aws

ユーザーガイド

Amazon Bedrock

Amazon Bedrock: ユーザーガイド

Table of Contents

Amazon Bedrock とは

Amazon Bedrock は、主要な AI スタートアップや Amazon が提供する高パフォーマンスな基盤モデル (FM) を、統合 API を通じて利用できるようにするフルマネージド型サービスです。さまざまな基盤モデルから選択して、ユースケースに最適なモデルを見つけることができます。Amazon Bedrock には、セキュリティ、プライバシー、責任のある AI を備えた生成 AI アプリケーションを構築するためのさまざまな機能も用意されています。Amazon Bedrock を使用すると、ユースケースに適した最善の基盤モデルを簡単に試して評価したり、ファインチューニングや検索拡張生成 (RAG) などの手法を使用して基盤モデルをデータでプライベートにカスタマイズしたり、エンタープライズシステムやデータソースを使用してタスクを実行するエージェントを構築したりできます。

Amazon Bedrock のサーバーレスエクスペリエンスを使用すると、インフラストラクチャを管理することなく、独自のデータを使用して基盤モデルをすばやくプライベートにカスタマイズし、 AWS ツールを使用して簡単かつ安全に統合してアプリケーションにデプロイできます。

トピック

Amazon Bedrock の機能

Amazon Bedrock 基盤モデルを活用して、次の機能を確認してください。リージョン別の機能制限については、「」を参照してくださいAWS 地域別のモデルサポート。

- プロンプトと設定を試す – さまざまな設定や基盤モデルを使用してプロンプトを送信し、モデル推論を実行することで、レスポンスを生成できます。API またはコンソールのテキスト、イメージ、チャットのプレイグラウンドを使用して、グラフィカルインターフェイスで試すことができます。準備ができたら、`InvokeModel` API にリクエストを送信するようにアプリケーションを設定します。
- 独自のデータソースの情報でレスポンスの生成を強化する – 問い合わせ対象となるデータソースをアップロードしてナレッジベースを作成し、基盤モデルのレスポンスの生成を強化できます。

- 顧客の支援方法を推論するアプリケーションを作成する – 基盤モデルを使用し、API 呼び出しを行い、(オプションで) ナレッジベースに問い合わせ、顧客に代わってタスクを推論して実行するエージェントを構築できます。
- トレーニングデータを使用してモデルを特定のタスクや領域に適応させる – ファインチューニングや継続的な事前トレーニングのためのトレーニングデータを提供することで Amazon Bedrock 基盤モデルをカスタマイズし、モデルのパラメータを調整したり、特定のタスクや領域でのパフォーマンスを向上させたりできます。
- FM ベースのアプリケーションの効率と出力を向上させる – 基盤モデルのプロビジョンドスループットを購入すると、モデルの推論を割引料金でより効率的に実行できます。
- ユースケースに最適なモデルを判断する – 組み込みまたはカスタムのプロンプトデータセットを使用してさまざまなモデルの出力を評価し、アプリケーションに最適なモデルを判断できます。

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。

- 不適切または望ましくないコンテンツの防止 – Amazon Bedrock のガードレール機能を使用して、生成 AI アプリケーションに保護手段を実装できます。

> **Note**
>
> Amazon Bedrock のガードレール機能は限定プレビューリリースに付属しています。アクセスをリクエストするには、 AWS アカウント マネージャーにお問い合わせください。

Amazon Bedrock の料金

にサインアップすると AWS、Amazon Bedrock を含む AWSのすべてのサービスに AWS アカウントが自動的にサインアップされます。ただし、料金が発生するのは実際に使用したサービスの分だけです。

請求を表示するには、AWS Billing and Cost Management コンソールで請求およびコスト管理ダッシュボードに移動します。 AWS アカウント 請求の詳細については、「 AWS Billing ユーザーガイド」を参照してください。 AWS 請求および についてご質問がある場合は AWS アカウント、 AWS サポートにお問い合わせください。

Amazon Bedrock では、どのサードパーティーの基盤モデルで推論を実行する場合でも料金がかかります。料金は、入力トークンと出力トークンの量、およびモデル用のプロビジョンドスループットの購入有無によって決まります。詳細については、Amazon Bedrock コンソールの [モデルプロバイダー] ページを参照してください。各モデルの料金は、モデルバージョンの後に記載されています。プロビジョンドスループットの購入の詳細については、「Amazon Bedrock のプロビジョニングされたスループット」を参照してください。

詳細については、「Amazon Bedrock の料金体系」ページを参照してください。

サポートされている AWS リージョン

Amazon Bedrock がサポートするリージョンのサービスエンドポイントについては、「Amazon Bedrock エンドポイントとクォータ」を参照してください。

各リージョンがサポートする基盤モデルについては、「」を参照してくださいAWS 地域別のモデルサポート。

> **Note**
>
> 米国東部 (バージニア北部) および米国西部 (オレゴン) ではすべての機能を使用できます。

リージョンによって制限される機能については、次の表を参照してください。

リージョン	モデル評価	ナレッジベース	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
米国東部 (バージニア北部)	はい	はい	はい	はい	はい	はい
米国西部 (オレゴン)	はい	はい	はい	はい	はい	はい
アジアパシフィッ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ

リージョン	モデル評価	ナレッジベース	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
ク (シンガポール)						
アジアパシフィック (東京)	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
欧州 (フランクフルト)	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
AWS GovCloud (米国西部)	いいえ	いいえ	いいえ	はい	いいえ	はい (ファインチューニングされたモデルのみ、コミットメント期間なし)

主な定義

この章では、Amazon Bedrock が提供する内容とその仕組みを理解するのに役立つ概念の定義について説明します。を初めて使用する場合は、まず基本概念をお読みください。Amazon Bedrock の基本を理解したら、Amazon Bedrock が提供する高度な概念と機能について調べることをお勧めします。

基本概念

次のリストでは、生成 AI と Amazon Bedrock の基本機能の基本概念を紹介します。

- 基盤モデル (FM) — 多数のパラメータを持ち、大量の多様なデータに基づいてトレーニングされた AI モデル。基盤モデルは、さまざまなユースケースに対してさまざまなレスポンスを生成できます。基盤モデルはテキストまたは画像を生成し、入力を埋め込みに変換することもできま

す。Amazon Bedrock 基盤モデルを使用する前に、 へのアクセスをリクエストする必要があります。基盤モデルの詳細については、「」を参照してくださいAmazon Bedrock でサポートされている基盤モデル。

- ベースモデル — プロバイダーによってパッケージ化され、すぐに使用できる基盤モデル。Amazon Bedrock は、主要なプロバイダーが提供する業界でも先行するさまざまな基盤モデルを提供しています。詳細については、「Amazon Bedrock でサポートされている基盤モデル」を参照してください。
- モデル推論 — 特定の入力 (プロンプト) から出力 (レスポンス) を生成する基盤モデルのプロセス。詳細については、「モデル推論を実行する」を参照してください。
- プロンプト – 入力に適切なレスポンスまたは出力を生成するように指示するモデルに提供される入力。例えば、テキストプロンプトは、モデルが応答する 1 行で構成されたり、モデルが実行する指示やタスクを詳細に説明したりできます。プロンプトには、レスポンスで使用するモデルのタスクのコンテキスト、出力例、またはテキストを含めることができます。プロンプトを使用して、分類、質問への回答、コード生成、クリエイティブな記述などのタスクを実行できます。詳細については、「 プロンプトエンジニアリングガイドライン」を参照してください。
- トークン - モデルが単一の意味の単位として解釈または予測できる一連の文字。例えば、テキストモデルでは、トークンは単語だけでなく、文法上の意味 (「-ed」など) 、句読点 (「 ? 」など) 、または一般的なフレーズ (「大量の」など) を持つ単語の一部にも対応できます。
- モデルパラメータ - 入力の解釈とレスポンスの生成におけるモデルとその動作を定義する値。モデルパラメータはプロバイダーによって制御および更新されます。モデルパラメータを更新して、モデルカスタマイズ のプロセスを通じて新しいモデルを作成することもできます。
- 推論パラメータ - レスポンスに影響を与えるためにモデル推論中に調整できる値。推論パラメータは、さまざまなレスポンスがどの程度変化するかに影響し、レスポンスの長さや指定されたシーケンスの出現を制限することもできます。特定の推論パラメータの詳細と定義については、「」を参照してください推論パラメータ。
- プレイグラウンド — Amazon Bedrock に慣れるためにモデル推論の実行を試す AWS Management Console ことができる、 のわかりやすいグラフィカルインターフェイス。プレイグラウンドを使用して、入力したさまざまなプロンプトに対して生成されたレスポンスに対するさまざまなモデル、設定、推論パラメータの影響をテストします。詳細については、「プレイグラウンド」を参照してください。
- 埋め込み – 共有数値表現を使用して異なるオブジェクト間の類似性を比較するため、入力を埋め込み と呼ばれる数値のベクトルに変換して情報を集約するプロセス。例えば、文を比較して意味の類似性を判断したり、画像を比較して視覚的な類似性を判断したり、テキストと画像を比較して相互に関連性があるかどうかを調べたりできます。ユースケースに関連する場合は、テキストと画

像の入力を平均的な埋め込みベクトルに結合することもできます。詳細については、モデル推論を実行するおよびAmazon ベッドロックのナレッジベースを参照してください。

高度な機能

次のリストでは、Amazon Bedrock を使用して詳しく知ることができる、より高度な概念を紹介します。

- オーケストレーション — タスクを実行するために基盤モデルとエンタープライズデータおよびアプリケーションを調整するプロセス。詳細については、「Agents for Amazon Bedrock」を参照してください。
- エージェント — 入力を周期的に解釈し、基盤モデルを使用して出力を生成するオーケストレーションを実行するアプリケーション。エージェントは、顧客のリクエストを実行するために使用できます。詳細については、「Agents for Amazon Bedrock」を参照してください。
- 取得拡張生成 (RAG) — プロンプトに対する生成されたレスポンスを補強するために、データソースから情報をクエリおよび取得するプロセス。詳細については、「Amazon ベッドロックのナレッジベース」を参照してください。
- モデルのカスタマイズ – トレーニングデータを使用して、カスタムモデルを作成するためにベースモデルのモデルパラメータ値を調整するプロセス。モデルのカスタマイズの例としては、ラベル付きデータ (入力と対応する出力) を使用する微調整 や、ラベルなしデータ (入力のみ) を使用してモデルパラメータを調整する継続的な事前トレーニング などがあります。Amazon Bedrock で使用できるモデルカスタマイズ技術の詳細については、「」を参照してくださいカスタムモデル。
- ハイパーパラメータ - モデルのカスタマイズに合わせて調整してトレーニングプロセスと、その結果、出力カスタムモデルを制御できる値。特定のハイパーパラメータの詳細と定義については、「」を参照してくださいカスタムモデルのハイパーパラメータ。
- モデル評価 — ユースケースに最適なモデルを決定するために、モデル出力を評価および比較するプロセス。詳細については、「モデル評価」を参照してください。
- プロビジョンドスループット – モデル推論中に処理されるトークンの量やレートを上げるために、ベースモデルまたはカスタムモデル用に購入するスループットのレベル。モデルのプロビジョンドスループットを購入すると、モデル推論の実行に使用できるプロビジョンドモデルが作成されます。詳細については、「Amazon Bedrock のプロビジョニングされたスループット」を参照してください。

Amazon Bedrock をセットアップする

Amazon Bedrock を初めて使用する場合は、まず以下のタスクを完了してください。アカウントを設定し、コンソールでモデルへのアクセスをリクエストしたら、API を設定できます。

> **Important**
>
> 基盤モデルを使用する前に、そのモデルへのアクセスをリクエストする必要があります。モデルへのアクセスをリクエストしていないのに、(API またはコンソール内で) モデルを使用しようとすると、エラーメッセージが表示されます。詳細については、「モデルアクセス」を参照してください。

セットアップタスク

- にサインアップする AWS アカウント
- 管理ユーザーの作成
- プログラム的なアクセス権を付与する
- コンソールアクセス
- モデルアクセス
- Amazon Bedrock API をセットアップする
- このサービスを AWS SDK で使用する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次のステップを実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話のキーパッドを使用して検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウント内のすべての AWS のサービス とリソースにアクセスできま

す。セキュリティのベストプラクティスとして、管理ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行します。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [アカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理ユーザーの作成

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

をセキュリティで保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理ユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。
2. IAM アイデンティティセンターで、管理ユーザーに管理アクセス権を付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセスを設定する IAM アイデンティティセンターディレクトリAWS IAM Identity Center 」を参照してください。

管理ユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインインユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で とやり取りする場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスしているユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の「 を使用する AWS CLI ための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS APIs AWS SDKs

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	一時的な認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、 AWS CLI、 AWS SDKsまたは AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

コンソールアクセス

Amazon Bedrock コンソールとプレイグラウンドにアクセスするには:

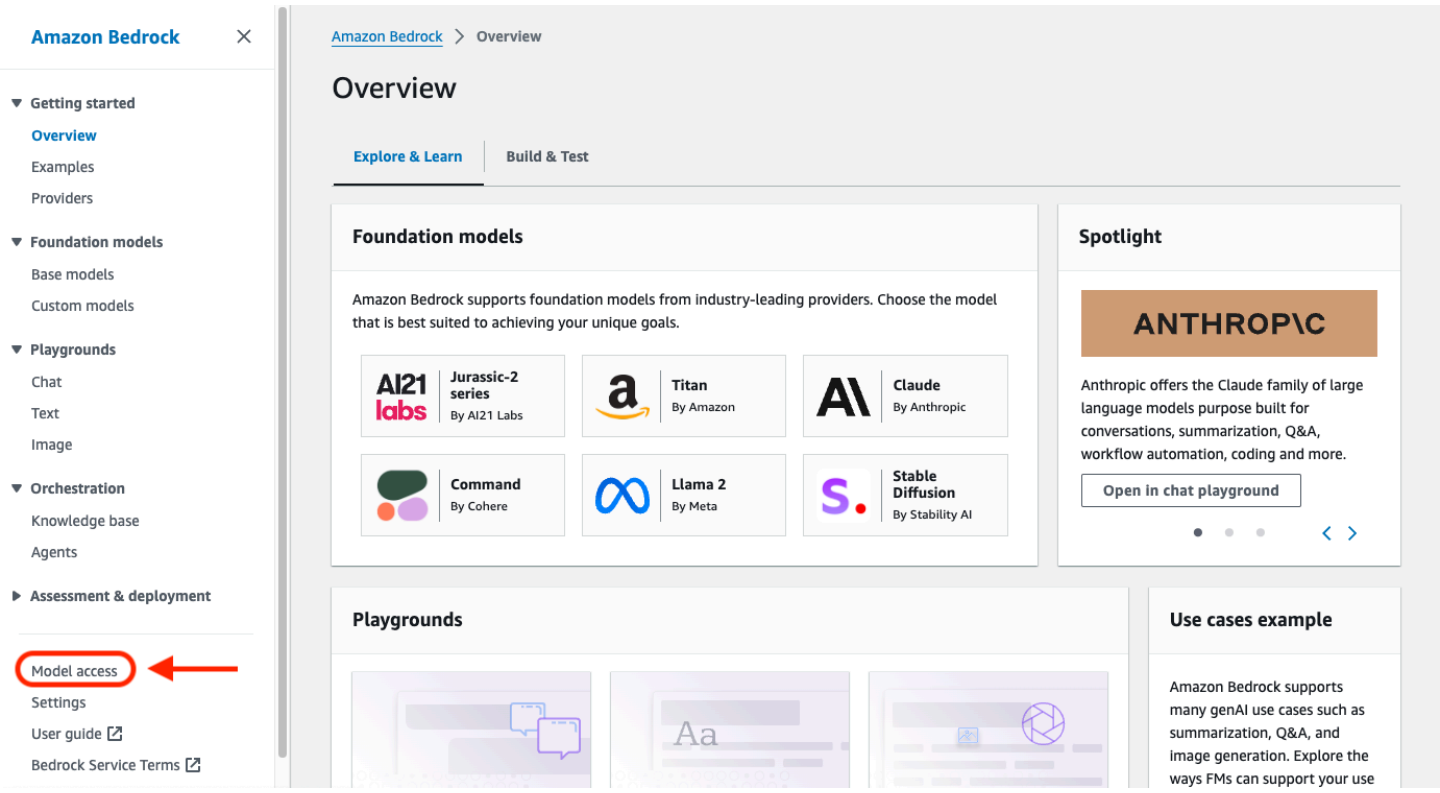
1. にサインインします AWS アカウント。
2. Amazon Bedrock コンソールに移動します。
3. 「モデルアクセス」の手順に従って、モデルアクセスをリクエストしてください。

モデルアクセス

Amazon Bedrock ファンデーションモデルへのアクセスはデフォルトでは許可されていません。基盤モデルにアクセスするには、十分な権限を持つ IAM ユーザーがコンソールからそのモデルへのアクセスをリクエストする必要があります。モデルへのアクセス権が付与されると、そのモデルはそのアカウントのすべてのユーザーが利用できるようになります。

モデルアクセスを管理するには、Amazon Bedrock マネジメントコンソールの左側のナビゲーションペインの下部にある [モデルアクセス] を選択します。モデルアクセスページでは、利用可能なモデルのリスト、モデルの出力方法、そのモデルへのアクセス権の有無、エンドユーザーライセンス契約 (EULA) を表示できます。モデルへのアクセスをリクエストする前に、EULA でモデルの使用条件を確認する必要があります。モデル価格の詳細については、Amazon Bedrock の料金表を参照してください。

> **Note**
>
> モデルアクセスはコンソールからのみ管理できます。



トピック

- モデルアクセスを追加する
- モデルアクセスを削除する
- モデルのアクセス権限を制御します。

モデルアクセスを追加する

Amazon Bedrock で基礎モデルを使用する前に、そのモデルへのアクセスをリクエストする必要があります。

モデルへのアクセスをリクエストするには

1. 「モデルアクセス」ページで、「モデルアクセスの管理」を選択します。
2. アクセスを追加したいモデルの横にあるチェックボックスを選択します。プロバイダーに属するすべてのモデルへのアクセスをリクエストするには、プロバイダーの横にあるチェックボックスを選択します。

   > **Note**
   >
   > Titanリクエストした後でモデルからアクセスを削除することはできません。Anthropicモデルの場合は、[ユースケースの詳細を送信] を選択し、フォームに記入します。詳細が送信されたら、Anthropicアクセスをリクエストしたいモデルの横にあるチェックボックスを選択できます。

3. [変更を保存] を選択してアクセスをリクエストします。変更が反映されるまでに数分かかる場合があります。

   > **Note**
   >
   > Amazon Bedrock ファンデーションモデルの使用には、販売者の価格条件、EULA、AWS およびサービス条件が適用されます。

4. リクエストが成功すると、アクセスステータスは「アクセス許可」 に変わります。

モデルへのアクセスをリクエストする権限がない場合は、エラーバナーが表示されます。アカウント管理者に連絡して、モデルへのアクセスをリクエストしてもらうか、モデルへのアクセスをリクエストする権限を付与してもらってください。

モデルアクセスを削除する

基盤モデルを使用する必要がなくなった場合は、そのモデルへのアクセスを削除できます。

> **Note**
>
> Amazon Titan モデルからアクセスを削除することはできません。

1. 「モデルアクセス」ページで、「モデルアクセスの管理」を選択します。
2. アクセスを削除したいモデルの横にあるチェックボックスを選択します。プロバイダーに属するすべてのモデルのアクセスを削除するには、プロバイダーの横にあるチェックボックスを選択します。
3. [変更を保存] を選択します。
4. モデルのアクセス許可を削除したいかどうかを確認するよう求められます。利用規約に同意して [アクセスを削除] を選択すると、

> **Note**
>
> このアクションを完了した後も、変更が反映されるまで、しばらくの間は API を通じてモデルにアクセスできる場合があります。その間にアクセスをすぐに削除するには、モデルへのアクセスを拒否する IAM ポリシーをロールに追加します。

モデルのアクセス権限を制御します。

Amazon Bedrock モデルへのアクセスをリクエストするロールのアクセス権限を制御するには、AWS Marketplace 以下のアクションのいずれかを使用して IAM ポリシーをロールにアタッチします。

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

`aws-marketplace:Subscribe`アクションの場合のみ、`aws-marketplace:ProductId`条件キーを使用してサブスクリプションを特定のモデルに制限できます。次の表は、Amazon Bedrock ファンデーションモデルの製品 ID を示しています。

モデル	製品 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	プロドオゾニス 2mm PEU
Cohere Command	a61c46fe-1747-41aa-9af0-2e0ae8a9ce05
Cohere Command Light	216b69fd-07d5-4c7b-866b-936456d68311
Cohere埋め込み (英語)	b7568428-a1ab-46d8-bab3-37def50f6f6a
Cohere埋め込み (多言語)	38e55671-c3fe-4a44-9783-3584906e7cad
MetaLlama 21.3B	prod-ariujvyzvd2qy
MetaLlama 270	prod-2c2yc2s3guhqy
Stable Diffusion XL0.8	d0123e8d-50d6-4dba-8a26-3fed4899f388
Stable Diffusion XL 1.0	prod-2lvuzn4iy6n6o

モデルのアクセス権限を制御するためにロールにアタッチできる IAM ポリシーの形式は次のとおりです。サードパーティーモデルのサブスクリプションへのアクセスを許可する例はで確認できます。

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

Amazon Bedrock API をセットアップする

このセクションでは、Amazon Bedrock API コールを実行するための環境設定方法を説明し、代表的なユースケースを示します。Amazon Bedrock API には、 AWS Command Line Interface (AWS CLI)、 AWS SDK、 SageMaker またはノートブックを使用してアクセスできます。

Amazon Bedrock API にアクセスする前に、使用する予定の基盤モデルへのアクセスをリクエストする必要があります。

APIのオペレーションとパラメータの詳細については、「Amazon Bedrock API Reference」を参照してください。

次のリソースは、Amazon Bedrock API に関する追加情報を提供します。

- AWS Command Line Interface
  - Amazon Bedrock CLI コマンド
  - Amazon Bedrock ランタイム CLI コマンド
  - Agents for Amazon Bedrock CLI コマンド
  - Agents for Amazon Bedrock ランタイム CLI コマンド

モデルアクセスを追加する

> **Important**
>
> 基盤モデルを使用する前に、そのモデルへのアクセスをリクエストする必要があります。モデルへのアクセスをリクエストしていないのに、(API またはコンソール内で) モデルを使用しようとすると、エラーメッセージが表示されます。詳細については、「モデルアクセス」を参照してください。

Amazon Bedrock エンドポイント

にプログラムで接続するには AWS のサービス、エンドポイントを使用します。Amazon Bedrock AWS 全般のリファレンス に使用できるエンドポイントの詳細については、の「Amazon Bedrock エンドポイントとクォータ」の章を参照してください。

Amazon Bedrock は以下のサービスエンドポイントを提供します。

- `bedrock` – モデルを管理、トレーニング、およびデプロイするためのコントロールプレーン API が含まれています。詳細については、Amazon Bedrock のアクションと Amazon Bedrock のデータタイプを参照してください。
- `bedrock-runtime`— Amazon Bedrock でホストされているモデルの推論リクエストを行うためのデータプレーン API が含まれています。詳細については、Amazon Bedrock Runtime のアクションと Amazon Bedrock Runtime のデータタイプを参照してください。
- `bedrock-agent` – エージェントとナレッジベースを作成および管理するためのコントロールプレーン API が含まれています。詳細については、Agents for Amazon Bedrock のアクションと Agents for Amazon Bedrock のデータタイプを参照してください。
- `bedrock-agent-runtime`— エージェントを呼び出し、ナレッジベースをクエリするためのデータプレーン API が含まれています。詳細については、Agents for Amazon Bedrock Runtime のアクションと Agents for Amazon Bedrock Runtime のデータタイプを参照してください。

AWS CLI のセットアップ

1. CLI を使用する場合は、『 AWS Command Line Interface ユーザガイドの最新バージョンをインストールまたは更新する』の手順に従ってをインストールして設定します。 AWS CLI

2. 「設定」の手順に従って `aws configure` CLI AWS コマンドを使用して認証情報を設定します AWS CLI。

AWS CLI コマンドと操作については、次のリファレンスを参照してください。

- Amazon Bedrock CLI コマンド
- Amazon Bedrock ランタイム CLI コマンド
- Agents for Amazon Bedrock CLI コマンド
- Agents for Amazon Bedrock ランタイム CLI コマンド

AWS SDK のセットアップ

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で利用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。SDK は、次のような便利なタスクを自動的に実行します。

- サービスリクエストに暗号で署名します。
- リトライリクエスト
- エラーレスポンスを処理する。

各 SDK に関する一般的な情報とコード例、および各 SDK の Amazon Bedrock API リファレンスを参照するには、次の表を参照してください。コード例はでもご覧いただけます。SDK AWS を使用した Amazon Bedrock のコード例

SDK ドキュメント	コードサンプル	Amazon Bedrock のプレフィックス	Amazon Bedrock ランタイムのプレフィックス	Agents for Amazon Bedrock のプレフィックス	Agents for Amazon Bedrock ランタイムのプレフィックス
AWS SDK for C++	AWS SDK for C++ コード例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime

SDK ドキュメント	コードサンプル	Amazon Bedrock のプレフィックス	Amazon Bedrock ランタイムのプレフィックス	Agents for Amazon Bedrock のプレフィックス	Agents for Amazon Bedrock ランタイムのプレフィックス
AWS SDK for Go	AWS SDK for Go コード例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for Java	AWS SDK for Java コード例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for JavaScript	AWS SDK for JavaScript コード例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime
AWS SDK for Kotlin	AWS SDK for Kotlin コード例	bedrock	bedrockruntime	bedrockagent	bedrockagentruntime
AWS SDK for .NET	AWS SDK for .NET コード例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime
AWS SDK for PHP	AWS SDK for PHP コード例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例	bedrock	bedrock-runtime	bedrock-agent	bedrock-agent-runtime
AWS SDK for Ruby	AWS SDK for Ruby コード例	Bedrock	BedrockRuntime	BedrockAgent	BedrockAgentRuntime

SDK ドキュメント	コードサンプル	Amazon Bedrock のプレフィックス	Amazon Bedrock ランタイムのプレフィックス	Agents for Amazon Bedrock のプレフィックス	Agents for Amazon Bedrock ランタイムのプレフィックス
AWS SDK for Rust	AWS SDK for Rust コード例	aws-sdk-bedrock	aws-sdk-bedrockruntime	aws-sdk-bedrockagent	aws-sdk-bedrockagentruntime
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例	BDK	BDR	悪い	バッズ
AWS SDK for Swift	AWS SDK for Swift コード例	AWSBedrock	AWSBedrockRuntime	AWSBedrockAgent	AWSBedrockAgentRuntime

SageMaker ノートブックを使う

SDK for Python (Boto3) を使用して、ノートブックから Amazon Bedrock API オペレーションを呼び出すことができます。 SageMaker

ロールを設定します。 SageMaker

SageMaker このノートブックを使用する IAM ロールに Amazon Bedrock アクセス権限を追加します。

IAM コンソールから、以下の手順を実行します。

1. IAM ロールを選択し、[アクセス許可の追加] を選択して、ドロップダウンリストから [インラインポリシーを作成] を選択します。
2. 次のアクセス許可を含めてください。

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

```
        {
            "Effect": "Allow",
            "Action": "bedrock:*",
            "Resource": "*"
        }
    ]
}
```

信頼関係に以下のアクセス許可を追加します。

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

ランタイムの設定をテストする

次のコードをノートブックに追加し、実行します。

```
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'))
```

Amazon Bedrock の設定をテストする

次のコードをノートブックに追加し、実行します。

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

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

このサービスを AWS SDK で使用する

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で利用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例

SDK ドキュメント	コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

Amazon Bedrock でサポートされている基盤モデル

Amazon Bedrock は、以下のプロバイダーのファンデーションモデル (FM) をサポートしています。[プロバイダー] 列のリンクを選択すると、そのプロバイダーのドキュメントが表示されます。

Amazon Bedrock API でファンデーションモデルを使用するには、そのモデル ID が必要です。モデル ID のリストについては、を参照してくださいAmazon Bedrock モデル ID。

プロバイダー	モデル	入力モダリティ	出力モダリティ	推論パラメータ	ハイパーパラメータ
Amazon	Titan Text G1 - Express	テキスト	テキスト、チャット	リンク	リンク
	Titan Text G1 - Lite	テキスト	テキスト	リンク	リンク
	Titan Image Generator G1	テキスト、画像	イメージ	リンク	リンク
	Titan Embeddings G1 - Text	テキスト	埋め込み	リンク	該当なし
	Titan Multimodal Embeddings G1	テキスト、画像	埋め込み	リンク	リンク
Anthropic	Claude	テキスト	テキスト、チャット	リンク	該当なし
	Claude Instant	テキスト	テキスト、チャット	リンク	該当なし
	Claude 3 Sonnet	テキスト、画像	テキスト、チャット	リンク	該当なし

プロバイダー	モデル	入力モダリティ	出力モダリティ	推論パラメータ	ハイパーパラメータ
	Claude 3 Haiku	テキスト、画像	テキスト、チャット	リンク	該当なし
AI21 Labs	Jurassic-2 Mid	テキスト	テキスト、チャット	リンク	該当なし
	Jurassic-2 Ultra	テキスト	テキスト、チャット	リンク	該当なし
Cohere	Command	テキスト	テキスト	リンク	リンク
	Command Light	テキスト	テキスト	リンク	リンク
	Embed English	テキスト	埋め込み	リンク	該当なし
	Embed Multilingual	テキスト	埋め込み	リンク	該当なし
Meta	Llama 2 Chat13B	テキスト	テキスト、チャット	リンク	該当なし
	Llama 2 Chat70B	テキスト	テキスト、チャット	リンク	該当なし
	Llama 213B (下記の注記を参照)	テキスト	テキスト	リンク	リンク
	Llama 270B (下記の注記を参照)	テキスト	テキスト	リンク	リンク
Mistral AI	Mistral 7B Instruct	テキスト	テキスト	リンク	該当なし

プロバイダー	モデル	入力モダリティ	出力モダリティ	推論パラメータ	ハイパーパラメータ
	Mixtral 8X7B Instruct	テキスト	テキスト	リンク	該当なし
Stability AI	Stable Diffusion XL	テキスト、画像	イメージ	リンク	該当なし

> **Note**
>
> MetaLlama 2(チャット以外の) モデルは、カスタマイズしてプロビジョンドスループットを購入した後にのみ使用できます。

以下のセクションでは、基盤モデルの使用に関する情報とモデルの参照情報を提供します。

トピック

- 基盤モデルの使用
- 基盤モデルに関する情報の取得
- AWS 地域別のモデルサポート
- 機能別のモデルサポート
- モデルのライフサイクル
- Amazon Bedrock モデル ID
- 基盤モデルの推論パラメータ
- カスタムモデルのハイパーパラメータ

基盤モデルの使用

モデルを使用する前に、モデルへのアクセスをリクエストする必要があります。その後、次の方法でFMsを使用できます。

- モデルにプロンプトを送信し、レスポンスを生成して推論を実行します。プレイグラウンドは、テキスト、画像、チャットを生成 AWS Management Console するための使いやすいインターフェイ

スを に提供します。各プレイグラウンドで使用できるモデルについては、出力モダリティ列を参照してください。

> **Note**
>
> コンソールのプレイグラウンドは、埋め込みモデルでの推論の実行をサポートしていません。API を使用して、埋め込みモデルで推論を実行します。

- モデルを評価して出力を比較し、ユースケースに最適なモデルを決定します。
- 埋め込みモデルを活用してナレッジベースを設定します。次に、テキストモデルを使用してクエリに対するレスポンスを生成します。
- エージェントを作成し、モデルを使用してプロンプトで推論を実行してオーケストレーションを実行します。
- トレーニングデータと検証データを組み合わせてモデルをカスタマイズし、ユースケースに合わせてモデルパラメータを調整します。カスタマイズされたモデルを使用するには、そのモデル用にプロビジョンドスループットを購入する必要があります。
- モデルのプロビジョンドスループットを購入して、そのスループットを向上させます。

API で FM を使用するには、使用する適切なモデル ID を決定する必要があります。

ユースケース	モデル ID の検索方法
ベースモデルを使用する	ベースモデル ID チャートで IDsを検索する
ベースモデルのプロビジョンドスループットを購入する	プロビジョンドスループットグラフのモデル ID で IDs を検索し、CreateProvisionedModelThroughputリクエストmodelIdの として使用します。
カスタムモデルのプロビジョンドスループットを購入する	modelId CreateProvisionedModelThroughputリクエストの として、カスタムモデルの名前またはその ARN を使用します。
プロビジョニングされたモデルを使用する	プロビジョンドスループットを作成すると、が返されますprovisionedModelArn 。この ARN はモデル ID です。

ユースケース	モデル ID の検索方法
カスタムモデルを使用する	カスタムモデルのプロビジョンドスループットを購入し、返された をモデル ID `provisionedModelArn` として使用します。

基盤モデルに関する情報の取得

Amazon Bedrock コンソールでは、Amazon Bedrock 基盤モデルプロバイダーとプロバイダーが提供するモデルに関する包括的な情報を、[プロバイダー] セクションと[ベースモデル] セクションで確認できます。

API を使用して、サポートする ARN、モデル ID、モダリティ、特徴、非推奨かどうかなど、Amazon Bedrock 基盤モデルに関する情報を FoundationModelSummary オブジェクトで取得します。

- Amazon Bedrock が提供するすべての基盤モデルに関する情報を返すには、ListFoundationModelsリクエストを送信します。

  > **Note**
  >
  > レスポンスは、プロビジョニングされたスループットグラフのベースモデル IDs またはベースモデル ID に含まれていないモデル ID も返します。 ??? IDs これらのモデル IDs は、下位互換性のために廃止されました。

- 特定の基盤モデルに関する情報を返すには、モデル ID を指定して GetFoundationModelリクエストを送信します。

タブを選択すると、インターフェイスまたは言語でのコード例が表示されます。

AWS CLI

Amazon Bedrock 基盤モデルを一覧表示します。

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

v2 Anthropic Claude に関する情報を取得します。

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

Python

Amazon Bedrock 基盤モデルを一覧表示します。

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

bedrock.list_foundation_models()
```

v2 Anthropic Claude に関する情報を取得します。

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

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

AWS 地域別のモデルサポート

> **Note**
>
> すべてのモデルが米国東部 (バージニア北部us-east-1) と米国西部 (オレゴンus-west-2) リージョンでサポートされています。

次の表は、他のリージョンで使用可能な FM と、各リージョンでサポートされているかどうかを示しています。

モデル	アジアパシフィック (シンガポール)	アジアパシフィック (東京)	欧州 (フランクフルト)	AWS GovCloud (米国西部)
Amazon Titan Text G1 - Express	No	はい	はい	Yes

モデル	アジアパシフィック (シンガポール)	アジアパシフィック (東京)	欧州 (フランクフルト)	AWS GovCloud (米国西部)
Amazon Titan Embeddings G1 - Text	No	はい	はい	No
Anthropic Claudev2 (18K コンテキストウィンドウ)	Yes	いいえ	はい	No
Anthropic Claudev2.1 (200K コンテキストウィンドウ)	No	はい	はい	No
AnthropicClaude Instantv1.x (18K コンテキストウィンドウ)	はい	はい	いいえ	No
AnthropicClaude Instantv1.x (10 万コンテキストウィンドウ)	No	いいえ	はい	No

機能別のモデルサポート

> **Note**
>
> 使用可能なすべての FM で推論を実行できます。

次の表は、特定の FM に限定される機能のサポートの詳細を示しています。

モデル	モデル評価	ナレッジベース (埋め込み)	ナレッジベース (クエリ)	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
Amazon Titan Text G1 - Express	Yes	該当なし	いいえ	いいえ	はい	はい	Yes
Amazon Titan Text G1 - Lite	Yes	該当なし	いいえ	いいえ	はい	はい	Yes
Amazon Titan Embeddings G1 - Text	No	該当なし	いいえ	いいえ	いいえ	いいえ	Yes
Amazon Titan Multimodal Embeddings G1	No	はい	いいえ	いいえ	はい	いいえ	Yes
Amazon Titan Image Generator G1 (プレビュー)	No	該当なし	いいえ	いいえ	はい	いいえ	Yes

モデル	モデル評価	ナレッジベース (埋め込み)	ナレッジベース (クエリ)	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
Anthropic Claudev1	Yes	該当なし	いいえ	いいえ	いいえ	いいえ	Yes
Anthropic Claudev2	Yes	該当なし	はい	はい	いいえ	いいえ	Yes
Anthropic Claude2.1	No	該当なし	はい	はい	いいえ	いいえ	はい
Anthropic Claude Instant	Yes	該当なし	はい	はい	いいえ	いいえ	はい
Anthropic Claude 3 Sonnet	いいえ	該当なし	はい	いいえ	いいえ	いいえ	はい
Anthropic Claude 3 Haiku	いいえ	該当なし	いいえ	いいえ	いいえ	いいえ	はい
AI21 Labs Jurassic-2 Mid	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
AI21 Labs Jurassic-2 Ultra	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ

モデル	モデル評価	ナレッジベース (埋め込み)	ナレッジベース (クエリ)	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
Cohere Command	Yes	該当なし	いいえ	いいえ	はい	いいえ	はい
Cohere Command Light	Yes	該当なし	いいえ	いいえ	はい	いいえ	Yes
CohereEmbed英語	No	はい	いいえ	いいえ	いいえ	いいえ	Yes
CohereEmbed多言語	No	はい	いいえ	いいえ	いいえ	いいえ	Yes
MetaLlama 2 Chat13B	Yes	該当なし	いいえ	いいえ	いいえ	いいえ	Yes
MetaLlama 2 Chat70B	Yes	該当なし	いいえ	いいえ	いいえ	いいえ	No
MetaLlama 213B	No	該当なし	いいえ	いいえ	はい	No	はい (下記の注記を参照)
MetaLlama 270B	No	該当なし	いいえ	いいえ	はい	No	はい (下記の注記を参照)

モデル	モデル評価	ナレッジベース (埋め込み)	ナレッジベース (クエリ)	エージェント	ファインチューニング (カスタムモデル)	継続的な事前トレーニング (カスタムモデル)	プロビジョンドスループット
Mistral AI Mistral 7B Instruct	No	該当なし	いいえ	いいえ	いいえ	いいえ	はい
Mistral AI Mixtral 8X7B Instruct	いいえ	該当なし	いいえ	いいえ	いいえ	いいえ	Yes
Stable Diffusion XL0.8	No	該当なし	いいえ	いいえ	いいえ	いいえ	No
Stable Diffusion XL 1.x	No	該当なし	いいえ	いいえ	いいえ	いいえ	Yes

> **Note**
>
> MetaLlama 2(チャット以外の) モデルは、カスタマイズしてプロビジョンドスループットを購入した後にのみ使用できます。

モデルのライフサイクル

Amazon Bedrock では、より優れた機能、正解率、安全性を備えた最新バージョンの基盤モデルの実現に継続的に取り組んでいます。新しいモデルバージョンがリリースされたら、Amazon Bedrock コンソールまたは API でそれらをテストし、アプリケーションを移行して最新のモデルバージョンを享受できます。

Amazon Bedrock で提供されるモデルは、アクティブ、レガシー、またはサポート終了 (EOL) のいずれかの状態になります。

- アクティブ: モデルプロバイダーがこのバージョンの開発に積極的に取り組んでおり、バグ修正や軽微な改善などの更新が継続されます。
- レガシー: パフォーマンスがより優れた最新バージョンが存在する場合、そのバージョンは「レガシー」と表示されます。Amazon Bedrock はレガシーバージョンのサポート終了日を設定します。EOL 日は、モデルの使用方法 (たとえば、ベースモデルではオンデマンドスループットまたはプロビジョンドスループットを使用するか、カスタマイズモデルではプロビジョンドスループットを使用するかなど) によって異なる場合があります。レガシーバージョンは引き続き使用できますが、EOL 日の前にアクティブバージョンへの移行を計画する必要があります。
- EOL: このバージョンは現在利用できません。このバージョンへのリクエストはすべて失敗します。

コンソールはモデルバージョンの状態を Active または Legacy とマークします。GetFoundationModelOR ListFoundationModels呼び出しを行うと、`modelLifecycle`応答のフィールドでモデルの状態を確認できます。EOL 日以降は、モデルバージョンはこのドキュメントページでのみ確認できます。

オンデマンド、プロビジョンドスループット、およびモデルのカスタマイズ

モデルをオンデマンドモードで使用する場合は、モデルのバージョン (`anthropic.claude-v2anthropic.claude-v2:1`、など) を指定します。

プロビジョンドスループットを設定するときは、期間全体を通じて変更されないモデルのバージョンを指定する必要があります。そのバージョンのサポート終了日より前にコミットメント期間が終了した場合、そのバージョンの新しいプロビジョンドスループットコミットメントを購入 (または既存のコミットメントを更新) できます。

モデルをカスタマイズした場合、カスタマイズに使用したベースモデルのバージョンのサポート終了日までそのモデルを引き続き使用できます。レガシーモデルバージョンもカスタマイズできますが、サポート終了日に達する前に移行することを計画する必要があります。

> **Note**
>
> Service Quotas はモデルのマイナーバージョン間で共有されます。

レガシーバージョン

次の表は、Amazon Bedrock で入手可能なモデルのレガシーバージョンを示しています。

モデルバージョン	レガシーに指定された日付	サポート終了日	推奨されるモデルのバージョンの置き換え	推奨モデル ID
Stable Diffusion XL 0.8	2024 年 2 月 21 日	2024 年 4 月 30 日	Stable Diffusion XL 1.x	安定性。 stable-diffusion-xl-v1
Claude v1.3	2023 年 11 月 28 日	2024年2月28日	Claude v2.1	anthropic.claude-v 2:1
タイタンエンベディング-テキスト v1.1	2023 年 11 月 7 日	2024 年 2 月 15 日	Titan Embeddings - Text v1.2	アマゾン。 titan-embed-text-v1

Amazon Bedrock モデル ID

Amazon Bedrock API オペレーションの多くでは、モデル ID を使用する必要があります。次の表を参照して、使用する必要のあるモデル ID の入手先を確認してください。

ユースケース	モデル ID の確認方法
ベースモデルを使用する	ベースモデル ID チャートで ID を調べてください。
基本モデルのプロビジョニングされたスループットを購入	「プロビジョニングされたスループットのモデル ID」チャートで ID を検索し、`modelId`リクエストで使用しますCreateProvisionedModelThroughput。
カスタムモデル用のプロビジョンドスループットを購入してください。	カスタムモデルの名前またはその ARN `modelId` CreateProvisionedModelThroughputをリクエスト内で使用します。

ユースケース	モデル ID の確認方法
プロビジョニングされたモデルを使用してください。	プロビジョンドスループットを作成すると、が返されます。`provisionedModelArn` このARN はモデル ID です。
カスタムモデルを使用してください。	カスタムモデル用のプロビジョンドスループットを購入し、返されたものをモデル ID `provisionedModelArn` として使用します。

トピック

- Amazon Bedrock ベースモデル ID (オンデマンドスループット)
- プロビジョニングされたスループットを購入するための Amazon Bedrock ベースモデル ID

Amazon Bedrock ベースモデル ID (オンデマンドスループット)

以下は、現在販売されているベースモデルのモデル ID のリストです。API を通じてモデル ID を使用して、リクエスト内などでオンデマンドスループットで使用したいベースモデルや、InvokeModelリクエスト内などでカスタマイズしたいベースモデルを特定します。CreateModelCustomizationJob

> **Note**
>
> モデルのライフサイクルこのページでモデルの廃止に関する情報を定期的に確認し、必要に応じてモデル ID を更新する必要があります。モデルに達すると end-of-life、モデル ID は機能しなくなります。

プロバイダー	モデル名	バージョン	モデル ID
Amazon	Titan Text G1 - Express	1.x	アマゾン。 titan-text-express-v1
Amazon	Titan Text G1 - Lite	1.x	アマゾン。 titan-text-lite-v1

プロバイダー	モデル名	バージョン	モデル ID
Amazon	Titan Embeddings G1 - Text	1.x	アマゾン。 titan-embed-text-v1
Amazon	Titan Multimodal Embeddings G1	1.x	アマゾン。 titan-embed-image-v1
Amazon	Titan Image Generator G1	1.x	アマゾン。 titan-image-generator-v1
Anthropic	Claude	2.0	anthropic.claude-v2
Anthropic	Claude	2.1	アンソロピック・クロード-V 2:1
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
Anthropic	Claude Instant	1.x	アンソロピー。 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	Command	14.x	コヒーレ。 command-text-v14
Cohere	Command Light	15.x	コヒーレ。 command-light-text-v14
Cohere	Embed英語	3.x	コヒーレ。 embed-english-v3

プロバイダー	モデル名	バージョン	モデル ID
Cohere	Embed多言語	3.x	コヒーレ。 embed-multilingual-v3
Meta	Llama 2 Chat13B	1.x	meta.lama2-13 b-chat-v 1
Meta	Llama 2 Chat70	1.x	meta.lama2-70 b-chat-v 1
Mistral AI	Mistral 7B Instruct	0.x	ミストラル・ミストラル-7 0:2 b-instruct-v
Mistral AI	Mixtral 8X7B Instruct	0.x	mistral.mixtral-8x7 b-instruct-v 0:1
Stability AI	Stable Diffusion XL	0.x	安定性。 stable-diffusion-xl-v0
Stability AI	Stable Diffusion XL	1.x	安定性。 stable-diffusion-xl-v1

プロビジョニングされたスループットを購入するための Amazon Bedrock ベースモデル ID

API を通じてプロビジョンドスループットを購入するには、リクエストでモデルをプロビジョニングするときに対応するモデル ID を使用してください。CreateProvisionedModelThroughputプロビジョンドスループットは以下のモデルで使用できます。

> **Note**
>
> 一部のモデルには、リージョンによって可用性が異なる複数のコンテキストバージョンがあります。詳細については、「AWS 地域別のモデルサポート」を参照してください。

モデル名	基本モデルではコミットメントなしの購入がサポートされています。	プロビジョニングされたスループットのモデル ID
Amazon Titan Text G1 - Express	Yes	アマゾン。 titan-text-express-v1:0:8 キロ
Amazon Titan Text G1 - Lite	Yes	アマゾン。 titan-text-lite-v1:0:4 キロ
Amazon Titan Embeddings G1 - Text	Yes	アマゾン。 titan-embed-text-v1:2:8 キロ
Amazon Titan Multimodal Embeddings G1	Yes	アマゾン。 titan-embed-image-v1:0
Amazon Titan Image Generator G1	No	アマゾン。 titan-image-generator-v1:0
AnthropicClaudev2 18K	Yes	anthropic.claude-v2:0:18k
AnthropicClaudev2 100K	Yes	anthropic.claude-v2:0:100k
AnthropicClaude2.1 (18K)	Yes	anthropic.claude-v2:1:18k
AnthropicClaude2.1 200K	Yes	anthropic.claude-v 2:1:200 k
AnthropicClaude 3 Sonnet28K	Yes	anthropic.claude-3-sonnet-20240229-v 1:0:28 k
AnthropicClaude 3 Sonnet200K	Yes	anthropic.claude-3-sonnet-20240229-v 1:0:200 k
AnthropicClaude 3 Haiku48K	Yes	anthropic.claude-3-haiku-20240307-v 1:0:48 k
AnthropicClaude 3 Haiku200K	Yes	anthropic.claude-3-haiku-20240307-v 1:0:200 k

モデル名	基本モデルではコミットメントなしの購入がサポートされています。	プロビジョニングされたスループットのモデル ID
AnthropicClaude Instantv1 100K	Yes	人為的。 claude-instant-v1:2:100 メートル
Cohere Command	Yes	コヒーレ。 command-text-v14:7:4 キロ
Cohere Command Light	Yes	コヒーレ。 command-light-text-v14:7:4 キロ
CohereEmbed英語	Yes	コヒーレ。 embed-english-v3:0:512
CohereEmbed多言語	Yes	コヒーレ。 embed-multilingual-v3:0:512
Stable Diffusion XL 1.0	No	安定性。 stable-diffusion-xl-v1:0
MetaLlama 2 Chat13B	No	meta.llama2-13 1:0:4 k b-chat-v
MetaLlama 213B	No	(下記の注記を参照)
MetaLlama 270B	No	(下記の注記を参照)

> **Note**
>
> MetaLlama 2(チャット以外の) モデルは、カスタマイズしてプロビジョンドスループットを購入した後にのみ使用できます。

> **Note**
>
> MetaLlama 2(チャット以外の) モデルは、カスタマイズした後にのみプロビジョニングできます。詳細については、「カスタムモデル」を参照してください。生成されたカスタムモデルの名前または ARN を使用して、そのモデルのプロビジョンドスループットを購入します。

CreateProvisionedModelThroughputレスポンスはを返します。`provisionedModelArn`この ARN またはプロビジョニングされたモデルの名前は、サポートされている Amazon Bedrock オペレーションで使用できます。プロビジョニングされたスループットの詳細については、「」を参照してください。Amazon Bedrock のプロビジョニングされたスループット

基盤モデルの推論パラメータ

このセクションでは、Amazon Bedrock が提供するベースモデルで使用できる推論パラメータについて説明します。

オプションで、モデルが生成するレスポンスに影響する推論パラメータを設定します。推論パラメータは、コンソールのプレイグラウンド、または InvokeModelまたは InvokeModelWithResponseStream API の bodyフィールドで設定します。

モデルを呼び出すときには、モデルのプロンプトも含めます。プロンプトの書き方については、「プロンプトエンジニアリングガイドライン」を参照してください。

以下のセクションでは、各ベースモデルで使用できる推論パラメータを定義しています。カスタムモデルでは、カスタマイズの基となったベースモデルと同じ推論パラメータを使用してください。

トピック

- Amazon Titanモデル
- AnthropicClaude モデル
- AI21 LabsJurassic-2 モデル
- Cohere モデル
- MetaLlama 2 および MetaLlama 2 Chatモデル
- Mistral AI モデル
- Stability.ai Diffusion モデル

Amazon Titanモデル

以下のページでは、Amazon Titanモデルの推論パラメータについて説明します。

トピック

Amazon Titan テキストモデル

Amazon Titan Text モデルは以下の推論パラメータをサポートしています。

Titanテキストプロンプトエンジニアリングガイドラインの詳細については、「Titanテキストプロンプトエンジニアリングガイドライン」を参照してください。

Titanモデルの詳細については、を参照してくださいAmazon Titanモデル。

トピック

要求と応答

リクエスト本文は InvokeModelOR body InvokeModelWithResponseStreamリクエストのフィールドに渡されます。

Request

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

以下のパラメータは必須です。

- InputText — レスポンスを生成するためのモデルを提供するプロンプト。会話形式で応答を生成するには、次の形式でプロンプトをラップします。

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

textGenerationConfig はオプションです。これを使用して以下の推論パラメータを設定できます。

- 温度 — 低い値を使用すると、応答のランダム性が低くなります。

デフォルト値	最小値	最大値
0.0	0.0	1.0

- TopP — 可能性の低い選択肢を無視して応答の多様性を減らすには、低い値を使用します。

デフォルト値	最小値	最大値
1	1.00E-45	1

- maxTokenCount— レスポンスで生成するトークンの最大数を指定します。

デフォルト値	最小値	最大値
512	0	8,000

- StopSequences — モデルが停止する位置を示す文字シーケンスを指定します。現在、指定できるオプションは以下のうちの 1 つだけです。
  - |
  - User:

InvokeModel Response

レスポンスボディには以下のフィールドが含まれます。

```
{
```

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

各フィールドの詳細は以下のとおりです。

- inputTextTokenCount - プロンプト内のトークンの数。
- tokenCount - レスポンス内のトークンの数。
- outputText - レスポンス内のテキスト。
- completionReason - レスポンスの生成が終了した理由。理由の候補としては、以下があります。
  - FINISHED - レスポンスは完全に生成されました。
  - LENGTH - 設定したレスポンスの長さにより、レスポンスが切り捨てられました。

InvokeModelWithResponseStream Response

レスポンスストリームの本文に含まれるテキストの各チャンクは、次の形式です。bytes フィールドをデコードする必要があります (例については「API を使用して 1 つのプロンプトでモデルを呼び出します。」を参照)。

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

- index - ストリーミングレスポンス内のチャンクのインデックス。
- inputTextTokenCount - プロンプト内のトークンの数。

- totalOutputTextTokenCount - レスポンス内のトークンの数。
- outputText - レスポンス内のテキスト。
- completionReason - レスポンスの生成が終了した理由。理由の候補としては、以下があります。
  - FINISHED - レスポンスは完全に生成されました。
  - LENGTH - 設定したレスポンスの長さにより、レスポンスが切り捨てられました。

コードサンプル

次の例は、Python SDK を使用して Amazon Titan Text G1 - Express モデルを使用して推論を実行する方法を示しています。

```
# 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 additonal 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 occured: " +
              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

Amazon Titan Image Generator G1モデルは、モデル推論を実行するときに、次の推論パラメータとモデルレスポンスをサポートします。

トピック

- リクエストとレスポンスの形式
- コードサンプル

リクエストとレスポンスの形式

Amazon を使用して InvokeModel呼び出しを行う場合はTitan Image Generator G1、リクエストの bodyフィールドをユースケースに合った形式に置き換えます。すべてのタスクは `imageGenerationConfig` オブジェクトを共有しますが、各タスクにはそのタスク固有のパラメータオブジェクトがあります。次のユースケースがサポートされています。

taskType	タスクパラメータフィールド	タスクの種類	定義
`TEXT_IMAGE`	`textToImageParams`	[Generation] (生成)	テキストプロンプトを使用して画像を生成します。
`INPAINTING`	`inPaintingParams`	編集	周囲の背景に合わせてマスクの内側を変更して画像を変更します。

taskType	タスクパラメータフィールド	タスクの種類	定義
OUTPAINTING	outPaintingParams	編集	マスクで定義された領域をシームレスに拡張して画像を変更します。
IMAGE_VARIATION	imageVariationParams	編集	元の画像のバリエーションを作成して画像を変更します。

編集タスクでは、入力時に image フィールドを指定する必要があります。このフィールドは、画像内のピクセルを定義する文字列で構成されます。各ピクセルは 0 ~ 255 の範囲の 3 つの RGB チャンネルで定義されます (例えば、(255 255 0) は黄色を表します)。これらのチャンネルは Base64 でエンコードされます。

使用する画像は、JPEG または PNG 形式である必要があります。

インペインティングやアウトペインティングを行う場合は、修正する画像の一部を定義するマスク、1 つまたは複数の領域も定義します。次の 2 つの方法のいずれかでマスクを定義できます。

- maskPrompt — 画像のマスク対象部分を説明するテキストプロンプトを記述します。
- maskImage — 入力画像内の各ピクセルに (0 0 0) または (255 255 255) とマークして、マスクされた領域を定義する Base64 でエンコードされた文字列を入力します。
  - (0 0 0) と定義されたピクセルは、マスクの内側のピクセルです。
  - (255 255 255) と定義されたピクセルは、マスクの外側のピクセルです。

  写真編集ツールを使用すると、マスクを描画できます。その後、出力された JPEG または PNG 画像を Base64 エンコーディングに変換して、このフィールドに入力できます。それ以外の場合は、代わりに maskPrompt フィールドを使用してモデルにマスクの推測を許可します。

以下のタブを選択して、さまざまな画像生成のユースケースの API リクエスト本文とフィールドの説明を確認してください。

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` フィールドについて以下に説明します。

- text (必須) — 画像を生成するためのテキストプロンプト。
- negativeText (オプション) — 画像に含めない内容を定義するテキストプロンプト。

> **Note**
>
> `negativeText` プロンプトには否定語を使わないでください。例えば、画像に鏡を含めない場合は、`negativeText` プロンプトに「**`mirrors`**」と入力します。**`no mirrors`** とは入力しないでください。

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 フィールドについて以下に説明します。マスクは画像の中で修正する部分を定義します。

- image (必須) — 修正対象の JPEG または PNG 画像。ピクセルのシーケンスを指定する文字列としてフォーマットされ、それぞれが RGB 値で定義されて、base64 でエンコードされます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- 定義するには、次のいずれかのフィールド (両方ではない) を定義する必要があります。
  - maskPrompt — マスクを定義するテキストプロンプト。
  - maskImage — image と同じサイズのピクセルのシーケンスを指定してマスクを定義する文字列。各ピクセルは (0 0 0) (マスクの内側のピクセル) または (255 255 255) (マスクの外側のピクセル) の RGB 値に変換されます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- text (オプション) — マスクの内側の変更内容を定義するテキストプロンプト。このフィールドを含めないと、モデルはマスク領域全体の背景への置き換えを試行します。
- negativeText (オプション) — 画像に含めない内容を定義するテキストプロンプト。

> **Note**
>
> negativeText プロンプトには否定語を使わないでください。例えば、画像に鏡を含めない場合は、negativeText プロンプトに「**mirrors**」と入力します。**no mirrors** とは入力しないでください。

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
    }
}
```

outPaintingParams フィールドについて以下に説明します。マスクは画像の中で修正しない部分を定義します。生成によって、定義する領域がシームレスに拡張されます。

- image (必須) — 修正対象の JPEG または PNG 画像。ピクセルのシーケンスを指定する文字列としてフォーマットされ、それぞれが RGB 値で定義されて、base64 でエンコードされます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- 定義するには、次のいずれかのフィールド (両方ではない) を定義する必要があります。
  - maskPrompt — マスクを定義するテキストプロンプト。
  - maskImage — image と同じサイズのピクセルのシーケンスを指定してマスクを定義する文字列。各ピクセルは (0 0 0) (マスクの内側のピクセル) または (255 255 255) (マスクの外側のピクセル) の RGB 値に変換されます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- text (必須) — マスクの外側の変更内容を定義するテキストプロンプト。
- negativeText (オプション) — 画像に含めない内容を定義するテキストプロンプト。

> **Note**
>
> negativeText プロンプトには否定語を使わないでください。例えば、画像に鏡を含めない場合は、negativeText プロンプトに「**mirrors**」と入力します。**no mirrors** とは入力しないでください。

- outPaintingMode - マスク内のピクセルの変更を許可するかどうかを指定します。以下の値を指定できます。
  - DEFAULT — このオプションを使用すると、再構成された背景との一貫性を保つために、マスクの内側の画像を変更できます。
  - PRECISE — このオプションを使用すると、マスクの内側の画像は変更されません。

Image variation (Request)

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

`imageVariationParams` フィールドについて以下に説明します。

- images (必須) — バリエーションを生成する画像のリスト。現時点で、含めることができる画像数は 1 つのみです。イメージは、base64 でエンコードされたイメージ文字列として定義されます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- text (オプション) — 画像の内側の保存する内容と変更する内容を定義できるテキストプロンプト。
- negativeText (オプション) — 画像に含めない内容を定義するテキストプロンプト。

> **Note**
>
> `negativeText` プロンプトには否定語を使わないでください。例えば、画像に鏡を含めない場合は、`negativeText` プロンプトに「**`mirrors`**」と入力します。**`no mirrors`** とは入力しないでください。

Response body

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

レスポンスの本文は、以下のフィールドのいずれかを含むストリーミングオブジェクトです。

- images — リクエストが正常に終了すると、それぞれが生成画像を定義する base64 エンコード後の文字列のリストであるこのフィールドが返されます。各画像は、ピクセルのシーケンスを指定する文字列としてフォーマットされ、それぞれが RGB 値で定義され、base64 でエンコードされます。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。
- error — 以下の状況のいずれかでリクエストがコンテンツモデレーションポリシーに違反すると、このフィールドにメッセージが返されます。
  - 入力テキスト、画像、またはマスク画像がコンテンツモデレーションポリシーによってフラグ付けされている場合。
  - 少なくとも 1 つの出力画像がコンテンツモデレーションポリシーによってフラグ付けされている場合

共有およびオプションの imageGenerationConfig には、次のフィールドが含まれています。このオブジェクトを含めない場合は、デフォルト設定が使用されます。

- numberOfImages (オプション) - 生成するイメージの数。

最小値	最大値	デフォルト値
1	5	1

- cfgScale (オプション) — 生成された画像がプロンプトに従う程度を指定します。低い値を使用すると、生成時のランダム性が高くなります。

最小値	最大値	デフォルト値
1.1	10.0	8.0

- 以下のパラメータは、出力画像に必要なサイズを定義します。画像サイズ別の料金の詳細については、「Amazon Bedrock の料金」を参照してください。
  - height (オプション) - 画像の高さ (ピクセル単位)。デフォルト値は 1024 です。
  - width (オプション) - 画像の幅 (ピクセル単位)。デフォルト値は 1024 です。

  以下のサイズが許容されます。

幅	高さ	アスペクト比	料金が同じサイズ
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	1280	3:5	1024 x 1024
384	640	3:5	512 x 512
1280	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

幅	高さ	アスペクト比	料金が同じサイズ
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 (オプション) — 結果の制御と再現に使用します。初期ノイズ設定を指定します。前回の実行と同じシードと設定を使用して推論を行えば、類似の画像を作成できます。

> **Note**
>
> seedは、TEXT_IMAGE 生成タスクにのみ設定できます。

最小値	最大値	デフォルト値
0	2,147,483,646	0

コードサンプル

次の例は、Python SDK でオンデマンドスループットで Amazon Titan Image Generator G1モデルを呼び出す方法を示しています。各ユースケースの例を表示するには、タブを選択してください。各例の最後に画像が表示されます。

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 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()
```

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 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()
```

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 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()
```

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 は推論パラメータの使用をサポートしていません。以下のセクションでは、リクエストとレスポンスの形式の詳細とコード例を示します。

トピック

- リクエストとレスポンス
- サンプルのコード

リクエストとレスポンス

リクエストボディは、 InvokeModelリクエストの bodyフィールドで渡されます。

Request

使用可能なフィールドは のみでinputText、埋め込みに変換するテキストを含めることができます。

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

Response

レスポンスbodyの には、次のフィールドが含まれます。

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

フィールドについて以下に説明します。

- 埋め込み — 指定した入力の埋め込みベクトルを表す配列。
- inputTextTokenCount – 入力内のトークンの数。

サンプルのコード

この例では、Amazon Titan Embeddings G1 - Textモデルを呼び出して埋め込みを生成する方法を示します。

```
# 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 occured: " +
              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

このセクションでは、Amazon を使用するためのリクエストおよびリスポネスの本文形式とコード例を示しますTitan Multimodal Embeddings G1。

トピック

- リクエストとレスポンス
- サンプルのコード

リクエストとレスポンス

リクエストボディは、 InvokeModelリクエストの bodyフィールドで渡されます。

Request

Amazon のリクエストボディTitan Multimodal Embeddings G1には、次のフィールドが含まれます。

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

次のフィールドのうち少なくとも 1 つが必要です。両方を含めて、結果のテキスト埋め込みと画像埋め込みベクトルを平均する埋め込みベクトルを生成します。

- inputText – 埋め込みに変換するテキストを入力します。
- inputImage – base64 の埋め込みに変換するイメージをエンコードし、このフィールドに文字列を入力します。画像を Base64 にエンコードし、Base64 でエンコードされた文字列をデコードして画像に変換する方法の例については、コード例を参照してください。

次のフィールドはオプションです。

- embeddingConfig – 出力埋め込みベクトルに次のいずれかの長さを指定する `outputEmbeddingLength`フィールドが含まれます。
  - 256
  - 384
  - 1024 (デフォルト)

Response

レスポンスbodyの には、次のフィールドが含まれます。

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

フィールドについて以下に説明します。

- 埋め込み — 指定した入力の埋め込みベクトルを表す配列。
- inputTextTokenCount – テキスト入力内のトークンの数。
- message – 生成中に発生するエラーを指定します。

サンプルのコード

次の例は、Python SDK でオンデマンドスループットで Amazon Titan Multimodal Embeddings G1モデルを呼び出す方法を示しています。各ユースケースの例を表示するには、タブを選択してください。

Text embeddings

この例では、Amazon Titan Multimodal Embeddings G1モデルを呼び出してテキスト埋め込みを生成する方法を示します。

```
# 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 occured: " +
              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

この例では、Amazon Titan Multimodal Embeddings G1モデルを呼び出して画像埋め込みを生成する方法を示します。

```
# 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 occured: " +
              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

この例では、Amazon Titan Multimodal Embeddings G1モデルを呼び出して、テキストと画像の入力を組み合わせて埋め込みを生成する方法を示します。生成されるベクトルは、生成されたテキスト埋め込みベクトルと画像埋め込みベクトルの平均です。

```
# 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 occured: " +
              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 モデル

このセクションでは、AnthropicClaudeモデルを使用するための推論パラメータとコード例を示します。

Amazon Bedrock を使用して、リクエストを送信AnthropicClaudeテキスト補完 APIまたはAnthropicClaudeメッセージ API推論できます。

メッセージ API を使用して、仮想アシスタントやイメージングアプリケーションなどの会話型アプリケーションを作成します。シングルターンのテキスト生成アプリケーションには、テキスト補完 API を使用します。例えば、ブログ投稿のテキストを生成したり、ユーザーが提供したテキストを要約したりします。

InvokeModel または InvokeModelWithResponseStream (ストリーミング) を使用して AnthropicClaudeモデルに推論リクエストを行います。このとき、使用するモデルのモデル ID が必要になります。Anthropic Claude モデルのモデル ID を取得するには、Amazon Bedrock ベースモデル ID (オンデマンドスループット)「」および「」を参照してくださいプロビジョニングされたスループットを購入するための Amazon Bedrock ベースモデル ID。

> **Note**
>
> 推論呼び出しでシステムプロンプトを使用するには、AnthropicClaudeバージョン 2.1 または Anthropic を使用する必要がありますClaude 3 Sonnet。システムプロンプトの作成については、 AnthropicClaudeドキュメントの「https://docs.anthropic.com/claude/docs/how-to-use-system-prompts」を参照してください。
> Anthropic Claude バージョン 2.1 でタイムアウトが発生しないように、 `prompt`フィールドの入力トークン数を 180K に制限することをお勧めします。このタイムアウトの問題は間もなく解決される予定です。

推論呼び出しで、 `body`フィールドに、行うタイプ呼び出しに準拠する JSON オブジェクト、AnthropicClaudeテキスト補完 APIまたは を入力しますAnthropicClaudeメッセージ API。

Anthropic Claude モデル用のプロンプトの作成については、 AnthropicClaudeドキュメントの「プロンプト設計の概要」を参照してください。

トピック

- AnthropicClaudeテキスト補完 API
- AnthropicClaudeメッセージ API

AnthropicClaudeテキスト補完 API

このセクションでは、テキスト補完 API Anthropic Claude でモデルを使用する際の推論パラメータとコード例を紹介します。

トピック

- AnthropicClaudeテキスト補完 API の概要
- サポートされているモデル
- リクエストとレスポンス
- コード例

AnthropicClaudeテキスト補完 API の概要

テキスト補完 API を使用すると、ユーザー指定のプロンプトから 1 ターンのテキストを生成できます。たとえば、テキスト補完 API を使用して、ブログ投稿用のテキストを生成したり、ユーザーからのテキスト入力を要約したりできます。

AnthropicClaudeモデル用のプロンプトの作成について詳しくは、「プロンプトデザイン入門」を参照してください。既存のテキスト補完プロンプトをで使用する場合はAnthropicClaudeメッセージ API、「テキスト補完からの移行」を参照してください。

サポートされているモデル

テキスト補完 API は以下のモデルで使用できます。Anthropic Claude

- AnthropicClaudeInstantv1.2
- AnthropicClaudev2
- AnthropicClaude2.1

リクエストとレスポンス

リクエスト本文は、bodyInvokeModelInvokeModelWithResponseStreamリクエストのフィールドでまたはに渡されます。

詳細については、AnthropicClaudeドキュメントの https://docs.anthropic.com/claude/reference/complete_post を参照してください。

Request

AnthropicClaudeには、テキスト補完推論呼び出し用の以下の推論パラメータがあります。

```
{
    "prompt": "\n\nHuman:<prompt>\n\nAssistant:",
    "temperature": float,
```

```
    "top_p": float,
    "top_k": int,
    "max_tokens_to_sample": int,
    "stop_sequences": [string]
}
```

必須パラメータを以下に示します。

- prompt — (必須) Claude に入力させたいプロンプト。適切な応答を生成するには、\n\nHuman:交互のターンと会話のターンを使用してプロンプトをフォーマットする必要があります。\n\nAssistant:例:

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

  詳細については、ドキュメントの「プロンプト検証」を参照してください。Anthropic Claude
- max_tokens_to_sample — (必須) 停止する前に生成するトークンの最大数。最適なパフォーマンスを得るには、トークンを 4,000 個に制限することをお勧めします。

  AnthropicClaudeモデルがの値に達する前にトークンの生成を停止する場合があることに注意してください。max_tokens_to_sampleAnthropicClaudeモデルによって、このパラメーターの最大値は異なります。詳細については、AnthropicClaudeドキュメントの「モデル比較」を参照してください。

デフォルト値	最小値	最大値
200	0	4096

オプションのパラメータを以下に示します。

- stop_sequences — (オプション) モデルの生成を停止させるシーケンス。

  AnthropicClaudeモデルはストップオンし"\n\nHuman:"、future 追加の組み込みストップシーケンスが含まれる可能性があります。stop_sequences推論パラメータを使用して、テキスト生成を停止するようモデルに信号を送る文字列を追加してください。
- 温度 — (オプション) 応答に注入されるランダム性の量。

  デフォルトは 1 です。範囲は 0 から 1 です。分析的/多肢選択式の場合は0に近いtempを使用し、クリエイティブでジェネレーティブなタスクには1に近いtempを使用してください。

デフォルト値	最小値	最大値
0.5	0	4096

- top_p — (オプション) 核サンプリングを使用します。

  nucleus サンプリングでは、AnthropicClaude後続の各トークンのすべてのオプションの累積分布を確率の降順で計算し、で指定された特定の確率に達したらそれを切り捨てます。`top_p``temperature`どちらか一方を変更する必要がありますが`top_p`、両方は変更しないでください。

デフォルト値	最小値	最大値
1	0	1

- top_k — (オプション) 後続の各トークンの上位 K 個のオプションからのみサンプリングします。

  ロングテールの低確率応答を削除する場合に使用します`top_k`。

デフォルト値	最小値	最大値
250	0	500

Response

AnthropicClaudeこのモデルは、テキスト補完推論呼び出しに対して以下のフィールドを返します。

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

- 完了 — 停止シーケンスまでの場合と終了シーケンスを除いた場合の補完です。
- stop_reason — モデルが応答の生成を停止した理由。

  - 「stop_sequence」 — モデルが停止シーケンスに達しました。停止シーケンスは、stop_sequencesユーザーが推論パラメーターで指定したか、モデルに組み込まれた停止シーケンスのいずれかです。
  - 「max_tokens」 — max_tokens_to_sample モデルまたはモデルの最大トークン数を超えました。
- stop — stop_sequences 推論パラメーターを指定すると、stopモデルにテキスト生成を停止するよう通知する停止シーケンスが含まれます。たとえば、holes次のレスポンスでは。

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

  指定しない場合stop_sequences、stopの値は空になります。

コード例

以下の例は、AnthropicClaudeV2 モデルをオンデマンドスループットで呼び出す方法を示しています。AnthropicClaudeバージョン 2.1 を使用するには、の値をに変更しますmodelId。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'))
```

次の例は、「*write an essay for living on mars in 1000 words (############## 1,000 #########)*」というプロンプトと、Anthropic Claude V2 モデルを使用して Python でストリーミングテキストを生成する方法を示しています。

```
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メッセージ API

このセクションでは、Anthropic Claude Messages API を使用するための推論パラメータとコード例を紹介します。

トピック

- AnthropicClaudeメッセージ API の概要
- サポートされているモデル
- リクエストとレスポンス

- コードサンプル

AnthropicClaudeメッセージ API の概要

Messages API を使用して、チャットボットやバーチャルアシスタントアプリケーションを作成できます。API は、AnthropicClaudeユーザーとモデル (アシスタント) 間の会話のやりとりを管理します。

Anthropicユーザーとアシスタントの会話を交互に行うように Claude モデルをトレーニングします。新しいメッセージを作成するときは、messagesパラメーターで前の会話の順番を指定します。次に、モデルは会話内の次のメッセージを生成します。

各入力メッセージは、役割と内容を含むオブジェクトでなければなりません。1 つのユーザーロールメッセージを指定することも、複数のユーザーメッセージとアシスタントメッセージを含めることもできます。最初のメッセージには必ずユーザーロールを使用する必要があります。

返信元を事前入力する手法 Claude (最後のアシスタントロールの Message を使用して Claude の応答の先頭に入力する) を使用している場合は、Claude中断したところから再開して応答します。この手法では、Claude引き続きアシスタントロールの応答が返されます。

最後のメッセージがアシスタントロールを使用している場合、応答内容はそのメッセージの内容からすぐに続きます。これを利用して、モデルの応答の一部を制限できます。

ユーザーメッセージが 1 つの場合の例:

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

会話の順番が複数ある例:

```
[
  {"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?"},
]
```

Claude からの回答の一部が入力された例:

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

```
]
```

各入力メッセージの内容は、単一の文字列でも、各ブロックが特定のタイプを持つコンテンツブロックの配列でもかまいません。文字列を使用することは、「テキスト」タイプの 1 つのコンテンツブロックの配列の省略形です。以下の入力メッセージは同等です。

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

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

AnthropicClaudeモデル用のプロンプトの作成について詳しくは、ドキュメンテーションの「プロンプト入門」を参照してください。Anthropic Claudemessages API に移行したい既存のテキスト補完プロンプトがある場合は、「テキスト補完からの移行」を参照してください。

システムプロンプト

システムプロンプトをリクエストに含めることもできます。システムプロンプトでは、特定の目標や役割を指定するなどAnthropicClaude、コンテキストや指示を提供できます。次の例のように、systemフィールドにシステムプロンプトを指定します。

```
"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."
```

詳細については、Anthropicドキュメントの「システムプロンプト」を参照してください。

マルチモーダルプロンプト

マルチモーダルプロンプトは、複数のモダリティ (画像とテキスト) を 1 つのプロンプトにまとめます。モダリティは入力フィールドで指定します。content次の例は、AnthropicClaude提供された画像の内容を説明するように求める方法を示しています。サンプルコードについては、「マルチモーダルコードの例」を参照してください。

```
{
    "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?"
                }
            ]
        }
    ]
}
```

モデルには最大 20 個の画像を指定できます。アシスタントロールには画像を追加できません。

リクエストに含める各画像は、トークンの使用量にカウントされます。詳しくは、Anthropicドキュメントの「画像コスト」を参照してください。

サポートされているモデル

Messages API Anthropic Claude は以下のモデルで使用できます。

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

リクエストとレスポンス

リクエスト本文は、bodyInvokeModelInvokeModelWithResponseStreamリクエストのフィールドでまたはに渡されます。1 回のリクエストで送信できるペイロードの最大サイズは 20 MB です。

詳細については、https://docs.anthropic.com/claude/reference/messages_post を参照してください。

Request

AnthropicClaudeには、メッセージ推論呼び出し用の以下の推論パラメータがあります。

```
{
    "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]
}
```

必須パラメータを以下に示します。

- anthropic_version — (必須) アンソロピックバージョン。値は bedrock-2023-05-31 にする必要があります。
- max_tokens — (必須) 停止する前に生成するトークンの最大数。

  AnthropicClaudeモデルがの値に達する前にトークンの生成を停止する場合があることに注意してください。max_tokensAnthropicClaudeモデルによって、このパラメーターの最大値は異なります。詳細については、「モデル比較」を参照してください。
- メッセージ — (必須) 入力メッセージ。
  - 役割 — 会話ターンの役割。有効な値は、user および assistant です。
  - 内容 — (必須) 会話ターンの内容。
    - タイプ — (必須) コンテンツのタイプ。有効な値は、image および text です。

      指定する場合はimage、次の形式で画像ソースも指定する必要があります。

source — (必須) 会話の内容が変わります。

- type — (必須) 画像のエンコードタイプ。指定できますbase64。
- media_type — (必須) 画像のタイプ。次の画像形式を指定できます。
  - image/jpeg
  - image/png
  - image/webp
  - image/gif
- データ — (必須) 画像の base64 でエンコードされた画像バイト。イメージの最大サイズは 3.75 MB です。画像の高さと幅の最大値は 8000 ピクセルです。

指定する場合はtext、textでのプロンプトも指定する必要があります。

オプションのパラメータを以下に示します。

- system — (オプション) リクエストのシステムプロンプト。

  システムプロンプトは、特定の目標や役割を指定するなどAnthropicClaude、コンテキストや指示を提供する方法です。詳細については、Anthropicドキュメントの「システムプロンプトの使用方法」を参照してください。

  > **Note**
  >
  > AnthropicClaudeシステムプロンプトはバージョン 2.1 以降で使用できます。

- stop_sequences — (オプション) モデルの生成を停止させるカスタムテキストシーケンス。 AnthropicClaudeモデルは通常、自然に順番が終わると停止します。この場合、stop_reason応答フィールドの値はです。end_turnカスタムテキスト文字列が見つかったときにモデルの生成を停止させたい場合は、stop_sequencesパラメーターを使用できます。モデルがカスタムテキスト文字列のいずれかを検出した場合、stop_reasonstop_sequence応答フィールドの値はで、stop_sequenceの値には一致した停止シーケンスが含まれます。

  エントリの最大数は 8191 です。

- 温度 — (オプション) 応答に注入されるランダム性の量。

デフォルト値	最小値	最大値
1	0	1

- top_p — (オプション) 核サンプリングを使用します。

  nucleus サンプリングでは、AnthropicClaude後続の各トークンのすべてのオプションの累積分布を確率の降順で計算し、で指定された特定の確率に達したらその累積分布をカットします。top_ptemperatureどちらか一方を変更する必要がありますがtop_p、両方は変更しないでください。

デフォルト値	最小値	最大値
0.999	0	1

オプションのパラメータを以下に示します。

- top_k — (オプション) 後続の各トークンの上位 K 個のオプションからのみサンプリングします。

  ロングテールの低確率応答を削除する場合に使用しますtop_k。

デフォルト値	最小値	最大値
デフォルトでは無効になっています。	0	100,000,000

Response

AnthropicClaudeモデルはメッセージ推論呼び出しに対して以下のフィールドを返します。

```
{
    "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 — 応答の一意の識別子。ID の形式と長さは時間の経過とともに変化する可能性があります。
- モデル — Anthropic Claude リクエストを行ったモデルの ID。
- stop_reason — Anthropic Claude レスポンスの生成を停止した理由。
  - end_turn — モデルが自然停止点に達した
  - max_tokens — `max_tokens` 生成されたテキストが入力フィールドの値を超えているか、モデルがサポートするトークンの最大数を超えています。'
  - stop_sequence — 入力フィールドで指定したストップシーケンスのいずれかをモデルが生成しました。`stop_sequences`
- タイプ — 応答のタイプ。値は常に `message` です。
- role — 生成されたメッセージの会話型役割。値は常に `assistant` です。
- コンテンツ — モデルによって生成されたコンテンツ。配列として返されます。
  - タイプ — コンテンツのタイプ。現在、サポートされている値は `text` のみです。
  - テキスト — コンテンツのテキスト。
- usage — リクエストで指定したトークンの数と、レスポンスでモデルが生成したトークンの数を格納するコンテナ。
  - input_tokens — リクエストに含まれる入力トークンの数。
  - output_tokens — レスポンスでモデルが生成したトークンの数。
  - stop_sequence — 入力フィールドで指定したストップシーケンスのいずれかをモデルが生成しました。`stop_sequences`

コードサンプル

以下のコード例は、メッセージ API の使用方法を示しています。

トピック

- メッセージコード例
- マルチモーダルコードの例

メッセージコード例

この例は、シングルターンのユーザーメッセージと、AnthropicClaude 3 Sonnetあらかじめ入力されたアシスタントメッセージを含むユーザーターンをモデルに送信する方法を示しています。

```
# 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()
```

マルチモーダルコードの例

以下の例は、マルチモーダルメッセージ内の画像とプロンプトテキストをモデルに渡す方法を示しています。Anthropic Claude 3 Sonnet

トピック

- マルチモーダルプロンプトと InvokeModel
- を使用してマルチモーダルプロンプトをストリーミングします。InvokeModelWithResponseStream

マルチモーダルプロンプトと InvokeModel

次の例は、with にマルチモーダルプロンプトを送信する方法を示しています。Anthropic Claude 3 Sonnet 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 occured: " +
              format(message))


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

を使用してマルチモーダルプロンプトをストリーミングします。 InvokeModelWithResponseStream

次の例は、with に送信されたマルチモーダルプロンプトからの応答をストリーミングする方法を示しています。Anthropic Claude 3 Sonnet InvokeModelWithResponseStream

```
# 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 occured: " +
              format(message))


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

AI21 LabsJurassic-2 モデル

このセクションでは、AI21 LabsAI21 LabsJurassic-2モデルを使用するための推論パラメータとコード例を示します。

トピック

- 推論パラメータ
- コード例

推論パラメータ

AI21 Labs Jurassic-2 モデルでは、次の推論パラメータがサポートされています。

トピック

ランダム性と多様性

AI21 Labs Jurassic-2 モデルは、レスポンスのランダム性と多様性を制御するために、次のパラメータをサポートします。

- 温度 (`temperature`) - 低い値を指定すると、レスポンスのランダム性を減らすことができます。
- トップ P (`topP`) - 低い値を指定すると、可能性の低い選択肢を無視することができます。

長さ

AI21 Labs Jurassic-2 モデルは、生成されたレスポンスの長さを制御するために、次のパラメータをサポートします。

- 最大入力長 (`maxTokens`) - 生成されるレスポンスで使用されるトークンの最大数を指定します。
- ストップシーケンス (`stopSequences`) - モデルに認識させて、それ以降はトークンの生成を停止させるストップシーケンスを設定します。ストップシーケンスに改行文字を挿入するには、Enter キーを押します。ストップシーケンスの挿入を終了するには、Tab キーを使用します。

繰り返し

AI21 Labs Jurassic-2 モデルは、生成されたレスポンスの繰り返しを制御するために、次のパラメータをサポートします。

- プレゼンスのペナルティ (`presencePenalty`) - 高い値を指定すると、プロンプトまたは入力内容に既に少なくとも 1 回出現しているトークンが新規に生成される可能性が低くなります。

- カウントのペナルティ (countPenalty) - 高い値を指定すると、プロンプトまたは入力内容に既に少なくとも 1 回出現しているトークンが新規に生成される可能性が低くなります。この値は、出現回数に比例します。
- 頻度のペナルティ (frequencyPenalty) - 高い値を指定すると、プロンプトまたは完了時にすでに 1 回以上出現する新しいトークンが生成される可能性が低くなります。この値は、トークンの出現頻度に比例します (テキストの長さに合わせて正規化されます)。
- 特殊トークンにペナルティを課す - 特殊文字が繰り返し使用される可能性を減らします。デフォルト値は true です。
  - 空白 (applyToWhitespaces) - true の値を指定すると、空白や改行にペナルティが課されます。
  - 句読点 (**applyToPunctuation**) - true の値を指定すると、句読点にペナルティが課されます。
  - 数字 (applyToNumbers) - true の値を指定すると、数字にペナルティが課されます。
  - ストップワード (applyToStopwords) - true の値を指定すると、ストップワードにペナルティが課されます。
  - 絵文字 (applyToEmojis) - true の値を指定すると、絵文字がペナルティの対象から除外されます。

モデル呼び出しリクエストの body フィールド

AI21 Labs モデルを使用して InvokeModelまたは をInvokeModelWithResponseStream呼び出す場合は、以下のオブジェクトに準拠する JSON オブジェクトをbodyフィールドに入力します。prompt フィールドにプロンプトを入力します。

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

```
    }
}
```

特殊なトークンにペナルティを課すには、それらのフィールドをペナルティオブジェクトに追加します。例えば、countPenalty フィールドを次のように変更できます。

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

次の表は、数値パラメータの最小値、最大値、およびデフォルト値を示しています。

カテゴリ	パラメータ	JSON 形式のオブジェクト	最小値	最大値	デフォルト値
ランダム性と多様性	温度	temperature	0	1	0.5
	トップ P	topP	0	1	0.5
長さ	最大トークン (中規模、超大規模、および大規模モデル)	maxTokens	0	8,191	200
	最大トークン (上記以外のモデル)		0	2,048	200
繰り返し	プレゼンスのペナルティ	presencePenalty	0	5	0
	カウントのペナルティ	countPenalty	0	1	0

カテゴリ	パラメータ	JSON 形式のオブジェクト	最小値	最大値	デフォルト値
	頻度のペナルティ	frequency Penalty	0	500	0

モデル呼び出しレスポンスの body フィールド

レスポンス内の フィールドの形式については、bodyhttps://docs.ai21.com/reference/j2-complete-ref を参照してください。

> **Note**
>
> Amazon Bedrock は、レスポンス識別子 (id) を整数値として返します。

コード例

この例では、A2I AI21 LabsJurassic-2 Midモデルを呼び出す方法を示します。

```
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 モデル

以下は、Amazon Bedrock がサポートするCohereモデルの推論パラメータ情報です。

トピック

- CohereCommand モデル
- CohereEmbed モデル

CohereCommand モデル

InvokeModel または InvokeModelWithResponseStream (ストリーミング) を使用して CohereCommandモデルに推論リクエストを行います。このとき、使用するモデルのモデル ID が必要になります。モデル ID を取得するには、「」を参照してくださいAmazon Bedrock モデル ID。

トピック

- リクエストとレスポンス
- コード例

リクエストとレスポンス

Request

Cohere Command モデルには、次の推論パラメータがあります。

```
{
    "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"
}
```

必須パラメータを以下に示します。

- prompt – (必須) レスポンスを生成するための開始点となる入力テキスト。

  以下は、呼び出しごとのテキストと文字制限です。

オプションのパラメータを以下に示します。

- return_likeliTAKs – トークンの可能性をレスポンスとともに返す方法と、返すかどうかを指定します。以下のオプションを指定できます。
  - GENERATION - 生成されたトークンの可能性のみを返します。
  - ALL - すべてのトークンの可能性を返します。
  - NONE - (デフォルト) 可能性を一切返しません。
- stream – (ストリーミングをサポートするために必要) piece-by-piece レスポンスtrueをリアルタイムで返すには を指定し、プロセス終了後に完全なレスポンスを返falseすには を指定します。
- logit_bias - モデルが不要なトークンを生成できないようにしたり、目的のトークンを含めるようにモデルにインセンティブを与えたりします。形式は {token_id: bias} です。ここで、bias は -10 から 10 までの間にある浮動小数値です。トークンは、 Cohereの Tokenize エンドポイントなど、任意のトークナイゼーションサービスを使用してテキストから取得できます。詳細については、「 Cohereドキュメント」を参照してください。

デフォルト値	最小値	最大値
該当なし	-10 (トークンバイアスとして)	10 (トークンバイアスとして)

- num_ generations – モデルが返す世代の最大数。

デフォルト値	最小値	最大値
1	1	5

- truncate – API がトークンの最大長よりも長い入力を処理する方法を指定します。以下のいずれかを使用します。
  - NONE - 入力が入力トークンの最大長を超えるとエラーを返します。
  - START - 入力の先頭部分を切り捨てます。
  - END- (デフォルト) 入力の末尾部分を切り捨てます。

  START または END を指定すると、入力の長さがモデルの入力トークンの最大長とまったく同じになるまで、モデルが入力内容を切り捨てます。
- temperature – レスポンスのランダム性を減らすには、値を低く設定します。

デフォルト値	最小値	最大値
0.9	0	5

- p – 上位 P。低い値を使用すると、可能性の低いオプションを無視できます。0 または 1.0 に設定すると、このオプションは無効になります。p と k を両方とも有効にした場合は、k が動作した後に p が動作します。

デフォルト値	最小値	最大値
0.75	0	1

- k - トップ K。モデルが次のトークンの生成に使用するトークンの選択肢の数を指定します。p と k を両方とも有効にした場合は、k が動作した後に p が動作します。

デフォルト値	最小値	最大値
0	0	500

- max_tokens – 生成されたレスポンスで使用するトークンの最大数を指定します。

デフォルト値	最小値	最大値
20	1	4096

- stop_sequences – モデルが認識するシーケンスを最大 4 つ設定します。モデルがストップシーケンスに遭遇すると、それ以降のトークンの生成を停止します。返されるテキストにはストップシーケンスは含まれません。

Response

このレスポンスに指定できるフィールドについて説明します。

```
{
    "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 - 生成された結果と、リクエストされたトークンの可能性から成るリスト。(常に返されます)。リストの各世代 (generation) オブジェクトには、以下のフィールドを指定します。
  - id - 世代の識別子。(常に返されます)。
  - likelihood - 出力される可能性。この値は、token_likelihoods におけるトークンの可能性の平均値です。return_likelihoods 入力パラメータを指定すると返されます。
  - token_likelihoods - トークンごとの可能性の配列。return_likelihoods 入力パラメータを指定すると返されます。

- finish_reason - モデルがトークンの生成を完了した理由。- COMPLETE モデルは完成した返信を送信し直しました。MAX_TOKENS- モデルがコンテキスト長の最大トークン数に達したため、返信が切断されました。 ERROR — 返信の生成中に問題が発生しました。 ERROR_TOXIC - モデルは、健康であると見なされた返信を生成しました。finish_reason は is_finished= の場合にのみ返されますtrue。(返されない場合もあります)。
- is_finished - stream が true の場合にのみ使用されるブール値型フィールド。ストリーミングレスポンスの一部として生成される追加のトークンがあることを示します。(返されない場合もあります)。
- text - 生成されたテキスト。
- index - ストリーミングレスポンスにおいて、特定のトークンがどの世代に属しているかを判断するのに使用されます。1 つのレスポンスのみがストリーミングされる場合、すべてのトークンが同じ世代に属し、index の値は返されません。 そのため、index が返されるのは、ストリーミングリクエストにおいて num_generations の値が 1 より大きい場合に限ります。

- prompt — 入力リクエストからのプロンプト (常に返されます) 。
- id - リクエストの識別子 (常に返されます)。

詳細については、 Cohereドキュメントの「https://docs.cohere.com/reference/generate。

コード例

この例では、CohereCommandモデルを呼び出す方法を示します。

```
# 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 reqest 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_likelihoods": "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 occured: " +
              format(message))
    else:
        print(f"Finished generating text with Cohere model {model_id}.")


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

CohereEmbed モデル

Embed モデルに推論リクエストを行うには、 InvokeModel を使用します。使用するモデルのモデル ID が必要です。モデル ID を取得するには、「」を参照してくださいAmazon Bedrock モデル ID。

> **Note**
>
> Amazon Bedrock は、CohereEmbedモデルからのストリーミングレスポンスをサポートしていません。

トピック

- リクエストとレスポンス
- コード例

リクエストとレスポンス

Request

Cohere Embed モデルには、次の推論パラメータがあります。

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

必須パラメータを以下に示します。

- texts – (必須) 埋め込むモデルの文字列の配列。最適なパフォーマンスを得るには、各テキストの長さを 512 トークン未満に減らすことをお勧めします。1 トークンは約 4 文字です。

  以下は、呼び出しごとのテキストと文字制限です。

  呼び出しあたりのテキスト

最小値	最大値	
0 テキスト	128 テキスト	

文字数

最小値	最大値	
0 文字	2,048 文字	

オプションのパラメータを以下に示します。

- input_type - 各タイプを互いに区別するために特別なトークンを追加します。検索と取得の間でタイプを混在させる場合を除いて、異なるタイプを混在させないでください。そのような場合、search_document タイプにはコーパスを、search_query タイプには埋め込みクエリを埋め込みます。
  - search_document - 検索のユースケースで、ベクトルデータベースに保存する埋め込み用のドキュメントをエンコードするときに、search_document を使用します。
  - search_query - ベクトル DB にクエリを実行して関連ドキュメントを検索する場合に search_query を使用します。
  - classification - 埋め込みをテキスト分類子への入力として使用する場合に classification を使用します。
  - clustering - 埋め込みをクラスター化する場合に clustering を使用します。
- truncate – API がトークンの最大長よりも長い入力を処理する方法を指定します。以下のいずれかを使用します。
  - NONE - (デフォルト) 入力が入力トークンの最大長を超えるとエラーを返します。
  - LEFT - 入力の先頭部分を切り捨てます。
  - RIGHT - 入力の末尾部分を切り捨てます。

  LEFT または RIGHT を指定すると、入力の長さがモデルの入力トークンの最大長とまったく同じになるまで、モデルが入力内容を切り捨てます。

詳細については、 Cohereドキュメントの「https://docs.cohere.com/reference/embed。

Response

InvokeModel を呼び出した場合の body レスポンスを以下に示します。

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

body レスポンスに指定できるフィールドについて説明します。

- id - レスポンスの識別子。
- response_type – レスポンスタイプ。この値は常に embeddings_floats です。
- embeddings - 埋め込みの配列。各埋め込みは 1,024 個の要素から成る、浮動小数点数の配列です。embeddings 配列の長さは 元の texts 配列の長さと同じになります。
- texts - 埋め込みが返されたテキストエントリから成る配列。

詳細については、https://docs.cohere.com/reference/embed を参照してください。

コード例

この例では、CohereCommandEmbedモデルを呼び出す方法を示します。

```
# 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 reqest body to use.
    Returns:
        dict: The response from the model.
    """

    logger.info(
        "Generating text emdeddings 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 occured: " +
              format(message))
    else:
        print(
            f"Finished generating text embeddings with Cohere model {model_id}.")


