Agentic AI勉強会

Amazon Bedrock
マルチエージェントコラボレーション
サンプルアプリを実装してみる

第4回（全7回）　|　2026年05月21日

AWS（Amazon Web Services）

Amazonが提供するクラウドコンピューティングサービス

オンデマンド利用

サーバーやストレージなどのITリソースを必要な時に必要な分だけ利用でき、使った分だけ支払う従量課金制

豊富なサービス

- EC2（仮想サーバー）
- S3（オブジェクトストレージ）
- RDS（データベース）
- Lambda（サーバーレスコンピューティング）

グローバルインフラ

世界中のデータセンター（リージョン）から利用可能

スケーラビリティ

需要に応じて自動的にリソースを拡張・縮小できる

物理的なサーバーを購入・管理する必要がなく、Webサービス、データ分析、機械学習、ストレージなど幅広い用途に使われており、クラウド市場で世界トップシェアを誇る。

Amazon Bedrock

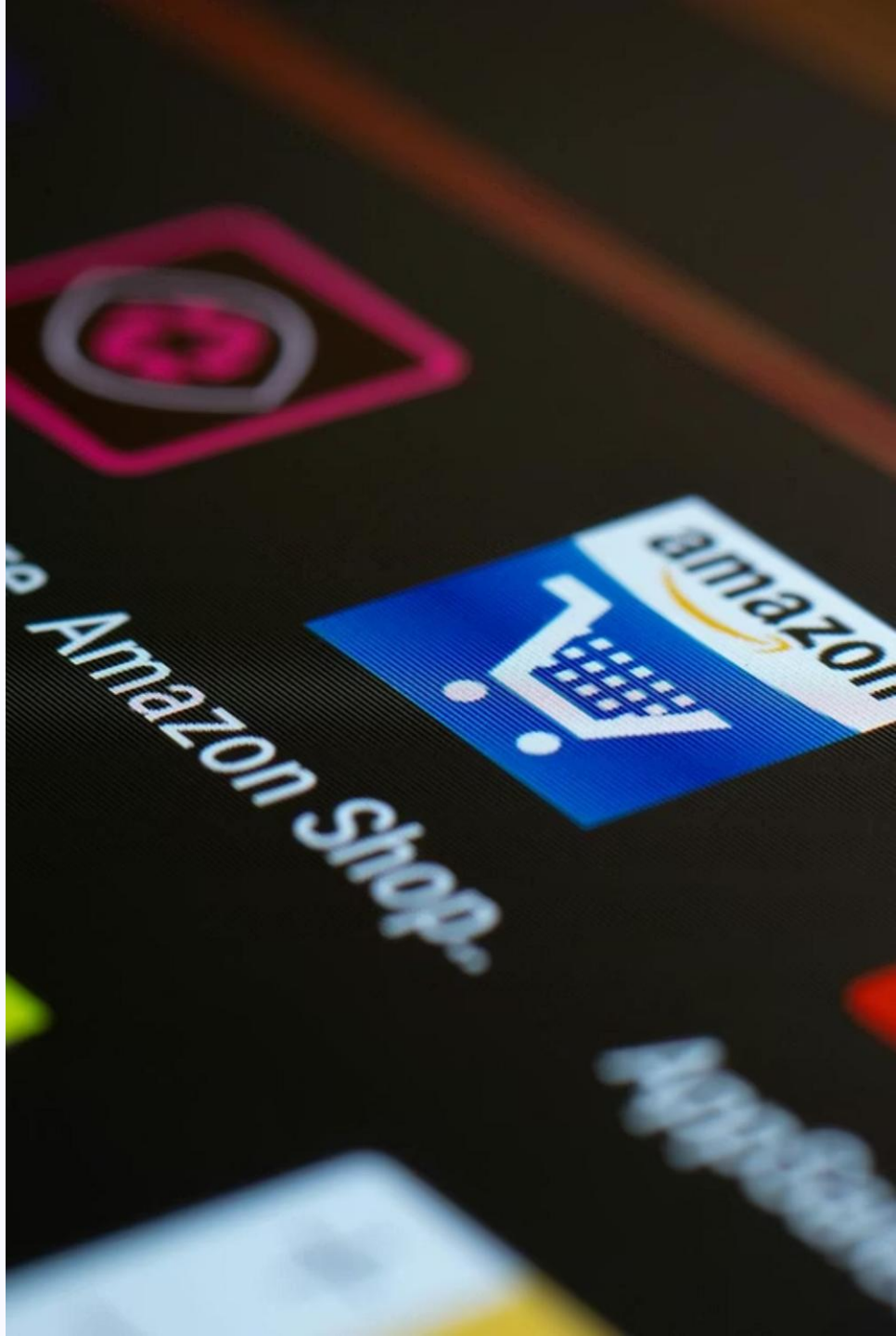
生成AIアプリとAIエージェントをAWS上で作るための基盤

基盤モデル
会話・要約・分類・生成の土台となるAIモデル

Knowledge Base
社内文書を検索して回答する仕組み

Agents
APIやツールを呼び出して業務を実行

Guardrails
安全対策・出力制御の仕組み



サンプルアプリを作ってみよう！

Amazon Bedrock の新機能マルチエージェントで「わが家の AI 技術顧問」を作ろう！
https://aws.amazon.com/jp/builders-flash/202503/create-ai-advisor-with-bedrock/

アプリの概要

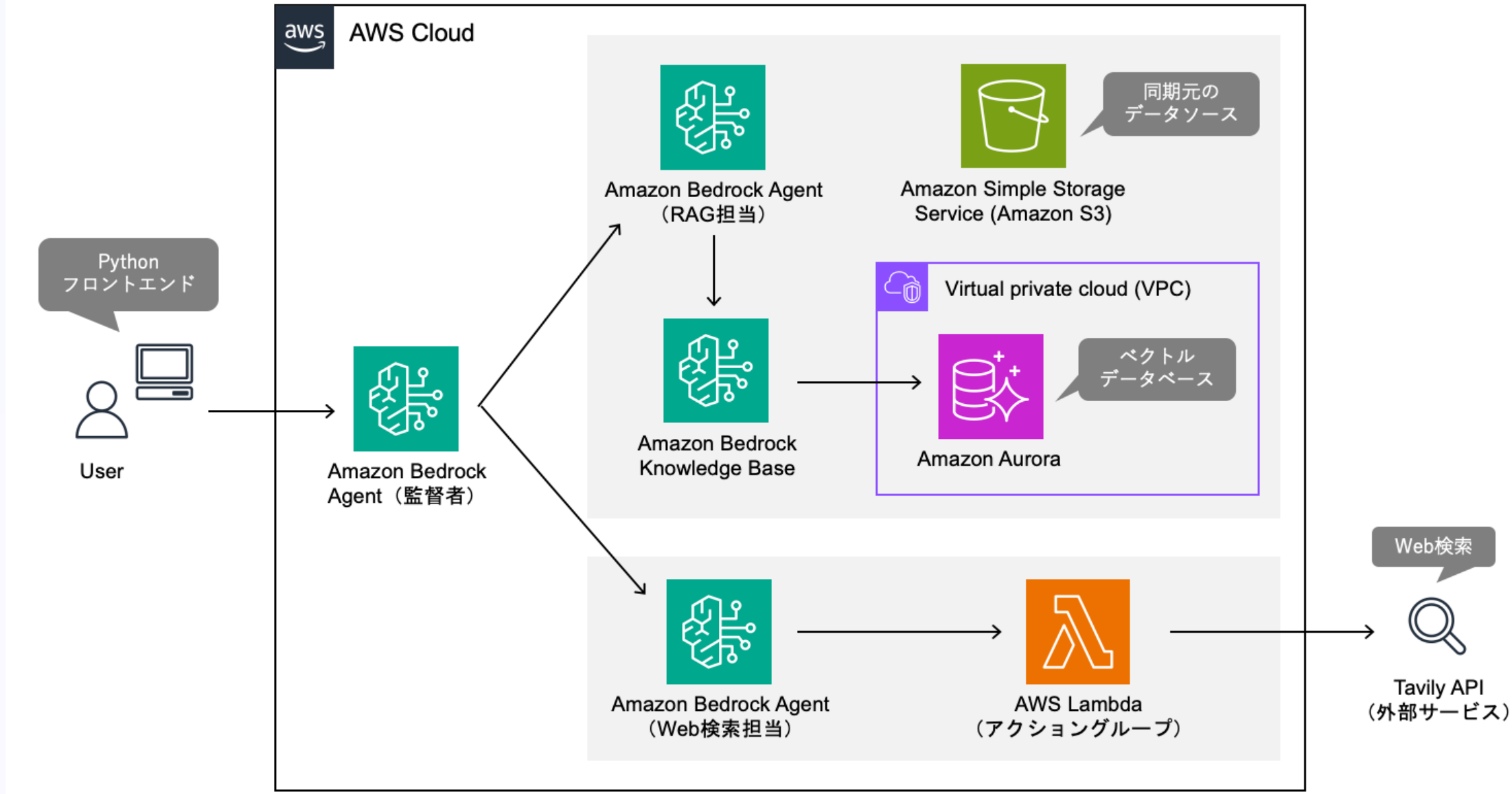
このアプリケーションに質問すると裏で 3 体の AI エージェントが動作します。

