

ENOG87 LT

AWS で運用するということ

2025年8月

株式会社サーバーワークス 前田友広

自己紹介

名前: 前田友広 (まえだともひろ)

所属: 株式会社サーバーワークス

好きなAWSサービス: IAM



2025年3月より子会社スマートオペレーションズとともに
新潟駅北側すぐのところにオフィスを開設いたしました。



目次 AWSで運用するということ

**AWSで運用するということは、AWS環境の運用監視・維持保守だけでなくAWSを利用するお客様のご支援も含まれます。
お客様のご支援の中で今流行りのAI・MCPサーバーで省力化できそうなポイントをご紹介します。**

- 1. AWS料金の見積り (AWS Pricing MCP Server)**
- 2. AWS環境の構成情報出力 (AWS Diagram MCP Server)**

AWS料金の見積り (AWS Pricing MCP Server)

お客様「こういう構成だとAWS料金はいくら？」

お客様「このサーバのスペックを上げる／下げるといくら変わる？」



AWS料金の見積り (AWS Pricing MCP Server)

AWSのウェブサイトの料金表で見積り

AWS pricing calculatorで見積り

Amazon EC2 のオンデマンドプラン

場所のタイプとリージョンを選択する

場所のタイプ

AWS リージョン

リージョン

アジアパシフィック (東京)

オペレーティングシステム、インスタンスタイプ、vCPU を選択して料金を表示する

オペレーティングシステム

Linux

インスタンスタイプ

すべて

vCPU

すべて

815 個の利用可能なインスタンスを表示

検索欄とページネーション (1, 2, 3, 4, 5, 6, 7, ..., 41)

インスタンス名	オンデマンドの時間単価	vCPU	メモリ	ストレージ	ネットワークパフォーマンス
t4g.nano	USD 0.0054	2	0.5 GiB	EBS のみ	最大 5 ギガビット
t4g.micro	USD 0.0108	2	1 GiB	EBS のみ	最大 5 ギガビット
t4g.small	USD 0.0216	2	2 GiB	EBS のみ	最大 5 ギガビット
t4g.medium	USD 0.0432	2	4 GiB	EBS のみ	最大 5 ギガビット

AWSのウェブサイト

aws pricing calculator

Create estimate: Configure Amazon EC2

説明: 見積りの説明を入力します

ロケーションタイプを選択: リージョン (US East (Ohio))

EC2 の仕様

テナンシー: 共有インスタンス

オペレーティングシステム: Linux

ワークロード: 二つの使用量 (毎日のスパイクトラフィック)

インスタンス数: 1

EC2 インスタンス (859)

Based on your inputs, this is the lowest-cost EC2 instance: t4g.nano

選択されたインスタンス: t4g.nano | Family: t4g | 2vCPU | 0.5 GiB メモリ

インスタンスタイプを検索: t4g.nano

インスタンスファミリー: Any Instance Family | vCPU: Any vCPUs | メモリ (GiB): Any Memory (GiB)

AWS pricing calculator

AWS料金の見積り (AWS Pricing MCP Server)

AWS Pricing MCP Server

<https://awslabs.github.io/mcp/servers/aws-pricing-mcp-server/>

AWS Pricing MCP Server

MCP server for accessing real-time AWS pricing information and providing cost analysis capabilities

Important Note: This server provides real-time pricing data from the AWS Pricing API. We cannot guarantee that AI assistants will always construct filters correctly or identify the absolute cheapest options. All calls are free of charge.

Features

AWS Pricing Discovery & Information

- **Service catalog exploration:** Discover all AWS services with available pricing information
- **Pricing attribute discovery:** Identify filterable dimensions (instance types, regions, storage classes, etc.) for any AWS service
- **Real-time pricing queries:** Access current pricing data with advanced filtering capabilities including multi-option comparisons and pattern matching
- **Multi-region pricing comparisons:** Compare pricing across different AWS regions in a single query
- **Bulk pricing data access:** Download complete pricing datasets in CSV/JSON formats for historical analysis and offline processing

Cost Analysis & Planning

- **Detailed cost report generation:** Create comprehensive cost analysis reports with unit pricing, calculation breakdowns, and usage scenarios
- **Infrastructure project analysis:** Scan CDK and Terraform projects to automatically identify AWS services and their configurations

