ワークフローエンジンよもやま噺

と言っていたなあれは嘘だ
(ごめんなさい)

大量のcronジョブを AWS StepFunctionsに 移行できなかった噺

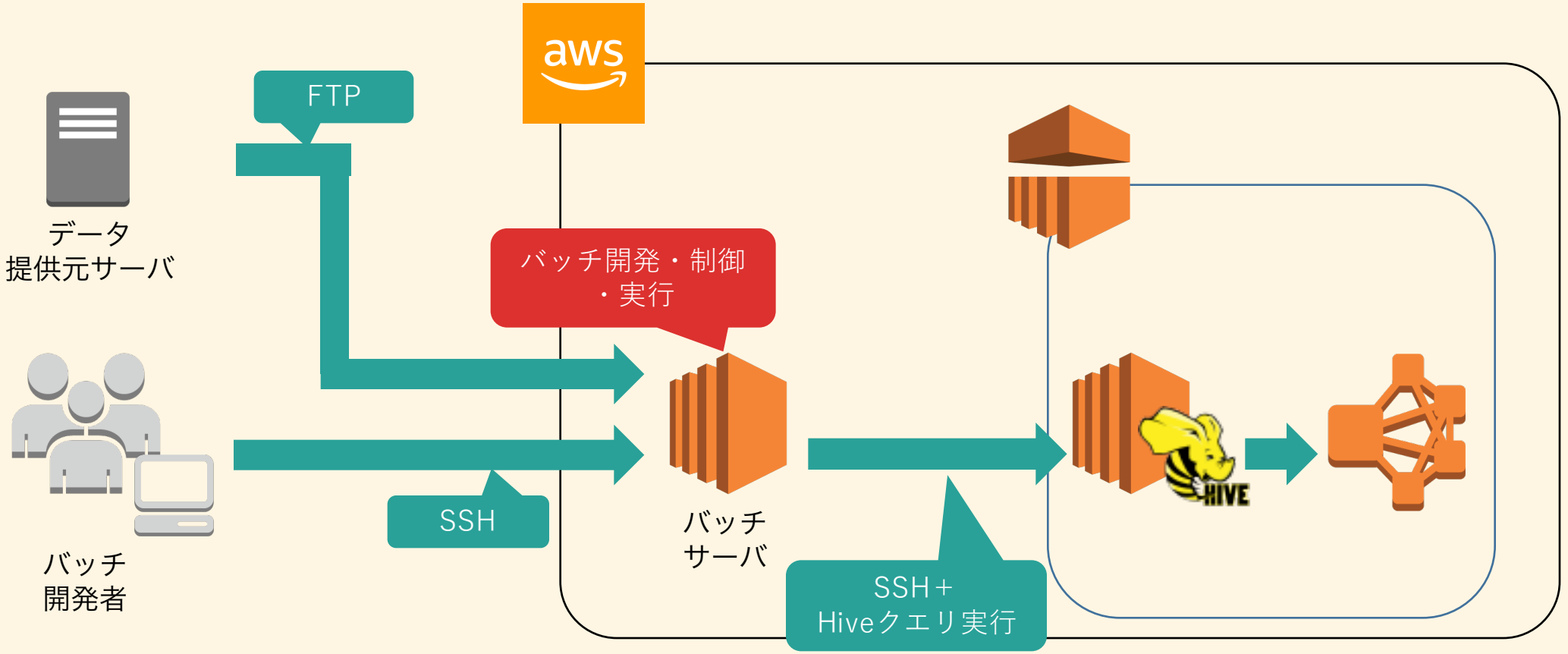
Recruit Technologies Co.,Ltd

Takumi Kitazawa

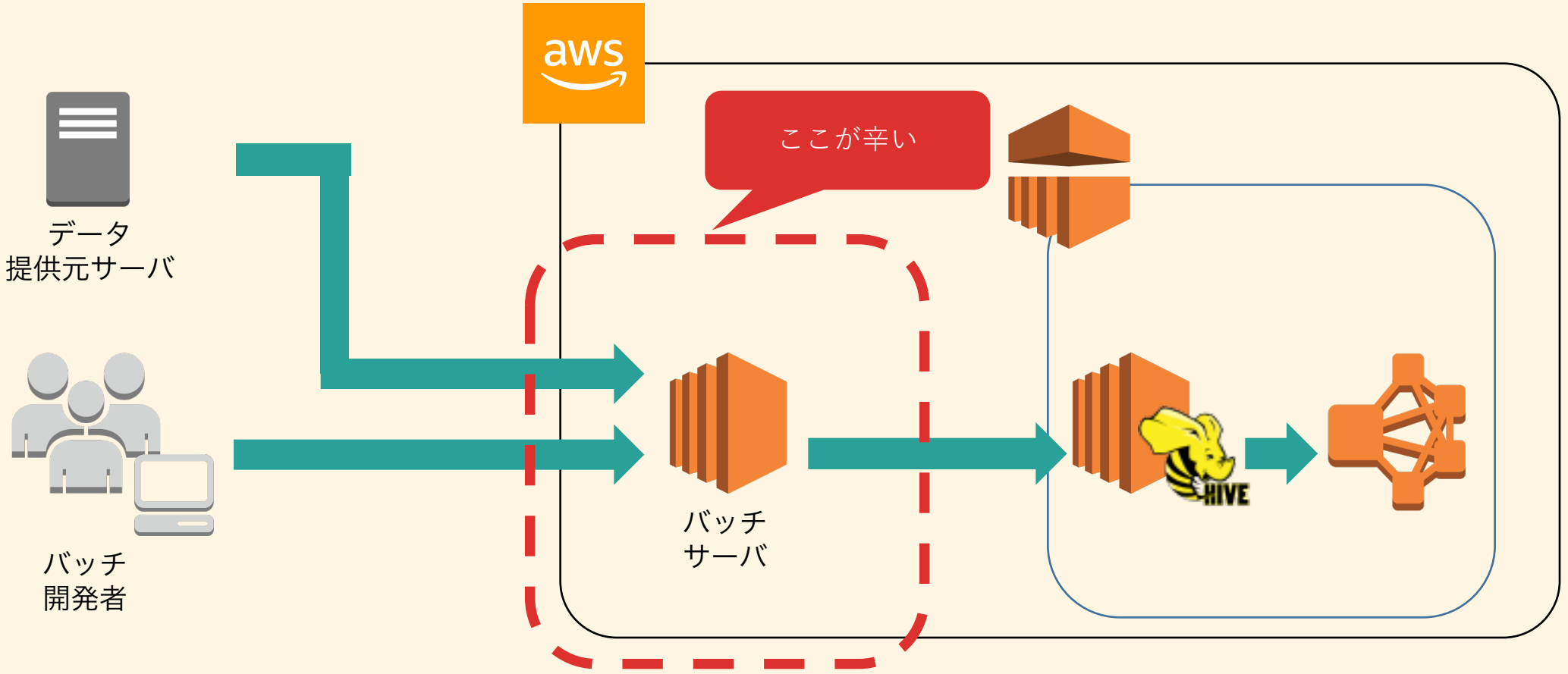
お前は誰だ

- Takumi Kitazawa(@substance626)
- リクルートテクノロジーズでインフラ運用とかマイグレとか
- 最近はスクラムとかSREとかも
- ワークフローエンジンは横目で見ている程度

プロジェクト環境


aws
FTP
データ
提供元サーバ
バッチ開発・制御
・実行
バッチ
開発者
SSH
バッチ
サーバ
SSH＋
Hiveクエリ実行
HIVE

プロジェクト環境



プロジェクト状況

- 古の時に創られ、継ぎ足され続けたcronジョブ(100+)
  - 1ジョブの実行時間は数分〜6時間程度
  - 大抵は数十分〜数時間おきに呼び出し
  - FTPで置かれたファイルがあれば起動
- ワークフローの進捗状態はメールで把握

問題意識

- ワークフロー管理がし辛い
  - 100行以上のcronファイル
  - Excelによるジョブ、ワークフロー管理
  - ~~依存関係はコードを見ればわかる~~
- ワークフローの実行状況が把握し辛い
  - ~~メールを見ればわかる~~
- 本格的に改修する余裕はない

