CloudShift

Stylez

株式会社スタイルズ

2023年3月7日

CloudFormationで始める IoTシステムのIaC

aws partner network

aws PARTNER Select Tier Services

RANCHER PARTNER NETWORK

ZABBIX CERTIFIED PARTNER

Nextcloud Platinum Partner

vmware PARTNER PROFESSIONAL SOLUTION PROVIDER

自己紹介

- **皆川　秀洋（Minagawa Hidehiro）**

- **株式会社スタイルズ**
  - CI/CDパイプライン、データETL実装、IaCを担当
- **好きな技術領域**
  - AI
  - IaC

本日、お話する内容

対象

- IaCという言葉をよく聞くが、どういうものかよくわからない人
- IaCツールの導入を考えている人
- IoTシステムのコード化に興味がある人

ゴール

- IaCはどういうものか理解する
- IaCツール導入のメリットとデメリットについて理解する
- IoTシステムコード化のワークフローがどういうものか理解する

本日のアジェンダ

- IaCとは
- IaCのメリット・デメリット
- IoTシステムのコード化のワークフロー紹介

IaCとは

IaCとは

- Infrastructure as Codeの略
  - 「インフラのコード化」という意味
  - IaCの考え方ではクラウドサービス全体をインフラとみなすのでインフラの範囲が広い
- サーバーの構成やクラウドサービスの構築をコードを基にして行おうという考え方

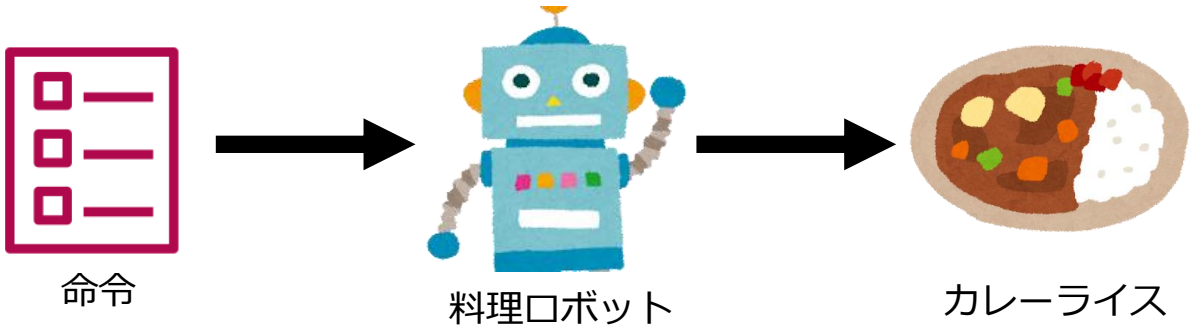
ITシステム近年における変化

	従来のITシステム	最近のITシステム
おおまかな構成	アプリケーションソースコード ＋ インフラ	アプリケーションソースコード ＋ インフラコード
GitHub等でバージョン管理する対象	アプリケーションソースコードのみ	アプリケーションソースコードとインフラコード
インフラ構築	担当者がパラメーターシートと手順書を元に手作業で構築	コマンドラインや画面から自動で構築

- こういった取り組みがインフラのコード化（IaC）

IaCの特徴：宣言型

- プログラミングのパラダイムとして「命令型」（imperative）と「宣言型」（declarative）というものがある
- IaCツールは宣言型のものが多い
- 例）ロボットが人間の命令を元に料理を作る場合



命令型と宣言型

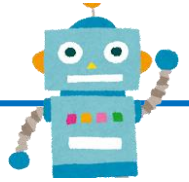
- 命令型
  - 「どのように作るか」が書いてある
    - １．フライパンで油を温める
    - ２．野菜を切って炒める
    - ３．ルーを入れる

- 宣言型
  - 「最終結果」のみが書いてある
    - 中辛のルー
    - ライス100ｇ
    - ジャガイモ多め

IaCとは？カレーライスを例に

- 宣言型の良さ
  - 「どのようにして作るか」の部分は考えなくていい
  - カレーの例：料理ロボットが考える
  - CloudFormation：CloudFormationの内部ロジック

```
材料:
  油:
    種類:
      - サラダ油:
          # 単位はml
          量: 5
      - ごま油:
          量: 10
          会社: ○○精油

  ジャガイモ:
    前処理:
      切り方:
        形: ぶつ切り
    #後処理:

  にんじん:
    産地: 静岡県産
    # 単位はg
    量: 20

  肉:
    種類: 豚肉
    量: 50

  フライパン:
    材質: 鉄
    新しさ: 普通
    半径: 30
```



1．サラダ油とごま油を混ぜて油を作る
2．フライパンに油を敷く
3．玉ねぎを１㎝の薄さで切る
・
・
・

宣言型の命令

命令を手続きに変換

カレーライス

IaCとは

- インフラ・アズ・コードの略
- 従来では手のかかっていたインフラの構築がコードを元に自動でできるようになった
- IaCは宣言型のコードを用いる場合が多く、宣言型の場合 「どのように」の部分は考えなくてよい