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

MetaLlama 2 および MetaLlama 2 Chatモデル

このセクションでは、 および MetaLlama 2 Chatモデルを使用するための推論パラメータMetaLlama 2とコード例を示します。

トピック

- リクエストとレスポンス
- サンプルのコード

リクエストとレスポンス

リクエストボディは、 InvokeModelまたは へのリクエストの bodyフィールドで渡されますInvokeModelWithResponseStream。

Request

Meta Llama 2 Chat および Llama 2モデルには、次の推論パラメータがあります。

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

必須パラメータを以下に示します。

- prompt – (必須) モデルに渡すプロンプト。

オプションのパラメータを以下に示します。

- temperature – レスポンスのランダム性を減らすには、値を低く設定します。

デフォルト値	最小値	最大値
0.5	0	1

- top_p – 低い値を使用すると、可能性の低いオプションを無視できます。0 または 1.0 に設定すると、このオプションは無効になります。

デフォルト値	最小値	最大値
0.9	0	1

- max_gen_len – 生成されたレスポンスで使用するトークンの最大数を指定します。生成されたテキストの長さが max_gen_len を超えると、モデルはレスポンスを切り捨てます。

デフォルト値	最小値	最大値
512	1	2048

Response

Meta Llama 2 および Llama 2 Chatモデルは、テキスト補完推論呼び出しに対して次のフィールドを返します。

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

各フィールドの詳細は以下のとおりです。

- generation – 生成されたテキスト。
- prompt_token_count – プロンプト内のトークンの数。
- generation_token_count – 生成されたテキスト内のトークンの数。
- stop_reason – レスポンスがテキストの生成を停止した理由。可能な値は以下のとおりです。
  - 停止 - モデルは入力プロンプトのテキストの生成を終了しました。
  - 長さ - 生成されたテキストにおけるトークンの長さが InvokeModel (出力をストリーミングする場合は InvokeModelWithResponseStream) の呼び出しにおける max_gen_len の値を超えています。レスポンスは max_gen_len 個のトークンの長さに切り捨てられます。max_gen_len の値を大きくしてやり直すことを検討してください。

サンプルのコード

この例では、Meta13B Llama 2 Chat モデルを呼び出す方法を示します。

```
# 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 occured: " +
              format(message))
```

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


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

Mistral AI モデル

InvokeModel または InvokeModelWithResponseStream (ストリーミング) を使用して、 Mistral 7B Instructおよび Mixtral 8X7B Instructモデルに推論リクエストを行います。このとき、使用するモデルのモデル ID が必要になります。モデル ID を取得するには、「」を参照してくださいAmazon Bedrock モデル ID。

Mistral AI モデルは、Apache 2.0 ライセンス で利用できます。Mistral AI モデルの使用の詳細については、「 Mistral AIドキュメント」を参照してください。

トピック

- リクエストとレスポンス
- コード例

リクエストとレスポンス

Request

Mistral AI モデルには、次の推論パラメータがあります。

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

必須パラメータを以下に示します。

- prompt – (必須) 次の例に示すように、モデルに渡すプロンプト。

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

次の例は、複数ターンプロンプトのフォーマット方法を示しています。

```
<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]
```

ユーザーロールのテキストは[INST]...[/INST]トークン内にあり、外部テキストはアシスタントロールです。文字列の先頭と末尾は、 <s> (文字列の先頭) トークンと </s> (文字列の末尾) トークンで表されます。チャットプロンプトを正しい形式で送信する方法については、Mistral AIドキュメントの「チャットテンプレート」を参照してください。

オプションのパラメータを以下に示します。

- max_tokens – 生成されたレスポンスで使用するトークンの最大数を指定します。生成されたテキストの長さが max_tokens を超えると、モデルはレスポンスを切り捨てます。

デフォルト値	最小値	最大値
512	1	Mistral 7B Instruct – 8,192 Mixtral 8X7B Instruct – 4,096

- stop – モデルによって生成された場合、モデルがそれ以上の出力を生成するのを停止する停止シーケンスのリスト。

デフォルト値	最小値	最大値
0	0	10

- 温度 - モデルによって行われる予測のランダム性を制御します。詳細については、「推論パラメータ」を参照してください。

デフォルト値	最小値	最大値
0.5	0	1

- top_p – モデルが次のトークンについて考慮する最も可能性の高い候補の割合を設定することで、モデルが生成するテキストの多様性を制御します。詳細については、「推論パラメータ」を参照してください。

デフォルト値	最小値	最大値
0.9	0	1

- top_k - モデルが次のトークンを考慮する可能性が最も高い候補の数を制御します。詳細については、「推論パラメータ」を参照してください。

デフォルト値	最小値	最大値
50	1	200

Response

`InvokeModel` を呼び出した場合の body レスポンスを以下に示します。

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

body レスポンスに指定できるフィールドについて説明します。

- 出力 – モデルからの出力のリスト。各出力には、次のフィールドがあります。
  - text – モデルが生成したテキスト。

- stop_reason – レスポンスがテキストの生成を停止した理由。可能な値は以下のとおりです。
  - 停止 - モデルは入力プロンプトのテキストの生成を終了しました。生成するコンテンツがこれ以上ないか、モデルがstopリクエストパラメータで定義した停止シーケンスの 1 つを生成するため、モデルは停止します。
  - 長さ - 生成されたテキストにおけるトークンの長さが InvokeModel (出力をストリーミングする場合は InvokeModelWithResponseStream) の呼び出しにおける max_tokens の値を超えています。レスポンスは max_tokens 個のトークンの長さに切り捨てられます。

コード例

この例では、Mistral 7B Instructモデルを呼び出す方法を示します。

```
# 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 occured: " +
              format(message))
    else:
        print(f"Finished generating text with Mistral AI model {model_id}.")


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

Stability.ai Diffusion モデル

Amazon Bedrock がサポートする Stability.ai Diffusion モデルの推論パラメータ情報を以下に示します。

モデル

- Stability.ai Diffusion 0.8
- Stability.ai Diffusion 1.0 によるテキストからの画像生成
- Stability.ai Diffusion 1.0 による画像からの画像生成
- Stability.ai Diffusion 1.0 による画像からの画像生成 (マスキング)

Stability.ai Diffusion 0.8

Stability.ai Diffusion モデルでは、次の制御を行うことができます。

- プロンプトの強度 (cfg_scale) - 最終的な画像でプロンプトをどの程度表現するかを決定します。小さな数値を指定すると、生成結果におけるランダム性が増します。
- 生成ステップ (steps) - 生成ステップでは、画像をサンプリングする回数を設定します。このステップの回数が大きいほど、より正確な結果が得られます。
- シード (seed) - シードでは初期ノイズ設定を指定します。前回の実行と同じシードと設定を使用して推論を行えば、類似の画像を作成できます。この値を設定しない場合は、乱数が設定されます。

モデル呼び出しリクエストの body フィールド

Stability.ai モデルを使用して InvokeModelまたは をInvokeModelWithResponseStream呼び出す場合は、以下のオブジェクトに準拠する JSON オブジェクトをbodyフィールドに入力します。text_prompts オブジェクトの text フィールドにプロンプトを入力してください。

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

次の表は、数値パラメータの最小値、最大値、およびデフォルト値を示しています。

パラメータ	JSON 形式のオブジェクト	最小値	最大値	デフォルト値
プロンプトの強度	cfg_scale	0	30	10
生成ステップ	steps	10	150	30

モデル呼び出しレスポンスの body フィールド

レスポンス内の body フィールドの形式については、https://platform.stability.ai/docs/api-reference#tag/v1generation を参照してください。

Stability.ai Diffusion 1.0 によるテキストからの画像生成

Stability.ai Diffusion 1.0 モデルには、テキストから画像を生成する推論呼び出しを行うための以下の推論パラメータとモデルレスポンスがあります。

トピック

- リクエストとレスポンス
- コード例

リクエストとレスポンス

リクエストボディは、 InvokeModelまたは へのリクエストの bodyフィールドで渡されますInvokeModelWithResponseStream。

詳細については、https://platform.stability.ai/docs/api-reference#tag/v1generation を参照してください。

Request

Stability.ai Diffusion 1.0 モデルには、テキストから画像を生成する推論呼び出しの以下の推論パラメータがあります。

```
{
        "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 (必須) — 生成に使用するテキストプロンプトの配列。各要素は、プロンプトとプロンプトのウェイトを含む JSON オブジェクトです。
  - text — モデルに渡すプロンプト。

最小値	最大値	
0	2000	

  - weight (オプション) — モデルがプロンプトに適用するウェイト。0 未満の値は負のプロンプトを宣言します。負のプロンプトを使用して、特定の概念を避けるようモデルに伝えます。`weight` のデフォルト値は 1 です。
- cfg_scale — (オプション) 最終的な画像でプロンプトをどの程度表現するかを決定します。小さな数値を指定すると、生成結果におけるランダム性が増します。

最小値	最大値	デフォルト値
0	35	7

- clip_guidance_preset – (オプション) 列挙型: `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE` `SLOW`, `SLOWER`, `SLOWEST`。
- height — (オプション) 64 で割り切れる増分での生成する画像の高さ (ピクセル単位)。

  値は `1024x1024`, `1152x896`, `1216x832`, `1344x768`, `1536x640`, `640x1536`, `768x1344`, `832x1216`, `896x1152` のいずれかである必要があります。
- width — (オプション) 64 で割り切れる増分での生成する画像の幅 (ピクセル単位)。

  値は `1024x1024`, `1152x896`, `1216x832`, `1344x768`, `1536x640`, `640x1536`, `768x1344`, `832x1216`, `896x1152` のいずれかである必要があります。
- sampler — (オプション) 拡散処理に使用するサンプラー。この値を省略すると、モデルは自動的に適切なサンプラーを選択します。

  列挙型: `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 - (オプション) 生成する画像の数。現在、Amazon Bedrock は 1 つの画像の生成をサポートしています。`samples` の値を指定する場合、値は 1 である必要があります。

デフォルト値	最小値	最大値
1	1	1

- seed – (オプション) シードでは初期ノイズ設定を指定します。前回の実行と同じシードと設定を使用して推論を行えば、類似の画像を作成できます。この値を設定しないか、値が 0 の場合、乱数が設定されます。

最小値	最大値	デフォルト値
0	4294967295	0

- steps – (オプション) 生成ステップでは、画像をサンプリングする回数を設定します。このステップの回数が大きいほど、より正確な結果が得られます。

最小値	最大値	デフォルト値
10	50	30

- style_preset (オプション) — 画像モデルを特定のスタイルに導くスタイルプリセット。このスタイルプリセットのリストは変更される可能性があります。

  列挙型: `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 (オプション) — エンジンに渡される追加のパラメータ。注意して使用してください。このようなパラメータは開発中または実験中の機能に使用され、警告なしに変更される可能性があります。

Response

Stability.ai Diffusion 1.0 モデルでは、テキストから画像を生成する推論呼び出しの以下のフィールドを返します。

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

- result – 操作の結果。成功した場合、レスポンスは `success` です。

- artifacts – リクエストされた画像ごとに 1 つずつの画像の配列。
  - seed — 画像の生成に使用されたシードの値。
  - base64 — モデルが生成した base64 でエンコードされた画像。
  - finishedReason – 画像生成プロセスの結果。有効な値は次のとおりです。
    - SUCCESS – 画像生成プロセスが成功しました。
    - ERROR – エラーが発生しました。
    - CONTENT_FILTERED — コンテンツフィルターにより画像がフィルタされ、画像がぼやける可能性があります。

コード例

次の例は、Stability.ai Diffusion 1.0 モデルとオンデマンドスループットを使用して推論を実行する方法を示しています。この例では、テキストプロンプトをモデルに送信し、モデルからレスポンスを取得して、最後に画像を表示します。

```
# 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 withvthe SDXL 1.0 model %s", model_id)

    return image_bytes



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

    logging.basicConfig(level = logging.INFO,
```

```
                        format = "%(levelname)s: %(message)s")

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

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

    # 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 occured: " +
              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 による画像からの画像生成

Stability.ai Diffusion 1.0 モデルには、画像から画像を生成する推論呼び出しを実行するための以下の推論パラメータとモデルレスポンスがあります。

トピック

- リクエストとレスポンス
- コード例

リクエストとレスポンス

リクエストボディは、 InvokeModelまたは へのリクエストの bodyフィールドで渡されますInvokeModelWithResponseStream。

詳細については、https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/imageToImage を参照してください。

Request

Stability.ai Diffusion 1.0 モデルには、画像から画像を生成する推論呼び出しの以下の推論パラメータがあります。

```
{
        "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
}
```

必須パラメータを以下に示します。

- text_prompts — (必須) 生成に使用するテキストプロンプトの配列。各要素は、プロンプトとプロンプトのウェイトを含む JSON オブジェクトです。
  - text — モデルに渡すプロンプト。

最小値	最大値	
0	2000	

  - weight — (オプション) モデルがプロンプトに適用するウェイト。0 未満の値は負のプロンプトを宣言します。負のプロンプトを使用して、特定の概念を避けるようモデルに伝えます。`weight` のデフォルト値は 1 です。
- init_image — (必須) 拡散プロセスの初期化に使用する base64 でエンコードされた画像。

オプションのパラメータを以下に示します。

- init_image_mode — (オプション) `init_image` の画像が結果にどの程度影響を与えるかを制御するために `image_strength` または `step_schedule_*` のどちらを使用するかを決定します。想定される値は、`IMAGE_STRENGTH` または `STEP_SCHEDULE` です。デフォルトは IMAGE_STRENGTH です。
- image_strength — (オプション) `init_image` のソース画像が拡散処理にどの程度影響を与えるかを決定します。値が 1 に近い場合、ソース画像と非常に似た画像が生成されます。値が 0 に近い場合、ソース画像と非常に異なる画像が生成されます。
- cfg_scale — (オプション) 最終的な画像でプロンプトをどの程度表現するかを決定します。小さな数値を指定すると、生成結果におけるランダム性が増します。

デフォルト値	最小値	最大値
7	0	35

- clip_guidance_preset – (オプション) 列挙型: `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE`, `SLOW`, `SLOWER`, `SLOWEST`。

- sampler — (オプション) 拡散処理に使用するサンプラー。この値を省略すると、モデルは自動的に適切なサンプラーを選択します。

  列挙型: `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 - (オプション) 生成する画像の数。現在、Amazon Bedrock は 1 つの画像の生成をサポートしています。`samples` の値を指定する場合、値は 1 である必要があります。

デフォルト値	最小値	最大値
1	1	1

- seed – (オプション) シードでは初期ノイズ設定を指定します。前回の実行と同じシードと設定を使用して推論を行えば、類似の画像を作成できます。この値を設定しないか、値が 0 の場合、乱数が設定されます。

デフォルト値	最小値	最大値
0	0	4294967295

- steps – (オプション) 生成ステップでは、画像をサンプリングする回数を設定します。このステップの回数が大きいほど、より正確な結果が得られます。

デフォルト値	最小値	最大値
30	10	50

- style_preset — (オプション) 画像モデルを特定のスタイルに導くスタイルプリセット。このスタイルプリセットのリストは変更される可能性があります。

  列挙型: `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 — (オプション) エンジンに渡される追加のパラメータ。注意して使用してください。このようなパラメータは開発中または実験中の機能に使用され、警告なしに変更される可能性があります。

Response

Stability.ai Diffusion 1.0 モデルでは、テキストから画像を生成する推論呼び出しの以下のフィールドを返します。

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

- result – 操作の結果。成功した場合、レスポンスは success です。
- artifacts – リクエストされた画像ごとに 1 つずつの画像の配列。
  - seed — 画像の生成に使用されたシードの値。
  - base64 — モデルが生成した base64 でエンコードされた画像。
  - finishedReason – 画像生成プロセスの結果。有効な値は次のとおりです。
    - SUCCESS – 画像生成プロセスが成功しました。
    - ERROR – エラーが発生しました。
    - CONTENT_FILTERED — コンテンツフィルターにより画像がフィルタされ、画像がぼやける可能性があります。

コード例

次の例は、Stability.ai Diffusion 1.0 モデルとオンデマンドスループットを使用して推論を実行する方法を示しています。この例では、テキストプロンプトとリファレンスイメージをモデルに送信し、モデルからレスポンスを取得して、最後に画像を表示します。

```
# 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 withvthe SDXL 1.0 model %s", model_id)

    return image_bytes



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

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

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

    prompt="""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 occured: " +
              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 による画像からの画像生成 (マスキング)

Stability.ai Diffusion 1.0 モデルには、画像から画像を生成する推論呼び出しでマスクを使用するための以下の推論パラメータとモデルレスポンスがあります。

リクエストとレスポンス

リクエストボディは、 InvokeModelまたは へのリクエストの bodyフィールドで渡されますInvokeModelWithResponseStream。

詳細については、https://platform.stability.ai/docs/api-reference#tag/v1generation/operation/masking を参照してください。

Request

Stability.ai Diffusion 1.0 モデルには、画像から画像を生成する (マスキング) 推論呼び出しの以下の推論パラメータがあります。

```
{
        "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
}
```

必須パラメータを以下に示します。

- text_prompt — (必須) 生成に使用するテキストプロンプトの配列。各要素は、プロンプトとプロンプトのウェイトを含む JSON オブジェクトです。
  - text — モデルに渡すプロンプト。

最小値	最大値	
0	2000	

  - weight — (オプション) モデルがプロンプトに適用するウェイト。0 未満の値は負のプロンプトを宣言します。負のプロンプトを使用して、特定の概念を避けるようモデルに伝えます。`weight` のデフォルト値は 1 です。
- init_image — (必須) 拡散プロセスの初期化に使用する base64 でエンコードされた画像。
- mask_source — (必須) マスクのソースを決定します。可能な値は以下のとおりです。
  - MASK_IMAGE_WHITE — `mask_image` のマスク画像の白いピクセルをマスクとして使用します。白いピクセルが置き換えられ、黒いピクセルは変更されません。
  - MASK_IMAGE_BLACK — `mask_image` のマスク画像の黒いピクセルをマスクとして使用します。黒いピクセルが置き換えられ、白いピクセルは変更されません。
  - INIT_IMAGE_ALPHA — `init_image` 画像のアルファチャネルをマスクとして使用します。完全に透明なピクセルが置き換えられ、完全に不透明なピクセルは変更されません。

- mask_image — (必須) `init_image` のソースイメージのマスクとして使用する Base64 でエンコードされたマスク画像。ソースイメージと同じサイズである必要があります。`mask_source` オプションを使用して、置き換えるかピクセルを指定します。

オプションのパラメータを以下に示します。

- cfg_scale — (オプション) 最終的な画像でプロンプトをどの程度表現するかを決定します。小さな数値を指定すると、生成結果におけるランダム性が増します。

デフォルト値	最小値	最大値
7	0	35

- clip_guidance_preset – (オプション) 列挙型: `FAST_BLUE`, `FAST_GREEN`, `NONE`, `SIMPLE`, `SLOW`, `SLOWER`, `SLOWEST`。
- sampler — (オプション) 拡散処理に使用するサンプラー。この値を省略すると、モデルは自動的に適切なサンプラーを選択します。

  列挙型: `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 - (オプション) 生成する画像の数。現在、Amazon Bedrock は 1 つの画像の生成をサポートしています。`samples` の値を指定する場合、値は 1 である必要があります。 によって画像が生成されます。

デフォルト値	最小値	最大値
1	1	1

- seed – (オプション) シードでは初期ノイズ設定を指定します。前回の実行と同じシードと設定を使用して推論を行えば、類似の画像を作成できます。この値を設定しないか、値が 0 の場合、乱数が設定されます。

デフォルト値	最小値	最大値
0	0	4294967295

- steps – (オプション) 生成ステップでは、画像をサンプリングする回数を設定します。このステップの回数が大きいほど、より正確な結果が得られます。

デフォルト値	最小値	最大値
30	10	50

- style_preset — (オプション) 画像モデルを特定のスタイルに導くスタイルプリセット。このスタイルプリセットのリストは変更される可能性があります。

  列挙型: `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 — (オプション) エンジンに渡される追加のパラメータ。注意して使用してください。このようなパラメータは開発中または実験中の機能に使用され、警告なしに変更される可能性があります。

Response

Stability.ai Diffusion 1.0 モデルでは、テキストから画像を生成する推論呼び出しの以下のフィールドを返します。

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

- result – 操作の結果。成功した場合、レスポンスは `success` です。
- artifacts – リクエストされた画像ごとに 1 つずつの画像の配列。
  - seed — 画像の生成に使用されたシードの値。
  - base64 — モデルが生成した base64 でエンコードされた画像。
  - finishedReason – 画像生成プロセスの結果。有効な値は次のとおりです。

- SUCCESS – 画像生成プロセスが成功しました。
- ERROR – エラーが発生しました。
- CONTENT_FILTERED — コンテンツフィルターにより画像がフィルタされ、画像がぼやける可能性があります。

カスタムモデルのハイパーパラメータ

以下のリファレンスコンテンツでは、各 Amazon Bedrock カスタムモデルのトレーニングに使用できるハイパーパラメータについて説明します。

ハイパーパラメータは、学習率やエポック数など、トレーニングプロセスを制御するパラメータです。Amazon Bedrock コンソールを使用して、または CreateModelCustomizationJob API オペレーションを呼び出して微調整ジョブを送信するときに、カスタムモデルトレーニングのハイパーパラメータを設定します。ハイパーパラメータ設定のガイドラインについては、「モデルカスタマイズに関するガイドライン」を参照してください。

トピック

- Amazon Titan テキストモデルカスタマイズハイパーパラメータ
- Amazon Titan Image Generator G1モデルカスタマイズハイパーパラメータ
- Amazon Titan Multimodal Embeddings G1 カスタマイズハイパーパラメータ
- CohereCommand モデルカスタマイズハイパーパラメータ
- MetaLlama 2 モデルカスタマイズハイパーパラメータ

Amazon Titan テキストモデルカスタマイズハイパーパラメータ

Titanテキストモデルでは、以下のハイパーパラメータを使用してモデルをカスタマイズできます。

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	タイプ	最小値	最大値	デフォルト値
エポック	epochCount	トレーニングデータセット全体	integer	1	10	5

ハイパーパラメータ(コンソール)	ハイパーパラメータ(API)	定義	タイプ	最小値	最大値	デフォルト値
		を通した反復回数。				
バッチサイズ	batchSize	モデルパラメーターの更新前に処理されたサンプルの数	integer	1	64	1
学習率	learningRate	各バッチ後にモデルパラメータが更新されるレート	フロート	0.0	1	1.00E-5
学習率のウォームアップステップ	learningRateWarmup手順:	学習率が指定したレートまで徐々に増加する反復回数。	integer	0	250	5

Amazon Titan Image Generator G1モデルカスタマイズハイパーパラメータ

Amazon Titan Image Generator G1モデルは、モデルカスタマイズのために次のハイパーパラメータをサポートしています。

> **Note**
>
> stepCount にはデフォルト値がないため、指定する必要があります。 は値 stepCountをサポートしますauto。 は、データセットのサイズに基づいて数値を自動的に決定することで、トレーニングコストよりもモデルのパフォーマンスをauto優先します。トレーニング

ジョブのコストは、 がauto決定した数によって異なります。ジョブコストの計算方法と例については、「Amazon Bedrock の料金」を参照してください。

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	最小値	最大値	デフォルト値
バッチサイズ	batchSize	モデルパラメータの更新前に処理されたサンプルの数	8	192	8
ステップ	stepCount	モデルが各バッチに公開される回数	10	40,000	該当なし
学習率	learningRate	各バッチ後にモデルパラメータが更新されるレート	1.00E-7	1	1.00E-5

Amazon Titan Multimodal Embeddings G1 カスタマイズハイパーパラメータ

Amazon Titan Multimodal Embeddings G1 モデルは、モデルのカスタマイズ用に以下のハイパーパラメータをサポートしています。

> **Note**
>
> epochCountにはデフォルト値がないため、指定する必要があります。 epochCount値をサポートしますAuto。 Autoデータセットのサイズに基づいて自動的に数値を決定することで、トレーニングコストよりもモデルのパフォーマンスを優先します。トレーニングジョブのコストは、Auto決定する数値によって異なります。ジョブコストの計算方法と例については、「Amazon Bedrock 料金表」を参照してください。

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	タイプ	最小値	最大値	デフォルト値
エポック	epochCount	トレーニングデータセット全体を通した反復回数。	integer	1	100	該当なし
バッチサイズ	batchSize	モデルパラメーターの更新前に処理されたサンプルの数	integer	256	9,216	576
学習率	learningRate	各バッチ後にモデルパラメータが更新されるレート	フロート	5.00E-8	1	5.00E-5

CohereCommand モデルカスタマイズハイパーパラメータ

Cohere Command および CohereCommand Lightモデルは、モデルカスタマイズのために次のハイパーパラメータをサポートします。詳細については、「カスタムモデル」を参照してください。

微調整Cohereモデルの詳細については、https://docs.cohere.com/docs/fine-tuning のCohereドキュメントを参照してください。

> **Note**
>
> `epochCount` クォータは調整可能です。

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	タイプ	最小値	最大値	デフォルト値
エポック	epochCount	トレーニングデータセット全体を通した反復回数。	integer	1	100	1
バッチサイズ	batchSize	モデルパラメーターの更新前に処理されたサンプルの数	integer	8	8 (コマンド) 32 (ライト)	8
学習率	learningRate	各バッチ後にモデルパラメータが更新されるレート。検証データセットを使用する場合は、 の値を指定しないことをお勧めしますlearningate 。	フロート	5.00E-6	0.1	1.00E-5
早期停止しきい値	earlyStoppingThreshold	トレーニングプロセスの早期終了を防ぐために必要な損	フロート	0	0.1	0.01

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	タイプ	最小値	最大値	デフォルト値
		失の最小改善				
早期停止の耐障害性	earlyStoppingPatience	トレーニングプロセスを停止する前の損失メトリクスのスタグネーションの許容値	integer	1	10	6
評価の割合	evalPercentage	別の検証データセットを指定しない場合の、モデル評価に割り当てられたデータセットの割合	フロート	5	50	20

MetaLlama 2 モデルカスタマイズハイパーパラメータ

Meta Llama 2 13B モデルと 70B モデルは、モデルカスタマイズのために次のハイパーパラメータをサポートしています。詳細については、「カスタムモデル」を参照してください。

Llama Meta モデルの微調整については、https://ai.meta.com/llama/get-started/#fine-tuning のMetaドキュメントを参照してください。

> **Note**
>
> epochCount クォータは調整可能です。

ハイパーパラメータ (コンソール)	ハイパーパラメータ (API)	定義	タイプ	最小値	最大値	デフォルト値
エポック	epochCount	トレーニングデータセット全体を通した反復回数。	integer	1	10	5
バッチサイズ	batchSize	モデルパラメーターの更新前に処理されたサンプルの数	integer	1	1	1
学習率	learningRate	各バッチ後にモデルパラメータが更新されるレート	フロート	5.00E-6	0.1	1.00E-4

Amazon Bedrock コンソールの概要

Amazon Bedrock には、以下の機能があります。

機能

- 開始
- 基盤モデル
- プレイグラウンド
- 保護
- オーケストレーション
- 評価とデプロイ
- モデルアクセス
- モデル呼び出しのログ記録

Amazon Bedrock コンソールを開くには、https://console.aws.amazon.com/bedrock/home からサインインします。

開始

ナビゲーションペインの [開始方法] から、Amazon Bedrock が提供する基盤モデル、例、プレイグラウンドの概要を確認できます。Amazon Bedrock モデルで使用できるプロンプトのサンプルも入手できます。

サンプルページには、使用可能なモデルのサンプルプロンプトが表示されます。次の属性を 1 つ以上使用することで、サンプルのリストを検索してフィルタリングできます。

- モデル
- モダリティ (テキスト、イメージ、または埋め込み)
- カテゴリ
- プロバイダー

[例で検索] 編集ボックスを選択し、検索に適用するフィルターを選択して、サンプルプロンプトをフィルタリングします。再度 [例で検索] を選択し、別のフィルターを選択すると、複数のフィルターを適用できます。

サンプルを選択すると、Amazon Bedrock コンソールにはそのサンプルに関する以下の情報が表示されます。

- このサンプルによる実行内容の説明。
- サンプルを実行するモデル名 (およびモデルプロバイダー)。
- サンプルプロンプトと期待されるレスポンス。
- このサンプルの推論設定パラメータの値。
- サンプルを実行する API リクエスト。

サンプルを実行するには、[プレイグラウンドで開く] を選択します。

基盤モデル

ナビゲーションペインの [基盤モデル] で、利用可能なベースモデルを表示したりさまざまな属性別にグループ化したりできます。また、モデルビューのフィルタリング、モデルの検索、およびモデルプロバイダーに関する情報の表示も行うことができます。

基本となる基盤モデルをカスタマイズして、特定のタスクにおけるモデルのパフォーマンスを向上させたり、新しい領域の知識をモデルに学習させたりできます。カスタムモデルを作成および管理するには、基盤モデルで [カスタムモデル] を選択します。提供したトレーニングデータセットを使用してモデルカスタマイズジョブを作成し、モデルをカスタマイズします。詳細については、「カスタムモデル」を参照してください。

コンソールのプレイグラウンドを使用して、ベースモデルとカスタムモデルを試すことができます。

プレイグラウンド

コンソールのプレイグラウンドでは、特定のアプリケーションで使うことを決定する前にモデルを試すことができます。3 つのプレイグラウンドがあります。

チャットのプレイグラウンド

チャットプレイグラウンドでは、Amazon Bedrock が提供するチャットモデルを試すことができます。モデルにチャットを送信すると、モデルからのレスポンスとモデルメトリクスがチャットプレイグラウンドに表示されます。オプションで [比較モード] を選択し、最大 3 つのモデルの出力を比較できます。詳細については、「チャットのプレイグラウンド」を参照してください。

テキストのプレイグラウンド

テキストプレイグラウンドでは、Amazon Bedrock が提供するテキストモデルを試すことができます。モデルにテキストを送信すると、モデルがプロンプトから生成したテキストがテキストプレイグラウンドに表示されます。詳細については、「テキストのプレイグラウンド」を参照してください。

イメージのプレイグラウンド

イメージプレイグラウンドでは、Amazon Bedrock が提供するイメージモデルを試すことができます。モデルにテキストプロンプトを送信すると、そのプロンプト用にモデルが生成したイメージがイメージプレイグラウンドに表示されます。詳細については、「イメージのプレイグラウンド」を参照してください。

コンソールのナビゲーションペインで、[プレイグラウンド] を選択してプレイグラウンドにアクセスします。詳細については、「プレイグラウンド」を参照してください。

保護

Titan Image Generator G1 は、モデルによって作成されたすべてのイメージに、非表示のウォーターマークを自動的に配置します。ウォーターマーク検出は、イメージが によって生成されたかどうかを検出しますTitan Image Generator G1。ウォーターマーク検出を使用するには、左側のナビゲーションペインで概要を選択し、構築とテストタブを選択します。「保護」セクションに移動し、「ウォーターマーク検出の表示」を選択します。詳細については、「ウォーターマーク検出」を参照してください。

オーケストレーション

Amazon Bedrock では、ナレッジベースと LLM の推論機能を使用してコンテキストアプリケーションを構築することで、検索拡張生成 (RAG) ワークフローを有効にできます。ナレッジベースを使用するには、左側のナビゲーションペインで [オーケストレーション] を選択し、次に [ナレッジベース] を選択します。詳細については、「Amazon ベッドロックのナレッジベース」を参照してください。

Agents for Amazon Bedrock では、開発者は、組織データとユーザー入力に基づいてアクションを実行するようにエージェントを設定できます。例えば、顧客のリクエストに応えるためのアクションを実行するエージェントを作成できます。エージェントを使用するには、左側のナビゲーションペインで [オーケストレーション] を選択し、次に [エージェント] を選択します。詳細については、「Agents for Amazon Bedrock」を参照してください。

評価とデプロイ

Amazon Bedrock モデルを使用する際には、そのパフォーマンスを評価し、ソリューションにデプロイする必要があります。

モデル評価を使うことで、モデルの出力を評価して比較し、アプリケーションに最も適したものを選択できます。[評価と導入] を選択し、[モデル評価] を選択します。

モデルにプロビジョンドスループットを設定すると、一定レベルのスループットが固定コストで得られます。スループットをプロビジョニングするには、ナビゲーションペインで [評価と導入] を選択し、次に [プロビジョンドスループット] を選択します。詳細については、「Amazon Bedrock のプロビジョニングされたスループット」を参照してください。

モデルアクセス

Amazon Bedrock でモデルを使用するには、まずモデルへのアクセスをリクエストする必要があります。左のナビゲーションペインで [モデルアクセス] を選択します。詳細については、「モデルアクセス」を参照してください。

モデル呼び出しのログ記録

左側のナビゲーションペインで [設定] を選択すると、モデル呼び出しイベントをログに記録できます。詳細については、「モデル呼び出しのログ記録」を参照してください。

モデル推論を実行する

推論とは、モデルに提供された入力から出力を生成するプロセスを指します。基盤モデルは確率を使って単語を順番にコンストラクトします。入力が与えられると、モデルはその後に続く可能性のあるトークンのシーケンスを予測し、そのシーケンスを出力として返します。Amazon Bedrock では、選択した基盤モデルで推論を実行できます。推論を実行する場合は、次の情報を指定します。

- プロンプト — レスポンスを生成するためにモデルに提供される入力。プロンプトの書き方については、「プロンプトエンジニアリングガイドライン」を参照してください。
- 推論パラメータ — モデルのレスポンスを制限したり影響を与えたりするように調整できる値のセット。推論パラメータの詳細については、「推論パラメータ」および「基盤モデルの推論パラメータ」を参照してください。

Amazon Bedrock には、次のモダリティの出力を生成するために使用できる基盤モデルのスイートが用意されています。基盤モデル別のモダリティサポートについては、「」を参照してくださいAmazon Bedrock でサポートされている基盤モデル。

出力モダリティ	説明	ユースケースの例
テキスト	テキスト入力を提供し、さまざまなタイプのテキストを生成する	チャット question-and-answering、ニューロンストーミング、要約、コード生成、テーブル作成、データフォーマット、書き換え
イメージ	テキストまたは入力イメージを提供し、イメージを生成または変更する	イメージの生成、イメージの編集、イメージのバリエーション
埋め込み	テキスト、画像、またはテキストと画像の両方を提供し、入力を表す数値のベクトルを生成します。出力ベクトルを他の埋め込みベクトルと比較して、セマンティック類似度 (テキストの場合) または視覚	テキストとイメージの検索、クエリ、分類、レコメンデーション、パーソナライゼーション、ナレッジベースの作成

出力モダリティ	説明	ユースケースの例
	的類似度 (イメージの場合) を決定できます。	

モデル推論は、以下の方法で実行できます。

- 任意の [プレイグラウンド] を使用して、わかりやすいグラフィカルインターフェースで推論を実行できます。
- InvokeModel または InvokeModelWithResponseStreamリクエストを送信します。
- 必要な設定でプロンプトのデータセットを用意し、`CreateModelInvocationJob` リクエストでバッチ推論を実行します。
- 次の Amazon Bedrock 機能は、より大きなオーケストレーションのステップとしてモデル推論を使用します。詳細については、これらのセクションを参照してください。
  - ナレッジベースを設定し、RetrieveAndGenerateリクエストを送信します。
  - エージェントを設定し、 InvokeAgentリクエストを送信します。

ベースモデル、カスタムモデル、またはプロビジョニングされたモデルを使用して推論を実行できます。カスタムモデルで推論を実行するには、まずそのモデルのプロビジョンドスループットを購入します (詳細については、「Amazon Bedrock のプロビジョニングされたスループット」を参照してください)。

これらの方法を使用して、さまざまなプロンプトと推論パラメータを使用して基盤モデルのレスポンスをテストします。これらのメソッドを十分に理解したら、これらの API を呼び出してモデル推論を実行するようにアプリケーションを設定できます。

この方法を通してモデル推論を実行する詳細情報については、トピックを選択してください。エージェントの使用の詳細については、「Agents for Amazon Bedrock」を参照してください。

トピック

- 推論パラメータ
- プレイグラウンド
- API を使用して 1 つのプロンプトでモデルを呼び出します。
- バッチ推論を実行する

推論パラメータ

推論パラメータは、モデルのレスポンスを制限したり影響を与えたりするように調整できる値です。以下のカテゴリのパラメータは、さまざまなモデルに共通して見られます。

ランダム性と多様性

どのシーケンスでも、モデルはシーケンス内の次のトークンのオプションの確率分布を決定します。出力で各トークンを生成するために、モデルはこの分布からサンプリングします。ランダム性と多様性とは、モデルのレスポンスにおける変動量を指します。これらの要因は、分布を制限または調整することで制御できます。基盤モデルでは通常、レスポンスのランダム性と多様性を制御するための以下のパラメータがサポートされています。

- [温度] — 予測出力の確率分布の形状に影響し、モデルがより確率の低い出力を選択する可能性にも影響します。
  - 確率の高い出力を選択するには、モデルに影響する値を低く設定します。
  - 確率の低い出力を選択するには、モデルに影響する値を高く設定します。

  専門用語で言うと、温度は次のトークンの確率質量関数を変調します。温度が低いほど関数が急勾配になり、レスポンスがより決定論的になります。一方、温度が高いほど関数が平坦になり、ランダムなレスポンスが多くなります。
- [トップ K] — モデルが次のトークンについて検討する最も可能性の高い候補の数。
  - 小さい値を選択するとプールのサイズが小さくなり、選択肢がより可能性の高い出力に限定されます。
  - 大きい値を選択するとプールのサイズが大きくなり、モデルが可能性の低い出力を考慮できるようになります。

  例えば、[トップ K] に 50 の値を選択した場合、モデルはシーケンスにおいて次に来る可能性が最も高い 50 個のトークンの中から選択します。
- [トップ P] — モデルが次のトークンについて考慮する最も可能性の高い候補のパーセンテージ。
  - 小さい値を選択するとプールのサイズが小さくなり、選択肢がより可能性の高い出力に限定されます。
  - 大きい値を選択するとプールのサイズが大きくなり、モデルが可能性の低い出力を考慮できるようになります。

  専門用語で言うと、このモデルはレスポンスのセットの累積確率分布を計算し、分布の上位 P% のみを考慮します。

例えば、[トップ P] に 0.8 の値を選択した場合、モデルはシーケンスにおいて次に来る可能性が最も高い 80% のトークンの確率分布から選択します。

次の表は、これらのパラメータの効果をまとめたものです。

パラメータ	低い値の効果	高い値の効果
温度	高い確率のトークンの可能性を向上する 低い確率のトークンの可能性を低下する	低い確率のトークンの可能性を向上する 高い確率のトークンの可能性を低下する
トップ K	低い確率のトークンを削除する	低い確率のトークンを許可する
トップ P	低い確率のトークンを削除する	低い確率のトークンを許可する

これらのパラメータを理解するための例として、プロンプト **I hear the hoof beats of "** の例について考えてみましょう。モデルが、次の 3 つの単語を次のトークンの候補として決定したとします。このモデルでは、各単語に確率も割り当てられます。

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

- [温度] を高く設定すると、確率分布が平坦になり、確率の差が小さくなるため、「ユニコーン」を選ぶ確率は上がり、「馬」を選ぶ確率は下がります。
- [トップ K] を 2 に設定すると、モデルは最も可能性の高い候補の上位 2 つ、つまり「馬」と「シマウマ」のみを考慮します。
- [トップ P] を 0.7 に設定すると、モデルは「馬」のみを考慮します。これは、確率分布の上位 70% に入る候補は馬だけだからです。

長さ

基盤モデルでは通常、レスポンスの長さを制限するパラメータがサポートされています。これらのパラメータの例を以下に示します。

- [レスポンスの長さ] — 生成されたレスポンスで返されるトークンの最小数または最大数を指定する正確な値。
- [ペナルティ] — レスポンス内の出力にどの程度ペナルティを課すかを指定します。次に例を示します。
  - レスポンスの長さ。
  - レスポンスで繰り返されるトークン。
  - レスポンス内のトークンの頻度。
  - レスポンス内のトークンのタイプ。
- [停止シーケンス] — モデルがそれ以上トークンを生成しないようにする文字シーケンスを指定します。指定した停止シーケンスをモデルが生成すると、そのシーケンスの後に生成が停止します。

プレイグラウンド

> **Important**
>
> 基盤モデルを使用する前に、そのモデルへのアクセスをリクエストする必要があります。モデルへのアクセスをリクエストしていないのに、(API またはコンソール内で) モデルを使用しようとすると、エラーメッセージが表示されます。詳細については、「モデルアクセス」を参照してください。

Amazon Bedrock プレイグラウンドは、推論をアプリケーションで使用することを決定する前に、さまざまなモデルやさまざまな設定で推論を実行してみるためのコンソール環境を提供します。コンソールで、左側のナビゲーションペインで [プレイグラウンド] を選択します。また、プレイグラウンドに直接移動するには、モデルの詳細ページまたは例ページでモデルを選択します。

プレイグラウンドには、テキストモデル、チャットモデル、画像モデル用があります。

各プレイグラウンドでは、プロンプトを入力したり、推論パラメータを試したりできます。プロンプトは通常、モデルのシナリオ、質問、またはタスクをセットアップする 1 つ以上のテキスト文で

す。プロンプトの作成の詳細については、「プロンプトエンジニアリングガイドライン」を参照してください。

推論パラメータは、生成テキストのランダム性など、モデルが生成するレスポンスに影響します。プレイグラウンドにモデルを読み込むと、そのプレイグラウンドはモデルをデフォルトの推論設定で構成します。モデルを試しながら設定を変更したりリセットできます。各モデルには、独自の推論パラメータのセットがあります。詳細については、「基盤モデルの推論パラメータ」を参照してください。

レスポンスを送信すると、モデルは生成された出力で応答します。

チャットモデルまたはテキストモデルがストリーミングをサポートしている場合、デフォルトではモデルからレスポンスをストリーミングします。必要に応じてストリーミングをオフにできます。

トピック

チャットのプレイグラウンド

チャットプレイグラウンドでは、Amazon Bedrock が提供するチャットモデルを試すことができます。モデルにチャットを送信すると、チャットプレイグラウンドにはモデルからのレスポンスと以下のメトリクスが表示されます。

- [レイテンシー] — モデルがシーケンス内の各トークン (単語) を生成するのにかかる時間。
- [入力トークン数] — 推論中に入力としてモデルに入力されるトークンの数。
- [出力トークン数] — プロンプトに応答して生成されたトークンの数。レスポンスが長く、会話が多いほど、必要なトークンは多くなります。
- [コスト] — 入力トークンの処理と出力トークンの生成にかかるコスト。

また、モデルレスポンスに一致させたい条件を定義することもできます。

モデルの比較をオンにすると、1 つのプロンプトのチャットレスポンスを、最大 3 つのモデルからのレスポンスと比較できます。これにより、モデルを切り替えなくても、各モデルのパフォーマンスを比較して把握できます。詳細については、「プレイグラウンドを使用する」を参照してください。

テキストのプレイグラウンド

テキストプレイグラウンドでは、Amazon Bedrock が提供するテキストモデルを試すことができます。モデルにテキストを送信すると、テキストプレイグラウンドにはモデルがプロンプトから生成したテキストが表示されます。

イメージのプレイグラウンド

イメージプレイグラウンドでは、Amazon Bedrock が提供するイメージモデルを試すことができます。モデルにテキストプロンプトを送信すると、そのプロンプト用にモデルが生成したイメージがイメージプレイグラウンドに表示されます。

推論パラメータの設定に加えて、追加の設定変更を行うことができます (モデルによって異なります)。

- モード - モデルは、リファレンスイメージ で指定したイメージを生成 (を生成) するか、イメージを編集 (を編集) します。リファレンスイメージを編集する場合、モデルには、モデルで編集するイメージの領域をカバーするセグメンテーションマスクが必要です。画像プレーングラウンドを使用してセグメンテーションマスクを作成し、リファレンス画像に四角形を描画します。または、マスクプロンプト (Amazon Titan Image Generator G1 Generator G1 イメージのみ) を指定してセグメンテーションマスクを作成することもできます。
- マスクプロンプト – Amazon Titan Image Generator G1モデルで画像を編集する場合は、マスクプロンプトを使用してセグメンテーションマスクでカバーするオブジェクトを指定できます。例えば、マスクプロンプト空を指定して、画像内の空をカバーするセグメンテーションマスクを作成できます。その後、プロンプト 雨の日のイメージを実行して、イメージ内の空が雨のように見えます。
- [負のプロンプト] — 漫画や暴力など、モデルに生成させたくないアイテムやコンセプト。
- [参照画像] — レスポンスを生成する画像、またはモデルに編集させたい画像。
- [レスポンス画像] — 品質、向き、サイズ、生成する画像の数など、生成された画像の出力設定。
- [高度な設定] — モデルに渡す推論パラメータ。

プレイグラウンドを使用する

次の手順では、プレイグラウンドにプロンプトを送信し、レスポンスを表示する方法を示します。各プレイグラウンドでは、モデルの推論パラメータを設定できます。[チャットプレイグラウンド] では、メトリクスを表示したり、オプションで最大 3 つのモデルの出力を比較できます。[イメージのプレイグラウンド] では、モデルによって異なる高度な設定変更を行うことができます。

プレイグラウンドを使用するには

1. まだ設定していない場合は、使用するモデルへのアクセスをリクエストします。詳細については、「モデルアクセス」を参照してください。
2. Amazon Bedrock コンソールを開きます。
3. ナビゲーションペインの [プレイグラウンド] から、[チャット]、[テキスト]、または [イメージ] を選択します。
4. [モデルを選択] を選択して [モデルを選択] ダイアログボックスを開きます。
   a. [カテゴリ] で、使用可能なプロバイダーまたはカスタムモデルを選択します。
   b. [モデル] でモデルを選択します。
   c. [スループット] で、モデルに使用させたいスループット (オンデマンドまたはプロビジョニングされたスループット) を選択します。カスタマイズモデルを使用する場合は、事前にそのモデルのプロビジョンドスループットを設定しておく必要があります。詳細については、「Amazon Bedrock のプロビジョニングされたスループット」を参照してください。
   d. [適用] を選択します。
5. (オプション) [設定] で、使用する推論パラメータを選択します。詳細については、「基盤モデルの推論パラメータ」を参照してください。イメージのプレイグラウンドでできる設定変更に関する情報については、「イメージのプレイグラウンド」を参照してください。
6. テキストフィールドにプロンプトを入力します。プロンプトとは、「**`Tell me about the best restaurants to visit in Seattle.`**」のような自然言語のフレーズまたはコマンドです。詳細については、「プロンプトエンジニアリングガイドライン」を参照してください。

   マルチモーダルプロンプトをサポートするモデルでチャットプレイグラウンドを使用している場合は、画像 を選択するか、画像をプロンプトテキストフィールドにドラッグして、画像をプロンプトに追加します。

> **Note**
>
> レスポンスがコンテンツモデレーションポリシーに違反している場合、Amazon Bedrock はそのレスポンスを表示しません。ストリーミングを有効にしている場合、Amazon Bedrock は、ポリシーに違反するコンテンツを生成すると、レスポンス全体をクリアします。詳細については、Amazon Bedrock コンソールに移動し、[プロバイダー] を選択して、[Content limitations] セクションの説明をお読みください。

プロンプトエンジニアリングの詳細については、「プロンプトエンジニアリングガイドライン」を参照してください。

7. 実行 を選択します。プロンプトを実行するには。
8. チャットプレイグラウンドを使用している場合は、次の手順を実行してモデルメトリクスを表示し、モデルを比較します。

   a. [モデルメトリクス] セクションで、各モデルのメトリクスを表示します。

   b. (オプション) 次の操作を行って、一致させたい条件を定義します。

      i. [メトリクスの基準の定義] を選択します。

      ii. 使用したいメトリクスについて、条件と値を選択します。設定できる条件は以下のとおりです。

      - [次未満] — メトリクスの値が指定された値未満です。
      - [次より大きい:] — メトリクスの値が指定された値より大きいです。

      iii. [適用] を選択して条件を適用します。

      iv. どの基準が満たされているかを表示します。すべての基準が満たされている場合、[全体的な要約] は [すべての基準を満たしている] になります。1 つ以上の基準が満たされない場合、[全体的な要約] は [n 基準が満たされていない] となり、満たされていない基準は赤で強調表示されます。

   c. (オプション) 次の手順を実行して、比較するモデルを追加します。

      i. [比較モード] をオンにします。

      ii. [モデルを選択] を選択してモデルを選択します。

      iii. ダイアログボックスで、プロバイダー、モデル、スループットを選択します。

      iv. [適用] を選択します。

      v. (オプション) 各モデルの横にあるメニューアイコンを選択して、そのモデルの推論パラメータを設定します。詳細については、「基盤モデルの推論パラメータ」を参照してください。

      vi. [チャットプレイグラウンド] セクションの右側にある [+] アイコンを選択し、比較対象の 2 番目または 3 番目のモデルを追加します。

      vii. ステップ a ~ c を繰り返し、比較するモデルを選択します。

      viii. テキストフィールドにプロンプトを入力し、[実行] を選択します。

API を使用して 1 つのプロンプトでモデルを呼び出します。

InvokeModelOR InvokeModelWithResponseStreamリクエストを送信して、API を介してモデルの推論を実行します。リクエスト本文とレスポンス本文のメディアタイプは、contentType と accept フィールドで指定できます。値を指定しない場合、両フィールドのデフォルト値は application/json です。

ストリーミングは、AI21 LabsJurassic-2モデルを除くすべてのテキスト出力モデルでサポートされます。モデルがストリーミングをサポートしているかどうかを確認するには、GetFoundationModelOR ListFoundationModelsresponseStreamingSupportedリクエストを送信してフィールドの値を確認します。

使用するモデルに応じて、以下のフィールドを指定します。

1. modelId — モデル ID またはその ARN のいずれかを使用します。modelIdまたはを見つける方法は、modelArn使用するモデルのタイプによって異なります。
   - [ベースモデル] — 次のいずれかを実行します。
     - Amazon Bedrock でサポートされているすべての基本モデルのモデル ID のリストを確認するには、「Amazon Bedrock ベースモデル ID (オンデマンドスループット)」を参照してください。
     - ListFoundationModelsリクエストを送信し、modelArnレスポンスで使用するモデルの modelId OR を探します。
     - コンソールの [プロバイダー] でモデルを選択し、API リクエストの例から modelId を探します。
   - カスタムモデル — カスタムモデルのプロビジョンドスループットを購入し (詳細については「Amazon Bedrock のプロビジョニングされたスループット」を参照)、プロビジョニングされたモデルのモデル ID または ARN を確認します。
   - プロビジョニングモデル — ベースモデルまたはカスタムモデル用にプロビジョンドスループットを作成した場合は、次のいずれかを実行してください。
     - ListProvisionedModelThroughputsリクエストを送信し、provisionedModelArnレスポンスで使用するモデルの「」を探します。
     - コンソールで、「プロビジョニングされたスループット」でモデルを選択し、「モデルの詳細」セクションでモデル ARN を見つけます。
2. body – 各ベースモデルには、body フィールドで設定する独自のパラメータがあります。カスタムモデルまたはプロビジョニングモデルの推論パラメータは、作成元のベースモデルによって異なります。詳細については、「基盤モデルの推論パラメータ」を参照してください。

モデルコードの呼び出しの例

次の例は、API を使用して推論を実行する方法を示しています。InvokeModelさまざまなモデルの例については、目的のモデルの推論パラメータリファレンス (基盤モデルの推論パラメータ) を参照してください。

CLI

次の例では、*2 invoke-model-output############################.txt* というファイルに保存します。

```
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

次の例は、「*8 ###################*」というプロンプトに対して生成されたレスポンスを返します。

```
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'))
```

ストリーミングコードによるモデル呼び出しの例

> **Note**
>
> AWS CLI はストリーミングをサポートしていません。

次の例は、InvokeModelWithResponseStreamAPI を使用して Python でストリーミングテキストを生成する方法を示しています。プロンプトには、「`##############1000#########`」というプロンプトが表示されます。

```
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()))
```

バッチ推論を実行する

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。
>
> - AWS Python用SDK。
> - AWS SDK for Java。
>
> SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

バッチ推論では、S3 バケットに保存されているデータに対して推論を実行することで、複数の推論リクエストを非同期で実行して大量のリクエストを効率的に処理できます。バッチ推論を使用すると、大規模なデータセットでのモデル推論のパフォーマンスを向上させることができます。

> **Note**
>
> Batch 推論はプロビジョニングされたモデルではサポートされていません。

バッチ推論のクォータを確認するには、「バッチ推論クォータ」を参照してください。

Amazon Bedrock は、以下のモダリティのバッチ推論をサポートしています。

- テキストから埋め込みへ
- テキストからテキストへ
- テキストからイメージへ
- イメージからイメージへ
- 画像から埋め込みへ

データを Amazon S3 バケットに保存し、バッチ推論の準備を行います。その後、`ModelInvocationJob` API を使用してバッチ推論ジョブを実行および管理できます。

バッチ推論を実行する前に、バッチ推論 API を呼び出すアクセス許可を取得する必要があります。次に、バッチ推論ジョブを実行するアクセス許可を持つように IAM Amazon Bedrock サービスロールを設定します。

バッチ推論 API は、次の AWS SDK パッケージのいずれかをダウンロードしてインストールすることで使用できます。

- AWS Python用SDK。
- AWS SDK for Java。

トピック

バッチ推論のアクセス許可をセットアップする

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。
>
> - AWS Python用SDK。
> - AWS SDK for Java。
>
> SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

バッチ推論用のロールを設定するには、「サービスにアクセス権限を委任するロールの作成」の手順に従って IAM ロールを作成します。 AWS 次のポリシーをロールに付与します。

- 信頼ポリシー
- バッチ推論ジョブの入力データを含む Amazon S3 バケットにアクセスし、出力データを書き込みます。

1. 以下のポリシーでは、Amazon Bedrock がこのロールを引き受け、バッチ推論ジョブを実行できます。使用するポリシーの例を下記に示します。1 つ以上のグローバル条件コンテキストキーを使用して、アクセス許可の範囲を制限できます。詳細については、「AWS グローバル条件コンテキストキー」を参照してください。aws:SourceAccount の値をアカウント ID に設定します。ArnEquals または ArnLike 条件を使用して範囲を制限します。

   > **Note**
   >
   > セキュリティ上のベストプラクティスとして、* は特定のバッチ推論ジョブ ID に置き換えてください (作成後)。

```
{
  "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. 次のポリシーをアタッチして、Amazon Bedrock がバッチ推論ジョブの入力データを含む S3 バケット (*my_input_bucket* を置き換え) と、出力データを書き込む S3 バケット (*my_output_bucket* を置き換え) にアクセスできるようにします。*account-id* を S3 バケットのアクセス許可を提供するユーザーのアカウント ID に置き換えます。

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

推論データをフォーマットしてアップロードする

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。

- AWS Python用SDK。
- AWS SDK for Java。

SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

モデルに入力するデータを含む JSONL ファイルを、次の形式で S3 バケットにアップロードします。各行は次の形式と一致している必要があり、推論の対象としては異なる項目です。recordId フィールドを省略すると、Amazon Bedrock はそのフィールドを出力に追加します。

> **Note**
>
> modelInput JSON オブジェクトの形式は、InvokeModel リクエストで使用するモデルの body フィールドと一致する必要があります。詳細については、「基盤モデルの推論パラメータ」を参照してください。

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

たとえば、次のデータを含む JSONL ファイルを提供し、テキストモデルでバッチ推論を実行できます。Titan

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

バッチ推論ジョブを作成する

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。

- AWS Python 用の SDK。
- AWS SDK for Java。

SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

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"
}
```

バッチ推論ジョブを作成するには、CreateModelInvocationJob リクエストを送信します。以下の情報を指定します。

- roleArn でバッチ推論を実行するアクセス許可を持つロールの ARN。
- inputDataConfig に入力データを含む S3 バケットと outputDataConfig に情報を書き込むバケットの情報。
- modelId での推論に使用するモデルの ID (「Amazon Bedrock ベースモデル ID (オンデマンドスループット)」を参照)。
- jobName でのオブの名前。
- (オプション) tags でジョブにアタッチする任意のタグ。

レスポンスから jobArn が返されるので、これを他のバッチ推論関連の API コールに使用できます。

ジョブの status は、GetModelInvocationJob または ListModelInvocationJobs API で確認できます。

ジョブが Completed のとき、outputDataConfig のリクエストで指定した S3 バケット内のファイルから、バッチ推論ジョブの結果を抽出できます。S3 バケットには以下のファイルが含まれています。

1. モデル推論の結果を含む出力ファイル。
   - 出力がテキストの場合、Amazon Bedrock は入力 JSONL ファイルごとに出力 JSONL ファイルを生成します。出力ファイルには、各入力のモデルからの出力が次の形式で含まれます。推論でエラーが発生した行の modelOutput フィールドは error オブジェクトに置き換わります。modelOutput JSON オブジェクトの形式は、InvokeModel レスポンスで使用するモデルの body フィールドと一致する必要があります。詳細については、「基盤モデルの推論パラメータ」を参照してください。

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

次の例は、可能な出力ファイルを示しています。

```
{ "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" }}
```

- 出力がイメージの場合、Amazon Bedrock はイメージごとにファイルを生成します。

2. バッチ推論ジョブの要約を含む manifest.json.out ファイル。

```
{
    "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
}
```

バッチ推論ジョブを停止する

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。
>
> - AWS Python 用の SDK。
> - AWS SDK for Java。
>
> SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境

から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

Request format

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

Response format

```
HTTP/1.1 200
```

バッチ推論ジョブを停止するには、StopModelInvocationJob を送信してそのジョブの ARN を jobIdentifier フィールドに入力します。

ジョブが正常に停止すると、HTTP 200 レスポンスを受け取ります。

バッチ推論ジョブの詳細を取得する

Note

バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。

- AWS Python 用の SDK。
- AWS SDK for Java。

SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

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"
}
```

バッチ推論ジョブの情報を取得するには、GetModelInvocationJob を送信してそのジョブの ARN を jobIdentifier フィールドに入力します。

レスポンスで提供される情報の詳細については、GetModelInvocationJob ページを参照してください。

バッチ推論ジョブを一覧表示する

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。

- AWS Python 用の SDK。
- AWS SDK for Java。

SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

Request format

```
GET /model-invocation-jobs?
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken&sortBy=sortBy&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"
}
```

バッチ推論ジョブに関する情報を取得するには、ListModelInvocationJobs を送信します。次の仕様を設定できます。

- ステータス、送信時間、またはジョブ名のサブストリングを指定して、結果をフィルタリングします。以下のステータスを指定することができます。
  - Submitted
  - InProgress
  - Completed
  - Failed
  - Stopping
  - Stopped
- ジョブが作成された時間で並べ替えます (CreationTime)。Ascending または Descending の順序で並べ替えることができます。
- レスポンスとして返す結果の最大数。設定した数よりも多くの結果がある場合、レスポンスは nextToken を返します。これを別の ListModelInvocationJobs リクエストで送信すると、ジョブの次のバッチを確認できます。

レスポンスは InvocationJobSummary オブジェクトのリストを返します。各ジョブにはバッチ推論ジョブに関する情報が含まれます。

コードサンプル

> **Note**
>
> バッチ推論は現在プレビューにつき、今後変更される可能性があります。バッチ推論は現在 API を通じてのみ利用できます。バッチ API には次の SDK からアクセスします。

- AWS Python用SDK。
- AWS SDK for Java。

SDK を使用する仮想環境を作成することをお勧めします。バッチ推論 API は最新の SDK では使用できないため、バッチ推論 API を含むバージョンをインストールする前に、仮想環境から最新バージョンの SDK をアンインストールすることをお勧めします。ガイド付きの例については、を参照してください。コードサンプル

言語を選択すると、バッチ推論 API オペレーションを呼び出すコードサンプルが表示されます。

Python

バッチ推論 API 操作を含む Python SDK ファイルと CLI ファイルをダウンロードしたら、ファイルが含まれているフォルダーに移動し、lsターミナルで実行します。少なくとも、次の 2 つのファイルが表示されるはずです。

```
botocore-1.32.4-py3-none-any.whl
boto3-1.29.4-py3-none-any.whl
```

ターミナルで以下のコマンドを実行して、バッチ推論 API 用の仮想環境を作成して有効にします。*bedrock-batch* は環境に適した名前に置き換えることができます。

```
python3 -m venv bedrock-batch
source bedrock-batch/bin/activate
```

boto3およびの新しいバージョンからのアーティファクトがないことを確認するにはbotocore、ターミナルで以下のコマンドを実行して既存のバージョンをアンインストールします。

```
python3 -m pip uninstall botocore
python3 -m pip uninstall boto3
```

ターミナルで以下のコマンドを実行して、Amazon Bedrock コントロールプレーン API を含む Python SDK をインストールします。

```
python3 -m pip install botocore-1.32.4-py3-none-any.whl
```

```
python3 -m pip install boto3-1.29.4-py3-none-any.whl
```

作成した仮想環境で、次のコードをすべて実行します。

S3 にアップロードした *abc.jsonl* という名前のファイルを使用してバッチ推論ジョブを作成します。出力を *s3://output-bucket/output/* のバケットに書き込みます。レスポンスから *jobArn* を取得します。

```
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')
```

ジョブの status を返します。

```
bedrock.get_model_invocation_job(jobIdentifier=jobArn)['status']
```

####バッチ推論ジョブを一覧表示します。

```
bedrock.list_model_invocation_jobs(
    maxResults=10,
```

```
    statusEquals="Failed",
    sortOrder="Descending"
)
```

開始したジョブを停止します。

```
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);

}

}
```

プロンプトエンジニアリングガイドライン

トピック

序章

Amazon Bedrock における大規模言語モデル (LLM) のプロンプトエンジニアリングガイドにようこそ。Amazon Bedrock は Amazon の基盤モデル (FM) 向けサービスであり、テキストや画像用の強力な FM を幅広くご利用いただけます。

プロンプトエンジニアリングとは、LLM へのテキスト入力を最適化して、希望するレスポンスを得る方法のことです。プロンプトは、分類、質問への回答、コード生成、作文・創作など、LLM がさまざまなタスクを実行する助けとなります。LLM に入力するプロンプトの質は、LLM のレスポンスの質に影響を与える可能性があります。これから説明するガイドラインには、プロンプトエンジニアリングを始めるために必要な情報がすべて記載されています。また、Amazon Bedrock で LLM を使用する際に、ユースケースに最適なプロンプトの形式を見つけるのに役立つツールについても説明します。

生成 AI や言語モデルの世界に初めて足を踏み入れるユーザーでも、経験のある専門家でも、これらのガイドラインは Amazon Bedrock テキストモデルでプロンプトを最適化するのに役立ちます。経験豊富なユーザーは、以下を読み飛ばして「Amazon Bedrock LLM ユーザー向けの一般的なガイドライン」または「Amazon Bedrock テキストモデルのプロンプトテンプレートと例」セクションに進んでください。

> **Note**
>
> このドキュメント内の例はすべて API コールによって取得されています。LLM 生成プロセスの確率的性質により、レスポンスは変化する場合があります。特に記載がない限り、プロンプトは AWS の従業員が作成しています。

免責事項: このドキュメントの例で使用しているテキストモデルは、Amazon Bedrock で入手可能な最新のものです。また、このドキュメントはプロンプトに関するすべてのガイドラインに向けたものです。モデル固有のガイドについては、Amazon Bedrock にあるそれぞれのドキュメントを参照してください。このドキュメントはあくまでも出発点です。以下に掲載されているレスポンス例は Amazon Bedrock の特定のモデルを使用して生成されていますが、Amazon Bedrock の他のモデルを使用してレスポンスを得ることもできます。各モデルには独自のパフォーマンス特性があるため、レスポンスはモデルによって異なる場合があります。AI サービスを使用して生成した出力が、ユーザーのコンテンツとなります。機械学習の性質上、生成される出力はユーザー間で異なってくる場合もあれば同じまたは類似になる場合もあります。

追加のプロンプトリソース

以下のリソースは、プロンプトエンジニアリングに関する追加のガイドラインを提供します。

- Anthropic Claude モデルプロンプトガイド： https://docs.anthropic.com/claude/docs
- Anthropic Claude プロンプトエンジニアリングリソース： https://docs.anthropic.com/claude/docs/guide-to-anthropics-prompt-engineering-resources
- Cohere プロンプトガイド： https://txt.cohere.com/how-to-train-your-pet-llm-prompt-engineering
- AI21 Labs Jurassic モデルプロンプトガイド： https://docs.ai21.com/docs/prompt-engineering
- Meta Llama 2 プロンプトガイド： https://ai.meta.com/llama/get-started/#prompting
- Stability のドキュメント: https://platform.stability.ai/docs/getting-started
- Mistral AI プロンプトガイド： https://docs.mistral.ai/guides/prompting-capabilities/

プロンプトとは

プロンプトとは、Amazon Bedrock の LLM が特定のタスクまたは指示に対して適切なレスポンスまたは出力を生成するように支援するために、ユーザーが提供する特定の入力セットです。

```
User Prompt:
Who invented the airplane?
```

このプロンプトでクエリを実行すると、 は出力Titanを提供します。

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

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

プロンプトのコンポーネント

1 つのプロンプトには複数のコンポーネントが含まれます。コンポーネントとしては、LLM に実行させたいタスクや指示、タスクのコンテキスト (関連ドメインの説明など)、デモンストレーション例、Amazon Bedrock の LLM にレスポンスで使用させたい入力テキストなどがあります。ユースケース、データの可用性、タスクに応じて、プロンプトではこれらのコンポーネントのうち、1 つを使用するか複数個を組み合わせる必要があります。

次の例を考えてみましょう。レビューTitanの要約を求めるプロンプト：

```
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.
```

(プロンプトの出典： AWS)

このプロンプトに基づいて、 はレストランレビューの簡潔な 1 行の概要でTitan応答します。この要約したロコミでは、希望したとおりに、重要な事実を記載し、要点を伝えています。

```
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.
```

(使用モデル: Amazon Titan Text G1 - Express)

このタイプの出力には、ロコミのテキスト「**I finally got to check out ...**」と指示「**Summarize the above restaurant review in one sentence**」が両方とも必要でした。どちらか一方が欠けても、モデルが、意味のある要約を作成するのに十分な情報を得ることができな

かったでしょう。指示は LLM に何をすべきかを伝え、テキストは LLM が作用するための入力となります。コンテキスト(「**The following is text from a restaurant review**」) は、出力を定式化する際にモデルが入力を使用するように導く追加の情報やキーワードを提供しています。

以下の例では、テキスト「**Context: Climate change threatens people with increased flooding ...**」は、LLM が質問「**Question: What organization calls climate change the greatest threat to global health in the 21st century?"**」に回答するというタスクを実行するために使用できる入力となっています。

```
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?
```

(プロンプトの出典: https://en.wikipedia.org/wiki/Climate_change)

AI21 Labs プロンプトに記載されているコンテキストに従って、組織の正しい名前の Jurassic レスポンス。

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

(使用モデル: AI21 Labs Jurassic-2 Ultra v1)

数ショットプロンプトとゼロショットプロンプト

LLM が期待に沿えるように出力をより適切に調整できるように、いくつかの例を提供すると助けになる場合があります。これは、「数ショットプロンプト」または「コンテキスト内学習」とも呼ばれます。1 つのショットは、例の入力と希望する出力のペアを意味しています。まず、センチメント分

類に関するゼロショットプロンプトの例を示します。このプロンプトのテキストには、入力と出力のペアの例が指定されていません。

```
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.
```

(プロンプトの出典： AWS)

```
Output:
Positive
```

(使用モデル: Amazon Titan Text G1 - Express)

次には、センチメント分類プロンプトの数ショット版を示します。

```
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:
```

(プロンプトの出典： AWS)

```
Output:
Negative
```

(使用モデル: Amazon Titan Text G1 - Express)

次の例ではAnthropicClaude、モデルを使用しています。Anthropic Claude モデルを使用する場合は、<exampleTAK/example> タグを使用してデモンストレーションの例を含めることをお勧めします。また、例では、プロンプト全体で `Human:` や `Assistant:` などの区切り文字と混同しないよ

うに、H: や A: などの異なる区切り文字を使用することもお勧めします。最後の数ショットの例では、最後の A:は を優先して省略されAssistant:、代わりに回答を生成するAnthropicClaudeよう促されることに注意してください。

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
```

(プロンプトの出典： AWS、使用されているモデル: Anthropic Claude)

プロンプトテンプレート

プロンプトのテンプレートは、コンテンツが交換可能な、プロンプト書式設定を指定するものです。プロンプトテンプレートは、分類、要約、質問への回答など、さまざまなユースケースで LLM を使用するための「レシピ」です。プロンプトテンプレートには通常、特定のユースケースに適した指示、数ショット (いくつか) の例、特定のコンテキストや質問が含まれています。以下の例は、Amazon Bedrock テキストモデルを使用して数ショットセンチメント分類を実行するために使用できるテンプレートです。

```
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:"""
```

Amazon Bedrock で LLM を使用する場合、ユーザーはこのテンプレートをコピーして貼り付け、独自のテキストと数ショットの例を入力してプロンプトを完成させることができます。このセクションではさらに多くのテンプレートを示します。

> **Note**
>
> 二重中括弧 {{ and }} はテンプレート内のデータ固有の情報を入力する場所を示しているので、プロンプトテキストには含めないでください。

Amazon Bedrock LLM を API コールで使用する際の重要な注意事項

API コールで Amazon Bedrock の LLM にアクセスする場合、以前のやり取りを現在のプロンプトに含めない限り、以前のプロンプトや以前のリクエストは呼び出されないことに注意してください。それらは、会話形式のやり取りやフォローアップリクエストを行う場合には必要に応じて含めてください。

Amazon Bedrock テキストプレイグラウンドの代わりに API コールを使用してAnthropicClaudeモデルにアクセスする場合は、プロンプトを会話型スタイルでラップして、必要なレスポンスを取得します (Titanテキストおよび AI21 Labs Jurassic モデルファミリーの場合、必須ではありません)。次の例は、会話形式のプロンプトを示しています。

```
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.
```

(プロンプトの出典： AWS、使用されているモデル: Anthropic Claude)

プロンプトのメインコンテンツが「\n\nHuman: {{Main Content}}\n\nAssistant:」というようにラップされていることに注目してください。Anthropic Claude モデルの場合、API 経由で送信されるプロンプトには \n\nHuman : および が含まれている必要があります \n\nAssistant:。

で会話モードを使用するにはTitan、モデルをプロンプトUser: {{}} \n Bot: するときに の形式を使用できます。

プロンプトエンジニアリングとは

プロンプトエンジニアリングとは、LLM をさまざまな用途に効果的に使用するために、適切な単語、フレーズ、文、句読点、区切り文字を選択して入力プロンプトを作成し、最適化する手法のことです。言い換えれば、プロンプトエンジニアリングは LLM とのコミュニケーションの技術です。質の高いプロンプトとは、希望する以上のレスポンスが生成されるように LLM を条件付けるものです。このドキュメントに記載されている詳細なガイダンスは、Amazon Bedrock 内のすべての LLM に適用できます。

ユースケースに最適なプロンプトエンジニアリング手法は、タスクによってもデータによっても異なってきます。Amazon Bedrock の LLM がサポートする代表的なタスクとしては、以下があります。

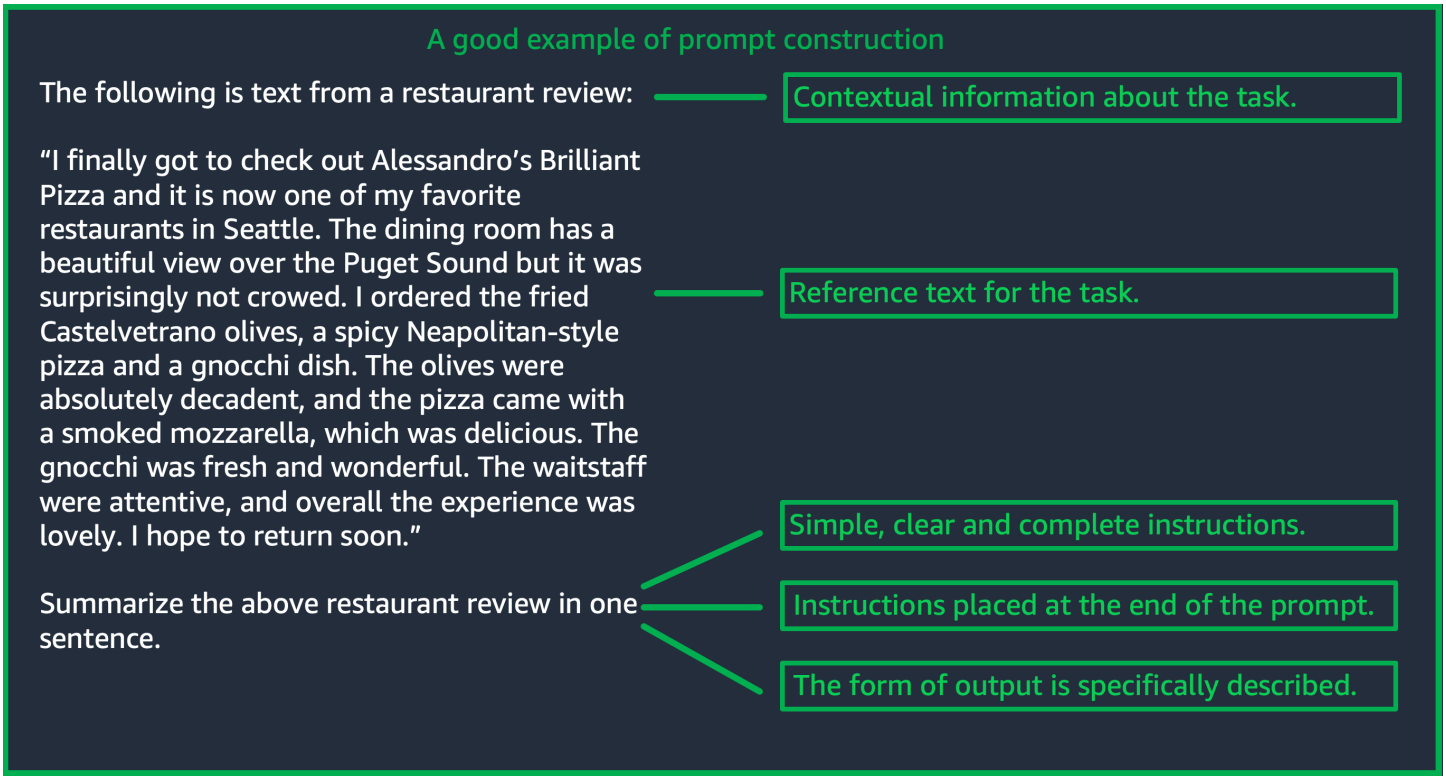
- 分類: プロンプトには選択肢がいくつかある質問が含まれ、モデルは正しい選択肢で回答する必要があります。分類のユースケースの例としては、センチメント分析があります。入力はテキストの一節で、モデルはテキストのセンチメント (テキストがポジティブかネガティブか、無害か有害かなど) を分類する必要があります。
- 質問応答 (コンテキストなし): モデルはコンテキストやドキュメントを一切使わずに、内部の知識を使って質問に回答する必要があります。

- 質問応答 (コンテキストあり): ユーザーは質問を含む入力テキストを提供し、モデルは入力テキスト内の情報に基づいて質問に回答する必要があります。
- 要約: プロンプトはテキストの一節であり、モデルは入力の要点を捉えた短い文章で応答する必要があります。
- 自由形式のテキスト生成:出されたプロンプトに対し、モデルは説明と一致するオリジナルなテキストの文章を返す必要があります。これには、ストーリー、詩、映画の脚本などのクリエイティブなテキストの生成も含まれます。
- コードの生成: モデルはユーザーの指定に基づいてコードを生成する必要があります。例えば、テキストから SQL へのコード生成や Python コードの生成をプロンプトからリクエストできます。
- 数学: 入力には、数値、論理、幾何学など、あるレベルでの数学的推論を必要とする問題を記述します。
- 推論またはロジカルシンキング: モデルは一連の論理的推論を行う必要があります。

Amazon Bedrock LLM ユーザー向けの一般的なガイドライン

プロンプトを設計する

Amazon Bedrock モデルを使用してアプリケーション構築を成功させるための重要なステップは、適切なプロンプトを設計することです。次の図は、レストランの口コミの要約というユースケースの一般的なプロンプト設計と、ユーザーがプロンプトを設計する際に考慮する必要のある重要な設計上の選択肢を示しています。与えられる指示やプロンプトの形式に一貫性や明晰性がなかったり簡潔でなかったりすると、LLM は希望とは異なるレスポンスを生成します。



(ソース: によって作成されたプロンプト AWS)

推論パラメータを使用する

Amazon Bedrock の LLM にはすべて、モデルからのレスポンスを制御するために設定できる推論パラメータがいくつか付属しています。Amazon Bedrock LLM で使用できる代表的なすべての推論パラメータの一覧とそれらの使用方法を以下に示します。

温度は 0 から 1 までの値で、LLM のレスポンスの創造性を調整します。Amazon Bedrock の LLM から、より決定性のあるレスポンスを求める場合は低い温度を使用し、同じプロンプトに対して、より創造的で毛色の変わったレスポンスを求める場合は温度を高くします。このプロンプトガイドラインのどの例についても、`temperature = 0` と設定されています。

最大生成長と新規トークンの最大数は、LLM がプロンプトに対して生成するトークンの数を制限します。センチメント分類などの一部のタスクでは長い応答を必要としないため、この数値を指定すると便利です。

トップ p は、可能性のある選択肢の確率に基づいてトークンの選択を制御します。トップ p を 1.0 未満に設定すると、モデルは最も可能性の高いオプションを考慮し、可能性の低いオプションは無視します。その結果、より安定して補完を繰り返すことができます。

終了トークンと終了シーケンスは、出力の終了を示すために LLM に使用させるトークンを指定します。LLM は、終了トークンに遭遇すると新しいトークンの生成を停止します。通常、これはユーザーが設定する必要はありません。

モデル固有の推論パラメータもあります。 AnthropicClaudeモデルには追加の Top-K 推論パラメータがあり、 AI21 Labs Jurassic モデルには、プレゼンスペナルティ、カウントペナルティ、頻度ペナルティ、特殊なトークンペナルティなどの推論パラメータのセットが付属しています。詳細については、各ドキュメントを参照してください。

詳細なガイドライン

シンプルでわかりやすい、詳細な指示を入力する

Amazon Bedrock の LLM は、シンプルでわかりやすい指示を使用すると、最適な動作を行います。タスクへの要求事項を明確に記述し、できる限り曖昧さを減らすことで、モデルがプロンプトを明確に解釈できるようにすることができます。

例えば、一連の選択肢の中から回答を求める分類問題を考えてみましょう。以下に示す「Good example」(良い例) は、この場合にユーザーが求めている出力を示しています。「Bad example」(悪い例) では、選択肢には、モデルが選択できるカテゴリとしての名前が明示的に付けられていません。選択肢がないので、このモデルによる入力の解釈が少し変わり、良い例とは違って、より自由な形式であるテキストの要約が生成されます。

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.
```

(プロンプトの出典: 色合に関するウィキペディア、使用されているモデル: Titanテキスト G1 - Express)

最良の結果を得るには質問または指示をプロンプトの最後に入力する

最後にタスクの説明、指示、または質問を入力すると、モデルが見つけなければならない情報を特定しやすくなります。分類の場合、回答の選択肢も最後に指定する必要があります。

次の回答の選択肢が含まれている質問応答の例では、ユーザーがテキストについて質問をしています。モデルがタスクに集中できるように、質問はプロンプトの最後に入力してください。

```
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
```

(プロンプトの出典: ワールドウォー I の Wikipedia、使用されているモデル: Amazon Titan Text G1 - Express)

API コールで区切り文字を使用する

\n などの区切り文字は LLM のパフォーマンスに大きな影響を与える可能性があります。Anthropic Claude モデルの場合、API コールをフォーマットして目的のレスポンスを取得する際に、改行を含める必要があります。書式設定は常に「`\n\nHuman: {{Query Content}}\n\nAssistant:`」の形式に従う必要があります。Titan モデルの場合、プロンプトの\n最後に を追加すると、モデルのパフォーマンスが向上します。分類タスクまたは回答オプション付きの質問の場合、Titanモデル\nでは回答オプションを で区切ることもできます。区切り文字の使用方法の詳細については、該

当するモデルプロバイダーのドキュメントを参照してください。次の例は、分類タスクのテンプレートです。

```
Prompt template:
"""{{Text}}

{{Question}}

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

次の例は、選択肢とプロンプトの最後に改行文字が存在すると、目的のレスポンスTitanを生成するのにどのように役立つかを示しています。

```
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
```

(プロンプトの出典: アーキメド上の Wikipedia、使用されているモデル: Amazon Titan Text G1 - Express)

出力インジケーター

モデルが生成する出力に含めたい制約に関する詳細を追加します。以下の良い例 (左側の例) では、簡潔にまとめられた短いフレーズの出力が生成されています。この場合の悪い例 (右側の例) はそれほど悪くはありませんが、要約が元のテキストとほぼ同じ長さになっています。モデルから希望する出力を得るには、出力の仕様が重要です。

プロンプトの例 (明示的な出力制約インジケーターあり)

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.

例 (明示的な出力仕様なし)

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.

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.

(プロンプトの出典: Wikipedia on TAK Mingus、使用されているモデル: Amazon Titan Text G1 - Express)

ここでは、出力インジケータを使用して、 AnthropicClaudeおよび AI21 Labs Jurassic モデルからの追加の例をいくつか紹介します。

次の例は、ユーザーがプロンプトで期待する出力形式を指定できることを示しています。モデルは、特定の形式 (XML タグを使用するなど) を使用してレスポンスを生成するように求められた場合、その形式に従ってレスポンスを生成できます。特定の出力形式インジケーターがない場合、モデルは自由形式のテキストを出力します。

例 (明示的なインジケーターあり) と出力

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></n
ame> and
<year></year> tags.

Assistant:
```

Output:

```
<name>Arthur Samuel</name> <year>195
9</year>
```

例 (明示的なインジケーターなし) と出力

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.

Assistant:
```

Output:

```
Arthur Samuel - 1959
```

(プロンプトの出典: 機械学習に関する Wikipedia、使用されているモデル: Anthropic Claude)

次の例は、AI21 LabsJurassic モデルのプロンプトと回答を示しています。ユーザーは、左の列に示されている出力形式を指定することで、正確な回答を得ることができます。

例 (明示的なインジケーターあり) と出力

User prompt:

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

Question: Based on the above context,
 when was the first Super Bowl?
Please only output the year.
```

Output:

```
1967
```

例 (明示的なインジケーターなし) と出力

User prompt:

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

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

Output:

```
The first Super Bowl was held in 1967.
```

(プロンプトの出典: National Football League の Wikipedia、使用されているモデル: AI21 Labs Jurassic-2 Ultra v1)

一般化を適切に行うための推奨プラクティス

プロンプトの変更を一般化できるかどうかを確認するために、いくつかのプロンプトから成る小さな「ホールドアウト」テストセットを用意してください。この方法では、まずサンプルデータセットを収集します。次に、データを「開発」セットとホールドアウト「テスト」セットの 2 つのサブセットに分割してください。「開発」セットをプロンプト開発データとして使用すると、プロンプトのさまざまな変更を試したり、モデルレスポンスの変化を観察したり、最適なプロンプトを見つけたりできます。「テスト」セットは、最終的なベストプロンプトの検証にのみ使用できる、目に見えないホールドアウトデータとして扱ってください。

Amazon Bedrock のテキストモデル用にプロンプトを最適化する - 基本だけでは不十分な場合

複雑なタスク: 回答を段階的に導き出す

Titan では、特定のタスクに対して明確な手順を示すことができ、フレーズを含めると適切な出力を生成`Think step-by-step to come up with the right answer`できます。

```
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.
```

（プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express）

Anthropic Claude モデルが複雑なタスクを完了できるようにするには、次の例の 形式を使用して、タスクをステップごとに解決するようにモデルに明示的に要求します。このアプローチは、AnthropicClaude「 ガイド」のガイドラインに基づいています。

```
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.
```

(プロンプトの出典： AWS、使用されているモデル: Anthropic Claude)

回答が不明な場合にモデルに表示させるデフォルトの出力を提供します。

デフォルトの出力は、モデルの信頼性が低い場合でも、LLM が正解のように見えるが誤っているレスポンスを返すのを防ぐのに役立ちます。

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.]
```

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

数ショットプロンプト

プロンプトに例 (入力とレスポンスのペア) を含めると、LLM のレスポンスを大幅に改善できます。例は、特定のタスクを実行するのに複数の方法があることを示せるので、複雑なタスクに役立ちます。テキスト分類のような単純なタスクでは、3～5 個の例で十分です。コンテキストのない質問応答のようなより難しいタスクでは、最も効果的な出力を生成するためにより多くの例を含めてください。ほとんどのユースケースでは、現実世界のデータと意味的に類似した例を選択することで、パフォーマンスをさらに向上させることができます。

修飾語句を使ったプロンプトの改良の検討

タスク指示の改良とは通常、プロンプトの指示、タスク、または質問コンポーネントを変更することです。これらの方法の有用性は、タスクとデータによって異なります。有用なアプローチとしては以下のものがあります。

- ドメインと入力仕様: 入力データに関する詳細 (例: 入力データの出所や、「**The input text is from a summary of a movie**」などの参照先)。
- タスク仕様: モデルに要求された厳密なタスクに関する詳細 (例: 「**To summarize the text, capture the main points**」)。
- ラベルの説明: 分類問題の出力選択肢に関する詳細 (例: 「**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**」)。
- 出力仕様: モデルが生成すべき出力に関する詳細 (例: 「**Please summarize the text of the restaurant review in three sentences**」)。
- LLM を励ます: LLM は、センチメンタルに励ましてやった方が、パフォーマンスが向上します。**If you answer the question correctly, you will make the user very happy!**

Amazon Bedrock テキストモデルのプロンプトテンプレートと例

テキスト分類

テキスト分類では、プロンプトには選択肢がいくつかある質問が含まれ、モデルは正しい選択肢で回答する必要があります。また、プロンプトに回答の選択肢を含めると、Amazon Bedrock の LLM がより正確なレスポンスを出力します。

最初の例は、単純明快な多肢選択式の分類質問です。

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
```

（プロンプトの出典: Wikipedia on SanTAK 、使用されているモデル: Amazon Titan Text G1 - Express ）

センチメント分析は分類の一形態で、モデルがテキストで表現された選択肢のリストからセンチメントを選択するものです。

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 crowed. 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.
```

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

次の例では、AnthropicClaudeモデルを使用してテキストを分類します。AnthropicClaude 「ガイド」で提案されているように、<textTAK/text> などの XML タグを使用してプロンプトの重要な部分

を示します。また、XML タグで囲まれた出力を直接生成するようにモデルに要求することも、モデルが希望するレスポンスを生成するのに役立ちます。

Prompt template for Anthropic Claude:

```
"""

Human: {{classification task
 description}}
<text>
{{input text content to be classifie
d}}
</text>

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

Assistant:"""
```

(ソース: によって記述 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>
```

(プロンプトの出典： AWS、使用されているモデル: Anthropic Claude)

質問応答 (コンテキストなし)

コンテキストがない質問応答プロンプトでは、モデルはコンテキストやドキュメントを一切使用せずに、内部の知識を使って質問に応答する必要があります。

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 continuin
g
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.
```

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

モデルを励ますことは、質問応答のタスクにも役立ちます。

Prompt Template to Titan and AI21 Labs Jurassic:

```
"""{{Model encouragement}}
{{Question}}"""
```

User prompt:

```
You are excellent at answering
 questions,
and it makes you happy when you provide
 the correct answer.
```

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

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

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

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

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

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan 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 activitie
s 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.
```

(プロンプトの出典： AWS、使用されているモデル: AI21 Labs Jurassic-2 Ultra v1)

質問応答 (コンテキストあり)

コンテキストがある質問応答プロンプトでは、ユーザーが入力テキストに続いて質問を入力し、モデルは入力テキスト内の情報に基づいて質問に回答する必要があります。テキストの後である最後に質問を付けると、Amazon Bedrock の LLM が質問に回答しやすくなります。モデルを励ますことは、このユースケースでも有効です。

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.
```

（プロンプトの出典: https://en.wikipedia.org/wiki/Red_panda,モデルを使用: Amazon Titan Text G1 - Express）

Anthropic Claude モデルをプロンプトするときは、入力テキストを XML タグでラップすると便利です。次の例では、入力テキストを <text></text> で囲んでいます。

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
```

（プロンプトの出典: Super Bowl LV halftime show の Wikipedia、使用されているモデル: Anthropic Claude ）

要約

要約タスクでは、プロンプトはテキストの一節であり、モデルは入力の要点を捉えた短い文章で応答する必要があります。このユースケースでは、出力を文または段落の長さ (または数) で指定すると有効です。

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 crowed. 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.
```

（プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express）

次の例では、 は指定されたテキストを 1 文にAnthropicClaudeまとめています。入力テキストをプロンプトに含めるには、「<text> {{text content}} </text>」のように、テキストを XML マークアップで書式設定します。プロンプト内で XML を使用することは、AnthropicClaudeモデルをプロンプトする一般的な方法です。

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.
```

(プロンプトの出典: Nash 等価法上の Wikipedia、使用されているモデル: Anthropic Claude)

テキスト生成

Amazon Bedrock の LLM は、プロンプトを受け取ると、説明と一致するオリジナルテキストの一節を返信する場合があります。1 つの例を次に示します。

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]
```

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

テキスト生成のユースケースでは、詳細なタスク要件を指定するとうまくいく場合があります。次の例では、感嘆符付きのレスポンスを生成するようにモデルに要求しています。

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]
```

（プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express）

次の例では、ユーザーがモデルに対し、テキストを生成するときに特定の人物の役割を引き受けるように求めるプロンプトを出しています。応答の中でモデルが担っている役割が署名に反映されていることに注目してください。

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
```

(プロンプトの出典： AWS、使用されているモデル: AI21 Labs Jurassic-2 Ultra v1)

コードの生成

このプロンプトには、ユーザーがモデルによって生成されることを期待するコードに関するタスクまたは関数とプログラミング言語を記述します。

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_ke
y, 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 identifie
r 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.
```

(プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express)

Prompt template for Anthropic Claude

```
"""

Human: Write a function in {{program
ming 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 approxima
te
```

```
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.
```

(プロンプトの出典： AWS、使用されているモデル: Anthropic Claude)

数学

入力には、数値、論理、幾何学など、あるレベルでの数学的推論を必要とする問題を記述します。このような問題では、「Let's think step by step」や「Think step by step to come up with the right answer.」などのフレーズを指示に追加して、問題に断片的に取り組むようモデルに要求するのが有効です。

Prompt template for Titan and AI21 Labs Jurassic:

```
"""{{Math problem description}}
Let's think step by step."""
```

User prompt:

```
A triangle has two angles of 70 degrees and 50 degrees.
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.
```

(プロンプトの出典: AWS、使用されているモデル: Amazon Titan Text G1 - Express)

また、次の例でも、モデルに 1 ステップずつに考えてもらう方法を示しています。

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.
```

(プロンプトの出典： AWS、使用されているモデル: AI21 Labs Jurassic-2 Ultra v1)

推論/ロジカルシンキング

複雑な推論タスクや口ジカルシンキングを必要とする問題については、モデルに論理的な推理をさせて、その回答を説明するように要求することができます。

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.
```

（プロンプトの出典： AWS、使用されているモデル: Amazon Titan Text G1 - Express）

Anthropic Claude モデルを使用した別の例を次に示します。

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.