問題意識

- ワークフロー管理がし辛い
  - 100行以上のcronファイル
  - Excelによるジョブ、ワークフロー管理
  - ~~依存関係はコードを見ればわかる~~
- ワークフローの実行状況が把握し辛い
  - ~~メールを見ればわかる~~
- 本格的に改修する余裕はない

そもそもここが
問 d (ry

要件

✓ワークフローの管理がし易いこと
- 依存関係の明示化

✓ワークフローの実行状況が把握できること
- 実行状態の可視化

✓学習・運用コストが少ないこと
- cron同様バッチ開発者が登録、運用する

✓今までのshellを使いまわせること
- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

要件

✓ワークフローの管理がし易いこと
- 依存関係の明示化

✓ワークフローの実行状況が把握できること

そうだ、ワークフローエンジン使おう

✓今までのshellを使いまわせること
- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

ワークフローエンジン候補

- AWS StepFunctions
  - AWS提供のワークフロー制御サービス
  - トリガはAPI Gateway、CloudWatch
  - Lambdaや任意のコードを含んだワークフローを構成可能

- JP1
  - フル社内マネージドサービス(運用要らず)
  - 20年以上の豊富な実績
  - 利用したことのあるユーザが周りに多い(気がする)

ワークフローエンジン候補

- AWS StepFunctions
  - AWS提供のワークフロー制御サービス
  - トリガはAPI Gateway、CloudWatch
  - Lambdaや任意のコードを含んだワークフローを構成可能

- JP1
  - フル社内マネージドサービス(運用要らず)
  - 20年以上の豊富な実績
  - 利用したことのあるユーザが周りに多い(気がする)

要件

✓ワークフローの管理がし易いこと
- 依存関係の明示化

✓ワークフローの実行状況が把握できること
- 実行状態の可視化

✓学習・運用コストが少ないこと
- cron同様バッチ開発者が登録、運用する

✓今までのEC2＋shellを使いまわせること
- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

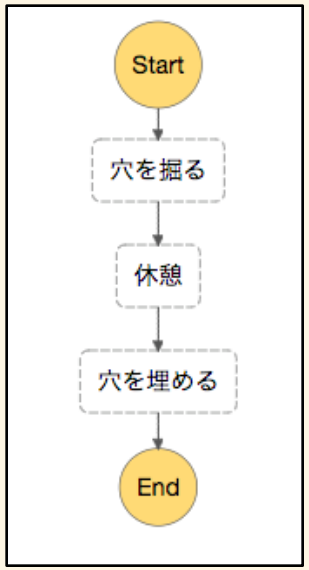
StepFunctionsのワークフロー管理

- AWSコンソール利用の場合（APIでも操作可）

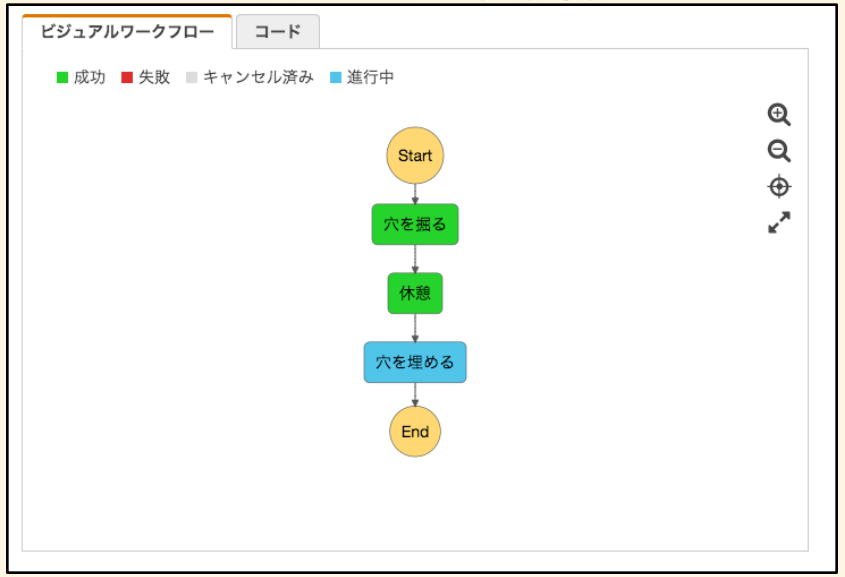
1.Amazon States Languageで定義

```
{
  "Comment": "Send mail",
  "StartAt": "穴を掘る",
  "States": {
    "穴を掘る": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:ap-northeast-1:459006417540:function:DigFunction",
      "Next": "休憩"
    },
    "休憩": {
      "Type": "Wait",
      "Seconds": 10,
      "Next": "穴を埋める"
    },
    "穴を埋める": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:ap-northeast-1:459008822350:function:FillMailFunction",
      "End": true
    }
  }
}
```