IaC導入のメリット・デメリット

IaC導入のメリット：インフラ構築コストの低下

	IaCなしのインフラ構築	IaCありのインフラ構築
手順書作成コスト	高	低
インフラコード作成コスト	なし	高
作業員の確保・教育・コミュニケーションコスト	高	低
ヒューマンエラーが起こる可能性	高	低
複数環境時の構築コスト	高	低

- インフラ構築の準備コスト
  - IaCなし：手順書作成
  - IaCあり：インフラコードの作成
- インフラ構築の実行コスト
  - IaCなし＞IaCあり

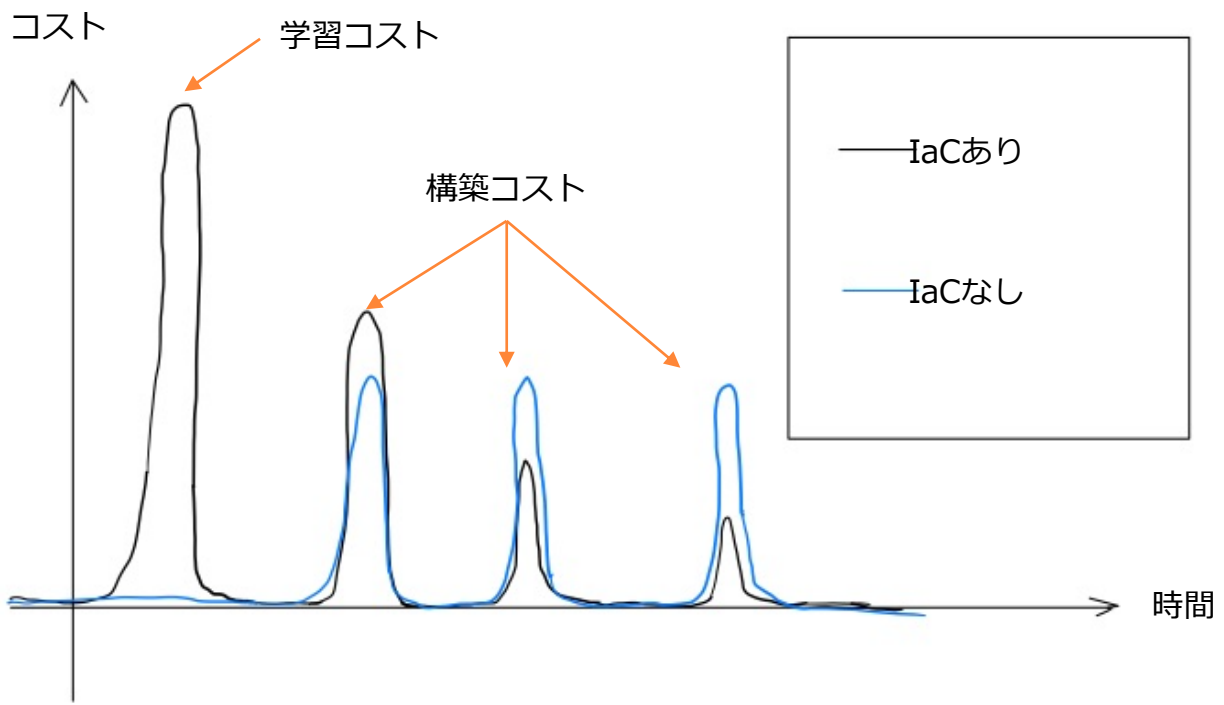
IaC導入のデメリット

- IaCに取り組む体制が作りにくい
  - IaCは新しくかつユーザー数が限られている
- インフラコード作成の難易度低くない
  - （AWSシステムの場合）AWSに対する知識、コード対象となるインフラ固有の知識だけでなく、それをコード化する際の調査、試行錯誤が必要
  - 学習環境整備されているとは言えない（Terraformは例外）

    例：日本語のドキュメント、StackOverflow等の質問サイトでの投稿数の少なさ

IaCの長期的に見たコスト傾向

- IaCは初期に大きなコスト
  - 学習コスト
  - 初期の構築時
    - インフラのコード化に関わるコスト大
- それ以降は下がる傾向
  - 同じ構成なら構築コストほぼゼロ
  - 違う構成でも構築のコスト低下
    - インフラコードの使いまわしが可能なケース



- 長期的な視点で見るとIaC導入のコスト削減効果増大

参考:「Infrastructure as Code Patterns and Practices」Manning

IaC導入のメリット・デメリット

- メリット
  - インフラ構築の実行コストが大きく下がる
- デメリット
  - 導入が難しい
  - 導入初期のコストリターンだけを見ると導入に踏み切れない

IoTシステムコード化のワークフロー

IaCツールの３つのカテゴリー

カテゴリー名	例	コード化対象	備考
構成管理ツール	Ansible, Chef, Puppet	サーバーOS、ミドルウェアの設定	・IaCツールの中では比較的古い ・EC2やGreengrassデバイスなどの設定で使用（口頭でセミナー範囲外説明）
プロビジョンツール	AWS CloudFormation, Azure ARM, GCP CDM, Terraform	クラウドサービス	・本セミナーで扱う
イメージビルダー	Docker, EC2 Image Builder	コンテナイメージ、サーバーイメージ	

