AWS Data Pipeline

AWS Black Belt Tech Webinar 2015

Yuta Imai
Solutions Architect, Amazon Data Services Japan

アジェンダ

1. What is AWS Data Pipeline
2. When not to use
3. Case studies

アジェンダ

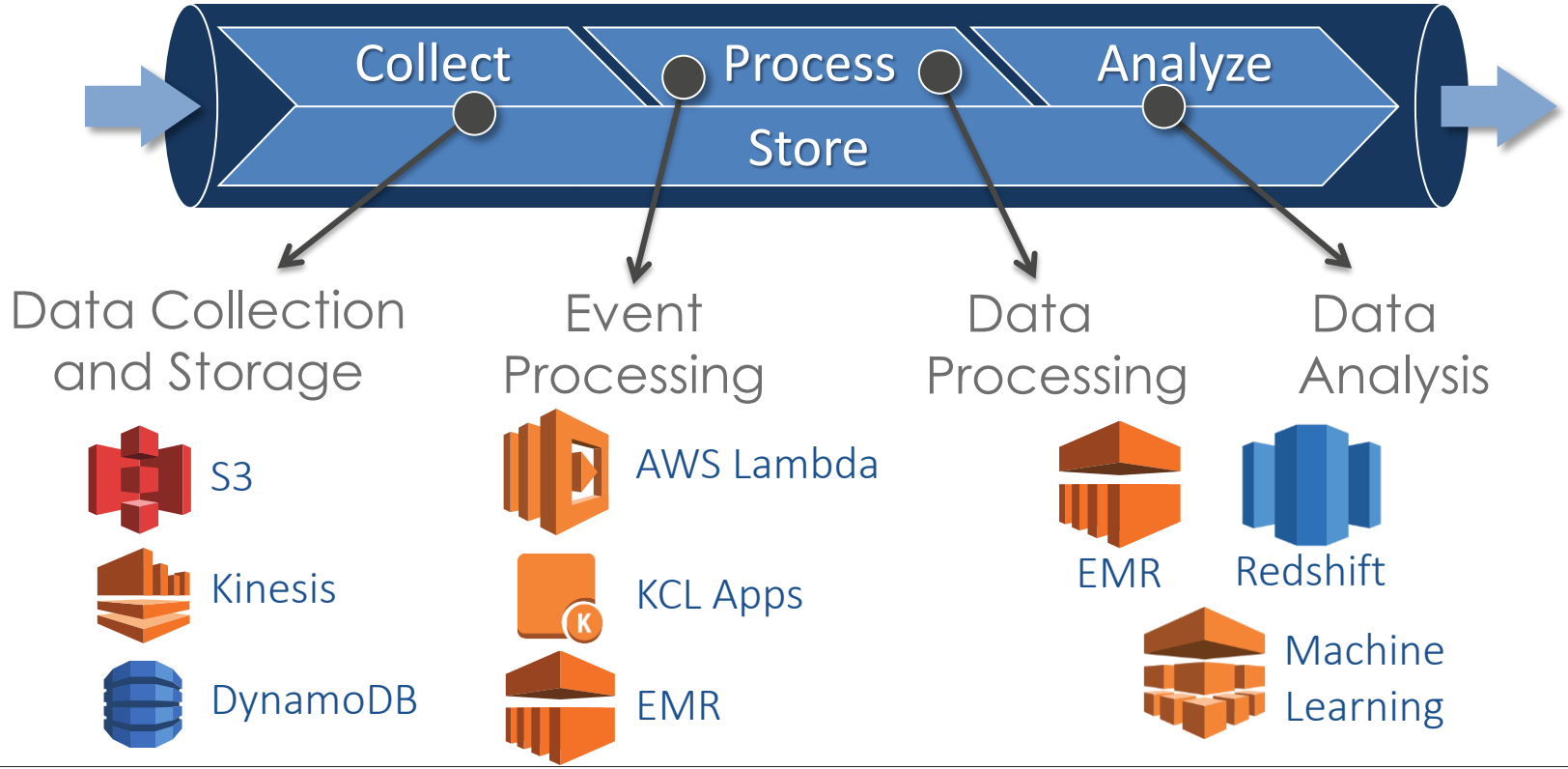
1. What is AWS Data Pipeline
2. When not to use
3. Case studies

AWS Data Pipelineでできること

- さまざまなサーバーやデータベースからのデータの回収
- ETL
- Redshiftへのデータロード

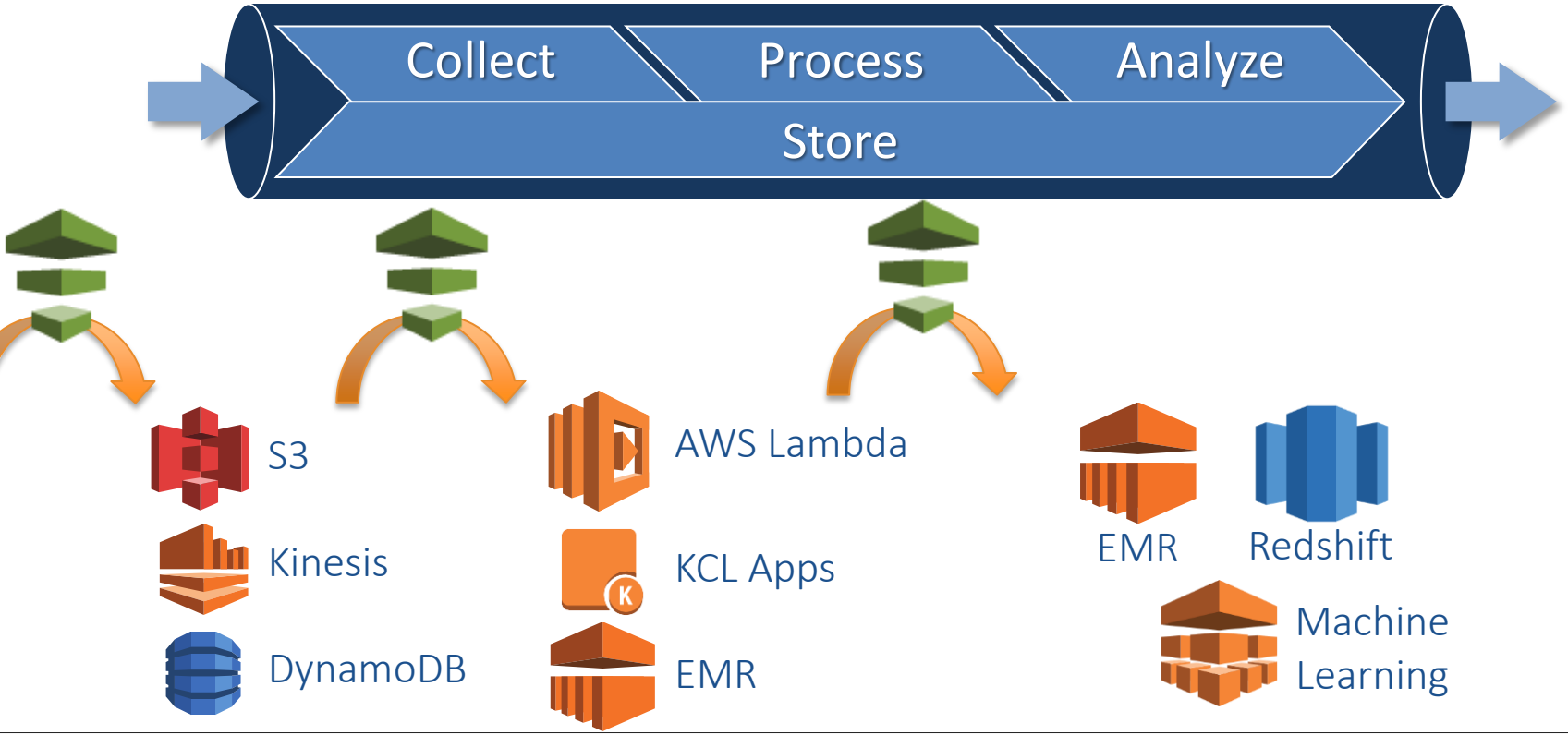
データ利用のためのマエショリ！！！

AWS のビッグデータ・プラットフォーム



amazon web services

AWS のビッグデータ・プラットフォームとData Pipeline



amazon web services

前処理ってめんどくさい

こういうのやめたいですよね・・・

前処理は処理コードを書くだけではなく下記が必要！

- スケジュール管理
- 状態管理

```
s3 = boto.s3.connect_to_region()
key = create_key_from_timestamp() # some_key_2014112500.tar.gz

if(s3.get_key(key)):

  resutl1 = do_first_step(key)

  if(result1):
    do_second_step(result1)
    ...

  else:
    print 'Error in first step...'
    sys.exit()

else:

  print 'error: no target object in s3 bucket. The key to be ...'
  sys.exit()
```

