# [AWS Black Belt Online Seminar]

# AWS Glue

サービスカットシリーズ

Solutions Architect　倉光 怜
2019/08/06

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

# 自己紹介

## 倉光　怜

所属：ソリューションアーキテクト

経歴：

SIer、クラウドインテグレータを経てAWS入社

前職ではお客様のAWS導入のご支援、設計・構築

好きなサービス：

AWS Glue

Amazon Kinesis

Amazon S3

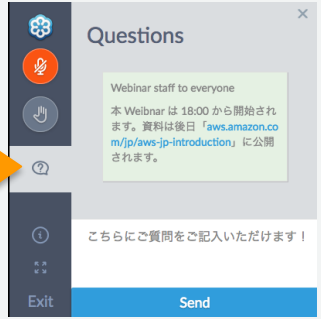
# AWS Black Belt Online Seminar とは

**「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。**

**質問を投げることができます！**

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック




Twitter 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2019年08月06日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。最新の情報はAWS公式ウェブサイト(http://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が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

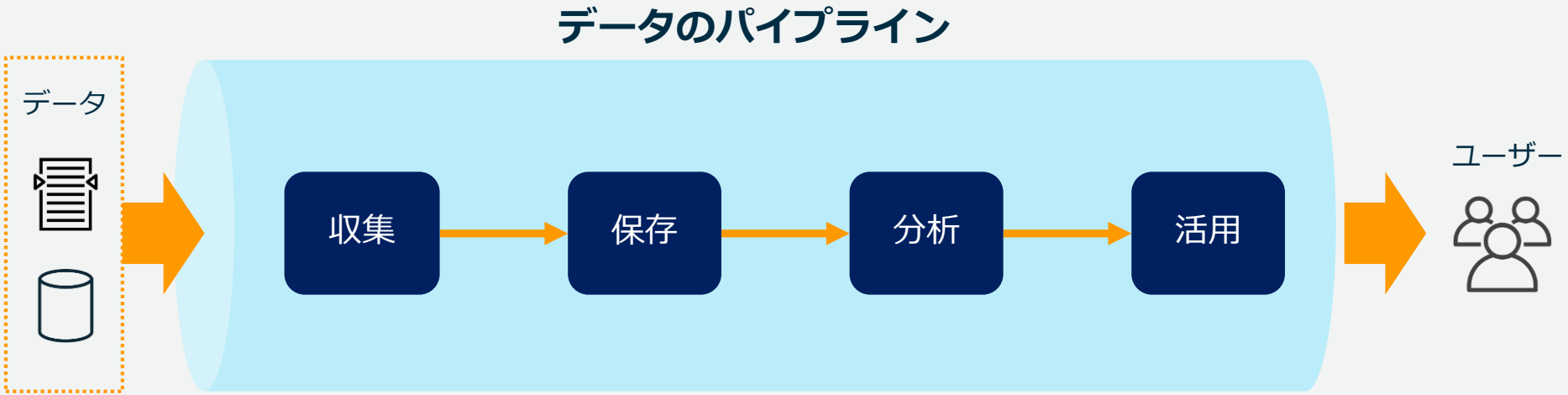
- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