- それぞれ得意分野が違う
- システム全体のコード化にはプロビジョンツールだけでは足りず、他のカテゴリーのIaCツールも必要になることが多い
- IoTシステムだとセンサーデバイスのIoT Coreへの登録やゲートウェイデバイスへのGreengrassのインストールに構成管理ツールが必要になってくる

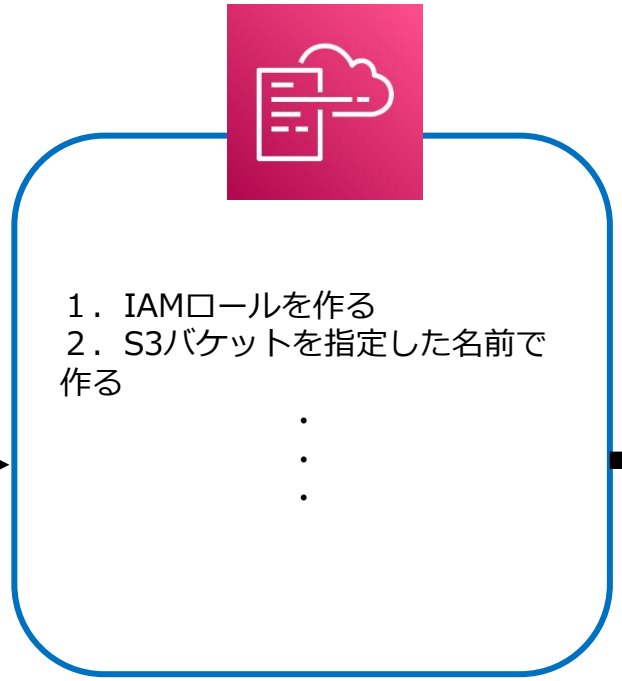
参考：「Infrastructure as Code Patterns and Practices」Manning

CloudFormationとは？そのしくみ

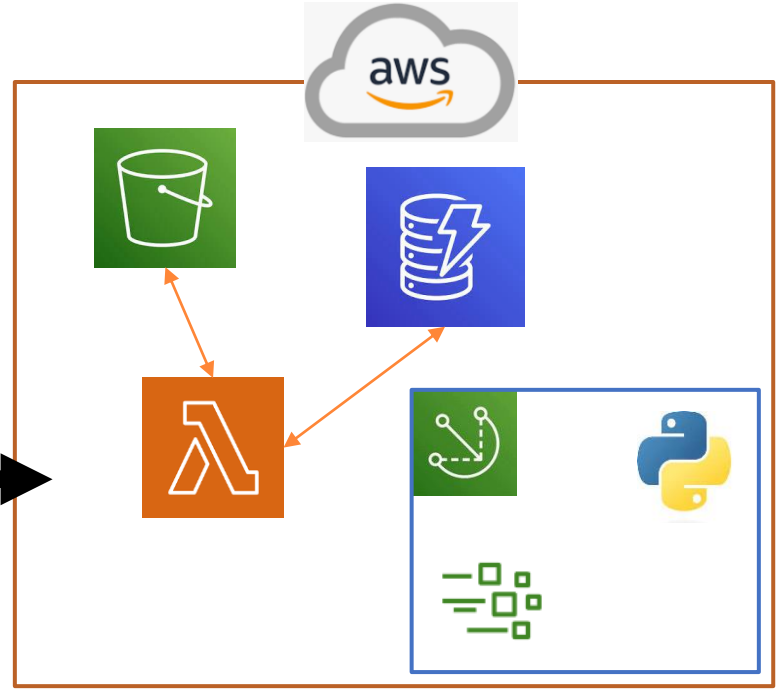
- 宣言型のテンプレートが一連のAPIリクエストに変換され、AWSシステムが作成される

```yaml
MyKinesisStream:
  Type: 'AWS::KinesisFirehose::DeliveryStream'
  DependsOn: MyKinesisRole
  Properties:
    DeliveryStreamName: minagawa-kinesis-strean
    ExtendedS3DestinationConfiguration:
      BucketARN: !GetAtt MyS3Bucket.Arn
      RoleARN: !GetAtt MyKinesisRole.Arn
      BufferingHints:
        IntervalInSeconds: 60
        SizeInMBs: 1
      CompressionFormat: UNCOMPRESSED
      Prefix: firehose/
      CloudWatchLoggingOptions:
        Enabled: true
        LogGroupName: my-log-group
        LogStreamName: my-log-stream

MyS3Bucket:
  Type: 'AWS::S3::Bucket'
  Properties:
    BucketName: minagawa-bucket-29847247294

MyKinesisRole:
  Type: 'AWS::IAM::Role'
  Properties:
    RoleName: minagawa-kinesis-role
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
```