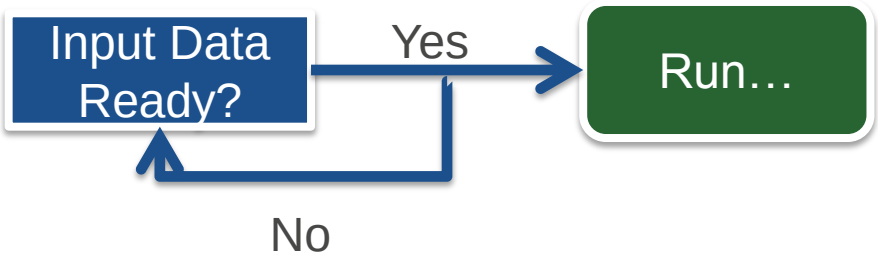
ETLってめんどくさい

データのマイグレーションが簡単じゃない

- 異なる場所、異なるフォーマット
  - S3, RDS, EMR, Redshift, DynamoDB
  - オンプレミス

依存管理が複雑

- 入力データが存在
- 前の処理が成功



異常処理が必要

- 失敗時のリトライ
- タイムアウト
- イベント通知

たとえば・・・
定期的に起動中のEC2のリストを取得してS3に保存

定期的に起動中のEC2のリストを取得してS3に保存

1. スクリプトを書く

2. EC2にそのスクリプトを配置してcronで動かす
   - EC2が落ちたらどうする？
   - cronのログはどこで管理する？
   - EC2の時計がずれたらどうする

定期的に起動中のEC2のリストを取得してS3に保存

1. スクリプトを書く

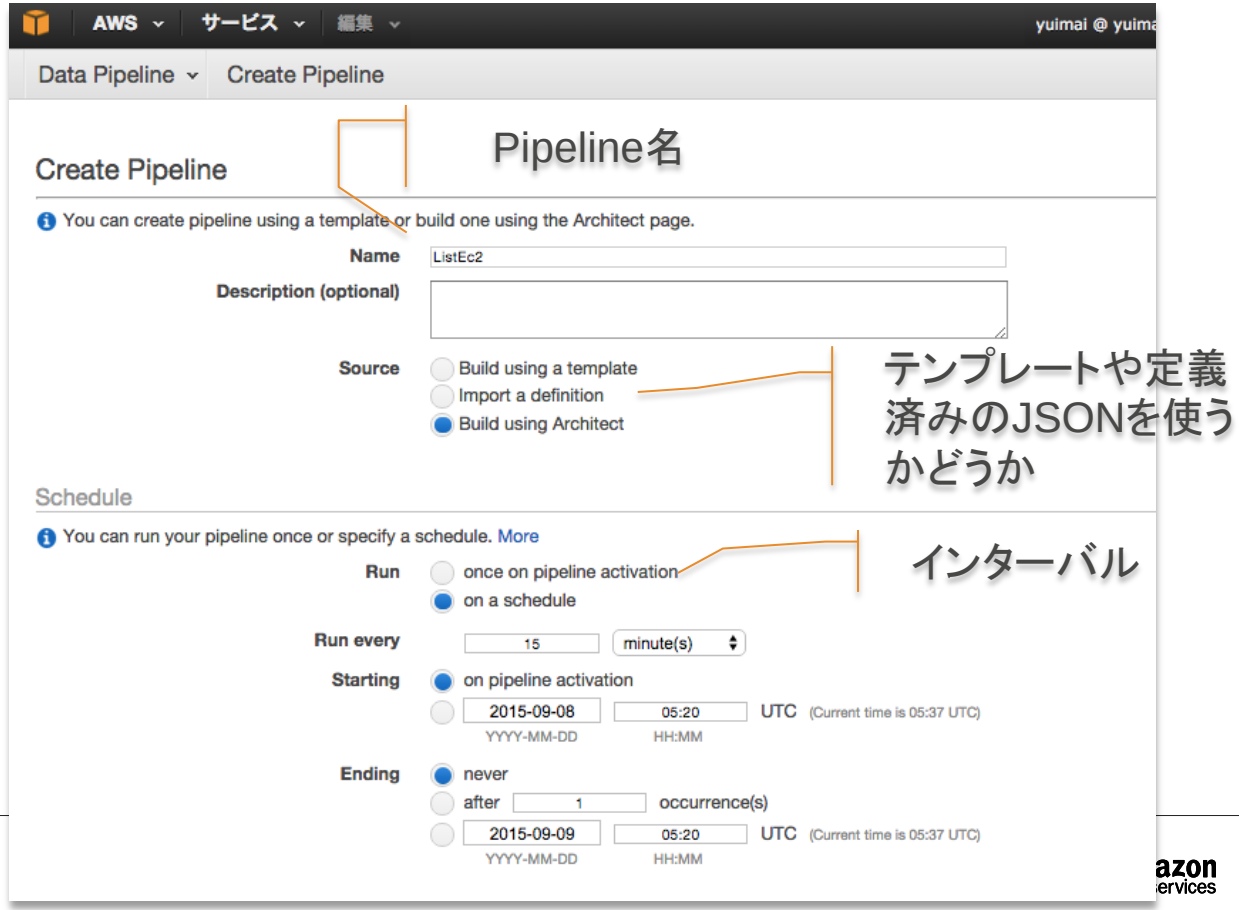
~~2. EC2にそのスクリプトを配置してcronで動かす~~

- ~~EC2が落ちたらどうする？~~
- ~~cronのログはどこで管理する？~~