- 1 体目は監督者エージェントで、残りの 2 体のサブエージェントへ専門的な仕事をルーティングします。
- 2 体目は RAG（Retrieval-Augmented Generation、検索によって強化された生成）担当のサブエージェントで、Amazon Bedrock のドキュメントを検索して情報提供を行います。
- 3 体目は Web 検索担当のサブエージェントで、Tavily という検索エンジンの API を使って最新情報を取得します。

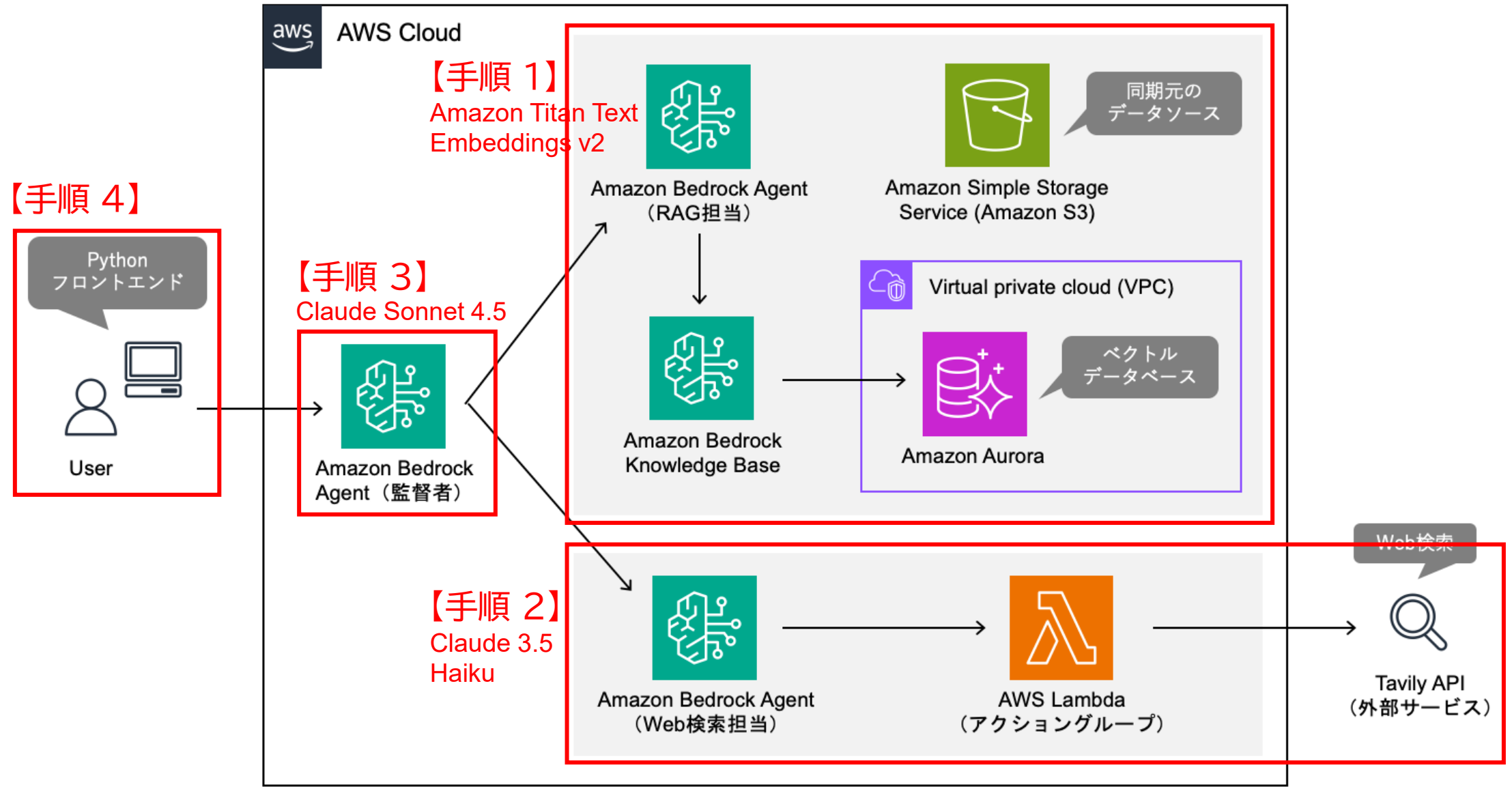
環境の前準備

- Linux 実行環境
  - ➢ Python 3系
  - ➢ AWS CLI 2系
- AWS アカウント　※別途アナウンスします
  - ➢ IAMユーザー登録
    - ✓ AWS CLI 初期設定

サンプルアプリ　アーキテクチャ図