AWS料金の見積り (AWS Pricing MCP Server)

用意するもの

- ・ AWSアカウント
- ・ AWSスキルビルダーのアカウント
- ・ Amazon Q Developer CLI (インストールしておく)
- ・ Visual Studio Code (あると便利)

AWS料金の見積り (AWS Pricing MCP Server)

導入・初期設定

- ① uvをインストール 「`curl -LsSf https://astral.sh/uv/install.sh | sh`」
- ② Python 3.10をインストール 「`uv python install 3.10`」
- ③ AWSアカウントにて「pricing」をAllowするIAMポリシーを付加したIAMユーザーを作成しアクセスキーを払い出す
- ④ AWSプロファイルを編集
- ⑤ `~/.aws/amazonq/mcp.json` を作成
- ⑥ Visual Studio CodeにAmazon Qをインストール

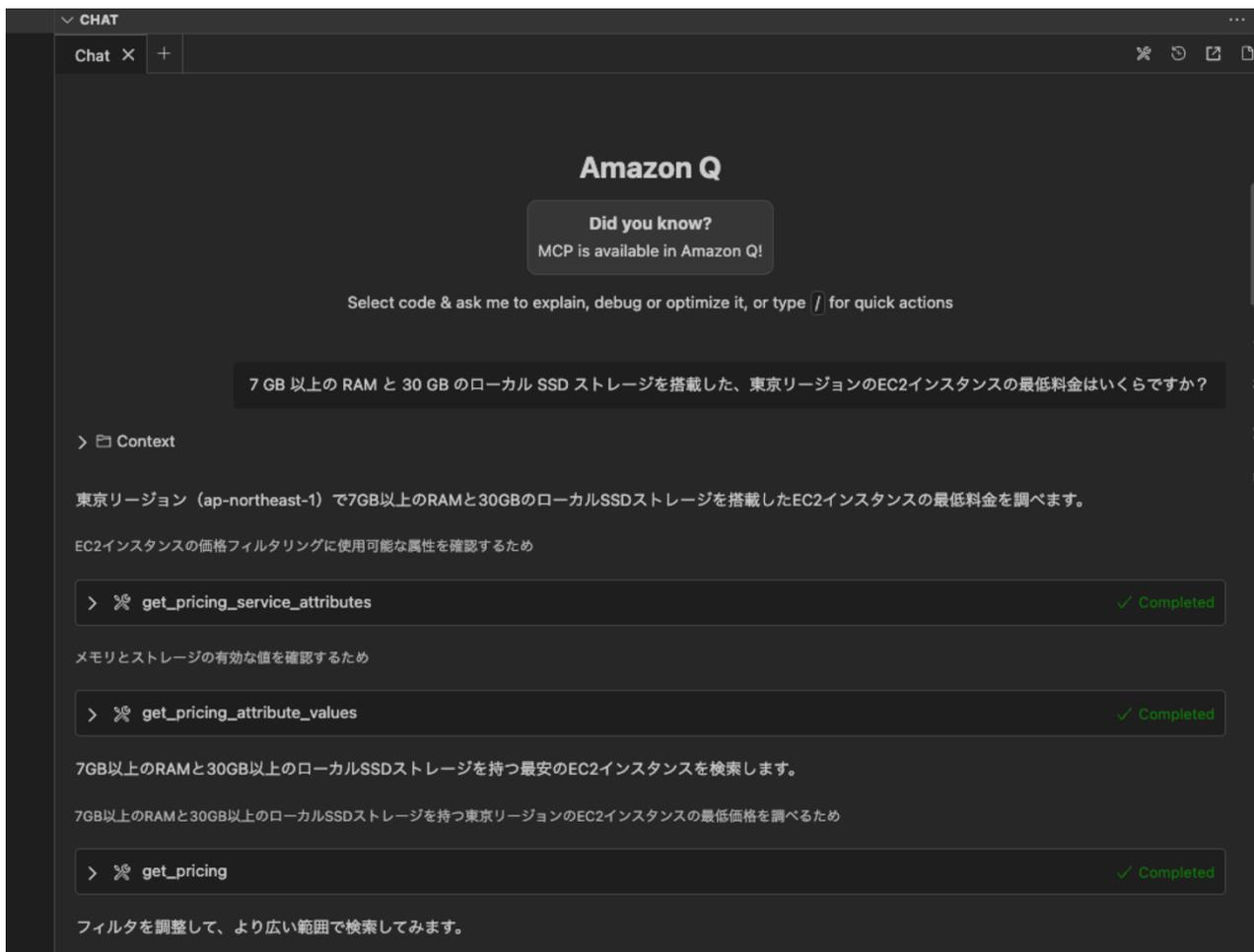


- ⑦ Visual Studio CodeでAWSスキルビルダーにログイン



AWS料金の見積り (AWS Pricing MCP Server)