2.StepFunctions Consoleで可視化



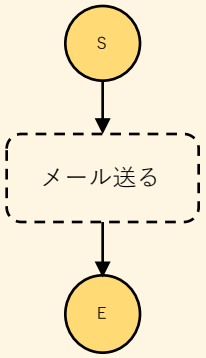
3.実行&進捗・結果確認



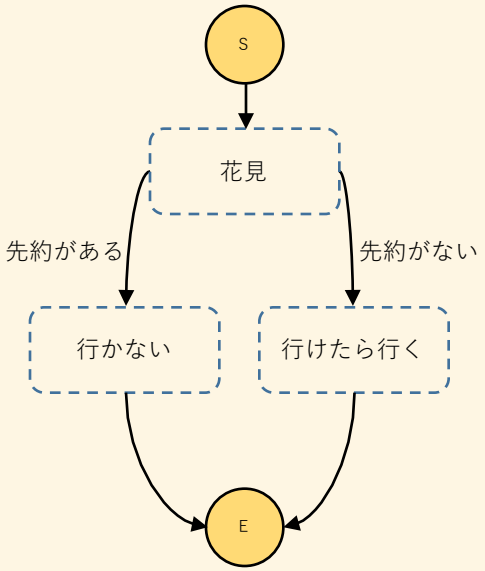
StepFunctionsのワークフロー管理(con't)

基本的な制御はAmazon States Language中のTypeで指定

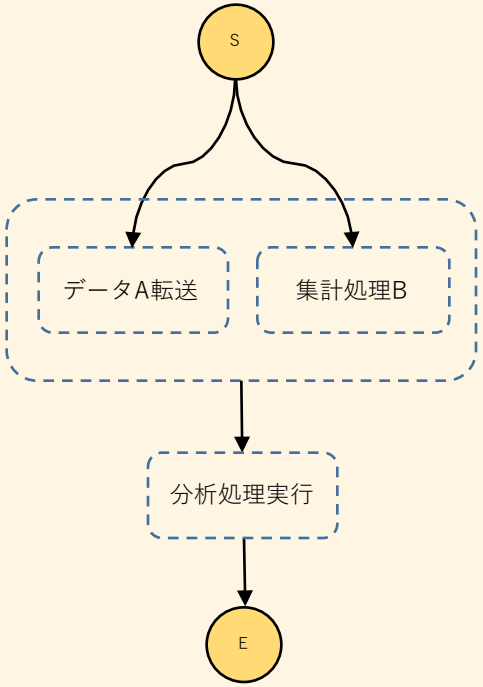
Task : 特定の処理を実行



Choice : 分岐



Parallel : 並列実行

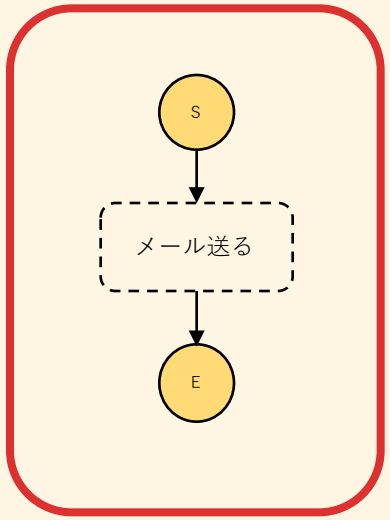


制御やinput/outputを組み合わせることでループ、エラーハンドリングも可能

StepFunctionsのワークフロー管理(con't)