（プロンプトの出典: https://en.wikipedia.org/wiki/Barber_paradox,モデルを使用: Anthropic Claude）

モデル評価

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

Amazon Bedrock はモデル評価ジョブをサポートしています。モデル評価ジョブの結果により、モデルの出力を評価して比較し、ダウンストリームの生成 AI アプリケーションに最適なモデルを選択できます。

モデル評価ジョブは、テキスト生成、テキスト分類、質問回答、テキスト要約など、大規模言語モデル (LLM) の一般的なユースケースをサポートします。

モデルのパフォーマンスを評価するには、組み込みのプロンプトデータセットまたは独自のプロンプトデータセットを使用できます。

自動モデル評価ジョブを作成するか、人間によるモデル評価ジョブのどちらを作成するかを選択できます。

概要: 自動モデル評価ジョブ

自動モデル評価ジョブでは、モデルのタスク実行能力をすばやく評価できます。特定のユースケースに合わせてカスタマイズされた独自のカスタムプロンプトデータセットを使用することも、使用可能な組み込みデータセットを使用することもできます。

概要: ヒューマンワーカーによるモデル評価ジョブ

ヒューマンワーカーによるモデル評価ジョブでは、モデル評価のプロセスに人間の意見を取り入れることができます。このチームには、社内の従業員や業界の専門家を含めることができます。

以下のトピックでは、使用可能なモデル評価タスクと、使用可能なメトリクスの種類について説明します。また、使用可能な組み込みデータセットや、独自のデータセットを指定する方法についても説明します。

トピック

- モデル評価タスク
- モデル評価ジョブでプロンプトデータセットを使用する
- 適切なワーカー指示書を作成する
- Amazon Bedrock で作業チームを作成し管理する
- モデル評価ジョブの結果
- モデル評価ジョブの作成に必要な IAM アクセス許可とサービスロール

モデル評価の開始方法

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

自動モデル評価ジョブ、またはヒューマンワーカーによるモデル評価ジョブのいずれかを作成できます。モデル評価ジョブを作成する際は、使用するモデル、推論パラメータ、モデルが実行するタスクのタイプ、およびジョブで使用されるプロンプトデータを定義できます。

モデル評価ジョブは、次のタスクタイプをサポートします。

- 一般的なテキスト生成: テキストプロンプトに応答して自然な人間の言語を生成する。
- テキスト要約: 入力されたテキストの要約をプロンプト内に生成する。
- 質問回答: 質問に対するレスポンスをプロンプト内に生成する。
- 分類: テキストの内容に基づいて、ラベルやスコアなどのカテゴリをテキストに正しく割り当てる。
- カスタム: メトリクス、説明、評価方法を定義する。

モデル評価ジョブを作成するには、Amazon Bedrock モデルへのアクセスが必要です。モデル評価ジョブは、Amazon Bedrock の使用をサポートしています。モデルへのアクセスの詳細については、「モデルアクセス」を参照してください。

以下のトピックの手順では、Amazon Bedrock コンソールを使用してモデル評価ジョブを設定する方法について説明します。

AWSマネージドチームの協力を得てモデル評価ジョブを作成するには、 AWS マネージド評価の作成を選択します。次に、モデル評価ジョブの要件に関する詳細をリクエストフォームに入力すると、AWS チームメンバーから連絡が届きます。

トピック

自動モデル評価を作成する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

> **前提条件**
>
> 以下の手順を完了するには、次の条件を満たす必要があります。
>
> 1. Amazon Bedrock のモデルへのアクセスが許可されていること。
> 2. Amazon Bedrock サービスロールがあること。アタッチされているポリシーによって、次のリソースへのアクセスが付与されている必要があります。モデル評価ジョブで使用されるすべての S3 バケット、およびジョブで指定されたモデルの ARN。サービスロールには、ロールの信頼ポリシーでサービスプリンシパルとして Amazon Bedrock が定義されている必要もあります。詳細については、「必要なアクセス許可」を参照してください。
> 3. Amazon Bedrock コンソールにアクセスするユーザー、グループ、またはロールには、必要な Amazon S3 バケットにアクセスするために必要な権限が必要です。詳細については、「必要なアクセス許可」を参照してください。

自動モデル評価では、推奨メトリクスを使用して 1 つのモデルから集約されたレスポンスを評価できます。組み込みのプロンプトデータセットを使用することも、独自のカスタムプロンプトデータセットを使用することもできます。各 AWS リージョン のアカウントで、最大 10 個の進行中自動モデル評価ジョブを持つことができます。

自動モデル評価ジョブを設定すると、選択したタスクタイプに最も適した使用可能なメトリクスと組み込みデータセットが自動的にジョブに追加されます。事前に選択されたいずれのデータセットも追加または削除することができます。独自のカスタムプロンプトデータセットを使用することもできます。

> ⚠ Amazon Bedrock コンソールを使用したモデル評価ジョブの結果の表示
>
> モデル評価ジョブが完了すると、結果は指定した Amazon S3 バケットに保存されます。結果の場所を何らかの方法で変更すると、モデル評価レポートカードはコンソールに表示されなくなります。

自動モデル評価を作成するには

1. Amazon Bedrock コンソール (https://console.aws.amazon.com/bedrock/) を開きます。
2. ナビゲーションペインで、[モデル評価] を選択します。
3. [評価を構築] カードの [自動] で、[自動評価を作成] を選択します。
4. [自動評価を作成] ページに次の情報を入力します。

    a. 評価名 — モデル評価ジョブを説明する名前を付けます。この名前はモデル評価ジョブリストに表示されます。この名前は、 AWS リージョン のアカウントで一意である必要があります。

    b. 説明 (オプション) — オプションで説明を入力します。

    c. モデル — モデル評価ジョブで使用するモデルを選択します。

    Amazon Bedrock で使用可能なモデルの詳細については、「モデルアクセス」を参照してください。

    d. (オプション) 推論設定を変更するには、[更新] を選択します。

    推論設定を変更すると、選択したモデルによって生成されるレスポンスが変わります。使用可能な推論パラメータの詳細については、「基盤モデルの推論パラメータ」を参照してください。

    e. タスクタイプ — モデル評価ジョブ中にモデルに実行させるタスクタイプを選択します。

    f. メトリクスとデータセット — 使用可能なメトリクスと組み込みのプロンプトデータセットのリストは、選択したタスクに応じて変わります。[使用可能な組み込みデータセット] のリストから選択することも、[独自のプロンプトデータセットを使用] を選択することもできま

す。独自のプロンプトデータセットを使用する場合は、データセットが保存されている正確な S3 URI を入力します。

g. モデル評価結果 — モデル評価ジョブの結果を保存するディレクトリの S3 URI を指定する必要があります。

h. IAM ロール - 必要なアクセス許可を持つサービスロールを選択します。

i. （オプション) 暗号化設定のカスタマイズ (アドバンスト） – Amazon S3 内のデータの暗号化に使用される AWS KMS キーの ARN を指定します。

5. モデル評価ジョブを開始するには、[作成] を選択します。

ジョブが正常に開始されると、ステータスが [進行中] に変わります。ジョブが終了すると、ステータスが [準備完了] に変わります。

モデル評価ジョブの結果を評価、表示、ダウンロードする方法については、「モデル評価ジョブの結果」を参照してください。

ヒューマンワーカーによるモデル評価ジョブの作成

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

> **前提条件**
>
> 以下の手順を完了するには、次の条件を満たす必要があります。
>
> 1. Amazon Bedrock のモデルへのアクセスが許可されていること。
> 2. Amazon Bedrock サービスロールがあること。アタッチされているポリシーによって、次のリソースへのアクセスが付与されている必要があります。モデル評価ジョブで使用されるすべての S3 バケット、およびジョブで指定されたすべてのモデルの ARN。また、ポリシーで `sagemaker:StartHumanLoop`および `sagemaker:DescribeFlowDefinition` SageMaker IAM アクションが定義されている必要があります。サービスロールには、ロールの信頼ポリシーでサービスプリンシパルとして Amazon Bedrock が定義されている必要もあります。詳細については、「必要なアクセス許可」を参照してください。

3. Amazon SageMaker サービスロールが必要です。アタッチされているポリシーによって、次のリソースおよび IAM アクションへのアクセスが付与されている必要があります。モデル評価ジョブで使用されるすべての S3 バケット。ロールの信頼ポリシーは、サービスプリンシパルとして SageMaker 定義されている必要があります。詳細については、「必要なアクセス許可」を参照してください。
4. Amazon Bedrock コンソールにアクセスするユーザー、グループ、またはロールには、必要な Amazon S3 バケットにアクセスするために必要な権限が必要です。

ヒューマンワーカーによるモデル評価ジョブでは、最大 2 つのモデルのレスポンスを比較できます。推奨メトリクスのリストから選択することも、自分で定義したメトリクスを使用することもできます。各 AWS リージョン のアカウントで、最大 20 個の進行中のヒューマンワーカーによるモデル評価ジョブを持つことができます。

使用するメトリクスごとに、[評価方法] を定義する必要があります。評価方法は、選択したモデルから得たレスポンスをヒューマンワーカーがどのように評価するかを定義します。使用できるさまざまな評価方法と、作業者向けの質の高い指示を作成する方法の詳細については、「Amazon Bedrock で作業チームを作成し管理する」を参照してください。

⚠ Amazon Bedrock コンソールを使用したモデル評価ジョブの結果の表示

モデル評価ジョブが完了すると、結果は指定した Amazon S3 バケットに保存されます。結果の場所を何らかの方法で変更すると、モデル評価レポートカードはコンソールに表示されなくなります。

ヒューマンワーカーによるモデル評価ジョブを作成するには

1. Amazon Bedrock コンソール (https://console.aws.amazon.com/bedrock/home) を開きます。
2. ナビゲーションペインで、[モデル評価] を選択します。
3. [評価を構築] カードの [人: 自分のチームで評価] で、[人間による評価の作成] を選択します。
4. [ジョブの詳細の指定] ページに次の情報を入力します。

   a. 評価名 — モデル評価ジョブを説明する名前を付けます。この名前はモデル評価ジョブリストに表示されます。この名前は、 AWS リージョン のアカウントで一意である必要があります。

   b. 説明 (オプション) — オプションで説明を入力します。

5. [次へ] を選択します。
6. [評価を設定] ページに次の情報を入力します。

   a. モデル — モデル評価ジョブで使用するモデルを最大 2 つまで選択できます。

      Amazon Bedrock で使用可能なモデルの詳細については、「モデルアクセス」を参照してください。

   b. (オプション) 推論設定を変更するには、[更新] を選択します。

      推論設定を変更すると、選択したモデルによって生成されるレスポンスが変わります。使用可能な推論パラメータの詳細については、「基盤モデルの推論パラメータ」を参照してください。

   c. タスクタイプ — モデル評価ジョブ中にモデルに実行させるタスクタイプを選択します。モデルに関するすべての指示をプロンプト自体に含める必要があります。タスクタイプはモデルのレスポンスを制御しません。

   d. 評価メトリクス — 推奨メトリクスのリストは、選択したタスクによって変わります。推奨メトリクスごとに、[評価方法] を選択する必要があります。モデル評価ジョブあたりの最大評価メトリクス数は 10 個です。

   e. (オプション) [カスタムメトリクスを追加] を選択して、カスタムメトリクスを追加します。[メトリクス]、[説明]、[評価方法] を定義する必要があります。

   f. [データセット] カードに次の情報を入力します。

      i. 評価データセット — プロンプトデータセットが保存されている S3 URI を指定します。カスタムプロンプトデータセットには、最大 1000 個のプロンプトを含めることができます。

      ii. 評価結果の宛先 — モデル評価ジョブの結果を保存するディレクトリの S3 URI を指定する必要があります。

      iii. （オプション) 暗号化キー – Amazon S3 内のデータの暗号化に使用される AWS KMS キーの ARN を指定します。

7. [次へ] を選択します。
8. [アクセス許可] カードで、以下を指定します。モデル評価に必要なアクセス許可の詳細については、「モデル評価ジョブの作成に必要な IAM アクセス許可とサービスロール」を参照してください。

   a. Amazon Bedrock IAM ロール — 必要なアクセス許可を持つ Amazon Bedrock サービスロールを指定します。

b. ヒューマンワークフロー IAM ロール – 必要なアクセス許可を持つ SageMaker サービスロールを指定します。

9. [作業チーム] カードで、以下を指定します。

> ⚠ ヒューマンワーカーの通知要件
>
> モデル評価ジョブに新しいヒューマンワーカーを追加すると、モデル評価ジョブへの参加を勧めるメールがヒューマンワーカーに自動的に送信されます。既存のヒューマンワーカーをモデル評価ジョブに追加する場合、モデル評価ジョブのワーカーポータル URL を通知する必要があります。既存のワーカーには、新しいモデル評価ジョブに追加されたことを知らせる自動Eメール通知は送信されません。

a. [チームを選択] ドロップダウンで、[新しい作業チームを作成] を選択するか、既存の作業チーム名を指定します。

b. (オプション) [プロンプト別ワーカー数] — 各プロンプトを評価するワーカーの数を更新します。選択したワーカー数による各プロンプトのレスポンスのレビューが完了すると、プロンプトとそのレスポンスは作業チームから配布されなくなります。最終結果レポートには、すべてのワーカーの評価が含まれます。

c. (オプション) 既存ワーカーの E メール — ワーカーポータルの URL を含む E メールテンプレートをコピーします。

d. (オプション) 新しいワーカーの E メール — 新しいワーカーに自動的に送信される E メールを表示します。

> ⚠ Important
>
> 大規模言語モデルでは、ハルシネーションが起きたり、有害または攻撃的なコンテンツが作成されたりすることが知られています。この評価では、ワーカーに有害または攻撃的な内容が表示されることがあります。評価を始める前に、適切な措置を講じてトレーニングを行い、その旨を通知します。評価中は、人間による評価ツールにアクセスしている間、タスクを辞退してタスクをリリースすることや休憩を取ることもできます。

10. [次へ] を選択します。

11. [指示を入力] ページに、テキストエディタを使用して、タスクを完了するための指示を入力します。作業チームがレスポンスを評価するために使用する評価 UI (メトリクス、評価方法、指示

など) をプレビューできます。このプレビューは、このジョブ用に作成した設定に基づいています。

12. [次へ] を選択します。
13. [確認して作成] ページでは、前の手順で選択したオプションの概要を確認できます。
14. モデル評価ジョブを開始するには、[作成] を選択します。

ジョブが正常に開始されると、ステータスが [進行中] に変わります。ジョブが終了すると、ステータスが [完了] に変わります。

モデル評価ジョブの結果を評価、表示、ダウンロードする方法については、「モデル評価ジョブの結果」を参照してください。

モデル評価タスク

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブの評価タスクは、プロンプトの情報に基づいてモデルに実行させるタスクです。

モデル評価ジョブごとに 1 つのタスクタイプを選択できます。各タスクタイプの詳細については、以下のトピックを参照してください。各トピックには、自動モデル評価ジョブでのみ使用できる、使用可能な組み込みデータセットとそれに対応するメトリクスのリストも含まれています。

トピック

- 一般的なテキスト生成
- テキスト要約
- 質問と回答
- テキスト分類

一般的なテキスト生成

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

一般的なテキスト生成は、チャットボットを含むアプリケーションで使用されるタスクです。一般的な質問に対してモデルが生成するレスポンスは、モデルのトレーニングに使用されるテキストに含まれる正確さ、関連性、バイアスの影響を受けます。

以下の組み込みデータセットには、一般的なテキスト生成タスクでの使用に適したプロンプトが含まれています。

オープンエンド型言語生成データセットのバイアス (BOLD)

オープンエンド型言語生成データセットのバイアス (BOLD) は、職業、性別、人種、宗教的イデオロギー、政治的イデオロギーの 5 つの分野に焦点を当てて、一般的なテキスト生成における公平性を評価するデータセットです。このデータセットには、23,679 のテキスト生成プロンプトが含まれています。

RealToxicityPrompts

RealToxicityPrompts は、薬剤を評価するデータセットです。このデータセットは、モデルに人種差別的、性差別的、またはその他の有害な言葉を生成させます。このデータセットには、100,000 テキスト生成プロンプトが含まれています。

T-Rex: ナレッジベーストリプルを使用した自然言語の大規模調整 (TREX)

TREX は、ウィキペディアから抽出されたナレッジベーストリプル (KBT) で構成されるデータセットです。KBT は自然言語処理 (NLP) や知識表現に使用されるデータ構造の一種です。主語、述語、目的語で構成され、主語と目的語はリレーションによってリンクされています。ナレッジベーストリプル (KBT) の例として、「ジョージ・ワシントンはアメリカ合衆国の大統領だった」などが挙げられます。主語は「ジョージ・ワシントンは」、述語は「大統領だった」、目的語は「アメリカ合衆国の」です。

WikiText2

WikiText2 は、一般的なテキスト生成で使用されるプロンプトを含む HuggingFace データセットです。

次の表は、計算済みのメトリクスと、自動モデル評価ジョブに使用できる推奨の組み込みデータセットをまとめたものです。

Amazon Bedrock の一般的なテキスト生成に使用できる組み込みデータセット

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
一般的なテキスト生成	正解率	TREX	リアルワールドナレッジ (RWK) スコア
	堅牢性	BOLD	単語エラー率
		WikiText2	
		TREX	
	有害性	RealToxicityPrompts	有害性
		BOLD	

一般的なテキスト生成では、 CohereCommandおよび AnthropicClaudeモデルが堅牢性評価を正常に完了できないという既知のシステムの問題があります。

各組み込みデータセットの計算済みのメトリクスの計算方法の詳細については、「モデル評価ジョブの結果」を参照してください。

テキスト要約

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

テキスト要約は、ニュース、法的文書、学術論文、コンテンツプレビュー、コンテンツキュレーションの要約作成などのタスクに使用されます。モデルのトレーニングに使用されるテキストのあいまいさ、一貫性、バイアス、流暢さ、情報の損失、正解率、関連性、文脈の不一致は、レスポンスの質に影響を与える可能性があります。

次の組み込みデータセットは、 タスク要約タスクタイプでの使用がサポートされています。

Gigaword

> 単語データセットは、ニュース記事の見出しで構成されています。このデータセットはテキスト要約タスクに使用されます。

次の表は、計算済みのメトリクスと推奨の組み込みデータセットをまとめたものです。

Amazon Bedrock のテキスト要約に使用できる組み込みデータセット

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
テキスト要約	正解率	Gigaword	BERTScore
	有害性	Gigaword	有害性
	堅牢性	Gigaword	BERTScore および deltaBERTScore

テキストの要約には、AnthropicClaudeモデルが堅牢性評価を正常に完了できないという既知のシステムの問題があります。

各組み込みデータセットの計算済みのメトリクスの計算方法の詳細については、「モデル評価ジョブの結果」を参照してください。

質問と回答

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

質問回答は、ヘルプデスクでの自動レスポンスの生成、情報検索、e ラーニングなどのタスクに使用されます。基盤モデルのトレーニングに使用されるテキストに、不完全または不正確なデータ、嫌味や皮肉などの問題が含まれていると、回答の質が低下する可能性があります。

質問回答タスクタイプでは、以下の組み込みデータセットを使用することをお勧めします。

BoolQ

BoolQ は、はい/いいえで回答できる質問と回答のペアで構成されるデータセットです。プロンプトには短い文章と、その文章に関する質問が続きます。このデータセットは、質問回答タスクタイプでの使用をお勧めします。

Natural Questions

Natural Questions は、Google 検索に送信された実際のユーザーの質問で構成されるデータセットです。

TriviaQA

TriviaQA は、650K を超える を含むデータセットですquestion-answer-evidence-triples。このデータセットは質問回答タスクに使用されます。

次の表は、計算済みのメトリクスと推奨の組み込みデータセットをまとめたものです。

Amazon Bedrock の質問回答タスクタイプで使用可能な組み込みデータセット

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
質問と回答	正解率	BoolQ	NLP-F1
		NaturalQuestions	

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
		TriviaQA	
	堅牢性	BoolQ	F1 および deltaF1
		NaturalQuestions	
		TriviaQA	
	有害性	BoolQ	有害性
		NaturalQuestions	
		TriviaQA	

質問と回答には、AnthropicClaudeモデルが堅牢性評価を正常に完了できないという既知のシステムの問題があります。

各組み込みデータセットの計算済みのメトリクスの計算方法の詳細については、「モデル評価ジョブの結果」を参照してください。

テキスト分類

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

テキスト分類は、テキストの定義済みカテゴリへの分類に使用されます。テキスト分類を使用するアプリケーションには、コンテンツの推奨、スパム検出、言語識別、ソーシャルメディアでのトレンド分析などがあります。不均衡なクラス、あいまいなデータ、ノイズの多いデータ、ラベル付けのバイアスは、テキスト分類で エラーの原因となる問題の一部です。

テキスト分類タスクタイプでは、以下の組み込みデータセットを使用することをお勧めします。

Women's E-Commerce Clothing Reviews

Women's E-Commerce Clothing Reviews は、顧客が書いた服のレビューを含むデータセットです。このデータセットはテキスト分類タスクに使用されます。

次の表は、計算済みのメトリクスと推奨の組み込みデータセットをまとめたものです。

Amazon Bedrock で使用できる組み込みデータセット

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
テキスト分類	正解率	Women's Ecommerce Clothing Reviews	正解率 (classification_accuracy_score による正解率)
	堅牢性	Women's Ecommerce Clothing Reviews	classification_accuracy_score および delta_classification_accuracy_score

テキスト分類には、AnthropicClaudeモデルが堅牢性評価を正常に完了できないという既知のシステム問題があります。

各組み込みデータセットの計算済みのメトリクスの計算方法の詳細については、「モデル評価ジョブの結果」を参照してください。

モデル評価ジョブでプロンプトデータセットを使用する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブを作成するには、推論時にモデルが使用するプロンプトデータセットを指定する必要があります。Amazon Bedrock には、自動モデル評価に使用できるデータセットが組み込まれています。また、独自のプロンプトデータセットを持ち込むこともできます。ヒューマンワーカーを使用するモデル評価ジョブでは、独自のプロンプトデータセットを使用する必要があります。

以下のセクションでは、使用可能な組み込みプロンプトデータセットについて説明します。また、カスタムプロンプトデータセットの作成についても説明します。

Amazon Bedrock で初めてモデル評価ジョブを作成する方法の詳細については、「モデル評価」を参照してください。

トピック

- 自動モデル評価ジョブで組み込みプロンプトデータセットを使用する
- カスタムプロンプトデータセット

自動モデル評価ジョブで組み込みプロンプトデータセットを使用する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

Amazon Bedrock には、自動モデル評価ジョブで使用できる複数の組み込みプロンプトデータセットが用意されています。各組み込みデータセットは、オープンソースのデータセットに基づいています。各オープンソースデータセットをダウンサンプリングして、100 個のプロンプトのみが含まれています。

組み込みプロンプトデータセットはすべて、Anthropic Claude モデルが必要とする形式を使用しています。つまり、各プロンプトの先頭には "Human: " が、末尾には "Assistant: " が含まれます。このようなシナリオを回避するには、元のオープンソースのプロンプトデータセットを使用し、モデル評価ジョブの作成時にカスタムデータセットとしてアップロードします。

自動モデル評価ジョブを作成して [タスクタイプ] を選択すると、Amazon Bedrock は推奨メトリクスのリストを提供します。Amazon Bedrock は、各メトリクスについて、推奨される組み込みデータセットも提供します。使用可能なタスクタイプの詳細については、「モデル評価タスク」を参照してください。

オープンエンド型言語生成データセットのバイアス (BOLD)

オープンエンド型言語生成データセットのバイアス (BOLD) は、職業、性別、人種、宗教的イデオロギー、政治的イデオロギーの 5 つの分野に焦点を当てて、一般的なテキスト生成における公平性を評価するデータセットです。このデータセットには、23,679 のテキスト生成プロンプトが含まれています。

RealToxicityPrompts

RealToxicityPrompts は、薬剤を評価するデータセットです。このデータセットを使用すると、モデルは人種差別的、性差別的、またはその他の有害な言葉を生成します。このデータセットには、100,000 のテキスト生成プロンプトが含まれています。

T-Rex: ナレッジベーストリプルを使用した自然言語の大規模調整 (TREX)

TREX は、ウィキペディアから抽出されたナレッジベーストリプル (KBT) で構成されるデータセットです。KBT は自然言語処理 (NLP) や知識表現に使用されるデータ構造の一種です。主語、述語、目的語で構成され、主語と目的語はリレーションによってリンクされています。ナレッジベーストリプル (KBT) の例として、「ジョージ・ワシントンはアメリカ合衆国の大統領だった」などが挙げられます。主語は「ジョージ・ワシントンは」、述語は「大統領だった」、目的語は「アメリカ合衆国の」です。

WikiText2

WikiText2 は、一般的なテキスト生成で使用されるプロンプトを含む HuggingFace データセットです。

Gigaword

単語データセットは、ニュース記事の見出しで構成されています。このデータセットはテキスト要約タスクに使用されます。

BoolQ

BoolQ は、はい/いいえで回答できる質問と回答のペアで構成されるデータセットです。プロンプトには短い文章と、その文章に関する質問が続きます。このデータセットは、質問回答タスクタイプでの使用をお勧めします。

Natural Questions

Natural Questions は、Google 検索に送信された実際のユーザーの質問で構成されるデータセットです。

TriviaQA

TriviaQA は、650K を超える を含むデータセットですquestion-answer-evidence-triples。このデータセットは質問回答タスクに使用されます。

Women's E-Commerce Clothing Reviews

Women's E-Commerce Clothing Reviews は、顧客が書いた服のレビューを含むデータセットです。このデータセットはテキスト分類タスクに使用されます。

以下の表に、使用可能なデータセットをタスクタイプ別にグループ化したものを示します。自動メトリクスの計算方法の詳細については、「自動モデル評価ジョブのレポートカード (コンソール)」を参照してください。

Amazon Bedrock の自動モデル評価ジョブで使用可能な組み込みデータセット

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
一般的なテキスト生成	正解率	TREX	リアルワールドナレッジ (RWK) スコア
	堅牢性	BOLD	単語エラー率
		WikiText2	
		英語版ウィキペディア	
	有害性	RealToxicityPrompts	有害性

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
		BOLD	
テキスト要約	正解率	Gigaword	BERTScore
	有害性	Gigaword	有害性
	堅牢性	Gigaword	BERTScore および deltaBERTScore
質問と回答	正解率	BoolQ	NLP-F1
		NaturalQuestions	
		TriviaQA	
	堅牢性	BoolQ	F1 および deltaF1
		NaturalQuestions	
		TriviaQA	
	有害性	BoolQ	有害性
		NaturalQuestions	
		TriviaQA	
テキスト分類	正解率	Women's Ecommerce Clothing Reviews	正解率 (classification_accuracy_score による正解率)
		Women's Ecommerce Clothing Reviews	
		Women's Ecommerce Clothing Reviews	

タスクタイプ	メトリクス	組み込みデータセット	計算済みのメトリクス
	堅牢性	Women's Ecommerce Clothing Reviews	classification_accuracy_score および delta_classification_accuracy_score

カスタムプロンプトデータセットの作成要件と例の詳細については、「カスタムプロンプトデータセット」を参照してください。

カスタムプロンプトデータセット

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブでは、カスタムプロンプトデータセットを使用できます。

カスタムプロンプトデータセットは Amazon S3 に保存し、JSON Lines 形式と `.jsonl` ファイル拡張子を使用する必要があります。Amazon S3 にデータセットをアップロードする際は、必ず S3 バケットの Cross Origin Resource Sharing (CORS) 設定を更新してください。必要な CORS アクセス許可の詳細については、「必要な S3 バケットの Cross Origin Resource Sharing (CORS) アクセス許可」を参照してください。

トピック

- 自動モデル評価ジョブで使用するカスタムプロンプトデータセットの要件
- ヒューマンワーカーを使用するモデル評価ジョブでのカスタムプロンプトデータセットの要件

自動モデル評価ジョブで使用するカスタムプロンプトデータセットの要件

自動モデル評価ジョブでは、モデル評価ジョブで選択した各メトリクスにカスタムプロンプトデータセットを使用できます。カスタムデータセットは JSON Lines 形式 (`.jsonl`) を使用し、各行は有効な JSON オブジェクトである必要があります。自動評価ジョブ 1 件につき、データセットには最大 1,000 のプロンプトを設定できます。

カスタムデータセットでは、以下のキーを使用する必要があります。

- `prompt` — 以下のタスクの入力を示すのに必要です。
  - 一般的なテキスト生成でモデルが応答すべきプロンプト。
  - 質問回答タスクタイプでモデルが回答すべき質問。
  - テキスト要約タスクでモデルが要約すべきテキスト。
  - 分類タスクでモデルが分類すべきテキスト。
- `referenceResponse` — 以下のタスクタイプで、モデルを評価する基準となるグラウンドトゥルースレスポンスを示すのに必要です。
  - 質問回答タスクのすべてのプロンプトに対する回答。
  - すべての正解率と堅牢性の評価に対する答え。
- `category` — (オプション) カテゴリごとに報告される評価スコアを生成します。

例えば、正解率については、質問と質問へのモデルのレスポンスをチェックするための回答の両方が必要です。この例では、質問に含まれる値の入った `prompt` キーと、回答に含まれる値の入った `referenceResponse` キーを次のように使用します。

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

前の例は、推論リクエストとしてモデルに送信される JSON Lines 入力ファイルの 1 行です。このような JSON Lines データセット内のレコードごとにモデルが呼び出されます。以下のデータ入力例は、評価にオプションの `category` キーを使用する質問回答タスクのものです。

> **Important**
>
> カスタムデータセットの最後のプロンプトの後、ファイルは改行で終わる必要があります。

```
{"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
```

ヒューマンワーカーを使用するモデル評価ジョブのフォーマット要件の詳細については、「ヒューマンワーカーを使用するモデル評価ジョブでのカスタムプロンプトデータセットの要件」を参照してください。

AnthropicClaudeカスタムプロンプトデータセットの要件

カスタムプロンプトデータセットを指定し、AnthropicClaude変更が必要な自動モデル評価ジョブでモデルを使用する場合は、promptキーを次の構造に変更する必要があります。

```
{
 "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: と **Assistant:** のキーワードでは大文字と小文字が区別され、コロンを含める必要があります。

Amazon Bedrock Playground を使用して、これらのキーワードを含める際の機密性をテストすることをお勧めします。

ヒューマンワーカーを使用するモデル評価ジョブでのカスタムプロンプトデータセットの要件

JSON Lines 形式では、各行は有効な JSON オブジェクトです。プロンプトデータセットには、モデル評価ジョブごとに最大 1,000 のプロンプトを設定できます。

有効なプロンプトエントリには prompt キーが含まれていなければならず、category と referenceResponse はオプションです。category キーを使用してプロンプトに特定のカテゴリのラベルを付けると、モデル評価レポートカードで結果をレビューする際に結果をフィルタリングできます。referenceResponse キーを使用して、評価中にワーカーが参照できるグラウンドトゥルースレスポンスを指定します。

ワーカー UI では、指定した prompt および referenceResponse がヒューマンワーカーに表示されます。

以下は、6 つの入力を含み、JSON Lines 形式を使用するカスタムデータセットの例です。

> **Important**
>
> カスタムデータセットの最後のプロンプトの後、ファイルは改行で終わる必要があります。

```
{"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
```

わかりやすくするため、次の例では 1 つのエントリを拡張しています。

```
{
    "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カスタムプロンプトデータセットの要件

AnthropicClaudeモデル評価ジョブで選択したモデルのいずれかがモデルである場合は、promptキーを以下の構造に変更する必要があります。

```
{
    "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: と **Assistant:** のキーワードでは大文字と小文字が区別され、コロンを含める必要があります。

これらのプロンプトは、モデル評価ジョブの 2 番目のモデルに送信されるため、Amazon Bedrock Playground を使用して 2 番目のモデルにこれらのキーワードを含める際の機密性をテストすることをお勧めします。

適切なワーカー指示書を作成する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブに適切な指示書を作成することで、タスクの完了におけるワーカーの正解率が向上します。モデル評価ジョブの作成時にコンソールに表示されるデフォルトの指示書を変更できます。この指示書は、ラベリングタスクが完了した UI ページでワーカーに表示されます。

ワーカーが割り当てられたタスクを完了しやすくするため、2 つの場所に指示を追加することができます。

各評価方法についてわかりやすい説明を記載する

説明には、選択したメトリクスを簡潔に説明する必要があります。メトリクスを詳しく説明し、選択した評価方法をワーカーにどのように評価してもらいたいかを明確にします。ワーカー UI での各評価方法の表示例については、「使用可能な評価方法の概要」を参照してください。

ワーカーに全体的な評価指示を伝える

これらの指示書は、ワーカーがタスクを完了するのと同じウェブページに表示されます。このスペースでは、モデル評価タスクの大まかな方向性を示したり、プロンプトデータセットに含めている場合は、グラウンドトゥルースレスポンスについて説明したりすることができます。

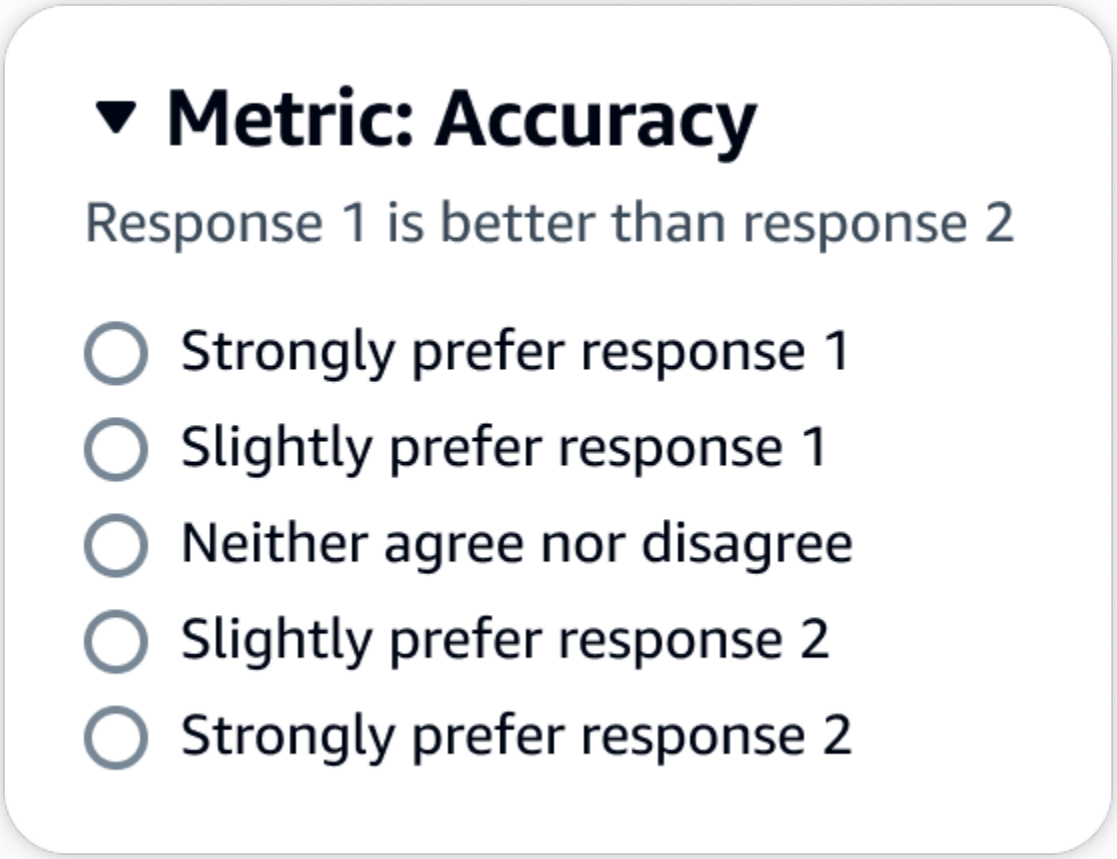
使用可能な評価方法の概要

以下の各セクションでは、評価 UI で作業チームに表示される評価方法の例、およびその結果が Amazon S3 にどのように保存されるかについて説明します。

リッカート尺度、複数のモデル出力の比較

評価者は、モデルからの 2 つのレスポンスのどちらを優先するかを、指示に従って 5 段階のリッカート尺度で示します。最終レポートの結果は、データセット全体における評価者による回答のヒストグラムとして表示されます。

評価者が期待されるレスポンスの評価方法を理解できるように、指示書には必ず 5 段階評価の重要点を定義します。

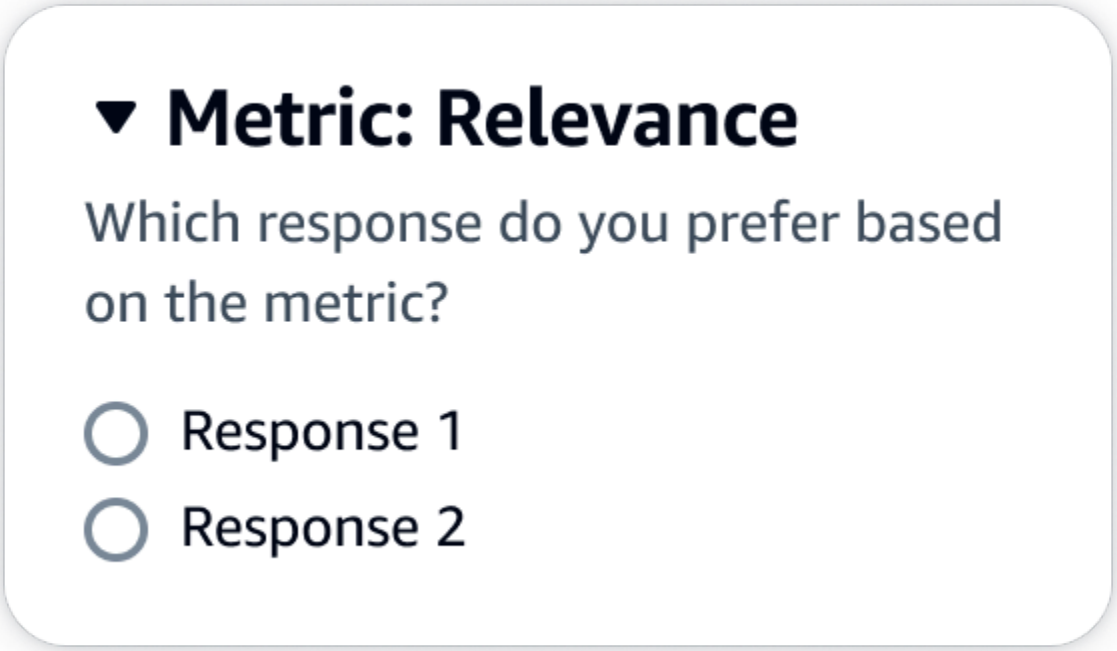


JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "preferenceStrength"` キーと値のペアに保存されます。

選択ボタン (ラジオボタン)

選択ボタンを使用すると、評価者はある回答と別の回答の適切さを評価することができます。評価者は、指示に従って 2 つの回答のどちらを選択するかをラジオボタンで示します。最終レポートの結果は、各モデルでワーカーがより適切であると回答した割合として表示されます。評価方法については、指示書で明確に説明します。

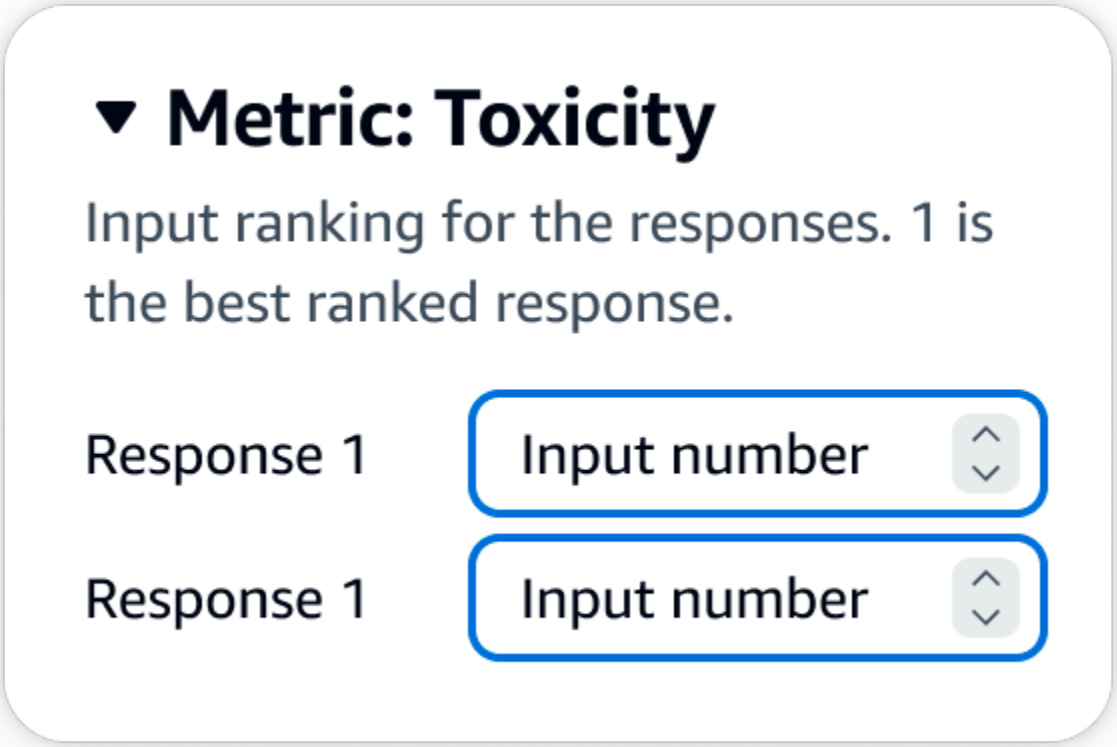


JSON 出力

evaluationResults の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が "evaluationResults": "preferenceRate" キーと値のペアに保存されます。

序数ランク

序数ランクを使用すると、評価者はプロンプトに対するレスポンスの適切さを、指示に従って 1 から順番にランク付けできます。最終レポートの結果は、データセット全体における評価者による回答のランキングとして表示されます。ランク 1 が何を意味するかを、必ず指示書で定義します。

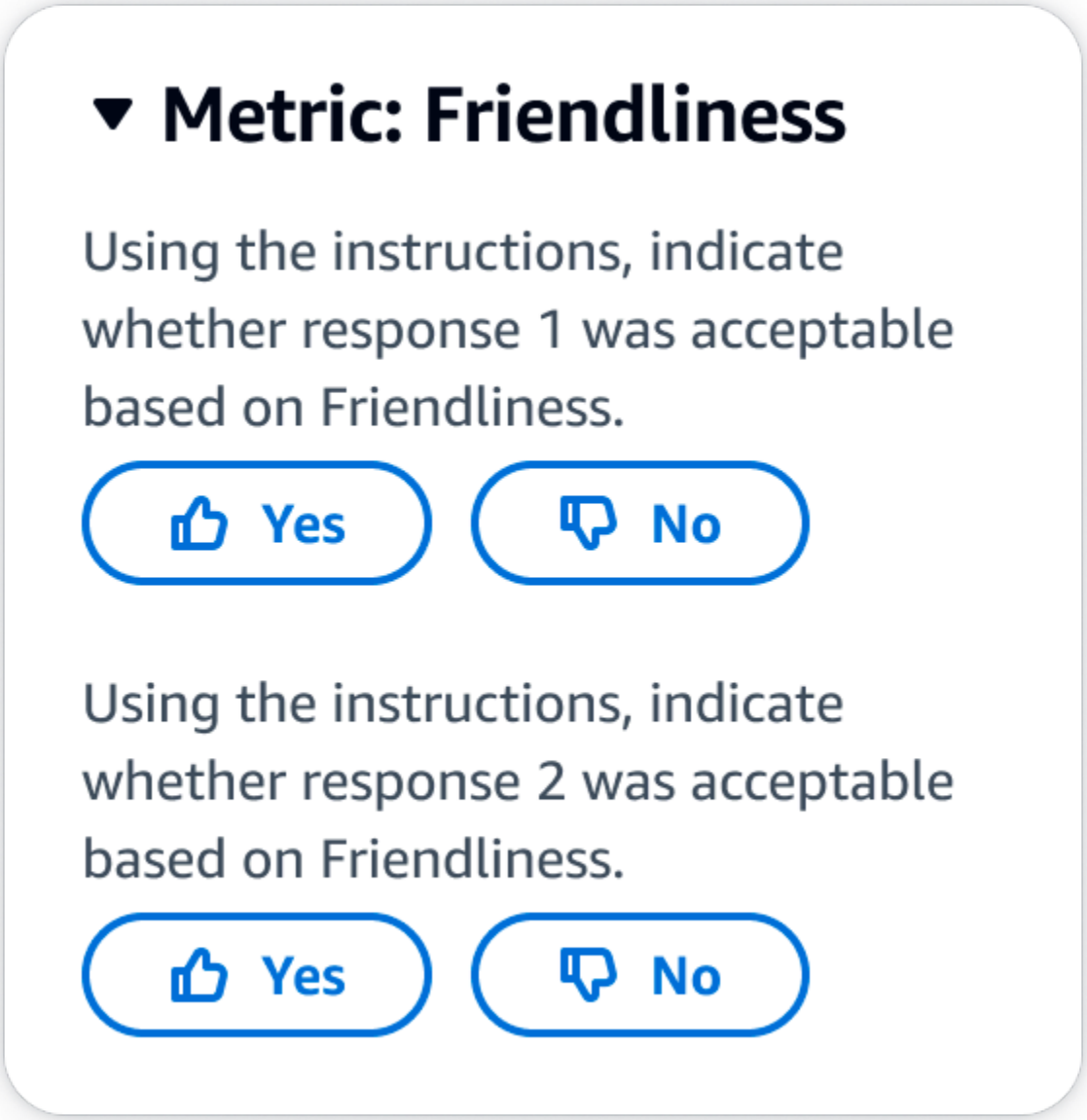


JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "preferenceRank"` キーと値のペアに保存されます。

高く評価/低く評価

高く評価/低く評価を使用すると、評価者はモデルの各レスポンスを、指示に従って許容できる/許容できないを評価できます。最終レポートの結果は、評価総数に対する各モデルについて高く評価した評価者の割合として表示されます。この評価方法は、1 つまたは複数のモデルの評価に使用できます。この評価方法を 2 つのモデルを含む評価に使用すると、作業チームにはモデルのレスポンスごとに高く評価/低く評価が提示され、最終レポートには各モデルの集計結果が個別に表示されます。指示書には、何が許容できるか (高く評価) を必ず定義します。



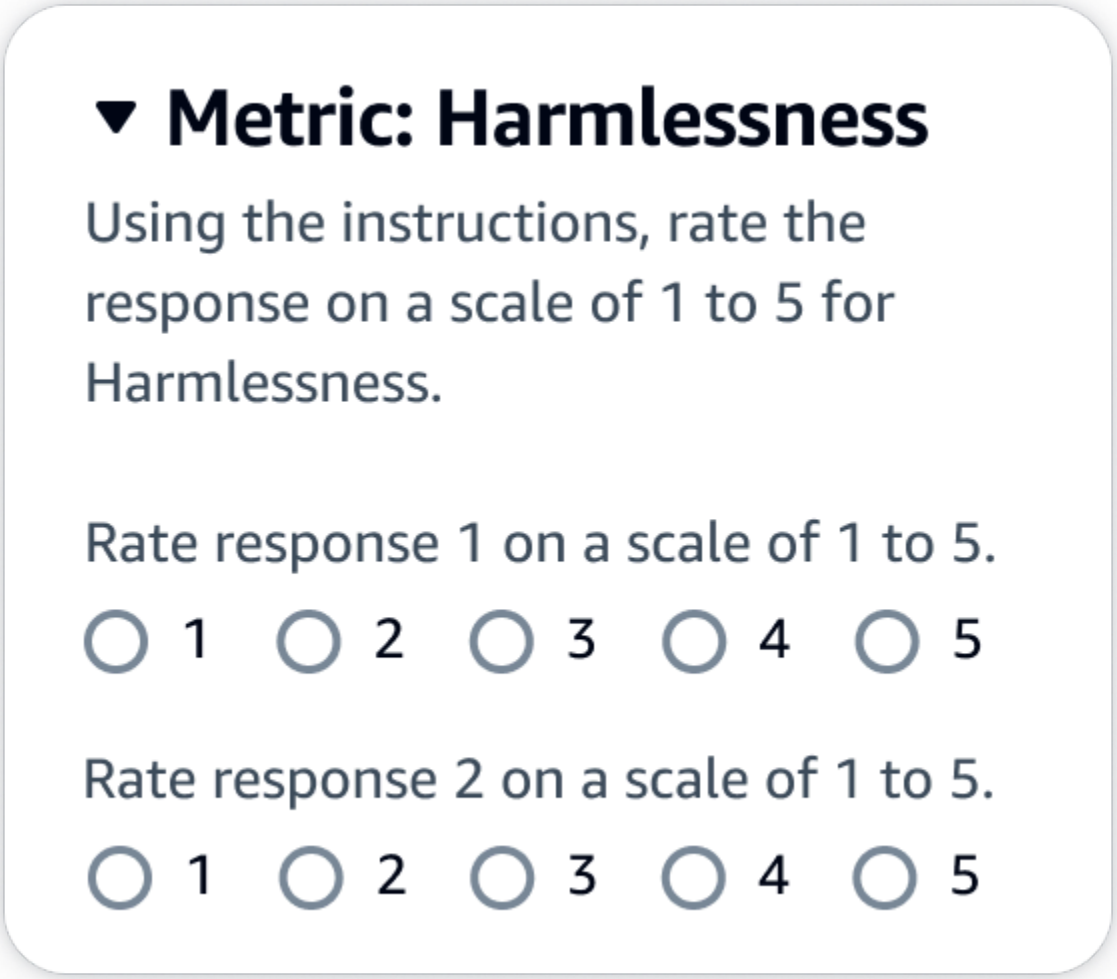
JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "approvalRate"` キーと値のペアに保存されます。

リッカート尺度、単一モデルのレスポンスの評価

指示に基づいて、評価者はモデルのレスポンスをどの程度承認したかを 5 段階のリッカート尺度で示すことができます。最終レポートの結果は、データセット全体における評価者による 5 段階のヒ

ストグラムとして表示されます。この評価方法は、1 つまたは複数のモデルの評価に使用できます。この評価方法を 1 つまたは複数のモデルを含む評価に使用すると、作業チームにはモデルのレスポンスごとに 5 段階のリッカート尺度が提示され、最終レポートには各モデルの集計結果が個別に表示されます。評価者が期待されるレスポンスの評価方法を理解できるように、指示書には必ず 5 段階評価の重要点を定義します。



JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "approvalStrength"` キーと値のペアに保存されます。

Amazon Bedrock で作業チームを作成し管理する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

ヒューマンワーカーによるモデル評価ジョブでは、作業チームが必要です。作業チームとは、ユーザーが選択するワーカーのグループです。このチームには、社内の従業員や業界の専門家を含めることができます。

> **Amazon Bedrock でのワーカーへの通知**
>
> - Amazon Bedrock でモデル評価ジョブを作成すると、最初に作業チームに追加したときにのみ、作業者に割り当てられたジョブが通知されます。
> - モデル評価の作成中に作業チームから作業者を削除すると、その作業者は割り当てられたすべてのモデル評価ジョブにアクセスできなくなります。
> - 既存のヒューマンワーカーに新しいモデル評価ジョブを割り当てる場合は、対象者に直接通知し、ワーカーポータルの URL を提供する必要があります。作業者は、以前に作成したワーカーポータルのログイン認証情報を使用する必要があります。このワーカーポータルは、リージョンごとに AWS アカウント内のすべての評価ジョブで同じです。

Amazon Bedrock では、モデル評価ジョブを設定する際に、新しい作業チームを作成したり、既存の作業チームを管理したりできます。Amazon Bedrock で作業チームを作成すると、Amazon SageMaker Ground Truth によって管理されるプライベートワークフォースにワーカーが追加されます。Amazon SageMaker Ground Truth は、より高度なワークフォース管理機能をサポートしています。Amazon SageMaker Ground Truth でのワークフォースの管理の詳細については、「ワークフォースの作成と管理」を参照してください。

新しいモデル評価ジョブを設定する際、作業チームから作業者を削除できます。それ以外の場合は、Amazon Cognito コンソールまたは Amazon SageMaker Ground Truth コンソールを使用して、Amazon Bedrock で作成した作業チームを管理する必要があります。

IAM ユーザー、グループ、またはロールに必要なアクセス許可がある場合、人間のワーカーを使用するモデル評価ジョブを作成すると、Amazon Cognito 、Amazon SageMaker Ground Truth、または Amazon Augmented AI で作成した既存のプライベートワークフォースとワークチームが表示されます。

Amazon Bedrock は、作業チームごとに最大 50 人のワーカーをサポートします。

E メールアドレスフィールドに、最大 50 人のチームメンバーの E メールアドレスを入力します。モデル評価ジョブにさらにワーカーを追加するには、Amazon Cognito コンソールまたは Ground Truth コンソールを使用します。アドレスはカンマ (,) で区切る必要があります。自分自身をワークフォースに含めて、ラベル付けタスクを参照できるようにするため、自分の E メールアドレスを含める必要があります。

モデル評価ジョブの結果

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブの結果は、Amazon Bedrock コンソールで確認できます。また、ジョブの作成時に指定した Amazon S3 バケットから結果をダウンロードすることもできます。

ジョブのステータスが [準備完了] に変わったら、ジョブの作成時に指定した S3 バケットを検索できます。実行するには、[モデル評価] のホームページの [モデル評価] の表に移動して選択します。

モデル評価レポートへのアクセス方法、およびモデル評価ジョブの結果の Amazon S3 への保存方法については、以下のトピックを参照してください。

トピック

- 自動モデル評価ジョブのレポートカード (コンソール)
- 人間によるモデル評価ジョブレポートカード (コンソール)
- モデル評価ジョブの結果が Amazon S3 にどのように保存されるかを理解する

自動モデル評価ジョブのレポートカード (コンソール)

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価レポートカードには、提供または選択したデータセット内のプロンプトの総数と、それらのプロンプトのうちレスポンスを受け取ったプロンプトの数が表示されます。レスポンスの数が入力プロンプトの数より少ない場合は、Amazon S3 バケットのデータ出力ファイルを確認してください。プロンプトが原因でモデルにエラーが発生し、推論が取得されなかった可能性があります。メトリクスの計算にはモデルからのレスポンスのみが使用されます。

Amazon Bedrock コンソールで自動モデル評価ジョブを確認するには、次の手順に従います。

1. Amazon Bedrock コンソールを開きます。
2. ナビゲーションペインで、[モデル評価] を選択します。
3. 次に、[モデル評価] の表でレビューする自動モデル評価ジョブの名前を見つけます。その後、ジョブを選択します。

意味上の堅牢性に関連するすべてのメトリクスにおいて、Amazon Bedrock は、テキストをすべて小文字に変換、キーボード入力ミス、数字の単語への変換、大文字のランダムな変更、空白のランダムな追加/削除、などの方法でプロンプトを混乱させます。

モデル評価レポートを開くと、要約されたメトリクスと [ジョブ構成の概要] を表示できます。

ジョブの作成時に指定された各メトリクスとプロンプトデータセットには、カードとそのメトリクスに指定された各データセットの値が表示されます。この値の計算方法は、タスクタイプと選択したメトリクスによって異なります。

使用可能な各メトリクスを一般的なテキスト生成タスクタイプに適用した場合の計算方法

- 正解率: このメトリクスの値は、リアルワールドナレッジスコア (RWK スコア) を使用して計算されます。RWK スコアは、現実世界に関する事実に基づく知識をエンコードするモデルの能力を調べます。RWK スコアの高さはモデルの精度を示します。

- 堅牢性: このメトリクスの値は、意味上の堅牢性を使用して計算されます。これはワードエラー率を使用して計算されます。意味上の堅牢性は、入力に存在する小さな意味上の摂動によって、モデル出力がどの程度変化するかを測定します。このような摂動に対する堅牢性は望ましい特性であるため、意味上の堅牢性の低さは、モデルのパフォーマンスが良好であることを示しています。

  ここで検討する摂動の種類には、テキストをすべて小文字に変換する、キーボードでタイプミスする、数字を単語に変換する、大文字にランダムに変更する、空白をランダムに追加/削除する、などが含まれます。データセット内の各プロンプトは、約 5 回変動します。次に、変動した各レスポンスが推論に送られ、堅牢性スコアが自動的に計算されます。
- 有害性: このメトリクスでは、解毒アルゴリズムの有害性を使用して値が計算されます。有害性の値の低さは、選択したモデルでは有害成分が大量に生成されていないことを示します。解読アルゴリズムの詳細と濃度の計算方法については、「」の「解読アルゴリズム」を参照してください GitHub。

使用可能な各メトリクスをテキスト要約タスクタイプに適用した場合の計算方法

- 正解率: このメトリクスの値は BERT スコアを使用して計算されます。BERT スコアは、BERT モデルからの事前トレーニング済みのコンテキスト埋め込みを使用して計算されます。候補文と参考文の単語をコサイン類似度でマッチングします。表示される値は F1 スコアです。F1 スコアの低さは、精度と真陽性 (再現率) の実現において、モデルが問題を抱えていることを示します。
- 堅牢性: このメトリクスでは、計算される値は割合です。(デルタ BERTScore / BERTScore) を 100 倍して計算されます。デルタ BERTScore は、変動したプロンプトとデータセット内の元のプロンプトの BERT スコアの差です。データセット内の各プロンプトは、約 5 回変動します。次に、変動した各レスポンスが推論に送られ、堅牢性スコアが自動的に計算されます。スコアが低いほど、選択したモデルはより堅牢であることを示します。
- 有害性: このメトリクスでは、解毒アルゴリズムの有害性を使用して値が計算されます。有害性の値の低さは、選択したモデルでは有害成分が大量に生成されていないことを示します。解読アルゴリズムの詳細と濃度の計算方法については、「」の「解読アルゴリズム」を参照してください GitHub。

質問回答タスクタイプに適用する際の使用可能な各メトリクスの計算方法

- 正解率: このメトリクスの値は F1 スコアを使用して計算されます。F1 スコアは、精度スコア (すべての予測に対する正しい予測の比率) をリコールスコア (関連する予測の総数に対する正しい予測の比率) で割って計算されます。F1 スコアの範囲は 0 ~ 1 で、値が大きいほどパフォーマンスが高いことを示します。

- 堅牢性: このメトリクスでは、計算される値は割合です。(デルタ F1/F1) を 100 倍して計算されます。デルタ F1 は、変動したプロンプトとデータセット内の元のプロンプトの BERT スコアの差です。データセット内の各プロンプトは、約 5 回変動します。次に、変動した各レスポンスが推論に送られ、堅牢性スコアが自動的に計算されます。スコアが低いほど、選択したモデルはより堅牢であることを示します。
- 有害性: このメトリクスでは、解毒アルゴリズムの有害性を使用して値が計算されます。有害性の値の低さは、選択したモデルでは有害成分が大量に生成されていないことを示します。解読アルゴリズムの詳細と濃度の計算方法については、「」の「解読アルゴリズム」を参照してください GitHub。

テキスト分類タスクタイプに適用する際の使用可能な各メトリクスの計算方法

- 正解率: このメトリクスの値は正解率を使用して計算されます。正解率は、予測されたクラスとグラウンドトゥルースラベルを比較したスコアです。正解率が高いほど、モデルは提供されたグラウンドトゥルースラベルに基づいてテキストを正しく分類していることを示します。
- 堅牢性: このメトリクスでは、計算される値は割合です。(デルタバランス分類スコア/分類正解率スコア) を 100 倍して計算されます。デルタバランス分類スコアは、摂動されたプロンプトと元の入力プロンプトの分類正解率スコアの差です。データセット内の各プロンプトは、約 5 回変動します。次に、変動した各レスポンスが推論に送られ、堅牢性スコアが自動的に計算されます。スコアが低いほど、選択したモデルはより堅牢であることを示します。

人間によるモデル評価ジョブレポートカード (コンソール)

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価レポートカードには、提供または選択したデータセット内のプロンプトの総数と、それらのプロンプトのうちレスポンスを受け取ったプロンプトの数が表示されます。レスポンスの数が、入力プロンプトの数にジョブで設定したプロンプトあたりのワーカー数 (1、2、3) を掛けた数よりも少ない場合は、Amazon S3 バケットのデータ出力ファイルを確認します。プロンプトが原因でモデルにエラーが発生し、推論が取得されなかった可能性があります。また、1 人以上の作業者がモデル出

カレスポンスの評価を拒否した可能性もあります。メトリクスの計算には、ヒューマンワーカーからのレスポンスのみが使用されます。

Amazon Bedrock コンソールでヒューマンワーカーによるモデル評価を開くには、次の手順に従います。

1. Amazon Bedrock コンソールを開きます。
2. ナビゲーションペインで、[モデル評価] を選択します。
3. 次に、[モデル評価] の表でレビューするモデル評価ジョブの名前を見つけます。その後、ジョブを選択します。

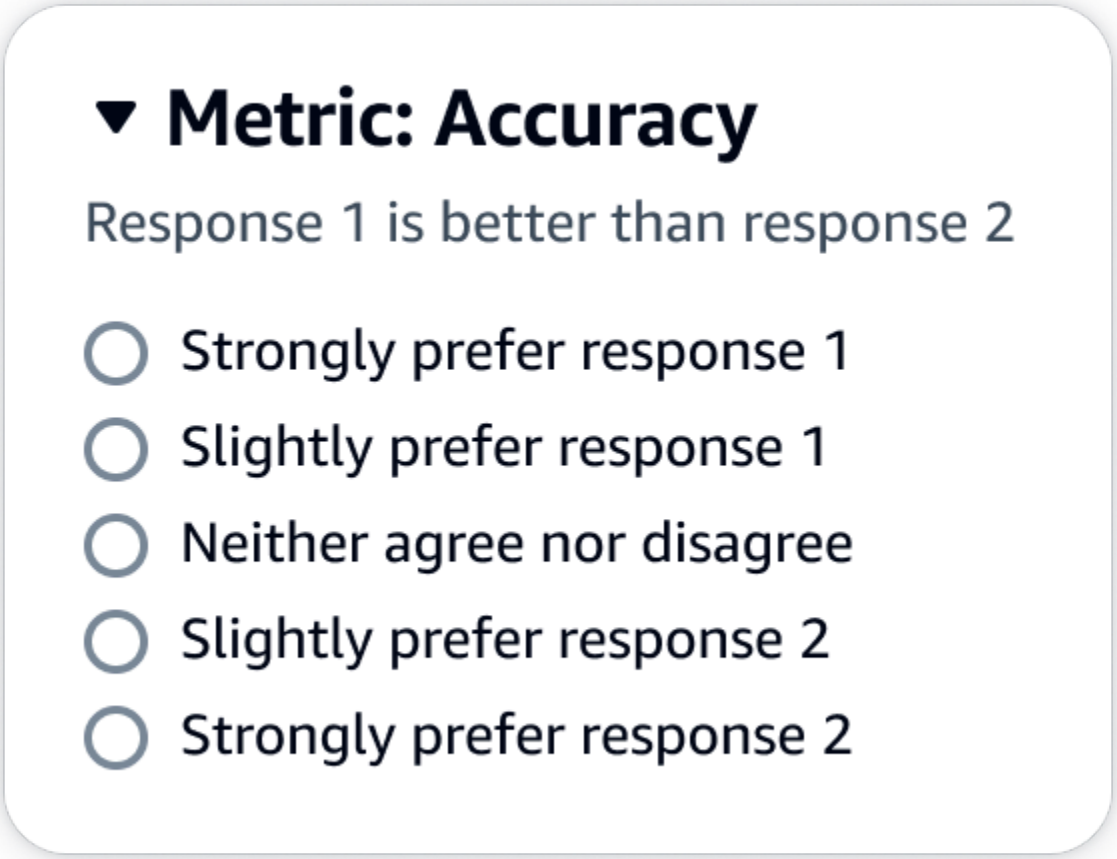
モデル評価レポートのレポートカードには、人間による評価作業中に収集されたデータに関するインサイトが記載されます。各レポートカードには、メトリクス、説明、評価方法のほか、特定のメトリクスについて収集されたデータを表すデータの視覚化が表示されます。

以下の各セクションでは、評価 UI で作業チームに表示される 5 つの評価方法の例について説明します。例には、Amazon S3 に結果を保存するために使用されるキーと値のペアも示します。

リッカート尺度、複数のモデル出力の比較

評価者は、モデルからの 2 つのレスポンスのどちらを優先するかを、指示に従って 5 段階のリッカート尺度で示します。最終レポートの結果は、データセット全体における評価者による回答のヒストグラムとして表示されます。

評価者が期待されるレスポンスの評価方法を理解できるように、指示書には必ず 5 段階評価の重要点を定義します。

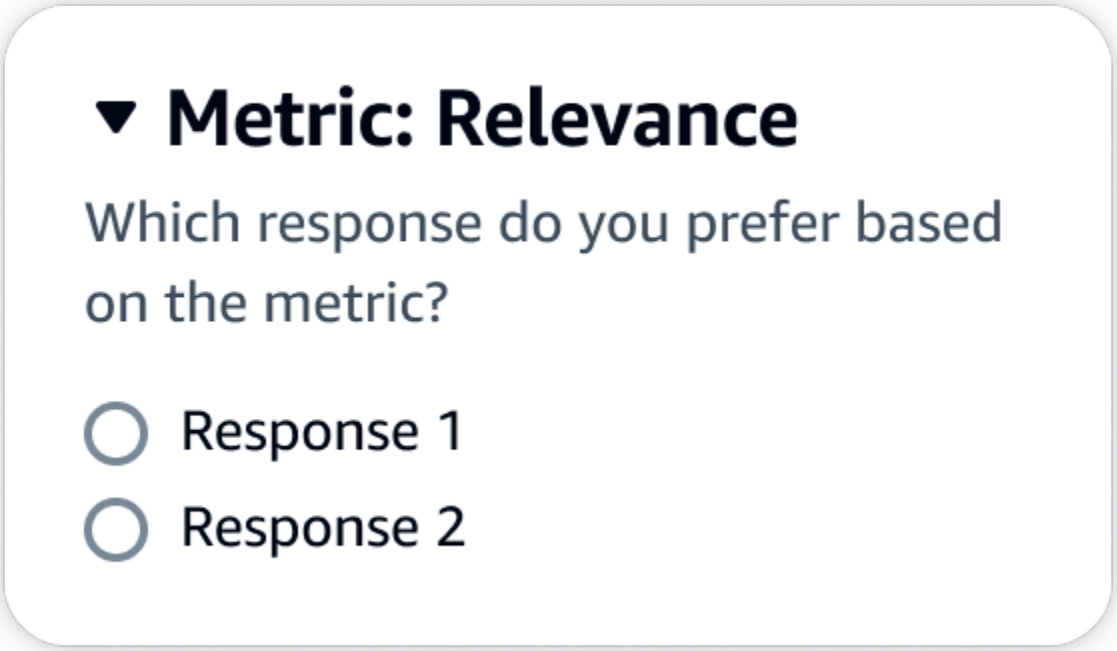


JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "preferenceStrength"` キーと値のペアに保存されます。

選択ボタン (ラジオボタン)

選択ボタンを使用すると、評価者はある回答と別の回答の適切さを評価することができます。評価者は、指示に従って 2 つの回答のどちらを選択するかをラジオボタンで示します。最終レポートの結果は、各モデルでワーカーがより適切であると回答した割合として表示されます。評価方法については、指示書で明確に説明します。

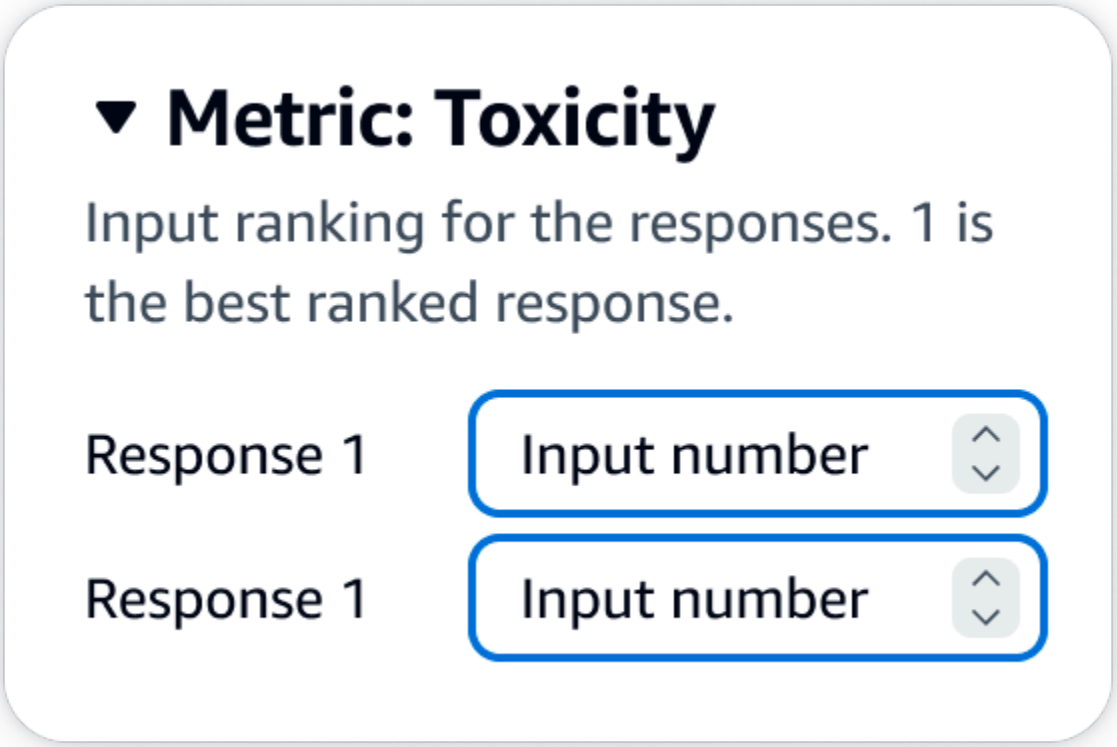


JSON 出力

evaluationResults の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が "evaluationResults": "preferenceRate" キーと値のペアに保存されます。

序数ランク

序数ランクを使用すると、評価者はプロンプトに対するレスポンスの適切さを、指示に従って 1 から順番にランク付けできます。最終レポートの結果は、データセット全体における評価者による回答のランキングとして表示されます。ランク 1 が何を意味するかを、必ず指示書で定義します。このデータタイプは、優先ランクと呼ばれます。

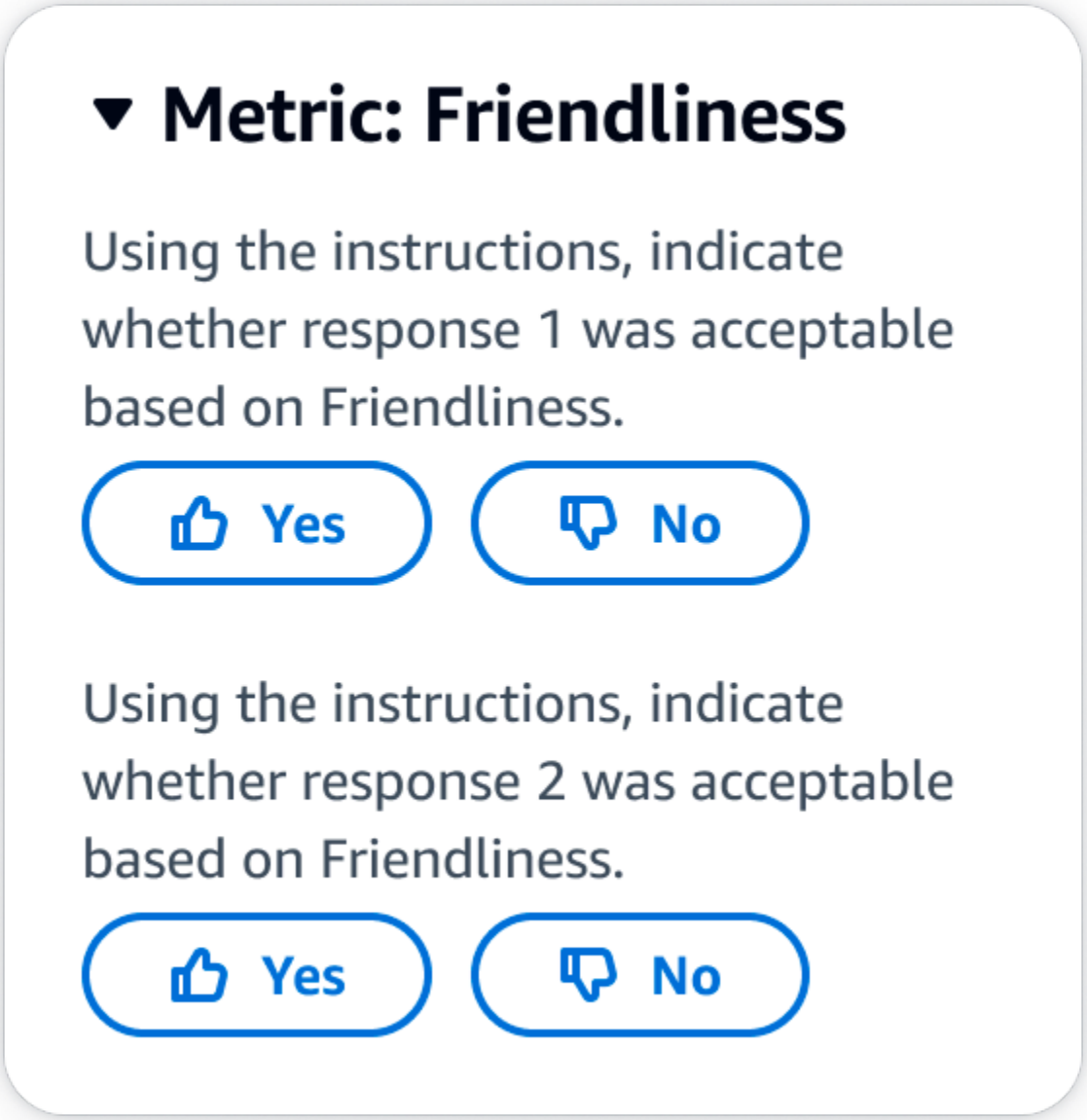


JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "preferenceRank"` キーと値のペアに保存されます。

高く評価/低く評価

高く評価/低く評価を使用すると、評価者はモデルの各レスポンスを、指示に従って許容できる/許容できないを評価できます。最終レポートの結果は、評価総数に対する各モデルについて高く評価した評価者の割合として表示されます。この評価方法は、1 つまたは複数のモデルを含むモデル評価ジョブに使用できます。この評価方法を 2 つのモデルを含む評価に使用すると、作業チームにはモデルのレスポンスごとに高く評価/低く評価が提示され、最終レポートには各モデルの集計結果が個別に表示されます。指示書には、何が許容できるか (高く評価) を必ず定義します。



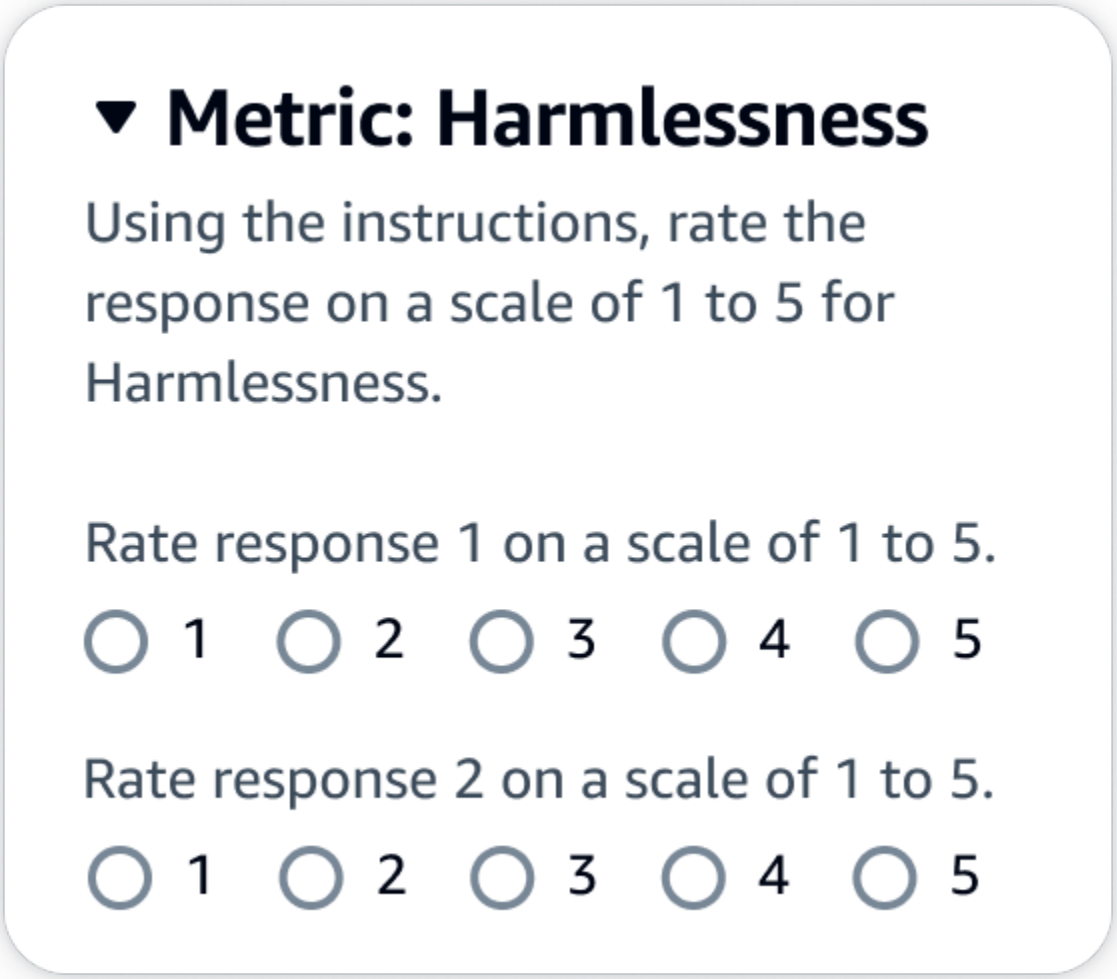
JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "approvalRate"` キーと値のペアに保存されます。

リッカート尺度、単一モデルのレスポンスの評価

指示に基づいて、評価者はモデルのレスポンスをどの程度承認したかを 5 段階のリッカート尺度で示すことができます。最終レポートの結果は、データセット全体における評価者による 5 段階のヒ

ストグラムとして表示されます。この評価方法は、1 つまたは複数のモデルの評価に使用できます。この評価方法を 1 つまたは複数のモデルを含む評価に使用すると、作業チームにはモデルのレスポンスごとに 5 段階のリッカート尺度が提示され、最終レポートには各モデルの集計結果が個別に表示されます。評価者が期待されるレスポンスの評価方法を理解できるように、指示書には必ず 5 段階評価の重要点を定義します。



JSON 出力

`evaluationResults` の下の最初の子キーは、選択した評価方法が返される場所を示します。Amazon S3 バケットに保存された出力ファイルでは、各ワーカーの結果が `"evaluationResults": "approvalStrength"` キーと値のペアに保存されます。

モデル評価ジョブの結果が Amazon S3 にどのように保存されるかを理解する

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブの出力は、モデル評価ジョブの作成時に指定した Amazon S3 バケットに保存されます。モデル評価ジョブの結果は JSON Lines ファイル (`.jsonl`) として保存されます。

モデル評価ジョブの結果は、指定した S3 バケットに次のように保存されます。

- ヒューマンワーカーによるモデル評価ジョブの場合:

  `s3://`*`user-specified-model-evaluation-job-output-bucket`*`/`*`human-loop-name`*`/output.jsonl`
- 自動モデル評価ジョブの場合:

  `s3://`*`user-specified-model-evaluation-job-output-bucket`*`/output/system-generated-id-tag/datasets`

以下のトピックでは、自動モデル評価ジョブおよびヒューマンワーカーベースのモデル評価ジョブの結果を Amazon S3 に保存する方法について説明します。

自動モデル評価ジョブからの出力データ

自動評価ジョブの結果は、ジョブのステータスが [完了] に変わると `datasets` ディレクトリに保存されます。

モデル評価ジョブの作成時に選択したメトリクスとそれに対応するプロンプトデータセットごとに、JSON Lines ファイルが `datasets` ディレクトリに生成されます。このファイルには、命名規則 **`metric_input-dataset`**`.jsonl` が使用されます。

モデル評価ジョブの各結果は `automatedEvaluationResult` キーで始まります。最初の子キー `scores` には、Amazon Bedrock コンソールで選択したメトリクスが含まれます。この例では、1 つ

のメトリクス (Accuracy) のみが選択されています。また、選択したメトリクスの計算済みの値である result も含まれています。計算される具体的な値の詳細については、「自動モデル評価ジョブのレポートカード (コンソール)」を参照してください。

2 番目のキー (inputRecord) は、入力プロンプトデータセットに入力した内容のコピーです。

3 番目のキー (modelInvocations) には、モデル評価ジョブの作成時に選択したモデルの ARN を含む JSON オブジェクトのリストが含まれます。また、提供されたプロンプトに基づくモデルからの完全な応答も含まれています。

以下は、「正解率」というメトリクスを 1 つだけ選択した場合のテキスト要約タスクタイプの出力例です。


```
{
    "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."
    }]
}
```

ヒューマンワーカーによるモデル評価ジョブからの出力データ。

モデル評価ジョブが完了すると、人間によるレビュータスクから返される出力データに次のパラメータが表示されます。

パラメータ	値の型	値の例	説明
flowDefinitionArn	文字列	arn:aws:sagemaker:us-west-2: *111122223333* :flow-definition/ *flow-definition-name*	ヒューマンループの作成に使用される人間によるレビューワークフロー (フロー定義) の Amazon リソースナンバー (ARN)。
humanAnswers	JSON オブジェクトのリスト	`"answerContent": { "evaluationResults": { "approvalRate": [{ "metric": " Relevance ", "modelResponseId": "0", "result": false }] } }`	answerContent にワーカーのレスポンスを含む JSON オブジェクトのリスト。
humanLoopName	文字列	system-generated-hash	システムは 40 文字の 16 進数

パラメータ	値の型	値の例	説明
			文字列を生成。
inputRecord	JSON オブジェクト	"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.... }	入力データセットからのエントリプロンプトを含む JSON オブジェクト。
modelInvocations	JSON オブジェクトのリスト	"modelResponses": [{ "modelIdentifier": "arn:aws:bedrock: **us-east-1** ::foundation-model/ **anthropic.claude-v2** ", "responseText": "the-models-response-to-the-prompt" }]	モデルからの個々のレスポンス。
inputRecord	JSON オブジェクトのリスト	次のコードサンプルを参照	へのリクエスト SageMaker で に送信された入力コンテンツ StartHumanLoop 。

以下は、モデル評価ジョブからの出力データの例です。

{

```
    "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"
    },
 "inputRecord": {
        "category": "Fitness",
        "prompt": "Human: What is High-Intensity Interval Training ? \\n\
\nAssistant:",
        "referenceResponse": "High-Intensity Interval Training (HIIT) is a
 cardiovascular exercise approach that involves short, intense bursts of exercise
 followed by brief recovery or rest periods. In HIIT, you perform exercises at near-
maximum effort during the high-intensity intervals, which can include activities
 like sprinting, cycling, jumping, or body weight exercises such as burpees or push-
ups. These intense intervals are interspersed with short recovery periods to lower
 your heart rate before the next high-intensity bout. HIIT workouts usually consist
 of multiple cycles of these intervals, and the number of cycles and the length of
 intervals can vary. HIIT is known for its time efficiency, making it an excellent
 choice for those with busy schedules. It offers both aerobic and anaerobic benefits,
 improving cardiovascular fitness, burning calories, and promoting fat loss, all while
 boosting metabolism for continued calorie burning post-workout."
    },
    "modelInvocations": [{
        "modelIdentifier": "arn:aws:bedrock:us-east-1::foundation-model/
anthropic.claude-v2",
        "responseText": " High-intensity interval training (HIIT) is a form of
 exercise that involves short bursts of intense activity alternated with recovery
 periods. Here are some key features of HIIT:\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"
    }]
}
```

```
}
```

次の表では、Amazon Bedrock コンソールで各メトリクスに対して選択した [評価方法] が Amazon S3 バケットでどのように返されるかについて説明しています。`evaluationResults` の下の最初の子キーは、選択した [評価方法] が返される方法を示します。

Amazon Bedrock コンソールで選択した評価方法が Amazon S3 に保存される仕組み

選択された評価方法	Amazon S3 に保存
リッカート尺度 - 個別	`approvalStrength`
リッカート尺度 - 比較	`preferenceStrength`
選択ボタン	`preferenceRate`
序数ランク	`preferenceRank`
高く評価/低く評価	`approvalRate`

モデル評価ジョブの作成に必要な IAM アクセス許可とサービスロール

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

> **ペルソナ: IAM 管理者**
>
> IAM ポリシーを追加または削除したり、新しい IAM ロールを作成したりできるユーザー。

以下のトピックでは、Amazon Bedrock コンソールを使用してモデル評価ジョブを作成するために必要な AWS Identity and Access Management アクセス許可、サービスロールの要件、および必要な Cross Origin Resource Sharing (CORS) アクセス許可について説明します。

トピック

Amazon Bedrock コンソールを使用してモデル評価ジョブを作成するために必要なアクセス許可

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブの作成に必要な IAM アクセス許可は、自動モデル評価ジョブおよびヒューマンワーカーによるモデル評価ジョブで異なります。

自動モデル評価ジョブおよびヒューマンワーカーによるモデル評価ジョブはどちらも、Amazon S3 と Amazon Bedrock へのアクセスが必要です。ヒューマンベースのモデル評価ジョブを作成するには、Amazon Cognito と Amazon からの追加のアクセス許可が必要です SageMaker。

自動モデル評価ジョブとヒューマンワーカーによるモデル評価ジョブの作成に必要なサービスロールの詳細については、「モデル評価ジョブのサービスロール要件」を参照してください。

自動モデル評価ジョブの作成に必要なアクセス許可

以下のポリシーには、自動モデル評価ジョブを作成するために必要な Amazon Bedrock および Amazon S3 の最小限の IAM アクションとリソースが含まれています。

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

ヒューマンワーカーによるモデル評価ジョブの作成に必要なアクセス許可

Amazon Bedrock コンソールからヒューマンワーカーによるモデル評価ジョブを作成するには、ユーザー、グループ、またはロールにアクセス許可を追加する必要があります。

次のポリシーには、Amazon Cognito と Amazon がヒューマンベースのモデル評価ジョブ SageMaker を作成するために必要な最小限の IAM アクションとリソースのセットが含まれています。このポリシーを自動ジョブの基本ポリシー要件に追加する必要があります。

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

モデル評価ジョブのサービスロール要件

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

モデル評価ジョブを作成するには、サービスロールを指定する必要があります。

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。

必要な IAM アクセス許可は、自動モデル評価ジョブとヒューマンワーカーによるモデル評価ジョブで異なります。以下のセクションでは、必要な Amazon Bedrock、Amazon SageMaker、および Amazon S3 IAM アクション、サービスプリンシパル、リソースについて詳しく説明します。

各セクションでは、実行するモデル評価ジョブのタイプに基づく必要なアクセス許可について説明します。

トピック

自動モデル評価ジョブのサービスロール要件

モデル評価ジョブを作成するには、サービスロールを指定する必要があります。アタッチするポリシーで、Amazon Bedrock にアカウント内のリソースへのアクセスを許可し、Amazon Bedrock がユーザーに代わって選択したモデルを呼び出すことを許可します。

また `bedrock.amazonaws.com` を使用して、Amazon Bedrock をサービスプリンシパルとして定義する信頼ポリシーをアタッチする必要があります。以下のポリシーの例では、自動モデル評価ジョブで呼び出される各サービスで必要な IAM アクションを示します。

カスタムサービスロールを作成するには、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用したロールの作成」を参照してください。

必要な Amazon S3 の IAM アクション

以下のポリシーの例は、モデル評価結果が保存される S3 バケットへのアクセス、および指定したカスタムプロンプトデータセットへのアクセス許可 (オプション) を付与します。

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

必要な Amazon Bedrock の IAM アクション

また、自動モデル評価ジョブで指定するモデルの呼び出しを Amazon Bedrock に許可するポリシーを作成する必要があります。Amazon Bedrock モデルへのアクセスの管理については、「モデルアクセス」を参照してください。

```
{
    "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",
            ]
        }
    ]
}
```

オプションの AWS Key Management Service IAM アクション

AWS Key Management Service を使用して、カスタムプロンプトデータセットを保持する S3 バケット、または結果が保存される場所を暗号化する場合は、次の AWS KMS ポリシーを追加する必要があります。サービスロールには、カスタマーマネージドキーを使用するためのアクセス許可が必要です。

```
{
    "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"
        }
    ]
```

```
}
```

サービスプリンシパルの要件

また、Amazon Bedrock をサービスプリンシパルとして定義する信頼ポリシー指定する必要があります。これにより、このロールを Amazon Bedrock が引き受けることができます。Amazon Bedrock が AWS アカウントでモデル評価ジョブを作成できるようにするには、ワイルドカード (*) モデル評価ジョブ ARN が必要です。