やってみた



素晴らしい！結果が得られました。30GB以上のローカルSSDストレージを持つインスタンスを特定してみます。

検索結果から、7GB以上のRAMと30GB以上のローカルSSDストレージを搭載した東京リージョンのEC2インスタンスの最低料金は以下の通りです：

最安のインスタンス

r7gd.medium - \$0.0822/時間

- メモリ: 8 GiB
- ストレージ: 1 x 59 NVMe SSD (59GB > 30GB要件を満たす)
- 月額料金 (24時間x30日) : 約 **\$59.18**

その他の候補インスタンス (30GB以上のローカルSSD付き)

- m5d.large - \$0.156/時間**
 - メモリ: 8 GiB
 - ストレージ: 1 x 75 NVMe SSD
- c5d.xlarge - \$0.244/時間**
 - メモリ: 8 GiB
 - ストレージ: 1 x 100 NVMe SSD
- m6id.large - \$0.649/時間**
 - メモリ: 8 GiB
 - ストレージ: 1 x 118 NVMe SSD

まとめ

要件 (7GB以上のRAM + 30GB以上のローカルSSD) を満たす最安のEC2インスタンスは **r7gd.medium** で、時間あたり**\$0.0822**です。

このインスタンスは8GiBのメモリと59GBのNVMe SSDストレージを搭載しており、ご要件を満たしています。

AWS料金の見積り (AWS Pricing MCP Server)

やってみた

メッセージングシステムを構築しています。Amazon SQS スタンダードキューと Amazon SQS FIFO キューの料金モデルを、1 か月あたり 1 億件のメッセージを含むワークロードの料金モデルと比較してください。主な機能上の違いは何ですか？ また、どちらを選択すればよいでしょうか？

> Context

Amazon SQSの料金とスタンダードキューとFIFOキューの比較を調べます。

Amazon SQSの価格フィルタリングに使用可能な属性を確認するため

> ☞ get_pricing_service_attributes

✓ Completed

正しいサービスコードを確認します。

SQSの正しいサービスコードを見つけるため

> ☞ get_pricing_service_codes

✓ Completed

SQSのサービスコードは「AWSQueueService」です。

Amazon SQSの価格フィルタリングに使用可能な属性を確認するため

> ☞ get_pricing_service_attributes

✓ Completed

SQSのキュータイプとメッセージ配信の有効な値を確認するため

> ☞ get_pricing_attribute_values

✓ Completed

東京リージョンでスタンダードキューとFIFOキューの料金を取得します。

東京リージョンでSQSスタンダードキューとFIFOキューの料金を比較するため

> ☞ get_pricing

✓ Completed

FIFOキューの料金も取得します。



1億件のメッセージワークロードでの料金比較と機能の違いを分析します。

Amazon SQS 料金比較 (東京リージョン)

1億件/月のメッセージワークロードでの料金

スタンダードキュー:

- Tier1 (0-1000億リクエスト) : \$0.40/100万リクエスト
- 1億件 = 100,000,000リクエスト = 100 × 100万リクエスト
- 月額料金: \$40.00

FIFOキュー:

- Tier1 (0-1000億リクエスト) : \$0.50/100万リクエスト
- 1億件 = 100,000,000リクエスト = 100 × 100万リクエスト
- 月額料金: \$50.00

料金差: FIFOキューはスタンダードキューより25%高い (+\$10.00/月)

主な機能上の違い

スタンダードキュー

- 配信順序: 保証されない (ベストエフォート)
- 配信頻度: 最低1回 (重複の可能性あり)
- スループット: ほぼ無制限
- メッセージグループ: サポートなし
- 重複排除: なし

