AWS Black Belt Online Seminar

AWS CloudFormation

よくあるユースケースと質問

木村 友則

Solutions Archtect
2023/10

自己紹介

名前：**木村 友則 (きむら とものり)**

所属：クラウドソリューション統括本部
　　ソリューションアーキテクト

好きなAWSサービス：

AWS CloudFormation, AWS CDK, AWS CLI

本セミナーの対象者

想定聴講者

- CloudFormation のユースケースやよく聞かれる疑問に興味のある方

前提知識

- AWS の基本的な概要や操作を理解していること
- CloudFormation の用語 (スタック、テンプレート、変更セットなど) を理解していること

※ 次の Black Belt Online Seminar で解説しています
AWS CloudFormation #1 基礎編
資料 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_CloudFormation-1_0731_v1.pdf
動画 https://youtu.be/4dyiPsYXG8I

アジェンダ

- よくあるユースケース別の使い方
- よくある質問

※ 本資料では CloudFormation = CFn と略記することがあります

よくあるユースケース別の使いかた

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

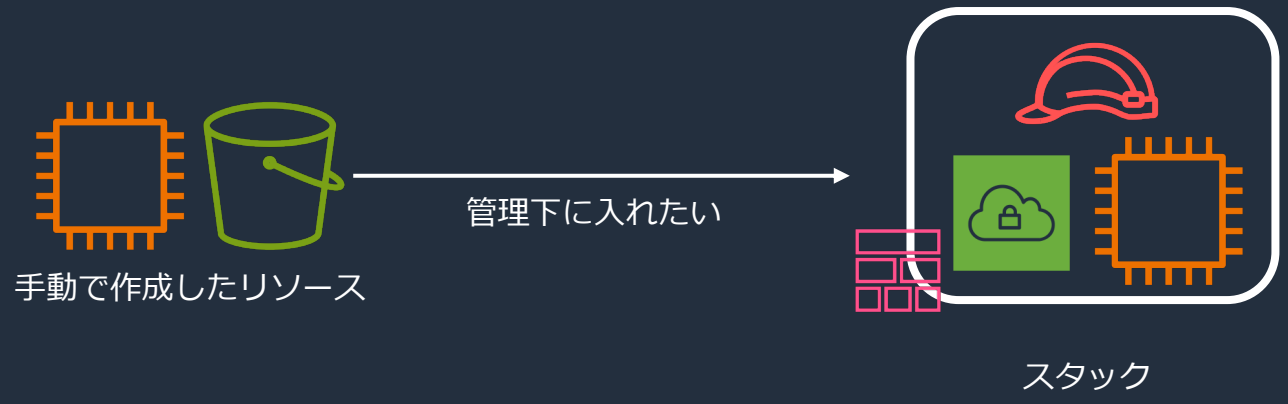
手動で作った既存リソースを CFn の管理下に入れる

実現したいこと

- 手動で作った既存リソースを新規または既存のスタック管理下に入れたい

解決策

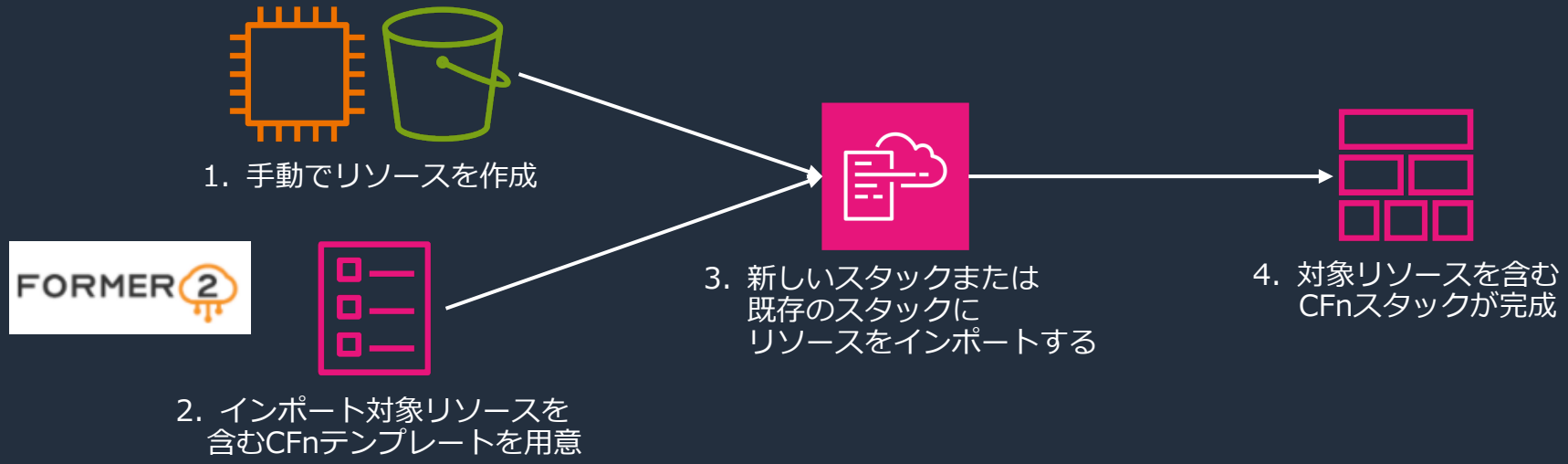
- リソースのインポート機能を利用する



リソースのインポート機能について

手動で作成した AWS リソースを
あとから CloudFormation スタックにインポートして管理可能

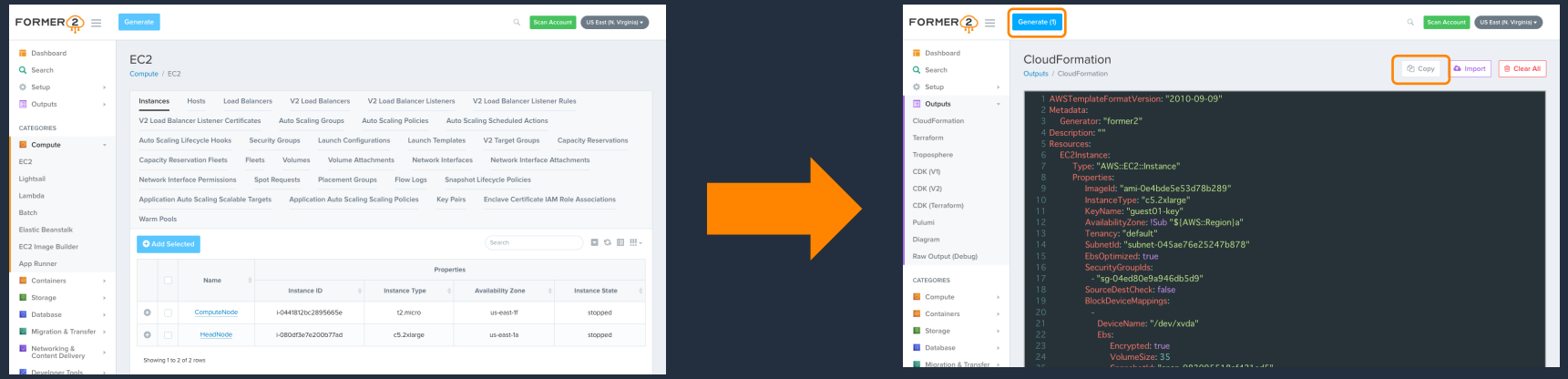
- リソースをスタックの管理下から切り離したり、別のスタック管理下に移動することも可能
- インポート対象のリソースの実際の設定と一致するようテンプレートを用意する
  - AWS 公式のツールではないが、Former2 でテンプレート作成の省力化が可能
    - https://github.com/iann0036/former2