- ~~EC2の時計がずれたらどうする~~

2. Data Pipelineで動かす！

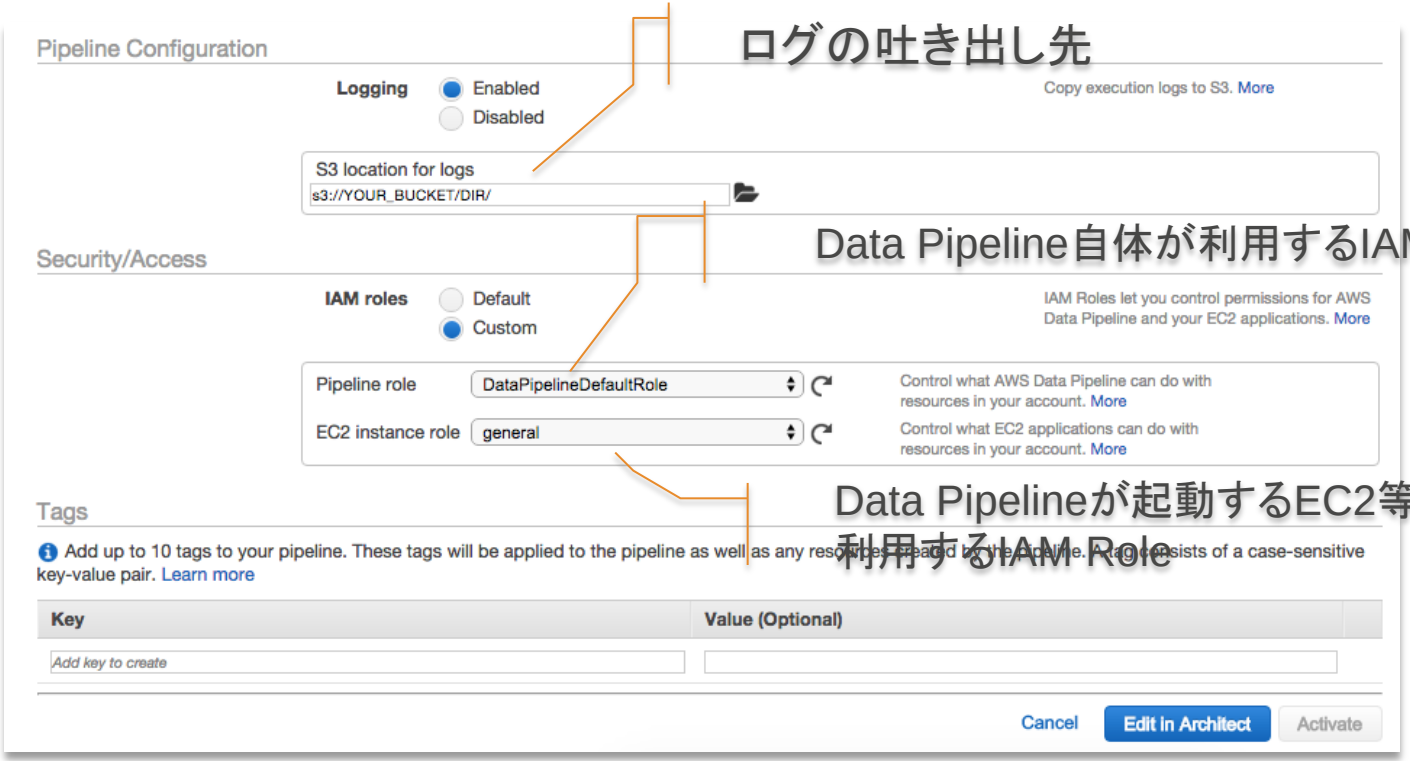
定期的に起動中のEC2のリストを取得してS3に保存

Pipeline作成 - 1/2



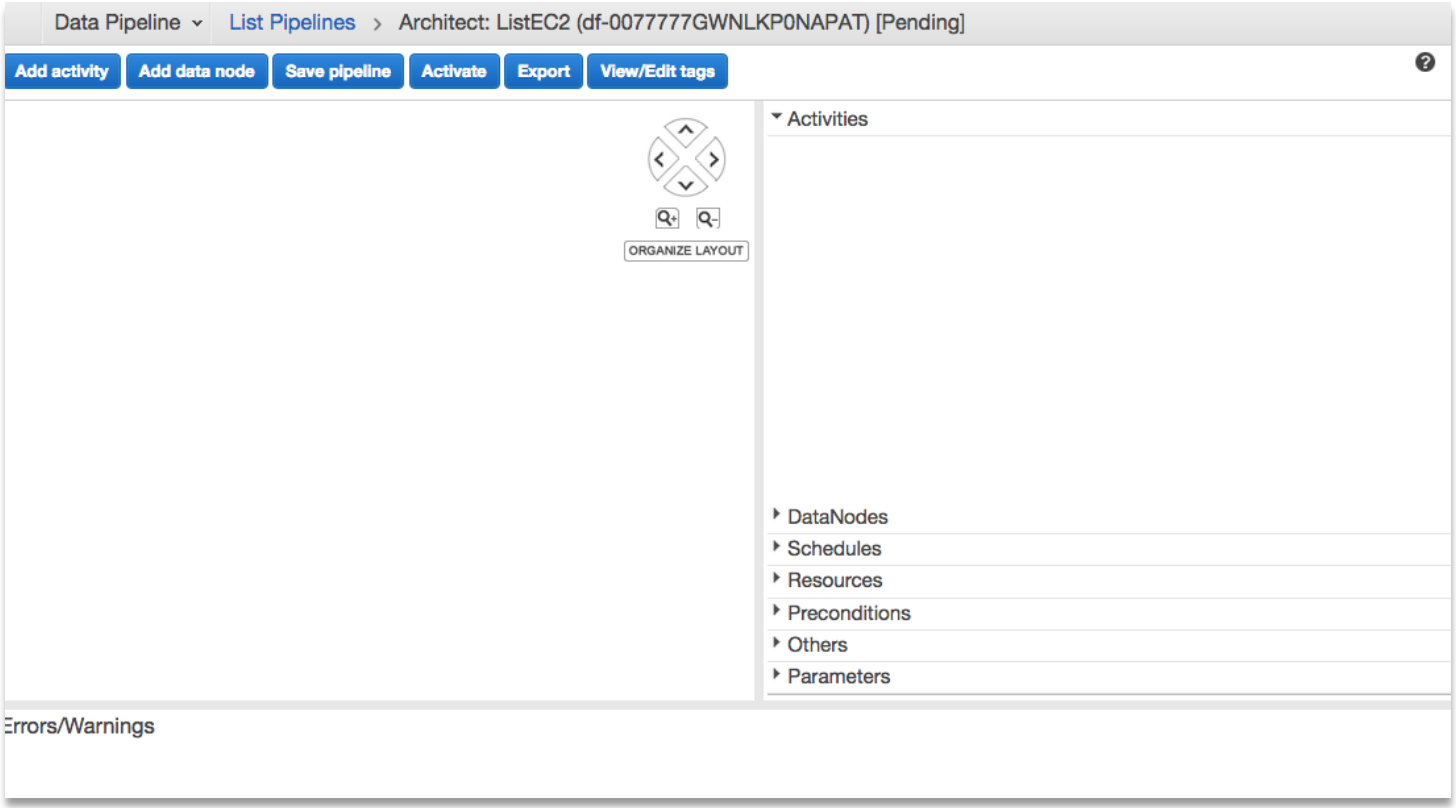
定期的に起動中のEC2のリストを取得してS3に保存

Pipeline作成 - 2/2



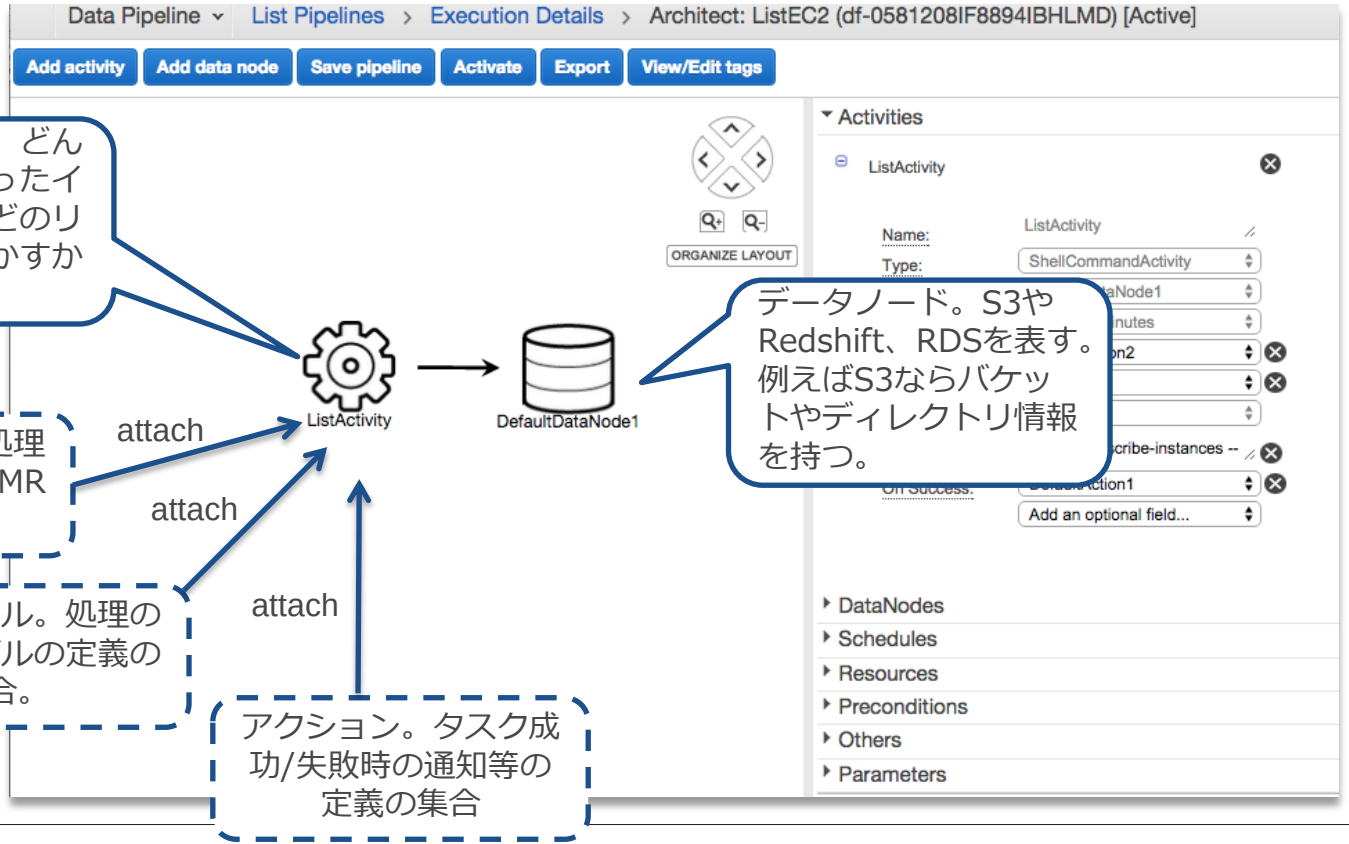
定期的に起動中のEC2のリストを取得してS3に保存

Architect



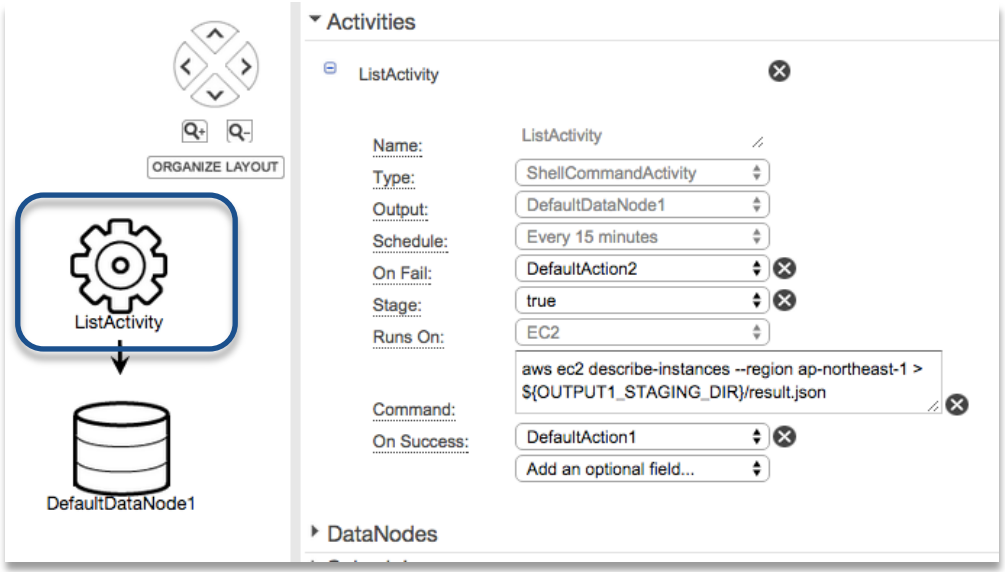
定期的に起動中のEC2のリストを取得してS3に保存

Architect



定期的に起動中のEC2のリストを取得してS3に保存

Architect



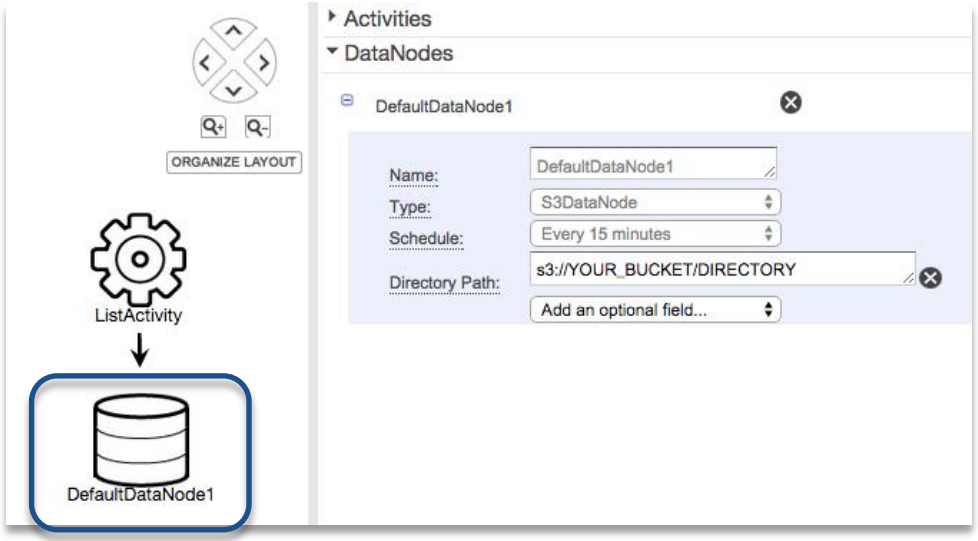
- Output: データの出力先
  - 結果を出力するS3バケット。詳細DataNotdeで説明。
- On Success/On Fail
  - 成功/失敗時のアクション