FIFOキュー

- 配信順序: 厳密に保証 (First-In-First-Out)
- 配信頻度: 正確に1回 (重複なし)
- スループット:
 - バッチなし: 300 TPS
 - バッチあり: 3,000 TPS
- メッセージグループ: サポート (並列処理可能)
- 重複排除: 自動 (5分間のウィンドウ)

どちらを選択すべきか

スタンダードキューを選ぶべき場合:

- 高スループットが必要
- メッセージの順序が重要でない
- 重複処理に対応できる
- コスト重視

AWS環境の構成情報出力（AWS Diagram MCP Server）

お客様「こういう構成で考えてみてよ」

お客様「こういう構成で合ってる？」

お客様「今の構成の一覧や構成図ください」

AWS環境の構成情報出力（AWS Diagram MCP Server）

用意するもの

- ・ AWS Pricing MCP Serverと同じ

導入・初期設定

- ・ AWS Pricing MCP Serverで実施済みなら追加作業なし

AWS環境の構成情報出力 (AWS Diagram MCP Server)

やってみた

スクリプトの概要

スクリプトは /Users/maedatomohiro/Documents/testbuild.sh に保存され、実行権限が付与されています。以下のリソースを順番に作成します：

1. VPC

- 名前: MyVPC
- CIDR: 10.0.0.0/16
- DNS設定を有効化

2. サブネット

- PublicSubnet1 (10.0.0.0/24) - ap-northeast-1a
- PublicSubnet2 (10.0.2.0/24) - ap-northeast-1c
- PrivateSubnet1 (10.0.1.0/24) - ap-northeast-1a
- PrivateSubnet2 (10.0.3.0/24) - ap-northeast-1c
- パブリックサブネットには自動パブリックIP割り当てを有効化

3. インターネットゲートウェイ

- 名前: MyIGW
- VPCにアタッチ

4. Elastic IP

- NATゲートウェイ用に2つのElastic IPを割り当て

5. NATゲートウェイ

- NatGateway1 - PublicSubnet1に配置
- NatGateway2 - PublicSubnet2に配置

6. ルートテーブル

- PublicRouteTable - インターネットゲートウェイへのルート
- PrivateRouteTable1 - NATゲートウェイ1へのルート
- PrivateRouteTable2 - NATゲートウェイ2へのルート

7. ルートテーブルの関連付け

- パブリックサブネットをPublicRouteTableに関連付け
- プライベートサブネットをそれぞれのPrivateRouteTableに関連付け

AWS環境の構成情報出力 (AWS Diagram MCP Server)

