguration` 字段中提供包含数据源文件的 S3 存储桶的信息。
- 指定如何对 `vectorIngestionConfiguration` 字段中的数据源进行分块。有关更多信息，请参阅 [为您的知识库设置数据源](#)。

 Note

创建数据源后，您无法更改分块配置。

- ( 可选 ) 在将您的数据转换为嵌入式数据时，Amazon Bedrock 默认使用 AWS 拥有和管理的密钥对您的数据进行加密。要使用自己的 KMS 密钥，请将其包含在 `serverSideEncryptionConfiguration` 对象中。有关更多信息，请参阅 [知识库资源的加密](#)。

## 为您的知识库设置安全配置

创建知识库后，可能需要设置以下安全配置：

### 主题

- [为您的知识库设置数据访问策略](#)
- [为您的 Amazon OpenSearch 无服务器知识库设置网络访问策略](#)

## 为您的知识库设置数据访问策略

如果您使用的是 [自定义角色](#)，请为新创建的知识库设置安全配置。如果您让 Amazon Bedrock 为您创建服务角色，则可以跳过此步骤。按照与您设置的数据库相对应的选项卡中的步骤进行操作。

### Amazon OpenSearch Serverless

要将对 Amazon OpenSearch Serverless 集合的访问权限限制为知识库服务角色，请创建数据访问策略。您可以通过以下方式执行此操作：

- 按照亚马逊 OpenSearch 服务开发者指南中的 [创建数据访问策略 \( 控制台 \)](#) 中的步骤使用亚马逊 OpenSearch 服务控制台。
- 通过发送带有 [OpenSearch 无服务器端点](#) 的 [CreateAccessPolicy](#) 请求来使用 AWS API。有关 AWS CLI 示例，请参阅 [创建数据访问策略 \( AWS CLI \)](#)。

使用以下数据访问策略，指定 Amazon OpenSearch Serverless 集合和您的服务角色：

```
[
 {
 "Description": "${data_access_policy_description}",
 "Rules": [
 {
 "Resource": [
 "index/${collection_name}/*"
],
 "Permission": [
 "aoss:DescribeIndex",
 "aoss:ReadDocument",
 "aoss:WriteDocument"
],
 "ResourceType": "index"
 }
],
 "Principal": [
 "arn:aws:iam::${account-id}:role/${kb-service-role}"
]
 }
]
```

## Pinecone or Redis Enterprise Cloud

要集成Pinecone或Redis Enterprise Cloud向量索引，请将以下基于身份的策略附加到您的知识库服务角色，以允许其访问向量索引的 AWS Secrets Manager 密钥。

```
{
 "Version": "2012-10-17",
 "Statement": [{
 "Effect": "Allow",
 "Action": [
 "bedrock:AssociateThirdPartyKnowledgeBase"
],
 "Resource": [
 "arn:aws:iam::${region}:${account-id}:knowledge-base/${knowledge-base-id}"
],
 "Condition": {
 "StringEquals": {
```

```

 "bedrock:ThirdPartyKnowledgeBaseCredentialsSecretArn":
 "arn:aws:iam::${region}:${account-id}:secret:${secret-id}"
 }
 }
}]]
}

```

## 为您的 Amazon OpenSearch 无服务器知识库设置网络访问策略

如果您使用私有 Amazon OpenSearch Serverless 集合作为知识库，则只能通过 AWS PrivateLink VPC 终端节点对其进行访问。您可以在[设置亚马逊 OpenSearch 无服务器矢量集合时创建私有的 Amazon OpenSearch Serverless 集合](#)，也可以在配置其网络访问策略时将现有的亚马逊 OpenSearch 无服务器集合（包括亚马逊 Bedrock 控制台为您创建的集合）设为私有。

《亚马逊 OpenSearch 服务开发者指南》中的以下资源将帮助您了解私有 Amazon OpenSearch Serverless 集合所需的设置：

- 有关为私有 Amazon OpenSearch Serverless 集合设置 VPC 终端节点的更多信息，请参阅[使用接口终端节点访问 Amazon OpenSearch Serverless \( \) AWS PrivateLink](#)。
- 有关 Amazon Serverless 中网络访问策略的更多信息，请参阅 Amazon OpenSearch Serv [OpenSearch 无服务器的网络访问权限](#)。

要允许 Amazon Bedrock 知识库访问亚马逊 OpenSearch 无服务器的私有馆藏，您必须编辑亚马逊 OpenSearch 无服务器集合的网络访问策略，以允许 Amazon Bedrock 作为源服务。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

1. 打开亚马逊 OpenSearch 服务控制台，[网址为 https://console.aws.amazon.com/aos/](https://console.aws.amazon.com/aos/)。
2. 在左侧导航窗格中，选择“收藏夹”。然后选择您的收藏。
3. 在“网络”部分，选择“关联策略”。
4. 选择编辑。
5. 在“选择策略定义方法”中，选择 JSON。
6. 在 JSON 编辑器中，粘贴以下策略。

```

[
 {

```

```

 "AllowFromPublic": false,
 "Description": "${network access policy description}",
 "Rules": [
 {
 "ResourceType": "collection",
 "Resource": [
 "collection/${collection-id}"
]
 },
],
 "SourceServices": [
 "bedrock.amazonaws.com"
]
 }
]

```

7. 选择更新。

## API

要编辑您的 Amazon OpenSearch Serverless 馆藏的网络访问策略，请执行以下操作：

1. 使用 [OpenSearch 无服务器端点](#) 发送 [GetSecurityPolicy](#) 请求。指定策略name的，然后指定type为network。记下响应中的 policyVersion。
2. 使用 [OpenSearch 无服务器端点](#) 发送 [UpdateSecurityPolicy](#) 请求。至少要指定以下字段：

| 字段   | 描述                                       |
|------|------------------------------------------|
| name | 策略的名称                                    |
| 策略版本 | 从policyVersion 回复中返回给你。GetSecurityPolicy |
| type | 安全策略的类型。指定 network。                      |
| 策略   | 要使用的策略。指定以下 JSON 对象                      |

```

[
 {
 "AllowFromPublic": false,

```

```

 "Description": "${network_access_policy_description}",
 "Rules": [
 {
 "ResourceType": "collection",
 "Resource": [
 "collection/${collection-id}"
]
 },
],
 "SourceServices": [
 "bedrock.amazonaws.com"
]
 }
]

```

有关 AWS CLI 示例，请参阅[创建数据访问策略 \( AWS CLI \)](#)。

- 按照[创建网络策略 \( 控制 OpenSearch 台 \)](#)中的步骤使用 Amazon Service 控制台。与其创建网络策略，不如记下集合详细信息的“网络”小节中的关联策略。

## 同步以将您的数据源提取到知识库中

创建知识库后，将数据源提取到知识库中，以便对它们进行索引并可供查询。Ingestion 会将数据源中的原始数据转换为矢量嵌入。它还会将原始文本和[您设置为筛选的任何相关元数据关联起来](#)，以增强查询流程。在开始摄取之前，请检查您的数据源是否满足以下条件：

- 数据源的 Amazon S3 存储桶与知识库位于同一区域。
- 这些文件采用支持的格式。有关更多信息，请参阅[在支持的矢量存储中为您的知识库设置矢量索引](#)。
- 文件大小不超过 50 MB 的最大文件大小。有关更多信息，请参阅[知识库配额](#)。
- 如果您的数据源包含[元数据文件](#)，请检查以下条件以确保元数据文件不会被忽略：
  - 每个 .metadata.json 文件都与其关联的源文件同名。
  - 如果知识库的向量索引位于 Amazon OpenSearch Serverless 矢量存储中，请检查该 faiss 引擎是否配置了向量索引。如果使用引 nmslib 引擎配置了向量索引，则必须执行以下操作之一：
    - 在控制台中@@@ [创建新的知识库](#)，让 Amazon Bedrock 在 Amazon OpenSearch Serverless 中自动为您创建向量索引。

- 在@@ [向量存储中创建另一个向量索引](#)，然后选择faiss作为引擎。然后[创建新的知识库](#)并指定新的向量索引。
- 如果知识库的向量索引位于 Amazon Aurora 数据库集群中，请在开始摄取之前，检查您的索引表是否包含元数据文件中每个元数据属性的列。

### Note

每次在数据源的 S3 存储桶中添加、修改或删除文件时，都必须同步该数据源，以便将其重新索引到知识库中。同步是增量的，因此 Amazon Bedrock 仅处理自上次同步以来添加、修改或删除的 S3 存储桶中的对象。

要了解如何将数据源提取到知识库中，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 摄取数据来源

1. 通过以下网址打开 Amazon Bedrock 控制台：<https://console.aws.amazon.com/bedrock/>。
2. 从左侧导航窗格中选择知识库，然后选择您的知识库。
3. 在数据来源部分中，选择同步以开始数据摄取。
4. 数据摄取完成后，如果成功则会显示绿色成功横幅。
5. 您可以选择一个数据来源，查看其同步历史记录。选择查看警告以查看数据摄取作业失败的原因。

## API

要将数据源提取到您为知识库配置的向量存储中，请使用适用于 [Amazon Bedrock 的代理构建时](#) 终端节点发送 [StartIngestionJob](#) 请求。指定 `knowledgeBaseId` 和 `dataSourceId`。

在 [Amazon Bedrock 代理构建时终端节点的 GetIngestionJob 请求中使用](#) [响应中 ingestionJobId 返回的内容](#) 来跟踪摄取任务的状态。此外，请指定 `knowledgeBaseId` 和 `dataSourceId`。

- 提取作业完成后，响应中的 `status` 为 `COMPLETE`。
- 响应中的 `statistics` 对象会返回有关数据来源中文档的提取是否成功的信息。



您还可以使用适用于 [Amazon Bedrock 的代理](#) 构建时终端节点发送 [ListIngestionJobs](#) 请求，从而查看数据源的所有摄取任务的信息。指定要将 knowledgeBaseId 数据采集到的知识库的 dataSourceId 和。

- 通过在 filters 对象中指定要搜索的状态来筛选结果。
- 通过指定 sortBy 对象，按作业启动时间或作业状态进行排序。您可以按升序或降序进行排序。
- 在 maxResults 字段中设置要在响应中返回的结果数量上限。如果结果多于您设置的数字，则响应会返回一个 nextToken，您可以在另一个 [ListIngestionJobs](#) 请求中发送该请求以查看下一批作业。

## 在 Amazon Bedrock 中测试知识库

设置完知识库后，您可以通过发送查询和查看响应来测试其行为。您还可以设置查询配置以自定义信息检索。当您对知识库的行为感到满意时，可以将应用程序设置为查询知识库或将知识库附加到代理。

选择一个主题以了解更多信息。

### 主题

- [查询知识库并返回结果或生成响应](#)
- [查询配置](#)

## 查询知识库并返回结果或生成响应

要学习如何查询知识库，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 测试知识库

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择知识库。
3. 在知识库部分，执行以下操作之一：
  - 选择要测试的知识库旁边的单选按钮，然后选择测试知识库。测试窗口从右侧展开。
  - 选择要测试的知识库。测试窗口从右侧展开。
4. 根据您的用例，选择或清除“为查询生成响应”。

- 要返回直接从知识库中检索到的信息，请关闭“生成响应”。Amazon Bedrock 将从您的数据源中返回与查询相关的文本块。
  - 要根据从知识库中检索到的信息生成响应，请打开“生成响应”。Amazon Bedrock 将根据您的数据源生成响应，并在脚注中引用其提供的信息。
5. 如果您打开生成响应，请选择选择模型以选择用于生成响应的模型。然后选择“应用”。
  6. ( 可选 ) 选择配置图标



( )  
以打开配置。您可以修改以下配置：

- 搜索类型-指定如何查询您的知识库。有关更多信息，请参阅 [搜索类型](#)。
  - 源区块的最大数量-指定要检索的最大源区块数。有关更多信息，请参阅 [源区块的最大数量](#)。
  - 筛选器-指定最多 5 个筛选器组，在每个组中最多指定 5 个筛选器，用于文件的元数据。有关更多信息，请参阅 [元数据和筛选](#)。
  - 知识库提示模板-如果您开启了“生成响应”，则可以将默认的提示模板替换为自己的模板，以自定义发送到模型以生成响应的提示。有关更多信息，请参阅 [知识库提示模板](#)。
7. 在聊天窗口的文本框中输入查询，然后选择运行，让知识库返回响应。
  8. 您可以通过以下方式检查响应。
    - 如果您没有生成响应，则会按相关性顺序直接返回文本块。
    - 如果您生成了回复，请选择脚注以查看该部分回复的引用来源摘录。选择链接以导航到包含该文件的 S3 对象。
    - 要查看有关每个脚注中引用的区块的详细信息，请选择“显示来源详细信息”。您可以在来源详细信息窗格中执行以下操作：
      - 要查看您为查询设置的配置，请展开查询配置。
      - 要查看有关源区块的详细信息，请选择其旁边的右箭头



( )  
将其展开。您可以看到以下信息：

- 来自源代码块的原始文本。要复制此文本，请选择复制图标



)。

要导航到包含该文件的 S3 对象，请选择外部链接图标



)。

- 与源区块关联的元数据。属性键和值在与源文档关联 `.metadata.json` 的文件中定义。有关更多信息，请参阅 [元数据文件要求](#)。

## 聊天选项

- 如果您正在生成响应，则可以选择更改模型以使用不同的模型生成响应。如果更改模型，聊天窗口中的文本将被完全清除。
- 通过选择或清除“生成回复”，在为查询生成回复和直接返回报价之间切换。如果更改设置，聊天窗口中的文本将被完全清除。
- 要清除聊天窗口，请选择扫帚图标



)。

- 要复制聊天窗口中的所有输出，请选择复制图标



)。

## API

### 检索

要查询知识库并仅从数据源返回相关文本，请使用 [Amazon Bedrock 代理运行时端点](#) 发送 [Retrieve](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。

下表简要描述了参数和请求正文（有关详细信息和请求结构，请参阅[检索请求语法](#)：

| Variable         | 必填？ | 应用场景              |
|------------------|-----|-------------------|
| knowledgeBaselId | 支持  | 指定要查询的知识库         |
| 检索查询             | 支持  | 包含用于指定查询的 text 字段 |
| nextToken        | 不支持 | 返回下一批回复           |

| Variable | 必填？ | 应用场景                                |
|----------|-----|-------------------------------------|
| 检索配置     | 不支持 | 包括用于自定义矢量搜索的 <a href="#">查询配置</a> 。 |

下表简要描述了响应正文（有关详细信息和响应结构，请参阅[检索响应语法](#)）：

| Variable  | 应用场景                                |
|-----------|-------------------------------------|
| 检索结果      | 包含源区块、源的 Amazon S3 位置以及区块score的相关性。 |
| nextToken | 在另一个请求中使用以返回下一批结果。                  |

## RetrieveAndGenerate

要查询知识库并使用基础模型根据数据源的结果生成响应，请使用 [Amazon Bedrock 代理运行时终端节点](#) 发送 [RetrieveAndGenerate](#) 请求。

下表简要描述了参数和请求正文（有关详细信息和请求结构，请参阅[RetrieveAndGenerate 请求语法](#)）：

| Variable              | 必填？ | 应用场景                                              |
|-----------------------|-----|---------------------------------------------------|
| input                 | 支持  | 包含用于指定查询的text字段                                   |
| retrieveAndGenerate配置 | 支持  | 用于指定要查询的知识库、用于生成响应的模型以及 <a href="#">可选的查询配置</a> 。 |
| sessionId             | 不支持 | 使用相同的值继续相同的会话并维护信息                                |
| 会话配置                  | 不支持 | 包括用于加密会话的 KMS 密钥                                  |

下表简要描述了响应正文（有关详细信息和响应结构，请参阅[检索响应语法](#)）：

| Variable  | 应用场景                                                                                                                                                  |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 引用        | 在每个对象中包含生成的响应的一部分 <code>generatedResponsePart</code> ，以及对象中的源代码块和 <code>content</code> 对象中源的 Amazon S3 位置。 <code>location retrieved References</code> |
| output    | 包含生成的整个响应。                                                                                                                                            |
| sessionId | 包含会话的 ID，您可以在另一个请求中重复使用该会话以保持相同的对话                                                                                                                    |

### Note

如果您在生成响应时收到提示超过字符限制的错误消息，则可以通过以下方式缩短提示：

- 减少源区块的最大数量（这会缩短中为 `$search_results$` 占位符填写的内容）。[知识库提示模板](#)
- 使用使用较小区块的分块策略重新创建数据源（这会缩短中为 `$search_results$` 占位符填写的内容）。[知识库提示模板](#)
- 缩短提示模板。
- 缩短用户查询（这会缩短中为 `$query$` 占位符填写的内容）。[知识库提示模板](#)

## 查询配置

在查询知识库时，您可以修改配置，以自定义检索和响应生成。要了解有关配置以及如何在控制台或 API 中修改配置的更多信息，请从以下主题中进行选择。

### 搜索类型

搜索类型定义了如何查询知识库中的数据源。以下搜索类型是可能的：

- 默认 — Amazon Bedrock 为您决定搜索策略。

- Hybrid — 将搜索向量嵌入（语义搜索）与搜索原始文本相结合。目前，只有包含可筛选文本字段的 Amazon OpenSearch Serverless 矢量存储才支持混合搜索。如果您使用其他矢量存储，或者您的 Amazon OpenSearch Serverless 矢量存储不包含可筛选的文本字段，则查询将使用语义搜索。
- 语义-仅搜索向量嵌入。

要了解如何定义搜索类型，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

按照中的控制台步骤进行操作[查询知识库并返回结果或生成响应](#)。打开“配置”窗格时，您将看到以下搜索类型选项：

- 默认 — Amazon Bedrock 决定哪种搜索策略最适合您的矢量商店配置。
- 混合 — Amazon Bedrock 同时使用向量嵌入和原始文本来查询知识库。只有当您使用配置了可筛选文本字段的 Amazon OpenSearch Serverless 矢量存储时，此选项才可用。
- 语义 — Amazon Bedrock 使用其向量嵌入来查询知识库。

## API

[RetrieveAndGenerate](#)发出[Retrieve](#)或请求时，请添加一个映射到[KnowledgeBaseRetrievalConfiguration](#)对象的[retrievalConfiguration](#)字段。要查看此字段的位置，请参阅 API 参考中的[Retrieve](#)和[RetrieveAndGenerate](#)请求正文。

以下 JSON 对象显示了该[KnowledgeBaseRetrievalConfiguration](#)对象中设置搜索类型配置所需的最少字段：

```
"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "overrideSearchType": "HYBRID | SEMANTIC"
 }
}
```

在[overrideSearchType](#)字段中指定搜索类型。您有以下选项：

- 如果您未指定值，Amazon Bedrock 将决定哪种搜索策略最适合您的矢量商店配置。
- HYBRID — Amazon Bedrock 同时使用向量嵌入和原始文本来查询知识库。只有当您使用配置了可筛选文本字段的 Amazon OpenSearch Serverless 矢量存储时，此选项才可用。
- 语义 — Amazon Bedrock 使用其向量嵌入来查询知识库。

## 源区块的最大数量

当您查询知识库时，Amazon Bedrock 会在响应中返回最多五个源块。要修改要返回的最大源区块数，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

按照中的控制台步骤进行操作[查询知识库并返回结果或生成响应](#)。在“配置”窗格中，展开“源区块的最大数量”。

### API

[RetrieveAndGenerate](#)发出[Retrieve](#)或请求时，请添加一个映射到[KnowledgeBaseRetrievalConfiguration](#)对象的retrievalConfiguration字段。要查看此字段的位置，请参阅 API 参考中的[Retrieve](#)和[RetrieveAndGenerate](#)请求正文。

以下 JSON 对象显示了[KnowledgeBaseRetrievalConfiguration](#)对象中设置要返回的最大源区块数所需的最小字段：

```
"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "numberOfResults": number
 }
}
```

指定要在numberOfResults字段中返回的最大源区块数（有关[KnowledgeBaseRetrievalConfiguration](#)可接受值的范围，请参阅中的numberOfResults字段）。

## 元数据和筛选

您的数据源可以包括与源文档关联的元数据文件。元数据文件包含作为键值对的属性，这些属性是您为源文档定义的。有关为数据源文件创建元数据的更多信息，请参阅[向文件添加元数据以允许筛选](#)。要在知识库查询期间使用过滤器，请检查您的知识库是否满足以下要求：

- 包含您的数据源的 Amazon S3 存储桶至少包含一个与其关联的源文档同名的 .metadata.json 文件。
- 如果您知识库的向量索引位于 Amazon OpenSearch Serverless 矢量存储中，请检查该faiss引擎是否配置了向量索引。如果使用引nmslib引擎配置了向量索引，则必须执行以下操作之一：

- 在控制台中@@ [创建新的知识库](#)，让 Amazon Bedrock 在 Amazon OpenSearch Serverless 中自动为您创建矢量索引。
- 在@@ [矢量存储中创建另一个向量索引](#)，然后选择faiss作为引擎。然后[创建新的知识库](#)并指定新的向量索引。

在修改用于筛选的查询配置时，可以使用以下筛选运算符：

#### 筛选运算符

| 运算符          | 控制台 | API 筛选器名称                       | 支持的属性数据类型  | 筛选结果                    |
|--------------|-----|---------------------------------|------------|-------------------------|
| 等于           | =   | <a href="#">equals</a>          | 字符串、数字、布尔值 | 属性与您提供的值相匹配             |
| 不等于          | !=  | <a href="#">noteEquals</a>      | 字符串、数字、布尔值 | 属性与您提供的值不匹配             |
| Greater than | >   | <a href="#">大于</a>              | number     | 属性大于您提供的值               |
| 大于或等于        | >=  | <a href="#">greaterThanOr等于</a> | number     | 属性大于或等于您提供的值            |
| Less than    | <   | <a href="#">小于</a>              | number     | 属性小于您提供的值               |
| 小于或等于        | <=  | <a href="#">lessThanOr等于</a>    | number     | 属性小于或等于您提供的值            |
| In           | :   | <a href="#">在</a>               | 字符串列表      | 属性在您提供的列表中              |
| 不在           | !:  | <a href="#">notIn</a>           | 字符串列表      | 属性不在您提供的列表中             |
| 开始于          | ^   | <a href="#">开头为</a>             | 字符串        | 属性以您提供的字符串开头（仅支持 Amazon |



| 运算符 | 控制台 | API 筛选器名称 | 支持的属性数据类型 | 筛选结果                         |
|-----|-----|-----------|-----------|------------------------------|
|     |     |           |           | OpenSearch Serverless 矢量存储 ) |

要组合筛选运算符，可以使用以下逻辑运算符：

### 逻辑运算符

| 运算符 | 控制台 | API 筛选器字段名称            | 筛选结果            |  |
|-----|-----|------------------------|-----------------|--|
| And | 以及  | <a href="#">andAll</a> | 结果满足群组中的所有筛选表达式 |  |
| Or  | 或者  | <a href="#">orAll</a>  | 结果满足组中至少一个筛选表达式 |  |

要了解如何使用元数据筛选结果，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

按照中的控制台步骤进行操作[查询知识库并返回结果或生成响应](#)。当你打开“配置”窗格时，你会看到“筛选器”部分。以下过程描述了不同的用例：

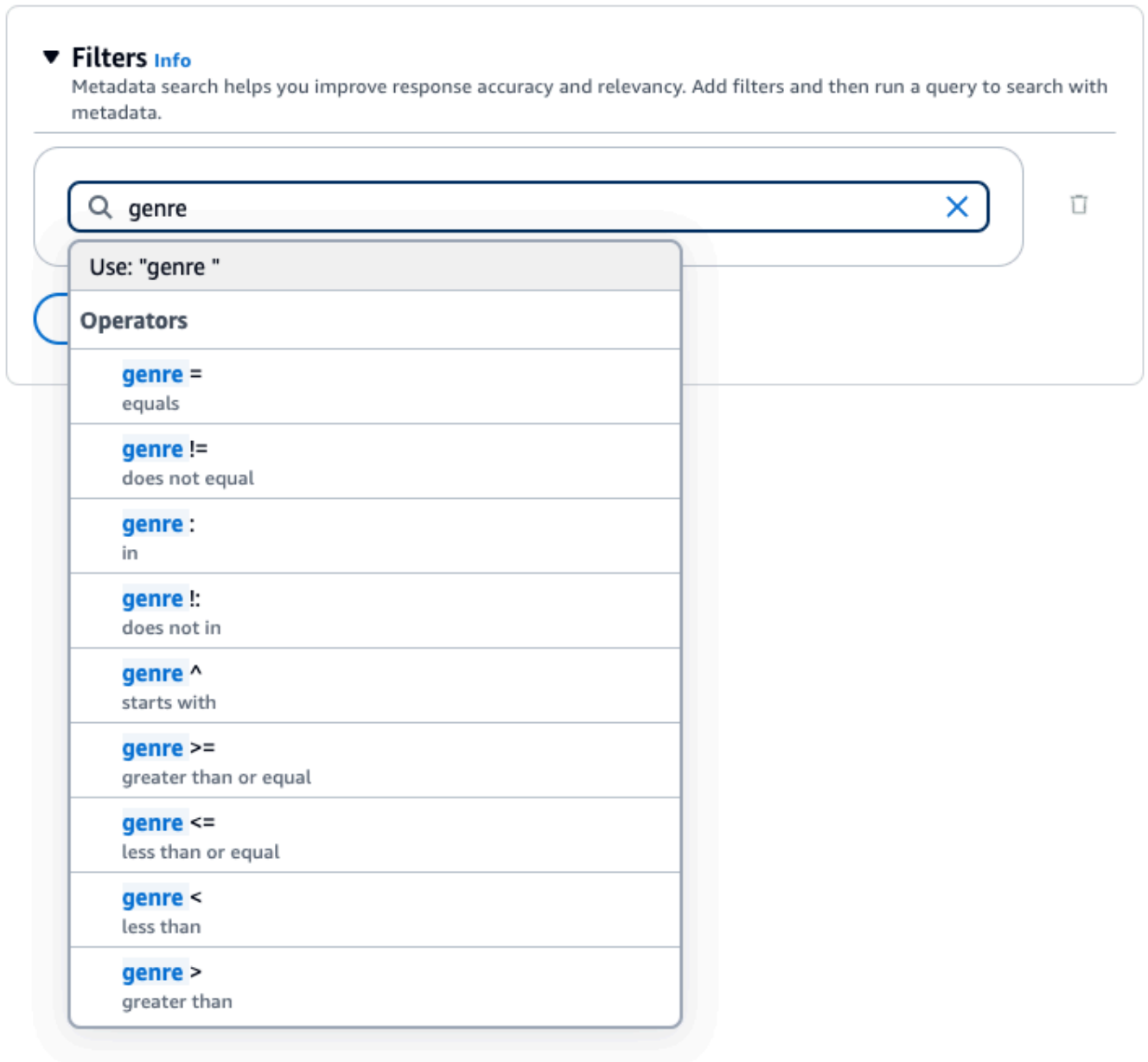
- 要添加筛选条件，请在框中输入元数据属性、筛选运算符和值来创建筛选表达式。用空格分隔表达式的每个部分。按 Enter 键添加过滤器。

有关可接受的筛选运算符的列表，请参阅上面的筛选运算符表。在元数据属性后面添加空格时，还可以看到筛选运算符列表。

#### Note

必须用引号将字符串括起来。

例如，您可以通过添加以下筛选器来筛选来自包含genre元数据属性的源文档"entertainment"的结果，该属性的值为：**genre = "entertainment"**。



- 要添加其他筛选器，请在框中输入另一个筛选表达式，然后按 Enter。您最多可以在群组中添加 5 个过滤器。

▼ **Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

Q Enter

genre = "entertainment" × and ▼ year > 2018 ×

+ Add Group

- 默认情况下，查询将返回满足您提供的所有筛选表达式的结果。要返回至少满足其中一个筛选表达式的结果，请在任意两个筛选操作之间选择和下拉菜单，然后选择或。

▼ **Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

Q Enter

genre = "entertainment" × and ▲ and ✓ or

+ Add Group

- 要组合不同的逻辑运算符，请选择 + Add Group 以添加筛选器组。在新组中输入筛选表达式。您最多可以添加 5 个筛选器组。

▼ **Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

🗑️

genre = "entertainment" ✕ and ▼ year > 2018 ✕ |

AND ▼

🗑️

genre : ["cooking", "sports"] ✕ and ▼ author ^ "C" ✕ |

+ Add Group

- 要更改所有筛选组之间使用的逻辑运算符，请在任意两个筛选器组之间选择 AND 下拉菜单，然后选择 OR。

▼ **Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

Q Enter

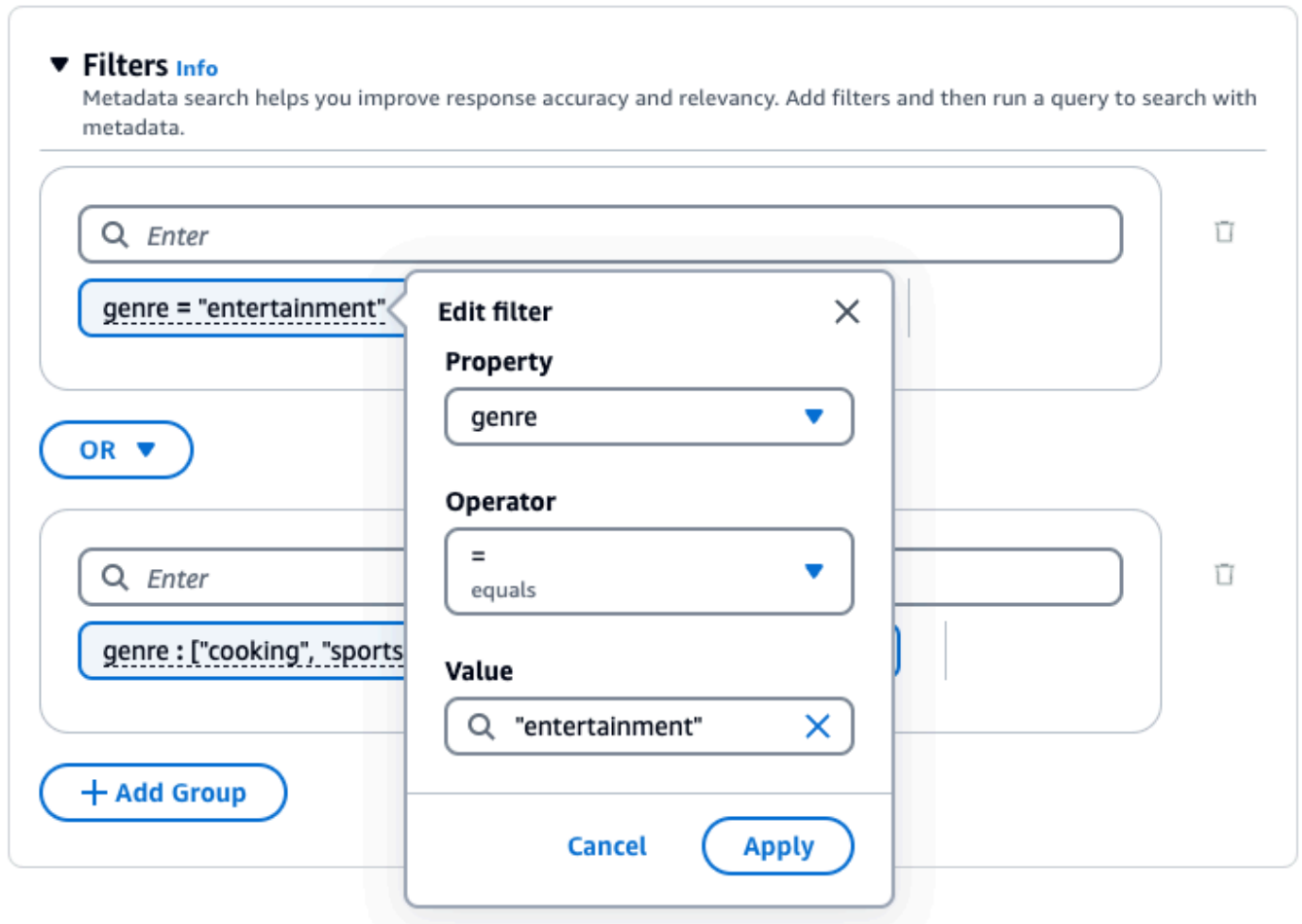
genre = "entertainment" × and ▼ year > 2018 ×

AND ▲  
AND  
OR

genre : ["cooking", "sports"] × and ▼ author ^ "C" ×

+ Add Group

- 要编辑筛选器，请将其选中，修改筛选操作，然后选择 Apply。



- 要移除筛选器组，请选择该组旁边的垃圾桶图标



( )。

要移除过滤器，请选择过滤器旁边的删除图标



( )。

▼ **Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

🗑️

genre = "entertainment" ✕ and ▼ year > 2018 ✕ |

OR ▼

🗑️

genre : ["cooking", "sports"] ✕ and ▼ author ^ "C" ✕ |

+ Add Group

下图显示了一个筛选器配置示例，除了类型为或2018且作者以开头的文档外"entertainment"，该配置还返回所有以其类型为"cooking"或"sports"的文档编写的文档"C"。

**▼ Filters Info**  
Metadata search helps you improve response accuracy and relevancy. Add filters and then run a query to search with metadata.

---

genre = "entertainment"
✕

and ▼

year > 2018
✕

**OR ▼**

genre : ["cooking", "sports"]
✕

and ▼

author ^ "C"
✕

**+ Add Group**

## API

[RetrieveAndGenerate](#)发出[Retrieve](#)或请求时，请添加一个映射到[KnowledgeBaseRetrievalConfiguration](#)对象的`retrievalConfiguration`字段。要查看此字段的位置，请参阅 API 参考中的[Retrieve](#)和[RetrieveAndGenerate](#)请求正文。

以下 JSON 对象显示了[KnowledgeBaseRetrievalConfiguration](#)对象中为不同用例设置过滤器所需的最少字段：

1. 使用一个过滤运算符（参见上面的过滤运算符表）。

```

"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "filter": {
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string", "string", ...]
 }
 }
 }
}

```



```

 }
 }
}

```

2. 使用逻辑运算符（参见上面的逻辑运算符表）最多可组合 5 个。

```

"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "filter": {
 "andAll | orAll": [
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 ...
]
 }
 }
}

```

3. 使用逻辑运算符将最多 5 个筛选运算符组合到一个筛选器组中，使用第二个逻辑运算符将该筛选器组与其他筛选运算符合并。

```

"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "filter": {
 "andAll | orAll": [
 "andAll | orAll": [
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 }
],
],
 }
 }
}

```

```

 ...
],
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 }
]
 }
}
}
}

```

4. 将最多 5 个筛选器组嵌入到另一个逻辑运算符中，从而合并多达 5 个过滤器组。您可以创建一层嵌入。

```

"retrievalConfiguration": {
 "vectorSearchConfiguration": {
 "filter": {
 "andAll | orAll": [
 "andAll | orAll": [
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 ...
],
 "andAll | orAll": [
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 "<filter-type>": {
 "key": "string",
 "value": "string" | number | boolean | ["string",
"string", ...]
 },
 ...
],
],
 },
 },
}

```



| 字段    | 地图到            | 筛选结果            |
|-------|----------------|-----------------|
| orAll | 最多 5 种过滤器类型的列表 | 结果满足组中至少一个筛选表达式 |

有关示例，请参阅[发送查询并包含筛选条件 \(检索\)](#)和[发送查询并包含筛选条件 \(RetrieveAndGenerate\)](#)。

## 知识库提示模板

当您查询知识库并请求生成响应时，Amazon Bedrock 使用将说明和上下文与用户查询相结合的提示模板来构造发送到模型以生成响应的提示。您可以使用以下工具设计提示模板：

- 提示占位符 — Amazon Bedrock 知识库中的预定义变量，这些变量在知识库查询期间在运行时动态填充。在系统提示符中，您会看到这些占位符被\$符号包围。以下列表描述了您可以使用的占位符：

| Variable                       | 取而代之的是                                                             | 模型                                            | 必填？            |
|--------------------------------|--------------------------------------------------------------------|-----------------------------------------------|----------------|
| \$query\$                      | 发送到知识库的用户查询。                                                       | AnthropicClaude Instant , AnthropicClaudev2.x | 支持             |
|                                |                                                                    | Anthropic Claude 3 Sonnet                     | 否 (自动包含在模型输入中) |
| \$search_results\$             | 检索到的用户查询结果。                                                        | 全部                                            | 支持             |
| \$output_format_instructions\$ | 格式化响应生成和引文的基本说明。因型号而异。如果您自己定义格式化说明，我们建议您删除此占位符。如果没有这个占位符，响应将不包含引文。 | 全部                                            | 不支持            |

| Variable         | 取而代之的是 | 模型 | 必填？ |
|------------------|--------|----|-----|
| \$current_time\$ | 当前时间。  | 全部 | 不支持 |

- XML 标签 — Anthropic 模型支持使用 XML 标签来构造和描述您的提示。使用描述性标签名称以获得最佳结果。例如，在默认的系统提示中，您将看到用于描述先前问过的问题数据库的<database>标签)。有关更多信息，请参阅[Anthropic 用户指南](#)中的[使用 XML 标签](#)。

有关一般提示工程指南，请参阅 [提示工程准则](#)。

选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

按照中的控制台步骤进行操作[查询知识库并返回结果或生成响应](#)。在测试窗口中，打开“生成响应”。然后，在配置窗格中，展开知识库提示模板部分。

1. 选择编辑。
2. 根据需要，在文本编辑器中编辑系统提示，包括提示占位符和 XML 标签。要恢复到默认提示模板，请选择“重置为默认值”。
3. 编辑完成后，选择保存更改。要在不保存系统提示的情况下退出，请选择“放弃更改”。

## API

当您提出[RetrieveAndGenerate](#)请求时，请添加一个映射到[GenerationConfiguration](#)对象的generationConfiguration字段。要查看此字段的位置，请参阅 API 参考中的[RetrieveAndGenerate](#)请求正文。

以下 JSON 对象显示了[GenerationConfiguration](#)对象中设置要返回的最大源区块数所需的最小字段：

```

"generationConfiguration": {
 "promptTemplate": {
 "textPromptTemplate": "string"
 }
}

```

在textPromptTemplate字段中输入您的自定义提示模板，必要时包括提示占位符和XML 标签。有关系统提示符中允许的最大字符数，请参阅中的textPromptTemplate字段[GenerationConfiguration](#)。

## 管理数据源

创建数据源后，您可以查看有关它的详细信息、更新或删除它。

### 查看有关数据源的信息

您可以查看有关您的数据源及其同步历史记录的信息。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

#### Console

查看有关数据源的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择知识库。
3. 在数据源部分，选择要查看其详细信息的数据源。
4. 数据源概述包含有关数据源的详细信息。
5. 同步历史记录包含有关数据源何时同步的详细信息。要查看同步事件失败的原因，请选择一个同步事件并选择查看警告。

#### API

要获取有关数据源的信息，请使用 [Amazon Bedrock 代理构建时终端节点](#) 发送 [GetDataSource](#) 请求，dataSourceId并指定其所属知识库knowledgeBaseId的和。

要列出有关知识库数据源的信息，请使用[适用于 Amazon Bedrock 的代理构建时终端节点](#) 发送 [ListDataSources](#) 请求并指定知识库的 ID。

- 要设置响应中返回的最大结果数，请使用字maxResults段。
- 如果结果多于您设置的数字，则响应会返回nextToken。您可以在另一个ListDataSources请求中使用此值来查看下一批结果。

要获取数据源的同步事件信息，请使用适用于 [Amazon Bedrock 构建时终端节点的代理](#) 发送 [GetIngestionJob](#) 请求。指定 `dataSourceId`、`knowledgeBaseId` 和 `ingestionJobId`。

要在知识库中列出数据源的同步历史记录，请使用 [Amazon Bedrock 代理构建](#) 时终端节点发送 [ListIngestionJobs](#) 请求。指定知识库和数据来源的 ID。您可以设置以下规范。

- 通过在 `filters` 对象中指定要搜索的状态来筛选结果。
- 通过指定 `sortBy` 对象，按作业启动时间或作业状态进行排序。您可以按升序或降序进行排序。
- 在 `maxResults` 字段中设置要在响应中返回的结果数量上限。如果结果多于您设置的数字，则响应会返回一个 `nextToken`，您可以在另一个 [ListIngestionJobs](#) 请求中发送该请求以查看下一批作业。

## 删除数据源

您可以通过以下方式更新数据源：

- 在包含数据源文件的 S3 存储桶中添加、更改或删除文件。
- 更改数据源的名称或 S3 存储桶，或者更改用于在数据摄取期间加密临时数据的 KMS 密钥。

每次在数据源的 S3 存储桶中添加、修改或删除文件时，都必须同步该数据源，以便将其重新索引到知识库中。同步是增量的，因此 Amazon Bedrock 仅处理自上次同步以来添加、修改或删除的 S3 存储桶中的对象。在开始摄取之前，请检查您的数据源是否满足以下条件：

- 这些文件采用支持的格式。有关更多信息，请参阅 [在支持的矢量存储中为您的知识库设置矢量索引](#)。
- 文件大小不超过 50 MB 的最大文件大小。有关更多信息，请参阅 [知识库配额](#)。
- 如果您的数据源包含 [元数据文件](#)，请检查以下条件以确保元数据文件不会被忽略：
  - 每个 `.metadata.json` 文件都与其关联的源文件同名。
  - 如果知识库的向量索引位于 Amazon OpenSearch Serverless 矢量存储中，请检查该 `faiss` 引擎是否配置了向量索引。如果使用 `hnswlib` 引擎配置了向量索引，则必须执行以下操作之一：
    - 在控制台中 [@@ 创建新的知识库](#)，让 Amazon Bedrock 在 Amazon OpenSearch Serverless 中自动为您创建向量索引。
    - 在 [@@ 矢量存储中创建另一个向量索引](#)，然后选择 `faiss` 作为引擎。然后 [创建新的知识库](#) 并指定新的向量索引。

- 如果知识库的向量索引位于 Amazon Aurora 数据库集群中，请在开始摄取之前，检查您的索引表是否包含元数据文件中每个元数据属性的列。

要了解如何更新数据源，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 更新数据源

1. （可选）对包含数据源文件的 S3 存储桶中的文件进行必要的更改。
2. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。
3. 从左侧导航窗格中选择知识库。
4. 在数据源部分，选择要同步的数据源旁边的单选按钮。
5. （可选）选择编辑，更改所有必要的配置，然后选择提交。
6. 选择“同步”。
7. 同步完成且状态变为“就绪”时，会出现绿色横幅。

## API

### 更新数据源

1. （可选）对包含数据源文件的 S3 存储桶中的文件进行必要的更改。
2. （可选）使用 [Amazon Bedrock 代理构建时终端节点](#) 发送 [UpdateDataSource](#) 请求，更改必要的配置并指定您不想更改的相同配置。

#### Note

你无法更改 chunkingConfiguration。使用现有请求发送请求 chunkingConfiguration。

3. 使用 [适用于 Amazon Bedrock 的代理构建时终端节点](#) 发送 [StartIngestionJob](#) 请求，指定和。dataSourceId knowledgeBaseId



## 删除数据源

如果您不再需要数据源，可以将其删除。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 删除数据来源

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择知识库。
3. 在数据源部分，选择要删除的数据源旁边的单选按钮。
4. 选择 Delete (删除)。
5. 成功删除数据源后，会出现绿色横幅。

### API

要从知识库中删除数据源，请发送 [DeleteDataSource](#) 请求，指定 `dataSourceId` 和 `knowledgeBaseId`。

## 管理知识库

设置知识库后，您可以查看有关知识库的信息、修改知识库或将其删除。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### 查看有关知识库的信息

您可以查看有关知识库的信息。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 查看有关知识库的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择知识库。
3. 要查看知识库的详细信息，请选择来源的名称，或选择来源旁边的单选按钮并选择编辑。
4. 在详细信息页面上，您可以执行以下操作：

- 要更改知识库的详细信息，请在知识库概述部分中选择编辑。
- 要更新知识库中附加的标签，请在标签部分中选择管理标签。
- 如果您更新创建知识库所依据的数据来源并且需要同步更改，请在数据来源部分中选择同步。
- 要查看数据来源的详细信息，请选择数据来源名称。在详细信息中，您可以在同步历史记录部分选择某同步事件旁边的单选按钮，然后选择查看警告以查看无法同步数据摄取作业中文件的原因。
- 要管理用于知识库的嵌入模型，请选择编辑预调配吞吐量。
- 编辑完成后，选择保存更改。

## API

要获取有关知识库的信息，请使用[适用于 Amazon Bedrock 的代理构建时终端节点发送GetKnowledgeBase请求](#)，指定 `knowledgeBaseId`

要列出有关您的知识库的信息，请使用[适用于 Amazon Bedrock 的代理构建时终端节点发送ListKnowledgeBases请求](#)。您可以设置要在响应中返回的结果数量上限。如果结果多于您设置的数字，则响应会返回 `nextToken`。您可以在另一个 [ListKnowledgeBases](#) 请求的 `nextToken` 字段中使用此值来查看下一批结果。

## 更新知识库

### Console

#### 更新知识库

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择知识库。
3. 选择知识库以查看其详细信息，或者选择知识库旁边的单选按钮并选择编辑。
4. 您可以通过以下方式修改知识库。
  - 通过在知识库概述部分选择编辑来更改知识库的配置。
  - 在“标签”部分中选择“管理标签”，更改附加到知识库的标签
  - 在“数据源”部分中管理数据源。有关更多信息，请参阅[管理数据源](#)。

5. 编辑完成后，选择保存更改。

## API

要更新知识库，请使用[适用于 Amazon Bedrock 的代理构建](#)时终端节点发送 `UpdateKnowledgeBase` 请求。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。

## 删除知识库

如果您不再需要知识库，可以将其删除。删除知识库时，还应执行以下操作以完全删除与知识库关联的所有资源。

- 将知识库与其关联的任何代理分开。
- 从知识库中编制索引的基础数据仍保留在您设置的矢量存储中，并且仍然可以检索。要删除数据，还需要删除包含数据嵌入的向量索引。

选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 删除知识库

1. 执行以下步骤之前，请务必先从与知识库关联的所有代理中删除该知识库。为此，请执行以下步骤：
  - a. 从左侧导航窗格中选择代理。
  - b. 选择要从中删除知识库的代理的名称。
  - c. 此时将出现红色横幅，警告您要从代理中删除已不存在的知识库引用。
  - d. 选中要删除的知识库旁边的单选按钮。选择更多，然后选择删除。
2. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
3. 从左侧导航窗格中选择知识库。
4. 选择知识库或选择知识库旁边的单选按钮。然后选择删除。
5. 查看有关删除知识库的警告。如果您接受这些条件，请在输入框中输入 **delete** 并选择删除进行确认。

6. 要完全删除知识库的矢量嵌入，您需要删除包含数据嵌入的向量索引。

## API

在删除知识库之前，请向适用于 [Amazon Bedrock 的代理构建时终端节点](#) 发出 [DisassociateAgentKnowledgeBase](#) 请求，从而取消知识库与其关联的任何代理的关联。

要删除知识库，请使用 [适用于 Amazon Bedrock 的代理构建时终端节点](#) 发送 [DeleteKnowledgeBase](#) 请求。

要完全删除知识库的矢量嵌入，您需要删除包含数据嵌入的向量索引。

## 部署知识库

要在应用程序中部署知识库，请将其设置为向知识库发出 [Retrieve](#) 或 [RetrieveAndGenerate](#) 请求。要了解如何使用这些 API 操作，请选择中的“API”选项卡 [在 Amazon Bedrock 中测试知识库](#)。

您也可以将知识库与代理相关联，代理将在编排过程中必要时调用该知识库。有关更多信息，请参阅 [Agents for Amazon Bedrock](#)。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 将知识库与代理关联

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择代理。
3. 选择要向其添加知识库的代理。
4. 在“工作草稿”部分，选择工作草稿。
5. 在“知识库”部分，选择“添加”。
6. 从“选择知识库”下的下拉列表中选择知识库，并指定代理应如何与知识库交互并返回结果的说明。

### 解除知识库与代理的关联

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console

2. 从左侧导航窗格中选择代理。
3. 选择要向其添加知识库的代理。
4. 在“工作草稿”部分，选择工作草稿。
5. 在知识库部分，选择知识库。
6. 选择删除。

## API

要将知识库与代理关联，[AssociateAgentKnowledgeBase](#)请发送请求。

- 包括详细内容description，说明代理应如何与知识库交互并返回结果。
- 将设置ENABLED为knowledgeBaseState以允许代理查询知识库。

您可以通过发送[UpdateAgentKnowledgeBase](#)请求来更新与代理关联的知识库。例如，您可能需要将设置knowledgeBaseState为ENABLED以解决问题。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。

要解除知识库与代理的关联，请发送请求。[DisassociateAgentKnowledgeBase](#)

# Agents for Amazon Bedrock

借助 Amazon Bedrock 代理，您可以在应用程序中构建和配置自主代理。代理可帮助您的最终用户根据组织数据和用户输入完成操作。代理协调基础模型 (FM)、数据源、软件应用程序和用户对话之间的交互。此外，代理会自动调用 API 来采取行动，并调用知识库来补充这些操作的信息。开发人员可以通过集成代理来加速生成式人工智能（生成人工智能）应用程序的交付，从而节省数周的开发工作量。

借助客服，您可以为客户自动执行任务并为他们回答问题。例如，您可以创建帮助客户处理保险索赔的代理或帮助客户进行旅行预订的代理。您无需配置容量、管理基础架构或编写自定义代码。Amazon Bedrock 可管理提示工程、内存、监控、加密、用户权限和 API 调用。

代理执行以下任务：

- 扩展基础模型以了解用户请求，并将代理必须执行的任务分解为较小的步骤。
- 通过自然对话从用户那里收集其他信息。
- 通过向公司系统发出 API 调用，采取措施满足客户的请求。
- 通过查询数据来源来提高性能和准确性。

要使用代理，请执行以下步骤：

1. （可选）创建知识库，以将私有数据存储在数据库。有关更多信息，请参阅[Amazon Bedrock 知识库](#)。
2. 为您的用例配置代理并添加代理可以执行的操作。要定义代理如何处理操作，请使用您选择的编程语言编写 Lambda 函数。
3. 将知识库与代理关联以提高代理的绩效。有关更多信息，请参阅[在 Amazon Bedrock 中创建代理](#)。
4. （可选）要根据您的特定用例自定义代理的行为，请修改代理执行的预处理、编排、知识库响应生成和后处理步骤的提示模板。有关更多信息，请参阅[Amazon Bedrock 中的高级提示](#)。
5. 在 Amazon Bedrock 控制台中或通过 API 调用测试您的代理。TSTALIASID 根据需要修改配置。使用跟踪来检查代理在编排过程中每个步骤的推理过程。有关更多信息，请参阅[测试 Amazon Bedrock 代理](#) 和 [在 Amazon Bedrock 中追踪事件](#)。
6. 当您对代理进行了充分的修改并准备将其部署到您的应用程序时，请创建一个别名以指向代理的某个版本。有关更多信息，请参阅[部署 Amazon Bedrock 代理](#)。
7. 将您的应用程序设置为对您的代理别名进行 API 调用。
8. 对代理进行迭代，并根据需要创建更多版本和别名。

## 主题

- [Amazon Bedrock 代理的工作原理](#)
- [Amazon Bedrock 代理支持的区域和型号](#)
- [Amazon Bedrock 代理的先决条件](#)
- [在 Amazon Bedrock 中创建代理](#)
- [为 Amazon Bedrock 代理创建行动组](#)
- [将知识库与 Amazon Bedrock 代理关联](#)
- [测试 Amazon Bedrock 代理](#)
- [管理 Amazon Bedrock 代理](#)
- [自定义 Amazon Bedrock 代理](#)
- [部署 Amazon Bedrock 代理](#)

## Amazon Bedrock 代理的工作原理

Amazon Bedrock 代理由以下两组主要的 API 操作组成，可帮助您设置和运行代理：

- 用于创建、配置和管理代理及其相关资源的 [@@ 构建时 API 操作](#)
- [运行时 API 操作](#)，用于使用用户输入调用您的代理，并启动编排以执行任务。

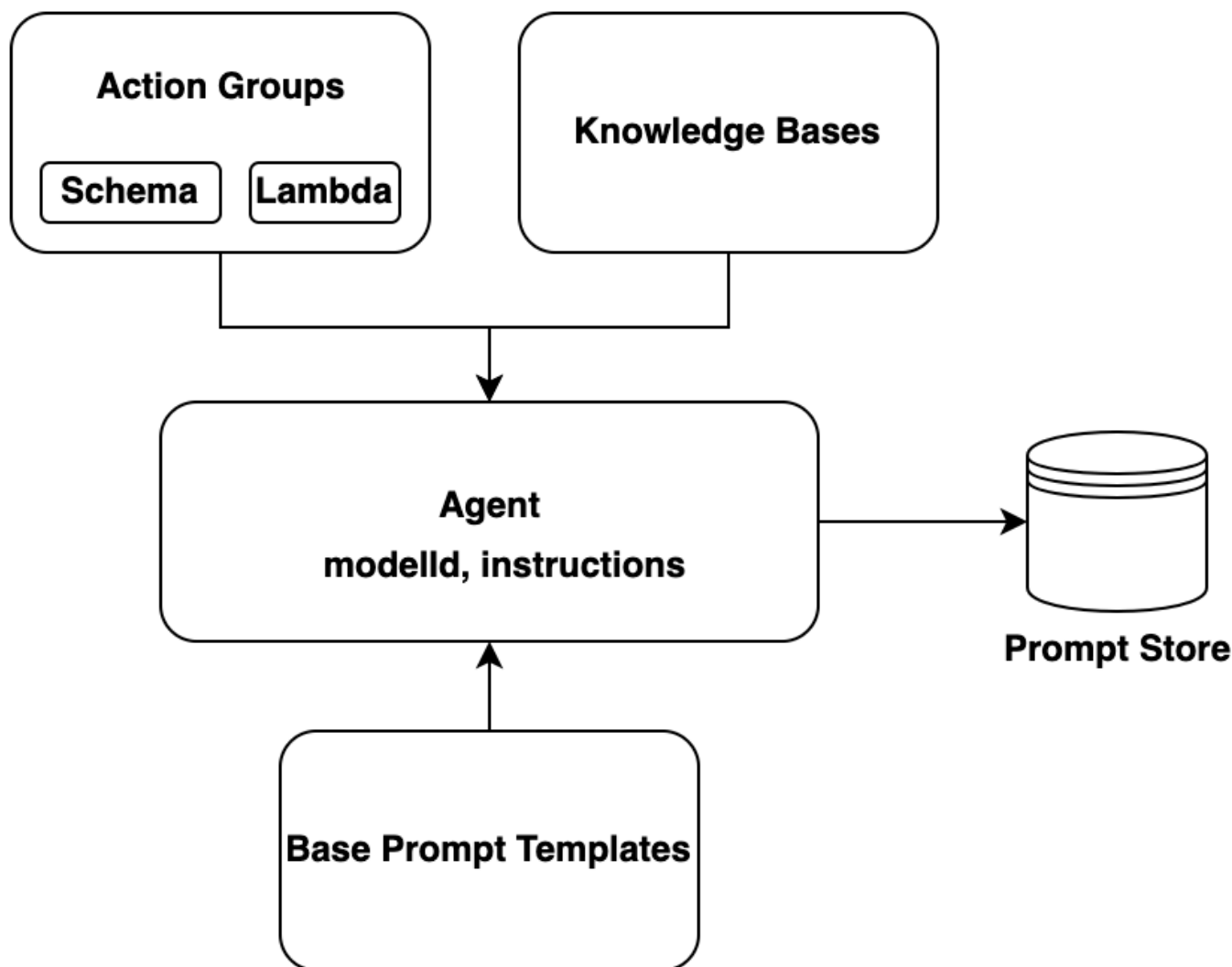
## 构建时配置

代理包含以下组件：

- **基础模型** — 您可以选择一个基础模型 (FM)，代理调用该模型来解释其编排过程中的用户输入和后续提示。代理还调用 FM 以生成响应和流程中的后续步骤。
- **说明**-您编写说明来描述代理的设计用途。借助高级提示，您可以在编排的每个步骤中为代理进一步自定义指令，并包含用于解析每个步骤输出的 Lambda 函数。
- **操作组 (可选)** — 您可以通过提供以下资源来定义代理应执行的操作：
  - 一种 OpenAPI 架构，用于定义代理可以调用以执行其任务的 API 操作。
  - 具有以下输入和输出的 Lambda 函数：
    - 输入-编排过程中识别的 API 操作和参数。
    - 输出 – API 调用的结果。

- 知识库 ( 可选 ) -将知识库与代理关联起来。代理会查询知识库以获取额外的上下文，以增强响应生成和对编排过程步骤的输入。
- 提示模板 — 提示模板是创建要提供给 FM 的提示的基础。Amazon Bedrock 代理公开了预处理、编排、知识库响应生成和后处理期间使用的默认四个基本提示模板。您可以选择编辑这些基本提示模板，以自定义代理在其序列的每个步骤中的行为。您也可以出于故障排除目的或在您认为不需要某个步骤时关闭步骤。有关更多信息，请参阅[Amazon Bedrock 中的高级提示](#)。

在构建时，将收集所有这些组件来构造基本提示，供代理执行编排，直到用户请求完成。通过高级提示，您可以根据其他逻辑和少样本示例来修改这些基本提示，以提高代理调用每个步骤的准确性。基本提示模板包含说明、操作描述、知识库描述和对话历史记录，您可以自定义所有这些内容以修改代理以满足您的需求。然后，您需要准备代理，该代理将代理的所有组件打包，包括安全配置。准备代理会使其进入可以在运行时对其进行测试的状态。下图显示了构建时 API 操作是如何构造代理的。





## 运行时进程

运行时由 [InvokeAgent](#) API 操作管理。此操作将启动代理序列，该序列由以下三个主要步骤组成。

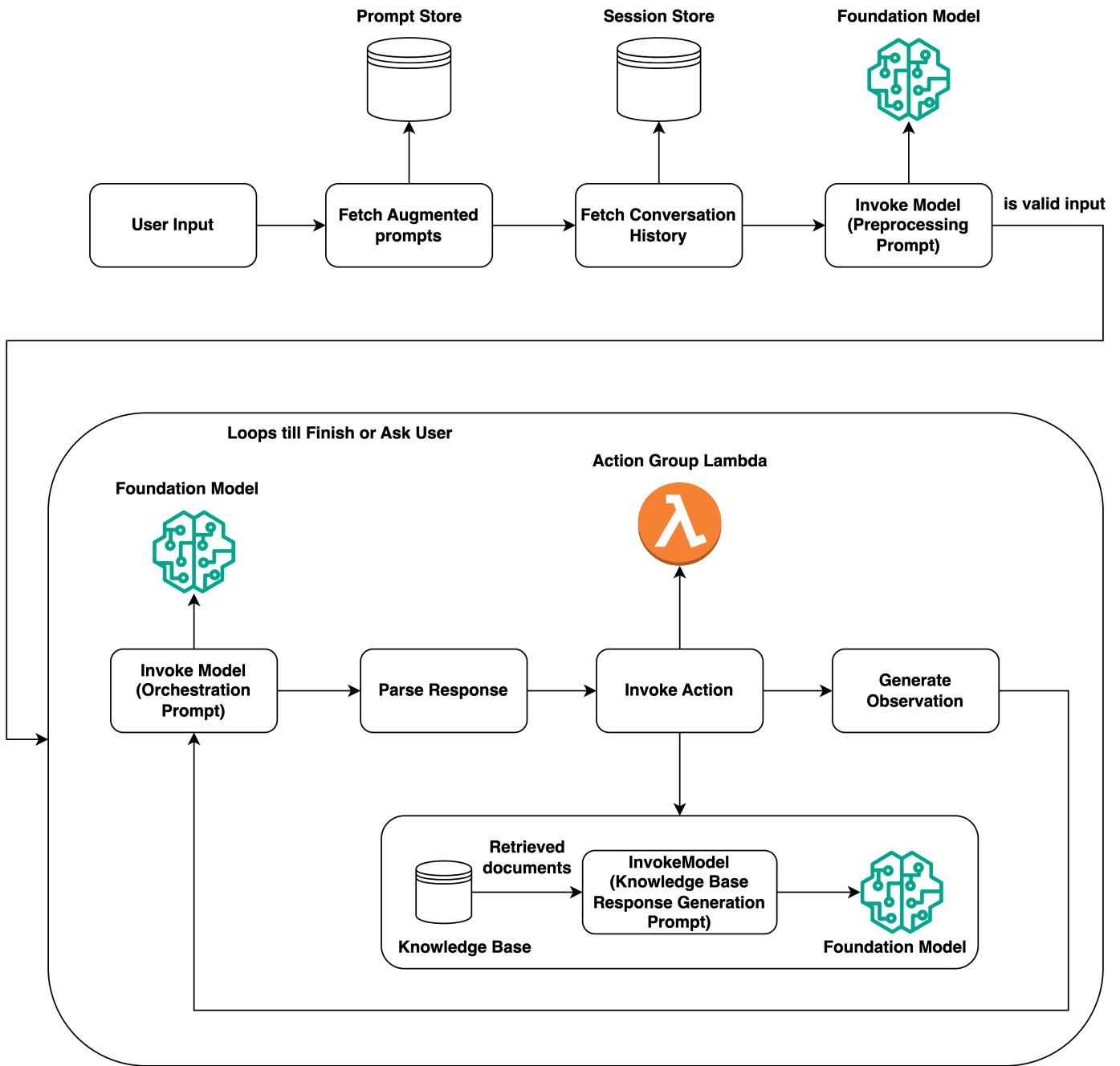
1. 预处理-管理代理如何对用户输入进行情境化和分类，并可用于验证输入。
2. 编排 — 解释用户输入，调用操作组和查询知识库，并将输出返回给用户或作为输入返回给用户或作为输入继续编排。编排包括以下步骤：
  - a. 代理通过根基模型解释输入，并生成理由来阐述下一步行动的逻辑依据。
  - b. 代理调用操作组并查询知识库（知识库响应生成）以检索其他上下文并汇总数据以增强其生成。
  - c. 代理通过调用操作组和汇总知识库的结果来生成输出，也称为观察结果。代理使用观察结果来增强基本提示，然后使用根基模型解释基本提示。然后，代理确定是否需要重复编排过程。
  - d. 此循环一直持续到代理向用户返回响应或需要提示用户提供额外信息为止。

在编排过程中，基本提示模板会使用您添加到代理中的代理说明、操作组和知识库进行扩展。然后，使用增强的基本提示符来调用 FM。FM 可以预测满足用户输入的最佳步伐和轨迹。在编排的每次迭代中，FM 都会预测要调用的 API 操作或要查询的知识库。

3. 后处理-代理格式化最终响应以返回给用户。默认情况下，此步骤处于关闭状态。

调用代理时，可以在运行时开启跟踪。通过跟踪，您可以在代理序列的每个步骤中跟踪代理的理由、操作、查询和观察结果。跟踪包括每个步骤向基础模型发送的完整提示以及基础模型的输出、API 响应和知识库查询。您可以使用跟踪来了解代理在每个步骤中的推理。有关更多信息，请参阅 [在 Amazon Bedrock 中追踪事件](#)。

随着用户与代理的会话继续 InvokeAgent 处理更多请求，对话历史记录会被保留。对话历史记录不断为编排基础提示模板添加上下文，有助于提高代理的准确性和性能。下图显示了代理在运行时的进程：



## Amazon Bedrock 代理支持的区域和型号

以下区域支持 Amazon Bedrock 代理：

## 区域

美国东部 ( 弗吉尼亚州北部 )

US West ( Oregon )

您可以将 Amazon Bedrock 代理与以下型号配合使用：

| 模型名称                      | 型号标识                  |
|---------------------------|-----------------------|
| AnthropicClaude Instantv1 | 人类。 claude-instant-v1 |
| AnthropicClaudev2.0       | anthropic.claude-v2   |
| AnthropicClaudev2.1       | anthropic.claude-v 2: |

## Amazon Bedrock 代理的先决条件

确保您的 IAM 角色具有执行与 Amazon Bedrock 代理相关的操作[所需的权限](#)。

客服人员使用操作组和知识库来帮助您的客户执行任务。以下是每种资源的简短描述：

- 操作组-定义代理可以帮助用户执行的操作。包括可以调用的 API、如何处理操作以及如何返回响应。
- 知识库-提供一个信息存储库，工程师可以查询这些信息来回答客户的疑问并改进其生成的响应。