https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/resource-import.html

Former2 を利用したインポート用テンプレートの生成

1. リソース読み取り用の IAM ユーザーを作成する
   - ReadOnlyAccess ポリシーを付与し、アクセスキーを発行する
2. Former2 を開く
   - https://former2.com/ または ローカル環境で起動した Former2
   - ブラウザに適した拡張機能をインストール
3. 先の手順で用意したアクセスキーを適用する
4. 対象リソースを選択して、CloudFormation テンプレートを生成、コピーする



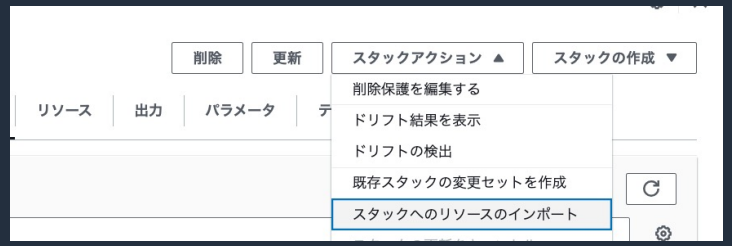
生成したテンプレートを既存スタックにインポートする

1. 生成したテンプレートのResources以下にある各リソースに DeletionPolicy 属性を追加して、既存のインポート先テンプレートの Resources 以下に追記する

```
EC2Instance:
  Type: "AWS::EC2::Instance"
  DeletionPolicy: Retain
  Properties:
    ImageId: "ami-0e4bde5e53d78b289"
```

※ Retain、Delete 等、用途に応じて指定する

2. CloudFormation のマネジメントコンソールでインポートを実行する



インポート先のスタックを選択し、
「スタックへのリソースのインポート」を実行する
※ 新規スタックの場合は「スタックの作成」から

対象リソースの識別子を指定する

3. ドリフト検出を実行し、テンプレートとリソースが一致していることを確認、一致しない場合は修正を実施する

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/resource-import-existing-stack.html

リソースをインポートする際の考慮事項

全てのリソースのインポートに対応しているわけではない

- 対応リソース一覧 (最新の情報は英語版をご覧ください)
- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resource-import-supported-resources.html
- 対応リソースのリクエストは Public Coverage Roadmap まで
- https://github.com/aws-cloudformation/cloudformation-coverage-roadmap

リソースのインポートには、識別子の指定が必須

- テンプレートの生成だけで作業は完了せず、対象リソースの量によって相応の手間もかかる

EC2 や RDS 等の依存リソースが多いものは、インポートが困難になりがち

- S3 や Lambda 等のサーバーレスなリソースは、比較的容易にインポートできる

※ インポートの実施に困難が大きい場合は、再作成もご検討ください。

参考：ローカル環境での Former2 起動方法

実行例

```
$ git clone https://github.com/iann0036/former2
$ cd former2
$ docker-compose up -d
```

※ 詳しくはドキュメントも参照ください
https://github.com/iann0036/former2/blob/master/HOSTING.md

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

肥大化したスタックを分割する（リファクタリング）

実現したいこと

- 運用過程のリソース追加によってスタックが肥大化、その結果スタック変更に時間がかかる、スタック変更の影響範囲が大きい、クォータの上限に達しそう等の顕在化した課題を解消したい

解決策

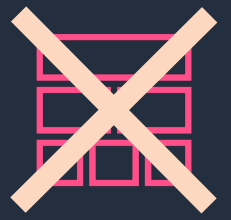
- リソースを残したままスタックを削除し、
  別のテンプレートでリソースのインポートを行うことでスタックをリファクタリングする
  (インポートに対応しているリソースに限られる)

実行手順

1. 既存スタックのテンプレートを編集し更新する
全てのリソースに “DeletionPolicy: Retain” を追記、テンプレートをスタックに適用する。Retain と指定することで、スタックを削除してもリソースは削除されない。(リソースは複数スタックに属せない)

```
EC2Instance:
  Type: "AWS::EC2::Instance"
  DeletionPolicy: Retain
  Properties:
    ImageId: "ami-0e4bde5e53d78b289"
```