アクティビティ

- Type: ShellCommandActivity
  - 指定したシェルスクリプトを実行するアクティビティ
- Schedule: 15分毎
- Runs On
  - アクティビティを実行するEC2。詳細はResourceで説明。
- Command
  - 実行するコマンド
  - CommandではなくてScriptUriを使うと、S3に配置したスクリプトを指定可能
- Stage: True
  - ${INPUT1_STAGING_DIR}, ${OUTPUT1_STAGING_DIR}という変数が使えるようになる。S3からのデータ、S3へ送るデータが配置される/配置するディレクトリ。

定期的に起動中のEC2のリストを取得してS3に保存

Architect

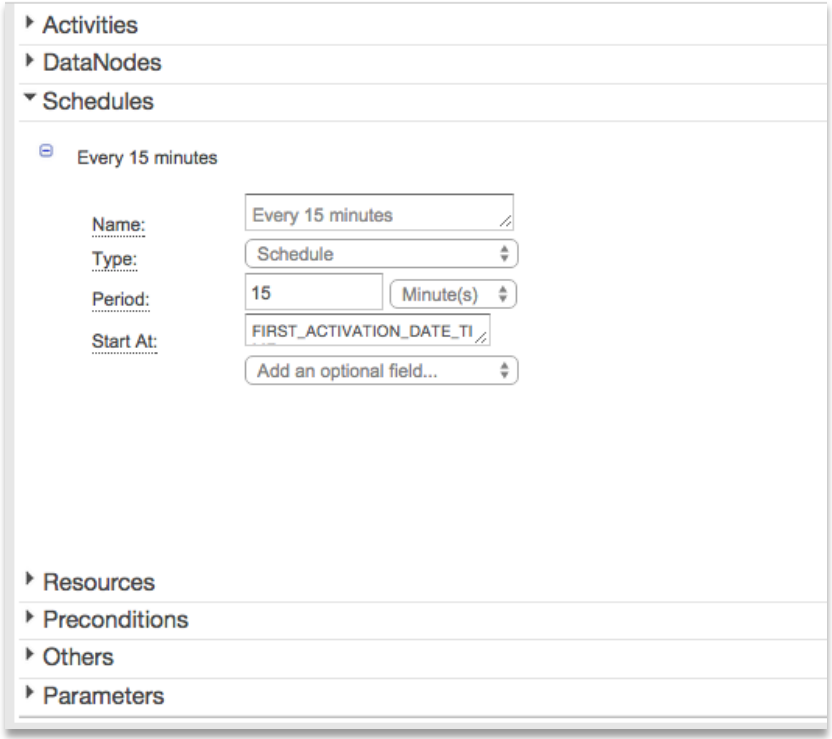


データノード

- Type: S3
  - S3, Redshift, RDSなどの選択肢がある
- Directory Path:
  - S3のパス

定期的に起動中のEC2のリストを取得してS3に保存

Architect

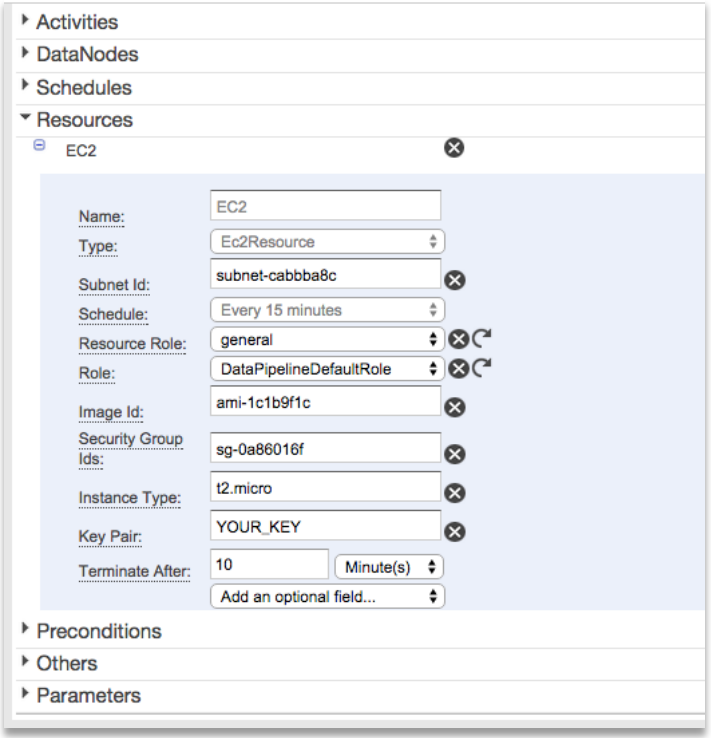


スケジュール

- Type: Schedule
- Period
  - ここで実際のインターバルを設定する。
  - 最小インターバルは15分。
- Start At
  - スケジュールの開始日時

定期的に起動中のEC2のリストを取得してS3に保存

Architect

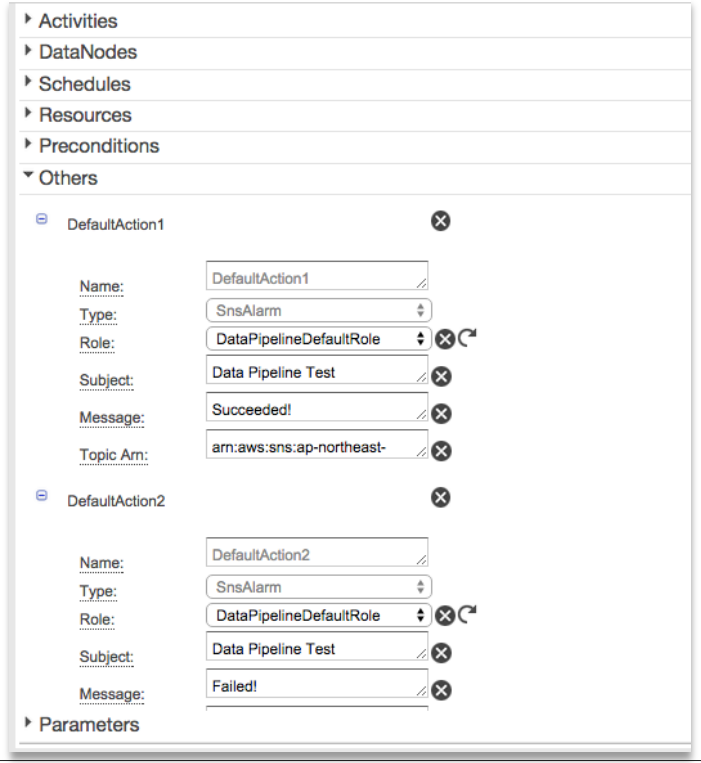


リソース

- Type: Ec2Resource
  - EC2もしくはEMRから選択
- Subnet Id
  - リソースを起動するSubnet
- Schedule
  - リソースを起動するインターバル
- Terminate After
  - 起動したリソースの削除時間の設定
- Image Id/Instance Type
  - 起動するリソースのAMIとインスタンスタイプ

定期的に起動中のEC2のリストを取得してS3に保存

Architect

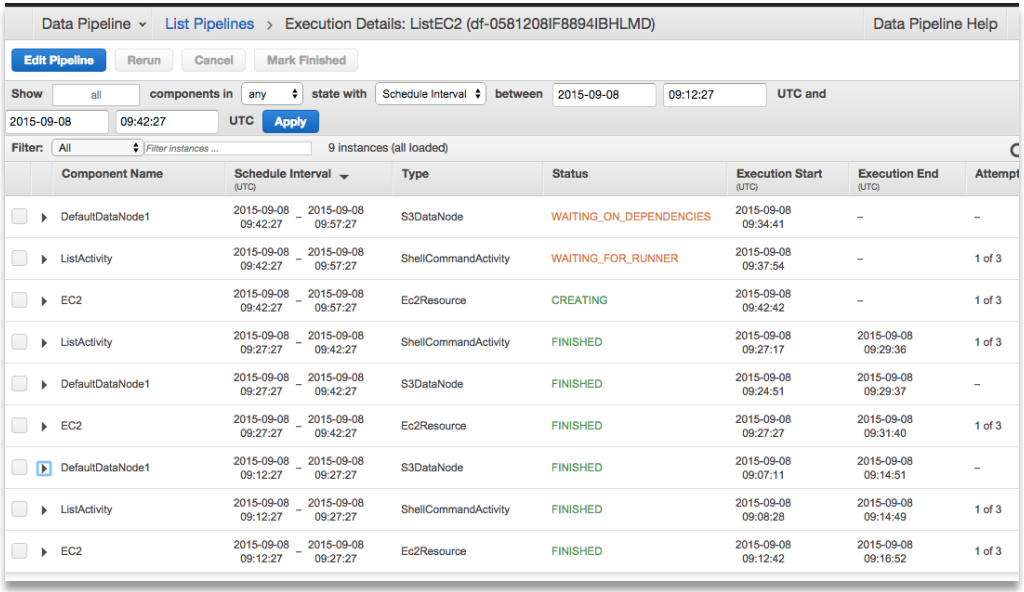


アクション

- Type: SnsAlarm
  - SnsAlarmもしくはTerminateから選択
- Role
  - アクションを実行するIAM Role
- Subject
  - メッセージのタイトル
- Message
  - メッセージの内容
- Topic Arn
  - SNSのトピックのARN(識別子)