# Agenda

- AWS Glue登場の背景
- AWS Glueの機能
- 開発
- ネットワーク／セキュリティ／監視
- ユースケース
- 料金
- まとめ

# AWS Glue登場の背景

# データ分析のプロセス例



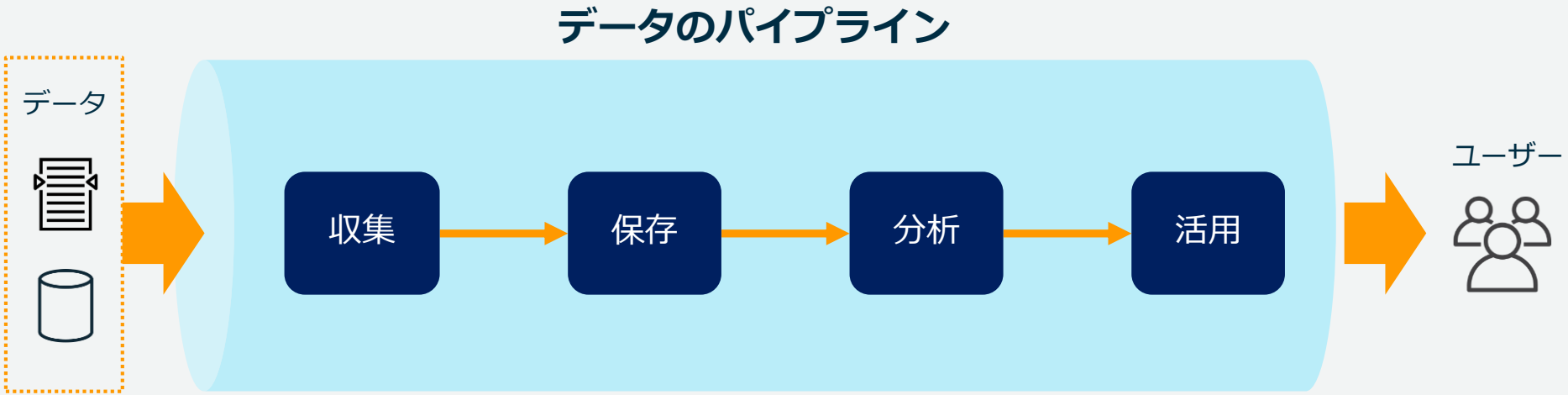

収集・・・データベースやファイルなどからデータを集める

保存・・・分析対象のデータをデータベースやストレージに保存する

分析・・・過去・現在のデータから状況を可視化して、未来を予測する

活用・・・予測結果を社内、または他システムに連携する

# データ分析のプロセス例(Big Dataが注目される前)

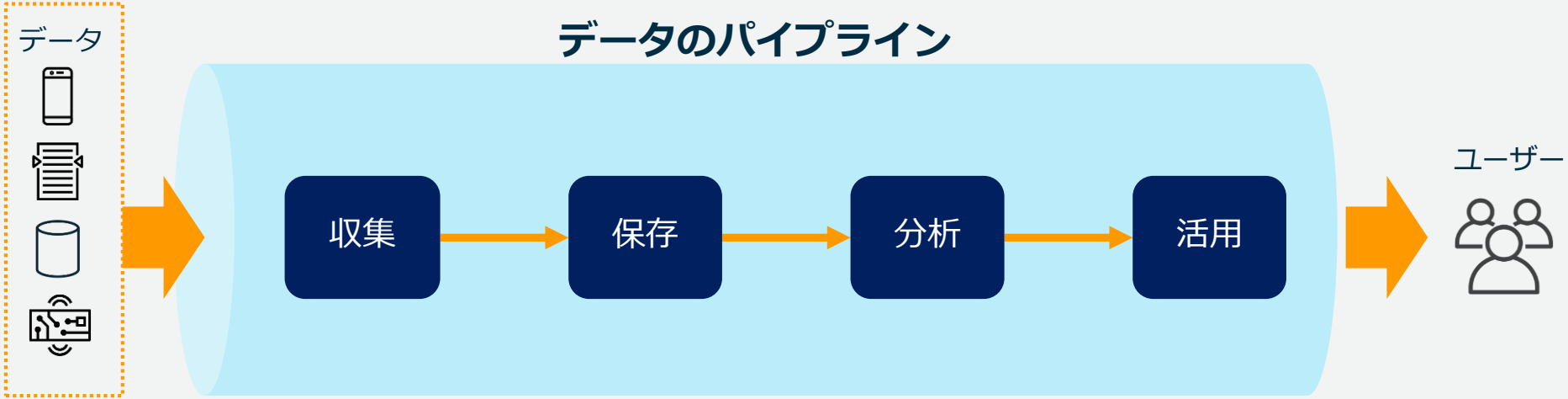



分析対象のほとんどはCSVやRDB上のデータ

分析対象のほとんどは構造データで、それに対してETL処理(※1)を実施していた

(※1)ETL処理：**E**xtract(抽出)、**T**ransform(変換)、**L**oad(ロード)の略

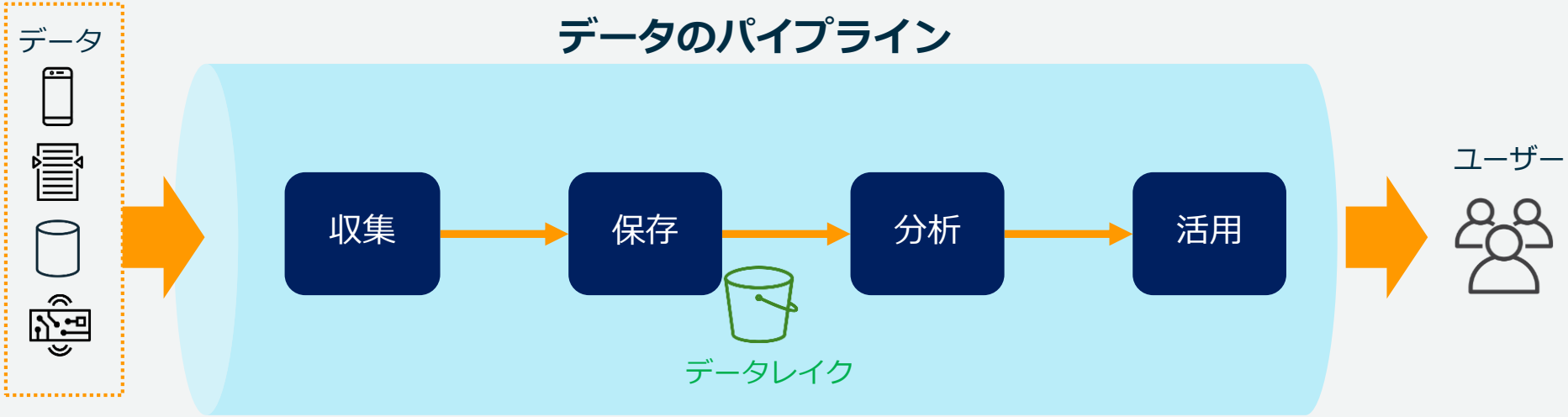
# データ分析のプロセス例(現在)