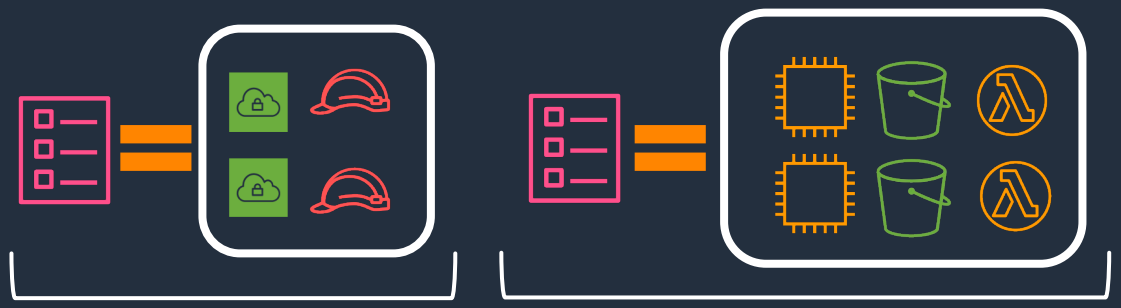
2. 既存スタックを削除する



スタックは無くなる

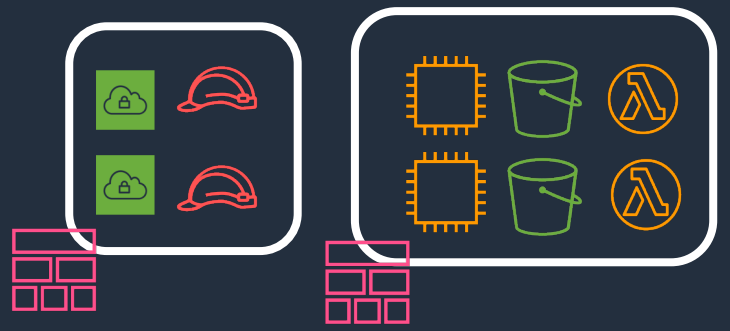
リソースは残る

3. 既存スタックのテンプレートを分割する



※ 必要に応じてクロススタックリファレンスやネストスタックを利用

4. 分割したテンプレートでリソースをインポートする



https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/refactor-stacks.html

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

各AWSアカウント・リージョンに基本設定を展開する

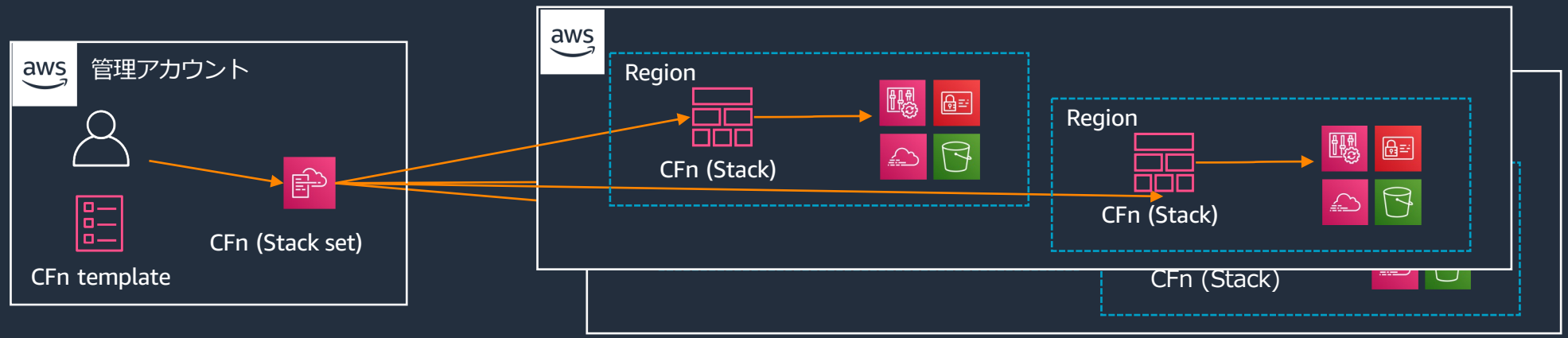
実現したいこと

- マルチアカウント・マルチリージョン環境で、基本設定を展開したい
- 基本設定として、監査ログ、構成のチェック、基本の IAM Role を設定したい

解決策

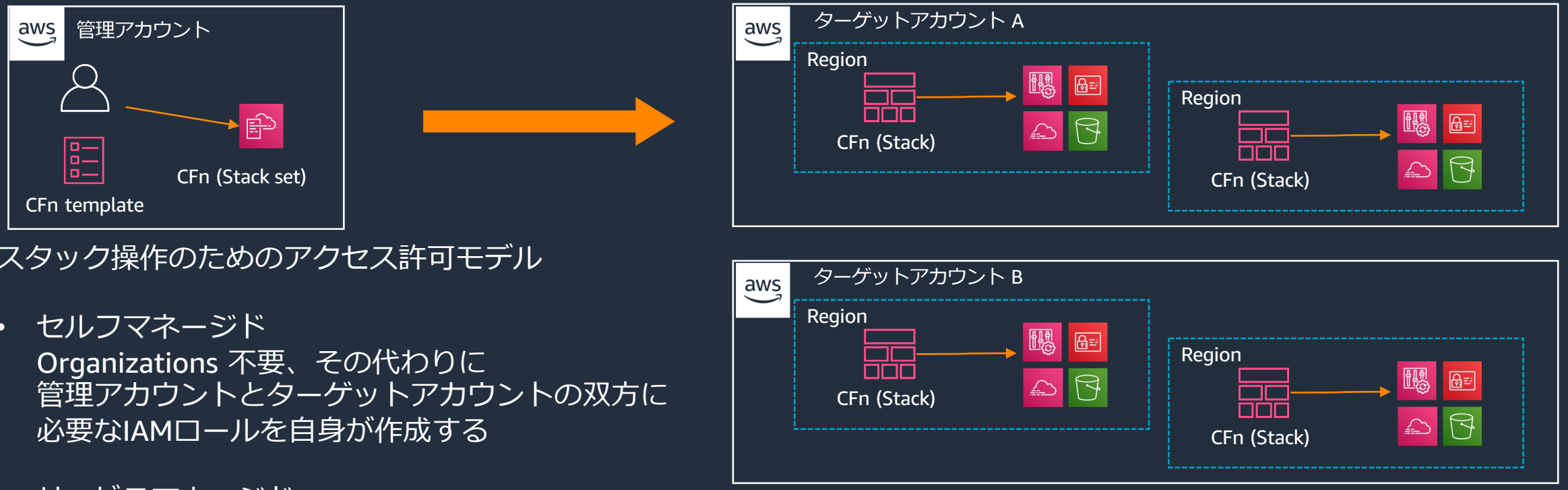
- StackSets を使って対象アカウント・リージョンに基本設定のスタックを作成する

  ※ AWS Organizations が利用できる状況ならば、AWS ControlTower の利用もお勧めです。
  ControlTower では、例示したような基本設定の展開を StackSets を活用して実現しています。



StackSets について

一回の操作で複数のアカウントやリージョンへ スタックを作成、更新、削除できるCloudFormationの機能



スタック操作のためのアクセス許可モデル

- セルフマネージド
  Organizations 不要、その代わりに
  管理アカウントとターゲットアカウントの双方に
  必要なIAMロールを自身が作成する

- サービスマネージド
  Organizations を通じて必要なIAMロールを自動で作成する

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/what-is-cfnstacksets.html

実行手順 – アクセス許可モデルの選択と設定 (初回のみ)

セルフマネージド

管理アカウントとターゲットアカウントの双方に
指定された名称・内容でIAMロールの作成が必要です。

IAMロール作成のための CFn テンプレートが
配布されています (ドキュメント参照)。

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/stacksets-prereqs-self-managed.html

サービスマネージド

Organizations で
CloudFormation StackSets の 信頼されたアクセスを
有効化することで、以下のIAMロールが自動的に作成されます。

管理アカウント： AWSServiceRoleFor
CloudFormationStackSetsOrgAdmin (実際には一行)
ターゲットアカウント： AWSServiceRoleFor
CloudFormationStackSetsOrgMember (実際には一行)
及び stacksets-exec-*

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/stacksets-orgs-activate-trusted-access.html