宣言型のテンプレート



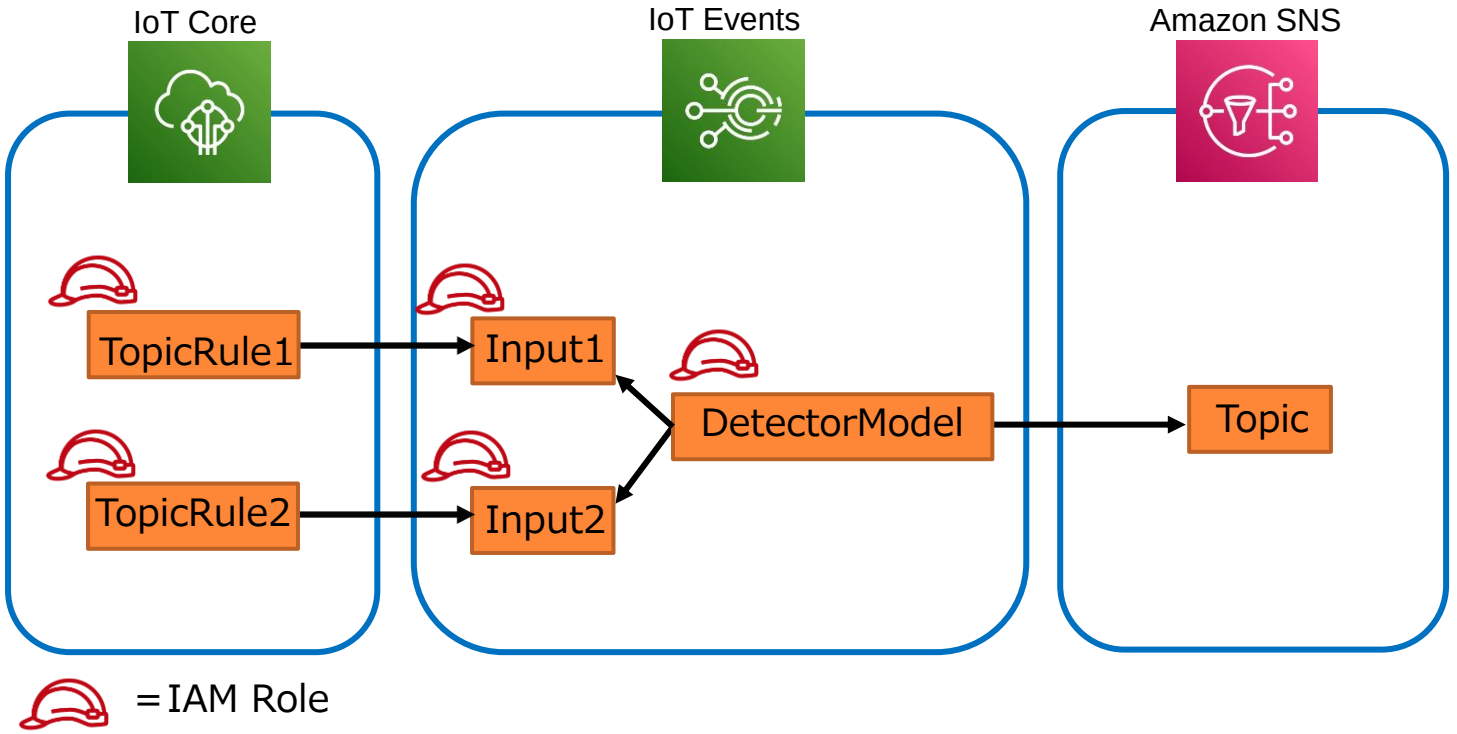
テンプレートを
一連のAPIリクエストに変換



スタック

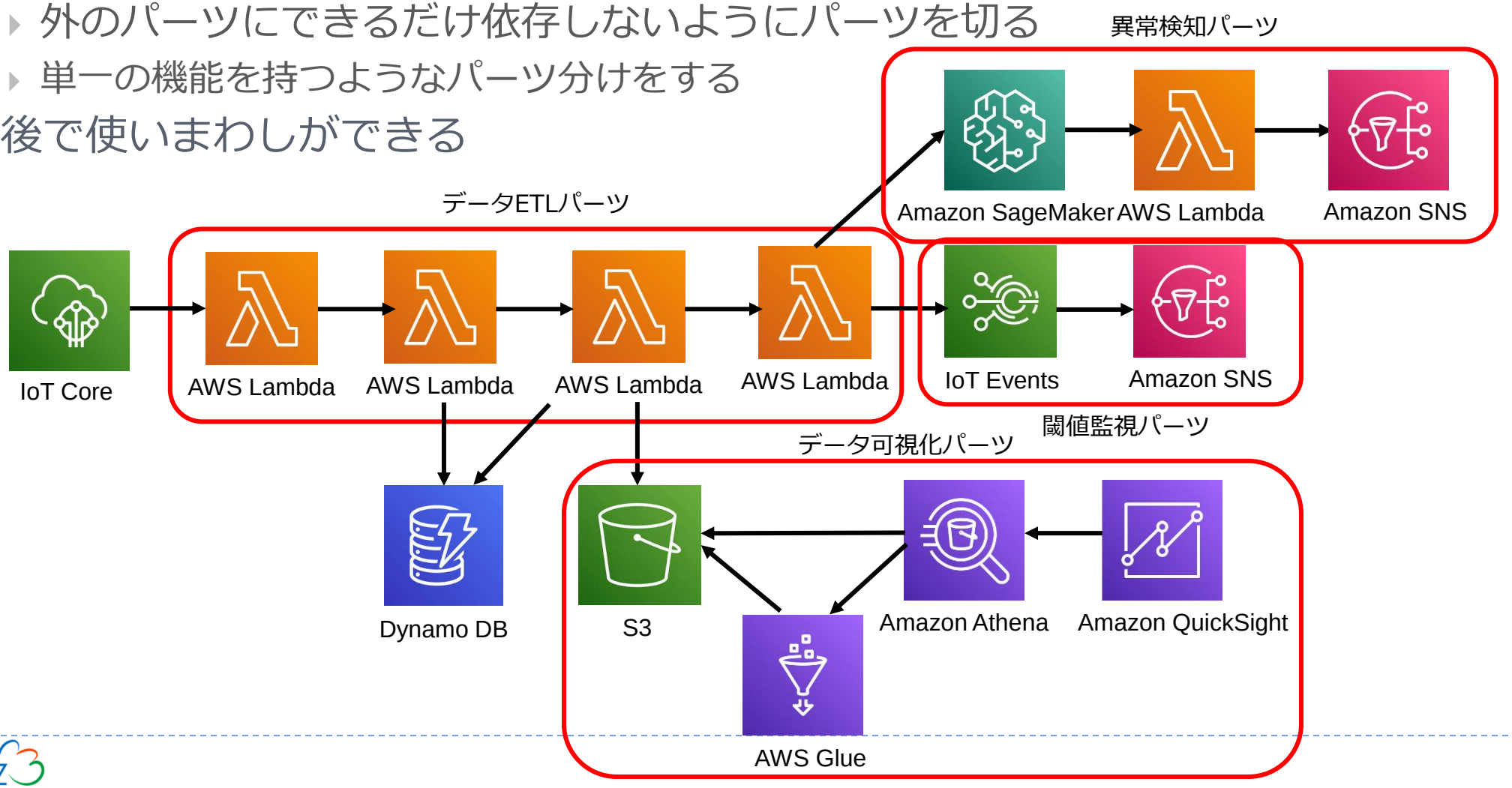
IoTシステムコード化のワークフロー：閾値監視機能のCloudFormationを使った構築

- 本セミナーで説明するもの：閾値監視機能
  - TopicRuleが特定のMQTTトピックに届いたデータをInputに送信
  - DetectorModelで設定された閾値を超えるとTopicに指定したアドレスにメール通知