基本的な制御はAmazon States Language中のTypeで指定

Task : 特定の処理を実行




```
{
  "Comment": "Send mail",
  "StartAt": "メール送る",
  "States": {
    "メール送る": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:ap-northeast-
1:459006417540:function:SendMailFunction",
      "End": true
    }
  }
}
```

列実行

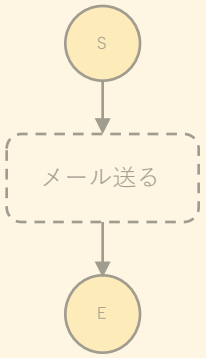
集計処理B

制御やinput/outputを組み合わせることでループ、エラーハンドリングも可能

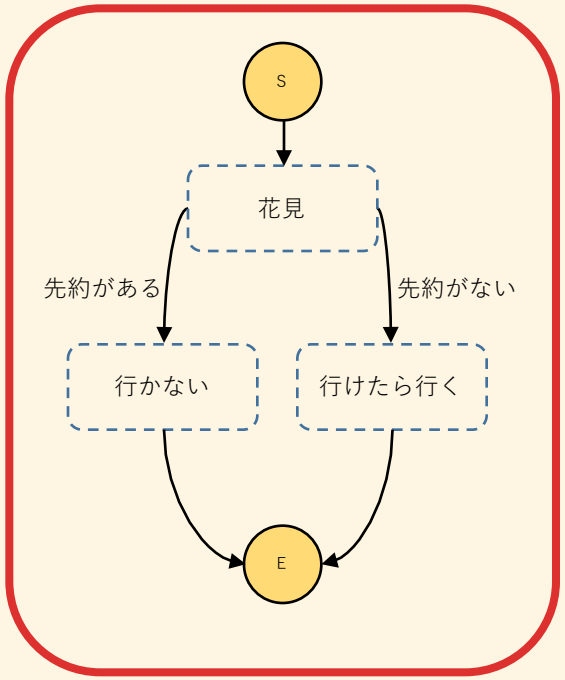
StepFunctionsのワークフロー管理(con't)

基本的な制御はAmazon States Language中のTypeで指定

Task : 特定の処理を実行



Choise : 分岐




```
{
  "StartAt": "花見",
  "States": {
    "花見": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.foo",
          "NumericEquals": 1,
          "Next": "行かない"
        },
        {
          "Variable": "$.foo",
          "NumericEquals": 2,
          "Next": "行けたら行く"
        }
      ]
    },
    "行かない": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:OnFirstMatch",
      "End": true
    },
    "行けたら行く": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:OnSecondMatch",
      "End": true
    }
  }
}
```

制御やinput/outputを組み合わせることでループ、エラーハンドリングも可能

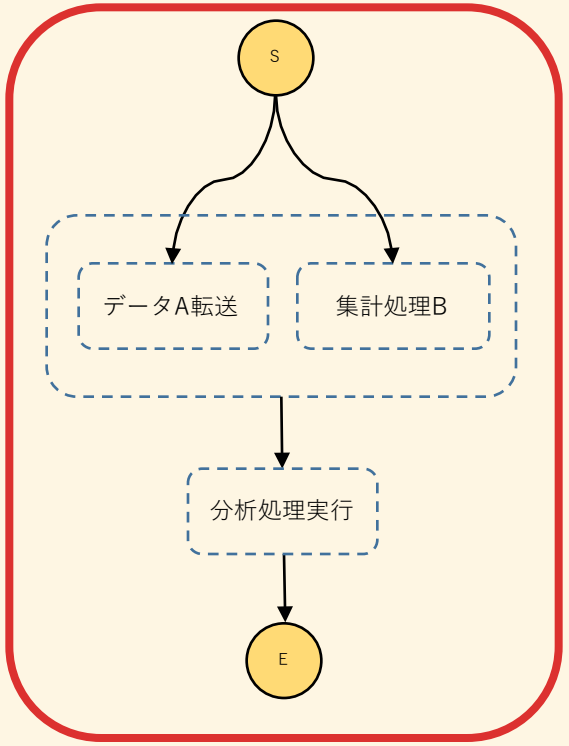
StepFunctionsのワークフロー管理(con't)