定期的に起動中のEC2のリストを取得してS3に保存

Execution Details



- いわゆる実行状況や実行ログのが確認できる
- うまく動かないときはここからリソース起動やアクティビティのログを調査することができる。

定期的に起動中のEC2のリストを取得してS3に保存

Behind the scene

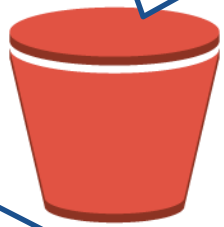
スケジュールで定義した「15分ごと」に・・

リソースで定義にしたがってのEC2を起動。(Terminate Afterで定義した時間がくると削除される)

アクティビティで定義されたコマンドを実行

```
aws ec2 describe-instances --region ap-northeast-1 \
> ${OUTPUT1_STAGING_DIR}/result.json
```

アップロード先はデータノードで定義



${OUTPUT1_STAGING_DIR}に配置されたファイルをS3へアップロード

定期的に起動中のEC2のリストを取得してS3に保存

1. スクリプトを書く

~~2. EC2にそのスクリプトを配置してcronで動かす~~

- ~~EC2が落ちたらどうする？~~
- ~~cronのログはどこで管理する？~~
- ~~EC2の時計がずれたらどうする~~

2. Data Pipelineで動かす！

AWS Data Pipeline: 例えばの使い道・・

Pipelineのテンプレートより→

Getting Started
- Getting Started using ShellCommandActivity

AWS Command Line Interface (CLI) Templates
- Run AWS CLI command

DynamoDB Templates
- DynamoDB cross-regional table copy
- Export DynamoDB table to S3
- Import DynamoDB backup data from S3

Elastic MapReduce (EMR) Templates
- Run job on an Elastic MapReduce cluster

RDS Templates
- Full copy of RDS MySQL table to S3
- Incremental copy of RDS MySQL table to S3
- Load S3 data into RDS MySQL table

Redshift Templates
- Full copy of RDS MySQL table to Redshift
- Incremental copy of RDS MySQL table to Redshift
- Load AWS detailed billing report into Redshift
- Load data from S3 into Redshift

AWS Data Pipeline

AWS Data Pipelineとは

- サービス間のデータ統合・処理をスケジュールベースで自動化してくれるサービス
- 使い始めるにはワークフローを定義する
  1. データ：データソースや出力先の定義
  2. アクティビティ：処理の内容を定義
  3. スケジュールと依存関係：処理の依存関係とスケジュールを定義
  4. 通知：イベントの通知先を定義
- オンプレミス環境との連携も可能

Pipelineの定義

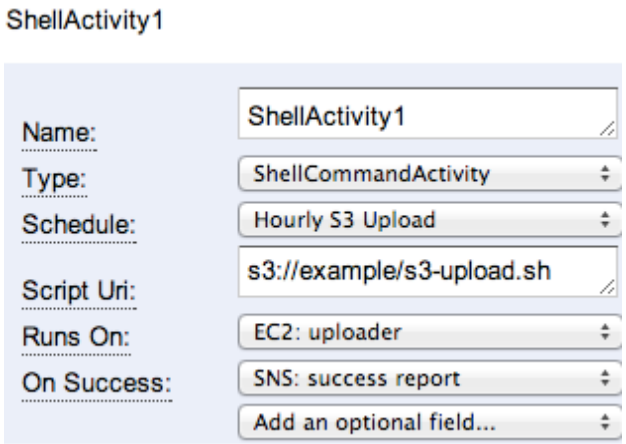
ワークフローを定義するための登場人物

- Data Node: データの場所、フォーマット
- Activity: データ処理のアクティビティ
- Schedule: 処理実行のスケジュール
- Resource: 処理や条件チェックを行うリソース
- Precondition: 処理実行の条件
- Action: 通知を送る方法

アクティビティ(Activities)

データ移動や処理の全体を管理
- 入出力、スケジュール、処理内容、リソース、通知アクション
- AWSとオンプレミスにて実行可能

サポートするアクティビティ一覧
- CopyActivity
- EmrActivity
- HiveActivity
- HiveCopyActivity
- PigActivity
- RedshiftCopyActivity
- SqlActivity
- ShellCommandActivity



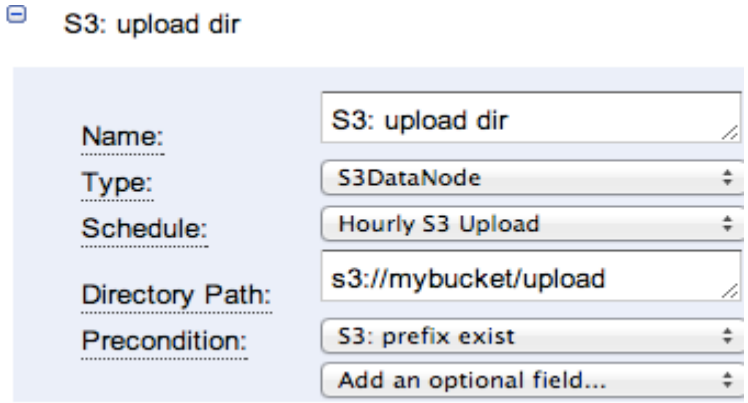
データノード

Input / Outputデータの場所やタイプを定義

- S3パス
- SQL データベース
- DynamoDB
- Redshift

フォーマット指定は自由

- CSV Data Format
- カスタマイズ



データとテーブルのステージング

データにアクセスしやすいための仕組み

- データを自動的にリソースにコピー / テーブルを自動的に作成
  - データがリソースのローカルにあるように
- サポートデータノード: S3DataNode, SqlDataNode
- サポートアクティビティ
  - ShellCommandActivity
    - 初期値: off。Stage = true にセット
    - 変数: ${INPUT**x**_STAGING_DIR} , ${OUTPUT**x**_STAGING_DIR}
  - HiveActivity
    - 初期値: on.
    - 変数: ${input**x**}, ${output**x**}

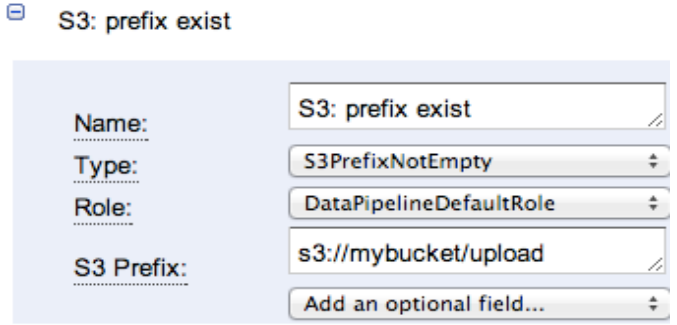
Table作成は不要

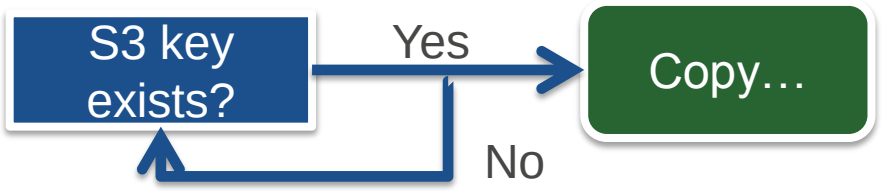
```
{
  "id": "MyHiveActivity",
  …
  "hiveScript": "INSERT OVERWRITE TABLE ${output1} select * from ${input1};"
},
```

依存関係(Preconditions)

条件が成立した場合のみ後継タスクを実行

- DynamoDB tableが存在／データがある
- S3キーが存在
- S3プリフィックスが存在
- 独自のShellコマンド実行が成功
- 依存するpipelineタスクが成功

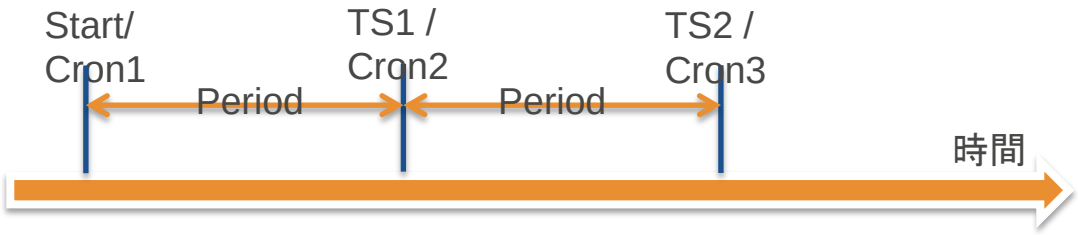


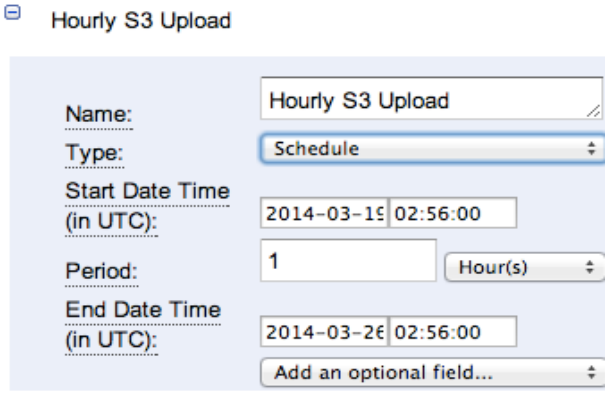


スケジュール

処理を実行するタイミング

- Cronスタイル: 指定した間隔のstart時点で起動
- Time Seriesスタイル: 指定した間隔のend時点で起動
  - ただし、EC2やEMRリソースは常にstart時点に作成
- 間隔: 15分、時、日、週など
  - 15min ~ 3year





スケジュール(2)

Backfillタスク

- 開始時間に過去を指定した場合、現在まで遡ってタスクを繰り返し実行
  - テストに便利
- 開始時間が1日以前の場合、タスク起動しない
  - CLIで *--force* 引数を指定して起動可能