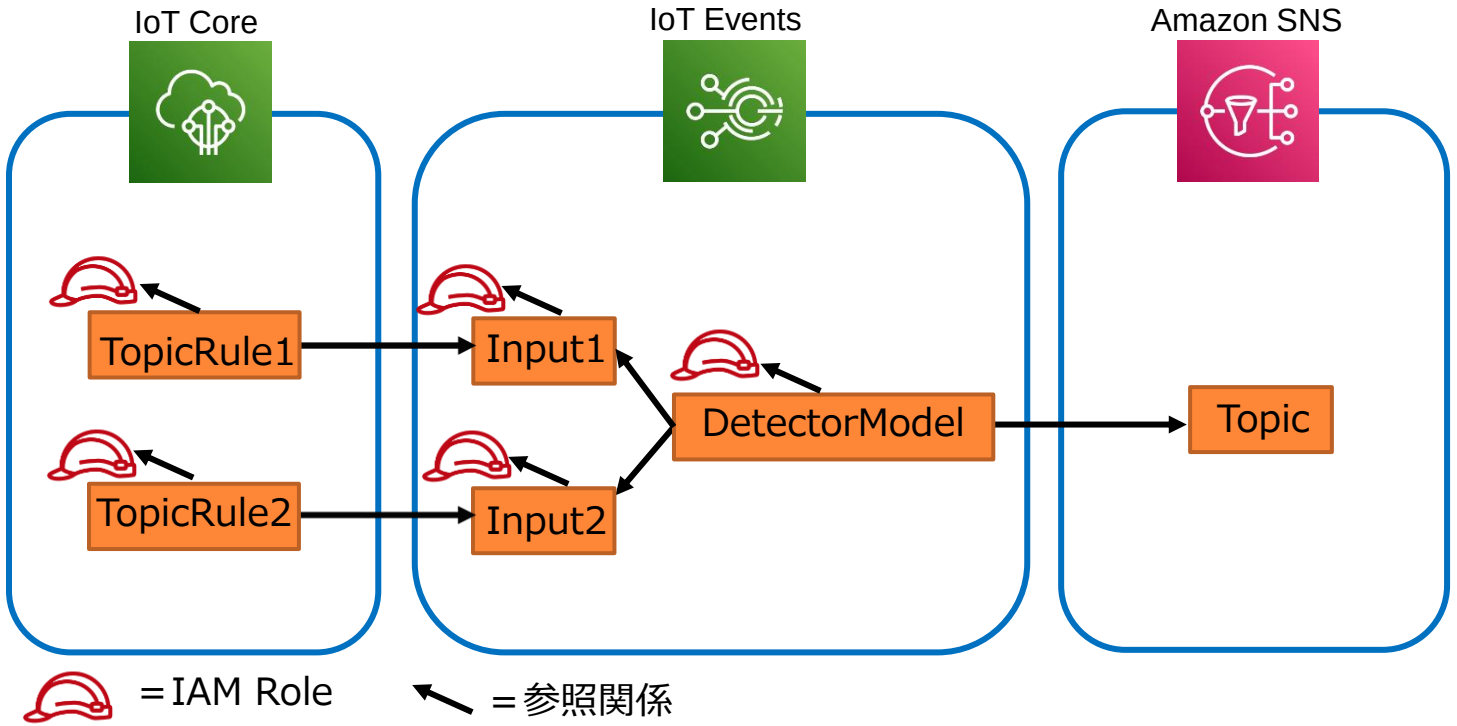
閾値監視機能のコード化のワークフロー：ステップ 1

- システム全体をパーツに分ける
  - 外のパーツにできるだけ依存しないようにパーツを切る
  - 単一の機能を持つようなパーツ分けをする
- 後で使いまわしができる



閾値監視機能のコード化のワークフロー：ステップ 2

- AWSはサービスの単位より一つ細かいリソースという単位が存在する
  - コード化の際は作成したいパーツのリソースの構成がどうなっているかを理解することが大事
    ※特に参照関係



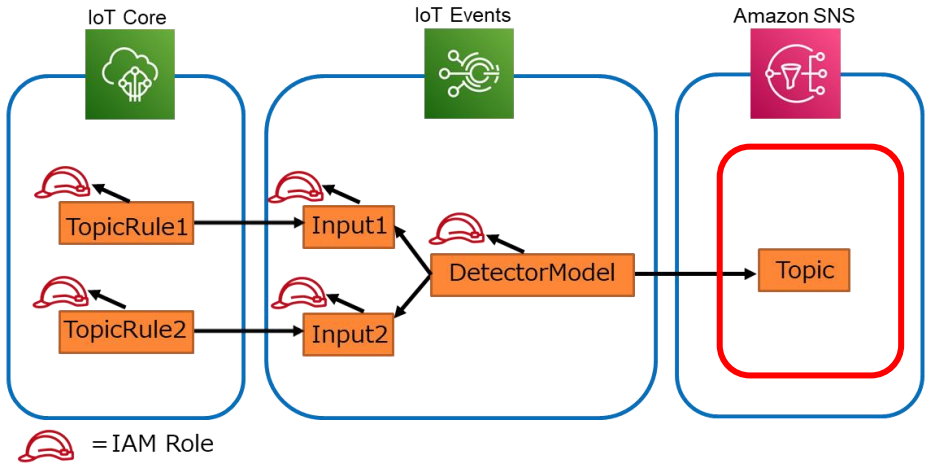
閾値監視機能のコード化のワークフロー：ステップ 3

- テンプレートの作成
  - リソース
    - パーツ内のすべてのリソースを記述
  - プロパティ
    - リソースの細かい設定項目

```
#####################################
#              SNS                  #
#####################################
  SNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: iot-webinar-minagawa@outlook.com
          Protocol: email

#####################################
#             TopicRule1            #
#####################################
  SensorValueTopicRule:
    Type: AWS::IoT::TopicRule
    Properties:
      RuleName: webinar_topic_rule_sensor_value