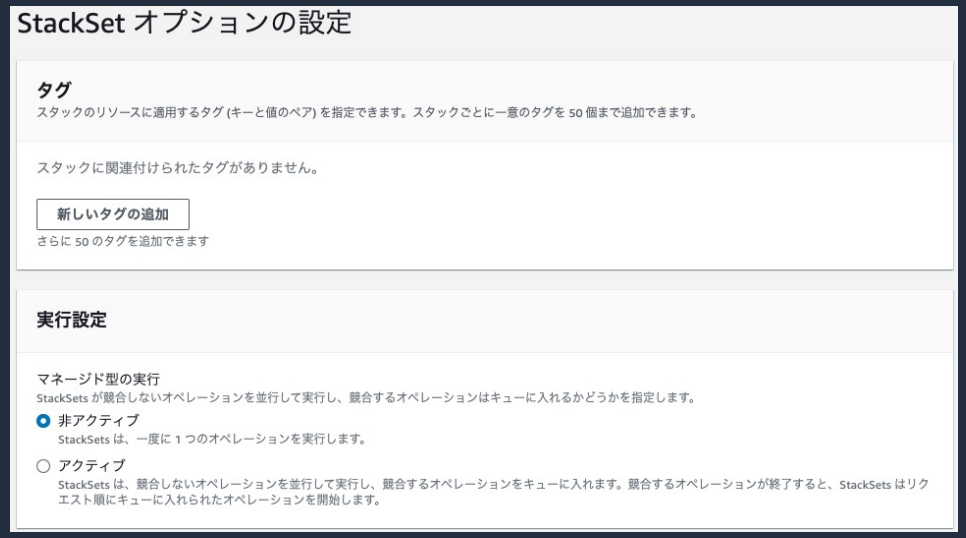
実行手順 1/2

1. テンプレートの選択
アクセス許可モデルを選んだ上で、
自身で用意したものかサンプルテンプレートを
指定します。

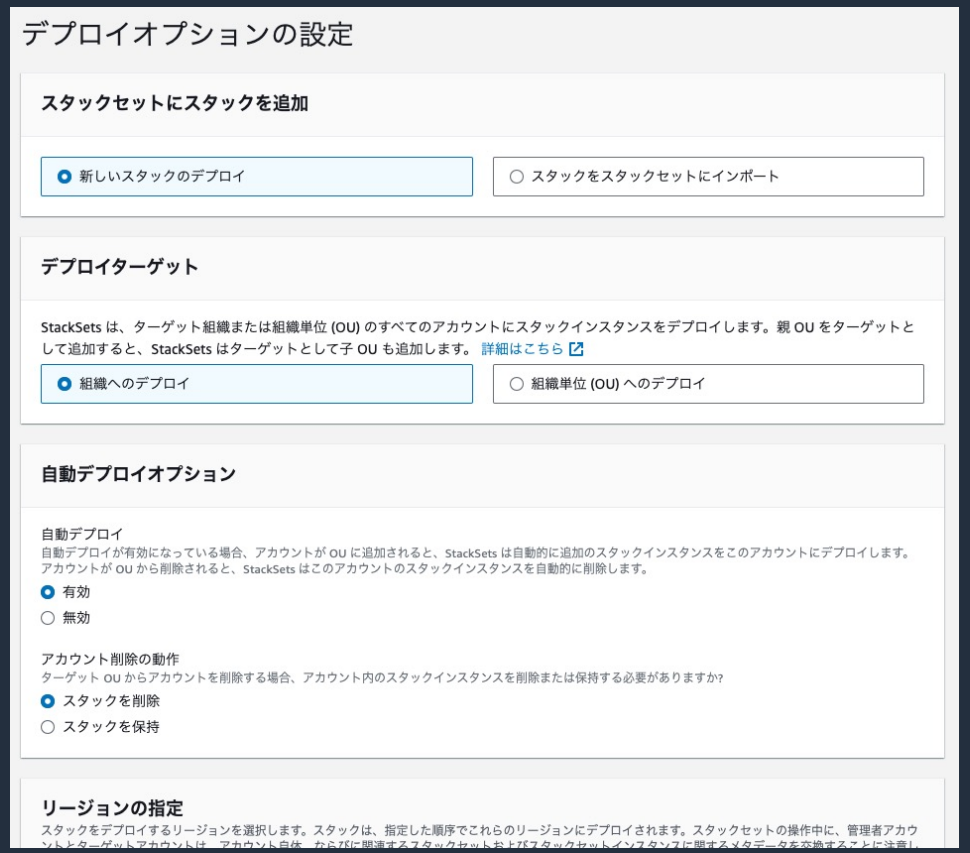
2. StackSets の詳細を指定
StackSetsの名称や説明、
テンプレートのパラメータを指定します。

実行手順 2/2

3. StackSets オプションの設定
タグや実行設定を指定します。



4. デプロイオプションの設定
デプロイ先 (組織・OU・アカウントID) やリージョン、自動デプロイオプション(サービスマネージドの場合)、同時実行するアカウントやリージョン等を指定します。



StackSets を利用する際の考慮事項

デプロイ対象は複数のアカウントの複数のリージョンを指定できる

- Organizations があるとデプロイ対象の指定とロールの管理が楽だが必須ではない
- OUまたは個別アカウントのいずれかを指定することが多い

テンプレートとパラメータ

- テンプレートは全ての環境に同じものが利用される
- パラメータはデプロイ対象に対して一括で指定するが、特定のアカウント×リージョンに対して個別にデプロイする際のパラメータを上書きできる

StackSets 間の連携

- StackSets 同士は依存関係を指定できないため、1アカウントあたり StackSets 1つにしておくと安全

参考：基本設定の例 1/2

AWS ControlTower の設定

- ControlTower ドキュメントに記載の CFn テンプレート
  https://docs.aws.amazon.com/ja_jp/controltower/latest/userguide/controls-reference.html
- ControlTower が生成するスタック
  https://docs.aws.amazon.com/ja_jp/controltower/latest/userguide/account-factory-considerations.html
  ControlTower 環境を作ると CFn 経由でテンプレートを参照できます

ロギングの有効化

- AWS CloudTrail および AWS Configの有効化
- AWS ConfigRulesでCloudTrailが有効であることを確認
- これらの設定は、サンプルテンプレートが StackSets から利用可能
  https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/stacksets-sampletemplates.html

参考：基本設定の例 2/2

各アカウントの管理用 IAM Role

- 例: baseline-admin、baseline-abc などの名称で IAM Role を作成する
- 用途に合わせてAdministratorAccess や、ReadOnlyAccess ポリシーを持つユーザを作成
- ただし… Organizations SCP (または追加のポリシー) で以下を制限
  - Role名に baseline-* がついたロールの変更禁止 (権限昇格の禁止)
  - https://docs.aws.amazon.com/ja_jp/controltower/latest/userguide/mandatory-controls.html#iam-disallow-changes

CloudTrailおよびConfigの設定変更を禁止

- https://docs.aws.amazon.com/ja_jp/controltower/latest/userguide/mandatory-controls.html#cloudtrail-configuration-changes
- https://docs.aws.amazon.com/ja_jp/controltower/latest/userguide/mandatory-controls.html#config-disallow-changes