在创建代理之前，请查看以下先决条件并确定需要满足哪些先决条件：

1. [设置一个行动组](#)。要让您的代理协调对系统的 API 调用，您必须至少添加一个操作组。如果您想稍后添加操作组，或者计划不为代理设置任何操作组，则可以跳过此先决条件。
2. [建立知识库](#)。要使用私有数据源改善对客户查询的响应，您可以关联至少一个知识库。如果您计划让代理没有与之相关的知识库，则可以跳过此先决条件
3. [为您的代理创建具有适当权限的自定义 AWS Identity and Access Management \(IAM\) 服务角色](#)。如果您计划使用自动为您创建服务角色，AWS Management Console 则可以跳过此先决条件。

## 在 Amazon Bedrock 中创建代理

要使用 Amazon Bedrock 创建代理，您需要设置以下组件：

- 代理的配置，它定义了代理的用途，并指明了代理用于生成提示和响应的基础模型 (FM)。
- ( 可选 ) 定义代理要执行的操作的操作组。
- ( 可选 ) 数据来源知识库，用于增强代理的生成式功能。

您可以在控制台或 API 中创建代理。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

视具体情况而定

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择“代理”。
3. 在“代理”部分，选择“创建代理”。
4. 选择 创建。
5. 您将被带到新创建的代理生成器，您可以在其中配置代理。
6. 在代理详细信息部分，您可以设置以下配置：
  - a. 您可以编辑名称或描述。
  - b. 在“选择模型”中，选择“更改”以选择 FM 供您的代理在编排期间调用。
  - c. 在代理说明中，输入详细信息以告知代理应该做什么以及如何与用户交互。[这些指令取代了编排提示模板中的 \\$instructions\\$ 占位符。](#) 以下是说明示例：

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

- d. 如果展开其他设置，则可以修改以下配置：

用户输入-选择是否允许代理在用户没有足够信息时向其请求更多信息。

- 如果您选择“是”，则如果代理需要在操作组中调用 API，但没有足够的信息来完成 API 请求，则会返回[观察](#)结果，重新提示用户提供更多信息。

- 如果您选择“否”，则代理不会要求用户提供其他详细信息，而是通知用户它没有足够的信息来完成任务。
  - 空闲会话超时-默认情况下，如果用户在与 Amazon Bedrock 代理的会话中有 30 分钟没有响应，则该代理将不再维护对话历史记录。对话历史记录既可用于恢复交互，也可用于根据对话的上下文来增强回应。要更改此默认时间长度，请在“会话超时”字段中输入一个数字，然后选择一个时间单位。
- e. 设置完代理配置后，选择“下一步”。
7. 在操作组部分，您可以向代理添加操作组。有关设置操作组的更多信息，请参阅[the section called “创建操作组”](#)。要添加操作组，请选择“创建”或“创建操作组”。
- a. 在操作组调用部分，您可以设置代理在预测应调用的 API 或函数以及必须传递的参数之后执行的操作。您可以通过定义 Lambda 函数来实现此目的，该函数提供调用操作时执行的业务逻辑。为 Lambda 函数选择以下选项之一：
  - b. 要为您的代理设置另一个操作组，请选择添加其他操作组。添加完操作组后，选择“下一步”。

## API

要创建代理，请使用适用于 [Amazon Bedrock 的代理构建时终端节点发送 CreateAgent 请求](#)（[有关请求和响应格式以及字段详情](#)，请参阅[链接](#)）。

### [参见代码示例](#)

以下列表描述了请求中的字段：

- 至少提供以下必填字段：

| 字段                   | 简短描述         |
|----------------------|--------------|
| agentName            | 代理的名字        |
| agentResourceRoleArn | 代理的服务角色的 ARN |
| 基础模型                 | 代理编排的基础模型    |

- 以下字段为可选字段，但建议填写：

| 字段          | 简短描述                    |
|-------------|-------------------------|
| description | 描述代理的用途                 |
| 指令          | 告诉代理该怎么做的说明。在编排提示模板中使用。 |

- 出于安全考虑，可以指定以下字段：

| 字段                       | 简短描述                  |
|--------------------------|-----------------------|
| clientToken              | 用于确保 API 请求仅完成一次的标识符。 |
| customerEncryptionKeyArn | 用于加密代理资源的 KMS 密钥的 ARN |
| IdleSessionTTL InSeconds | 代理结束会话并删除所有存储信息的持续时间。 |

- 要通过覆盖默认提示模板来自定义代理的行为，请添加一个 `promptOverrideConfiguration` 对象。有关更多信息，请参阅 [Amazon Bedrock 中的高级提示](#)。
- 要将标签附加到您的代理，请使用 `Tags` 字段。有关更多信息，请参阅 [标记资源](#)。

如果您的代理创建失败，则响应中的 `CreateAgent` 对象会返回一个列表 `failureReasons` 和一个列表 `recommendedActions` 供您进行故障排除。

## 为 Amazon Bedrock 代理创建行动组

操作组定义代理可以帮助用户执行的操作。您可以通过在系统中指定可调用的 API、编写 Lambda 函数来处理操作以及如何返回响应来定义操作组。要创建操作组，请准备以下组件：

- 使用操作 [@@ 组的 API 描述、结构和参数设置架 OpenAPI 构](#)。您可以通过以下方式之一将 API 架构添加到操作组：
  - 将您创建的架构上传到亚马逊简单存储服务 (Amazon S3) 存储桶。
  - 添加操作组 AWS Management Console 时，在内联 OpenAPI 架构编辑器中编写架构。只有在操作组所属的代理已经创建之后，此选项才可用。

- [创建一个 Lambda 函数](#)来定义操作组的业务逻辑。

要详细了解操作组的组成部分以及在设置操作组后如何创建操作组，请从以下主题中进行选择：

主题

- [在 Amazon Bedrock 中 OpenAPI 为代理的行动组定义架构](#)
- [在 Amazon Bedrock 中为代理的操作组定义 Lambda 函数](#)
- [在 Amazon Bedrock 中向你的代理添加一个行动组](#)

## 在 Amazon Bedrock 中 OpenAPI 为代理的行动组定义架构

在 Amazon Bedrock 中创建操作组时，必须定义代理可以调用的 API 操作。要定义 API 操作，请创建 JSON 或 YAML 格式的 OpenAPI 架构。您可以创建 OpenAPI 架构文件并将其上传到亚马逊简单存储服务 (Amazon S3) Simple Service。或者，您可以使用控制台中的 OpenAPI 文本编辑器来验证您的架构。创建代理后，您可以使用文本编辑器向代理添加操作组或编辑现有操作组。有关更多信息，请参阅[编辑代理](#)。

有关 API 架构的更多信息，请参阅以下资源：

- 有关 OpenAPI 架构的更多详细信息，请参阅 Swagger 网站上的[OpenAPI 规范](#)。
- 有关编写 API 架构的最佳实践，请参阅 Swagger 网站上的[API 设计最佳实践](#)。

以下是操作组 OpenAPI 架构的一般格式。

```
{
 "openapi": "3.0.0",
 "paths": {
 "/path": {
 "method": {
 "description": "string",
 "operationId": "string",
 "parameters": [...],
 "requestBody": { ... },
 "responses": { ... }
 }
 }
 }
}
```

以下列表描述了OpenAPI架构中的字段

- `openapi`— (必填) 正在使用的版本。OpenAPI此值必须等于或大于 "3.0.0"，操作组才能发挥作用。
- `paths`— (必填) 包含指向各个端点的相对路径。每条路径必须以正斜杠 (/) 开头。
- `method`— (必填) 定义要使用的方法。

最低限度，每种方法都需要以下字段：

- `description`— API 操作的描述。使用此字段通知代理何时调用此 API 操作以及该操作的用途。
- `responses`— 包含代理在 API 响应中返回的属性。代理使用响应属性来构造提示，准确处理 API 调用的结果，并确定执行任务的正确步骤集。代理可以使用来自一个操作的响应值作为编排中后续步骤的输入。

以下两个对象中的字段可为代理提供更多信息，让其有效利用操作组。对于每个字段，`true`如果需要，将该`required`字段的值设置为，如果可选，`false`则设置为。

- `parameters`— 包含可在请求中添加的参数的相关信息。
- `requestBody`— 包含操作请求正文中的字段。不要在 GET 和 DELETE 方法中包含此字段。

要了解有关结构的更多信息，请从以下选项卡中进行选择。

responses

```
"responses": {
 "200": {
 "content": {
 "<media type>": {
 "schema": {
 "properties": {
 "<property>": {
 "type": "string",
 "description": "string"
 },
 ...
 }
 }
 }
 }
 },
}
```



```

 },
 ...
}

```

responses对象中的每个键都是一个响应代码，用于描述响应的状态。响应代码映射到一个包含以下响应信息的对象：

- content – ( 每个响应的必填项 ) 响应内容。
- *<media type>* – 响应正文的格式。有关更多信息，请参阅Swagger网站上的[媒体类型](#)。
- schema – ( 每个媒体类型的必填项 ) 用于定义响应正文及其字段的数据类型。
- properties – ( 如果架构中有 items ，则为必填项 ) 代理使用您在架构中定义的属性来确定需要返回给最终用户的信息，以便完成任务。每个属性都包含以下字段：
  - type – ( 每个属性的必填项 ) 响应字段的数据类型。
  - description – ( 可选 ) 用于描述属性。代理可以使用这些信息来确定需要返回给最终用户的信息。

## parameters

```

"parameters": [
 {
 "name": "string",
 "description": "string",
 "required": boolean,
 "schema": {
 ...
 }
 },
 ...
]

```

您的代理使用以下字段来确定为执行操作组的要求而必须从最终用户那里获得的信息。

- name – ( 必填 ) 参数的名称。
- description – ( 必填 ) 参数的描述。使用此字段可以帮助代理了解如何从用户那里获取该参数，或者确定是否已经从先前的操作或用户对代理的请求中获得该参数值。
- required – ( 可选 ) API 请求是否需要该参数。使用此字段向代理指明每次调用是否都需要此参数，或者该参数是否为可选参数。

- `schema` – ( 可选 ) 输入和输出数据类型的定义。有关更多信息，请参阅Swagger网站上的[数据模型 \( 架构 \)](#)。

## requestBody

以下是`requestBody`字段的一般结构：

```
"requestBody": {
 "required": boolean,
 "content": {
 "<media type>": {
 "schema": {
 "properties": {
 "<property>": {
 "type": "string",
 "description": "string"
 },
 ...
 }
 }
 }
 }
}
```

以下列表描述了每个字段：

- `required`— ( 可选 ) API 请求是否需要请求正文。
- `content` – ( 必填 ) 请求正文的内容。
- `<media type>` – ( 可选 ) 请求正文的格式。有关更多信息，请参阅Swagger网站上的[媒体类型](#)。
- `schema` – ( 可选 ) 用于定义请求正文及其字段的数据类型。
- `properties`— ( 可选 ) 您的代理使用您在架构中定义的属性来确定发出 API 请求时必须从最终用户那里获得的信息。每个属性都包含以下字段：
  - `type` – ( 可选 ) 请求字段的数据类型。
  - `description` – ( 可选 ) 用于描述属性。代理可以使用这些信息来确定需要返回给最终用户的信息。

## 示例 API 架构

以下示例 API 架构定义了一组有助于处理保险索赔的 API 操作。三个 API 的定义如下：

- `getAllOpenClaims`— 如果需要未结索赔列表，您的代理可以使用该 `description` 字段来确定是否应调用此 API 操作。 `responses` 中的 `properties` 用于指定要返回 ID、保单持有人以及理赔状态。代理将这些信息返回给用户，或者使用部分或全部响应作为后续 API 调用的输入。
- `identifyMissingDocuments`— 如果必须识别保险索赔中缺少的文件，您的代理可以使用该 `description` 字段来确定是否应调用此 API 操作。 `name`、`description` 和 `required` 字段用于告诉代理必须从客户那里获取未结理赔的唯一标识符。 `responses` 中的 `properties` 用于指定要返回未结保险理赔的 ID。代理将此信息返回给最终用户，或者使用部分或全部响应作为后续 API 调用的输入。
- `sendReminders`— 如果需要向客户发送提醒，您的代理可以使用该 `description` 字段来确定是否应调用此 API 操作。例如，提醒他们有待处理的未决索赔文件。 `properties` 中的 `requestBody` 告知代理人必须找到索赔编号和待处理的文件。 `properties` 中的 `responses` 指定返回提醒的 ID 及其状态。代理将此信息返回给最终用户，或者使用部分或全部响应作为后续 API 调用的输入。

```
{
 "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"
 }
 }
 }
 },
 "/claims/{claimId}/identify-missing-documents": {
 "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."
 }
 }
 }
 }
 }
 }
 },
 "/send-reminders": {
 "post": {
 "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"
]
}
}
},
"responses": {
 "200": {
 "description": "Reminders sent successfully",
 "content": {
 "application/json": {
 "schema": {
 "type": "object",
 "properties": {
 "sendReminderTrackingId": {
 "type": "string",
 "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."
 }
}
}
}

```

```
 }
 }
}
```

有关 OpenAPI 架构的更多示例，请参阅 GitHub 网站上的 <https://github.com/OAI/OpenAPI-Specification/tree/main/examples/v3.0>。

## 在 Amazon Bedrock 中为代理的操作组定义 Lambda 函数

您必须定义 Lambda 函数来编程操作组的业务逻辑。在 Amazon Bedrock 代理确定需要在操作组中调用的 API 操作后，它会将来自 API 架构的信息以及相关的元数据作为输入事件发送到 Lambda 函数。要编写您的函数，您必须了解 Lambda 函数的以下组成部分：

- 输入事件-包含代理确定必须调用的 API 操作请求正文中的相关元数据和填充字段。
- 响应-包含 API 操作返回的响应正文的相关元数据和填充字段。

您可以编写 Lambda 函数来定义如何处理操作组，并自定义您希望如何返回 API 响应。您可以使用输入事件中的变量来定义函数，并向代理返回响应。

### Note

一个操作组最多可以包含 5 个 API 操作，但您只能编写一个 Lambda 函数。由于 Lambda 函数一次只能接收输入事件并返回一个 API 操作的响应，因此在编写函数时应考虑可能调用的不同 API 操作。

要让您的代理使用 Lambda 函数，您必须将基于资源的策略附加到该函数以为代理提供权限。有关更多信息，请按照中的步骤操作[基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数](#)。有关 Lambda 中基于资源的策略的更多信息，[请参阅开发人员指南中的对 Lambda 使用基于资源的策略](#)。AWS Lambda

### 主题

- [来自 Amazon Bedrock 的 Lambda 输入事件](#)
- [对 Amazon Bedrock 的 Lambda 响应事件](#)
- [操作组 Lambda 函数示例](#)

## 来自 Amazon Bedrock 的 Lambda 输入事件

调用使用 Lambda 函数的操作组时，Amazon Bedrock 会发送以下一般格式的 Lambda 输入事件。您可以将 Lambda 函数定义为使用任何输入事件字段来操纵函数中的业务逻辑以成功执行操作。有关 Lambda 函数的更多信息，请参阅开发人员指南中的[事件驱动调用](#)。AWS Lambda

```
{
 "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"
 },
 ...
]
 }
 }
 },
 "sessionAttributes": {
 "string": "string",
```



```

 },
 "promptSessionAttributes": {
 "string": "string"
 }
 }
}

```

以下列表描述了输入事件字段；

- `messageVersion` – 消息的版本，用于标识进入 Lambda 函数的事件数据格式以及 Lambda 函数的预期响应格式。Amazon Bedrock 仅支持版本 1.0。
- `agent` – 包含操作组所属代理的名称、ID、别名和版本的相关信息。
- `inputText` – 用户在对话回合中的输入。
- `sessionId` – 代理会话的唯一标识符。
- `actionGroup` – 操作组的名称。
- `apiPath` – API 操作的路径，如 OpenAPI 架构中所定义。
- `httpMethod` – API 操作的方法，如 OpenAPI 架构中所定义。
- `parameters` – 包含对象列表。每个对象都包含 API 操作中参数的名称、类型和值，如 OpenAPI 架构中所定义。
- `requestBody` – 包含 OpenAPI 架构中定义请求正文及其属性。
- `sessionAttributes` – 包含[会话属性](#)及其值。这些属性存储在[会话](#)中，为代理提供上下文。
- `promptSessionAttributes` – 包含[提示会话属性](#)及其值。这些属性是按[回合](#)存储的，为代理提供上下文。

## 对 Amazon Bedrock 的 Lambda 响应事件

Amazon Bedrock 希望从 Lambda 函数获得符合以下格式的响应。响应由从 API 操作返回的参数组成。代理可以使用 Lambda 函数的响应进行进一步编排或借助它向客户返回响应。

### Note

[代理配额](#)有关 Lambda 有效负载响应配额，请参阅。

```

{
 "messageVersion": "1.0",

```

```

"response": {
 "actionGroup": "string",
 "apiPath": "string",
 "httpMethod": "string",
 "statusCode": number,
 "responseBody": {
 "<contentType>": {
 "body": "JSON-formatted string"
 }
 }
},
"sessionAttributes": {
 "string": "string",
},
"promptSessionAttributes": {
 "string": "string"
}
}

```

以下列表描述了响应字段：

- `messageVersion` – 消息的版本，用于标识进入 Lambda 函数的事件数据格式以及 Lambda 函数的预期响应格式。Amazon Bedrock 仅支持版本 1.0。
- `response` – 包含有关 API 响应的以下信息。
  - `actionGroup` – 操作组的名称。
  - `apiPath`— API 操作的路径，如 OpenAPI 架构中所定义。
  - `httpMethod`— API 操作的方法，如 OpenAPI 架构中所定义。
  - `responseBody`— 包含 OpenAPI 架构中定义的响应正文。
- ( 可选 ) `sessionAttributes` – 包含会话属性及其值。
- ( 可选 ) `promptSessionAttributes` – 包含提示属性及其值。

## 操作组 Lambda 函数示例

以下是如何在中定义 Lambda 函数的最小示例。Python

```

def lambda_handler(event, context):

 response_body = {
 'application/json': {

```

```
 'body': "sample response"
 }
}

action_response = {
 'actionGroup': event['actionGroup'],
 'apiPath': event['apiPath'],
 'httpMethod': event['httpMethod'],
 'statusCode': 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
```

## 在 Amazon Bedrock 中向你的代理添加一个行动组

为操作组设置 OpenAPI 架构和 Lambda 函数后，您可以创建操作组。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

[创建代理时](#)，可以在工作草案中添加操作组。

创建代理后，您可以通过执行以下步骤向其添加操作组：

向已创建的代理添加操作组

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 从“代理”部分选择代理人，然后在“工作草稿”部分选择工作草稿。
4. 在操作组部分选择添加。

5. 填写操作组详细信息
6. 要使用行内架构编辑器定义操作组的OpenAPI架构，请执行以下操作。有关操作组的 API 架构的更多信息，请参阅[在 Amazon Bedrock 中OpenAPI为代理的行动组定义架构](#)。
  - a. 在“选择 API 架构”下选择“使用内联OpenAPI架构编辑器定义”。此时将出现一个您可以编辑的示例架构。
  - b. 使用格式旁边的下拉菜单选择架构的格式。
  - c. 要从 S3 导入现有架构进行编辑，请选择导入架构，提供 S3 URI，然后选择导入。
  - d. 要将架构恢复为原始示例架构，请选择重置，然后再次选择重置来确认显示的消息。
7. 选择添加。如果没有问题，将会出现绿色成功横幅。如果验证架构时出现问题，将会出现红色横幅。验证过程中发现了以下问题。
  - 滚动浏览架构，查看存在有关格式的错误或警告的行。X 表示存在格式错误，而感叹号表示存在有关格式的警告。
  - 选择红色横幅中的查看详细信息，查看有关 API 架构内容的错误列表。
8. 选择准备以应用对代理所做的更改，然后再测试代理。

## API

要创建操作组，请使用[适用于 Amazon Bedrock 的代理构建](#)时终端节点发送[CreateAgentActionGroup](#)请求。

### [参见代码示例](#)

以下列表描述了请求中的字段：

- 以下字段是必填字段：

| 字段 | 简短描述 |
|----|------|
|    |      |
|    |      |
|    |      |

| 字段 | 简短描述 |
|----|------|
|    |      |

- 以下字段为可选字段：
- 指定用于定义apiSchema对象中操作组要调用的 API 的OpenAPI架构。您可以直接将架构作为JSON 负载提供，也可以提供包含OpenAPI架构的 Amazon S3 存储桶。
- （可选）指定 Lambda 函数的 ARN，以执行调用操作时执行的业务逻辑。
- 将 actionGroupState 设置为 ENABLED 以允许代理调用操作组。
- 要允许您的代理在尝试完成任务时请求用户提供更多信息，请添加一个parentActionGroupSignature字段设置为的操作组AMAZON.UserInput。必须将此操作组的descriptionapiSchema、和actionGroupExecutor字段留空。

在编排过程中，如果您的代理确定需要在操作组中调用 API，但没有足够的信息来完成 API 请求，它将改为调用此操作组并返回 Observation，重新提示用户获取更多信息。

## 将知识库与 Amazon Bedrock 代理关联

如果您尚未创建知识库，请参阅[Amazon Bedrock 知识库](#)以了解知识库并创建知识库。您可以在[创建代理期间或在创建代理](#)之后关联知识库。要将知识库与现有代理关联。请选择与您选择的方法相对应的选项卡，然后按照步骤操作...

### Console

#### 添加知识库

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择代理。然后，在“代理”部分中选择一个代理。
3. 在代理详细信息页面上，从工作草稿部分选择工作草稿。
4. 在“知识库”部分，选择“添加”。
5. 选择您创建的知识库，并提供有关代理应如何与之交互的说明。
6. 选择 添加。成功横幅出现在顶部。
7. 要应用您在测试代理之前对其所做的更改，请选择测试前准备。

## API

要将知识库与代理关联，请向 [Amazon Bedrock 构建时终端节点的代理](#) 发送 [AssociateAgentKnowledgeBase](#) 请求。

以下列表描述了请求中的字段：

- 以下字段是必填字段：

| 字段              | 简短描述    |
|-----------------|---------|
| agentId         | 代理人的 ID |
| agentVersion    | 代理的版本   |
| knowledgeBaseId | 知识库的 ID |

- 以下字段为可选字段：

| 字段                 | 简短描述                    |
|--------------------|-------------------------|
| description        | 描述代理如何使用知识库             |
| knowledgeBaseState | 要防止代理查询知识库，请指定 DISABLED |

## 测试 Amazon Bedrock 代理

创建代理后，您将获得一份工作草稿。工作草稿是代理的一个版本，可用于以迭代方式构建代理。每次您对代理进行更改时，工作草稿都会更新。当您对代理的配置感到满意时，可以创建一个版本（即代理的快照）和指向该版本的别名。然后，您可以通过调用别名将代理部署到您的应用程序中。有关更多信息，请参阅 [部署 Amazon Bedrock 代理](#)。

以下列表描述了如何测试代理：

- 在 Amazon Bedrock 控制台中，您可以打开侧面的测试窗口，然后将输入内容发送给您的代理进行回复。您可以选择工作草稿或已创建的版本。
- 在 API 中，工作草案就是 DRAFT 版本。您可以通过使用 [InvokeAgent](#) 测试别名 TSTALIASID、或指向静态版本的其他别名向代理发送输入。

为了帮助解决您的代理的行为，Amazon Bedrock 代理提供了在与您的代理会话期间查看跟踪的功能。轨迹显示了代理的 step-by-step 推理过程。有关跟踪的更多信息，请参阅[在 Amazon Bedrock 中追踪事件](#)。

以下是测试代理的步骤。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 测试代理

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在“代理”部分，从代理列表中选择要测试的代理的链接。
4. “测试”窗口出现在右侧的窗格中。

#### Note

如果“测试”窗口已关闭，则可以通过选择代理详细信息页面顶部的测试或其中的任何页面来重新打开该窗口。

5. 创建代理后，必须通过以下方式之一进行准备，将其与工作草稿更改一起打包：
  - 在“测试”窗口中，选择“准备”。
  - 在“工作草稿”页面中，选择页面顶部的准备。

#### Note

每次更新工作草稿时，都必须让代理做好准备，以便将最新更改打包给代理。作为最佳实践，我们建议您始终在工作草稿页面的代理概述部分中查看代理的上次准备时间，以确认您正在使用最新的配置测试代理。

6. 要选择要测试的别名和关联版本，请使用测试窗口顶部的下拉菜单。默认情况下，选择 TestAlias：工作草稿组合。
7. 要测试代理，请输入一条消息并选择运行。在等待生成响应时或生成响应之后，您可以选择以下选项：
  - 要查看代理协调流程中每个步骤的详细信息，包括每个步骤的提示、推理配置和代理的推理过程以及其操作组和知识库的使用情况，请选择显示跟踪。跟踪会实时更新，因此您可以在

返回响应之前进行查看。要展开或折叠步骤的轨迹，请选择步骤旁边的箭头。有关“跟踪”窗口和显示的详细信息的更多信息，请参阅[在 Amazon Bedrock 中追踪事件](#)。

- 如果代理调用知识库，则响应中包含脚注。要查看包含响应特定部分引用信息的 S3 对象的链接，请选择相关的脚注。

您可以在“测试”窗口中执行以下操作：

- 要与代理开始新的对话，请选择刷新图标。
- 要查看“跟踪”窗口，请选择展开图标。要关闭“跟踪”窗口，请选择缩小图标。
- 要关闭“测试”窗口，请选择向右箭头图标。

您可以启用或禁用操作组和知识库。使用此功能通过评估代理在不同设置下的行为来确定哪些操作组或知识库需要更新，从而对代理进行故障排除。

### 启用操作组或知识库

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在“代理”部分，从代理列表中选择要测试的代理的链接。
4. 在代理的详细信息页面上，在工作草稿部分，选择工作草稿的链接。
5. 在“操作组”或“知识库”部分，将鼠标悬停在要更改其状态的操作组或知识库的状态上。
6. 此时将出现一个编辑按钮。选择编辑图标，然后从下拉菜单中选择操作组或知识库是启用还是禁用。
7. 如果操作组处于禁用状态，则代理将不使用该操作组。如果知识库已禁用，则代理将不使用知识库。启用或禁用操作组或知识库，然后使用“测试”窗口对代理进行故障排除。
8. 选择“准备”，应用您在测试代理之前对代理所做的更改。

## API

在首次测试代理之前，必须使用适用于 [Amazon Bedrock 的代理构建](#) 时终端节点发送 [PrepareAgent](#) 请求（请求和响应格式以及字段详情参见链接），将其与工作草稿更改一起打包。在请求 agentId 中包括。更改适用于 TSTALIASID 别名指向的 DRAFT 版本。

### [参见代码示例](#)



**Note**

每次更新工作草稿时，都必须让代理做好准备，以便将最新更改打包给代理。作为最佳实践，我们建议您使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时终端节点发送GetAgent请求](#)（[有关](#)请求和响应格式以及字段详情，请参阅链接），并查看代理验证您是否正在使用最新配置测试代理的preparedAt时间。

要测试您的代理，请使用适用于 Amazon Bedrock 的 [代理运行时终端节点发送InvokeAgent请求](#)（[有关](#)请求和响应格式以及字段详情，请参阅链接）。

[参见代码示例](#)**Note**

AWS CLI 不支持 [InvokeAgent](#)。

[参见代码示例](#)

请求中存在以下字段：

- 至少提供以下必填字段：

| 字段           | 简短描述                          |
|--------------|-------------------------------|
| agentId      | 代理人的 ID                       |
| agentAliasId | 别名的 ID。TSTALIASID 用于调用DRAFT版本 |
| sessionId    | 会话的字母数字 ID（2—100 个字符）         |
| 输入文本         | 用户向代理发送的提示                    |

- 以下字段为可选字段：

| 字段           | 简短描述                                                                                          |
|--------------|-----------------------------------------------------------------------------------------------|
| 启用跟踪         | 指定TRUE以查看 <a href="#">跟踪</a> 。                                                                |
| 结束会话         | 指定TRUE在此请求之后结束与代理的会话。                                                                         |
| sessionState | 包括在sessionAttributes 整个会话中保留的和/或promptSessionAttributes 在整个InvokeAgent回合中保留的内容。这些属性为代理提供了上下文。 |

响应以chunk对象中的字节为单位返回。如果代理查询知识库，则chunk包含citations。如果您启用了跟踪，则还会返回一个trace对象。如果发生错误，则会返回包含错误消息的字段。有关如何读取跟踪的更多信息，请参阅[在 Amazon Bedrock 中追踪事件](#)。

## 在 Amazon Bedrock 中追踪事件

Amazon Bedrock 代理的每一次回复都附有一条跟踪记录，详细说明了该代理正在策划的步骤。跟踪记录可以帮助您跟踪代理的推理过程，该过程可引导它做出它当时在对话中做出的响应。

使用跟踪记录来跟踪代理从用户输入到它返回的响应的路径。跟踪提供有关代理调用的操作组的输入信息，以及代理为响应用户而查询的知识库的信息。此外，跟踪还提供有关行动组和知识库返回的产出的信息。您可以查看代理用来确定其采取的操作或对知识库的查询的推理。如果跟踪中的某个步骤失败，则跟踪功能会返回失败的原因。使用跟踪中的详细信息对您的代理进行故障排除。您可以确定代理在哪些步骤中遇到问题或在哪些步骤中会产生意外行为。然后，您可以使用这些信息来考虑如何改善代理的行为。

### 查看踪迹

以下介绍如何查看跟踪。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

#### Console

在与代理对话期间查看跟踪

登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console

1. 在“代理”部分，从代理列表中选择要测试的代理的链接。
2. “测试”窗口出现在右侧的窗格中。
3. 输入一条消息，然后选择 Run。在生成响应时或响应完成生成后，选择显示跟踪。
4. 当您的代理执行编排时，您可以实时查看每个步骤的跟踪。

## API

要查看跟踪，请使用[适用于 Amazon Bedrock 的代理运行时终端节点](#)发送 `InvokeAgent` 请求，并将 `enableTrace` 字段设置为 `TRUE`。默认情况下禁用跟踪。

如果您启用跟踪，则在 `InvokeAgent` 响应中，流 `chunk` 中的每一个都附带一个映射到 `TracePart` 对象的 `trace` 字段。里面 `TracePart` 有一个映射到 `Trace` 对象的 `trace` 字段。

## 轨迹的结构

该跟踪在控制台和 API 中均以 JSON 对象的形式显示。控制台或 API `Trace` 中的每个步骤都可以是以下轨迹之一：

- [PreProcessingTrace](#)— 跟踪预处理步骤的输入和输出，在该步骤中，代理对用户输入进行情境化和分类，并确定其是否有效。
- [编排](#)— 跟踪编排步骤的输入和输出，在该步骤中，代理解释输入并调用 API 操作和查询知识库。然后，代理返回输出以继续编排或响应用户。
- [PostProcessingTrace](#)— 跟踪后处理步骤的输入和输出，在该步骤中，代理处理编排的最终输出并确定如何将响应返回给用户。
- [FailureTrace](#)— 追踪步骤失败的原因。

每条轨迹（除外 `FailureTrace`）都包含一个 `ModelInvocationInput` 对象。该 `ModelInvocationInput` 对象包含在该步骤的提示模板中设置的配置，以及在此步骤中向代理提供的提示。有关如何修改提示模板的更多信息，请参阅[Amazon Bedrock 中的高级提示](#)。该 `ModelInvocationInput` 对象的结构如下所示：

```
{
 "traceId": "string",
 "text": "string",
 "type": "PRE_PROCESSING | ORCHESTRATION | KNOWLEDGE_BASE_RESPONSE_GENERATION | POST_PROCESSING",
```

```

 "inferenceConfiguration": {
 "maxLength": number,
 "stopSequences": ["string"],
 "temperature": float,
 "topK": float,
 "topP": float
 },
 "promptCreationMode": "DEFAULT | OVERRIDDEN",
 "parserMode": "DEFAULT | OVERRIDDEN",
 "overrideLambda": "string"
 }
}

```

以下列表描述了 [ModelInvocationInput](#) 对象的字段：

- `traceId` – 跟踪的唯一标识符。
- `text` – 在相应步骤中向代理提供的提示文本。
- `type` – 代理过程的当前步骤。
- `inferenceConfiguration` – 影响响应生成的推理参数。有关更多信息，请参阅 [推理参数](#)。
- `promptCreationMode`— 此步骤中代理的默认基本提示模板是否被覆盖。有关更多信息，请参阅 [Amazon Bedrock 中的高级提示](#)。
- `parserMode`— 此步骤中代理的默认响应解析器是否被覆盖。有关更多信息，请参阅 [Amazon Bedrock 中的高级提示](#)。
- `overrideLambda`— 如果默认解析器被覆盖，则用于解析响应的解析器 Lambda 函数的亚马逊资源名称 (ARN)。有关更多信息，请参阅 [Amazon Bedrock 中的高级提示](#)。

有关每种跟踪类型的更多信息，请参阅以下各节：

### PreProcessingTrace

```

{
 "modelInvocationInput": { // see above for details }
 "modelInvocationOutput": {
 "parsedResponse": {
 "isValid": boolean,
 "rationale": "string"
 },
 "traceId": "string"
 }
}

```

[PreProcessingTrace](#) 由一个 [ModelInvocationInput](#) 对象和一个 [PreProcessingModelInvocationOutput](#) 对象组成。[PreProcessingModelInvocationOutput](#) 包含以下字段。

- `parsedResponse` – 包含已解析的用户提示的以下相关详细信息。
  - `isValid`— 指定用户提示是否有效。
  - `rationale` – 用于指定代理对后续行动的推理。
- `traceId` – 跟踪的唯一标识符。

## OrchestrationTrace

[编排](#) 由 [ModelInvocationInput](#) 对象以及“[原理](#)”和“[观察](#)”对象的任意组合组成。[InvocationInput](#) 有关每个对象的更多信息，请从以下选项卡中进行选择：

```
{
 "modelInvocationInput": { // see above for details },
 "rationale": { ... },
 "invocationInput": { ... },
 "observation": { ... }
}
```

## Rationale

Rational 对象包含代理给定用户输入的[推理](#)。以下是结构：

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

以下列表描述了 Rational 对象的[字段](#)：

- `traceId` – 跟踪步骤的唯一标识符。
- `text`— 基于输入提示的代理推理过程。

## InvocationInput

[InvocationInput](#) 对象包含将输入到要调用或查询的操作组或知识库中的信息。以下是结构：

```
{
```

```

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

```

以下列表描述了 [InvocationInput](#) 对象的字段：

- `traceId` – 跟踪的唯一标识符。
- `invocationType`— 指定代理是在调用操作组或知识库，还是正在结束会话。
- `actionGroupInvocationInput` – 如果 `type` 为 `ACTION_GROUP`，则会显示此字段。有关更多信息，请参阅在 [Amazon Bedrock 中 OpenAPI 为代理的行动组定义架构](#)。包含被调用的操作组的以下输入：
  - `actionGroupName`— 代理将调用的操作组的名称。
  - `apiPath`— 根据 API 架构，要调用的 API 操作的路径。

- `verb`— 根据 API 架构，正在使用的 API 方法。
- `parameters` – 包含对象列表。每个对象都包含 API 操作中参数的名称、类型和值，如 API 架构中所定义。
- `requestBody`— 包含请求正文及其属性，如 API 架构中所定义。
- `knowledgeBaseLookupInput` – 如果 `type` 为 `KNOWLEDGE_BASE`，则会显示此字段。有关更多信息，请参阅[Amazon Bedrock 知识库](#)。包含有关知识库和知识库搜索查询的以下信息：
  - `knowledgeBaseId` – 代理将查询的知识库的唯一标识符。
  - `text` – 对知识库的查询。

## Observation

`Observation` 对象包含操作组或知识库的结果或输出，或者对用户的响应。以下是结构：

```
{
 "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"
 }
}

```

以下列表描述了[观测](#)对象的字段：

- `traceId` – 跟踪的唯一标识符。
- `type`— 指定代理的观察结果是从操作组还是知识库的结果中返回，以及代理是否在重新提示用户、请求更多信息或结束对话。
- `actionGroupInvocationOutput`— 包含操作组调用的 API 操作返回的 JSON 格式的字符串。如果 `type` 为 `ACTION_GROUP`，则显示此字段。有关更多信息，请参阅[在 Amazon Bedrock 中 OpenAPI 为代理的行动组定义架构](#)。
- `knowledgeBaseLookupOutput`— 包含从知识库中检索到的与响应提示相关的文本，以及数据源的 Amazon S3 位置。如果 `type` 为 `KNOWLEDGE_BASE`，则显示此字段。有关更多信息，请参阅[Amazon Bedrock 知识库](#)。列表中的每个对象都 `retrievedReferences` 包含以下字段：
  - `content` – 包含从知识库查询返回的知识库 `text`。
  - `location`— 包含从中找到返回文本的数据源的 Amazon S3 URI。
- `repromptResponse` – 如果 `type` 为 `REPROMPT`，则会显示此字段。包含要求再次提示的 `text`，以及代理需要重新提示的原因 `source`。
- `finalResponse` – 如果 `type` 为 `ASK_USER` 或 `FINISH`，则会显示此字段。包含要求用户提供更多信息或回应用户的 `text`。

## PostProcessingTrace

```

{
 "modelInvocationInput": { // see above for details }
 "modelInvocationOutput": {
 "parsedResponse": {
 "text": "string"
 },
 "traceId": "string"
 }
}

```

[PostProcessingTrace](#) 由一个 [ModelInvocationInput](#) 对象和一个 [PostProcessingModelInvocationOutput](#) 对象组成。[PostProcessingModelInvocationOutput](#) 包含以下字段：



- `parsedResponse`— 包含`text`要在解析器函数处理文本后返回给用户的。
- `traceId` – 跟踪的唯一标识符。

## FailureTrace

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

以下列表描述了[FailureTrace](#)对象的字段：

- `failureReason` – 步骤失败的原因。
- `traceId` – 跟踪的唯一标识符。

## 管理 Amazon Bedrock 代理

创建代理后，您可以根据需要查看或更新其配置。配置适用于工作草稿。如果您不再需要代理，可以将其删除。

### 主题

- [查看有关代理的信息](#)
- [编辑代理](#)
- [删除代理](#)
- [管理代理的操作组](#)
- [管理代理与知识库的关联](#)

## 查看有关代理的信息

要了解如何查看有关代理的信息，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 查看有关代理的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console

2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理详细信息上，您可以查看以下信息：
  - 代理概述部分包含代理配置。
  - 标签部分包含与代理关联的标签。有关更多信息，请参阅[标记资源](#)。
  - 工作草案部分载有工作草案。如果您选择工作草稿，则可以查看以下信息：
    - 模型详细信息部分包含代理工作草案使用的模型和说明。
    - “操作组”部分包含代理使用的操作组。有关更多信息，请参阅[为 Amazon Bedrock 代理创建行动组](#)和[管理代理的操作组](#)。
    - 知识库部分包含与代理相关的知识库。有关更多信息，请参阅[将知识库与 Amazon Bedrock 代理关联](#)和[管理代理与知识库的关联](#)。
    - 高级提示部分包含代理编排的每个步骤的提示模板。有关更多信息，请参阅[Amazon Bedrock 中的高级提示](#)。
  - 版本和别名部分包含可用于部署到应用程序的代理的版本和别名。有关更多信息，请参阅[部署 Amazon Bedrock 代理](#)。

## API

要获取有关代理的信息，请使用适用于 [Amazon Bedrock 的代理构建时终端节点](#) 发送 [GetAgent](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)），并指定 `agentId` [参见代码示例](#)。

要列出有关您的代理的信息，请使用适用于 [Amazon Bedrock 的代理构建时终端节点](#) 发送 [ListAgents](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。[参见代码示例](#)。您可以指定以下可选参数：

| 字段                      | 简短描述                                                                                                                       |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>maxResults</code> | 要在响应中返回的结果数量上限。                                                                                                            |
| <code>nextToken</code>  | 如果结果多于您在 <code>maxResults</code> 字段中指定的数字，则响应会返回一个 <code>nextToken</code> 值。要查看下一批结果，请在另一个请求中发送该 <code>nextToken</code> 值。 |

要列出代理的所有标签，请使用 [Amazon Bedrock 代理构建时终端节点发送ListTagsForResource请求](#)（请求和响应格式以及字段详情参见链接），并附上代理的亚马逊资源名称 (ARN)。

## 编辑代理

要了解如何编辑代理，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 编辑代理配置

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理概述部分中，选择编辑。
4. 根据需要编辑字段中的现有信息。
5. 编辑完信息后，选择“保存”以保持在同一窗口中，或者选择“保存并退出”返回代理详细信息页面。成功横幅出现在顶部。要将新配置应用于您的代理，请在标题中选择“准备”。

您可能需要为代理尝试不同的根基模型或更改对代理的指令。这些更改仅适用于工作草稿。

更改您的代理使用的根基模型或对代理的指令。

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在模型详细信息部分，选择编辑
6. 根据需要选择其他型号或编辑给代理的指令。

#### Note

如果您更改基础模型，则您修改的所有[提示模板](#)都将设置为该模型的默认值。

7. 编辑完信息后，选择“保存”以保持在同一窗口中，或者选择“保存并退出”返回代理详细信息页面。成功横幅出现在顶部。
8. 要应用您在测试代理之前对其所做的更改，请在“测试”窗口或“工作草稿”页面顶部选择“准备”。

### 编辑与代理关联的标签

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在标签部分中，选择管理标签。
5. 要添加标签，请选择添加新标签。然后输入密钥并根据需要输入值。要删除标签，请选择移除。有关更多信息，请参阅[标记资源](#)。
6. 编辑完标签后，选择提交。

## API

要编辑代理，请使用适用于 [Amazon Bedrock 的代理构建](#) 时终端节点发送 [UpdateAgent](#) 请求（请求和响应格式以及字段详情参见链接）。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。有关必填字段和可选字段的更多信息，请参阅 [在 Amazon Bedrock 中创建代理](#)。

要将更改应用于工作草稿，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时端点发送 PrepareAgent 请求](#)（[有关](#) 请求和响应格式以及字段详情，请参阅链接）。在请求 agentId 中包含。更改适用于 TSTALIASID 别名指向的 DRAFT 版本。

要向代理添加标签，请使用 [Amazon Bedrock 代理构建时终端节点发送 TagResource 请求](#)（[请求和响应格式以及字段详情参见链接](#)），并附上代理的亚马逊资源名称 (ARN)。请求正文包含一个 tags 字段，该字段是一个包含您为每个标签指定的键值对的对象。

要从代理中移除标签，请使用适用于 [Amazon Bedrock 的代理构建时终端节点发送 UntagResource 请求](#)（[请求和响应格式以及字段详情参见链接](#)），并附上代理的亚马逊资源名称 (ARN)。tagKeys 请求参数是一个列表，其中包含您要删除的标签的密钥。

## 删除代理

要了解如何删除代理，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 要删除代理

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择“代理”。
3. 要删除代理，请选择要删除的代理旁边的选项按钮。
4. 将出现一个对话框，警告您注意删除的后果。要确认要删除代理，请在输入字段 `delete` 中输入，然后选择删除。
5. 删除完成后，将显示成功横幅。

### API

要删除代理，请使用适用于 [Amazon Bedrock 的代理构建时终端节点发送 DeleteAgent 请求](#)（[有关请求和响应格式以及字段详情，请参阅链接](#)），并指定 `agentId`

默认情况下，`skipResourceInUseCheck` 参数为 `false`，如果资源正在使用中，则会停止删除。如果设置 `skipResourceInUseCheck` 为 `true`，则即使资源正在使用中，也会删除该资源。

#### [参见代码示例](#)

选择一个主题以了解如何管理代理的操作组或知识库。

#### 主题

- [管理代理的操作组](#)
- [管理代理与知识库的关联](#)

## 管理代理的操作组

创建操作组后，您可以查看、编辑或删除它。这些更改适用于代理的工作草稿版本。

#### 主题

- [查看有关操作组的信息](#)
- [编辑操作组](#)
- [删除操作组](#)

## 查看有关操作组的信息

要了解如何查看有关操作组的信息，请选择与您选择的方法相对应的选项卡，然后按照步骤进行操作。

### Console

#### 查看有关操作组的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在“操作组”部分，选择要查看其信息的操作组。

### API

要获取有关操作组的信息，请使用 [Amazon Bedrock 代理构建时终端节点发送GetAgentActionGroup请求](#)（[有关请求和响应格式以及字段详情，请参阅链接](#)），并指定actionGroupId、agentId和agentVersion

要列出有关代理操作组的信息，请使用 [Amazon Bedrock 代理构建时终端节点发送ListAgentActionGroups请求](#)（[有关请求和响应格式以及字段详情，请参阅链接](#)）。指定要查看其操作组的agentId和agentVersion。您可以包括以下参数：

| 字段         | 简短描述                                                                             |
|------------|----------------------------------------------------------------------------------|
| maxResults | 要在响应中返回的结果数量上限。                                                                  |
| nextToken  | 如果结果多于您在maxResults 字段中指定的数字，则响应会返回一个nextToken 值。要查看下一批结果，请在另一个请求中发送该nextToken 值。 |

## [参见代码示例](#)

### 编辑操作组

要了解如何编辑操作组，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

#### Console

##### 编辑操作组

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在“操作组”部分，选择要编辑的操作组。然后选择编辑。
6. 根据需要编辑现有字段。有关更多信息，请参阅[Amazon Bedrock 代理创建行动组](#)。
7. 要使用行内架构编辑器为操作组定义架构，请在“选择 API OpenAPI 架构”中，选择“使用内联 OpenAPI 架构编辑器定义”。此时将出现一个您可以编辑的示例架构。您可以配置以下选项：
  - 要从 Amazon S3 导入现有架构进行编辑，请选择导入架构，提供 Amazon S3 URI，然后选择导入。
  - 要将架构恢复到原始示例架构，请选择“重置”，然后选择“确认”以确认显示的消息。
  - 要为架构选择其他格式，请使用标有 JSON 的下拉菜单。
  - 要更改架构的视觉外观，请选择架构下方的齿轮图标。
8. 要控制代理能否使用操作组，请选择“启用”或“禁用”。使用此功能来帮助排除代理的行为。
9. 要保持在同一窗口中以便您可以测试更改，请选择“保存”。要返回到操作组详细信息页面，请选择保存并退出。
10. 如果没有问题，则会显示成功横幅。如果验证架构时出现问题，则会显示错误标语。要查看错误列表，请选择在横幅中显示详细信息。
11. 要应用您在测试代理之前对其所做的更改，请在“测试”窗口或“工作草稿”页面顶部选择“准备”。

## API

要编辑操作组，请使用适用于 [Amazon Bedrock 的代理构建](#) 时终端节点发送 [UpdateAgentActionGroup](#) 请求（请求和响应格式以及字段详情参见链接）。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。必须将 `agentVersion` 指定为 `DRAFT`。有关必填字段和可选字段的更多信息，请参阅 [Amazon Bedrock 代理创建行动组](#)。

要将更改应用于工作草稿，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时端点发送 PrepareAgent](#) 请求（[有关](#) 请求和响应格式以及字段详情，请参阅链接）。在请求 `agentId` 中包含。更改适用于 `TSTALIASID` 别名指向的 `DRAFT` 版本。

## 删除操作组

要了解如何删除操作组，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 删除操作组

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在操作组部分，选择要删除的操作组旁边的选项按钮。
6. 将出现一个对话框，警告您注意删除的后果。要确认要删除操作组，请在输入字段 `delete` 中输入，然后选择删除。
7. 删除完成后，将显示成功横幅。
8. 要应用您在测试代理之前对其所做的更改，请在“测试”窗口或“工作草稿”页面顶部选择“准备”。

## API

要删除操作组，[DeleteAgentActionGroup](#) 请发送请求。指定要 `actionGroupId` 和 `agentVersion` 从中删除它的 `agentId`。默认情况下，`skipResourceInUseCheck` 参数为 `false`，如果资源正在使用中，则会停止删除。如果设置 `skipResourceInUseCheck` 为 `true`，则即使资源正在使用中，也会删除该资源。



要将更改应用于工作草稿，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时端点发送PrepareAgent请求](#)（有关请求和响应格式以及字段详情，请参阅链接）。在请求agentId中包含。更改适用于TSTALIASID别名指向的DRAFT版本。

## 管理代理与知识库的关联

创建代理后，您可以添加更多知识库或对其进行编辑。添加和编辑操作在工作草稿中执行。要执行这些操作，请从代理部分选择代理，然后在工作草稿部分选择工作草稿。

### 主题

- [查看有关代理与知识库关联的信息](#)
- [编辑代理与知识库的关联](#)
- [取消知识库与代理的关联](#)

## 查看有关代理与知识库关联的信息

要了解如何查看知识库信息，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

查看与代理关联的知识库的相关信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在知识库部分，选择要查看其信息的知识库。

### API

要获取有关与代理关联的知识库的信息，请使用 [Amazon Bedrock 代理构建时终端节点发送GetAgentKnowledgeBase请求](#)（有关请求和响应格式以及字段详情，请参阅链接）。指定以下字段：

要列出与代理相关的知识库信息，请使用 A [mazon Bedrock 代理构建](#) 时终端节点发送 [ListAgentKnowledgeBases](#) 请求（请求和响应格式以及字段详情参见链接）。指定要查看相关知识库的 `agentId` 和 `agentVersion`。

| 字段                      | 简短描述                                                                                                                       |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>maxResults</code> | 要在响应中返回的结果数量上限。                                                                                                            |
| <code>nextToken</code>  | 如果结果多于您在 <code>maxResults</code> 字段中指定的数字，则响应会返回一个 <code>nextToken</code> 值。要查看下一批结果，请在另一个请求中发送该 <code>nextToken</code> 值。 |

### [参见代码示例](#)

## 编辑代理与知识库的关联

要了解如何编辑代理与知识库的关联，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 编辑代理与知识库的关联

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在“操作组”部分，选择要编辑的操作组。然后选择编辑。
6. 根据需要编辑现有字段。有关更多信息，请参阅[将知识库与 Amazon Bedrock 代理关联](#)。
7. 要控制代理能否使用知识库，请选择“启用”或“禁用”。使用此功能来帮助排除代理的行为。
8. 要保持在同一窗口中以便您可以测试更改，请选择“保存”。要返回工作草稿页面，请选择保存并退出。
9. 要应用您在测试代理之前对其所做的更改，请在“测试”窗口或“工作草稿”页面顶部选择“准备”。

## API

要编辑与代理关联的知识库的配置，请使用适用于 [Amazon Bedrock 的代理构建](#) 时端点发送 [UpdateAgentKnowledgeBase](#) 请求（请求和响应格式以及字段详情参见链接）。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。必须将 `agentVersion` 指定为 `DRAFT`。有关必填字段和可选字段的更多信息，请参阅 [将知识库与 Amazon Bedrock 代理关联](#)。

要将更改应用于工作草稿，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时端点发送 PrepareAgent](#) 请求（[有关](#) 请求和响应格式以及字段详情，请参阅链接）。在请求 `agentId` 中包含更改适用于 `TSTALIASID` 别名指向的 `DRAFT` 版本。

## 取消知识库与代理的关联

要了解如何取消知识库与代理的关联，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 解除知识库与代理的关联

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在代理部分选择一个代理。
4. 在代理详细信息页面上，在“工作草稿”部分，选择工作草稿。
5. 在知识库部分，选择要删除的知识库旁边的选项按钮。然后选择删除。
6. 确认出现的消息，然后选择删除。
7. 要应用您在测试代理之前对其所做的更改，请在“测试”窗口或“工作草稿”页面顶部选择“准备”。

## API

要取消知识库与代理的关联，请使用适用于 [Amazon Bedrock 的代理构建](#) 时终端节点发送 [DisassociateAgentKnowledgeBase](#) 请求（请求和响应格式以及字段详情参见链接）。指定要 `knowledgeBaseId` 与其取消关联 `agentVersion` 的代理的 `and` 和。

要将更改应用于工作草稿，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时端点发送 PrepareAgent](#) 请求（[有关](#) 请求和响应格式以及字段详情，请参阅链接）。在请求 `agentId` 中包含更改适用于 `TSTALIASID` 别名指向的 `DRAFT` 版本。

# 自定义 Amazon Bedrock 代理

设置代理后，您可以使用以下功能进一步自定义其行为：

- 高级提示允许您修改提示模板，以确定在运行的每个步骤中发送给代理的提示。
- 会话状态是一个包含属性的字段，您可以在构建时在发送请求时定义这些属性，也可以在运行时使用 [CreateAgent](#) 请求发送这些属性。 [InvokeAgent](#) 您可以使用这些属性在用户和代理之间的对话中提供和管理上下文。

选择一个主题以了解有关该功能的更多信息。

## 主题

- [Amazon Bedrock 中的高级提示](#)
- [控制会话上下文](#)

## Amazon Bedrock 中的高级提示

创建后，将使用以下四个默认基本提示模板配置代理，这些模板概述了代理序列的每一步如何构造要发送到基础模型的提示。有关每个步骤所包含内容的详细信息，请参阅 [运行时进程](#)。

- 预处理
- 编排
- 知识库响应生成
- 后期处理 ( 默认禁用 )

提示模板定义了代理如何执行以下操作：

- 处理基础模型 (FM) 中的用户输入文本和输出提示
- 在 FM、行动小组和知识库之间进行协调
- 格式化回复并将其返回给用户

通过使用高级提示，您可以修改这些提示模板以提供详细的配置，从而提高代理的准确性。您还可以提供精心策划的少镜头提示示例，在这些示例中，您可以通过为特定任务提供带标签的示例来提高模型性能。

选择一个主题以了解有关高级提示的更多信息。

## 主题

- [高级提示术语](#)
- [配置提示模板](#)
- [Amazon Bedrock 代理提示模板中的占位符变量](#)
- [Amazon Bedrock 代理中的解析器 Lambda 函数](#)

## 高级提示术语

以下术语有助于理解高级提示的工作原理。

- 会话-向具有相同会话 ID 的同一个代理发出的一组 [InvokeAgent](#) 请求。发出 InvokeAgent 请求时，可以重复使用上一次调用的响应返回的 sessionId，以便继续与代理进行同一会话。只要 [代理](#) 配置中的 idleSessionTTLInSeconds 时间未到期，您就可以与代理保持相同的会话。
- 回合 – 一个 InvokeAgent 调用。一个会话由一个或多个回合组成。
- 迭代-以下操作的序列：
  1. ( 必要 ) 对根基模型的调用
  2. ( 可选 ) 操作组调用
  3. ( 可选 ) 知识库调用
  4. ( 可选 ) 对用户要求提供更多信息的回应

可能会跳过某项操作，具体取决于代理的配置或代理当时的要求。一个回合包含一次或多次迭代。

- 提示 – 提示包含对代理的指令、上下文和文本输入。文本输入可以来自用户，也可以来自代理序列中另一个步骤的输出。向基础模型提供提示，以确定代理在响应用户输入时要采取的下一步行动
- 基本提示模板 – 构成提示的结构元素。该模板由占位符组成，这些占位符在运行时填充了用户输入、代理配置和上下文，用于为基础模型创建当代理到达该步骤时要处理的提示。有关这些占位符的更多信息，请参阅 [Amazon Bedrock 代理提示模板中的占位符变量](#) )。使用高级提示，您可以编辑这些模板。

## 配置提示模板

使用高级提示，您可以执行以下操作：

- 打开或关闭代理序列中不同步骤的调用。

- 配置他们的推理参数。
- 编辑代理使用的默认基本提示模板。通过使用自己的配置覆盖逻辑，您可以自定义代理的行为。

对于代理序列的每个步骤，您可以编辑以下部分：

- 提示模板-描述代理应如何评估和使用在编辑模板的步骤中收到的提示。编辑模板时，您可以使用以下工具设计提示：
  - 提示模板占位符 — Amazon Bedrock 代理中的预定义变量，这些变量在代理调用期间在运行时动态填充。在提示模板中，您会看到这些占位符周围环绕着\$ (例如，\$instructions\$)。有关可在模板中使用的占位符变量的信息，请参阅[Amazon Bedrock 代理提示模板中的占位符变量](#)。
  - XML 标签 — Anthropic 模型支持使用 XML 标签来构造和描述您的提示。使用描述性标签名称以获得最佳结果。例如，在默认的编排提示模板中，您将看到用于描述几个镜头示例的<examples>标签)。有关更多信息，请参阅[Anthropic用户指南](#)中的[使用 XML 标签](#)。

您可以启用或禁用代理序列中的任何步骤。下表显示了每个步骤的默认状态。

| 提示模板    | 默认设置 |
|---------|------|
| 预处理     | 已启用  |
| 编排      | 已启用  |
| 知识库响应生成 | 已启用  |
| 后处理     | 已禁用  |

#### Note

如果您禁用编排步骤，则代理会将原始用户输入发送到基础模型，并且不会使用基本提示模板进行编排。

如果您禁用任何其他步骤，代理将完全跳过相应步骤。

- 推理配置-影响您使用的模型生成的响应。有关推理参数的定义，以及不同模型所支持参数的更多详细信息，请参阅[根基模型的推理参数](#)。
- (可选) 解析器 Lambda 函数 – 定义如何解析原始根基模型输出以及如何在运行时流程中使用它。此函数作用于已启用该函数的步骤的输出，并按照函数中的定义返回解析后的响应。

根据您的自定义基本提示模板的方式，原始基础模型输出可能特定于该模板。因此，代理的默认解析器可能难以正确解析输出。通过编写自定义解析器 Lambda 函数，您可以帮助代理根据您的用例解析原始基础模型输出。有关解析器 Lambda 函数及其编写方法的更多信息，请参阅 [Amazon Bedrock 代理中的解析器 Lambda 函数](#)

### Note

您可以为所有基本模板定义一个解析器 Lambda 函数，但可以在每个步骤中配置是否调用该函数。请务必为您的 Lambda 函数配置基于资源的策略，以便您的代理可以调用该函数。有关更多信息，请参阅 [基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数](#)。

编辑提示模板后，您可以测试您的代理。要分析代理的 step-by-step 过程并确定其是否按预期运行，请打开跟踪并进行检查。有关更多信息，请参阅 [在 Amazon Bedrock 中追踪事件](#)。

您可以在 AWS Management Console 或通过 API 配置高级提示。

## Console

在控制台中，您可以在创建代理后配置高级提示。您可以在编辑代理时进行配置。


### 查看或编辑代理的高级提示

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 在左侧导航窗格中，选择代理。然后在“代理”部分中选择一个代理。
3. 在代理详细信息页面上，在工作草稿部分，选择工作草稿。
4. 在工作草稿页面的高级提示部分，选择编辑。
5. 在“编辑高级提示”页面上，选择与要编辑的代理序列步骤相对应的选项卡。
6. 要启用模板编辑功能，请打开“覆盖模板默认值”。在“覆盖模板默认值”对话框中，选择“确认”。

### Warning


如果您关闭覆盖模板默认值或更改模型，则会使用默认的 Amazon Bedrock 模板，并且您的模板将立即被删除。要进行确认，请在文本框中输入 **confirm** 以确认出现的消息。

7. 要允许代理在生成响应时使用模板，请打开激活模板。如果此配置被关闭，代理将不使用该模板。
8. 要修改示例提示模板，请使用提示模板编辑器。
9. 在配置中，您可以修改提示的推理参数。有关参数的定义，以及不同模型所支持参数的更多信息，请参阅[根基模型的推理参数](#)。
10. ( 可选 ) 要使用您定义的 Lambda 函数来解析原始基础模型输出，请执行以下操作：

 Note

所有提示模板均使用一个 Lambda 函数。

- a. 在配置部分，选择使用 Lambda 函数进行解析。如果清除此设置，您的代理将使用默认解析器作为提示。
- b. 对于解析器 Lambda 函数，请从下拉菜单中选择一个 Lambda 函数。

 Note

您必须为代理附加权限，这样它才能访问 Lambda 函数。有关更多信息，请参阅[基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数](#)。

11. 要保存您的设置，请选择以下选项之一：
  - a. 要保持在同一窗口中，以便在测试更新的代理时动态更新提示设置，请选择保存。
  - b. 要保存设置并返回工作草稿页面，请选择保存并退出。
12. 要测试更新的设置，请在“测试”窗口中选择“准备”。



5 Pre-processing | **Orchestration** | KB response generation | Post-processing - Inactive

6 **Override orchestration template defaults**  
This template defines the order in which actions are executed. Enabling this will allow you to edit the template and override its default values. Disabling this means the agent will revert back to the default Bedrock template.

7 **Activate orchestration template**  
Enabling this means this template is used in generating agent responses. When disabled, this template will not affect agent responses regardless of how it is configured.

8 **Prompt template editor**  
1  
2  
3 Human:  
4 \* You are a research assistant AI that has been equipped with one or more functions to help you answer a <questions>. Your goal is to answer the user's question to the best of your ability, using the function(s) to gather more information if necessary to better answer the question. If you choose to call a function, the result of the function call will be added to the conversation history in <function\_results> tags (if the call succeeded) or <errors> tags (if the function failed). \$ask\_user\_missing\_parameters\$  
5 You were created with these instructions to consider as well:  
6 <auxiliary\_instructions>\$instructions\$/auxiliary\_instructions\$  
7  
8 Here are some examples of correct action by other, different agents with access to functions that may or may not be similar to ones you are provided.  
9  
10 \* <examples>  
11 <example\_docstring> Here is an example of how you would correctly answer a question using a <function\_call> and the corresponding <function\_result> >. Notice that you are free to think before deciding to make a <function\_call> in the <scratchpad>. <example\_docstring>  
12 \* <example>  
13 \* <functions>  
14 \* <function>

9 **Configurations**

▼ Randomness & Diversity

Temperature

Top P

Top K

▼ Length

Max completion length

Stop sequences

10a  Use Lambda function for parsing  
Parse Foundation model output to get the next action group/knowledge base to be invoked or check if the orchestration should end for the current user input.