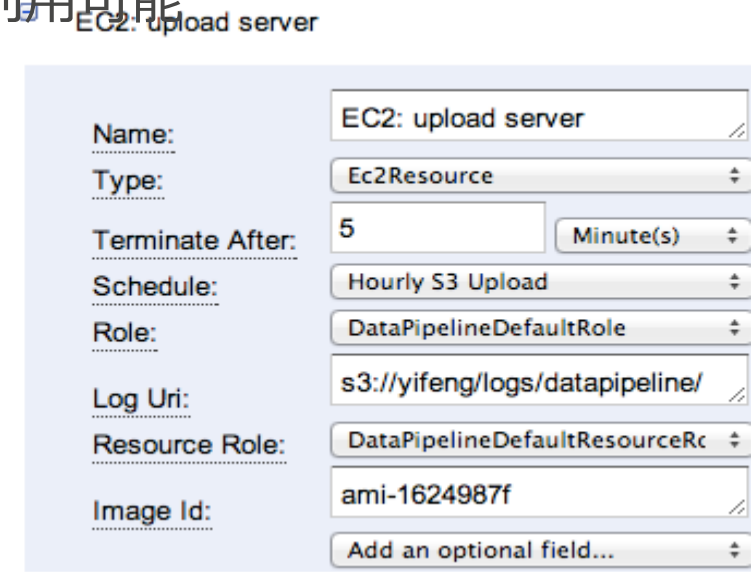
タイムゾーン

- 初期値: UTC, “YYYY-MM-DDTHH:MM:SS” フォーマット
- 変数にタイムゾーン指定可能
  - #{inTimeZone(myDateTime,’Asia/Tokyo')}

リソース

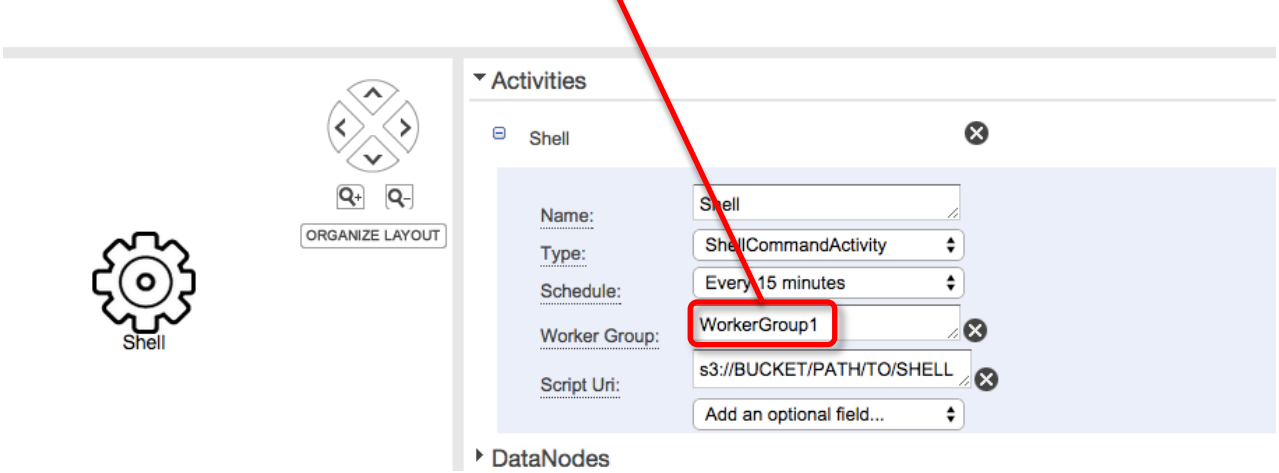
タスクを実行するリソース

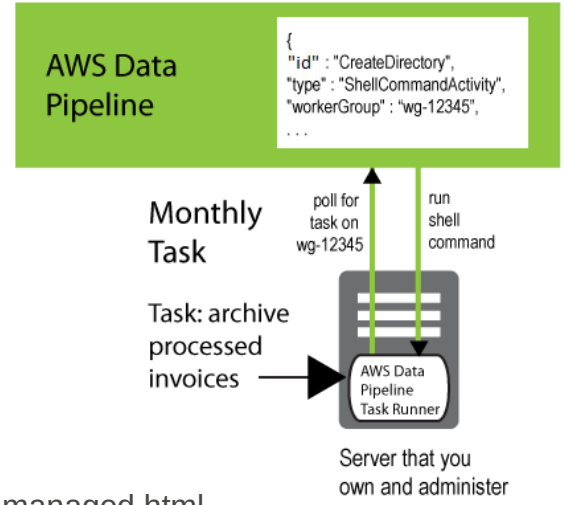
- Task Runnerと呼ばれるAWS Data Pipelineのエージェントプロセスが実体
- EC2: EC2-ClassicとEC2-VPC両方サポート
- EMR: タスクノードにspot instance利用可能
- Multi-regionのリソース管理が可能



Task Runnerは
既存EC2やオンプレミスサーバーでも動く

```
java -jar TaskRunner-1.0.jar --config ~/credentials.json -workerGroup=WorkerGroup1 --
region=MyRegion --logUri=s3://mybucket/foldername
```



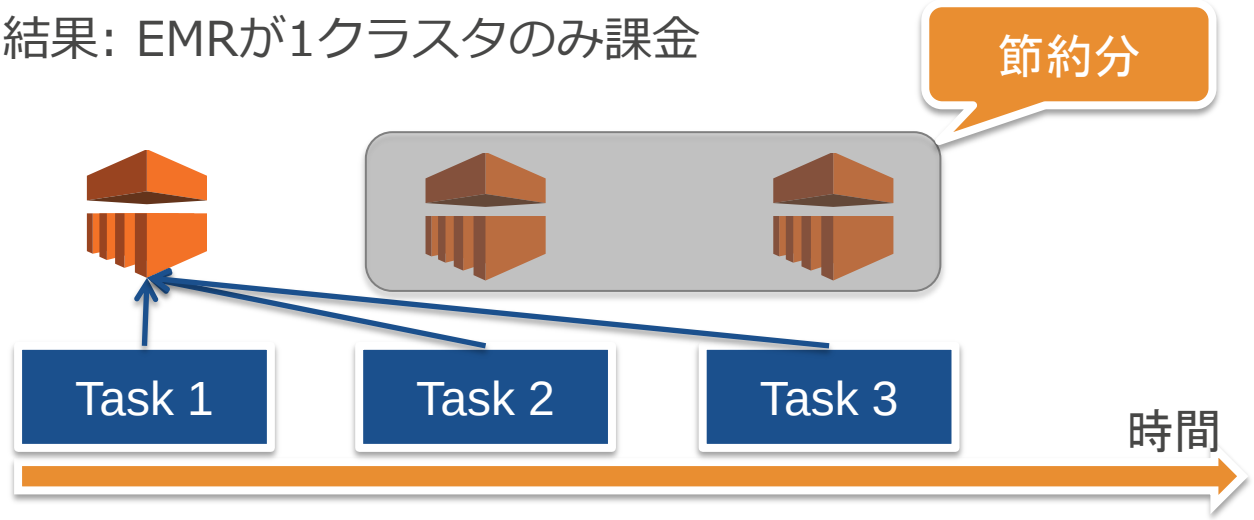


http://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-how-task-runner-user-managed.html

リソース(2)

アクティビティとリソースのスケジュールを別々に指定可能

- リソースを最大限に利用
  - Activityスケジュール: 20分
  - Resourceスケジュール: 1 時間
  - 結果: EMRが1クラスタのみ課金



イベントと異常管理

イベントが発生するタイミング

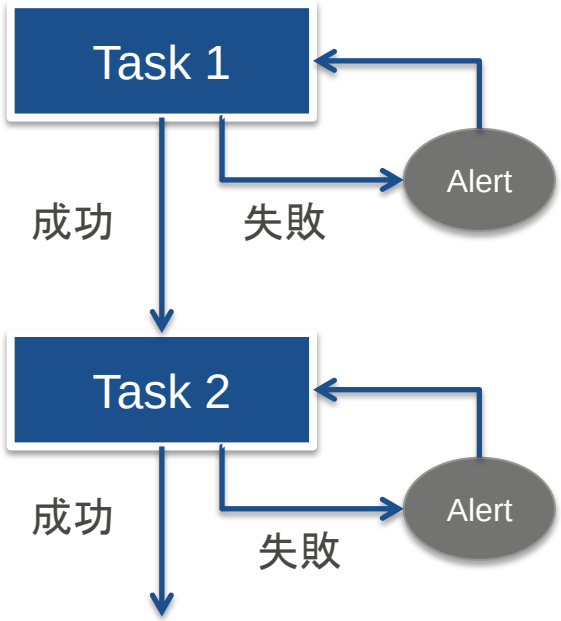
- 成功時
- 失敗時
- 遅れが発生した場合

設定可能なアクション

- SNS通知
- リソースを削除

失敗時自動リトライ

- 初回実行含め、1~6回の実行回数を設定可能
- 初期値は3回



ファイル出力

GUIの操作結果をJSONに出力可能

- Pipelineのバージョン管理
- 出力したファイルは

```
./datapipeline --create pipeline_name --put pipeline_file
--activate --force
```

- インポート前にバリデーションできる

```
./datapipeline –validate my-pipeline.json –credential
credetials.json --force --id df-0123456789ABCD
```

```
{
  "objects": [
    {
      "id": "ActivityId_YYbJV",
      "schedule": {
        "ref": "ScheduleId_X8kbH"
      },
      "scriptUri":
"s3://mybucket/myscript.sh",
      "name": "ShellActivity1",
      "runsOn": {
        "ref": "ResourceId_5nJIh"
      },
      ...
  ]
}
```