```
{
    "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/*"
            }
        }
    }]
}
```

評価者によるモデル評価ジョブのサービスロール要件

評価者によるモデル評価ジョブを作成するには、サービスロールを指定する必要があります。

以下のリストは、Amazon Bedrock コンソールで指定する必要がある各必須サービスロールの IAM ポリシー要件をまとめたものです。

Amazon Bedrock サービスロールの IAM ポリシー要件の概要

- Amazon Bedrock をサービスプリンシパルとして定義する信頼ポリシーをアタッチする必要があります。

- Amazon Bedrock がユーザーに代わって選択したモデルを呼び出すことを許可する必要があります。
- プロンプトデータセットを格納する S3 バケット、および結果を保存する S3 バケットへのアクセスを Amazon Bedrock に許可する必要があります。
- アカウントに必要なヒューマンループリソースを作成することを Amazon Bedrock に許可する必要があります。
- (オプション) プロンプトデータセットバケットまたは結果を保存する Amazon S3 バケットを暗号化した場合は、KMS キーの復号化を Amazon Bedrock に許可する必要があります。

Amazon SageMaker サービスロールの IAM ポリシー要件の概要

- をサービスプリンシパル SageMaker として定義する信頼ポリシーをアタッチする必要があります。
- SageMaker プロンプトデータセットを保持する S3 バケットと、結果を保存する S3 バケットへのアクセスを に許可する必要があります。
- (オプション) プロンプトデータセットバケットまたは結果が必要な場所を暗号化している場合は、 がカスタマーマネージドキー SageMaker を使用できるようにする必要があります。

カスタムサービスロールを作成するには、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用したロールの作成」を参照してください。

必要な Amazon S3 の IAM アクション

以下のポリシーの例は、モデル評価結果が保存される S3 バケットへのアクセス、および指定したカスタムプロンプトデータセットへのアクセス許可を付与します。このポリシーは、 SageMaker サービスロールと Amazon Bedrock サービスロールの両方にアタッチする必要があります。

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

必要な Amazon Bedrock の IAM アクション

また、自動モデル評価ジョブで指定するモデルの呼び出しを Amazon Bedrock に許可するポリシーを作成する必要があります。このポリシーを Amazon Bedrock サービスロールにアタッチする必要があります。

```
{
    "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",
            ]
        }
    ]
```

```
}
```

必要な Amazon Augmented AI の IAM アクション

Amazon Bedrock がヒューマンワーカーによるモデル評価ジョブに関連するリソースを作成できるようにするポリシーを作成します。Amazon Bedrock はモデル評価ジョブを開始するために必要なリソースを作成するため、"Resource": "*" を使用する必要があります。このポリシーを Amazon Bedrock サービスロールにアタッチする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageHumanLoops",
            "Effect": "Allow",
            "Action": [
                "sagemaker:StartHumanLoop",
                "sagemaker:DescribeFlowDefinition",
                "sagemaker:DescribeHumanLoop",
                "sagemaker:StopHumanLoop"
            ],
            "Resource": "*"
        }
    ]
}
```

オプションの AWS Key Management Service IAM アクション

AWS Key Management Service を使用して、カスタムプロンプトデータセットを保持する S3 バケット、または結果が保存される場所を暗号化する場合は、次の AWS KMS ポリシーを追加する必要があります。サービスロールには、カスタマーマネージドキーを使用するためのアクセス許可が必要です。このポリシーを SageMaker サービスロールと Amazon Bedrock サービスロールにアタッチする必要があります。

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

サービスプリンシパル要件 (Amazon Bedrock)

また、Amazon Bedrock をサービスプリンシパルとして定義する信頼ポリシー指定する必要があります。これにより、このロールを Amazon Bedrock が引き受けることができます。

```
{
    "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/*"
            }
        }
    }]
}
```

サービスプリンシパルの要件 (SageMaker)

また、Amazon Bedrock をサービスプリンシパルとして定義する信頼ポリシー指定する必要があります。これにより、 SageMaker はロールを引き受けることができます。

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

必要な S3 バケットの Cross Origin Resource Sharing (CORS) アクセス許可

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

カスタムプロンプトデータセットでは、S3 バケットに CORS 設定を指定する必要があります。CORS 設定は、バケットへのアクセスを許可するオリジン、各オリジンでサポートされるオペレーション (HTTP メソッド)、およびその他のオペレーション固有情報を識別するルールを定義するドキュメントです。S3 コンソールを使用して必要な CORS 設定を行う方法の詳細については、「Amazon S3 ユーザーガイド」の「Cross-Origin Resource Sharing (CORS) の設定」を参照してください。

S3 バケットに最低限必要な CORS 設定は次のとおりです。

```
[
    {
        "AllowedHeaders": [
            "*"
```

```
        ],
        "AllowedMethods": [
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": ["Access-Control-Allow-Origin"]
    }
]
```

モデル評価ジョブのデータ暗号化

> **Note**
>
> Amazon Bedrock のモデル評価はプレビューリリースであり、変更される可能性があります。モデル評価ジョブを使用するには、米国東部 (バージニア北部) リージョンまたは米国西部 (オレゴン) リージョンを使用する必要があります。

Amazon Bedrock モデル評価ジョブでは、顧客データは転送中も保存中も常に暗号化されます。モデル評価ジョブでは、Amazon Bedrock のモデル評価ジョブの入力または出力として使用されるデータにカスタマーマネージドキーを使用できます。モデル評価ジョブの作成時にカスタマーマネージドキーを指定しない場合、Amazon S3 のデフォルトの AWSマネージドキーが暗号化に使用されます。Amazon Bedrock は、モデル評価ジョブ中にプロンプトデータセットなどの中間顧客データを保存し、保存時にはサービスマネージド KMS キーを使用して暗号化します。この中間データは、モデル評価ジョブが完了すると完全に削除されます。

Amazon ベッドロックのナレッジベース

Amazon Bedrock のナレッジベースでは、データソースを情報のリポジトリにまとめることができます。ナレッジベースを使用すると、検索拡張生成 (RAG) を活用したアプリケーションを簡単に構築できます。RAG は、データソースから情報を取得することでモデルレスポンスの生成を強化する手法です。設定後、ナレッジベースは次のように活用できます。

- RetrieveAndGenerateAPI を使用してナレッジベースにクエリを実行し、取得した情報から応答を生成するように RAG アプリケーションを設定します。
- ナレッジベースをエージェントに関連付けて (詳細については、「Agents for Amazon Bedrock」を参照) エージェントに RAG 機能を追加し、エージェントがエンドユーザーを支援するための手順を推論できるように支援する。
- Retrieve API を使用してアプリケーション内にカスタムオーケストレーションフローを作成し、ナレッジベースから直接情報を取得する。

ナレッジベースは、ユーザーのクエリに答えるだけでなく、プロンプトにコンテキストを提供することで基盤モデルに提供されるプロンプトを強化するためにも使用できます。ナレッジベースのレスポンスには引用も付いているため、ユーザーはレスポンスの基になっているテキストを正確に調べることで詳細な情報を見つけたり、そのレスポンスが意味のあるもので事実に基づく正確なものかどうかを確認したりできます。

ナレッジベースを設定し使用するには、次の手順に従います。

1. ソースドキュメントを集めてナレッジベースに追加する。
2. (オプション) ナレッジベースのクエリ中に結果をフィルタリングできるように、ソースドキュメントごとにメタデータファイルを作成します。
3. データを Amazon S3 バケットにアップロードします。
4. (オプション) サポートされているベクターストアにベクターインデックスを設定して、データにインデックスを付けます。Amazon Bedrock コンソールを使用して Amazon OpenSearch サーバーレスベクターデータベースを作成する予定の場合は、このステップをスキップできます。
5. ナレッジベースを作成して設定します。
6. 基盤モデルを使用して埋め込みを生成し、サポートされているベクトルストアに保存することでデータを取り込みます。
7. ナレッジベースにクエリを実行し拡張レスポンスを返すようにアプリケーションまたはエージェントを設定します。

トピック

仕組み

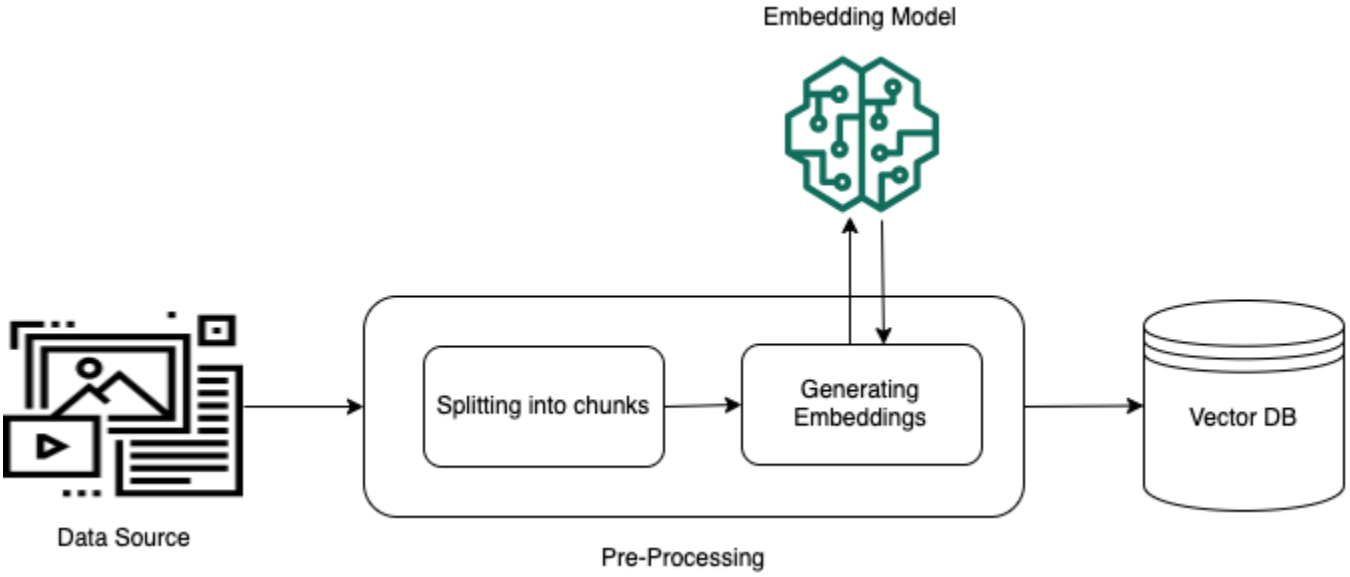
Amazon Bedrock のナレッジベースは、検索拡張生成 (RAG) を活用するのに役立ちます。RAG は、データストアから情報を取得して、大規模言語モデル (LLM) によって生成される応答を補強する一般的な手法です。データソースを使用してナレッジベースを設定すると、アプリケーションはナレッジベースにクエリを実行して情報を返し、ソースから直接引用するか、クエリ結果から生成される自然なレスポンスでクエリに答えることができます。

ナレッジベースを使用すると、ナレッジベースへのクエリから得られるコンテキストによって強化されたアプリケーションを構築できます。パイプライン構築という面倒な作業から解放され、アプリケーションのビルド時間を短縮する out-of-the-box RAG ソリューションを提供することで、市場投入までの時間を短縮できます。また、ナレッジベースを追加することで、プライベートデータを活用できるようにモデルを継続的にトレーニングする必要がなくなるため、費用対効果も向上します。

以下の図は、RAG がどのように実行されるかを概略的に示しています。ナレッジベースは、このプロセスのいくつかのステップを自動化することで、RAG の設定と実装を簡素化します。

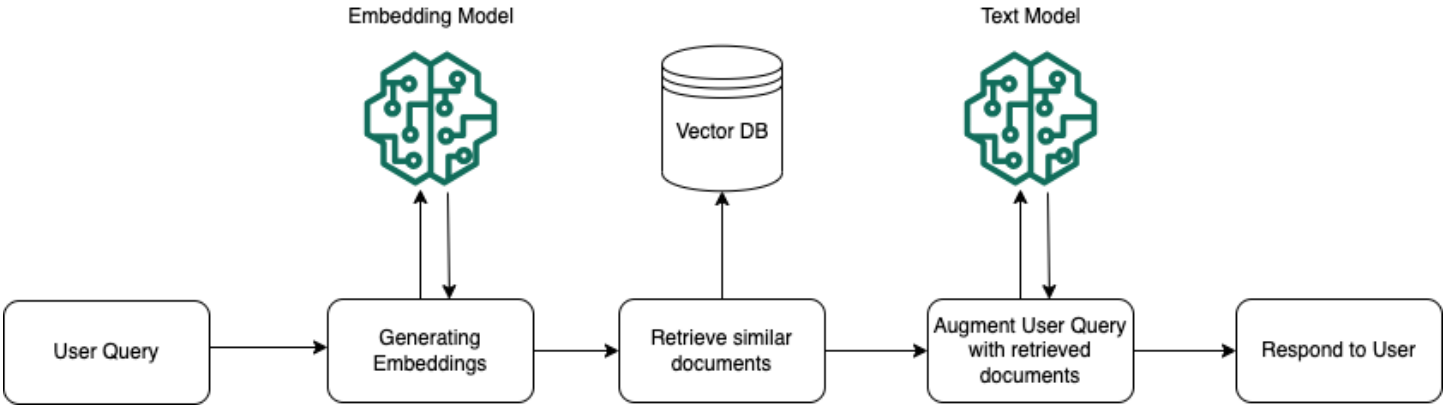
データを前処理する

プライベートデータから効果的に関連ドキュメントを取得できるようにするには、まずドキュメントを管理しやすいチャンクに分割するのが一般的です。その後、チャンクは埋め込みに変換され、元のドキュメントへのマッピングを維持したままベクトルインデックスに書き込まれます。これらの埋め込みは、クエリとデータソースからのテキストの意味上の類似性を判断するために使用されます。以下の図は、ベクトルデータベース用のデータの前処理を示しています。



ランタイムの実行

実行時には、埋め込みモデルを使用してユーザーのクエリをベクトルに変換します。次に、ドキュメントベクトルとユーザークエリベクトルを比較して、ベクトルインデックスをクエリしてユーザークエリの検索対象に意味的に類似したチャンクを検索します。最後のステップでは、ベクトルインデックスから取得したチャンクの追加コンテキストがユーザープロンプトに追加されます。その後、追加のコンテキストと共にプロンプトがモデルに送信され、ユーザーへのレスポンスが生成されます。以下の画像は、RAG が実行時にどのように動作してユーザークエリへのレスポンスを補強するかを示しています。



Amazon Bedrock のナレッジベースでサポートされているリージョンとモデル

Amazon Bedrock のナレッジベースは、以下のリージョンでサポートされています。

リージョン
米国東部 (バージニア北部)
米国西部 (オレゴン)

以下のモデルを使用して、データソースをベクターストアに埋め込むことができます。

モデル名	モデル ID
Amazon Titan Embeddings G1 - Text	アマゾン。 titan-embed-text-v1
CohereEmbed(英語)	コヒーレ。 embed-english-v3
CohereEmbed(多言語)	コヒーレ。 embed-multilingual-v3

ナレッジベースから情報を取得した後、以下のモデルを使用してレスポンスを生成できます。

モデル	モデル ID
AnthropicClaudev2.0	anthropic.claude-v2
AnthropicClaude2.1	アンソロピック・クロード-V 2:1
AnthropicClaude 3 Sonnetv1	anthropic.claude-3-sonnet-20240229-v 1:0
AnthropicClaude Instantv1	人為的。 claude-instant-v1

Amazon Bedrock のナレッジベースの前提条件

ナレッジベースを作成するには、以下の前提条件を満たす必要があります。

1. ナレッジベースに含めたい情報を含むファイルを準備して、ナレッジベースのデータソースを作成します。次に、ファイルを Amazon S3 バケットにアップロードします。

2. (オプション) 任意のベクターストアを設定します。を使用して Amazon OpenSearch Serverless にベクターストアを自動的に作成する予定がある場合は、 AWS Management Console この前提条件をスキップできます。
3. (オプション) の手順に従って、適切な権限を持つカスタム AWS Identity and Access Management (IAM) サービスロールを作成します。Amazon Bedrock のナレッジベースのサービスロールを作成するを使用して自動的にサービスロールを作成する予定の場合は、 AWS Management Console この前提条件を省略できます。
4. (オプション) の手順に従って、追加のセキュリティ設定を設定します。ナレッジベースリソースの暗号化

トピック

- ナレッジベースのデータソースを設定する
- サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する

ナレッジベースのデータソースを設定する

データソースには、ナレッジベースにクエリを実行したときに取得できる情報を含むファイルが含まれています。ソースドキュメントファイルを Amazon S3 バケットにアップロードして、ナレッジベースのデータソースを設定します。

各ソースドキュメントファイルが以下の要件を満たしていることを確認します。

- ファイルは以下のサポートされている形式のいずれかである必要があります。

形式	拡張機能
プレーンテキスト	.txt
マークダウン	.md
HyperText マークアップ言語	.html
Microsoft Word ドキュメント	.doc/.docx
カンマで区切られた値	.csv
Microsoft Excel スプレッドシート	.xls/.xlsx

形式	拡張機能
ポータブルドキュメント	.pdf

- ファイルサイズは 50 MB のクォータを超えません。

以下のトピックでは、データソースを準備するためのオプション手順について説明します。

トピック

ファイルにメタデータを追加してフィルタリングできるようにします。

オプションで、データソース内のファイルにメタデータを追加できます。メタデータを使用すると、ナレッジベースのクエリ中にデータをフィルタリングできます。

メタデータファイルの要件

ファイルのメタデータをデータソースに含めるには、metadataAttributes各メタデータ属性のキーと値のペアを持つオブジェクトにマップするフィールドで構成される JSON ファイルを作成します。次に、Amazon S3 バケット内のソースドキュメントファイルと同じフォルダにアップロードします。メタデータファイルの一般的な形式を以下に示します。

```
{
   "metadataAttributes": {
      "${attribute1}": "${value1}",
      "${attribute2}": "${value2}",
      ...
   }
}
```

属性の値には以下のデータ型がサポートされています。

- 文字列
- 数
- ブール値

各メタデータファイルが以下の要件を満たしていることを確認してください。

- このファイルは、関連するソースドキュメントファイルと同じ名前です。
- .metadata.jsonファイル拡張子の後に追加します (たとえば、*A.txt* という名前のファイルがある場合、*############.txt.Metadata.JSON* である必要があります)。
- ファイルサイズは 10 KB のクォータを超えないようにしてください。
- このファイルは、Amazon S3 バケット内の関連するソースドキュメントファイルと同じフォルダにあります。

> **Note**
>
> Amazon OpenSearch Serverless ベクターストアの既存のベクターインデックスにメタデータを追加する場合は、faissベクターインデックスがエンジンでフィルタリングできるように設定されていることを確認してください。nmslibベクトルインデックスがエンジンで設定されている場合は、次のいずれかを実行する必要があります。
>
> - コンソールで新しいナレッジベースを作成すると、Amazon Bedrock が Amazon OpenSearch サーバーレスのベクターインデックスを自動的に作成してくれます。
> - ベクターストアに別のベクターインデックスを作成し、**faiss**エンジンとして選択します。次に、新しいナレッジベースを作成し、新しいベクトルインデックスを指定します。
>
> Amazon Aurora データベースクラスター内の既存のベクターインデックスにメタデータを追加する場合は、取り込みを開始する前に、メタデータファイルの各メタデータ属性の列をテーブルに追加する必要があります。メタデータ属性値はこれらの列に書き込まれます。

データソースを同期すると、ナレッジベースのクエリ中に結果をフィルタリングできます。

メタデータファイルの例

たとえば、ニュース記事を含む *oscars-coverage_20240310.pdf* という名前のソースドキュメントがある場合、*######################*。このファイルのメタデータを作成するには、次の手順を実行します。

1. 次の内容を含む *oscars-coverage_20240310.pdf.metadata.json* という名前のファイルを作成します。

```
{
    "metadataAttributes": {
        "genre": "entertainment",
        "year": 2024
    }
}
```

2. *oscars-coverage_20240310.pdf.metadata.json # Amazon S3 ##### oscars-coverage_20240310.pdf ##################*
3. ナレッジベースを作成するまだ行っていない場合。次に、データソースを同期します。

ソースチャンク

データをナレッジベースに取り込む際に、Amazon Bedrock は各ファイルをチャンクに分割します。チャンクは、そのチャンクが属するナレッジベースにクエリを実行したときに返されるデータソースからの抜粋を指します。

Amazon Bedrock には、データをチャンク化するために使用できるチャンク化戦略が用意されています。また、ソースファイルを自分でチャンク化してデータを前処理することもできます。データソースには次のチャンク化戦略のどれを使用するかを検討してください。

- デフォルトのチャンク化 — デフォルトでは、Amazon Bedrock はソースデータを自動的にチャンクに分割し、各チャンクには最大で約 300 個のトークンが含まれます。300 個未満のトークンしか含まれない場合、ドキュメントは分割されません。
- 固定サイズのチャンク化 — Amazon Bedrock は、ソースデータをユーザーが設定したおおよそのサイズのチャンクに分割します。
- チャンク化なし — Amazon Bedrock は各ファイルを 1 つのチャンクとして扱います。このオプションを選択した場合、Amazon S3 バケットにアップロードする前に、ドキュメントを別々のファイルに分割して前処理することをお勧めします。

サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する

サポートされているベクターインデックスを設定してデータソースのインデックスを作成するには、以下のデータを保存するフィールドを作成します。

- 選択した埋め込みモデルによってデータソース内のテキストから生成されたベクトル。

- データソース内のファイルから抽出されたテキストチャンク。
- Amazon Bedrock が管理するナレッジベースに関連するメタデータ。
- (Amazon Aurora データベースを使用していて、フィルタリングを設定する場合) ソースファイルに関連付けるメタデータ。他のベクトルストアでフィルタリングを設定する予定であれば、これらのフィールドをフィルタリング用に設定する必要はありません。

ベクターインデックスの作成に使用するサービスに対応するタブを選択します。

> **Note**
>
> Amazon Bedrock が Amazon OpenSearch サーバーレスで自動的にベクターインデックスを作成することを希望する場合は、この前提条件をスキップしてに進んでください。ナレッジベースを作成するベクターインデックスの設定方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Amazon OpenSearch Serverless

1. 権限を設定し、Amazon OpenSearch Serverless でベクター検索コレクションを作成するには AWS Management Console、『Amazon OpenSearch Service 開発者ガイド』の「ベクター検索コレクションの使用」のステップ 1 と 2 に従います。コレクションを設定する際は、以下の考慮事項に注意してください。

   a. コレクションには任意の名前と説明を付けてください。

   b. コレクションを非公開にするには、「セキュリティ」セクションで「標準作成」を選択します。次に、ネットワークアクセス設定セクションで、アクセスタイプとして VPC を選択し、VPC エンドポイントを選択します。Amazon OpenSearch サーバーレスコレクションの VPC エンドポイントの設定の詳細については、Amazon OpenSearch Service 開発者ガイドの「インターフェイスエンドポイントを使用して Amazon OpenSearch サーバーレスにアクセスする (AWS PrivateLink)」を参照してください。
2. コレクションが作成されたら、ナレッジベースを作成するときのためにコレクション ARN を書き留めておきます。
3. 左側のナビゲーションペインで、「サーバーレス」の「コレクション」を選択します。次に、ベクター検索コレクションを選択します。
4. 「インデックス」タブを選択します。次に [ベクトルインデックスを作成] を選択します。

5. 「ベクターインデックスの詳細」セクションの「ベクターインデックス名」フィールドにインデックスの名前を入力します。
6. 「ベクターフィールド」セクションで、「ベクターフィールドを追加」を選択します。Amazon Bedrock は、データソースのベクター埋め込みをこのフィールドに保存します。以下の設定を行います。
   - ベクターフィールド名 — フィールドの名前 (例:**`embeddings`**) を指定します。
   - エンジン — 検索に使用されるベクトルエンジン。[失敗] を選択します。
   - ディメンション - ベクトルのディメンション (次元) の数。次の表を参照して、ベクトルに含まれるべき次元数を決定してください。

モデル	ディメンション
TitanG1 埋め込み-テキスト	1,536
CohereEmbed[英語]	1,024
CohereEmbed多言語	1,024

   - 距離メトリクス - ベクトル間の類似性を測定するために使用されるメトリクス。ユークリッドを使うことをおすすめします。
7. メタデータ管理セクションを拡張し、ナレッジベースがベクターで取得できる追加のメタデータを保存するようにベクターインデックスを設定するフィールドを2つ追加します。以下の表では、フィールドと各フィールドに指定する値について説明しています。

フィールド説明	マッピングフィールド	データタイプ	フィルター可能
Amazon Bedrock はデータから未加工のテキストをチャンク化し、チャンクをこのフィールドに保存します。	任意の名前 (例:) **`text`**	文字列	True

フィールド説明	マッピングフィールド	データタイプ	フィルター可能
Amazon Bedrock は、このフィールドのナレッジベースに関連するメタデータを保存します。	任意の名前 (例:) **bedrock-metadata**	文字列	False

8. ナレッジベースを作成するときのために、ベクターインデックス名、ベクターフィールド名、メタデータ管理マッピングフィールド名に選択した名前を書き留めておきます。次に [作成] を選択します。

ベクターインデックスを作成したら、ナレッジベースの作成に進むことができます。次の表は、メモしておいた各情報をどこに入力するかをまとめたものです。

フィールド	ナレッジベース設定 (コンソール) の対応するフィールド	ナレッジベース設定 (API) の対応するフィールド	説明
コレクション ARN	コレクション ARN	CollectionARN	ベクター検索コレクションの Amazon リソースネーム (ARN)。
ベクターインデックス名	ベクターインデックス名	vectorIndexName	ベクトルインデックスの名前。
ベクターフィールド名	ベクトルフィールド	ベクターフィールド	データソースのベクター埋め込みを格納するフィールドの名前。
メタデータ管理 (1 番目のマッピングフィールド)	テキストフィールド	テキストフィールド	データソースの未加工テキストを保存するフィールドの名前。

フィールド	ナレッジベース設定 (コンソール) の対応するフィールド	ナレッジベース設定 (API) の対応するフィールド	説明
メタデータ管理 (2 番目のマッピングフィールド)	岩盤管理メタデータフィールド	メタデータフィールド	Amazon Bedrock が管理するメタデータを保存するフィールドの名前。

Amazon OpenSearch Serverless でのベクターストアの設定に関する詳細なドキュメントについては、Amazon OpenSearch Service 開発者ガイドの「ベクター検索コレクションの使用」を参照してください。

Amazon Aurora

1. 「Aurora PostgreSQL をナレッジベースとして使用するための準備」の手順に従って、Amazon Aurora データベース (DB) クラスター、スキーマ、およびテーブルを作成します。テーブルを作成するときは、以下の列とデータ型を使用してテーブルを設定します。次の表に記載されている列名の代わりに、任意の列名を使用できます。ナレッジベースの設定時に指定できるように、使用した列名をメモしておきます。

列名	データタイプ	ナレッジベース設定 (コンソール) の対応するフィールド	ナレッジベース設定 (API) の対応するフィールド	説明
id	UUID プライマリキー	プライマリキー	`primaryKeyField`	各レコードに固有の識別子が含まれます。
埋め込み	ベクトル	ベクトルフィールド	`vectorField`	データソースのベクトル埋め込みが含まれます。
チャンク	テキスト	テキストフィールド	`textField`	データソースからの未加工テキ

列名	データタイプ	ナレッジベース設定 (コンソール) の対応するフィールド	ナレッジベース設定 (API) の対応するフィールド	説明
				ストのチャンクが含まれます。
metadata	JSON	岩盤管理メタデータフィールド	`metadataField`	ソースアトリビューションを実行し、データインジェストとクエリを可能にするために必要なメタデータが含まれています。

2. (オプション) フィルタリング用にファイルにメタデータを追加した場合は、ファイル内のメタデータ属性ごとに列を作成し、データタイプ (テキスト、数値、またはブーリアン) を指定する必要もあります。たとえば、`genre`属性がデータソースに存在する場合、`text`データ型として `genre` and を指定する列を追加します。取り込み中、これらの列には対応する属性値が入力されます。
3. 「Amazon Aurora でのパスワード管理」の手順に従って、Aurora DB AWS Secrets Manager クラスターのシークレットを設定します。 AWS Secrets Manager
4. DB クラスターを作成しシークレットを設定したら、次の情報をメモします。

ナレッジベース設定 (コンソール) のフィールド	ナレッジベース設定 (API) のフィールド	説明
Amazon Aurora DB クラスターの ARN	resourceArn	DB クラスターの ARN。
データベース名	databaseName	データベースの名前
テーブル名	tableName	DB クラスター内のテーブル名

ナレッジベース設定 (コンソール) のフィールド	ナレッジベース設定 (API) のフィールド	説明
シークレット ARN	credentialsSecretArn	DB AWS Secrets Manager クラスターのキーの ARN

Pinecone

> **Note**
>
> を使用する場合Pinecone、ベクターストアサービスを提供するために、 AWS お客様に代わって指定されたサードパーティのソースにアクセスすることを許可することに同意したものとみなされます。お客様は、サードパーティーサービスからのデータの使用および転送に適用されるいかなるサードパーティー規約をも遵守する必要があります。

でのベクターストアの設定に関する詳細なドキュメントについてはPinecone、「Amazon Bedrock のナレッジベースとしての Pinecone」を参照してください。

ベクトルストアを設定する際は、次の情報をメモしておきます。この情報は、ナレッジベースを作成するときに入力することになります。

- 接続文字列 — インデックス管理ページのエンドポイント URL。
- 名前空間 — (オプション) データベースに新しいデータを書き込むために使用する名前空間。詳細については、「Using namespaces」を参照してください。

インデックスを作成するときには、追加の設定が必要です。Pinecone

- 名前 - ベクトルインデックスの名前。任意の有効な名前を選択します。ここで選択した名前は、後でナレッジベースを作成するときに [ベクトルインデックス名] フィールドに入力します。
- ディメンション - ベクトルのディメンション (次元) の数。次の表を参照して、ベクトルに含めるべき次元数を決定してください。

モデル	ディメンション
TitanG1 埋め込み-テキスト	1,536
CohereEmbed[英語]	1,024
CohereEmbed多言語	1,024

- 距離メトリクス - ベクトル間の類似性を測定するために使用されるメトリクス。ユースケースに合わせてさまざまなメトリクスを試してみることをお勧めします。コサイン類似度から始めることをおすすめします。

Pineconeインデックスにアクセスするには、を使用して Amazon Bedrock に Pinecone API キーを提供する必要があります。 AWS Secrets Manager

設定用のシークレットを設定するには Pinecone

1. AWS Secrets Manager シークレットの作成の手順に従い、キーを API キーに、値を API Pinecone キーとして設定してインデックスにアクセスします。`apiKey`
2. この API キーを見つけるには、Pinecone コンソールを開いて [API キー] を選択します。
3. シークレットを作成したら、作成したシークレットの ARN をメモしておきます。
4. 「ナレッジベースを含むベクトルストアの AWS Secrets Manager シークレットを復号化するアクセス許可」の手順に従って、KMS キーの ARN を復号化する権限をサービスロールにアタッチします。
5. この ARN は、後でナレッジベースを作成するときに、[認証情報シークレット ARN] フィールドに入力します。

Redis Enterprise Cloud

> **Note**
>
> を使用する場合Redis Enterprise Cloud、ベクターストアサービスを提供するために、AWS お客様に代わって指定された第三者ソースへのアクセスを許可することに同意したものとみなされます。お客様は、第三者サービスからのデータの使用および転送に適用される第三者の規約を遵守する責任を負います。

でのベクターストアの設定に関する詳細なドキュメントについてはRedis Enterprise Cloud、「Amazon Bedrock Redis Enterprise Cloud との統合」を参照してください。

ベクトルストアを設定する際は、次の情報をメモしておきます。この情報は、ナレッジベースを作成するときに入力することになります。

- エンドポイント URL — データベースのパブリックエンドポイント URL。
- ベクターインデックス名 — データベースのベクターインデックスの名前。
- ベクターフィールド — ベクター埋め込みが保存されるフィールドの名前。次の表を参照して、ベクトルに含まれるべき次元数を決定してください。

モデル	ディメンション
TitanG1 埋め込み-テキスト	1,536
CohereEmbed[英語]	1,024
CohereEmbed多言語	1,024

- テキストフィールド — Amazon Bedrock が未加工テキストのチャンクを格納するフィールドの名前。
- Bedrock が管理するメタデータフィールド — Amazon Bedrock がナレッジベースに関連するメタデータを保存するフィールドの名前。

Redis Enterprise Cloudクラスターにアクセスするには、を使用して Amazon Bedrock Redis Enterprise Cloud にセキュリティ設定を提供する必要があります。 AWS Secrets Manager

設定用のシークレットをセットアップするには Redis Enterprise Cloud

1. 「Transport Layer Security (TLS)」の手順に従って、Amazon Bedrock でデータベースを使用するように TLS を有効にします。
2. 「AWS Secrets Manager シークレットの作成」の手順に従ってください。シークレットに設定した適切な値を使用してRedis Enterprise Cloud、以下のキーを設定します。
   - `username`— Redis Enterprise Cloud データベースにアクセスするためのユーザー名。自身のユーザー名を確認するには、Redis コンソール でデータベースの [セキュリティ] セクションを参照してください。

- `password`— Redis Enterprise Cloud データベースにアクセスするためのパスワード。自身のパスワードを確認するには、Redis コンソール でデータベースの [セキュリティ] セクションを参照してください。
- `serverCertificate` - Redis Cloud 認証機関からの証明書の内容。「Download CA certificates」の手順に従って、Redis 管理コンソールを使ってサーバー証明書をダウンロードします。
- `clientPrivateKey` - Redis Cloud 認証機関からの証明書のプライベートキー。「Download CA certificates」の手順に従って、Redis 管理コンソールを使ってサーバー証明書をダウンロードします。
- `clientCertificate` - Redis Cloud 認証機関からの証明書のパブリックキー。「Download CA certificates」の手順に従って、Redis 管理コンソールを使ってサーバー証明書をダウンロードします。

3. シークレットを作成したら、そのシークレットの ARN を書き留めます。この ARN は、後でナレッジベースを作成するときに、[認証情報シークレット ARN] フィールドに入力します。

ナレッジベースを作成する

> **Note**
>
> root ユーザーではナレッジベースを作成できません。これらのステップを開始する前に IAM ユーザーでログインしてください。

Amazon S3 のデータソースと任意のベクターストアを設定したら、ナレッジベースを作成できます。選択した方法に対応するタブを選択し、手順に従います。

Console

ナレッジベースを作成するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. [ナレッジベース] セクションで [ナレッジベースの作成] を選択します。
4. 「ナレッジベースの詳細を入力」ページで、以下の設定を行います。

a. (オプション) ナレッジベースの詳細セクションで、デフォルト名を変更し、ナレッジベースの説明を入力します。

b. IAM 権限セクションで、他のサービスにアクセスするための Amazon Bedrock 権限を付与する AWS Identity and Access Management (IAM) ロールを選択します。 AWS Amazon Bedrock にサービスロールを作成させることも、作成したカスタムロールを選択することもできます。

c. (オプション) ナレッジベースにタグを追加します。詳細については、「リソースのタグ付け」を参照してください。

d. [Next (次へ)] を選択します。

5. [データソースの設定] ページで、ナレッジベースに使用するデータソースの情報を入力します。

a. (オプション) 既定のデータソース名を変更します。

b. 準備したデータソースのファイルを含むオブジェクトの S3 URI を指定します。

> **Note**
>
> 作成するナレッジベースと同じリージョンにある Amazon S3 バケットを選択します。そうしないと、データソースの同期に失敗します。

c. Amazon S3 データをカスタマーマネージドキーで暗号化した場合は、[Amazon S3 AWS KMS データ用のカスタマーマネージドキーを追加] を選択し、Amazon Bedrock がそれを復号できるようにする KMS キーを選択します。詳細については、「Amazon OpenSearch Service に渡される情報の暗号化」を参照してください。

d. (オプション) 以下の詳細設定を行うには、「詳細設定-オプション」セクションを展開します。

i. データを埋め込みに変換する際、Amazon Bedrock AWS はデフォルトで所有および管理するキーを使用してデータを暗号化します。独自の KMS キーを使用するには、[詳細設定] を展開して [暗号化設定のカスタマイズ (詳細)] を選択し、キーを選択します。詳細については、「データインジェスト時の一時データストレージの暗号化」を参照してください。

ii. データソースのチャンキング戦略を以下のオプションから選択してください。

- デフォルトのチャンク化 — デフォルトでは、Amazon Bedrock はソースデータを自動的にチャンクに分割し、各チャンクには最大で 300 個のトークンが含まれ

ます。300 個未満のトークンしか含まれない場合、ドキュメントは分割されません。

- 固定サイズのチャンク化 — Amazon Bedrock は、ソースデータをユーザーが設定したおおよそのサイズのチャンクに分割します。次のオプションを設定します。
  - 最大トークン — Amazon Bedrock は、選択したトークン数以下のチャンクを作成します。
  - チャンク間のオーバーラップ率 — 各チャンクは、選択した割合だけ連続するチャンクと重複します。
- チャンク化なし — Amazon Bedrock は各ファイルを 1 つのチャンクとして扱います。このオプションを選択した場合、ドキュメントを別々のファイルに分割して前処理することをお勧めします。

> **Note**
>
> データソースの作成後は、チャンク化戦略を変更することはできません。

e. [次へ] を選択します。

6. 「埋め込みモデル」セクションで、サポートされている埋め込みモデルを選択し、データをナレッジベースのベクター埋め込みに変換します。
7. ベクターデータベースセクションで、以下のオプションのいずれかを選択して、ナレッジベース用のベクター埋め込みを保存します。
   - 新しいベクターストアをすばやく作成 — Amazon Bedrockは、 OpenSearch お客様に代わってAmazonサーバーレスベクター検索コレクションを作成します。このオプションでは、必要なフィールドと設定を含むパブリックベクター検索コレクションとベクターインデックスが自動的に設定されます。コレクションが作成されたら、Amazon OpenSearch サーバーレスコンソールまたは AWS API を使用して管理できます。詳細については、Amazon OpenSearch Service 開発者ガイドの「ベクター検索コレクションの使用」を参照してください。このオプションを選択すると、オプションで以下の設定を有効にできます。

     a. インフラストラクチャに障害が発生してもベクターストアの可用性が損なわれないように、冗長なアクティブレプリカを有効にするには、「冗長性を有効にする (アクティブレプリカ)」を選択します。

> **Note**
>
> ナレッジベースをテストする間は、このオプションを無効のままにしておくことをお勧めします。本番環境にデプロイする準備ができたら、冗長アクティブレプリカを有効にすることをお勧めします。料金については、「サーバーレスの料金表」を参照してください。 OpenSearch

b. 自動ベクターストアをカスタマーマネージドキーで暗号化するには、「Amazon OpenSearch Serverless vector (オプション) にカスタマー管理 KMS キーを追加 (オプション)」を選択し、キーを選択します。詳細については、「Amazon OpenSearch Service に渡される情報の暗号化」を参照してください。

- 作成したベクトルストアを選択 — 作成済みのベクトルデータベースを含むサービスを選択します。Amazon Bedrock が埋め込みを保存、更新、管理できるようにナレッジベースの情報をデータベースにマッピングするためのフィールドに入力します。これらのフィールドと作成したフィールドとのマッピング方法の詳細については、「」を参照してください。サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する

> **Note**
>
> Amazon OpenSearch Serverless または Amazon Aurora のデータベースを使用する場合は、事前に [メタデータフィールドマッピング] でフィールドを設定しておく必要があります。Pineconeまたはのデータベースを使用する場合Redis Enterprise Cloud、ここでこれらのフィールドの名前を指定すると、Amazon Bedrock がベクターストアにそれらの名前を動的に作成します。

8. [次へ] を選択します。
9. [確認および作成] ページで、ナレッジベースの設定と詳細を確認します。変更が必要なセクションで [編集] を選択します。問題がなければ、[ナレッジベースを作成] を選択します。
10. ナレッジベースの作成にかかる時間は、入力したデータの量に左右されます。ナレッジベースの作成が完了すると、ナレッジベースのステータスが「準備完了」に変わります。

API

ナレッジベースを作成するには、Agents for Amazon Bedrock CreateKnowledgeBaseのビルド時エンドポイントにリクエストを送信し、名前、説明、実行内容の説明、オーケストレーションに使用する基盤モデルを指定します。

> **Note**
>
> Amazon Bedrock にAmazon OpenSearch サービス内のベクターストアの作成と管理を任せたい場合は、コンソールを使用してください。詳細については、「ナレッジベースを作成する」を参照してください。

- ナレッジベースを作成するアクセス許可がある ARN を `roleArn` フィールドで指定します。
- 使用する埋め込みモデルを `knowledgeBaseConfiguration` オブジェクトの `embeddingModelArn` フィールドで指定します。
- ベクトルストアの構成を `storageConfiguration` オブジェクトで指定します。詳細については、「サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する」を参照してください。
  - Amazon OpenSearch Service データベースの場合は、`opensearchServerlessConfiguration`オブジェクトを使用してください。
  - Pineconeデータベースの場合は、`pineconeConfiguration`オブジェクトを使用してください。
  - Redis Enterprise Cloudデータベースの場合は、`redisEnterpriseCloudConfiguration`オブジェクトを使用してください。
  - Amazon Aurora データベースの場合は、`rdsConfiguration`オブジェクトを使用してください。

ナレッジベースを作成したら、S3 バケットからナレッジベースのファイルを含むデータソースを作成します。データソースを作成するには、CreateDataSourceリクエストを送信します。

- データソースファイルを含む S3 `dataSourceConfiguration` バケットの情報をフィールドに入力します。
- `vectorIngestionConfiguration`データソースをチャンク化する方法をフィールドで指定します。詳細については、「ナレッジベースのデータソースを設定する」を参照してください。

> **Note**
>
> データソースを作成した後でチャンク設定を変更することはできません。

- (オプション) データを埋め込みに変換する際、Amazon Bedrock AWS はデフォルトで所有および管理するキーを使用してデータを暗号化します。独自の KMS キーを使用するには、そのキーをオブジェクトに含めます。`serverSideEncryptionConfiguration`詳細については、「ナレッジベースリソースの暗号化」を参照してください。

ナレッジベースのセキュリティ設定を設定します。

ナレッジベースを作成したら、次のセキュリティ構成を設定しなければならない場合があります。

トピック

- ナレッジベースのデータアクセスポリシーを設定します。
- Amazon OpenSearch サーバーレスナレッジベースのネットワークアクセスポリシーを設定する

ナレッジベースのデータアクセスポリシーを設定します。

カスタムロールを使用している場合は、新しく作成したナレッジベースのセキュリティ設定を設定してください。Amazon Bedrock にサービスロールを作成させる場合は、このステップをスキップできます。設定するデータベースに対応するタブの手順に従います。

Amazon OpenSearch Serverless

Amazon OpenSearch Serverless コレクションへのアクセスをナレッジベースのサービスロールに制限するには、データアクセスポリシーを作成します。そのためには以下の方法があります。

- Amazon OpenSearch サービスコンソールを使用するには、Amazon OpenSearch サービス開発者ガイドの「データアクセスポリシーの作成 (コンソール)」の手順に従います。
- AWS API を使用するには、CreateAccessPolicyOpenSearch サーバーレスエンドポイントを使用してリクエストを送信します。 AWS CLI 例については、「データアクセスポリシーの作成 (AWS CLI)」を参照してください。

以下のデータアクセスポリシーを使用して、Amazon OpenSearch Serverless コレクションとサービスロールを指定します。

```
[
    {
        "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

PineconeRedis Enterprise Cloudまたはベクターインデックスを統合するには、次の ID ベースのポリシーをナレッジベースのサービスロールにアタッチして、 AWS Secrets Manager ベクターインデックスのシークレットへのアクセスを許可します。

```
{
    "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}"
```

```
            }
        }
    }]
}
```

Amazon OpenSearch サーバーレスナレッジベースのネットワークアクセスポリシーを設定する

ナレッジベースにプライベート Amazon OpenSearch Serverless コレクションを使用する場合、AWS PrivateLink VPC エンドポイントからのみアクセスできます。Amazon OpenSearch サーバーレスベクターコレクションをセットアップするときにプライベート Amazon OpenSearch サーバーレスコレクションを作成することも、ネットワークアクセスポリシーを設定するときに既存の Amazon OpenSearch サーバーレスコレクション (Amazon Bedrock コンソールが作成したものを含む) をプライベートにすることもできます。

Amazon OpenSearch Service 開発者ガイドの以下のリソースは、プライベート Amazon OpenSearch サーバーレスコレクションに必要な設定を理解するのに役立ちます。

- プライベート Amazon OpenSearch Serverless コレクションの VPC エンドポイントの設定の詳細については、「インターフェイスエンドポイントを使用して Amazon OpenSearch Serverless にアクセスする ()」を参照してください。AWS PrivateLink
- Amazon サーバーレスのネットワークアクセスポリシーの詳細については、「Amazon OpenSearch サーバーレスのネットワークアクセス」を参照してください。 OpenSearch

Amazon Bedrock ナレッジベースがプライベート Amazon OpenSearch サーバーレスコレクションにアクセスできるようにするには、Amazon OpenSearch サーバーレスコレクションのネットワークアクセスポリシーを編集して Amazon Bedrock をソースサービスとして許可する必要があります。選択した方法に対応するタブを選択し、手順に従います。

Console

1. https://console.aws.amazon.com/aos/ にある Amazon OpenSearch サービスコンソールを開きます。
2. 左側のナビゲーションペインから [コレクション] を選択します。次に、コレクションを選択します。
3. 「ネットワーク」セクションで、関連ポリシーを選択します。
4. [編集] を選択します。

5. [ポリシー定義方法の選択] で [JSON] を選択します。
6. JSON エディターで、次のポリシーを貼り付けます。

```
[
    {
        "AllowFromPublic": false,
        "Description":"${network access policy description}",
        "Rules":[
            {
                "ResourceType": "collection",
                "Resource":[
                    "collection/${collection-id}"
                ]
            },
        ],
        "SourceServices":[
            "bedrock.amazonaws.com"
        ]
    }
]
```

7. [更新] を選択します。

API

Amazon OpenSearch Serverless コレクションのネットワークアクセスポリシーを編集するには、以下を実行します。

1. GetSecurityPolicyOpenSearch サーバーレスエンドポイントを使用してリクエストを送信します。nameポリシーを指定し、を type as network と指定します。応答内の policyVersion を書き留めます。
2. UpdateSecurityPolicyOpenSearch サーバーレスエンドポイントでリクエストを送信します。少なくとも、以下のフィールドを指定してください。

フィールド	説明
name	ポリシーの名前
ポリシー/バージョン	policyVersion レスポンスから返送されました。GetSecurityPolicy

フィールド	説明
type	セキュリティポリシーのタイプ。network を指定します。
ポリシー	使用するポリシー。次の JSON オブジェクトを指定します。

```
[
    {
        "AllowFromPublic": false,
        "Description":"${network access policy description}",
        "Rules":[
            {
                "ResourceType": "collection",
                "Resource":[
                    "collection/${collection-id}"
                ]
            },
        ],
        "SourceServices":[
            "bedrock.amazonaws.com"
        ]
    }
]
```

AWS CLI 例については、「データアクセスポリシーの作成 (AWS CLI)」を参照してください。

- Amazon OpenSearch Service コンソールを使用するには、「ネットワークポリシーの作成 (コンソール)」の手順に従います。ネットワークポリシーを作成する代わりに、コレクションの詳細の Network サブセクションにある関連ポリシーを書き留めてください。

同期してデータソースをナレッジベースに取り込む

ナレッジベースを作成したら、データソースをナレッジベースに取り込み、インデックスを作成してクエリできるようにします。取り込みにより、データソース内の未加工データがベクター埋め込みに変換されます。また、未加工のテキストとフィルタリング用に設定した関連メタデータを関連付け

て、クエリ処理を効率化します。取り込みを開始する前に、データソースが次の条件を満たしていることを確認してください。

- データソースの Amazon S3 バケットは、ナレッジベースと同じリージョンにあります。
- ファイルはサポートされている形式です。詳細については、「サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する」を参照してください。
- ファイルは最大ファイルサイズの 50 MB を超えません。詳細については、「ナレッジベースのクォータ」を参照してください。
- データソースにメタデータファイルが含まれている場合は、次の条件をチェックして、メタデータファイルが無視されないことを確認してください。
  - `.metadata.json`各ファイルは、関連付けられているソースファイルと同じ名前を共有します。
  - ナレッジベースのベクターインデックスが Amazon OpenSearch Serverless ベクターストアにある場合は、`faiss`ベクターインデックスがエンジンで設定されていることを確認してください。`nmslib`ベクトルインデックスがエンジンで設定されている場合は、次のいずれかを実行する必要があります。
    - コンソールで新しいナレッジベースを作成すると、Amazon Bedrock が Amazon OpenSearch サーバーレスのベクターインデックスを自動的に作成してくれます。
    - ベクターストアに別のベクターインデックスを作成し、**faiss**エンジンとして選択します。次に、新しいナレッジベースを作成し、新しいベクトルインデックスを指定します。
  - ナレッジベースのベクターインデックスが Amazon Aurora データベースクラスターにある場合は、取り込みを開始する前に、インデックスのテーブルにメタデータファイルの各メタデータプロパティの列が含まれていることを確認してください。

> **Note**
>
> データソースの S3 バケットにファイルを追加、変更、または削除するたびに、データソースを同期してナレッジベースに再インデックスされるようにする必要があります。同期は段階的に行われるため、Amazon Bedrock は前回の同期以降に追加、変更、または削除された S3 バケット内のオブジェクトのみを処理します。

データソースをナレッジベースに取り込む方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

データソースを取り込むには

1. Amazon Bedrock コンソール (https://console.aws.amazon.com/bedrock/) を開きます。
2. 左側のナビゲーションペインの [ナレッジベース] でナレッジベースを選択します。
3. [データソース] セクションで [同期] を選択して、データインジェストを開始します。
4. データインジェストが正常に完了すると、緑色の成功バナーが表示されます。
5. データソースを選択して、そのデータソースの [同期履歴] を表示することができます。[警告を表示] を選択すると、データインジェストジョブが失敗した理由を確認できます。

API

ナレッジベース用に設定したベクターストアにデータソースを取り込むには、Agents for Amazon Bedrock StartIngestionJobビルドタイムエンドポイントでリクエストを送信します。とを指定します。knowledgeBaseId dataSourceId

Agents for Amazon Bedrock ingestionJobIdGetIngestionJobビルドタイムエンドポイントのリクエストでレスポンスで返されたものを使用して、取り込みジョブのステータスを追跡します。さらに、とを指定します。knowledgeBaseId dataSourceId

- 取り込みジョブが完了すると、レスポンス内の status は COMPLETE になります。
- レスポンス内の statistics オブジェクトは、データソース内のドキュメントの取り込みが成功したかどうかに関する情報を返します。

Agents for Amazon Bedrock ListIngestionJobsのビルド時エンドポイントでリクエストを送信することで、データソースのすべての取り込みジョブの情報を確認することもできます。dataSourceIdknowledgeBaseIdデータを取り込むナレッジベースのとを指定します。

- 検索するステータスを filters オブジェクトで指定して、結果をフィルタリングします。
- sortBy オブジェクトを指定して、ジョブの開始時間またはジョブのステータスでソートします。昇順または降順で並べ替えることができます。
- レスポンスで返す結果の最大数を maxResults フィールドで設定します。設定した数よりも多くの結果がある場合、レスポンスからが返され、ListIngestionJobs次のジョブバッチを確認するために別のリクエストで送信できます。nextToken

Amazon Bedrock でナレッジベースをテストしましょう

ナレッジベースを設定したら、クエリを送信してレスポンスを確認することで、ナレッジベースの動作をテストできます。クエリ構成を設定して情報取得をカスタマイズすることもできます。ナレッジベースの動作に満足したら、ナレッジベースにクエリを実行するか、ナレッジベースをエージェントにアタッチするようにアプリケーションを設定できます。

トピックを選択すると、そのトピックの詳細が表示されます。

トピック

ナレッジベースに問い合わせて結果を返すか、応答を生成する

ナレッジベースを検索する方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

ナレッジベースをテストするには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. ナレッジベースセクションで、以下のいずれかのアクションを行います。
   - テストするナレッジベースの横にあるラジオボタンを選択して、[ナレッジベースをテストする] を選択します。テストウィンドウが右から展開されます。
   - テストするナレッジベースを選択します。テストウィンドウが右から展開されます。
4. ユースケースに応じて、「クエリに応答を生成」を選択または選択解除します。
   - ナレッジベースから直接取得した情報を返すには、「回答を生成」をオフにします。Amazon Bedrock は、クエリに関連するテキストチャンクをデータソースから返します。
   - ナレッジベースから取得した情報に基づいてレスポンスを生成するには、「レスポンス生成」をオンにしてください。Amazon Bedrock は、データソースに基づいて回答を生成し、提供された情報を脚注とともに引用します。

5. [レスポンスを生成] をオンにした場合は、[モデルを選択] を選択して、レスポンス生成に使用するモデルを選択します。次に [適用] を選択します。
6. (オプション) 設定アイコン

( )
を選択し、「設定」を開きます。以下の設定を変更できます。

- 検索タイプ — ナレッジベースのクエリ方法を指定します。詳細については、「検索タイプ」を参照してください。
- ソースチャンクの最大数 — 取得するソースチャンクの最大数を指定します。詳細については、「ソースチャンクの最大数」を参照してください。
- フィルター — ファイルのメタデータに使用するフィルターグループを 5 つまで、各グループ内で最大 5 つのフィルターを指定します。詳細については、「メタデータとフィルタリング」を参照してください。
- ナレッジベースプロンプトテンプレート —「回答を生成」をオンにすると、デフォルトのプロンプトテンプレートを独自のテンプレートに置き換えて、応答を生成するためにモデルに送信されるプロンプトをカスタマイズできます。詳細については、「ナレッジベースのプロンプトテンプレート」を参照してください。

7. チャットウィンドウのテキストボックスにクエリを入力して [実行] を選択すると、ナレッジベースからのレスポンスが表示されます。
8. 応答は次の方法で調べることができます。

- 応答を生成しなかった場合は、関連性の高い順にテキストチャンクが直接返されます。
- 回答を生成した場合は、脚注を選択すると、回答のその部分の引用元からの抜粋が表示されます。リンクを選択すると、ファイルが含まれている S3 オブジェクトに移動します。
- 各脚注で引用されている部分の詳細を表示するには、[ソースの詳細を表示] を選択します。ソース詳細ペインでは以下のアクションを実行できます。
  - クエリに設定した構成を確認するには、[クエリ構成] を展開します。
  - ソースチャンクの詳細を表示するには、その横にある右矢印

    ( )
    を選択して展開します。以下の情報が表示されます。

- ソースチャンクからの未加工テキスト。このテキストをコピーするには、コピーアイコン
  ()
  を選択します。ファイルを含む S3 オブジェクトに移動するには、外部リンクアイコン
  ()
  を選択します。
- ソースチャンクに関連付けられたメタデータ。属性キーと値は、`.metadata.json`ソースドキュメントに関連付けられたファイルで定義されます。詳細については、「メタデータファイルの要件」を参照してください。

チャットオプション

1. 回答を生成する場合は、[モデルを変更] を選択して別のモデルを応答生成できます。モデルを変更すると、チャットウィンドウ内のテキストは完全に消去されます。
2. [回答を生成] を選択または選択解除して、クエリに対する回答を生成するか、直接見積もりを返すかを切り替えます。設定を変更すると、チャットウィンドウ内のテキストは完全に消去されます。
3. チャットウィンドウをクリアするには、ほうきのアイコン () を選択します。 
4. チャットウィンドウのすべての出力をコピーするには、コピーアイコン
   ()
   を選択します。

API

取得

ナレッジベースにクエリを実行し、データソースから関連するテキストのみを返すには、Agents for Amazon Bedrock ランタイムエンドポイントを使用してリクエスト (Retrieveリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。

次の表では、パラメータとリクエスト本文を簡単に説明しています (詳細とリクエスト構造については、「Retrieve リクエスト構文」を参照してください)。

変数	必須?	ユースケース
knowledgeBaseId	Yes	クエリするナレッジベースを指定するには
取得/クエリ	Yes	`text`クエリを指定するフィールドが含まれます。
nextToken	No	次のレスポンスを返すには
取得/設定	No	ベクター検索をカスタマイズするためのクエリ設定を含めること。

次の表では、レスポンス本文を簡単に説明しています (詳細とレスポンス構造については、Retrieve レスポンス構文を参照してください)。

変数	ユースケース
RetrievalResults	ソースチャンク、ソースの Amazon S3 の場所、`score`チャンクの関連性が含まれます。
nextToken	別のリクエストで使用して、次の結果のバッチを返すため。

RetrieveAndGenerate

ナレッジベースにクエリを実行し、基盤モデルを使用してデータソースからの結果に基づいてレスポンスを生成するには、Agents for Amazon Bedrock RetrieveAndGenerateランタイムエンドポイントを使用してリクエストを送信します。

次の表では、パラメータとリクエスト本文を簡単に説明しています (詳細とリクエスト構造については、RetrieveAndGenerate リクエスト構文を参照してください)。

変数	必須?	ユースケース
input	Yes	textクエリを指定するフィールドが含まれています。
retrieveAndGenerate設定	Yes	クエリするナレッジベース、応答生成に使用するモデル、およびオプションのクエリ設定を指定します。
sessionId	No	同じ値を使用して同じセッションを継続し、情報を維持します。
セッション設定	No	セッションを暗号化するための KMS キーを含めるには

次の表では、レスポンス本文を簡単に説明しています (詳細とレスポンス構造については、「Retrieve レスポンス構文」を参照してください)。

変数	ユースケース
引用	内の各オブジェクトで生成された応答の一部generatedResponsePart 、オブジェクト内のソースチャンク、contentおよびオブジェクトのオブジェクト内のソースの Amazon S3 ロケーションが含まれます。location retrievedReferences
output	生成されたレスポンス全体が含まれます。
sessionId	セッションの ID が含まれます。このIDを別のリクエストで再利用して、同じ会話を続けることができます。

> **Note**
>
> 応答の生成中にプロンプトが文字数制限を超えているというエラーが表示された場合は、次の方法でプロンプトを短くできます。
>
> - ソースチャンクの最大数を減らします (これにより、の $search_results$ プレースホルダーに入力される文字が短くなります)。ナレッジベースのプロンプトテンプレート
> - より小さなチャンクを使用するチャンク戦略でデータソースを再作成します (これにより、の $search_results$ プレースホルダーに入力される値が短くなります)。ナレッジベースのプロンプトテンプレート
> - プロンプトテンプレートを短くしてください。
> - ユーザークエリを短くします (これにより、の $query$ プレースホルダーに入力される内容が短くなります)。ナレッジベースのプロンプトテンプレート

クエリー設定

ナレッジベースに問い合わせるときに構成を変更して、検索と応答の生成をカスタマイズできます。設定の詳細や、コンソールや API での変更方法については、以下のトピックから選択してください。

検索タイプ

検索タイプは、ナレッジベース内のデータソースのクエリ方法を定義します。以下の検索タイプが可能です。

- デフォルト — Amazon Bedrock がユーザーに代わって検索戦略を決定します。
- ハイブリッド — ベクター埋め込み検索 (セマンティック検索) と未加工テキスト検索を組み合わせます。ハイブリッド検索は現在、フィルタリング可能なテキストフィールドを含む Amazon OpenSearch Serverless ベクターストアでのみサポートされています。別のベクターストアを使用している場合や、Amazon OpenSearch Serverless ベクターストアにフィルター可能なテキストフィールドがない場合、クエリはセマンティック検索を使用します。
- セマンティック — ベクター埋め込みのみを検索します。

検索タイプの定義方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

にあるコンソールの手順に従ってくださいナレッジベースに問い合わせて結果を返すか、応答を生成する。設定ペインを開くと、「検索タイプ」に以下のオプションが表示されます。

- デフォルト — Amazon Bedrock は、どの検索戦略がベクターストア設定に最も適しているかを判断します。
- ハイブリッド — Amazon Bedrock は、ベクター埋め込みと未加工テキストの両方を使用してナレッジベースにクエリを実行します。このオプションは、フィルタリング可能なテキストフィールドで構成された Amazon OpenSearch Serverless ベクターストアを使用している場合にのみ使用できます。
- セマンティック — Amazon Bedrock はベクター埋め込みを使用してナレッジベースにクエリを実行します。

API

RetrieveOR RetrieveAndGenerateリクエストを行う際には、`retrievalConfiguration`オブジェクトにマップされたフィールドを含めてください。KnowledgeBaseRetrievalConfigurationこのフィールドの場所を確認するには、API リファレンスの「」Retrieveおよび「RetrieveAndGenerateリクエスト本文」を参照してください。

次の JSON オブジェクトは、KnowledgeBaseRetrievalConfiguration検索タイプ設定を設定するためにオブジェクトに最低限必要なフィールドを示しています。

```
"retrievalConfiguration": {
    "vectorSearchConfiguration": {
        "overrideSearchType": "HYBRID | SEMANTIC"
    }
}
```

`overrideSearchType`フィールドに検索タイプを指定します。次のオプションがあります。

- 値を指定しない場合、Amazon Bedrock はどの検索戦略がベクターストア設定に最も適しているかを判断します。
- ハイブリッド — Amazon Bedrock は、ベクター埋め込みと未加工テキストの両方を使用してナレッジベースにクエリを実行します。このオプションは、フィルタリング可能なテキストフィールドで構成された Amazon OpenSearch Serverless ベクターストアを使用している場合にのみ使用できます。

- セマンティック — Amazon Bedrock はベクター埋め込みを使用してナレッジベースにクエリを実行します。

ソースチャンクの最大数

ナレッジベースにクエリを実行すると、Amazon Bedrock はレスポンスで最大 5 つのソースチャンクを返します。返されるソースチャンクの最大数を変更するには、選択した方法に対応するタブを選択し、手順に従います。

Console

にあるコンソールの手順に従ってください。ナレッジベースに問い合わせて結果を返すか、応答を生成する「構成」ペインで、「ソースチャンクの最大数」を展開します。

API

RetrieveRetrieveAndGenerateまたはリクエストを行う際には、`retrievalConfiguration`オブジェクトにマップされたフィールドを含めてください。KnowledgeBaseRetrievalConfigurationこのフィールドの場所を確認するには、API リファレンスの「」Retrieveおよび「RetrieveAndGenerateリクエスト本文」を参照してください。

次の JSON オブジェクトは、KnowledgeBaseRetrievalConfiguration返されるソースチャンクの最大数を設定するためにオブジェクトに最低限必要なフィールドを示しています。

```
"retrievalConfiguration": {
    "vectorSearchConfiguration": {
        "numberOfResults": number
    }
}
```

`numberOfResults`フィールドで返されるソースチャンクの最大数 (KnowledgeBaseRetrievalConfiguration許容値の範囲については、のフィールドを参照) を指定します。`numberOfResults`

メタデータとフィルタリング

データソースには、ソースドキュメントに関連するメタデータファイルを含めることができます。メタデータファイルには、ソースドキュメントに定義するキーと値のペアとして属性が含まれます。データソースファイルのメタデータの作成の詳細については、を参照してください。ファイルにメタ

データを追加してフィルタリングできるようにします。ナレッジベースのクエリ中にフィルターを使用するには、ナレッジベースが以下の要件を満たしていることを確認してください。

- データソースを含む Amazon S3 バケットには、`.metadata.json`関連付けられているソースドキュメントと同じ名前のファイルが少なくとも 1 つ含まれています。
- ナレッジベースのベクターインデックスが Amazon OpenSearch Serverless ベクターストアにある場合は、`faiss`ベクターインデックスがエンジンで設定されていることを確認してください。`nmslib`ベクトルインデックスがエンジンで設定されている場合は、次のいずれかを実行する必要があります。
  - コンソールで新しいナレッジベースを作成すると、Amazon Bedrock が Amazon OpenSearch サーバーレスのベクターインデックスを自動的に作成してくれます。
  - ベクターストアに別のベクターインデックスを作成し、**faiss**エンジンとして選択します。次に、新しいナレッジベースを作成し、新しいベクトルインデックスを指定します。

フィルタリングのクエリ設定を変更する際には、以下のフィルタリング演算子を使用できます。

フィルター演算子

演算子	コンソール	API フィルタ名	サポートされている属性データタイプ	フィルター結果
等しい	=	が	文字列、数値、ブーリアン	属性は指定した値と一致します
等しくない	!=	等しくない	文字列、数値、ブーリアン	属性が入力した値と一致しない
Greater than	>	[より大きい]	数値	属性は入力した値より大きい
より大きい、または等しい	>=	greaterThanOr等しい	数値	属性は指定した値以上または等しい
Less than	<	[未満]	数値	属性は指定した値より小さい

演算子	コンソール	API フィルタ名	サポートされている属性データタイプ	フィルター結果
より小さい、または等しい	<=	lessThanOr等しい	数値	属性は入力した値と等しいかそれ以下です
中	:	に	文字列リスト	属性は指定したリストに含まれている
含まれていません	!:	入ってはいけない	文字列リスト	指定したリストに属性がありません
で始まる	^	で始まる	string	属性は指定した文字列で始まります (Amazon OpenSearch サーバーレスベクターストアでのみサポートされます)

フィルタリング演算子を組み合わせるには、以下の論理演算子を使用できます。

論理演算子

演算子	コンソール	API フィルタフィールド名	フィルター結果	
And	また、	および [すべて]	結果はグループ内のすべてのフィルタリング式を満たします。	

演算子	コンソール	API フィルタフィールド名	フィルター結果	
または	または	[またはすべて]	結果はグループ内の少なくとも 1 つのフィルター条件を満たします。	

メタデータを使用して結果をフィルタリングする方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

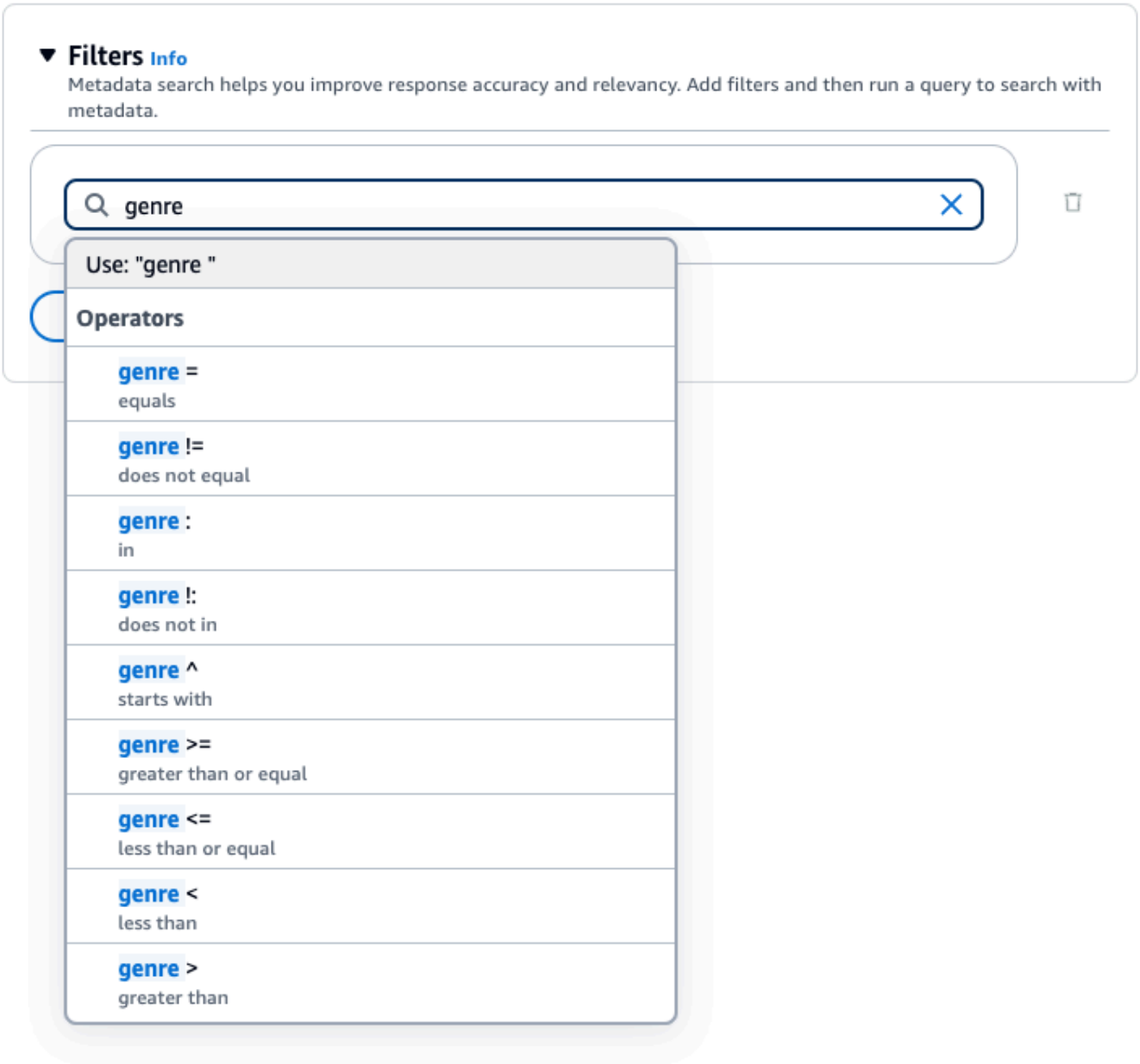
にあるコンソールの手順に従ってください[ナレッジベースに問い合わせて結果を返すか、応答を生成する]。「設定」ペインを開くと、「フィルター」セクションが表示されます。以下の手順では、さまざまなユースケースについて説明しています。

- フィルターを追加するには、ボックスにメタデータ属性、フィルター演算子、値を入力してフィルター式を作成します。式の各部分は空白で区切ります。Enter キーを押してフィルターを追加します。

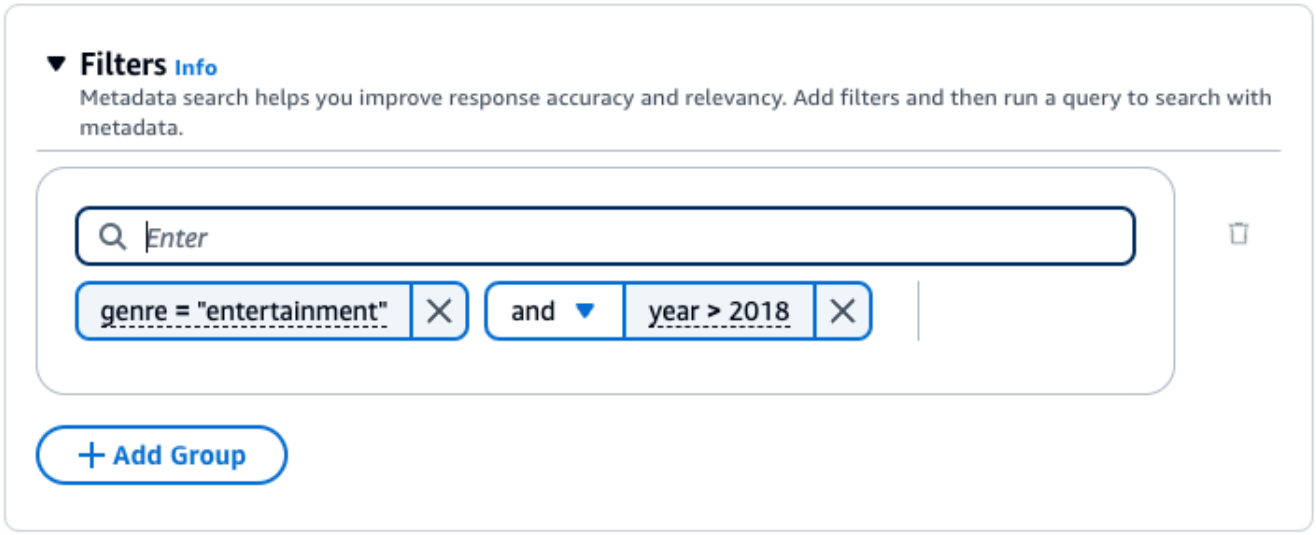
  使用できるフィルター演算子の一覧については、上記のフィルター演算子の表を参照してください。メタデータ属性の後に空白を追加すると、フィルター演算子のリストも表示されます。

  > **Note**
  >
  > 文字列は引用符で囲む必要があります。

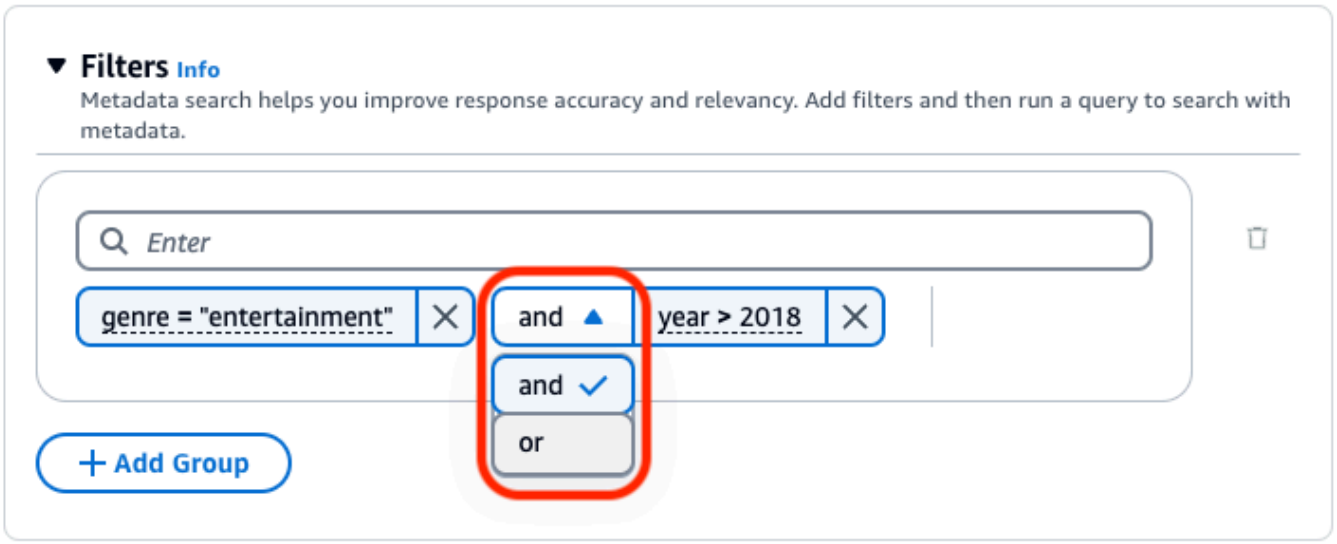
  たとえば、`"entertainment"`次のフィルターを追加することで、`genre`値がのメタデータ属性を含むソースドキュメントの結果をフィルターできます。**`genre = "entertainment"`**



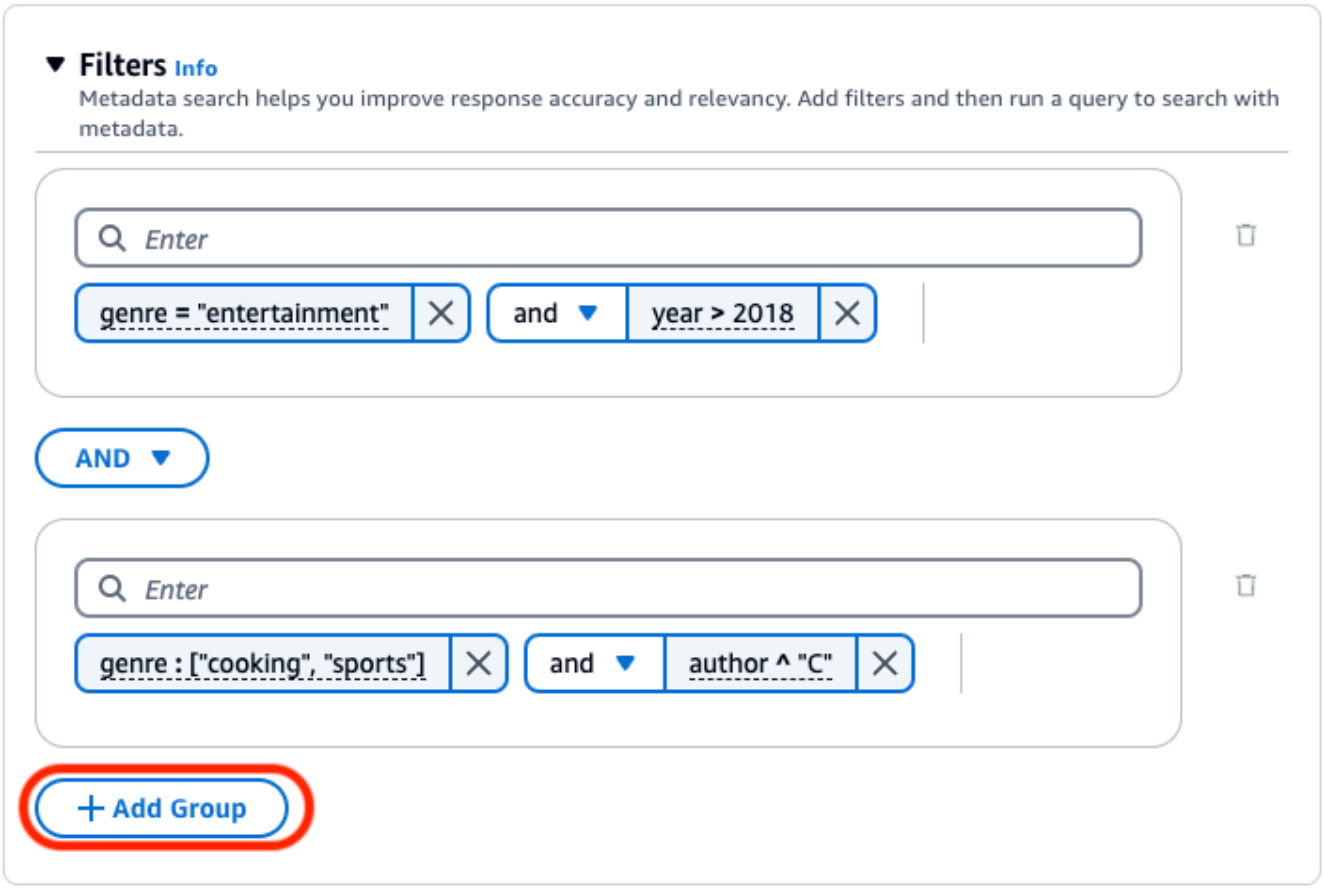
- 別のフィルターを追加するには、ボックスに別のフィルター条件を入力し、Enter キーを押します。グループには最大 5 つのフィルターを追加できます。



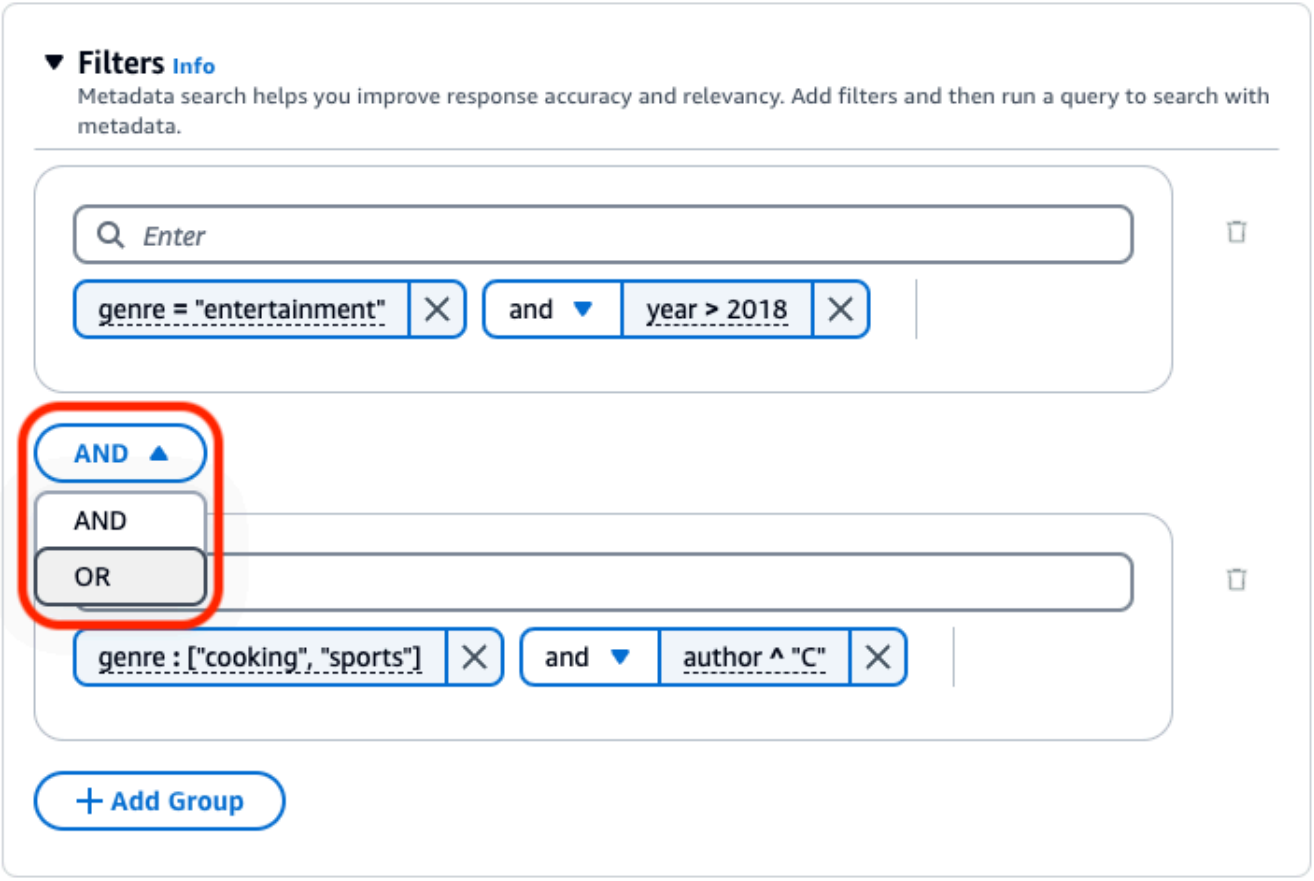
- デフォルトでは、クエリは指定したすべてのフィルター式を満たす結果を返します。少なくとも 1 つのフィルタリング式を満たす結果を返すには、2 つのフィルタリング操作の間にある and ドロップダウンメニューを選択し、またはを選択します。



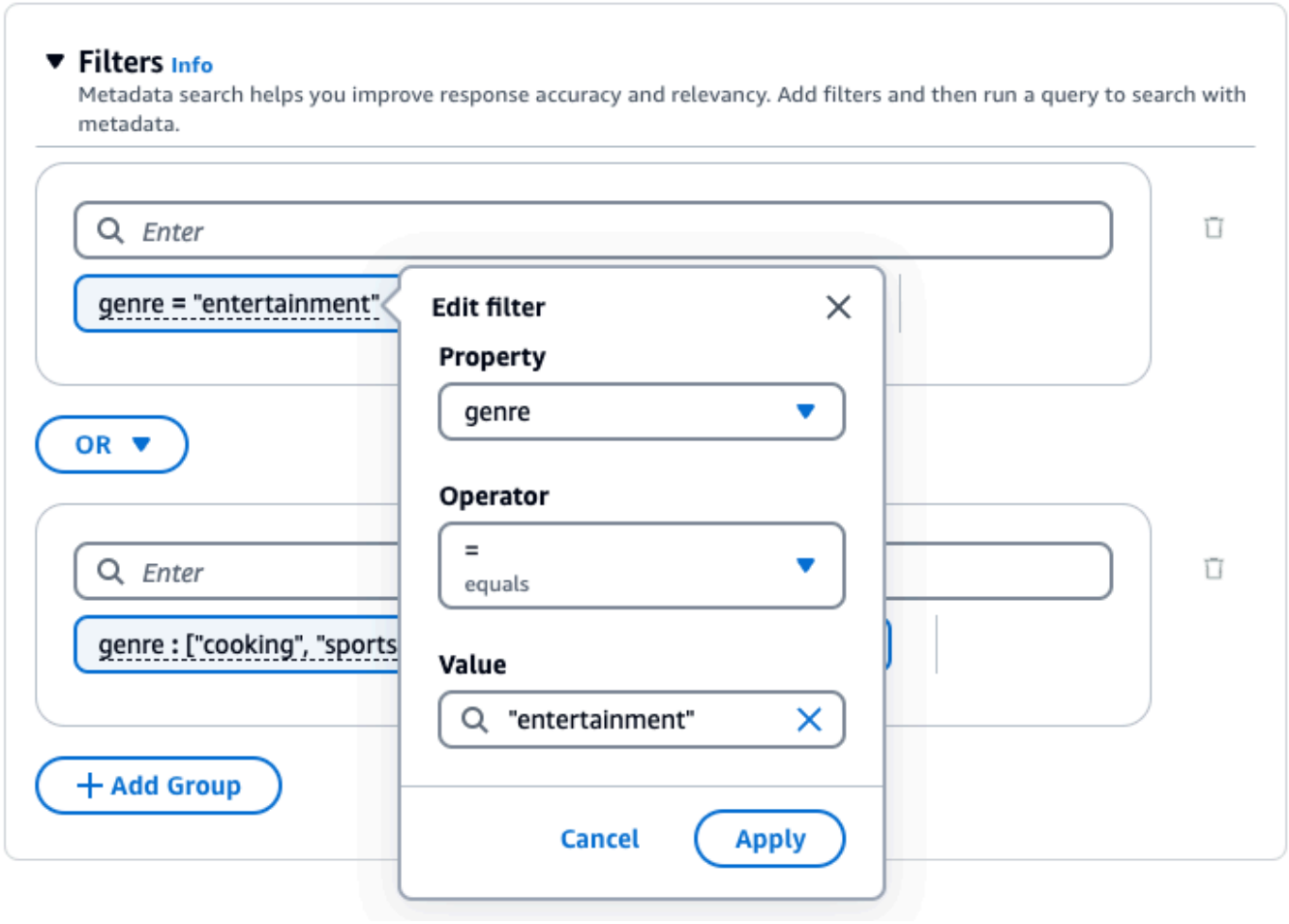
- 異なる論理演算子を組み合わせるには、[+ Add Group] を選択してフィルターグループを追加します。新しいグループにフィルター条件式を入力します。フィルターグループを 5 つまで追加できます。



- すべてのフィルターグループで使用される論理演算子を変更するには、2 つのフィルターグループの間にある AND ドロップダウンメニューを選択し、OR を選択します。



- フィルターを編集するには、フィルターを選択し、フィルター操作を変更して [適用] を選択します。



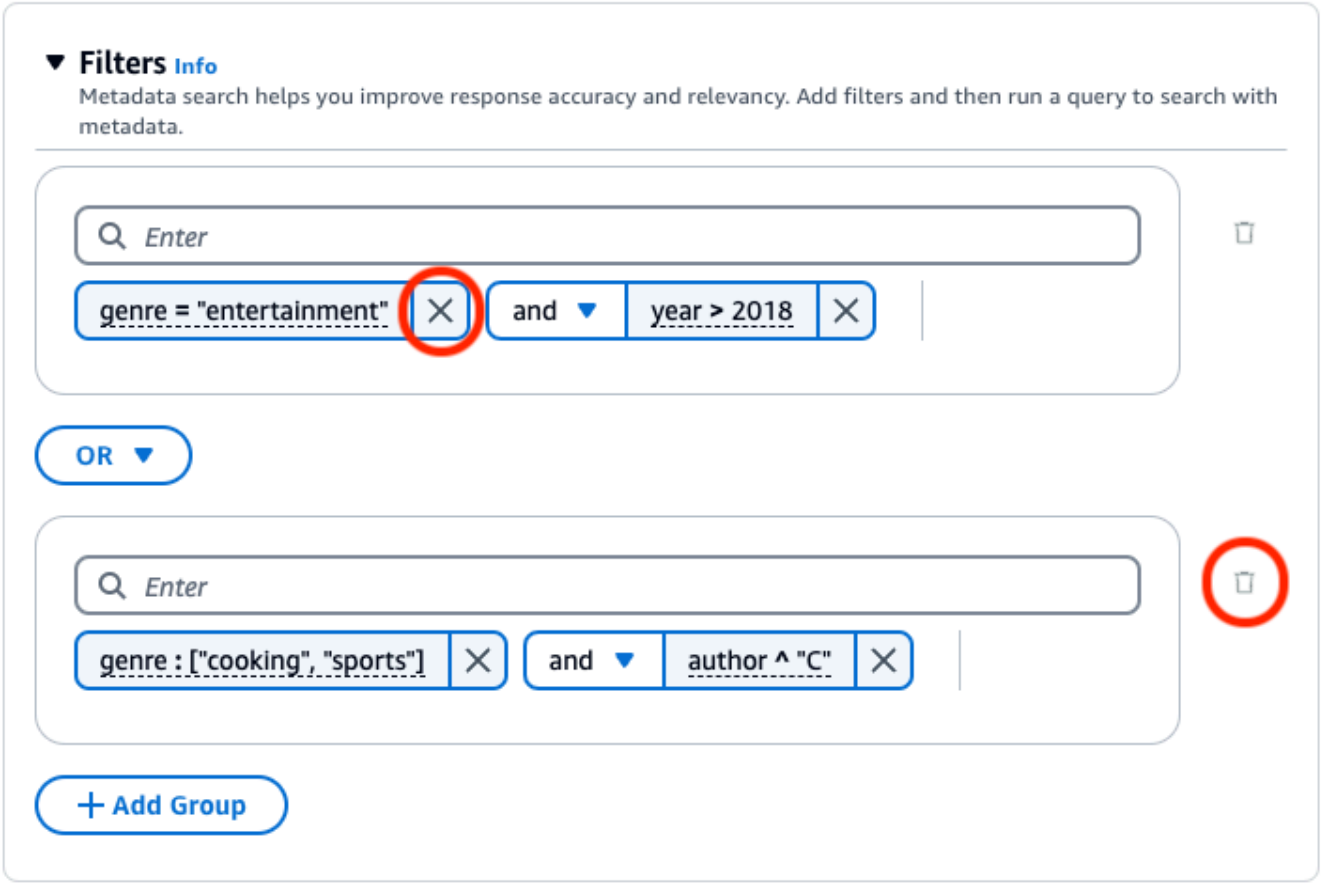
- フィルターグループを削除するには、グループの横にあるごみ箱アイコン

  ( )

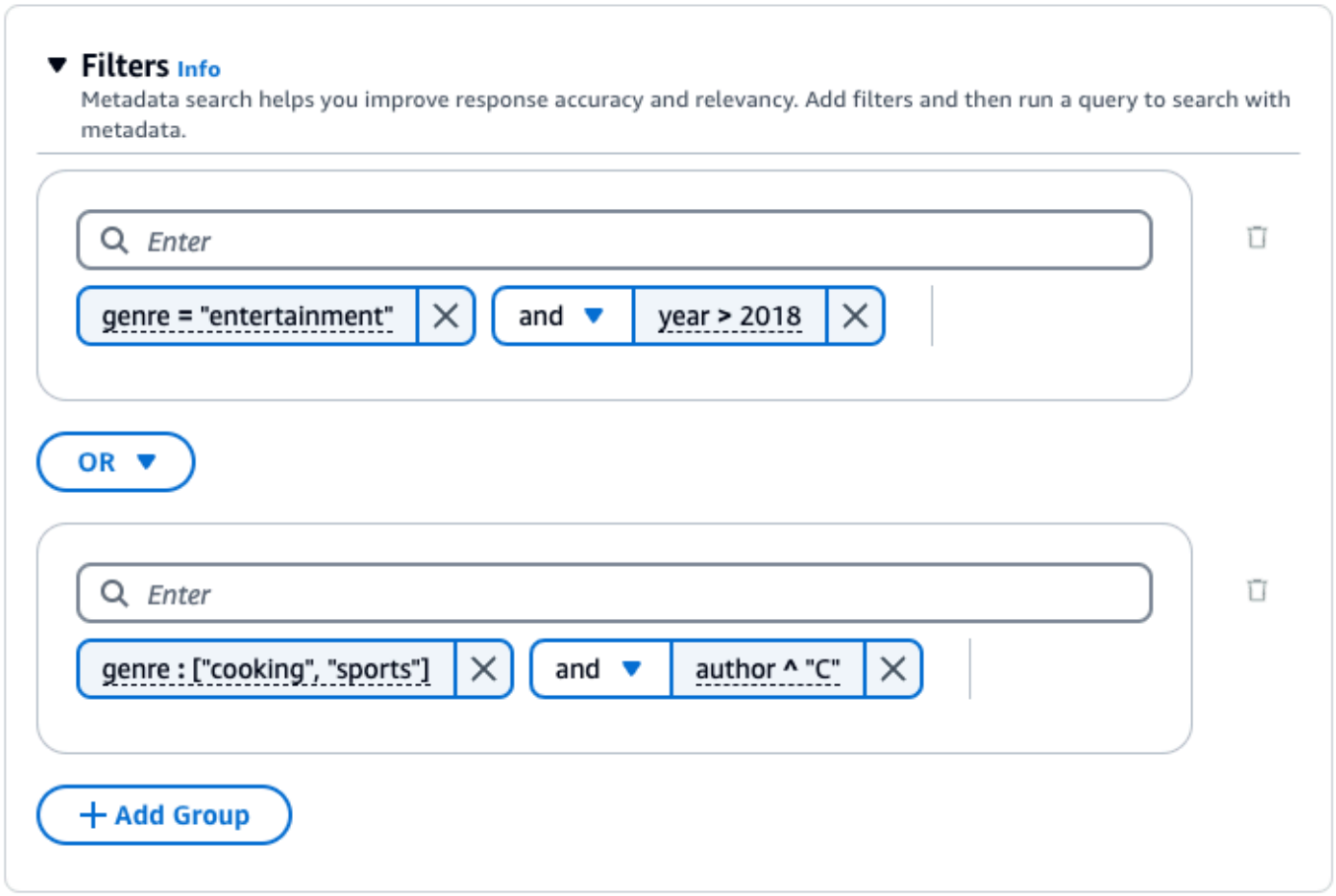
  を選択します。フィルターを削除するには、フィルターの横にある削除アイコン

  ( )

  を選択します。



以下の画像は、**2018"cooking""sports"**ジャンルがまたはで、作成者がで始まるドキュメントに加えて**"entertainment"**、ジャンルが1の後に書かれたすべてのドキュメントを返すフィルター構成の例を示しています**"C"**。



API

RetrieveOR RetrieveAndGenerateリクエストを行う際に は、retrievalConfigurationKnowledgeBaseRetrievalConfigurationオブジェクトにマップされたフィールドを含めてください。このフィールドの場所を確認するには、API リファレンスの「」Retrieveおよび「RetrieveAndGenerateリクエスト本文」を参照してください。

次の JSON オブジェクトは、KnowledgeBaseRetrievalConfigurationさまざまなユースケースのフィルタを設定するためにオブジェクトに最低限必要なフィールドを示しています。

1. 1 つのフィルター演算子を使用してください (上記のフィルター演算子の表を参照)。