10b **Parser Lambda function**

11

## API

要使用 API 操作配置高级提示，请发送 [CreateAgent](#) 或 [UpdateAgent](#) 调用并修改以下 `promptOverrideConfiguration` 对象。

```
"promptOverrideConfiguration": {
 "overrideLambda": "string",
 "promptConfigurations": [
 {
 "basePromptTemplate": "string",
 "inferenceConfiguration": {
 "maxLength": 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. 在 `promptConfigurations` 列表中，为要编辑的每个提示模板添加 `promptConfiguration` 对象。
2. 在 `promptType` 字段中指定要修改的提示。
3. 通过以下步骤修改提示模板：
  - a. 使用提示模板指定 `basePromptTemplate` 字段。
  - b. 在 `inferenceConfiguration` 对象中包含推理参数。有关推理配置的更多信息，请参阅[根基模型的推理参数](#)。
4. 要启用提示模板，`promptCreationMode` 请将设置为 `OVERRIDDEN`。
5. 要允许或阻止代理执行 `promptType` 字段中的步骤，请修改该 `promptState` 值。此设置可用于对代理的行为进行故障排除。
  - 如果 `promptState` 将 `DISABLED`、或 `POST_PROCESSING` 步骤设置为 `PRE_PROCESSINGKNOWLEDGE_BASE_RESPONSE_GENERATION`，则代理会跳过该步骤。
  - 如果将 `ORCHESTRATION` 步骤设置 `promptState` `DISABLED` 为，则代理在编排中仅将用户输入发送到基础模型。此外，代理按原样返回响应，无需编排 API 操作和知识库之间的调用。
  - 默认情况下，该 `POST_PROCESSING` 步骤为 `DISABLED`。默认情况下，`PRE_PROCESSING` 或 `ORCHESTRATION`、和 `KNOWLEDGE_BASE_RESPONSE_GENERATION` 步骤为 `ENABLED`。
6. 要使用您定义的 Lambda 函数来解析原始基础模型输出，请执行以下步骤：
  - a. 对于要为其启用 Lambda 函数的每个提示模板，请将设置为 `parserMode`。 `OVERRIDDEN`
  - b. 在对象的 `overrideLambda` 字段中指定 Lambda 函数的亚马逊资源名称 (ARN)。 `promptOverrideConfiguration`

## Amazon Bedrock 代理提示模板中的占位符变量

可以在代理提示模板中使用占位符变量。调用提示模板时，变量将由预先存在的配置填充。选择一个选项卡，查看可用于每个提示模板的变量。

## Pre-processing

| Variable      | 支持的型号                                | 取而代之的是                    |
|---------------|--------------------------------------|---------------------------|
| \$functions\$ | AnthropicClaude Instant , Claudev2.0 | 为代理配置的操作组 API 操作和知识库。     |
| \$tools\$     | AnthropicClaudev2.1                  |                           |
| \$对话_历史\$     | 全部                                   | 当前会话的对话历史记录               |
| \$问题\$        | 全部                                   | 会话中当前InvokeAgent 呼叫的用户输入。 |

## Orchestration

| Variable              | 支持的型号                                | 取而代之的是                                                                          |
|-----------------------|--------------------------------------|---------------------------------------------------------------------------------|
| \$functions\$         | AnthropicClaude Instant , Claudev2.0 | 为代理配置的操作组 API 操作和知识库。                                                           |
| \$tools\$             | AnthropicClaudev2.1                  |                                                                                 |
| \$agent_scratchpad\$  | 全部                                   | 为模型指定一个区域，用于写下其想法和所采取的行动。取而代之的是当前回合中先前迭代的预测和输出。为模型提供给定用户输入所取得的成果以及下一步应该做什么的上下文。 |
| \$any_function_name\$ | AnthropicClaude Instant , Claudev2.0 | 从代理操作组中存在的 API 名称中随机选择的 API 名称。                                                 |
| \$对话_历史\$             | 全部                                   | 当前会话的对话历史记录                                                                     |
| \$指令\$                | 全部                                   | 为代理配置的模型指令。                                                                     |

| Variable                | 支持的型号 | 取而代之的是                    |
|-------------------------|-------|---------------------------|
| \$prompt_session_attrib | 全部    | 在提示中保留会话属性                |
| \$问题\$                  | 全部    | 会话中当前InvokeAgent 呼叫的用户输入。 |

如果您允许代理通过执行以下操作之一向用户询问更多信息，则可以使用以下占位符变量：

- 在控制台中，在代理详细信息中的用户输入中进行设置。
- AMAZON.UserInput使用[CreateAgentActionGroup](#)或[UpdateAgentActionGroup](#)请求parentActionGroupSignature将设置为。

| Variable                         | 支持的型号                                | 取而代之的是                       |
|----------------------------------|--------------------------------------|------------------------------|
| \$ask_user_missing_parameters\$  | AnthropicClaude Instant , Claudev2.0 | 模特要求用户提供所需的缺失信息的说明           |
| \$ask_user_missing_information\$ | AnthropicClaudev2.1                  |                              |
| \$ask_user_confirm_parameters\$  | 全部                                   | 模型要求用户确认代理尚未收到或不确定的参数的说明。    |
| \$ask_user_function\$            | 全部                                   | 向用户提问的功能。                    |
| \$ask_user_function_format\$     | 全部                                   | 向用户提问的函数的格式。                 |
| \$ask_user_input_examples\$      | 全部                                   | 为模型提供一些示例，告诉模型如何预测何时应该向用户提问。 |

## Knowledge base response generation

| Variable           | 模型 | 取而代之的是                          |
|--------------------|----|---------------------------------|
| \$query\$          | 全部 | 当编排预测下一步是知识库查询时，由编排提示模型响应生成的查询。 |
| \$search_results\$ | 全部 | 检索到的用户查询结果                      |

## Post-processing

| Variable            | 模型 | 取而代之的是                    |
|---------------------|----|---------------------------|
| \$latest_response\$ | 全部 | 最后一次编排提示模型响应              |
| \$问题\$              | 全部 | 会话中当前InvokeAgent 呼叫的用户输入。 |
| \$responses\$       | 全部 | 行动组和知识库从当前回合中输出。          |

## Amazon Bedrock 代理中的解析器 Lambda 函数

每个提示模板都包含一个您可以修改的解析器 Lambda 函数。要编写自定义解析器 Lambda 函数，您必须了解代理发送的输入事件以及代理期望作为 Lambda 函数输出的响应。您可以编写处理程序函数来处理输入事件中的变量并返回响应。有关 AWS Lambda 工作原理的更多信息，请参阅《开发者指南》AWS Lambda 中的[事件驱动调用](#)。

### 主题

- [解析器 Lambda 输入事件](#)
- [解析器 Lambda 响应](#)
- [解析器 Lambda 示例](#)

## 解析器 Lambda 输入事件

以下是来自代理的输入事件的一般结构。使用以下字段编写 Lambda 处理程序函数。

```
{
 "messageVersion": "1.0",
 "agent": {
 "name": "string",
 "id": "string",
 "alias": "string",
 "version": "string"
 },
 "invokeModelRawResponse": "string",
 "promptType": "ORCHESTRATION | POST_PROCESSING | PRE_PROCESSING |
KNOWLEDGE_BASE_RESPONSE_GENERATION ",
 "overrideType": "OUTPUT_PARSER"
}
```

以下列表描述了输入事件字段：

- `messageVersion` – 消息的版本，用于标识进入 Lambda 函数的事件数据格式以及 Lambda 函数的预期响应格式。Amazon Bedrock 代理仅支持版本 1.0。
- `agent` – 包含提示所属代理的名称、ID、别名和版本的相关信息。
- `invokeModelRawResponse` – 要解析其输出的提示的原始根基模型输出。
- `promptType` – 要解析其输出的提示类型。
- `overrideType` – 此 Lambda 函数覆盖的构件。当前，`OUTPUT_PARSER`仅支持，这表示要重写默认解析器。

## 解析器 Lambda 响应

代理希望从 Lambda 函数获得符合以下格式的响应。代理使用响应进行进一步的编排或帮助其向用户返回响应。使用 Lambda 函数响应字段配置如何返回输出。

```
{
 "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",
 "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"},
 ...
]
 }
]
 },
 "postProcessingParsedResponse": {
 "responseText": "string",
 "citations": {
 "generatedResponseParts": [{
 "text": "string",
 "references": [{
 "sourceId": "string"
 }]
 }]
 }
 }
}

```

以下列表描述了 Lambda 响应字段：

- `messageVersion` – 消息的版本，用于标识进入 Lambda 函数的事件数据格式以及 Lambda 函数的预期响应格式。Amazon Bedrock 代理仅支持版本 1.0。
- `promptType` – 当前回合的提示类型。
- `preProcessingParsedResponse` – PRE\_PROCESSING 提示类型的解析后响应。
- `orchestrationParsedResponse` – ORCHESTRATION 提示类型的解析后响应。有关更多详细信息，请参阅下文。
- `knowledgeBaseResponseGenerationParsedResponse` – KNOWLEDGE\_BASE\_RESPONSE\_GENERATION 提示类型的解析后响应。
- `postProcessingParsedResponse` – POST\_PROCESSING 提示类型的解析后响应。

有关四个提示模板的解析响应的更多详细信息，请参阅以下选项卡。

### preProcessingParsedResponse

```

{
 "isValidInput": "boolean",

```



```
"rationale": "string"
}
```

`preProcessingParsedResponse` 包含以下字段。

- `isValidInput` – 用于指定用户输入是否有效。您可以定义函数来确定如何描述用户输入的有效性。
- `rationale` – 对用户输入分类的推理。这个原理由模型在原始响应中提供，Lambda 函数对其进行解析，代理将其呈现在跟踪中进行预处理。

## `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": [
 {
 "text": "string",
 "references": [
 {"sourceId": "string"},
 ...
]
 },
 ...
]
 },
 ...
]
}
}
}
}

```

`orchestrationParsedResponse`包含以下字段：

- `rationale` – 基于根基模型输出对后续行动的推理。您可以定义要从模型输出中解析的函数。
- `parsingErrorDetails` – 包含 `repromptResponse`，这是一条消息，用于在无法解析模型响应时重新提示模型更新其原始响应。您可以定义函数来处理如何重新提示模型。
- `responseDetails` – 包含有关如何处理根基模型输出的详细信息。包含 `invocationType`，这是代理要采取的下一步行动，还有一个应该与 `invocationType` 匹配的字段。可能有以下对象。
  - `agentAskUser` – 与 `ASK_USER` 调用类型兼容。此调用类型用于结束编排步骤。包含用于向用户询问更多信息的 `responseText`。您可以定义函数来处理这个字段。
  - `actionGroupInvocation` – 与 `ACTION_GROUP` 调用类型兼容。您可以定义函数来确定要调用的操作组和要传递的参数。包含以下字段：
    - `actionGroupName` – 要调用的操作组。
    - `apiName`— 要在操作组中调用的 API 操作的名称。
    - `verb`— 要使用的 API 操作的方法。
    - `actionGroupInput`— 包含要在 API 操作请求中指定的参数。
  - `agentKnowledgeBase` – 与 `KNOWLEDGE_BASE` 调用类型兼容。您可以定义函数来确定如何查询知识库。包含以下字段：
    - `knowledgeBaseId` – 知识库的唯一标识符。
    - `searchQuery`— 包含要发送到该value领域知识库的查询。
  - `agentFinalResponse` – 与 `FINISH` 调用类型兼容。此调用类型用于结束编排步骤。在 `responseText` 字段中包含给用户的响应，在 `citations` 对象中包含响应的引文。

## knowledgeBaseResponseGenerationParsedResponse

```
{
 "generatedResponse": {
 "generatedResponseParts": [
 {
 "text": "string",
 "references": [
 { "sourceId": "string" },
 ...
]
 },
 ...
]
 }
}
```

knowledgeBaseResponseGenerationParsedResponse 包含查询知识库的摘录和数据源的参考文献。generatedResponse

## postProcessingParsedResponse

```
{
 "responseText": "string",
 "citations": {
 "generatedResponseParts": [
 {
 "text": "string",
 "references": [
 { "sourceId": "string" },
 ...
]
 },
 ...
]
 }
}
```

postProcessingParsedResponse 包含以下字段：

- **responseText** – 返回给最终用户的响应。您可以定义函数来确定响应的格式。
- **citations** – 包含响应的引文列表。每个引文都显示被引文本及其参考文献。

## 解析器 Lambda 示例

要查看特定提示模板的示例解析器 Lambda 函数，请从以下选项卡中进行选择。还显示了发送到函数的示例输入事件和来自函数的响应。lambda\_handler 函数会将解析后的响应返回给代理。

### Pre-processing

#### 示例函数

```
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"
 },
 "invokeModelRawResponse": " <thinking>\n\nThe user is asking about the
instructions provided to the function calling agent. This input is trying to gather
information about what functions/API's or instructions our function calling agent
has access to. Based on the categories provided, this input belongs in Category B.
\n</thinking>\n\n\n<category>B</category>",
 "messageVersion": "1.0",
 "overrideType": "OUTPUT_PARSER",
 "promptType": "PRE_PROCESSING"
}

```

```

{
 "promptType": "PRE_PROCESSING",
 "preProcessingParsedResponse": {

```

```

 "rationale": "\n\nThe user is asking about the instructions provided to the
function calling agent. This input is trying to gather information about what
functions/API's or instructions our function calling agent has access to. Based on
the categories provided, this input belongs in Category B.\n\n",
 "isValidInput": false
 }
}

```

## Orchestration

以下是 Anthropic Claude 2 和 Anthropic Claude 2.1 的示例函数。

### AnthropicClaude2

```

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+)::(.+)\(((+)\)"

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_REPROMPT_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)

 return final_response, generated_response_parts

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()
 raise ValueError(MISSING_API_INPUT_FOR_USER_REPROMPT_MESSAGE)
 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
 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,
 "actionGroupName": resource_name,
 "apiName": function,
 "actionGroupInput": parameters
}

return parsed_response

def addRepromptResponse(parsed_response, error):
 error_message = str(error)
 logger.warn(error_message)

 parsed_response['orchestrationParsedResponse']['parsingErrorDetails'] = {
 'repromptResponse': error_message
 }

```

## AnthropicClaude2.1

```

import logging
import re
import xml.etree.ElementTree as ET

RATIONALE_REGEX_LIST = [
 "(.*?)(<function_calls>)",
 "(.*?)(<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_calls>"

ASK_USER_FUNCTION_CALL_REGEX = r"<tool_name>user::askuser</tool_name>"
ASK_USER_FUNCTION_CALL_PATTERN = re.compile(ASK_USER_FUNCTION_CALL_REGEX, re.DOTALL)

ASK_USER_TOOL_NAME_REGEX = r"<tool_name>((.|\n)*?)</tool_name>"
ASK_USER_TOOL_NAME_PATTERN = re.compile(ASK_USER_TOOL_NAME_REGEX, re.DOTALL)

TOOL_PARAMETERS_REGEX = r"<parameters>((.|\n)*?)</parameters>"
TOOL_PARAMETERS_PATTERN = re.compile(TOOL_PARAMETERS_REGEX, re.DOTALL)

ASK_USER_TOOL_PARAMETER_REGEX = r"<question>((.|\n)*?)</question>"
ASK_USER_TOOL_PARAMETER_PATTERN = re.compile(ASK_USER_TOOL_PARAMETER_REGEX,
re.DOTALL)

KNOWLEDGE_STORE_SEARCH_ACTION_PREFIX = "x_amz_knowledgebase_"

FUNCTION_CALL_REGEX = r"(?<=<function_calls>)(.*)"

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 parameter 'question' for
user::askuser function call. Please try again with the correct argument added."
ASK_USER_FUNCTION_CALL_STRUCTURE_REPROMPT_MESSAGE = "The function call format is
incorrect. The format for function calls to the askuser function must be: <invoke>
<tool_name>user::askuser</tool_name><parameters><question>$QUESTION</question></
parameters></invoke>."
FUNCTION_CALL_STRUCTURE_REPROMPT_MESSAGE = "The function call format is incorrect.
The format for function calls must be: <invoke> <tool_name>$TOOL_NAME</tool_name>
<parameters> <$PARAMETER_NAME>$PARAMETER_VALUE</$PARAMETER_NAME>...</parameters></
invoke>."

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)

return final_response, generated_response_parts

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:
 parameters_matches =
TOOL_PARAMETERS_PATTERN.search(sanitized_llm_response)
 params = parameters_matches.group(1).strip()
 ask_user_question_matcher =
ASK_USER_TOOL_PARAMETER_PATTERN.search(params)
 if ask_user_question_matcher:
 ask_user_question = ask_user_question_matcher.group(1)
 return ask_user_question
 raise ValueError(MISSING_API_INPUT_FOR_USER_REPROMPT_MESSAGE)
 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_REMPROMPT_MESSAGE)

 tool_name_matches = ASK_USER_TOOL_NAME_PATTERN.search(sanitized_response)
 tool_name = tool_name_matches.group(1)
 parameters_matches = TOOL_PARAMETERS_PATTERN.search(sanitized_response)
 params = parameters_matches.group(1).strip()

 action_split = tool_name.split(':::')
 verb = action_split[0].strip()
 resource_name = action_split[1].strip()
 function = action_split[2].strip()

 xml_tree = ET.ElementTree(ET.fromstring("<parameters>{}/</parameters>".format(params)))
 parameters = {}
 for elem in xml_tree.iter():
 if elem.text:
 parameters[elem.tag] = {'value': elem.text.strip('" ')}

 parsed_response['orchestrationParsedResponse']['responseDetails'] = {}

 # Function calls can either invoke an action group or a knowledge base.
 # Mapping to the correct variable names accordingly
 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,
 "actionGroupName": resource_name,
 "apiName": function,
 "actionGroupInput": parameters
}

return parsed_response

def addRepromptResponse(parsed_response, error):
 error_message = str(error)
 logger.warn(error_message)

 parsed_response['orchestrationParsedResponse']['parsingErrorDetails'] = {
 'repromptResponse': error_message
 }

```

## 示例请求

```

{
 'agent': {
 'alias': 'TSTALIASID',
 'id': 'AGENTID123',
 'name': 'InsuranceAgent',
 'version': 'DRAFT'
 },
 'invokeModelRawResponse': ' To answer this question, I will:\n\n1.
Call the GET::x_amz_knowledgebase_KBID123456::Search function to search
for a phone number to call.\n\nI have checked that I have access to the
GET::x_amz_knowledgebase_KBID23456::Search function.\n\n</scratchpad>\n\n
\n<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'
}

```

## 响应示例

```
{
 'promptType': 'ORCHESTRATION',
 'orchestrationParsedResponse': {
 'rationale': 'To answer this question, I will:\\n\\n1. Call the
GET::x_amz_knowledgebase_KBID123456::Search function to search for a phone
number to call Farmers.\\n\\nI 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

### 示例函数

```
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
 }

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

```

### 示例请求

```
{
```

```

 '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.
\\\\\\n</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\\":\\"\\\\\\n\\\\\\
\\nHuman:\\"}',
 'messageVersion': '1.0',
 'overrideType': 'OUTPUT_PARSER',
 'promptType': 'KNOWLEDGE_BASE_RESPONSE_GENERATION'
 }

```

## 响应示例

```

{
 'promptType': 'KNOWLEDGE_BASE_RESPONSE_GENERATION',
 'knowledgeBaseResponseGenerationParsedResponse': {
 'generatedResponse': {
 'generatedResponseParts': [
 {
 '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.
\\\\\\n',
 'references': [
 {'sourceId': '1234567-1234-1234-1234-123456789abc'},
 {'sourceId': '2345678-2345-2345-2345-23456789abcd'},
]
 }
]
 }
 }
}

```

```

 {'sourceId': '3456789-3456-3456-3456-3456789abcde'}
]
}
]
}
}
}

```

## Post-processing

### 示例函数

```

import json
import re
import logging

FINAL_RESPONSE_REGEX = r"<final_response>([\s\S]*?)</final_response>"
FINAL_RESPONSE_PATTERN = re.compile(FINAL_RESPONSE_REGEX, re.DOTALL)

logger = logging.getLogger()

This parser lambda is an example of how to parse the LLM output for the default
PostProcessing prompt
def lambda_handler(event, context):
 logger.info("Lambda input: " + str(event))
 raw_response = event['invokeModelRawResponse']

 parsed_response = {
 'promptType': 'POST_PROCESSING',
 'postProcessingParsedResponse': {}
 }

 matcher = FINAL_RESPONSE_PATTERN.search(raw_response)
 if not matcher:
 raise Exception("Could not parse raw LLM output")
 response_text = matcher.group(1).strip()

 parsed_response['postProcessingParsedResponse']['responseText'] = response_text

 logger.info(parsed_response)
 return parsed_response

```

### 示例请求

```
{
 'agent': {
 'alias': 'TSTALIASID',
 'id': 'AGENTID123',
 'name': 'InsuranceAgent',
 'version': 'DRAFT'
 },
 'invokeModelRawResponse': ' <final_response>\n\nBased 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.\n\n</
final_response>',
 'messageVersion': '1.0',
 'overrideType': 'OUTPUT_PARSER',
 'promptType': 'POST_PROCESSING'
}
```

## 响应示例

```
{
 '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.'
 }
}
```

## 控制会话上下文

为了更好地控制会话上下文，可以在代理中修改[SessionState](#)对象。该[SessionState](#)对象包含两种类型的属性，可用于在用户对话期间为代理提供对话上下文。

- 会话属性-在用户和代理之间的会话中持续存在的属性。只要未超过会话时间限制 (theIdleSessionTTLInSeconds)，所有使用相同的[InvokeAgent](#)请求都sessionId属于同一个会话。
- promptSessionAttributes— 在一回合（一次[InvokeAgent](#)调用）内持续存在的属性。编辑编排基础提示模板时，可以使用 \$prompt\_session\_attributes\$ [占位符](#)。此占位符将在运行时使用您在promptSessionAttributes字段中指定的属性进行填充。

该[SessionState](#)对象的一般格式如下。

```
{
 "sessionAttributes": {
 "<attributeName1>": "<attributeValue1>",
 "<attributeName2>": "<attributeValue2>",
 ...
 },
 "promptSessionAttributes": {
 "<attributeName3>": "<attributeValue3>",
 "<attributeName4>": "<attributeValue4>",
 ...
 }
}
```

您可以通过两个不同的步骤定义会话状态属性。

- 当您设置操作组并[编写 Lambda 函数](#)时，请在返回给 Amazon Bedrock 的[响应事件](#)promptSessionAttributes中包含sessionAttributes或。
- 在运行时，当您发送[InvokeAgent](#)请求时，请在请求正文中包含一个sessionState对象，以便在对话中动态更改会话状态属性。

## 会话属性示例

以下示例使用会话属性对发送给用户的消息进行个性化设置。

1. 编写您的应用程序代码，要求用户提供他们的名字和他们想要向代理发出的请求，并将答案存储为变量<first\_name>和<request>。
2. 编写您的应用程序代码以发送带有以下正文的[InvokeAgent](#)请求：

```
{
 "inputText": "<request>",
 "sessionState": {
 "sessionAttributes": {
 "firstName": "<first_name>"
 }
 }
}
```

3. 当用户使用您的应用程序并提供他们的名字时，您的代码将发送名字作为会话属性，代理将在会话期间存储他们的名字。
4. 由于会话属性是在 [Lambda 输入事件](#) 中发送的，因此您可以在 Lambda 函数中为操作组引用这些会话属性。例如，如果操作 [API 架构](#) 要求在请求正文中使用名字，则可以在为操作组编写 Lambda 函数时使用 firstName 会话属性，以便在发送 API 请求时自动填充该字段。

## 提示会话属性示例

以下一般示例使用提示会话属性为代理提供临时上下文。

1. 编写应用程序代码，将用户请求存储在名为的变量中<request>。
2. 如果用户在中使用表示相对时间的单词（例如“tomorrow”），则编写应用程序代码以检索用户所在地的时区<request>，并存储在名为的变量中<timezone>。
3. 编写您的应用程序以发送带有以下正文的[InvokeAgent](#)请求：

```
{
 "inputText": "<request>",
 "sessionState": {
 "promptSessionAttributes": {
 "timeZone": "<timezone>"
 }
 }
}
```

```
}
```

4. 如果用户使用表示相对时间的单词，则您的代码将发送 `prompt_session_attributes.timeZone` 属性，代理将在[回合](#)期间将其存储。
5. 例如，如果用户询问 **I need to book a hotel for tomorrow**，您的代码会将用户的时区发送给代理，代理可以确定“明天”所指的确切日期。
6. 提示会话属性可以在以下步骤中使用。
  - 如果您在编排提示模板中包含 `$prompt_session_attributes` [占位符](#)，则 FM 的编排提示将包含提示会话属性。
  - [提示会话属性在 Lambda 输入事件中发送，可用于帮助填充 API 请求或在响应中返回。](#)

## 部署 Amazon Bedrock 代理

当你第一次创建 Amazon Bedrock 代理时，你有一个工作草稿版本 (DRAFT) 和一个指向工作草稿版本的测试别名 (TSTALIASID)。当您代理进行更改时，更改将应用于工作草稿。在对代理人的行为感到满意之前，您可以反复修改工作草稿。然后，您可以通过创建代理的别名来设置代理以进行部署和集成到应用程序中。

要部署代理，必须创建别名。在创建别名期间，Amazon Bedrock 会自动为您的代理创建一个版本。别名指向这个新创建的版本。或者，您可以将别名指向先前创建的代理版本。然后，您将应用程序配置为对该别名进行 API 调用。

版本是一种将资源保留在创建时存在的快照。如有必要，您可以继续修改工作草稿并创建代理的新别名（以及相应的版本）。在 Amazon Bedrock 中，您可以通过创建默认指向新版本的别名来创建代理的新版本。Amazon Bedrock 创建的版本按数字顺序排列，从 1 开始。

版本是不可变的，因为它们充当你创建代理时的快照。要在生产环境中对代理进行更新，必须创建一个新版本并将应用程序设置为调用指向该版本的别名。

使用别名，您可以在代理的不同版本之间高效切换，而无需应用程序跟踪版本。例如，如果有需要快速恢复的更改，则可以将别名更改为指向代理的先前版本。

### 部署您的代理

1. 创建代理的别名和版本。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

创建别名 ( 也可以创建新版本 )

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在“别名”部分中，选择“创建”。
4. 输入别名的唯一名称并提供可选描述。
5. 请选择以下选项之一：
  - 要创建新版本，请选择创建新版本并将其与此别名相关联。
  - 要使用现有版本，请选择使用现有版本关联此别名。从下拉菜单中，选择要将别名关联到的版本。
6. 选择创建别名。成功横幅出现在顶部。

## API

要为代理创建别名，请使用适用于 Amazon Bedrock 的 [Amazon Bedrock 代理构建时终端节点发送CreateAgentAlias请求](#) ( [有关](#) 请求和响应格式以及字段详情，请参阅链接 )。要创建新版本并将此别名与之关联，请将该routingConfiguration对象保留为未指定。

### [参见代码示例](#)

2. 通过将您的应用程序设置为使用适用于 Amazon Bedrock 的 [代理运行时终端节点发送InvokeAgent请求](#) ( [有关](#) 请求和响应格式以及字段详情，请参阅链接 ) 来部署您的代理。在该agentAliasId字段中，指定指向要使用的代理版本的别名的 ID。

要了解如何管理代理的版本和别名，请从以下主题中进行选择。

### 主题

- [在 Amazon Bedrock 中管理代理的版本](#)
- [在 Amazon Bedrock 中管理代理的别名](#)

## 在 Amazon Bedrock 中管理代理的版本

创建代理版本后，您可以查看有关该版本的信息或将其删除。您只能通过创建新别名来创建代理的新版本。

### 主题

- [在 Amazon Bedrock 中查看有关代理版本的信息](#)
- [在 Amazon Bedrock 中删除代理的某个版本](#)

## 在 Amazon Bedrock 中查看有关代理版本的信息

要了解如何查看有关代理版本的信息，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 查看有关代理版本的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 从“版本”部分选择要查看的版本。
4. 要查看有关代理版本所附模型、操作组或知识库的详细信息，请选择要查看的信息的名称。您无法修改版本的任何部分。要对代理进行修改，请使用工作草稿并创建新版本。

### API

要获取有关代理版本的信息，请使用适用于 [Amazon Bedrock 的代理构建时终端节点](#) 发送 [GetAgentVersion](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。指定 `agentId` 和 `agentVersion`。

要列出有关代理版本的信息，请使用 [Amazon Bedrock 代理构建时终端节点](#) 发送 [ListAgentVersions](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)），并指定 `agentId`。您可以指定以下可选参数：

| 字段                      | 简短描述            |
|-------------------------|-----------------|
| <code>maxResults</code> | 要在响应中返回的结果数量上限。 |

| 字段        | 简短描述                                                                             |
|-----------|----------------------------------------------------------------------------------|
| nextToken | 如果结果多于您在maxResults 字段中指定的数字，则响应会返回一个nextToken 值。要查看下一批结果，请在另一个请求中发送该nextToken 值。 |

## 在 Amazon Bedrock 中删除代理的某个版本

要了解如何删除代理版本，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 删除代理的某个版本

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 要选择要删除的版本，请在“版本”部分中，选择要删除的版本旁边的选项按钮。
4. 选择 Delete ( 删除)。
5. 将出现一个对话框，警告您注意删除的后果。要确认要删除版本，请在输入字段**delete**中输入并选择删除。
6. 此时会出现一个横幅，通知您该版本正在被删除。删除完成后，将显示成功横幅。

### API

要删除某个版本的代理，请使用适用于 [Amazon Bedrock 的代理构建](#) 时终端节点发送 [DeleteAgentVersion](#) 请求（请求和响应格式以及字段详情参见链接）。默认情况下，skipResourceInUseCheck 参数为 false，如果资源正在使用中，则会停止删除。如果设置 skipResourceInUseCheck 为 true，则即使资源正在使用中，也会删除该资源。

## 在 Amazon Bedrock 中管理代理的别名

创建代理的别名后，您可以查看有关该代理的信息、对其进行编辑或删除。

### 主题

- [在 Amazon Bedrock 中查看有关代理别名的信息](#)
- [在 Amazon Bedrock 中编辑代理的别名](#)
- [在 Amazon Bedrock 中删除代理的别名](#)

## 在 Amazon Bedrock 中查看有关代理别名的信息

要了解如何查看有关代理别名的信息，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 查看别名的详细信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 从“别名”部分选择要查看的别名。
4. 您可以查看与别名关联的别名和标签的名称和描述。

### API

要获取有关代理别名的信息，请使用适用于 [Amazon Bedrock 的代理构建时终端节点](#) 发送 [GetAgentAlias](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。指定 `agentId` 和 `agentAliasId`。

要列出有关代理别名的信息，请使用 [Amazon Bedrock 代理构建时终端节点](#) 发送 [ListAgentVersions](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)），并指定 `agentId`。您可以指定以下可选参数：

| 字段                      | 简短描述                                                                                                                       |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>maxResults</code> | 要在响应中返回的结果数量上限。                                                                                                            |
| <code>nextToken</code>  | 如果结果多于您在 <code>maxResults</code> 字段中指定的数字，则响应会返回一个 <code>nextToken</code> 值。要查看下一批结果，请在另一个请求中发送该 <code>nextToken</code> 值。 |

要查看别名的所有标签，请使用 [Amazon Bedrock 代理构建时终端节点发送ListTagsForResource](#) 请求（请求和响应格式以及字段详情参见链接），并附上别名的亚马逊资源名称 (ARN)。

## 在 Amazon Bedrock 中编辑代理的别名

要了解如何编辑代理的别名，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 编辑别名

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 在“别名”部分中，选择要编辑的别名旁边的选项按钮。
4. 您可以编辑别名的名称和描述。此外，您还可以执行以下操作之一：
  - 要创建新版本并将此别名与该版本关联，请选择创建新版本并将其与该别名关联。
  - 要将此别名与其他现有版本相关联，请选择使用现有版本并关联此别名。

#### 添加或移除与别名关联的标签

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 从“别名”部分选择要为其管理标签的别名。
4. 在标签部分中，选择管理标签。
5. 要添加标签，请选择添加新标签。然后输入密钥并根据需要输入值。要删除标签，请选择移除。有关更多信息，请参阅[标记资源](#)。
6. 编辑完标签后，选择提交。

### API

要编辑代理别名，[UpdateAgentAlias](#) 请发送请求。由于所有字段都将被覆盖，因此请同时包括要更新的字段以及要保持不变的字段。



要向别名添加标签，请使用 [Amazon Bedrock 代理构建时终端节点发送TagResource](#) 请求（请求和响应格式以及字段详情参见链接），并附上别名的亚马逊资源名称 (ARN)。请求正文包含一个 tags 字段，该字段是一个包含您为每个标签指定的键值对的对象。

要从别名中删除标签，请使用 [Amazon Bedrock 代理构建时终端节点发送UntagResource](#) 请求（请求和响应格式以及字段详情参见链接），并附上别名的亚马逊资源名称 (ARN)。tagKeys 请求参数是一个列表，其中包含您要删除的标签的密钥。

## 在 Amazon Bedrock 中删除代理的别名

要了解如何删除代理的别名，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

#### 删除别名

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 从左侧导航窗格中选择“代理”。然后，在“代理”部分中选择一个代理。
3. 要选择要删除的别名，请在“别名”部分中，选择要删除的别名旁边的选项按钮。
4. 选择 Delete（删除）。
5. 将出现一个对话框，警告您注意删除的后果。要确认要删除别名，请在输入字段 **delete** 中输入并选择删除。
6. 此时会出现一条横幅，通知您该别名已被删除。删除完成后，将显示成功横幅。

### API

要删除代理的别名，请使用适用于 [Amazon Bedrock 的代理构建时终端节点发送DeleteAgentAlias](#) 请求（有关请求和响应格式以及字段详情，请参阅链接）。默认情况下，skipResourceInUseCheck 参数为 false，如果资源正在使用中，则会停止删除。如果设置 skipResourceInUseCheck 为 true，则即使资源正在使用中，也会删除该资源。

#### [参见代码示例](#)

# 自定义模型

模型定制是向模型提供训练数据以提高其在特定用例中的性能的过程。您可以自定义 Amazon Bedrock 基础模型，以提高其性能并创造更好的客户体验。Amazon Bedrock 目前提供以下自定义方法。

- 继续预训练

通过熟悉基础模型的某些类型的输入，提供未标记的数据，以便对基础模型进行预训练。您可以提供来自特定主题的数据，以便向这些区域展示模型。持续的预训练过程将调整模型参数以适应输入数据并提高其领域知识。

例如，您可以使用私有数据（例如业务文档）训练模型，这些数据并未公开用于训练大型语言模型。此外，您还可以使用更多可用的未标记数据重新训练模型，从而不断改进模型。

- 微调

提供带标签的数据，以便训练模型以提高特定任务的性能。通过提供带有标签的示例的训练数据集，模型可以学会关联应为某些类型的输入生成哪些类型的输出。在此过程中会调整模型参数，并提高模型在训练数据集所代表的任务中的性能。

有关模型自定义配额的信息，请参阅[模型自定义配额](#)。

## Note

模型训练费用基于模型处理的令牌数量（训练数据语料库中的令牌数量 × 周期数）和每个模型每月收取的模型存储费用。有关更多信息，请参阅 [Amazon Bedrock 定价](#)。

在模型自定义中，您需要执行以下步骤。

1. 为您的自定义任务@@ [创建训练和验证数据集（如果适用）](#)。
2. 如果您计划使用新的自定义 IAM 角色，请[设置 IAM 权限](#)以访问您的数据的 S3 存储桶。您也可以使用现有角色或让控制台自动创建具有适当权限的角色。
3. （可选）配置 [KMS 密钥](#)和/或 [VPC](#) 以提高安全性。
4. [创建微调或继续预训练作业](#)，通过调整[超](#)参数值来控制训练过程。
5. 通过查看训练或验证指标或使用模型评估来@@ [分析结果](#)。
6. 为您新创建@@ [的自定义模型购买预配置吞吐量](#)。

7. 像在 Amazon Bedrock 任务（例如模型推断）中[使用基础模型一样使用您的自定义模型](#)。

## 主题

- [支持模型定制的区域和型号](#)
- [模型定制的先决条件](#)
- [提交模型定制任务](#)
- [管理模型定制作业](#)
- [分析模型定制作业的结果](#)
- [使用自定义模型](#)
- [模型定制的代码示例](#)
- [模型自定义准则](#)
- [故障排除](#)

## 支持模型定制的区域和型号

下表显示了每种自定义方法的区域支持：

| 区域                 | 微调 | 继续进行预训练 |
|--------------------|----|---------|
| 美国东部（弗吉尼亚州北部）      | 是  | 支持      |
| US West（Oregon）    | 支持 | 支持      |
| AWS GovCloud（美国西部） | 支持 | 不支持     |

下表显示了每种自定义方法的模型支持：

| 模型名称                           | 型号标识                       | 微调 | 继续进行预训练 |
|--------------------------------|----------------------------|----|---------|
| Amazon Titan Text G1 - Express | 亚马逊。 titan-text-express-v1 | 支持 | 支持      |
| Amazon Titan Text G1 - Lite    | 亚马逊。 titan-text-lite-v1    | 支持 | 支持      |

| 模型名称                                  | 型号标识                          | 微调 | 继续进行预训练 |
|---------------------------------------|-------------------------------|----|---------|
| Amazon Titan Image Generator G1       | 亚马逊。 titan-image-generator-v1 | 支持 | 不支持     |
| 亚马逊 Titan Multimodal Embeddings G1 G1 | 亚马逊。 titan-embed-image-v1     | 支持 | 不支持     |
| Cohere Command                        | 一致。 command-text-v14          | 支持 | 不支持     |
| Cohere Command Light                  | 一致。 command-light-text-v14    | 支持 | 不支持     |
| MetaLlama 213B                        | meta.llama2-131b-chat-v       | 支持 | 不支持     |
| MetaLlama 270B                        | meta.llama2-70b-chat-v        | 支持 | 不支持     |

## 模型定制的先决条件

在开始模型自定义任务之前，您需要满足以下先决条件：

1. 确定您计划进行微调还是继续预训练作业，以及计划使用哪种模型。您所做的选择决定了您输入到自定义作业的数据集的格式。
2. 准备训练数据集文件。如果您选择的自定义方法和模型支持验证数据集，则也可以准备验证数据集文件。按照中的以下步骤操作，[准备数据集](#)然后[将文件上传到](#) Amazon S3 存储桶。
3. ( 可选 ) 按照中的说明创建具有适当权限的自定义 AWS Identity and Access Management (IAM) [服务角色为模型自定义创建服务角色](#)来设置该角色。如果您计划使用自动为您创建服务角色，AWS Management Console 则可以跳过此先决条件。
4. ( 可选 ) 设置额外的安全配置。
  - 您可以对输入和输出数据、自定义任务或向自定义模型发出的推理请求进行加密。有关更多信息，请参阅[对模型自定义任务和构件进行加密](#)。
  - 您可以创建虚拟私有云 (VPC) 来保护您的自定义任务。有关更多信息，请参阅[使用 VPC 保护模型自定义任务](#)。

## 主题

- [准备数据集](#)
- [使用 VPC 保护模型自定义任务](#)

## 准备数据集

在开始模型自定义任务之前，您需要最低限度地准备训练数据集。是否支持验证数据集以及您的训练和验证数据集的格式取决于以下因素。

- 自定义任务的类型（微调或持续的预训练）。
- 数据的输入和输出模式。

要查看不同模型的数据集和文件要求，请参阅[模型自定义配额](#)。

选择与您的用例相关的选项卡。

### Fine-tuning: Text-to-text

要对 text-to-text 模型进行微调，请创建一个包含多行 JSON 的 JSONL 文件来准备训练和可选的验证数据集。每个 JSON 行都是一个同时包含 prompt 和 completion 字段的示例。针对每个令牌使用 6 个字符，作为令牌数量的近似值。格式如下所示。

```
{"prompt": "<prompt1>", "completion": "<expected generated text>"}
{"prompt": "<prompt2>", "completion": "<expected generated text>"}
{"prompt": "<prompt3>", "completion": "<expected generated text>"}
```

以下是问答任务的示例项目：

```
{"prompt": "what is AWS", "completion": "it's Amazon Web Services"}
```

### Fine-tuning: Text-to-image & Image-to-embeddings

要微调 text-to-image 或 image-to-embedding 模型，请通过创建包含多行 JSON 的 JSONL 文件来准备训练数据集。不支持验证数据集。每行 JSON 内容都是一个样本，其中包含 image-ref（图像的 Amazon S3 URI）和 caption（可作为图像的提示）。

这些图像必须是 JPEG 或 PNG 格式。

```
{"image-ref": "s3://bucket/path/to/image001.png", "caption": "<prompt text>"}
{"image-ref": "s3://bucket/path/to/image002.png", "caption": "<prompt text>"}
{"image-ref": "s3://bucket/path/to/image003.png", "caption": "<prompt text>"}
```

以下是示例项目：

```
{"image-ref": "s3://my-bucket/my-pets/cat.png", "caption": "an orange cat with white spots"}
```

要允许 Amazon Bedrock 访问图像文件，请添加与您在控制台中[访问训练和验证文件以及在 S3 中写入输出文件的权限](#)设置或自动设置的 Amazon Bedrock 模型自定义服务角色中的策略相似的 IAM 策略。您在训练数据集内提供的 Amazon S3 路径必须位于您在策略中指定的文件夹中。

### Continued Pre-training: Text-to-text

要对 text-to-text 模型进行持续预训练，请通过创建包含多行 JSON 行的 JSONL 文件来准备训练和可选的验证数据集。由于持续预训练涉及未标记的数据，因此每个 JSON 行都是仅包含一个字段的 input 样本。针对每个令牌使用 6 个字符，作为令牌数量的近似值。格式如下所示。

```
{"input": "<input text>"}
{"input": "<input text>"}
{"input": "<input text>"}
```

以下是训练数据中可能存在的示例项目。

```
{"input": "AWS stands for Amazon Web Services"}
```

## 使用 VPC 保护模型自定义任务

当您运行模型自定义作业时，该作业会访问您的 Amazon S3 存储桶来下载输入数据和上传作业指标。为了控制对数据的访问，我们建议您使用带有 [Amazon VPC 的虚拟私有云 \(VPC\)](#)。您可以进一步保护您的数据，方法是配置您的 VPC，使您的数据无法通过 Internet 获取，然后使用创建 VPC 接口终端节点 [AWS PrivateLink](#) 来建立与您的数据的私有连接。有关 Amazon VPC 和如何与 Amazon Bedrock AWS PrivateLink 集成的更多信息，请参阅 [使用 Amazon VPC 保护您的数据以及 AWS PrivateLink](#)。

执行以下步骤，配置和使用 VPC，用于模型自定义任务的训练、验证和输出数据。

### 主题

- [设置 VPC](#)
- [创建 Amazon S3 VPC 端点](#)
- [\( 可选 \) 使用 IAM 策略限制对您的 S3 文件的访问](#)
- [将 VPC 权限附加到模型自定义角色](#)
- [提交模型自定义任务时添加 VPC 配置](#)

## 设置 VPC

您可以将[默认 VPC](#)用于模型自定义数据，也可以按照[Amazon VPC 入门和创建 VPC 中的指南创建新的 VPC](#)。

在创建 VPC 时，我们建议您使用终端节点路由表的默认 DNS 设置，以便标准的 Amazon S3 网址（例如<http://s3-aws-region.amazonaws.com/training-bucket>）解析。

## 创建 Amazon S3 VPC 端点

如果您将 VPC 配置为无法访问互联网，则需要创建一个[Amazon S3 VPC 终端节点](#)，以允许您的模型自定义任务访问存储您的训练和验证数据以及存储模型工件的 S3 存储桶。

按照为 Amazon S3 创建[网关终端节点中的步骤创建 S3 VPC 终端节点](#)。

### Note

如果您的 VPC 不使用默认 DNS 设置，则需要通过配置终端节点路由表来确保训练作业中数据位置的 URL 能够解析。有关 VPC 终端节点[路由表的信息](#)，请参阅[网关终端节点路由](#)。

## ( 可选 ) 使用 IAM 策略限制对您的 S3 文件的访问

您可以使用[基于资源的策略](#)来更严格地控制对 S3 文件的访问。您可以使用以下类型的基于资源的策略的任意组合。

- [终端节点策略-终端节点策略限制](#)通过 VPC 终端节点进行访问。默认端点策略允许您的 VPC 中的任何用户或服务完全访问 Amazon S3。在创建终端节点时或创建终端节点之后，您可以选择将基于资源的策略附加到终端节点以添加限制，例如仅允许终端节点访问特定存储桶或仅允许特定的 IAM 角色访问终端节点。有关示例，请参阅[编辑 VPC 终端节点策略](#)。

以下是您可以附加到您的 VPC 终端节点的示例策略，仅允许其访问包含您的训练数据的存储桶。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "RestrictAccessToTrainingBucket",
 "Effect": "Allow",
 "Principal": "*",
 "Action": [
 "s3:GetObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::training-bucket",
 "arn:aws:s3:::training-bucket/*"
]
 }
]
}
```

- 存储桶策略-存储桶策略限制对 S3 存储桶的访问。您可以使用存储桶策略来限制对来自您的 VPC 的流量的访问。[要附加存储桶策略，请按照使用存储桶策略中的步骤进行操作，然后使用 aws: SourceVPC、aws: SourceVPCE 或 aws: 条件密钥。VpcSourceIp 有关示例，请参阅使用存储桶策略控制访问权限。](#)

以下是您可以附加到 S3 存储桶的策略示例，该策略将包含您的输出数据，以拒绝所有流向存储桶的流量，除非流量来自您的 VPC。

```
{
 "Version": "2012-10-17",
 "Statement": [{
 "Sid": "RestrictAccessToOutputBucket",
 "Effect": "Deny",
 "Principal": "*",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::output-bucket",
 "arn:aws:s3:::output-bucket/*"
]
 }],
}
```



```

 "Condition": {
 "StringNotEquals": {
 "aws:sourceVpc": "your-vpc-id"
 }
 }
]
}

```

## 将 VPC 权限附加到模型自定义角色

完成 VPC 和终端节点的设置后，您需要将以下权限附加到您的[模型自定义 IAM 角色](#)。修改此策略以仅允许访问您的任务所需的 VPC 资源。将 `## ID` 替换为您的 V `security-group-id`PC 中的值。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
],
 "Resource": [
 "arn:aws:ec2:region:account-id:network-interface/*"
],
 "Condition": {
 "StringEquals": {
 "aws:RequestTag/BedrockManaged": ["true"]
 },
 "ArnEquals": {

```

```

 "aws:RequestTag/BedrockModelCustomizationJobArn":
["arn:aws:bedrock:region:account-id:model-customization-job/*"]
 }
}
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
],
 "Resource": [
 "arn:aws:ec2:region:account-id:subnet/subnet-id",
 "arn:aws:ec2:region:account-id:subnet/subnet-id2",
 "arn:aws:ec2:region:account-id:security-group/security-group-id"
]
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterfacePermission",
 "ec2>DeleteNetworkInterface",
 "ec2>DeleteNetworkInterfacePermission",
],
 "Resource": "*",
 "Condition": {
 "ArnEquals": {
 "ec2:Subnet": [
 "arn:aws:ec2:region:account-id:subnet/subnet-id",
 "arn:aws:ec2:region:account-id:subnet/subnet-id2"
],
 "ec2:ResourceTag/BedrockModelCustomizationJobArn":
["arn:aws:bedrock:region:account-id:model-customization-job/*"]
 },
 "StringEquals": {
 "ec2:ResourceTag/BedrockManaged": "true"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": "arn:aws:ec2:region:account-id:network-interface/*",

```

```
 "Condition": {
 "StringEquals": {
 "ec2:CreateAction": [
 "CreateNetworkInterface"
]
 },
 "ForAllValues:StringEquals": {
 "aws:TagKeys": [
 "BedrockManaged",
 "BedrockModelCustomizationJobArn"
]
 }
 }
]
 }
 }
```

## 提交模型自定义任务时添加 VPC 配置

按照前几节所述配置 VPC 以及所需的角色和权限后，您可以创建使用此 VPC 的模型自定义作业。

当您为作业指定 VPC 子网和安全组时，Amazon Bedrock 会创建与其中一个子网中的安全组关联的弹性网络接口 (ENI)。ENI 允许 Amazon Bedrock 作业连接到您的 VPC 中的资源。有关 ENI 的信息，请参阅《Amazon VPC 用户指南》中的[弹性网络接口](#)。Amazon Bedrock 会使用 `BedrockManaged` 和 `BedrockModelCusomizationJobArn` 标签标记它所创建的 ENI。

我们建议您至少在每个可用区中提供一个子网。

您可以使用安全组来制定规则，以控制 Amazon Bedrock 对您的 VPC 资源的访问。

您可以将 VPC 配置为在控制台中使用，也可以通过 API 进行配置。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

对于 Amazon Bedrock 控制台，您可以在创建模型自定义作业时，在可选的 VPC 设置部分中指定 VPC 子网和安全组。有关配置作业的更多信息，请参阅[提交模型定制任务](#)。

#### Note

对于包含 VPC 配置的任务，控制台无法自动为您创建服务角色。按照中的指导[为模型自定义创建服务角色](#)创建自定义角色。

## API

提交请求时，您可以将VpcConfig作为[CreateModelCustomizationJob](#)请求参数包括在内，以指定要使用的 VPC 子网和安全组，如以下示例所示。

```
"VpcConfig": {
 "SecurityGroupIds": [
 "sg-0123456789abcdef0"
],
 "Subnets": [
 "subnet-0123456789abcdef0",
 "subnet-0123456789abcdef1",
 "subnet-0123456789abcdef2"
]
}
```

## 提交模型定制任务

您可以在 Amazon Bedrock 控制台或 API 中使用微调或继续预训练来创建自定义模型。自定义工作可能需要几个小时。作业的持续时间取决于训练数据的大小（记录数、输入令牌数和输出令牌数）、周期数和批次大小。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

要在控制台中提交模型定制任务，请执行以下步骤。

1. 在 Amazon Bedrock 控制台中，从左侧导航窗格中选择基础模型下的自定义模型。
2. 在“模型”选项卡中，选择“自定义模型”，然后选择“创建微调作业”或“创建持续预训练作业”，具体取决于要训练的模型类型。
3. 在模型详细信息部分中，执行以下操作。
  - a. 选择要使用自己的数据自定义的模型，然后为生成的模型命名。
  - b. （可选）默认情况下，Amazon Bedrock 会使用由其拥有和管理的密钥对您的模型进行加密。AWS 要使用[自定义 KMS 密钥](#)，请选择模型加密并选择密钥。
  - c. （可选）要将[标签](#)与自定义模型相关联，请展开标签部分并选择添加新标签。
4. 在作业配置部分，输入作业的名称，并可选择添加任何标签以与该作业关联。
5. （可选）要使用[虚拟私有云 \(VPC\) 保护您的训练数据和自定义任务](#)，请在 VPC 设置部分选择包含输入数据和输出数据 Amazon S3 位置、其子网和安全组的 VPC。

**Note**

如果您包含 VPC 配置，则控制台无法为该任务创建新的服务角色。[创建自定义服务角色](#)并添加与中所述示例类似的权限[将 VPC 权限附加到模型自定义角色](#)。

- 在输入数据部分，选择训练数据集文件的 S3 位置，以及验证数据集文件（如果适用）。
- 在“超参数”部分中，输入要在训练中使用的[超参数](#)的值。
- 在输出数据部分，输入 Amazon Bedrock 应保存任务输出的 Amazon S3 位置。Amazon Bedrock 会将每个周期的训练损失指标和验证损失指标存储在指定位置的单独文件中。
- 在服务访问权限部分，选择以下选项之一：
  - 使用现有服务角色 — 从下拉列表中选择一个服务角色。有关设置具有相应权限的自定义角色的更多信息，请参阅[为模型自定义创建服务角色](#)。
  - 创建和使用新的服务角色 — 输入服务角色的名称。
- 选择 Fine-tune 模型或创建持续的预训练作业开始作业。

## API

### 请求

使用 [Amazon Bedrock 控制平面终端节点](#) 发送请求 [CreateModelCustomizationJob](#)（有关请求和响应格式以及字段详情，请参阅[链接](#)），以提交模型自定义任务。您必须至少提供以下字段。

- roleArn— 具有自定义模型权限的服务角色的 ARN。如果您使用控制台，Amazon Bedrock 可以自动创建具有相应权限的角色，也可以按照中的步骤创建自定义角色。[为模型自定义创建服务角色](#)

**Note**

如果您包含字 vpcConfig 段，请确保该角色具有访问 VPC 的适当权限。有关示例，请参阅[将 VPC 权限附加到模型自定义角色](#)。

- baseModelIdentifier— 要[自定义的基础模型的模型 ID](#) 或 ARN。
- customModelName – 要为新自定义的模型使用的名称。
- jobName – 要为训练作业使用的名称。
- hyperParameters— 影响模型定制过程的[超参数](#)。

- `trainingDataConfig`— 一个包含训练数据集的 Amazon S3 URI 的对象。根据自定义方法和型号，您还可以包括 `validationDataConfig`。有关准备数据集的更多信息，请参阅[准备数据集](#)。
- `outputDataConfig`— 一个包含用于向其写入输出数据的 Amazon S3 URI 的对象。

如果您未指定 `customizationType`，则模型自定义方法默认为 `FINE_TUNING`。

为防止请求多次完成，请添加一个 `clientRequestToken`。

您可以包括以下可选字段以进行额外配置。

- `jobTags` 和/或 `customModelTags`-将[标签](#)与自定义任务或生成的自定义模型相关联。
- `customModelKmsKeyId`— 包括用于加密您的[自定义模型的自定义 KMS 密钥](#)。
- `vpcConfig`— 包括[虚拟私有云 \(VPC\) 的配置，以保护您的训练数据和自定义任务](#)。

响应

响应会返回一个 `jobArn`，你可以用它来[监视](#)或[停止](#)作业。

[参见代码示例](#)

## 管理模型定制作业

启动模型自定义任务后，您可以跟踪其进度或停止它。如果您通过 API 执行此操作，则需要 `jobArn`。您可以通过以下方式之一查找它：

1. 在 Amazon Bedrock 主机中
  1. 从左侧导航窗格中选择基础模型下的自定义模型。
  2. 从“训练作业”表中选择作业以查看详细信息，包括作业的 ARN。
2. 查看创建任务的呼叫或[CreateModelCustomizationJob](#)呼叫返回的响应中的 `jobArn` 字段。[CreateModelCustomizationJob](#)

## 监控模型定制作业

开始作业后，您可以在控制台或 API 中监控其进度。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

### 监控微调任务的状态

1. 在 Amazon Bedrock 控制台中，从左侧导航窗格中选择基础模型下的自定义模型。
2. 选择“训练作业”选项卡以显示您已启动的微调作业。查看状态列以监控作业的进度。
3. 选择一项作业以查看您的训练输入的详细信息。

## API

要列出有关您的所有模型自定义任务的信息，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [CreateModelCustomizationJob](#) 请求。[CreateModelCustomizationJob](#) 有关可以使用的过滤器，请参阅。

要监控模型自定义任务的状态，请使用任务的 [Amazon Bedrock 控制平面终端节点](#) 发送 [GetModelCustomizationJob](#) 请求。jobArn

要列出模型自定义任务的所有标签，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [ListTagsForResource](#) 请求（请求和响应格式以及字段详情参见链接），并附上任务的亚马逊资源名称 (ARN)。

[参见代码示例](#)

## 停止模型定制作业

您可以在 Amazon Bedrock 模型自定义作业进行期间停止该作业。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Warning

您无法恢复已停止的作业。Amazon Bedrock 会对停止工作之前用来训练模型的令牌收费。Amazon Bedrock 不会为已停止的作业创建中间自定义模型。

## Console

### 停止模型自定义作业

1. 在 Amazon Bedrock 控制台中，从左侧导航窗格中选择基础模型下的自定义模型。

2. 在“训练作业”选项卡中，选择要停止的作业旁边的单选按钮，或者选择要停止的作业以导航到详细信息页面。
3. 选择停止作业按钮。只有当任务的状态为时，您才能停止该作业Training。
4. 模态似乎警告您，如果您停止训练作业，则无法恢复它。选择停止作业进行确认。

## API

要停止模型自定义任务，请使用任务的 [Amazon Bedrock 控制平面终端节点](#) 发送 [CreateModelCustomizationJob](#) ( 请求和响应格式以及字段详情参见链接 ) 请求。jobArn

只有当任务的状态为时，您才能停止该作业IN\_PROGRESS。请status通过[GetModelCustomizationJob](#)请求查看。系统会将作业标记为终止，并将状态设置为 STOPPING。作业停止后，状态变为STOPPED。

[参见代码示例](#)

## 分析模型定制作业的结果

模型自定义任务完成后，您可以通过查看您在提交作业时指定的输出 S3 文件夹中的文件来分析训练过程的结果，或者查看有关模型的详细信息。Amazon Bedrock 将您的定制模型存储在 AWS 仅限您账户的托管存储空间中。

您也可以通过运行模型评估任务来评估您的模型。有关更多信息，请参阅 [模型评估](#)。

模型自定义任务的 S3 输出包含您的 S3 文件夹中的以下输出文件。仅当包含验证数据集时，验证工件才会出现。

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

使用 `step_wise_training_metrics.csv` 和 `validation_metrics.csv` 文件可分析模型自定义作业，并根据需要借助它们调整模型。

`step_wise_training_metrics.csv` 文件中的列如下所示。

- `step_number` — 训练过程中的步骤。从 0 开始。



- `epoch_number` — 训练过程中的时代。
- `training_loss` — 表示模型与训练数据的拟合程度。值越低表示拟合度越高。
- `perplexity` — 表示模型预测代币序列的效果如何。值越低表示预测能力越好。

`validation_metrics.csv` 文件中的列与训练文件相同，唯一的区别是 `validation_loss` (模型与验证数据的拟合程度) 代替 `training_loss` 了。

您可以通过直接打开 <https://console.aws.amazon.com/s3> 或在模型详细信息中找到输出文件夹的链接来找到输出文件。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

1. 在 Amazon Bedrock 控制台中，从左侧导航窗格中选择基础模型下的自定义模型。
2. 在模型选项卡中，选择一个模型以查看其详细信息。Job 名称可以在模型详情部分找到。
3. 要查看输出 S3 文件，请在输出数据部分中选择 S3 位置。
4. 在名称与模型的 Job 名称匹配的文件夹中查找训练和验证指标文件。

## API

要列出有关您的所有自定义模型的信息，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送请求 [ListCustomModels](#) (查看请求和响应格式以及字段详情链接)。 [ListCustomModels](#) 有关可以使用的过滤器，请参阅。

要列出自定义模型的所有标签，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [ListTagsForResource](#) 请求 (请求和响应格式以及字段详情参见链接)，并附上自定义模型的亚马逊资源名称 (ARN)。

要监控模型自定义任务的状态，请使用带有 [Amazon Bedrock 控制平面终端节点](#) 的请求发送 [GetCustomModel](#) (请求和响应格式以及字段详情参见链接) `modelIdentifier`，即以下任一方式。

- 你给模型起的名字。
- 模型的 ARN。

您可以在 [GetModelCustomizationJob](#) 或 [GetCustomModel](#) 响应中查看模型自定义任务的 `trainingMetrics` 和 `validationMetrics`。

要下载训练和验证指标文件，请按照[下载对象](#)中的步骤操作。使用您在中提供的 S3 URI `outputDataConfig`。

[参见代码示例](#)

## 使用自定义模型

在使用自定义模型之前，您需要为其购买预置吞吐量。有关预配置吞吐量的更多信息，请参阅[Amazon Bedrock 的预配置吞吐量](#)。然后，您可以使用生成的预配置模型进行推理。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

### Console

为自定义模型购买预调配吞吐量。

1. 在 Amazon Bedrock 控制台中，从左侧导航窗格中选择基础模型下的自定义模型。
2. 在模型选项卡中，选择要为其购买预配置吞吐量的型号旁边的单选按钮，或者选择型号名称以导航到详细信息页面。
3. 选择“购买预配置吞吐量”。
4. 有关更多详细信息，请按照中的步骤操作为 [Amazon Bedrock 模型购买预配置吞吐量](#)。
5. 为您的自定义模型购买预配置吞吐量后，请按照中的步骤操作。 [使用预配置吞吐量运行推理](#)

当您执行任何支持使用自定义模型的操作时，您将在模型选择菜单中看到您的自定义模型作为一个选项。

### API

要购买自定义模型的预配置吞吐量，请按照中的步骤使用 [Amazon Bedrock 控制平面](#) 终端节点发送 [CreateProvisionedModelThroughput](#) ( 请求和响应格式以及字段详情链接 ) 请求。为 [Amazon Bedrock 模型购买预配置吞吐量](#) 使用您的自定义模型的名称或 ARN 作为 `modelId` 响应会返回一个 `provisionedModelArn`，您可以在发出 [InvokeModel](#) 或 [InvokeModelWithResponseStream](#) 请求 `modelId` 时将其用作的。

[参见代码示例](#)

## 模型定制的代码示例

以下代码示例展示了如何准备基本数据集、设置权限、创建自定义模型、查看输出文件、购买模型吞吐量以及对模型运行推理。您可以根据自己的特定用例修改这些代码片段。

### 1. 准备训练数据集。

- a. 创建一个包含以下一行的训练数据集文件，并将其命名为 `train.jsonl#`

```
{"prompt": "what is AWS", "completion": "it's Amazon Web Services"}
```

- b. 为训练数据创建一个 S3 存储桶，为输出数据创建另一个 S3 存储桶（名称必须是唯一的）。
  - c. 将 `train.jsonl #####` 桶中。
- ### 2. 创建访问您的培训的策略，并将其附加到具有 Amazon Bedrock 信任关系的 IAM 角色。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

#### Console

### 1. 创建 S3 策略。

- a. 导航到 IAM 控制台 <https://console.aws.amazon.com/iam>，然后从左侧导航窗格中选择策略。
- b. 选择创建策略，然后选择 JSON 以打开策略编辑器。
- c. 粘贴以下策略，将 `${training-bucket}` 和 `${output-bucket}` 替换为您的存储桶名称，然后选择下一步。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${training-bucket}",
 "arn:aws:s3:::${training-bucket}/*"
]
 }
],
}
```

```

 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${output-bucket}",
 "arn:aws:s3:::${output-bucket}/*"
]
 }
]
}

```

- d. 命名策略 *MyFineTuningDataAccess* 并选择创建策略。
2. 创建 IAM 角色并附加策略。
    - a. 从左侧导航窗格中选择“角色”，然后选择“创建角色”。
    - b. 选择自定义信任策略，粘贴以下策略，然后选择下一步。

```

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

```

- c. 搜索您创建的 *MyFineTuningDataAccess* 策略，选中该复选框，然后选择下一步。
- d. 为角色命名 *MyCustomizationRole* 并选择####。

## CLI

1. 创建一个名为 *BedrockTrust.json* 的文件并将以下策略粘贴到其中。

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

2. 创建另一个名为 *MyFineTuningDataAccess.json* 的文件并将以下策略粘贴到其中，将 `${raining-bucket} # $ {output-bucket} #####`。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${training-bucket}",
 "arn:aws:s3:::${training-bucket}/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${training-bucket}",
 "arn:aws:s3:::${training-bucket}/*"
]
 }
]
}
```

```
]
}
```

3. 在终端中，导航到包含您创建的策略的文件夹。
4. [CreateRole](#)请求创建名为的 IAM 角色*MyCustomizationRole*并附加您创建的*BedrockTrust.json*信任策略。

```
aws iam create-role \
 --role-name MyCustomizationRole \
 --assume-role-policy-document file://BedrockTrust.json
```

5. 使用您创建的 *MyFineTuningDataAccess.js* on 文件[CreatePolicy](#)请求创建 S3 数据访问策略。该策略Arn的响应会返回一个。

```
aws iam create-policy \
 --policy-name MyFineTuningDataAccess \
 --policy-document file://myFineTuningDataAccess.json
```

6. [AttachRolePolicy](#)请求将 S3 数据访问策略附加到您的角色，将policy-arn替换为上一步响应中的 ARN：

```
aws iam attach-role-policy \
 --role-name MyCustomizationRole \
 --policy-arn #{policy-arn}
```

## Python

1. 运行以下代码，[CreateRole](#)请求创建名为的 IAM 角色*MyCustomizationRole*并[CreatePolicy](#)请求创建名为的 S3 数据访问策略*MyFineTuningDataAccess*。对于 S3 数据访问策略，请将 *#{training-bucket} # #{output-bucket} ##### S3 ###*名称。

```
import boto3
import json

iam = boto3.client("iam")

iam.create_role(
 RoleName="MyCustomizationRole",
 AssumeRolePolicyDocument=json.dumps({
```

```

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

iam.create_policy(
 PolicyName="MyFineTuningDataAccess",
 PolicyDocument=json.dumps({
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${training-bucket}",
 "arn:aws:s3:::${training-bucket}/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::${output-bucket}",
 "arn:aws:s3:::${output-bucket}/*"
]
 }
]
 })
)

```

```
)
```

2. 响应Arn中返回。运行以下代码片段[AttachRolePolicy](#)发出请求，将 `${policy-arn}` 替换为返回的。Arn

```
iam.attach_role_policy(
 RoleName="MyCustomizationRole",
 PolicyArn="${policy-arn}"
)
```

3. 选择一种语言以查看调用模型自定义 API 操作的代码示例。

## CLI

首先，创建一个名为 `FineTuningData.json` 的文本文件。将 JSON 代码从下方复制到文本文件中，将 `${training-bucket}` # `${output-bucket}` ##### `S3` ###名称。

```
{
 "trainingDataConfig": {
 "s3Uri": "s3://${training-bucket}/train.jsonl"
 },
 "outputDataConfig": {
 "s3Uri": "s3://${output-bucket}"
 }
}
```

要提交模型自定义作业，请在终端中导航到包含 `FineTuningData.json` 的文件夹，然后在命令行中运行以下命令，将 `${your-customization-role-arn}` 替换为您设置的模型自定义角色。

```
aws bedrock create-model-customization-job \
 --customization-type FINE_TUNING \
 --base-model-identifier arn:aws:bedrock:us-east-1::foundation-model/
amazon.titan-text-express-v1 \
 --role-arn ${your-customization-role-arn} \
 --job-name MyFineTuningJob \
 --custom-model-name MyCustomModel \
 --hyper-parameters
epochCount=1,batchSize=1,learningRate=.0005,learningRateWarmupSteps=0 \
 --cli-input-json file://FineTuningData.json
```



响应返回一个 *jobArn* 留出一些时间来完成任务。您可以使用以下命令检查其状态。

```
aws bedrock get-model-customization-job \
 --job-identifier "jobArn"
```

当status为时COMPLETE，你可以在响应trainingMetrics中看到。#####  
### *aet.et-bucket* ##### *jobID* ##### *ID*##### *jobArn*