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

OS上の設定も含めてサーバー構成を管理する

実現したいこと

- CloudFormation でサーバー構成を管理するだけで無く、
  起動したサーバー(EC2)のOS上の各種設定についても一気通貫で管理したい

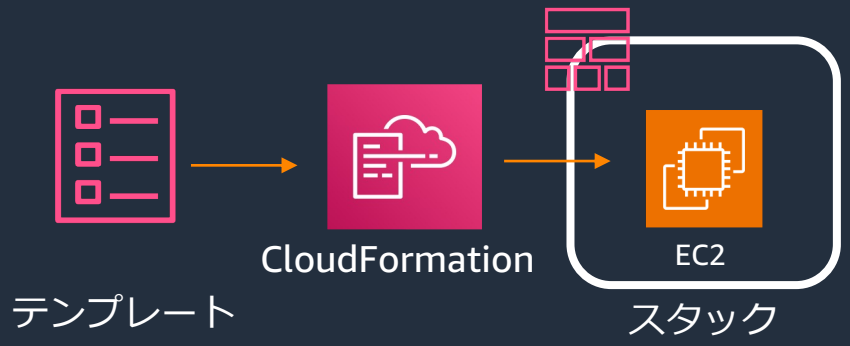
解決策

- CloudFormation と Systems Manager (以降SSM) の State Manager を併用する
- State Manager では、予め用意されている SSM ドキュメント を利用して、
  OS上の設定管理ツール (Ansible や PowerShell DSC 等) を定期的に実行する

※ State Manager について
資料 https://pages.awscloud.com/rs/112-TZM-766/images/AWS-Black-Belt_2023_AWS-SystemsManager-StateManager_0630_v1.pdf
動画 https://youtu.be/vSAbhWZFtKU

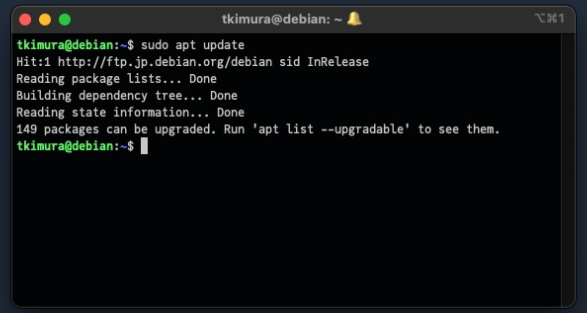
CloudFormation を利用したサーバーの構築の流れ

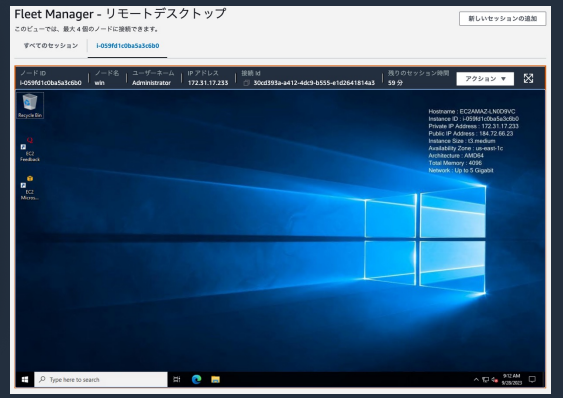
1. EC2 の起動



2. OS上の設定

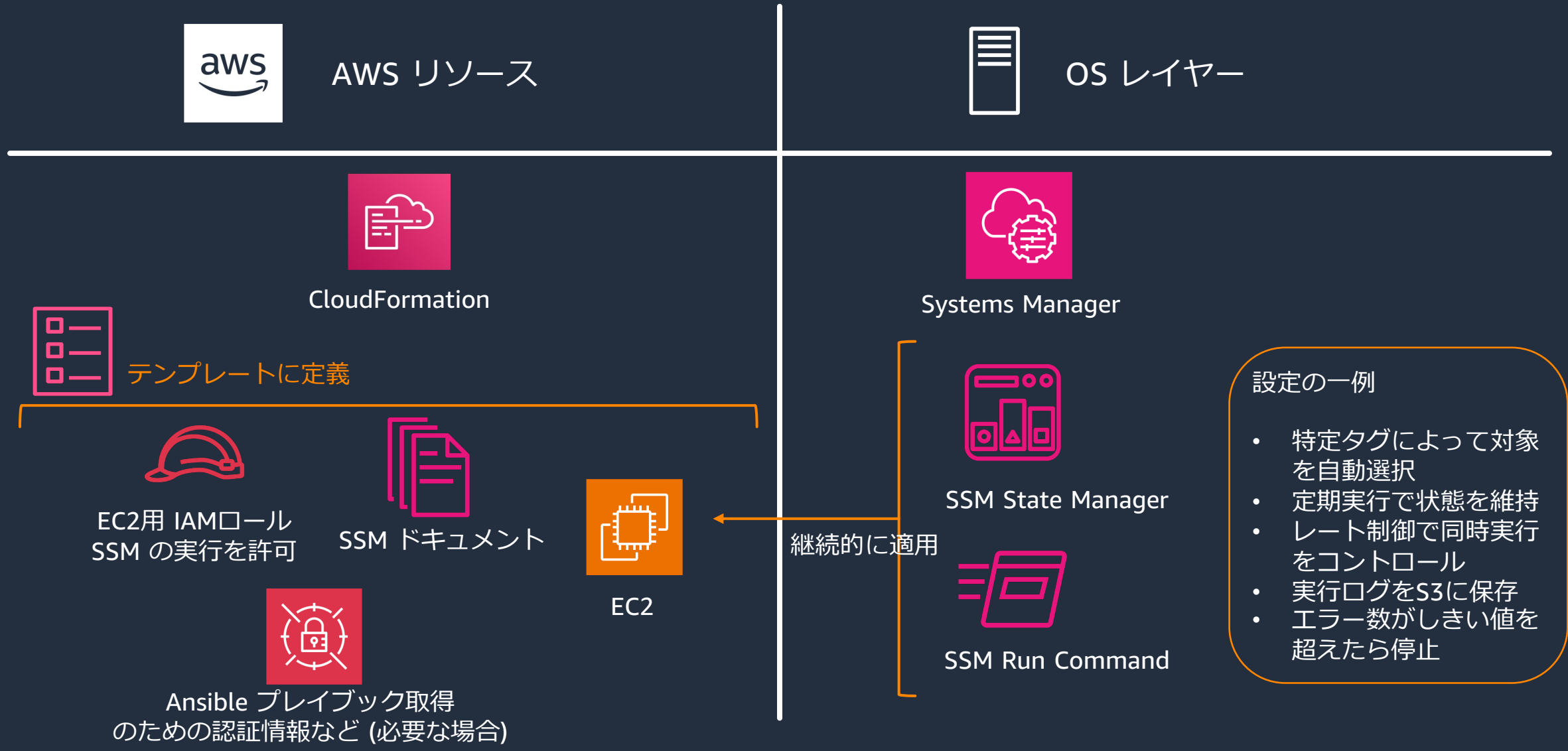
SSH やSSM Session Manager、SSM Fleet Manager などで接続し、
手動 または Ansible や PowerShell DSC などの設定管理ツールで設定を実施





この領域についても一気通貫で管理出来れば、
個々のサーバーに対する個別オペレーションなく構築が完了出来る

実現方法



テンプレートの例

EC2を含むテンプレートに次のような内容を追加する

```
  AnsibleAssociation:
    Type: AWS::SSM::Association
    Properties:
      # ここではAWS-ApplyAnsiblePlaybooksを利用しています
      Name: AWS-ApplyAnsiblePlaybooks
      # CloudFormationは関連付けが行われるのを待ちます
      WaitForSuccessTimeoutSeconds: 120
      Targets:
        - Key: InstanceIds
          Values: [ !Ref EC2Instance ]
      OutputLocation:
        S3Location:
          OutputS3BucketName: !Ref SSMAssocLogs
          OutputS3KeyPrefix: 'logs/'
```