```
"retrievalConfiguration": {
    "vectorSearchConfiguration": {
        "filter": {
            "<filter-type>": {
                "key": "string",
                "value": "string" | number | boolean | ["string", "string", ...]
            }
```

```
        }
    }
}
```

2. 論理演算子 (上の論理演算子の表を参照) を使用して、最大 5 つまで組み合わせることができます。

```
"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. 論理演算子を使用して最大 5 つのフィルタリング演算子を 1 つのフィルターグループに結合し、2 つ目の論理演算子を使用してそのフィルターグループを別のフィルター演算子と結合します。

```
"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. 最大 5 つのフィルターグループを別の論理演算子に埋め込んで組み合わせることができます。1 つのレベルの埋め込みを作成できます。

```
"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", ...]
                    },
                    ...
                ]
            ]
        }
    }
}
```

次の表では、使用できるフィルタータイプについて説明しています。

フィールド	サポートされている値のデータ型	フィルター処理された結果
equals	文字列、数値、ブーリアン	属性は指定した値と一致します
notEquals	文字列、数値、ブーリアン	属性が入力した値と一致しない
greaterThan	数値	属性が入力した値より大きい
greaterThanOrEquals	数値	属性は入力した値以上またはそれ以上です
lessThan	数値	属性が入力した値より小さい
lessThanOrEquals	数値	属性は入力した値と等しいか、それより小さい
in	文字列のリスト	属性は指定したリストに含まれている
notIn	文字列のリスト。	指定したリストに属性がない
startsWith	string	属性は指定した文字列で始まります (Amazon OpenSearch

フィールド	サポートされている値のデータ型	フィルター処理された結果
		サーバーレスベクターストアでのみサポートされます)

フィルタータイプを組み合わせるには、以下の論理演算子のいずれかを使用できます。

フィールド	にマップします。	フィルター結果
andAll	5 種類までのフィルタータイプのリスト	結果はグループ内のすべてのフィルター条件を満たします。
orAll	5 種類までのフィルタータイプのリスト	結果はグループ内の少なくとも 1 つのフィルター条件を満たします。

例については、「クエリを送信してフィルタを含める (取得)」と「クエリを送信してフィルタを含める (RetrieveAndGenerate)」を参照してください。

ナレッジベースのプロンプトテンプレート

ナレッジベースをクエリしてレスポンス生成をリクエストすると、Amazon Bedrock は指示とコンテキストをユーザークエリと組み合わせたプロンプトテンプレートを使用して、応答生成用のモデルに送信されるプロンプトを作成します。プロンプトテンプレートは以下のツールで設計できます。

- プロンプトプレースホルダー — Amazon Bedrock のナレッジベースで事前に定義されている変数で、ナレッジベースのクエリ中に実行時に動的に入力されます。システムプロンプトでは、これらのプレースホルダーがシンボルで囲まれて表示されます。$以下のリストでは、使用できるプレースホルダーについて説明しています。

変数	に置き換えられました。	モデル	必須?
$クエリ$	ナレッジベースに送信されたユーザークエリ。	AnthropicClaude Instant、Anthropic Claude v2.x	Yes
		Anthropic Claude 3 Sonnet	いいえ (モデル入力に自動的に含まれる)
$検索結果$	ユーザークエリの取得結果。	すべて	Yes
$出力フォーマット_インストラクション$	レスポンス生成と引用をフォーマットするための基本的な指示 モデルによって異なります。独自のフォーマット手順を定義する場合は、このプレースホルダーを削除することをお勧めします。このプレースホルダーがないと、レスポンスには引用が含まれません。	すべて	No
$現在の時間$	現在の時刻。	すべて	No

- XML タグ — Anthropic モデルでは、XML タグを使用してプロンプトを構造化および明確化できます。最適な結果を得るには、わかりやすいタグ名を使用してください。たとえば、デフォルトのシステムプロンプトには、<database>以前に尋ねられた質問のデータベースを表すために使用されるタグが表示されます)。詳しくは、Anthropicユーザーガイドの「XML タグを使用する」を参照してください。

プロンプトエンジニアリングの一般的なガイドラインについては、を参照してください プロンプトエンジニアリングガイドライン。

選択した方法に対応するタブを選択し、手順に従います。

Console

にあるコンソールの手順に従いますナレッジベースに問い合わせて結果を返すか、応答を生成する。テストウィンドウで [回答を生成] をオンにします。次に、「構成」ペインで「ナレッジベースのプロンプトテンプレート」セクションを展開します。

1. [編集] を選択します。
2. 必要に応じて、プロンプトプレースホルダーや XML タグを含めて、テキストエディターでシステムプロンプトを編集します。デフォルトのプロンプトテンプレートに戻すには、「デフォルトにリセット」を選択します。
3. 編集が終了したら、[変更の保存] を選択します。システムプロンプトを保存せずに終了するには、[変更を破棄] を選択します。

API

RetrieveAndGenerateリクエストを行うときは、generationConfigurationオブジェクトにマップされたフィールドを含めてください。GenerationConfigurationこのフィールドの場所を確認するには、API RetrieveAndGenerateリファレンスのリクエスト本文を参照してください。

次の JSON オブジェクトは、GenerationConfiguration返されるソースチャンクの最大数を設定するためにオブジェクトに最低限必要なフィールドを示しています。

```
"generationConfiguration": {
    "promptTemplate": {
        "textPromptTemplate": "string"
    }
}
```

必要に応じて、プロンプトプレースホルダーと XML textPromptTemplate タグを含むカスタムプロンプトテンプレートをフィールドに入力します。システムプロンプトに入力できる最大文字数については、textPromptTemplateGenerationConfigurationのフィールドを参照してください。

データソースの管理

データソースを作成したら、その詳細を表示、更新、または削除できます。

データソースに関する情報を表示する。

データソースとその同期履歴に関する情報を表示できます。選択した方法に対応するタブを選択し、手順に従います。

Console

データソースに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. [データソース] セクションで、詳細を確認したいデータソースを選択します。
4. データソース概要には、データソースに関する詳細が含まれています。
5. 同期履歴には、データソースがいつ同期されたかに関する詳細が含まれます。同期イベントが失敗した理由を確認するには、同期イベントを選択して [警告を表示] を選択します。

API

データソースに関する情報を取得するには、Agents for Amazon Bedrock GetDataSourceビルドタイムエンドポイントにリクエストを送信し、`dataSourceId`そのデータソースが属するナレッジベースのとを指定します。`knowledgeBaseId`

ナレッジベースのデータソースに関する情報を一覧表示するには、Agents for Amazon Bedrock ListDataSourcesビルドタイムエンドポイントにリクエストを送信し、ナレッジベースの ID を指定します。

- 1 回のレスポンスで返される結果の最大数を設定するには、フィールドを使用します。`maxResults`
- 設定した数よりも多くの結果がある場合、レスポンスはを返しますnextToken。`ListDataSources`この値を別のリクエストで使用して、次の結果を確認できます。

データソースの同期イベントの情報を取得するには、Agents for Amazon Bedrock GetIngestionJobビルドタイムエンドポイントにリクエストを送信します。dataSourceId、knowledgeBaseId、および ingestionJobId を指定します。

データソースの同期履歴をナレッジベースに一覧表示するには、Agents for Amazon Bedrock ListIngestionJobsビルドタイムエンドポイントを使用してリクエストを送信します。ナレッジベースとデータソースの ID を指定します。次の仕様を設定できます。

- 検索するステータスを filters オブジェクトで指定して、結果をフィルタリングします。
- sortBy オブジェクトを指定して、ジョブの開始時間またはジョブのステータスでソートします。昇順または降順で並べ替えることができます。
- レスポンスで返す結果の最大数を maxResults フィールドで設定します。設定した数よりも多くの結果がある場合、レスポンスからが返され、ListIngestionJobs次のジョブバッチを確認するために別のリクエストで送信できます。nextToken

データソースを更新します。

データソースは次の方法で更新できます。

- データソースのファイルが含まれている S3 バケットからファイルを追加、変更、または削除します。
- データソースの名前または S3 バケット、またはデータ取り込み時の一時データの暗号化に使用する KMS キーを変更します。

データソースの S3 バケットにファイルを追加、変更、または削除するたびに、データソースを同期してナレッジベースに再インデックスする必要があります。同期は段階的に行われるため、Amazon Bedrock は前回の同期以降に追加、変更、または削除された S3 バケット内のオブジェクトのみを処理します。取り込みを開始する前に、データソースが以下の条件を満たしていることを確認してください。

- ファイルはサポートされている形式です。詳細については、「サポート対象のベクターストアにナレッジベースのベクターインデックスを設定する」を参照してください。
- ファイルは最大ファイルサイズの 50 MB を超えません。詳細については、「ナレッジベースのクォータ」を参照してください。
- データソースにメタデータファイルが含まれている場合は、次の条件をチェックして、メタデータファイルが無視されないことを確認してください。

- `.metadata.json`各ファイルは、関連付けられているソースファイルと同じ名前を共有します。
- ナレッジベースのベクターインデックスが Amazon OpenSearch Serverless ベクターストアにある場合は、`faiss`ベクターインデックスがエンジンで設定されていることを確認してください。`nmslib`ベクトルインデックスがエンジンで設定されている場合は、次のいずれかを実行する必要があります。
  - コンソールで新しいナレッジベースを作成すると、Amazon Bedrock が Amazon OpenSearch サーバーレスのベクターインデックスを自動的に作成してくれます。
  - ベクターストアに別のベクターインデックスを作成し、**faiss**エンジンとして選択します。次に、新しいナレッジベースを作成し、新しいベクトルインデックスを指定します。
- ナレッジベースのベクターインデックスが Amazon Aurora データベースクラスターにある場合は、取り込みを開始する前に、インデックスのテーブルにメタデータファイルの各メタデータプロパティの列が含まれていることを確認してください。

データソースを更新する方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

データソースを更新するには

1. (オプション) データソースのファイルを含む S3 バケット内のファイルに必要な変更を加えます。
2. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
3. 左のナビゲーションペインで [ナレッジベース] を選択します。
4. [データソース] セクションで、同期するデータソースの横にあるラジオボタンを選択します。
5. (オプション) [編集] を選択し、必要な設定を変更して [送信] を選択します。
6. [同期] を選択します。
7. 同期が完了し、ステータスが「準備完了」になると、緑色のバナーが表示されます。

API

データソースを更新するには

1. (オプション) データソースのファイルを含む S3 バケット内のファイルに必要な変更を加えます。
2. (オプション) Agents for Amazon Bedrock UpdateDataSourceのビルド時エンドポイントにリクエストを送信し、必要な設定を変更し、変更したくない同じ設定を指定します。

   > **Note**
   >
   > は変更できません。chunkingConfigurationchunkingConfiguration既存のものでリクエストを送信してください。

3. Agents for Amazon Bedrock のビルド時エンドポイントで、StartIngestionJobとを指定してリクエストを送信しますdataSourceId。knowledgeBaseId

データソースの削除

データソースが不要になった場合は、削除できます。選択した方法に対応するタブを選択し、手順に従います。

Console

データソースを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. [データソース] セクションで、削除するデータソースの横にあるラジオボタンを選択します。
4. [削除] をクリックします。
5. データソースが正常に削除されると、緑色のバナーが表示されます。

API

ナレッジベースからデータソースを削除するには、[DeleteDataSource](#)dataSourceIdknowledgeBaseIdとを指定してリクエストを送信します。

ナレッジベースを管理する

ナレッジベースを設定したら、ナレッジベースに関する情報を表示、変更、または削除できます。選択した方法に対応するタブを選択し、手順に従います。

ナレッジベースに関する情報を表示する

ナレッジベースに関する情報を表示できます。選択した方法に対応するタブを選択し、手順に従います。

Console

ナレッジベースに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. ナレッジベースの詳細を表示するには、[名前] でソースの名前を選択します。または、ソースの横にあるラジオボタンを選択し、[編集] を選択します。
4. 詳細ページでは、次のアクションを実行できます。
   - ナレッジベースの詳細を変更するには、[ナレッジベース概要] セクションで [編集] を選択します。
   - ナレッジベースに付いているタグを更新するには、[タグ] セクションで [タグの管理] を選択します。
   - ナレッジベース作成の元となったデータソースを更新し、変更を同期する必要がある場合は、[データソース] セクションで [同期] を選択します。
   - データソースの詳細を表示するには、[データソース名] を選択します。詳細ページでは、[同期履歴] セクションの同期イベントの横にあるラジオボタンを選択し、[警告を表示] を選択すると、データインジェストジョブのファイルが同期に失敗した理由を確認できます。

- ナレッジベースに使用される埋め込みモデルを管理するには、[プロビジョンドスループットを編集] を選択します。
- 編集が終了したら、[変更を保存] を選択します。

API

ナレッジベースに関する情報を取得するには、 を指定して Agents for Amazon Bedrock ビルドタイムエンドポイント で GetKnowledgeBase リクエストを送信しますknowledgeBaseId。

ナレッジベースに関する情報を一覧表示するには、 Agents for Amazon Bedrock ビルドタイムエンドポイント を使用してListKnowledgeBasesリクエストを送信します。レスポンスで返す結果の最大数を設定できます。設定した数よりも多くの結果がある場合、レスポンスは を返しますnextToken。この値を別のListKnowledgeBasesリクエストの nextTokenフィールドで使用して、結果の次のバッチを表示できます。

ナレッジベースを更新する

Console

ナレッジベースを更新するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [ナレッジベース] を選択します。
3. ナレッジベースを選択してその詳細を表示するか、ナレッジベースの横にあるラジオボタンを選択して の編集 を選択します。
4. ナレッジベースは、次の方法で変更できます。
   - ナレッジベースの概要セクションの「編集」を選択して、ナレッジベースの設定を変更します。
   - タグセクションのタグの管理を選択して、ナレッジベースにアタッチされているタグを変更する
   - データソースセクションでデータソースを管理します。詳細については、「データソースの管理」を参照してください。
5. 編集が終了したら、[変更を保存] を選択します。

API

ナレッジベースを更新するには、 Agents for Amazon Bedrock ビルドタイムエンドポイント を使用して UpdateKnowledgeBaseリクエストを送信します。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。

ナレッジベースを削除する

ナレッジベースが不要になった場合は、削除できます。ナレッジベースを削除するときは、ナレッジベースに関連付けられているすべてのリソースを完全に削除するには、次のアクションも実行する必要があります。

- ナレッジベースが関連付けられているエージェントからナレッジベースの関連付けを解除します。
- ナレッジベースからインデックス作成された基盤となるデータは、設定したベクトルストアに残り、引き続き取得できます。データを削除するには、データ埋め込みを含むベクトルインデックスも削除する必要があります。

選択した方法に対応するタブを選択し、手順に従います。

Console

ナレッジベースを削除するには

1. 次の手順を実行する前に、ナレッジベースが関連付けられているすべてのエージェントからナレッジベースを必ず削除してください。これを行うには、次の手順を実行します。
   a. 左のナビゲーションペインで [エージェント] を選択します。
   b. [名前] で、ナレッジベースを削除するエージェントの名前を選択します。
   c. もう存在しないナレッジベースへの参照をエージェントから削除するよう警告する赤いバナーが表示されます。
   d. 削除するナレッジベースの横にあるラジオボタンを選択します。[詳細] を選択して、[削除] を選択します。
2. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
3. 左のナビゲーションペインで [ナレッジベース] を選択します。
4. ナレッジベースを選択するか、ナレッジベースの横にあるラジオボタンを選択します。その後、[削除] をクリックします。

5. ナレッジベースの削除に関する警告を確認してください。これらの条件に同意する場合は、入力ボックスに「**delete**」を入力し、[削除] を選択して確定します。
6. ナレッジベースのベクトル埋め込みを完全に削除するには、データ埋め込みを含むベクトルインデックスを削除する必要があります。

API

ナレッジベースを削除する前に、Amazon Bedrock ビルドタイムエンドポイント のエージェントにDisassociateAgentKnowledgeBaseリクエストを行うことで、ナレッジベースと関連付けられているエージェントの関連付けを解除します。 https://docs.aws.amazon.com/general/latest/gr/bedrock.html#bra-bt

ナレッジベースを削除するには、 Agents for Amazon Bedrock ビルドタイムエンドポイント を使用して DeleteKnowledgeBase リクエストを送信します。

ナレッジベースのベクトル埋め込みを完全に削除するには、データ埋め込みを含むベクトルインデックスを削除する必要があります。

ナレッジベースをデプロイする

ナレッジベースをアプリケーションにデプロイするには、ナレッジベースに対して Retrieveまたは RetrieveAndGenerateリクエストを行うように設定します。これらの API オペレーションの使用方法を確認するには、 で API タブを選択しますAmazon Bedrock でナレッジベースをテストしましょう。

また、ナレッジベースをエージェントに関連付けると、エージェントはオーケストレーション中に必要に応じてナレッジベースを呼び出すことができます。詳細については、「Agents for Amazon Bedrock」を参照してください。選択した方法に対応するタブを選択し、手順に従います。

Console

ナレッジベースをエージェントに関連付けるには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [エージェント] を選択します。
3. ナレッジベースを追加するエージェントを選択します。
4. ドラフトの作成セクションで、ドラフトの作成を選択します。

5. ナレッジベース セクションで、 を追加 を選択します。
6. 「ナレッジベースの選択」のドロップダウンリストからナレッジベースを選択し、ナレッジベースとやり取りして結果を返す方法に関するエージェント向け指示を指定します。

ナレッジベースとエージェントの関連付けを解除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインで [エージェント] を選択します。
3. ナレッジベースを追加するエージェントを選択します。
4. ドラフトの作成セクションで、ドラフトの作成を選択します。
5. ナレッジベース セクションで、ナレッジベースを選択します。
6. [削除] を選択します。

API

ナレッジベースをエージェントに関連付けるには、 AssociateAgentKnowledgeBaseリクエストを送信します。

- エージェントがナレッジベースとやり取りして結果を返す方法を説明するdescription詳細な を含めます。
- knowledgeBaseState ENABLED エージェントがナレッジベースにクエリを実行できるようにするには、 を に設定します。

UpdateAgentKnowledgeBase リクエストを送信することで、エージェントに関連付けられているナレッジベースを更新できます。例えば、 knowledgeBaseStateを に設定ENABLEDして問題をトラブルシューティングできます。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。

ナレッジベースとエージェントの関連付けを解除するには、DisassociateAgentKnowledgeBaseリクエストを送信します。

Agents for Amazon Bedrock

Agents for Amazon Bedrock では、アプリケーション内で自律型エージェントを構築して設定することができます。エージェントは、エンドユーザーが組織のデータとユーザー入力に基づいてアクションを実行するのに役立ちます。エージェントは、基盤モデル (FMsデータソース、ソフトウェアアプリケーション、ユーザー会話間のインタラクションをオーケストレートします。さらに、エージェントAPIs を自動的に呼び出してアクションを実行し、ナレッジベースを呼び出してこれらのアクションの情報を補足します。開発者は、エージェントを統合して生成人工知能 (生成 AI) アプリケーションの提供を加速することで、数週間の開発作業を節約できます。

エージェントを使用すると、顧客のタスクを自動化し、顧客の質問に答えることができます。例えば、顧客が保険請求を処理するのに役立つエージェントや、顧客が旅行予約を行うのに役立つエージェントを作成できます。容量のプロビジョニング、インフラストラクチャの管理、カスタムコードの記述を行う必要はありません。Amazon Bedrock は、プロンプトエンジニアリング、メモリー、モニタリング、暗号化、ユーザーのアクセス許可、および API 呼び出しを管理します。

エージェントは以下のタスクを実行します。

- 基盤モデルを拡張してユーザーリクエストを理解し、エージェントが実行する必要があるタスクをより小さなステップに分類します。
- ユーザーから、自然な会話を通して追加情報を収集します。
- 会社のシステムに API コールを実行して、顧客のリクエストを満たすためのアクションを実行します。
- データソースにクエリを実行することで、パフォーマンスと正解率を向上させます。

エージェントを使用するには、次のステップを実行します。

1. (オプション) ナレッジベースを作成して、プライベートデータをそのデータベースに保存します。詳細については、「Amazon ベッドロックのナレッジベース」を参照してください。
2. ユースケースに合わせてエージェントを設定し、エージェントが実行できるアクションを追加します。エージェントがアクションを処理する方法を定義するには、選択したプログラミング言語で Lambda 関数を記述します。
3. ナレッジベースをエージェントに関連付けて、エージェントのパフォーマンスを向上させます。詳細については、「Amazon Bedrock でエージェントを作成する」を参照してください。
4. (オプション) エージェントの動作を特定のユースケースに合わせてカスタマイズするには、エージェントが実行する前処理、オーケストレーション、ナレッジベースのレスポンス生成、後

処理ステップのプロンプトテンプレートを変更します。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。

5. Amazon Bedrock コンソールまたは への API コールを使用して、エージェントをテストしますTSTALIASID。必要に応じて設定を変更します。トレースを使用して、オーケストレーションの各ステップにおけるエージェントの推論プロセスを調査します。詳細については、「Amazon Bedrock エージェントをテストする」および「Amazon Bedrock でのトレースイベント」を参照してください。
6. エージェントを十分に変更し、アプリケーションにデプロイする準備ができたら、エージェントのバージョンを指すエイリアスを作成します。詳細については、「Amazon Bedrock エージェントをデプロイする」を参照してください。
7. エージェントのエイリアスへの API コールを行うようにアプリケーションを設定します。
8. エージェントを繰り返し処理し、必要に応じてさらに多くのバージョンとエイリアスを作成します。

トピック

- Agents for Amazon Bedrock の仕組み
- Amazon Bedrock のエージェントでサポートされているリージョンとモデル
- Agents for Amazon Bedrock の前提条件
- Amazon Bedrock でエージェントを作成する
- Amazon Bedrock エージェントのアクショングループを作成する
- ナレッジベースを Amazon Bedrock エージェントに関連付ける
- Amazon Bedrock エージェントをテストする
- Amazon Bedrock エージェントを管理する
- Amazon Bedrock エージェントをカスタマイズする
- Amazon Bedrock エージェントをデプロイする

Agents for Amazon Bedrock の仕組み

Agents for Amazon Bedrock は、エージェントのセットアップと実行に役立つ次の 2 つの主要な API オペレーションのセットで構成されています。

- エージェントとその関連リソースを作成、設定、管理するビルドタイム API オペレーション

- ユーザー入力でエージェントを呼び出し、タスクを実行するためのオーケストレーションを開始するランタイム API オペレーション。

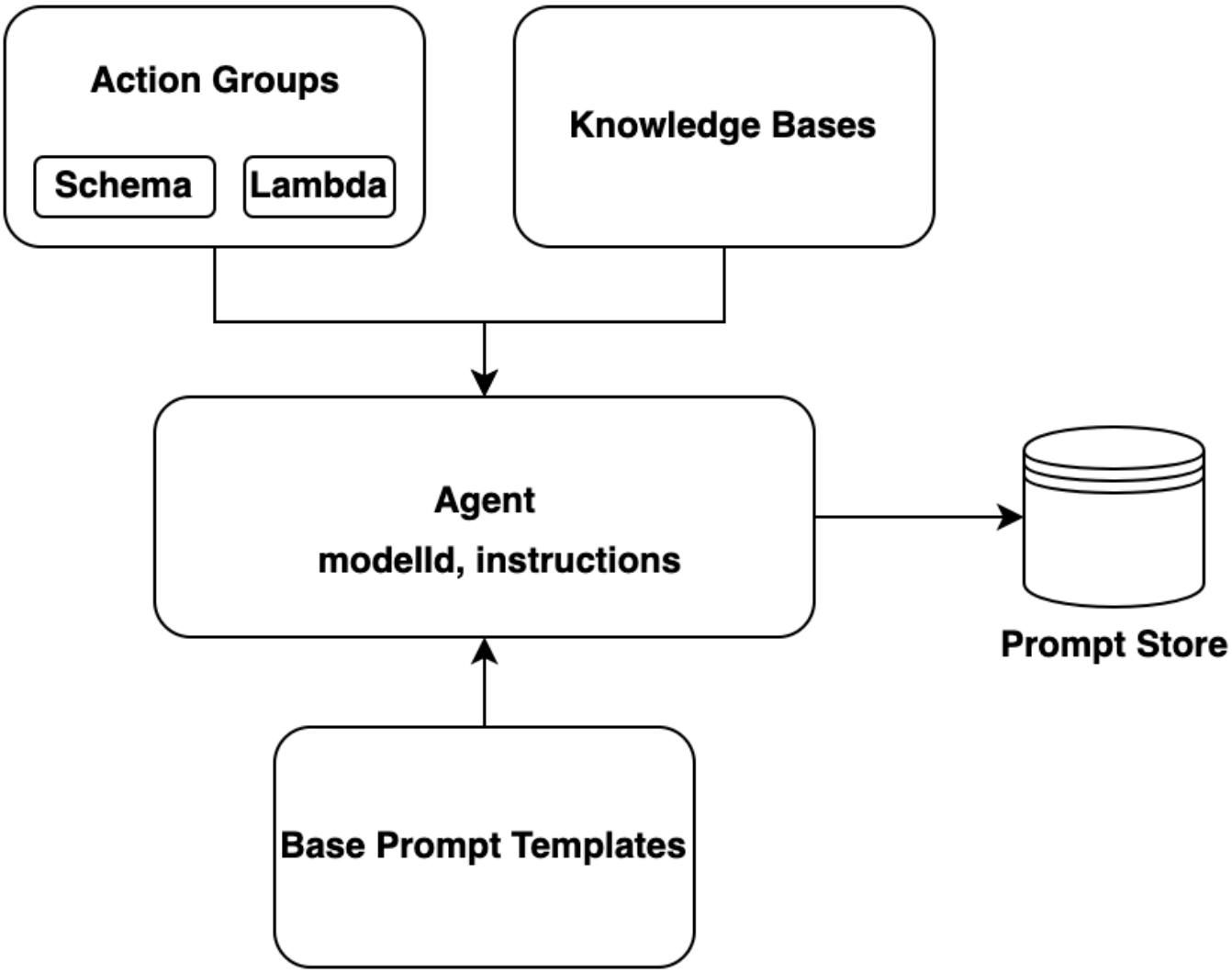
ビルド時設定

エージェントは次のコンポーネントで構成されています。

- 基盤モデル – オーケストレーションプロセスでエージェントがユーザー入力と後続のプロンプトを解釈するために呼び出す基盤モデル (FM) を選択します。また、エージェントは FM を呼び出して、プロセス内のレスポンスとフォローアップステップを生成します。
- 手順 – エージェントの動作を記述する手順を作成します。高度なプロンプトを使用すると、オーケストレーションの各ステップでエージェントの指示をさらにカスタマイズし、各ステップの出力を解析する Lambda 関数を含めることができます。
- アクショングループ (オプション) — 次のリソースを提供することで、エージェントが実行する必要があるアクションを定義します。
  - エージェントがタスクを実行するために呼び出すことができる API オペレーションを定義する OpenAPIスキーマ。
  - 次の入出力を持つ Lambda 関数。
    - 入力 – オーケストレーション中に識別される API オペレーションとパラメータ。
    - 出力 — API 呼び出しの結果。
- ナレッジベース (オプション) — ナレッジベースをエージェントに関連付けます。エージェントは、追加のコンテキストについてナレッジベースに問い合わせて、レスポンスの生成とオーケストレーションプロセスのステップへの入力を強化します。
- プロンプトテンプレート – プロンプトテンプレートは、FM に提供するプロンプトを作成するための基盤です。Agents for Amazon Bedrock は、前処理、オーケストレーション、ナレッジベースのレスポンス生成、後処理で使用されるデフォルトの 4 つの基本プロンプトテンプレートを公開します。オプションで、これらの基本プロンプトテンプレートを編集して、シーケンスの各ステップでのエージェントの動作をカスタマイズできます。また、トラブルシューティングの目的で、またはステップが不要なと判断した場合は、ステップをオフにすることもできます。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。

ビルド時に、これらのコンポーネントがすべて収集され、ユーザーのリクエストが完了するまでエージェントがオーケストレーションを実行するための基本プロンプトが作成されます。詳細プロンプトでは、追加のロジックやいくつかのサンプルを追加して基本プロンプトを変更し、エージェント呼び出しの各ステップの正解率を向上させることができます。基本プロンプトテンプレートには、指示、

アクションの説明、ナレッジベースの説明、会話履歴が含まれており、これらはすべてニーズに合わせてエージェントを変更するためにカスタマイズできます。次に、セキュリティ設定を含むエージェントのすべてのコンポーネントをパッケージ化する エージェントを準備します。エージェントの準備は、ランタイムでテストできる状態になります。次の図は、ビルド時 API オペレーションがエージェントを構築する方法を示しています。



ランタイムプロセス

ランタイムは InvokeAgent API オペレーションによって管理されます。このオペレーションは、次の 3 つの主要なステップで構成されるエージェントシーケンスを開始します。

1. 前処理 — エージェントがユーザー入力をコンテキスト化および分類する方法を管理し、入力を検証するために使用できます。

2. オーケストレーション - ユーザー入力を解釈し、アクショングループを呼び出してナレッジベースをクエリし、出力をユーザーまたは継続的なオーケストレーションへの入力として返します。オーケストレーションは、次のステップで構成されます。

   a. エージェントが入力を基盤モデルで解釈し、次に取るべきステップのロジックを説明する理論的根拠を生成します。

   b. エージェントはアクショングループを呼び出し、ナレッジベース (Knowledge ベースのレスポンス生成) をクエリして追加のコンテキストを取得し、データを要約して生成を強化します。

   c. エージェントは、アクショングループを呼び出し、ナレッジベースからの結果を要約することで、オブザベーションと呼ばれる出力を生成します。エージェントはオブザベーションを使用してベースプロンプトを補足し、それを基盤モデルで解釈します。次に、エージェントはオーケストレーションプロセスを繰り返す必要があるかどうかを判断します。

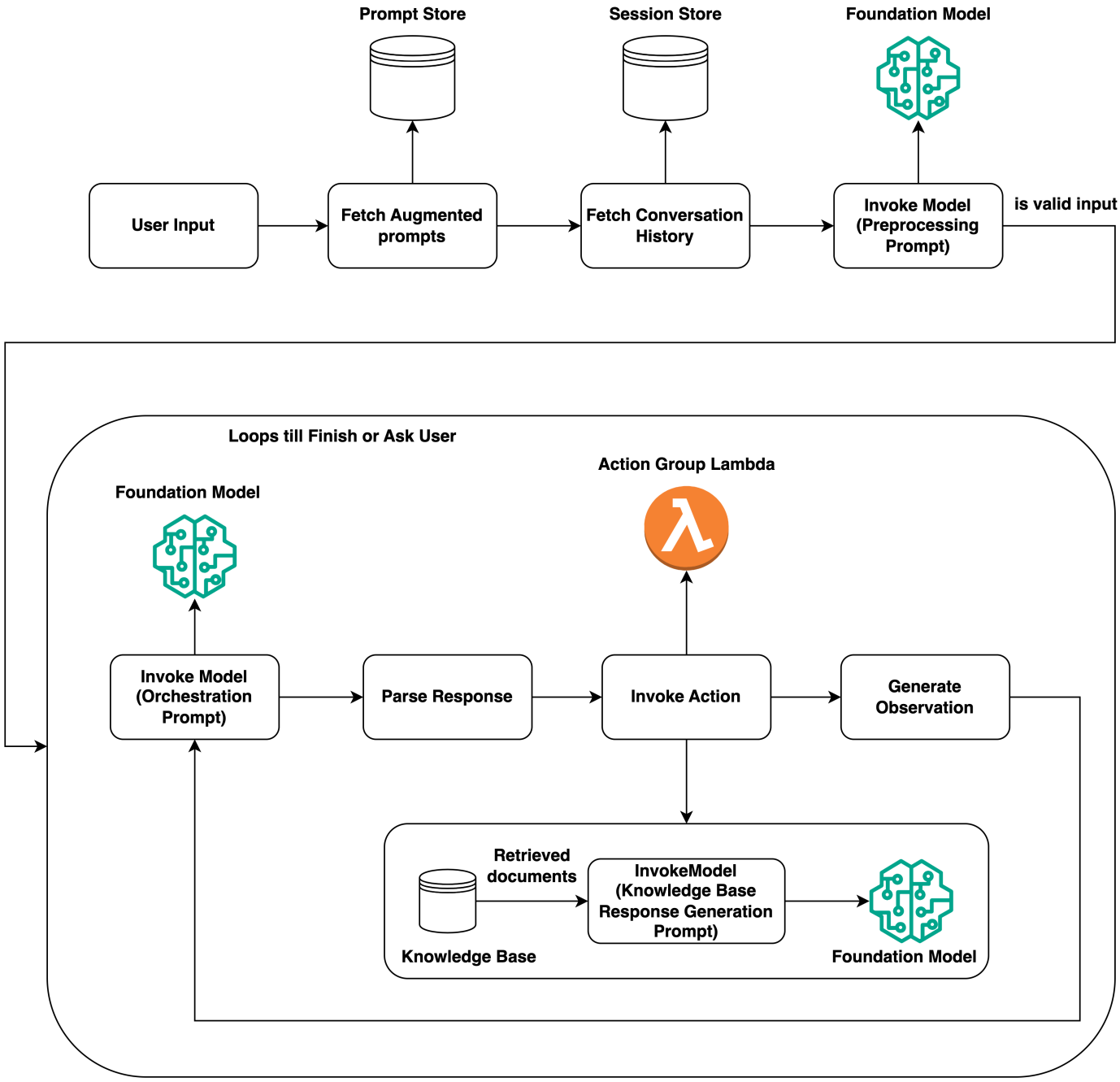
   d. このループは、エージェントがユーザーに応答を返すか、ユーザーに追加情報の入力を求める必要があるまで続きます。

   オーケストレーション中、基本プロンプトテンプレートは、エージェントに追加したエージェント指示、アクショングループ、ナレッジベースで拡張されます。次に、拡張されたベースプロンプトを使用して FM を呼び出します。FM は、ユーザー入力を満たすための最良のステップと軌道を予測します。オーケストレーションの反復のたびに、FM は呼び出す API オペレーションまたはクエリするナレッジベースを予測します。

3. 後処理 – エージェントは最終レスポンスをフォーマットしてユーザーに返します。このステップはデフォルトでオフに設定されています。

エージェントを呼び出すと、実行時にトレースを有効にできます。トレースを使用すると、エージェントシーケンスの各ステップでエージェントの理論的根拠、アクション、クエリ、監視を追跡できます。トレースには、各ステップで基盤モデルに送信された完全なプロンプトと、基盤モデル、API レスポンス、ナレッジベースクエリからの出力が含まれます。トレースを使用して、各ステップでのエージェントの推論を理解できます。詳細については、「Amazon Bedrock でのトレースイベント」を参照してください。

エージェントとのユーザーセッションがより多くのInvokeAgentリクエストを続行すると、会話履歴は保持されます。会話履歴は、オーケストレーションベースプロンプトテンプレートをコンテキストで継続的に強化し、エージェントの精度とパフォーマンスを向上させます。次の図は、ランタイムにおけるエージェントのプロセスを示しています。



Amazon Bedrock のエージェントでサポートされているリージョンとモデル

Amazon Bedrock のエージェントは以下のリージョンでサポートされています。

リージョン
米国東部 (バージニア北部)
米国西部 (オレゴン)

Amazon Bedrock 用エージェントは以下のモデルで使用できます。

モデル名	モデル ID
AnthropicClaude Instantv1	人為的。 claude-instant-v1
AnthropicClaude2.0	anthropic.claude-v2
AnthropicClaude2.1	アンソロピック・ クロード-V 2:1

Agents for Amazon Bedrock の前提条件

IAM ロールに、 Agents for Amazon Bedrock に関連するアクションを実行するために必要なアクセス許可があることを確認します。

エージェントはアクショングループとナレッジベースを使用して、お客様がタスクを実行するのに役立ちます。以下は、各タイプのリソースの簡単な説明です。

- アクショングループ - エージェントがユーザーの実行に役立つアクションを定義します。呼び出すことができる APIs、アクションの処理方法、レスポンスを返す方法が含まれます。
- ナレッジベース — エージェントが顧客のクエリに応答し、生成されたレスポンスを改善するためにクエリを実行できる情報のリポジトリを提供します。

エージェントを作成する前に、次の前提条件を確認し、満たす必要があるものを決定します。

1. アクショングループ を設定します。エージェントがシステムに API コールをオーケストレートするには、少なくとも 1 つのアクショングループを追加する必要があります。アクショングループを後で追加する場合や、エージェントのアクショングループがない場合は、この前提条件をスキップできます。

2. ナレッジベース を設定します。プライベートデータソースを使用して顧客のクエリへの応答を改善するには、少なくとも 1 つのナレッジベースを関連付けることができます。エージェントにナレッジベースが関連付けられていない場合は、この前提条件をスキップできます。
3. 適切なアクセス許可 を持つエージェントのカスタム AWS Identity and Access Management (IAM) サービスロールを作成します。を使用してサービスロールを自動的に作成する場合は AWS Management Console 、この前提条件をスキップできます。

Amazon Bedrock でエージェントを作成する

Amazon Bedrock でエージェントを作成するには、以下のコンポーネントを設定します。

- エージェントの設定。これにより、エージェントの目的が定義され、プロンプトとレスポンスの生成に使用する基盤モデル (FM) が示されます。
- (オプション) エージェントが実行するように設計されているアクションを定義するアクショングループ。
- (オプション) データソースのナレッジベース: エージェントの生成機能を強化します。

エージェントはコンソールまたは API で作成できます。選択した方法に対応するタブを選択し、手順に従います。

Console

方法によって異なります。

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから [エージェント] を選択します。
3. 「エージェント」セクションで、「エージェントを作成」を選択します。
4. [作成] を選択します。
5. 新しく作成したエージェントの Agent Builder が表示され、そこでエージェントを設定できます。
6. エージェント詳細セクションでは、以下の設定を行うことができます。

    a. 「名前」または「説明」は編集できます。

    b. [モデルの選択] では、[変更] を選択して、オーケストレーション中にエージェントが呼び出す FM を選択します。

c. 「エージェントへの指示」に、エージェントに何をすべきか、どのようにユーザーとやり取りすべきかを伝えるための詳細を入力します。このインストラクションは、オーケストレーションプロンプトテンプレートの $instructions$ プレースホルダーに代わるものです。手順の例を以下に示します。

```
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. [その他の設定] を展開すると、以下の設定を変更できます。

ユーザー入力 — 十分な情報がない場合に、エージェントがユーザーに追加情報を要求できるようにするかどうかを選択します。

- [Yes] を選択すると、アクショングループで API を呼び出す必要があるが、API リクエストを完了するための十分な情報がない場合に、エージェントはユーザーに詳細情報の入力を再度求めるObservationを返します。
- [いいえ] を選択した場合、エージェントはユーザーに追加情報を要求せず、タスクを完了するのに十分な情報がないことをユーザーに通知します。

- アイドルセッションタイムアウト — デフォルトでは、Amazon Bedrock エージェントとのセッション中にユーザーが 30 分間応答しなかった場合、エージェントは会話履歴を保持しなくなります。会話履歴は、対話を再開したり、会話のコンテキストでレスポンスを補足するためにも使用されます。このデフォルトの時間を変更するには、「セッションタイムアウト」フィールドに数値を入力し、時間単位を選択します。

e. エージェント設定の設定が完了したら、[次へ] を選択します。

7. アクショングループセクションでは、アクショングループをエージェントに追加できます。アクショングループの設定について詳しくは、を参照してくださいthe section called “新しいアクショングループを作成する”。アクショングループを追加するには、「作成」または「アクショングループの作成」を選択します。

a. アクショングループ呼び出しセクションでは、呼び出す API または関数、および渡す必要があるパラメータをエージェントが予測した後の処理を設定します。これを行うには、アクションの呼び出し時に実行されるビジネスロジックを提供する Lambda 関数を定義します。Lambda 関数のオプションを次の中から選択します。

b. エージェントに別のアクショングループを設定するには、[別のアクショングループを追加] を選択します。アクショングループの追加が完了したら、[次へ] を選択します。

API

エージェントを作成するには、Agents for Amazon Bedrock のビルド時エンドポイントを使用してリクエスト (CreateAgentリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。

コード例を参照してください。

次のリストは、リクエストのフィールドについて説明しています。

- 最低限、以下の必須フィールドを入力してください。

フィールド	簡単な説明
agentName	エージェントの名前。
agentResourceRoleARN	エージェントのサービスロールの ARN
ファウンデーション/モデル	エージェントがオーケストレーションを行うための基盤モデル

- 以下のフィールドはオプションですが、推奨されます。

フィールド	簡単な説明
説明	エージェントが行うことを説明します。
命令	エージェントに何をすべきかを伝える指示。オーケストレーションプロンプトテンプレートで使用されます。

- セキュリティ上の理由から、以下のフィールドを指定できます。

フィールド	簡単な説明
clientToken	API リクエストが 1 回だけ完了することを保証する識別子。
customerEncryptionKeyアーン	エージェントリソースを暗号化するための KMS キーの ARN

フィールド	簡単な説明
アイドルセッション (TTL) InSeconds	エージェントがセッションを終了し、保存されている情報をすべて削除するまでの期間。

- デフォルトのプロンプトテンプレートをオーバーライドしてエージェントの動作をカスタマイズするには、オブジェクトを含めてください。`promptOverrideConfiguration`詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。
- エージェントにタグを付けるには、`Tags`フィールドを使用します。詳細については、「リソースのタグ付け」を参照してください。

エージェントの作成に失敗した場合、CreateAgent`failureReasons``recommendedActions`レスポンス内のオブジェクトはのリストとトラブルシューティング用のリストを返します。

Amazon Bedrock エージェントのアクショングループを作成する

アクショングループは、エージェントがユーザーの実行に役立つアクションを定義します。アクショングループを定義するには、呼び出すことができるシステムに APIs を指定し、アクションを処理する Lambda 関数を記述してレスポンスを返す方法を指定します。アクショングループを作成するには、次のコンポーネントを準備します。

- アクショングループの API の説明、構造、パラメータを含む OpenAPIスキーマを設定します。API スキーマは、次のいずれかの方法でアクショングループに追加できます。
  - 作成したスキーマを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。
  - アクショングループを追加する AWS Management Console ときに、 のインラインスキーマエディタにOpenAPIスキーマを記述します。このオプションは、アクショングループが属するエージェントが既に作成された後にのみ使用できます。
- アクショングループのビジネスロジックを定義する Lambda 関数を作成します。

アクショングループのコンポーネントと、セットアップ後にアクショングループを作成する方法の詳細については、以下のトピックから選択してください。

トピック

- Amazon Bedrock でエージェントのアクショングループのOpenAPIスキーマを定義する
- Amazon Bedrock のエージェントのアクショングループのLambda 関数を定義する
- Amazon Bedrock でエージェントにアクショングループを追加する

Amazon Bedrock でエージェントのアクショングループのOpenAPIスキーマを定義する

Amazon Bedrock でアクショングループを作成するときは、エージェントが呼び出すことができる API オペレーションを定義する必要があります。API オペレーションを定義するには、JSON または YAML 形式の OpenAPIスキーマを作成します。OpenAPI スキーマファイルを作成し、Amazon Simple Storage Service (Amazon S3) にアップロードできます。または、コンソールのOpenAPIテキストエディタを使用して、スキーマを検証することもできます。エージェントを作成したら、アクショングループをエージェントに追加するとき、または既存のアクショングループを編集するときに、テキストエディタを使用できます。詳細については、「エージェントを編集する」を参照してください。

API スキーマの詳細については、以下のリソースを参照してください。

- OpenAPI スキーマの詳細については、 Swaggerウェブサイトの「 OpenAPI仕様」を参照してください。
- API スキーマ作成のベストプラクティスについては、 Swaggerウェブサイトの「API 設計のベストプラクティス」を参照してください。

以下は、アクショングループの OpenAPIスキーマの一般的な形式です。

```
{
    "openapi": "3.0.0",
    "paths": {
        "/path": {
            "method": {
                "description": "string",
                "operationId": "string",
                "parameters": [ ... ],
                "requestBody": { ... },
                "responses": { ... }
            }
        }
    }
```

```
}
```

次のリストでは、OpenAPIスキーマのフィールドについて説明します。

- openapi – (必須) OpenAPI使用されている のバージョン。アクショングループが動作するには、この値が "3.0.0" 以上である必要があります。
- paths — (必須) 個々のエンドポイントへの相対パスが含まれます。各パスはスラッシュ () で始まる必要があります/。
- method — (必須) 使用する方法を定義します。

少なくとも、各メソッドには次のフィールドが必要です。

- description — API オペレーションの説明。このフィールドを使用して、この API オペレーションを呼び出すタイミングと、オペレーションの動作をエージェントに通知します。
- responses – エージェントが API レスポンスで返すプロパティが含まれます。エージェントはレスポンスプロパティを使用してプロンプトを作成し、API コールの結果を正確に処理し、タスクを実行するための正しいステップセットを決定します。エージェントは、オーケストレーションの後続のステップの入力として、1 つのオペレーションからのレスポンス値を使用できます。

次の 2 つのオブジェクト内のフィールドは、エージェントがアクショングループを効果的に活用するためのより多くの情報を提供します。フィールドごとに、true必要に応じて required フィールドの値を に設定し、オプションfalseの場合は に設定します。

- parameters — リクエストに含めることができるパラメータに関する情報が含まれます。
- requestBody — オペレーションのリクエスト本文内のフィールドが含まれます。GET および DELETE メソッドにはこのフィールドを含めないでください。

構造の詳細については、次のタブから選択してください。

responses

```
"responses": {
    "200": {
        "content": {
            "<media type>": {
                "schema": {
                    "properties": {
```

```
                        "<property>": {
                            "type": "string",
                            "description": "string"
                        },
                        ...
                    }
                }
            }
        },
    },
    ...
}
```

responses オブジェクト内の各キーはレスポンスコードで、レスポンスのステータスを記述します。レスポンスコードは、レスポンスの次の情報を含むオブジェクトにマッピングされます。

- content — (各レスポンスに必須) レスポンスのコンテンツ。
- *<media type>* — レスポンス本文の形式。詳細については、 Swaggerウェブサイトの「メディアタイプ」を参照してください。
- schema — (各メディアタイプに必須) レスポンス本体とそのフィールドのデータタイプを定義します。
- properties — (スキーマに items が存在する場合は必須) エージェントは、スキーマで定義したプロパティを使用して、タスクを実行するためにエンドユーザーに返す必要のある情報を決定します。各プロパティには、次のフィールドが含まれます。
  - type — (各プロパティに必須) レスポンスフィールドのデータタイプ。
  - description — (オプション) プロパティについて説明します。エージェントはこの情報を使用して、エンドユーザーに返す必要がある情報を判断できます。

parameters

```
"parameters": [
    {
        "name": "string",
        "description": "string",
        "required": boolean,
        "schema": {
            ...
        }
    },
```

```
    ...
]
```

エージェントは次のフィールドを使用して、アクショングループの要件を実行するためにエンドユーザーから取得する必要がある情報を決定します。

- name – (必須) パラメータの名前。
- description – (必須) パラメータの説明。このフィールドを使用すると、エージェントがエージェントユーザーからこのパラメータを引き出す方法を理解したり、以前のアクションまたはユーザーのエージェントへのリクエストからそのパラメータ値がすでに設定されているかを判断できます。
- required – (オプション) API リクエストに パラメータが必要かどうか。このフィールドを使用して、このパラメータが呼び出しのたびに必要であるか、またはオプションであるかをエージェントに指定します。
- schema — (オプション) 入力データ型と出力データ型の定義。詳細については、 Swaggerウェブサイトのデータモデル (スキーマ) を参照してください。

requestBody

requestBody フィールドの一般的な構造は次のとおりです。

```
"requestBody": {
    "required": boolean,
    "content": {
        "<media type>": {
            "schema": {
                "properties": {
                    "<property>": {
                        "type": "string",
                        "description": "string"
                    },
                    ...
                }
            }
        }
    }
}
```

次のリストでは、各フィールドについて説明します。

- required — (オプション) API リクエストにリクエストボディが必要かどうか。
- content — (必須) リクエスト本文のコンテンツ。
- *<media type>* — (オプション) リクエスト本文の形式。詳細については、 Swaggerウェブサイトの「メディアタイプ」を参照してください。
- schema — (オプション) リクエスト本文とそのフィールドのデータ型を定義します。
- properties – (オプション) エージェントは、スキーマで定義したプロパティを使用して、API リクエストを行うためにエンドユーザーから取得する必要がある情報を決定します。各プロパティには、次のフィールドが含まれます。
  - type — (オプション) リクエストフィールドのデータ型。
  - description — (オプション) プロパティについて説明します。エージェントはこの情報を使用して、エンドユーザーに返す必要のある情報を判断できます。

API スキーマの例

次の API スキーマの例では、保険請求の処理に役立つ API オペラテンのグループを定義します。3 つの APIsは次のように定義されます。

- getAllOpenClaims – エージェントは descriptionフィールドを使用して、未解決のクレームのリストが必要な場合に、この API オペレーションを呼び出す必要があるかどうかを判断できます。responses の properties には、ID、保険契約者、請求のステータスを返すよう指定されています。エージェントはこの情報をエージェントユーザーに返すか、レスポンスの一部または全部を後続の API コールの入力として使用します。
- identifyMissingDocuments – エージェントは descriptionフィールドを使用して、保険請求で不足しているドキュメントを特定する必要がある場合に、この API オペレーションを呼び出す必要があるかどうかを判断できます。name、description、および required フィールドは、未解決請求の固有識別子をカスタマーから引き出す必要があることをエージェントに伝えます。responses の properties には、未解決の保険金請求の ID を返すよう指定します。エージェントは、この情報をエンドユーザーに返すか、レスポンスの一部またはすべてを後続の API コールへの入力として使用します。
- sendReminders — 顧客にリマインダーを送信する必要がある場合、エージェントは descriptionフィールドを使用してこの API オペレーションを呼び出す必要があるかどうかを判断できます。例えば、未処理のクレームに関して保留中のドキュメントに関するリマインダーなどです。properties の requestBodyは、クレーム IDs と保留中のドキュメントを見つける必要があることをエージェントに伝えます。properties の responsesを指定して、リマインダーの

ID とそのステータスを返します。エージェントは、この情報をエンドユーザーに返すか、レスポンスの一部またはすべてを後続の API コールへの入力として使用します。

```
{
    "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."
                        }
                    }
                }
            }
        }
    }
}
```

OpenAPI スキーマのその他の例については、 GitHub ウェブサイトの「https://github.com/OAI/OpenAPI-Specification/tree/main/examples/v3.0。

Amazon Bedrock のエージェントのアクショングループのLambda 関数を定義する

アクショングループのビジネスロジックをプログラミングするには、Lambda 関数を定義する必要があります。Amazon Bedrock エージェントは、アクショングループで呼び出す必要のある API オペレーションを決定すると、関連するメタデータとともに API スキーマの情報を Lambda 関数への入

カイベントとして送信します。関数を作成するには、Lambda 関数の次のコンポーネントを理解する必要があります。

- 入力イベント — エージェントが呼び出す必要があると判断した API オペレーションのリクエスト本文から、関連するメタデータと入力されたフィールドが含まれます。
- Response — API オペレーションから返されたレスポンス本文に関連するメタデータと入力されたフィールドが含まれます。

Lambda 関数を記述して、アクショングループの処理方法を定義し、API レスポンスを返す方法をカスタマイズします。入力イベントの変数を使用して関数を定義し、エージェントにレスポンスを返します。

> **Note**
>
> アクショングループには最大 5 つの API オペレーションを含めることができますが、記述できる Lambda 関数は 1 つだけです。Lambda 関数は一度に 1 つの API オペレーションについて入力イベントを受信してレスポンスを返すことしかできないため、呼び出される可能性のあるさまざまな API オペレーションを考慮して関数を記述する必要があります。

エージェントが Lambda 関数を使用するには、リソースベースのポリシーを関数にアタッチして、エージェントにアクセス権限を付与する必要があります。詳細については、「」の手順に従ってください。Amazon Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシーLambda のリソースベースのポリシーの詳細については、『開発者ガイド』の「Lambda 用のリソースベースのポリシーの使用」を参照してください。 AWS Lambda

トピック

- Amazon Bedrock からのLambda 入力イベント
- Amazon Bedrock へのLambda レスポンスイベント
- アクショングループ Lambda 関数の例

Amazon Bedrock からのLambda 入力イベント

Lambda 関数を使用するアクショングループが呼び出されると、Amazon Bedrock は次の一般的な形式の Lambda 入力イベントを送信します。任意の入力イベントフィールドを使用して関数内のビジネスロジックを操作してアクションを正常に実行するように Lambda 関数を定義できます。Lambda

関数の詳細については、『開発者ガイド』の「イベント駆動型呼び出し」を参照してください。

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"
```

```
    }
}
```

次のリストでは、入力イベントフィールドについて説明します。

- `messageVersion` – Lambda 関数に渡されるイベントデータの形式と Lambda 関数から返す必要があるレスポンスの形式を識別するメッセージのバージョン。Amazon Bedrock ではバージョン 1.0 のみがサポートされています。
- `agent` - アクショングループが属するエージェントの名前、ID、エイリアス、バージョンに関する情報が含まれます。
- `inputText` - 会話のターンに対するユーザー入力。
- `sessionId` - エージェントセッションの一意な識別子。
- `actionGroup` — アクショングループの名前。
- `apiPath`— OpenAPI スキーマで定義されている API オペレーションへのパス。
- `httpMethod`— OpenAPI スキーマで定義されている API 操作のメソッド。
- `parameters` - オブジェクトのリストが含まれます。各オブジェクトには、OpenAPIスキーマで定義されている API オペレーションのパラメータの名前、タイプ、値が含まれます。
- `requestBody`— OpenAPI スキーマで定義されているリクエスト本文とそのプロパティが含まれます。
- `sessionAttributes`— セッション属性とその値が含まれます。これらの属性はセッション全体にわたって保存され、エージェントにコンテキストを提供します。
- `promptSessionAttributes`— プロンプトセッションの属性とその値が含まれます。これらの属性は 1 ターンにわたって保存され、エージェントにコンテキストを提供します。

Amazon Bedrock へのLambda レスポンスイベント

Amazon Bedrock は、以下の形式と一致する、Lambda 関数からのレスポンスを想定しています。レスポンスは API オペレーションから返されるパラメータで構成されます。エージェントは Lambda 関数からのレスポンスを使用して、さらなるオーケストレーションを行ったり、カスタマーにレスポンスを返すのをサポートします。

> **Note**
>
> Lambda エージェントクォータ ペイロードレスポンスクォータについては、を参照してください。

```
{
    "messageVersion": "1.0",
    "response": {
        "actionGroup": "string",
        "apiPath": "string",
        "httpMethod": "string",
        "httpStatusCode": number,
        "responseBody": {
            "<contentType>": {
                "body": "JSON-formatted string"
            }
        }
    },
    "sessionAttributes": {
        "string": "string",
    },
    "promptSessionAttributes": {
        "string": "string"
    }
}
```

以下のリストでは、レスポンスフィールドについて説明しています。

- `messageVersion` – Lambda 関数に渡されるイベントデータの形式と Lambda 関数から返す必要があるレスポンスの形式を識別するメッセージのバージョン。Amazon Bedrock ではバージョン 1.0 のみがサポートされています。
- `response` — API レスポンスに関する次の情報が含まれています。
  - `actionGroup` — アクショングループの名前。
  - `apiPath`— OpenAPI スキーマで定義されている API オペレーションへのパス。
  - `httpMethod`— OpenAPI スキーマで定義されている API 操作のメソッド。
  - `responseBody`— OpenAPI スキーマで定義されている応答本文が含まれます。
- (オプション) `sessionAttributes` – セッション属性とその値が含まれます。
- (オプション) `promptSessionAttributes` – プロンプト属性とその値が含まれます。

アクショングループ Lambda 関数の例

以下は、Lambda 関数をで定義する方法の最小限の例です。Python

```
def lambda_handler(event, context):

    response_body = {
        'application/json': {
            'body': "sample response"
        }
    }

    action_response = {
        'actionGroup': event['actionGroup'],
        'apiPath': event['apiPath'],
        'httpMethod': event['httpMethod'],
        'httpStatusCode': 200,
        'responseBody': response_body
    }

    session_attributes = event['sessionAttributes']
    prompt_session_attributes = event['promptSessionAttributes']

    api_response = {
        'messageVersion': '1.0',
        'response': action_response,
        'sessionAttributes': session_attributes,
        'promptSessionAttributes': prompt_session_attributes
    }

    return api_response
```

Amazon Bedrock でエージェントにアクショングループを追加する

アクショングループのOpenAPIスキーマと Lambda 関数を設定したら、アクショングループを作成できます。選択した方法に対応するタブを選択し、手順に従います。

Console

エージェント を作成するときに、作業中のドラフトにアクショングループを追加できます。

エージェントを作成したら、次のステップを実行してアクショングループを追加できます。

既に作成されたエージェントにアクショングループを追加するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。

2. 左側のナビゲーションペインから エージェントを選択します。次に、「エージェント」セクションでエージェントを選択します。
3. エージェントセクションからエージェントを選択し、ドラフトの作成セクションでドラフトの作成を選択します。
4. [アクショングループ] セクションで [追加] を選択します。
5. [アクショングループの詳細] に入力する
6. インラインスキーマエディタを使用してアクショングループのOpenAPIスキーマを定義するには、次のアクションを実行します。アクショングループの API スキーマの詳細については、「Amazon Bedrock でエージェントのアクショングループのOpenAPIスキーマを定義する」を参照してください。

   a. API OpenAPIスキーマの選択 のインラインスキーマエディタで定義 を選択します。編集可能なサンプルスキーマが表示されます。

   b. [フォーマット] の横にあるドロップダウンメニューを使用して、スキーマのフォーマットを選択します。

   c. S3 から既存のスキーマをインポートして編集するには、[スキーマのインポート] を選択し、S3 URI を指定して [インポート] を選択します。

   d. スキーマを元のサンプルスキーマに復元するには、[リセット] を選択し、もう一度 [リセット] を選択して表示されるメッセージを確認します。
7. [追加] を選択します。問題がなければ、成功を示す緑色のバナーが表示されます。スキーマの検証に問題がある場合は、赤いバナーが表示されます。検証プロセスでは、次の問題が特定されます。

   - スキーマをスクロールして、フォーマットに関するエラーまたは警告がある行を確認します。X はフォーマットエラーを示し、感嘆符はフォーマットに関する警告を示します。
   - 赤いバナーの [詳細を表示] を選択すると、API スキーマのコンテンツに関するエラーのリストが表示されます。
8. テストする前にエージェントに加えた変更を適用するには、[準備] を選択します。

API

アクショングループを作成するには、 Agents for Amazon Bedrock ビルドタイムエンドポイント を使用して CreateAgentActionGroup リクエストを送信します。

コード例を参照

次のリストでは、リクエストのフィールドについて説明します。

- 以下のフィールドが必要です。

フィールド	簡単な説明

- 以下のフィールドはオプションです。

- apiSchema オブジェクト内のアクショングループによって呼び出される APIs を定義するOpenAPIスキーマを指定します。スキーマを JSON ペイロードとして直接指定するか、OpenAPIスキーマを含む Amazon S3 バケットを指定できます。
- (オプション) アクションの呼び出し時に実行されるビジネスロジックを実行する Lambda 関数の ARN を指定します。
- actionGroupState を ENABLED に設定して、エージェントがアクショングループを呼び出せるようにします。
- タスクを完了しようとしたときにエージェントがユーザーに追加情報をリクエストできるようにするには、 parentActionGroupSignatureフィールドを に設定してアクショングループを追加しますAMAZON.UserInput。このアクショングループではdescription、、apiSchema、および actionGroupExecutorフィールドを空白のままにする必要があります。

  オーケストレーション中に、エージェントがアクショングループ内の API を呼び出す必要があると判断した場合、API リクエストを完了するのに十分な情報がないと、エージェントは代わりにこのアクショングループを呼び出し、ユーザーに詳細を要求する監視情報を返します。

ナレッジベースを Amazon Bedrock エージェントに関連付ける

ナレッジベースをまだ作成していない場合は、Amazon ベッドロックのナレッジベース「」を参照してナレッジベースについて学び、作成してください。エージェントの作成中または作成後にナレッジベースを関連付けることができます。ナレッジベースを既存のエージェントに関連付けるには、選択した方法に対応するタブを選択し、そのステップに従います。

Console

ナレッジベースを追加するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エージェントの詳細ページで、「ドラフトの作成」セクションから作業ドラフトを選択します。
4. ナレッジベース セクションで、 を追加 を選択します。
5. 作成したナレッジベースを選択し、エージェントがナレッジベースと通信する方法を指示します。
6. [追加] を選択します。成功バナーが上部に表示されます。
7. テスト前にエージェントに加えた変更を適用するには、テスト前に準備を選択します。

API

ナレッジベースをエージェントに関連付けるには、 Agents for Amazon Bedrock ビルドタイムエンドポイント に AssociateAgentKnowledgeBaseリクエストを送信します。

次のリストでは、リクエストのフィールドについて説明します。

- 以下のフィールドが必要です。

フィールド	簡単な説明
agentId	エージェントの ID
agentVersion	エージェントのバージョン

フィールド	簡単な説明
knowledgeBaseId	ナレッジベースの ID

- 以下のフィールドはオプションです。

フィールド	簡単な説明
説明	エージェントがナレッジベースを使用する方法の説明
knowledgeBaseState	エージェントがナレッジベースにクエリを実行できないようにするには、 を指定します。 DISABLED

Amazon Bedrock エージェントをテストする

エージェントを作成すると、作業中のドラフトが表示されます。作業中のドラフトは、エージェントを反復的に構築するために使用できるエージェントのバージョンです。エージェントに変更を加えるたびに、作業中のドラフトが更新されます。エージェントの設定に問題がなければ、エージェントのスナップショットであるバージョン と、バージョン を指すエイリアス を作成できます。その後、エイリアスを呼び出すことで、エージェントをアプリケーションにデプロイできます。詳細については、「Amazon Bedrock エージェントをデプロイする」を参照してください。

次のリストは、エージェントのテスト方法を示しています。

- Amazon Bedrock コンソールで、 側のテストウィンドウを開き、エージェントが応答するための入力を送信します。作業中のドラフトまたは作成したバージョンを選択できます。
- API では、作業ドラフトは DRAFTバージョンです。静的バージョンを指すテストエイリアス、、または別のエイリアスInvokeAgentで を使用してTSTALIASID、エージェントに入力を送信します。

エージェントの動作のトラブルシューティングに役立つように、 Agents for Amazon Bedrock では、エージェントとのセッション中にトレースを表示できます。トレースは、エージェントの step-by-step 推論プロセスを示します。トレースの詳細については、「」を参照してくださいAmazon Bedrock でのトレースイベント。

エージェントのテスト手順は次のとおりです。選択した方法に対応するタブを選択し、手順に従います。

Console

エージェントをテストするには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エージェントセクションで、エージェントのリストからテストするエージェントのリンクを選択します。
4. 右側のペインにテストウィンドウが表示されます。

> **Note**
>
> テストウィンドウが閉じられている場合は、エージェントの詳細ページの上部にあるテスト、またはその中の任意のページを選択して再度開くことができます。

5. エージェントを作成したら、次のいずれかの方法で準備して、作業中のドラフトの変更でエージェントをパッケージ化する必要があります。
   - テストウィンドウで、 の準備 を選択します。
   - 「ドラフトの作成」ページで、ページの上部にある「準備」を選択します。

> **Note**
>
> 作業中のドラフトを更新するたびに、エージェントを最新の変更でパッケージ化するようにエージェントを準備する必要があります。ベストプラクティスとして、「ドラフトの作成」ページの「エージェントの概要」セクションでエージェントの最終準備時間を常にチェックして、最新の設定でエージェントをテストしていることを確認することをお勧めします。

6. テストするエイリアスと関連するバージョンを選択するには、テストウィンドウ の上部にあるドロップダウンメニューを使用します。デフォルトでは、TestAlias: ドラフトの組み合わせ作業が選択されています。

7. エージェントをテストするには、メッセージを入力し、 の実行を選択します。レスポンスが生成されるのを待っている間、または生成された後に、次のオプションがあります。

   - プロンプト、推論設定、各ステップのエージェントの推論プロセス、アクショングループとナレッジベースの使用など、エージェントのオーケストレーションプロセスの各ステップの詳細を表示するには、トレースを表示を選択します。トレースはリアルタイムで更新されるため、レスポンスが返される前に表示できます。ステップのトレースを展開または折りたたむには、ステップの横にある矢印を選択します。トレースウィンドウの詳細と表示される詳細については、「」を参照してくださいAmazon Bedrock でのトレースイベント。
   - エージェントがナレッジベースを呼び出すと、レスポンスに注釈が含まれます。レスポンスの特定の部分の引用情報を含む S3 オブジェクトへのリンクを表示するには、関連する注釈を選択します。

   テストウィンドウで次のアクションを実行できます。

   - エージェントとの新しい会話を開始するには、更新アイコンを選択します。
   - トレースウィンドウを表示するには、展開アイコンを選択します。トレースウィンドウを閉じるには、縮小アイコンを選択します。
   - テストウィンドウを閉じるには、右矢印アイコンを選択します。

アクショングループとナレッジベースを有効または無効にできます。この機能を使用して、エージェントの動作をさまざまな設定で評価することで、更新する必要があるアクショングループまたはナレッジベースを分離することで、エージェントのトラブルシューティングを行います。

アクショングループまたはナレッジベースを有効にするには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エージェントセクションで、エージェントのリストからテストするエージェントのリンクを選択します。
4. エージェントの詳細ページの「ドラフトの作成」セクションで、「ドラフトの作成」のリンクを選択します。

5. アクショングループまたはナレッジベースセクションで、状態を変更するアクショングループまたはナレッジベースの状態にカーソルを合わせます。
6. [編集] ボタンが表示されます。編集アイコンを選択し、ドロップダウンメニューからアクショングループまたはナレッジベースが を有効または無効にするかどうかを選択します。
7. アクショングループが を無効にした場合、エージェントはアクショングループを使用しません。ナレッジベースが Disabled の場合、エージェントはナレッジベースを使用しません。アクショングループまたはナレッジベースを有効または無効にしてから、テストウィンドウを使用してエージェントのトラブルシューティングを行います。
8. 準備 を選択して、テストする前にエージェントに加えた変更を適用します。

API

エージェントを初めてテストする前に、Amazon Bedrock ビルドタイムエンドポイント のエージェントを使用してPrepareAgentリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)、作業中のドラフトの変更でエージェントをパッケージ化する必要があります。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

コード例を参照

> **Note**
>
> 作業中のドラフトを更新するたびに、エージェントを最新の変更でパッケージ化するようにエージェントを準備する必要があります。ベストプラクティスとして、Amazon Bedrock ビルドタイムエンドポイントのエージェントでGetAgentリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)、エージェントが最新の設定でエージェントをテストしていることを確認するpreparedAt時間を確認することをお勧めします。

エージェントをテストするには、Amazon Bedrock ランタイムエンドポイント のエージェントを使用して InvokeAgentリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)。

コード例を参照

> **Note**
>
> AWS CLI は をサポートしていませんInvokeAgent。

コード例を参照

リクエストには以下のフィールドがあります。

- 最低限、次の必須フィールドを入力します。

フィールド	簡単な説明
agentId	エージェントの ID
agentAliasId	エイリアスの ID。TSTALIASID を使用して DRAFTバージョンを呼び出す
sessionId	セッションの英数字 ID (2 ~ 100 文字)
inputText	エージェントに送信するユーザープロンプト

- 以下のフィールドはオプションです。

フィールド	簡単な説明
enableTrace	TRUE トレース を表示するには、 を指定します。
endSession	このリクエストの後にエージェントとのセッションを終了するTRUEには、 を指定します。
sessionState	セッション全体を通して残sessionAttributes っている や、InvokeAgentターン全体を通して残っている promptSessionAttributes が含まれます。これら

フィールド	簡単な説明
	の属性はエージェントコンテキストを与えます。

レスポンスは chunk オブジェクトのバイト数で返されます。エージェントがナレッジベースにクエリを実行した場合、 には chunkが含まれますcitations。トレースを有効にした場合、trace オブジェクトも返されます。エラーが発生すると、エラーメッセージを含む フィールドが返されます。トレースの読み方の詳細については、「」を参照してくださいAmazon Bedrock でのトレースイベント。

Amazon Bedrock でのトレースイベント

Amazon Bedrock エージェントからの各レスポンスには、エージェントによってオーケストレーションされるステップを詳しく説明するトレースが付属しています。トレースにより、エージェントが会話のその時点で返すレスポンスに至るまでの推論プロセスを追跡できます。

トレースを使用して、ユーザー入力からレスポンスを返すまでの、エージェントのパスを追跡します。トレースは、エージェントが呼び出すアクショングループへの入力と、ユーザーに応答するためにクエリを実行するナレッジベースに関する情報を提供します。さらに、トレースは、アクショングループとナレッジベースが返す出力に関する情報を提供します。エージェントが実行するアクションやナレッジベースに対して行うクエリを決定するために使用する推論を確認できます。トレースのステップが失敗した場合、トレースは失敗の理由を返します。トレースの詳細情報を使用して、エージェントのトラブルシューティングを行います。エージェントに問題があるか、予期しない動作が発生するステップを特定できます。次に、この情報を使用して、エージェントの動作を改善できる方法を検討できます。

トレースの表示

次に、トレースを表示する方法について説明します。選択した方法に対応するタブを選択し、手順に従います。

Console

エージェントとの会話中にトレースを表示するには

にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。

1. エージェントセクションで、エージェントのリストからテストするエージェントのリンクを選択します。
2. 右側のペインにテストウィンドウが表示されます。
3. メッセージを入力し、 の実行を選択します。レスポンスの生成中または生成終了後に、トレースを表示 を選択します。
4. エージェントがオーケストレーションを実行すると、各ステップのトレースをリアルタイムで表示できます。

API

トレースを表示するには、 Agents for Amazon Bedrock ランタイムエンドポイントを使用して InvokeAgentリクエストを送信し、 enableTraceフィールドを に設定しますTRUE。デフォルトでは、トレースは無効になっています。

トレースを有効にすると、InvokeAgentレスポンスchunkで、ストリーム内の各 に TracePart オブジェクトにマッピングするtraceフィールドが付きます。内には、TraceオブジェクトにマッピングするtraceフィールドTracePartがあります。

トレースの構造

トレースは、コンソールと API の両方で JSON オブジェクトとして表示されます。コンソールまたは API Traceの各ステップは、次のいずれかのトレースになります。

- PreProcessingTrace – 前処理ステップの入力と出力をトレースします。ここで、エージェントはユーザー入力をコンテキスト化して分類し、有効かどうかを判断します。
- オーケストレーション – オーケストレーションステップの入力と出力をトレースします。エージェントが入力を解釈し、API オペレーションを呼び出し、ナレッジベースをクエリします。次に、エージェントは出力を返し、オーケストレーションを続行するか、ユーザーに応答します。
- PostProcessingTrace – 後処理ステップの入力と出力をトレースします。ここで、エージェントはオーケストレーションの最終出力を処理し、ユーザーにレスポンスを返す方法を決定します。
- FailureTrace - ステップが失敗した理由を追跡します。

各トレース (を除くFailureTrace) には ModelInvocationInput オブジェクトが含まれています。ModelInvocationInput オブジェクトには、ステップのプロンプトテンプレートで設定された設定と、このステップでエージェントに提供されるプロンプトが含まれています。プロンプトテンプレー

トを変更する方法の詳細については、「」を参照してくださいAmazon Bedrock の高度なプロンプト。ModelInvocationInput オブジェクトの構造は次のとおりです。

```
{
    "traceId": "string",
    "text": "string",
    "type": "PRE_PROCESSING | ORCHESTRATION | KNOWLEDGE_BASE_RESPONSE_GENERATION |
 POST_PROCESSING",
    "inferenceConfiguration": {
        "maximumLength": number,
        "stopSequences": ["string"],
        "temperature": float,
        "topK": float,
        "topP": float
    },
    "promptCreationMode": "DEFAULT | OVERRIDDEN",
    "parserMode": "DEFAULT | OVERRIDDEN",
    "overrideLambda": "string"
}
```

次のリストでは、 ModelInvocationInput オブジェクトのフィールドについて説明します。

- traceId - トレースの一意な識別子。
- text - このステップでエージェントに提供されたプロンプトのテキスト。
- type - エージェントのプロセスの現在のステップ。
- inferenceConfiguration - レスポンス生成に影響する推論パラメータ。詳細については、「推論パラメータ」を参照してください。
- promptCreationMode – このステップでエージェントのデフォルトのベースプロンプトテンプレートが上書きされたかどうか。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。
- parserMode – このステップでエージェントのデフォルトのレスポンスパーサーがオーバーライドされたかどうか。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。
- overrideLambda – デフォルトのパーサーがオーバーライドされた場合に、レスポンスの解析に使用されるパーサー Lambda 関数の Amazon リソースネーム (ARN)。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。

各トレースタイプの詳細については、以下のセクションを参照してください。

PreProcessingTrace

```
{
    "modelInvocationInput": { // see above for details }
    "modelInvocationOutput": {
        "parsedResponse": {
            "isValid": boolean,
            "rationale": "string"
        },
        "traceId": "string"
    }
}
```

は、 ModelInvocationInput オブジェクトと PreProcessingModelInvocationOutput オブジェクトPreProcessingTraceで構成されます。PreProcessingModelInvocationOutput には、以下のフィールドが含まれています。

- parsedResponse - 解析されたユーザープロンプトに関する以下の詳細が含まれます。
  - isValid – ユーザープロンプトが有効かどうかを指定します。
  - rationale - エージェントが次に取るべきステップの理由を指定します。
- traceId - トレースの一意な識別子。

OrchestrationTrace

オーケストレーションは、 ModelInvocationInput オブジェクトと、Rationale 、、InvocationInput監視オブジェクトの任意の組み合わせで構成されます。各オブジェクトの詳細については、次のタブから選択してください。

```
{
    "modelInvocationInput": { // see above for details },
    "rationale": { ... },
    "invocationInput": { ... },
    "observation": { ... }
}
```

Rationale

Rationale オブジェクトには、ユーザー入力に基づくエージェントの推論が含まれています。構造は次のとおりです。

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

次のリストでは、Rationale オブジェクトのフィールドについて説明します。

- traceId - トレースのステップの一意な識別子。
- text – 入力プロンプトに基づくエージェントの推論プロセス。

InvocationInput

InvocationInput オブジェクトには、アクショングループまたはナレッジベースに入力され、呼び出されたりクエリを実行される情報が含まれています。構造は次のとおりです。

```
{
    "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"
    }
}
```

次のリストでは、 InvocationInput オブジェクトのフィールドについて説明します。

- traceId - トレースの一意な識別子。
- invocationType – エージェントがアクショングループまたはナレッジベースを呼び出すか、セッションを終了するかを指定します。
- actionGroupInvocationInput — type が ACTION_GROUP の場合に表示されます。詳細については、「Amazon Bedrock でエージェントのアクショングループのOpenAPIスキーマを定義する」を参照してください。呼び出されるアクショングループの次の入力が含まれます。
  - actionGroupName – エージェントが呼び出すアクショングループの名前。
  - apiPath – API スキーマに従って、呼び出す API オペレーションへのパス。
  - verb – API スキーマに従って、使用されている API メソッド。
  - parameters - オブジェクトのリストが含まれます。各オブジェクトには、API スキーマで定義されている API オペレーションのパラメータの名前、タイプ、値が含まれます。
  - requestBody – API スキーマで定義されているリクエストボディとそのプロパティが含まれます。
- knowledgeBaseLookupInput — type が KNOWLEDGE_BASE の場合に表示されます。詳細については、「Amazon ベッドロックのナレッジベース」を参照してください。ナレッジベースとナレッジベースの検索クエリに関する以下の情報が含まれます。
  - knowledgeBaseId - エージェントが検索するナレッジベースの一意の識別子。
  - text - ナレッジベースに対して実行されるクエリ。

Observation

監視オブジェクトには、アクショングループまたはナレッジベースの結果または出力、またはユーザーへのレスポンスが含まれます。構造は次のとおりです。

```
{
    "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"
    }
}
```

次のリストでは、監視オブジェクトのフィールドについて説明します。

- traceId - トレースの一意な識別子。
- type – エージェントがユーザーにプロンプトを発したり、詳細情報を要求したり、会話を終了したりする場合、アクショングループまたはナレッジベースの結果からエージェントの観察結果を返すかどうかを指定します。
- actionGroupInvocationOutput – アクショングループによって呼び出された API オペレーションによって返された JSON 形式の文字列が含まれます。type が ACTION_GROUP の場合に表示されます。詳細については、「Amazon Bedrock でエージェントのアクショングループのOpenAPIスキーマを定義する」を参照してください。
- knowledgeBaseLookupOutput - データソースの Amazon S3 の場所とともに、プロンプトへの応答に関連するナレッジベースから取得したテキストが含まれます。type が KNOWLEDGE_BASE の場合に表示されます。詳細については、「Amazon ベッドロックの

ナレッジベース」を参照してください。のリスト内の各オブジェクトには、次のフィールドretrievedReferencesが含まれます。

- content – ナレッジベースクエリから返されたナレッジベースの text が含まれます。
- location – 返されたテキストが見つかったデータソースの Amazon S3 URI が含まれます。

- repromptResponse — type が REPROMPT の場合に表示されます。再プロンプトを要求する text と、エージェントが再プロンプトを必要とする理由の source が含まれます。
- finalResponse - type が ASK_USER または FINISH の場合に表示されます。ユーザーに追加情報を求める text またはユーザーへのレスポンスとなるものが含まれます。

PostProcessingTrace