サンプルアプリ　アーキテクチャ図



【手順 1】サブエージェント 1（RAG 担当）の作成

Amazon S3 バケットの作成

- バケット名： bedrock-docs-(あなたのニックネーム)-(YYYYMMDD)
  （例） bedrock-docs-ttanaka-20260521

- データソース： Amazon Bedrock のユーザーガイド（PDF版）を利用

＜注意事項＞

各ファイルサイズは 50 MB のクォータを超えないようにしてください。

参考： Amazon Bedrock ナレッジベースデータの前提条件
https://docs.aws.amazon.com/ja_jp/bedrock/latest/userguide/knowledge-base-ds.html

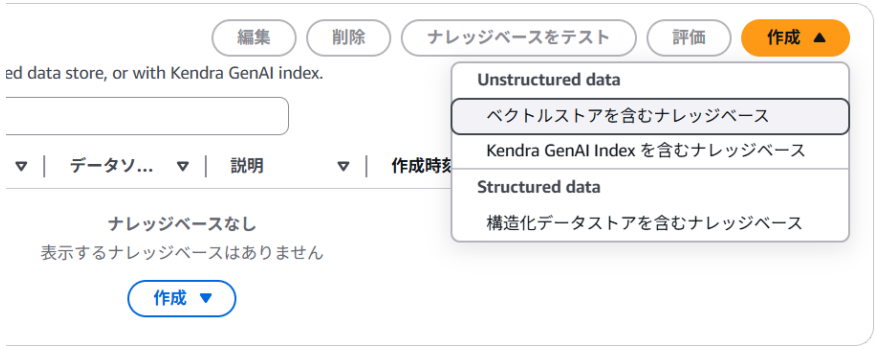
対策:

- ファイルを50MB未満になるよう圧縮する
- ファイルを分割し、1ファイルあたり50MB未満になるよう調節する

  例）PDFgear：オンラインPDF分割ツール　https://www.pdfgear.com/jp/split-pdf/

ナレッジベースの作成

左メニュー ［構築］＞［ナレッジベース］
右画面　　【作成】＞＞［ベクトルストアを含むナレッジベース］



ステップ 1

- ナレッジベース名：bedrock-docs-(あなたのニックネーム)-(YYYYMMDD)
  （例） bedrock-docs-ttanaka-20260521
- データソースを選択:S3　※default

ステップ 2

- データソース名：bedrock-docs
- S3 URI：s3://bedrock-docs-(あなたのニックネーム)-(YYYYMMDD)　※前項で作成

ステップ 3

- 埋め込みモデル：Amazon Titan Text Embeddings v2
- Vector store：Amazon Aurora PostgreSQL Serverless

エージェントの作成

左メニュー ［構築］＞［エージェント］
右画面　　【エージェントを作成】

- エージェント名：bedrock-master-(あなたのニックネーム)-(YYYYMMDD)
  （例） bedrock-master-ttanaka-20260521

エージェントビルダー

- モデルを選択：Anthropic > Claude 3.5 Haiku > (推論プロファイル) US Anthropic Claude 3.5
- エージェント向けの指示：※指定通り入力