      TopicRulePayload:
        AwsIotSqlVersion: "2016-03-23"
        RuleDisabled: false
        Sql: >-
          SELECT
            *
          FROM
            'webinar/sensor/value'
        Actions:
          - IotEvents:
              InputName: !Ref SensorValueInput
              RoleArn: !GetAtt SampleTopicRuleRole.Arn
```

閾値監視機能のコード化のワークフロー：ステップ 3

- リソースの説明：SNS Topic
  - 設定項目
    - Subscription
      - アラート通知をする先のメールアドレス
      - 変数化して外出しすることでプログラムによる制御が可能（→後述）



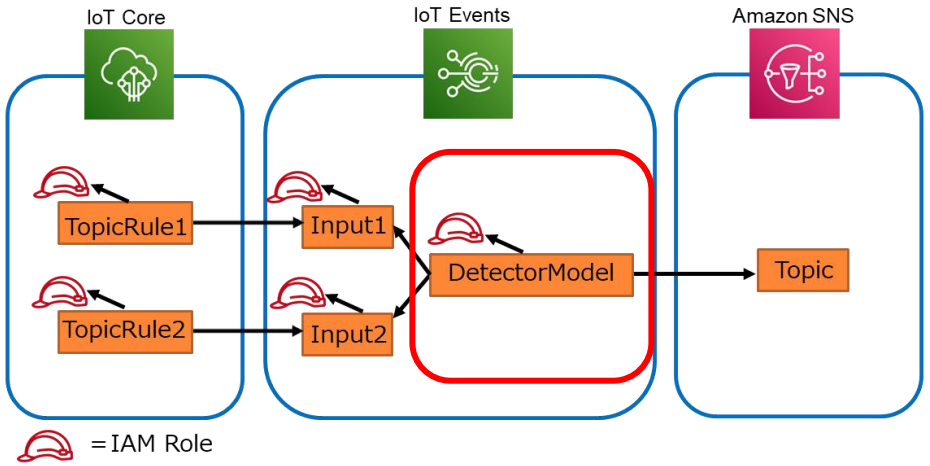

```yaml
Parameters:
  EmailAddress:
    Type: String
    Default: default-address@example.com

Resources:
#########################################
#              SNSTopic                 #
#########################################
  SNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: iot-webinar-minagawa@outlook.com
          Protocol: email
```