基本的な制御はAmazon States Language中のTypeで指定

Task：特定

```
{
  "StartAt": "Parallel",
  "States": {
    "Parallel": {
      "Type": "Parallel",
      "Next": "分析処理実行",
      "Branches": [
        {
          "StartAt": "データAの到着を待つ",
          "States": {
            "データAの到着を待つ": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:ap-northeast-1:459006417540:function:hello",
              "End": true
            }
          }
        },
        {
          "StartAt": "集計処理Bの完了を待つ",
          "States": {
            "集計処理Bの完了を待つ": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:ap-northeast-1:459006417540:function:FailFunction",
              "End": true
            }
          }
        }
      ]
    },
    "分析処理実行": {
      "Type": "Pass",
      "End": true
    }
  }
}
```

Parallel：並列実行



制御やinput/outputを組み合わせることでループ、エラーハンドリングも可能

要件

✓ワークフローの管理がし易いこと　⇒　○
- 依存関係の明示化

✓ワークフローの実行状況が把握できること　⇒　○
- 実行状態の可視化

✓学習・運用コストが少ないこと
- cron同様バッチ開発者が登録、運用する

✓今までのEC2＋shellを使いまわせること
- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

StepFunctionsによるワークフロー定義

- Amazon States languageが少しとっつき辛い?
  - 今のところはGUIからのフロー登録は無い
  - JSONに目を慣らす必要がある
  - 覚えるべき要素はそこまで多くない
  - テンプレ、Best Practiceもある