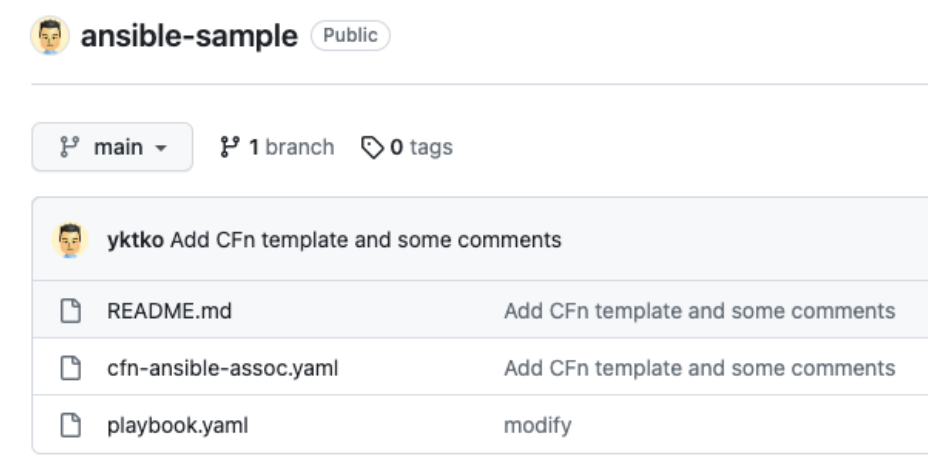
右へ続く

```
      Parameters:
        # GitHubからAnsibleプレイブックを取得します
        SourceType:
          - 'GitHub‘
        SourceInfo:
          - '{"owner":"<Insert your GitHub Owner Name>",
            "repository":"<Insert your GitHub Repo>",
            "path":"",
            "getOptions":"branch:master" }‘
        # Ansibleと依存関係のインストール
        InstallDependencies:
          - 'True‘
        # 実行するプレイブック
        PlaybookFile:
          - 'playbook.yml‘
        ExtraVariables:
          - 'SSM=True‘
        Check:
          - 'False'
```

https://aws.amazon.com/jp/blogs/news/using-state-manager-over-cfn-init-in-cloudformation-and-its-benefits/

参考： サンプルテンプレート

https://github.com/yktko/ansible-sample



cfn-ansible-assoc.yaml を利用してスタックを作成することで、EC2 の起動を行い、SSM State Manager 経由の Ansibleプレイブックの実行の流れを確認出来ます。

起動した EC2 には、SSM Session Manager で接続出来ます。

参考： CFn + cfn-init によるサーバー構成管理

EC2 の UserData にスクリプトを記述する方式

- 初期起動時に指定したスクリプトを実行する
- テンプレートにサーバーの起動処理をまとめて記載可能
- cfn-init はヘルパースクリプトの一つ
- 慣れ親しんだシェルスクリプトが使える
- 従来から利用されてきており情報が豊富

課題

- インスタンス初期起動時しか動作できず
  起動後のメンテナンスは他の仕組みが必要
- CFn テンプレート内にコードを書くことで
  複雑な処理や多数のサーバーの管理ではメンテナンス性に課題
- 処理中のエラーメッセージが
  インスンスのローカルディスク上のログに保持される

CFn テンプレートでの指定例

```
UserData:
  Fn::Base64: !Sub |
    #!/bin/bash -xe
    yum -y update
    yum -y install aws-cfn-bootstrap
    /opt/aws/bin/cfn-init -v ¥
    --stack ${AWS::StackName} ¥
    --resource ServerInstance ¥
    --region ${AWS::Region}
```

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/cfn-init.html

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

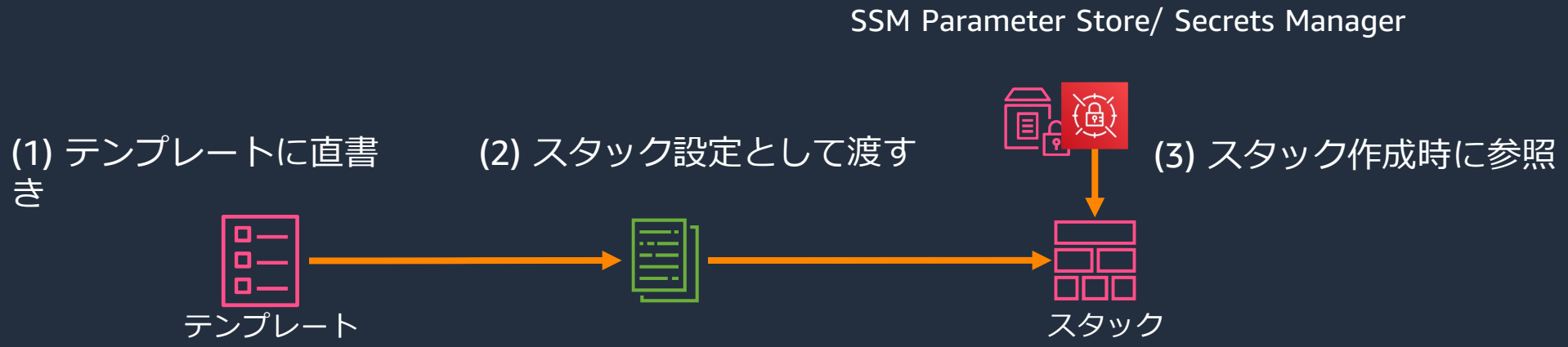
場面に応じてパラメータの指定方法を使い分ける

実現したいこと

- パラメータによって作成するAWSリソースを変更したい
- 例) 同じテンプレートを使って、開発 や 本番 環境を作り分けたい

解決策

- 複数あるパラメータの指定方法を把握し、用途に応じて使い分ける



指定方法 1/2

a. テンプレート直書き

EC2のAMIを指定する場合

```
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: "ami-79fd7eee"
      KeyName: "testkey"
```

テンプレート作成時に決定したい値の例

- 自社専用に用意したEC2のカスタムAMI
- Lambdaのランタイムバージョン
- 等々

直書きすることでレビュープロセスで指摘出来る

b. スタック設定として渡す

デプロイ先の環境名を設定する場合

```
Parameters:
  EnvironmentTag:
    Default: development
    AllowedValues:
      - development
      - staging
      - production
    Type: String
```

スタック作成時に決定したい値の例

- 環境を示すタグ (各リソースのタグに指定)
- マルチAZの要否 (True/False)
- 等々

指定方法 2/2

c. スタック作成時に外部を参照

RDSの認証情報を指定する場合

```
Resources:
  MyDB:
    Type: AWS::RDS::DBInstance
    Properties:
      Engine: MySQL
      EngineVersion: "8.0.34"
      MasterUsername: dbadmin
      MasterUserPassword: '{{resolve:secretsmanager:MyDBAuth:SecretString:password}}'
```