やってみた

```
#!/bin/bash

# VPC作成
VPC_ID=$(aws ec2 create-vpc --cidr-block 10.0.0.0/16 --enable-dns-hostnames --enable-dns-support --tag-specifications 'ResourceType=vpc,Tags=[{Key=Name,Value=MyVPC}]' --query 'Vpc.VpcId' --output text)
echo "VPC作成完了: $VPC_ID"

# サブネット作成
PUBLIC_SUBNET1_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.0.0.0/24 --availability-zone ap-northeast-1a --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=PublicSubnet1}]' --query 'Subnet.SubnetId' --output text)
PUBLIC_SUBNET2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.0.2.0/24 --availability-zone ap-northeast-1c --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=PublicSubnet2}]' --query 'Subnet.SubnetId' --output text)
PRIVATE_SUBNET1_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.0.1.0/24 --availability-zone ap-northeast-1a --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=PrivateSubnet1}]' --query 'Subnet.SubnetId' --output text)
PRIVATE_SUBNET2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.0.3.0/24 --availability-zone ap-northeast-1c --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=PrivateSubnet2}]' --query 'Subnet.SubnetId' --output text)

# パブリックサブネットの自動パブリックIP割り当て有効化
aws ec2 modify-subnet-attribute --subnet-id $PUBLIC_SUBNET1_ID --map-public-ip-on-launch
aws ec2 modify-subnet-attribute --subnet-id $PUBLIC_SUBNET2_ID --map-public-ip-on-launch

# インターネットゲートウェイ作成・アタッチ
IGW_ID=$(aws ec2 create-internet-gateway --tag-specifications 'ResourceType=internet-gateway,Tags=[{Key=Name,Value=MyIGW}]' --query 'InternetGateway.InternetGatewayId' --output text)
aws ec2 attach-internet-gateway --vpc-id $VPC_ID --internet-gateway-id $IGW_ID

# Elastic IP割り当て
EIP1_ALLOC_ID=$(aws ec2 allocate-address --domain vpc --query 'AllocationId' --output text)
EIP2_ALLOC_ID=$(aws ec2 allocate-address --domain vpc --query 'AllocationId' --output text)

# NATゲートウェイ作成
NAT_GW1_ID=$(aws ec2 create-nat-gateway --subnet-id $PUBLIC_SUBNET1_ID --allocation-id $EIP1_ALLOC_ID --tag-specifications 'ResourceType=nat-gateway,Tags=[{Key=Name,Value=NatGateway1}]' --query 'NatGateway.NatGatewayId' --output text)
NAT_GW2_ID=$(aws ec2 create-nat-gateway --subnet-id $PUBLIC_SUBNET2_ID --allocation-id $EIP2_ALLOC_ID --tag-specifications 'ResourceType=nat-gateway,Tags=[{Key=Name,Value=NatGateway2}]' --query 'NatGateway.NatGatewayId' --output text)

# NATゲートウェイの作成完了を待機
echo "NATゲートウェイの作成を待機中..."
aws ec2 wait nat-gateway-available --nat-gateway-ids $NAT_GW1_ID $NAT_GW2_ID

# ルートテーブル作成
PUBLIC_RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=PublicRouteTable}]' --query 'RouteTable.RouteTableId' --output text)
PRIVATE_RT1_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=PrivateRouteTable1}]' --query 'RouteTable.RouteTableId' --output text)
PRIVATE_RT2_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=PrivateRouteTable2}]' --query 'RouteTable.RouteTableId' --output text)

# ルート追加
aws ec2 create-route --route-table-id $PUBLIC_RT_ID --destination-cidr-block 0.0.0.0/0 --gateway-id $IGW_ID
aws ec2 create-route --route-table-id $PRIVATE_RT1_ID --destination-cidr-block 0.0.0.0/0 --nat-gateway-id $NAT_GW1_ID
aws ec2 create-route --route-table-id $PRIVATE_RT2_ID --destination-cidr-block 0.0.0.0/0 --nat-gateway-id $NAT_GW2_ID

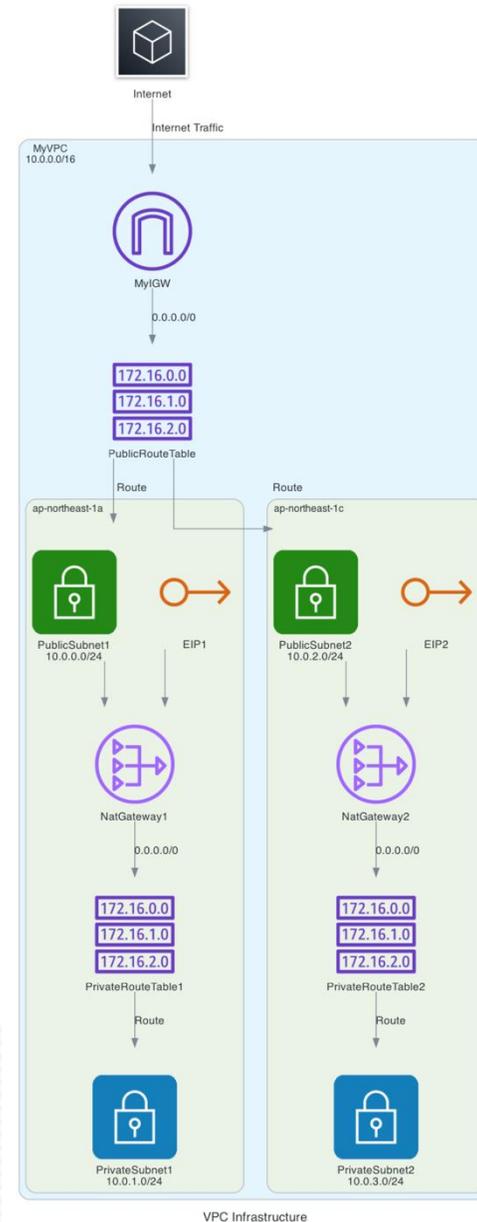
# サブネットとルートテーブルの関連付け
aws ec2 associate-route-table --subnet-id $PUBLIC_SUBNET1_ID --route-table-id $PUBLIC_RT_ID
aws ec2 associate-route-table --subnet-id $PUBLIC_SUBNET2_ID --route-table-id $PUBLIC_RT_ID
aws ec2 associate-route-table --subnet-id $PRIVATE_SUBNET1_ID --route-table-id $PRIVATE_RT1_ID
aws ec2 associate-route-table --subnet-id $PRIVATE_SUBNET2_ID --route-table-id $PRIVATE_RT2_ID

echo "VPCインフラストラクチャの構築が完了しました"
echo "VPC ID: $VPC_ID"
echo "パブリックサブネット1: $PUBLIC_SUBNET1_ID"
echo "パブリックサブネット2: $PUBLIC_SUBNET2_ID"
echo "プライベートサブネット1: $PRIVATE_SUBNET1_ID"
echo "プライベートサブネット2: $PRIVATE_SUBNET2_ID"
```