要件

✓ワークフローの管理がし易いこと　⇒　○
- 依存関係の明示化

✓ワークフローの実行状況が把握できること　⇒　○
- 実行状態の可視化

✓学習・運用コストが少ないこと　⇒　△
- cron同様バッチ開発者が登録、運用する

✓今までのEC2＋shellを使いまわせること
- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

StepFunctionsにおけるshell実行

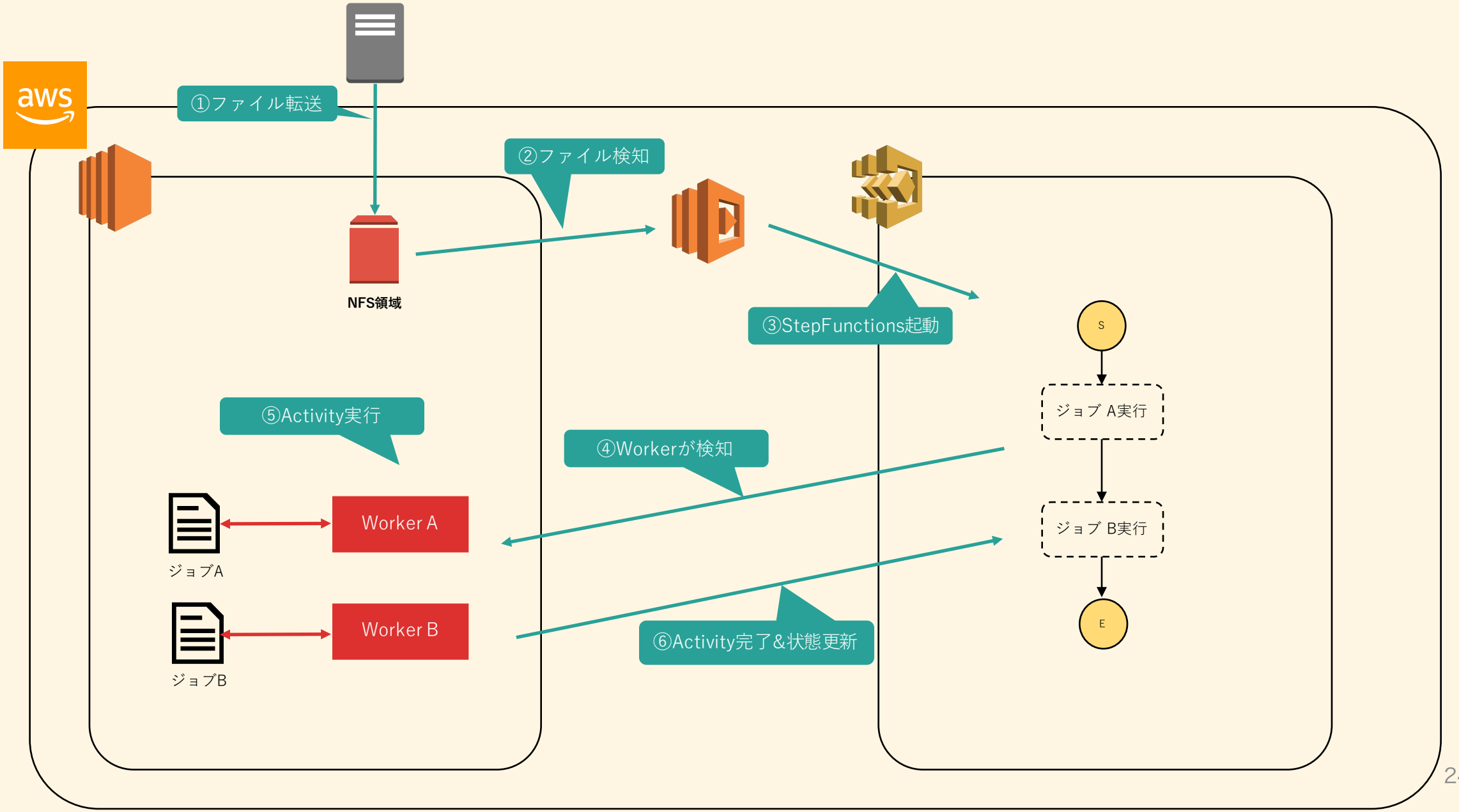
- 既存のEC2＋shellを使う場合Type：Activityを利用
  - EC2やECSなどでWorkerを動かす仕組み
  - Worker⇒ジョブキック(or実行)&StepFunctionsの状態更新する常プロセス
  - ただしWorkerは実装が必要