閾値監視機能のコード化のワークフロー：ステップ３

```
#####################################
#            DetectorModel          #
#####################################
  GeneralSensorModel:
    Type: AWS::IoTEvents::DetectorModel
    Properties:
      DetectorModelName:
      Key: sensorIDType
      EvaluationMethod: SERIAL
      RoleArn: !GetAtt DetectorModelRole.Arn
      DetectorModelDefinition:
        InitialStateName: Excluded
        States:
          - StateName: Normal
            OnInput:
              Events:
```

```
- EventName: minagawa_test
  Actions:
    - SetVariable:
        VariableName: test_var1
        Value: "'${$input.                    .sensorID}'"
    - Sns:
        TargetArn: !Ref SNSTopic
        Payload:
```

- リソースの説明：IoT Events DetectorModel
  - Ref関数でSNS Topicを参照している
  - モデル定義（DetectorModelDefinition）は以下の手順が楽
    - GUIで作成
    - JSONでエクスポート
    - モデル定義のNotNull制約に合わせて項目を削除



CloudFormation：２つの構築方法

- テンプレートの作成が終わったら構築は楽
  - 構築方法は２つ存在する

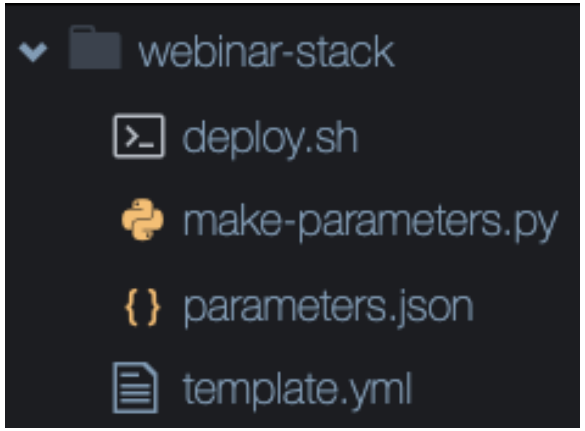
	コマンドライン	コンソール画面
必要なファイル	・テンプレート用YAMLファイル ・デプロイ用Shellスクリプト（※） ・変数用JSONファイル（※） ・変数生成用Pythonファイル（※）	テンプレート用YAMLファイルのみ
プログラムを使った変数の生成	できる	できない
途中経過、エラー内容	表示されない	詳細に表示

- 「構築自体はコマンドラインから、途中経過・エラー内容はコンソール画面で確認」がおすすめ

（※）厳密にはこれらのファイルがなくても構築はできるがワークフローが複雑になる

コマンドラインから構築する際のフォルダ構成

- テンプレートファイル
  - YAMLファイル
- 変数用ファイル
  - JSONファイル
  - JSON作成用のPythonファイル
    - 例）リソース名の接頭辞にuuidをつける
- デプロイ用スクリプト
  - テンプレートをCloudFormationに渡すための Shellスクリプト



Shellスクリプト参考：「The Good Parts of AWS」

変数用Pythonスクリプト参考：「Infrastructure as Code Patterns and Practices」Manning

構築手順

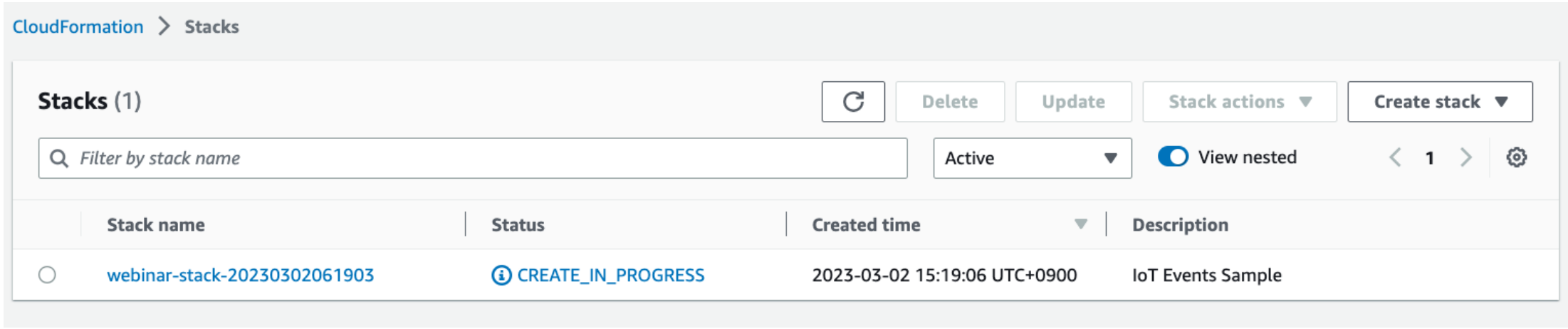
- コマンドラインから構築を行う
- （必要に応じて）デプロイ用Shellスクリプトで作成リージョンやスタック名を確認
- コマンドラインからデプロイ用のShellスクリプトを起動