AWS環境の構成情報出力（AWS Diagram MCP Server）

やってみた

「スクリプト /Users/maedatomohiro/Documents/testbuild.sh をもとにMCPのAWS Diagram Toolを利用して構成図を作図してください。構成図はpngファイルで出力してください。」



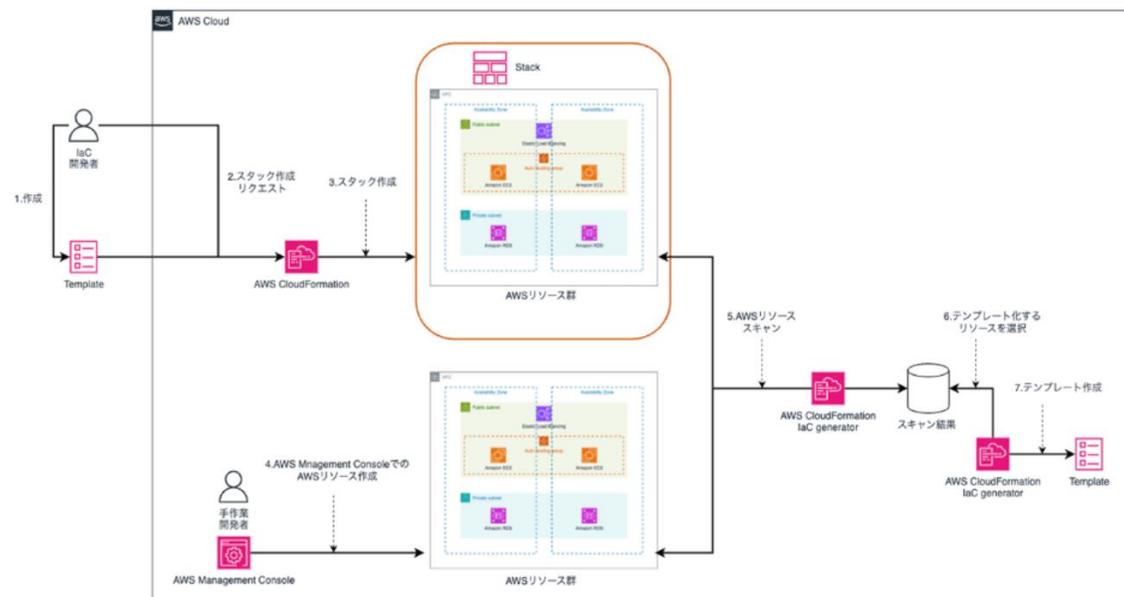
AWS環境の構成情報出力（AWS Diagram MCP Server）

既存環境の出力は、CloudFormation IaCジェネレータでテンプレートを出力して、このテンプレートから構成図を出力させるといったことができるかも。

IaC ジェネレーターの概要

IaC ジェネレーターは、AWS リソースをスキャンし、その結果から CloudFormation テンプレートを作成することができます。これにより、マネジメントコンソールで作成された AWS リソースを Infrastructure as Code の原則に則って管理することが可能になります。このプロセスにより、一貫性があり管理しやすいインフラストラクチャの展開を促進します。

今回はスキャン結果から AWS リソースの棚卸を行う方法を紹介します。



クラウドで、
世界を、もっと、
はたらきやすく