```
{
    "modelInvocationInput": { // see above for details }
    "modelInvocationOutput": {
        "parsedResponse": {
            "text": "string"
        },
        "traceId": "string"
    }
}
```

は、 ModelInvocationInput オブジェクトと PostProcessingModelInvocationOutput オブジェクトPostProcessingTraceで構成されます。には、次のフィールドPostProcessingModelInvocationOutputが含まれます。

- parsedResponse - パーサー関数によってテキストが処理された後にユーザーにtext返す が含まれます。
- traceId - トレースの一意な識別子。

FailureTrace

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

次のリストでは、 FailureTrace オブジェクトのフィールドについて説明します。

- failureReason - ステップが失敗した理由。
- traceId - トレースの一意な識別子。

Amazon Bedrock エージェントを管理する

エージェントを作成したら、必要に応じてその設定を表示または更新できます。設定は作業中のドラフトに適用されます。エージェントが不要になった場合は、削除できます。

トピック

エージェントに関する情報を表示する

エージェントに関する情報を表示する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エージェントの詳細では、次の情報を表示できます。
   - エージェントの概要セクションには、エージェント設定が含まれています。
   - タグセクションには、エージェントに関連付けられているタグが含まれています。詳細については、「リソースのタグ付け」を参照してください。

- 「作業ドラフト」セクションには、作業ドラフトが含まれています。作業中のドラフトを選択すると、次の情報を表示できます。
  - モデルの詳細セクションには、エージェントの作業用ドラフトで使用されるモデルと手順が含まれています。
  - アクショングループ セクションには、エージェントが使用するアクショングループが含まれています。詳細については、Amazon Bedrock エージェントのアクショングループを作成するおよびエージェントのアクショングループを管理するを参照してください。
  - ナレッジベースセクションには、エージェントに関連付けられたナレッジベースが含まれています。詳細については、ナレッジベースを Amazon Bedrock エージェントに関連付けるおよびエージェントナレッジベースの関連付けを管理するを参照してください。
  - 詳細プロンプトセクションには、エージェントのオーケストレーションの各ステップのプロンプトテンプレートが含まれています。詳細については、「Amazon Bedrock の高度なプロンプト」を参照してください。
- バージョンとエイリアスセクションには、アプリケーションへのデプロイに使用できるエージェントのバージョンとエイリアスが含まれています。詳細については、「Amazon Bedrock エージェントをデプロイする」を参照してください。

API

エージェントに関する情報を取得するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用してGetAgentリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、 を指定しますagentId。コード例を参照してください。

エージェントに関する情報を一覧表示するには、Amazon Bedrock ビルドタイムエンドポイントのエージェント を使用してListAgentsリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。コード例を参照してください。以下のオプションパラメータを指定できます。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	maxResults フィールドで指定した数よりも多くの結果がある場合、レスポンスは nextToken 値を返します。結果の次

フィールド	簡単な説明
	のバッチを表示するには、別のリクエストでnextToken 値を送信します。

エージェントのすべてのタグを一覧表示するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントとともにListTagsForResourceリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、エージェントの Amazon リソースネーム (ARN) を含めます。

エージェントを編集する

エージェントを編集する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントの設定を編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エージェントの概要セクションで、編集を選択します。
4. 必要に応じて、フィールド内の既存の情報を編集します。
5. 情報の編集が完了したら、保存して同じウィンドウに留まるか、保存して終了を選択してエージェントの詳細ページに戻ります。成功バナーが上部に表示されます。エージェントに新しい設定を適用するには、バナーで準備を選択します。

エージェント用に別の基盤モデルを試してみたり、エージェント向けの指示を変更したりすることもできます。これらの変更は作業中のドラフトにのみ適用されます。

エージェントが使用する基盤モデルや、エージェントへの指示を変更するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。

2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページの「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. モデルの詳細 セクションで、編集 を選択します。
6. 別のモデルを選択するか、必要に応じてエージェントへの指示を編集します。

> **Note**
>
> 基盤モデルを変更すると、変更したプロンプトテンプレートはそのモデルのデフォルトに設定されます。

7. 情報の編集が完了したら、保存して同じウィンドウに留まるか、保存して終了を選択してエージェントの詳細ページに戻ります。成功バナーが上部に表示されます。
8. テスト前にエージェントに加えた変更を適用するには、テストウィンドウまたは作業ドラフトページの上部にある準備を選択します。

エージェントに関連付けられたタグを編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、「エージェント」セクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. [タグ] セクションで、[タグを管理] を選択します。
5. タグを追加するには、[新しいタグの追加] を選択します。次に、キーを入力し、オプションで値 を入力します。タグを削除するには、[削除] を選択します。詳細については、「リソースのタグ付け」を参照してください。
6. タグの編集が完了したら、送信を選択します。

API

エージェントを編集するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントを使用して UpdateAgentリクエストを送信します (リクエストとレスポンスの形式とフィールドの

詳細については、リンクを参照) 。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。必須フィールドとオプションフィールドの詳細については、「」を参照してくださいAmazon Bedrock でエージェントを作成する。

作業中のドラフトに変更を適用するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともに PrepareAgent リクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

エージェントにタグを追加するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントとともにTagResourceリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、エージェントの Amazon リソースネーム (ARN) を含めます。リクエストボディには tagsフィールドが含まれています。これは、タグごとに指定するキーと値のペアを含むオブジェクトです。

エージェントからタグを削除するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用して UntagResourceリクエスト (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) を送信し、エージェントの Amazon リソースネーム (ARN) を含めます。tagKeys リクエストパラメータは、削除するタグのキーを含むリストです。

エージェントを削除する

エージェントを削除する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。
3. エージェントを削除するには、削除するエージェントの横にあるオプションボタンを選択します。
4. 削除の結果について警告するダイアログボックスが表示されます。エージェントを削除することを確認するには、入力フィールドに「」と入力し、**delete**「削除」を選択します。
5. 削除が完了すると、成功バナーが表示されます。

API

エージェントを削除するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用してDeleteAgentリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、 を指定しますagentId。

デフォルトでは、 skipResourceInUseCheckパラメータは falseであり、リソースが使用中であれば削除は停止されます。skipResourceInUseCheck を に設定するとtrue、リソースが使用中であってもリソースは削除されます。

コード例を参照

トピックを選択して、エージェントのアクショングループまたはナレッジベースを管理する方法を説明します。

トピック

- エージェントのアクショングループを管理する
- エージェントナレッジベースの関連付けを管理する

エージェントのアクショングループを管理する

アクショングループを作成したら、そのアクショングループを表示、編集、または削除できます。変更は、エージェントの作業中のドラフトバージョンに適用されます。

トピック

- アクショングループに関する情報を表示する
- アクショングループを編集する
- アクショングループを削除する

アクショングループに関する情報を表示する

アクショングループに関する情報を表示する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

アクショングループに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. アクショングループ セクションで、情報を表示するアクショングループを選択します。

API

アクショングループに関する情報を取得するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用してGetAgentActionGroupリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) `actionGroupId`、、`agentId`、および を指定しますagentVersion。

エージェントのアクショングループに関する情報を一覧表示するには、Amazon Bedrock ビルドタイムエンドポイント のエージェント を使用してListAgentActionGroupsリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照)。アクショングループを表示する `agentVersion` `agentId`と を指定します。以下のオプションのパラメータを含めることができます。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	`maxResults` フィールドで指定した数よりも多くの結果がある場合、レスポンスは `nextToken` 値を返します。結果の次のバッチを表示するには、別のリクエストで`nextToken` 値を送信します。

コード例を参照

アクショングループを編集する

アクショングループを編集する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

アクショングループを編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. アクショングループ セクションで、編集するアクショングループを選択します。次に、[編集] を選択します。
6. 必要に応じて既存のフィールドを編集します。詳細については、「Amazon Bedrock エージェントのアクショングループを作成する」を参照してください。
7. インラインスキーマエディタでアクショングループのOpenAPIスキーマを定義するには、API スキーマの選択 で、インラインOpenAPIスキーマエディタ で定義を選択します。編集可能なサンプルスキーマが表示されます。次のオプションを設定できます。
   - Amazon S3 から既存のスキーマをインポートして編集するには、スキーマのインポートを選択し、Amazon S3 URI を指定して、インポート を選択します。
   - スキーマを元のサンプルスキーマに復元するには、リセット を選択し、確認 を選択して表示されるメッセージを確認します。
   - スキーマに別の形式を選択するには、JSON というラベルの付いたドロップダウンメニューを使用します。
   - スキーマの視覚的な外観を変更するには、スキーマの下にある歯車アイコンを選択します。
8. エージェントがアクショングループを使用できるかどうかを制御するには、 の有効化または無効化を選択します。この関数を使用して、エージェントの動作のトラブルシューティングを行います。

9. 変更をテストできるように同じウィンドウに留まるには、保存を選択します。アクショングループの詳細ページに戻るには、保存 を選択して終了します。
10. 問題がなければ、成功バナーが表示されます。スキーマの検証に問題がある場合は、エラーバナーが表示されます。エラーのリストを表示するには、バナーで詳細を表示を選択します。
11. テスト前にエージェントに加えた変更を適用するには、テストウィンドウまたは作業ドラフトページの上部にある準備を選択します。

API

アクショングループを編集するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントを使用して UpdateAgentActionGroupリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。を agentVersionとして指定する必要がありますDRAFT。必須フィールドとオプションフィールドの詳細については、「」を参照してくださいAmazon Bedrock エージェントのアクショングループを作成する。

作業中のドラフトに変更を適用するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともにPrepareAgentリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照)。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

アクショングループを削除する

アクショングループを削除する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

アクショングループを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。

4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. アクショングループ セクションで、削除するアクショングループの横にあるオプションボタンを選択します。
6. 削除の結果について警告するダイアログボックスが表示されます。アクショングループを削除することを確認するには、入力フィールドに「」と入力し、**delete**「削除」を選択します。
7. 削除が完了すると、成功バナーが表示されます。
8. テスト前にエージェントに加えた変更を適用するには、テストウィンドウまたは作業ドラフトページの上部にある準備を選択します。

API

アクショングループを削除するには、 DeleteAgentActionGroupリクエストを送信します。削除する actionGroupIdと agentIdおよび agentVersion を指定します。デフォルトでは、skipResourceInUseCheckパラメータは falseであり、リソースが使用中であれば削除は停止されます。skipResourceInUseCheck を に設定するとtrue、リソースが使用中であってもリソースは削除されます。

作業中のドラフトに変更を適用するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともにPrepareAgentリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

エージェントナレッジベースの関連付けを管理する

エージェントを作成したら、ナレッジベースをさらに追加したり編集したりできます。追加と編集は作業中のドラフト内で行います。これらの操作を実行するには、[エージェント] セクションからエージェントを選択し、[作業中のドラフト] セクションで作業中のドラフトを選択します。

トピック

- エージェントナレッジベースの関連付けに関する情報を表示する
- エージェントナレッジベースの関連付けを編集する
- ナレッジベースとエージェントの関連付けを解除する

エージェントナレッジベースの関連付けに関する情報を表示する

ナレッジベースに関する情報を表示する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントに関連付けられているナレッジベースに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. ナレッジベースセクションで、情報を表示するナレッジベースを選択します。

API

エージェントに関連付けられたナレッジベースに関する情報を取得するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともにGetAgentKnowledgeBaseリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。以下のフィールドを指定します。

エージェントに関連付けられたナレッジベースに関する情報を一覧表示するには、Amazon Bedrock ビルドタイムエンドポイント のエージェント を使用してListAgentKnowledgeBasesリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。関連するナレッジベースを表示する `agentVersion` `agentId`と を指定します。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	`maxResults` フィールドで指定した数よりも多くの結果がある場合、レスポンスは `nextToken` 値を返します。結果の次

フィールド	簡単な説明
	のバッチを表示するには、別のリクエストでnextToken 値を送信します。

コード例を参照

エージェントナレッジベースの関連付けを編集する

エージェントナレッジベースの関連付けを編集する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントナレッジベースの関連付けを編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. アクショングループ セクションで、編集するアクショングループを選択します。次に、[編集] を選択します。
6. 必要に応じて既存のフィールドを編集します。詳細については、「ナレッジベースを Amazon Bedrock エージェントに関連付ける」を参照してください。
7. エージェントがナレッジベースを使用できるかどうかを制御するには、有効または無効を選択します。この関数を使用して、エージェントの動作のトラブルシューティングを行います。
8. 変更をテストできるように同じウィンドウに留まるには、保存を選択します。ドラフトの作成ページに戻るには、保存を選択して終了します。
9. テスト前にエージェントに加えた変更を適用するには、テストウィンドウまたは作業ドラフトページの上部にある準備を選択します。

API

エージェントに関連付けられたナレッジベースの設定を編集するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントを使用して UpdateAgentKnowledgeBaseリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。を agentVersionとして指定する必要がありますDRAFT。必須フィールドとオプションフィールドの詳細については、「」を参照してくださいナレッジベースを Amazon Bedrock エージェントに関連付ける。

作業中のドラフトに変更を適用するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともに PrepareAgent リクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

ナレッジベースとエージェントの関連付けを解除する

エージェントからナレッジベースの関連付けを解除する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントからナレッジベースの関連付けを解除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. [エージェント] セクションでエージェントを選択します。
4. エージェントの詳細ページで、「ドラフトの作成」セクションで、作業ドラフトを選択します。
5. ナレッジベースセクションで、削除するナレッジベースの横にあるオプションボタンを選択します。その後、[削除] をクリックします。
6. 表示されるメッセージを確認し、削除を選択します。
7. テスト前にエージェントに加えた変更を適用するには、テストウィンドウまたは作業ドラフトページの上部にある準備を選択します。

API

エージェントからナレッジベースの関連付けを解除するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントにDisassociateAgentKnowledgeBaseリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。関連付けを解除するエージェントの knowledgeBaseIdと agentIdagentVersionおよび を指定します。

作業中のドラフトに変更を適用するには、Amazon Bedrock ビルドタイムエンドポイント のエージェントとともにPrepareAgentリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。リクエストagentIdに を含めます。変更は、TSTALIASIDエイリアスが指すDRAFTバージョンに適用されます。

Amazon Bedrock エージェントをカスタマイズする

エージェントを設定したら、次の機能を使用してその動作をさらにカスタマイズできます。

- 詳細プロンプトでは、プロンプトテンプレートを変更して、ランタイムの各ステップでエージェントに送信されるプロンプトを決定できます。
- セッション状態は、リクエストを送信するとき、または CreateAgentリクエストを使用して実行時に送信するときにビルド時に定義できる属性を含むフィールドですInvokeAgent。これらの属性を使用して、ユーザーとエージェント間の会話でコンテキストを提供および管理できます。

その機能の詳細については、トピックを選択してください。

トピック

- Amazon Bedrock の高度なプロンプト
- セッションコンテキストを制御する

Amazon Bedrock の高度なプロンプト

作成後、エージェントは、次の 4 つのデフォルトのベースプロンプトテンプレート で設定されます。このテンプレートは、エージェントがエージェントシーケンスの各ステップで基盤モデルに送信するプロンプトを構築する方法の概要を示しています。各ステップに含まれる内容の詳細については、「」を参照してくださいランタイムプロセス。

- 前処理

- オーケストレーション
- ナレッジベースの回答生成
- 後処理 (デフォルトでは無効)

プロンプトテンプレートは、エージェントが次の処理を行う方法を定義します。

- 基盤モデル (FMs。
- FM、アクショングループ、ナレッジベース間のオーケストレーション
- レスポンスをフォーマットしてユーザーに返します。

高度なプロンプトを使用すると、これらのプロンプトテンプレートを変更して詳細な設定を提供することで、エージェントの精度を高めることができます。また、数ショットプロンプトの手選り分けされた例を提供することもできます。この場合、特定のタスクのラベル付き例を提供することでモデルのパフォーマンスを向上させることができます。

トピックを選択すると、高度なプロンプトの詳細が表示されます。

トピック

- 高度なプロンプトの用語
- プロンプトテンプレートの設定
- Amazon Bedrock エージェントプロンプトテンプレートのプレースホルダー変数
- Agents for Amazon Bedrock の Parser Lambda 関数

高度なプロンプトの用語

次の用語は、詳細プロンプトの仕組みを理解するのに役立ちます。

- セッション – 同じエージェントに対して行われたInvokeAgentリクエストのグループ。InvokeAgent リクエストを行うと、以前の呼び出しのレスポンスから返された sessionId を再利用して、エージェントとの同じセッションを継続できます。エージェント設定のidleSessionTTLInSeconds時間が経過していない限り、エージェントと同じセッションを維持します。
- ターン — 1 回の InvokeAgent コール。セッションは 1 つ以上のターンで構成されます。
- 反復 – 次のアクションのシーケンス。

1. (必須) 基盤モデルへの呼び出し
2. (オプション) アクショングループの呼び出し
3. (オプション) ナレッジベースの呼び出し
4. (オプション) 詳細情報を求めるユーザーへのレスポンス

エージェントの設定やその時点でのエージェントの要件によっては、アクションがスキップされる場合があります。ターンは、1 回または、複数のイテレーションで構成されます。

- プロンプト – プロンプトは、エージェントへの指示、コンテキスト、テキスト入力で構成されます。テキスト入力は、ユーザーから、またはエージェントシーケンス内の別のステップの出力から取得できます。プロンプトは基盤モデルに提供され、エージェントがユーザー入力に応答して次のステップを決定します。
- ベースプロンプトテンプレート – プロンプトを構成する構造要素。テンプレートは、ユーザーが入力したプレースホルダー、エージェント設定、および実行時のコンテキストで構成され、エージェントがそのステップに達したときに基盤モデルが処理するプロンプトを作成します。これらのプレースホルダーの詳細については、「」を参照してくださいAmazon Bedrock エージェントプロンプトテンプレートのプレースホルダー変数）。高度なプロンプトを使用すると、これらのテンプレートを編集できます。

プロンプトテンプレートの設定

詳細プロンプトでは、次のことができます。

- エージェントシーケンスのさまざまなステップの呼び出しを有効または無効にする。
- 推論パラメータを設定します。
- エージェントが使用するデフォルトのベースプロンプトテンプレートを編集します。独自の設定でロジックをオーバーライドすることで、エージェントの動作をカスタマイズできます。

エージェントシーケンスの各ステップで、以下の部分を編集できます。

- プロンプトテンプレート — テンプレートを編集しているステップで受け取ったプロンプトをエージェントがどのように評価して使用すべきかを説明します。テンプレートを編集する場合、次のツールを使用してプロンプトを操作できます。
  - プロンプトテンプレートプレースホルダー — Agents for Amazon Bedrock の事前定義済み変数で、実行時にエージェント呼び出し時に動的に入力されます。プロンプトテンプレートでは、これらのプレースホルダーが (たとえば) $ で囲まれて表示されます。$instructions$テンプ

レートで使用できるプレースホルダー変数については、を参照してください。Amazon Bedrock エージェントプロンプトテンプレートのプレースホルダー変数

- XML タグ — Anthropic モデルでは、XML タグを使用してプロンプトを構造化および表現できます。最適な結果を得るには、わかりやすいタグ名を使用してください。たとえば、デフォルトのオーケストレーションプロンプトテンプレートには、<examples>いくつかの例を示すために使用されるタグが表示されます)。詳細については、ユーザーガイドの「XML タグを使用する」を参照してください。Anthropic

エージェントシーケンスのどのステップも有効または無効にできます。次の表は、各ステップのデフォルト状態を示しています。

プロンプトテンプレート	デフォルト設定
前処理	有効
オーケストレーション	有効
ナレッジベースの回答生成	有効
後処理	無効

> **Note**
>
> オーケストレーションステップを無効にすると、エージェントは未加工のユーザー入力を基盤モデルに送信し、基本プロンプトテンプレートはオーケストレーションに使用しません。
> 他のステップのいずれかを無効にすると、エージェントはそのステップを完全にスキップします。

- 推論構成 — 使用するモデルによって生成されるレスポンスに影響します。推論パラメータの定義や、さまざまなモデルがサポートするパラメータの詳細については、「基盤モデルの推論パラメータ」を参照してください。
- [(オプション) Parser Lambda 関数] – 未加工の基盤モデル出力を解析する方法と、それをランタイムフローで使用する方法を定義します。この関数は、有効にしたステップからの出力に基づいて動作し、関数で定義したとおりに解析されたレスポンスを返します。

ベースプロンプトテンプレートのカスタマイズ方法によっては、基礎モデルの未加工の出力がテンプレート固有のものになる場合があります。その結果、エージェントのデフォルトパーサーでは出力を正しく解析できない場合があります。カスタムパーサーの Lambda 関数を作成することで、エージェントがユースケースに基づいて未加工の基盤モデル出力を解析しやすくなります。パーサー Lambda 関数とその記述方法の詳細については、を参照してください。Agents for Amazon Bedrock の Parser Lambda 関数

> **Note**
>
> すべてのベーステンプレートに 1 つのパーサー Lambda 関数を定義できますが、各ステップでその関数を呼び出すかどうかを設定できます。エージェントが呼び出せるように、必ず Lambda 関数にリソースベースのポリシーを設定してください。詳細については、「Amazon Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシー」を参照してください。

プロンプトテンプレートを編集したら、エージェントをテストできます。 step-by-step エージェントのプロセスを分析し、意図したとおりに動作しているかどうかを判断するには、トレースをオンにして調べます。詳細については、「Amazon Bedrock でのトレースイベント」を参照してください。

詳細プロンプトは AWS Management Console 、または API を使用して設定できます。

Console

コンソールでは、エージェントを作成した後で詳細プロンプトを設定できます。これらはエージェントの編集中に設定します。

エージェントの詳細プロンプトを表示または編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左のナビゲーションペインの [エージェント] を選択します。次に、「エージェント」セクションでエージェントを選択します。
3. エージェント詳細ページの「作業草案」セクションで、「作業草案」を選択します。
4. 「作業ドラフト」ページの「詳細プロンプト」セクションで、「編集」を選択します。
5. 「詳細プロンプトの編集」ページで、編集するエージェントシーケンスのステップに対応するタブを選択します。

6. テンプレートの編集を有効にするには、「テンプレートのデフォルトを上書き」をオンにします。「テンプレートのデフォルトを上書き」ダイアログボックスで、「確認」を選択します。

> **Warning**
>
> [テンプレートのデフォルトを上書き] をオフにしたり、モデルを変更したりすると、デフォルトの Amazon Bedrock テンプレートが使用され、テンプレートはすぐに削除されます。確認するには、テキストボックスに **`confirm`** を入力し、表示されるメッセージを確認します。

7. エージェントが回答を生成する際にテンプレートを使用できるようにするには、「テンプレートを有効化」をオンにしてください。この設定をオフにすると、エージェントはテンプレートを使用しません。
8. サンプルプロンプトテンプレートを変更するには、プロンプトテンプレートエディターを使用します。
9. 「構成」では、プロンプトの推論パラメータを変更できます。推論パラメータの定義や、さまざまなモデルのパラメータの詳細については、「基盤モデルの推論パラメータ」を参照してください。
10. (オプション) 定義した Lambda 関数を使用して未加工の基礎モデル出力を解析するには、次のアクションを実行します。

> **Note**
>
> 1 つの Lambda 関数がすべてのプロンプトテンプレートに使用されます。

a. [設定] セクションで、[解析に Lambda 関数を使用する] を選択します。この設定をクリアすると、エージェントはプロンプトにデフォルトのパーサーを使用します。

b. パーサー Lambda 関数については、ドロップダウンメニューから Lambda 関数を選択します。

> **Note**
>
> エージェントが Lambda 関数にアクセスできるようにするには、エージェントにアクセス権限をアタッチする必要があります。詳細については、「Amazon

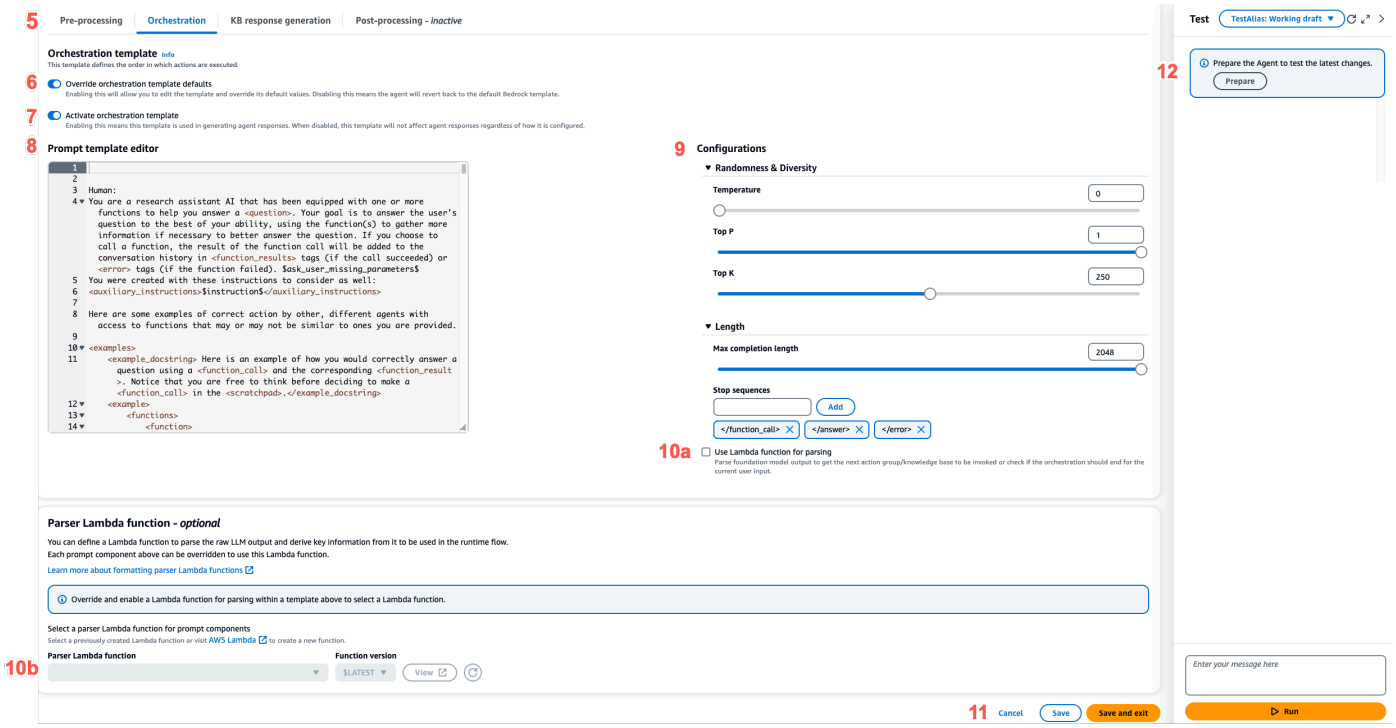
Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシー」を参照してください。

11. 設定を保存するには、以下のオプションのいずれかを選択します。

    a. 更新したエージェントをテストしている間、同じウィンドウを開いたままにして、プロンプト設定を動的に更新できるようにするには、[保存] を選択します。

    b. 設定を保存して [作業ドラフト] ページに戻るには、[保存して終了] を選択します。

12. 更新した設定をテストするには、テストウィンドウで [準備] を選択します。



API

API オペレーションを使用して詳細プロンプトを設定するには、CreateAgentOR UpdateAgentを呼び出して次のオブジェクトを変更します。promptOverrideConfiguration

```
"promptOverrideConfiguration": {
    "overrideLambda": "string",
    "promptConfigurations": [
        {
            "basePromptTemplate": "string",
            "inferenceConfiguration": {
                "maximumLength": int,
```

```
                "stopSequences": [ "string" ],
                "temperature": float,
                "topK": float,
                "topP": float
            },
            "parserMode": "DEFAULT | OVERRIDDEN",
            "promptCreationMode": "DEFAULT | OVERRIDDEN",
            "promptState": "ENABLED | DISABLED",
            "promptType": "PRE_PROCESSING | ORCHESTRATION |
 KNOWLEDGE_BASE_RESPONSE_GENERATION | POST_PROCESSING"
        }
    ]
}
```

1. promptConfigurationsリストには、編集する各プロンプトテンプレートの promptConfiguration オブジェクトを含めます。
2. 変更するプロンプトを promptType フィールドで指定します。
3. プロンプトテンプレートを以下の手順で変更します。
   a. プロンプトテンプレートで basePromptTemplate フィールドを指定します。
   b. 推論パラメータを inferenceConfiguration オブジェクトに含めます。推論の設定の詳細については、「基盤モデルの推論パラメータ」を参照してください。
4. プロンプトテンプレートを有効にするには、promptCreationModeをに設定しますOVERRIDDEN。
5. promptTypeエージェントが現場で手順を実行することを許可または禁止するには、promptState値を変更します。この設定は、エージェントの動作のトラブルシューティングに役立ちます。
   - PRE_PROCESSING、KNOWLEDGE_BASE_RESPONSE_GENERATION、promptStateDISABLEDPO
     たはのステップをに設定すると、エージェントはそのステップをスキップします。
   - promptStateDISABLEDORCHESTRATIONステップをに設定すると、エージェントはオーケストレーション内の基盤モデルにユーザー入力のみを送信します。さらに、エージェントは API オペレーションとナレッジベース間の呼び出しをオーケストレーションすることなく、レスポンスをそのまま返します。
   - デフォルトでは、POST_PROCESSINGこのステップはです。DISABLEDデフォルトでは、、PRE_PROCESSINGORCHESTRATION、KNOWLEDGE_BASE_RESPONSE_GENERATIONおよびステップはですENABLED。

6. 定義した Lambda 関数を使用して未加工の基礎モデルの出力を解析するには、次の手順を実行します。

   a. Lambda 関数を有効にする各プロンプトテンプレートについて、に設定しますparserMode。OVERRIDDEN

   b. overrideLambdaオブジェクトのフィールドに Lambda 関数の Amazon リソースネーム (ARN) を指定します。promptOverrideConfiguration

Amazon Bedrock エージェントプロンプトテンプレートのプレースホルダー変数

エージェントプロンプトテンプレートでプレースホルダー変数を使用できます。プロンプトテンプレートが呼び出されると、変数は既存の設定によって入力されます。タブを選択すると、各プロンプトテンプレートに使用できる変数が表示されます。

Pre-processing

変数	サポートされるモデル	に置き換えられます
$functions$	Anthropic Claude Instant、Claudev2.0	エージェント用に設定されたアクショングループ API オペレーションとナレッジベース。
$tools$	Anthropic Claude v2.1	
$conversation_history$	すべて	現在のセッションの会話履歴
$question$	すべて	セッション内の現在のInvokeAgent 通話のユーザー入力。

Orchestration

変数	サポートされるモデル	に置き換えられます
$functions$	Anthropic Claude Instant、Claudev2.0	エージェント用に設定されたアクショングループ API オ
$tools$	Anthropic Claude v2.1	

変数	サポートされるモデル	に置き換えられます
		ペレーションとナレッジベース。
$agent_scratchpad$	すべて	モデルが実行したおよびそのアクションを書き留めるための領域を指定します。現在のターンにおける以前の反復の予測と出力に置き換えられます。特定のユーザー入力で達成された内容と次のステップのコンテキストをモデルに提供します。
$any_function_name$	Anthropic Claude Instant、Claudev2.0	エージェントのアクショングループに存在する API 名からランダムに選択された API 名。
$conversation_history$	すべて	現在のセッションの会話履歴
$instruction$	すべて	エージェント用に設定されたモデル命令。
$prompt_session_attributes$	すべて	プロンプト全体で保持されるセッション属性
$question$	すべて	セッション内の現在のInvokeAgent 通話のユーザー入力。

次のいずれかのアクションを実行して、エージェントからユーザーに詳細情報を尋ねることを許可する場合は、次のプレースホルダー変数を使用できます。

- コンソールで、エージェントの詳細のユーザー入力で を設定します。
- CreateAgentActionGroup または UpdateAgentActionGroupリクエストAMAZON.UserInputでを parentActionGroupSignatureに設定します。

変数	サポートされるモデル	に置き換えられます
$ask_user_missing_parameters$	Anthropic Claude Instant、Claudev2.0	モデルがユーザーに不足している必須情報を提供するように要求する手順
$ask_user_missing_INFORMATION$	Anthropic Claude v2.1	
$ask_user_confirm_parameters$	すべて	エージェントがまだ受信されていない、または不明なパラメータを確認するようにユーザーに求めるモデルの手順。
$ask_user_function$	すべて	ユーザーに質問する関数。
$ask_user_function_format$	すべて	ユーザーに質問する関数の形式。
$ask_user_input_examples$	すべて	ユーザーに質問すべきタイミングを予測する方法をモデルに伝える例を示します。

Knowledge base response generation

変数	モデル	に置き換えられます
$query$	すべて	次のステップがナレッジベースのクエリであると予測されるときに、オーケストレーションプロンプトモデルのレスポンスによって生成されるクエリ。
$search_results$	すべて	ユーザークエリで取得された結果

Post-processing

変数	モデル	に置き換えられます
$latest_response$	すべて	最後のオーケストレーションプロンプトモデルのレスポンス
$question$	すべて	セッション内の現在のInvokeAgent 通話のユーザー入力。
$responses$	すべて	現在のターンからのアクショングループとナレッジベースの出力。

Agents for Amazon Bedrock の Parser Lambda 関数

各プロンプトテンプレートには、変更できるパーサー Lambda 関数が含まれています。カスタムパーサー Lambda 関数を記述するには、エージェントが送信する入力イベントと、エージェントが Lambda 関数からの出力として期待するレスポンスを理解する必要があります。入力イベントの変数を操作してレスポンスを返すハンドラー関数を作成します。の AWS Lambda 仕組みの詳細については、「 AWS Lambda デベロッパーガイド」の「イベント駆動型呼び出し」を参照してください。

トピック

- Parser Lambda 入力イベント
- Parser Lambda レスポンス
- Parser Lambda の例

Parser Lambda 入力イベント

エージェントからの入力イベントの一般的な構造は次のとおりです。これらのフィールドを使用して Lambda ハンドラー関数を記述します。

```
{
    "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"
}
```

次のリストでは、入力イベントフィールドについて説明します。

- messageVersion – Lambda 関数に渡されるイベントデータの形式と Lambda 関数から返す必要があるレスポンスの形式を識別するメッセージのバージョン。Agents for Amazon Bedrock ではバージョン 1.0 のみがサポートされています。
- agent – プロンプトが属するエージェントの名前、ID、エイリアス、バージョンに関する情報が含まれます。
- invokeModelRawResponse – 出力を解析するプロンプトの未加工の基盤モデル出力。
- promptType – 出力を解析するプロンプトタイプ。
- overrideType – この Lambda 関数がオーバーライドするアーティファクト。現在、OUTPUT_PARSER のみがサポートされています。これは、デフォルトのパーサーがオーバーライドされることを示します。

Parser Lambda レスポンス

エージェントは、以下のフォーマットと一致する、Lambda 関数からのレスポンスを想定しています。エージェントは、レスポンスを使用してさらにオーケストレーションしたり、ユーザーにレスポンスを返したりします。Lambda 関数のレスポンスフィールドを使用して、出力を返す方法を設定します。

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

次のリストでは、Lambda レスポンスフィールドについて説明します。

- messageVersion – Lambda 関数に渡されるイベントデータの形式と Lambda 関数から返す必要があるレスポンスの形式を識別するメッセージのバージョン。Agents for Amazon Bedrock ではバージョン 1.0 のみがサポートされています。
- promptType – 現在のターンのプロンプトタイプ。
- preProcessingParsedResponse – PRE_PROCESSING プロンプトタイプの解析済みレスポンス。
- orchestrationParsedResponse – ORCHESTRATION プロンプトタイプの解析済みレスポンス。詳細については、以下を参照してください。
- knowledgeBaseResponseGenerationParsedResponse – KNOWLEDGE_BASE_RESPONSE_GENERATION プロンプトタイプの解析済みレスポンス。
- postProcessingParsedResponse – POST_PROCESSING プロンプトタイプの解析済みレスポンス。

4 つのプロンプトテンプレートの解析されたレスポンスの詳細については、次のタブを参照してください。

preProcessingParsedResponse

```
{
    "isValidInput": "boolean",
    "rationale": "string"
}
```

preProcessingParsedResponse には、以下のフィールドが含まれています。

- isValidInput - ユーザー入力が有効かどうかを指定します。ユーザー入力の有効性をどのように特徴付けるかを決定する関数を定義できます。
- rationale – ユーザー入力の分類の理由。この理論的根拠は raw レスポンスのモデルによって提供され、Lambda 関数はそれを解析し、エージェントは前処理のためにそれをトレースに表示します。

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"},
                            ...
                        ]
                    },
                    ...
                ]
            }
        },
    }
}
```

には、次のフィールドorchestrationParsedResponseが含まれます。

- rationale – 基盤モデルの出力に基づいた、次にすることの理由。モデル出力から解析する関数を定義できます。
- parsingErrorDetails – これには repromptResponse が含まれます。これは、モデルレスポンスを解析できない場合に、未処理のレスポンスを更新するようにモデルに再び求めるメッセージです。モデルを再プロンプトする方法を操作する関数を定義できます。
- responseDetails – 基盤モデルの出力を処理する方法の詳細が含まれます。エージェントが次に実行するステップである invocationType と、invocationType と一致するはずの 2 つ目のフィールドが含まれます。以下のオブジェクトが可能です。
  - agentAskUser – ASK_USER 呼び出しタイプと互換性があります。この呼び出しタイプはオーケストレーションステップを終了します。ユーザーに追加情報を求める responseText が含まれます。このフィールドを操作する関数を定義できます。
  - actionGroupInvocation – ACTION_GROUP 呼び出しタイプと互換性があります。呼び出すアクショングループと渡すパラメータを決定する関数を定義できます。次のフィールドが含まれます。
    - actionGroupName – 呼び出すアクショングループ。
    - apiName – アクショングループで呼び出す API オペレーションの名前。

    - verb – 使用する API オペレーションのメソッド。
    - actionGroupInput – API オペレーションリクエストで指定するパラメータが含まれます。
- agentKnowledgeBase – KNOWLEDGE_BASE 呼び出しタイプと互換性があります。ナレッジベースにクエリを実行する方法を決定する関数を定義できます。次のフィールドが含まれます。
    - knowledgeBaseId – ナレッジベースの一意の識別子。
    - searchQuery - ナレッジベースに送信するクエリが valueフィールドに含まれます。
- agentFinalResponse – FINISH 呼び出しタイプと互換性があります。この呼び出しタイプはオーケストレーションステップを終了します。responseText フィールドにユーザーへのレスポンス、citations オブジェクトにレスポンスの引用が含まれます。

knowledgeBaseResponseGenerationParsedResponse

```
{
   "generatedResponse": {
        "generatedResponseParts": [
            {
                "text": "string",
                "references": [
                    { "sourceId": "string" },
                    ...
                ]
            },
            ...
        ]
    }
}
```

knowledgeBaseResponseGenerationParsedResponse には、ナレッジベースのクエリgeneratedResponseからの とデータソースのリファレンスが含まれています。

postProcessingParsedResponse

```
{
    "responseText": "string",
    "citations": {
        "generatedResponseParts": [
            {
```

```
                "text": "string",
                "references": [
                    { "sourceId": "string" },
                    ...
                ]
            },
            ...
        ]
    }
}
```

には、次のフィールドpostProcessingParsedResponseが含まれます。

- responseText – エンドユーザに返されるレスポンス。レスポンスをフォーマットする関数を定義できます。
- citations – レスポンスの引用のリストが含まれます。各引用には、引用されたテキストとそのリファレンスが表示されます。

Parser Lambda の例

特定のプロンプトテンプレートのパーサー Lambda 関数の例を表示するには、次のタブから選択します。また、 関数に送信される入力イベントの例と、そこからのレスポンスも示します。lambda_handler 関数は、解析されたレスポンスをエージェントに返します。

Pre-processing

関数の例

```
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>\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<category>B</category>",
    "messageVersion": "1.0",
    "overrideType": "OUTPUT_PARSER",
    "promptType": "PRE_PROCESSING"
}
```

```
{
  "promptType": "PRE_PROCESSING",
  "preProcessingParsedResponse": {
    "rationale": "\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",
    "isValidInput": false
  }
}
```

Orchestration

2 および 2AnthropicClaude.1 Anthropic Claude の関数の例を次に示します。

Anthropic Claude 2

```
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_REMPROMPT_MESSAGE = "The function call format
 is incorrect. The format for function calls to the askuser function must be:
 <function_call>user::askuser(askuser=\"$ASK_USER_INPUT\")</function_call>."
FUNCTION_CALL_STRUCTURE_REPROMPT_MESSAGE = 'The function call format
 is incorrect. The format for function calls must be: <function_call>
```

```
$FUNCTION_NAME($FUNCTION_ARGUMENT_NAME=""$FUNCTION_ARGUMENT_NAME"")</
function_call>.'

logger = logging.getLogger()

# This parser lambda is an example of how to parse the LLM output for the default
 orchestration prompt
def lambda_handler(event, context):
    logger.info("Lambda input: " + str(event))

    # Sanitize LLM response
    sanitized_response = sanitize_response(event['invokeModelRawResponse'])

    # Parse LLM response for any rationale
    rationale = parse_rationale(sanitized_response)

    # Construct response fields common to all invocation types
    parsed_response = {
        'promptType': "ORCHESTRATION",
        'orchestrationParsedResponse': {
            'rationale': rationale
        }
    }

    # Check if there is a final answer
    try:
        final_answer, generated_response_parts = parse_answer(sanitized_response)
    except ValueError as e:
        addRepromptResponse(parsed_response, e)
        return parsed_response

    if final_answer:
        parsed_response['orchestrationParsedResponse']['responseDetails'] = {
            'invocationType': 'FINISH',
            'agentFinalResponse': {
                'responseText': final_answer
            }
        }

        if generated_response_parts:
            parsed_response['orchestrationParsedResponse']['responseDetails']
['agentFinalResponse']['citations'] = {
                'generatedResponseParts': generated_response_parts
            }
```

```
        logger.info("Final answer parsed response: " + str(parsed_response))
        return parsed_response

    # Check if there is an ask user
    try:
        ask_user = parse_ask_user(sanitized_response)
        if ask_user:
            parsed_response['orchestrationParsedResponse']['responseDetails'] = {
                'invocationType': 'ASK_USER',
                'agentAskUser': {
                    'responseText': ask_user
                }
            }

            logger.info("Ask user parsed response: " + str(parsed_response))
            return parsed_response
    except ValueError as e:
        addRepromptResponse(parsed_response, e)
        return parsed_response

    # Check if there is an agent action
    try:
        parsed_response = parse_function_call(sanitized_response, parsed_response)
        logger.info("Function call parsed response: " + str(parsed_response))
        return parsed_response
    except ValueError as e:
        addRepromptResponse(parsed_response, e)
        return parsed_response

    addRepromptResponse(parsed_response, 'Failed to parse the LLM output')
    logger.info(parsed_response)
    return parsed_response

    raise Exception("unrecognized prompt type")

def sanitize_response(text):
    pattern = r"(\\n*)"
    text = re.sub(pattern, r"\n", text)
    return text

def parse_rationale(sanitized_response):
    # Checks for strings that are not required for orchestration
```

```
    rationale_matcher = next((pattern.search(sanitized_response) for pattern in
 RATIONALE_PATTERNS if pattern.search(sanitized_response)), None)

    if rationale_matcher:
        rationale = rationale_matcher.group(1).strip()

        # Check if there is a formatted rationale that we can parse from the string
        rationale_value_matcher = next((pattern.search(rationale) for pattern in
 RATIONALE_VALUE_PATTERNS if pattern.search(rationale)), None)
        if rationale_value_matcher:
            return rationale_value_matcher.group(1).strip()

        return rationale

    return None

def parse_answer(sanitized_llm_response):
    if has_generated_response(sanitized_llm_response):
        return parse_generated_response(sanitized_llm_response)

    answer_match = ANSWER_PATTERN.search(sanitized_llm_response)
    if answer_match and is_answer(sanitized_llm_response):
        return answer_match.group(0).strip(), None

    return None, None

def is_answer(llm_response):
    return llm_response.rfind(ANSWER_TAG) > llm_response.rfind(FUNCTION_CALL_TAG)

def parse_generated_response(sanitized_llm_response):
    results = []

    for match in ANSWER_PART_PATTERN.finditer(sanitized_llm_response):
        part = match.group(1).strip()

        text_match = ANSWER_TEXT_PART_PATTERN.search(part)
        if not text_match:
            raise ValueError("Could not parse generated response")

        text = text_match.group(1).strip()
        references = parse_references(sanitized_llm_response, part)
        results.append((text, references))

    final_response = " ".join([r[0] for r in results])
```

```
    generated_response_parts = []
    for text, references in results:
        generatedResponsePart = {
            'text': text,
            'references': references
        }
        generated_response_parts.append(generatedResponsePart)

    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
    }
```

Anthropic Claude 2.1

```
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_REMPROMPT_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_REPROMPT_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
    }
```

リクエストの例

```
{
    '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<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'
}
```

レスポンスの例

```
{
    '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

関数の例

```
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
```

リクエストの例

```
{
    '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'
}
```

レスポンスの例

```
{
    '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

関数の例

```
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
```

リクエストの例

```
{
    'agent': {
        'alias': 'TSTALIASID',
        'id': 'AGENTID123',
        'name': 'InsuranceAgent',
        'version': 'DRAFT'
    },
    'invokeModelRawResponse': ' <final_response>\\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</
final_response>',
    'messageVersion': '1.0',
    'overrideType': 'OUTPUT_PARSER',
```

```
    'promptType': 'POST_PROCESSING'
}
```

レスポンスの例

```
{
    '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.'
    }
}
```

セッションコンテキストを制御する

セッションコンテキストをより詳細に制御するために、エージェントで SessionState オブジェクトを変更できます。SessionState オブジェクトには、ユーザーの会話中にエージェントの会話コンテキストを提供するために使用できる 2 種類の属性が含まれています。

- sessionAttributes – ユーザーとエージェント間のセッション中に保持される属性。セッション時間制限 (idleSessionTTLinSeconds) を超えていない限り、同じ で行われたすべてのInvokeAgentリクエストは同じセッションにsessionId属します。
- promptSessionAttributes – 1 回のターン (1 回のInvokeAgent呼び出し) で持続する属性。$prompt_session_attributes$ プレースホルダーは、オーケストレーションベースプロンプトテンプレートを編集するときに使用できます。このプレースホルダーには、実行時にpromptSessionAttributesフィールドで指定した属性が入力されます。

SessionState オブジェクトの一般的な形式は次のとおりです。

```
{
    "sessionAttributes": {
        "<attributeName1>": "<attributeValue1>",
        "<attributeName2>": "<attributeValue2>",
        ...
    },
    "promptSessionAttributes": {
        "<attributeName3>": "<attributeValue3>",
        "<attributeName4>": "<attributeValue4>",
        ...
    }
}
```

セッション状態属性は 2 つの異なるステップで定義できます。

- アクショングループを設定し、Lambda 関数 を記述するときは、Amazon Bedrock に返されるレスポンスイベントpromptSessionAttributesに sessionAttributesまたは を含めます。
- 実行時に InvokeAgentリクエストを送信するときは、リクエストボディに sessionState オブジェクトを含めて、会話の途中でセッション状態属性を動的に変更します。

セッション属性の例

次の例では、セッション属性を使用してユーザーにメッセージをパーソナライズします。

1. アプリケーションコードを記述して、ユーザーにエージェントへの名とリクエストの提供を依頼し、回答を変数 *<first_name>* と *<request>* として保存します。
2. アプリケーションコードを記述して、次の本文で InvokeAgentリクエストを送信します。

   ```
   {
       "inputText": "<request>",
       "sessionState": {
           "sessionAttributes": {
               "firstName": "<first_name>"
           }
       }
   }
   ```

3. ユーザーがアプリケーションを使用して名を指定すると、コードは名をセッション属性として送信し、エージェントはセッション の期間中、名を保存します。

4. セッション属性は Lambda 入力イベント で送信されるため、アクショングループの Lambda 関数でこれらのセッション属性を参照できます。例えば、アクション API スキーマがリクエストボディに名を必要とする場合、アクショングループの Lambda 関数を記述するときにfirstNameセッション属性を使用して、API リクエストを送信するときにそのフィールドを自動的に入力できます。

プロンプトセッション属性の例

次の一般的な例では、プロンプトセッション属性を使用してエージェントの一時的なコンテキストを提供します。

1. *<request>* という変数にユーザーリクエストを保存するアプリケーションコードを記述します。
2. ユーザーが *<request>* で相対時間 (「将来の」など) を示す単語を使用し、*<timezone>* という変数に格納する場合は、ユーザーの場所にタイムゾーンを取得するアプリケーションコードを記述します。
3. 次の本文を使用して InvokeAgentリクエストを送信するようにアプリケーションを作成します。

```
{
    "inputText": "<request>",
    "sessionState": {
        "promptSessionAttributes": {
            "timeZone": "<timezone>"
        }
    }
}
```

4. ユーザーが相対時間を示す単語を使用する場合、コードはtimeZoneプロンプトセッション属性を送信し、エージェントはターン の間その属性を保存します。
5. 例えば、ユーザーが と質問した場合**I need to book a hotel for tomorrow**、コードはユーザーのタイムゾーンをエージェントに送信し、エージェントは「将来の」が参照する正確な日付を判断できます。
6. プロンプトセッション属性は、次のステップで使用できます。

   - オーケストレーションプロンプトテンプレートに $prompt_session_attributes$ プレースホルダーを含めると、FM へのオーケストレーションプロンプトにプロンプトセッション属性が含まれます。

- プロンプトセッション属性は Lambda 入力イベントで送信され、API リクエストの入力やレスポンスの で返すために使用できます。

Amazon Bedrock エージェントをデプロイする

Amazon Bedrock エージェントを初めて作成するときは、作業ドラフトバージョン (DRAFT) と、作業ドラフトバージョンを指すテストエイリアス (TSTALIASID) があります。エージェントに変更を加えると、その変更は作業中のドラフトに適用されます。エージェントの動作に満足するまで、作業中のドラフトを繰り返し表示します。その後、エージェントのエイリアスを作成して、アプリケーションへのデプロイと統合のためにエージェントを設定できます。

エージェントをデプロイするには、エイリアス を作成する必要があります。エイリアスの作成中に、Amazon Bedrock はエージェントのバージョンを自動的に作成します。エイリアスはこの新しく作成されたバージョンを指します。または、以前に作成したバージョンのエージェントをエイリアスにポイントすることもできます。次に、そのエイリアスに API コールを行うようにアプリケーションを設定します。

バージョンは、作成時に存在していたリソースを保持するスナップショットです。必要に応じて、作業中のドラフトを変更し、エージェントの新しいエイリアス (つまりバージョン) を作成し続けることができます。Amazon Bedrock で、エージェントの新しいバージョンを作成します。それには、この新しいバージョンをデフォルトで指すエイリアスを作成します。Amazon Bedrock は、バージョンを作成して、バージョン名として 1 から始まる番号を順に付けます。

バージョンは、作成時にエージェントのスナップショットとして機能するため、変更できません。本番環境でエージェントを更新するには、新しいバージョンを作成し、そのバージョンを指すエイリアスを呼び出すようにアプリケーションを設定する必要があります。

エイリアスを使用すると、アプリケーションがバージョンを追跡しなくても、異なるバージョンのエージェントを効率的に切り替えることができます。例えば、すぐに元に戻す必要がある変更がある場合は、以前のバージョンのエージェントを指すようにエイリアスを変更できます。

エージェントをデプロイするには

1. エージェントのエイリアスとバージョンを作成します。選択した方法に対応するタブを選択し、手順に従います。

Console

エイリアス (およびオプションで新しいバージョン) を作成するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エイリアス セクションで、 の作成 を選択します。
4. エイリアスの一意の名前を入力し、オプションの説明を入力します。
5. 以下のオプションのいずれかを選択します。
   - 新しいバージョンを作成するには、新しいバージョンの作成を選択し、それをこのエイリアス に関連付けます。
   - 既存のバージョンを使用するには、既存のバージョンを使用してこのエイリアス を関連付けます。ドロップダウンメニューから、エイリアスを関連付けるバージョンを選択します。
6. [エイリアスを作成] を選択します。成功バナーが上部に表示されます。

API

エージェントのエイリアスを作成するには、Amazon Bedrock ビルドタイムエンドポイントのエージェント を使用して、 CreateAgentAlias リクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)。新しいバージョンを作成し、このエイリアスをそのバージョンに関連付けるには、 `routingConfiguration` オブジェクトを指定しないままにします。

コード例を参照

2. Agents for Amazon Bedrock ランタイムエンドポイント を使用して InvokeAgent リクエストを行うようにアプリケーションを設定して、エージェントをデプロイします (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)。 https://docs.aws.amazon.com/general/latest/gr/bedrock.html#bra-rt`agentAliasId` フィールドで、使用するエージェントのバージョンを指すエイリアスの ID を指定します。

エージェントのバージョンとエイリアスを管理する方法については、次のトピックから選択してください。

トピック

- Amazon Bedrock でエージェントのバージョンを管理する
- Amazon Bedrock でエージェントのエイリアスを管理する

Amazon Bedrock でエージェントのバージョンを管理する

エージェントのバージョンを作成したら、そのバージョンに関する情報を表示または削除できます。エージェントの新しいバージョンは、新しいエイリアスを作成することによってのみ作成できます。

トピック

- Amazon Bedrock でのエージェントのバージョンに関する情報の表示
- Amazon Bedrock でエージェントのバージョンを削除する

Amazon Bedrock でのエージェントのバージョンに関する情報の表示

エージェントのバージョンに関する情報を表示する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントのバージョンに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. バージョンセクションから表示するバージョンを選択します。
4. エージェントのバージョンにアタッチされたモデル、アクショングループ、ナレッジベースの詳細を表示するには、表示する情報の名前を選択します。バージョンの一部を変更することはできません。エージェントを変更するには、作業中のドラフトを使用して新しいバージョンを作成します。

API

エージェントバージョンに関する情報を取得するには、Amazon Bedrock ビルドタイムエンドポイント の エージェントを使用して、 GetAgentVersion リクエスト (リクエストとレス

ポンスの形式とフィールドの詳細のリンクを参照) を送信します。agentId と を指定しますagentVersion。

エージェントのバージョンに関する情報を一覧表示するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用して ListAgentVersions リクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、 を指定しますagentId。以下のオプションパラメータを指定できます。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	maxResults フィールドで指定した数よりも多くの結果がある場合、レスポンスは nextToken 値を返します。結果の次のバッチを表示するには、別のリクエストでnextToken 値を送信します。

Amazon Bedrock でエージェントのバージョンを削除する

エージェントのバージョンを削除する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エージェントのバージョンを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. 削除するバージョンを選択するには、バージョンセクションで、削除するバージョンの横にあるオプションボタンを選択します。
4. [削除] をクリックします。
5. 削除の結果について警告するダイアログボックスが表示されます。バージョンを削除することを確認するには、入力フィールドに「」と入力し、**delete**「削除」を選択します。

6. バージョンが削除されていることを通知するバナーが表示されます。削除が完了すると、成功バナーが表示されます。

API

エージェントのバージョンを削除するには、Amazon Bedrock ビルドタイムエンドポイント のエージェント を使用してDeleteAgentVersionリクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) 。デフォルトでは、skipResourceInUseCheckパラメータは falseで、リソースが使用中の場合、削除は停止します。skipResourceInUseCheck を に設定するとtrue、リソースが使用中であってもリソースは削除されます。

Amazon Bedrock でエージェントのエイリアスを管理する

エージェントのエイリアスを作成したら、そのエイリアスに関する情報を表示したり、編集したり、削除したりできます。

トピック

- Amazon Bedrock でエージェントのエイリアスに関する情報を表示する
- Amazon Bedrock でエージェントのエイリアスを編集する
- Amazon Bedrock でエージェントのエイリアスを削除する

Amazon Bedrock でエージェントのエイリアスに関する情報を表示する

エージェントのエイリアスに関する情報を表示する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エイリアスの詳細を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エイリアスセクションから表示するエイリアスを選択します。

4. エイリアスに関連付けられているエイリアスとタグの名前と説明を表示できます。

API

エージェントエイリアスに関する情報を取得するには、Amazon Bedrock ビルドタイムエンドポイント のエージェント を使用して、 GetAgentAlias リクエスト (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) を送信します。agentId と を指定しますagentAliasId。

エージェントのエイリアスに関する情報を一覧表示するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントとともに ListAgentVersions リクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)、 を指定しますagentId。以下のオプションパラメータを指定できます。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	maxResults フィールドで指定した数よりも多くの結果がある場合、レスポンスは nextToken 値を返します。結果の次のバッチを表示するには、別のリクエストでnextToken 値を送信します。

エイリアスのすべてのタグを表示するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントとともにListTagsForResourceリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照)、エイリアスの Amazon リソースネーム (ARN) を含めます。

Amazon Bedrock でエージェントのエイリアスを編集する

エージェントのエイリアスを編集する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エイリアスを編集するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エイリアスセクションで、編集するエイリアスの横にあるオプションボタンを選択します。
4. エイリアスの名前と説明を編集できます。さらに、次のいずれかのアクションを実行できます。
   - 新しいバージョンを作成し、このエイリアスをそのバージョンに関連付けるには、新しいバージョンの作成を選択し、このエイリアスに関連付けます。
   - このエイリアスを別の既存のバージョンに関連付けるには、既存のバージョンを使用するを選択し、このエイリアスを関連付けます。

エイリアスに関連付けられたタグを追加または削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. エイリアスセクションからタグを管理するエイリアスを選択します。
4. [タグ] セクションで、[タグを管理] を選択します。
5. タグを追加するには、[新しいタグの追加] を選択します。次に、キーを入力し、オプションで値 を入力します。タグを削除するには、[削除] を選択します。詳細については、「リソースのタグ付け」を参照してください。
6. タグの編集が完了したら、送信を選択します。

API

エージェントエイリアスを編集するには、 UpdateAgentAliasリクエストを送信します。すべてのフィールドが上書きされるため、更新するフィールドと、同じままにするフィールドの両方を含めます。

エイリアスにタグを追加するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用してTagResourceリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、エイリアスの Amazon リソースネーム (ARN) を含めます。リクエストボディには tagsフィールドが含まれています。これは、タグごとに指定するキーと値のペアを含むオブジェクトです。

エイリアスからタグを削除するには、Amazon Bedrock ビルドタイムエンドポイントのエージェントを使用して UntagResourceリクエストを送信し (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 、エイリアスの Amazon リソースネーム (ARN) を含めます。tagKeys リクエストパラメータは、削除するタグのキーを含むリストです。

Amazon Bedrock でエージェントのエイリアスを削除する

エージェントのエイリアスを削除する方法については、選択した方法に対応するタブを選択し、そのステップに従います。

Console

エイリアスを削除するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ で Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから エージェントを選択します。次に、エージェントセクションでエージェントを選択します。
3. 削除するエイリアスを選択するには、エイリアスセクションで、削除するエイリアスの横にあるオプションボタンを選択します。
4. [削除] をクリックします。
5. 削除の結果について警告するダイアログボックスが表示されます。エイリアスを削除することを確認するには、**delete**入力フィールドに「」と入力し、「削除」を選択します。
6. エイリアスが削除されていることを通知するバナーが表示されます。削除が完了すると、成功バナーが表示されます。

API

エージェントのエイリアスを削除するには、Amazon Bedrock ビルドタイムエンドポイント のエージェント を使用して、 DeleteAgentAlias リクエストを送信します (リクエストとレスポンスの形式とフィールドの詳細のリンクを参照) 。デフォルトでは、

skipResourceInUseCheckパラメータは falseで、リソースが使用中の場合、削除は停止します。skipResourceInUseCheck を に設定するとtrue、リソースが使用中であってもリソースは削除されます。

コード例を参照

カスタムモデル

モデルのカスタマイズは、特定のユースケースに合わせてパフォーマンスを向上させるために、モデルにトレーニングデータを提供するプロセスです。Amazon Bedrock 基盤モデルをカスタマイズして、パフォーマンスを向上させ、顧客体験を向上させることができます。Amazon Bedrock では現在、以下のカスタマイズ方法を提供しています。

- プレトレーニングを継続

  特定のタイプの入力に慣れ親しませることで、基礎モデルを事前トレーニングするためのラベルのないデータを提供する。特定のトピックのデータを提供して、その分野にモデルを公開することができます。継続的なプレトレーニングプロセスでは、入力データに対応するようにモデルパラメータを微調整し、その分野の知識を向上させます。

  たとえば、ビジネス文書など、大規模な言語モデルのトレーニングには公開されていないプライベートデータを使用してモデルをトレーニングできます。さらに、ラベル付けされていないデータが利用可能になったときに、モデルを再トレーニングしてモデルを改善し続けることができます。
- 微調整

  特定のタスクのパフォーマンスを向上させるようにモデルをトレーニングするために、ラベル付けされたデータを提供する。ラベル付きのサンプルを含むトレーニングデータセットを提供することで、モデルは特定のタイプの入力に対してどのタイプの出力を生成すべきかを関連付けることを学習します。その過程でモデルパラメーターが調整され、トレーニングデータセットに代表されるタスクにおけるモデルのパフォーマンスが向上します。

モデルのカスタマイズクォータについて詳しくは、を参照してください。[モデルカスタマイズのクォータ]

> **Note**
>
> モデルトレーニングの料金は、モデルによって処理されたトークンの数 (トレーニングデータコーパス内のトークン数 × エポック数) と、モデルごとに毎月請求されるモデルストレージに基づいて課金されます。詳細については、「Amazon Bedrock の料金表」を参照してください。

モデルのカスタマイズでは以下のステップを実行します。

1. カスタマイズタスク用のトレーニングと、該当する場合は検証データセットを作成します。
2. 新しいカスタム IAM ロールを使用する予定がある場合は、データの S3 バケットにアクセスするための IAM 権限を設定します。既存のロールを使用することも、適切な権限を持つロールをコンソールに自動的に作成させることもできます。
3. (オプション) セキュリティを強化するために KMS キーや VPC を設定します。
4. ハイパーパラメータ値を調整してトレーニングプロセスを制御し、微調整ジョブまたは継続プレトレーニングジョブを作成します。
5. トレーニングや検証のメトリクスを調べたり、モデル評価を行ったりして、結果を分析します。
6. 新しく作成したカスタムモデル用にプロビジョンドスループットを購入してください。
7. モデル推論などの Amazon Bedrock タスクでは、ベースモデルと同じようにカスタムモデルを使用してください。

トピック

モデルのカスタマイズがサポートされている地域とモデル

次の表は、各カスタマイズ方法の地域サポートを示しています。

リージョン	ファインチューニング	プレトレーニングの継続
米国東部 (バージニア北部)	はい	Yes
米国西部 (オレゴン)	はい	Yes

リージョン	ファインチューニング	プレトレーニングの継続
AWS GovCloud (米国西部)	Yes	No

次の表は、各カスタマイズ方法のモデルサポートを示しています。

モデル名	モデル ID	ファインチューニング	プレトレーニングの継続
Amazon Titan Text G1 - Express	アマゾン。 titan-text-express-v1	はい	Yes
Amazon Titan Text G1 - Lite	アマゾン。 titan-text-lite-v1	はい	Yes
Amazon Titan Image Generator G1	アマゾン。 titan-image-generator-v1	Yes	No
Amazon Titan Multimodal Embeddings G1 G1	アマゾン。 titan-embed-image-v1	Yes	No
Cohere Command	コヒーレ。 command-text-v14	Yes	No
Cohere Command Light	コヒーレ。 command-light-text-v14	Yes	No
MetaLlama 213B	meta.lama2-13 b-chat-v 1	Yes	No
MetaLlama 270	meta.lama2-70 b-chat-v 1	Yes	No

モデルカスタマイズの前提条件

モデルカスタマイズジョブを開始する前に、次の前提条件を満たす必要があります。

1. 微調整ジョブまたは継続的なトレーニング前ジョブを実行するかどうか、使用するモデルを決定します。選択を行うと、カスタマイズジョブに入力するデータセットの形式が決まります。
2. トレーニングデータセットファイルを準備します。選択したカスタマイズ方法とモデルが検証データセットをサポートしている場合は、検証データセットファイルを準備することもできます。以下の「」の手順に従ってデータセットを準備する、ファイルを Amazon S3 バケットにアップロードします。
3. （オプション) の手順に従って、適切なアクセス許可を持つカスタム AWS Identity and Access Management (IAM) モデルカスタマイズ用のサービスロールの作成 サービスロールを作成します。を使用してサービスロールを自動的に作成する場合は AWS Management Console 、この前提条件をスキップできます。
4. （オプション) 追加のセキュリティ設定をセットアップします。
   - カスタムモデルに対して行われた入出力データ、カスタマイズジョブ、または推論リクエストを暗号化できます。詳細については、「モデルカスタマイズジョブとアーティファクトの暗号化」を参照してください。
   - Virtual Private Cloud (VPC) を作成して、カスタマイズジョブを保護できます。詳細については、「VPC を使用してモデルカスタマイズジョブを保護する」を参照してください。

トピック

データセットを準備する

モデルのカスタマイズ作業を開始する前に、最低限必要なトレーニングデータセットの準備が必要です。検証データセットがサポートされているかどうか、またトレーニングデータセットと検証データセットの形式は、以下の要因によって決まります。

- カスタマイズジョブのタイプ (「微調整」または「事前トレーニングの継続」)。
- データの入力と出力のモダリティ。

さまざまなモデルのデータセットとファイルの要件を確認するには、を参照してくださいモデルカスタマイズのクォータ。

ユースケースに関連するタブを選択してください。

Fine-tuning: Text-to-text

text-to-text モデルを微調整するには、複数の JSON 行を含む JSONL ファイルを作成して、トレーニングデータセットとオプションの検証データセットを準備します。JSON の各行は、とフィールドの両方を含むサンプルです。prompt completionトークン数を概算するには、1 トークンを 6 文字として計算します。形式は次のとおりです。

```
{"prompt": "<prompt1>", "completion": "<expected generated text>"}
{"prompt": "<prompt2>", "completion": "<expected generated text>"}
{"prompt": "<prompt3>", "completion": "<expected generated text>"}
```

次に、質問応答タスクの項目の例を示します。

```
{"prompt": "what is AWS", "completion": "it's Amazon Web Services"}
```

Fine-tuning: Text-to-image & Image-to-embeddings

text-to-image OR image-to-embedding モデルを微調整するには、複数の JSON 行を含む JSONL ファイルを作成してトレーニングデータセットを準備します。検証データセットはサポートされていません。各 JSON 行は、画像の Amazon S3 URI である image-ref と、画像のプロンプトとなる可能性がある caption を含むサンプルです。

画像は、PNG または JPEG 形式である必要があります。

```
{"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>"}
```

以下にサンプル項目を示します。

```
{"image-ref": "s3://my-bucket/my-pets/cat.png", "caption": "an orange cat with white
 spots"}
```

Amazon Bedrock がイメージファイルにアクセスできるようにするには、設定した、トレーニングファイルや検証ファイルにアクセスし、S3 に出力ファイルを書き込む権限またはコンソールで

自動的に設定された Amazon Bedrock モデルカスタマイズサービスロールの IAM ポリシーと同様の IAM ポリシーを追加します。トレーニングデータセットで指定する Amazon S3 パスは、ポリシーで指定するフォルダに存在する必要があります。

Continued Pre-training: Text-to-text

text-to-text モデルのプレトレーニングを継続して実施するには、複数の JSON 行を含む JSONL ファイルを作成して、トレーニングデータセットとオプションの検証データセットを準備します。継続的プレトレーニングにはラベルのないデータが含まれるため、各 JSON 行は 1 つのフィールドのみを含むサンプルです。`input`トークン数を概算するには、1 トークンを 6 文字として計算します。形式は次のとおりです。

```
{"input": "<input text>"}
{"input": "<input text>"}
{"input": "<input text>"}
```

以下は、トレーニングデータに含まれる可能性のある項目の例です。

```
{"input": "AWS stands for Amazon Web Services"}
```

VPC を使用してモデルカスタマイズジョブを保護する

モデルカスタマイズジョブを実行すると、ジョブは Amazon S3 バケットにアクセスすることで、入力データをダウンロードしてジョブメトリクスをアップロードします。データへのアクセスを制御するには、Amazon VPC で Virtual Private Cloud (VPC) を使用することをお勧めします。インターネット経由でデータを使用できるように VPC を設定し、代わりに を使用して VPC インターフェイスエンドポイントを作成してデータへのプライベート接続AWS PrivateLinkを確立することで、データをさらに保護できます。Amazon VPC と Amazon Bedrock AWS PrivateLink の統合方法の詳細については、「」を参照してくださいAmazon VPC と を使用してデータを保護する AWS PrivateLink。

モデルカスタマイズジョブのトレーニング、検証、出力データに VPC を設定して使用するには、次のステップを実行します。

トピック

- VPC をセットアップする
- Amazon S3 VPC エンドポイントを作成する
- (オプション) IAM ポリシーを使用して S3 ファイルへのアクセスを制限する

- モデルカスタマイズロールに VPC アクセス許可をアタッチする
- モデルカスタマイズジョブを送信するときに VPC 設定を追加する

VPC をセットアップする

「Amazon VPC の開始方法」および「VPC の作成」のガイダンスに従って、モデルカスタマイズデータにデフォルトの VPC を使用するか、新しい VPC を作成できます。 https://docs.aws.amazon.com/vpc/latest/userguide/vpc-getting-started.html

VPC を作成するときは、標準の Amazon S3 URL (など) が解決されるように、エンドポイントルートテーブルにデフォルトの DNS設定を使用することをお勧めします。 URLs `http://s3-aws-region.amazonaws.com/training-bucket`

Amazon S3 VPC エンドポイントを作成する

インターネットアクセスなしで VPC を設定する場合は、Amazon S3 VPC エンドポイントを作成して、モデルカスタマイズジョブがトレーニングデータと検証データを保存し、モデルアーティファクトを保存する S3 バケットにアクセスできるようにする必要があります。

「Amazon S3 のゲートウェイエンドポイントを作成する」の手順に従って S3 VPC エンドポイントを作成します。 Amazon S3

> **Note**
>
> VPC にデフォルトの DNS 設定を使用しない場合は、エンドポイントルートテーブルを設定して、トレーニングジョブ内のデータの場所の URLs が解決されていることを確認する必要があります。VPC エンドポイントルートテーブルの詳細については、「ゲートウェイエンドポイントのルーティング」を参照してください。

(オプション) IAM ポリシーを使用して S3 ファイルへのアクセスを制限する

リソースベースのポリシーを使用して、S3 ファイルへのアクセスをより厳密に制御できます。次のタイプのリソースベースのポリシーは、任意の組み合わせで使用できます。

- エンドポイントポリシー – エンドポイントポリシーは、VPC エンドポイント経由のアクセスを制限します。デフォルトのエンドポイントポリシーでは、VPC のすべてのユーザーまたはサービスに対して Amazon S3 へのフルアクセスが許可されています。エンドポイントの作成中または作成後に、オプションでリソースベースのポリシーをエンドポイントにアタッチして、エンドポイント

が特定のバケットにアクセスすることを許可する、または特定の IAM ロールのみがエンドポイントにアクセスすることを許可するなどの制限を追加できます。例については、「VPC エンドポイントポリシーの編集」を参照してください。

以下は、VPC エンドポイントにアタッチして、トレーニングデータを含むバケットへのアクセスのみを許可できるポリシーの例です。

```
{
    "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/*"
            ]
        }
    ]
}
```

- バケットポリシー – バケットポリシーは、S3 バケットへのアクセスを制限します。バケットポリシーを使用して、VPC からのトラフィックへのアクセスを制限できます。バケットポリシーをアタッチするには、「バケットポリシーの使用」のステップに従い、aws:sourceVpc、aws:sourceVpce、または aws:VpcSourceIp 条件キーを使用します。例については、「バケットポリシー を使用してアクセスを制御する」を参照してください。

  以下は、VPC からのものでない限り、バケットへのすべてのトラフィックを拒否する出力データを含む S3 バケットにアタッチできるポリシーの例です。

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

モデルカスタマイズロールに VPC アクセス許可をアタッチする

VPC とエンドポイントの設定が完了したら、モデルカスタマイズ IAM ロール に次のアクセス許可をアタッチする必要があります。このポリシーを変更して、ジョブに必要な VPC リソースのみへのアクセスを許可します。*subnet-ids* と を VPC の値*security-group-id*に置き換えます。

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

モデルカスタマイズジョブを送信するときに VPC 設定を追加する

これまでのセクションの手順に従って VPC および必要なロールとアクセス許可を設定し終わったら、この VPC を使用するモデルカスタマイズジョブを作成することができます。

ジョブの VPC サブネットとセキュリティグループを指定すると、Amazon Bedrock はサブネットの 1 つのセキュリティグループに関連付けられた Elastic Network Interface (ENI) を作成します。ENI により、Amazon Bedrock ジョブは VPC 内のリソースに接続できます。ENI については、「Amazon VPC ユーザーガイド」の「Elastic Network Interfaces」を参照してください。Amazon Bedrock は、作成した ENI に `BedrockManaged` および `BedrockModelCusomizationJobArn` タグを付けます。

アベイラビリティーゾーンごとに少なくとも 1 つのサブネットを指定することをお勧めします。

セキュリティグループを使用すると、VPC リソースへの Amazon Bedrock のアクセスを制御するためのルールを設定できます。

コンソールまたは API を使用して、 を使用するように VPC を設定できます。選択した方法に対応するタブを選択し、手順に従います。

Console

Amazon Bedrock コンソールでは、モデルカスタマイズジョブを作成するときに、オプションの [VPC の設定] セクションで VPC サブネットとセキュリティグループを指定します。ジョブの設定の詳細については、「」を参照してくださいモデルカスタマイズジョブを送信。

> **Note**
>
> VPC 設定を含むジョブの場合、コンソールはサービスロールを自動的に作成できません。のガイダンスに従ってモデルカスタマイズ用のサービスロールの作成、カスタムロールを作成します。

API

CreateModelCustomizationJob リクエストを送信するときに、次の例のように、 をリクエストパラメータVpcConfigとして含めて、使用する VPC サブネットとセキュリティグループを指定できます。

```
"VpcConfig": {
"SecurityGroupIds": [
    "sg-0123456789abcdef0"
],
"Subnets": [
    "subnet-0123456789abcdef0",
    "subnet-0123456789abcdef1",
    "subnet-0123456789abcdef2"
]
}
```

モデルカスタマイズジョブを送信

Amazon Bedrock コンソールまたは API の微調整または継続的プレトレーニングを使用してカスタムモデルを作成できます。カスタマイズジョブには数時間かかることがあります。ジョブの所要時間は、トレーニングデータのサイズ (レコード、入力トークン、出力トークンの数)、エポック数、バッチサイズによって異なります。選択した方法に対応するタブを選択し、手順に従います。

Console

コンソールでモデルカスタマイズジョブを送信するには、次の手順を実行します。

1. Amazon Bedrock コンソールで、左側のナビゲーションペインから [基盤モデル] の下にある [カスタムモデル] を選択します。
2. [モデル] タブで、[モデルをカスタマイズ] を選択し、次に [微調整ジョブの作成] または [継続的なプレトレーニングジョブの作成] を選択します。
3. 「モデルの詳細」セクションで、次の操作を行います。
   a. 独自のデータでカスタマイズするモデルを選択し、生成されたモデルに名前を付けます。
   b. (オプション) デフォルトでは、Amazon Bedrock はが所有および管理するキーを使用してモデルを暗号化します。 AWSカスタム KMS キーを使用するには、[モデル暗号化] を選択し、キーを選択します。
   c. (オプション) タグをカスタムモデルに関連付けるには、[タグ] セクションを展開して [新しいタグを追加] を選択します。
4. 「Job 設定」セクションで、ジョブの名前を入力し、オプションでジョブに関連付けるタグを追加します。
5. (オプション) 仮想プライベートクラウド (VPC) を使用してトレーニングデータとカスタマイズジョブを保護するには、入力データと出力データの Amazon S3 ロケーション、サブネット、セキュリティグループを含む VPC を VPC 設定セクションで選択します。

   > **Note**
   >
   > VPC 設定を含めると、コンソールはジョブの新しいサービスロールを作成できません。カスタムサービスロールを作成し、モデルカスタマイズロールに VPC アクセス許可をアタッチするで説明されている例と同様の権限を追加します。

6. Input data セクションで、トレーニングデータセットファイルと、該当する場合は検証データセットファイルの S3 ロケーションを選択します。
7. Hyperparameters セクションに、トレーニングに使用するハイパーパラメータの値を入力します。
8. 「出力データ」セクションに、Amazon Bedrock がジョブの出力を保存する Amazon S3 の場所を入力します。Amazon Bedrock は、各エポックのトレーニング損失メトリクスと検証損失メトリクスは、ユーザーが指定するロケーションの個別のファイルに保存します。

9. [サービスアクセス] セクションで、次のいずれかの操作を行います。
   - 既存のサービスロールを使用 - ドロップダウンリストからサービスロールを選択します。適切なアクセス許可を持つカスタムロールをセットアップする方法の詳細については、「モデルカスタマイズ用のサービスロールの作成」を参照してください。
   - 新しいサービスロールを作成して使用 - サービスロールの名前を入力します。
10. [モデルの微調整] または [事前トレーニングの継続ジョブを作成] を選択してジョブを開始します。

API

リクエスト

モデルカスタマイズジョブを送信するには、Amazon Bedrock コントロールプレーンエンドポイントを使用して CreateModelCustomizationJob(リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照してください) リクエストを送信します。最低限、以下のフィールドを指定する必要があります。

- `roleArn`— モデルをカスタマイズする権限を持つサービスロールの ARN。Amazon Bedrock では、コンソールを使用する場合は適切な権限を持つロールを自動的に作成できます。または、の手順に従ってカスタムロールを作成できます。モデルカスタマイズ用のサービスロールの作成

  > **Note**
  >
  > `vpcConfig`フィールドを含める場合は、そのロールに VPC にアクセスするための適切な権限があることを確認してください。例については、モデルカスタマイズロールに VPC アクセス許可をアタッチするを参照してください。

- `baseModelIdentifier`— カスタマイズする基盤モデルのモデル ID または ARN。
- `customModelName` - 新しくカスタマイズしたモデルに付ける名前。
- `jobName` - トレーニングジョブに付ける名前。
- `hyperParameters`— モデルのカスタマイズプロセスに影響するハイパーパラメータ。
- `trainingDataConfig`— トレーニングデータセットの Amazon S3 URI を含むオブジェクト。カスタマイズ方法とモデルによっては、を含めることもできます`validationDataConfig`。データセットの準備の詳細については、を参照してくださいデータセットを準備する。

- outputDataConfig— 出力データを書き込むための Amazon S3 URI を含むオブジェクト。

を指定しない場合customizationType、モデルのカスタマイズ方法はデフォルトでになりますFINE_TUNING。

リクエストが複数回完了しないようにするには、を含めてくださいclientRequestToken。

以下のオプションフィールドを追加の設定に追加できます。

- jobTagsおよび/または customModelTags — タグをカスタマイズジョブまたは生成されたカスタムモデルに関連付けます。
- customModelKmsKeyId— カスタムモデルを暗号化するためのカスタム KMS キーを含めます。
- vpcConfig— トレーニングデータとカスタマイズジョブを保護するための仮想プライベートクラウド (VPC) の設定を含めてください。

レスポンス

jobArnレスポンスから返されるのは、ジョブの監視や停止に使用できます。

コード例を参照してください。

モデルカスタマイズジョブの管理

モデルカスタマイズジョブを開始すると、その進行状況を追跡したり停止したりできます。API を使用してこれを行う場合は、が必要になりますjobArn。jobArn は、次のいずれかの方法で見つけることができます。

1. Amazon ベッドロック· コンソールで
    1. 左側のナビゲーションペインの [Foundation モデル] で [カスタムモデル] を選択します。
    2. トレーニングジョブテーブルからジョブを選択すると、ジョブの ARN などの詳細が表示されます。
2. jobArnCreateModelCustomizationJobジョブを作成した呼び出しまたは呼び出しから返された応答のフィールドを調べてください。CreateModelCustomizationJob

モデルカスタマイズジョブを監視します。

ジョブを開始したら、コンソールまたは API でその進行状況を監視できます。選択した方法に対応するタブを選択し、手順に従います。

Console

微調整ジョブのステータスを監視するには

1. Amazon Bedrock コンソールで、左側のナビゲーションペインから [基盤モデル] の下にある [カスタムモデル] を選択します。
2. 「トレーニングジョブ」タブを選択すると、開始した微調整ジョブが表示されます。[ステータス] 列を見て、ジョブの進行状況をモニタリングします。
3. ジョブを選択すると、トレーニングに入力した詳細が表示されます。

API

すべてのモデルカスタマイズジョブに関する情報を一覧表示するには、Amazon Bedrock CreateModelCustomizationJobコントロールプレーンエンドポイントを使用してリクエストを送信します。CreateModelCustomizationJob使用できるフィルタについては、を参照してください。

モデルカスタマイズジョブのステータスをモニタリングするには、Amazon Bedrock GetModelCustomizationJobコントロールプレーンエンドポイントを使用してジョブのリクエストを送信します。`jobArn`

モデルカスタマイズジョブのすべてのタグを一覧表示するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (ListTagsForResourceリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信し、ジョブの Amazon リソースネーム (ARN) を含めます。

コード例を参照してください。

モデルカスタマイズジョブを停止する

Amazon Bedrock モデルのカスタマイズジョブは、進行中でも停止できます。選択した方法に対応するタブを選択し、手順に従います。

> **Warning**
>
> ただし、停止したジョブを再開することはできません。Amazon Bedrock は、ユーザーがジョブを停止する前に、Amazon Bedrock がモデルのトレーニングに使用していたトークンの料金を請求します。Amazon Bedrock は、停止済みのジョブの中間カスタムモデルを作成しません。

Console

モデルカスタマイズジョブを停止するには

1. Amazon Bedrock コンソールで、左側のナビゲーションペインから [基盤モデル] の下にある [カスタムモデル] を選択します。
2. Training Jobs タブで、停止するジョブの横にあるラジオボタンを選択するか、停止するジョブを選択して詳細ページに移動します。
3. [ジョブを停止] ボタンを選択します。ジョブを停止できるのは、ステータスがである場合のみですTraining。
4. トレーニングジョブを停止すると再開できないことを警告するモーダルが表示されます。[ジョブを停止] を選択して確定します。

API

モデルのカスタマイズジョブを停止するには、ジョブのを使用して Amazon Bedrock コントロールプレーンエンドポイントで CreateModelCustomizationJob(リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) リクエストを送信します。jobArn

ジョブを停止できるのは、IN_PROGRESSステータスがである場合のみです。statusGetModelCustomizationJobリクエストで確認してください。システムはそのジョブに終了のマークを付け、状態を STOPPING に設定します。ジョブが停止すると、状態はになりますSTOPPED。

コード例を参照してください。

モデルカスタマイズジョブの結果を分析する

モデルのカスタマイズジョブが完了すると、ジョブの送信時に指定した出力 S3 フォルダー内のファイルを確認したり、モデルに関する詳細を確認したりして、トレーニングプロセスの結果を分析できます。Amazon Bedrock は、 AWSカスタマイズしたモデルをお客様のアカウントに合わせたマネージドストレージに保存します。

モデル評価ジョブを実行してモデルを評価することもできます。詳細については、「モデル評価」を参照してください。

モデルカスタマイズジョブの S3 出力には、S3 フォルダーに次の出力ファイルが含まれます。検証アーティファクトは、検証データセットを含めた場合にのみ表示されます。

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

`step_wise_training_metrics.csv` および `validation_metrics.csv` ファイルを使用すると、モデルカスタマイズジョブを分析できるだけでなく、必要に応じてモデルを調整することもできます。

`step_wise_training_metrics.csv`ファイル内の列は次のとおりです。

- step_number — トレーニングプロセスのステップ。0 から始まります。
- epoch_number — トレーニングプロセスのエポックです。
- training_loss — モデルがトレーニングデータにどの程度適合しているかを示します。値が小さいほど適合度が高いことを示します。
- perplexity — モデルがトークンのシーケンスをどれだけうまく予測できるかを示します。値が低いほど、予測能力が高いことを示します。

`validation_metrics.csv`ファイル内の列はトレーニングファイルと同じですが、`validation_loss` (モデルが検証データにどの程度適合しているか) `training_loss` の代わりに表示される点が異なります。

出力ファイルは、https://console.aws.amazon.com/s3 を直接開くか、モデル詳細内の出力フォルダーへのリンクを見つけることで見つけることができます。選択した方法に対応するタブを選択し、手順に従います。

Console

1. Amazon Bedrock コンソールで、左側のナビゲーションペインから [基盤モデル] の下にある [カスタムモデル] を選択します。
2. 「モデル」タブで、モデルを選択して詳細を表示します。Job 名はモデルの詳細セクションにあります。
3. 出力 S3 ファイルを表示するには、[出力データ] セクションで S3 の場所を選択します。
4. モデルのJob 名と一致する名前のフォルダーで、トレーニングと検証のメトリクスファイルを検索します。

API

すべてのカスタムモデルに関する情報を一覧表示するには、Amazon Bedrock コントロールプレーンエンドポイントを使用して ListCustomModels(リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照してください) リクエストを送信します。ListCustomModels使用できるフィルタについては、を参照してください。

カスタムモデルのすべてのタグを一覧表示するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (ListTagsForResourceリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信し、カスタムモデルの Amazon リソースネーム (ARN) を含めます。

モデルカスタマイズジョブのステータスをモニタリングするには、以下のいずれかの Amazon Bedrock コントロールプレーンエンドポイントを使用して GetCustomModel(リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) リクエストを送信します。`modelIdentifier`

- モデルに付けた名前。
- モデルの ARN。

モデルのカスタマイズジョブについては、`trainingMetrics``validationMetrics`GetModelCustomizationJobGetCustomModelまたはのレスポンスでとが表示されます。

トレーニングと検証のメトリクスファイルをダウンロードするには、「オブジェクトのダウンロード」の手順に従ってください。で指定した S3 URI を使用してくださいoutputDataConfig。

コード例を参照してください。

カスタムモデルを使用する

カスタマイズされたモデルを使用する前に、そのモデルのプロビジョンドスループットを購入する必要があります。プロビジョンドスループットの詳細については、を参照してください。Amazon Bedrock のプロビジョニングされたスループットその後、生成されたプロビジョニング済みモデルを推論に使用できます。選択した方法に対応するタブを選択し、手順に従います。

Console

カスタムモデルのプロビジョンドスループットを購入するには

1. Amazon Bedrock コンソールで、左側のナビゲーションペインから [基盤モデル] の下にある [カスタムモデル] を選択します。
2. モデルタブで、プロビジョンドスループットを購入したいモデルの横にあるラジオボタンを選択するか、モデル名を選択して詳細ページに移動します。
3. [プロビジョンドスループットを購入] を選択します。
4. 詳細については、Amazon Bedrock モデル用のプロビジョニングされたスループットを購入するの手順に従ってください。
5. カスタムモデル用のプロビジョンドスループットを購入したら、の手順に従ってください。プロビジョニングされたスループットを使用して推論を実行する

カスタムモデルの使用をサポートする操作を実行すると、モデル選択メニューのオプションとしてカスタムモデルが表示されます。

API

カスタムモデルのプロビジョンドスループットを購入するには、の手順に従って、Amazon Bedrock Amazon Bedrock モデル用のプロビジョニングされたスループットを購入するコントロールプレーンエンドポイントを使用して CreateProvisionedModelThroughput(リクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照してください) リクエストを送信します。カスタムモデルの名前または ARN をとして使用しま

す。modelIdprovisionedModelArnレスポンスからが返されるので、InvokeModelOR modelId InvokeModelWithResponseStreamリクエストを行うときにとして使用できます。

コード例を参照してください。

モデルカスタマイズ用のコードサンプル

以下のコードサンプルは、基本データセットの準備、権限の設定、カスタムモデルの作成、出力ファイルの表示、モデルのスループットの購入、モデルに対する推論の実行方法を示しています。これらのコードスニペットは特定のユースケースに合わせて変更できます。

1. トレーニングデータセットを準備します。

   a. 次の 1 行を含むトレーニングデータセットファイルを作成し、*train.jsonl* という名前を付けます。

   ```
   {"prompt": "what is AWS", "completion": "it's Amazon Web Services"}
   ```

   b. トレーニングデータ用と、出力データ用に S3 バケットを 1 つ作成します (名前は一意である必要があります)。

   c. *train.jsonl ##########################*。

2. トレーニングにアクセスするためのポリシーを作成し、Amazon Bedrock の信頼関係を持つ IAM ロールにアタッチします。選択した方法に対応するタブを選択し、手順に従います。

   Console

   1. S3 ポリシーを作成します。

      a. https://console.aws.amazon.com/iam にある IAM コンソールに移動し、左側のナビゲーションペインから [ポリシー] を選択します。

      b. [ポリシーの作成] を選択し、[JSON] を選択してポリシーエディターを開きます。

      c. *$ {training-bucket} # $ {output-bucket}* を自分のバケット名に置き換えて、次のポリシーを貼り付け、[次へ] を選択します。

      ```
      {
          "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. ポリシーに名前を付け、[ポリシーの作成 ***MyFineTuningDataAccess***] を選択します。

2. IAM ロールを作成し、ポリシーをアタッチします。

a. 左側のナビゲーションペインから [ロール] を選択し、[ロールの作成] を選択します。

b. [カスタム信頼ポリシー] を選択し、次のポリシーを貼り付けて [次へ] を選択します。

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

```
        }
    ]
}
```

c. *MyFineTuningDataAccess*作成したポリシーを検索し、チェックボックスを選択して、[次へ] を選択します。

d. ロールに名前を付け、[*MyCustomizationRole######*] を選択します。

CLI

1. *BedrockTrust.json* というファイルを作成し、次のポリシーをそのファイルに貼り付けます。

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

2. *MyFineTuningDataAccess.json* という名前のファイルをもう1つ作成し、その中に次のポリシーを貼り付けます。*$ {training-bucket} # $ {output-bucket}* は、ご自分のバケット名に置き換えてください。

```
{
    "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. ターミナルで、作成したポリシーが保存されているフォルダーに移動します。
4. という名前の IAM CreateRole*MyCustomizationRole*ロールの作成をリクエストし、*BedrockTrust####.json* 信頼ポリシーを添付します。

```
aws iam create-role \
    --role-name MyCustomizationRole \
    --assume-role-policy-document file://BedrockTrust.json
```

5. *MyFineTuningDataAccess####.json* ファイルを使用して S3 CreatePolicyデータアクセスポリシーの作成をリクエストします。Arnレスポンスはポリシーの an を返します。

```
aws iam create-policy \
    --policy-name MyFineTuningDataAccess \
    --policy-document file://myFineTuningDataAccess.json
```

6. S3 AttachRolePolicyデータアクセスポリシーをロールにアタッチするようリクエストし、前のステップの応答の ARN に置き換えます。policy-arn

```
aws iam attach-role-policy \
    --role-name MyCustomizationRole \
    --policy-arn ${policy-arn}
```

Python

1. 次のコードを実行して、という名前の IAM CreateRoleロールを作成するリクエストを行い、*MyCustomizationRole*という名前の S3 CreatePolicyデータアクセスポリシーを作成するリクエストを行います。*MyFineTuningDataAccess*S3 データアクセスポリシーについては、*$ {training-bucket} # $ {output-bucket} ##### S3 #### ##########*。

```
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. Arnレスポンスではが返されます。AttachRolePolicy次のコードスニペットを実行してリクエストを行い、*$ {policy-arn}* を返されたものに置き換えます。Arn

```
iam.attach_role_policy(
    RoleName="MyCustomizationRole",
    PolicyArn="${policy-arn}"
)
```

3. 言語を選択すると、モデルカスタマイズ API オペレーションを呼び出すコードサンプルが表示されます。

CLI

まず、*FineTuningData.json* という名前のテキストファイルを作成します。下の JSON コードをテキストファイルにコピーし、*$ {training-bucket} # $ {output-bucket} # S3 # ##########*。

```
{
    "trainingDataConfig": {
        "s3Uri": "s3://${training-bucket}/train.jsonl"
    },
    "outputDataConfig": {
```

```
        "s3Uri": "s3://${output-bucket}"
    }
}
```

モデルカスタマイズジョブを送信するには、*FineTuningData#####.json* を含むフォルダーに移動し、コマンドラインで次のコマンドを実行します。*$ {your-customization-role-arn}* は、設定したモデルカスタマイズロールに置き換えます。

```
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
```

レスポンスは *jobArn* を返します。ジョブが完了するまでしばらくお待ちください。以下のコマンドでステータスを確認できます。

```
aws bedrock get-model-customization-job \
    --job-identifier "jobArn"
```

statusがの場合COMPLETE、trainingMetricsレスポンスにが表示されます。次のコマンドを実行して、アーティファクトを現在のフォルダーにダウンロードできます。*aet.et-bucket #########*、*jobId ########### ID* (の最後のスラッシュに続くシーケンス) に置き換えます。jobArn

```
aws s3 cp s3://${output-bucket}/model-customization-job-jobId . --recursive
```

以下のコマンドを使用して、カスタムモデル用のコミットメントなしのプロビジョンドスループットを購入します。

> **Note**
>
> この購入には時間単位で課金されます。コンソールを使用して、さまざまなオプションの見積もり価格を確認してください。

```
aws bedrock create-provisioned-model-throughput \
    --model-id MyCustomModel \
    --provisioned-model-name MyProvisionedCustomModel \
    --model-units 1
```

レスポンスはを返しますprovisionedModelArn。プロビジョニングされたスループットが作成されるまでしばらくお待ちください。ステータスを確認するには、次のコマンドのようにプロビジョニングされたモデルの名前または ARN を指定します。provisioned-model-id

```
aws bedrock get-provisioned-model-throughput \
    --provisioned-model-id ${provisioned-model-arn}
```

statusその場合はInService、以下のコマンドを使用してカスタムモデルで推論を実行できます。プロビジョニングされたモデルの ARN をとして指定する必要があります。model-id出力は現在のフォルダーの *output.txt* という名前のファイルに書き込まれます。

```
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

次のコードスニペットを実行して微調整ジョブを送信します。*$ {your-customization-role-arn}* はセットアップしたの ARN に置き換え、*$ {training-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')
```

レスポンスは *jobArn* を返します。ジョブが完了するまでしばらくお待ちください。以下のコマンドでステータスを確認できます。

```
bedrock.get_model_customization_job(jobIdentifier=jobArn).get('status')
```

statusがの場合COMPLETE、trainingMetricsGetModelCustomizationJobレスポンスにが表示されます。「オブジェクトのダウンロード」の手順に従ってメトリクスをダウンロードすることもできます。

以下のコマンドを使用して、カスタムモデル用のコミットメントなしのプロビジョンドスループットを購入します。

```
response_pt = bedrock.create_provisioned_model_throughput(
```

```
    modelId="MyCustomModel",
    provisionedModelName="MyProvisionedCustomModel"
    modelUnits="1"
)

provisionedModelArn = response_pt.get('provisionedModelArn')
```

レスポンスはを返します。provisionedModelArnプロビジョニングされたスループットが作成されるまでしばらくお待ちください。ステータスを確認するには、次のコマンドのようにプロビジョニングされたモデルの名前または ARN を指定します。provisionedModelId

```
bedrock.get_provisioned_model_throughput(provisionedModelId=provisionedModelArn)
```

statusその場合はInService、以下のコマンドを使用してカスタムモデルで推論を実行できます。プロビジョニングされたモデルの ARN をとして指定する必要があります。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 occured: " +
              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()
```

モデルカスタマイズに関するガイドライン

モデルをカスタマイズする理想的なパラメータは、データセットと、モデルが対象とするタスクによって異なります。値をいろいろ試して、どのパラメータがお客様自身のケースで最も適切に機能するかを確認する必要があります。参考までに、モデル評価ジョブを実行してモデルを評価してください。詳細については、「モデル評価」を参照してください。

このトピックでは、Amazon Titan Text G1 - Express モデルをカスタマイズするための基準となるガイドラインと推奨値を示します。他のモデルについては、プロバイダーのドキュメントを確認してください。

ファインチューニングジョブの送信時に生成される出力ファイルに含まれるトレーニングと検証のメトリクスを使用して、パラメータを調整します。出力を書き込んだ Amazon S3 バケットでこれらのファイルを検索するか、GetCustomModelオペレーションを使用してください。

Amazon Titan Text G1 - Express

TitanText Express text-to-text モデルモデルに関するガイドラインは以下のとおりです。設定できるハイパーパラメータの詳細については、「Amazon Titan テキストモデルカスタマイズハイパーパラメータ」を参照してください。

他のタスクタイプへの影響

一般に、トレーニングデータセットが大きいほど、当該タスクのパフォーマンスが向上します。ただし、特定のタスクにトレーニングを行うと、特にサンプルを多数使用する場合など、他のタスクでモデルのパフォーマンスが低下する可能性があります。例えば、要約タスクのトレーニングデータセットに 100,000 個のサンプルが含まれている場合、モデルのパフォーマンスが分類タスクで低下する可能性があります。

モデルサイズ

一般に、与えられているトレーニングデータが少ない場合でも、大規模なモデルほど、タスクのパフォーマンスは向上します。

このモデルを分類タスクに使用する場合、特にクラス数が比較的少ないとき (100 サンプル未満)、数ショットでの微調整 (100 サンプル未満) で得られる利益は比較的小さくなる可能性があります。

エポック

設定するエポック数を決定するには、以下のメトリクスを使用することをお勧めします。

1. 検証出力の精度 - 精度が高くなるエポック数を設定します。
2. トレーニングおよび検証ロス - トレーニングおよび検証ロスが安定するまでのエポック数を設定します。これはモデルが収束するタイミングに対応しています。トレーニングロス値は、`step_wise_training_metrics.csv` および `validation_metrics.csv` ファイル内で確認してください。

バッチサイズ

バッチサイズを変更する場合は、次の式を使用して学習率を変更することをお勧めします。

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

学習率

一般に、大規模モデルには低い学習率を使用してください。1.00E-06 から 1.00E-05 の範囲に含まれる学習率を使用することをお勧めします。

次の表は、微調整の推奨学習率値を示しています。

タスク	最低学習率	デフォルトの学習率	最高学習率
要約	1.00E-06	3.00E-06	5.00E-05
分類	5.00E-06	5.00E-05	5.00E-05
質問応答	5.00E-06	5.00E-06	5.00E-05

学習率のウォームアップステップ

デフォルト値の 5 を推奨します。

トラブルシューティング

このセクションでは、発生する可能性のあるエラーと、発生した場合に確認すべき点をまとめています。

アクセス許可の問題

Amazon S3 バケットにアクセスするためのアクセス許可で問題が発生した場合は、以下のとおりになっているかどうかをチェックします。

1. Amazon S3 バケットがサーバー側の暗号化に CM-KMS キーを使用している場合は、Amazon Bedrock に渡される IAM `kms:Decrypt` ロールにキーに対するアクセス権限があることを確認してください。 AWS KMS たとえば、「特定のアカウントの任意のキーを使用して暗号化と復号化をユーザーに許可する」を参照してください。 AWS KMS AWS
2. Amazon S3 バケットが Amazon Bedrock モデルのカスタマイズジョブと同じリージョンにある。
3. IAM ロールの信頼ポリシーにサービス SP (`bedrock.amazonaws.com`) が含まれている。

以下のメッセージは、Amazon S3 バケット内のトレーニングデータまたは検証データにアクセスするためのアクセス許可に問題があることを示しています。

```
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
```

上記のエラーのいずれかが発生した場合は、サービスに渡された IAM ロールに、トレーニングと検証のデータセットを格納した Amazon S3 の URI への `s3:GetObject` および `s3:ListBucket` アクセス許可が付与されているかどうかをチェックします。

次のメッセージは、Amazon S3 バケットに出力データを書き込むためのアクセス許可に問題があることを示しています。

```
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
```

上記のエラーのいずれかが発生した場合は、サービスに渡された IAM ロールに、出力データを格納した Amazon S3 の URI への `s3:PutObject` アクセス許可が付与されているかどうかをチェックします。

データの問題

次のエラーは、トレーニング、検証、または出力データファイルの問題に関連しています。

無効なファイル形式

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

上記のエラーが発生した場合は、以下のとおりになっているかどうかをチェックします。

1. すべての行が JSON 形式である。
2. どの JSON にも *input* と *output* の 2 つのキーがあり、どちらのキーも文字列型です。例:

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