```
aws s3 cp s3://{output-bucket}/model-customization-job-jobId . --recursive
```

使用以下命令为您的自定义模型购买无承诺预置吞吐量。

### Note

此次购买将按小时向您收费。使用控制台查看不同选项的估算价格。

```
aws bedrock create-provisioned-model-throughput \
 --model-id MyCustomModel \
 --provisioned-model-name MyProvisionedCustomModel \
 --model-units 1
```

响应返回 a *provisionedModelArn*。留出一段时间来创建预配置吞吐量。要检查其状态，请在以下命令*provisioned-model-id*中提供已配置模型的名称或 ARN。

```
aws bedrock get-provisioned-model-throughput \
 --provisioned-model-id {provisioned-model-arn}
```

如果status是InService，则可以使用以下命令对自定义模型进行推理。您必须提供预配置模型的 ARN 作为。model-id输出将写入当前文件夹中名为 *output.txt* 的文件中。

```
aws bedrock-runtime invoke-model \
 --model-id {provisioned-model-arn} \
 --body '{"inputText": "What is AWS?", "textGenerationConfig": {"temperature":
0.5}}' \
 --cli-binary-format raw-in-base64-out \
 output.txt
```

## Python

运行以下代码片段以提交微调作业。将 `${your-customization-role-arn}` 替换为您设置的 ARN，将 `${raining-bucket} # ${output-bucket} ##### S3 #####`。 `MyCustomizationRole`

```
import boto3
import json

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

Set parameters
customizationType = "FINE_TUNING"
baseModelIdentifier = "arn:aws:bedrock:us-east-1::foundation-model/amazon.titan-text-express-v1"
roleArn = "${your-customization-role-arn}"
jobName = "MyFineTuningJob"
customModelName = "MyCustomModel"
hyperParameters = {
 "epochCount": "1",
 "batchSize": "1",
 "learningRate": ".0005",
 "learningRateWarmupSteps": "0"
}
trainingDataConfig = {"s3Uri": "s3://${training-bucket}/myInputData/train.jsonl"}
outputDataConfig = {"s3Uri": "s3://${output-bucket}/myOutputData"}

Create job
response_ft = bedrock.create_model_customization_job(
 jobName=jobName,
 customModelName=customModelName,
 roleArn=roleArn,
 baseModelIdentifier=baseModelIdentifier,
 hyperParameters=hyperParameters,
 trainingDataConfig=trainingDataConfig,
 outputDataConfig=outputDataConfig
)

jobArn = response_ft.get('jobArn')
```

响应返回一个 `jobarn#` 留出一些时间来完成任务。您可以使用以下命令检查其状态。

```
bedrock.get_model_customization_job(jobIdentifier=jobArn).get('status')
```

当status为时COMPLETE，您可以在[GetModelCustomizationJob](#)响应trainingMetrics中看到。您也可以按照[下载对象中的](#)步骤下载指标。

使用以下命令为您的自定义模型购买无承诺预置吞吐量。

```
response_pt = bedrock.create_provisioned_model_throughput(
 modelId="MyCustomModel",
 provisionedModelName="MyProvisionedCustomModel"
 modelUnits="1"
)

provisionedModelArn = response_pt.get('provisionedModelArn')
```

响应返回 a provisionedModelArn。留出一段时间来创建预配置吞吐量。要检查其状态，请在以下命令provisionedModelId中提供已配置模型的名称或 ARN。

```
bedrock.get_provisioned_model_throughput(provisionedModelId=provisionedModelArn)
```

如果status是InService，则可以使用以下命令对自定义模型进行推理。您必须提供预配置模型的 ARN 作为。modelId

```
import json
import logging
import boto3

from botocore.exceptions import ClientError

class ImageError(Exception):
 "Custom exception for errors returned by the model"

 def __init__(self, message):
 self.message = message

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

```
def generate_text(model_id, body):
 """
 Generate text using your provisioned custom model.
 Args:
 model_id (str): The model ID to use.
 body (str) : The request body to use.
 Returns:
 response (json): The response from the model.
 """

 logger.info(
 "Generating text with your provisioned custom model %s", model_id)

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

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

 response = brt.invoke_model(
 body=body, modelId=model_id, accept=accept, contentType=content_type
)
 response_body = json.loads(response.get("body").read())

 finish_reason = response_body.get("error")

 if finish_reason is not None:
 raise ImageError(f"Text generation error. Error is {finish_reason}")

 logger.info(
 "Successfully generated text with provisioned custom model %s", model_id)

 return response_body

def main():
 """
 Entrypoint for example.
 """
 try:
 logging.basicConfig(level=logging.INFO,
 format="%(levelname)s: %(message)s")

 model_id = provisionedModelArn
```

```
body = json.dumps({
 "inputText": "what is AWS?"
})

response_body = generate_text(model_id, body)
print(f"Input token count: {response_body['inputTextTokenCount']}")

for result in response_body['results']:
 print(f"Token count: {result['tokenCount']}")
 print(f"Output text: {result['outputText']}")
 print(f"Completion reason: {result['completionReason']}")

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 your provisioned custom model
{model_id}.")

if __name__ == "__main__":
 main()
```

## 模型自定义准则

用于自定义模型的理想参数取决于数据集以及模型所要执行的任务。您应该对值进行试验，以确定哪些参数最适合您的具体情况。为了获取帮助，请运行模型评估作业来评估模型。有关更多信息，请参阅[模型评估](#)。

本主题提供指南和建议值，以此作为自定义 Amazon Titan Text G1 - Express 模型的基准。对于其他模型，请查看提供商的文档。

使用[提交](#)微调作业时生成的[输出文件](#)中的训练和验证指标来帮助您调整参数。在您写入输出的 Amazon S3 存储桶中找到这些文件，或者使用[GetCustomModel](#)操作。

## Amazon Titan Text G1 - Express

以下指南适用于 [Text Titan t Express](#) text-to-text 模型模型。有关可以设置的超参数的信息，请参阅 [Amazon Titan 文本模型自定义超参数](#)。

### 对其他任务类型的影响

通常，训练数据集越大，在特定任务上的性能就越好。不过，针对特定任务的训练可能会使模型在不同任务中的性能变差，尤其是在使用大量示例的情况下。例如，如果汇总任务的训练数据集包含 100,000 个样本，则模型在分类任务上的性能可能会较差。

### 模型大小

通常，对于给定的有限训练数据，模型越大，执行任务的性能就越好。

如果您将模型用于分类任务，则通过少样本微调（少于 100 个样本）实现的增益可能相对较小，尤其是在类数相对较少（少于 100 个）的情况下。

### 周期

我们建议使用以下指标来确定要设置的周期数：

1. 验证输出准确性 – 将周期数设置为可实现高准确性的值。
2. 训练和验证损失 – 确定训练和验证损失将在多少个周期后开始稳定。这与模型收敛时间相对应。  
在 `step_wise_training_metrics.csv` 和 `validation_metrics.csv` 文件中查找训练损失值。

### 批次大小

更改批次大小时，我们建议您使用以下公式更改学习率：

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

### 学习率

通常，对于较大的模型，请使用较小的学习率。我们建议使用位于 1.00E-06 到 1.00E-05 范围内的学习率。

下表显示了用于微调的推荐学习率值：

| 任务 | 最低学习率    | 默认学习率    | 最大学习率    |
|----|----------|----------|----------|
| 总结 | 1.00E-06 | 3.00E-06 | 5.00E-05 |
| 分类 | 5.00E-06 | 5.00E-05 | 5.00E-05 |
| 问答 | 5.00E-06 | 5.00E-06 | 5.00E-05 |

## 学习预热步骤

我们建议使用默认值 5。

## 故障排除

本部分总结了您可能遇到的错误以及遇到错误时需要检查的内容。

### 权限问题

如果您对 Amazon S3 存储桶的访问权限有问题，请检查以下内容是否正确：

1. 如果 Amazon S3 存储桶使用 CM-KMS 密钥进行服务器端加密，请确保传递给 Amazon Bedrock 的 IAM 角色拥有该密钥的 `kms:Decrypt` 权限。AWS KMS 例如，请参阅 [允许用户使用特定账户中的任何密 AWS KMS 钥进行加密和解密](#)。AWS
2. Amazon S3 存储桶与 Amazon Bedrock 模型自定义作业位于同一区域。
3. IAM 角色信任策略包括服务 SP (`bedrock.amazonaws.com`)。

以下消息表明对 Amazon S3 存储桶中训练或验证数据的访问权限有问题：

```
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
```

如果您遇到上述错误之一，请检查传递给服务的 IAM 角色是否对训练和验证数据集 Amazon S3 URI 拥有 `s3:GetObject` 和 `s3:ListBucket` 权限。

以下消息表明在 Amazon S3 存储桶中写入输出数据的权限有问题：

```
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
```

如果您遇到上述错误，请检查传递给服务的 IAM 角色是否对输出数据 Amazon S3 URI 拥有 s3:PutObject 权限。

## 数据问题

以下错误与训练、验证或输出数据文件的问题有关：

### 文件格式无效

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

如果您遇到上述错误，请检查以下内容是否正确：

1. 每行都采用 JSON 格式。
2. 每行 JSON 都有两个键，一个 `##` 键和一个 `##` 键，且每个键都是一个字符串。例如：

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

3. 没有其他新行或空行。

### 已超出字符配额

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

如果您遇到以上述文本开头的错误消息，请确保字符数符合 [模型自定义配额](#) 中的字符配额。

### 已超过令牌计数

```
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
```



如果您遇到与上述示例类似的错误，请确保词元数量符合[模型自定义配额](#)中的词元配额。

## 内部错误

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

如果您遇到上述错误，则可能是服务出现了问题。重试该作业。如果问题仍然存在，请与联系 AWS Support。

# Amazon Bedrock 的预配置吞吐量

吞吐量是指模型处理和返回的输入和输出的数量和速率。您可以购买预配置吞吐量，以固定成本为模型预置更高的吞吐量。如果您自定义了模型，则必须购买预配置吞吐量才能使用它。

您购买的预配置吞吐量按小时计费。有关定价的详细信息，请参阅 [Amazon Bedrock 定价](#)。每小时的价格取决于以下因素：

1. 您选择的型号（对于自定义型号，定价与其定制的基本型号相同）。
2. 您为预配置吞吐量指定的模型单位 (MU) 的数量。MU 为指定模型提供特定的吞吐量级别。MU 的吞吐量级别指定了以下内容：
  - MU 在一分钟内可以在所有请求中处理的输入令牌数量。
  - MU 在一分钟内可以在所有请求中生成的输出令牌的数量。

## Note

有关 MU 指定内容的更多信息，请联系您的 AWS 账户经理。

3. 您承诺保持预配置吞吐量的持续时间。承诺期限越长，每小时价格的折扣就越多。您可以在以下承诺级别之间进行选择：
  - 无需承诺 — 删除预配置吞吐量后，账单即告终止。
  - 1 个月-账单将在一个月后结束。在承诺期限结束之前，您无法删除预配置吞吐量。
  - 6 个月-账单将在六个月后结束。在承诺期限结束之前，您无法删除预配置吞吐量。

以下步骤概述了设置和使用预配置吞吐量的过程。

1. 确定您要为预配置吞吐量购买的 MU 数量，以及您要承诺使用预配置吞吐量的时间。
2. 为基本型号或自定义型号购买预置吞吐量。
3. 创建预配置模型后，您可以使用它来[运行模型推理](#)。

## 主题

- [预配置吞吐量支持的区域和型号](#)
- [先决条件](#)
- [为 Amazon Bedrock 模型购买预配置吞吐量](#)

- [管理预配置吞吐量](#)
- [使用预配置吞吐量运行推理](#)
- [Amazon Bedrock 中预配置吞吐量的代码示例](#)

## 预配置吞吐量支持的区域和型号

以下区域支持预置吞吐量：

| 区域                                    |
|---------------------------------------|
| 美国东部（弗吉尼亚州北部）                         |
| US West（Oregon）                       |
| AWS GovCloud（美国西部）<br>（仅适用于没有承诺的定制型号） |

如果您通过 Amazon Bedrock API 购买预配置吞吐量，则必须为型号 ID 指定 Amazon Bedrock FM 的上下文变体。下表显示了您可以购买预置吞吐量的模型、是否可以在不承诺购买基本型号的情况下购买以及购买预置吞吐量时要使用的模型 ID。

| 模型名称                                  | 基本型号支持无承诺购买 | 预置吞吐量的型号 ID                     |
|---------------------------------------|-------------|---------------------------------|
| Amazon Titan Text G1 - Express        | 支持          | 亚马逊。 titan-text-express-v1:08 k |
| Amazon Titan Text G1 - Lite           | 支持          | 亚马逊。 titan-text-lite-v1:0:4 k   |
| Amazon Titan Embeddings G1 - Text     | 支持          | 亚马逊。 titan-embed-text-v1:2:8 k  |
| Amazon Titan Multimodal Embeddings G1 | 支持          | 亚马逊。 titan-embed-image-v1:0     |
| Amazon Titan Image Generator G1       | 不支持         | 亚马逊。 titan-image-generator-v1:0 |

| 模型名称                           | 基本型号支持无承诺购买 | 预置吞吐量的型号 ID                                    |
|--------------------------------|-------------|------------------------------------------------|
| AnthropicClaudev2 18K          | 支持          | anthropic.claude-v2:0:18k                      |
| AnthropicClaudev2 100K         | 支持          | anthropic.claude-v2:0:100k                     |
| AnthropicClaudev2.1 18K        | 支持          | anthropic.claude-v2:1:18k                      |
| AnthropicClaudev2.1 200K       | 支持          | anthropic.claude-v 2:1:200k                    |
| AnthropicClaude 3 Sonnet28K    | 支持          | anthropic.claude-3-sonnet-20240229-v 1:0:28 k  |
| AnthropicClaude 3 Sonnet200K   | 支持          | anthropic.claude-3-sonnet-20240229-v 1:0:200 k |
| AnthropicClaude 3 Haiku48K     | 支持          | anthropic.claude-3-haiku-20240307-v 1:0:48 k   |
| AnthropicClaude 3 Haiku200K    | 支持          | anthropic.claude-3-haiku-20240307-v 1:0:200 k  |
| AnthropicClaude Instantv1 100K | 支持          | 人类。 claude-instant-v1:2:100 k                  |
| Cohere Command                 | 支持          | 一致。 command-text-v14:7:4 k                     |
| Cohere Command Light           | 支持          | 一致。 command-light-text-v14:7:4 k               |
| CohereEmbed英语                  | 支持          | 一致。 embed-english-v3:0:512                     |
| CohereEmbed多语言                 | 支持          | 一致。 embed-multilingual-v3:0:512                |
| Stable Diffusion XL 1.0        | 不支持         | 稳定性。 stable-diffusion-xl-v1:0                  |

| 模型名称                | 基本型号支持无承诺购买 | 预置吞吐量的型号 ID                     |
|---------------------|-------------|---------------------------------|
| MetaLlama 2 Chat13B | 不支持         | meta.llama2-13 1:0:4 k b-chat-v |
| MetaLlama 213B      | 不支持         | ( 参见下面的注释 )                     |
| MetaLlama 270B      | 不支持         | ( 参见下面的注释 )                     |

### Note

MetaLlama 2 ( 非聊天 ) 模型只能在[经过自定义并购买预配置吞吐量](#)后才能使用。

## 先决条件

在购买和管理预配置吞吐量之前，您需要满足以下先决条件：

1. [请求访问您要为其购买预置吞吐量的一个或多个模型](#)。授予访问权限后，您可以为基本模型以及从中自定义的任何模型购买预配置吞吐量。
2. 确保您的 IAM 角色具有执行与预配置吞吐量相关的操作[所需的权限](#)。
3. 如果您为使用客户管理 AWS KMS 密钥加密的自定义模型购买预配置吞吐量，则您的 IAM 角色必须具有解密密钥的权限。您可以在上使用该模板[创建密钥策略并将其附加到客户托管密钥](#)。对于最低权限，您只能使用#####策略声明。

## 为 Amazon Bedrock 模型购买预配置吞吐量

购买模型的预配置吞吐量时，需要指定其承诺级别和要分配的模型单位 (MU) 数量。有关 MU 配额的信息，请参阅[预调配吞吐量配额](#)。您可以为预配置吞吐量分配的 MU 数量取决于预配置吞吐量的承诺期限：

- 默认情况下，您的账户为您提供 2 个 MU，可在预配置吞吐量之间进行分配，无需承诺。
- 如果您要使用承诺购买预配置吞吐量，则必须先访问[AWS 支持中心](#)，为您的账户申请 MU，以便在预配置吞吐量与承诺之间进行分配。您的请求获得批准后，您可以通过承诺购买预配置吞吐量。

**Note**

购买预配置吞吐量后，只有选择自定义型号，才能更改关联的模型。您可以将关联的模型更改为以下模型之一：

- 它所依据的基本模型。
- 另一个源自同一个基础模型的自定义模型。

要了解如何为模型购买预置吞吐量，请选择与您选择的方法相对应的选项卡，然后按照步骤操作。

**Console**

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 在左侧导航窗格的“评估和部署”下选择“预配置吞吐量”。
3. 在预配置吞吐量部分，选择购买预配置吞吐量。
4. 要查看预置吞吐量详细信息部分，请执行以下操作：
  - a. 在预配置吞吐量名称字段中，输入预配置吞吐量的名称。
  - b. 在选择模型下，选择基础模型提供者或自定义模型类别。然后选择要为其预配置吞吐量的模型。

**Note**

要查看无需承诺即可购买预配置吞吐量的基本型号，请参阅[预配置吞吐量支持的区域和型号](#)。

在该 AWS GovCloud (US) 地区，您只能为自定义模型购买预置吞吐量，无需承诺。

- c. （可选）要将标签与您的预配置吞吐量相关联，请展开标签部分并选择添加新标签。有关更多信息，请参阅[标记资源](#)。
5. 在“承诺期限和模型单位”部分，请执行以下操作：
    - a. 在选择承诺期限部分，选择您要承诺使用预配置吞吐量的时间。
    - b. 在模型单位字段中，输入所需的模型单位 (MU) 数量。如果您要使用承诺配置模型，则必须先访问[AWS 支持中心](#)，申请增加可以购买的 MU 数量。

6. 在估算购买摘要下，查看估算成本。
7. 选择购买预调配吞吐量。
8. 查看所显示的备注，并通过选中复选框确认承诺期限和价格。然后选择确认购买。
9. 控制台显示预配置吞吐量概述页面。预配置吞吐量表中预置吞吐量的状态变为正在创建。预配置吞吐量创建完成后，状态变为服务中。如果更新失败，则状态变为“失败”。

## API

要购买预配置吞吐量，请使用 [Amazon Bedrock 控制平面](#) 终端节点发送 [CreateProvisionedModelThroughput](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。

### Note

要查看无需承诺即可购买预配置吞吐量的基本型号，请参阅[预配置吞吐量支持的区域和型号](#)。

在该 AWS GovCloud (US) 地区，您只能为自定义模型购买预置吞吐量，无需承诺。

下表简要描述了参数和请求正文（有关详细信息和请求结构，请参阅[CreateProvisionedModelThroughput 请求语法](#)）：

| Variable             | 必填？ | 应用场景                                                                              |
|----------------------|-----|-----------------------------------------------------------------------------------|
| modelId              | 支持  | 指定 <a href="#">用于购买预置吞吐量的基本型号 ID 或 ARN</a> ，或自定义型号名称或 ARN                         |
| 模型单位                 | 支持  | 指定要购买的模型单位 (MU) 的数量。要增加您可以购买的 MU 数量，请访问 <a href="#">AWS 支持中心</a> 申请增加您可以购买的 MU 数量 |
| provisionedModelName | 支持  | 为预配置吞吐量指定名称                                                                       |

| Variable           | 必填？ | 应用场景                          |
|--------------------|-----|-------------------------------|
| 承诺期限               | 不支持 | 指定提交预配置吞吐量的持续时间。省略此字段可选择无承诺定价 |
| tags               | 不支持 | 将标签与您的预配置吞吐量相关联               |
| clientRequestToken | 不支持 | 为了防止重复请求                      |

响应会返回provisionedModelArn一个可用作[模型modelId内推断的](#)。要检查预配置吞吐量何时可供使用，请发送[GetProvisionedModelThroughput](#)请求并检查状态是否为InService。如果更新失败，则其状态将为Failed，[GetProvisionedModelThroughput](#)响应中将包含failureMessage。

[参见代码示例](#)

## 管理预配置吞吐量

购买预配置吞吐量后，您可以查看其详细信息、更新或删除预配置吞吐量。

主题

- [查看有关预配置吞吐量的信息](#)
- [编辑预调配吞吐量](#)
- [删除预调配吞吐量](#)

### 查看有关预配置吞吐量的信息

要了解如何查看有关您已购买的预配置吞吐量的信息，请选择与您选择的方法对应的选项卡，然后按照步骤进行操作。



## Console

查看有关预配置吞吐量的信息

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 在左侧导航窗格的“评估和部署”下选择“预配置吞吐量”。
3. 从预配置吞吐量部分，选择预置吞吐量。
4. 在预配置吞吐量概述部分查看预配置吞吐量的详细信息，在标签部分查看与您的预配置吞吐量关联的标签。

## API

要检索有关特定预配置吞吐量的信息，请使用 [Amazon Bedrock 控制平面](#) 终端节点发送 [GetProvisionedModelThroughput](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。将预配置吞吐量的名称或其 ARN 指定为 `provisionedModelId`

要列出有关账户中所有预配置吞吐量的信息，请使用 [Amazon Bedrock 控制平面](#) 终端节点发送 [ListProvisionedModelThroughputs](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。要控制返回的结果数量，可以指定以下可选参数：

| 字段                      | 简短描述                                                                                                                       |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>maxResults</code> | 要在响应中返回的结果数量上限。                                                                                                            |
| <code>nextToken</code>  | 如果结果多于您在 <code>maxResults</code> 字段中指定的数字，则响应会返回一个 <code>nextToken</code> 值。要查看下一批结果，请在另一个请求中发送该 <code>nextToken</code> 值。 |

有关您可以指定的用于对结果进行排序和筛选的其他可选参数，请参阅 [GetProvisionedModelThroughput](#)。

要列出代理的所有标签，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [ListTagsForResource](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)），并附上预配置吞吐量的亚马逊资源名称 (ARN)。

[查看代码示例](#)

## 编辑预调配吞吐量

您可以编辑现有预配置吞吐量的名称或标签。

以下限制适用于更改与预配置吞吐量关联的模型：

- 您无法更改与基本模型关联的预配置吞吐量的模型。
- 如果预配置吞吐量与自定义模型相关联，则可以将关联更改为其自定义的基本模型，或者更改为从同一基础模型派生的另一个自定义模型。

当预配置吞吐量更新时，您可以使用预配置吞吐量进行推理，而不会中断来自最终客户的持续流量。如果您更改了与预配置吞吐量关联的模型，则在更新完全部署之前，您可能会收到旧模型的输出。

要了解如何编辑预配置吞吐量，请选择与您选择的方法对应的选项卡，然后按照步骤操作。

### Console

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。  
AWS Management Console
2. 在左侧导航窗格的“评估和部署”下选择“预配置吞吐量”。
3. 从预配置吞吐量部分，选择预置吞吐量。
4. 选择编辑。您可以编辑以下字段：
  - 预配置吞吐量名称-更改预配置吞吐量的名称。
  - 选择模型-如果预配置吞吐量与自定义模型相关联，则可以更改关联的模型。
5. 您可以在标签部分编辑与您的预配置吞吐量关联的标签。有关更多信息，请参阅 [标记资源](#)。
6. 要保存更改，请选择保存编辑内容。
7. 控制台显示预配置吞吐量概述页面。预配置吞吐量表中的预配置吞吐量的状态变为正在更新。当预配置吞吐量完成更新后，状态变为服务中。如果更新失败，则状态变为“失败”。

### API

要编辑预配置吞吐量，请使用 [Amazon Bedrock 控制平面](#) 终端节点发送 [UpdateProvisionedModelThroughput](#) 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。

下表简要描述了参数和请求正文（有关详细信息和请求结构，请参阅 [UpdateProvisionedModelThroughput 请求语法](#)）：

| Variable                  | 必填？ | 应用场景                                 |
|---------------------------|-----|--------------------------------------|
| provisionedModelId        | 支持  | 指定要更新的预配置吞吐量的名称或 ARN                 |
| desiredModelId            | 不支持 | 指定与预配置吞吐量关联的新模型（不适用于与基本模型关联的预配置吞吐量）。 |
| desiredProvisionedModel姓名 | 不支持 | 为预配置吞吐量指定新名称                         |

如果操作成功，则响应将返回 HTTP 200 状态响应。要检查预配置吞吐量何时可供使用，请发送 [GetProvisionedModelThroughput](#) 请求并检查状态是否为 InService。当预配置吞吐量的状态为时，您无法更新或删除该吞吐量。Updating 如果更新失败，则其状态将为 Failed，[GetProvisionedModelThroughput](#) 响应中将包含 failureMessage。

要向预配置吞吐量添加标签，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [TagResource](#) 请求（有关请求和响应格式以及字段详情，请参阅链接），并附上预配置吞吐量的亚马逊资源名称 (ARN)。请求正文包含一个 tags 字段，该字段是一个包含您为每个标签指定的键值对的对象。

要从预配置吞吐量中移除标签，请使用 [Amazon Bedrock 控制平面终端节点](#) 发送 [UntagResource](#) 请求（有关请求和响应格式以及字段详情，请参阅链接），并附上预配置吞吐量的亚马逊资源名称 (ARN)。tagKeys 请求参数是一个列表，其中包含您要删除的标签的密钥。

[查看代码示例](#)

## 删除预调配吞吐量

要了解如何删除预配置吞吐量，请选择与您选择的方法对应的选项卡，然后按照步骤操作。

### Note

在承诺期限结束之前，您不能删除带有承诺的预调配吞吐量。

## Console

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 在左侧导航窗格的“评估和部署”下选择“预配置吞吐量”。
3. 从预配置吞吐量部分，选择预置吞吐量。
4. 选择 Delete（删除）。
5. 控制台会显示一个模态表单，警告您删除是永久性的。选择确认继续。
6. 预配置吞吐量将立即删除。

## API

要删除预配置吞吐量，请使用 [Amazon Bedrock 控制平面](#) 终端节点发送 `DeleteProvisionedModelThroughput` 请求（有关请求和响应格式以及字段详情，请参阅[链接](#)）。将预配置吞吐量的名称或其 ARN 指定为 `provisionedModelId`。如果删除成功，则响应将返回 HTTP 200 状态码。

[查看代码示例](#)

## 使用预配置吞吐量运行推理

购买预配置吞吐量后，可以在模型推理中使用它来增加吞吐量。如果你愿意，你可以先在 Amazon Bedrock 控制台平台上测试预配置吞吐量。当您准备好部署预配置吞吐量时，您可以将应用程序设置为调用预配置模型。选择与您选择的方法相对应的选项卡，然后按照步骤操作。

## Console

在 Amazon Bedrock 控制台平台中使用预配置吞吐量

1. 登录并打开 Amazon Bedrock 控制台，[网址为 https://console.aws.amazon.com/bedrock/](https://console.aws.amazon.com/bedrock/)。AWS Management Console
2. 根据您的用例，在左侧导航窗格中，在 Playgroun ds 下选择“聊天”、“文本”或“图片”。
3. 选择“选择型号”。
4. 在 1. 类别列，选择提供者或自定义模型类别。然后，在 2 中。模型列中，选择与您的预配置吞吐量关联的模型。
5. 在 3 中。吞吐量列，选择您的预配置吞吐量。

## 6. 选择 应用。

要了解如何使用 Amazon Bedrock 游乐场，请参阅 [操场](#)

### API

要使用预配置吞吐量运行推理，请使用 [Amazon Bedrock](#) 运行时终端节点发送 [InvokeModel](#) 或 [InvokeModelWithResponseStream](#) 请求（有关请求和响应格式以及字段详情，请参阅 [链接](#)）。将 `modelId` 参数指定为预调配模型 ARN。要查看不同模型对请求正文的要求，请参阅 [根基模型的推理参数](#)。

[参见代码示例](#)

## Amazon Bedrock 中预配置吞吐量的代码示例

以下代码示例演示如何使用和 Python SDK 创建、使用和管理预配置吞吐量。AWS CLI

### AWS CLI

通过在终端中运行以下命令，创建一个 MyPT 基于名为的自定义模型调用的无承诺预配置吞吐量 `MyCustomModel`，该模型是从 Anthropic Claude v2.1 模型中自定义的。

```
aws bedrock create-provisioned-model-throughput \
 --model-units 1 \
 --provisioned-model-name MyPT \
 --model-id arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-v2:1:200k/
MyCustomModel
```

响应返回 `a provisioned-model-arn`。留出一些时间让创作完成。要检查其状态，请在以下命令 `provisioned-model-id` 中提供已配置模型的名称或 ARN。

```
aws bedrock get-provisioned-model-throughput \
 --provisioned-model-id MyPT
```

更改预配置吞吐量的名称，并将其与从 Anthropic Claude v2.1 自定义的其他模型相关联。

```
aws bedrock update-provisioned-model-throughput \
 --provisioned-model-id MyPT \
 --desired-provisioned-model-name MyPT2 \
 --model-id arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-v2:1:200k/
MyCustomModel
```

```
--desired-model-id arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-
v2:1:200k/MyCustomModel2
```

使用以下命令对更新的预配置模型运行推理。您必须提供UpdateProvisionedModelThroughput响应中返回的预配置模型的 ARN 作为。model-id 输出将写入当前文件夹中名为 *output.txt* 的文件中。

```
aws bedrock-runtime invoke-model \
 --model-id ${provisioned-model-arn} \
 --body '{"inputText": "What is AWS?", "textGenerationConfig": {"temperature":
0.5}}' \
 --cli-binary-format raw-in-base64-out \
 output.txt
```

使用以下命令删除预配置吞吐量。您将不再需要为预配置吞吐量付费。

```
aws bedrock delete-provisioned-model-throughput
 --provisioned-model-id MyPT2
```

## Python (Boto)

通过运行以下代码片段，创建一个MyPT基于名为的自定义模型调用的无承诺预配置吞吐量MyCustomModel，该模型是从 Anthropic Claude v2.1 模型中自定义的。

```
import boto3

bedrock = boto3.client(service_name='bedrock')
bedrock.create_provisioned_model_throughput(
 modelUnits=1,
 provisionedModelName='MyPT',
 modelId='arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-v2:1:200k/
MyCustomModel'
)
```

响应返回 a provisionedModelArn。留出一些时间让创作完成。您可以使用以下代码片段检查其状态。您可以提供预配置吞吐量的名称或响应中返回的 ARN [CreateProvisionedModelThroughput](#) 作为。provisionedModelId

```
bedrock.get_provisioned_model_throughput(provisionedModelId='MyPT')
```

更改预配置吞吐量的名称，并将其与从 Anthropic Claude v2.1 自定义的其他模型相关联。然后发送 [GetProvisionedModelThroughput](#) 请求并将已配置模型的 ARN 保存到变量中以用于推理。

```
bedrock.update_provisioned_model_throughput(
 provisionedModelId='MyPT',
 desiredProvisionedModelName='MyPT2',
 desiredModelId='arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-
v2:1:200k/MyCustomModel2'
)

arn_MyPT2 =
 bedrock.get_provisioned_model_throughput(provisionedModelId='MyPT2').get('provisionedModelArn')
```

使用以下命令对更新的预配置模型运行推理。您必须提供预配置模型的 ARN 作为 `modelId`

```
import json
import logging
import boto3

from botocore.exceptions import ClientError

class ImageError(Exception):
 "Custom exception for errors returned by the model"

 def __init__(self, message):
 self.message = message

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

def generate_text(model_id, body):
 """
 Generate text using your provisioned custom model.
 Args:
 model_id (str): The model ID to use.
 body (str) : The request body to use.
 Returns:
 response (json): The response from the model.
 """
```

```
logger.info(
 "Generating text with your provisioned custom model %s", model_id)

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

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

response = brt.invoke_model(
 body=body, modelId=model_id, accept=accept, contentType=content_type
)
response_body = json.loads(response.get("body").read())

finish_reason = response_body.get("error")

if finish_reason is not None:
 raise ImageError(f"Text generation error. Error is {finish_reason}")

logger.info(
 "Successfully generated text with provisioned custom model %s", model_id)

return response_body

def main():
 """
 Entrypoint for example.
 """
 try:
 logging.basicConfig(level=logging.INFO,
 format="%(levelname)s: %(message)s")

 model_id = arn_myPT2

 body = json.dumps({
 "inputText": "what is AWS?"
 })

 response_body = generate_text(model_id, body)
 print(f"Input token count: {response_body['inputTextTokenCount']}")

 for result in response_body['results']:
 print(f"Token count: {result['tokenCount']}")
 print(f"Output text: {result['outputText']}")
```



```
 print(f"Completion reason: {result['completionReason']}")

 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 your provisioned custom model
{model_id}.")

if __name__ == "__main__":
 main()
```

使用以下代码片段删除预配置吞吐量。您将不再需要为预配置吞吐量付费。

```
bedrock.delete_provisioned_model_throughput(provisionedModelId='MyPT2')
```

# 标记资源

为了帮助您管理 Amazon Bedrock 资源，您可以将元数据以标签形式分配给每个资源。标签是您分配给 AWS 资源的标签。每个标签均包含一个键和一个值。

标签使您能够以不同的方式对 AWS 资源进行分类，例如按用途、所有者或应用程序。标签可帮助您执行以下操作：

- 识别和整理您的 AWS 资源。许多 AWS 资源都支持标记，因此您可以为不同服务中的资源分配相同的标签，以表明这些资源是相同的。
- 分配成本。您可以在 AWS Billing and Cost Management 控制板上激活标签。AWS 使用标签对您的成本进行分类，并向您提供每月成本分配报告。有关更多信息，请参阅 AWS Billing and Cost Management 用户指南中的[使用成本分配标签](#)。
- 控制对资源的访问。您可以在 Amazon Bedrock 中使用标签来创建策略以控制对 Amazon Bedrock 资源的访问。这些策略可以附加到 IAM 角色或用户，以启用基于标签的访问控制。

您可以标记的 Amazon Bedrock 资源包括：

- 自定义模型
- 模型自定义作业
- 预调配模型
- 批量推理作业 ( 仅限 API )
- 代理
- 代理别名
- 知识库
- 模型评估 ( 仅限控制台 )

主题

- [使用控制台](#)
- [使用 API](#)
- [最佳实践和限制](#)

## 使用控制台

在创建或编辑支持的资源时，您可以随时添加、修改和删除标签。

## 使用 API

要执行标记操作，您需要想对其执行标记操作的资源的 Amazon 资源名称 (ARN)。有两组标记操作，具体取决于您要为其添加或管理标签的资源。

1. 以下资源使用 Amazon Bedrock [TagResourceUntagResource](#)、和 [ListTagsForResource](#) 操作。
  - 自定义模型
  - 模型自定义作业
  - 预调配模型
  - 批量推理作业
2. 以下资源使用适用于 Amazon Bedrock [TagResource](#) 的代理和 [ListTagsForResource](#) 操作。 [UntagResource](#)
  - 代理
  - 代理别名
  - 知识库

要为资源添加标签，请发送 Amazon Bedrock [TagResource](#) 或 Amazon Bedrock [TagResource](#) 代理请求。

要取消资源标签，请发送 [UntagResource](#) 或 [UntagResource](#) 请求。

要列出资源的标签，请发送 [ListTagsForResource](#) 或 [ListTagsForResource](#) 请求。

选择一个选项卡，查看界面或语言中的代码示例。

### AWS CLI

为代理添加两个标签。用空格分隔键/值对。

```
aws bedrock-agent tag-resource \
 --resource-arn "arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345" \
 --tags key=department,value=billing key=facing,value=internal
```

从代理中删除标签。用空格分隔键。

```
aws bedrock-agent untag-resource \
 --resource-arn "arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345" \
 --tag-keys key=department facing
```

列出代理的标签。

```
aws bedrock-agent list-tags-for-resource \
 --resource-arn "arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345"
```

## Python (Boto)

为代理添加两个标签。

```
import boto3

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

tags = [
 {
 'key': 'department',
 'value': 'billing'
 },
 {
 'key': 'facing',
 'value': 'internal'
 }
]

bedrock.tag_resource(resourceArn='arn:aws:bedrock:us-east-1:123456789012:agent/
AGENT12345', tags=tags)
```

从代理中删除标签。

```
bedrock.untag_resource(
 resourceArn='arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345',
 tagKeys=['department', 'facing']
)
```

列出代理的标签。

```
bedrock.list_tags_for_resource(resourceArn='arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345')
```

## 最佳实践和限制

有关标记的最佳做法和限制，请参阅为资源[添加 AWS 标签](#)。

# 亚马逊Titan模特

Amazon Titan 基础模型 (FM) 是一系列 FM，经过大型数据集的 AWS 预训练，因此它们是为支持各种用例而构建的强大通用模型。您可以按原样使用它们，也可以用自己的数据对其进行量身定制。

亚马逊Titan支持以下型号的 Amazon Bedrock。

- Amazon Titan 文本
- Amazon Titan Embeddings G1 - Text
- Amazon Titan Multimodal Embeddings G1
- 亚马逊Titan Image Generator G1 ( 预览 )

## Note

Amazon Titan Image Generator G1 现已发布公开预览版。

## 主题

- [Amazon Titan Text 模型](#)
- [亚马逊Titan Embeddings G1 - Text模型](#)
- [亚马逊Titan Multimodal Embeddings G1模型](#)
- [亚马逊Titan Image Generator G1模型](#)

## Amazon Titan Text 模型

亚马逊Titan文本模型包括亚马逊Titan Text G1 - Express和亚马逊Titan Text G1 - Lite。

### Amazon Titan Text G1 - Express

Amazon Titan Text G1 - Express 是一种用于生成文本的大型语言模型。它适用于各种高级通用语言任务，例如开放式文本生成和对话聊天，以及检索增强生成 (RAG) 内的支持。发布时，该模型针对英文版进行了优化，预览版中还提供 100 多种其他语言的多语言支持。

- 模型 ID – `amazon.titan-text-express-v1`

- 最大词元数 – 8K
- 语言 – 英语 ( GA 版 ) , 其他 100 种语言 ( 预览版 )
- 支持的用例 – 检索增强生成、开放式文本生成、头脑风暴、摘要提取、代码生成、表格创建、数据格式化、释义、思维链、重写、提取、问答和聊天。

## Amazon Titan Text G1 - Lite

Amazon Titan Text G1 - Lite 是一种轻量级高效的模型，非常适合微调英语任务，包括摘要和文案写作，在这些任务中，客户需要一种更小、更具成本效益且高度可定制化的模型。

- 模型 ID – `amazon.titan-text-lite-v1`
- 最大词元数 – 4K
- 语言 – 英语
- 支持的用例 – 开放式文本生成、头脑风暴、摘要提取、代码生成、表格创建、数据格式化、释义、思维链、重写、提取、问答和聊天。

## Amazon Titan 文本模型定制

有关自定义 Amazon Titan 文本模型的更多信息，请参阅以下页面。

- [准备数据集](#)
- [Amazon Titan 文本模型自定义超参数](#)

## Amazon Titan 短信提示工程指南

Amazon Titan 文本模型可用于各种不同用例的应用程序。Amazon Titan Text 模型具有适用于以下应用程序的即时工程指南，包括：

- Chatbot
- Text2SQL
- 函数调用
- RAG ( 检索增强生成 )

有关亚马逊Titan文本提示工程指南的更多信息，请参阅[亚马逊Titan文本提示工程指南](#)。

有关提示工程的一般性准则，请参阅[提示工程准则](#)。

## AWS AI 服务卡——A [amazon Titan Text](#)

AI Service Cards 提供了透明度，并记录了我们的 AWS AI 服务的预期用例和公平性注意事项。借助人工智能服务卡，您可以在一个位置查找有关预期用例、负责任的人工智能设计选择、最佳实践和一组人工智能服务用例性能的信息。

## 亚马逊Titan Embeddings G1 - Text模型

Amazon Em Titan beddings 文本模型包括亚马逊 Titan Embeddings G1 - Text G1。

文本嵌入表示非结构化文本（例如文档、段落和句子）的有意义的向量表示形式。您可以输入文本正文，输出为  $(1 \times n)$  向量。嵌入向量适用于各种应用程序。

亚马逊Titan Embeddings G1 - Text模型 (`amazon.titan-embed-text-v1`)。Amazon Titan Embeddings G1 - Text — Text v1.2 最多可以吸收 8k 个代币，并输出一个 1,536 个维度的向量。该模型还支持 25 种以上的不同语言。该模型针对文本检索任务进行了优化，但也可以执行其他任务，例如语义相似度和聚类。Amazon Titan Embeddings G1 - Text — Text v1.2 也支持长文档，但是，对于检索任务，建议将文档分成逻辑段（例如段落或部分）。请采纳我们的建议。

### Note

Titan Embeddings G1 - Text模型不支持推理参数，例如`maxTokenCount`或`topP`。

要使用文本或图像嵌入模型，请使用 `Invoke Model API` 操作（将 `amazon.titan-embed-text-v1` 或 `amazon.titan-embed-image-v1` 当作 `model Id`）并在响应中检索嵌入对象。

查看 Jupyter 笔记本示例：

1. 通过以下网址登录 Amazon Bedrock 控制台：<https://console.aws.amazon.com/bedrock/home>。
2. 从左侧菜单中，选择基础模型。
3. 向下滚动并选择 Amazon Titan Embeddings G1 - Text 型号
4. 在 Amazon Titan Embeddings G1 - Text 选项卡（取决于您选择的型号）中，选择查看示例笔记本以查看嵌入式示例笔记本。

有关为多模态训练准备数据集的更多信息，请参阅[准备数据集](#)。



## 亚马逊Titan Multimodal Embeddings G1模型

Amaz Titan Multimodal Embeddings G1 on Generation 1 (G1) 是一种多模式嵌入模型，适用于通过文本、图像或文本和图像组合搜索图像等用例。该模型专为高准确性和快速响应设计，是搜索和推荐用例的理想选择。

- 模型 ID – `amazon.titan-embed-image-v1`
- 最大输入文本词元数 – 128
- 语言 – 英语
- 最大输入图像大小 – 5 MB
- 输出向量大小 – 1,024 (默认)、384、256
- 推理类型 – 按需吞吐量、预调配吞吐量
- 支持的用例 – 图像搜索、推荐和个性化

### 嵌入长度

可以选择是否设置自定义嵌入长度。嵌入的默认长度为 1024 个字符，适用于大多数用例。嵌入长度可以设置为 256、384 或 1024 个字符。较大的嵌入尺寸会生成更详细的响应，但也会增加计算时间。较短的嵌入长度生成的响应不那么详细，但会缩短响应时间。

```
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
 }
})
```

## 微调

- Amazon Titan Multimodal Embeddings G1 微调的输入是图像-文本对。
- 图像格式：PNG、JPEG
- 输入图像大小限制：5 MB
- 图像尺寸：最小 128 像素，最大 4,096 像素
- 字幕中的最大词元数：128
- 训练数据集大小范围：1000 - 500,000
- 验证数据集大小范围：8 - 50,000
- 字幕长度（以字符为单位）：0 - 2,560
- 每个图像的最大总像素数：2048\*2048\*3
- 纵横比 (w/h)：最小 0.25，最大 4

## 准备数据集

对于训练数据集，请创建一个包含多个 JSON 行的 .jsonl 文件。每个 JSON 行都包含一个 image-ref 和多个 caption 属性，类似于 [Sagemaker 增强清单格式](#)。需要一个验证数据集。当前不支持自动字幕功能。

```
{"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"}
```

对于训练数据集和验证数据集，需要创建一个包含多个 JSON 行的 .jsonl 文件。

Amazon S3 路径所在的文件夹，需要与您通过将 IAM 策略附加到 Amazon Bedrock 服务角色，以便向 Amazon Bedrock 提供数据访问权限所在的文件夹相同。有关授予 IAM 策略以访问训练数据的更多信息，请参阅[为自定义作业授予访问训练数据的权限](#)。

## 超参数

可以针对 Multimodal Embeddings 模型的超参数调整这些值。默认值适用于大多数用例。

- 学习速率 - (最小/最大学习速率) – 默认：5.00E-05，最小：5.00E-08，最大：1

- 批量大小 - 有效批量大小 – 默认：576，最小：256，最大：9,216
- 最大周期数 – 默认：“auto”，最小：1，最大：100

## 亚马逊Titan Image Generator G1模型

Amazon Titan Image Generator G1 是一种图像生成模型。它根据文本生成图像，并允许用户上传和编辑现有图像。用户可以使用文本提示编辑图像（不使用掩膜），也可以使用图像掩膜编辑图像的某些部分。您可以使用扩绘功能来扩展图像的边界，也可以用补绘功能来填充图像。它还能根据可选的文本提示生成图像的变体。Amazon Titan Image Generator G1 on Generator 在输出文件上添加了水印。

### Note

Amazon 目前Titan Image Generator G1处于预览版。不建议将其用于生产工作。某些功能可能无法按预期运行。图像和输出数据产生的结果可能不够准确。

有关亚马逊Titan Image Generator G1提示工程指南的更多信息，请参阅[亚马逊Titan Image Generator G1提示工程最佳实践](#)。

- 模型 ID – amazon.titan-image-generator-v1
- 最大输入字符数 – 1,024 个字符
- 最大输入图像大小 – 50 MB ( 仅支持某些特定的分辨率 )
- 使用补绘/扩绘的最大图像尺寸 – 1,024 x 1,024 像素
- 使用图像变体的最大图像尺寸 – 4,096 x 4,096 像素
- 语言 – 英语
- 输出类型 – 图像
- 支持的图像类型 – JPEG、JPG、PNG
- 推理类型 – 按需吞吐量、预调配吞吐量
- 支持的用例 – 图像生成、图像编辑、图像变体

## 功能

- Text-to-image (T2I) 生成-输入文本提示并生成新图像作为输出。生成的图像捕捉了文本提示所描述的概念。

- T2I 模型的微调 – 导入多个图像以捕捉自己的风格和个性化，然后微调核心 T2I 模型。经过微调的模型将生成符合特定用户风格和个性化的图像。
- 图像编辑选项 – 包括补绘、扩绘、生成变体以及不使用图像掩膜的自动编辑。
- 补绘 – 使用图像和分割掩膜作为输入（来自用户或由模型估计），并重建掩膜内的区域。可使用补绘功能移除掩膜元素并将其替换为背景像素。
- 扩绘 – 使用图像和分割掩膜作为输入（来自用户或由模型估计），并生成可无缝扩展区域的新像素。将图像扩展到边界时，可使用精准的扩绘功能来保留被掩膜遮盖图像的像素。根据分割设置，使用默认的扩绘功能将被掩膜遮盖图像的像素扩展到图像边界。
- 图像变体 – 使用图像和可选提示作为输入。它会生成一个新图像，该图像将保留输入图像的内容，但会改变其样式和背景。

### Note

如果您使用的是经过微调的模型，则不能使用 API 或模型的修复或外涂功能。

## 参数

有关 Amazon Titan Image Generator G1 推理参数的信息，请参阅[亚马逊 Titan Image Generator G1 推理参数](#)。

## 微调

有关微调 Amazon Titan Image Generator G1 模型的更多信息，请参阅以下页面。

- [准备数据集](#)
- [Amazon Titan Image Generator G1 模型自定义超参数](#)

### Titan Image Generator G1 微调和定价

该模型使用以下示例公式来计算每份工作的总价格：

总价格 = 步数 \* 批次大小 \* 每张图片的价格

最小值 ( auto ) ：

- 最小步数 (auto)-500

- 最小批量大小-8
- 默认学习率-0.00001
- 每张图片的价格-0.005

## 微调超参数设置

**步骤-模型在每个批次中暴露的次数。** 没有设置默认的步数。您必须选择一个介于 10-40,000 之间的数字，或者选择“自动”的字符串值。

**步骤设置-自动** — Amazon Bedrock 根据训练信息确定合理的值。选择此选项可优先考虑模型性能而不是训练成本。步数是自动确定的。根据您的数据集，此数字通常介于 1,000 到 8,000 之间。Job 成本受用于向数据公开模型的步骤数的影响。请参阅定价详情的定价示例部分，了解工作成本是如何计算的。（参见上面的示例表，了解选择“自动”时步数与图像数量的关系。）

**步骤设置-自定义**- 您可以输入希望 Bedrock 向训练数据公开自定义模型的步数。此值可以介于 10 到 40,000 之间。您可以使用较低的步数值来降低模型生成的每张图像的成本。

**Batch size-更新模型参数之前处理的样本数量。** 该值介于 8 和 192 之间，是 8 的倍数。

**学习率-每批训练数据之后模型参数的更新速率。** 这是一个介于 0 和 1 之间的浮点值。默认情况下，学习率设置为 0.00001。

有关微调程序的更多信息，请参阅[提交模型定制任务](#)。

## 输出

Titan Image Generator G1使用输出图像的大小和质量来确定图像的定价方式。Titan Image Generator G1有两个基于尺寸的定价区段：一个用于 512\*512 张图片，另一个用于 1024\*1024 张图片。价格是根据图像尺寸（高度\*宽度）确定的，是小于、等于 512\*512 还是大于 512\*512。

有关亚马逊 Bedrock 定价的更多信息，请参阅[亚马逊 Bedrock 定价](#)。

## 水印检测

### Note

水印检测处于公开预览版中，只能检测从中生成的水印Titan Image Generator G1。此功能目前仅在 PDX 和 IAD 区域可用。水印检测是对生成的水印的高精度检测Titan Image Generator G1。修改原始图像的图像可能会产生不太准确的检测结果。

该模型为所有生成的图像添加了不可见的水印，以减少错误信息的传播，协助版权保护并跟踪内容的使用情况。将来，将提供水印检测 API 来帮助您确认图像是否由模型生成，该 Titan Image Generator G1 模型会检查该水印是否存在。

您可以上传图像以检测图像上 Titan Image Generator G1 是否存在来自的水印。按照以下步骤，使用控制台检测此模型的水印。

要检测水印，请执行 Titan Image Generator G1 以下操作：

1. 在 [Amazon Bedrock 控制台](#) 中打开 Amazon Bedrock 控制台
2. 从 Amazon Bedrock 的导航窗格中选择“概览”。选择“生成和测试”选项卡。
3. 在“安全措施”部分中，转到“水印检测”，然后选择“查看水印检测”。
4. 选择“上传图片”，然后找到 JPG 或 PNG 格式的文件。允许的最大文件大小为 5 MB。
5. 上传后，将显示图像的缩略图，其中包含名称、文件大小和上次修改日期。选择 X 可从“上传”部分删除或替换图片。
6. 选择“分析”开始水印检测分析。
7. 图像在“结果”下预览，并指示是否检测到水印，图像下方检测到水印，图像上有横幅。如果未检测到水印，则图像下方的文字将显示“未检测到水印”。
8. 要加载下一张图片，请在“上传”部分的图像缩略图中选择 X，然后选择要分析的新图片。

## 提示工程准则

**掩膜提示** – 此算法将把像素划分为不同的概念。用户可提供一个文本提示，系统将根据对掩膜提示的解释，将图像的区域进行分类以使用掩膜遮盖。提示选项可以解释更复杂的提示，并将掩膜编码到分割算法中。

**图像掩膜** – 也可以使用图像掩膜来设置掩膜值。可以将图像掩膜与掩膜的提示输入结合起来以提高准确性。图像掩膜文件必须遵守以下参数要求：

- 掩膜图像的值必须为 0 (黑色) 或 255 (白色)。值为 0 的图像掩膜区域将使用用户提示和/或输入图像中的图像重新生成。
- maskImage 字段必须为 base64 编码的图像字符串。
- 掩膜图像的尺寸必须与输入图像的尺寸相同 (高度和宽度相同)。
- 只能使用 PNG 或 JPG 文件作为输入图像和掩膜图像。
- 掩膜图像只能使用黑白像素值。
- 掩膜图像只能使用 RGB 通道 (不支持 Alpha 通道)。

有关亚马逊Titan Image Generator G1提示工程的更多信息，请参阅[亚马逊Titan Image Generator G1提示工程最佳实践](#)。

有关提示工程的一般性准则，请参阅[提示工程准则](#)。

# Amazon Bedrock 中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方 AWS 的共同责任。[责任共担模式](#)将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云中运行 AWS 服务的基础架构 AWS Cloud。AWS 还为您提供可以安全使用的服务。作为[AWS 合规计划](#)的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于 Amazon Bedrock 的合规计划，请参阅按合规计划提供的[范围内的 AWS 服务按合规计划](#)分的范围内服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您的公司的要求以及适用的法律法规。

此文档可帮助您了解如何在使用 Amazon Bedrock 时应用责任共担模式。以下主题介绍如何配置 Amazon Bedrock 以实现您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon Bedrock 资源。

## 主题

- [数据保护](#)
- [适用于 Amazon Bedrock 的身份和访问管理](#)
- [Amazon Bedrock 的合规性验证](#)
- [Amazon Bedrock 中的事件响应](#)
- [Amazon Bedrock 中的故障恢复能力](#)
- [Amazon Bedrock 中的基础设施安全性](#)
- [防止跨服务混淆座席](#)
- [Amazon Bedrock 中的配置和漏洞分析](#)
- [使用接口 VPC 端点 \(AWS PrivateLink\)](#)

## 数据保护

AWS [分担责任模式](#)适用于 Amazon Bedrock 中的数据保护。如本模型所述 AWS，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。您



还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅[数据隐私常见问题解答](#)。有关欧洲数据保护的信息，请参阅 AWS 安全性博客上的 [AWS 责任共担模式和 GDPR 博客文章](#)。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与资源通信。AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅 [《美国联邦信息处理标准 \( FIPS \) 第 140-2 版》](#)。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您 AWS 服务 使用控制台、API 或 AWS 软件开发工具包与 Amazon Bedrock 或其他 AWS CLI 机构合作的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

## Amazon Bedrock 中的数据保护

Amazon Bedrock 不会使用您的提示和延续来训练任何 AWS 模型或将其分发给第三方。

每个模型提供商都有一个托管账户，他们可以将模型上传到该账户。Amazon Bedrock 推理账户有权调用这些模型，但托管账户本身对 Amazon Bedrock 账户没有出站权限。此外，模型提供商无法访问 Amazon Bedrock 日志，也无权访问客户提示和延续。

Amazon Bedrock 不会在其服务日志中存储或记录您的数据。

## Amazon Bedrock 模型定制中的数据保护

您的训练数据不会用于训练基础 Titan 模型或分发给第三方。其他使用数据（例如使用时间戳、记录的账户 ID 以及服务记录的其他信息）也不会用于训练模型。

Amazon Bedrock 仅使用您提供的微调数据来微调亚马逊 Bedrock 基础模型。Amazon Bedrock 不会将微调数据用于任何其他用途，例如，训练基础根基模型。

Bedrock 在[CreateModelCustomizationJob](#)操作或[控制台](#)中使用您的训练数据来创建自定义模型，该模型是 Amazon Bedrock 基础模型的微调版本。您的自定义模型由管理和存储 AWS。默认情况下，自定义模型使用 AWS 拥有的 AWS Key Management Service 密钥进行加密，但您可以使用自己的 AWS KMS 密钥对自定义模型进行加密。当您通过控制台或以编程方式使用 `CreateModelCustomizationJob` 操作提交微调作业时，可以对自定义模型进行加密。

微调作业完成后，您为进行微调而提供的训练或验证数据都不会存储在 Amazon Bedrock 账户中。训练期间，您的数据存在于 AWS 服务管理连接器 实例内存中，但是在这些计算机上，会使用实例本身上的硬件模块中实施的 AES-256 密码来为这些数据加密。

我们不建议使用机密数据来训练自定义模型，因为该模型可能会根据这些机密数据生成推理响应。如果您使用机密数据来训练自定义模型，则防止基于这些数据做出响应的唯一方法是删除自定义模型，从训练数据集中删除机密数据，然后重新训练自定义模型。

自定义模型元数据（名称和 Amazon 资源名称）和预调配模型的元数据存储于 Amazon DynamoDB 表中，对于该表，使用 Amazon Bedrock 服务拥有的密钥进行了加密。

## 主题

- [数据加密](#)
- [使用 Amazon VPC 保护您的数据以及 AWS PrivateLink](#)

## 数据加密

Amazon Bedrock 使用加密来保护静态数据和传输中数据。

## 主题

- [传输中加密](#)
- [静态加密](#)
- [密钥管理](#)
- [对模型自定义任务和构件进行加密](#)
- [代理资源的加密](#)
- [知识库资源的加密](#)

## 传输中加密

在内部 AWS，所有传输中的网络间数据都支持 TLS 1.2 加密。

通过安全 (SSL) 连接发出对 Amazon Bedrock API 和控制台的请求。您将 AWS Identity and Access Management (IAM) 角色传递给 Amazon Bedrock，以提供代表您访问资源以进行培训和部署的权限。

## 静态加密

Amazon Bedrock 提供静态[对模型自定义任务和构件进行加密](#)。

## 密钥管理

使用 AWS Key Management Service 来管理用于加密资源的密钥。有关更多信息，请参阅 [AWS Key Management Service 概念](#)。您可以使用 KMS 密钥来加密以下资源。

- 通过 Amazon Bedrock
  - 模型自定义任务及其输出自定义模型-在控制台中创建任务期间或通过 [CreateModelCustomizationJob](#) API 调用中指定 `customModelKmsKeyId` 字段。
  - 代理-在控制台中创建代理期间，或者在 [CreateAgent](#) API 调用中指定字段。
  - 知识库的数据源提取作业-在控制台中创建知识库期间，或者通过在 [CreateDataSource](#) 或 [UpdateDataSource](#) API 调用中指定 `kmsKeyArn` 字段。
  - Amazon S OpenSearch ervice 中的矢量存储-在创建矢量商店期间。有关更多信息，请参阅 [创建、列出和删除亚马逊 OpenSearch 服务馆藏](#) 以及 [加密亚马逊 OpenSearch 服务的静态数据](#)。
- 通过 Amazon S3 — 有关更多信息，请参阅 [使用带 AWS KMS 密钥的服务器端加密 \(SSE- KMS\)](#)。
  - 用于模型自定义的训练、验证和输出数据
  - 知识库的数据来源
- 通过 AWS Secrets Manager — 有关更多信息，请参阅中的 [秘密加密和解密 AWS Secrets Manager](#)
  - 第三方模型的向量存储

加密资源后，您可以通过选择资源并在控制台中查看其详细信息或使用以下 Get API 调用来查找 KMS 密钥的 ARN。

- [GetModelCustomizationJob](#)
- [GetAgent](#)
- [GetIngestionJob](#)

## 对模型自定义任务和构件进行加密

默认情况下，Amazon Bedrock 使用 AWS 托管密钥对模型自定义任务中的以下模型工件进行加密。

- 模型定制任务
- 模型自定义作业的输出文件（训练和验证指标）
- 生成的自定义模型

或者，您可以通过创建客户托管密钥来加密模型工件。有关更多信息 AWS KMS keys，请参阅《AWS Key Management Service 开发人员指南》中的[客户托管密钥](#)。要使用客户管理的密钥，请执行以下步骤。

1. 使用创建客户托管密钥 AWS Key Management Service。
2. 附加[基于资源的策略](#)，允许指定角色创建或使用自定义模型。

### 主题

- [创建客户托管密钥](#)
- [创建密钥策略并将其附加到客户托管密钥](#)
- [对训练、验证和输出数据进行加密](#)

### 创建客户托管密钥

首先，请确保您拥有CreateKey权限。然后按照[创建密钥](#)中的步骤在 AWS KMS 控制台或 [CreateKey](#) API 操作中创建客户托管密钥。确保创建对称加密密钥。

创建密钥时会返回密钥Arn的值，您可以在[提交模型自定义任务customModelKmsKeyId](#)时将其用作密钥。

### 创建密钥策略并将其附加到客户托管密钥

按照[创建密钥策略中的步骤](#)，将以下基于资源的策略附加到 KMS 密钥。该策略包含两个声明。

1. 角色对模型自定义构件进行加密的权限。将自定义模型构建者角色的 ARN 添加到该Principal字段。
2. 角色在推理中使用自定义模型的权限。将自定义模型用户角色的 ARN 添加到该Principal字段。

```
{
```

```
"Version": "2012-10-17",
"Id": "KMS Key Policy",
"Statement": [
 {
 "Sid": "Permissions for custom model builders",
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::account-id:user/role"
 },
 "Action": [
 "kms:Decrypt",
 "kms:GenerateDataKey",
 "kms:DescribeKey",
 "kms:CreateGrant"
],
 "Resource": "*"
 },
 {
 "Sid": "Permissions for custom model users",
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::account-id:user/role"
 },
 "Action": "kms:Decrypt",
 "Resource": "*"
 }
]
```

## 对训练、验证和输出数据进行加密

当您使用 Amazon Bedrock 运行模型自定义任务时，您将输入（训练/验证数据）文件存储在您的 Amazon S3 存储桶中。任务完成后，Amazon Bedrock 会将输出指标文件存储在您在创建任务时指定的 S3 存储桶中，并将生成的自定义模型项目存储在由控制的 Amazon S3 存储桶中。AWS

默认情况下，输入和输出文件使用 Amazon S3 SSE-S3 服务器端加密进行加密，AWS 托管式密钥使用。此类密钥由代表您创建、管理和使用 AWS。

相反，您可以选择使用您自己创建、拥有和管理的客户托管密钥来加密这些文件。请参阅前面的章节和以下链接，了解如何创建客户托管密钥和密钥策略。

- 要了解有关 Amazon S3 SSE-S3 服务器端加密的更多信息，请参阅对[亚马逊 S3 托管密钥使用服务器端加密 \(SSE-S3\)](#)

- 要详细了解用于加密 S3 对象的客户托管密钥，请参阅[使用 KMS 密钥的服务器端加密 \(SSE-AWS KMS\)](#)

## 代理资源的加密

Amazon Bedrock 会加密代理的会话信息。默认情况下，Amazon Bedrock 使用 AWS 托管密钥对这些数据进行加密。或者，您可以使用客户托管式密钥来加密代理构件。

有关更多信息 AWS KMS keys，请参阅《AWS Key Management Service 开发人员指南》中的[客户托管密钥](#)。

如果您使用自定义 KMS 密钥对与代理的会话进行加密，则必须设置以下基于身份的策略和基于资源的策略，以允许 Amazon Bedrock 代表您加密和解密代理资源。

1. 将以下基于身份的策略附加到具有 InvokeAgent 调用权限的 IAM 角色或用户。此策略会验证发出 InvokeAgent 调用的用户是否具有 KMS 权限。将 *region*、*account-id*、*agent-id* 和 *key-id* 替换为相应值。

```
{
 "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. 将以下基于资源的策略附加到 KMS 密钥。根据需要更改权限的范围。将 *region*、*account-id*、*agent-id* 和 *key-id* 替换为相应值。

```
{
 "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:*",
 "Resource": "arn:aws:kms:region:account-id:key/key-id"
 },
 {
 "Sid": "Allow Amazon Bedrock to encrypt and decrypt Agent resources on behalf of authorized users",
 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 },
 "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"
 }
 }
 }
]
}
```

## 知识库资源的加密

Amazon Bedrock 会加密与知识库相关的资源。默认情况下，Amazon Bedrock 使用 AWS 托管密钥对这些数据进行加密。或者，您可以使用客户托管式密钥对模型构件进行加密。

支持使用 KMS 密钥进行加密的过程如下：

- 提取数据来源时的临时数据存储
- 如果您让 Amazon Bedrock 设置矢量数据库，则会将信息传递给 OpenSearch 服务部门
- 查询知识库

知识库使用的下列资源也可以使用 KMS 密钥进行加密。如果加密这些资源，则需要添加权限以解密 KMS 密钥。

- 存储在 Amazon S3 存储桶中的数据来源
- 第三方向量存储

有关更多信息 AWS KMS keys，请参阅《AWS Key Management Service 开发人员指南》中的[客户托管密钥](#)。

## 主题

- [加密数据提取期间的临时数据存储](#)
- [对传递给 Amazon OpenSearch 服务的信息进行加密](#)
- [加密知识库检索](#)
- [解密您在 Amazon S3 中的数据源的 AWS KMS 密钥的权限](#)
- [解密包含知识库的矢量存储的 AWS Secrets Manager 密钥的权限](#)

## 加密数据提取期间的临时数据存储

在为知识库设置数据提取作业时，可以使用自定义 KMS 密钥加密作业。

要允许在摄取数据源的过程中为临时数据存储创建 AWS KMS 密钥，请将以下策略附加到您的 Amazon Bedrock 服务角色。将 *region*、*account-id* 和 *key-id* 替换为适当的值。

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