テンプレートに書きたくない値、既存のParameter Store、Secrets Managerから取得したい値

- 認証情報のような機密情報
- リージョンをまたぐスタック同士での値の参照
- その他、既にParameter Store、Secrets Managerで管理されている値を参照したい場合

※ 例では別途 SSM Secrets Manager で設定済みの認証情報を使う形にしていますが、
RDS では ManageMasterUserPassword プロパティを使うことで Secrets Manager へのシークレットの作成も自動的に行われます。
https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/dynamic-references.html

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

最新のAMI IDを取得して指定する

実現したいこと

- 常に最新のAMI IDを利用してEC2を起動したい
- 例) 一時的に利用する環境を CloudFormation で作成する

解決策

- SSMのパブリックパラメータからAMI IDを取得して指定する

テンプレートの例

SSMパラメータタイプを利用し、パブリックパラメータからAMI IDを参照する

```
AWSTemplateFormatVersion: "2010-09-09"

Parameters:
  EC2ImageId:
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref EC2ImageId
      InstanceType: t3.micro
```

例では Amazon Linux を指定していますが、Windows (ami-windows-latest) についても参照可能です。

※ 注意：常に最新の AMI ID を指定できる反面、AMI ID が更新された場合は EC2 の再作成が行われます。
スタック更新の際には、変更セットで意図しない EC2 の再作成が行われないことをご確認ください。

https://docs.aws.amazon.com/ja_jp/systems-manager/latest/userguide/parameter-store-public-parameters-ami.html

よくあるユースケース

1. 手動で作った既存リソースをCFnの管理下に入れる
2. 既存のスタックを分割する(テンプレートのリファクタリング)
3. 各AWSアカウントに基本設定を展開する
4. OS上の設定も含めてサーバー構成を管理する
5. 場面に応じてパラメータの指定方法を使い分ける
6. 最新のAMI IDを取得して指定する
7. CloudFormationで作成したリソースのバックアップを取得する

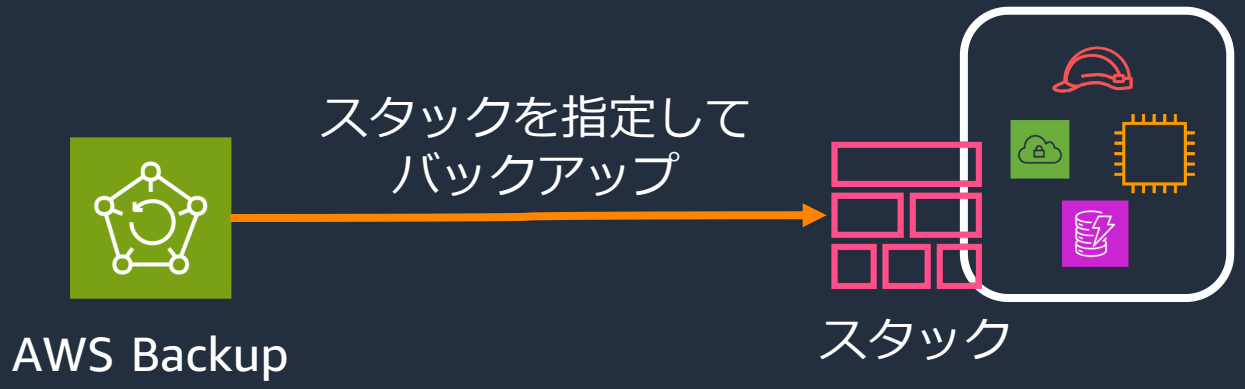
CloudFormation で作成したリソースのバックアップを取得する

実現したいこと

- スタックに含まれるデータベースやファイルシステム等のステートフルコンポーネントのバックアップを取得したい
- スタック全体をまとめてバックアップ対象として設定したい

解決策

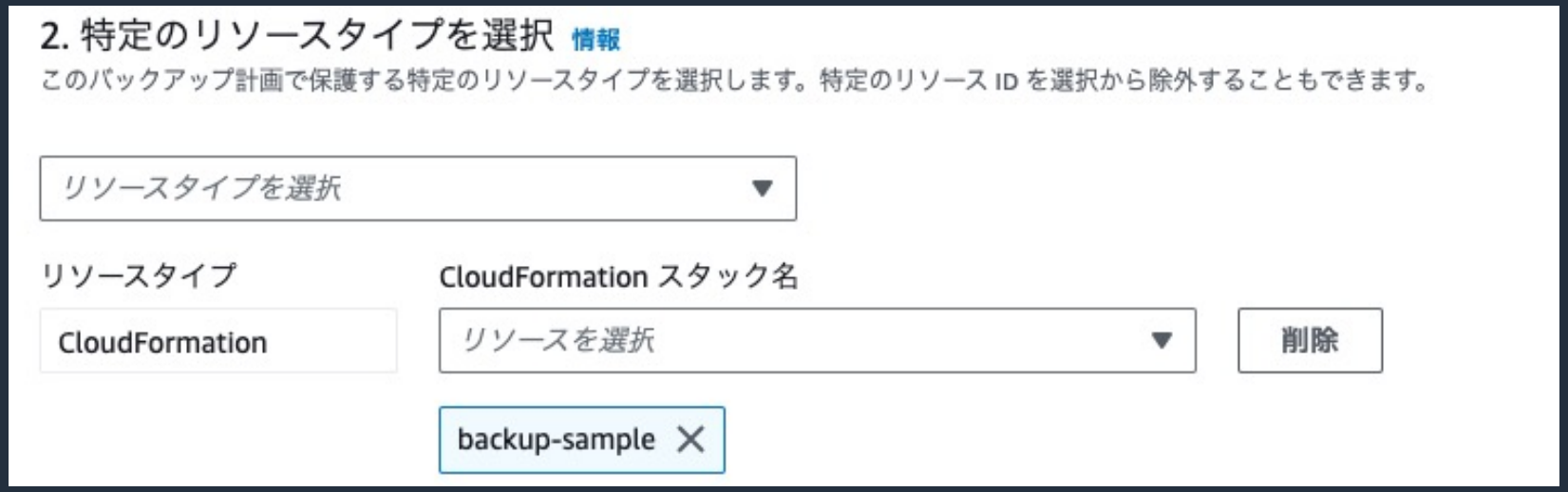
- AWS Backupを利用して、バックアッププランを作成する



https://aws.amazon.com/jp/blogs/news/new-for-aws-backup-protect-and-restore-your-cloudformation-stacks/

バックアップの取得

バックアッププランの作成を行い、CloudFormation スタックを選択する



- バックアップには、テンプレートや各パラメータの他、
  AWS Backupがサポートする、RDSやS3等の全てのステートフルコンポーネントが含まれます
- IAM RoleやVPC等のステートレスコンポーネントも含まれます
- スケジュールや保存期間、ライフサイクルルール等、他リソース同様に指定可能

https://docs.aws.amazon.com/ja_jp/aws-backup/latest/devguide/applicationstackbackups.html