一旦【保存】後、

作成が済んだ「ナレッジベース」を指示文と共に追加（add）し、

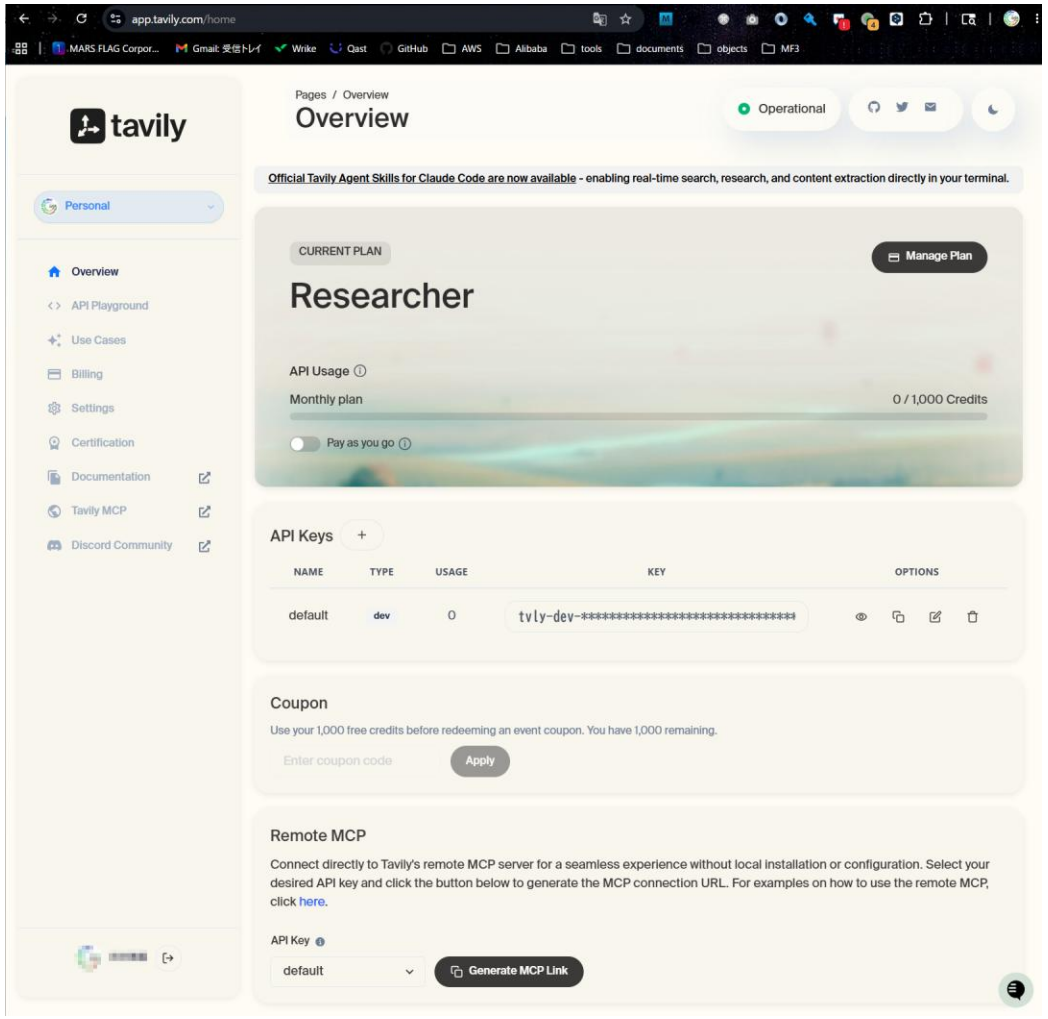
【保存して終了】
画面右側のテスト用サイドバー【準備】
【エイリアスを作成】

※ 作成が済んだ「ナレッジベース」は【同期】を実行しておくこと

【手順 2】サブエージェント 2（web検索担当）の作成

Tavily https://www.tavily.com/

- APIキーの取得



エージェントの作成

左メニュー ［構築］＞［エージェント］
右画面　　【エージェントを作成】

- エージェント名：web-search-master-(あなたのニックネーム)-(YYYYMMDD)
  （例） web-search-master-ttanaka-20260521

エージェントビルダー

- モデルを選択：“最適化済みBedrockエージェント” の ✅ を外してから

  Anthropic > Claude Sonnet 4.5 > (推論プロファイル) US Anthropic Claude Sonnet 4.5

- エージェント向けの指示：※指定通り入力

一旦【保存】後、

「アクショングループ」を【追加】

名前とパラメータを指示通り入力して【作成】

【保存して終了】
画面右側のテスト用サイドバー【準備】
【エイリアスを作成】

AWS Lambda の設定

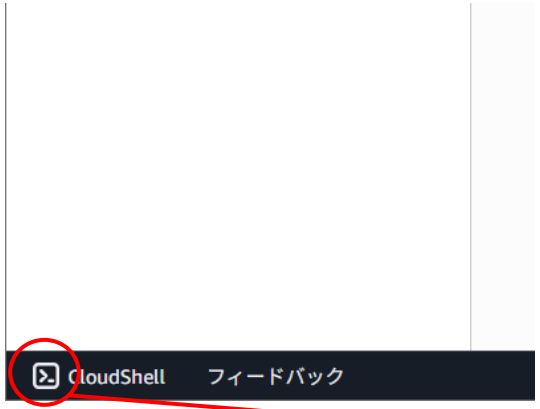
左上検索窓に “lambda” 入力 → 表示されたサービス「Lambda」をクリック