Writing a Worker

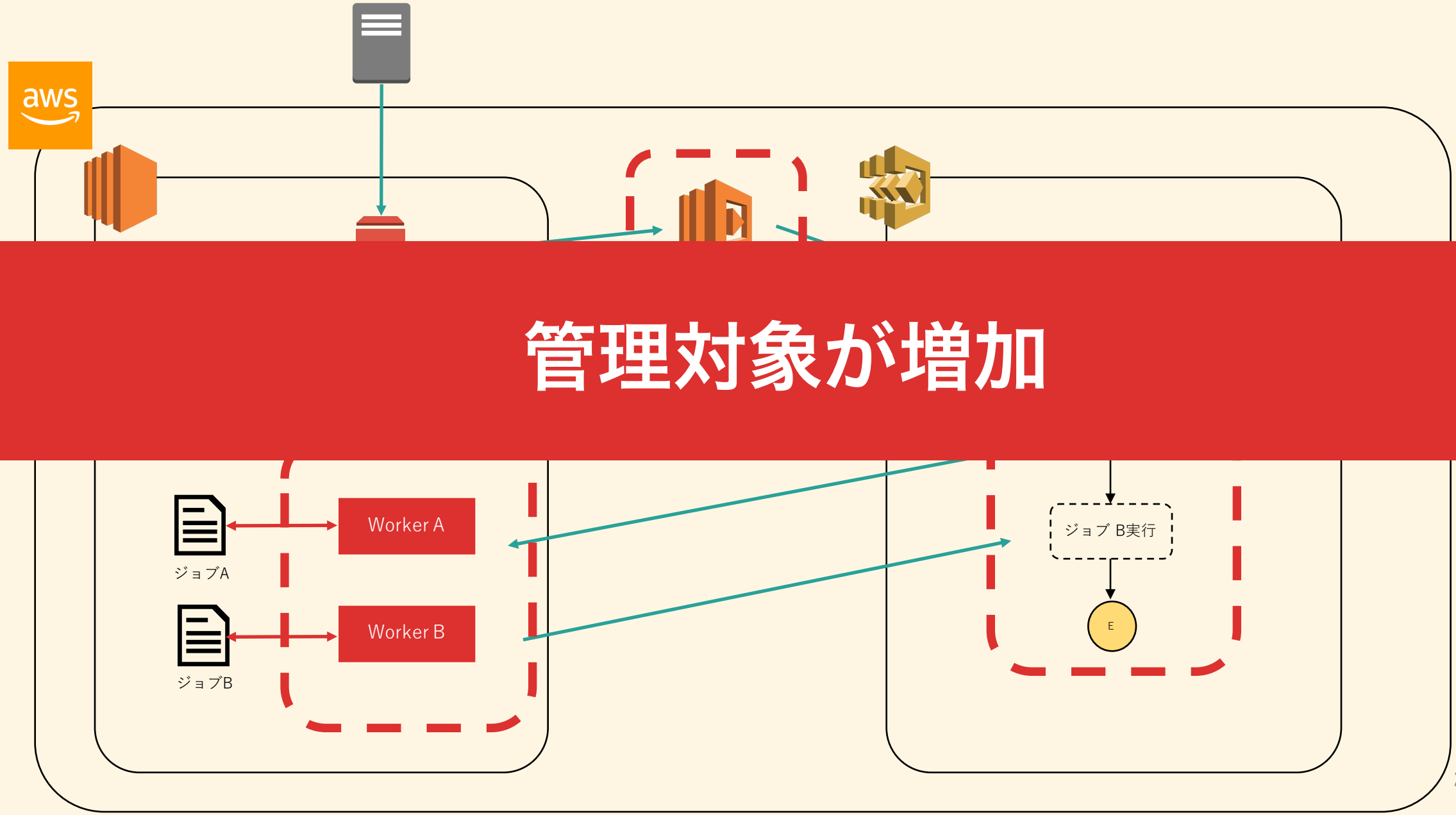
Workers can be implemented in any language that can make AWS Step Functions API actions. Workers should repeatedly poll for work by implementing the following pseudo-code algorithm:

```
[taskToken, jsonInput] = GetActivityTask();
try {
     // Do some work...
     SendTaskSuccess(taskToken, jsonOutput);
 } catch (Exception e) {
     SendTaskFailure(taskToken, reason, errorCode);
}
```

StepFunctionsを使った場合？



StepFunctionsを使った場合？



要件

- ✓ワークフローの管理がし易いこと ⇒ ○
  - 依存関係の明示化
- ✓ワークフローの実行状況が把握できること ⇒ ○
  - 実行状態の可視化
- ✓学習・運用コストが少ないこと ⇒ △
  - cron同様バッチ開発者が登録、運用する
- ✓今までのshell＋EC2を使いまわせること ⇒ △
  - 時刻起動
  - FTPでファイルが置かれたのを検知⇒実行

要件

✓ワークフローの管理がし易いこと • 依存関係の明示化	⇒	○	○
✓ワークフローの実行状況が把握できること • 実行状態の可視化	⇒	○	○
✓学習・運用コストが少ないこと • cron同様バッチ開発者が登録、運用する	⇒	△	○
✓今までのshell＋EC2を使いまわせること • 時刻起動 • FTPでファイルが置かれたのを検知⇒実行	⇒	△	○

JP1

要件

✓ワークフローの管理がし易いこと ⇒ ○ ○

- 依存関係の明示化

✓ワークフローの実行状況が把握できること ⇒ ○ ○

JP1

JP1採用

✓今までのshell＋EC2を使いまわせること ⇒ △ ○

- 時刻起動
- FTPでファイルが置かれたのを検知⇒実行

見送った理由

- StepFunctionsの用途とズレ
  - 機能は自分で好きに実装する必要
- EC2＋shellを改修する余裕が無かった
  - AWS Lambda、AWS Batch化ができれば使いやすそう
  - Workerは…
  - FTP領域⇒S3になればイベント起動可能
- 何だかんだJP1は機能が充実

お粗末さまでした