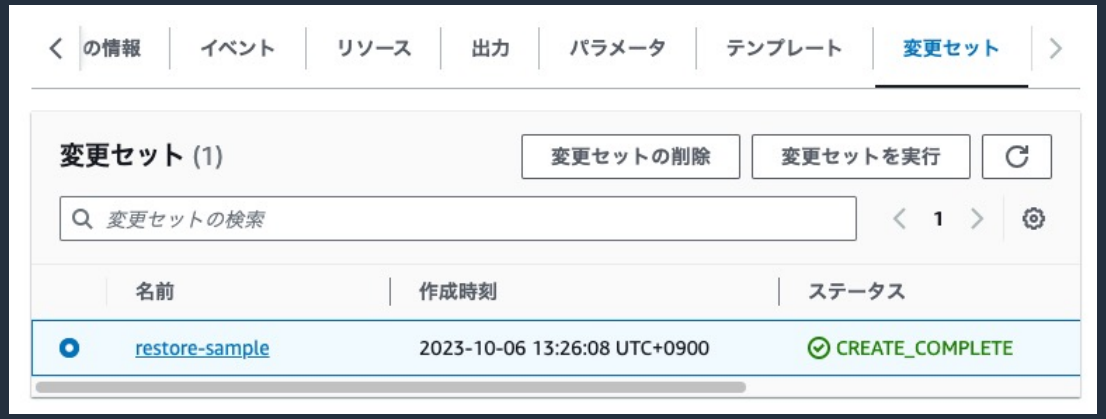
バックアップから復元

(1) バックアップから復旧ポイントを選択して復元
リソース単独またはスタックを選択可能

リソース単独の場合、新たに名称を指定したり、
既存のS3バケットに復元したり等を選択可能。

スタックの場合、スタックそのものを復元するため、
データベースなどのステートフルコンポーネントは、
中身が空の状態で再作成される。
そのため、データそのものは別途復元が必要となる。

(2) 作成されたスタックの変更セットを確認して実行
※ スタックを復元する場合のみ



CloudFormationにスタックと変更セットが作成され、
変更セットの実行待ちになる。

実行すればスタックに含まれるリソースが作成される。
なおスタックの復元時、リソースの物理IDが重複する場合、
復元時にエラーとなります。
テンプレート作成の段階で、重複しないような対策が必要です。

https://docs.aws.amazon.com/ja_jp/aws-backup/latest/devguide/restore-application-stacks.html

よくある質問

よくある質問

1. セキュリティグループが循環参照になって作成出来ない
2. 依存関係が残っていてスタックの削除に失敗する
3. CloudFormation で注意が必要な Quota

循環参照になってセキュリティグループが作成できない

課題

- AWS::EC2::SecurityGroup で定義しても循環参照になってしまいスタックが作れない
  - 例1）セキュリティグループAにおいて、
    ソース が セキュリティグループA のインバウンドアクセスを許可したい
    (デフォルトセキュリティグループと同じ。セキュリティグループAが設定されているもの同士は通信OK。)
  - 例2) セキュリティグループAおよびBがあり、
    A はソースが B のインバウンドアクセスを許可、
    B はソースが A のインバウンドアクセスを許可したい

解決策

- AWS::EC2::SecurityGroup で SecurityGroup を作成した後に、
  AWS::EC2::SecurityGroupIngress でインバウンドルールを定義する

  ※１つのテンプレート内で表現できない場合は、別スタックにするかCLIやカスタムリソースなど手続き型の処理を行う

サンプル

課題の例1

```
sgSelfReference:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Self Reference

sgSelfReferenceAllowAll:
  Type: AWS::EC2::SecurityGroupIngress
  Properties:
    IpProtocol: -1
    GroupId: !GetAtt sgSelfReference.GroupId
    SourceSecurityGroupId: !GetAtt sgSelfReference.GroupId
```

課題の例2

```
sgCrossReference1:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cross Reference - 1

sgCrossReference2:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cross Reference - 2

sgCrossReference1AllowAllFrom2:
  Type: AWS::EC2::SecurityGroupIngress
  Properties:
    IpProtocol: -1
    GroupId: !GetAtt sgCrossReference1.GroupId
    SourceSecurityGroupId: !GetAtt sgCrossReference2.GroupId

sgCrossReference2AllowAllFrom1:
  Type: AWS::EC2::SecurityGroupIngress
  Properties:
    IpProtocol: -1
    GroupId: !GetAtt sgCrossReference2.GroupId
    SourceSecurityGroupId: !GetAtt sgCrossReference1.GroupId
```

依存関係が残っていてスタックの削除に失敗する

課題

- スタックを削除するとリソースが利用中のため途中でエラーになる
  - CloudFormation で作ったリソースを別スタックやスタック外で使ってしまった場合
    ENI周りやセキュリティグループなど
  - CloudFormation で作ったS3バケットにデータが残っている場合

解決策

- 依存関係を解消してスタックを削除する
- 消せないリソースを保持してスタックを削除する
- どこに依存関係があるかわからない場合
  - AWS Configで削除に失敗するリソースを探し、依存関係を調べる
  - AWS CLI の describe- で始まるコマンドを実行して確認する

  参考: VPCの依存関係を確認する方法
  https://aws.amazon.com/jp/premiumsupport/knowledge-center/troubleshoot-dependency-error-delete-vpc/

リソース保持の例

CloudFormation で注意が必要な Quota

課題

- Quota 上限に引っかかりエラーが発生してしまう

解決策

- 予め注意が必要なところを確認しておく
  https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html
- 注意が必要な項目
  - テンプレートで宣言できるリソースの最大数 ... 500
  - テンプレート本文の最大サイズ ... 51,200 bytes
  - S3上のテンプレート本文の最大サイズ ... 1 MB
- ネストスタックを利用することで上限を回避することも出来る

Thank you!

AWS Black Belt Online Seminar とは

- 「サービス別」「ソリューション別」「業種別」などのテーマに分け、アマゾン ウェブ サービス ジャパン合同会社が提供するオンラインセミナーシリーズです
- AWS の技術担当者が、 AWS の各サービスやソリューションについてテーマごとに動画を公開します
- 以下の URL より、過去のセミナー含めた資料などをダウンロードすることができます
  - https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/
  - https://www.youtube.com/playlist?list=PLzWGOASvSx6FIwIC2X1nObr1KcMCBBlqY

ご感想は X (Twitter) へ！ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では資料作成時点のサービス内容および価格についてご説明しています。AWS のサービスは常にアップデートを続けているため、最新の情報は AWS 公式ウェブサイト (https://aws.amazon.com/) にてご確認ください
- 資料作成には十分注意しておりますが、資料内の価格と AWS 公式ウェブサイト記載の価格に相違があった場合、AWS 公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます
- 技術的な内容に関しましては、有料の AWS サポート窓口へお問い合わせください
- 料金面でのお問い合わせに関しましては、カスタマーサポート窓口へお問い合わせください (マネジメントコンソールへのログインが必要です)