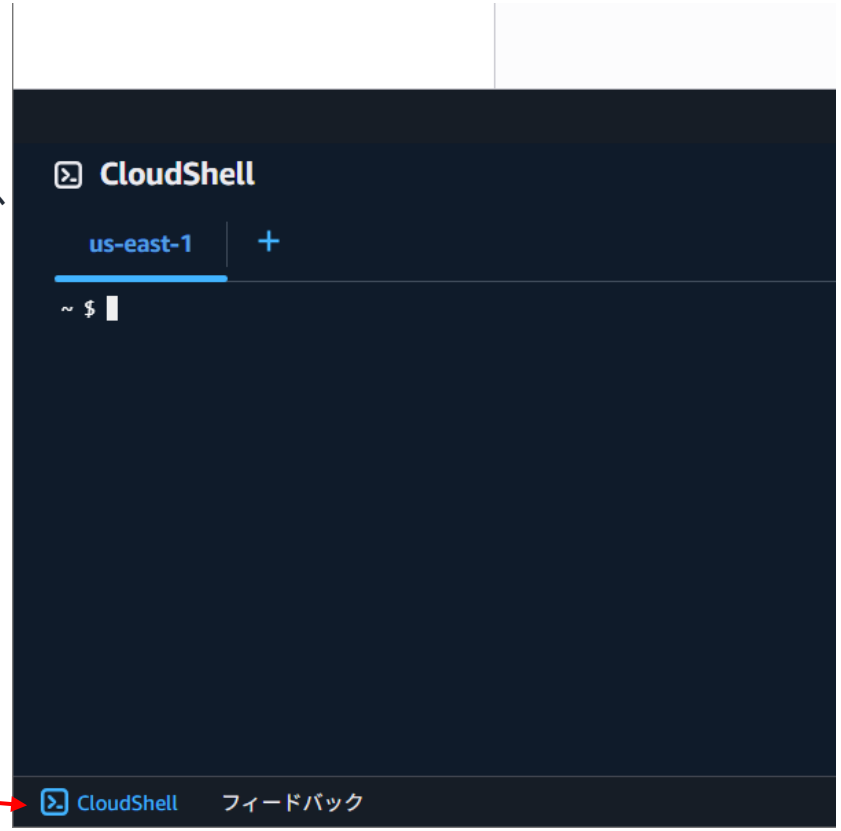
- 関数名：tavily_search-*****　が存在していることを確認

＜記述の訂正＞

（誤）最初に画面右上の［>_］アイコンをクリックして　を起動し、

（正）最初に画面左下の［>_］アイコンをクリックして CloudShell を起動し、





AWS Lambda の設定

- コマンドの実行
  - 確認のため、最後に以下を実行

```
$ ls
layer.zip  python
$ python --version
Python 3.13.13
```

- レイヤーの作成
  - ➢ 「アクション ＞ ファイルのダウンロード」→ layer.zip と入力し、ZIP ファイルをダウンロード
  - ➢ Lambda コンソールの「関数リソース ＞ レイヤー」→【レイヤーを作成】
    - ✓ 互換性のあるランタイム： Python 3.13 ※上記で確認したバージョンにする
- dummy_lambda.py を編集
- Lambda 関数の設定
  - ➢ ランタイム設定： ランタイム： Python 3.13
- レイヤー
  - 【編集】→【追加】

【手順 3】監督者サブエージェントの作成

エージェントの作成

- エージェント名：your-tech-advisor-(あなたのニックネーム)-(YYYYMMDD)
  （例） your-tech-advisor-ttanaka-20260521
- “マルチエージェントコラボレーションを有効にする” に✅を入れる

エージェントビルダー

- モデルを選択：“最適化済みBedrockエージェント” の✅を外してから
  Anthropic > Claude Sonnet 4 >（推論プロファイル）US Anthropic Claude Sonnet 4
- エージェント向けの指示・その他の設定: ※指定通り入力

一旦【保存】、以後指示通り設定

※監督者エージェントのIDと、「エイリアスID」の2つをメモ、または参照可能な状態にしておく

【手順 4】Python フロントエンドから動作確認

- frontend.py を作成
- .env ファイルを作成　※前項の[監督者エージェントのID][エイリアスID]を設定
- AWS IAM ユーザー認証設定

コード実行例：

```
$ python3 -m venv .venvs/streamlit
$ source .venvs/streamlit/bin/activate
(streamlit)$ pip install dotenv boto3 streamlit
  :
(streamlit)$ streamlit run frontend.py &
```

終了:

```
(streamlit)$ kill %
[1]+  終了          streamlit run frontend.py
(streamlit)$ deactivate
$
```

【参考】ガードレールの設定

以下のポリシーを含んだガードレールを作成することが出来る。

- Content filters
- Denied Topics
- Word filters
- 機密情報フィルター
- Contextual grounding
- Blocked messaging

作成したガードレールを特定のエージェントに紐づけて適用することが出来る。

フロントエンドから任意のタイミングでガードレールを適用することも可能。