```

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

```

对传递给 Amazon OpenSearch 服务的信息进行加密

如果您选择让 Amazon Bedrock 在亚马逊 OpenSearch 服务中为您的知识库创建向量存储，Amazon Bedrock 可以将您选择的 KMS 密钥传递给亚马逊 OpenSearch 服务进行加密。要了解有关亚马逊 OpenSearch 服务加密的更多信息，请参阅[亚马逊 OpenSearch 服务中的加密](#)。

### 加密知识库检索

您可以使用 KMS 密钥加密在查询知识库时生成响应的会话。为此，请在发出请求时在 `kmsKeyArn` 字段中包含 KMS 密钥的 ARN。[RetrieveAndGenerate](#) 附加以下策略，根据情况替换 *values* 以允许 Amazon Bedrock 加密会话上下文。

```

{
 "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"
 }
]
}

```

### 解密您在 Amazon S3 中的数据源的 AWS KMS 密钥的权限

您将知识库的数据来源存储在 Amazon S3 存储桶中。要对这些静态文档进行加密，您可以使用 Amazon S3 SSE-S3 服务器端加密选项。通过此选项，将使用由 Amazon S3 服务管理的服务密钥为对象加密。

有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的[使用采用 Amazon S3 托管加密密钥的服务器端加密 \(SSE-S3\) 保护数据](#)。

如果您使用自定义 AWS KMS 密钥加密了 Amazon S3 中的数据源，请将以下策略附加到您的 Amazon Bedrock 服务角色，以允许 Amazon Bedrock 解密您的密钥。将 *region* 和 *account-id* 替换为密钥所属的区域和账户 ID。将 *key-id* 替换为密钥的 ID AWS KMS。

```
{
 "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"
]
 }
 }
]
}
```

解密包含知识库的矢量存储的 AWS Secrets Manager 密钥的权限

如果包含知识库的矢量存储配置了 AWS Secrets Manager 密钥，则可以按照中的密钥加密和[解密中的步骤使用自定义密 AWS KMS 钥对密钥进行加密](#)。AWS Secrets Manager

如果您这样做，请将以下策略附加到 Amazon Bedrock 服务角色，以允许它将密钥解密。将 *region* 和 *account-id* 替换为密钥所属的区域和账户 ID。将 *key-id* 替换为密钥的 ID AWS KMS。

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

```

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

```

## 使用 Amazon VPC 保护您的数据以及 AWS PrivateLink

为了控制对数据的访问，我们建议您使用带有 [Amazon VPC 的虚拟私有云 \(VPC\)](#)。使用 VPC 可以保护您的数据，并允许您使用 [VPC 流日志监控进出 AWS 任务容器的所有网络流量](#)。您可以进一步保护您的数据，方法是配置您的 VPC，使您的数据无法通过 Internet 获取，然后使用创建 VPC 接口终端节点 [AWS PrivateLink](#) 来建立与您的数据的私有连接。

有关使用 VPC 保护您与 Amazon Bedrock 集成的数据的示例，请参阅 [使用 VPC 保护模型自定义任务](#)。

### 使用接口 VPC 端点 (AWS PrivateLink)

您可以使用 AWS PrivateLink 在您的 VPC 和 Amazon Bedrock 之间创建私有连接。您可以像在您的 VPC 中一样访问 Amazon Bedrock，无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 Amazon Bedrock。

您可以通过创建由 AWS PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管式网络接口，用作发往 Amazon Bedrock 的流量的入口点。

有关更多信息，请参阅 [AWS PrivateLink 指南](#) AWS PrivateLink 中的 [AWS 服务 通过访问](#)。

### Amazon Bedrock VPC 端点注意事项

在为 Amazon Bedrock 设置接口端点之前，请查看《[AWS PrivateLink 指南](#)》中的 [注意事项](#)。

Amazon Bedrock 支持通过 VPC 端点进行以下 API 调用。

| 类别                                         | 端点前缀            |
|--------------------------------------------|-----------------|
| <a href="#">Amazon Bedrock 控制面板 API 操作</a> | bedrock         |
| <a href="#">Amazon Bedrock 运行时 API 操作</a>  | bedrock-runtime |

| 类别                                           | 端点前缀                  |
|----------------------------------------------|-----------------------|
| <a href="#">Amazon Bedrock 构建时 API 操作的代理</a> | bedrock-agent         |
| <a href="#">Amazon Bedrock 代理运行时 API 操作</a>  | bedrock-agent-runtime |

## 可用区

Amazon Bedrock 和 Amazon Bedrock 终端节点代理可在多个可用区域使用。

### 为 Amazon Bedrock 创建接口端点

您可以使用亚马逊 VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 Amazon Bedrock 创建接口终端节点。有关更多信息，请参阅《AWS PrivateLink 指南》中的[创建接口端点](#)。

使用以下任一服务名称为 Amazon Bedrock 创建接口端点：

- com.amazonaws.*region*.bedrock
- com.amazonaws.*region*.bedrock-runtime
- com.amazonaws.*region*.bedrock-agent
- com.amazonaws.*region*.bedrock-agent-runtime

创建终端节点后，您可以选择启用私有 DNS 主机名。在创建 VPC 终端节点时，通过在 VPC 控制台中选择启用私有 DNS 名称，可启用此设置。

如果为接口端点启用私有 DNS，则可以使用区域的默认 DNS 名称向 Amazon Bedrock 发出 API 请求。以下示例显示了默认区域 DNS 名称的格式。

- bedrock.*region*.amazonaws.com
- bedrock-runtime.*region*.amazonaws.com
- bedrock-agent.*region*.amazonaws.com
- bedrock-agent-runtime.*region*.amazonaws.com

### 为接口端点创建端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略提供通过接口端点访问 Amazon Bedrock 的完全访问权限。要控制允许从 VPC 访问 Amazon Bedrock 的访问权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体 ( AWS 账户、IAM 用户和 IAM 角色 )。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《AWS PrivateLink 指南》中的[使用端点策略控制对服务的访问权限](#)。

示例：适用于 Amazon Bedrock 操作的 VPC 端点策略

以下是自定义端点策略的示例。当您将此基于资源的策略附加到接口终端节点时，它会向所有委托人授予所有资源上列出的 Amazon Bedrock 操作的访问权限。

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

## 适用于 Amazon Bedrock 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证 ( 登录 ) 和获得授权 ( 拥有权限 ) 以使用 Amazon Bedrock 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- [受众](#)
- [使用身份进行身份验证](#)
- [使用策略管理访问](#)

- [Amazon Bedrock 如何与 IAM 搭配使用](#)
- [Amazon Bedrock 基于身份的策略示例](#)
- [AWS Amazon Bedrock 的托管政策](#)
- [服务角色](#)
- [Amazon Bedrock 身份和访问问题排查](#)

## 受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Amazon Bedrock 中所做的工作。

**服务用户** – 如果您使用 Amazon Bedrock 服务来完成作业，则您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon Bedrock 功能来执行工作，您可能需要更多权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon Bedrock 中的功能，请参阅[Amazon Bedrock 身份和访问问题排查](#)。

**服务管理员** – 如果您在公司负责管理 Amazon Bedrock 资源，您可能对 Amazon Bedrock 具有完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon Bedrock 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Amazon Bedrock 搭配使用的更多信息，请参阅[Amazon Bedrock 如何与 IAM 搭配使用](#)。

**IAM 管理员** - 如果您是 IAM 管理员，您可能希望了解有关如何编写策略来管理对 Amazon Bedrock 的访问的详细信息。要查看您可以在 IAM 中使用的 Amazon Bedrock 基于身份的策略示例，请参阅[Amazon Bedrock 基于身份的策略示例](#)。

## 使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担任 AWS 账户根用户担任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。AWS IAM Identity Center（IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》[中的如何登录到您 AWS 账户的](#)。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的[签署 AWS API 请求](#)。

无论使用何种身份验证方法，您可能需要提供其它安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的[多重身份验证](#)和《IAM 用户指南》中的[在 AWS 中使用多重身份验证 \(MFA\)](#)。

## AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的[需要根用户凭证的任务](#)。

## 联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity Center 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。AWS Directory Service 当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的[什么是 IAM Identity Center ?](#)

## IAM 用户和组

[IAM 用户](#)是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果有一些特定的使用场景需要长期凭证以及 IAM 用户，我们建议轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的[对于需要长期凭证的使用场景定期轮换访问密钥](#)。

[IAM 组](#)是一个用于指定一组 IAM 用户的身份。您不能使用组的身份登录。可以使用群组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，可能具有一个名为 IAMAdmins 的群组，并为该群组授予权限以管理 IAM 资源。



用户与角色不同。用户唯一地与某个人或应用程序关联，而角色旨在让需要它的任何人担任。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的[何时创建 IAM 用户（而不是角色）](#)。

## IAM 角色

[IAM 角色](#)是您内部具有特定权限 AWS 账户的身份。它类似于 IAM 用户，但与特定人员不关联。您可以使用 AWS Management Console 通过[切换角色在中临时担任 IAM 角色](#)。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的[使用 IAM 角色](#)。

具有临时凭证的 IAM 角色在以下情况下很有用：

- **联合用户访问** – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的[为第三方身份提供商创建角色](#)。如果使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的[权限集](#)。
- **临时 IAM 用户权限** – IAM 用户或角色可代入 IAM 角色，以暂时获得针对特定任务的不同权限。
- **跨账户存取**：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户存取的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的[IAM 角色与基于资源的策略有何不同](#)。
- **跨服务访问** — 有些 AWS 服务使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - **转发访问会话 (FAS)** — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务向下游服务发出请求的请求。AWS 服务只有当服务收到需要与其他 AWS 服务或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅[转发访问会话](#)。
- **服务角色** - 服务角色是服务代表您在您的账户中执行操作而分派的 [IAM 角色](#)。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的[创建向 AWS 服务委派权限的角色](#)。
- **服务相关角色**-服务相关角色是一种与服务相关联的服务角色。AWS 服务服务可以担任代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。



- 在 Amazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的[使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限](#)。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的[何时创建 IAM 角色（而不是用户）](#)。

## 使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS，当与身份或资源关联时，它会定义其权限。AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的[JSON 策略概览](#)。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。然后，管理员可以向角色添加 IAM 策略，并且用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

## 基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的[创建 IAM 策略](#)。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的[在托管式策略与内联策略之间进行选择](#)。

## 基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。必须在基于资源的策略中[指定主体](#)。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

## 访问控制列表 ( ACL )

访问控制列表 ( ACL ) 控制哪些主体 ( 账户成员、用户或角色 ) 有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。AWS WAF 要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的[访问控制列表 \(ACL\) 概览](#)。

## 其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型所授予的最大权限。

- **权限边界**：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体 ( IAM 用户或角色 ) 授予的最大权限。可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 Principal 字段中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的[IAM 实体的权限边界](#)。
- **服务控制策略 (SCP)**-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 ( SCP )。SCP 限制成员账户中的实体 ( 包括每个 AWS 账户根用户实体 ) 的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的[SCP 的工作原理](#)。
- **会话策略** – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的[会话策略](#)。

## 多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的[策略评估逻辑](#)。

## Amazon Bedrock 如何与 IAM 搭配使用

在使用 IAM 管理对 Amazon Bedrock 的访问之前，您应该了解哪些 IAM 功能可与 Amazon Bedrock 搭配使用。

可与 Amazon Bedrock 搭配使用的 IAM 功能

| IAM 功能                        | Amazon Bedrock 支持 |
|-------------------------------|-------------------|
| <a href="#">基于身份的策略</a>       | 是                 |
| <a href="#">基于资源的策略</a>       | 否                 |
| <a href="#">策略操作</a>          | 是                 |
| <a href="#">策略资源</a>          | 是                 |
| <a href="#">策略条件键</a>         | 是                 |
| <a href="#">ACL</a>           | 否                 |
| <a href="#">ABAC (策略中的标签)</a> | 是                 |
| <a href="#">临时凭证</a>          | 是                 |
| <a href="#">主体权限</a>          | 是                 |
| <a href="#">服务角色</a>          | 是                 |
| <a href="#">服务相关角色</a>        | 否                 |

要全面了解 Amazon Bedrock 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM [配合使用的AWS 服务](#)。

## Amazon Bedrock 基于身份的策略

支持基于身份的策略 是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的[创建 IAM policy](#)。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的[IAM JSON 策略元素引用](#)。

### Amazon Bedrock 基于身份的策略示例

要查看 Amazon Bedrock 基于身份的策略示例，请参阅[Amazon Bedrock 基于身份的策略示例](#)。

## Amazon Bedrock 基于资源的策略

支持基于资源的策略 否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。必须在基于资源的策略中[指定主体](#)。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅 IAM 用户指南中的[IAM 角色与基于资源的策略有何不同](#)。

### Amazon Bedrock 的策略操作

支持策略操作 是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 Action 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Amazon Bedrock 操作列表，请参阅《[服务授权参考](#)》中的 [Amazon Bedrock 定义的操作](#)。

Amazon Bedrock 中的策略操作在操作前使用以下前缀：

```
bedrock
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

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

要查看 Amazon Bedrock 基于身份的策略示例，请参阅[Amazon Bedrock 基于身份的策略示例](#)。

## Amazon Bedrock 的策略资源

支持策略资源

是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

ResourceJSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实操，请使用 [Amazon 资源名称 \(ARN\)](#) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (\*) 指示语句应用于所有资源。

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

要查看 Amazon Bedrock 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 [Amazon Bedrock 定义的资源](#)。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅 [Amazon Bedrock 定义的操作](#)。

某些 Amazon Bedrock API 操作支持多种资源。例如，[AssociateAgentKnowledgeBase](#) 访问 [AGENT12345](#) 和 [KB12345678](#)，因此委托人必须具有访问这两个资源的权限。要在单个语句中指定多个资源，请使用逗号分隔 ARN。

```
"Resource": [
 "arn:aws:bedrock:aws-region:111122223333:agent/AGENT12345",
 "arn:aws:bedrock:aws-region:111122223333:knowledge-base/KB12345678"
]
```

要查看 Amazon Bedrock 基于身份的策略示例，请参阅 [Amazon Bedrock 基于身份的策略示例](#)。

## Amazon Bedrock 的策略条件键

支持特定于服务的策略条件键 **是**

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素 ( 或 Condition 块 ) 中，可以指定语句生效的条件。Condition 元素是可选的。可以创建使用 [条件运算符](#) ( 例如，等于或小于 ) 的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评测它们。如果您为单个条件键指定多个值，则使用逻辑 OR 运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 [IAM 策略元素：变量和标签](#)。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的 [AWS 全局条件上下文密钥](#)。

要查看 Amazon Bedrock 条件密钥列表，请参阅《服务授权参考》中的 [Amazon Bedrock 条件密钥](#)。要了解您可以使用哪些操作和资源使用条件键，请参阅 [Amazon Bedrock 定义的操作](#)。

所有 Amazon Bedrock 操作均支持使用 Amazon Bedrock 模型作为资源的条件键。

要查看 Amazon Bedrock 基于身份的策略示例，请参阅[Amazon Bedrock 基于身份的策略示例](#)。

## Amazon Bedrock 中的 ACL

|        |   |
|--------|---|
| 支持 ACL | 否 |
|--------|---|

访问控制列表 (ACL) 控制哪些主体 ( 账户成员、用户或角色 ) 有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

## ABAC 与 Amazon Bedrock

|                    |   |
|--------------------|---|
| 支持 ABAC ( 策略中的标签 ) | 是 |
|--------------------|---|

基于属性的访问权限控制 ( ABAC ) 是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以将标签附加到 IAM 实体 ( 用户或角色 ) 和许多 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的[条件元素](#)中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的[什么是 ABAC？](#) 要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的[使用基于属性的访问权限控制 \( ABAC \)](#)。

## 将临时凭证与 Amazon Bedrock 搭配使用

|        |   |
|--------|---|
| 支持临时凭证 | 是 |
|--------|---|

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的[AWS 服务与 IAM 配合使用的信息](#)。



如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的[切换到角色 \(控制台\)](#)。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 [IAM 中的临时安全凭证](#)。

## Amazon Bedrock 的跨服务主体权限

支持转发访问会话 (FAS) 是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用调用委托人的权限 AWS 服务，再加上 AWS 服务 向下游服务发出请求的请求。只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅[转发访问会话](#)。

## Amazon Bedrock 的服务角色

支持服务角色 是

服务角色是由一项服务代入、代表您执行操作的 [IAM 角色](#)。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的 [创建向 AWS 服务 委派权限的角色](#)。

### Warning

更改服务角色的权限可能会破坏 Amazon Bedrock 的功能。仅在 Amazon Bedrock 提供相关指导时编辑服务角色。

## Amazon Bedrock 的服务相关角色

支持服务相关角色 否



服务相关角色是一种与服务相关联的 AWS 服务角色。服务可以担任代表您执行操作的角色。服务相关角色出现在您的 AWS 账户，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

## Amazon Bedrock 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon Bedrock 资源的权限。他们也无法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的[创建 IAM 策略](#)。

有关 Amazon Bedrock 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅《服务授权参考》中的[Amazon Bedrock 的操作、资源和条件键](#)。

### 主题

- [策略最佳实践](#)
- [使用 Amazon Bedrock 控制台](#)
- [允许用户查看他们自己的权限](#)
- [允许访问第三方模型订阅](#)
- [拒绝访问以对特定模型进行推理](#)
- [Amazon Bedrock 代理基于身份的策略示例](#)
- [预配置吞吐量的基于身份的策略示例](#)

### 策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon Bedrock 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的 AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的[AWS 托管策略](#) 或 [工作职能的 AWS 托管策略](#)。

- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 [IAM 中的策略和权限](#)。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 [IAM JSON 策略元素：条件](#)。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy 语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 [IAM Access Analyzer 策略验证](#)。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 [配置受 MFA 保护的 API 访问](#)。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 [IAM 中的安全最佳实操](#)。

## 使用 Amazon Bedrock 控制台

要访问 Amazon Bedrock 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您 AWS 账户的 Amazon Bedrock 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用 Amazon Bedrock 控制台，还需要将 Amazon Bedrock [AmazonBedrockFullAccess](#) 或 [AmazonBedrockReadOnly](#) AWS 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的 [为用户添加权限](#)。

## 允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

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

```

 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam:ListGroupsForUser",
 "iam:ListAttachedUserPolicies",
 "iam:ListUserPolicies",
 "iam:GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",
 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam:GetPolicy",
 "iam:ListAttachedGroupPolicies",
 "iam:ListGroupPolicies",
 "iam:ListPolicyVersions",
 "iam:ListPolicies",
 "iam:ListUsers"
],
 "Resource": "*"
 }
]
}

```

## 允许访问第三方模型订阅

首次访问 Amazon Bedrock 模型时，需要使用 Amazon Bedrock 控制台订阅第三方模型。您的 IAM 用户或控制台用户担任的角色需要访问订阅 API 操作的权限。

以下示例展示允许访问订阅 API 操作的基于身份的策略。

使用条件密钥（如示例所示）将政策的范围限制在 Marketplace 中的 Amazon Bedrock 基础模型的子集。要查看产品 ID 列表以及它们对应的基础模型，请参阅中的表格[控制模型访问权限](#)。

```

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

```

```

 "Effect": "Allow",
 "Action": [
 "aws-marketplace:Subscribe"
],
 "Resource": "*",
 "Condition": {
 "ForAnyValue:StringEquals": {
 "aws-marketplace:ProductId": [
 "1d288c71-65f9-489a-a3e2-9c7f4f6e6a85",
 "cc0bdd50-279a-40d8-829c-4009b77a1fcc",
 "c468b48a-84df-43a4-8c46-8870630108a7",
 "99d90be8-b43e-49b7-91e4-752f3866c8c7",
 "b0eb9475-3a2c-43d1-94d3-56756fd43737",
 "d0123e8d-50d6-4dba-8a26-3fed4899f388",
 "a61c46fe-1747-41aa-9af0-2e0ae8a9ce05",
 "216b69fd-07d5-4c7b-866b-936456d68311",
 "b7568428-a1ab-46d8-bab3-37def50f6f6a",
 "38e55671-c3fe-4a44-9783-3584906e7cad",
 "prod-ariujvyzvd2qy",
 "prod-2c2yc2s3guhqy",
 "prod-6dw3qvchef7zy",
 "prod-ozonys2hmmpeu"
]
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "aws-marketplace:Unsubscribe",
 "aws-marketplace:ViewSubscriptions"
],
 "Resource": "*"
 }
]
}

```

## 拒绝访问以对特定模型进行推理

以下示例展示基于身份的策略，该策略拒绝访问以对特定模型进行推理。

```

{
 "Version": "2012-10-17",

```

```

 "Statement": {
 "Sid": "DenyInference",
 "Effect": "Deny",
 "Action": [
 "bedrock:InvokeModel",
 "bedrock:InvokeModelWithResponseStream"
],
 "Resource": "arn:aws:bedrock:*::foundation-model/model-id"
 }
 }
}

```

## Amazon Bedrock 代理基于身份的策略示例

选择一个主题以查看您可以附加到 IAM 角色的 IAM 策略示例，以便为中的操作配置权限[Agents for Amazon Bedrock](#)。

### 主题

- [Amazon Bedrock 代理所需的权限](#)
- [允许用户查看有关代理的信息并调用代理](#)

### Amazon Bedrock 代理所需的权限

要让 IAM 身份使用适用于 Amazon Bedrock 的代理，您必须为其配置必要的权限。您可以附加[AmazonBedrockFullAccess](#)策略以向该角色授予适当的权限。

要将权限限制为仅限在 Amazon Bedrock 代理中使用的操作，请将以下基于身份的策略附加到 IAM 角色：

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "Agents for Amazon Bedrock permissions",
 "Effect": "Allow",
 "Action": [
 "bedrock:ListFoundationModels",
 "bedrock:GetFoundationModel",
 "bedrock:TagResource",
 "bedrock:UntagResource",
 "bedrock:ListTagsForResource",
 "bedrock:CreateAgent",

```

```

 "bedrock:UpdateAgent",
 "bedrock:GetAgent",
 "bedrock:ListAgents",
 "bedrock>DeleteAgent",
 "bedrock:CreateAgentActionGroup",
 "bedrock:UpdateAgentActionGroup",
 "bedrock:GetAgentActionGroup",
 "bedrock:ListAgentActionGroups",
 "bedrock>DeleteAgentActionGroup",
 "bedrock:GetAgentVersion",
 "bedrock:ListAgentVersions",
 "bedrock>DeleteAgentVersion",
 "bedrock:CreateAgentAlias",
 "bedrock:UpdateAgentAlias",
 "bedrock:GetAgentAlias",
 "bedrock:ListAgentAliases",
 "bedrock>DeleteAgentAlias",
 "bedrock:AssociateAgentKnowledgeBase",
 "bedrock:DisassociateAgentKnowledgeBase",
 "bedrock:GetKnowledgeBase",
 "bedrock:ListKnowledgeBases",
 "bedrock:PrepareAgent",
 "bedrock:InvokeAgent"
],
 "Resource": "*"
}
]
}

```

您可以通过省略[操作](#)或指定[资源](#)和[条件键](#)来进一步限制权限。IAM 身份可以对特定资源调用 API 操作。例如，该[UpdateAgent](#)操作只能在代理资源上使用，而该[InvokeAgent](#)操作只能在别名资源上使用。对于未在特定资源类型上使用的 API 操作（例如 [CreateAgent](#)），请将\* 指定为Resource。如果您指定的 API 操作不能用于策略中指定的资源，Amazon Bedrock 将返回错误。

允许用户查看有关代理的信息并调用代理

以下是您可以附加到 IAM 角色的示例策略，以允许该角色查看有关编号# *AGENT12345* 的代理的信息或编辑，以及与 ID # *ALIAS12345* 的代理的别名进行交互。例如，您可以将此策略附加到您只想拥有代理故障排除和更新代理的权限的角色。

```

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

```

```

 {
 "Sid": "Get information about and update an agent",
 "Effect": "Allow",
 "Action": [
 "bedrock:GetAgent",
 "bedrock:UpdateAgent"
],
 "Resource": "arn:aws:bedrock:aws-region:111122223333:agent/AGENT12345"
 },
 {
 "Sid": "Invoke an agent",
 "Effect": "Allow",
 "Action": [
 "bedrock:InvokeAgent"
],
 "Resource": "arn:aws:bedrock:aws-region:111122223333:agent-alias/AGENT12345/ALIAS12345"
 },
]
}

```

## 预配置吞吐量的基于身份的策略示例

选择一个主题以查看您可以附加到 IAM 角色的 IAM 策略示例，以便为与之相关的操作配置权限 [Amazon Bedrock 的预配置吞吐量](#)。

### 主题

- [预配置吞吐量所需的权限](#)
- [允许用户调用预配置的模型](#)

### 预配置吞吐量所需的权限

要让 IAM 身份使用预配置吞吐量，您必须为其配置必要的权限。您可以附加 [AmazonBedrockFullAccess](#) 策略以向该角色授予适当的权限。

要将权限限制为仅在预配置吞吐量中使用的操作，请将以下基于身份的策略附加到 IAM 角色：

```

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

```

```

 "Sid": "Provisioned Throughput permissions",
 "Effect": "Allow",
 "Action": [
 "bedrock:GetFoundationModel",
 "bedrock:ListFoundationModels",
 "bedrock:InvokeModel",
 "bedrock:InvokeModelWithResponseStream",
 "bedrock:ListTagsForResource",
 "bedrock:UntagResource",
 "bedrock:TagResource",
 "bedrock:CreateProvisionedModelThroughput",
 "bedrock:GetProvisionedModelThroughput",
 "bedrock:ListProvisionedModelThroughputs",
 "bedrock:UpdateProvisionedModelThroughput",
 "bedrock>DeleteProvisionedModelThroughput"
],
 "Resource": "*"
 }
]
}

```

您可以通过省略[操作](#)或指定[资源](#)和[条件键](#)来进一步限制权限。IAM 身份可以对特定资源调用 API 操作。例如，该[CreateProvisionedModelThroughput](#)操作只能用于自定义模型和基础模型资源，并且该[DeleteProvisionedModelThroughput](#)操作只能用于预配置的模型资源。对于未在特定资源类型上使用的 API 操作（例如 [ListProvisionedModelThroughputs](#)），请将\*指定为Resource。如果您指定的 API 操作不能用于策略中指定的资源，Amazon Bedrock 将返回错误。

### 允许用户调用预配置的模型

以下是一个示例策略，您可以将其附加到 IAM 角色以允许其在模型推断中使用预配置的模型。例如，您可以将此策略附加到您只想拥有使用预配置模型权限的角色。该角色将无法管理或查看有关预配置吞吐量的信息。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "Use a Provisioned Throughput for model inference",
 "Effect": "Allow",
 "Action": [
 "bedrock:InvokeModel",
 "bedrock:InvokeModelWithResponseStream"
]
 }
]
}

```



```
],
 "Resource": "arn:aws:bedrock:aws-region:111122223333:provisioned-
model/${my-provisioned-model}"
 }
]
}
```

## AWS Amazon Bedrock 的托管政策

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建仅为团队提供所需权限的 [IAM 客户管理型策略](#) 需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅 IAM 用户指南中的 [AWS 托管策略](#)。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管策略添加额外权限以支持新功能。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新功能或新操作可用时，服务最有可能更新 AWS 托管策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动一项新功能时，AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅《IAM 用户指南》中的 [适用于工作职能的 AWS 托管策略](#)。

### AWS 托管策略：AmazonBedrockFullAccess

您可以将 AmazonBedrockFullAccess 策略附加到 IAM 身份。

此策略可授予管理权限，允许用户创建、读取、更新和删除 Amazon Bedrock 资源。

#### Note

微调和访问模型需要额外的权限。有关更多信息，请参阅 [允许访问第三方模型订阅](#) 和 [访问训练和验证文件以及在 S3 中写入输出文件的权限](#)。

#### 权限详细信息

该策略包含以下权限：

- ec2 (Amazon Elastic Compute Cloud) – 允许描述 VPC、子网和安全组的权限。
- iam ( Identity and Access Management ) — 允许委托人传递角色，但仅允许将带有“Amazon Bedrock”的 IAM 角色传递给 Amazon Bedrock 服务。对于 Amazon Bedrock 操作，这些权限仅限于 bedrock.amazonaws.com。
- kms ( AWS 密钥管理服务 ) -允许委托人描述 AWS KMS 密钥和别名。
- bedrock (Amazon Bedrock) – 允许主体读取和写入 Amazon Bedrock 控制面板和运行时服务中的所有操作。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "BedrockAll",
 "Effect": "Allow",
 "Action": [
 "bedrock:*"
],
 "Resource": "*"
 },
 {
 "Sid": "DescribeKey",
 "Effect": "Allow",
 "Action": [
 "kms:DescribeKey"
],
 "Resource": "arn:*:kms:*:*:*"
 },
 {
 "Sid": "APIsWithAllResourceAccess",
 "Effect": "Allow",
 "Action": [
 "iam:ListRoles",
 "ec2:DescribeVpcs",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
],
 "Resource": "*"
 },
 {
 "Sid": "PassRoleToBedrock",
 "Effect": "Allow",
```

```

 "Action": [
 "iam:PassRole"
],
 "Resource": "arn:aws:iam::*:role/*AmazonBedrock*",
 "Condition": {
 "StringEquals": {
 "iam:PassedToService": [
 "bedrock.amazonaws.com"
]
 }
 }
 }
]
}

```

## AWS 托管策略：AmazonBedrockReadOnly

您可以将 AmazonBedrockReadOnly 策略附加到 IAM 身份。

此策略可授予只读权限，允许用户查看 Amazon Bedrock 中的所有资源。

```

{
 "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 更新了托 AWS 管政策

查看自该服务开始跟踪这些更改以来，Amazon Bedrock AWS 托管政策更新的详细信息。要获得有关此页面更改的自动提示，请订阅 [《Amazon Bedrock 用户指南》的文档历史记录](#) 上的 RSS 源。

| 更改                                            | 描述                                            | 日期               |
|-----------------------------------------------|-----------------------------------------------|------------------|
| <a href="#">AmazonBedrockFullAccess</a> - 新策略 | Amazon Bedrock 添加了一项新策略，授予用户创建、读取、更新和删除资源的权限。 | 2023 年 12 月 12 日 |
| <a href="#">AmazonBedrockReadOnly</a> : 新策略   | Amazon Bedrock 添加了一项新策略，授予用户对所有操作的只读权限。       | 2023 年 12 月 12 日 |
| Amazon Bedrock 已开启跟踪更改                        | Amazon Bedrock 开始跟踪其 AWS 托管政策的变更。             | 2023 年 12 月 12 日 |

## 服务角色

Amazon Bedrock 将 [IAM 服务角色](#) 用于以下功能，让 Amazon Bedrock 代表您执行任务。

控制台会自动为支持的功能创建服务角色。

您还可以创建自定义服务角色并根据您的特定用例自定义附加权限。如果您使用控制台，则可以选择此角色，而不必让 Amazon Bedrock 为您创建角色。

要设置自定义服务角色，请执行以下常规步骤。

1. 按照创建角色中的步骤 [创建角色以向 AWS 服务委派权限](#)。
2. 附上信任策略。
3. 附加相关的基于身份的权限。

有关与设置服务角色权限相关的 IAM 概念的更多信息，请参阅以下链接。

- [AWS 服务角色](#)

- [基于身份的策略和基于资源的策略](#)
- [对 Lambda 使用基于资源的策略](#)
- [AWS 全局条件上下文密钥](#)
- [Amazon Bedrock 的条件密钥](#)

选择一个主题以了解有关特定功能的服务角色的更多信息。

#### 主题

- [为模型自定义创建服务角色](#)
- [为 Amazon Bedrock 的代理创建服务角色](#)
- [为 Amazon Bedrock 知识库创建服务角色](#)

### 为模型自定义创建服务角色

要使用自定义角色而不是 Amazon Bedrock 自动创建的角色进行模型定制，请按照创建角色[向 AWS 服务委派权限中的步骤创建一个 IAM 角色并附加以下权限](#)。

- 信任关系
- 访问 S3 中的训练和验证数据以及将输出数据写入 S3 的权限
- ( 可选 ) 如果您使用 KMS 密钥加密以下任何资源，则还有解密密钥的权限 ( 请参阅[对模型自定义任务和构件进行加密](#) )
  - 模型自定义作业或生成的自定义模型
  - 用于模型自定义作业的训练、验证和输出数据

#### 主题

- [信任关系](#)
- [访问训练和验证文件以及在 S3 中写入输出文件的权限](#)

#### 信任关系

以下策略允许 Amazon Bedrock 担任此角色并执行模型自定义作业。下面所示为您可以使用的示例策略。

您可以选择通过在 Condition 字段中使用一个或多个全局条件上下文密钥来限制[跨服务混淆副手预防](#)的权限范围。有关更多信息，请参阅 [AWS 全局条件上下文键](#)。

- 将 `aws:SourceAccount` 值设置为您的账户 ID。
- ( 可选 ) 使用 `ArnEquals` 或 `ArnLike` 条件将范围限制在您的账户 ID 中的特定模型自定义任务范围内。

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

访问训练和验证文件以及在 S3 中写入输出文件的权限

附加以下策略以允许该角色访问您的训练和验证数据以及向其写入输出数据的存储桶。将 `Resource` 列表中的值替换为您实际的存储桶名称。

要限制对存储桶中特定文件夹的访问权限，请在您的文件夹路径中添加 `s3:prefix` 条件密钥。您可以按照 [示例 2 中的用户策略示例进行操作：获取带有特定前缀的存储桶中的对象列表](#)

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
```

```

 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::training-bucket",
 "arn:aws:s3:::training-bucket/*",
 "arn:aws:s3:::validation-bucket",
 "arn:aws:s3:::validation-bucket/*"
]
},
{
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::output-bucket",
 "arn:aws:s3:::output-bucket/*"
]
}
]
}

```

## 为 Amazon Bedrock 的代理创建服务角色

要为代理使用自定义服务角色而不是 Amazon Bedrock 自动创建的角色，请创建带有前缀的 IAM 角色 `AmazonBedrockExecutionRoleForAgents_` 并按照 [创建角色向 AWS 服务委派权限中的步骤附加以下权限](#)。

- 信任策略
- 包含以下基于身份的权限的策略
  - 对 Amazon Bedrock 基础模型的访问权限
  - 访问包含代理中操作组 OpenAPI 架构的 Amazon S3 对象
  - Amazon Bedrock 有权查询您要附加到代理的知识库
  - ( 可选 ) 如果您使用 KMS 密钥加密代理，则还有解密密钥的权限 ( 请参阅 [代理资源的加密](#) )

无论您是否使用自定义角色，您还需要将基于资源的策略附加到代理中的操作组的 Lambda 函数，以便为服务角色提供访问这些函数的权限。有关更多信息，请参阅 [基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数](#)。

## 主题

- [信任关系](#)
- [代理服务角色的基于身份的权限。](#)
- [基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数](#)

## 信任关系

以下信任策略允许 Amazon Bedrock 担任此角色并创建和管理代理。根据需要替换这些#。该策略在Condition字段中包含可选的[条件键](#)（[请参阅 Amazon Bedrock 的条件密钥和AWS 全局条件上下文密钥](#)），我们建议您将其用作安全最佳实践。

### Note

出于安全考虑，最佳做法是在创建完特定的代理 ID 后将 \* 替换为该 ID。

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

代理服务角色的基于身份的权限。

附加以下策略以为服务角色提供权限，并根据需要替换#。该策略包含以下声明。如果语句不适用于您的用例，则将其省略。该策略在Condition字段中包含可选的[条件键](#)（[请参阅 Amazon Bedrock 的条件密钥和AWS 全局条件上下文密钥](#)），我们建议您将其用作安全最佳实践。



**Note**

如果您使用客户管理的 KMS 密钥加密代理，请参阅[代理资源的加密](#)以了解需要添加的更多权限。

- 使用 Amazon Bedrock 基础模型对代理编排中使用的提示进行模型推断的权限。
- 在 Amazon S3 中访问您的代理的操作组 API 架构的权限。如果您的代理没有操作组，请省略此语句。
- 访问与您的代理相关的知识库的权限。如果您的代理没有相关的知识库，请省略此声明。
- 访问与您的代理关联的第三方（Pinecone或Redis Enterprise Cloud）知识库的权限。如果您的知识库是第一方（Amazon OpenSearch Serverless 或 Amazon Aurora），或者您的代理没有相关的知识库，请省略此声明。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "Allow model invocation for orchestration",
 "Effect": "Allow",
 "Action": [
 "bedrock:InvokeModel"
],
 "Resource": [
 "arn:aws:bedrock:region::foundation-model/anthropic.claude-v2",
 "arn:aws:bedrock:region::foundation-model/anthropic.claude-v2:1",
 "arn:aws:bedrock:region::foundation-model/anthropic.claude-instant-v1"
]
 },
 {
 "Sid": "Allow access to action group API schemas in S3",
 "Effect": "Allow",
 "Action": [
 "s3:GetObject"
],
 "Resource": [
 "arn:aws:s3:::bucket/path/to/schema"
],
 "Condition": {
 "StringEquals": {
```

```

 "aws:ResourceAccount": "account-id"
 }
 }
 },
 {
 "Sid": "Query associated knowledge bases",
 "Effect": "Allow",
 "Action": [
 "bedrock:Retrieve",
 "bedrock:RetrieveAndGenerate"
],
 "Resource": [
 "arn:aws:bedrock:region:account-id:knowledge-base/knowledge-base-id"
]
 },
 {
 "Sid": "Associate a third-party knowledge base with your agent",
 "Effect": "Allow",
 "Action": [
 "bedrock:AssociateThirdPartyKnowledgeBase",
],
 "Resource": "arn:aws:bedrock:region:account-id:knowledge-base/knowledge-
base-id",
 "Condition": {
 "StringEquals" : {
 "bedrock:ThirdPartyKnowledgeBaseCredentialsSecretArn":
"arn:aws:kms:region:account-id:key/key-id"
 }
 }
 }
]
}

```

基于资源的策略，允许 Amazon Bedrock 调用操作组 Lambda 函数

### [Lambda](#) ##### [Lambda](#) ##### [Amazon Bedrock](#) ##### [Lambda](#) ##### 该策略在 Condition 字段中包含可选的 [条件键](#) ( 请参阅 [Amazon Bedrock](#) 的 [条件密钥和AWS 全局条件上下文密钥](#) ) ，我们建议您将其用作安全最佳实践。

```

{
 "Version": "2012-10-17",
 "Statement": [{
 "Sid": "Allow Amazon Bedrock to access action group Lambda function",

```

```

 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 },
 "Action": "lambda:InvokeFunction",
 "Resource": "arn:aws:lambda:region:account-id:function:function-name",
 "Condition": {
 "StringEquals": {
 "AWS:SourceAccount": "account-id"
 },
 "ArnLike": {
 "AWS:SourceArn": "arn:aws:bedrock:region:account-id:agent/agent-id"
 }
 }
 }
}

```

## 为 Amazon Bedrock 知识库创建服务角色

要使用知识库的自定义角色而不是 Amazon Bedrock 自动创建的角色，请按照创建角色[向 AWS 服务委派权限中的步骤创建一个 IAM 角色并附加以下权限](#)。您可以在知识库中使用相同的角色。

- 信任关系
- 对 Amazon Bedrock 基础模型的访问权限
- 对包含数据来源的 Amazon S3 对象的访问权限
- ( 如果您在 Amazon OpenSearch 服务中创建矢量数据库 ) 访问您的 OpenSearch 服务集合
- ( 如果您在 Amazon Aurora 中创建向量数据库 )
- ( 如果您在 Pinecone 或中创建矢量数据库 Redis Enterprise Cloud ) 对您的 Pinecone 或 Redis Enterprise Cloud 账户 AWS Secrets Manager 进行身份验证的权限
- ( 可选 ) 如果您使用 KMS 密钥加密以下任何资源，则还有解密密钥的权限 ( 请参阅[知识库资源的加密](#) )。
  - 知识库
  - 知识库的数据来源
  - 您在 Amazon OpenSearch 服务中的矢量数据库
  - 您的第三方矢量数据库的秘密 AWS Secrets Manager
  - 数据提取作业

## 主题

- [信任关系](#)
- [访问 Amazon Bedrock 模型的权限](#)
- [访问 Amazon S3 中数据来源的权限](#)
- [\( 可选 \) 在 Amazon OpenSearch 服务中访问您的矢量数据库的权限](#)
- [\( 可选 \) 访问 Amazon Aurora 数据库集群的权限](#)
- [\( 可选 \) 访问配置了 AWS Secrets Manager 密钥的矢量数据库的权限](#)
- [\( 可选 \) 在数据摄取 AWS 期间管理用于临时数据存储的 AWS KMS 密钥的权限](#)

## 信任关系

以下策略允许 Amazon Bedrock 担任此角色，并创建和管理知识库。下面所示为您可以使用的示例策略。您可以使用一个或多个全局条件上下文键来限制权限的范围。有关更多信息，请参阅 [AWS 全局条件上下文键](#)。将 `aws:SourceAccount` 值设置为您的账户 ID。使用 `ArnEquals` 或 `ArnLike` 条件将范围限制为特定的知识库。

### Note

出于安全考虑，最佳做法是在创建完特定的知识库 ID 后将 \* 替换为该 ID。

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

```
}

```

## 访问 Amazon Bedrock 模型的权限

附加以下策略，以向角色提供使用 Amazon Bedrock 模型嵌入源数据的权限。

```
{
 "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",
 "arn:aws:bedrock:region::foundation-model/cohere.embed-english-v3",
 "arn:aws:bedrock:region::foundation-model/cohere.embed-multilingual-v3"
]
 }
]
}
```

## 访问 Amazon S3 中数据来源的权限

附加以下策略，以向角色提供访问包含知识库的数据来源文件的 Amazon S3 URI 的权限。在 Resource 字段中，提供包含数据来源的 Amazon S3 对象，或者将每个数据来源的 URI 添加到列表中。

如果您使用 AWS KMS 密钥加密了这些数据源，请按照中的步骤为角色授予解密密钥的权限。[解密您在 Amazon S3 中的数据源的 AWS KMS 密钥的权限](#)

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

```

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

```

( 可选 ) 在 Amazon OpenSearch 服务中访问您的矢量数据库的权限

如果您在亚马逊 OpenSearch 服务中为知识库创建了矢量数据库，请将以下策略附加到您的 Amazon Bedrock 知识库服务角色中，以允许访问该馆藏。将 *region* 和 *account-id* 替换为数据库所属的区域和账户 ID。在产品系列编号中输入您的亚马逊 OpenSearch 服务产品#####您可以通过将多个集合添加到Resource列表中来允许访问这些集合。

```

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

```

( 可选 ) 访问 Amazon Aurora 数据库集群的权限

如果您在 Amazon Aurora 中为知识库创建了数据库 (DB) 集群，请将以下策略附加到您的 Amazon Bedrock 知识库服务角色中，以允许访问该数据库集群并为其提供读写权限。将 *region* 和

*account-id* 替换为 DB 集群所属的区域和账户 ID。在中输入您的亚马逊 Aurora 数据库集群的 ID *db-cluster-id*。您可以通过将多个 DB 集群添加到 Resource 列表来允许访问这些集群。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "RdsDescribeStatementID",
 "Effect": "Allow",
 "Action": [
 "rds:DescribeDBClusters"
],
 "Resource": [
 "arn:aws:rds:region:account-id:cluster:db-cluster-id"
]
 },
 {
 "Sid": "DataAPIStatementID",
 "Effect": "Allow",
 "Action": [
 "rds-data:BatchExecuteStatement",
 "rds-data:ExecuteStatement"
],
 "Resource": [
 "arn:aws:rds:region:account-id:cluster:db-cluster-id"
]
 }
]
}
```

( 可选 ) 访问配置了 AWS Secrets Manager 密钥的矢量数据库的权限

如果您的矢量数据库配置了 AWS Secrets Manager 密钥，请将以下策略附加到您的 Amazon Bedrock 服务角色知识库中，AWS Secrets Manager 以允许对您的账户进行身份验证以访问数据库。将 *region* 和 *account-id* 替换为数据库所属的区域和账户 ID。将 *secret-id* 替换为您的密钥的 ID。

```
{
 "Version": "2012-10-17",
 "Statement": [{
 "Effect": "Allow",
 "Action": [
 "secretsmanager:GetSecretValue"
],
 "Resource": [
```

```

 "arn:aws:secretsmanager:region:account-id:secret:secret-id"
]
 }]
}

```

如果您使用密钥加密了您的密 AWS KMS 钥，请按照中的步骤为该角色授予解密密钥的权限。[解密包含知识库的矢量存储的 AWS Secrets Manager 密钥的权限](#)

( 可选 ) 在数据摄取 AWS 期间管理用于临时数据存储的 AWS KMS 密钥的权限

要允许在提取数据源的过程中创建用于临时数据存储的 AWS KMS 密钥，请将以下策略附加到您的 Amazon Bedrock 服务角色知识库中。将 *region*、*account-id* 和 *key-id* 替换为适当的值。

```

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

```

## Amazon Bedrock 身份和访问问题排查

以下信息可帮助您诊断和修复在使用 Amazon Bedrock 和 IAM 时可能遇到的常见问题。

### 主题

- [我无权在 Amazon Bedrock 中执行操作](#)
- [我无权执行 iam : PassRole](#)
- [我想允许我以外的人访问我 AWS 账户 的 Amazon Bedrock 资源](#)

### 我无权在 Amazon Bedrock 中执行操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。



当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 `bedrock:GetWidget` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
bedrock:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 `bedrock:GetWidget` 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

## 我无权执行 iam : PassRole

如果您收到一条错误信息，指明您无权执行 `iam:PassRole` 操作，则必须更新策略以使其允许您将角色传递给 Amazon Bedrock。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon Bedrock 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

## 我想允许我以外的人访问我 AWS 账户 的 Amazon Bedrock 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 ( ACL ) 的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon Bedrock 是否支持这些功能，请参阅 [Amazon Bedrock 如何与 IAM 搭配使用](#)。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户，请参阅 [IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问](#) 权限。

- 要了解如何向第三方提供对您的资源的访问[权限 AWS 账户](#)，请参阅 [IAM 用户指南中的向第三方提供访问权限](#)。AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的[为经过外部身份验证的用户 \(联合身份验证\) 提供访问权限](#)。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 [IAM 角色与基于资源的策略有何不同](#)。

## Amazon Bedrock 的合规性验证

要了解是否属于特定合规计划的范围，请参阅 AWS 服务“[按合规计划划分的范围](#)”，然后选择您感兴趣的合规计划。AWS 服务 有关一般信息，请参阅[AWS 合规计划AWS](#)。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的“[下载报告](#)”中的“[AWS Artifact](#)”。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。AWS 提供了以下资源来帮助实现合规性：

- [安全与合规性快速入门指南](#) — 这些部署指南讨论了架构注意事项，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 [Amazon Web Services 上构建 HIPAA 安全与合规性](#) — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

### Note

并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅[符合 HIPAA 要求的服务参考](#)。

- [AWS 合规资源](#) — 此工作簿和指南集可能适用于您的行业和所在地区。
- [AWS 客户合规指南](#) — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践，AWS 服务 并将指南映射到跨多个框架 (包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO) ) 的安全控制。
- [使用 AWS Config 开发人员指南中的规则评估资源](#) — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- [AWS Security Hub](#) — 这 AWS 服务 可以全面了解您的安全状态 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实操。有关受支持服务及控制的列表，请参阅 [Security Hub 控制参考](#)。

- [AWS Audit Manager](#)— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

## Amazon Bedrock 中的事件响应

AWS 非常重视安全性。作为 AWS 云[责任共担模式](#)的一部分，AWS 管理满足大多数安全敏感组织要求的数据中心、网络和软件架构。AWS 负责与 Amazon Bedrock 服务本身有关的任何事件响应。此外，作为 AWS 客户，您也有责任维护云端的安全。这意味着你可以从你有权访问的 AWS 工具和功能中控制你选择实施的安全性。此外，在分担责任模式中，你还要对事件响应负责。

通过建立符合云端运行应用程序目标的安全基准，您可以检测出可以响应的偏差。为了帮助您了解事件响应和您的选择对企业目标的影响，我们鼓励您查看以下资源：

- [AWS 安全事件响应指南](#)
- [AWS 安全、身份和合规性最佳实践](#)
- [AWS 云采用框架 \(CAF\) 的安全视角](#) 白皮书

## Amazon Bedrock 中的故障恢复能力

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。AWS 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域 和可用区的更多信息，请参阅[AWS 全球基础设施](#)。

## Amazon Bedrock 中的基础设施安全性

作为一项托管服务，Amazon Bedrock 受到 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅[AWS 云安全](#)。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS security Pillar Well-Architected Framework 中的[基础设施保护](#)。

您可以使用 AWS 已发布的 API 调用通过网络访问 Amazon Bedrock。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE (临时 Diffie-Hellman) 或 ECDHE (临时椭圆曲线 Diffie-Hellman)。大多数现代系统 (如 Java 7 及更高版本) 都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 [AWS Security Token Service](#) ( AWS STS ) 生成临时安全凭证来对请求进行签名。

## 防止跨服务混淆座席

混淆座席问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。一个服务（呼叫服务）调用另一项服务（所谓的“服务”）时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全局条件上下文键，以限制 Amazon Bedrock 为其他服务提供的资源权限。如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符 (\*) 的 `aws:SourceArn` 全局上下文条件键。例如，`arn:aws:bedrock:*:123456789012:*`。

如果 `aws:SourceArn` 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文键来限制权限。

`aws:SourceArn` 的值必须为 `ResourceDescription`。

以下示例演示如何使用 Bedrock 中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键防止出现代理混淆问题。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "aws:SourceAccount": "111122223333"
 }
 }
 }
]
}
```

```

 },
 "ArnEquals": {
 "aws:SourceArn": "arn:aws:bedrock:us-east-1:111122223333:model-
customization-job/*"
 }
 }
]
}

```

## Amazon Bedrock 中的配置和漏洞分析

配置和 IT 控制由您 ( 我们的客户 ) 共同 AWS 负责。有关更多信息，请参阅[责任 AWS 共担模型](#)。

### 使用接口 VPC 端点 (AWS PrivateLink)

您可以使用 AWS PrivateLink 在您的 VPC 和 Amazon Bedrock 之间创建私有连接。您可以像在您的 VPC 中一样访问 Amazon Bedrock，无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 Amazon Bedrock。

您可以通过创建由 AWS PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管式网络接口，用作发往 Amazon Bedrock 的流量的入口点。

有关更多信息，请参阅[AWS PrivateLink 指南](#)中的[AWS 服务 通过访问](#)。

### Amazon Bedrock VPC 端点注意事项

在为 Amazon Bedrock 设置接口端点之前，请查看《[AWS PrivateLink 指南](#)》中的[注意事项](#)。

Amazon Bedrock 支持通过 VPC 端点进行以下 API 调用。

| 类别                                           | 端点前缀            |
|----------------------------------------------|-----------------|
| <a href="#">Amazon Bedrock 控制面板 API 操作</a>   | bedrock         |
| <a href="#">Amazon Bedrock 运行时 API 操作</a>    | bedrock-runtime |
| <a href="#">Amazon Bedrock 构建时 API 操作的代理</a> | bedrock-agent   |

| 类别                                          | 端点前缀                  |
|---------------------------------------------|-----------------------|
| <a href="#">Amazon Bedrock 代理运行时 API 操作</a> | bedrock-agent-runtime |

## 可用区

Amazon Bedrock 和 Amazon Bedrock 终端节点代理可在多个可用区域使用。

## 为 Amazon Bedrock 创建接口端点

您可以使用亚马逊 VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 Amazon Bedrock 创建接口终端节点。有关更多信息，请参阅《AWS PrivateLink 指南》中的[创建接口端点](#)。

使用以下任一服务名称为 Amazon Bedrock 创建接口端点：

- com.amazonaws.*region*.bedrock
- com.amazonaws.*region*.bedrock-runtime
- com.amazonaws.*region*.bedrock-agent
- com.amazonaws.*region*.bedrock-agent-runtime

创建终端节点后，您可以选择启用私有 DNS 主机名。在创建 VPC 终端节点时，通过在 VPC 控制台中选择启用私有 DNS 名称，可启用此设置。

如果为接口端点启用私有 DNS，则可以使用区域的默认 DNS 名称向 Amazon Bedrock 发出 API 请求。以下示例显示了默认区域 DNS 名称的格式。

- bedrock.*region*.amazonaws.com
- bedrock-runtime.*region*.amazonaws.com
- bedrock-agent.*region*.amazonaws.com
- bedrock-agent-runtime.*region*.amazonaws.com

## 为接口端点创建端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略提供通过接口端点访问 Amazon Bedrock 的完全访问权限。要控制允许从 VPC 访问 Amazon Bedrock 的访问权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体 ( AWS 账户、IAM 用户和 IAM 角色 ) 。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《AWS PrivateLink 指南》中的[使用端点策略控制对服务的访问权限](#)。

示例：适用于 Amazon Bedrock 操作的 VPC 端点策略

以下是自定义端点策略的示例。当您将此基于资源的策略附加到接口终端节点时，它会向所有委托人授予所有资源上列出的 Amazon Bedrock 操作的访问权限。

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



# 监控 Amazon Bedrock

您可以通过亚马逊 CloudWatch 和亚马逊监控 Amazon Bedrock。EventBridge

## 主题

- [模型调用日志记录](#)
- [使用亚马逊监控亚马逊 Bedrock CloudWatch](#)
- [监控亚马逊中的亚马逊 Bedrock 事件 EventBridge](#)
- [使用记录亚马逊 Bedrock API 调用 AWS CloudTrail](#)

## 模型调用日志记录

模型调用日志可用于收集您在 Amazon Bedrock 中使用的所有调用的调用 AWS 账户 日志、模型输入数据和模型输出数据。默认情况下，日志记录处于禁用状态。

通过调用日志记录，您可以收集与在您的账户中执行的所有调用相关的完整请求数据、响应数据和元数据。可以配置日志记录以提供日志数据发布到的目标资源。支持的目标包括亚马逊 CloudWatch 日志和亚马逊简单存储服务 (Amazon S3) Service。仅支持来自相同账户和区域的目标。

在启用调用日志之前，您需要设置 Amazon S3 或 CloudWatch 日志目标。您可以通过控制台或 API 启用调用日志记录。

## 主题

- [设置 Amazon S3 目标](#)
- [设置 CloudWatch 日志目标](#)
- [使用 控制台](#)
- [将 API 与调用日志记录搭配使用](#)

## 设置 Amazon S3 目标

您可以按照以下步骤在 Amazon Bedrock 中设置 S3 日志记录目标：

1. 创建要将日志传输到 S3 存储桶。
2. 向其添加诸如下面的存储桶策略（替换 *accountId*、*region*、*bucketName* 以及可选 *prefix* 的值）：



**Note**

当您使用权限 `S3:GetBucketPolicy` 和 `S3:PutBucketPolicy` 配置日志记录时，系统会代表您自动将存储桶策略附加到存储桶。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AmazonBedrockLogsWrite",
 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 },
 "Action": [
 "s3:PutObject"
],
 "Resource": [
 "arn:aws:s3:::bucketName/prefix/AWSLogs/accountId/BedrockModelInvocationLogs/*"
],
 "Condition": {
 "StringEquals": {
 "aws:SourceAccount": "accountId"
 },
 "ArnLike": {
 "aws:SourceArn": "arn:aws:bedrock:region:accountId:*"
 }
 }
 }
]
}
```

3. (可选) 如果在存储桶上配置 SSE-KMS，请在 KMS 密钥上添加以下策略：

```
{
 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 }
}
```

```

 },
 "Action": "kms:GenerateDataKey",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "aws:SourceAccount": "accountId"
 },
 "ArnLike": {
 "aws:SourceArn": "arn:aws:bedrock:region:accountId:"
 }
 }
 }
}

```

有关 S3 SSE-KMS 配置的更多信息，请参阅[指定 KMS 加密](#)。

#### Note

必须禁用存储桶 ACL 才能使存储桶策略生效。有关更多信息，请参阅[禁用所有新存储桶的 ACL 并强制执行对象所有权](#)。

## 设置 CloudWatch 日志目标

您可以按照以下步骤设置用于登录 Amazon Bedrock 的 Amazon Log CloudWatch s 目标：

1. 创建 CloudWatch 将在其中发布日志的日志组。
2. 创建具有以下 CloudWatch 日志权限的 IAM 角色。

可信实体：

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "bedrock.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {

```

```

 "aws:SourceAccount": "accountId"
 },
 "ArnLike": {
 "aws:SourceArn": "arn:aws:bedrock:region:accountId:*"
 }
}
]
}

```

角色策略：

```

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

```

有关为 CloudWatch 日志设置 SSE 的更多信息，请参阅[使用 AWS Key Management Service 加密 CloudWatch 日志中的日志数据](#)。

## 使用 控制台

要启用模型调用日志记录，请拖动设置页面中日志记录切换开关旁边的滑块按钮。日志记录的其他配置设置将显示在面板上。

选择要将哪些数据请求和响应发布到日志中。您可以选择以下输出选项的任何组合：

- 文本
- 图像

- 嵌入

选择日志的发布位置：

- 仅限 Amazon S3
- CloudWatch 仅限日志
- Amazon S3 和 CloudWatch 日志

调用 CloudWatch 日志以及小型输入和输出数据支持 Amazon S3 和日志目标。对于大型输入和输出数据或二进制图像输出，仅支持 Amazon S3。以下详细信息总结了目标位置的数据表示形式。

- S3 目标 — Gzip 类型的 JSON 文件（每个文件都包含一批调用日志记录）被传输到指定的 S3 存储桶。与 CloudWatch 日志事件类似，每条记录都将包含调用元数据以及大小不超过 100 KB 的输入和输出 JSON 正文。大于 100 KB 的二进制数据或 JSON 正文将作为单个对象上传到相应数据前缀下的指定 Amazon S3 存储桶中。可以使用 Amazon S3 Select 和 Amazon Athena 对数据进行查询，也可以使用 AWS Glue 对 ETL 进行编目。数据可以加载到 OpenSearch 服务中，也可以由任何 Amazon EventBridge 目标进行处理。
- CloudWatch 日志目标-JSON 调用日志事件被传送到日志中的 CloudWatch 指定日志组。日志事件包含调用元数据以及大小不超过 100 KB 的输入和输出 JSON 正文。如果提供了用于传输大量数据的 Amazon S3 位置，则大于 100 KB 的二进制数据或 JSON 正文将改为上传到数据前缀下的 Amazon S3 存储桶。可以使用 Logs Insights 查询数据，也可以使用 CloudWatch 日志进一步实时流式传输到各种服务。CloudWatch

## 将 API 与调用日志记录搭配使用

可以使用以下 API 配置模型调用日志记录：

- PutModelInvocationLoggingConfiguration
- GetModelInvocationLoggingConfiguration
- DeleteModelInvocationLoggingConfiguration

有关如何将 API 和调用日志记录搭配使用的更多信息，请参阅《Bedrock API 指南》。

## 使用亚马逊监控亚马逊 Bedrock CloudWatch

您可以使用 Amazon 监控 Amazon Bedrock CloudWatch，亚马逊会收集原始数据并将其处理为可读的近乎实时的指标。您可以使用 CloudWatch 控制台绘制指标的图表。此外，您还可以设置警报来监视特定阈值，并在值超出这些阈值时发送通知或采取措施。

有关更多信息，请参阅《[亚马逊 CloudWatch 用户指南](#)》CloudWatch 中的“[什么是亚马逊](#)”。

### 主题

- [运行时指标](#)
- [记录 CloudWatch 指标](#)
- [使用 Amazon Bedrock 的 CloudWatch 指标](#)
- [查看 Amazon Bedrock 指标](#)

## 运行时指标

下表描述了 Amazon Bedrock 提供的运行时指标。

| 指标名称                   | 单位           | 描述                                                                                        |
|------------------------|--------------|-------------------------------------------------------------------------------------------|
| Invocations            | SampleCount  | 对 <a href="#">InvokeModel</a> 或 <a href="#">InvokeModelWithResponseStream</a> API 操作的请求数。 |
| InvocationLatency      | Milliseconds | 调用的延迟时间。                                                                                  |
| InvocationClientErrors | SampleCount  | 导致发生客户端错误的调用次数。                                                                           |
| InvocationServerErrors | SampleCount  | 导致 AWS 服务器端错误的调用次数。                                                                       |
| InvocationThrottles    | SampleCount  | 系统限制的调用次数。                                                                                |
| InputTokenCount        | SampleCount  | 文本输入的词元数。                                                                                 |
| LegacyModelInvocations | SampleCount  | 使用 <a href="#">旧版</a> 模型的调用次数                                                             |

| 指标名称             | 单位          | 描述        |
|------------------|-------------|-----------|
| OutputTokenCount | SampleCount | 文本输出的词元数。 |
| OutputImageCount | SampleCount | 输出映像数。    |

## 记录 CloudWatch 指标

对于每次成功或失败的配送尝试，都会在命名空间AWS/Bedrock和Across all model IDs维度下发出以下 Amazon CloudWatch 指标：

- ModelInvocationLogsCloudWatchDeliverySuccess
- ModelInvocationLogsCloudWatchDeliveryFailure
- ModelInvocationLogsS3DeliverySuccess
- ModelInvocationLogsS3DeliveryFailure
- ModelInvocationLargeDataS3DeliverySuccess
- ModelInvocationLargeDataS3DeliveryFailure

如果由于权限配置错误或暂时性故障而导致无法传输日志，则会定期重试，最长是在 24 小时后重试。

## 使用 Amazon Bedrock 的 CloudWatch 指标

要检索 Amazon Bedrock 操作的指标，请指定以下信息：

- 指标维度。维度是用于标识指标的一组名称-值对。Amazon Bedrock 支持以下维度：
  - ModelId – 所有指标
  - ModelId + ImageSize + BucketedStepSize – OutputImageCount
- 指标名称，如 InvocationClientErrors。

您可以通过 AWS Management Console、或 CloudWatch API 获取 Amazon Bedrock 的指标。AWS CLI您可以通过其中一个 AWS 软件开发套件 (SDK) 或 CloudWatch API 工具来使用 API。  
CloudWatch

您必须拥有相应的 CloudWatch 权限才能监控 Amazon Bedrock。CloudWatch 有关更多信息，请参阅亚马逊 CloudWatch 用户指南 CloudWatch 中的[亚马逊身份验证和访问控制](#)。

## 查看 Amazon Bedrock 指标

在 CloudWatch 控制台中查看 Amazon Bedrock 指标。

查看指标 ( CloudWatch 控制台 )

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，[网址为 https://console.aws.amazon.com/cloudwatch/](https://console.aws.amazon.com/cloudwatch/)。
2. 选择“指标”，选择“所有指标”，然后搜索ModelId。

## 监控亚马逊中的亚马逊 Bedrock 事件 EventBridge

您可以使用亚马逊监控 Amazon EventBridge Bedrock 中的状态变更事件。借助亚马逊 EventBridge，您可以 SageMaker 将亚马逊配置为自动响应 Amazon Bedrock 中模型定制任务状态的变化。来自亚马逊 Bedrock 的活动以近乎实时 EventBridge 的方式发送到亚马逊。您可以编写简单规则，以在事件匹配规则时执行自动执行操作。如果您将亚马逊 EventBridge 与 Amazon Bedrock 配合使用，则可以：

- 在您触发的模型自定义作业发生状态更改事件时发布通知，而无论将来是否添加新的异步工作流。发布的事件应为您提供足够的信息，以便您对下游工作流中的事件做出反应。
- 在不调用 GetModelCustomizationJob API 的情况下提供任务状态更新，这可能意味着要处理 API 速率限制问题、API 更新和减少额外的计算资源。

从 Amazon 接收 AWS 活动不收取任何费用 EventBridge。有关亚马逊的更多信息 EventBridge，请参阅 [Amazon EventBridge](#)

### Note

- Amazon Bedrock 会尽最大努力发出事件。活动以近乎实时 EventBridge 的方式发送到 Amazon。借助 Amazon EventBridge，您可以创建触发程序操作以响应事件的规则。例如，您可以配置一条规则，以调用 SNS 主题来发送电子邮件通知或调用函数来执行某操作。有关更多信息，请参阅 Amazon EventBridge 用户指南。
- 每当您触发的模型自定义作业发生状态更改时，Amazon Bedrock 都会创建一个新事件，并尽最大努力传输此类事件。