料金

- 無料使用枠あり
- アクティビティ、または依存関係の従量課金
- 実際に利用したリソース(EC2, S3など)の課金

	高頻度 (>1日1回)	低頻度
On AWS	$1.00	$0.60
オンプレミス	$2.50	$1.50
実行しないpipeline	$1.00	

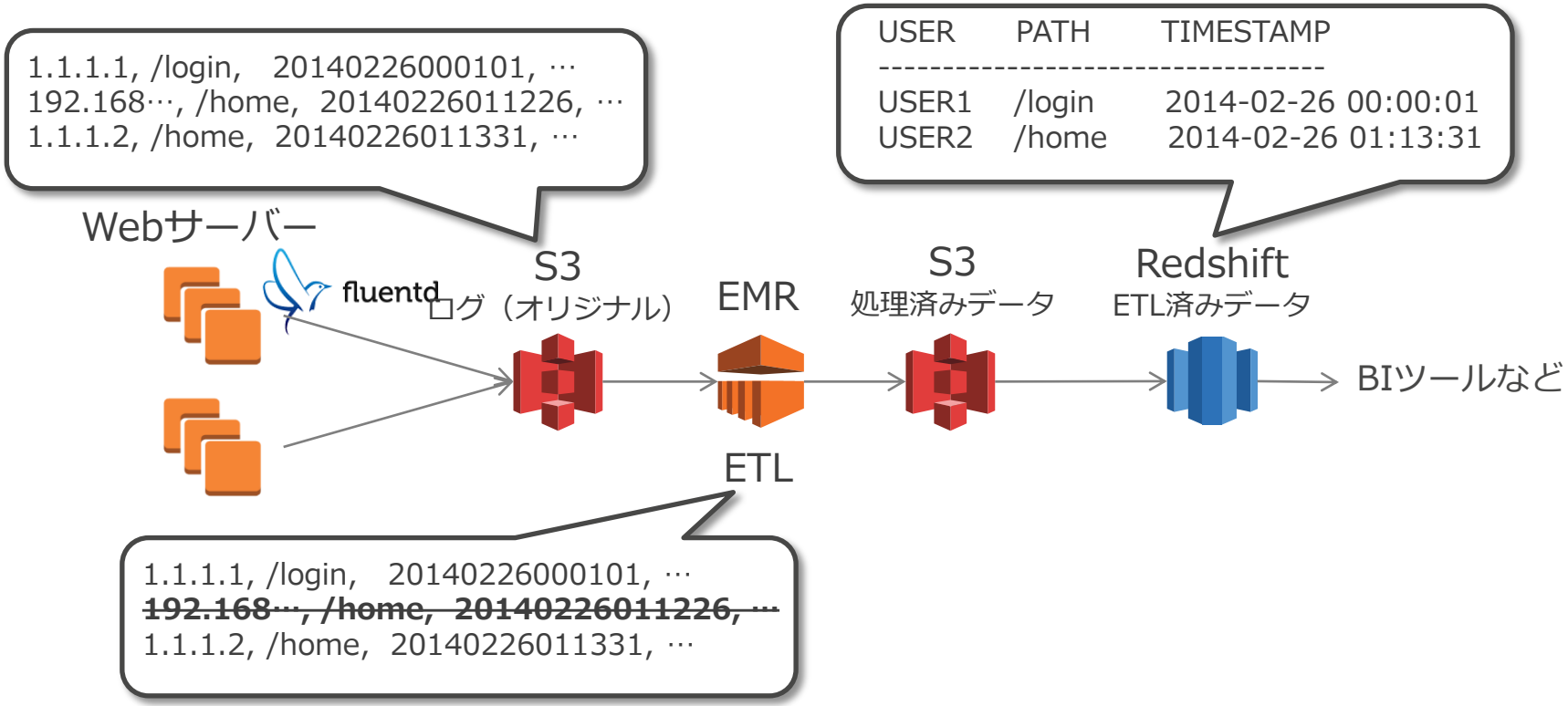
* 2014/3/19時点、東京リージョンのアクティビティ / 依存関係の月額単価

AWS Data Pipelineがもたらしてくれるもの

- 分離
  - データと処理リソースの分離
  - 処理リソースと処理ロジックの分離
  - 処理ロジックとスケジュールの分離
- 統合
  - 分散環境での整合性
  - 一環したエラー処理や処理のやりなおし