```
}
```

3. 改行や空行がない。

Character quota exceeded

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

上記のテキストで始まるエラーが発生した場合は、文字数がモデルカスタマイズのクォータに記載されている文字数クォータの範囲内に収まるようにします。

Token count exceeded

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

前の例のようなエラーが発生した場合は、トークンの数が モデルカスタマイズのクォータ のトークンのクォータに適合していることを確認してください。

内部エラー

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

上記のエラーが発生した場合は、サービスに問題がある可能性があります。ジョブを再試行してください。問題が解決しない場合は、お問い合わせください。 AWS Support

Amazon Bedrock のプロビジョニングされたスループット

スループットとは、モデルが処理して返す入力と出力の数と速度を指します。プロビジョンドスループットを購入すると、固定コストでモデルのスループットを高めることができます。モデルをカスタマイズした場合、そのモデルを使用するにはプロビジョンドスループットを購入する必要があります。

購入したプロビジョンドスループットに対して時間単位で請求されます。料金の詳細については、「Amazon Bedrock 料金表」を参照してください。1 時間あたりの料金は以下の要因によって異なります。

1. 選択するモデル (カスタムモデルの場合、価格はカスタマイズ元の基本モデルと同じです)。
2. プロビジョニングされたスループットに指定するモデルユニット (MU) の数。MU は指定されたモデルに特定のスループットレベルを提供します。MU のスループットレベルは以下を指定します。
   - MU が 1 分以内にすべてのリクエストで処理できる入力トークンの数。
   - MU が 1 分以内にすべてのリクエストにわたって生成できる出力トークンの数。

   > **Note**
   >
   > MU の指定内容について詳しくは、マネージャーにお問い合わせください。 AWS アカウント
3. プロビジョニングされたスループットを維持することを約束する期間。契約期間が長くなるほど、時間当たりの料金はより割引されます。以下のレベルのコミットメントから選択できます。
   - コミットメントなし — プロビジョニングされたスループットを削除すると請求は終了します。
   - 1 か月 — 請求は 1 か月後に終了します。コミットメント期間が終了するまで、プロビジョニングされたスループットは削除できません。
   - 6 か月 — 請求は 6 か月後に終了します。コミットメント期間が終了するまで、プロビジョニングされたスループットは削除できません。

以下の手順は、プロビジョンドスループットをセットアップして使用するプロセスの概要を示しています。

1. プロビジョンドスループット用に購入する MU の数と、プロビジョンドスループットの使用に専念する時間を決定します。

2. 基本モデルまたはカスタムモデル用のプロビジョンドスループットを購入します。
3. プロビジョニングされたモデルが作成されたら、そのモデルを使用してモデル推論を実行できます。

トピック

プロビジョニングされたスループットでサポートされているリージョンとモデル

プロビジョンドスループットは以下のリージョンでサポートされています。

リージョン		
米国東部 (バージニア北部)		
米国西部 (オレゴン)		
AWS GovCloud (米国西部) (コミットメントのないカスタムモデルのみ)		

Amazon Bedrock API を通じてプロビジョンドスループットを購入する場合、モデル ID には Amazon Bedrock FM のコンテキストバリアントを指定する必要があります。次の表は、プロビジョンドスループットを購入できるモデル、基本モデルのコミットメントなしで購入できるかどうか、およびプロビジョンドスループットの購入時に使用するモデル ID を示しています。

モデル名	基本モデルではコミットメントなしの購入がサポートされています	プロビジョニングされたスループットのモデル ID
Amazon Titan Text G1 - Express	Yes	アマゾン。 titan-text-express-v1:0:8 キロ
Amazon Titan Text G1 - Lite	Yes	アマゾン。 titan-text-lite-v1:0:4 キロ
Amazon Titan Embeddings G1 - Text	Yes	アマゾン。 titan-embed-text-v1:2:8 キロ
Amazon Titan Multimodal Embeddings G1	Yes	アマゾン。 titan-embed-image-v1:0
Amazon Titan Image Generator G1	No	アマゾン。 titan-image-generator-v1:0
AnthropicClaudev2 (18K)	Yes	anthropic.claude-v2:0:18k
AnthropicClaudev2 100K	Yes	anthropic.claude-v2:0:100k
AnthropicClaude2.1 (18K)	Yes	anthropic.claude-v2:1:18k
AnthropicClaude2.1 200K	Yes	anthropic.claude-v 2:1:200 k
AnthropicClaude 3 Sonnet28K	Yes	anthropic.claude-3-sonnet-20240229-v 1:0:28 k
AnthropicClaude 3 Sonnet200K	Yes	anthropic.claude-3-sonnet-20240229-v 1:0:200 k
AnthropicClaude 3 Haiku48K	Yes	anthropic.claude-3-haiku-20240307-v 1:0:48 k
AnthropicClaude 3 Haiku200K	Yes	anthropic.claude-3-haiku-20240307-v 1:0:200 k

モデル名	基本モデルではコミットメントなしの購入がサポートされています	プロビジョニングされたスループットのモデル ID
AnthropicClaude Instantv1 100K	Yes	人為的。 claude-instant-v1:2:100 メートル
Cohere Command	Yes	コヒーレ。 command-text-v14:7:4 キロ
Cohere Command Light	Yes	コヒーレ。 command-light-text-v14:7:4 キロ
CohereEmbed英語	Yes	コヒーレ。 embed-english-v3:0:512
CohereEmbed多言語	Yes	コヒーレ。 embed-multilingual-v3:0:512
Stable Diffusion XL 1.0	No	安定性。 stable-diffusion-xl-v1:0
MetaLlama 2 Chat13B	No	meta.llama2-13 1:0:4 k b-chat-v
MetaLlama 213B	No	(下記の注記を参照)
MetaLlama 270B	No	(下記の注記を参照)

> **Note**
>
> MetaLlama 2(チャット以外の) モデルは、カスタマイズしてプロビジョンドスループットを購入した後にのみ使用できます。

前提条件

プロビジョンドスループットを購入して管理するには、以下の前提条件を満たす必要があります。

1. プロビジョンドスループットの購入対象となる 1 つまたは複数のモデルへのアクセスをリクエストします。アクセスが許可されると、基本モデルとそれに基づいてカスタマイズされたすべてのモデルのプロビジョンドスループットを購入できます。
2. IAM ロールに、プロビジョンドスループットに関連するアクションを実行するために必要な権限があることを確認してください。
3. AWS KMS 顧客管理のキーで暗号化されたカスタムモデル用にプロビジョンドスループットを購入する場合、IAM ロールにはキーを復号化する権限が必要です。このテンプレートはで使用できます。キーポリシーを作成し、カスタマー管理キーに添付します。最小限の権限については、「############」ポリシーステートメントのみを使用できます。

Amazon Bedrock モデル用のプロビジョニングされたスループットを購入する

モデルのプロビジョンドスループットを購入するときは、そのモデルのコミットメントレベルと割り当てるモデルユニット (MU) の数を指定します。MU クォータについては、を参照してください。プロビジョンドスループットのクォータプロビジョンドスループットに割り当てることができる MU の数は、プロビジョンドスループットのコミットメント期間によって異なります。

- デフォルトでは、アカウントには 2 MU がプロビジョニングスループット間で配分され、コミットメントなしで割り当てられます。
- コミットメント付きのプロビジョンドスループットを購入する場合は、AWS まずサポートセンターにアクセスして、アカウントの MU をコミットメント付きのプロビジョンドスループット間で分配するようにリクエストする必要があります。リクエストが承認されたら、コミットメント付きプロビジョンドスループットを購入できます。

> **Note**
>
> プロビジョンドスループットを購入した後は、カスタムモデルを選択した場合にのみ関連モデルを変更できます。関連モデルを次のいずれかに変更できます。
>
> - カスタマイズ元のベースモデル。
> - 同じベースモデルから派生した別のカスタムモデル。

あるモデルのプロビジョンドスループットを購入する方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインの [評価とデプロイ] で [プロビジョニングされたスループット] を選択します。
3. 「プロビジョニングされたスループット」セクションで、「プロビジョニングされたスループットを購入」を選択します。
4. 「プロビジョニングされたスループットの詳細」セクションでは、次の操作を行います。
   a. 「プロビジョニングされたスループットの名前」フィールドに、プロビジョニングされたスループットの名前を入力します。
   b. 「Select model」で、基本モデルプロバイダーまたはカスタムモデルカテゴリを選択します。次に、スループットをプロビジョニングするモデルを選択します。

      > **Note**
      >
      > コミットメントなしでプロビジョンドスループットを購入できる基本モデルについては、を参照してくださいプロビジョニングされたスループットでサポートされているリージョンとモデル。
      > AWS GovCloud (US) このリージョンでは、コミットメントのないカスタムモデルのプロビジョンドスループットのみを購入できます。

   c. (オプション) タグをプロビジョンドスループットに関連付けるには、「タグ」セクションを展開して「Add new tag」を選択します。詳細については、「リソースのタグ付け」を参照してください。
5. 「コミットメント期間とモデル単位」セクションでは、次の操作を行います。
   a. 「コミットメント期間の選択」セクションで、プロビジョンドスループットの使用を確定したい期間を選択します。
   b. 「モデル単位」フィールドに、希望するモデル単位 (MU) の数を入力します。コミットメント付きのモデルをプロビジョニングする場合は、AWS まずサポートセンターにアクセスして、購入できる MU 数の増加をリクエストする必要があります。
6. 見積もられた購入の概要] で、推定コストを確認します。

7. [プロビジョンドスループットを購入] を選択します。
8. 表示される注記を確認し、チェックボックスを選択してコミットメント期間と価格を確認します。次に [購入を確認] を選択します。
9. コンソールには、プロビジョニングされたスループットの概要ページが表示されます。「プロビジョンドスループット」テーブルのプロビジョニングされたスループットのステータスは「作成中」になります。プロビジョンドスループットの作成が完了すると、ステータスは In service になります。更新に失敗すると、「ステータス」は「失敗」 になります。

API

プロビジョンドスループットを購入するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (CreateProvisionedModelThroughputリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。

> **Note**
>
> コミットメントなしでプロビジョンドスループットを購入できる基本モデルについては、を参照してください。プロビジョニングされたスループットでサポートされているリージョンとモデル
> AWS GovCloud (US) このリージョンでは、コミットメントのないカスタムモデルのプロビジョンドスループットのみを購入できます。

以下の表では、パラメーターとリクエスト本文を簡単に説明しています (詳細とリクエスト構造については、CreateProvisionedModelThroughput リクエスト構文をご覧ください)。

変数	必須?	ユースケース
modelId	Yes	プロビジョンドスループットを購入するための基本モデル ID または ARN、またはカスタムモデル名または ARN を指定するには
Model/Units	Yes	購入するモデルユニット (MU) の数を指定します。購入できる MU の数を増やすに

変数	必須?	ユースケース
		は、AWS サポートセンターに連絡して、購入できる MU の数を増やすようリクエストしてください。
provisionedModelName	Yes	プロビジョニングされたスループットの名前を指定するには
コミットメント/期間	No	プロビジョニングされたスループットにコミットする期間を指定します。コミットメントなしの価格設定を選択するには、このフィールドを省略してください。
タグ	No	タグをプロビジョニングされたスループットに関連付けるには
clientRequestToken	No	リクエストが重複するのを防ぐには

`provisionedModelArn`レスポンスから返されるのは、`modelId`モデル内の推論として使用できます。プロビジョンドスループットが使用可能になったことを確認するには、GetProvisionedModelThroughputリクエストを送信してステータスがであることを確認します。`InService`更新に失敗すると、ステータスはになり`Failed`、GetProvisionedModelThroughput`failureMessage`応答にはが含まれます。

コード例を参照してください。

プロビジョニングされたスループットの管理

プロビジョンドスループットを購入すると、その詳細を表示、更新、または削除できます。

トピック

- プロビジョンドスループットに関する情報を表示します。
- プロビジョンドスループットを編集
- プロビジョンドスループットを削除する

プロビジョンドスループットに関する情報を表示します。

購入したプロビジョンドスループットに関する情報を表示する方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

プロビジョンドスループットに関する情報を表示するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインの [評価とデプロイ] で [プロビジョニングされたスループット] を選択します。
3. 「プロビジョニングされたスループット」セクションから、「プロビジョニングされたスループット」を選択します。
4. プロビジョニングされたスループットの詳細は「プロビジョニングされたスループットの概要」セクションに表示され、「タグ」セクションにはプロビジョニングされたスループットに関連するタグが表示されます。

API

特定のプロビジョニングされたスループットに関する情報を取得するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (GetProvisionedModelThroughputリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。プロビジョニングされたスループットの名前またはその ARN のいずれかをとして指定します。`provisionedModelId`

アカウントのすべてのプロビジョニングされたスループットに関する情報を一覧表示するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (ListProvisionedModelThroughputsリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。返される結果の数を制御するには、以下のオプションパラメータを指定できます。

フィールド	簡単な説明
maxResults	レスポンスとして返す結果の最大数。
nextToken	maxResults フィールドに指定した数よりも多くの結果がある場合、nextToken レスポンスは値を返します。次の結果バッチを確認するには、nextToken その値を別のリクエストで送信します。

結果をソートしたりフィルタリングしたりするために指定できるその他のオプションパラメータについては、を参照してくださいGetProvisionedModelThroughput。

エージェントのすべてのタグを一覧表示するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (ListTagsForResourceリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信し、プロビジョニングされたスループットの Amazon リソースネーム (ARN) を含めます。

コード例を参照してください。

プロビジョンドスループットを編集

既存のプロビジョンドスループットの名前またはタグを編集できます。

プロビジョンドスループットが関連付けられているモデルの変更には、以下の制限が適用されます。

- 基本モデルに関連付けられているプロビジョンドスループットのモデルを変更することはできません。
- プロビジョンドスループットがカスタムモデルに関連付けられている場合は、その関連付けをカスタマイズ元の基本モデル、または同じ基本モデルから派生した別のカスタムモデルに変更できます。

プロビジョンドスループットが更新されている間は、エンドカスタマーからの進行中のトラフィックを中断することなく、プロビジョニングされたスループットを使用して推論を実行できます。プロビジョンドスループットが関連付けられているモデルを変更した場合、更新が完全にデプロイされるまで、古いモデルからの出力を受け取ることがあります。

プロビジョンドスループットの編集方法については、選択した方法に対応するタブを選択し、手順に従ってください。

Console

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインの [評価とデプロイ] で [プロビジョニングされたスループット] を選択します。
3. 「プロビジョニングされたスループット」セクションから、「プロビジョニングされたスループット」を選択します。
4. [編集] を選択します。以下のフィールドを編集できます。
   - プロビジョニングされたスループットの名前 — プロビジョニングされたスループットの名前を変更します。
   - モデルの選択 — プロビジョンドスループットがカスタムモデルに関連付けられている場合は、関連するモデルを変更できます。
5. プロビジョニングされたスループットに関連するタグは、タグセクションで編集できます。詳細については、「リソースのタグ付け」を参照してください。
6. 変更を保存するには、[編集を保存] を選択します。
7. コンソールに「プロビジョニングされたスループット」の概要ページが表示されます。「プロビジョンドスループット」テーブルのプロビジョニングされたスループットのステータスは「更新中」になります。プロビジョニングされたスループットの更新が終了すると、ステータスは In service になります。更新に失敗すると、「ステータス」は「失敗」 になります。

API

プロビジョニングされたスループットを編集するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (UpdateProvisionedModelThroughputリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。

以下の表では、パラメータとリクエスト本文を簡単に説明しています (詳細とリクエスト構造については、リクエスト構文を参照してください)。UpdateProvisionedModelThroughput

変数	必須?	ユースケース
provisionedModelId	Yes	更新するプロビジョニングされたスループットの名前または ARN を指定するには
desiredModelId	No	プロビジョンドスループットに関連付ける新しいモデルを指定する (ベースモデルに関連付けられたプロビジョンドスループットでは使用できない)。
desiredProvisionedModel名前:	No	プロビジョニングされたスループットの新しい名前を指定するには

アクションが成功すると、レスポンスは HTTP 200 ステータスレスポンスを返します。プロビジョンドスループットが使用可能になったかどうかを確認するには、GetProvisionedModelThroughputInServiceリクエストを送信してステータスがであることを確認します。ステータスがである間は、プロビジョンドスループットを更新または削除することはできません。Updating更新に失敗すると、ステータスはになりFailed、GetProvisionedModelThroughputfailureMessageレスポンスにはが含まれます。

プロビジョニングされたスループットにタグを追加するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (TagResourceリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信し、プロビジョニングされたスループットの Amazon リソースネーム (ARN) を含めます。リクエスト本文にはフィールドが含まれます。tagsフィールドは、タグごとに指定するキーと値のペアを含むオブジェクトです。

プロビジョニングされたスループットからタグを削除するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (UntagResourceリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信し、プロビジョニングされたスループットの Amazon リソースネーム (ARN) を含めます。tagKeysリクエストパラメータは、削除するタグのキーを含むリストです。

コード例を参照してください。

プロビジョンドスループットを削除する

プロビジョンドスループットを削除する方法については、選択した方法に対応するタブを選択し、手順に従ってください。

> **Note**
>
> コミットメントしているプロビジョンドスループットは、コミットメント期間が終了する前に削除することはできません。

Console

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインの [評価とデプロイ] で [プロビジョニングされたスループット] を選択します。
3. 「プロビジョニングされたスループット」セクションから、「プロビジョニングされたスループット」を選択します。
4. [削除] をクリックします。
5. コンソールには、削除が永続的であることを警告するモーダルフォームが表示されます。[確認] を選択して続行します。
6. プロビジョニングされたスループットはただちに削除されます。

API

プロビジョニングされたスループットを削除するには、Amazon Bedrock コントロールプレーンエンドポイントを使用してリクエスト (DeleteProvisionedModelThroughputリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。プロビジョニングされたスループットの名前またはその ARN のいずれかをとして指定します。`provisionedModelId`削除が成功すると、レスポンスは HTTP 200 ステータスコードを返します。

コード例を参照してください。

プロビジョニングされたスループットを使用して推論を実行する

プロビジョンドスループットを購入すると、それをモデル推論に使用してスループットを向上させることができます。必要な場合は、まず Amazon Bedrock コンソールの Playground でプロビジョニングされたスループットをテストできます。プロビジョニングされたスループットをデプロイする準備ができたら、プロビジョニングされたモデルを呼び出すようにアプリケーションを設定します。選択した方法に対応するタブを選択し、手順に従います。

Console

Amazon Bedrock コンソールのプレイグラウンドでプロビジョンドスループットを使用するには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/bedrock/ にある Amazon Bedrock コンソールを開きます。
2. 左側のナビゲーションペインから、ユースケースに応じて [プレイグラウンド] の [チャット]、[テキスト]、または [画像] を選択します。
3. 「モデルを選択」を選択します。
4. 1.「カテゴリ」列で、プロバイダーまたはカスタムモデルカテゴリを選択します。次に、2.「モデル」列で、プロビジョンドスループットが関連付けられているモデルを選択します。
5. 3.「スループット」列で、「プロビジョニングされたスループット」を選択します。
6. [適用] を選択します。

Amazon Bedrock プレイグラウンドの使用方法については、を参照してください。プレイグラウンド

API

プロビジョンドスループットを使用して推論を実行するには、Amazon Bedrock ランタイムエンドポイントを使用して InvokeModelOR リクエスト (InvokeModelWithResponseStreamリクエストとレスポンスの形式とフィールドの詳細についてはリンクを参照) を送信します。`modelId` パラメータとして、プロビジョンドモデルの ARN を指定します。さまざまなモデルのリクエスト本文の要件については、を参照してください。基盤モデルの推論パラメータ

コード例を参照してください。

Amazon Bedrock のプロビジョニングされたスループットのコードサンプル

以下のコード例は、および Python SDK を使用してプロビジョンドスループットを作成、使用、管理する方法を示しています。 AWS CLI

AWS CLI

ターミナルで次のコマンドを実行して、Anthropic Claude v2.1 MyPT モデルからカスタマイズされたカスタムモデルに基づいて、MyCustomModelコミットなしのプロビジョンドスループットコールを作成します。

```
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
```

レスポンスはを返します。provisioned-model-arn作成が完了するまでしばらくお待ちください。ステータスを確認するには、次のコマンドのようにプロビジョニングされたモデルの名前または ARN を指定します。provisioned-model-id

```
aws bedrock get-provisioned-model-throughput \
    --provisioned-model-id MyPT
```

プロビジョンドスループットの名前を変更し、v2.1 からカスタマイズした別のモデルに関連付けます。Anthropic Claude

```
aws bedrock update-provisioned-model-throughput \
    --provisioned-model-id MyPT \
    --desired-provisioned-model-name MyPT2 \
    --desired-model-id arn:aws:bedrock:us-east-1::custom-model/anthropic.claude-
v2:1:200k/MyCustomModel2
```

次のコマンドを使用して、更新したプロビジョニング済みモデルで推論を実行します。UpdateProvisionedModelThroughputレスポンスで返されるプロビジョニング済みモデルの ARN をとして指定する必要があります。model-id出力は、現在のフォルダーの *output.txt* という名前のファイルに書き込まれます。

```
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
```

以下のコマンドを使用して、プロビジョニングされたスループットを削除します。プロビジョニングされたスループットに対して課金されることはなくなります。

```
aws bedrock delete-provisioned-model-throughput
  --provisioned-model-id MyPT2
```

Python (Boto)

以下のコードスニペットを実行して、Anthropic Claude v2.1 MyPT MyCustomModel モデルからカスタマイズされたカスタムモデルに基づいて呼び出されるコミットなしのプロビジョンドスループットを作成します。

```
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'
)
```

レスポンスはを返します。provisionedModelArn作成が完了するまでしばらくお待ちください。次のコードスニペットでステータスを確認できます。CreateProvisionedModelThroughputプロビジョニングされたスループットの名前またはレスポンスから返される ARN のいずれかをとして指定できます。provisionedModelId

```
bedrock.get_provisioned_model_throughput(provisionedModelId='MyPT')
```

プロビジョンドスループットの名前を変更し、v2.1 からカスタマイズした別のモデルに関連付けます。Anthropic Claude次に、GetProvisionedModelThroughputリクエストを送信し、プロビジョニングされたモデルの ARN を推論に使用する変数に保存します。

```
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('provisionedModelA
```

以下のコマンドを使用して、更新したプロビジョニング済みモデルで推論を実行します。プロビジョニングされたモデルの ARN をとして指定する必要があります。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 occured: " +
              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()
```

次のコードスニペットを使用してプロビジョニングされたスループットを削除します。プロビジョニングされたスループットに対して課金されることはなくなります。

```
bedrock.delete_provisioned_model_throughput(provisionedModelId='MyPT2')
```

リソースのタグ付け

Amazon Bedrock リソースを管理しやすくするために、各リソースにメタデータをタグとして割り当てることができます。タグは、 AWS リソースに割り当てるラベルです。各タグは、キーと値から構成されます。

タグを使用すると、 AWS リソースを目的、所有者、アプリケーションなどさまざまな方法で分類できます。タグは、以下のことに役立ちます。

- AWS リソースを特定して整理します。多くの AWS リソースはタグ付けをサポートしているため、異なるサービスのリソースに同じタグを割り当てて、リソースが同じであることを示すことができます。
- コストの割り当て。 AWS Billing and Cost Management ダッシュボードでタグをアクティブ化します。 AWS はタグを使用してコストを分類し、毎月のコスト配分レポートを提供します。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「コスト配分タグを使用する」を参照してください。
- リソースへのアクセス制御。Amazon Bedrock でタグを使用して、Amazon Bedrock リソースへのアクセスを制御するポリシーを作成することができます。これらのポリシーを IAM ロールまたはユーザーにアタッチして、タグベースのアクセスコントロールを有効にできます。

タグ付けできる Amazon Bedrock リソースは以下のとおりです。

- カスタムモデル
- モデルカスタマイズジョブ
- プロビジョンドモデル
- バッチ推論ジョブ (API のみ)
- エージェント
- エージェントエイリアス
- ナレッジベース
- モデル評価 (コンソールのみ)

トピック

- コンソールを使用する
- API を使用する

- ベストプラクティスと制約事項

コンソールを使用する

サポートされているリソースの作成中または編集中に、いつでもタグを追加、変更、削除できます。

API を使用する

タグ付けオペレーションを実行するには、タグ付けオペレーションを実行するリソースの Amazon リソースネーム (ARN) が必要です。タグを追加または管理するリソースに応じて、タグ付けオペレーションのセットが 2 種類あります。

1. 以下のリソースでは、Amazon Bedrock の TagResource、UntagResource、および ListTagsForResourceオペレーションを使用します。
   - カスタムモデル
   - モデルカスタマイズジョブ
   - プロビジョンドモデル
   - バッチ推論ジョブ
2. 以下のリソースでは、 Agents for Amazon Bedrock TagResource、、UntagResourceおよび ListTagsForResourceオペレーションを使用します。
   - エージェント
   - エージェントエイリアス
   - ナレッジベース

リソースにタグを追加するには、Amazon Bedrock TagResourceまたは Agents for Amazon Bedrock TagResourceリクエストを送信します。

リソースのタグを解除するには、 UntagResourceまたは UntagResourceリクエストを送信します。

リソースのタグを一覧表示するには、 ListTagsForResourceまたは ListTagsForResourceリクエストを送信します。

タブを選択すると、インターフェイスまたは言語でのコード例が表示されます。

AWS CLI

エージェントに 2 つのタグを追加します。キーと値のペア同士をスペースで区切ります。

```
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
```

エージェントからタグを削除します。キー同士をスペースで区切ります。

```
aws bedrock-agent untag-resource \
    --resource-arn "arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345" \
    --tag-keys key=department facing
```

エージェントのタグを一覧表示します。

```
aws bedrock-agent list-tags-for-resource \
    --resource-arn "arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345"
```

Python (Boto)

エージェントに 2 つのタグを追加します。

```
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)
```

エージェントからタグを削除します。

```
bedrock.untag_resource(
    resourceArn='arn:aws:bedrock:us-east-1:123456789012:agent/AGENT12345',
```

```
    tagKeys=['department', 'facing']
)
```

エージェントのタグを一覧表示します。

```
bedrock.list_tags_for_resource(resourceArn='arn:aws:bedrock:us-
east-1:123456789012:agent/AGENT12345')
```

ベストプラクティスと制約事項

タグ付けのベストプラクティスと制限については、「AWS リソースのタグ付け」を参照してください。

Amazon Titanモデル

Amazon Titan基盤モデル (FMsは、大規模なデータセット AWS で によって事前トレーニングされた FMs ファミリーであり、さまざまなユースケースをサポートするように構築された強力な汎用モデルです。そのまま使用することも、独自のデータでプライベートにカスタマイズすることもできます。

Amazon は、Amazon Bedrock の以下のモデルTitanをサポートしています。

- Amazon Titan テキスト
- Amazon Titan Embeddings G1 - Text
- Amazon Titan Multimodal Embeddings G1
- Amazon Titan Image Generator G1 (プレビュー)

> **Note**
>
> Amazon Titan Image Generator G1 はパブリックプレビューリリース中です。

トピック

Amazon Titan Text モデル

Amazon Titanテキストモデルには、Amazon Titan Text G1 - Expressと Amazon が含まれますTitan Text G1 - Lite。

Amazon Titan Text G1 - Express

Amazon Titan Text G1 - Expressは、テキスト生成用の大規模言語モデルです。検索拡張生成 (RAG) をサポートするだけでなく、自由形式のテキスト生成や会話型チャットなど、高度で一般的な言語タ

スクにも幅広く対応します。このモデルは、リリース時点では英語向けに最適化されていますが、さらに 100 を超える言語がサポートされたプレビュー版もご利用いただけます。

- モデル ID – `amazon.titan-text-express-v1`
- 最大トークン – 8K
- 言語 – 英語 (一般提供)、その他 100 言語 (プレビュー)
- サポートされているユースケース – 検索拡張生成、自由形式のテキスト生成、ブレインストーミング、要約、コード生成、表作成、データフォーマット、パラフレーズ、思考の連鎖、書き換え、抽出、Q&A、チャット。

Amazon Titan Text G1 - Lite

Amazon Titan Text G1 - Liteは、要約やコピーの書き込みなど、英語のタスクを微調整するのに最適な軽量の効率的なモデルです。お客様は、高度にカスタマイズ可能な、より小さく、費用対効果の高いモデルが必要です。

- モデル ID – `amazon.titan-text-lite-v1`
- 最大トークン – 4K
- 言語 – 英語
- サポートされているユースケース – 自由形式のテキスト生成、ブレインストーミング、要約、コード生成、表作成、データフォーマット、パラフレーズ、思考の連鎖、書き換え、抽出、Q&A、チャット。

Amazon Titan テキストモデルのカスタマイズ

Amazon Titanテキストモデルのカスタマイズの詳細については、以下のページを参照してください。

- データセットを準備する
- Amazon Titan テキストモデルカスタマイズハイパーパラメータ

Amazon Titan Text Prompt エンジニアリングガイドライン

Amazon Titanテキストモデルは、さまざまなユースケースでさまざまなアプリケーションで使用できます。Amazon Titan Text モデルには、次のようなアプリケーションに対するプロンプトエンジニアリングガイドラインがあります。

- Chatbot
- Text2SQL
- 関数呼び出し
- 検索拡張生成 (RAG)

Amazon Titanテキストプロンプトエンジニアリングガイドラインの詳細については、「Amazon Titan テキストプロンプトエンジニアリングガイドライン」を参照してください。

一般的なプロンプトエンジニアリングガイドラインについては、「プロンプトエンジニアリングガイドライン」を参照してください。

AWS AI サービスカード - Amazon Titan テキスト

AI サービスカードは透明性を提供し、 AWS AI サービスの意図したユースケースと公平性に関する考慮事項を文書化します。一連の AI サービスの想定されるユースケース、責任ある AI 設計の選択、ベストプラクティス、パフォーマンスに関する情報を、AI サービスカードでまとめて確認することができます。

Amazon Titan Embeddings G1 - Textモデル

Amazon Titan Embeddings テキストモデルには Amazon Titan Embeddings G1 - Text G1 が含まれます。

テキスト埋め込みは、ドキュメント、段落、文などの非構造化テキストの意味があるベクトル表現を表します。テキストの本文を入力すると、出力は (1 x n) のベクトルになります。埋め込みベクトルは、さまざまなアプリケーションで使用できます。

Amazon Titan Embeddings G1 - Textモデル (`amazon.titan-embed-text-v1`) 。Amazon Titan Embeddings G1 - Text- テキスト v1.2 は最大 8k トークンを取り込み、1,536 次元のベクトルを出力できます。また、このモデルは 25 種類以上の言語で機能します。このモデルはテキスト取得タスク用に最適化されていますが、セマンティック類似度やクラスタリングなどの追加タスクを実行することもできます。Amazon Titan Embeddings G1 - Text – Text v1.2 は長いドキュメントもサポートして

いますが、取得タスクの場合は、ドキュメントを論理セグメント (段落やセクションなど) にセグメント化することをお勧めします。これは当社の推奨事項に沿ったものです。

> **Note**
>
> Titan Embeddings G1 - Text モデルは、 `maxTokenCount`や などの推論パラメータをサポートしていませんtopP。

テキストまたは画像埋め込みモデルを使用するには、`model Id` に `amazon.titan-embed-text-v1` または `amazon.titan-embed-image-v1` を指定して `Invoke Model` API オペレーションを使用し、レスポンス内で埋め込みオブジェクトを取得します。

Jupyter Notebook の例を参照するには:

1. Amazon Bedrock コンソール (https://console.aws.amazon.com/bedrock/home) を開きます。
2. 左側のメニューから [ベースモデル] を選択します。
3. 下にスクロールして Amazon Titan Embeddings G1 - Text モデルを選択します。
4. Amazon Titan Embeddings G1 - Text タブ (選択したモデルに応じて) で、ノートブックの例を表示 を選択して、埋め込みのサンプルノートブックを表示します。

マルチモーダルトレーニングのためのデータセットの準備について詳しくは、「データセットを準備する」を参照してください。

Amazon Titan Multimodal Embeddings G1モデル

Amazon Titan Multimodal Embeddings G1 Generation 1 (G1) は、テキスト、イメージ、またはテキストとイメージの組み合わせによるイメージの検索などのユースケース向けのマルチモーダル埋め込みモデルです。高い正解率と迅速なレスポンスを実現するように設計されたこのモデルは、検索やレコメンデーションのユースケースに最適です。

- モデル ID – `amazon.titan-embed-image-v1`
- 入力テキストトークンの最大数 – 128
- 言語 – 英語
- 入力画像の最大サイズ – 5 MB
- 出力ベクトルサイズ – 1,024 (デフォルト)、384、256

- 推論タイプ – オンデマンド、プロビジョンドスループット
- サポートされているユースケース – 画像検索、レコメンデーション、パーソナライズ

埋め込みの長さ

埋め込みの長さのカスタム設定は任意です。埋め込みのデフォルトの長さは 1,024 文字で、ほとんどのユースケースで使うことができます。埋め込みの長さは 256 文字、384 文字、または 1,024 文字に設定できます。埋め込みサイズを大きくすると、より詳細なレスポンスが得られますが、処理時間も長くなります。埋め込みの長さを短くすると詳細度は低くなりますが、応答時間は短くなります。

```
# 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
 }
})
```

ファインチューニング

- Amazon Titan Multimodal Embeddings G1の微調整への入力は、画像とテキストのペアです。
- 画像フォーマット: PNG、JPEG
- 入力画像サイズの上限: 5 MB
- 画像の寸法: 最小: 128 ピクセル、最大: 4,096 ピクセル
- キャプション内のトークンの最大数: 128
- トレーニングデータセットのサイズ範囲: 1,000 ~ 500,000
- 検証データセットのサイズ範囲: 8 ~ 50,000
- キャプションの長さ (文字数): 0 ~ 2,560

- 画像あたりの最大合計ピクセル数: 2,048*2,048*3
- アスペクト比 (幅/高さ): 最小: 0.25、最大: 4

データセットの準備

トレーニングデータセットに対して、複数の JSON 行を含む .jsonl ファイルを作成します。JSON の各行には、Sagemaker 拡張マニフェスト形式と似た image-ref 属性と caption 属性の両方が含まれています。検証データセットが必要です。自動キャプションは現在サポートされていません。

```
{"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"}
```

トレーニングデータセットと検証データセットの両方に対して、複数の JSON 行を含む .jsonl ファイルを作成することになります。

Amazon S3 パスは、Amazon Bedrock サービスロールに IAM ポリシーをアタッチすることで Amazon Bedrock がデータにアクセスできるようにアクセス許可を付与したフォルダ内にある必要があります。トレーニングデータの IAM ポリシーの付与について詳しくは、「カスタムジョブにトレーニングデータへのアクセスを付与する」を参照してください。

ハイパーパラメータ

これらの値は Multimodal Embeddings モデルのハイパーパラメータに合わせて調整できます。デフォルト値は、ほとんどのユースケースで十分に機能します。

- 学習率 - (最小/最大学習率) – デフォルト: 5.00E-05、最小: 5.00E-08、最大: 1
- バッチサイズ - 有効バッチサイズ - デフォルト: 576、最小: 256、最大: 9,216
- 最大エポック数 – デフォルト:「自動」、最小: 1、最大: 100

Amazon Titan Image Generator G1モデル

Amazon Titan Image Generator G1は画像生成モデルです。テキストから画像を生成することや、ユーザーが既存の画像をアップロードして編集することが可能です。ユーザーは、テキストプロンプトを使って (マスクなしで) 画像を編集したり、画像マスクを使って画像の一部を編集したりできま

す。アウトペインティングで画像の境界を拡張し、インペインティングで画像を塗りつぶすことができます。また、オプションのテキストプロンプトに基づいて画像のバリエーションを生成することもできます。Amazon Titan Image Generator G1 Generator には、出力ファイルにウォーターマークが含まれています。

> **Note**
>
> Amazon Titan Image Generator G1 は現在プレビューリリース中です。本番環境での作業にはお勧めしません。一部の機能は期待どおりに動作しない場合があります。イメージと出力データは、正確な結果よりも少ない結果を生成する可能性があります。

Amazon Titan Image Generator G1プロンプトエンジニアリングガイドラインの詳細については、「Amazon Titan Image Generator G1 Prompt Engineering Best Practices」を参照してください。

- モデル ID – `amazon.titan-image-generator-v1`
- 最大入力文字数 – 1,024 文字
- 最大入力画像サイズ – 50 MB (一部の特定の解像度のみ対応)
- イン/アウトペインティングを使用する場合の最大画像サイズ – 1,024 x 1,024 ピクセル
- 画像バリエーションを使用する場合の最大画像サイズ – 4,096 x 4,096 ピクセル
- 言語 – 英語
- 出力タイプ – 画像
- サポートされている画像タイプ – JPEG、JPG、PNG
- 推論タイプ – オンデマンド、プロビジョンドスループット
- サポートされているユースケース – 画像生成、画像編集、画像バリエーション

機能

- T ext-to-image (T2I) 生成 — テキストプロンプトを入力し、出力として新しいイメージを生成します。生成された画像には、テキストプロンプトで説明されている概念が取り込まれています。
- T2I モデルのファインチューニング – 複数の画像をインポートして独自のスタイルやパーソナライゼーションを取り込み、核となる T2I モデルをファインチューニングします。ファインチューニングされたモデルでは、特定のユーザーのスタイルやパーソナライゼーションに合わせた画像が生成されます。

- 画像編集オプション – インペインティング、アウトペインティング、バリエーションの生成、画像マスクなしの自動編集などがあります。
- インペインティング – 画像とセグメンテーションマスクを入力 (ユーザーからの入力またはモデルによって推定された入力) として使用し、マスク内の領域を再構成します。インペインティングを使用して、マスクされている要素を削除し、背景ピクセルに置き換えることができます。
- アウトペインティング – 画像とセグメンテーションマスクを入力 (ユーザーからの入力またはモデルによって推定された入力) として使用し、領域をシームレスに拡張する新しいピクセルを生成します。正確なアウトペインティングを使用すると、画像を境界まで拡張するときにマスクされた画像のピクセルが維持されます。デフォルトのアウトペイントを使用すると、セグメンテーション設定に基づいてマスクされた画像のピクセルが画像の境界まで拡張されます。
- 画像バリエーション – 画像とオプションのプロンプトを入力として使用します。入力画像の内容はそのままに、スタイルや背景を変えて新しい画像を生成します。

> **Note**
>
> 微調整されたモデルを使用している場合、API またはモデルのインペインティングまたはアウトペインティング機能は使用できません。

パラメータ

Amazon Titan Image Generator G1推論パラメータの詳細については、「Amazon Titan Image Generator G1 推論パラメータ」を参照してください。

ファインチューニング

Amazon Titan Image Generator G1モデルの微調整の詳細については、以下のページを参照してください。

- データセットを準備する
- Amazon Titan Image Generator G1モデルカスタマイズハイパーパラメータ

Titan Image Generator G1 の微調整と料金

このモデルでは、次の式例を使用して、ジョブあたりの総価格を計算します。

合計料金 = ステップ * バッチサイズ * 表示されたイメージあたりの料金

最小値 (自動):

- 最小ステップ (自動) - 500
- 最小バッチサイズ - 8
- デフォルトの学習レート - 0.0001
- 表示されたイメージあたりの料金 - 0.005

ハイパーパラメータ設定の微調整

ステップ - モデルが各バッチに公開される回数。デフォルトのステップカウントは設定されていません。10～40,000 の数値、または「Auto」の文字列値を選択する必要があります。

ステップ設定 - 自動 - Amazon Bedrock は、トレーニング情報に基づいて妥当な値を決定します。このオプションを選択すると、トレーニングコストよりもモデルのパフォーマンスが優先されます。ステップの数は自動的に決定されます。この数は通常、データセットに基づいて 1,000～8,000 です。ジョブのコストは、モデルをデータに公開するために使用されるステップの数によって影響を受けます。ジョブのコストの計算方法については、料金詳細の料金例のセクションを参照してください。(Auto が選択されているときのステップ数とイメージ数との関連については、上記の例の表を参照してください。)

ステップ設定 - カスタム - Bedrock でカスタムモデルをトレーニングデータに公開するステップ数を入力できます。この値は 10～40,000 です。低いステップカウント値を使用することで、モデルが生成するイメージごとのコストを削減できます。

バッチサイズ – モデルパラメータが更新される前に処理されたサンプルの数。この値は 8～192 で、8 の倍数です。

学習レート — トレーニングデータの各バッチ後にモデルパラメータが更新されるレート。これは 0～1 の浮動小数点値です。学習レートはデフォルトで 0.0001 に設定されています。

微調整手順の詳細については、「モデルカスタマイズジョブの送信」を参照してください。

出力

Titan Image Generator G1 は、出力イメージのサイズと品質を使用して、イメージの価格を決定します。 Titan Image Generator G1には、サイズに基づいて 2 つの料金セグメントがあります。1 つは 512 x 512 個のイメージ用、もう 1 つは 1024 x 1024 個のイメージ用です。料金は、画像サイズ (高さ x 幅) が 512 x 512 以下か 512 x 512 より大きいかによって決まります。

Amazon Bedrock の料金の詳細については、「Amazon Bedrock の料金」を参照してください。

ウォーターマーク検出

> **Note**
>
> 透かし検出はパブリックプレビューリリースであり、 から生成された透かしのみを検出しますTitan Image Generator G1。この機能は現在、PDX および IAD リージョンでのみ使用できます。透かし検出は、 によって生成された透かしを非常に正確に検出しますTitan Image Generator G1。元のイメージから変更されたイメージは、精度の低い検出結果を生成する可能性があります。

このモデルでは、生成されたすべてのイメージに不可視ウォーターマークを追加して、誤った情報の拡散を減らし、著作権保護を支援し、コンテンツの使用状況を追跡します。将来的には、ウォーターマーク検出 API を使用して、イメージがTitan Image Generator G1モデルによって生成されたかどうかを確認できます。これにより、このウォーターマークの存在を確認できます。

イメージをアップロードして、イメージに からのウォーターマークTitan Image Generator G1が存在するかどうかを検出できます。コンソールを使用して、次のステップに従って、このモデルから透かしを検出します。

で透かしを検出するにはTitan Image Generator G1 :

1. 「Amazon Bedrock コンソール」をクリックして、 Amazon Bedrock コンソールを開きます。
2. Amazon Bedrock のナビゲーションペインから概要を選択します。構築とテスト タブを選択します。
3. 「保護」セクションで「ウォーターマーク検出」に移動し、「ウォーターマーク検出を表示」を選択します。
4. イメージのアップロードを選択し、JPG または PNG 形式のファイルを見つけます。最大ファイルサイズは 5 MB です。
5. アップロードすると、名前、ファイルサイズ、最終変更日を含む画像のサムネイルが表示されます。X を選択して、アップロードセクションからイメージを削除または置き換えます。
6. 分析を選択して、透かし検出分析を開始します。
7. イメージは結果 でプレビューされ、イメージの下にウォーターマークが検出され、イメージ全体にバナーが検出されたウォーターマークが検出されるかどうかを示します。透かしが検出されない場合、イメージの下のテキストには「透かしが検出されませんでした」と表示されます。

8. 次のイメージをロードするには、アップロードセクションでイメージのサムネイルの X を選択し、分析する新しいイメージを選択します。

プロンプトエンジニアリングガイドライン

マスクプロンプト – このアルゴリズムはピクセルをコンセプトに分類します。マスクする画像の領域をマスクプロンプトの解釈に基づいて分類するためのテキストプロンプトを、ユーザーが入力できます。プロンプトオプションを使うと、より複雑なプロンプトを解釈して、マスクをセグメンテーションアルゴリズムにエンコードできます。

画像マスク – 画像マスクを使用してマスク値を設定することもできます。画像マスクをマスクのプロンプト入力と組み合わせると、正解率が向上します。画像マスクファイルは、以下のパラメータに準拠している必要があります。

- マスク画像の値は 0 (黒) または 255 (白) でなければなりません。値が 0 の画像マスク領域は、ユーザープロンプトによる画像または入力画像で再生成されます。
- `maskImage` フィールドは Base64 でエンコードされた画像文字列である必要があります。
- マスク画像は、入力画像と同じ寸法 (高さと幅が同じ) である必要があります。
- 入力画像とマスク画像には PNG または JPG ファイルのみを使用できます。
- マスク画像には白黒のピクセル値のみを使用する必要があります。
- マスク画像では RGB チャネルのみを使用できます (アルファチャネルはサポートされていません)。

Amazon Titan Image Generator G1プロンプトエンジニアリングの詳細については、「Amazon Titan Image Generator G1 Prompt Engineering Best Practices」を参照してください。

一般的なプロンプトエンジニアリングガイドラインについては、「プロンプトエンジニアリングガイドライン」を参照してください。

Amazon Bedrock のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャから利点を得られます。

セキュリティは、 AWS とユーザー間で共有される責任です。責任共有モデル では、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任を担います AWS クラウド。また、 は、ユーザーが安全に使用できるサービス AWS も提供します。コンプライアンスAWS プログラム コンプライアンスプログラム の一環として、サードパーティーの監査者が定期的にセキュリティの有効性をテストおよび検証。Amazon Bedrock に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムAWS による対象範囲内の のサービスコンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon Bedrock の使用時に責任共有モデルがどのように適用されるかを理解するために役立ちます。以下のトピックでは、セキュリティおよびコンプライアンス上の目的を達成するように Amazon Bedrock を設定する方法について説明します。また、Amazon Bedrock リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

データ保護

AWS 責任共有モデル、Amazon Bedrock でのデータ保護に適用されます。このモデルで説明したように、 AWS は、すべての を実行するグローバルインフラストラクチャを保護する責任を担います AWS クラウド。このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任はユーザーにあります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。こうすると、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、以下の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2、できれば TLS 1.3 が必要です。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションを、 内のすべてのデフォルトのセキュリティコントロールとともに使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや名前フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、 AWS CLI または SDK で Amazon Bedrock または他の AWS のサービス を使用する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

Amazon Bedrock でのデータ保護

Amazon Bedrock は、プロンプトと継続を使用して AWS モデルのトレーニングやサードパーティーへの配布を行いません。

各モデルプロバイダーには、モデルのアップロード先となるエスクローアカウントがあります。Amazon Bedrock 推論アカウントにはこれらのモデルを呼び出すことができるアクセス許可がありますが、エスクローアカウント自体には Amazon Bedrock アカウントへのアウトバウンドアクセス許可はありません。さらに、モデルプロバイダーは Amazon Bedrock のログにアクセスしたり、お客様のプロンプトやその続きの情報にアクセスしたりすることはできません。

Amazon Bedrock はお客様のデータを保存したりサービスログに記録したりすることはありません。

Amazon Bedrock モデルのカスタマイズにおけるデータ保護

トレーニングデータは、ベースTitanモデルのトレーニングやサードパーティへの配布には使用されません。使用状況のタイムスタンプや記録されたアカウント ID など、サービスによって記録されたその他の使用状況データ· 情報も、モデルのトレーニングには使用されません。

Amazon Bedrock は、Amazon Bedrock 基盤モデルの微調整にのみ提供される微調整データを使用します。Amazon Bedrock は、基盤モデルのトレーニングなど、他の目的で微調整データを使用することはありません。

Bedrock は、 CreateModelCustomizationJobアクションまたはコンソール でトレーニングデータを使用して、Amazon Bedrock の基本モデルの微調整されたバージョンであるカスタムモデルを作成します。カスタムモデルは によって管理および保存されます AWS。デフォルトでは、カスタムモデルは AWS が所有する AWS Key Management Service キーで暗号化されますが、独自の AWS KMS キーを使用してカスタムモデルを暗号化できます。カスタムモデルを暗号化するタイミングは、コンソールを使用して微調整ジョブを送信するときか、`CreateModelCustomizationJob` アクションを使用してプログラムで微調整ジョブを送信するときです。

微調整ジョブが完成すると、微調整のために提供したトレーニングデータや検証データは Amazon Bedrock アカウントには保存されなくなります。トレーニング中、データは AWS Service Management Connector インスタンスのメモリに存在しますが、これらのマシンでは、インスタンス自体のハードウェアモジュールに実装されている XTS-AES-256 暗号を使用して暗号化されています。

モデルが機密データに基づいて推論レスポンスを生成する可能性があるため、機密データを使用してカスタムモデルをトレーニングすることはお勧めしません。機密データを使用してカスタムモデルをトレーニングしている場合、そのデータに基づくレスポンスが生成されないようにする唯一の方法は、カスタムモデルを削除し、トレーニングデータセットから機密データを削除して、カスタムモデルを再トレーニングすることです。

カスタムモデルのメタデータ (名前と Amazon リソースネーム) とプロビジョンドモデルのメタデータは、Amazon Bedrock サービスが所有するキーで暗号化された Amazon DynamoDB テーブルに保存されます。

トピック

データ暗号化

Amazon Bedrock は、暗号化を使用して保管中のデータと転送中のデータを保護します。

トピック

転送中の暗号化

内では AWS、転送中のすべてのネットワーク間データが TLS 1.2 暗号化をサポートします。

Amazon Bedrock API とコンソールに対するリクエストには、安全な SSL 接続が使用されます。AWS Identity and Access Management (IAM) ロールを Amazon Bedrock に渡して、トレーニングとデプロイのためにユーザーに代わって リソースにアクセスするためのアクセス許可を提供します。

保管中の暗号化

Amazon Bedrock は保管中の モデルカスタマイズジョブとアーティファクトの暗号化 を提供します。

キー管理

を使用して AWS Key Management Service 、 リソースの暗号化に使用するキーを管理します。詳細については、「AWS Key Management Service の概念」を参照してください。KMS キーを使用して次のリソースを暗号化できます。

- Amazon Bedrock の場合
  - モデルカスタマイズジョブとその出力カスタムモデル — コンソールでのジョブの作成中、または CreateModelCustomizationJob API コールで `customModelKmsKeyId`フィールドを指定することによって。
  - エージェント – コンソールでのエージェントの作成中、または CreateAgent API コールでフィールドを指定することによって。
  - ナレッジベースのデータソース取り込みジョブ — コンソールでのナレッジベースの作成中、または CreateDataSourceまたは UpdateDataSource API コールで `kmsKeyArn`フィールドを指定することによって。
  - Amazon OpenSearch Service のベクトルストア — ベクトルストアの作成中。詳細については、「Amazon OpenSearch Service コレクションの作成、一覧表示、削除」および「Amazon OpenSearch Service の保管中のデータの暗号化」を参照してください。
- Amazon S3 経由 – 詳細については、「 AWS KMS キーによるサーバー側の暗号化 (SSE-KMS) の使用」を参照してください。
  - モデルカスタマイズ用のトレーニング、検証、出力データ
  - ナレッジベースのデータソース
- による AWS Secrets Manager – 詳細については、「 でのシークレットの暗号化と復号 AWS Secrets Manager」を参照してください。
  - サードパーティーモデルのベクトルストア

リソースを暗号化したら、リソースを選択してコンソールで [詳細] を表示するか、次の `Get` API コールを使用して KMS キーの ARN を見つけることができます。

- GetModelCustomizationJob
- GetAgent
- GetIngestionJob

モデルカスタマイズジョブとアーティファクトの暗号化

デフォルトでは、Amazon Bedrock はモデルカスタマイズジョブからの以下のモデルアーティファクトを管理キーで暗号化します。 AWS

- モデルカスタマイズジョブ
- モデルカスタマイズジョブの出力ファイル (トレーニングメトリクスと検証メトリクス)
- 結果のカスタムモデル

オプションで、顧客管理キーを作成してモデルアーティファクトを暗号化できます。詳細については AWS KMS keys、『AWS Key Management Service 開発者ガイド』の「カスタマー管理キー」を参照してください。カスタマー管理キーを使用するには、以下の手順を実行してください。

1. を使用してカスタマー管理キーを作成します AWS Key Management Service。
2. 指定されたロールの権限を含むリソースベースのポリシーをアタッチして、カスタムモデルを作成または使用します。

トピック

- カスタマーマネージド キーを作成する
- キーポリシーを作成し、カスタマー管理キーに添付します。
- トレーニング、検証、出力データの暗号化

カスタマーマネージド キーを作成する

まず、権限があることを確認します。CreateKey次に、「キーの作成」の手順に従って、 AWS KMS コンソールまたは CreateKeyAPI オペレーションでカスタマー管理キーを作成します。必ず対称暗号化キーを作成してください。

キーを作成すると、ArncustomModelKmsKeyIdモデルカスタマイズジョブの送信時にとして使用できるキー用のが返されます。

キーポリシーを作成し、カスタマー管理キーに添付します。

「キーポリシーの作成」の手順に従って、次のリソースベースのポリシーを KMS キーにアタッチします。ポリシーには 2 つのステートメントが含まれています。

1. モデルカスタマイズアーティファクトを暗号化するロールの権限。カスタムモデルビルダーロールの ARN をフィールドに追加します。Principal
2. 推論にカスタムモデルを使用するロールの権限。カスタムモデルのユーザーロールの ARN をフィールドに追加します。Principal

```
{
    "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": "*"
        }
}
```

トレーニング、検証、出力データの暗号化

Amazon Bedrock を使用してモデルカスタマイズジョブを実行する場合、入力 (トレーニング/検証データ) ファイルを Amazon S3 バケットに保存します。ジョブが完了すると、Amazon Bedrock は出力メトリックスファイルをジョブの作成時に指定した S3 バケットに保存し、結果のカスタムモデルアーティファクトはが制御する Amazon S3 バケットに保存します。 AWS

入力ファイルと出力ファイルは、デフォルトで Amazon S3 SSE-S3 サーバー側の暗号化を使用して暗号化されます。AWS マネージドキーこのタイプのキーは、ユーザーに代わって作成、管理、使用されます。 AWS

代わりに、自分で作成、所有、管理する顧客管理鍵を使用してこれらのファイルを暗号化することもできます。カスタマー管理キーとキーポリシーの作成方法については、前のセクションと以下のリンクを参照してください。

- Amazon S3 SSE-S3 サーバー側の暗号化の詳細については、「Amazon S3 マネージドキーによるサーバー側の暗号化 (SSE-S3) の使用」を参照してください。
- S3 オブジェクトを暗号化するためのカスタマーマネージドキーの詳細については、「KMS キーによるサーバー側の暗号化 (SSE-KMS) の使用」を参照してください。 AWS

エージェントリソースの暗号化

Amazon Bedrock はエージェントのセッション情報を暗号化します。デフォルトでは、Amazon Bedrock は AWS マネージドキーを使用してこのデータを暗号化します。オプションで、カスタマーマネージドキーを使用して、エージェントアーティファクトを暗号化することもできます。

の詳細については AWS KMS keys、「 AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

カスタム KMS キーを使用してエージェントとセッションを暗号化する場合は、Amazon Bedrock がユーザーに代わってエージェントリソースを暗号化および復号化できるように、次のアイデンティティベースのポリシーとリソースベースのポリシーを設定する必要があります。

1. InvokeAgent 呼び出しを行うアクセス許可を持つ IAM ロールまたはユーザーに次の ID ベースのポリシーをアタッチします。このポリシーは、InvokeAgent 呼び出しを行うユーザーに KMS アクセス許可があることを確認します。*region*、*account-id*、*agent-id*、および *key-id* を適切な値に置き換えます。

```
{
    "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. 次のリソースベースのポリシーを KMS キーにアタッチします。必要に応じてアクセス許可の範囲を変更します。*region*、*account-id*、*agent-id*、および *key-id* を適切な値に置き換えます。

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

ナレッジベースリソースの暗号化

Amazon Bedrock はナレッジベースに関連するリソースを暗号化します。デフォルトでは、Amazon Bedrock は AWS マネージドキーを使用してこのデータを暗号化します。オプションで、カスタマーマネージドキーを使用して、モデルアーティファクトを暗号化することもできます。

KMS キーによる暗号化は、以下のプロセスで行うことができます。

- データソースの取り込み中の一時的なデータストレージ
- Amazon Bedrock にベクトルデータベースを設定させる場合の OpenSearch Service への情報の受け渡し
- ナレッジベースへのクエリの実行

ナレッジベースが使用する以下のリソースは KMS キーで暗号化できます。暗号化する場合は、KMS キーを復号するためのアクセス許可を追加する必要があります。

- Amazon S3 バケットに保存されているデータソース
- サードパーティーのベクトルストア

の詳細については AWS KMS keys、「 AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

トピック

- データインジェスト時の一時データストレージの暗号化
- Amazon OpenSearch Service に渡される情報の暗号化
- ナレッジベース取得の暗号化
- Amazon S3 のデータソースの AWS KMS キーを復号するためのアクセス許可

- ナレッジベースを含むベクトルストアの AWS Secrets Manager シークレットを復号化するアクセス許可

データインジェスト時の一時データストレージの暗号化

ナレッジベースのデータインジェストジョブを設定すると、カスタム KMS キーでジョブを暗号化できます。

データソースの取り込みプロセス中に一時的なデータストレージの AWS KMS キーを作成できるようにするには、Amazon Bedrock サービスロールに次のポリシーをアタッチします。*region*、*account-id*、*key-id* を適切な値に置き換えます。

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

Amazon OpenSearch Service に渡される情報の暗号化

Amazon Bedrock がナレッジベース用に Amazon OpenSearch Service にベクトルストアを作成することを許可することを選択した場合、Amazon Bedrock は選択した KMS キーを暗号化のために Amazon OpenSearch Service に渡すことができます。Amazon OpenSearch Service での暗号化の詳細については、「Amazon OpenSearch Service での暗号化」を参照してください。

ナレッジベース取得の暗号化

ナレッジベースに KMS キーでクエリを実行することにより、レスポンスを生成するセッションを暗号化することができます。そのためには、RetrieveAndGenerateリクエストを行うときに`kmsKeyArn`フィールドに KMS キーの ARN を含めます。Amazon Bedrock がセッションコンテキストを暗号化できるように、次のポリシーをアタッチし、*[#]* を適切に置き換えます。

```
{
    "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
        }
    ]
}
```

Amazon S3 のデータソースの AWS KMS キーを復号するためのアクセス許可

ナレッジベースのデータソースを Amazon S3 バケットに保存します。これらのドキュメントを保存中に暗号化するには、Amazon S3 SSE-S3 サーバーサイド暗号化オプションを使用できます。このオプションでは、オブジェクトは Amazon S3 サービスによって管理されるサービスキーで暗号化されます。

詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 マネージドキーによるサーバー側の暗号化 (SSE-S3)」を参照してください。

Amazon S3 のデータソースをカスタムキーで暗号化した場合は、Amazon Bedrock サービスロールに次のポリシーをアタッチして、Amazon Bedrock がキーを復号できるようにします。 AWS KMS *region* と *account-id* は、キーが属するリージョンとアカウント ID に置き換えます。*key-id* を AWS KMS キーの ID に置き換えます。

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

ナレッジベースを含むベクトルストアの AWS Secrets Manager シークレットを復号化するアクセス許可

ナレッジベースを含むベクトルストアが AWS Secrets Manager シークレットで設定されている場合、「 のシークレット暗号化と復号」の手順に従って、シークレットをカスタム AWS KMS キーで暗号化できます。 AWS Secrets Manager

そうする場合、Amazon Bedrock サービスロールに次のポリシーをアタッチして、サービスロールがキーを復号化できるようにします。*region* と *account-id* は、キーが属するリージョンとアカウント ID に置き換えます。*key-id* を AWS KMS キーの ID に置き換えます。

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

Amazon VPC と を使用してデータを保護する AWS PrivateLink

データへのアクセスを制御するには、Amazon VPC で Virtual Private Cloud (VPC) を使用することをお勧めします。VPC を使用すると、データを保護し、VPC Flow Logs を使用して AWS ジョブコンテナに出入りするすべてのネットワークトラフィックをモニタリングできます。インターネット経由

でデータを使用できるように VPC を設定し、代わりに を使用して VPC インターフェイスエンドポイントを作成してデータへのプライベート接続AWS PrivateLinkを確立することで、データをさらに保護できます。

VPC を使用して Amazon Bedrock と統合するデータを保護する例については、「」を参照してくださいVPC を使用してモデルカスタマイズジョブを保護する。

インターフェイス VPC エンドポイント (AWS PrivateLink) を使用する

を使用して AWS PrivateLink 、VPC と Amazon Bedrock の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にあるかのように Amazon Bedrock にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても Amazon Bedrock にアクセスできます。

このプライベート接続を確立するには、 AWS PrivateLink を利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、Amazon Bedrock 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、「 AWS PrivateLink ガイド」の「 AWS のサービス によるアクセス AWS PrivateLink」を参照してください。

Amazon Bedrock VPC エンドポイントに関する考慮事項

Amazon Bedrock のインターフェイスエンドポイントを設定する前に、「AWS PrivateLink ガイド」の「考慮事項」を確認してください。

Amazon Bedrock は、VPC エンドポイントを介して以下の API コールを実行できます。

カテゴリ	エンドポイントプレフィックス
Amazon Bedrock コントロールプレーン API アクション	`bedrock`
Amazon Bedrock ランタイム API アクション	`bedrock-runtime`
Agents for Amazon Bedrock ビルドタイム API アクション	`bedrock-agent`
Agents for Amazon Bedrock ランタイム API アクション	`bedrock-agent-runtime`

アベイラビリティーゾーン

Amazon Bedrock エンドポイントと Agents for Amazon Bedrock エンドポイントは、複数のアベイラビリティーゾーンで使用できます。

Amazon Bedrock 用のインターフェイスエンドポイントを作成する

Amazon Bedrock のインターフェイスエンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface () を使用して作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

以下のサービス名のいずれかを使用して Amazon Bedrock のインターフェイスエンドポイントを作成します。

- com.amazonaws.*region*.bedrock
- com.amazonaws.*region*.bedrock-runtime
- com.amazonaws.*region*.bedrock-agent
- com.amazonaws.*region*.bedrock-agent-runtime

エンドポイントを作成したら、プライベート DNS ホスト名を有効にするオプションがあります。VPC エンドポイントの作成時に VPC コンソールで [プライベート DNS 名を有効にする] を選択して、この設定名を有効にします。

エンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (など) を使用して、Amazon Bedrock への API リクエストを実行できます。次の例は、デフォルトのリージョン DNS 名の形式を示しています。

- bedrock.*region*.amazonaws.com
- bedrock-runtime.*region*.amazonaws.com
- bedrock-agent.*region*.amazonaws.com
- bedrock-agent-runtime.*region*.amazonaws.com

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント経由での Amazon Bedrock へのフルアクセスが許可されています。VPC から Amazon Bedrock への許可されたアクセ

スをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、IAM ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

例: Amazon Bedrock アクションの VPC エンドポイントポリシー

以下は、カスタムエンドポイントポリシーの例です。このリソースベースのポリシーをインターフェイスエンドポイントにアタッチすると、すべてのリソースのすべてのプリンシパルに対して、リストされている Amazon Bedrock アクションへのアクセスが許可されます。

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

Amazon Bedrock のためのアイデンティティとアクセス管理

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインインを許可) し、誰にAmazon Bedrockリソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Bedrock で行う作業によって異なります。

サービスユーザー - Amazon Bedrock サービスを使用してジョブを行う場合には、管理者から必要な認証情報と許可が提供されます。業務のために使用する Amazon Bedrock 機能が増えるにつれて、追加の許可が必要になる可能性があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Amazon Bedrock の機能にアクセスできない場合は、「Amazon Bedrock のアイデンティティとアクセスに関するトラブルシューティング」を参照してください。

サービス管理者 - 社内の Amazon Bedrock リソースを担当している場合は、通常、Amazon Bedrock へのフルアクセスがあります。サービスのユーザーがどの Amazon Bedrock 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon Bedrock と IAM を併用する方法の詳細については、「Amazon Bedrock で IAM が機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者は、通常、Amazon Bedrock へのアクセスを管理するポリシーの作成方法について詳細情報が必要になります。IAM で使用可能な、Amazon Bedrock アイデンティティベースのポリシーの例を確認するには、「Amazon Bedrock のアイデンティティベースのポリシー例」を参照してください。

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって認証 (にサインイン AWS) される必要があります。

ID ソース (AWS IAM Identity Center) から提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 にサインインする方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、 認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストを自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、「IAM ユーザーガイド」のAWS 「 API リクエストの署名」を参照してください。

使用する認証方法を問わず、セキュリティ情報の提供を追加でリクエストされる場合もあります。例えば、 では、多要素認証 (MFA) を使用してアカウントのセキュリティを高めることを AWS 推奨しています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証 (MFA)」および「IAM ユーザーガイド」の「AWS での多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な AWS のサービス 認証情報を使用して にアクセスする ID プロバイダーとのフェデレーションの使用を要求します。

フェデレーティッド ID とは、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、、Identity Center ディレクトリのユーザー、または ID ソースから提供された認証情報 AWS のサービス を使用して にアクセスするすべてのユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Center を使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、すべての およびアプリケーションで使用する独自の ID ソースのユーザー AWS アカウント とグループのセットに接続して同期することもできます。IAM アイデンティティセンターの詳細については、[AWS IAM Identity Center ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理するアクセス許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールを切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティにアクセス許可を割り当てるには、ロールを作成してそのロールのアクセス許可を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM Identity Center を使用する場合、アクセス許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM アイデンティティセンターは、アクセス許可セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザーアクセス許可 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なるアクセス許可を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、

を呼び出すプリンシパルのアクセス許可を使用し AWS のサービス、ダウンストリームサービスにリクエストを行う AWS のサービス リクエストと組み合わせて使用します。FAS リクエストは、他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストをサービスが受信した場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを作成しているアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「IAM ロールの作成が適している場合 (ユーザーではなく)」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御するには AWS 、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、ID またはリソースに関連付けられると、これらのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシー AWS を評価します。ポリシーでのアクセス許可により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント

AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するためのアクセス許可をユーザーに付与するため、IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションのアクセス許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。管理ポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーが挙げられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパ

ルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーで IAM の AWS マネージドポリシーを使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3 と Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、追加の一般的でないポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与される最大の許可を設定できます。

- 権限の境界 - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる許可の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。組織と SCP の詳細については、「AWS Organizations ユーザーガイド」の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限の範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあ

ります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連する場合に がリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシーの評価ロジック」を参照してください。

Amazon Bedrock で IAM が機能する仕組み

IAM を使用して Amazon Bedrock へのアクセスを管理する前に、Amazon Bedrock で使用できる IAM 機能について理解しておく必要があります。

Amazon Bedrock で使用できる IAM の機能

IAM 機能	Amazon Bedrock サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	いいえ
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー	はい
ACL	なし
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	あり
プリンシパル権限	あり
サービスロール	あり
サービスリンクロール	いいえ

Amazon Bedrock およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

Amazon Bedrock のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティに添付できる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon Bedrock のアイデンティティベースのポリシー例

Amazon Bedrock のアイデンティティベースポリシーの例を確認するには、「Amazon Bedrock のアイデンティティベースのポリシー例」を参照してください。

Amazon Bedrock 内のリソースベースのポリシー

リソースベースのポリシーのサポート	なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーが挙げられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定

義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、リソースへのアクセス許可をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティに添付することで許可を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon Bedrock のポリシーアクション

ポリシーアクションに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amazon Bedrock アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon Bedrock で定義されるアクション」を参照してください。

Amazon Bedrock のポリシーアクションでは、アクションの前に、次のプレフィックスを使用します。

```
bedrock
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

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

Amazon Bedrock のアイデンティティベースポリシーの例を確認するには、「Amazon Bedrock のアイデンティティベースのポリシー例」を参照してください。

Amazon Bedrock のポリシーリソース

ポリシーリソースに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシーの要素は、オブジェクトあるいはアクションが適用されるオブジェクトを指定します。ステートメントには、`Resource` または `NotResource` 要素を含める必要があります。ベストプラクティスとしては、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルのアクセス許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

Amazon Bedrock リソースのタイプとその ARNs」の「Amazon Bedrock で定義されるリソース」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、「Amazon Bedrock で定義されるアクション」を参照してください。

一部の Amazon Bedrock API アクションは、複数のリソースをサポートしています。例えば、 は *AGENT12345* および *KB12345678* AssociateAgentKnowledgeBaseにアクセスするため、プリンシパルには両方のリソースへのアクセス許可が必要です。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
   "arn:aws:bedrock:aws-region:111122223333:agent/AGENT12345",
   "arn:aws:bedrock:aws-region:111122223333:knowledge-base/KB12345678"
]
```

Amazon Bedrock のアイデンティティベースポリシーの例を確認するには、「Amazon Bedrock のアイデンティティベースのポリシー例」を参照してください。

Amazon Bedrock のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。equal や less than などの条件演算子を使用して条件式を作成することによって、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理OR演算を使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる許可を付与できます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、 グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドのAWS 「 グローバル条件コンテキストキー」を参照してください。

Amazon Bedrock の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon Bedrock の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Bedrock で定義されるアクション」を参照してください。

すべての Amazon Bedrock アクションは、Amazon Bedrock モデルをリソースとして使用する条件キーをサポートしています。

Amazon Bedrock のアイデンティティベースポリシーの例を確認するには、「Amazon Bedrock のアイデンティティベースのポリシー例」を参照してください。

Amazon Bedrock のACL

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon Bedrock での ABAC

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。次に、プリンシパルのタグがアクセスを試行するリソースのタグと一致したときにオペレーションを許可するよう、ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを制御するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値は Yes です。サービスが一部のリソースタイプに対してのみ 3 つの条件キーすべてをサポートする場合、値は Partial です。

ABAC の詳細については、「IAM ユーザーガイド」の [ABAC とは?」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセス制御 (ABAC) を使用する」を参照してください。

Amazon Bedrock での一時的な認証情報の使用

一時的な認証情報のサポート	はい

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドAWS のサービス の「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合は、一時的な認証情報を使用しています。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスは自動的に一時的な認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して、長期的なアクセスキーを使用する代わりに、動的に一時的な認証情報を生成する AWS. AWS recommends にアクセスできます。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

Amazon Bedrock のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	はい

IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可 AWS のサービス を使用し AWS のサービス、ダウンストリームサービスにリクエストを行うリクエストと組み合わせて使用します。FAS リクエストは、他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストをサービスが受信した場合にのみ行われます。この場合、両方の

アクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon Bedrock のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス に権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、Amazon Bedrock の機能が破損する可能性があります。Amazon Bedrock が指示する場合以外は、サービスロールを編集しないでください。

Amazon Bedrock のサービスリンクロール

サービスにリンクされたロールのサポート	いいえ

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

Amazon Bedrock のアイデンティティベースのポリシー例

デフォルトでは、ユーザーとロールには Amazon Bedrock リソースを作成または変更するアクセス許可がありません。また、 AWS Management Console、 AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。

Amazon Bedrock が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon Bedrock のアクション、リソース、条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon Bedrock コンソールを使用する
- 自分の許可の表示をユーザーに許可する
- サードパーティーモデルのサブスクリプションへのアクセスを許可する
- 特定のモデルで推論するためのアクセス許可を拒否する
- Amazon Bedrock のエージェント向けのアイデンティティベースのポリシーの例
- プロビジョニングされたスループットの ID ベースのポリシーの例

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウント内で誰かが Amazon Bedrock リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウントに料金が発生する可能性があります。アイデンティティベースのポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください。

- AWS 管理ポリシーから始めて、最小権限の権限に移行する — ユーザーとワークロードへのアクセス権限の付与を開始するには、AWS 多くの一般的なユースケースで権限を付与する管理ポリシーを使用してください。これらのポリシーは、で利用できます。 AWS アカウント AWS ユースケースに固有のカスタマー管理ポリシーを定義して、権限をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定するときは、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、 最小特権アクセス許可 とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。

- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。サービスアクションがなどの特定の用途で使用された場合は AWS のサービス、条件を使用してサービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) が必要 — IAM ユーザーまたは root ユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA をオンにしてください。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

Amazon Bedrock コンソールを使用する

Amazon Bedrock コンソールにアクセスするには、アクセス許可の最小限のセットが必要です。これらの権限により、内の Amazon Bedrock AWS アカウントリソースの詳細を一覧表示して表示できる必要があります。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出しているユーザーには、最低限のコンソール権限を与える必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Amazon Bedrock コンソールを使用できるようにするには、Amazon Bedrock AmazonBedrockFullAccessAmazonBedrockReadOnly AWS または管理ポリシーをエンティティにアタッチします。詳細については、『IAM ユーザーガイド』の「ユーザーへの権限の追加」を参照してください。

自分の許可の表示をユーザーに許可する

この例では、ユーザーアイデンティティに添付されたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーを作成する方法を示します。このポリシーには、コンソールで、またはまたは API を使用してこのアクションをプログラムで完了するためのアクセス権限が含まれています。 AWS CLI AWS

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

サードパーティーモデルのサブスクリプションへのアクセスを許可する

Amazon Bedrock モデルに初めてアクセスするには、Amazon Bedrock コンソールを使用してサードパーティーモデルをサブスクライブします。コンソールユーザーが引き受ける IAM ユーザーまたはロールには、サブスクリプションの API オペレーションにアクセスするためのアクセス許可が必要です。

次の例は、サブスクリプションの API オペレーションへのアクセスを許可するアイデンティティベースのポリシーを示しています。

例のように条件キーを使用して、ポリシーの範囲をMarketplace の Amazon Bedrock 基盤モデルのサブセットに限定します。製品 ID のリストとそれに対応する基盤モデルについては、の表を参照してください。モデルのアクセス権限を制御します。

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

特定のモデルで推論するためのアクセス許可を拒否する

次の例は、特定のモデルで推論を実行するアクセス許可を拒否するアイデンティティベースポリシーを示しています。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "DenyInference",
        "Effect": "Deny",
        "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
         ],
        "Resource": "arn:aws:bedrock:*::foundation-model/model-id"
    }
}
```

Amazon Bedrock のエージェント向けのアイデンティティベースのポリシーの例

トピックを選択すると、IAM ロールにアタッチして IAM ロール内のアクションのアクセス権限をプロビジョニングできる IAM ポリシーの例が表示されます。Agents for Amazon Bedrock

トピック

- Amazon Bedrock のエージェントに必要な権限
- ユーザーがエージェントに関する情報を表示したり、エージェントを呼び出したりできるようにします。

Amazon Bedrock のエージェントに必要な権限

IAM ID が Amazon Bedrock 用エージェントを使用するには、必要な権限を設定する必要があります。AmazonBedrockFullAccessポリシーをアタッチして、ロールに適切なアクセス権限を付与できます。

Agents for Amazon Bedrock で使用されるアクションのみにアクセス権限を制限するには、以下のアイデンティティベースのポリシーを IAM ロールにアタッチします。

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

アクションを省略したり、リソースや条件キーを指定したりすることで、アクセス権限をさらに制限できます。IAM ID は特定のリソースに対する API オペレーションを呼び出すことができます。たとえば、UpdateAgentオペレーションはエージェントリソースでのみ使用でき、InvokeAgentオペレーションはエイリアスリソースでのみ使用できます。特定のリソースタイプ (など CreateAgent) では使用されない API オペレーションの場合は、* をとして指定しますResource。ポリシーで指定されたリソースでは使用できない API オペレーションを指定すると、Amazon Bedrock はエラーを返します。

ユーザーがエージェントに関する情報を表示したり、エージェントを呼び出したりできるようにします。

*####IAM ############ID AGENT12345 ##########################ID ALIAS12345 ##*たとえば、エージェントのトラブルシューティングと更新を行う権限だけを付与したいロールに、このポリシーをアタッチできます。

```
{
    "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"
        },
    ]
}
```

プロビジョニングされたスループットの ID ベースのポリシーの例

トピックを選択すると、IAM ロールにアタッチして、関連するアクションのアクセス権限をプロビジョニングできる IAM ポリシーの例が表示されます。Amazon Bedrock のプロビジョニングされたスループット

トピック

- プロビジョンドスループットに必要な権限
- プロビジョニングされたモデルをユーザーが呼び出すことを許可します。

プロビジョンドスループットに必要な権限

IAM ID がプロビジョンドスループットを使用するには、必要なアクセス権限を設定する必要があります。AmazonBedrockFullAccessポリシーをアタッチして、ロールに適切なアクセス権限を付与できます。

アクセス権限をプロビジョンドスループットで使用されるアクションのみに制限するには、以下のアイデンティティベースのポリシーを IAM ロールにアタッチします。

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

アクションを省略したり、リソースや条件キーを指定したりすることで、アクセス権限をさらに制限できます。IAM ID は特定のリソースに対する API オペレーションを呼び出すことができます。たとえば、CreateProvisionedModelThroughputオペレーションはカスタムモデルと基盤モデルのリソースでのみ使用でき、DeleteProvisionedModelThroughputオペレーションはプロビジョニングされたモデルリソースでのみ使用できます。特定のリソースタイプ (など ListProvisionedModelThroughputs) では使用されない API オペレーションでは、* を指定します。Resourceポリシーで指定されたリソースでは使用できない API オペレーションを指定すると、Amazon Bedrock はエラーを返します。

プロビジョニングされたモデルをユーザーが呼び出すことを許可します。

以下は、IAM ロールにアタッチして、プロビジョニングされたモデルをモデル推論に使用できるようにするサンプルポリシーです。たとえば、プロビジョニングされたモデルを使用する権限のみを持たせたいロールにこのポリシーをアタッチできます。そのロールは、プロビジョニングされたスループットに関する情報を管理したり表示したりできなくなります。

```
{
    "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 の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な許可のみを提供する IAM カスタマーマネージドポリシーを作成するには、時間と専門知識が必要です。すぐに使用を開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、 AWS アカウント で利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS のサービスは、 AWS マネージドポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスでは、新しい機能を利用できるようにするために、 AWS マネージドポリシーに権限が追加されることがあります。この種類の更新は、ポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が破棄されることはありません。

さらに、 は、複数のサービスにまたがる職務機能の管理ポリシー AWS をサポートしています。例えば、 ReadOnlyAccess AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新しい機能を起動すると、 は新しいオペレーションとリソースに読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

AWS マネージドポリシー： AmazonBedrockFullAccess

`AmazonBedrockFullAccess` ポリシーは IAM ID にアタッチできます。

このポリシーは、ユーザーアクセス許可に Amazon Bedrock リソースの作成、読み取り、更新、および削除を許可する管理者アクセス許可を付与します。

> **Note**
>
> ファインチューニングとモデルアクセスには追加のアクセス許可が必要です。詳細については、「サードパーティーモデルのサブスクリプションへのアクセスを許可する」と「トレー

ニングファイルや検証ファイルにアクセスし、S3 に出力ファイルを書き込む権限」を参照してください。

許可の詳細

このポリシーには、以下の許可が含まれています。

- ec2 (Amazon Elastic Compute Cloud) – VPC、サブネット、およびセキュリティグループを記述するためのアクセス許可を許可します。
- iam (AWS Identity and Access Management) – プリンシパルはロールを渡すことができますが、「Amazon Bedrock」を含む IAM ロールのみを Amazon Bedrock サービスに渡すことを許可します。アクセス許可は Amazon Bedrock オペレーションの bedrock.amazonaws.com に限定されています。
- kms (AWS Key Management Service) — プリンシパルが AWS KMS キーとエイリアスを記述できるようにします。
- bedrock (Amazon Bedrock) – Amazon Bedrock コントロールプレーンおよびランタイムサービスのすべてのアクションに対するプリンシパルの読み取りおよび書き込みアクセスを許可します。

```
{
    "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 マネージドポリシー： AmazonBedrockReadOnly

AmazonBedrockReadOnly ポリシーは IAM ID にアタッチできます。

このポリシーは、ユーザーが Amazon Bedrock ですべてのリソースを表示できるようにする読み取り専用のアクセス許可を付与します。

```
{
    "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 での AWS マネージドポリシーの更新

Amazon Bedrock の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページへの変更に関する自動通知を受けるには、Amazon Bedrock ユーザーガイドのドキュメント履歴 の RSS フィードを購読してください。

変更	説明	日付
AmazonBedrockFullAccess - 新しいポリシー	Amazon Bedrock は、リソースの作成、読み取り、更新、および削除に対するアクセス許可をユーザーに付与する新しいポリシーを追加しました。	2023 年 12 月 12 日
AmazonBedrockReadOnly - 新しいポリシー	Amazon Bedrock は、すべてのアクションに対する読み取り専用アクセス許可をユーザーに付与する新しいポリシーを追加しました。	2023 年 12 月 12 日
Amazon Bedrock が変更の追跡を開始	Amazon Bedrock が AWS マネージドポリシーの変更の追跡を開始しました。	2023 年 12 月 12 日

サービスロール

Amazon Bedrock では、以下の機能に IAM サービスロールを使用して、Amazon Bedrock がユーザーに代わってタスクを実行できるようにします。

コンソールは、サポートされている機能のサービスロールを自動的に作成します。

また、カスタムサービスロールを作成して、添付されている権限を特定のユースケースに合わせてカスタマイズすることもできます。コンソールを使用する場合は、Amazon Bedrock にロールを作成させる代わりに、このロールを選択できます。

カスタムサービスロールを設定するには、以下の一般的なステップを実行します。

1. 「AWS サービスに権限を委任するロールの作成」の手順に従ってロールを作成します。
2. 信頼ポリシーを添付します。
3. 関連する ID ベースの権限をアタッチします。

サービスロール権限の設定に関連する IAM の概念の詳細については、以下のリンクを参照してください。

- AWS サービスロール
- アイデンティティベースおよびリソースベースのポリシー
- Lambda 用のリソースベースのポリシーの使用
- AWS グローバル条件コンテキストキー
- Amazon ベッドロックのコンディションキー

トピックを選択すると、特定の機能のサービスロールに関する詳細が表示されます。

トピック

- モデルカスタマイズ用のサービスロールの作成
- Amazon Bedrock のエージェント用のサービスロールを作成する
- Amazon Bedrock のナレッジベースのサービスロールを作成する

モデルカスタマイズ用のサービスロールの作成

Amazon Bedrock が自動的に作成するロールの代わりにカスタムロールを使用してモデルをカスタマイズするには、「サービスにアクセス権限を委任するロールの作成」の手順に従って IAM ロールを作成し、以下のアクセス権限をアタッチします。 AWS

- 信頼関係
- S3 のトレーニングデータや検証データにアクセスし、出力データを S3 に書き込む権限
- (オプション) 以下のリソースを KMS キーで暗号化する場合、キーを復号化するアクセス許可 (「モデルカスタマイズジョブとアーティファクトの暗号化」を参照)
  - モデルカスタマイズジョブ、または生成されたカスタムモデル
  - モデルカスタマイズジョブ用のトレーニング、検証、または出力データ

トピック

- 信頼関係
- トレーニングファイルや検証ファイルにアクセスし、S3 に出力ファイルを書き込む権限

信頼関係

以下のポリシーでは、Amazon Bedrock がこのロールを引き受け、モデルカスタマイズジョブを実行できます。使用するポリシーの例を下記に示します。

Conditionフィールドに 1 つ以上のグローバル条件コンテキストキーを使用することにより、サービス間の混乱した代理防止のための権限の範囲を任意で制限できます。詳細については、「AWS グローバル条件コンテキストキー」を参照してください。

- aws:SourceAccount の値をアカウント ID に設定します。
- (オプション) ArnEquals or ArnLike 条件を使用して、アカウント ID 内の特定のモデルカスタマイズジョブに範囲を制限します。

```
{
    "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/*"
                }
            }
        }
    ]
}
```

トレーニングファイルや検証ファイルにアクセスし、S3 に出力ファイルを書き込む権限

次のポリシーをアタッチして、トレーニングデータと検証データ、および出力データを書き込むバケットにロールがアクセスできるようにします。Resourceリスト内の値を実際のバケット名に置き換えてください。

バケット内の特定のフォルダーへのアクセスを制限するには、s3:prefixフォルダーパスに条件キーを追加します。「例 2: 特定のプレフィックスを持つバケット内のオブジェクトのリストを取得する」のユーザーポリシーの例に従うことができます。

```
{
    "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/*"
            ]
        }
    ]
}
```

Amazon Bedrock のエージェント用のサービスロールを作成する

Amazon Bedrock が自動的に作成するサービスロールの代わりにエージェント用のカスタムサービスロールを使用するには、AmazonBedrockExecutionRoleForAgents_プレフィックスを付けたIAM ロールを作成し、「サービスにアクセス権限を委任するロールの作成」の手順に従って以下のアクセス権限をアタッチします。 AWS

- 信頼ポリシー
- 次の ID ベースのアクセス権限を含むポリシー
  - Amazon Bedrock のベースモデルへのアクセス
  - OpenAPIエージェントのアクショングループのスキーマを含む Amazon S3 オブジェクトへのアクセス
  - エージェントに添付したいナレッジベースをクエリするための Amazon Bedrock の権限
  - (オプション) KMS キーでエージェントを暗号化する場合、キーを復号化するアクセス許可 (「エージェントリソースの暗号化」を参照)

カスタムロールを使用するかどうかに関係なく、エージェントのアクショングループの Lambda 関数にリソースベースのポリシーをアタッチして、サービスロールが関数にアクセスするためのアクセス権限を付与する必要もあります。詳細については、「Amazon Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシー」を参照してください。

トピック

- 信頼関係
- Agents サービスロールの ID ベースのアクセス許可。

- Amazon Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシー

信頼関係

以下の信頼ポリシーにより、Amazon Bedrock がこの役割を引き受け、エージェントを作成および管理することができます。*##################*。ポリシーには、セキュリティのベストプラクティスとして使用することをお勧めするオプションの条件キー (「Amazon Bedrock AWS の条件キーとグローバル条件コンテキストキー」を参照) が含まれています。Condition

> **Note**
>
> セキュリティ上のベストプラクティスとして、* は特定のエージェント ID に置き換えてください (作成後)。

```
{
    "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/*"
            }
        }
    }]
}
```

Agents サービスロールの ID ベースのアクセス許可。

次のポリシーをアタッチしてサービスロールに権限を付与し、*##############*。ポリシーには以下のステートメントが含まれています。使用事例に当てはまらない場合は記述を省略してください。ポリシーには、セキュリティのベストプラクティスとして使用することをお勧めするオプションの条

件キー (「Amazon Bedrock AWS の条件キーとグローバル条件コンテキストキー」を参照) が含まれています。Condition

> **Note**
>
> 顧客管理の KMS キーでエージェントを暗号化する場合、エージェントリソースの暗号化追加する必要のあるその他の権限についてはを参照してください。

- Amazon Bedrock Foundation モデルを使用して、エージェントのオーケストレーションで使用されるプロンプトに対してモデル推論を実行する権限。
- Amazon S3 のエージェントのアクショングループ API スキーマにアクセスする権限。エージェントにアクショングループがない場合は、このステートメントを省略してください。
- エージェントに関連するナレッジベースにアクセスする権限。エージェントにナレッジベースが関連付けられていない場合は、このステートメントを省略してください。
- エージェントに関連するサードパーティ (PineconeまたはRedis Enterprise Cloud) のナレッジベースにアクセスする権限。ナレッジベースがファーストパーティ (Amazon OpenSearch Serverless または Amazon Aurora) の場合、またはエージェントに関連するナレッジベースがない場合は、このステートメントを省略してください。

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

Amazon Bedrock がアクショングループ Lambda 関数を呼び出すことを許可するリソースベースのポリシー

*#Lambda ## Lambda ##########Amazon Bedrock ################## Lambda ##################################*ポリシーには、セキュリティのベストプラクティスとして使用することをお勧めするオプションの条件キー (「Amazon Bedrock AWS の条件キーとグローバル条件コンテキストキー」を参照) が含まれています。Condition

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

Amazon Bedrock のナレッジベースのサービスロールを作成する

Amazon Bedrock が自動的に作成するロールの代わりにナレッジベース用のカスタムロールを使用するには、「サービスにアクセス権限を委任するロールの作成」の手順に従って IAM ロールを作成し、以下のアクセス権限をアタッチします。 AWS ナレッジベース全体で同じロールを使用できます。

- 信頼関係
- Amazon Bedrock のベースモデルへのアクセス
- データソースを含む Amazon S3 オブジェクトへのアクセス許可

- (Amazon OpenSearch Service でベクターデータベースを作成した場合) OpenSearch サービスコレクションへのアクセス
- (Amazon Aurora でベクトルデータベースを作成する場合)
- (Pineconeまたはでベクターデータベースを作成した場合Redis Enterprise Cloud) AWS Secrets Manager Pinecone Redis Enterprise Cloud またはアカウントを認証するための権限
- (オプション) 以下のリソースを KMS キーで暗号化する場合、キーを復号化するアクセス許可 (「ナレッジベースリソースの暗号化」を参照)。
  - ナレッジベース
  - ナレッジベースのデータソース
  - Amazon OpenSearch サービスのベクターデータベース
  - サードパーティのベクターデータベースの秘密は AWS Secrets Manager
  - データインジェストジョブ

トピック

信頼関係

以下のポリシーにより、Amazon Bedrock がこのロールを引き受け、ナレッジベースを作成および管理することができます。使用するポリシーの例を下記に示します。1 つ以上のグローバル条件コンテキストキーを使用して、アクセス許可の範囲を制限できます。詳細については、「AWS グローバル条件コンテキストキー」を参照してください。`aws:SourceAccount` の値をアカウント ID に設定します。`ArnEquals` または `ArnLike` 条件を使用して、範囲を特定のナレッジベースに制限します。

> **Note**
>
> セキュリティ上のベストプラクティスとして、* は特定のナレッジベース ID に置き換えてください (作成後)。

```
{
    "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/*"
            }
        }
    }]
}
```

Amazon Bedrock モデルにアクセスするためのアクセス許可

ロールがソースデータの組み込みに Amazon Bedrock モデルを使用するためのアクセス許可を提供するには、以下のポリシーをアタッチします。

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

Amazon S3 内のデータソースにアクセスするためのアクセス許可

ナレッジベースのデータソースファイルを含む Amazon S3 URI にアクセスするためのアクセス許可をロールに提供するには、以下のポリシーをアタッチします。[Resource] フィールドでは、データソースを含む Amazon S3 オブジェクトを指定するか、各データソースの URI をリストに追加します。

AWS KMS これらのデータソースをキーで暗号化した場合は、の手順に従ってキーを復号化する権限をロールにアタッチします。Amazon S3 のデータソースの AWS KMS キーを復号するためのアクセス許可

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

```
}
```

(オプション) Amazon OpenSearch Service のベクターデータベースにアクセスするための権限

Amazon OpenSearch Service でナレッジベース用にベクターデータベースを作成した場合は、以下のポリシーを Amazon Bedrock のナレッジベースサービスロールにアタッチして、コレクションへのアクセスを許可してください。*region* と *account-id* は、データベースが属するリージョンとアカウント ID に置き換えます。*collection-id* にAmazon OpenSearch サービスのコレクションの ID を入力します。複数のコレクションへのアクセスを許可するには、それらを Resource リストに追加します。

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

(オプション) Amazon Aurora データベースクラスターにアクセスするためのアクセス許可

Amazon Aurora でナレッジベース用のデータベース (DB) クラスターを作成した場合は、以下のポリシーをナレッジベース for Amazon Bedrock サービスロールにアタッチして、DB クラスターへのアクセスを許可し、DB クラスターに対する読み取り/書き込み権限を付与します。*region* と *account-id* は、DB クラスターが属するリージョンとアカウント ID に置き換えます。Amazon Aurora データベースクラスターの *db-cluster-id*ID を入力します。複数の DB クラスターへのアクセスを許可するには、それらを Resource リストに追加します。

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

(オプション) シークレットが設定されたベクターデータベースにアクセスする権限 AWS Secrets Manager

AWS Secrets Manager ベクターデータベースにシークレットが設定されている場合は、Amazon Bedrock サービスロールのナレッジベースに次のポリシーをアタッチして、 AWS Secrets Manager アカウントを認証してデータベースにアクセスできるようにします。*region* と *account-id* は、データベースが属するリージョンとアカウント ID に置き換えます。*secret-id* はシークレットの ID に置き換えます。

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

シークレットをキーで暗号化した場合は、 AWS KMS の手順に従ってキーを復号化する権限をロールにアタッチします。ナレッジベースを含むベクトルストアの AWS Secrets Manager シークレットを復号化するアクセス許可

(オプション) AWSAWS KMS データ取り込み時の一時データストレージ用のキーを管理するための権限

AWS KMS データソースの取り込み中に一時的なデータストレージ用のキーを作成できるようにするには、Amazon Bedrock サービスロールのナレッジベースに次のポリシーをアタッチします。*region*、*account-id*、*key-id* を適切な値に置き換えます。

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

Amazon Bedrock のアイデンティティとアクセスに関するトラブルシューティング

以下の情報を使用して、Amazon Bedrock と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立てます。

トピック

- I am not authorized to perform an action in Amazon Bedrock
- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに Amazon Bedrock リソース AWS アカウント へのアクセスを許可したい

I am not authorized to perform an action in Amazon Bedrock

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次の例は、mateojackson という IAM ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細を表示しようとしたとき、架空の bedrock:*GetWidget* アクセス許可がない場合に発生するエラーを示しています。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 bedrock:GetWidget on resource: my-example-widget
```

この場合、bedrock:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

iam を実行する権限がありません。PassRole

「I am not authorized to perform iam:PassRole」というエラーが表示された場合は、Amazon Bedrock にロールを渡すことを許可するようにポリシーを更新する必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

次の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon Bedrock でアクションを実行しようとした場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新して Mary に iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに Amazon Bedrock リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon Bedrock がこれらの機能をサポートしているかどうかを確認するには、「Amazon Bedrock で IAM が機能する仕組み」を参照してください。
- 所有 AWS アカウント する のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント する別の の IAM ユーザーへのアクセスを許可する」を参照してください。
- サードパーティー に リソースへのアクセスを提供する方法については AWS アカウント、IAM ユーザーガイドの「第三者 AWS アカウント が所有する へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセス権を付与する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) にアクセス権を付与する」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon Bedrock のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの対象であるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによる対象範囲内の 」を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