### 主题

- [工作方式](#)
- [EventBridge 架构](#)
- [规则和目标](#)
- [创建规则以处理 Amazon Bedrock 事件](#)

## 工作方式

要接收来自 Amazon Bedrock 的事件，您需要创建规则和目标，以便通过亚马逊 EventBridge 匹配、接收和处理状态变化数据。Amazon EventBridge 是一个无服务器事件总线，用于接收来自 AWS 服务、SaaS 合作伙伴和客户应用程序的更改状态事件。它根据您创建的规则或模式处理事件，并将这些事件路由到您选择的一个或多个“目标”AWS Lambda，例如亚马逊简单队列服务和亚马逊简单通知服务。

每当模型定制任务的状态发生变化 EventBridge 时，Amazon Bedrock 都会通过亚马逊发布您的事件。在每种情况下，都会创建一个新事件并将其发送到 Amazon EventBridge，然后亚马逊会将该事件发送到您的默认事件总线。该事件显示哪个自定义作业的状态已更改，以及该作业的当前状态。当亚马逊 EventBridge 收到与您创建的规则匹配的事件时，亚马逊会将其 EventBridge 路由到您指定的目标。创建规则时，您可以根据事件的内容配置这些目标以及下游 workflows。

## EventBridge 架构

事件架构中的以下事件字段特定于 Amazon Bedrock。EventBridge

- `jobArn` — 模型自定义作业的 ARN。
- `outputModelArn` — 输出模型的 ARN。在训练作业完成时发布。
- `jobStatus` — 作业的当前状态。
- `FailureMessage` — 失败消息。在训练作业失败时发布。

## 事件示例

以下是失败的模型自定义作业的示例事件 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": ["arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghijklmxyz"],
"detail": {
 "version": "0.0",
 "jobName": "abcd-wxyz",
 "jobArn": "arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghijklmxyz",
 "outputModelName": "dummy-output-model-name",
 "outputModelArn": "arn:aws:bedrock:us-east-1:123412341234:dummy-output-model-
name",
 "roleArn": "arn:aws:iam::123412341234:role/JobExecutionRole",
 "jobStatus": "Failed",
 "failureMessage": "Failure Message here.",
 "creationTime": "2023-08-11T10:11:12Z",
 "lastModifiedTime": "2023-08-11T12:34:56Z",
 "endTime": "2023-08-11T12:34:56Z",
 "baseModelArn": "arn:aws:bedrock:us-east-1:123412341234:base-model-name",
 "hyperParameters": {
 "batchSize" : "batchSizeNumberUsed",
 "epochCount": "epochCountNumberUsed",
 "learningRate": "learningRateUsed",
 "learningRateWarmupSteps": "learningRateWarmupStepsUsed"
 },
 "trainingDataConfig": {
 "s3Uri": "s3://bucket/key",
 },
 "validationDataConfig": {
 "s3Uri": "s3://bucket/key",
 },
 "outputDataConfig": {
 "s3Uri": "s3://bucket/key",
 }
}
}
```

## 规则和目标

当传入事件与您创建的规则匹配时，这些事件将路由到您为该规则指定的目标，然后由该目标处理这些事件。目标支持 JSON 格式，可以包括亚马逊 EC2 实例、Lambda 函数、Kinesis 流、亚马逊 ECS 任务、Step Functions、亚马逊 SNS 主题和亚马逊 SQS 等 AWS 服务。要正确接收和处理事件，您需要

创建用于匹配、接收和正确处理事件数据的规则和目标。您可以通过 Amazon EventBridge 控制台或通过创建这些规则和目标 AWS CLI。

## 示例规则

该规则匹配 source ["aws.bedrock"] 发出的事件模式。该规则会捕获亚马逊 EventBridge 发送的所有源为 "aws.bedrock" 的事件到您的默认事件总线。

```
{
 "source": ["aws.bedrock"]
}
```

## 目标

在 Amazon 中创建规则时 EventBridge，您需要指定一个目标，EventBridge 将与您的规则模式匹配的事件发送到该目标。这些目标可以是 SageMaker 管道、Lambda 函数、SNS 主题、SQS 队列或当前支持的任何其他目标。EventBridge 您可以参阅 Amazon EventBridge 文档，了解如何为事件设定目标。有关如何使用 Amazon Simple Notification Service 作为目标的步骤，请参阅[创建规则以处理 Amazon Bedrock 事件](#)。

## 创建规则以处理 Amazon Bedrock 事件

要接收有关您的 Amazon Bedrock 事件的电子邮件通知，请完成以下过程。

### 创建 Amazon Simple Notification Service 主题

1. 通过 <https://console.aws.amazon.com/sns/v3/home> 打开 Amazon SNS 控制台。
2. 在导航窗格中，选择 Topics ( 主题 )。
3. 选择创建主题。
4. 对于类型，选择标准。
5. 对于 Name ( 名称 )，请为主题输入一个名称。
6. 选择创建主题。
7. 选择创建订阅。
8. 对于协议，选择电子邮件。
9. 对于 Endpoint ( 端点 )，请输入接收通知的电子邮件地址。
10. 选择创建订阅。

11. 您将收到电子邮件消息，其主题为：AWS Notification - Subscription Confirmation。请按照说明确认订阅。

使用以下过程创建用于处理 Amazon Bedrock 事件的规则。

创建规则以处理 Amazon Bedrock 事件

1. 打开亚马逊 EventBridge 控制台，[网址为 https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/)。
2. 选择 创建规则。
3. 对于 Name ( 名称 )，请为规则输入一个名称。
4. 对于规则类型，选择具有事件模式的规则。
5. 选择下一步。
6. 对于 Event pattern ( 事件模式 )，执行以下操作：
  - a. 对于事件源，选择 AWS 服务。
  - b. 对于 AWS 服务，选择 Amazon Bedrock。
  - c. 对于事件类型，选择模型自定义作业状态更改。
  - d. 默认情况下，我们会就每个事件发送通知。如果您愿意，可以创建一种事件模式来筛选特定作业状态的事件。
  - e. 选择下一步。
7. 按以下操作指定目标：
  - a. 对于目标类型，选择 AWS 服务。
  - b. 对于 Select a target ( 选择一个目标 )，选择 SNS topic ( SNS 主题 )。
  - c. 对于主题，选择您为通知创建的 SNS 主题。
  - d. 选择下一步。
8. ( 可选 ) 向规则添加标签。
9. 选择下一步。
10. 选择 创建规则。

## 使用记录亚马逊 Bedrock API 调用 AWS CloudTrail

Amazon Bedrock 与 AWS CloudTrail 一项服务集成，该服务提供用户、角色或 AWS 服务在 Amazon Bedrock 中采取的操作的记录。CloudTrail 将 Amazon Bedrock 的所有 API 调用捕获为事件。捕获

的调用中包括从 Amazon Bedrock 控制台进行的调用和通过代码对 Amazon Bedrock API 操作进行的调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括针对 Amazon Bedrock 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的“事件历史记录”中查看最新的事件。通过收集的信息 CloudTrail，您可以确定向 Amazon Bedrock 发出的请求、发出请求的 IP 地址、谁提出了请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅 [《AWS CloudTrail 用户指南》](#)。

## Amazon Bedrock 信息位于 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当 Amazon Bedrock 中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅[使用事件历史记录查看 CloudTrail 事件](#)。

要持续记录您的事件 AWS 账户，包括 Amazon Bedrock 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- [创建跟踪记录概览](#)
- [CloudTrail 支持的服务和集成](#)
- [配置 Amazon SNS 通知 CloudTrail](#)
- [接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件](#)

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 [CloudTrail userIdentity 元素](#)。

## Amazon Bedrock 中的数据事件 CloudTrail

[数据事件](#)可提供对资源或在资源中所执行资源操作（例如，读取或写入 Amazon S3 对象）的相关信息。这些也称为数据层面操作。数据事件通常是大容量活动，默认情况下 CloudTrail 不记录。

Amazon Bedrock 不会记录 [Amazon Bedrock 运行时 API 操作](#) ( `InvokeModel` 和 `InvokeModelWithResponseStream` )。

Amazon Bedrock 将 [Amazon Bedrock 运行时 API 的所有代理操作操作](#) 记录 CloudTrail 为数据事件。

- 要记录 [InvokeAgent](#) 呼叫，请配置高级事件选择器以记录 `AWS::Bedrock::AgentAlias` 资源类型的数据事件。
- 要记录 [Retrieve](#) 和 [RetrieveAndGenerate](#) 调用，请配置高级事件选择器以记录 `AWS::Bedrock::KnowledgeBase` 资源类型的数据事件。

在 CloudTrail 控制台中，为数据事件类型选择 Bedrock 代理别名或 Bedrock 知识库。此外，您可以通过选择自定义日志选择器模板，按 `eventName` 和 `resources.ARN` 字段进行筛选。有关更多信息，请参阅 [Logging data events with the AWS Management Console](#)。

从中 AWS CLI，将 `resource.type` 值设置为等于 `AWS::Bedrock::AgentAlias` 或 `AWS::Bedrock::KnowledgeBase` 并将 `eventCategory` 等于设置为 `Data`。有关更多信息，请参阅 [使用 AWS CLI 记录数据事件](#)。

以下示例展示了如何在 AWS CLI 中配置跟踪来记录所有 Amazon Bedrock 资源类型的所有 Amazon Bedrock 数据事件。

```
aws cloudtrail put-event-selectors --trail-name trailName \
--advanced-event-selectors \
'[
 {
 "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"] }
]
 }
]'
```

此外，您还可以按 `eventName` 和 `resources.ARN` 字段进行筛选。有关这些字段的更多信息，请参阅 [AdvancedFieldSelector](#)。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅 [AWS CloudTrail 定价](#)。

## 中的 Amazon Bedrock 管理活动 CloudTrail

[管理事件](#) 提供有关对您 AWS 账户中的资源执行的管理操作的信息。这些也称为控制平面操作。CloudTrail 默认情况下会记录管理事件 API 操作。

Amazon Bedrock 将剩余的 Amazon Bedrock API 操作记录为管理事件。有关 Amazon Bedrock 登录的亚马逊 Bedrock API 操作的列表 CloudTrail，请参阅亚马逊 Bedrock API 参考中的以下页面。

所有 [亚马逊 Bedrock API 操作](#) 和 [亚马逊 Bedrock API 操作的代理](#) 均由《[亚马逊 Bedrock API 参考](#)》记录 CloudTrail 并记录在案。例如，对 `InvokeModelStopModelCustomizationJob`、和 `CreateAgent` 操作的调用会在 CloudTrail 日志文件中生成条目。

## 了解 Amazon Bedrock 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该 `InvokeModel` 操作的 CloudTrail 日志条目。

```
{
 "eventVersion": "1.08",
 "userIdentity": {
 "type": "IAMUser",
 "principalId": "AROAIKFHPEXAMPLE",
 "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"
}
}
```

# 使用 AWS 软件开发工具包的 Amazon Bedrock 的代码示例

以下代码示例展示了如何将 Amazon Bedrock 与 AWS 软件开发套件 (SDK) 一起使用。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 代码示例

- [使用 AWS 软件开发工具包的 Amazon Bedrock 的代码示例](#)
  - [使用 AWS 软件开发工具包对 Amazon Bedrock 执行的操作](#)
    - [使用 SDK 获取有关 Amazon Bedrock 基础模型的 AWS 详细信息](#)
    - [使用软件开发工具包列出可用的 Amazon Bedrock 基础模型 AWS](#)
  - [使用 AWS SDK 的 Amazon Bedrock 场景](#)
    - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)
- [使用软件开发工具包的 Amazon Bedrock Runtime AWS 代码示例](#)
  - [使用软件开发工具包对 Amazon Bedrock 运行时的 AWS 操作](#)
    - [在 Amazon Bedrock 上调用 Amazon Titan Image Generator G1 模型来生成图像](#)
    - [在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 模型来生成图像](#)
    - [使用多模式提示在 Amazon Bedrock 上调用 Anthropic Claude 3](#)
    - [在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本](#)
    - [在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型生成文本](#)
    - [在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本](#)
    - [在亚马逊 Bedrock 上调用 Anthropic Claude 3 生成文本](#)
    - [在 Amazon Bedrock 上调用 Anthropic Claude Instant 模型进行文本](#)
    - [在 Amazon Bedrock 上调用 Anthropic Claude 以生成带有响应流的文本](#)
    - [在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本](#)
    - [在 Amazon Bedrock 上调用 Mistral 7B 模型生成文本](#)
    - [在 Amazon Bedrock 上调用 Mixtral 8x7b 模型生成文本](#)
  - [使用 SDK 的 Amazon Bedrock Runtime AWS 场景](#)
    - [使用软件开发工具包创建提供游乐场的示例应用程序，以便与 Amazon Bedrock 基础模型进行交互 AWS](#)
    - [适用于 Amazon Bedrock 的交互式文本生成平台](#)



- [在 Amazon Bedrock 上调用多个大型语言模型 \( LLM \)](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)
- [使用 AWS 软件开发工具包的 Amazon Bedrock 代理的代码示例](#)
  - [使用 AWS SDK 为 Amazon Bedrock 的代理提供的操作](#)
    - [使用软件开发工具包创建 Amazon Bedrock 代理 AWS](#)
    - [使用软件开发工具包创建 Amazon Bedrock 代理操作组 AWS](#)
    - [使用软件开发工具包创建 Amazon Bedrock 代理别名 AWS](#)
    - [使用软件开发工具包删除 Amazon Bedrock 代理 AWS](#)
    - [使用软件开发工具包删除 Amazon Bedrock 代理别名 AWS](#)
    - [使用软件开发工具包获取有关亚马逊 Bedrock 代理的信息 AWS](#)
    - [列出使用软件开发工具包的 Amazon Bedrock 代理的 AWS 操作组](#)
    - [使用软件开发工具包列出属于账户的 Amazon Bedrock 代理 AWS](#)
    - [列出与使用软件开发工具包的 Amazon Bedrock 代理相关的知识库 AWS](#)
    - [使用软件开发工具包准备 Amazon Bedrock 代理 AWS](#)
  - [使用 AWS SDK 的 Amazon Bedrock 代理的场景](#)
    - [演 end-to-end 示如何使用软件开发工具包创建和调用 Amazon Bedrock 代理的 AWS 示例](#)
    - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)
- [使用 AWS 软件开发工具包的 Amazon Bedrock Runtime 代理的代码示例](#)
  - [使用 AWS 软件开发工具包为 Amazon Bedrock Runtime 代理执行的操作](#)
    - [使用软件开发工具包调用 Amazon Bedrock 代理 AWS](#)
  - [使用 SDK 的 Amazon Bedrock Runtime AWS 代理的场景](#)
    - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用 AWS 软件开发工具包的 Amazon Bedrock 的代码示例

以下代码示例展示了如何将 Amazon Bedrock 与 AWS 软件开发套件 (SDK) 一起使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

**场景**是展示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 开始使用

### 开始使用 Amazon Bedrock

以下代码示例演示了如何开始使用 Amazon Bedrock。

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace ListFoundationModelsExample
{
 /// <summary>
 /// This example shows how to list foundation models.
 /// </summary>
 internal class HelloBedrock
 {
 /// <summary>
 /// Main method to call the ListFoundationModelsAsync method.
 /// </summary>
 /// <param name="args"> The command line arguments. </param>
 static async Task Main(string[] args)
 {
 // Specify a region endpoint where Amazon Bedrock is available.
 For a list of supported region see https://docs.aws.amazon.com/bedrock/latest/
 userguide/what-is-bedrock.html#bedrock-regions
 AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

 await ListFoundationModelsAsync(bedrockClient);
 }
 }
}
```

```
 }

 /// <summary>
 /// List foundation models.
 /// </summary>
 /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
 private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
 {
 Console.WriteLine("List foundation models with no filter");

 try
 {
 ListFoundationModelsResponse response = await
bedrockClient.ListFoundationModelsAsync(new ListFoundationModelsRequest()
 {
 });

 if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 foreach (var fm in response.ModelSummaries)
 {
 WriteToConsole(fm);
 }
 }
 else
 {
 Console.WriteLine("Something wrong happened");
 }
 }
 catch (AmazonBedrockException e)
 {
 Console.WriteLine(e.Message);
 }
 }

 /// <summary>
 /// Write the foundation model summary to console.
 /// </summary>
 /// <param name="foundationModel"> The foundation model summary to write
to console. </param>
```

```
private static void WriteToConsole(FoundationModelSummary
foundationModel)
{
 Console.WriteLine($"{foundationModel.ModelId}, Customization:
{String.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
{foundationModel.ResponseStreamingSupported}, Input: {String.Join(",
", foundationModel.InputModalities)}, Output: {String.Join(", ",
foundationModel.OutputModalities)}");
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFoundationModels](#) 中的。

## Go

### 适用于 Go V2 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/bedrock"
)

const region = "us-east-1"

// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock client and
// list the available foundation models in your account and the chosen region.
// This example uses the default settings specified in your shared credentials
// and config files.
```

```
func main() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO(),
config.WithRegion(region))
 if err != nil {
 fmt.Println("Couldn't load default configuration. Have you set up your
AWS account?")
 fmt.Println(err)
 return
 }
 bedrockClient := bedrock.NewFromConfig(sdkConfig)
 result, err := bedrockClient.ListFoundationModels(context.TODO(),
&bedrock.ListFoundationModelsInput{})
 if err != nil {
 fmt.Printf("Couldn't list foundation models. Here's why: %v\n", err)
 return
 }
 if len(result.ModelSummaries) == 0 {
 fmt.Println("There are no foundation models.")}
 for _, modelSummary := range result.ModelSummaries {
 fmt.Println(*modelSummary.ModelId)
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考[ListFoundationModels](#)中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

import { fileURLToPath } from "url";
```

```
import {
 BedrockClient,
 ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });

export const main = async () => {
 const command = new ListFoundationModelsCommand({});

 const response = await client.send(command);
 const models = response.modelSummaries;

 console.log("Listing the available Bedrock foundation models:");

 for (let model of models) {
 console.log("=".repeat(42));
 console.log(` Model: ${model.modelId}`);
 console.log("-".repeat(42));
 console.log(` Name: ${model.modelName}`);
 console.log(` Provider: ${model.providerName}`);
 console.log(` Model ARN: ${model.modelArn}`);
 console.log(` Input modalities: ${model.inputModalities}`);
 console.log(` Output modalities: ${model.outputModalities}`);
 console.log(` Supported customizations: ${model.customizationsSupported}`);
 console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
 console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
 console.log("=".repeat(42) + "\n");
 }

 const active = models.filter(
 (m) => m.modelLifecycle.status === "ACTIVE",
).length;
 const legacy = models.filter(
 (m) => m.modelLifecycle.status === "LEGACY",
).length;

 console.log(
 `There are ${active} active and ${legacy} legacy foundation models in ${REGION}.`,
);

 return response;
}
```

```
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 await main();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [ListFoundationModels](#) 中的。

## 代码示例

- [使用 AWS 软件开发工具包对 Amazon Bedrock 执行的操作](#)
  - [使用 SDK 获取有关 Amazon Bedrock 基础模型的 AWS 详细信息](#)
  - [使用软件开发工具包列出可用的 Amazon Bedrock 基础模型 AWS](#)
- [使用 AWS SDK 的 Amazon Bedrock 场景](#)
  - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用 AWS 软件开发工具包对 Amazon Bedrock 执行的操作

以下代码示例演示了如何使用 AWS 软件开发工具包执行单个 Amazon Bedrock 操作。这些摘录称为 Amazon Bedrock API，是必须在上下文中运行的大型程序的代码摘录。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。如需完整列表，请参阅 [Amazon Bedrock API 参考](#)。

### 示例

- [使用 SDK 获取有关 Amazon Bedrock 基础模型的 AWS 详细信息](#)
- [使用软件开发工具包列出可用的 Amazon Bedrock 基础模型 AWS](#)

## 使用 SDK 获取有关 Amazon Bedrock 基础模型的 AWS 详细信息

以下代码示例展示了如何获取有关 Amazon Bedrock 基础模型的详细信息。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用同步 Amazon Bedrock 客户端获取有关基础模型的详细信息。

```
/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @param bedrockClient The service client for accessing Amazon Bedrock.
 * @param modelIdentifier The model identifier.
 * @return An object containing the foundation model's details.
 */
public static FoundationModelDetails getFoundationModel(BedrockClient
bedrockClient, String modelIdentifier) {
 try {
 GetFoundationModelResponse response =
bedrockClient.getFoundationModel(
 r -> r.modelIdentifier(modelIdentifier)
);

 FoundationModelDetails model = response.modelDetails();

 System.out.println(" Model ID: " +
model.modelId());
 System.out.println(" Model ARN: " +
model.modelArn());
 System.out.println(" Model Name: " +
model.modelName());
 System.out.println(" Provider Name: " +
model.providerName());
 System.out.println(" Lifecycle status: " +
model.modelLifecycle().statusAsString());
 System.out.println(" Input modalities: " +
model.inputModalities());
 System.out.println(" Output modalities: " +
model.outputModalities());
 }
}
```



```

 System.out.println(" Supported customizations: " +
model.customizationsSupported());
 System.out.println(" Supported inference types: " +
model.inferenceTypesSupported());
 System.out.println(" Response streaming supported: " +
model.responseStreamingSupported());

 return model;

 } catch (ValidationException e) {
 throw new IllegalArgumentException(e.getMessage());
 } catch (SdkException e) {
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
 }
}

```

使用异步 Amazon Bedrock 客户端获取有关基础模型的详细信息。

```

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @param bedrockClient The async service client for accessing Amazon
Bedrock.
 * @param modelIdentifier The model identifier.
 * @return An object containing the foundation model's details.
 */
public static FoundationModelDetails getFoundationModel(BedrockAsyncClient
bedrockClient, String modelIdentifier) {
 try {
 CompletableFuture<GetFoundationModelResponse> future =
bedrockClient.getFoundationModel(
 r -> r.modelIdentifier(modelIdentifier)
);

 FoundationModelDetails model = future.get().modelDetails();

 System.out.println(" Model ID: " +
model.modelId());
 System.out.println(" Model ARN: " +
model.modelArn());
 }
}

```

```
 System.out.println(" Model Name: " +
model.modelName());
 System.out.println(" Provider Name: " +
model.providerName());
 System.out.println(" Lifecycle status: " +
model.modelLifecycle().statusAsString());
 System.out.println(" Input modalities: " +
model.inputModalities());
 System.out.println(" Output modalities: " +
model.outputModalities());
 System.out.println(" Supported customizations: " +
model.customizationsSupported());
 System.out.println(" Supported inference types: " +
model.inferenceTypesSupported());
 System.out.println(" Response streaming supported: " +
model.responseStreamingSupported());

 return model;

 } catch (ExecutionException e) {
 if (e.getMessage().contains("ValidationException")) {
 throw new IllegalArgumentException(e.getMessage());
 } else {
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
 }
 } catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [GetFoundationModel](#) 中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取有关基础模型的详细信息。

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

import { fileURLToPath } from "url";

import {
 BedrockClient,
 GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 models.
 */
export const getFoundationModel = async () => {
 const client = new BedrockClient();

 const command = new GetFoundationModelCommand({
 modelIdentifier: "amazon.titan-embed-text-v1",
 });

 const response = await client.send(command);

 return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const model = await getFoundationModel();
```

```
console.log(model);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考[GetFoundationModel](#)中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取有关基础模型的详细信息。

```
def get_foundation_model(self, model_identifier):
 """
 Get details about an Amazon Bedrock foundation model.

 :return: The foundation model's details.
 """

 try:
 return self.bedrock_client.get_foundation_model(
 modelIdentifier=model_identifier
)["modelDetails"]
 except ClientError:
 logger.error(
 f"Couldn't get foundation models details for {model_identifier}"
)
 raise
```

- 有关 API 的详细信息，请参阅适用[GetFoundationModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包列出可用的 Amazon Bedrock 基础模型 AWS

以下代码示例演示了如何列出可用的 Amazon Bedrock 基础模型。

.NET

AWS SDK for .NET

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Bedrock 基础模型。

```
/// <summary>
/// List foundation models.
/// </summary>
/// <param name="bedrockClient"> The Amazon Bedrock client. </param>
private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
{
 Console.WriteLine("List foundation models with no filter");

 try
 {
 ListFoundationModelsResponse response = await
bedrockClient.ListFoundationModelsAsync(new ListFoundationModelsRequest()
 {
 });

 if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 foreach (var fm in response.ModelSummaries)
 {
 WriteToConsole(fm);
 }
 }
 }
}
```

```
 else
 {
 Console.WriteLine("Something wrong happened");
 }
 }
 catch (AmazonBedrockException e)
 {
 Console.WriteLine(e.Message);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [ListFoundationModels](#) 中的。

## Go

### 适用于 Go V2 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Bedrock 基础模型。

```
// FoundationModelWrapper encapsulates Amazon Bedrock actions used in the
// examples.
// It contains a Bedrock service client that is used to perform foundation model
// actions.
type FoundationModelWrapper struct {
 BedrockClient *bedrock.Client
}

// ListPolicies lists Bedrock foundation models that you can use.
func (wrapper FoundationModelWrapper) ListFoundationModels()
([]types.FoundationModelSummary, error) {
```

```
var models []types.FoundationModelSummary

result, err := wrapper.BedrockClient.ListFoundationModels(context.TODO(),
&bedrock.ListFoundationModelsInput{})

if err != nil {
 log.Printf("Couldn't list foundation models. Here's why: %v\n", err)
} else {
 models = result.ModelSummaries
}
return models, err
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考[ListFoundationModels](#)中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用同步 Amazon Bedrock 客户端列出可用的 Amazon Bedrock 基础模型。

```
/**
 * Lists Amazon Bedrock foundation models that you can use.
 * You can filter the results with the request parameters.
 *
 * @param bedrockClient The service client for accessing Amazon Bedrock.
 * @return A list of objects containing the foundation models' details
 */
public static List<FoundationModelSummary> listFoundationModels(BedrockClient
bedrockClient) {

 try {
 ListFoundationModelsResponse response =
bedrockClient.listFoundationModels(r -> {});
```

```
 List<FoundationModelSummary> models = response.modelSummaries();

 if (models.isEmpty()) {
 System.out.println("No available foundation models in " +
region.toString());
 } else {
 for (FoundationModelSummary model : models) {
 System.out.println("Model ID: " + model.modelId());
 System.out.println("Provider: " + model.providerName());
 System.out.println("Name: " + model.modelName());
 System.out.println();
 }
 }

 return models;

 } catch (SdkClientException e) {
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
 }
}
```

使用异步 Amazon Bedrock 客户端列出可用的 Amazon Bedrock 基础模型。

```
/**
 * Lists Amazon Bedrock foundation models that you can use.
 * You can filter the results with the request parameters.
 *
 * @param bedrockClient The async service client for accessing Amazon
Bedrock.
 * @return A list of objects containing the foundation models' details
 */
public static List<FoundationModelSummary>
listFoundationModels(BedrockAsyncClient bedrockClient) {
 try {
 CompletableFuture<ListFoundationModelsResponse> future =
bedrockClient.listFoundationModels(r -> {});

 List<FoundationModelSummary> models = future.get().modelSummaries();

 if (models.isEmpty()) {
```



```
 System.out.println("No available foundation models in " +
region.toString());
 } else {
 for (FoundationModelSummary model : models) {
 System.out.println("Model ID: " + model.modelId());
 System.out.println("Provider: " + model.providerName());
 System.out.println("Name: " + model.modelName());
 System.out.println();
 }
 }

 return models;

} catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
} catch (ExecutionException e) {
 System.err.println(e.getMessage());
 throw new RuntimeException(e);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [ListFoundationModels](#) 中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的基础模型。

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

import { fileURLToPath } from "url";
```

```
import {
 BedrockClient,
 ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 * models.
 */
export const listFoundationModels = async () => {
 const client = new BedrockClient();

 const input = {
 // byProvider: 'STRING_VALUE',
 // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
 // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
 // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
 };

 const command = new ListFoundationModelsCommand(input);

 const response = await client.send(command);

 return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const models = await listFoundationModels();
 console.log(models);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [ListFoundationModels](#) 中的。

## Kotlin

### 适用于 Kotlin 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Amazon Bedrock 基础模型。

```
suspend fun listFoundationModels(): List<FoundationModelSummary>? {
 BedrockClient { region = "us-east-1" }.use { bedrockClient ->
 val response =
 bedrockClient.listFoundationModels(ListFoundationModelsRequest {})
 response.modelSummaries?.forEach { model ->
 println("=====")
 println(" Model ID: ${model.modelId}")
 println("-----")
 println(" Name: ${model.modelName}")
 println(" Provider: ${model.providerName}")
 println(" Input modalities: ${model.inputModalities}")
 println(" Output modalities: ${model.outputModalities}")
 println(" Supported customizations:
${model.customizationsSupported}")
 println(" Supported inference types:
${model.inferenceTypesSupported}")
 println("-----\n")
 }
 return response.modelSummaries
 }
 }
}
```

- 有关 API 的详细信息，请参阅适用 [ListFoundationModels](#) 于 Kotlin 的 AWS SDK API 参考。

## PHP

### 适用于 PHP 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Amazon Bedrock 基础模型。

```
public function listFoundationModels()
{
 $result = $this->bedrockClient->listFoundationModels();
 return $result;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考 [ListFoundationModels](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出可用的 Amazon Bedrock 基础模型。

```
def list_foundation_models(self):
 """
 List the available Amazon Bedrock foundation models.

 :return: The list of available bedrock foundation models.
 """
```

```
try:
 response = self.bedrock_client.list_foundation_models()
 models = response["modelSummaries"]
 logger.info("Got %s foundation models.", len(models))
 return models

except ClientError:
 logger.error("Couldn't list foundation models.")
 raise
```

- 有关 API 的详细信息，请参阅适用[ListFoundationModels](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 AWS SDK 的 Amazon Bedrock 场景

以下代码示例向您展示了如何使用 AWS 软件开发工具包在 Amazon Bedrock 中实现常见场景。这些场景向您展示了如何通过调用 Amazon Bedrock 中的多个函数来完成特定任务。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

### 示例

- [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序

以下代码示例展示了如何使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序。

### Python

#### SDK for Python (Boto3)

Amazon Bedrock Serverless Pro [AWS Step Functions](#) mpt Chaining 场景演示了如何使用 [Amazon Bedrock](#) 和 [Amazon Bedrock](#) 代理来构建和编排复杂、无服务器且高度可扩展的生成式人工智能应用程序。它包含以下工作示例：

- 为文学博客撰写对给定小说的分析。此示例说明了一个简单的、按顺序排列的提示链。

- 生成有关给定主题的短篇小说。此示例说明了 AI 如何迭代处理其先前生成的项目列表。
- 创建前往给定目的地的周末度假行程。此示例说明如何并行处理多个不同的提示。
- 向扮演电影制作人的人类用户推销电影创意。此示例说明了如何使用不同的推理参数对同一个提示进行并行处理，如何回溯到链中的上一个步骤，以及如何将人工输入作为工作流程的一部分。
- 根据用户手头的食材计划膳食。这个例子说明了提示链如何整合两个不同的人工智能对话，两个人工智能角色相互进行辩论以改善最终结果。
- 查找并总结当今最热门的 GitHub 存储库。此示例说明如何链接多个与外部 API 交互的 AI 代理。

有关完整的源代码以及设置和运行说明，请参阅上的完整项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock
- Amazon Bedrock 运行时系统
- Agents for Amazon Bedrock
- 亚马逊 Bedrock Runtime 的代理
- Step Functions

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包的 Amazon Bedrock Runtime AWS 代码示例

以下代码示例展示了如何将 Amazon Bedrock Runtime 与 AWS 软件开发套件 (SDK) 一起使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

## 开始使用 Amazon Bedrock

以下代码示例演示了如何开始使用 Amazon Bedrock。

Go

适用于 Go V2 的 SDK

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

```
package main

import (
 "context"
 "encoding/json"
 "flag"
 "fmt"
 "log"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// Each model provider defines their own individual request and response formats.
// For the format, ranges, and default values for the different models, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters.html

type ClaudeRequest struct {
 Prompt string `json:"prompt"`
 MaxTokensToSample int `json:"max_tokens_to_sample"`
 // Omitting optional request parameters
}

type ClaudeResponse struct {
 Completion string `json:"completion"`
}
```

```
}

// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock Runtime client
// and invokes Anthropic Claude 2 inside your account and the chosen region.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {

 region := flag.String("region", "us-east-1", "The AWS region")
 flag.Parse()

 fmt.Printf("Using AWS region: %s\n", *region)

 sdkConfig, err := config.LoadDefaultConfig(context.Background(),
 config.WithRegion(*region))
 if err != nil {
 fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
 fmt.Println(err)
 return
 }

 client := bedrockruntime.NewFromConfig(sdkConfig)

 modelId := "anthropic.claude-v2"

 prompt := "Hello, how are you today?"

 // Anthropic Claude requires you to enclose the prompt as follows:
 prefix := "Human: "
 postfix := "\n\nAssistant:"
 wrappedPrompt := prefix + prompt + postfix

 request := ClaudeRequest{
 Prompt: wrappedPrompt,
 MaxTokensToSample: 200,
 }

 body, err := json.Marshal(request)
 if err != nil {
 log.Panicln("Couldn't marshal the request: ", err)
 }
}
```



```
result, err := client.InvokeModel(context.Background(),
&bedrockruntime.InvokeModelInput{
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
 Body: body,
})

if err != nil {
 errMsg := err.Error()
 if strings.Contains(errMsg, "no such host") {
 fmt.Printf("Error: The Bedrock service is not available in the selected
region. Please double-check the service availability for your region at https://
aws.amazon.com/about-aws/global-infrastructure/regional-product-services/.\\n")
 } else if strings.Contains(errMsg, "Could not resolve the foundation model") {
 fmt.Printf("Error: Could not resolve the foundation model from model
identifier: \"%v\". Please verify that the requested model exists and is
accessible within the specified region.\\n", modelId)
 } else {
 fmt.Printf("Error: Couldn't invoke Anthropic Claude. Here's why: %v\\n", err)
 }
 os.Exit(1)
}

var response ClaudeResponse

err = json.Unmarshal(result.Body, &response)

if err != nil {
 log.Fatal("failed to unmarshal", err)
}
fmt.Println("Prompt:\\n", prompt)
fmt.Println("Response from Anthropic Claude:\\n", response.Completion)
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考 [InvokeModel](#) 中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "url";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
 console.log("=".repeat(35));
 console.log("Welcome to the Amazon Bedrock demo!");
 console.log("=".repeat(35));
}
```

```
console.log("Model: Anthropic Claude 3 Haiku");
console.log(`Prompt: ${PROMPT}\n`);
console.log("Invoking model...\n");

// Create a new Bedrock Runtime client instance.
const client = new BedrockRuntimeClient({ region: AWS_REGION });

// Prepare the payload for the model.
const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
};

// Invoke Claude with the payload and wait for the response.
const apiResponse = await client.send(
 new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId: MODEL_ID,
 }),
);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
const responses = responseBody.content;

if (responses.length === 1) {
 console.log(`Response: ${responses[0].text}`);
} else {
 console.log("Haiku returned multiple responses:");
 console.log(responses);
}

console.log(`\nNumber of input tokens: ${responseBody.usage.input_tokens}`);
console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
 await hello();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## 代码示例

- [使用软件开发工具包对 Amazon Bedrock 运行时的 AWS 操作](#)
  - [在 Amazon Bedrock 上调用 Amazon Titan Image Generator G1 模型来生成图像](#)
  - [在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 模型来生成图像](#)
  - [使用多模式提示在 Amazon Bedrock 上调用 Anthropic Claude 3](#)
  - [在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本](#)
  - [在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型生成文本](#)
  - [在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本](#)
  - [在亚马逊 Bedrock 上调用 Anthropic Claude 3 生成文本](#)
  - [在 Amazon Bedrock 上调用 Anthropic Claude Instant 模型进行文本](#)
  - [在 Amazon Bedrock 上调用 Anthropic Claude 以生成带有响应流的文本](#)
  - [在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本](#)
  - [在 Amazon Bedrock 上调用 Mistral 7B 模型生成文本](#)
  - [在 Amazon Bedrock 上调用 Mixtral 8x7b 模型生成文本](#)
- [使用 SDK 的 Amazon Bedrock Runtime AWS 场景](#)
  - [使用软件开发工具包创建提供游乐场的示例应用程序，以便与 Amazon Bedrock 基础模型进行交互 AWS](#)
  - [适用于 Amazon Bedrock 的交互式文本生成平台](#)
  - [在 Amazon Bedrock 上调用多个大型语言模型 \( LLM \)](#)
  - [在 Amazon Bedrock 上调用多个基础模型](#)
  - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用软件开发工具包对 Amazon Bedrock 运行时的 AWS 操作

以下代码示例演示了如何使用 AWS 软件开发工具包执行单个 Amazon Bedrock 运行时操作。这些摘录称为 Amazon Bedrock Runtime API，是必须在上下文中运行的大型程序的代码摘录。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。如需完整列表，请参阅 [Amazon Bedrock 运行时 API 参考](#)。

## 示例

- [在 Amazon Bedrock 上调用 Amazon Titan Image Generator G1 模型来生成图像](#)
- [在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 模型来生成图像](#)
- [使用多模式提示在 Amazon Bedrock 上调用 Anthropic Claude 3](#)
- [在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本](#)
- [在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型生成文本](#)
- [在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本](#)
- [在亚马逊 Bedrock 上调用 Anthropic Claude 3 生成文本](#)
- [在 Amazon Bedrock 上调用 Anthropic Claude Instant 模型进行文本](#)
- [在 Amazon Bedrock 上调用 Anthropic Claude 以生成带有响应流的文本](#)
- [在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本](#)
- [在 Amazon Bedrock 上调用 Mistral 7B 模型生成文本](#)
- [在 Amazon Bedrock 上调用 Mixtral 8x7b 模型生成文本](#)

## 在 Amazon Bedrock 上调用 Amazon Titan Image Generator G1 模型来生成图像

以下代码示例演示了如何在 Amazon Bedrock 上调用 Amazon Titan Image Generator G1 模型来生成图像。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Amazon Titan 图像生成器 G1 基础模型来生成图像。

```
 /// <summary>
 /// Asynchronously invokes the Amazon Titan Image Generator G1 model to
 run an inference based on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that describes the image Amazon Titan
 Image Generator G1 has to generate.</param>
 /// <returns>A base-64 encoded image generated by model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
 formats.
 /// For the format, ranges, and default values for Amazon Titan Image
 Generator G1, refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-titan-image.html
 /// </remarks>
 public static async Task<string?> InvokeTitanImageGeneratorG1Async(string
 prompt, int seed)
 {
 string titanImageGeneratorG1ModelId = "amazon.titan-image-generator-
 v1";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
 {
 { "taskType", "TEXT_IMAGE" },
 { "textToImageParams", new JsonObject()
 {
 { "text", prompt }
 }
 },
 { "imageGenerationConfig", new JsonObject()
 {
 { "numberOfImages", 1 },
 { "quality", "standard" },
 { "cfgScale", 8.0f },
 { "height", 512 },
 { "width", 512 },
 { "seed", seed }
 }
 }
 }
 }
}
```

```
 }.ToJsonString());

 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = titanImageGeneratorG1ModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 var results = JsonNode.ParseAsync(response.Body).Result?
["images"]?.ToArray();

 return results?[0]?.GetValue<string>();
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return null;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。

## Go

## 适用于 Go V2 的 SDK

**Note**

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Amazon Titan Image Generator G1 模型来生成图像。

```
type TitanImageRequest struct {
 TaskType string `json:"taskType"`
 TextToImageParams TextToImageParams `json:"textToImageParams"`
 ImageGenerationConfig ImageGenerationConfig `json:"imageGenerationConfig"`
}
type TextToImageParams struct {
 Text string `json:"text"`
}
type ImageGenerationConfig struct {
 NumberOfImages int `json:"numberOfImages"`
 Quality string `json:"quality"`
 CfgScale float64 `json:"cfgScale"`
 Height int `json:"height"`
 Width int `json:"width"`
 Seed int64 `json:"seed"`
}

type TitanImageResponse struct {
 Images []string `json:"images"`
}

// Invokes the Titan Image model to create an image using the input provided
// in the request body.
func (wrapper InvokeModelWrapper) InvokeTitanImage(prompt string, seed int64)
(string, error) {
 modelId := "amazon.titan-image-generator-v1"

 body, err := json.Marshal(TitanImageRequest {
 TaskType: "TEXT_IMAGE",
 TextToImageParams: TextToImageParams {
```



```
 Text: prompt,
 },
 ImageGenerationConfig: ImageGenerationConfig {
 NumberOfImages: 1,
 Quality: "standard",
 CfgScale: 8.0,
 Height: 512,
 Width: 512,
 Seed: seed,
 },
})

if err != nil { log.Fatal("failed to marshal", err) }

output, err := wrapper.BedrockRuntimeClient.InvokeModel(context.TODO(),
&bedrockruntime.InvokeModelInput{
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
 Body: body,
})

if err != nil { ProcessError(err, modelId) }

var response TitanImageResponse
if err := json.Unmarshal(output.Body, &response); err != nil {
 log.Fatal("failed to unmarshal", err)
}

base64ImageData := response.Images[0]

return base64ImageData, nil
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考 [InvokeModel](#) 中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Amazon Titan Image Generator G1 模型来生成图像。

```
/**
 * Invokes the Amazon Titan image generation model to create an image using
 the
 * input
 * provided in the request body.
 *
 * @param prompt The prompt that you want Amazon Titan to use for image
 * generation.
 * @param seed The random noise seed for image generation (Range: 0 to
 * 2147483647).
 * @return A Base64-encoded string representing the generated image.
 */
public static String invokeTitanImage(String prompt, long seed) {
 /**
 * The different model providers have individual request and response
 formats.
 * For the format, ranges, and default values for Titan Image models
 refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 titan-
 * image.html
 */
 String titanImageModelId = "amazon.titan-image-generator-v1";

 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_EAST_1)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 var textToImageParams = new JSONObject().put("text", prompt);
```

```
var imageGenerationConfig = new JSONObject()
 .put("numberOfImages", 1)
 .put("quality", "standard")
 .put("cfgScale", 8.0)
 .put("height", 512)
 .put("width", 512)
 .put("seed", seed);

JSONObject payload = new JSONObject()
 .put("taskType", "TEXT_IMAGE")
 .put("textToImageParams", textToImageParams)
 .put("imageGenerationConfig", imageGenerationConfig);

InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload.toString()))
 .modelId(titanImageModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

CompletableFuture<InvokeModelResponse> completableFuture =
client.invokeModel(request)
 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " +
exception);
 }
 });

String base64ImageData = "";
try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 base64ImageData = responseBody
 .getJSONArray("images")
 .getString(0);

} catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
} catch (ExecutionException e) {
 System.err.println(e.getMessage());
}
```

```
 return base64ImageData;
}
```

调用 Amazon Titan Image Generator G1 模型来生成图像。

```
/**
 * Invokes the Amazon Titan image generation model to create an image
 using the
 * input
 * provided in the request body.
 *
 * @param prompt The prompt that you want Amazon Titan to use for image
 * generation.
 * @param seed The random noise seed for image generation (Range: 0 to
 * 2147483647).
 * @return A Base64-encoded string representing the generated image.
 */
public static String invokeTitanImage(String prompt, long seed) {
 /**
 * The different model providers have individual request and
 response formats.
 * For the format, ranges, and default values for Titan Image
 models refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-
 image.html
 */
 String titanImageModelId = "amazon.titan-image-generator-v1";

 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_EAST_1)

 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 var textToImageParams = new JSONObject().put("text", prompt);

 var imageGenerationConfig = new JSONObject()
 .put("numberOfImages", 1)
 .put("quality", "standard")
 .put("cfgScale", 8.0)
}
```

```
 .put("height", 512)
 .put("width", 512)
 .put("seed", seed);

 JSONObject payload = new JSONObject()
 .put("taskType", "TEXT_IMAGE")
 .put("textToImageParams", textToImageParams)
 .put("imageGenerationConfig",
imageGenerationConfig);

 InvokeModelRequest request = InvokeModelRequest.builder()

 .body(SdkBytes.fromUtf8String(payload.toString()))
 .modelId(titanImageModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 InvokeModelResponse response = client.invokeModel(request);

 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());

 String base64ImageData = responseBody
 .getJSONArray("images")
 .getString(0);

 return base64ImageData;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[InvokeModel](#)中的。

## PHP

### 适用于 PHP 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Amazon Titan Image Generator G1 模型来生成图像。

```
public function invokeTitanImage(string $prompt, int $seed)
{
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Titan Image models refer
 # to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 # titan-image.html

 $base64_image_data = "";

 try {
 $modelId = 'amazon.titan-image-generator-v1';

 $request = json_encode([
 'taskType' => 'TEXT_IMAGE',
 'textToImageParams' => [
 'text' => $prompt
],
 'imageGenerationConfig' => [
 'numberOfImages' => 1,
 'quality' => 'standard',
 'cfgScale' => 8.0,
 'height' => 512,
 'width' => 512,
 'seed' => $seed
]
]);

 $result = $this->bedrockRuntimeClient->invokeModel([
 'contentType' => 'application/json',
 'body' => $request,
 'modelId' => $modelId,
]);

 $response_body = json_decode($result['body']);

 $base64_image_data = $response_body->images[0];
 } catch (Exception $e) {
 echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
 }
}
```

```
 return $base64_image_data;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考[InvokeModel](#)中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Amazon Titan Image Generator G1 模型来生成图像。

```
def invoke_titan_image(self, prompt, seed):
 """
 Invokes the Titan Image model to create an image using the input provided
 in the request body.

 :param prompt: The prompt that you want Amazon Titan to use for image
 generation.
 :param seed: Random noise seed (range: 0 to 2147483647)
 :return: Base64-encoded inference response from the model.
 """

 try:
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Titan Image models
 # refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 # parameters-titan-image.html

 request = json.dumps(
 {
 "taskType": "TEXT_IMAGE",
 "textToImageParams": {"text": prompt},
 "imageGenerationConfig": {
```

```
 "numberOfImages": 1,
 "quality": "standard",
 "cfgScale": 8.0,
 "height": 512,
 "width": 512,
 "seed": seed,
 },
}
)

response = self.bedrock_runtime_client.invoke_model(
 modelId="amazon.titan-image-generator-v1", body=request
)

response_body = json.loads(response["body"].read())
base64_image_data = response_body["images"][0]

return base64_image_data

except ClientError:
 logger.error("Couldn't invoke Titan Image generator")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 模型来生成图像

以下代码示例演示了如何在 Amazon Bedrock 上调用 Stability.ai Stable Diffusion XL 模型来生成图像。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)



## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```
 /// <summary>
 /// Asynchronously invokes the Stability.ai Stable Diffusion XLmodel to
 run an inference based on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that describes the image Stability.ai
 Stable Diffusion XL has to generate.</param>
 /// <returns>A base-64 encoded image generated by model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
 formats.
 /// For the format, ranges, and default values for Stability.ai Stable
 Diffusion XL, refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-stability-diffusion.html
 /// </remarks>
 public static async Task<string?> InvokeStableDiffusionXLG1Async(string
 prompt, int seed, string? stylePreset = null)
 {
 string stableDiffusionXLModelId = "stability.stable-diffusion-xl";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 var jsonPayload = new JsonObject()
 {
 { "text_prompts", new JsonArray() {
 new JsonObject()
 {
 { "text", prompt }
 }
 }
 }
 }
```

```
 },
 { "seed", seed }
 };

 if (!string.IsNullOrEmpty(stylePreset))
 {
 jsonPayload.Add("style_preset", stylePreset);
 }

 string payload = jsonPayload.ToString();

 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = stableDiffusionXLModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 var results = JsonNode.ParseAsync(response.Body).Result?
["artifacts"]?.AsArray();

 return results?[0]?["base64"]?.GetValue<string>();
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return null;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModel](#) 中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```
/**
 * Asynchronously invokes the Stability.ai Stable Diffusion XL model to
 * create
 * an image based on the provided input.
 *
 * @param prompt The prompt that guides the Stable Diffusion model.
 * @param seed The random noise seed for image generation (use 0 or
omit
 * for a random seed).
 * @param stylePreset The style preset to guide the image model towards a
 * specific style.
 * @return A Base64-encoded string representing the generated image.
 */
public static String invokeStableDiffusion(String prompt, long seed, String
stylePreset) {
 /**
 * The different model providers have individual request and response
 * formats.
 * For the format, ranges, and available style_presets of Stable
 * Diffusion
 * models refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 * stability-diffusion.html
 */

 String stableDiffusionModelId = "stability.stable-diffusion-xl";

 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
```

```
 .region(Region.US_EAST_1)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 JSONArray wrappedPrompt = new JSONArray().put(new
JSONObject().put("text", prompt));
 JSONObject payload = new JSONObject()
 .put("text_prompts", wrappedPrompt)
 .put("seed", seed);

 if (stylePreset != null && !stylePreset.isEmpty()) {
 payload.put("style_preset", stylePreset);
 }

 InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload.toString()))
 .modelId(stableDiffusionModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 CompletableFuture<InvokeModelResponse> completableFuture =
client.invokeModel(request)
 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " +
exception);
 }
 });

 String base64ImageData = "";
 try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 base64ImageData = responseBody
 .getJSONArray("artifacts")
 .getJSONObject(0)
 .getString("base64");
 } catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
 } catch (ExecutionException e) {
```

```
 System.err.println(e.getMessage());
 }

 return base64ImageData;
}
```

调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```
/**
 * Invokes the Stability.ai Stable Diffusion XL model to create an image
based
 * on the provided input.
 *
 * @param prompt The prompt that guides the Stable Diffusion model.
 * @param seed The random noise seed for image generation (use 0
or omit
 * for a random seed).
 * @param stylePreset The style preset to guide the image model towards a
 * specific style.
 * @return A Base64-encoded string representing the generated image.
 */
public static String invokeStableDiffusion(String prompt, long seed,
String stylePreset) {
 /**
 * The different model providers have individual request and
response formats.
 * For the format, ranges, and available style_presets of Stable
Diffusion
 * models refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-stability-diffusion.html
 */

 String stableDiffusionModelId = "stability.stable-diffusion-xl";

 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_EAST_1)

 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();
```

```
 JSONArray wrappedPrompt = new JSONArray().put(new
JSONObject().put("text", prompt));

 JSONObject payload = new JSONObject()
 .put("text_prompts", wrappedPrompt)
 .put("seed", seed);

 if (!(stylePreset == null || stylePreset.isEmpty())) {
 payload.put("style_preset", stylePreset);
 }

 InvokeModelRequest request = InvokeModelRequest.builder()

 .body(SdkBytes.fromUtf8String(payload.toString()))
 .modelId(stableDiffusionModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 InvokeModelResponse response = client.invokeModel(request);

 JSONObject responseBody = new
 JSONObject(response.body().asUtf8String());

 String base64ImageData = responseBody
 .getJSONArray("artifacts")
 .getJSONObject(0)
 .getString("base64");

 return base64ImageData;
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[InvokeModel](#)中的。

## PHP

## 适用于 PHP 的 SDK

**Note**

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```
public function invokeStableDiffusion(string $prompt, int $seed, string
$style_preset)
{
 # The different model providers have individual request and response
 formats.
 # For the format, ranges, and available style_presets of Stable Diffusion
 models refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 stability-diffusion.html

 $base64_image_data = "";

 try {
 $modelId = 'stability.stable-diffusion-xl';

 $body = [
 'text_prompts' => [
 ['text' => $prompt]
],
 'seed' => $seed,
 'cfg_scale' => 10,
 'steps' => 30
];

 if ($style_preset) {
 $body['style_preset'] = $style_preset;
 }

 $result = $this->bedrockRuntimeClient->invokeModel([
 'contentType' => 'application/json',
 'body' => json_encode($body),
```

```

 'modelId' => $modelId,
]);

 $response_body = json_decode($result['body']);

 $base64_image_data = $response_body->artifacts[0]->base64;
} catch (Exception $e) {
 echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
}

return $base64_image_data;
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```

def invoke_stable_diffusion(self, prompt, seed, style_preset=None):
 """
 Invokes the Stability.ai Stable Diffusion XL model to create an image
 using
 the input provided in the request body.

 :param prompt: The prompt that you want Stable Diffusion to use for
 image generation.
 :param seed: Random noise seed (omit this option or use 0 for a random
 seed)
 :param style_preset: Pass in a style preset to guide the image model
 towards
 a particular style.
 :return: Base64-encoded inference response from the model.
 """

```



```
"""

 try:
 # The different model providers have individual request and response
 formats.
 # For the format, ranges, and available style_presets of Stable
 Diffusion models refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-stability-diffusion.html

 body = {
 "text_prompts": [{"text": prompt}],
 "seed": seed,
 "cfg_scale": 10,
 "steps": 30,
 }

 if style_preset:
 body["style_preset"] = style_preset

 response = self.bedrock_runtime_client.invoke_model(
 modelId="stability.stable-diffusion-xl", body=json.dumps(body)
)

 response_body = json.loads(response["body"].read())
 base64_image_data = response_body["artifacts"][0]["base64"]


 return base64_image_data

 except ClientError:
 logger.error("Couldn't invoke Stable Diffusion XL")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的AWS SDK (Boto3) API 参考。

## SAP ABAP

## SDK for SAP ABAP

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Stability.ai Stable Diffusion XL 基础模型来生成图像。

```
"Stable Diffusion Input Parameters should be in a format like this:
* {
* "text_prompts": [
* {"text":"Draw a dolphin with a mustache"},
* {"text":"Make it photorealistic"}
*],
* "cfg_scale":10,
* "seed":0,
* "steps":50
* }
TYPES: BEGIN OF prompt_ts,
 text TYPE /aws1/rt_shape_string,
 END OF prompt_ts.

DATA: BEGIN OF ls_input,
 text_prompts TYPE STANDARD TABLE OF prompt_ts,
 cfg_scale TYPE /aws1/rt_shape_integer,
 seed TYPE /aws1/rt_shape_integer,
 steps TYPE /aws1/rt_shape_integer,
 END OF ls_input.

APPEND VALUE prompt_ts(text = iv_prompt) TO ls_input-text_prompts.
ls_input-cfg_scale = 10.
ls_input-seed = 0. "or better, choose a random integer.
ls_input-steps = 50.

DATA(lv_json) = /ui2/cl_json=>serialize(
 data = ls_input
 pretty_name = /ui2/cl_json=>pretty_mode-low_case).

TRY.
```

```

DATA(lo_response) = lo_bdr->invokemodel(
 iv_body = /aws1/cl_rt_util=>string_to_xstring(lv_json)
 iv_modelid = 'stability.stable-diffusion-xl-v0'
 iv_accept = 'application/json'
 iv_contenttype = 'application/json').

"Stable Diffusion Result Format:
*
* {
* "result": "success",
* "artifacts": [
* {
* "seed": 0,
* "base64": "iVBORw0KGgoAAAANSUUEUgAAAgAAA....
* "finishReason": "SUCCESS"
* }
*]
* }
TYPES: BEGIN OF artifact_ts,
 seed TYPE /aws1/rt_shape_integer,
 base64 TYPE /aws1/rt_shape_string,
 finishreason TYPE /aws1/rt_shape_string,
END OF artifact_ts.

DATA: BEGIN OF ls_response,
 result TYPE /aws1/rt_shape_string,
 artifacts TYPE STANDARD TABLE OF artifact_ts,
END OF ls_response.

/ui2/cl_json=>deserialize(
 EXPORTING jsonx = lo_response->get_body()
 pretty_name = /ui2/cl_json=>pretty_mode-camel_case
 CHANGING data = ls_response).
IF ls_response-artifacts IS NOT INITIAL.
 DATA(lv_image) =
 cl_http_utility=>if_http_utility~decode_x_base64(ls_response-artifacts[1]-
base64).
ENDIF.
CATCH /aws1/cx_bdraccessdeniedex INTO DATA(lo_ex).
WRITE / lo_ex->get_text().
WRITE / |Don't forget to enable model access at https://
console.aws.amazon.com/bedrock/home?#/modelaccess|.

ENDTRY.

```

调用 Stability.ai Stable Diffusion XL 基础模型，使用 L2 高级客户端生成图像。

```
TRY.
 DATA(lo_bdr_l2_sd) = /aws1/
cl_bdr_l2_factory=>create_stable_diffusion_10(lo_bdr).
 " iv_prompt contains a prompt like 'Show me a picture of a unicorn reading
 an enterprise financial report'.
 DATA(lv_image) = lo_bdr_l2_sd->text_to_image(iv_prompt).
 CATCH /aws1/cx_bdraccessdeniedex INTO DATA(lo_ex).
 WRITE / lo_ex->get_text().
 WRITE / |Don't forget to enable model access at https://
console.aws.amazon.com/bedrock/home?#/modelaccess|.