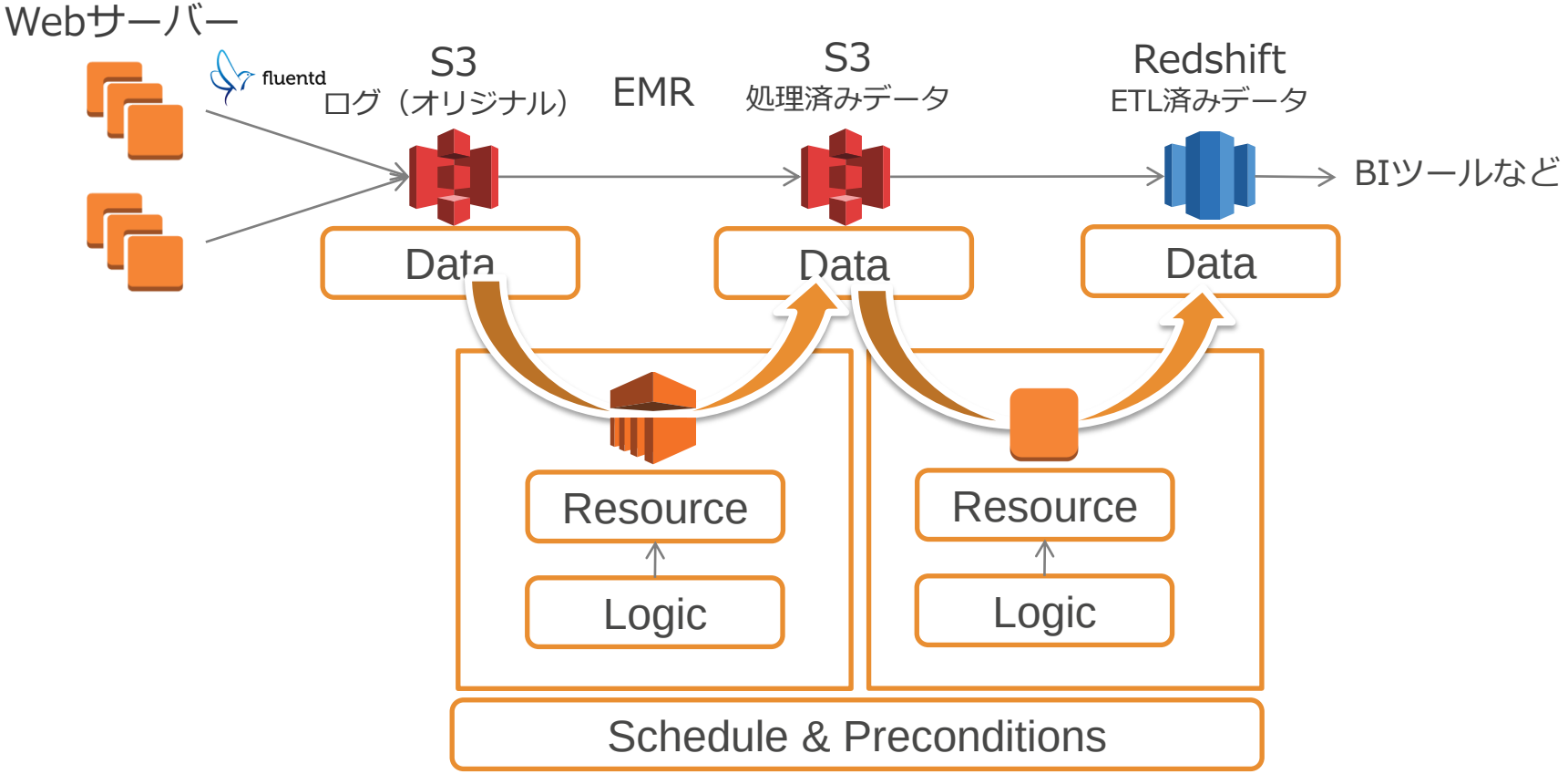
AWS Data Pipelineがもたらしてくれるもの

例えばこんなパイプラインなら・・・



AWS Data Pipelineがもたらしてくれるもの

例えばこんなパイプラインなら・・・



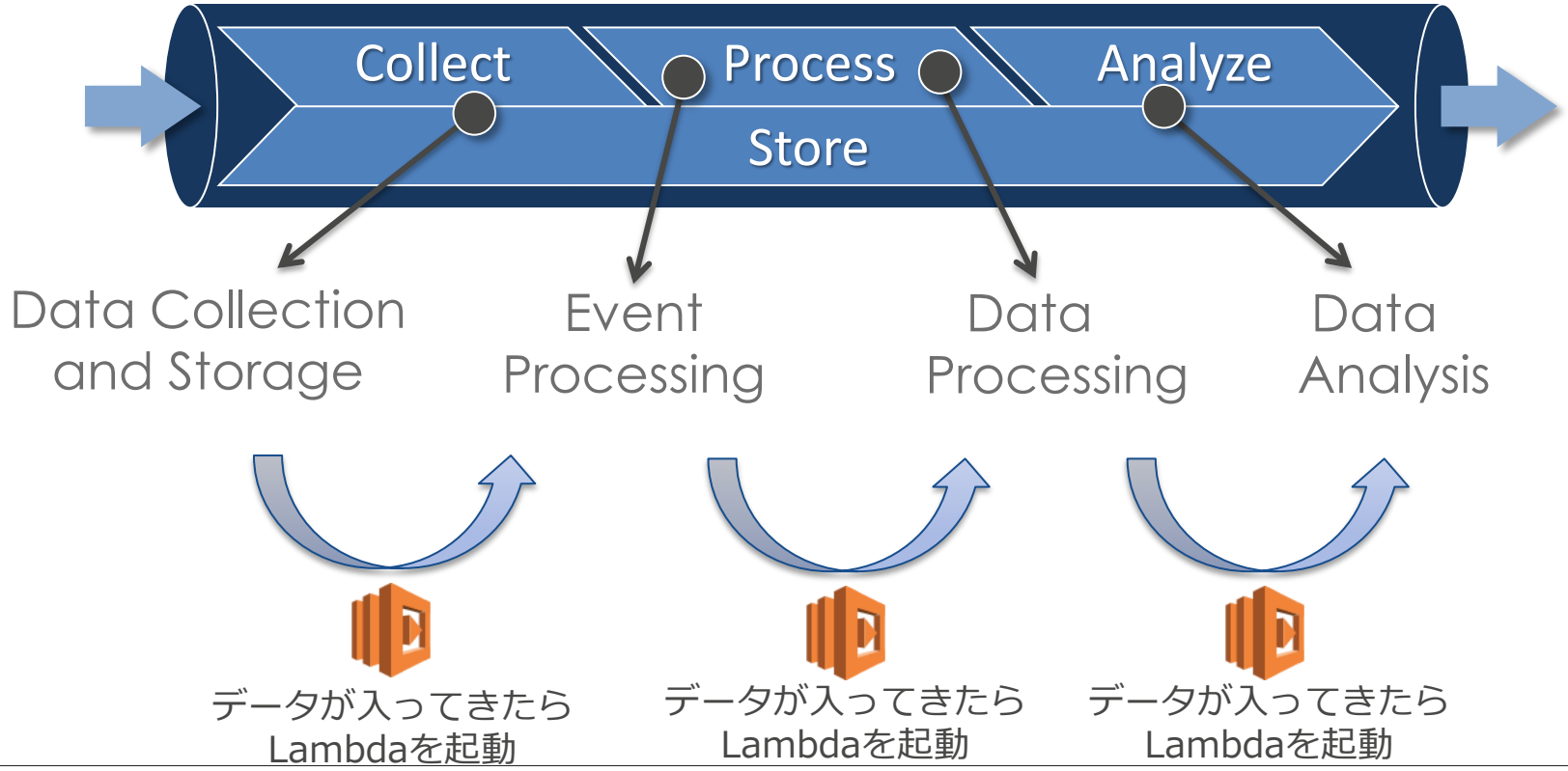
アジェンダ

1. What is AWS Data Pipeline
2. When not to use
3. Case studies

AWS Data Pipelineはスケジュールベースのサービス

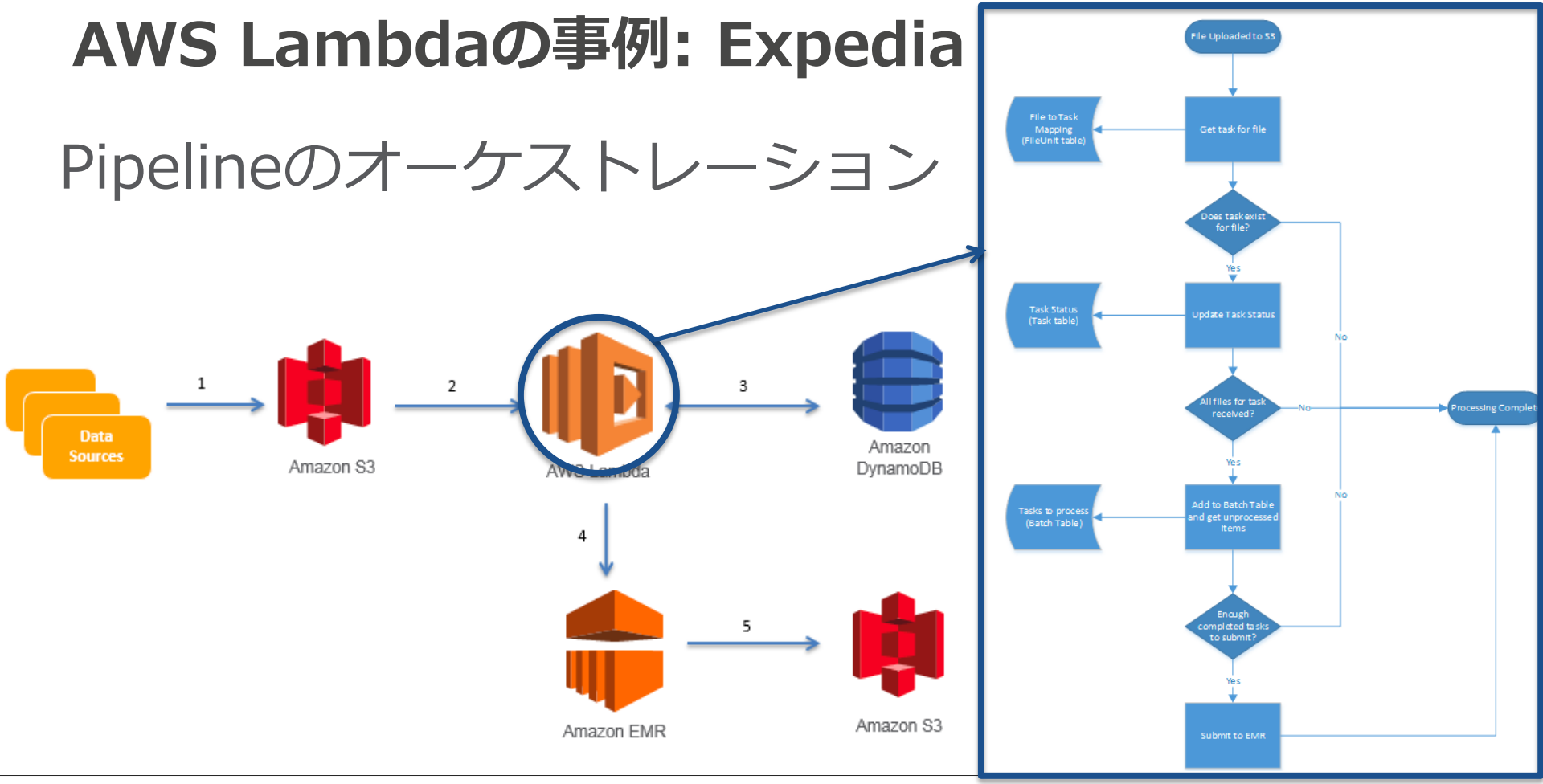
- イベントドリブンな処理を行いたければAWS Lambdaを使う

AWS Lambda as a pipeline glue



AWS Lambdaの事例: Expedia

Pipelineのオーケストレーション



http://blogs.aws.amazon.com/bigdata/post/Tx1R28PXR3NAO1I/How-Expedia-Implemented-Near-Real-time-Analysis-of-Interdependent-Datasets

アジェンダ

1. What is AWS Data Pipeline
2. When not to use
3. Case studies

AWS Data Pipelineがフィットするケース

- 1日に1回などの頻度で行われる処理
  - 複数の拠点やデータセンターからのデータをS3へ回収
    - そしてRedshiftへロード
  - 定期的なクロールの実行とS3への結果アップロード
    - そしてRedshiftへロード
  - データベースのバックアップ
    - RDSのようにマネージドサービスではないデータベースも、Data Pipelineを使えばバックアップを自動化できる

定義済みのアクティビティからユースケースを考えてみる

- CopyActivity
  - MysqlDataNodeとS3DataNode間、もしくはそれら同士のコピーを実施してくれる
  - RDBのバックアップ
- SqlActivity
  - データベースにSQLを発行してくれるアクティビティ。
  - 定期的にローデータテーブルを集計して結果をサマリテーブルに入れるなど。
- EmrActivity
  - EMR上でJARを指定して処理を実行させるためのアクティビティ。
  - 実行するJAR次第でなんでもできてしまうのである意味ShellCommandActivity並に自由度が高い。
  - 素直に使うなら大量のデータに対してのETLや集計、分析など。
- HiveActivity
  - EMR上でHiveを実行させるためのアクティビティ。
  - 生データが格納されているS3バケットからデータを取り出して整形したりフィルタして別バケットに格納、など。
  - S3とDynamoDB間のデータやりとりにも使えるので、DynamoDBのデータインポート/エクスポートにも。

定義済みのアクティビティからユースケースを考えてみる

- HiveCopyActivity
  - S3とDynamoDBのデータ連携に特化したアクティビティ。
- PigActivity
  - EMR上でPigを実行させるためのでアクティビティ。
- RedshiftCopyActivity
  - S3からRedshiftにデータを取り込むためのアクティビティ。
- ShellCommandActivity
  - シェルスクリプトを実行するアクティビティ。なんでもできる！

まとめ

AWS Data Pipelineがもたらしてくれるもの: バッチ処理を管理しやすくしてくれる！

- 分離
  - データと処理リソースの分離
  - 処理リソースと処理ロジックの分離
  - 処理ロジックとスケジュールの分離
- 統合
  - 分散環境での整合性
  - 一環したエラー処理や処理のやりなおし

まとめ: バッチ処理の管理省力化
- もう少し極端に言うとこういうこと -

1. スクリプトを書く

CopyActivity,EmrActivityなど、いろいろプリセットもある！

~~2. EC2にそのスクリプトを配置してcronで動かす~~

~~– EC2が落ちたらどうする？~~

~~– cronのログはどこで管理する？~~

~~– EC2の時計がずれたらどうする~~

2. Data Pipelineで動かす！

AWS Data Pipelineが面倒見てくれる！