途中経過確認

- コマンドラインからは詳細な情報はわからない

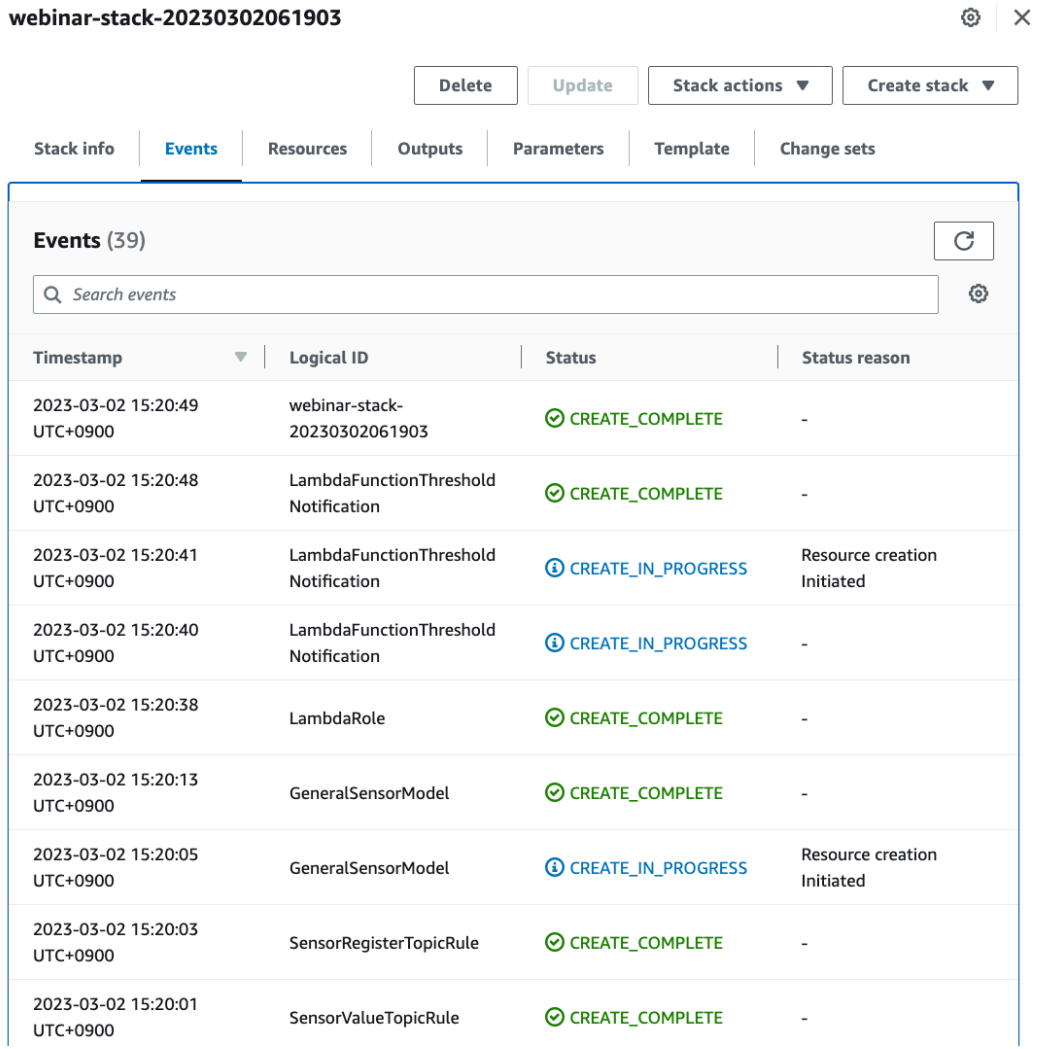
```
============ Deploying template ============

Waiting for changeset to be created..
Waiting for stack create/update to complete
```

- コンソール画面からはわかる
  - AWSコンソールのCloudFormationのページへ行き、該当するスタックをクリック



途中経過確認

- コンソール画面からは詳細情報が確認できる
  - 進捗状況の確認が可能
  - エラー内容の確認も可能
- この閾値監視機能は30秒ほどで構築が完了する



まとめ

まとめ

- IaCとは
  - 宣言型のファイルを用いたインフラのコード化
- メリット、デメリット
  - 従来、様々な種類のコストがかかっていたインフラの構築を自動化できる
  - 導入初期の学習やインフラコードの作成に関わるコストは大きいが、長期的にはIaC導入のコスト削減効果が増大していく
- IoTシステムコード化のワークフロー
  - ワークフローは以下の３ステップ
    - システムをパーツに分ける
    - リソースの構成を理解する
    - テンプレートを作成する

実績豊富なエンジニア集団の技術と開発ツールで「開発期間/コスト削減」「品質向上」を実現

株式会社スタイルズ

https://www.stylez.co.jp

東京都千代田区神田小川町1-2　風雲堂ビル6階

Tel:03-5244-4111

オープンソースソフトウェア推進