ENDTRY.
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用多模式提示在 Amazon Bedrock 上调用 Anthropic Claude 3

以下代码示例显示了如何使用多模式提示在 Amazon Bedrock 上调用 Anthropic Claude 3。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

使用多模态提示调用 Anthropic Claude 3 来分析图像。

```
def invoke_claude_3_multimodal(self, prompt, base64_image_data):
 """
 Invokes Anthropic Claude 3 Sonnet to run a multimodal inference using the
 input
 provided in the request body.

 :param prompt: The prompt that you want Claude 3 to use.
 :param base64_image_data: The base64-encoded image that you want to add
 to the request.
 :return: Inference response from the model.
 """

 # Initialize the Amazon Bedrock runtime client
 client = self.client or boto3.client(
 service_name="bedrock-runtime", region_name="us-east-1"
)

 # Invoke the model with the prompt and the encoded image
 model_id = "anthropic.claude-3-sonnet-20240229-v1:0"
 request_body = {
 "anthropic_version": "bedrock-2023-05-31",
 "max_tokens": 2048,
 "messages": [
 {
 "role": "user",
 "content": [
 {
 "type": "text",
 "text": prompt,
 },
],
 },
],
 }
```

```

 "type": "image",
 "source": {
 "type": "base64",
 "media_type": "image/png",
 "data": base64_image_data,
 },
 },
],
}

try:
 response = client.invoke_model(
 modelId=model_id,
 body=json.dumps(request_body),
)

 # Process and print the response
 result = json.loads(response.get("body").read())
 input_tokens = result["usage"]["input_tokens"]
 output_tokens = result["usage"]["output_tokens"]
 output_list = result.get("content", [])

 print("Invocation details:")
 print(f"- The input length is {input_tokens} tokens.")
 print(f"- The output length is {output_tokens} tokens.")

 print(f"- The model returned {len(output_list)} response(s):")
 for output in output_list:
 print(output["text"])

 return result
except ClientError as err:
 logger.error(
 "Couldn't invoke Claude 3 Sonnet. Here's why: %s: %s",
 err.response["Error"]["Code"],
 err.response["Error"]["Message"],
)
 raise

```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本

以下代码示例演示了如何在 Amazon Bedrock 上调用 AI21 Labs Jurassic-2 模型来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 AI21 Labs Jurassic-2 基础模型。

```
/// <summary>
/// Asynchronously invokes the AI21 Labs Jurassic-2 model to run an
inference based on the provided input.
/// </summary>
/// <param name="prompt">The prompt that you want Claude to complete.</
param>
/// <returns>The inference response from the model</returns>
/// <remarks>
/// The different model providers have individual request and response
formats.
/// For the format, ranges, and default values for AI21 Labs Jurassic-2,
refer to:
/// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-jurassic2.html
```

```
/// </remarks>
public static async Task<string> InvokeJurassic2Async(string prompt)
{
 string jurassic2ModelId = "ai21.j2-mid-v1";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
 {
 { "prompt", prompt },
 { "maxTokens", 200 },
 { "temperature", 0.5 }
 }.ToJsonString();

 string generatedText = "";
 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = jurassic2ModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 return JsonNode.ParseAsync(response.Body)
 .Result?["completions"]?
 .ToArray()[0]?["data"]?
 .AsObject()["text"]?.GetValue<string>() ?? "";
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return generatedText;
}
```




```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModel](#) 中的。

Go

适用于 Go V2 的 SDK

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 AI21 Labs Jurassic-2 基础模型以生成文本。

```
// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for AI21 Labs Jurassic-2, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html

type Jurassic2Request struct {
 Prompt string `json:"prompt"`
 MaxTokens int `json:"maxTokens,omitempty"`
 Temperature float64 `json:"temperature,omitempty"`
}

type Jurassic2Response struct {
 Completions []Completion `json:"completions"`
}

type Completion struct {
 Data Data `json:"data"`
}

type Data struct {
 Text string `json:"text"`
}

// Invokes AI21 Labs Jurassic-2 on Amazon Bedrock to run an inference using the
input
```

```
// provided in the request body.
func (wrapper InvokeModelWrapper) InvokeJurassic2(prompt string) (string, error)
{
 modelId := "ai21.j2-mid-v1"

 body, err := json.Marshal(Jurassic2Request {
 Prompt: prompt,
 MaxTokens: 200,
 Temperature: 0.5,
 })

 if err != nil { log.Fatal("failed to marshal", err) }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(context.TODO(),
 &bedrockruntime.InvokeModelInput{
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
 Body: body,
 })

 if err != nil { ProcessError(err, modelId) }

 var response Jurassic2Response
 if err := json.Unmarshal(output.Body, &response); err != nil {
 log.Fatal("failed to unmarshal", err)
 }

 return response.Completions[0].Data.Text, nil
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考[InvokeModel](#)中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 AI21 Labs Jurassic-2 基础模型来生成文本。

```
/**
 * Asynchronously invokes the AI21 Labs Jurassic-2 model to run an inference
 * based on the provided input.
 *
 * @param prompt The prompt that you want Jurassic to complete.
 * @return The inference response generated by the model.
 */
public static String invokeJurassic2(String prompt) {
 /**
 * The different model providers have individual request and response
 formats.
 * For the format, ranges, and default values for Anthropic Claude, refer
 to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-claude.html
 */

 String jurassic2ModelId = "ai21.j2-mid-v1";

 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_EAST_1)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 String payload = new JSONObject()
 .put("prompt", prompt)
 .put("temperature", 0.5)
 .put("maxTokens", 200)
 .toString();

 InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(jurassic2ModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 CompletableFuture<InvokeModelResponse> completableFuture =
 client.invokeModel(request)
 .whenComplete((response, exception) -> {
 if (exception != null) {
```

```

 System.out.println("Model invocation failed: " +
exception);
 }
});

String generatedText = "";
try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 generatedText = responseBody
 .getJSONArray("completions")
 .getJSONObject(0)
 .getJSONObject("data")
 .getString("text");

} catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
} catch (ExecutionException e) {
 System.err.println(e.getMessage());
}

return generatedText;
}

```

调用 AI21 Labs Jurassic-2 基础模型来生成文本。

```

/**
 * Invokes the AI21 Labs Jurassic-2 model to run an inference based on
the
 * provided input.
 *
 * @param prompt The prompt for Jurassic to complete.
 * @return The generated response.
 */
public static String invokeJurassic2(String prompt) {
 /*
 * The different model providers have individual request and
response formats.
 * For the format, ranges, and default values for AI21 Labs
Jurassic-2, refer

```

```
 * to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-jurassic2.html
 */

String jurassic2ModelId = "ai21.j2-mid-v1";

BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_EAST_1)

.credentialsProvider(ProfileCredentialsProvider.create())
 .build();

String payload = new JSONObject()
 .put("prompt", prompt)
 .put("temperature", 0.5)
 .put("maxTokens", 200)
 .toString();

InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(jurassic2ModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

InvokeModelResponse response = client.invokeModel(request);

JSONObject responseBody = new
JSONObject(response.body().asUtf8String());

String generatedText = responseBody
 .getJSONArray("completions")
 .getJSONObject(0)
 .getJSONObject("data")
 .getString("text");

return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [InvokeModel](#) 中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 AI21 Labs Jurassic-2 基础模型以生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "ai21.j2-
mid-v1".
 */
export const invokeModel = async (prompt, modelId = "ai21.j2-mid-v1") => {
 // Create a new Bedrock Runtime client instance.
```

```
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Prepare the payload for the model.
const payload = {
 prompt,
 maxTokens: 500,
 temperature: 0.5,
};

// Invoke the model with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s).
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.completions[0].data.text;
};


// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time..."';
 const modelId = FoundationModels.JURASSIC2_MID.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## PHP

## 适用于 PHP 的 SDK

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 AI21 Labs Jurassic-2 基础模型以生成文本。

```
public function invokeJurassic2($prompt)
{
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for AI21 Labs Jurassic-2,
 # refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 # jurassic2.html

 $completion = "";

 try {
 $modelId = 'ai21.j2-mid-v1';

 $body = [
 'prompt' => $prompt,
 'temperature' => 0.5,
 'maxTokens' => 200,
];

 $result = $this->bedrockRuntimeClient->invokeModel([
 'contentType' => 'application/json',
 'body' => json_encode($body),
 'modelId' => $modelId,
]);

 $response_body = json_decode($result['body']);

 $completion = $response_body->completions[0]->data->text;
 } catch (Exception $e) {
 echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
 }
}
```



```
 }

 return $completion;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 AI21 Labs Jurassic-2 基础模型以生成文本。

```
def invoke_jurassic2(self, prompt):
 """
 Invokes the AI21 Labs Jurassic-2 large-language model to run an inference
 using the input provided in the request body.

 :param prompt: The prompt that you want Jurassic-2 to complete.
 :return: Inference response from the model.
 """

 try:
 # The different model providers have individual request and response
 formats.
 # For the format, ranges, and default values for AI21 Labs
 Jurassic-2, refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-jurassic2.html

 body = {
 "prompt": prompt,
 "temperature": 0.5,
 "maxTokens": 200,
 }
```

```
response = self.bedrock_runtime_client.invoke_model(
 modelId="ai21.j2-mid-v1", body=json.dumps(body)
)

response_body = json.loads(response["body"].read())
completion = response_body["completions"][0]["data"]["text"]

return completion

except ClientError:
 logger.error("Couldn't invoke Jurassic-2")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型生成文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Amazon Titan Text G1 模型来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

## 异步调用 Amazon Titan Text G1 基础模型来生成文本。

```
 /// <summary>
 /// Asynchronously invokes the Amazon Titan Text G1 Express model to run
 an inference based on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that you want Amazon Titan Text G1
 Express to complete.</param>
 /// <returns>The inference response from the model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
 formats.
 /// For the format, ranges, and default values for Amazon Titan Text G1
 Express, refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-text.html
 /// </remarks>
 public static async Task<string> InvokeTitanTextG1Async(string prompt)
 {
 string titanTextG1ModelId = "amazon.titan-text-express-v1";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
 {
 { "inputText", prompt },
 { "textGenerationConfig", new JsonObject()
 {
 { "maxTokenCount", 512 },
 { "temperature", 0f },
 { "topP", 1f }
 }
 }
 }.ToJsonString();

 string generatedText = "";
 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
 InvokeModelRequest()
 {
 ModelId = titanTextG1ModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
```

```
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 var results = JsonNode.ParseAsync(response.Body).Result?
["results"]?.ToArray();

 return results is null ? "" : string.Join(" ",
results.Select(x => x?["outputText"]?.GetValue<string?>()));
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
}
catch (AmazonBedrockRuntimeException e)
{
 Console.WriteLine(e.Message);
}
return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Amazon Titan Text G1 基础模型来生成文本。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {Object[]} results
 */

/**
 * Invokes an Amazon Titan Text generation model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
"amazon.titan-text-express-v1".
 */
export const invokeModel = async (
 prompt,
 modelId = "amazon.titan-text-express-v1",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the model.
 const payload = {
 inputText: prompt,
 textGenerationConfig: {
 maxTokenCount: 4096,
 stopSequences: [],
 temperature: 0,
 topP: 1,
 },
 };

 // Invoke the model with the payload and wait for the response.
 const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 });
```

```
 modelId,
 });
 const apiResponse = await client.send(command);

 // Decode and return the response.
 const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
 /** @type {ResponseBody} */
 const responseBody = JSON.parse(decodedResponseBody);
 return responseBody.results[0].outputText;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time...";
 const modelId = FoundationModels.TITAN_TEXT_G1_EXPRESS.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考[InvokeModel](#)中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本

以下代码示例演示了如何在 Amazon Bedrock 上调用 Anthropic Claude 2 模型来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)

- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Anthropic Claude 2 基础模型来生成文本。

```
 /// <summary>
 /// Asynchronously invokes the Anthropic Claude 2 model to run an
 inference based on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that you want Claude to complete.</
param>
 /// <returns>The inference response from the model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
 formats.
 /// For the format, ranges, and default values for Anthropic Claude,
 refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-claude.html
 /// </remarks>
 public static async Task<string> InvokeClaudeAsync(string prompt)
 {
 string claudeModelId = "anthropic.claude-v2";

 // Claude requires you to enclose the prompt as follows:
 string enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
```

```
 {
 { "prompt", enclosedPrompt },
 { "max_tokens_to_sample", 200 },
 { "temperature", 0.5 },
 { "stop_sequences", new JSONArray("\n\nHuman:") }
 }.ToJsonString();

 string generatedText = "";
 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = claudeModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });


 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 return JsonNode.ParseAsync(response.Body).Result?
["completion"]?.GetValue<string>() ?? "";
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。



## Go

## 适用于 Go V2 的 SDK

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

```
// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for Anthropic Claude, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
// claude.html

type ClaudeRequest struct {
 Prompt string `json:"prompt"`
 MaxTokensToSample int `json:"max_tokens_to_sample"`
 Temperature float64 `json:"temperature,omitempty"`
 StopSequences []string `json:"stop_sequences,omitempty"`
}

type ClaudeResponse struct {
 Completion string `json:"completion"`
}

// Invokes Anthropic Claude on Amazon Bedrock to run an inference using the input
// provided in the request body.
func (wrapper InvokeModelWrapper) InvokeClaude(prompt string) (string, error) {
 modelId := "anthropic.claude-v2"

 // Anthropic Claude requires enclosing the prompt as follows:
 enclosedPrompt := "Human: " + prompt + "\n\nAssistant:"

 body, err := json.Marshal(ClaudeRequest {
 Prompt: enclosedPrompt,
 MaxTokensToSample: 200,
 Temperature: 0.5,
 StopSequences: []string{"\n\nHuman:"},
 })
}
```

```
if err != nil { log.Fatal("failed to marshal", err) }

output, err := wrapper.BedrockRuntimeClient.InvokeModel(context.TODO(),
&bedrockruntime.InvokeModelInput{
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
 Body: body,
})

if err != nil { ProcessError(err, modelId) }

var response ClaudeResponse
 if err := json.Unmarshal(output.Body, &response); err != nil {
 log.Fatal("failed to unmarshal", err)
 }

return response.Completion, nil
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考 [InvokeModel](#) 中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Anthropic Claude 2 基础模型来生成文本。

```
/**
 * Asynchronously invokes the Anthropic Claude 2 model to run an inference
 based
 * on the provided input.
 *
 * @param prompt The prompt that you want Claude to complete.
 * @return The inference response from the model.
```

```
*/
public static String invokeClaude(String prompt) {
 /*
 * The different model providers have individual request and response
 formats.
 * For the format, ranges, and default values for Anthropic Claude, refer
 to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-claude.html
 */

 String claudeModelId = "anthropic.claude-v2";

 // Claude requires you to enclose the prompt as follows:
 String enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_EAST_1)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 String payload = new JSONObject()
 .put("prompt", enclosedPrompt)
 .put("max_tokens_to_sample", 200)
 .put("temperature", 0.5)
 .put("stop_sequences", List.of("\n\nHuman:"))
 .toString();

 InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(claudeModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 CompletableFuture<InvokeModelResponse> completableFuture =
 client.invokeModel(request)
 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " +
 exception);
 }
 });
}
```

```
String generatedText = "";
try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 generatedText = responseBody.getString("completion");
} catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
} catch (ExecutionException e) {
 System.err.println(e.getMessage());
}

return generatedText;
}
```

调用 Anthropic Claude 2 基础模型来生成文本。

```
/**
 * Invokes the Anthropic Claude 2 model to run an inference based on the
 * provided input.
 *
 * @param prompt The prompt for Claude to complete.
 * @return The generated response.
 */
public static String invokeClaude(String prompt) {
 /**
 * The different model providers have individual request and
 response formats.
 * For the format, ranges, and default values for Anthropic
 Claude, refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-claude.html
 */

 String claudeModelId = "anthropic.claude-v2";

 // Claude requires you to enclose the prompt as follows:
 String enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_EAST_1)
```

```
.credentialsProvider(ProfileCredentialsProvider.create())
 .build();

String payload = new JSONObject()
 .put("prompt", enclosedPrompt)
 .put("max_tokens_to_sample", 200)
 .put("temperature", 0.5)
 .put("stop_sequences", List.of("\n\nHuman:"))
 .toString();

InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(claudeModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

InvokeModelResponse response = client.invokeModel(request);

JSONObject responseBody = new
JSONObject(response.body().asUtf8String());

String generatedText = responseBody.getString("completion");

return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[InvokeModel](#)中的。

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 */

/**
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 */

/**
 * @typedef {Object} TextCompletionsResponseBody
 * @property {completion} text
 */

/**
 * Invokes Anthropic Claude 2.x using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-v2".
 */
export const invokeModel = async (prompt, modelId = "anthropic.claude-v2") => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the Messages API request.
 const payload = {
```

```
anthropic_version: "bedrock-2023-05-31",
max_tokens: 1000,
messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
];

// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {MessagesResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 2.x using the Text Completions API.
 *
 * To learn more about the Anthropic Text Completions API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-text-completion.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-v2".
 */
export const invokeTextCompletionsApi = async (
 prompt,
 modelId = "anthropic.claude-v2",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });
```

```
// Prepare the payload for the Text Completions API, using the required prompt
template.
const enclosedPrompt = `Human: ${prompt}\n\nAssistant:`;
const payload = {
 prompt: enclosedPrompt,
 max_tokens_to_sample: 500,
 temperature: 0.5,
 stop_sequences: ["\n\nHuman:"],
};

// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response.
const decoded = new TextDecoder().decode(apiResponse.body);
/** @type {TextCompletionsResponseBody} */
const responseBody = JSON.parse(decoded);
return responseBody.completion;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time...";
 const modelId = FoundationModels.CLAUDE_2.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 console.log("Using the Messages API:");
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }

 try {
 console.log("-".repeat(53));
```



```
console.log("Using the Text Completions API:");
const response = await invokeTextCompletionsApi(prompt, modelId);
console.log(response);
} catch (err) {
 console.log(err);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## PHP

### 适用于 PHP 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

```
public function invokeClaude($prompt)
{
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Anthropic Claude, refer
 # to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 # claude.html

 $completion = "";

 try {
 $modelId = 'anthropic.claude-v2';

 # Claude requires you to enclose the prompt as follows:
 $prompt = "\n\nHuman: {$prompt}\n\nAssistant:";

 $body = [
 'prompt' => $prompt,
```

```
 'max_tokens_to_sample' => 200,
 'temperature' => 0.5,
 'stop_sequences' => ["\n\nHuman:"],
];

 $result = $this->bedrockRuntimeClient->invokeModel([
 'contentType' => 'application/json',
 'body' => json_encode($body),
 'modelId' => $modelId,
]);

 $response_body = json_decode($result['body']);

 $completion = $response_body->completion;
} catch (Exception $e) {
 echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
}

return $completion;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。

```
def invoke_claude(self, prompt):
 """
 Invokes the Anthropic Claude 2 model to run an inference using the input
 provided in the request body.

 :param prompt: The prompt that you want Claude to complete.
```

```
 :return: Inference response from the model.
 """

 try:
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Anthropic Claude,
 # refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 # parameters-claude.html

 # Claude requires you to enclose the prompt as follows:
 enclosed_prompt = "Human: " + prompt + "\n\nAssistant:"

 body = {
 "prompt": enclosed_prompt,
 "max_tokens_to_sample": 200,
 "temperature": 0.5,
 "stop_sequences": ["\n\nHuman:"],
 }

 response = self.bedrock_runtime_client.invoke_model(
 modelId="anthropic.claude-v2", body=json.dumps(body)
)

 response_body = json.loads(response["body"].read())
 completion = response_body["completion"]

 return completion

 except ClientError:
 logger.error("Couldn't invoke Anthropic Claude")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的AWS SDK (Boto3) API 参考。

## SAP ABAP

## SDK for SAP ABAP

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 2 基础模型以生成文本。此示例使用了 /US2/CL\_JSON 的功能，这些功能在某些版本上可能不可用。NetWeaver

```

" Claude V2 Input Parameters should be in a format like this:
* {
* "prompt": "\n\nHuman:\n\nTell me a joke\n\nAssistant:\n",
* "max_tokens_to_sample": 2048,
* "temperature": 0.5,
* "top_k": 250,
* "top_p": 1.0,
* "stop_sequences": []
* }

DATA: BEGIN OF ls_input,
 prompt TYPE string,
 max_tokens_to_sample TYPE /aws1/rt_shape_integer,
 temperature TYPE /aws1/rt_shape_float,
 top_k TYPE /aws1/rt_shape_integer,
 top_p TYPE /aws1/rt_shape_float,
 stop_sequences TYPE /aws1/rt_stringtab,
END OF ls_input.

"Leave ls_input-stop_sequences empty.
ls_input-prompt = |\n\nHuman:\n\n{ iv_prompt }\n\nAssistant:\n|.
ls_input-max_tokens_to_sample = 2048.
ls_input-temperature = '0.5'.
ls_input-top_k = 250.
ls_input-top_p = 1.

"Serialize into JSON with /ui2/cl_json -- this assumes SAP_UI is installed.
DATA(lv_json) = /ui2/cl_json=>serialize(
 data = ls_input

```

```

 pretty_name = /ui2/cl_json=>pretty_mode-low_case).

TRY.
 DATA(lo_response) = lo_bdr->invokemodel(
 iv_body = /aws1/cl_rt_util=>string_to_xstring(lv_json)
 iv_modelid = 'anthropic.claude-v2'
 iv_accept = 'application/json'
 iv_contenttype = 'application/json').

 "Claude V2 Response format will be:
* {
* "completion": "Knock Knock...",
* "stop_reason": "stop_sequence"
* }
 DATA: BEGIN OF ls_response,
 completion TYPE string,
 stop_reason TYPE string,
 END OF ls_response.

 /ui2/cl_json=>deserialize(
 EXPORTING jsonx = lo_response->get_body()
 pretty_name = /ui2/cl_json=>pretty_mode-camel_case
 CHANGING data = ls_response).

 DATA(lv_answer) = ls_response-completion.
 CATCH /aws1/cx_bdraccessdeniedex INTO DATA(lo_ex).
 WRITE / lo_ex->get_text().
 WRITE / |Don't forget to enable model access at https://
console.aws.amazon.com/bedrock/home?#/modelaccess|.

ENDTRY.

```

调用 Anthropic Claude 2 基础模型，使用 L2 高级客户端生成文本。

```

TRY.
 DATA(lo_bdr_l2_claude) = /aws1/
cl_bdr_l2_factory=>create_claude_2(lo_bdr).
 " iv_prompt can contain a prompt like 'tell me a joke about Java
 programmers'.
 DATA(lv_answer) = lo_bdr_l2_claude->prompt_for_text(iv_prompt).
 CATCH /aws1/cx_bdraccessdeniedex INTO DATA(lo_ex).
 WRITE / lo_ex->get_text().

```

```
WRITE / |Don't forget to enable model access at https://
console.aws.amazon.com/bedrock/home?#/modelaccess|.
```

```
ENDTRY.
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 S AP 的AWS SDK ABAP API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在亚马逊 Bedrock 上调用 Anthropic Claude 3 生成文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Anthropic Claude 3 来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 3 生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
```

```
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
 InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
 prompt,
 modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the model.
```

```
const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
};

// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {MessagesResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
 prompt,
 modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });
```



```
// Prepare the payload for the model.
const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
};

// Invoke Claude with the payload and wait for the API to respond.
const command = new InvokeModelWithResponseStreamCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

let completeMessage = "";

// Decode and process the response stream
for await (const item of apiResponse.body) {
 /** @type Chunk */
 const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
 const chunk_type = chunk.type;

 if (chunk_type === "content_block_delta") {
 const text = chunk.delta.text;
 completeMessage = completeMessage + text;
 process.stdout.write(text);
 }
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt = 'Write a paragraph starting with: "Once upon a time..."';
 const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
 console.log(`Prompt: ${prompt}`);
}
```

```
console.log(`Model ID: ${modelId}`);

try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log("\n" + "-".repeat(53));
 console.log("Final structured response:");
 console.log(response);
} catch (err) {
 console.log(`\n${err}`);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 3 生成文本。

```
def invoke_claude_3_with_text(self, prompt):
 """
 Invokes Anthropic Claude 3 Sonnet to run an inference using the input
 provided in the request body.

 :param prompt: The prompt that you want Claude 3 to complete.
 :return: Inference response from the model.
 """

 # Initialize the Amazon Bedrock runtime client
 client = self.client or boto3.client(
 service_name="bedrock-runtime", region_name="us-east-1"
)
```

```
Invoke Claude 3 with the text prompt
model_id = "anthropic.claude-3-sonnet-20240229-v1:0"

try:
 response = client.invoke_model(
 modelId=model_id,
 body=json.dumps(
 {
 "anthropic_version": "bedrock-2023-05-31",
 "max_tokens": 1024,
 "messages": [
 {
 "role": "user",
 "content": [{"type": "text", "text": prompt}],
 }
],
 }
),
)

 # Process and print the response
 result = json.loads(response.get("body").read())
 input_tokens = result["usage"]["input_tokens"]
 output_tokens = result["usage"]["output_tokens"]
 output_list = result.get("content", [])

 print("Invocation details:")
 print(f"- The input length is {input_tokens} tokens.")
 print(f"- The output length is {output_tokens} tokens.")

 print(f"- The model returned {len(output_list)} response(s):")
 for output in output_list:
 print(output["text"])

 return result

except ClientError as err:
 logger.error(
 "Couldn't invoke Claude 3 Sonnet. Here's why: %s: %s",
 err.response["Error"]["Code"],
 err.response["Error"]["Message"],
)
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Anthropic Claude Instant 模型进行文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Anthropic Claude Instant 模型来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude Instant 基础模型来生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
```

```
/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} MessageApiResponse
 * @property {Content[]} content
 */

/**
 * @typedef {Object} TextCompletionApiResponse
 * @property {string} completion
 */

/**
 * Invokes Anthropic Claude Instant using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-instant-v1".
 */
export const invokeModel = async (
 prompt,
 modelId = "anthropic.claude-instant-v1",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the Messages API request.
 const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
 };
};
```

```
// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {MessageApiResponse} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude Instant using the Text Completions API.
 *
 * To learn more about the Anthropic Text Completions API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-text-completion.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-instant-v1".
 */
export const invokeTextCompletionsApi = async (
 prompt,
 modelId = "anthropic.claude-instant-v1",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the Text Completions API, using the required prompt
 template.
 const enclosedPrompt = `Human: ${prompt}\n\nAssistant:`;
 const payload = {
 prompt: enclosedPrompt,
 max_tokens_to_sample: 500,
 temperature: 0.5,
 stop_sequences: ["\n\nHuman:"],
 };

 // Invoke Claude with the payload and wait for the response.
```

```
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response.
const decoded = new TextDecoder().decode(apiResponse.body);
/** @type {TextCompletionApiResponse} */
const responseBody = JSON.parse(decoded);
return responseBody.completion;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time...";
 const modelId = FoundationModels.CLAUDE_INSTANT.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 console.log("Using the Messages API:");
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }

 try {
 console.log("-".repeat(53));
 console.log("Using the Text Completions API:");
 const response = await invokeTextCompletionsApi(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Anthropic Claude 以生成带有响应流的文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Anthropic Claude 模型以生成带有响应流的文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [在 Amazon Bedrock 上调用多个基础模型](#)

### .NET

#### AWS SDK for .NET

##### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Anthropic Claude 并处理响应流。

```
/// <summary>
/// Asynchronously invokes the Anthropic Claude 2 model to run an
inference based on the provided input and process the response stream.
/// </summary>
/// <param name="prompt">The prompt that you want Claude to complete.</
param>
/// <returns>The inference response from the model</returns>
/// <remarks>
/// The different model providers have individual request and response
formats.
/// For the format, ranges, and default values for Anthropic Claude,
refer to:
/// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-claude.html
/// </remarks>
```



```
public static async IEnumerable<string>
InvokeClaudeWithResponseStreamAsync(string prompt, [EnumeratorCancellation]
Cancellation token cancellationToken = default)
{
 string claudeModelId = "anthropic.claude-v2";

 // Claude requires you to enclose the prompt as follows:
 string enclosedPrompt = "Human: " + prompt + "\n\nAssistant:";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
 {
 { "prompt", enclosedPrompt },
 { "max_tokens_to_sample", 200 },
 { "temperature", 0.5 },
 { "stop_sequences", new JSONArray("\n\nHuman:") }
 }.ToJsonString();

 InvokeModelWithResponseStreamResponse? response = null;

 try
 {
 response = await client.InvokeModelWithResponseStreamAsync(new
InvokeModelWithResponseStreamRequest()
 {
 ModelId = claudeModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }

 if (response is not null && response.HttpStatusCode ==
System.Net.HttpStatusCode.OK)
 {
 // create a buffer to write the event in to move from a push mode
to a pull mode
 Channel<string> buffer = Channel.CreateUnbounded<string>();
 bool isStreaming = true;
```

```
 response.Body.ChunkReceived += BodyOnChunkReceived;
 response.Body.StartProcessing();

 while ((!cancellationToken.IsCancellationRequested
&& isStreaming) || (!cancellationToken.IsCancellationRequested &&
buffer.Reader.Count > 0))
 {
 // pull the completion from the buffer and add it to the
IAsyncEnumerable collection
 yield return await
buffer.Reader.ReadAsync(cancellationToken);
 }
 response.Body.ChunkReceived -= BodyOnChunkReceived;

 yield break;

 // handle the ChunkReceived events
 async void BodyOnChunkReceived(object? sender,
EventStreamEventReceivedArgs<PayloadPart> e)
 {
 var streamResponse =
JsonSerializer.Deserialize<JsonObject>(e.EventStreamEvent.Bytes) ??
throw new NullReferenceException($"Unable to deserialize
{nameof(e.EventStreamEvent.Bytes)}");

 if (streamResponse["stop_reason"]?.GetValue<string?>() !=
null)
 {
 isStreaming = false;
 }


 // write the received completion chunk into the buffer
await
buffer.Writer.WriteAsync(streamResponse["completion"]?.GetValue<string>(),
cancellationToken);
 }
 }
 else if (response is not null)
 {
 Console.WriteLine("InvokeModelAsync failed with status code " +
response.HttpStatusCode);
 }
}
```

```
 yield break;
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考 [InvokeModelWithResponseStream](#) 中的。

Go

适用于 Go V2 的 SDK

 Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 并处理响应流。

```
// Each model provider defines their own individual request and response formats.
// For the format, ranges, and default values for the different models, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters.html

type Request struct {
 Prompt string `json:"prompt"`
 MaxTokensToSample int `json:"max_tokens_to_sample"`
 Temperature float64 `json:"temperature,omitempty"`
}

type Response struct {
 Completion string `json:"completion"`
}

// Invokes Anthropic Claude on Amazon Bedrock to run an inference and
// asynchronously
// process the response stream.

func (wrapper InvokeModelWithResponseStreamWrapper)
 InvokeModelWithResponseStream(prompt string) (string, error) {
```

```
modelId := "anthropic.claude-v2"

// Anthropic Claude requires you to enclose the prompt as follows:
prefix := "Human: "
postfix := "\n\nAssistant:"
prompt = prefix + prompt + postfix

request := ClaudeRequest{
 Prompt: prompt,
 MaxTokensToSample: 200,
 Temperature: 0.5,
 StopSequences: []string{"\n\nHuman:"},
}

body, err := json.Marshal(request)
if err != nil {
 log.Panicln("Couldn't marshal the request: ", err)
}

output, err :=
wrapper.BedrockRuntimeClient.InvokeModelWithResponseStream(context.Background(),
&bedrockruntime.InvokeModelWithResponseStreamInput{
 Body: body,
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
})

if err != nil {
 errMsg := err.Error()
 if strings.Contains(errMsg, "no such host") {
 log.Printf("The Bedrock service is not available in the selected region.
Please double-check the service availability for your region at https://
aws.amazon.com/about-aws/global-infrastructure/regional-product-services/.\n")
 } else if strings.Contains(errMsg, "Could not resolve the foundation model") {
 log.Printf("Could not resolve the foundation model from model identifier: \"%v
\". Please verify that the requested model exists and is accessible within the
specified region.\n", modelId)
 } else {
 log.Printf("Couldn't invoke Anthropic Claude. Here's why: %v\n", err)
 }
}

resp, err := processStreamingOutput(output, func(ctx context.Context, part
[]byte) error {
```

```
 fmt.Print(string(part))
 return nil
})

if err != nil {
 log.Fatal("streaming output processing error: ", err)
}

return resp.Completion, nil
}

type StreamingOutputHandler func(ctx context.Context, part []byte) error

func processStreamingOutput(output
 *bedrockruntime.InvokeModelWithResponseStreamOutput, handler
 StreamingOutputHandler) (Response, error) {

 var combinedResult string
 resp := Response{}

 for event := range output.GetStream().Events() {
 switch v := event.(type) {
 case *types.ResponseStreamMemberChunk:

 //fmt.Println("payload", string(v.Value.Bytes))

 var resp Response
 err := json.NewDecoder(bytes.NewReader(v.Value.Bytes)).Decode(&resp)
 if err != nil {
 return resp, err
 }

 err = handler(context.Background(), []byte(resp.Completion))
 if err != nil {
 return resp, err
 }

 combinedResult += resp.Completion

 case *types.UnknownUnionMember:
 fmt.Println("unknown tag:", v.Tag)

 default:
```

```
 fmt.Println("union is nil or unknown type")
 }
}

resp.Completion = combinedResult

return resp, nil
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考 [InvokeModelWithResponseStream](#) 中的。

## Java

适用于 Java 2.x 的 SDK

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 并处理响应流。

```
/**
 * Invokes the Anthropic Claude 2 model and processes the response
stream.
 *
 * @param prompt The prompt for Claude to complete.
 * @param silent Suppress console output of the individual response
stream
 *
 * chunks.
 * @return The generated response.
 */
public static String invokeClaude(String prompt, boolean silent) {

 BedrockRuntimeAsyncClient client =
BedrockRuntimeAsyncClient.builder()
 .region(Region.US_EAST_1)
```

```
.credentialsProvider(ProfileCredentialsProvider.create())
 .build();

var finalCompletion = new AtomicReference<>("");

var payload = new JSONObject()
 .put("prompt", "Human: " + prompt + "
Assistant:")
 .put("temperature", 0.8)
 .put("max_tokens_to_sample", 300)
 .toString();

var request = InvokeModelWithResponseStreamRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId("anthropic.claude-v2")
 .contentType("application/json")
 .accept("application/json")
 .build();

var visitor =
InvokeModelWithResponseStreamResponseHandler.Visitor.builder()
 .onChunk(chunk -> {
 var json = new
JSONObject(chunk.bytes().asUtf8String());
 var completion =
json.getString("completion");
 finalCompletion.set(finalCompletion.get()
+ completion);
 if (!silent) {
 System.out.print(completion);
 }
 })
 .build();

var handler =
InvokeModelWithResponseStreamResponseHandler.builder()
 .onEventStream(stream -> stream.subscribe(event -
> event.accept(visitor)))
 .onComplete(() -> {
 })
 .onError(e -> System.out.println("\n\nError: " +
e.getMessage()))
 .build();
```

```
 client.invokeModelWithResponseStream(request, handler).join();

 return finalCompletion.get();
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [InvokeModelWithResponseStream](#) 中的。

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 并处理响应流。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
 InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
```



```
* @property {string} text
*
* @typedef {Object} Message
* @property {string} role
*
* @typedef {Object} Chunk
* @property {string} type
* @property {Delta} delta
* @property {Message} message
*/

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
 prompt,
 modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the model.
 const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
 };

 // Invoke Claude with the payload and wait for the response.
 const command = new InvokeModelCommand({
 contentType: "application/json",
```

```
 body: JSON.stringify(payload),
 modelId,
 });
 const apiResponse = await client.send(command);

 // Decode and return the response(s)
 const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
 /** @type {MessagesResponseBody} */
 const responseBody = JSON.parse(decodedResponseBody);
 return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
 prompt,
 modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the model.
 const payload = {
 anthropic_version: "bedrock-2023-05-31",
 max_tokens: 1000,
 messages: [
 {
 role: "user",
 content: [{ type: "text", text: prompt }],
 },
],
 };

 // Invoke Claude with the payload and wait for the API to respond.
 const command = new InvokeModelWithResponseStreamCommand({
```

```
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
 });
const apiResponse = await client.send(command);

let completeMessage = "";

// Decode and process the response stream
for await (const item of apiResponse.body) {
 /** @type Chunk */
 const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
 const chunk_type = chunk.type;

 if (chunk_type === "content_block_delta") {
 const text = chunk.delta.text;
 completeMessage = completeMessage + text;
 process.stdout.write(text);
 }
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt = 'Write a paragraph starting with: "Once upon a time...";
 const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log("\n" + "-".repeat(53));
 console.log("Final structured response:");
 console.log(response);
 } catch (err) {
 console.log(`\n${err}`);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModelWithResponseStream](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Anthropic Claude 并处理响应流。

```
async def invoke_model_with_response_stream(self, prompt):
 """
 Invokes the Anthropic Claude 2 model to run an inference and process the
 response stream.

 :param prompt: The prompt that you want Claude to complete.
 :return: Inference response from the model.
 """

 try:
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Anthropic Claude,
 # refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 # parameters-claude.html

 # Claude requires you to enclose the prompt as follows:
 enclosed_prompt = "Human: " + prompt + "\n\nAssistant:"

 body = {
 "prompt": enclosed_prompt,
 "max_tokens_to_sample": 1024,
 "temperature": 0.5,
 "stop_sequences": ["\n\nHuman:"],
 }
```

```
 response =
self.bedrock_runtime_client.invoke_model_with_response_stream(
 modelId="anthropic.claude-v2", body=json.dumps(body)
)

 for event in response.get("body"):
 chunk = json.loads(event["chunk"]["bytes"])["completion"]
 yield chunk

 except ClientError:
 logger.error("Couldn't invoke Anthropic Claude v2")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModelWithResponseStream](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本

以下代码示例演示了如何在 Amazon Bedrock 上调用 Meta Llama 2 Chat 模型来生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Meta Llama 2 基础模型以生成文本。

```
 /// <summary>
 /// Asynchronously invokes the Meta Llama 2 Chat model to run an
inference based on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that you want Llama 2 to complete.</
param>
 /// <returns>The inference response from the model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
formats.
 /// For the format, ranges, and default values for Meta Llama 2 Chat,
refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-meta.html
 /// </remarks>
 public static async Task<string> InvokeLlama2Async(string prompt)
 {
 string llama2ModelId = "meta.llama2-13b-chat-v1";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USEast1);

 string payload = new JsonObject()
 {
 { "prompt", prompt },
 { "max_gen_len", 512 },
 { "temperature", 0.5 },
 { "top_p", 0.9 }
 }.ToJsonString();

 string generatedText = "";
 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = llama2ModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

```

```
 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 return JsonNode.ParseAsync(response.Body)
 .Result?["generation"]?.GetValue<string>() ?? "";
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return generatedText;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。

## Go

### 适用于 Go V2 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Meta Llama 2 Chat 基础模型以生成文本。

```
// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for Meta Llama 2 Chat, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
meta.html

type Llama2Request struct {
 Prompt string `json:"prompt"`
```

```
MaxGenLength int `json:"max_gen_len,omitempty"`
Temperature float64 `json:"temperature,omitempty"`
}

type Llama2Response struct {
 Generation string `json:"generation"`
}

// Invokes Meta Llama 2 Chat on Amazon Bedrock to run an inference using the
// input
// provided in the request body.
func (wrapper InvokeModelWrapper) InvokeLlama2(prompt string) (string, error) {
 modelId := "meta.llama2-13b-chat-v1"

 body, err := json.Marshal(Llama2Request {
 Prompt: prompt,
 MaxGenLength: 512,
 Temperature: 0.5,
 })

 if err != nil { log.Fatal("failed to marshal", err) }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(context.TODO(),
 &bedrockruntime.InvokeModelInput{
 ModelId: aws.String(modelId),
 ContentType: aws.String("application/json"),
 Body: body,
 })

 if err != nil { ProcessError(err, modelId) }

 var response Llama2Response
 if err := json.Unmarshal(output.Body, &response); err != nil {
 log.Fatal("failed to unmarshal", err)
 }

 return response.Generation, nil
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Go API 参考 [InvokeModel](#) 中的。



## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Meta Llama 2 Chat 基础模型以生成文本。

```
/**
 * Asynchronously invokes the Meta Llama 2 Chat model to run an inference
 based
 * on the provided input.
 *
 * @param prompt The prompt that you want Llama 2 to complete.
 * @return The inference response generated by the model.
 */
public static String invokeLlama2(String prompt) {
 /**
 * The different model providers have individual request and response
 formats.
 * For the format, ranges, and default values for Meta Llama 2 Chat,
 refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 meta.
 * html
 */

 String llama2ModelId = "meta.llama2-13b-chat-v1";

 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_EAST_1)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 String payload = new JSONObject()
 .put("prompt", prompt)
 .put("max_gen_len", 512)
 .put("temperature", 0.5)
 .put("top_p", 0.9)
```

```
 .toString());

 InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(llama2ModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 CompletableFuture<InvokeModelResponse> completableFuture =
client.invokeModel(request)
 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " +
exception);
 }
 });

 String generatedText = "";
 try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 generatedText = responseBody.getString("generation");

 } catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
 } catch (ExecutionException e) {
 System.err.println(e.getMessage());
 }

 return generatedText;
}
```

调用 Meta Llama 2 Chat 基础模型以生成文本。

```
/**
 * Invokes the Meta Llama 2 Chat model to run an inference based on the
provided
 * input.
 *

```

```
 * @param prompt The prompt for Llama 2 to complete.
 * @return The generated response.
 */
 public static String invokeLlama2(String prompt) {
 /*
 * The different model providers have individual request and
 response formats.
 * For the format, ranges, and default values for Meta Llama 2
 Chat, refer to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-meta.
 * html
 */

 String llama2ModelId = "meta.llama2-13b-chat-v1";

 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_EAST_1)

 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 String payload = new JSONObject()
 .put("prompt", prompt)
 .put("max_gen_len", 512)
 .put("temperature", 0.5)
 .put("top_p", 0.9)
 .toString();

 InvokeModelRequest request = InvokeModelRequest.builder()
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(llama2ModelId)
 .contentType("application/json")
 .accept("application/json")
 .build();

 InvokeModelResponse response = client.invokeModel(request);

 JSONObject responseBody = new
 JSONObject(response.body().asUtf8String());

 String generatedText = responseBody.getString("generation");

 return generatedText;
 }
}
```

```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [InvokeModel](#) 中的。

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Meta Llama 2 Chat 基础模型以生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} text
 */

/**
 * Invokes a Meta Llama 2 Chat model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "meta.llama2-13b-chat-v1".
 */
export const invokeModel = async (
 prompt,
```

```
modelId = "meta.llama2-13b-chat-v1",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Prepare the payload for the model.
 const payload = {
 prompt,
 temperature: 0.5,
 max_gen_len: 1000,
 };

 // Invoke the model with the payload and wait for the response.
 const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
 });
 const apiResponse = await client.send(command);

 // Decode and return the response.
 const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
 /** @type {ResponseBody} */
 const responseBody = JSON.parse(decodedResponseBody);
 return responseBody.generation;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time...";
 const modelId = FoundationModels.LLAMA2_CHAT_13B.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## PHP

### 适用于 PHP 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Meta Llama 2 Chat 基础模型以生成文本。

```
public function invokeLlama2($prompt)
{
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Meta Llama 2 Chat, refer
 # to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
 # meta.html

 $completion = "";

 try {
 $modelId = 'meta.llama2-13b-chat-v1';

 $body = [
 'prompt' => $prompt,
 'temperature' => 0.5,
 'max_gen_len' => 512,
];

 $result = $this->bedrockRuntimeClient->invokeModel([
 'contentType' => 'application/json',
 'body' => json_encode($body),
 'modelId' => $modelId,
]);
 }
```

```
 $response_body = json_decode($result['body']);

 $completion = $response_body->generation;
 } catch (Exception $e) {
 echo "Error: ({$e->getCode()}) - {$e->getMessage()}\n";
 }

 return $completion;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Meta Llama 2 Chat 基础模型以生成文本。

```
def invoke_llama2(self, prompt):
 """
 Invokes the Meta Llama 2 large-language model to run an inference
 using the input provided in the request body.

 :param prompt: The prompt that you want Llama 2 to complete.
 :return: Inference response from the model.
 """

 try:
 # The different model providers have individual request and response
 # formats.
 # For the format, ranges, and default values for Meta Llama 2 Chat,
 # refer to:
 # https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 # parameters-meta.html
```

```
body = {
 "prompt": prompt,
 "temperature": 0.5,
 "top_p": 0.9,
 "max_gen_len": 512,
}

response = self.bedrock_runtime_client.invoke_model(
 modelId="meta.llama2-13b-chat-v1", body=json.dumps(body)
)

response_body = json.loads(response["body"].read())
completion = response_body["generation"]

return completion

except ClientError:
 logger.error("Couldn't invoke Llama 2")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Mistral 7B 模型生成文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Mistral 7B 模型进行文本生成。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)



## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Mistral 7B 基础模型来生成文本。

```
 /// <summary>
 /// Asynchronously invokes the Mistral 7B model to run an inference based
 on the provided input.
 /// </summary>
 /// <param name="prompt">The prompt that you want Mistral 7B to
 complete.</param>
 /// <returns>The inference response from the model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
 formats.
 /// For the format, ranges, and default values for Mistral 7B, refer to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
 parameters-mistral.html
 /// </remarks>
 public static async Task<List<string?>> InvokeMistral7BAsync(string
 prompt)
 {
 string mistralModelId = "mistral.mistral-7b-instruct-v0:2";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USWest2);

 string payload = new JsonObject()
 {
 { "prompt", prompt },
 { "max_tokens", 200 },
 { "temperature", 0.5 }
 }.ToJsonString();

 List<string?>? generatedText = null;
 try
```

```
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = mistralModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 var results = JsonNode.ParseAsync(response.Body).Result?
["outputs"]?.ToArray();

 generatedText = results?.Select(x => x?
["text"]?.GetValue<string?>())?.ToList();
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
 }
 catch (AmazonBedrockRuntimeException e)
 {
 Console.WriteLine(e.Message);
 }
 return generatedText ?? [];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。

## Java

### 适用于 Java 2.x 的 SDK

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Mistral 7B 基础模型来生成文本。

```
/**
 * Asynchronously invokes the Mistral 7B model to run an inference based on
 the provided input.
 *
 * @param prompt The prompt for Mistral to complete.
 * @return The generated response.
 */
public static List<String> invokeMistral7B(String prompt) {
 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_WEST_2)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 // Mistral instruct models provide optimal results when
 // embedding the prompt into the following template:
 String instruction = "<s>[INST] " + prompt + " [/INST]";

 String modelId = "mistral.mistral-7b-instruct-v0:2";

 String payload = new JSONObject()
 .put("prompt", instruction)
 .put("max_tokens", 200)
 .put("temperature", 0.5)
 .toString();

 CompletableFuture<InvokeModelResponse> completableFuture =
client.invokeModel(request -> request
 .accept("application/json")
 .contentType("application/json")
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(modelId))
```

```

 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " + exception);
 }
 });

 try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 JSONArray outputs = responseBody.getJSONArray("outputs");

 return IntStream.range(0, outputs.length())
 .mapToObj(i -> outputs.getJSONObject(i).getString("text"))
 .toList();
 } catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
 } catch (ExecutionException e) {
 System.err.println(e.getMessage());
 }

 return List.of();
}

```

调用 Mistral 7B 基础模型生成文本。

```

/**
 * Invokes the Mistral 7B model to run an inference based on the provided
input.
 *
 * @param prompt The prompt for Mistral to complete.
 * @return The generated responses.
 */
public static List<String> invokeMistral7B(String prompt) {
 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_WEST_2)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 // Mistral instruct models provide optimal results when
 // embedding the prompt into the following template:

```

```
String instruction = "<s>[INST] " + prompt + " [/INST]";

String modelId = "mistral.mistral-7b-instruct-v0:2";

String payload = new JSONObject()
 .put("prompt", instruction)
 .put("max_tokens", 200)
 .put("temperature", 0.5)
 .toString();

InvokeModelResponse response = client.invokeModel(request ->
request
 .accept("application/json")
 .contentType("application/json")
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(modelId));

JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
JSONArray outputs = responseBody.getJSONArray("outputs");

return IntStream.range(0, outputs.length())
 .mapToObj(i ->
outputs.getJSONObject(i).getString("text"))
 .toList();

}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [InvokeModel](#) 中的。

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Mistral 7B 基础模型生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
 prompt,
 modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
 // Create a new Bedrock Runtime client instance.
 const client = new BedrockRuntimeClient({ region: "us-east-1" });

 // Mistral instruct models provide optimal results when embedding
 // the prompt into the following template:
 const instruction = `[INST] ${prompt} [/INST]`;

 // Prepare the payload.
 const payload = {
 prompt: instruction,
 max_tokens: 500,
 temperature: 0.5,
 };
};
```

```
// Invoke the model with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response.
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time..."';
 const modelId = FoundationModels.MISTRAL_7B.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Mistral 7B 基础模型生成文本。

```
def invoke_mistral_7b(self, prompt):
 """
 Invokes the Mistral 7B model to run an inference using the input
 provided in the request body.

 :param prompt: The prompt that you want Mistral to complete.
 :return: List of inference responses from the model.
 """

 try:
 # Mistral instruct models provide optimal results when
 # embedding the prompt into the following template:
 instruction = f"<s>[INST] {prompt} [/INST]"

 model_id = "mistral.mistral-7b-instruct-v0:2"

 body = {
 "prompt": instruction,
 "max_tokens": 200,
 "temperature": 0.5,
 }

 response = self.bedrock_runtime_client.invoke_model(
 modelId=model_id, body=json.dumps(body)
)

 response_body = json.loads(response["body"].read())
 outputs = response_body.get("outputs")

 completions = [output["text"] for output in outputs]
```



```
 return completions

 except ClientError:
 logger.error("Couldn't invoke Mistral 7B")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用 Mixtral 8x7b 模型生成文本

以下代码示例展示了如何在 Amazon Bedrock 上调用 Mixtral 8x7b 模型模型以生成文本。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- [互动文字游乐场](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [在 Amazon Bedrock 上调用多个 LLM](#)

## .NET

### AWS SDK for .NET

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Mixtral 8x7b 基础模型来生成文本。

```
 /// <summary>
 /// Asynchronously invokes the Mixtral 8x7B model to run an inference
 based on the provided input.
```

```
 /// </summary>
 /// <param name="prompt">The prompt that you want Mixtral 8x7B to
complete.</param>
 /// <returns>The inference response from the model</returns>
 /// <remarks>
 /// The different model providers have individual request and response
formats.
 /// For the format, ranges, and default values for Mixtral 8x7B, refer
to:
 /// https://docs.aws.amazon.com/bedrock/latest/userguide/model-
parameters-mistral.html
 /// </remarks>
 public static async Task<List<string?>> InvokeMixtral8x7BAsync(string
prompt)
 {
 string mixtralModelId = "mistral.mixtral-8x7b-instruct-v0:1";

 AmazonBedrockRuntimeClient client = new(RegionEndpoint.USWest2);

 string payload = new JsonObject()
 {
 { "prompt", prompt },
 { "max_tokens", 200 },
 { "temperature", 0.5 }
 }.ToJsonString();

 List<string?>? generatedText = null;
 try
 {
 InvokeModelResponse response = await client.InvokeModelAsync(new
InvokeModelRequest()
 {
 ModelId = mixtralModelId,
 Body = AWSSDKUtils.GenerateMemoryStreamFromString(payload),
 ContentType = "application/json",
 Accept = "application/json"
 });

 if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
 {
 var results = JsonNode.ParseAsync(response.Body).Result?
["outputs"]?.ToArray();
 }
 }
 }
}
```

```
 generatedText = results?.Select(x => x?
["text"]?.GetValue<string?>())?.ToList();
 }
 else
 {
 Console.WriteLine("InvokeModelAsync failed with status code "
+ response.HttpStatusCode);
 }
}
catch (AmazonBedrockRuntimeException e)
{
 Console.WriteLine(e.Message);
}
return generatedText ?? [];
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[InvokeModel](#)中的。

## Java

适用于 Java 2.x 的 SDK

### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

异步调用 Mistral 8x7B 基础模型来生成文本。

```
/**
 * Asynchronously invokes the Mixtral 8x7B model to run an inference based on
 the provided input.
 *
 * @param prompt The prompt for Mixtral to complete.
 * @return The generated response.
 */
public static List<String> invokeMixtral8x7B(String prompt) {
 BedrockRuntimeAsyncClient client = BedrockRuntimeAsyncClient.builder()
 .region(Region.US_WEST_2)
```

```
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

// Mistral instruct models provide optimal results when
// embedding the prompt into the following template:
String instruction = "<s>[INST] " + prompt + " [/INST]";

String modelId = "mistral.mixtral-8x7b-instruct-v0:1";

String payload = new JSONObject()
 .put("prompt", instruction)
 .put("max_tokens", 200)
 .put("temperature", 0.5)
 .toString();

CompletableFuture<InvokeModelResponse> completableFuture =
client.invokeModel(request -> request
 .accept("application/json")
 .contentType("application/json")
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(modelId))
 .whenComplete((response, exception) -> {
 if (exception != null) {
 System.out.println("Model invocation failed: " +
exception);
 }
 });

try {
 InvokeModelResponse response = completableFuture.get();
 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 JSONArray outputs = responseBody.getJSONArray("outputs");

 return IntStream.range(0, outputs.length())
 .mapToObj(i -> outputs.getJSONObject(i).getString("text"))
 .toList();
} catch (InterruptedException e) {
 Thread.currentThread().interrupt();
 System.err.println(e.getMessage());
} catch (ExecutionException e) {
 System.err.println(e.getMessage());
}
```

```

 return List.of();
}

```

调用 Mixtral 8x7b 基础模型来生成文本。

```

public static List<String> invokeMixtral8x7B(String prompt) {
 BedrockRuntimeClient client = BedrockRuntimeClient.builder()
 .region(Region.US_WEST_2)
 .credentialsProvider(ProfileCredentialsProvider.create())
 .build();

 // Mistral instruct models provide optimal results when
 // embedding the prompt into the following template:
 String instruction = "<s>[INST] " + prompt + " [/INST]";

 String modelId = "mistral.mixtral-8x7b-instruct-v0:1";

 String payload = new JSONObject()
 .put("prompt", instruction)
 .put("max_tokens", 200)
 .put("temperature", 0.5)
 .toString();

 InvokeModelResponse response = client.invokeModel(request ->
request
 .accept("application/json")
 .contentType("application/json")
 .body(SdkBytes.fromUtf8String(payload))
 .modelId(modelId));

 JSONObject responseBody = new
JSONObject(response.body().asUtf8String());
 JSONArray outputs = responseBody.getJSONArray("outputs");

 return IntStream.range(0, outputs.length())
 .mapToObj(i ->
outputs.getJSONObject(i).getString("text"))
 .toList();
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考 [InvokeModel](#) 中的。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Mistral 8x7b 基础模型来生成文本。

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

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
 BedrockRuntimeClient,
 InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 8x7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "mistral.mixtral-8x7b-instruct-v0:1".
 */
export const invokeModel = async (
 prompt,
 modelId = "mistral.mixtral-8x7b-instruct-v0:1",
) => {
 // Create a new Bedrock Runtime client instance.
```

```
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Mistral instruct models provide optimal results when embedding
// the prompt into the following template:
const instruction = `[INST] ${prompt} [/INST]`;

// Prepare the payload.
const payload = {
 prompt: instruction,
 max_tokens: 500,
 temperature: 0.5,
};

// Invoke the model with the payload and wait for the response.
const command = new InvokeModelCommand({
 contentType: "application/json",
 body: JSON.stringify(payload),
 modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response.
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const prompt =
 'Complete the following in one sentence: "Once upon a time...";
 const modelId = FoundationModels.MIXTRAL_8X7B.modelId;
 console.log(`Prompt: ${prompt}`);
 console.log(`Model ID: ${modelId}`);

 try {
 console.log("-".repeat(53));
 const response = await invokeModel(prompt, modelId);
 console.log(response);
 } catch (err) {
 console.log(err);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用 Mixtral 8x7b 基础模型来生成文本。

```
def invoke_mixtral_8x7b(self, prompt):
 """
 Invokes the Mixtral 8c7B model to run an inference using the input
 provided in the request body.

 :param prompt: The prompt that you want Mixtral to complete.
 :return: List of inference responses from the model.
 """

 try:
 # Mistral instruct models provide optimal results when
 # embedding the prompt into the following template:
 instruction = f"<s>[INST] {prompt} [/INST]"

 model_id = "mistral.mixtral-8x7b-instruct-v0:1"

 body = {
 "prompt": instruction,
 "max_tokens": 200,
 "temperature": 0.5,
 }

 response = self.bedrock_runtime_client.invoke_model(
 modelId=model_id, body=json.dumps(body)
)
```



```
response_body = json.loads(response["body"].read())
outputs = response_body.get("outputs")

completions = [output["text"] for output in outputs]

return completions

except ClientError:
 logger.error("Couldn't invoke Mixtral 8x7B")
 raise
```

- 有关 API 的详细信息，请参阅适用[InvokeModel](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 SDK 的 Amazon Bedrock Runtime AWS 场景

以下代码示例向您展示了如何使用 AWS 软件开发工具包在 Amazon Bedrock Runtime 中实现常见场景。这些场景向您展示了如何通过调用多个函数来完成特定任务。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

### 示例

- [使用软件开发工具包创建提供游乐场的示例应用程序，以便与 Amazon Bedrock 基础模型进行交互 AWS](#)
- [适用于 Amazon Bedrock 的交互式文本生成平台](#)
- [在 Amazon Bedrock 上调用多个大型语言模型 \(LLM\)](#)
- [在 Amazon Bedrock 上调用多个基础模型](#)
- [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用软件开发工具包创建提供游乐场的示例应用程序，以便与 Amazon Bedrock 基础模型进行交互 AWS

以下代码示例演示了如何创建操场，以通过不同模态与 Amazon Bedrock 基础模型交互。

## .NET

### AWS SDK for .NET

.NET Foundation Model (FM) Playground 是一个 .NET MAUI Blazor 示例应用程序，展示了如何通过 C# 代码使用 Amazon Bedrock。此示例展示了 .NET 和 C# 开发人员如何使用 Amazon Bedrock 来构建生成式人工智能赋能的应用程序。您可以使用以下四个平台测试 Amazon Bedrock 基础模型并与其交互：

- 文本平台。
- 聊天平台。
- 一个语音聊天平台。
- 图像平台。

该示例还列出并显示了您可以访问的基础模型及其特征。有关源代码和部署说明，请参阅中的项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock 运行时系统

## Java

### 适用于 Java 2.x 的 SDK

Java Foundation Model (FM) Playground 是一款 Spring Boot 示例应用程序，演示了如何将 Amazon Bedrock 与 Java 结合使用。此示例演示 Java 开发人员可如何使用 Amazon Bedrock 来构建支持生成式人工智能的应用程序。您可以使用以下三个操场测试 Amazon Bedrock 基础模型并与其交互：

- 文本操场。
- 聊天操场。
- 图像操场。

该示例还列出并显示您可以访问的基础模型及其特点。有关源代码和部署说明，请参阅中的项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock 运行时系统

## Python

### SDK for Python (Boto3)

Python Foundation Model (FM) Playground 是一款 Python/FastAPI 示例应用程序，演示如何将 Amazon Bedrock 与 Python 结合使用。此示例演示 Python 开发人员可如何使用 Amazon Bedrock 来构建支持生成式人工智能的应用程序。您可以使用以下三个操场测试 Amazon Bedrock 基础模型并与其交互：

- 文本操场。
- 聊天操场。
- 图像操场。

该示例还列出并显示您可以访问的基础模型及其特点。有关源代码和部署说明，请参阅中的项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock 运行时系统

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

### 适用于 Amazon Bedrock 的交互式文本生成平台

以下代码示例展示了如何在 Amazon Bedrock 上准备并向以下大型语言模型 (LLM) 发送提示：

- AI21 实验室：Jurassic2 中型和超级
- 亚马逊：Titan Text G1 Lite 和 Express
- Anthropic：克劳德·即时
- Anthropic：克劳德 2.0 和 2.1
- Anthropic：Claude 3 俳句和十四行诗
- 元数据：Llama2 Chat 13B 和 70B
- Mistral AI：Mistral 7B 和 Mixtral 8x7b

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

import { fileURLToPath } from "url";
import {
 Scenario,
 ScenarioAction,
 ScenarioInput,
 ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
 "greeting",
 "Welcome to the Amazon Bedrock Runtime client demo!",
 { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
 type: "select",
 choices: Object.values(FoundationModels).map((model) => ({
 name: model.modelName,
 value: model,
 })),
})),
```

```
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
 type: "input",
});

const printDetails = new ScenarioOutput(
 "print details",
 /**
 * @param {{ model: ModelConfig, prompt: string }} c
 */
 (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
 { slow: false },
);

const invokeModel = new ScenarioAction(
 "invoke model",
 /**
 * @param {{ model: ModelConfig, prompt: string, response: string }} c
 */
 async (c) => {
 const modelModule = await c.model.module();
 const invoker = c.model.invoker(modelModule);
 c.response = await invoker(c.prompt, c.model.modelId);
 },
);

const printResponse = new ScenarioOutput(
 "print response",
 /**
 * @param {{ response: string }} c
 */
 (c) => c.response,
 { slow: false },
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
 greeting,
 selectModel,
 enterPrompt,
 printDetails,
 invokeModel,
 printResponse,
]);
```

```
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 scenario.run();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeModel](#) 中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用多个大型语言模型 ( LLM )

以下代码示例显示了如何在 Amazon Bedrock 上调用多个 large-language-models (LLM)。

- 使用 Anthropic Claude 生成文本。
- 使用 AI21 Labs Jurassic-2 生成文本。
- 使用 Meta Llama 2 Chat 生成文本。

## PHP

### 适用于 PHP 的 SDK

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在 Amazon Bedrock 上调用多个 LLM。

```
namespace BedrockRuntime;

class GettingStartedWithBedrockRuntime
{
 protected BedrockRuntimeService $bedrockRuntimeService;

 public function runExample()
 {
 echo "\n";
 }
}
```

```
 echo
 "-----\n";
 echo "Welcome to the Amazon Bedrock Runtime getting started demo using
PHP!\n";
 echo
 "-----\n";

 $clientArgs = [
 'region' => 'us-east-1',
 'version' => 'latest',
 'profile' => 'default',
];

 $bedrockRuntimeService = new BedrockRuntimeService($clientArgs);

 $prompt = 'In one paragraph, who are you?';

 echo "\nPrompt: " . $prompt;

 echo "\n\nAnthropic Claude:";
 echo $bedrockRuntimeService->invokeClaude($prompt);

 echo "\n\nAI21 Labs Jurassic-2: ";
 echo $bedrockRuntimeService->invokeJurassic2($prompt);

 echo "\n\nMeta Llama 2 Chat: ";
 echo $bedrockRuntimeService->invokeLlama2($prompt);

 echo
 "\n-----\n";

 $image_prompt = 'stylized picture of a cute old steampunk robot';

 echo "\nImage prompt: " . $image_prompt;

 echo "\n\nStability.ai Stable Diffusion XL:\n";
 $diffusionSeed = rand(0, 4294967295);
 $style_preset = 'photographic';
 $base64 = $bedrockRuntimeService->invokeStableDiffusion($image_prompt,
$diffusionSeed, $style_preset);
 $image_path = $this->saveImage($base64, 'stability.stable-diffusion-xl');
 echo "The generated images have been saved to $image_path";

 echo "\n\nAmazon Titan Image Generation:\n";
```

```
 $titanSeed = rand(0, 2147483647);
 $base64 = $bedrockRuntimeService->invokeTitanImage($image_prompt,
 $titanSeed);
 $image_path = $this->saveImage($base64, 'amazon.titan-image-generator-
v1');
 echo "The generated images have been saved to $image_path";
 }

 private function saveImage($base64_image_data, $model_id): string
 {
 $output_dir = "output";

 if (!file_exists($output_dir)) {
 mkdir($output_dir);
 }

 $i = 1;
 while (file_exists("$output_dir/$model_id" . '_' . "$i.png")) {
 $i++;
 }

 $image_data = base64_decode($base64_image_data);

 $file_path = "$output_dir/$model_id" . '_' . "$i.png";

 $file = fopen($file_path, 'wb');
 fwrite($file, $image_data);
 fclose($file);

 return $file_path;
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for PHP API 参考[InvokeModel](#)中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 在 Amazon Bedrock 上调用多个基础模型


以下代码示例演示了如何在 Amazon Bedrock 上调用多个基础模型。



- 使用 Anthropic Claude 生成文本。
- 使用 AI21 Labs Jurassic-2 生成文本。
- 使用 Meta Llama 2 Chat 生成文本。
- 异步处理来自 Anthropic Claude 的响应流。
- 使用 Amazon Titan Image Generator 生成图像。

Go

适用于 Go V2 的 SDK

 Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

在 Amazon Bedrock 上调用多个基础模型。

```
// InvokeModelsScenario demonstrates how to use the Amazon Bedrock Runtime client
// to invoke various foundation models for text and image generation
//
// 1. Generate text with Anthropic Claude 2
// 2. Generate text with AI21 Labs Jurassic-2
// 3. Generate text with Meta Llama 2 Chat
// 4. Generate text and asynchronously process the response stream with Anthropic
// Claude 2
// 5. Generate and image with the Amazon Titan image generation model
type InvokeModelsScenario struct {
 sdkConfig aws.Config
 invokeModelWrapper actions.InvokeModelWrapper
 responseStreamWrapper actions.InvokeModelWithResponseStreamWrapper
 questioner demotools.IQuestioner
}

// NewInvokeModelsScenario constructs an InvokeModelsScenario instance from a
// configuration.
// It uses the specified config to get a Bedrock Runtime client and create
// wrappers for the
// actions used in the scenario.
```

```
func NewInvokeModelsScenario(sdkConfig aws.Config, questioner
demotools.IQuestioner) InvokeModelsScenario {
client := bedrockruntime.NewFromConfig(sdkConfig)
return InvokeModelsScenario{
sdkConfig: sdkConfig,
invokeModelWrapper: actions.InvokeModelWrapper{BedrockRuntimeClient:
client},
responseStreamWrapper:
actions.InvokeModelWithResponseStreamWrapper{BedrockRuntimeClient: client},
questioner: questioner,
}
}

// Runs the interactive scenario.
func (scenario InvokeModelsScenario) Run() {
defer func() {
if r := recover(); r != nil {
log.Printf("Something went wrong with the demo: %v\n", r)
}
}()

log.Println(strings.Repeat("=", 77))
log.Println("Welcome to the Amazon Bedrock Runtime model invocation demo.")
log.Println(strings.Repeat("=", 77))

log.Printf("First, let's invoke a few large-language models using the
synchronous client:\n\n")

text2textPrompt := "In one paragraph, who are you?"

log.Println(strings.Repeat("-", 77))
log.Printf("Invoking Claude with prompt: %v\n", text2textPrompt)
scenario.InvokeClaude(text2textPrompt)

log.Println(strings.Repeat("-", 77))
log.Printf("Invoking Jurassic-2 with prompt: %v\n", text2textPrompt)
scenario.InvokeJurassic2(text2textPrompt)

log.Println(strings.Repeat("-", 77))
log.Printf("Invoking Llama2 with prompt: %v\n", text2textPrompt)
scenario.InvokeLlama2(text2textPrompt)

log.Println(strings.Repeat("=", 77))
```

```
log.Printf("Now, let's invoke Claude with the asynchronous client and process
the response stream:\n\n")

log.Println(strings.Repeat("-", 77))
log.Printf("Invoking Claude with prompt: %v\n", text2textPrompt)
scenario.InvokeWithResponseStream(text2textPrompt)

log.Println(strings.Repeat("=", 77))
log.Printf("Now, let's create an image with the Amazon Titan image generation
model:\n\n")

text2ImagePrompt := "stylized picture of a cute old steampunk robot"
seed := rand.Int63n(2147483648)

log.Println(strings.Repeat("-", 77))
log.Printf("Invoking Amazon Titan with prompt: %v\n", text2ImagePrompt)
scenario.InvokeTitanImage(text2ImagePrompt, seed)

log.Println(strings.Repeat("=", 77))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("=", 77))
}

func (scenario InvokeModelsScenario) InvokeClaude(prompt string) {
 completion, err := scenario.invokeModelWrapper.InvokeClaude(prompt)
 if err != nil {
 panic(err)
 }
 log.Printf("\nClaude : %v\n", strings.TrimSpace(completion))
}

func (scenario InvokeModelsScenario) InvokeJurassic2(prompt string) {
 completion, err := scenario.invokeModelWrapper.InvokeJurassic2(prompt)
 if err != nil {
 panic(err)
 }
 log.Printf("\nJurassic-2 : %v\n", strings.TrimSpace(completion))
}

func (scenario InvokeModelsScenario) InvokeLlama2(prompt string) {
 completion, err := scenario.invokeModelWrapper.InvokeLlama2(prompt)
 if err != nil {
 panic(err)
 }
}
```

```
log.Printf("\nLlama 2 : %v\n\n", strings.TrimSpace(completion))
}

func (scenario InvokeModelsScenario) InvokeWithResponseStream(prompt string) {
 log.Println("\nClaude with response stream:")
 _, err := scenario.responseStreamWrapper.InvokeModelWithResponseStream(prompt)
 if err != nil {
 panic(err)
 }
 log.Println()
}

func (scenario InvokeModelsScenario) InvokeTitanImage(prompt string, seed int64)
{
 base64ImageData, err := scenario.invokeModelWrapper.InvokeTitanImage(prompt,
 seed)
 if err != nil {
 panic(err)
 }
 imagePath := saveImage(base64ImageData, "amazon.titan-image-generator-v1")
 fmt.Printf("The generated image has been saved to %s\n", imagePath)
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Go API 参考》中的以下主题。
  - [InvokeModel](#)
  - [InvokeModelWithResponseStream](#)

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序

以下代码示例展示了如何使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序。

## Python

### SDK for Python (Boto3)

Amazon Bedrock Serverless Pro [AWS Step Functions](#) [Prompt Chaining](#) 场景演示了如何使用 [Amazon Bedrock](#) 和 [Amazon Bedrock](#) 代理来构建和编排复杂、无服务器且高度可扩展的生成人工智能应用程序。它包含以下工作示例：

- 为文学博客撰写对给定小说的分析。此示例说明了一个简单的、按顺序排列的提示链。
- 生成有关给定主题的短篇小说。此示例说明了 AI 如何迭代处理其先前生成的项目列表。
- 创建前往给定目的地的周末度假行程。此示例说明如何并行处理多个不同的提示。
- 向扮演电影制作人的人类用户推销电影创意。此示例说明了如何使用不同的推理参数对同一个提示进行并行处理，如何回溯到链中的上一个步骤，以及如何将人工输入作为工作流程的一部分。
- 根据用户手头的食材计划膳食。这个例子说明了提示链如何整合两个不同的人工智能对话，两个人工智能角色相互进行辩论以改善最终结果。
- 查找并总结当今最热门的 GitHub 存储库。此示例说明如何链接多个与外部 API 交互的 AI 代理。

有关完整的源代码以及设置和运行说明，请参阅上的完整项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock
- Amazon Bedrock 运行时系统
- Agents for Amazon Bedrock
- 亚马逊 Bedrock Runtime 的代理
- Step Functions

有关 [AWS SDK 开发者指南](#) 和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 AWS 软件开发工具包的 Amazon Bedrock 代理的代码示例

以下代码示例展示了如何使用带有 AWS 软件开发套件 (SDK) 的 Amazon Bedrock 代理。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 开始使用

你好 Amazon Bedrock 的代理人

以下代码示例显示了如何开始使用适用于 Amazon Bedrock 的代理。

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

import { fileURLToPath } from "url";

import {
 BedrockAgentClient,
 GetAgentCommand,
 paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 * Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a
 * specific region.
```

```

* It then retrieves a list of existing agents using the streamlined paginator
approach.
* For each agent found, it retrieves detailed information using a command
object.
*
* Demonstrates:
* - Use of the Bedrock Agents client to initialize and communicate with the AWS
service.
* - Listing resources in a paginated response pattern.
* - Accessing an individual resource using a command object.
*
* @returns {Promise<void>} A promise that resolves when the function has
completed execution.
*/
export const main = async () => {
 const region = "us-east-1";

 console.log("=".repeat(68));

 console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
 const client = new BedrockAgentClient({ region });

 console.log(`Retrieving the list of existing agents...`);
 const paginatorConfig = { client };
 const pages = paginateListAgents(paginatorConfig, {});

 /** @type {AgentSummary[]} */
 const agentSummaries = [];
 for await (const page of pages) {
 agentSummaries.push(...page.agentSummaries);
 }

 console.log(`Found ${agentSummaries.length} agents in ${region}.`);

 if (agentSummaries.length > 0) {
 for (const agentSummary of agentSummaries) {
 const agentId = agentSummary.agentId;
 console.log("=".repeat(68));
 console.log(`Retrieving agent with ID: ${agentId}:`);
 console.log("-".repeat(68));

 const command = new GetAgentCommand({ agentId });
 const response = await client.send(command);
 const agent = response.agent;

```

```
 console.log(` Name: ${agent.agentName}`);
 console.log(` Status: ${agent.agentStatus}`);
 console.log(` ARN: ${agent.agentArn}`);
 console.log(` Foundation model: ${agent.foundationModel}`);
 }
}
console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 await main();
}
```

- 有关 API 详细信息，请参阅 [AWS SDK for JavaScript API 参考](#) 中的以下主题。
  - [GetAgent](#)
  - [ListAgents](#)

## 代码示例

- [使用 AWS SDK 为 Amazon Bedrock 的代理提供的操作](#)
  - [使用软件开发工具包创建 Amazon Bedrock 代理 AWS](#)
  - [使用软件开发工具包创建 Amazon Bedrock 代理操作组 AWS](#)
  - [使用软件开发工具包创建 Amazon Bedrock 代理别名 AWS](#)
  - [使用软件开发工具包删除 Amazon Bedrock 代理 AWS](#)
  - [使用软件开发工具包删除 Amazon Bedrock 代理别名 AWS](#)
  - [使用软件开发工具包获取有关亚马逊 Bedrock 代理的信息 AWS](#)
  - [列出使用软件开发工具包的 Amazon Bedrock 代理的 AWS 操作组](#)
  - [使用软件开发工具包列出属于账户的 Amazon Bedrock 代理 AWS](#)
  - [列出与使用软件开发工具包的 Amazon Bedrock 代理相关的知识库 AWS](#)
  - [使用软件开发工具包准备 Amazon Bedrock 代理 AWS](#)
- [使用 AWS SDK 的 Amazon Bedrock 代理的场景](#)
  - [演 end-to-end 示如何使用软件开发工具包创建和调用 Amazon Bedrock 代理的 AWS 示例](#)
  - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)



## 使用 AWS SDK 为 Amazon Bedrock 的代理提供的操作

以下代码示例演示了如何使用 AWS 软件开发工具包执行各个 Amazon Bedrock 代理操作。这些摘录调用 Amazon Bedrock 的 Agents API，是必须在上下文中运行的大型程序的代码摘录。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。如需完整列表，请参阅 [Amazon Bedrock 代理 API 参考](#)。

### 示例

- [使用软件开发工具包创建 Amazon Bedrock 代理 AWS](#)
- [使用软件开发工具包创建 Amazon Bedrock 代理操作组 AWS](#)
- [使用软件开发工具包创建 Amazon Bedrock 代理别名 AWS](#)
- [使用软件开发工具包删除 Amazon Bedrock 代理 AWS](#)
- [使用软件开发工具包删除 Amazon Bedrock 代理别名 AWS](#)
- [使用软件开发工具包获取有关亚马逊 Bedrock 代理的信息 AWS](#)
- [列出使用软件开发工具包的 Amazon Bedrock 代理的 AWS 操作组](#)
- [使用软件开发工具包列出属于账户的 Amazon Bedrock 代理 AWS](#)
- [列出与使用软件开发工具包的 Amazon Bedrock 代理相关的知识库 AWS](#)
- [使用软件开发工具包准备 Amazon Bedrock 代理 AWS](#)

### 使用软件开发工具包创建 Amazon Bedrock 代理 AWS

以下代码示例展示了如何创建 Amazon Bedrock 代理。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

#### 创建 代理

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
 BedrockAgentClient,
 CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with
permissions required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
containing details of the created agent.
 */
export const createAgent = async (
 agentName,
 foundationModel,
 agentResourceRoleArn,
 region = "us-east-1",
) => {
 const client = new BedrockAgentClient({ region });
```

```
const command = new CreateAgentCommand({
 agentName,
 foundationModel,
 agentResourceRoleArn,
});
const response = await client.send(command);

return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 // Replace the placeholders for agentName and accountId, and roleName with a
 // unique name for the new agent,
 // the id of your AWS account, and the name of an existing execution role that
 // the agent can use inside your account.
 // For foundationModel, specify the desired model. Ensure to remove the
 // brackets '[]' before adding your data.