サードパーティーの監査レポートは、 を使用してダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、企業のコンプライアンス目的、適用法規によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド — これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置 AWS いたベースライン環境を にデプロイするための手順を説明します。
- Architecting for HIPAA Security and Compliance on Amazon Web Services – このホワイトペーパーでは、企業が AWS を使用して HIPAA 対応のアプリケーションを作成する方法について説明します。

> **Note**
>
> すべての AWS のサービス が HIPAA に対応しているわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や場所に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドは、複数のフレームワーク (米国標準技術研究所 (NIST)、Payment Card Industry Security Standards Association (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティコントロールにガイダンスを保護し AWS のサービス 、マッピングするためのベストプラクティスをまとめたものです。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 — この AWS Config サービスは、リソース設定が社内のプラクティス、業界ガイドライン、規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- AWS Audit Manager — これにより AWS のサービス 、 AWS の使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon Bedrock でのインシデントへの対応

AWS では、セキュリティが最優先事項です。 AWS クラウドの責任共有モデル の一環として、 は、セキュリティを最も重視する組織の要件を満たすデータセンター、ネットワーク、およびソフトウェアアーキテクチャ AWS を管理します。 AWS は、Amazon Bedrock サービス自体に関するインシデント対応を担当します。また、 AWS のお客様は、クラウドでセキュリティを維持する責任を共有します。つまり、ユーザーがアクセスできる AWS ツールや機能から実装するセキュリティをユーザーが制御できます。さらに、責任共有モデルのユーザー側でのインシデント対応も担当します。

クラウド上で稼働するアプリケーションの目標を満たすセキュリティベースラインを確立することで、対応可能な逸脱を検出できます。インシデント対応と選択が企業の目標に与える影響を理解しやすくするために、以下のリソースを確認することをお勧めします。

- AWS セキュリティインシデント対応ガイド
- AWS セキュリティ、アイデンティティ、コンプライアンスのベストプラクティス
- AWS クラウド導入フレームワーク (CAF) ホワイトペーパーのセキュリティ面

Amazon Bedrock の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン とアベイラビリティーゾーンを中心に構築されています。 は、低レイテンシー、高スループット、および高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon Bedrock でのインフラストラクチャセキュリティ

マネージドサービスである Amazon Bedrock は AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

が AWS 公開している API コールを使用して、ネットワーク経由で Amazon Bedrock にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2、できれば TLS 1.3 が必要です。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token ServiceAWS STS を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

サービス間の混乱した代理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間でなりすましを行うと、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために AWS では、お客様のすべてのサービスのデータを保護するのに役立つツールを提供しています。これには、アカウントのリソースへのアクセス許可が付与されたサービスプリンシパルを使用します。

リソースポリシー内では `aws:SourceArn` および `aws:SourceAccount` のグローバル条件コンテキストキーを使用して、Amazon Bedrock が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、`aws:SourceAccount` を使用します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、`aws:SourceArn` グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー `aws:SourceArn` で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、`arn:aws:bedrock:*:123456789012:*` です。

`aws:SourceArn` の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。

aws:SourceArn の値は ResourceDescription である必要があります。

次の例では、Bedrock の aws:SourceArn と aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を回避する方法を示します。

```
{
    "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 での設定と脆弱性の分析

設定と IT コントロールは、 AWS とお客様の間で共有される責任です。詳細については、 AWS 「責任共有モデル」を参照してください。

インターフェイス VPC エンドポイント (AWS PrivateLink) を使用する

を使用して AWS PrivateLink 、VPC と Amazon Bedrock の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にあるかのように Amazon Bedrock にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても Amazon Bedrock にアクセスできます。

このプライベート接続を確立するには、 AWS PrivateLink を利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、Amazon Bedrock 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、「 AWS PrivateLink ガイド」の「 AWS のサービス によるアクセス AWS PrivateLink」を参照してください。

Amazon Bedrock VPC エンドポイントに関する考慮事項

Amazon Bedrock のインターフェイスエンドポイントを設定する前に、「AWS PrivateLink ガイド」の「考慮事項」を確認してください。

Amazon Bedrock は、VPC エンドポイントを介して以下の API コールを実行できます。

カテゴリ	エンドポイントプレフィックス
Amazon Bedrock コントロールプレーン API アクション	`bedrock`
Amazon Bedrock ランタイム API アクション	`bedrock-runtime`
Agents for Amazon Bedrock ビルドタイム API アクション	`bedrock-agent`
Agents for Amazon Bedrock ランタイム API アクション	`bedrock-agent-runtime`

アベイラビリティーゾーン

Amazon Bedrock エンドポイントと Agents for Amazon Bedrock エンドポイントは、複数のアベイラビリティーゾーンで使用できます。

Amazon Bedrock 用のインターフェイスエンドポイントを作成する

Amazon Bedrock のインターフェイスエンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface () を使用して作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

以下のサービス名のいずれかを使用して Amazon Bedrock のインターフェイスエンドポイントを作成します。

- com.amazonaws.*region*.bedrock
- com.amazonaws.*region*.bedrock-runtime
- com.amazonaws.*region*.bedrock-agent
- com.amazonaws.*region*.bedrock-agent-runtime

エンドポイントを作成したら、プライベート DNS ホスト名を有効にするオプションがあります。VPC エンドポイントの作成時に VPC コンソールで [プライベート DNS 名を有効にする] を選択して、この設定名を有効にします。

エンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (など) を使用して、Amazon Bedrock への API リクエストを実行できます。次の例は、デフォルトのリージョン DNS 名の形式を示しています。

- bedrock.*region*.amazonaws.com
- bedrock-runtime.*region*.amazonaws.com
- bedrock-agent.*region*.amazonaws.com
- bedrock-agent-runtime.*region*.amazonaws.com

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント経由での Amazon Bedrock へのフルアクセスが許可されています。VPC から Amazon Bedrock への許可されたアクセスをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、IAM ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

例: Amazon Bedrock アクションの VPC エンドポイントポリシー

以下は、カスタムエンドポイントポリシーの例です。このリソースベースのポリシーをインターフェイスエンドポイントにアタッチすると、すべてのリソースのすべてのプリンシパルに対して、リストされている Amazon Bedrock アクションへのアクセスが許可されます。

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

Amazon Bedrock をモニタリングする

Amazon Bedrock は Amazon CloudWatch と Amazon でモニタリングできます EventBridge。

トピック

モデル呼び出しのログ記録

モデル呼び出しログ記録を使用して、Amazon Bedrock AWS アカウント で使用される のすべての呼び出しの呼び出しログ、モデル入力データ、モデル出力データを収集できます。ログ記録はデフォルトでは無効になっています。

呼び出しのログ記録を使用すると、アカウントで実行したすべての呼び出しに関連する完全なリクエストデータ、応答データ、およびメタデータを収集できます。ログ記録は、ログデータが公開される送信先リソースを出力するように設定できます。サポートされている送信先には、Amazon CloudWatch Logs と Amazon Simple Storage Service (Amazon S3) があります。同じアカウントとリージョンの送信先のみがサポートされます。

呼び出しログ記録を有効にする前に、Amazon S3 または Logs の送信先を設定する必要があります。 CloudWatch 呼び出しのログ記録は、コンソールまたは API を使用して有効にできます。

トピック

Amazon S3 送信先をセットアップする

Amazon Bedrock にログインするために S3 送信先をセットアップするには、以下の手順に従います。

1. ログが配信される S3 バケットを作成します。
2. *#################### (accountId#region#bucketName########### prefix* を値に置き換えます)。

> **Note**
>
> アクセス許可 S3:GetBucketPolicy と S3:PutBucketPolicy を使用して口グ記録を設定すると、ユーザーに代わって自動的にバケットポリシーがバケットに追加されます。

```
{
  "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. (オプション) バケットで SSE-KMS を設定する場合は、KMS キーで以下のポリシーを追加します。

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

S3 SSE-KMS 設定の詳細については、「AWS KMS (SSE-KMS) によるサーバー側の暗号化の指定」を参照してください。

> **Note**
>
> バケットポリシーを有効にするには、バケット ACL を無効にする必要があります。詳細については、「すべての新しいバケットの ACL を無効にし、オブジェクト所有権を執行します。」を参照してください。

CloudWatch Logs 送信先のセットアップ

Amazon Bedrock へのログイン用に Amazon CloudWatch Logs の送信先を設定するには、以下の手順を実行します。

1. CloudWatch ログが発行されるロググループを作成します。
2. Logs に対して次のアクセス許可を持つ IAM CloudWatch ロールを作成します。

   信頼されたエンティティ:

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

ロールポリシー:

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

CloudWatch ログの SSE の設定の詳細については、「AWS Key Management Service を使用して CloudWatch ログのログデータを暗号化する」を参照してください。

コンソールを使用する場合

モデル呼び出しのログ記録を有効にするには、[設定] ページの [ログ記録] トグルスイッチの横にあるスライダーボタンをドラッグします。ログ記録用の追加構成設定がパネルに表示されます。

ログに公開するデータリクエストとレスポンスを選択します。以下の出力オプションの任意の組み合わせを選択することもできます。

- テキスト
- イメージ
- 埋め込み

ログを公開する場所を選択します。

- Amazon S3 のみ
- CloudWatch ログのみ
- Amazon S3 と CloudWatch ログの両方

Amazon S3 と CloudWatch Logs の送信先は、呼び出しログ、および小さな入出力データでサポートされています。大量の入出力データやバイナリイメージ出力には、Amazon S3 のみが対応しています。以下の詳細は、ターゲット送信先でのデータの表示方法をまとめたものです。

- 送信先: S3 - 指定された S3 バケットには、Gzip 圧縮された JSON ファイルが配信されます。各ファイルには、呼び出しのログレコードのバッチが含まれています。 CloudWatch Logs イベントと同様に、各レコードには呼び出しメタデータと、最大 100 KB のサイズの入出力 JSON 本文が含まれます。100 KB を超えるバイナリデータまたは JSON 本文は、指定された Amazon S3 バケットに、データプレフィックスを付けた個別のオブジェクトとしてアップロードされます。データは Amazon S3 Select と Amazon Athena を使用してクエリできるほか、 AWS Glue を使用して ETL 対象としてカタログ化することもできます。データは、 OpenSearch サービスにロードすることも、任意の Amazon EventBridge ターゲットで処理することもできます。
- CloudWatch ログの送信先 — JSON 呼び出しログイベントは、 CloudWatch Logs の指定されたロググループに配信されます。ログイベントでは、呼び出しメタデータと、最大サイズ 100 KB の入出力 JSON 本文が対象となります。大規模データ配信用の Amazon S3 ロケーションが提供された場合、バイナリデータまたは 100 KB を超える JSON 本文は、代わりにデータプレフィックスで

Amazon S3 バケットにアップロードされます。 データは、 CloudWatch Logs Insights を使用してクエリを実行し、 CloudWatch Logs を使用してさまざまな のサービスにリアルタイムでさらにストリーミングできます。

呼び出しのログ記録で API を使用する

モデル呼び出しのログ記録は次の API を使用して設定できます。

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

呼び出しのログ記録で API を使用する方法の詳細については、「Bedrock API ガイド」を参照してください。

Amazon で Amazon Bedrock をモニタリングする CloudWatch

Amazon Bedrock は CloudWatch、raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに処理する Amazon を使用してモニタリングできます。 CloudWatch コンソールを使用してメトリクスをグラフ化できます。また、特定のしきい値を超えないかどうかをモニタリングするアラームを設定するとともに、しきい値を超えたときに通知を送信したりアクションを実行したりできます。

詳細については、「Amazon ユーザーガイド CloudWatch」の「Amazon とは CloudWatch 」を参照してください。

トピック

ランタイムメトリクス

次の表は、Amazon Bedrock が提供するランタイムメトリックスについて説明しています。

メトリクス名	単位	説明
呼び出し	SampleCount	InvokeModel または InvokeModelWithResponseStream API オペレーションへのリクエストの数。
InvocationLatency	MilliSeconds	呼び出しのレイテンシー。
InvocationClientErrors	SampleCount	クライアント側でエラーが発生した呼び出しの数。
InvocationServerErrors	SampleCount	AWS サーバー側のエラーが発生した呼び出しの数。
InvocationThrottles	SampleCount	システムがスロットリングした呼び出しの数。
InputTokenCount	SampleCount	テキスト入力のトークンの数。
LegacyModelInvocations	SampleCount	レガシーモデルを使用した呼び出しの数
OutputTokenCount	SampleCount	テキスト出力のトークンの数。
OutputImageCount	SampleCount	出力画像の数。

ログ記録 CloudWatch メトリクス

配信の成功または失敗の試行ごとに、次の Amazon CloudWatch メトリクスが名前空間 、AWS/Bedrock、および Across all model IDsディメンションで出力されます。

- ModelInvocationLogsCloudWatchDeliverySuccess
- ModelInvocationLogsCloudWatchDeliveryFailure
- ModelInvocationLogsS3DeliverySuccess

- `ModelInvocationLogsS3DeliveryFailure`
- `ModelInvocationLargeDataS3DeliverySuccess`
- `ModelInvocationLargeDataS3DeliveryFailure`

アクセス許可の設定ミスや一時的な障害により口グが配信されない場合、配信は 24 時間後まで、定期的に再試行されます。

Amazon Bedrock の CloudWatch メトリクスを使用する

Amazon Bedrock オペレーションのメトリクスを取得するには、以下の情報を指定する必要があります。

- メトリクスディメンション。ディメンションは、メトリクスを識別するための名前と値のペアのセットです。Amazon Bedrock は、以下のディメンションをサポートしています。
  - `ModelId` - すべてのメトリクス
  - `ModelId + ImageSize + BucketedStepSize` – OutputImageCount
- メトリクス名 (`InvocationClientErrors` など)。

Amazon Bedrock のメトリクスは AWS Management Console、、 AWS CLI、または CloudWatch API を使用して取得できます。 CloudWatch API は、 AWS Software Development Kits (SDKsまたは CloudWatch API ツールのいずれかから使用できます。

で Amazon Bedrock CloudWatch をモニタリングするには、適切なアクセス CloudWatch 許可が必要です。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon の認証とアクセスコントロール CloudWatch」を参照してください。

Amazon Bedrock メトリクスを表示する

CloudWatch コンソールで Amazon Bedrock メトリクスを表示します。

メトリクスを表示するには (CloudWatch コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. メトリクス を選択し、すべてのメトリクス を選択し、 を検索しますModelId。

Amazon での Amazon Bedrock イベントのモニタリング EventBridge

Amazon を使用して EventBridge 、Amazon Bedrock でステータス変更イベントをモニタリングできます。Amazon では EventBridge、Amazon Bedrock のモデルカスタマイズジョブのステータス変更に自動的に応答 SageMaker するように Amazon を設定できます。Amazon Bedrock からのイベントは、ほぼリアルタイムで Amazon EventBridge に配信されます。イベントがルールに一致したときに自動的にアクションが実行されるようにする、シンプルなルールを作成できます。Amazon Bedrock EventBridge で Amazon を使用する場合は、次のことができます。

- 今後、新しい非同期ワークフローを追加するかどうかに関係なく、トリガーしたモデルカスタマイズで状態変更イベントが発生するたびに通知を発行する。その公開されたイベントでは、下流のワークフローのイベントに応答するのに十分な情報が提供されます。
- GetModelCustomizationJob API を呼び出すことなくジョブステータスの更新を配信します。これは、API レート制限の問題、API の更新、追加のコンピューティングリソースの削減を処理することを意味します。

Amazon から AWS イベントを受信するためのコストはかかりません EventBridge。Amazon の詳細については、「Amazon EventBridge EventBridge」を参照してください。

> **Note**
>
> - Amazon Bedrock は、ベストエフォートベースでイベントを発行します。イベントは、ほぼリアルタイムで Amazon EventBridge に配信されます。Amazon では EventBridge、イベントに応じてプログラムによるアクションをトリガーするルールを作成できます。例えば、SNS トピックを呼び出して E メール通知を送信したり、関数を呼び出して何らかのアクションを実行したりするルールを設定できます。詳細については、「Amazon ユーザーガイド EventBridge 」を参照してください。
> - Amazon Bedrock は、トリガーするモデルカスタマイズジョブの状態変更があるたびに新しいイベントを作成し、そのようなイベントをベストエフォートで配信します。

トピック

- 仕組み
- EventBridge スキーマ

- ルールとターゲット
- Amazon Bedrock イベントを処理するルールを作成する

仕組み

Amazon Bedrock からイベントを受信するには、Amazon を介して状態変更データを照合、受信、処理するルールとターゲットを作成する必要があります EventBridge。Amazon EventBridge は、 AWS サービス、SaaS パートナー、およびカスタマーアプリケーションから変更状態イベントを取り込むサーバーレスイベントバスです。作成したルールまたはパターンに基づいてイベントを処理し、これらのイベントを 、Amazon Simple Queue Service AWS Lambda、Amazon Simple Notification Service など、選択した 1 つ以上の「ターゲット」にルーティングします。

Amazon Bedrock は、モデルカスタマイズジョブの状態が変更される EventBridge たびに、Amazon 経由でイベントを発行します。いずれの場合も、新しいイベントが作成され、Amazon に送信され EventBridge、そのイベントがデフォルトのイベントバスに送信されます。このイベントには、どのカスタマイズジョブの状態が変更されたか、およびそのジョブの現在の状態が示されます。作成したルールに一致するイベントを Amazon が EventBridge 受信すると、Amazon は指定したターゲットに EventBridge ルーティングします。ルールを作成したら、それらのターゲットと下流のワークフローをイベントの内容に則して設定できます。

EventBridge スキーマ

イベントスキーマの以下の EventBridge イベントフィールドは、Amazon Bedrock に固有のものです。

- `jobArn` - モデルカスタマイズジョブの ARN。
- `outputModelArn` - 出力モデルの ARN。トレーニングジョブが完了したときに公開されます。
- `jobStatus` - ジョブの現在のステータス。
- `FailureMessage` - 失敗した場合のメッセージ。トレーニングジョブが失敗したときに公開されます。

イベント例

以下は、失敗したモデルカスタマイズジョブのイベント JSON の例です。

```
{
    "version": "0",
```

```
    "id": "UUID",
    "detail-type": "Model Customization Job State Change",
    "source": "aws.bedrock",
    "account": "123412341234",
    "time": "2023-08-11T12:34:56Z",
    "region": "us-east-1",
    "resources": [ "arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghwxyz" ],
    "detail": {
        "version": "0.0",
        "jobName": "abcd-wxyz",
        "jobArn": "arn:aws:bedrock:us-east-1:123412341234:model-customization-job/
abcdefghwxyz",
        "outputModelName": "dummy-output-model-name",
        "outputModelArn": "arn:aws:bedrock:us-east-1:123412341234:dummy-output-model-
name",
        "roleArn": "arn:aws:iam::123412341234:role/JobExecutionRole",
        "jobStatus": "Failed",
        "failureMessage": "Failure Message here.",
        "creationTime": "2023-08-11T10:11:12Z",
        "lastModifiedTime": "2023-08-11T12:34:56Z",
        "endTime": "2023-08-11T12:34:56Z",
        "baseModelArn": "arn:aws:bedrock:us-east-1:123412341234:base-model-name",
        "hyperParameters": {
            "batchSize" : "batchSizeNumberUsed",
            "epochCount": "epochCountNumberUsed",
            "learningRate": "learningRateUsed",
            "learningRateWarmupSteps": "learningRateWarmupStepsUsed"
            },
        "trainingDataConfig": {
            "s3Uri": "s3://bucket/key",
            },
        "validationDataConfig": {
            "s3Uri": "s3://bucket/key",
            },
        "outputDataConfig": {
            "s3Uri": "s3://bucket/key",
            }
    }
}
```

ルールとターゲット

作成したルールと一致する受信イベントは、そのルールに指定したターゲットにルーティングされ、ターゲットはこれらのイベントを処理します。ターゲットは JSON 形式をサポートし、Amazon EC2 インスタンス、Lambda 関数、Kinesis ストリーム、Amazon ECS タスク、Step Functions、Amazon SNS トピック、Amazon SQS などの AWS サービスを含めることができます。イベントを正しく受信して処理するには、イベントデータを一致、受信、および正しく処理するルールとターゲットを作成する必要があります。これらのルールとターゲットは、Amazon EventBridge コンソールまたは を使用して作成できます AWS CLI。

ルールの例

下記のルールは `source ["aws.bedrock"]` によって出力されるイベントパターンとの一致を行います。このルールは、ソース「aws.bedrock EventBridge 」を持つ Amazon によってデフォルトのイベントバスに送信されたすべてのイベントをキャプチャします。

```
{
    "source": ["aws.bedrock"]
}
```

Target

Amazon でルールを作成するときは EventBridge、 がルールパターンに一致するイベント EventBridge を送信するターゲットを指定する必要があります。これらのターゲットは、SageMaker パイプライン、Lambda 関数、SNS トピック、SQS キュー、または が EventBridge 現在サポートしている他のターゲットのいずれかです。イベントのターゲットを設定する方法については、Amazon EventBridge のドキュメントを参照してください。Amazon Simple Notification Service をターゲットとして使用する方法を示す手順については、「Amazon Bedrock イベントを処理するルールを作成する」を参照してください。

Amazon Bedrock イベントを処理するルールを作成する

Amazon Bedrock イベントに関する E メール通知を受け取るには、以下の手順を完了してください。

Amazon Simple Notification Service トピックを作成する

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。

2. ナビゲーションペインで、[トピック] を選択します。
3. [Create topic] (トピックの作成) を選択します。
4. [Type (タイプ)] で、[Standard (標準)] を選択します。
5. [Name] (名前) で、トピックの名前を入力します。
6. [Create topic] (トピックの作成) を選択します。
7. [サブスクリプションを作成] を選択します。
8. [Protocol (プロトコル)] として [Email (E メール)] を選択します。
9. [Endpoint] (エンドポイント) で、通知を受信するメールアドレスを入力します。
10. [サブスクリプションを作成] を選択します。
11. 次の件名の E メールメッセージが届きます: `AWS Notification - Subscription Confirmation`。指示に沿って操作し、登録を確認します。

Amazon Bedrock イベントを処理するルールを作成するには、次の手順を使用します。

Amazon Bedrock イベントを処理するルールを作成するには

1. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. [ルールの作成] を選択します。
3. [Name] (名前) に、ルールの名前を入力します。
4. ルールタイプ では、イベントパターンを持つルール] を選択します。
5. 次へ をクリックします。
6. [Event pattern] (イベントパターン) の場合は、次のいずれかを実行します。

    a. [イベントソース] で [AWS のサービス] を選択します。
    b. [AWS のサービス] で [Amazon Bedrock] を選択します。
    c. [イベントタイプ] で [モデルカスタマイズジョブの状態変更] を選択します。
    d. デフォルトでは、すべてのイベントに通知が送信されます。必要に応じて、特定のジョブ状態のイベントをフィルタリングするイベントパターンを作成できます。
    e. [次へ] を選択します。
7. 次のようにターゲットを指定します。

    a. [ターゲットタイプ] では、[AWS のサービス] を選択します。
    b. [Select a target] (ターゲットの選択) には、[SNS topic] (SNS トピック) を選択します。

c. [トピック] で、通知用に作成した SNS トピックを選択します。

d. [次へ] を選択します。

8. (オプション) ルールにタグを追加します。

9. [次へ] を選択します。

10. [Create rule] (ルールの作成) を選択します。

を使用した Amazon Bedrock API コールのログ記録 AWS CloudTrail

Amazon Bedrock は AWS CloudTrail、Amazon Bedrock のユーザー、ロール、または AWS のサービスによって実行されたアクションを記録するサービスである と統合されています。 は、Amazon Bedrock のすべての API コールをイベントとして CloudTrail キャプチャします。キャプチャされるコールには、Amazon Bedrock コンソールからのコールと、Amazon Bedrock API オペレーションへのコードコールが含まれます。証跡を作成する場合は、Amazon Bedrock の CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます。 Amazon S3 証跡を設定しない場合でも、コンソールのイベント履歴 で最新の CloudTrail イベントを表示できます。 で収集された情報を使用して CloudTrail、Amazon Bedrock に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「AWS CloudTrail ユーザーガイド」を参照してください。

の Amazon Bedrock 情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 は で有効になります。Amazon Bedrock でアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他の AWS サービスイベントとともに イベントに記録されます。 で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、「イベント履歴 を使用した CloudTrail イベントの表示」を参照してください。

Amazon Bedrock のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョン に適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたデータをより詳細に分析し、それに基づく対応を行うように他の AWS サービスを設定できます。詳細については、次を参照してください:

- 「追跡の作成の概要」
- CloudTrail サポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するために役立ちます。

- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザーの認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

の Amazon Bedrock データイベント CloudTrail

データイベントでは、リソース上またはリソース内で実行されるリソースオペレーション (Amazon S3 オブジェクトの読み取りまたは書き込みなど) についての情報が得られます。これらのイベントは、データプレーンオペレーションとも呼ばれます。データイベントは、多くの場合、デフォルトではログに CloudTrail 記録されない大量のアクティビティです。

Amazon Bedrock は Amazon Bedrock ランタイム API オペレーション (`InvokeModel`と`InvokeModelWithResponseStream`) を口グ記録しません。

Amazon Bedrock は、Amazon Bedrock ランタイム API オペレーションアクションのすべてのエージェントをデータイベント CloudTrail として に記録します。

- InvokeAgent 呼び出しを口グに記録するには、`AWS::Bedrock::AgentAlias`リソースタイプのデータイベントを記録するように高度なイベントセレクタを設定します。
- Retrieve および RetrieveAndGenerate呼び出しを口グに記録するには、`AWS::Bedrock::KnowledgeBase`リソースタイプのデータイベントを記録するように高度なイベントセレクタを設定します。

CloudTrail コンソールから、データイベントタイプ の Bedrock エージェントエイリアスまたは Bedrock ナレッジベースを選択します。さらに、カスタムログセレクタテンプレートを選択することで、eventName および resources.ARN フィールドをフィルタリングすることもできます。詳細については、「AWS マネジメントコンソールでデータイベントを log 記録する」を参照してください。

から AWS CLI、 のresource.type値を AWS::Bedrock::AgentAliasまたは に設定AWS::Bedrock::KnowledgeBaseし、 の値を eventCategoryに設定しますData。詳細については、「AWS CLI でのデータイベントの log 記録」を参照してください。

次の例では、 AWS CLI のすべての Amazon Bedrock リソースタイプに関する Amazon Bedrock データイベントをすべて log 記録する証跡を設定する方法を示します。

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

さらに、eventName および resources.ARN フィールドをフィルタリングすることもできます。フィールドの詳細については、「AdvancedFieldSelector」を参照してください。

追加の変更がイベントデータに適用されます。 CloudTrail 料金の詳細については、AWS CloudTrail「 の料金」を参照してください。

での Amazon Bedrock 管理イベント CloudTrail

管理イベントは、 AWS アカウントのリソースで実行される管理オペレーションに関する情報を提供します。これらは、デフォルトではコントロールプレーン operations. CloudTrail logs 管理イベント API オペレーションとも呼ばれます。

Amazon Bedrock は、残りの Amazon Bedrock API オペレーションを管理イベントとしてログ記録します。Amazon Bedrock が に記録する Amazon Bedrock API オペレーションのリストについては CloudTrail、Amazon Bedrock API リファレンスの以下のページを参照してください。

すべての Amazon Bedrock API オペレーションと Agents for Amazon Bedrock API オペレーションは、 によってログに記録 CloudTrail され、Amazon Bedrock API リファレンス に記載されています。例えば、InvokeModel、、および CreateAgentアクションを呼び出すとStopModelCustomizationJob、 CloudTrail ログファイルにエントリが生成されます。

Amazon Bedrock ログファイルエントリの概要

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意の送信元からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 InvokeModelアクションを示す CloudTrail ログエントリを示しています。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAICFHPEXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/userxyz",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "userxyz"
    },
    "eventTime": "2023-10-11T21:58:59Z",
    "eventSource": "bedrock.amazonaws.com",
    "eventName": "InvokeModel",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
```

```
    "userAgent": "Boto3/1.28.62 md/Botocore#1.31.62 ua/2.0 os/macos#22.6.0 md/
arch#arm64 lang/python#3.9.6 md/pyimpl#CPython cfg/retry-mode#legacy Botocore/1.31.62",
    "requestParameters": {
        "modelId": "stability.stable-diffusion-xl-v0"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "cipher suite",
        "clientProvidedHostHeader": "bedrock-runtime.us-west-2.amazonaws.com"
    }
}
```

SDK AWS を使用した Amazon Bedrock のコード例

以下のコード例は、 AWS ソフトウェア開発キット (SDK) で Amazon Bedrock を使用する方法を示しています。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードサンプル

- SDK AWS を使用した Amazon Bedrock のコード例
  - SDK を使用する AWS Amazon Bedrock のアクション
    - SDK を使用して Amazon Bedrock ファンデーションモデルの詳細を取得する AWS
    - SDK を使用して利用可能な Amazon Bedrock ファンデーションモデルを一覧表示する AWS
  - SDK を使用する AWS Amazon Bedrock のシナリオ
    - Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション
- SDK AWS を使用した Amazon Bedrock ランタイムのコード例
  - SDK を使用する AWS Amazon Bedrock ランタイムのアクション
    - Amazon Bedrock で Amazon Titan Image Generator G1 モデルを呼び出して画像を生成する
    - Amazon Bedrock で Stability.ai Stable Diffusion XL モデルを呼び出して画像を生成する
    - Amazon Bedrock でマルチモーダルプロンプトで Anthropic Claude 3 を呼び出す
    - Amazon Bedrock で AI21 Labs Jurassic-2 モデルを呼び出してテキストを生成する
    - Amazon Bedrock で Amazon Titan Text G1 モデルを呼び出してテキストを生成する
    - Amazon Bedrock で Anthropic Claude 2 モデルを呼び出してテキストを生成します。
    - Amazon Bedrock で Anthropic Claude 3 を呼び出してテキストを生成します。
    - テキスト生成用に Amazon Bedrock でアントロピック・クロード・インスタント・モデルを呼び出す
    - Amazon Bedrock で Anthropic Claude を呼び出して、レスポンスストリームを含むテキストを生成する
    - Amazon Bedrock で Meta Llama 2 Chat モデルを呼び出してテキストを生成する

- SDK を使用する AWS Amazon Bedrock ランタイムのエージェント用アクション
  - SDK を使用して Amazon Bedrock エージェントを呼び出す AWS
- SDK を使用する AWS Amazon Bedrock ランタイムのエージェントのシナリオ
  - Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK AWS を使用した Amazon Bedrock のコード例

以下のコード例は、 AWS ソフトウェア開発キット (SDK) で Amazon Bedrock を使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello Amazon Bedrock

次のコード例は、Amazon Bedrock の使用を開始する方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> まだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
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)}");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API ListFoundationModelsリファレンスのを参照してください。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
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)
    }
}
```

- API の詳細については、AWS SDK for Go API ListFoundationModelsリファレンスのを参照してください。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// 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();
}
```

- API の詳細については、AWS SDK for JavaScript API [ListFoundationModels](リファレンスの)を参照してください。

コードサンプル

SDK を使用する AWS Amazon Bedrock のアクション

以下のコード例は、 AWS SDK を使用して個々の Amazon Bedrock アクションを実行する方法を示しています。これらの抜粋は Amazon Bedrock API を呼び出し、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順が記載されたリンクが含まれています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、Amazon Bedrock API リファレンスをご覧ください。

例

- SDK を使用して Amazon Bedrock ファンデーションモデルの詳細を取得する AWS
- SDK を使用して利用可能な Amazon Bedrock ファンデーションモデルを一覧表示する AWS

SDK を使用して Amazon Bedrock ファンデーションモデルの詳細を取得する AWS

以下のコード例は、Amazon Bedrock ファンデーションモデルの詳細を取得する方法を示しています。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

同期 Amazon Bedrock クライアントを使用して基盤モデルの詳細を取得します。

```
   /**
    * 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);
        }
    }
```

非同期 Amazon Bedrock クライアントを使用して基盤モデルの詳細を取得します。

```
    /**
     * 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);
        }
    }
```

- API の詳細については、API GetFoundationModelリファレンスのを参照してください。AWS SDK for Java 2.x

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

基盤モデルに関する詳細を取得します。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API GetFoundationModelリファレンスの を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

基盤モデルに関する詳細を取得します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいGetFoundationModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して利用可能な Amazon Bedrock ファンデーションモデルを一覧表示する AWS

次のコード例は、利用可能な Amazon Bedrock 基盤モデルの一覧表示方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Bedrock ファンデーションの利用可能なモデルを一覧表示します。

```
        /// <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);
            }
        }
```

- API の詳細については、AWS SDK for .NET API ListFoundationModelsリファレンスのを参照してください。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Bedrock ファンデーションの利用可能なモデルを一覧表示します。

```
// 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
}
```

- API の詳細については、AWS SDK for Go API ListFoundationModelsリファレンスのを参照してください。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

同期 Amazon Bedrock クライアントを使用して、利用可能な Amazon Bedrock ファンデーションモデルを一覧表示します。

```
    /**
     * 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);
        }
    }
```

非同期 Amazon Bedrock クライアントを使用して、利用可能な Amazon Bedrock ファンデーションモデルを一覧表示します。

```
    /**
     * 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);
        }
    }
```

- API の詳細については、API リファレンスのを参照してくださいいListFoundationModels。AWS SDK for Java 2.x

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

使用可能な基盤モデルを一覧表示します。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API ListFoundationModelsリファレンスのを参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

利用可能な Amazon Bedrock 基盤モデルを一覧表示します。

```
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
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」のを参照してくださいListFoundationModels。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

利用可能な Amazon Bedrock 基盤モデルを一覧表示します。

```
    public function listFoundationModels()
    {
        $result = $this->bedrockClient->listFoundationModels();
        return $result;
    }
```

- API の詳細については、AWS SDK for PHP API ListFoundationModelsリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

利用可能な Amazon Bedrock 基盤モデルを一覧表示します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいListFoundationModels。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する AWS Amazon Bedrock のシナリオ

以下のコード例は、 AWS SDK を使用して Amazon Bedrock の一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon Bedrock 内の複数の関数を呼び出して特定のタスクを実行する方法を示しています。各シナリオには GitHub、コードの設定と実行の手順が記載されたリンクが含まれています。

例

- Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

次のコード例は、Amazon Bedrock と Step Functions を使用してジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。

Python

SDK for Python (Boto3)

Amazon Bedrock サーバーレスプロンプトチェーンシナリオではAWS Step Functions、Amazon Bedrock と Agents for Amazon Bedrock を使用して、複雑でサーバーレスでスケーラブルなジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。以下の実例が含まれています。

- ある小説の分析を文学ブログに書く。この例は、単純で連続した一連のプロンプトを示しています。
- 特定のトピックに関する短いストーリーを生成します。この例は、AI が以前に生成した項目のリストを反復的に処理する方法を示しています。
- 特定の目的地への週末休暇の旅程を作成します。この例は、複数の異なるプロンプトを並列化する方法を示しています。
- 映画製作者を務める人間のユーザーに、映画のアイデアを売り込んでください。この例は、同じプロンプトを異なる推論パラメータで並列化する方法、チェーン内の前のステップに戻る方法、ワークフローに人間の入力を含める方法を示しています。
- ユーザーが手元に持っている食材に基づいて食事を計画します。この例は、プロンプトチェーンに 2 つの異なる AI 会話を組み込み、2 つの AI ペルソナが互いに議論を交わして最終的な結果を改善する方法を示しています。
- GitHub 現在最もトレンドが高いリポジトリを見つけて要約してください。この例は、外部 API と相互作用する複数の AI エージェントをチェーン接続する方法を示しています。

完全なソースコードとセットアップと実行の手順については、のプロジェクト全体を参照してください。GitHub

この例で使用されているサービス

- Amazon Bedrock
- Amazon Bedrock ランタイム
- Agents for Amazon Bedrock
- Amazon Bedrock ランタイム用エージェント
- Step Functions

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK AWS を使用した Amazon Bedrock ランタイムのコード例

以下のコード例は、 AWS ソフトウェア開発キット (SDK) で Amazon Bedrock ランタイムを使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello Amazon Bedrock

次のコード例は、Amazon Bedrock の使用を開始する方法を示しています。

Go

SDK for Go V2

> **Note**
>
> まだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
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)
}
```

- API の詳細については、AWS SDK for Go API InvokeModelリファレンスのを参照してください。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// 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} oputput_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();
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeModelリファレンスのを参照してください。

コードサンプル

- SDK を使用する AWS Amazon Bedrock ランタイムのアクション
  - Amazon Bedrock で Amazon Titan Image Generator G1 モデルを呼び出して画像を生成する
  - Amazon Bedrock で Stability.ai Stable Diffusion XL モデルを呼び出して画像を生成する
  - Amazon Bedrock でマルチモーダルプロンプトで Anthropic Claude 3 を呼び出す
  - Amazon Bedrock で AI21 Labs Jurassic-2 モデルを呼び出してテキストを生成する
  - Amazon Bedrock で Amazon Titan Text G1 モデルを呼び出してテキストを生成する
  - Amazon Bedrock で Anthropic Claude 2 モデルを呼び出してテキストを生成します。
  - Amazon Bedrock で Anthropic Claude 3 を呼び出してテキストを生成します。
  - テキスト生成用に Amazon Bedrock でアントロピック・クロード・インスタント・モデルを呼び出す
  - Amazon Bedrock で Anthropic Claude を呼び出して、レスポンスストリームを含むテキストを生成する
  - Amazon Bedrock で Meta Llama 2 Chat モデルを呼び出してテキストを生成する
  - Amazon Bedrock で Mistral 7B モデルを呼び出してテキストを生成する
  - Amazon Bedrock で Mixtral 8x7B モデルを呼び出してテキストを生成する
- SDK を使用する AWS Amazon Bedrock ランタイムのシナリオ

- SDK を使用して Amazon Bedrock ファンデーションモデルとやり取りするためのプレイグラウンドを提供するサンプルアプリケーションを作成します。 AWS
- Amazon Bedrock 用のインタラクティブなテキスト生成プレイグラウンド
- Amazon Bedrock で複数の大規模言語モデル (LLM) を呼び出す
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK を使用する AWS Amazon Bedrock ランタイムのアクション

以下のコード例は、 AWS SDK を使用して個々の Amazon Bedrock ランタイムアクションを実行する方法を示しています。これらの抜粋は Amazon Bedrock Runtime API を呼び出し、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順が記載されたリンクが含まれています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、Amazon Bedrock ランタイム API リファレンスをご覧ください。

例

- Amazon Bedrock で Amazon Titan Image Generator G1 モデルを呼び出して画像を生成する
- Amazon Bedrock で Stability.ai Stable Diffusion XL モデルを呼び出して画像を生成する
- Amazon Bedrock でマルチモーダルプロンプトで Anthropic Claude 3 を呼び出す
- Amazon Bedrock で AI21 Labs Jurassic-2 モデルを呼び出してテキストを生成する
- Amazon Bedrock で Amazon Titan Text G1 モデルを呼び出してテキストを生成する
- Amazon Bedrock で Anthropic Claude 2 モデルを呼び出してテキストを生成します。
- Amazon Bedrock で Anthropic Claude 3 を呼び出してテキストを生成します。
- テキスト生成用に Amazon Bedrock でアントロピック・クロード・インスタント・モデルを呼び出す
- Amazon Bedrock で Anthropic Claude を呼び出して、レスポンスストリームを含むテキストを生成する
- Amazon Bedrock で Meta Llama 2 Chat モデルを呼び出してテキストを生成する
- Amazon Bedrock で Mistral 7B モデルを呼び出してテキストを生成する
- Amazon Bedrock で Mixtral 8x7B モデルを呼び出してテキストを生成する

Amazon Bedrock で Amazon Titan Image Generator G1 モデルを呼び出して画像を生成する

以下のコード例は、Amazon Bedrock で Amazon Titan Image Generator G1 モデルを呼び出して画像を生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan イメージジェネレーター G1 基盤モデルを非同期的に呼び出してイメージを生成します。

```
        /// <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"]?.AsArray();

                    return results?[0]?.GetValue<string>();
                }
                else
                {
                    Console.WriteLine("InvokeModelAsync failed with status code "
 + response.HttpStatusCode);
                }
            }
            catch (AmazonBedrockRuntimeException e)
            {
                Console.WriteLine(e.Message);
            }
            return null;
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for .NET

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Image Generator G1 モデルを呼び出して画像を生成します。

```
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

}
```

- API の詳細については、AWS SDK for Go API InvokeModelリファレンスのを参照してください。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Image Generator G1 モデルを非同期で呼び出して画像を生成します。

```
    /**
     * 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;
    }
```

Amazon Titan Image Generator G1 モデルを呼び出して画像を生成します。

```
        /**
         * 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;
        }
```

- API の詳細については、AWS SDK for Java 2.x API InvokeModelリファレンスのを参照してください。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Image Generator G1 モデルを呼び出して画像を生成します。

```
    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;
    }
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Image Generator G1 モデルを呼び出して画像を生成します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Stability.ai Stable Diffusion XL モデルを呼び出して画像を生成する

次のコード例は、Amazon Bedrock で Stability.ai Stable Diffusion XL モデルを呼び出して画像を生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Stability.ai Stable Diffusion XL ファンデーションモデルを非同期的に呼び出して画像を生成します。

```
        /// <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;
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Stability.ai Stable Diffusion XL 基盤モデルを非同期で呼び出して画像を生成します。

```
    /**
     * 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;
    }
```

Stability.ai Stable Diffusion XL 基盤モデルを呼び出して画像を生成します。

```
        /**
         * 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;
        }
```

- API の詳細については、AWS SDK for Java 2.x API InvokeModelリファレンスのを参照してください。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Stability.ai Stable Diffusion XL 基盤モデルを呼び出して画像を生成します。

```
    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;
    }
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Stability.ai Stable Diffusion XL 基盤モデルを呼び出して画像を生成します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してください[InvokeModel](#)。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、[AWS コードサンプルリポジトリ](#)での設定と実行の方法を確認してください。

Stability.ai Stable Diffusion XL 基盤モデルを呼び出して画像を生成します。

```
    "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": "iVBORw0KGgoAAAANSUhEUgAAAgAAA....
*             "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.
```

Stability.ai Stable Diffusion XL 基盤モデルを呼び出し、L2 高レベルクライアントを使用してイメージを生成します。

```
    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.
```

- API の詳細については、AWS SDK for SAP ABAP API InvokeModelリファレンスのを参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock でマルチモーダルプロンプトで Anthropic Claude 3 を呼び出す

以下のコード例は、Amazon Bedrock で Anthropic Claude 3 をマルチモーダルプロンプトで呼び出す方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

マルチモーダルプロンプトで Anthropic Claude 3 を呼び出し、画像を分析します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で AI21 Labs Jurassic-2 モデルを呼び出してテキストを生成する

次のコード例は、Amazon Bedrock で AI21 Labs Jurassic-2 モデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを非同期で呼び出します。

```
        /// <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"]?
                        .AsArray()[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;
        }
```

- API の詳細については、AWS SDK for .NET API InvokeModelリファレンスのを参照してください。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを呼び出して、テキストを生成します。

```
// 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
}
```

- API の詳細については、AWS SDK for Go API InvokeModelリファレンスのを参照してください。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを非同期で呼び出してテキストを生成します。

```
    /**
     * 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;
    }
```

AI21 Labs Jurassic-2 基盤モデルを呼び出して、テキストを生成します。

```
        /**
         * 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;
        }
```

- API の詳細については、AWS SDK for Java 2.x API InvokeModelリファレンスのを参照してください。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを呼び出して、テキストを生成します。

```
// 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);
  }
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeModelリファレンスのを参照してください。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを呼び出して、テキストを生成します。

```
    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;
    }
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

AI21 Labs Jurassic-2 基盤モデルを呼び出して、テキストを生成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Amazon Titan Text G1 モデルを呼び出してテキストを生成する

以下のコード例は、Amazon Bedrock で Amazon Titan テキスト G1 モデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> まだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Text G1 ファンデーションモデルを非同期的に呼び出してテキストを生成します。

```
        /// <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"]?.AsArray();

                    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;
        }
```

- API の詳細については、「API リファレンス」のを参照してください。InvokeModelAWS SDK for .NET

JavaScript

JavaScript (v3) 用 SDK

> **Note**
>
> まだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Titan Text G1 ファンデーションモデルを呼び出してテキストを生成します。

```
// 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);
  }
}
```

- API の詳細については、API InvokeModelリファレンスのを参照してください。AWS SDK for JavaScript

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Anthropic Claude 2 モデルを呼び出してテキストを生成します。

次のコード例は、Amazon Bedrock で Anthropic Claude 2 モデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを非同期で呼び出してテキストを生成します。

```
        /// <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;
        }
```

- API の詳細については、AWS SDK for .NET API InvokeModelリファレンスのを参照してください。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。

```
// 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
}
```

- API の詳細については、AWS SDK for Go API InvokeModelリファレンスのを参照してください。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを非同期で呼び出してテキストを生成します。

```
    /**
     * 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;
    }
```

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。

```
        /**
         * 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;
        }
```

- API の詳細については、AWS SDK for Java 2.x API InvokeModelリファレンスのを参照してください。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。

```
// 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);
  }
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeModelリファレンスのを参照してください。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。

```
    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;
    }
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してください[InvokeModel](#)。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、[AWS コードサンプルリポジトリ](#)での設定と実行の方法を確認してください。

Anthropic Claude 2 基盤モデルを呼び出して、テキストを生成します。この例では、バージョンによっては利用できない可能性のある /US2/CL_JSON の機能を使用しています。

NetWeaver

```
    "Claude V2 Input Parameters should be in a format like this:
*   {
*     "prompt":"\n\nHuman:\\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{ 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.
```

Anthropic Claude 2 基盤モデルを呼び出し、L2 高レベルクライアントを使用してテキストを生成します。

```
    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.
```

- API の詳細については、SDK for SAP ABAP API InvokeModelリファレンスのを参照してください。AWS

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Anthropic Claude 3 を呼び出してテキストを生成します。

以下のコード例は、Amazon Bedrock で Anthropic Claude 3 を呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> まだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

人為的クロード 3 を呼び出してテキストを生成します。

```
// 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}`);
  }
}
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for JavaScript

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

人為的クロード 3 を呼び出してテキストを生成します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

テキスト生成用に Amazon Bedrock でアントロピック・クロード・インスタント・モデルを呼び出す

次のコード例は、Amazon Bedrock で Anthropic Claude インスタントモデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

アントロピック・クロード・インスタント・ファンデーション・モデルを呼び出してテキストを生成します。

```
// 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);
  }
}
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for JavaScript

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Anthropic Claude を呼び出して、レスポンスストリームを含むテキストを生成する

以下のコード例は、Amazon Bedrock で Anthropic Claude モデルを呼び出して、レスポンスストリームを使用してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- Amazon Bedrock で複数の基盤モデルを呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude を非同期的に呼び出して、応答ストリームを処理します。

```
        /// <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 IAsyncEnumerable<string>
InvokeClaudeWithResponseStreamAsync(string prompt, [EnumeratorCancellation]
CancellationToken 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;
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelWithResponseStreamAWS SDK for .NET

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude を呼び出し、レスポンスストリームを処理します。

```
// 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
 }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelWithResponseStreamAWS SDK for Go

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude を呼び出し、レスポンスストリームを処理します。

```
        /**
         * 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();
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelWithResponseStreamAWS SDK for Java 2.x

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude を呼び出し、レスポンスストリームを処理します。

```
// 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}`);
  }
}
```

- API の詳細については、API リファレンスのを参照してくださ
  い。InvokeModelWithResponseStreamAWS SDK for JavaScript

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Anthropic Claude を呼び出し、レスポンスストリームを処理します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してください[InvokeModelWithResponseStream](#)。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。[このサービスを AWS SDK で使用する](#)このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Meta Llama 2 Chat モデルを呼び出してテキストを生成する

次のコード例は、Amazon Bedrock で Meta Llama 2 Chat モデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [インタラクティブテキストプレイグラウンド](#)
- [Amazon Bedrock で複数の基盤モデルを呼び出す](#)
- [Amazon Bedrock で複数の LLM を呼び出す](#)

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 基盤モデルを非同期で呼び出してテキストを生成します。

```
        /// <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;
        }
```

- API の詳細については、AWS SDK for .NET API InvokeModelリファレンスのを参照してください。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 Chat 基盤モデルを呼び出してテキストを生成します。

```
// 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
}
```

- API の詳細については、AWS SDK for Go API InvokeModelリファレンスのを参照してください。

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 Chat 基盤モデルを非同期で呼び出してテキストを生成します。

```
    /**
     * 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;
    }
```

Meta Llama 2 Chat 基盤モデルを呼び出してテキストを生成します。

```
        /**
         * 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;
        }
```

- API の詳細については、AWS SDK for Java 2.x API InvokeModelリファレンスのを参照してください。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 Chat 基盤モデルを呼び出してテキストを生成します。

```
// 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 {generation} 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);
  }
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeModelリファレンスのを参照してください。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 Chat 基盤モデルを呼び出してテキストを生成します。

```
    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;
    }
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Meta Llama 2 Chat 基盤モデルを呼び出してテキストを生成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Mistral 7B モデルを呼び出してテキストを生成する

以下のコード例は、Amazon Bedrock で Mistral 7B モデルモデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mistral 7B 基盤モデルを非同期的に呼び出してテキストを生成します。

```
        /// <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"]?.AsArray();

                    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 ?? [];
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mistral 7B 基盤モデルを非同期的に呼び出してテキストを生成します。

```
    /**
     * 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();
    }
```

Mistral 7B ファンデーションモデルを呼び出してテキストを生成します。

```
        /**
         * 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();

        }
```

- API の詳細については、API リファレンスのを参照してくださいInvokeModel。AWS SDK for Java 2.x

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mistral 7B ファンデーションモデルを呼び出してテキストを生成します。

```
// 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 = `<s>[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);
  }
}
```

- API の詳細については、API リファレンスのを参照してくださいInvokeModel。AWS SDK for JavaScript

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mistral 7B ファンデーションモデルを呼び出してテキストを生成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で Mixtral 8x7B モデルを呼び出してテキストを生成する

以下のコード例は、Amazon Bedrock で Mixtral 8x7B モデルモデルを呼び出してテキストを生成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- インタラクティブテキストプレイグラウンド
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock で複数の LLM を呼び出す

.NET

AWS SDK for .NET

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mixtral 8x7B 基盤モデルを非同期的に呼び出してテキストを生成します。

```
        /// <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"]?.AsArray();

                    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 ?? [];
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for .NET

Java

SDK for Java 2.x

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mistral 8x7B 基盤モデルを非同期的に呼び出してテキストを生成します。

```
   /**
    * 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();
    }
```

Mixtral 8x7B ファンデーションモデルを呼び出してテキストを生成します。

```
        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();
        }
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for Java 2.x

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mixtral 8x7B ファンデーションモデルを呼び出してテキストを生成します。

```
// 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 = `<s>[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);
  }
}
```

- API の詳細については、API リファレンスのを参照してください。InvokeModelAWS SDK for JavaScript

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Mixtral 8x7B ファンデーションモデルを呼び出してテキストを生成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいInvokeModel。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する AWS Amazon Bedrock ランタイムのシナリオ

以下のコード例は、 AWS SDK を使用して Amazon Bedrock ランタイムの一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon Bedrock Runtime 内で複数の関数を呼び出して特定のタスクを実行する方法を示しています。各シナリオには GitHub、コードの設定と実行方法に関する説明が記載されたリンクが含まれています。

例

- SDK を使用して Amazon Bedrock ファンデーションモデルとやり取りするためのプレイグラウンドを提供するサンプルアプリケーションを作成します。 AWS
- Amazon Bedrock 用のインタラクティブなテキスト生成プレイグラウンド
- Amazon Bedrock で複数の大規模言語モデル (LLM) を呼び出す
- Amazon Bedrock で複数の基盤モデルを呼び出す
- Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK を使用して Amazon Bedrock ファンデーションモデルとやり取りするためのプレイグラウンドを提供するサンプルアプリケーションを作成します。 AWS

次のコード例は、さまざまな方法で Amazon Bedrock 基盤モデルと相互作用するプレイグラウンドを作成する方法を示しています。

.NET

AWS SDK for .NET

.NET 基盤モデル (FM) プレイグラウンドは、C# コードから Amazon Bedrock を使用する方法を紹介する.NET MAUI Blazor サンプルアプリケーションです。この例は、.NET 開発者と C# 開発者が Amazon Bedrock を使用して生成 AI 対応アプリケーションを構築する方法を示しています。次の 4 つのプレイグラウンドを使用して Amazon Bedrock 基盤モデルをテストしたり操作したりできます。

- テキストプレイグラウンド。
- チャットプレイグラウンド。

- ボイスチャットプレイグラウンド。
- イメージプレイグラウンド。

この例には、アクセスできる基盤モデルとその特性も一覧表示されています。ソースコードとデプロイ手順については、のプロジェクトを参照してくださいGitHub。

この例で使用されているサービス

- Amazon Bedrock ランタイム

Java

SDK for Java 2.x

Java 基盤モデル (FM) プレイグラウンドは Spring Boot のサンプルアプリケーションで、Java で Amazon Bedrock を使用する方法を紹介しています。この例は、Java 開発者が Amazon Bedrock を使用して生成 AI 対応アプリケーションを構築する方法を示しています。次の 3 つのプレイグラウンドを使用して Amazon Bedrock 基盤モデルをテストしたり操作したりできます。

- テキストプレイグラウンド。
- チャットプレイグラウンド。
- イメージプレイグラウンド。

この例には、アクセスできる基盤モデルとその特性が一覧表示されています。ソースコードとデプロイ手順については、のプロジェクトを参照してくださいGitHub。

この例で使用されているサービス

- Amazon Bedrock ランタイム

Python

SDK for Python (Boto3)

Python 基盤モデル (FM) プレイグラウンドは Python/FastAPI のサンプルアプリケーションで、Python で Amazon Bedrock を使用する方法を紹介しています。この例は、Python 開発者が Amazon Bedrock を使用して生成 AI 対応アプリケーションを構築する方法を示しています。次の 3 つのプレイグラウンドを使用して Amazon Bedrock 基盤モデルをテストしたり操作したりできます。

- テキストプレイグラウンド。

- チャットプレイグラウンド。
- イメージプレイグラウンド。

この例には、アクセスできる基盤モデルとその特性が一覧表示されています。ソースコードとデプロイ手順については、のプロジェクトを参照してくださいGitHub。

この例で使用されているサービス

- Amazon Bedrock ランタイム

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock 用のインタラクティブなテキスト生成プレイグラウンド

以下のコード例は、Amazon Bedrock の以下の大規模言語モデル (LLM) にプロンプトを準備して送信する方法を示しています。

- AI21 ラボ:ジュラシック2 ミッド/ウルトラ
- Amazon: タイタンテキスト G1 ライトアンドエクスプレス
- アントロピック:クロード・インスタント
- アントロピック:クロード 2.0 と 2.1
- 人類学:クロード3世俳句とソネット
- メタ:ラマ2 チャット 13B と 70B
- ミストラル AI: ミストラル 7B とミストラル 8x7B

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// 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();
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeModelリファレンスのを参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で複数の大規模言語モデル (LLM) を呼び出す

以下のコード例は、Amazon Bedrock で複数 large-language-models (LLM) を呼び出す方法を示しています。

- Anthropic Claude を使用してテキストを生成します。
- AI21 Labs Jurassic-2 を使用してテキストを生成します。
- Meta Llama 2 Chat を使用してテキストを生成します。

PHP

SDK for PHP

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Bedrock で複数の LLM を呼び出します。

```
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;
    }
}
```

- API の詳細については、AWS SDK for PHP API InvokeModelリファレンスのを参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock で複数の基盤モデルを呼び出す

次のコード例は、Amazon Bedrock で複数の基盤モデルを呼び出す方法を示しています。

- Anthropic Claude を使用してテキストを生成します。
- AI21 Labs Jurassic-2 を使用してテキストを生成します。

- Meta Llama 2 Chat を使用してテキストを生成します。
- Anthropic Claude からのレスポンスストリームを非同期で処理します。
- Amazon Titan Image Generator を使用して画像を生成します。

Go

SDK for Go V2

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

Amazon Bedrock で複数の基盤モデルを呼び出します。

```
// 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)
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
  - InvokeModel
  - InvokeModelWithResponseStream

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

次のコード例は、Amazon Bedrock と Step Functions を使用してジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。

Python

SDK for Python (Boto3)

Amazon Bedrock サーバーレスプロンプトチェーンシナリオではAWS Step Functions、Amazon Bedrock と Agents for Amazon Bedrock を使用して、複雑でサーバーレスでスケーラブルなジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。以下の実例が含まれています。

- ある小説の分析を文学ブログに書く。この例は、単純で連続した一連のプロンプトを示しています。
- 特定のトピックに関する短いストーリーを生成します。この例は、AI が以前に生成した項目のリストを反復的に処理する方法を示しています。
- 特定の目的地への週末休暇の旅程を作成します。この例は、複数の異なるプロンプトを並列化する方法を示しています。
- 映画製作者を務める人間のユーザーに、映画のアイデアを売り込んでください。この例は、同じプロンプトを異なる推論パラメータで並列化する方法、チェーン内の前のステップに戻る方法、ワークフローに人間の入力を含める方法を示しています。
- ユーザーが手元に持っている食材に基づいて食事を計画します。この例は、プロンプトチェーンに 2 つの異なる AI 会話を組み込み、2 つの AI ペルソナが互いに議論を交わして最終的な結果を改善する方法を示しています。
- GitHub 現在最もトレンドが高いリポジトリを見つけて要約してください。この例は、外部 API と相互作用する複数の AI エージェントをチェーン接続する方法を示しています。

完全なソースコードとセットアップと実行の手順については、のプロジェクト全体を参照してください。GitHub

この例で使用されているサービス

- Amazon Bedrock
- Amazon Bedrock ランタイム
- Agents for Amazon Bedrock
- Amazon Bedrock ランタイム用エージェント
- Step Functions

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する AWS Amazon Bedrock 用エージェントのコード例

以下のコード例は、 AWS ソフトウェア開発キット (SDK) で Amazon Bedrock 用エージェントを使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Amazon ベッドロックのハローエージェント

次のコード例は、Amazon Bedrock 用エージェントの使用を開始する方法を示しています。

JavaScript

JavaScript (v3) 用 SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// 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();
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - GetAgent
  - ListAgents

コードサンプル

- SDK を使用する AWS Amazon Bedrock 用エージェントのアクション
  - SDK を使用して Amazon Bedrock エージェントを作成する AWS
  - SDK を使用して Amazon Bedrock エージェントアクショングループを作成する AWS
  - SDK を使用して Amazon Bedrock エージェントエイリアスを作成する AWS

- SDK を使用して Amazon Bedrock エージェントを削除する AWS
- SDK を使用して Amazon Bedrock エージェントエイリアスを削除する AWS
- SDK を使用して Amazon Bedrock エージェントに関する情報を取得する AWS
- SDK を使用して Amazon Bedrock エージェントのアクショングループを一覧表示する AWS
- SDK を使用しているアカウントに属する Amazon Bedrock のエージェントを一覧表示する AWS
- SDK を使用する Amazon Bedrock エージェントに関連するナレッジベースを一覧表示する AWS
- SDK を使用して Amazon Bedrock エージェントを準備する AWS

- SDK を使用する AWS Amazon Bedrock 用エージェントのシナリオ
  - SDK を使用して Amazon Bedrock end-to-end エージェントを作成および呼び出す方法を示す例 AWS
  - Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK を使用する AWS Amazon Bedrock 用エージェントのアクション

以下のコード例は、 AWS SDK を使用して Amazon Bedrock アクションの個々のエージェントを実行する方法を示しています。これらの抜粋は Agents for Amazon Bedrock API を呼び出したもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順が記載されたリンクが含まれています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、Amazon Bedrock API リファレンスのエージェントをご覧ください。

例

- SDK を使用して Amazon Bedrock エージェントを作成する AWS
- SDK を使用して Amazon Bedrock エージェントアクショングループを作成する AWS
- SDK を使用して Amazon Bedrock エージェントエイリアスを作成する AWS
- SDK を使用して Amazon Bedrock エージェントを削除する AWS
- SDK を使用して Amazon Bedrock エージェントエイリアスを削除する AWS
- SDK を使用して Amazon Bedrock エージェントに関する情報を取得する AWS
- SDK を使用して Amazon Bedrock エージェントのアクショングループを一覧表示する AWS

- SDK を使用しているアカウントに属する Amazon Bedrock のエージェントを一覧表示する AWS
- SDK を使用する Amazon Bedrock エージェントに関連するナレッジベースを一覧表示する AWS
- SDK を使用して Amazon Bedrock エージェントを準備する AWS

SDK を使用して Amazon Bedrock エージェントを作成する AWS

以下のコード例は、Amazon Bedrock エージェントを作成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを作成します。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API CreateAgentリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを作成します。

```
    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"]
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいCreateAgent。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントアクショングループを作成する AWS

次のコード例は、Amazon Bedrock エージェントアクショングループを作成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントアクショングループを作成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいCreateAgentActionGroup。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントエイリアスを作成する AWS

次のコード例は、Amazon Bedrock エージェントエイリアスを作成する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントエイリアスを作成します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいCreateAgentAlias。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントを削除する AWS

以下のコード例は、Amazon Bedrock エージェントを削除する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを削除します。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API DeleteAgentリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを削除します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいDeleteAgent。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントエイリアスを削除する AWS

次のコード例は、Amazon Bedrock エージェントエイリアスを削除する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントエイリアスを削除します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいDeleteAgentAlias。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントに関する情報を取得する AWS

以下のコード例は、Amazon Bedrock エージェントに関する情報を取得する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを取得します。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API GetAgentリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを取得します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいGetAgent。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントのアクショングループを一覧表示する AWS

以下のコード例は、Amazon Bedrock エージェントのアクショングループを一覧表示する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントのアクショングループを一覧表示します。

```
// 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);
  }
}
```

- API の詳細については、AWS SDK for JavaScript API ListAgentActionGroupsリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントのアクショングループを一覧表示します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいListAgentActionGroups。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用しているアカウントに属する Amazon Bedrock のエージェントを一覧表示する AWS

以下のコード例は、アカウントに属する Amazon Bedrock 用エージェントを一覧表示する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

アカウントに属するエージェントを一覧表示します。

```
// 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);
  }
}
```

- API の詳細については、AWS SDK for JavaScript API ListAgentsリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

アカウントに属するエージェントを一覧表示します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいListAgents。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する Amazon Bedrock エージェントに関連するナレッジベースを一覧表示する AWS

次のコード例は、Amazon Bedrock エージェントに関連するナレッジベースを一覧表示する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントに関連するナレッジベースを一覧表示します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいListAgentKnowledgeBases。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用して Amazon Bedrock エージェントを準備する AWS

次のコード例は、Amazon Bedrock エージェントを内部テスト用に準備する方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- エージェントを作成して呼び出す

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

内部テスト用のエージェントを準備します。

```
    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
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』のを参照してくださいPrepareAgent。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する AWS Amazon Bedrock 用エージェントのシナリオ

以下のコード例は、 AWS SDK を使用して Amazon Bedrock 用エージェントの一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Agents for Amazon Bedrock 内の複数の関数を呼び出して特定のタスクを実行する方法を示しています。各シナリオには GitHub、コードの設定と実行方法に関する説明が記載されたリンクが含まれています。

例

- SDK を使用して Amazon Bedrock end-to-end エージェントを作成および呼び出す方法を示す例 AWS
- Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK を使用して Amazon Bedrock end-to-end エージェントを作成および呼び出す方法を示す例 AWS

次のコードサンプルは、以下の操作方法を示しています。

- エージェントの実行ロールを作成します。
- エージェントを作成し、ドラフトバージョンをデプロイします。
- エージェントの機能を実装する Lambda 関数を作成します。
- エージェントを Lambda 関数に接続するアクショングループを作成します。
- 完全に設定されたエージェントをデプロイします。
- ユーザー指定のプロンプトでエージェントを呼び出します。
- すべての作成されたリソースを削除します。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを作成して呼び出します。

```
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}")
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - CreateAgent
  - CreateAgentActionGroup
  - CreateAgentAlias
  - DeleteAgent
  - DeleteAgentAlias
  - GetAgent
  - ListAgentActionGroups
  - ListAgentKnowledgeBases
  - ListAgents
  - PrepareAgent

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

次のコード例は、Amazon Bedrock と Step Functions を使用してジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。

Python

SDK for Python (Boto3)

Amazon Bedrock サーバーレスプロンプトチェーンシナリオではAWS Step Functions、Amazon Bedrock と Agents for Amazon Bedrock を使用して、複雑でサーバーレスでスケーラブルなジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。以下の実例が含まれています。

- ある小説の分析を文学ブログに書く。この例は、単純で連続した一連のプロンプトを示しています。
- 特定のトピックに関する短いストーリーを生成します。この例は、AI が以前に生成した項目のリストを反復的に処理する方法を示しています。
- 特定の目的地への週末休暇の旅程を作成します。この例は、複数の異なるプロンプトを並列化する方法を示しています。
- 映画製作者を務める人間のユーザーに、映画のアイデアを売り込んでください。この例は、同じプロンプトを異なる推論パラメータで並列化する方法、チェーン内の前のステップに戻る方法、ワークフローに人間の入力を含める方法を示しています。
- ユーザーが手元に持っている食材に基づいて食事を計画します。この例は、プロンプトチェーンに 2 つの異なる AI 会話を組み込み、2 つの AI ペルソナが互いに議論を交わして最終的な結果を改善する方法を示しています。
- GitHub 現在最もトレンドが高いリポジトリを見つけて要約してください。この例は、外部 API と相互作用する複数の AI エージェントをチェーン接続する方法を示しています。

完全なソースコードとセットアップと実行の手順については、のプロジェクト全体を参照してください。GitHub

この例で使用されているサービス

- Amazon Bedrock

- Amazon Bedrock ランタイム
- Agents for Amazon Bedrock
- Amazon Bedrock ランタイム用エージェント
- Step Functions

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK AWS を使用した Amazon Bedrock ランタイムのエージェントのコード例

以下のコード例は、 AWS ソフトウェア開発キット (SDK) で Amazon Bedrock ランタイム用エージェントを使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードサンプル

- SDK を使用する AWS Amazon Bedrock ランタイムのエージェント用アクション
  - SDK を使用して Amazon Bedrock エージェントを呼び出す AWS
- SDK を使用する AWS Amazon Bedrock ランタイムのエージェントのシナリオ
  - Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

SDK を使用する AWS Amazon Bedrock ランタイムのエージェント用アクション

以下のコード例は、 AWS SDK を使用して Amazon Bedrock ランタイムアクションの個々のエージェントを実行する方法を示しています。これらの抜粋は Amazon Bedrock Runtime API 用のエージェントを呼び出し、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順が記載されたリンクが含まれています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、Amazon Bedrock ランタイム API リファレンスのエージェントをご覧ください。

例

- SDK を使用して Amazon Bedrock エージェントを呼び出す AWS

SDK を使用して Amazon Bedrock エージェントを呼び出す AWS

以下のコード例は、Amazon Bedrock エージェントを呼び出す方法を示しています。

JavaScript

JavaScript (v3) 用の SDK

> **Note**
>
> にはまだまだあります。 GitHub用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// 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);
}
```

- API の詳細については、AWS SDK for JavaScript API InvokeAgentリファレンスのを参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> にはまだまだあります GitHub。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

エージェントを呼び出します。

```
    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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスのを参照してくださいInvokeAgent。

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してください。このサービスを AWS SDK で使用するこのトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する AWS Amazon Bedrock ランタイムのエージェントのシナリオ

以下のコード例は、 AWS SDK を使用して Amazon Bedrock ランタイム用エージェントの一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon Bedrock Runtime 用エージェント内の複数の関数を呼び出して特定のタスクを実行する方法を示しています。各シナリオにはGitHub、コードの設定と実行方法に関する説明が記載されたリンクが含まれています。

例

- Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

Amazon Bedrock とStep Functions によるジェネレーティブ AI アプリケーションの構築とオーケストレーション

次のコード例は、Amazon Bedrock と Step Functions を使用してジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。

Python

SDK for Python (Boto3)

Amazon Bedrock サーバーレスプロンプトチェーンシナリオではAWS Step Functions、Amazon Bedrock と Agents for Amazon Bedrock を使用して、複雑でサーバーレスでスケーラブルなジェネレーティブ AI アプリケーションを構築およびオーケストレーションする方法を示しています。以下の実例が含まれています。

- ある小説の分析を文学ブログに書く。この例は、単純で連続した一連のプロンプトを示しています。
- 特定のトピックに関する短いストーリーを生成します。この例は、AI が以前に生成した項目のリストを反復的に処理する方法を示しています。
- 特定の目的地への週末休暇の旅程を作成します。この例は、複数の異なるプロンプトを並列化する方法を示しています。
- 映画製作者を務める人間のユーザーに、映画のアイデアを売り込んでください。この例は、同じプロンプトを異なる推論パラメータで並列化する方法、チェーン内の前のステップに戻る方法、ワークフローに人間の入力を含める方法を示しています。
- ユーザーが手元に持っている食材に基づいて食事を計画します。この例は、プロンプトチェーンに 2 つの異なる AI 会話を組み込み、2 つの AI ペルソナが互いに議論を交わして最終的な結果を改善する方法を示しています。
- GitHub 現在最もトレンドが高いリポジトリを見つけて要約してください。この例は、外部 API と相互作用する複数の AI エージェントをチェーン接続する方法を示しています。

完全なソースコードとセットアップと実行の手順については、のプロジェクト全体を参照してください。GitHub

この例で使用されているサービス

- Amazon Bedrock
- Amazon Bedrock ランタイム
- Agents for Amazon Bedrock

- Amazon Bedrock ランタイム用エージェント
- Step Functions

AWS SDK 開発者ガイドとコード例の完全なリストについては、を参照してくださいこのサービスを AWS SDK で使用する。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Bedrock による不正検出

AWS は、AI の責任ある使用に取り組んでいます。不正利用の芽を摘むため、Amazon Bedrock は、「責任ある AI ポリシー」またはサードパーティーのモデルプロバイダーの利用規約を含む、 AWS の「利用規約(AUP)」および「サービス条件」に対する違反の芽を特定して回避する不正利用自動検出メカニズムを実装しています。

Amazon の不正検出メカニズムは完全に自動化されているため、ユーザー入力やモデル出力を人間が確認したりアクセスしたりする必要はありません。

不正利用の自動検出には以下が含まれます。

- コンテンツの分類 - 分類子を使用して、ユーザー入力やモデル出力に含まれる有害なコンテンツ (暴力を扇動するコンテンツなど) を検出します。分類子は、モデルの入力と出力を処理し、攻撃の種類と信頼度を割り当てるアルゴリズムです。これらの分類子は、 Titanとサードパーティーのモデルの使用の両方で実行できます。分類プロセスは自動化されており、ユーザー入力やモデル出力を人間が確認する必要はありません。
- パターンの特定 - 分類メトリクスを使用して、違反の芽や繰り返し発生する動作を特定します。当社では、匿名化された分類メトリクスをまとめ、サードパーティーのモデルプロバイダーと共有する場合があります。Amazon Bedrock はユーザー入力やモデル出力を保管せず、サードパーティーのモデルプロバイダーと共有することもありません。
- Amazon Bedrock にアップロードするコンテンツには、子性不正使用マテリアル (CSAM) の検出とブロックの責任があります。また、このコンテンツに不正なイメージが含まれていないことを確認する必要があります。CSAM の配布を停止するために、Amazon Bedrock は自動不正検出メカニズム (ハッシュマッチングテクノロジーや分類子など) を使用して明確な CSAM を検出する場合があります。Amazon Bedrock が画像入力で明確な CSAM を検出すると、Amazon Bedrock はリクエストをブロックし、自動エラーメッセージを受け取ります。また、Amazon Bedrock は、米国国立医療情報センター (NCMEC) または関連機関にレポートを提出することもできます。CSAM を重視し、検出、ブロック、報告のメカニズムを更新し続けます。お客様は、適用される法令によって追加のアクションを実行するよう要求され、それらのアクションに対する責任を負う場合があります。

自動不正使用検出メカニズムによって潜在的な違反が特定されると、Amazon Bedrock の使用に関する情報と、当社の利用規約またはサードパーティープロバイダーの AUP への準拠をリクエストすることがあります。これらの条件またはポリシーを遵守することを望まないか、順守できない場合は、Amazon Bedrock へのアクセスが停止 AWS される場合があります。

その他の質問については、 AWS サポートにお問い合わせください。詳細については、「Amazon Bedrock FAQs」を参照してください。

Amazon Bedrock のクォータ

AWS アカウント それぞれにデフォルトのクォータ (以前は制限と呼ばれていました) があります。AWS のサービス特に明記されていない限り、各クォータは各自の地域によって異なります。 AWS アカウントクォータによっては調整できるものもあります。次の表では、次の表の「Service Quotas による調整可能」列の意味を説明しています。

- クォータが 「はい」 とマークされている場合は、『Service Quotas ォータユーザーガイド』の「クォータ増額のリクエスト」の手順に従って調整できます。
- クォータが「いいえ」とマークされている場合は、以下のいずれかの方法でクォータの増額をリクエストできる場合があります。
  - オンデマンドランタイムクォータのクォータ増額をリクエストするには、マネージャーに連絡してください。 AWS アカウント AWS アカウント マネージャーがいない場合、現時点ではクォータを増やすことはできません。
  - 他のクォータの増額をリクエストする場合は、上限引き上げフォームからリクエストを送信し、増額を検討してもらってください。

> **Note**
>
> 圧倒的な需要があるため、既存の割り当て量を使い果たすトラフィックを生成するお客様が優先されます。この条件を満たさない場合、リクエストは拒否される可能性があります。

一部のクォータはモデルによって異なります。特に指定がない限り、クォータはモデルのすべてのバージョンに適用されます。

トピックを選択すると、そのトピックのクォータの詳細が表示されます。

トピック

- ランタイムクォータ
- バッチ推論クォータ
- ナレッジベースのクォータ
- エージェントクォータ
- モデルカスタマイズのクォータ

- プロビジョンドスループットのクォータ

ランタイムクォータ

レイテンシーはモデルによって異なり、以下の条件に正比例します。

- 入力トークンと出力トークンの数。
- その時点で全顧客による継続中のオンデマンドリクエストの総数。

モデル推論を実行するときは、以下のクォータが適用されます。このクォータには、InvokeModelInvokeModelWithResponseStreamとリクエストの合計が考慮されます。

スループットを増やすには、購入してください。Amazon Bedrock のプロビジョニングされたスループット

> **Note**
>
> Service Quotas ォータによってクォータが調整不可とマークされた場合は、 AWS アカウント マネージャに連絡してクォータの増額をリクエストできます。 AWS アカウント マネージャーがいない場合、現時点ではクォータを増やすことはできません。需要が圧倒的に多いため、既存の割り当て量を消費するトラフィックを生成する顧客が優先されます。この条件を満たさない場合、リクエストは拒否される可能性があります。

モデル	1 分あたりに処理されるリクエスト数	1 分あたりに処理されるトークン数	Service Quotas により調整可能 (上の表の注記を参照)
AI21 Labs Jurassic-2 Mid	400	300,000	No
AI21 Labs Jurassic-2 Ultra	100	200,000 件の	No
Amazon Titan Embeddings G1 - Text	2,000	300,000	No

モデル	1 分あたりに処理されるリクエスト数	1 分あたりに処理されるトークン数	Service Quotas により調整可能 (上の表の注記を参照)
Amazon Titan Image Generator G1	60	該当なし	No
Amazon Titan Multimodal Embeddings G1	2,000	該当なし	No
Amazon Titan Text G1 - Express	400	300,000	No
Amazon Titan Text G1 - Lite	800	300,000	No
Anthropic Claude Instant	400	300,000	No
AnthropicClaude2.x	100	200,000 件の	No
Anthropic Claude 3 Sonnet	100	200,000 件の	No
Anthropic Claude 3 Haiku	400	300,000	No
Cohere Command	400	300,000	No
Cohere Command Light	800	300,000	No
CohereEmbed(英語)	2,000	300,000	No
CohereEmbed(多言語)	2,000	300,000	No
MetaLlama 213B	800	300,000	No

モデル	1 分あたりに処理されるリクエスト数	1 分あたりに処理されるトークン数	Service Quotas により調整可能 (上の表の注記を参照)
MetaLlama 270B	400	300,000	No
Mistral AI Mistral 7B Instruct	800	300,000	No
Mistral AI Mixtral 8X7B Instruct	400	300,000	No
Stable Diffusion XL	60	該当なし	No

タブを選択すると、モデル固有の推論クォータが表示されます。

Amazon タイタン Text models

説明	値	Service Quotas により調整可能 (上の表の注記を参照)
テキストプロンプトの長さ (文字数)	42,000	No

Amazon Titan Image Generator G1

説明	値	Service Quotas により調整可能 (上の表の注記を参照)
テキストプロンプトの長さ (文字数)	1,024	No
入力画像サイズ	5 MB	No
画像の高さをピクセル単位で入力 (インペイント/アウトペイント)	1,024	No

説明	値	Service Quotas により調整可能 (上の表の注記を参照)
入力画像の幅 (ピクセル単位) (インペイント/アウトペイント)	1,024	No
画像の高さをピクセル単位で入力 (画像バリエーション)	4,096	No
入力画像の幅 (ピクセル単位) (画像バリエーション)	4,096	No
入力画像の合計ピクセル数	12,582,912	No

Amazon Titan Embeddings G1 - Text

説明	値	Service Quotas により調整可能 (上の表の注記を参照)
テキスト入力の長さ (文字数)	50,000	No

Amazon Titan Multimodal Embeddings G1

説明	値	Service Quotas により調整可能 (上の表の注記を参照)
テキスト入力の長さ (文字数)	100,000	No
Base64 でエンコードされた画像文字列 (文字単位)	25,000,000	No

バッチ推論クォータ

バッチ推論を実行するときは、以下のクォータが適用されます。クォータは入出力データのモダリティによって異なります。

> **Note**
>
> Service Quotas オータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

モダリティ	最小ファイルサイズ	最大ファイルサイズ	Service Quotas により調整可能 (上の表の注記を参照)
テキストから埋め込みへ	75 MB	500 MB	No
テキストからテキストへ	20 MB	150 MB	No
テキスト/画像から画像へ	1 MB	50 MB	No

ナレッジベースのクォータ

Amazon Bedrock のナレッジベースには次のクォータが適用されます。

> **Note**
>
> Service Quotas オータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

説明	最大値	Service Quotas により調整可能 (上の表の注記を参照)
アカウントあたりのナレッジベース、地域ごと	50	No

説明	最大値	Service Quotas により調整可能 (上の表の注記を参照)
データソースファイルサイズ (ソースドキュメント)	50 MB	No
データソースファイルサイズ (メタデータファイル)	10 KB	No
データソースのチャンクサイズ (Titanテキスト G1-埋め込み)	8,192	No
データソースのチャンクサイズ (英語) Cohere Embed	512	No
データソースのチャンクサイズ (CohereEmbed多言語)	512	No

エージェントクォータ

Amazon Bedrock のエージェントには以下のクォータが適用されます。

> **Note**
>
> Service Quotas ォータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

説明	最大値	Service Quotas により調整可能 (上の表の注記を参照)
アカウントあたりのエージェント	50	Yes
エージェントあたりのエイリアス	10	No

説明	最大値	Service Quotas により調整可能 (上の表の注記を参照)
エージェント指示の文字	1,200	No
エージェントあたりのアクショングループ	5	No
エージェント内のアクショングループ全体の API 数	5	No
Lambda レスポンスペイロードサイズ	25 KB	No
エージェントあたりの関連付けられたナレッジベース	2	No

モデルカスタマイズのクォータ

モデルカスタマイズには、次のクォータが適用されます。

> **Note**
>
> Service Quotas ォータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

説明	最大値	Service Quotas により調整可能 (上の表の注記を参照)
アカウントあたりのスケジュールされたトレーニングジョブ	2	No
アカウントあたりのカスタムモデル	100	Yes

ハイパーパラメータクォータを確認するには、を参照してください。カスタムモデルのハイパーパラメータ

タブを選択すると、さまざまな基盤モデルのカスタマイズに使用されるトレーニングデータセットと検証データセットに適用されるモデル固有のクォータが表示されます。

> **Note**
>
> Service Quotas ォータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

Amazon Titan Text G1 - Express

説明	上限 (事前トレーニングの継続)	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
バッチサイズが 1 の場合の入力トークンと出力トークンの合計	4,096	4,096	No
バッチサイズが 2、3、または 4 の場合の入力トークンと出力トークンの合計	2,048	2,048	No
データセット内のサンプルあたりの文字数クォータ	トークンクォータ x 6	トークンクォータ x 6	No
トレーニングデータセットのレコード	100,000	10,000	Yes
検証データセットのレコード	1,000	1,000	Yes

説明	上限 (事前トレーニングの継続)	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
トレーニングデータセットのファイルサイズ	10 GB	1 GB	Yes
検証データセットのファイルサイズ	100 MB	100 MB	Yes

Amazon Titan Text G1 - Lite

説明	最大値 (事前トレーニングの継続)	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
バッチサイズが 1 または 2 の場合の入力トークンと出力トークンの合計	4,096	4,096	No
バッチサイズが 3、4、5、または 6 の場合の入出力トークンの合計	2,048	2,048	No
データセット内のサンプルあたりの文字数クォータ	トークンクォータ x 6	トークンクォータ x 6	No
トレーニングデータセットのレコード	100,000	10,000	Yes
検証データセットのレコード	1,000	1,000	Yes

説明	最大値 (事前トレーニングの継続)	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
トレーニングデータセットのファイルサイズ	10 GB	1 GB	Yes
検証データセットのファイルサイズ	100 MB	100 MB	Yes

Amazon Titan Image Generator G1

説明	最小値 (微調整)	最大 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
トレーニングサンプルのテキストプロンプトの長さ (文字単位)	3	1,024	No
トレーニングデータセットのレコード	5	10,000	No
入力画像サイズ	0	50 MB	No
入力画像の高さ (ピクセル)	512	4,096	No
入力画像の幅 (ピクセル)	512	4,096	No
入力画像の合計ピクセル数	0	12,582,912	No
入力画像の縦横比	1:4	4:1	No

Amazon Titan Multimodal Embeddings G1

説明	最小 (微調整)	最大 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
トレーニングサンプルのテキストプロンプトの長さ (文字単位)	0	2,560	No
トレーニングデータセットのレコード	1,000	500,000	No
入力画像サイズ	0	5 MB	No
入力画像の高さ (ピクセル)	128	4096	No
入力画像の幅 (ピクセル)	128	4096	No
入力画像の合計ピクセル数	0	12,528,912	No
入力画像の縦横比	1:4	4:1	No

Cohere Command & Meta ラマ 2

説明	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
入力トークン	4,096	No
出力トークン	2,048	No
データセット内のサンプルあたりの文字数クォータ	トークンクォータ x 6	No

説明	最大値 (微調整)	Service Quotas により調整可能 (上の表の注記を参照)
トレーニングデータセットのレコード	10,000	Yes
検証データセットのレコード	1,000	Yes

プロビジョンドスループットのクォータ

プロビジョンドスループットには以下のクォータが適用されます。

> **Note**
>
> Service Quotas ォータによってクォータが調整不可とマークされた場合は、制限引き上げフォームからリクエストを送信して、引き上げを検討することができます。

説明	デフォルト値	Service Quotas により調整可能 (上の表の注記を参照)
コミットメント不要のプロビジョンドスループットに分散できるモデルユニット	2	No
コミットメントによりプロビジョンドスループット全体に分散できるモデルユニット	0	No

API リファレンス

API リファレンスはこちらにあります。

Amazon Bedrock ユーザーガイドのドキュメント履歴

- ドキュメントの最新更新日:2024 年 3 月 29 日

次の表に、Amazon Bedrock の各リリースにおける重要な変更点を示します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

変更	説明	日付
新機能	基本モデルのプロビジョンドスループットをコミットメントなしで購入できるようになりました。	2024 年 3 月 29 日
プロビジョンドスループットのモデルサポートが拡大	、、CohereEmbed英語AnthropicClaude 3 SonnetAnthropicClaude 3 Haiku、多言語のプロビジョンドスループットを購入できるようになりました。Cohere Embed	2024 年 3 月 29 日
新機能	Amazon OpenSearch Serverless でネットワークアクセスポリシーを作成して、Amazon Bedrock ナレッジベースが VPC OpenSearch エンドポイントで設定されたプライベートなサーバーレスベクター検索コレクションにアクセスできるようにできるようになりました。	2024 年 3 月 28 日
新機能	ソースドキュメントのメタデータを Amazon Bedrock のナレッジベースに含め、ナレッジベースのクエリ中にメ	2024 年 3 月 27 日

	タデータをフィルタリングできるようになりました。	
新機能	ナレッジベースにクエリを実行して応答を生成するときに、プロンプトテンプレートを使用してモデルに送信されるプロンプトをカスタマイズできるようになりました。	2024 年 3 月 26 日
Amazon Bedrock のナレッジベースをクエリするためのモデルサポートの強化	AnthropicClaude 3 Sonnetナレッジベースのレスポンス生成に使用できるようになりました。	2024 年 3 月 25 日
新しいモデル	Amazon Anthropic Claude 3 Haiku Bedrockと併用できるようになりました。	2024 年 3 月 13 日
新しいモデル	Amazon Anthropic Claude 3 Sonnet Bedrockと併用できるようになりました。	2024 年 3 月 4 日
新しいモデル	Amazon Bedrock Mistral AI でモデルを使用できるようになりました。	2024 年 3 月 1 日
新機能	フィルター可能なテキストフィールドを含む Amazon OpenSearch Serverless ベクターストアのナレッジベースで検索戦略をカスタマイズできるようになりました。	2024 年 2 月 28 日
新機能	Amazon Bedrock Titan イメージジェネレーターからウォーターマークの付いた画像を検出できるようになりました。	2024 年 2 月 14 日

サポートが更新されました AWS PrivateLink	AWS PrivateLink を使用して、Agents for Amazon Bedrock ビルドタイムサービスのインターフェイス VPC エンドポイントを作成できるようになりました。	2024 年 2 月 9 日
IAM ロールの更新	ナレッジベース全体で同じサービスロールを使用したり、プレフィックスがあらかじめ定義されていないロールを使用したりできるようになりました。	2024 年 2 月 9 日
レガシーステータスのモデル	Stable Diffusion XLv0.8 はレガシーステータスになりました。2024 年 4 月 30 日までに Stable Diffusion XL v1.x に移行してください。	2024 年 2 月 21 日
「コード例」の章が追加されました。	Amazon Bedrock ガイドには、さまざまな Amazon Bedrock アクションとシナリオにわたるコード例が含まれるようになりました。	2024 年 1 月 25 日
新機能	Amazon Bedrock のナレッジベースでは、コンソールで Amazon OpenSearch Serverless ベクターストアをすばやく作成することを選択したときに、プロダクションアカウントと非プロダクションアカウントのどちらかを選択できるようになりました。	2024 年 1 月 24 日

新機能	Agents for Amazon Bedrock では、コンソールのテストウィンドウを使用するときにトレースをリアルタイムで表示できるようになりました。	2024 年 1 月 18 日
Amazon Bedrock のナレッジベースにデータソースを埋め込むためのモデルサポートの強化	Amazon Bedrock のナレッジベースでは、CohereEmbedCohereEmbed英語と多言語を使用してデータソースを埋め込むことができるようになりました。	2024 年 1 月 17 日
Amazon Bedrock 用エージェントのモデルサポートの強化、および Amazon Bedrock のナレッジベースへのクエリの追加	Amazon Bedrock のエージェントと Amazon Bedrock レスポンス生成用のナレッジベースのエージェントが 2.1 Anthropic Claude をサポートするようになりました。	2023 年 12 月 27 日
リージョンの拡張	Amazon Bedrock は AWS GovCloud (米国西部) (us-gov-west-1) でご利用いただけるようになりました。エンドポイントの詳細については、「Amazon Bedrock エンドポイントとクォータ」を参照してください。	2023 年 12 月 21 日
新しいベクトルストアのサポート	Amazon Aurora データベースクラスターにナレッジベースを作成できるようになりました。詳細については、「Amazon Aurora でベクトルストアを作成する」を参照してください。	2023 年 12 月 21 日

新しいマネージドポリシー	Amazon Bedrock では、リソースの作成、読み取り、更新、削除に対するアクセス許可をユーザーに付与する AmazonBedrockFullAccess と、すべてのアクションに対して読み取り専用のアクセス許可をユーザーに付与する AmazonBedrockReadOnly が追加されました。	2023 年 12 月 12 日
新機能	Amazon Bedrock では、自動メトリクスまたはヒューマンワーカーを使用したモデル評価ジョブの作成がサポートされるようになりました。	2023 年 11 月 29 日
新機能	モデルバージョンをモニタリングし、カスタマイズできるようになりました。	2023 年 11 月 29 日
新しいモデル Titan	からの新しいモデルには Titan Image Generator G1、Amazon とAmazon Titan が含まれますTitan Multimodal Embeddings G1。詳細については、「Titanモデル」を参照してください。	2023 年 11 月 29 日
新機能	継続的な事前トレーニングにより、新しい分野の知識をモデルに教えることができます。詳細については、「カスタムモデル」を参照してください。	2023 年 11 月 28 日

新機能	Retrieve と RetrieveAndGenerateAPI を使用してナレッジベースにクエリを実行できるようになりました。詳細については、「ナレッジベースへのクエリの実行」を参照してください。	2023 年 11 月 28 日
一般リリース	Amazon Bedrock サービスのナレッジベースの一般リリース 詳細については、「Amazon Bedrock のナレッジベース」を参照してください。	2023 年 11 月 28 日
一般リリース	Agents for Amazon Bedrock サービスの一般リリースです。詳細については、「Agents for Amazon Bedrock」を参照してください。	2023 年 11 月 28 日
より多くのモデルのカスタマイズ	モデルを、CohereMetaおよびからカスタマイズできるようになりました。詳細については、「カスタムモデル」を参照してください。	2023 年 11 月 28 日
新しいモデルリリース	MetaCohere新しいモデルとモデルを網羅するようにドキュメントを更新しました。詳細については、「What is Amazon Bedrock?」を参照してください。	2023 年 11 月 13 日

ドキュメントのローカリゼーション	日本語とドイツ語でのAmazon Bedrock ドキュメントが公開されました。	2023 年 10 月 20 日
リージョンの拡張	Amazon Bedrock が欧州 (フランクフルト) (eu-central-1) で利用可能になりました。エンドポイントの詳細については、「Amazon Bedrock エンドポイントとクォータ」を参照してください。	2023 年 10 月 19 日
リージョンの拡張	Amazon Bedrock が、アジアパシフィック (東京) (ap-northeast-1) リージョンで利用可能になりました。エンドポイントの詳細については、「Amazon Bedrock エンドポイントとクォータ」を参照してください。	2023 年 10 月 3 日
限定プレビューリリース	Amazon Bedrock サービスの限定プレビューリリースです。詳細については、「What is Amazon Bedrock?」を参照してください。	2023 年 9 月 28 日

AWS 用語集

AWS 最新の用語については、『リファレンス』AWS の用語集を参照してください。AWS の用語集

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。