 // A string (max 100 chars) that can include letters, numbers, dashes '-', and
 // underscores '_'.
 const agentName = "[your-bedrock-agent-name]";

 // Your AWS account id.
 const accountId = "[123456789012]";

 // The name of the agent's execution role. It must be prefixed by
 // `AmazonBedrockExecutionRoleForAgents_`.
 const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

 // The ARN for the agent's execution role.
 // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
 const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

 // Specify the model for the agent. Change if a different model is preferred.
 const foundationModel = "anthropic.claude-v2";

 // Check for unresolved placeholders in agentName and roleArn.
 checkForPlaceholders([agentName, roleArn]);

 console.log(`Creating a new agent...`);

 const agent = await createAgent(agentName, foundationModel, roleArn);
 console.log(agent);
}
```

```
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [CreateAgent](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

## 创建 代理

```
def create_agent(self, agent_name, foundation_model, role_arn, instruction):
 """
 Creates an agent that orchestrates interactions between foundation
 models,
 data sources, software applications, user conversations, and APIs to
 carry
 out tasks to help customers.

 :param agent_name: A name for the agent.
 :param foundation_model: The foundation model to be used for
 orchestration by the agent.
 :param role_arn: The ARN of the IAM role with permissions needed by the
 agent.
 :param instruction: Instructions that tell the agent what it should do
 and how it should
 interact with users.
 :return: The response from Agents for Bedrock if successful, otherwise
 raises an exception.
 """
 try:
 response = self.client.create_agent(
 agentName=agent_name,
 foundationModel=foundation_model,
 agentResourceRoleArn=role_arn,
 instruction=instruction,
```

```
)
 except ClientError as e:
 logger.error(f"Error: Couldn't create agent. Here's why: {e}")
 raise
 else:
 return response["agent"]
```

- 有关 API 的详细信息，请参阅适用[CreateAgent](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包创建 Amazon Bedrock 代理操作组 AWS

以下代码示例演示了如何创建 Amazon Bedrock 代理操作组。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

### Python

#### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建代理操作组。

```
def create_agent_action_group(
 self, name, description, agent_id, agent_version, function_arn,
 api_schema
):
 """
```

```

 Creates an action group for an agent. An action group defines a set of
 actions that an
 agent should carry out for the customer.

 :param name: The name to give the action group.
 :param description: The description of the action group.
 :param agent_id: The unique identifier of the agent for which to create
 the action group.
 :param agent_version: The version of the agent for which to create the
 action group.
 :param function_arn: The ARN of the Lambda function containing the
 business logic that is
 carried out upon invoking the action.
 :param api_schema: Contains the OpenAPI schema for the action group.
 :return: Details about the action group that was created.
 """
 try:
 response = self.client.create_agent_action_group(
 actionGroupName=name,
 description=description,
 agentId=agent_id,
 agentVersion=agent_version,
 actionGroupExecutor={"lambda": function_arn},
 apiSchema={"payload": api_schema},
)
 agent_action_group = response["agentActionGroup"]
 except ClientError as e:
 logger.error(f"Error: Couldn't create agent action group. Here's why:
 {e}")
 raise
 else:
 return agent_action_group

```

- 有关 API 的详细信息，请参阅适用[CreateAgentActionGroup](#)于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包创建 Amazon Bedrock 代理别名 AWS

以下代码示例演示了如何创建 Amazon Bedrock 代理别名。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

### Python

#### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建代理别名。

```
def create_agent_alias(self, name, agent_id):
 """
 Creates an alias of an agent that can be used to deploy the agent.

 :param name: The name of the alias.
 :param agent_id: The unique identifier of the agent.
 :return: Details about the alias that was created.
 """
 try:
 response = self.client.create_agent_alias(
 agentAliasName=name, agentId=agent_id
)
 agent_alias = response["agentAlias"]
 except ClientError as e:
 logger.error(f"Couldn't create agent alias. {e}")
 raise
 else:
 return agent_alias
```

- 有关 API 的详细信息，请参阅适用[CreateAgentAlias](#)于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包删除 Amazon Bedrock 代理 AWS

以下代码示例展示了如何删除 Amazon Bedrock 代理。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

### JavaScript

适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

删除代理。

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
 BedrockAgentClient,
 DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
```



```

*
* @param {string} agentId - The unique identifier of the agent to delete.
* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<import("@aws-sdk/client-bedrock-agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
and some additional metadata.
*/
export const deleteAgent = (agentId, region = "us-east-1") => {
 const client = new BedrockAgentClient({ region });
 const command = new DeleteAgentCommand({ agentId });
 return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 // Replace the placeholders for agentId with an existing agent's id.
 // Ensure to remove the brackets (`[]`) before adding your data.

 // The agentId must be an alphanumeric string with exactly 10 characters.
 const agentId = "[ABC123DE45]";

 // Check for unresolved placeholders in agentId.
 checkForPlaceholders([agentId]);

 console.log(`Deleting agent with ID ${agentId}...`);

 const response = await deleteAgent(agentId);
 console.log(response);
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [DeleteAgent](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

## 删除代理。

```
def delete_agent(self, agent_id):
 """
 Deletes an Amazon Bedrock agent.

 :param agent_id: The unique identifier of the agent to delete.
 :return: The response from Agents for Bedrock if successful, otherwise
 raises an exception.
 """

 try:
 response = self.client.delete_agent(
 agentId=agent_id, skipResourceInUseCheck=False
)
 except ClientError as e:
 logger.error(f"Couldn't delete agent. {e}")
 raise
 else:
 return response
```

- 有关 API 的详细信息，请参阅适用[DeleteAgent](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包删除 Amazon Bedrock 代理别名 AWS

以下代码示例演示了如何删除 Amazon Bedrock 代理别名。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

删除代理别名。

```
def delete_agent_alias(self, agent_id, agent_alias_id):
 """
 Deletes an alias of an Amazon Bedrock agent.

 :param agent_id: The unique identifier of the agent that the alias
 belongs to.
 :param agent_alias_id: The unique identifier of the alias to delete.
 :return: The response from Agents for Bedrock if successful, otherwise
 raises an exception.
 """

 try:
 response = self.client.delete_agent_alias(
 agentId=agent_id, agentAliasId=agent_alias_id
)
 except ClientError as e:
 logger.error(f"Couldn't delete agent alias. {e}")
 raise
 else:
 return response
```

- 有关 API 的详细信息，请参阅适用 [DeleteAgentAlias](#) 于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包获取有关亚马逊 Bedrock 代理的信息 AWS

以下代码示例展示了如何获取有关 Amazon Bedrock 代理的信息。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

### JavaScript

适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取代理。

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
 BedrockAgentClient,
 GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
 containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
```

```
const client = new BedrockAgentClient({ region });

const command = new GetAgentCommand({ agentId });
const response = await client.send(command);
return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 // Replace the placeholders for agentId with an existing agent's id.
 // Ensure to remove the brackets '[]' before adding your data.

 // The agentId must be an alphanumeric string with exactly 10 characters.
 const agentId = "[ABC123DE45]";

 // Check for unresolved placeholders in agentId.
 checkForPlaceholders([agentId]);

 console.log(`Retrieving agent with ID ${agentId}...`);

 const agent = await getAgent(agentId);
 console.log(agent);
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [GetAgent](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

获取代理。

```
def get_agent(self, agent_id, log_error=True):
 """
 Gets information about an agent.
```

```
 :param agent_id: The unique identifier of the agent.
 :param log_error: Whether to log any errors that occur when getting the
agent.
 If True, errors will be logged to the logger. If False,
errors
 will still be raised, but not logged.
 :return: The information about the requested agent.
 """

 try:
 response = self.client.get_agent(agentId=agent_id)
 agent = response["agent"]
 except ClientError as e:
 if log_error:
 logger.error(f"Couldn't get agent {agent_id}. {e}")
 raise
 else:
 return agent
```

- 有关 API 的详细信息，请参阅适用[GetAgent](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 列出使用软件开发工具包的 Amazon Bedrock 代理的 AWS 操作组

以下代码示例展示了如何列出 Amazon Bedrock 代理的操作组。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出代理的操作组。

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

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
 BedrockAgentClient,
 ListAgentActionGroupsCommand,
 paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator
 * function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of`
 * loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
 agentId,
 agentVersion,
 region = "us-east-1",
) => {
```

```
const client = new BedrockAgentClient({ region });

// Create a paginator configuration
const paginatorConfig = {
 client,
 pageSize: 10, // optional, added for demonstration purposes
};

const params = { agentId, agentVersion };

const pages = paginateListAgentActionGroups(paginatorConfig, params);

// Paginate until there are no more results
const actionGroupSummaries = [];
for await (const page of pages) {
 actionGroupSummaries.push(...page.actionGroupSummaries);
}

return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 * ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the
 * client and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
 agentId,
 agentVersion,
 region = "us-east-1",
) => {
 const client = new BedrockAgentClient({ region });

 let nextToken;
```



```
const actionGroupSummaries = [];
do {
 const command = new ListAgentActionGroupsCommand({
 agentId,
 agentVersion,
 nextToken,
 maxResults: 10, // optional, added for demonstration purposes
 });

 /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}}
 */
 const response = await client.send(command);

 for (const actionGroup of response.actionGroupSummaries || []) {
 actionGroupSummaries.push(actionGroup);
 }

 nextToken = response.nextToken;
} while (nextToken);

return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 // Replace the placeholders for agentId and agentVersion with an existing
 agent's id and version.
 // Ensure to remove the brackets '[]' before adding your data.

 // The agentId must be an alphanumeric string with exactly 10 characters.
 const agentId = "[ABC123DE45]";

 // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123'
 or 'DRAFT').
 const agentVersion = "[DRAFT]";

 // Check for unresolved placeholders in agentId and agentVersion.
 checkForPlaceholders([agentId, agentVersion]);

 console.log("=".repeat(68));
 console.log(
 "Listing agent action groups using ListAgentActionGroupsCommand:",
);
};
```

```
for (const actionGroup of await listAgentActionGroupsWithCommandObject(
 agentId,
 agentVersion,
)) {
 console.log(actionGroup);
}

console.log("=".repeat(68));
console.log(
 "Listing agent action groups using the paginateListAgents function:",
);
for (const actionGroup of await listAgentActionGroupsWithPaginator(
 agentId,
 agentVersion,
)) {
 console.log(actionGroup);
}
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [ListAgentActionGroups](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出代理的操作组。

```
def list_agent_action_groups(self, agent_id, agent_version):
 """
 List the action groups for a version of an Amazon Bedrock Agent.

 :param agent_id: The unique identifier of the agent.
 :param agent_version: The version of the agent.
```

```
 :return: The list of action group summaries for the version of the agent.
 """

 try:
 action_groups = []

 paginator = self.client.get_paginator("list_agent_action_groups")
 for page in paginator.paginate(
 agentId=agent_id,
 agentVersion=agent_version,
 PaginationConfig={"PageSize": 10},
):
 action_groups.extend(page["actionGroupSummaries"])

 except ClientError as e:
 logger.error(f"Couldn't list action groups. {e}")
 raise
 else:
 return action_groups
```

- 有关 API 的详细信息，请参阅适用[ListAgentActionGroups](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包列出属于账户的 Amazon Bedrock 代理 AWS

以下代码示例展示了如何列出属于某个账户的 Amazon Bedrock 代理。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

## JavaScript

### 适用于 JavaScript (v3) 的软件开发工具包

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出属于某个账户的代理。

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

import { fileURLToPath } from "url";

import {
 BedrockAgentClient,
 ListAgentsCommand,
 paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 * function.
 *
 * * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * * a straightforward way to handle paginated results inside a `for await...of`
 * loop.
 *
 * * @param {string} [region='us-east-1'] - The AWS region in use.
 * * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
 const client = new BedrockAgentClient({ region });

 const paginatorConfig = {
 client,
 pageSize: 10, // optional, added for demonstration purposes
 };
```

```
const pages = paginateListAgents(paginatorConfig, {});

// Paginate until there are no more results
const agentSummaries = [];
for await (const page of pages) {
 agentSummaries.push(...page.agentSummaries);
}

return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 * ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the
 * client and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
 const client = new BedrockAgentClient({ region });

 let nextToken;
 const agentSummaries = [];
 do {
 const command = new ListAgentsCommand({
 nextToken,
 maxResults: 10, // optional, added for demonstration purposes
 });

 /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
 const paginatedResponse = await client.send(command);

 agentSummaries.push(...(paginatedResponse.agentSummaries || []));

 nextToken = paginatedResponse.nextToken;
 } while (nextToken);

 return agentSummaries;
};
```

```
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 console.log("=".repeat(68));
 console.log("Listing agents using ListAgentsCommand:");
 for (const agent of await listAgentsWithCommandObject()) {
 console.log(agent);
 }

 console.log("=".repeat(68));
 console.log("Listing agents using the paginateListAgents function:");
 for (const agent of await listAgentsWithPaginator()) {
 console.log(agent);
 }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [ListAgents](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出属于某个账户的代理。

```
def list_agents(self):
 """
 List the available Amazon Bedrock Agents.

 :return: The list of available bedrock agents.
 """

 try:
 all_agents = []
```

```
paginator = self.client.get_paginator("list_agents")
for page in paginator.paginate(PaginationConfig={"PageSize": 10}):
 all_agents.extend(page["agentSummaries"])

except ClientError as e:
 logger.error(f"Couldn't list agents. {e}")
 raise
else:
 return all_agents
```

- 有关 API 的详细信息，请参阅适用[ListAgents](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 列出与使用软件开发工具包的 Amazon Bedrock 代理相关的知识库 AWS

以下代码示例演示了如何列出与 Amazon Bedrock 代理关联的知识库。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)

### Python

#### SDK for Python (Boto3)

#### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

列出与代理关联的知识库。

```
def list_agent_knowledge_bases(self, agent_id, agent_version):
 """
```

```
List the knowledge bases associated with a version of an Amazon Bedrock Agent.

:param agent_id: The unique identifier of the agent.
:param agent_version: The version of the agent.
:return: The list of knowledge base summaries for the version of the agent.
"""

try:
 knowledge_bases = []

 paginator = self.client.get_paginator("list_agent_knowledge_bases")
 for page in paginator.paginate(
 agentId=agent_id,
 agentVersion=agent_version,
 PaginationConfig={"PageSize": 10},
):
 knowledge_bases.extend(page["agentKnowledgeBaseSummaries"])

except ClientError as e:
 logger.error(f"Couldn't list knowledge bases. {e}")
 raise
else:
 return knowledge_bases
```

- 有关 API 的详细信息，请参阅适用[ListAgentKnowledgeBases](#)于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用软件开发工具包准备 Amazon Bedrock 代理 AWS

以下代码示例演示了如何为内部测试准备 Amazon Bedrock 代理。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- [创建和调用代理](#)



## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

为内部测试准备代理。

```
def prepare_agent(self, agent_id):
 """
 Creates a DRAFT version of the agent that can be used for internal
 testing.

 :param agent_id: The unique identifier of the agent to prepare.
 :return: The response from Agents for Bedrock if successful, otherwise
 raises an exception.
 """
 try:
 prepared_agent_details = self.client.prepare_agent(agentId=agent_id)
 except ClientError as e:
 logger.error(f"Couldn't prepare agent. {e}")
 raise
 else:
 return prepared_agent_details
```

- 有关 API 的详细信息，请参阅适用 [PrepareAgent](#) 于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 AWS SDK 的 Amazon Bedrock 代理的场景

以下代码示例向您展示了如何使用 AWS 软件开发工具包在 Amazon Bedrock 代理中实现常见场景。这些场景向您展示了如何通过调用多个函数来完成特定任务。每个场景都包含一个指向的链接 [GitHub](#)，您可以在其中找到有关如何设置和运行代码的说明。

## 示例

- [演 end-to-end 示如何使用软件开发工具包创建和调用 Amazon Bedrock 代理的 AWS 示例](#)
- [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 演 end-to-end 示如何使用软件开发工具包创建和调用 Amazon Bedrock 代理的 AWS 示例

以下代码示例展示了如何：

- 为代理创建执行角色。
- 创建代理并部署草稿版本。
- 创建用于实施代理功能的 Lambda 函数。
- 创建将代理连接到 Lambda 函数的操作组。
- 部署完全配置的代理。
- 使用用户提供的提示调用代理。
- 删除所有已创建的资源。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

创建并调用代理。

```
REGION = "us-east-1"
ROLE_POLICY_NAME = "agent_permissions"

class BedrockAgentScenarioWrapper:
 """Runs a scenario that shows how to get started using Agents for Amazon
 Bedrock."""
```

```
def __init__(
 self, bedrock_agent_client, runtime_client, lambda_client, iam_resource,
 postfix
):
 self.iam_resource = iam_resource
 self.lambda_client = lambda_client
 self.bedrock_agent_runtime_client = runtime_client
 self.postfix = postfix

 self.bedrock_wrapper = BedrockAgentWrapper(bedrock_agent_client)

 self.agent = None
 self.agent_alias = None
 self.agent_role = None
 self.prepared_agent_details = None
 self.lambda_role = None
 self.lambda_function = None

def run_scenario(self):
 print("=" * 88)
 print("Welcome to the Amazon Bedrock Agents demo.")
 print("=" * 88)

 # Query input from user
 print("Let's start with creating an agent:")
 print("-" * 40)
 name, foundation_model = self._request_name_and_model_from_user()
 print("-" * 40)

 # Create an execution role for the agent
 self.agent_role = self._create_agent_role(foundation_model)

 # Create the agent
 self.agent = self._create_agent(name, foundation_model)

 # Prepare a DRAFT version of the agent
 self.prepared_agent_details = self._prepare_agent()

 # Create the agent's Lambda function
 self.lambda_function = self._create_lambda_function()

 # Configure permissions for the agent to invoke the Lambda function
 self._allow_agent_to_invoke_function()
 self._let_function_accept_invocations_from_agent()
```

```
Create an action group to connect the agent with the Lambda function
self._create_agent_action_group()

If the agent has been modified or any components have been added,
prepare the agent again
components = [self._get_agent()]
components += self._get_agent_action_groups()
components += self._get_agent_knowledge_bases()

latest_update = max(component["updatedAt"] for component in components)
if latest_update > self.prepared_agent_details["preparedAt"]:
 self.prepared_agent_details = self._prepare_agent()

Create an agent alias
self.agent_alias = self._create_agent_alias()

Test the agent
self._chat_with_agent(self.agent_alias)

print("=" * 88)
print("Thanks for running the demo!\n")

if q.ask("Do you want to delete the created resources? [y/N] ",
q.is_yesno):
 self._delete_resources()
 print("=" * 88)
 print(
 "All demo resources have been deleted. Thanks again for running
the demo!"
)
else:
 self._list_resources()
 print("=" * 88)
 print("Thanks again for running the demo!")

def _request_name_and_model_from_user(self):
 existing_agent_names = [
 agent["agentName"] for agent in self.bedrock_wrapper.list_agents()
]

 while True:
 name = q.ask("Enter an agent name: ", self.is_valid_agent_name)
 if name.lower() not in [n.lower() for n in existing_agent_names]:
```

```
 break
 print(
 f"Agent {name} conflicts with an existing agent. Please use a
different name."
)

 models = ["anthropic.claude-instant-v1", "anthropic.claude-v2"]
 model_id = models[
 q.choose("Which foundation model would you like to use? ", models)
]

 return name, model_id

def _create_agent_role(self, model_id):
 role_name = f"AmazonBedrockExecutionRoleForAgents_{self.postfix}"
 model_arn = f"arn:aws:bedrock:{REGION}::foundation-model/{model_id}*"

 print("Creating an an execution role for the agent...")

 try:
 role = self.iam_resource.create_role(
 RoleName=role_name,
 AssumeRolePolicyDocument=json.dumps(
 {
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {"Service":
"bedrock.amazonaws.com"},
 "Action": "sts:AssumeRole",
 }
],
 }
),
)

 role.Policy(ROLE_POLICY_NAME).put(
 PolicyDocument=json.dumps(
 {
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
```

```

 "Action": "bedrock:InvokeModel",
 "Resource": model_arn,
 }
],
}
)
)
except ClientError as e:
 logger.error(f"Couldn't create role {role_name}. Here's why: {e}")
 raise

return role

def _create_agent(self, name, model_id):
 print("Creating the agent...")

 instruction = """
 You are a friendly chat bot. You have access to a function called
that returns
 information about the current date and time. When responding with
date or time,
 please make sure to add the timezone UTC.
 """

 agent = self.bedrock_wrapper.create_agent(
 agent_name=name,
 foundation_model=model_id,
 instruction=instruction,
 role_arn=self.agent_role.arn,
)
 self._wait_for_agent_status(agent["agentId"], "NOT_PREPARED")

 return agent

def _prepare_agent(self):
 print("Preparing the agent...")

 agent_id = self.agent["agentId"]
 prepared_agent_details = self.bedrock_wrapper.prepare_agent(agent_id)
 self._wait_for_agent_status(agent_id, "PREPARED")

 return prepared_agent_details

def _create_lambda_function(self):
 print("Creating the Lambda function...")

```

```
function_name = f"AmazonBedrockExampleFunction_{self.postfix}"

self.lambda_role = self._create_lambda_role()

try:
 deployment_package = self._create_deployment_package(function_name)

 lambda_function = self.lambda_client.create_function(
 FunctionName=function_name,
 Description="Lambda function for Amazon Bedrock example",
 Runtime="python3.11",
 Role=self.lambda_role.arn,
 Handler=f"{function_name}.lambda_handler",
 Code={"ZipFile": deployment_package},
 Publish=True,
)

 waiter = self.lambda_client.get_waiter("function_active_v2")
 waiter.wait(FunctionName=function_name)

except ClientError as e:
 logger.error(
 f"Couldn't create Lambda function {function_name}. Here's why:
{e}"
)
 raise

return lambda_function

def _create_lambda_role(self):
 print("Creating an execution role for the Lambda function...")

 role_name = f"AmazonBedrockExecutionRoleForLambda_{self.postfix}"

 try:
 role = self.iam_resource.create_role(
 RoleName=role_name,
 AssumeRolePolicyDocument=json.dumps(
 {
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
```

```

 "Principal": {"Service": "lambda.amazonaws.com"},
 "Action": "sts:AssumeRole",
 }
],
}
),
)
role.attach_policy(
 PolicyArn="arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
)
print(f"Created role {role_name}")
except ClientError as e:
 logger.error(f"Couldn't create role {role_name}. Here's why: {e}")
 raise

print("Waiting for the execution role to be fully propagated...")
wait(10)

return role

def _allow_agent_to_invoke_function(self):
 policy = self.iam_resource.RolePolicy(
 self.agent_role.role_name, ROLE_POLICY_NAME
)
 doc = policy.policy_document
 doc["Statement"].append(
 {
 "Effect": "Allow",
 "Action": "lambda:InvokeFunction",
 "Resource": self.lambda_function["FunctionArn"],
 }
)

self.agent_role.Policy(ROLE_POLICY_NAME).put(PolicyDocument=json.dumps(doc))

def _let_function_accept_invocations_from_agent(self):
 try:
 self.lambda_client.add_permission(
 FunctionName=self.lambda_function["FunctionName"],
 SourceArn=self.agent["agentArn"],
 StatementId="BedrockAccess",
 Action="lambda:InvokeFunction",
 Principal="bedrock.amazonaws.com",

```



```
)
 except ClientError as e:
 logger.error(
 f"Couldn't grant Bedrock permission to invoke the Lambda
function. Here's why: {e}"
)
 raise

def _create_agent_action_group(self):
 print("Creating an action group for the agent...")

 try:
 with open("./scenario_resources/api_schema.yaml") as file:
 self.bedrock_wrapper.create_agent_action_group(
 name="current_date_and_time",
 description="Gets the current date and time.",
 agent_id=self.agent["agentId"],
 agent_version=self.prepared_agent_details["agentVersion"],
 function_arn=self.lambda_function["FunctionArn"],
 api_schema=json.dumps(yaml.safe_load(file)),
)
 except ClientError as e:
 logger.error(f"Couldn't create agent action group. Here's why: {e}")
 raise

def _get_agent(self):
 return self.bedrock_wrapper.get_agent(self.agent["agentId"])

def _get_agent_action_groups(self):
 return self.bedrock_wrapper.list_agent_action_groups(
 self.agent["agentId"], self.prepared_agent_details["agentVersion"]
)

def _get_agent_knowledge_bases(self):
 return self.bedrock_wrapper.list_agent_knowledge_bases(
 self.agent["agentId"], self.prepared_agent_details["agentVersion"]
)

def _create_agent_alias(self):
 print("Creating an agent alias...")

 agent_alias_name = "test_agent_alias"
 agent_alias = self.bedrock_wrapper.create_agent_alias(
 agent_alias_name, self.agent["agentId"]
)
```

```
)

self._wait_for_agent_status(self.agent["agentId"], "PREPARED")

return agent_alias

def _wait_for_agent_status(self, agent_id, status):
 while self.bedrock_wrapper.get_agent(agent_id)["agentStatus"] != status:
 wait(2)

def _chat_with_agent(self, agent_alias):
 print("-" * 88)
 print("The agent is ready to chat.")
 print("Try asking for the date or time. Type 'exit' to quit.")

 # Create a unique session ID for the conversation
 session_id = uuid.uuid4().hex

 while True:
 prompt = q.ask("Prompt: ", q.non_empty)

 if prompt == "exit":
 break

 response = asyncio.run(self._invoke_agent(agent_alias, prompt,
session_id))

 print(f"Agent: {response}")

 async def _invoke_agent(self, agent_alias, prompt, session_id):
 response = self.bedrock_agent_runtime_client.invoke_agent(
 agentId=self.agent["agentId"],
 agentAliasId=agent_alias["agentAliasId"],
 sessionId=session_id,
 inputText=prompt,
)

 completion = ""

 for event in response.get("completion"):
 chunk = event["chunk"]
 completion += chunk["bytes"].decode()

 return completion
```

```
def _delete_resources(self):
 if self.agent:
 agent_id = self.agent["agentId"]

 if self.agent_alias:
 agent_alias_id = self.agent_alias["agentAliasId"]
 print("Deleting agent alias...")
 self.bedrock_wrapper.delete_agent_alias(agent_id, agent_alias_id)

 print("Deleting agent...")
 agent_status = self.bedrock_wrapper.delete_agent(agent_id)
["agentStatus"]
 while agent_status == "DELETING":
 wait(5)
 try:
 agent_status = self.bedrock_wrapper.get_agent(
 agent_id, log_error=False
)["agentStatus"]
 except ClientError as err:
 if err.response["Error"]["Code"] ==
"ResourceNotFoundException":
 agent_status = "DELETED"

 if self.lambda_function:
 name = self.lambda_function["FunctionName"]
 print(f"Deleting function '{name}'...")
 self.lambda_client.delete_function(FunctionName=name)

 if self.agent_role:
 print(f"Deleting role '{self.agent_role.role_name}'...")
 self.agent_role.Policy(ROLE_POLICY_NAME).delete()
 self.agent_role.delete()

 if self.lambda_role:
 print(f"Deleting role '{self.lambda_role.role_name}'...")
 for policy in self.lambda_role.attached_policies.all():
 policy.detach_role(RoleName=self.lambda_role.role_name)
 self.lambda_role.delete()

def _list_resources(self):
 print("-" * 40)
 print(f"Here is the list of created resources in '{REGION}'.")
```

```

 print("Make sure you delete them once you're done to avoid unnecessary
costs.")
 if self.agent:
 print(f"Bedrock Agent: {self.agent['agentName']}")
 if self.lambda_function:
 print(f"Lambda function: {self.lambda_function['FunctionName']}")
 if self.agent_role:
 print(f"IAM role: {self.agent_role.role_name}")
 if self.lambda_role:
 print(f"IAM role: {self.lambda_role.role_name}")

 @staticmethod
 def is_valid_agent_name(answer):
 valid_regex = r"^[a-zA-Z0-9_-]{1,100}$"
 return (
 answer
 if answer and len(answer) <= 100 and re.match(valid_regex, answer)
 else None,
 "I need a name for the agent, please. Valid characters are a-z, A-Z,
0-9, _ (underscore) and - (hyphen).",
)

 @staticmethod
 def _create_deployment_package(function_name):
 buffer = io.BytesIO()
 with zipfile.ZipFile(buffer, "w") as zipped:
 zipped.write(
 "./scenario_resources/lambda_function.py", f"{function_name}.py"
)
 buffer.seek(0)
 return buffer.read()

if __name__ == "__main__":
 logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

 postfix = "".join(
 random.choice(string.ascii_lowercase + "0123456789") for _ in range(8)
)
 scenario = BedrockAgentScenarioWrapper(
 bedrock_agent_client=boto3.client(
 service_name="bedrock-agent", region_name=REGION
),
 runtime_client=boto3.client(

```

```
 service_name="bedrock-agent-runtime", region_name=REGION
),
 lambda_client=boto3.client(service_name="lambda", region_name=REGION),
 iam_resource=boto3.resource("iam"),
 postfix=postfix,
)
try:
 scenario.run_scenario()
except Exception as e:
 logging.exception(f"Something went wrong with the demo. Here's what:
{e}")
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。

- [CreateAgent](#)
- [CreateAgentActionGroup](#)
- [CreateAgentAlias](#)
- [DeleteAgent](#)
- [DeleteAgentAlias](#)
- [GetAgent](#)
- [ListAgentActionGroups](#)
- [ListAgentKnowledgeBases](#)
- [ListAgents](#)
- [PrepareAgent](#)

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序

以下代码示例展示了如何使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序。

## Python

### SDK for Python (Boto3)

Amazon Bedrock Serverless Pro [AWS Step Functions](#) `smpt Chaining` 场景演示了如何使用 [Amazon Bedrock](#) 和 [Amazon Bedrock](#) 代理来构建和编排复杂、无服务器且高度可扩展的生成人工智能应用程序。它包含以下工作示例：

- 为文学博客撰写对给定小说的分析。此示例说明了一个简单的、按顺序排列的提示链。
- 生成有关给定主题的短篇小说。此示例说明了 AI 如何迭代处理其先前生成的项目列表。
- 创建前往给定目的地的周末度假行程。此示例说明如何并行处理多个不同的提示。
- 向扮演电影制作人的人类用户推销电影创意。此示例说明了如何使用不同的推理参数对同一个提示进行并行处理，如何回溯到链中的上一个步骤，以及如何将人工输入作为工作流程的一部分。
- 根据用户手头的食材计划膳食。这个例子说明了提示链如何整合两个不同的人工智能对话，两个人工智能角色相互进行辩论以改善最终结果。
- 查找并总结当今最热门的 GitHub 存储库。此示例说明如何链接多个与外部 API 交互的 AI 代理。

有关完整的源代码以及设置和运行说明，请参阅上的完整项目 [GitHub](#)。

本示例中使用的服务

- Amazon Bedrock
- Amazon Bedrock 运行时系统
- Agents for Amazon Bedrock
- 亚马逊 Bedrock Runtime 的代理
- Step Functions

有关 `S AWS DK` 开发者指南和代码示例的完整列表，请参阅 [将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 AWS 软件开发工具包的 Amazon Bedrock Runtime 代理的代码示例

以下代码示例展示了如何使用带 AWS 软件开发套件 (SDK) 的 Amazon Bedrock Runtime 代理。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过同一服务中调用多个函数来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 代码示例

- [使用 AWS 软件开发工具包为 Amazon Bedrock Runtime 代理执行的操作](#)
  - [使用软件开发工具包调用 Amazon Bedrock 代理 AWS](#)
- [使用 SDK 的 Amazon Bedrock Runtime AWS 代理的场景](#)
  - [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用 AWS 软件开发工具包为 Amazon Bedrock Runtime 代理执行的操作

以下代码示例演示了如何使用 AWS 软件开发工具包为 Amazon Bedrock 运行时执行单个代理操作。这些摘录调用 Amazon Bedrock 运行时 API 的代理，是必须在上下文中运行的大型程序的代码摘录。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 [Amazon Bedrock 运行时代理 API 参考](#)。

## 示例

- [使用软件开发工具包调用 Amazon Bedrock 代理 AWS](#)

## 使用软件开发工具包调用 Amazon Bedrock 代理 AWS

以下代码示例展示了如何调用 Amazon Bedrock 代理。

## JavaScript

适用于 JavaScript (v3) 的软件开发工具包

### Note

还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

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

import {
 BedrockAgentRuntimeClient,
 InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
 const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
 // const client = new BedrockAgentRuntimeClient({
 // region: "us-east-1",
 // credentials: {
 // accessKeyId: "accessKeyId", // permission to invoke agent
 // secretAccessKey: "accessKeySecret",
 // },
 // });

 const agentId = "AJBHXXILZN";
 const agentAliasId = "AVKP1ITZAA";

 const command = new InvokeAgentCommand({
 agentId,
 agentAliasId,
 sessionId,
 inputText: prompt,
 });

 try {
 let completion = "";
 const response = await client.send(command);
 }
}
```



```

 if (response.completion === undefined) {
 throw new Error("Completion is undefined");
 }

 for await (let chunkEvent of response.completion) {
 const chunk = chunkEvent.chunk;
 console.log(chunk);
 const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
 completion += decodedResponse;
 }

 return { sessionId: sessionId, completion };
 } catch (err) {
 console.error(err);
 }
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
 const result = await invokeBedrockAgent("I need help.", "123");
 console.log(result);
}

```

- 有关 API 的详细信息，请参阅 AWS SDK for JavaScript API 参考 [InvokeAgent](#) 中的。

## Python

### SDK for Python (Boto3)

#### Note

还有更多相关信息 [GitHub](#)。在 [AWS 代码示例存储库](#) 中查找完整示例，了解如何进行设置和运行。

调用代理。

```

def invoke_agent(self, agent_id, agent_alias_id, session_id, prompt):
 """

```

```
Sends a prompt for the agent to process and respond to.

:param agent_id: The unique identifier of the agent to use.
:param agent_alias_id: The alias of the agent to use.
:param session_id: The unique identifier of the session. Use the same
value across requests
 to continue the same conversation.
:param prompt: The prompt that you want Claude to complete.
:return: Inference response from the model.
"""

try:
 response = self.agents_runtime_client.invoke_agent(
 agentId=agent_id,
 agentAliasId=agent_alias_id,
 sessionId=session_id,
 inputText=prompt,
)

 completion = ""

 for event in response.get("completion"):
 chunk = event["chunk"]
 completion = completion + chunk["bytes"].decode()

except ClientError as e:
 logger.error(f"Couldn't invoke agent. {e}")
 raise

return completion
```

- 有关 API 的详细信息，请参阅适用[InvokeAgent](#)于 Python 的 AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

## 使用 SDK 的 Amazon Bedrock Runtime AWS 代理的场景

以下代码示例向您展示了如何使用 AWS 软件开发工具包在 Amazon Bedrock Runtime 代理中实现常见场景。这些场景向您展示了如何通过调用多个函数来完成特定任务。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

### 示例

- [使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序](#)

## 使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序

以下代码示例展示了如何使用 Amazon Bedrock 和 Step Functions 构建和编排生成式人工智能应用程序。

### Python

#### SDK for Python (Boto3)

Amazon Bedrock Serverless Pro [AWS Step Functions](#) Chaining 场景演示了如何使用 Amazon Bedrock 和 Amazon Bedrock 代理来构建和编排复杂、无服务器且高度可扩展的生成式人工智能应用程序。它包含以下工作示例：

- 为文学博客撰写对给定小说的分析。此示例说明了一个简单的、按顺序排列的提示链。
- 生成有关给定主题的短篇小说。此示例说明了 AI 如何迭代处理其先前生成的项目列表。
- 创建前往给定目的地的周末度假行程。此示例说明如何并行处理多个不同的提示。
- 向扮演电影制作人的人类用户推销电影创意。此示例说明了如何使用不同的推理参数对同一个提示进行并行处理，如何回溯到链中的上一个步骤，以及如何将人工输入作为工作流程的一部分。
- 根据用户手头的食材计划膳食。这个例子说明了提示链如何整合两个不同的人工智能对话，两个人工智能角色相互进行辩论以改善最终结果。
- 查找并总结当今最热门的 GitHub 存储库。此示例说明如何链接多个与外部 API 交互的 AI 代理。

有关完整的源代码以及设置和运行说明，请参阅上的完整项目 [GitHub](#)。

#### 本示例中使用的服务

- Amazon Bedrock
- Amazon Bedrock 运行时系统

- Agents for Amazon Bedrock
- 亚马逊 Bedrock Runtime 的代理
- Step Functions

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅[将此服务与 AWS SDK 配合使用](#)。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

# Amazon Bedrock 滥用检测

AWS 致力于负责任地使用人工智能。为了帮助防止潜在的滥用行为，Amazon Bedrock 实施了自动化滥用检测机制，以识别可能违反 AWS 的 [Acceptable Use Policy \(AUP\)](#) 和服务条款的行为，包括 [Responsible AI Policy](#) 或第三方模型提供商的 AUP。

我们的滥用检测机制是完全自动化的，因此无需人工审核或者访问用户输入或模型输出。

自动化滥用检测包括：

- 对内容进行分类 — 我们使用分类器来检测用户输入和模型输出中的有害内容（例如，煽动暴力的内容）。分类器是一种处理模型输入和输出，并指定危害类型和置信度级别的算法。我们可能会对两个模型 Titan 和第三方模型的使用情况运行这些分类器。分类过程是自动化的，不涉及对用户输入或模型输出进行人工审核。
- 识别模式 — 我们使用分类器指标来识别潜在的违规行为和反复出现的行为。我们可能会编译匿名的分类器指标，并与第三方模型提供商共享。Amazon Bedrock 不存储用户输入或模型输出，也不会与第三方模型提供商共享这些信息。
- 检测和屏蔽儿童性虐待材料 (CSAM) — 您应对您（和您的最终用户）上传到 Amazon Bedrock 的内容负责，并且必须确保这些内容不含非法图片。为了帮助阻止 CSAM 的传播，Amazon Bedrock 可能会使用自动滥用检测机制（例如哈希匹配技术或分类器）来检测明显的 CSAM。如果 Amazon Bedrock 在您的图片输入中检测到明显的 CSAM，Amazon Bedrock 将阻止该请求，并且您将收到一条自动错误消息。Amazon Bedrock 还可能向国家失踪与被剥削儿童中心 (NCMEC) 或相关机构提交报告。我们认真对待 CSAM，并将继续更新我们的检测、屏蔽和报告机制。适用法律可能会要求您采取其他行动，并且您应对这些行为负责。

一旦我们的自动滥用行为检测机制发现了潜在的违规行为，我们可能会要求您提供有关您使用 Amazon Bedrock 以及遵守我们的服务条款或第三方提供商的 AUP 的信息。如果您不愿或无法遵守这些条款或政策，AWS 可以暂停您对 Amazon Bedrock 的访问权限。

如果您还有其他问题，请联系 AWS Support。有关更多信息，请参阅 [Amazon Bedrock 常见问题](#)。

# Amazon Bedrock 的配额

您的每个配额 AWS 账户 都有默认配额，以前称为限制 AWS 服务。除非另有说明，否则您的每个配额都是针对特定区域的。AWS 账户有些配额可能是可调整的。下表说明了下表中“通过 Service Quotas 可调整”列的含义：

- 如果将配额标记为“是”，则可以按照 [Service Quotas 用户指南中请求增加配额](#) 中的步骤进行调整。
- 如果将配额标记为“否”，则可以通过以下任一方式申请增加配额：
  - 要申请增加 [按需运行时配额的配额](#)，请联系您的 AWS 账户 经理。如果您没有 AWS 账户 经理，则此时无法增加配额。
  - 要申请其他配额上调，请通过 [提高限额表](#) 提交申请，以考虑增加配额。

## Note

由于需求过大，将优先考虑那些产生的流量消耗了现有配额分配的客户。如果您不符合此条件，您的请求可能会被拒绝。

有些配额因型号而异。除非另有说明，否则配额适用于模型的所有版本。

选择一个主题以了解有关其配额的更多信息。

## 主题

- [运行时限额](#)
- [批量推理配额](#)
- [知识库配额](#)
- [代理配额](#)
- [模型自定义配额](#)
- [预调配吞吐量配额](#)

## 运行时限额

延迟因型号而异，并且与以下条件成正比：

- 输入和输出令牌的数量

- 当时所有客户持续的按需请求总数。

在您进行模型推理时，以下配额适用。这些配额考虑了 [InvokeModel](#) 和 [InvokeModelWithResponseStream](#) 请求的总和。

要提高吞吐量，请购买 [Amazon Bedrock 的预配置吞吐量](#)。

#### Note

如果通过 Service Quotas 将配额标记为不可调整，您可以联系您的 AWS 账户经理申请增加配额。如果您没有 AWS 账户经理，则此时无法增加配额。由于需求过大，将优先考虑那些产生的流量消耗了现有配额分配的客户。如果您不符合此条件，您的请求可能会被拒绝。

| 模型                                    | 每分钟处理的请求数 | 每分钟处理的令牌数 | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------------------------------|-----------|-----------|---------------------------------------|
| AI21 Labs Jurassic-2 Mid              | 400       | 300,000   | 不支持                                   |
| AI21 Labs Jurassic-2 Ultra            | 100       | 200,000   | 不支持                                   |
| Amazon Titan Embeddings G1 - Text     | 2000      | 300,000   | 不支持                                   |
| Amazon Titan Image Generator G1       | 60        | 不适用       | 不支持                                   |
| Amazon Titan Multimodal Embeddings G1 | 2000      | 不适用       | 不支持                                   |
| Amazon Titan Text G1 - Express        | 400       | 300,000   | 不支持                                   |

| 模型                               | 每分钟处理的请求数 | 每分钟处理的令牌数 | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|----------------------------------|-----------|-----------|---------------------------------------|
| Amazon Titan Text G1 - Lite      | 800       | 300,000   | 不支持                                   |
| Anthropic Claude Instant         | 400       | 300,000   | 不支持                                   |
| AnthropicClaude2.x               | 100       | 200,000   | 不支持                                   |
| Anthropic Claude 3 Sonnet        | 100       | 200,000   | 不支持                                   |
| Anthropic Claude 3 Haiku         | 400       | 300,000   | 不支持                                   |
| Cohere Command                   | 400       | 300,000   | 不支持                                   |
| Cohere Command Light             | 800       | 300,000   | 不支持                                   |
| CohereEmbed ( 英语 )               | 2000      | 300,000   | 不支持                                   |
| CohereEmbed ( 多语言 )              | 2000      | 300,000   | 不支持                                   |
| MetaLlama 213B                   | 800       | 300,000   | 不支持                                   |
| MetaLlama 270B                   | 400       | 300,000   | 不支持                                   |
| Mistral AI Mistral 7B Instruct   | 800       | 300,000   | 不支持                                   |
| Mistral AI Mixtral 8X7B Instruct | 400       | 300,000   | 不支持                                   |
| Stable Diffusion XL              | 60        | 不适用       | 不支持                                   |



选择一个选项卡可查看特定于模型的推理配额。

### Amazon 泰坦 Text models

| 描述            | 值      | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------|--------|---------------------------------------|
| 文本提示长度，以字符为单位 | 42,000 | 不支持                                   |

### Amazon Titan Image Generator G1

| 描述                          | 值          | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|-----------------------------|------------|---------------------------------------|
| 文本提示长度，以字符为单位               | 1024       | 不支持                                   |
| 输入图像大小                      | 5MB        | 不支持                                   |
| 以像素为单位的输入图像高度 ( 画内/外绘 )     | 1024       | 不支持                                   |
| 输入图像宽度 ( 以像素为单位 ) ( 画内/外绘 ) | 1024       | 不支持                                   |
| 以像素为单位的输入图像高度 ( 图像变化 )      | 4,096      | 不支持                                   |
| 以像素为单位的输入图像宽度 ( 图像变化 )      | 4,096      | 不支持                                   |
| 输入图像的总像素数                   | 12,582,912 | 不支持                                   |

## Amazon Titan Embeddings G1 - Text


| 描述            | 值     | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------|-------|---------------------------------------|
| 文本输入长度，以字符为单位 | 50000 | 不支持                                   |

## Amazon Titan Multimodal Embeddings G1

| 描述                       | 值          | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|--------------------------|------------|---------------------------------------|
| 文本输入长度，以字符为单位            | 100000     | 不支持                                   |
| 以 Base64 编码的图像字符串，以字符为单位 | 25,000,000 | 不支持                                   |

## 批量推理配额

在您进行批量推理时，以下配额适用。配额取决于输入和输出数据的模式。

 Note

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

| 模式       | 最小文件大小 | 最大文件大小 | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|----------|--------|--------|---------------------------------------|
| 文本到嵌入    | 75 MB  | 500 MB | 不支持                                   |
| 文本到文本    | 20 MB  | 150 MB | 不支持                                   |
| 文本/图像到图像 | 1 MB   | 50 MB  | 不支持                                   |

## 知识库配额

以下配额适用于 Amazon Bedrock 的知识库。

### Note

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

| 描述                         | 最高    | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|----------------------------|-------|---------------------------------------|
| 每个地区每个账户的知识库               | 50    | 不可以                                   |
| 数据源文件大小 ( 源文档 )            | 50 MB | 不支持                                   |
| 数据源文件大小 ( 元数据文件 )          | 10 KB | 不支持                                   |
| 数据源区块大小 ( Titan文本 G1-嵌入 )  | 8192  | 不支持                                   |
| 数据源区块大小 ( CohereEmbed英语 )  | 512   | 不支持                                   |
| 数据源区块大小 ( CohereEmbed多语言 ) | 512   | 不支持                                   |

## 代理配额

以下配额适用于 Amazon Bedrock 的代理。

### Note

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

| 描述              | 最高    | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|-----------------|-------|---------------------------------------|
| 每个账户的代理数        | 50    | 可以                                    |
| 每个代理的别名数        | 10    | 不支持                                   |
| 代理说明中的字符数       | 1,200 | 不支持                                   |
| 每个代理的操作组数       | 5     | 不支持                                   |
| 代理中跨操作组的 API 数量 | 5     | 不支持                                   |
| Lambda 响应有效负载大小 | 25 KB | 不支持                                   |
| 每个代理的相关知识库数     | 2     | 不支持                                   |

## 模型自定义配额

以下配额适用于模型自定义。

### Note

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

| 描述           | 最高  | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|--------------|-----|---------------------------------------|
| 每个账户的预定训练作业数 | 2   | 不支持                                   |
| 每个账户的自定义模型数  | 100 | 是                                     |

要查看超参数配额，请参阅[自定义模型超参数](#)。

选择一个选项卡可查看适用于用于自定义不同基础模型的训练和验证数据集的模型特定配额。

**Note**

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

## Amazon Titan Text G1 - Express

| 描述                        | 最大值 ( 继续预训练 ) | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------------------|---------------|------------|---------------------------------------|
| 批次大小为 1 时输入和输出令牌数总和       | 4,096         | 4,096      | 不支持                                   |
| 批量大小为 2、3 或 4 时输入和输出令牌的总和 | 2,048         | 2,048      | 不支持                                   |
| 数据集中每个样本的字符配额             | 词元配额 x 6      | 词元配额 x 6   | 不支持                                   |
| 训练数据集中的记录                 | 100000        | 10000      | 支持                                    |
| 验证数据集中的记录                 | 1000          | 1000       | 是                                     |
| 训练数据集文件大小                 | 10 GB         | 1 GB       | 支持                                    |
| 验证数据集文件大小                 | 100 MB        | 100 MB     | 支持                                    |

## Amazon Titan Text G1 - Lite

| 描述                          | 最大值 ( 继续预训练 ) | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|-----------------------------|---------------|------------|---------------------------------------|
| 批量大小为 1 或 2 时输入和输出令牌的总和     | 4,096         | 4,096      | 不支持                                   |
| 批量大小为 3、4、5 或 6 时输入和输出令牌的总和 | 2,048         | 2,048      | 不支持                                   |
| 数据集中每个样本的字符配额               | 词元配额 x 6      | 词元配额 x 6   | 不支持                                   |
| 训练数据集中的记录                   | 100000        | 10000      | 支持                                    |
| 验证数据集中的记录                   | 1000          | 1000       | 是                                     |
| 训练数据集文件大小                   | 10 GB         | 1 GB       | 支持                                    |
| 验证数据集文件大小                   | 100 MB        | 100 MB     | 支持                                    |

## Amazon Titan Image Generator G1

| 描述                  | 最小值 ( 微调 ) | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------------|------------|------------|---------------------------------------|
| 训练样本中的文本提示长度，以字符为单位 | 3          | 1024       | 不支持                                   |
| 训练数据集中的记录           | 5          | 10000      | 不支持                                   |
| 输入图像大小              | 0          | 50 MB      | 不支持                                   |

| 描述                | 最小值 ( 微调 ) | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|-------------------|------------|------------|---------------------------------------|
| 输入图像高度 ( 以像素为单位 ) | 512        | 4,096      | 不支持                                   |
| 输入图像宽度 ( 以像素为单位 ) | 512        | 4,096      | 不支持                                   |
| 输入图像的总像素数         | 0          | 12,582,912 | 不支持                                   |
| 输入图像的纵横比          | 1:4        | 4:1        | 不支持                                   |

### Amazon Titan Multimodal Embeddings G1


| 描述                   | 最小值 ( 微调 ) | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|----------------------|------------|------------|---------------------------------------|
| 训练样本中的文本提示长度, 以字符为单位 | 0          | 2,560      | 不支持                                   |
| 训练数据集中的记录            | 1000       | 500,000    | 不支持                                   |
| 输入图像大小               | 0          | 5MB        | 不支持                                   |
| 输入图像高度 ( 以像素为单位 )    | 128        | 4096       | 不支持                                   |
| 输入图像宽度 ( 以像素为单位 )    | 128        | 4096       | 不支持                                   |
| 输入图像的总像素数            | 0          | 12,528,912 | 不支持                                   |
| 输入图像的纵横比             | 1:4        | 4:1        | 不支持                                   |

## Cohere 命令 &amp; Meta Llama 2

| 描述            | 最大值 ( 微调 ) | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|---------------|------------|---------------------------------------|
| 输入令牌          | 4,096      | 不支持                                   |
| 输出令牌          | 2,048      | 不支持                                   |
| 数据集中每个样本的字符配额 | 词元配额 x 6   | 不支持                                   |
| 训练数据集中的记录     | 10000      | 支持                                    |
| 验证数据集中的记录     | 1000       | 是                                     |

## 预调配吞吐量配额

以下配额适用于预调配吞吐量。

 Note

如果通过 Service Quotas 将配额标记为不可调整，则您可以通过[提高限额表](#)提交申请，以考虑增加限额。

| 描述                      | 默认 | 可通过 Service Quotas 进行调整 ( 参见表格上方的备注 ) |
|-------------------------|----|---------------------------------------|
| 可以在无承诺预置吞吐量之间分配的模型单元    | 2  | 不支持                                   |
| 对可按承诺在预配置吞吐量之间分配的单位进行建模 | 0  | 不支持                                   |



# API 参考

有关 API 参考，请点击[此处](#)。

# 《Amazon Bedrock 用户指南》的文档历史记录

- 最新文档更新：2024 年 3 月 29 日

下表介绍了每个 Amazon Bedrock 版本中的重大更改。要获得本文档的更新通知，您可以订阅 RSS 源。

| 变更                             | 说明                                                                                                              | 日期         |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------|------------|
| <a href="#">新特征</a>            | 现在，您无需承诺即可为基本型号购买预配置吞吐量。                                                                                        | 2024年3月29日 |
| <a href="#">对预配置吞吐量的更多模型支持</a> | 现在，您可以购买AnthropicClaude 3 Sonnet、AnthropicClaude 3 Haiku、CohereEmbed 英语和CohereEmbed多语言的预配置吞吐量。                  | 2024年3月29日 |
| <a href="#">新特征</a>            | 现在，您可以在 Amazon OpenSearch Serverless 中创建网络访问策略，以允许您的 Amazon Bedrock 知识库访问配置有 VPC 终端节点的私有 OpenSearch 无服务器矢量搜索集合。 | 2024年3月28日 |
| <a href="#">新特征</a>            | 现在，您可以在 Amazon Bedrock 的知识库中包含源文档的元数据，并在 <a href="#">知识库查询期间筛选元数据</a> 。                                         | 2024年3月27日 |
| <a href="#">新特征</a>            | 现在，当你查询知识库并生成响应时，你可以使用提示模板来自定义发送给模型的提示。                                                                         | 2024年3月26日 |

|                                               |                                                                                   |            |
|-----------------------------------------------|-----------------------------------------------------------------------------------|------------|
| <a href="#">更多模型支持在 Amazon Bedrock 中查询知识库</a> | 现在，您可以使用AnthropicClaude 3 Sonnet生成知识库响应。                                          | 2024年3月25日 |
| <a href="#">新模型</a>                           | 您现在可以在 Amazon Anthropic Claude 3 Haiku Bedrock 上使用。                               | 2024年3月13日 |
| <a href="#">新模型</a>                           | 您现在可以在 Amazon Anthropic Claude 3 Sonnet Bedrock 上使用。                              | 2024年3月4日  |
| <a href="#">新模型</a>                           | 现在，您可以在 Amazon Bedrock 上使用Mistral AI模型。                                           | 2024年3月1日  |
| <a href="#">新特征</a>                           | 现在，您可以在知识库中包含可筛选文本字段的 Amazon OpenSearch Serverless 矢量存储自定义搜索策略。                   | 2024年2月28日 |
| <a href="#">新特征</a>                           | 现在，您可以从 Amazon Bedrock Titan 图像生成器中检测带有水印的图像。                                     | 2024年2月14日 |
| <a href="#">更新了 AWS PrivateLink 支持</a>        | 现在，您可以使用 AWS PrivateLink 为 <a href="#">Amazon Bedrock 代理构建</a> 时服务创建接口 VPC 终端节点。  | 2024年2月9日  |
| <a href="#">IAM 角色更新</a>                      | 现在，您可以在知识库中使用相同的服务角色，也可以使用不带预定义前缀的角色。                                             | 2024年2月9日  |
| <a href="#">处于旧版状态的模型</a>                     | Stable Diffusion XLv0.8 现在处于旧版状态。在 2024 年 4 月 30 日之前迁移到 Stable Diffusion XL v1.x。 | 2024年2月2日  |

|                                                                      |                                                                                                              |                  |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">添加了代码示例章节</a>                                            | Amazon Bedrock 指南现在包括各种亚马逊 Bedrock 操作和场景的代码示例。                                                               | 2024 年 1 月 25 日  |
| <a href="#">新特征</a>                                                  | 现在，当你选择在控制台中快速创建 Amazon OpenSearch Serverless 矢量存储时，Amazon Bedrock 的知识库可以让你在生产账户和非生产账户之间进行选择。                | 2024 年 1 月 24 日  |
| <a href="#">新特征</a>                                                  | Amazon Bedrock 代理现在允许您在使用控制台中的测试窗口时实时查看跟踪。                                                                   | 2024 年 1 月 18 日  |
| <a href="#">更多模型支持在 Amazon Bedrock 的知识库中嵌入数据源</a>                    | Amazon Bedrock 的知识库现在支持使用 CohereEmbed 英语和 CohereEmbed 多语言嵌入您的数据源。                                            | 2024 年 1 月 17 日  |
| <a href="#">为 Amazon Bedrock 代理提供更多模型支持，并在 Amazon Bedrock 中查询知识库</a> | Amazon Bedrock 代理和亚马逊 Bedrock 响应生成的知识库现在支持 Anthropic Claude 2.1。                                             | 2023 年 12 月 27 日 |
| <a href="#">区域扩展</a>                                                 | Amazon Bedrock 现已在 AWS GovCloud (美国西部) (us-gov-west 1) 上市。有关端点的信息，请参阅 <a href="#">Amazon Bedrock 端点和配额</a> 。 | 2023 年 12 月 21 日 |
| <a href="#">新增对向量存储的支持</a>                                           | 您现在可以在 Amazon Aurora 数据库集群中创建知识库。有关更多信息，请参阅在 <a href="#">Amazon Aurora 中创建向量存储</a> 。                         | 2023 年 12 月 21 日 |

|                          |                                                                                                                                  |                  |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">全新托管策略</a>   | Amazon Bedrock 新增了 AmazonBedrockFullAccess 和 AmazonBedrockReadOnly ，前者用于向用户授予创建、读取、更新和删除资源的权限，后者用于向用户授予对所有操作的只读权限。               | 2023 年 12 月 12 日 |
| <a href="#">新特征</a>      | Amazon Bedrock 现在支持使用自动指标或人工创建模型评估作业。                                                                                            | 2023 年 11 月 29 日 |
| <a href="#">新特征</a>      | 您现在可以监控和自定义 <a href="#">模型版本</a> 。                                                                                               | 2023 年 11 月 29 日 |
| <a href="#">新Titan车型</a> | 来自的新车型Titan包括亚马逊Titan Image Generator G1 和亚马逊Titan Multimodal Embeddings G1。有关更多信息，请参阅 <a href="#">Titan模型</a> 。                 | 2023 年 11 月 29 日 |
| <a href="#">新特征</a>      | 通过持续预训练，您可以向模型传授新的领域知识。有关更多信息，请参阅 <a href="#">Custom Models</a> 。                                                                | 2023 年 11 月 28 日 |
| <a href="#">新特征</a>      | 现在，您可以通过“ <a href="#">检索</a> ”和 <a href="#">RetrieveAndGenerateAPI</a> 查询知识库。有关更多信息，请参阅 <a href="#">Query a knowledge base</a> 。 | 2023 年 11 月 28 日 |
| <a href="#">常规版本</a>     | Amazon Bedrock 服务知识库的正式发布。有关更多信息，请参阅 <a href="#">Amazon Bedrock 知识库</a> 。                                                        | 2023 年 11 月 28 日 |

|                         |                                                                                                      |                  |
|-------------------------|------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">常规版本</a>    | Amazon Bedrock 代理服务的常规版本。有关更多信息，请参阅 <a href="#">Agents for Amazon Bedrock</a> 。                      | 2023 年 11 月 28 日 |
| <a href="#">自定义更多模型</a> | 现在，您可以从Cohere和自定义模型Meta。有关更多信息，请参阅 <a href="#">Custom Models</a> 。                                   | 2023 年 11 月 28 日 |
| <a href="#">新模型版本</a>   | 更新了文档以涵盖新产品Meta和Cohere型号。有关更多信息，请参阅 <a href="#">Amazon Bedrock</a> 。                                 | 2023 年 11 月 13 日 |
| <a href="#">文档本地化</a>   | Amazon Bedrock 文档现提供 <a href="#">日语</a> 和 <a href="#">德语</a> 版本。                                     | 2023 年 10 月 20 日 |
| <a href="#">区域扩展</a>    | Amazon Bedrock 现已在欧洲地区 ( 法兰克福 ) (eu-central-1) 发售。有关端点的信息，请参阅 <a href="#">Amazon Bedrock 端点和配额</a> 。 | 2023 年 10 月 19 日 |
| <a href="#">区域扩展</a>    | Amazon Bedrock 现已在亚太地区 ( 东京 ) (ap-northeast-1) 发售。有关端点的信息，请参阅 <a href="#">Amazon Bedrock 端点和配额</a> 。 | 2023 年 10 月 3 日  |
| <a href="#">封闭式常规版本</a> | Amazon Bedrock 服务的封闭式常规版本。有关更多信息，请参阅 <a href="#">Amazon Bedrock</a> 。                                | 2023 年 9 月 28 日  |

# AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的[AWS 词汇表](#)。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。