・従来の保存方法では最低限必要なデータに絞って、加工整形済みのデータを保存することしかできなかった
・データの種類や量の増加、非常に速いスピードでデータが生成されるようになり、大量データを分析してビジネス価値を生み出す動きが活発化、加えて、お客様ビジネス自体の変化も早くなり始めた

大量のデータが保存でき、かつ必要なときに必要分のデータを取得して、活用できる保存場所が求められた

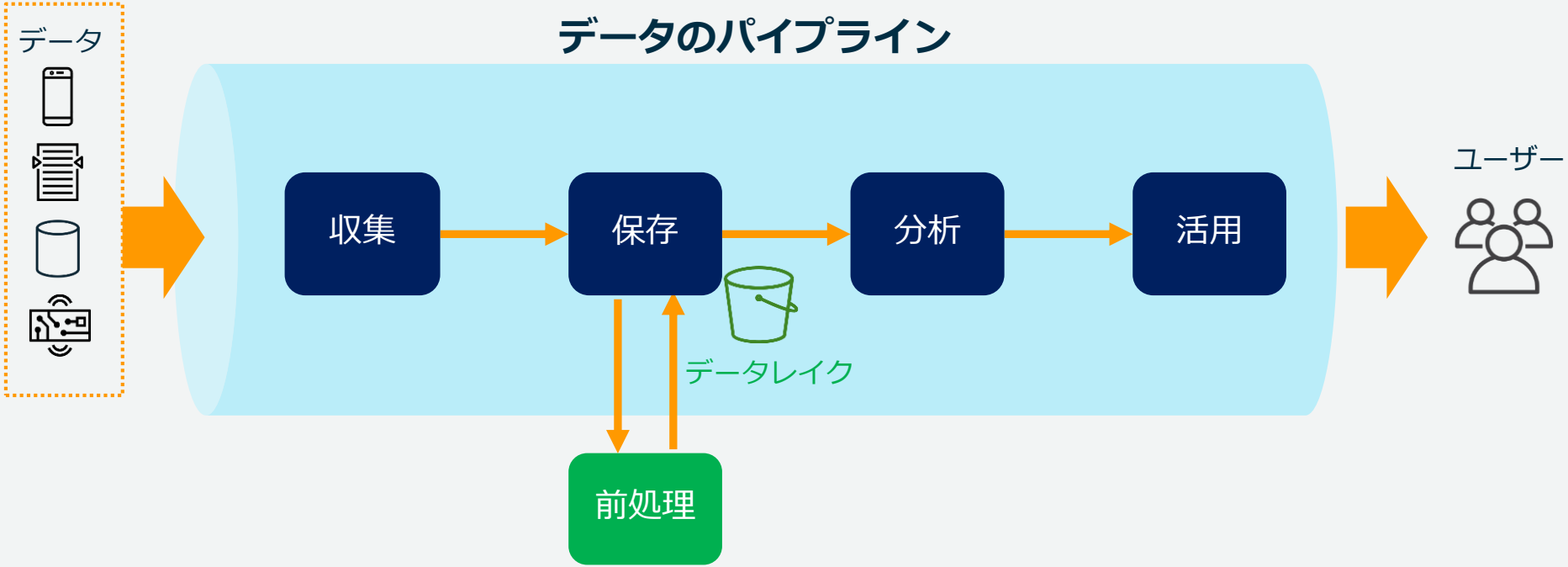
# データ分析のプロセス例(現在)




様々なデータソースから生成される生データをそのまま保存する「データレイク」の概念が登場

生データを分析対象のデータにするためには・・・？

# データ分析のプロセス例(現在)




**データレイク上のデータを分析するために前処理(=ETL処理)を実施する**

# AWS Glue

様々なデータソースの**メタデータを管理する**、
**フルマネージド**で**サーバーレス**な**ETLサービス**

# AWS Glueの特徴

AWS Glue

サーバーレス

柔軟な起動方法

コードに集中

データソースの
メタデータ管理

VPC内からのアクセス

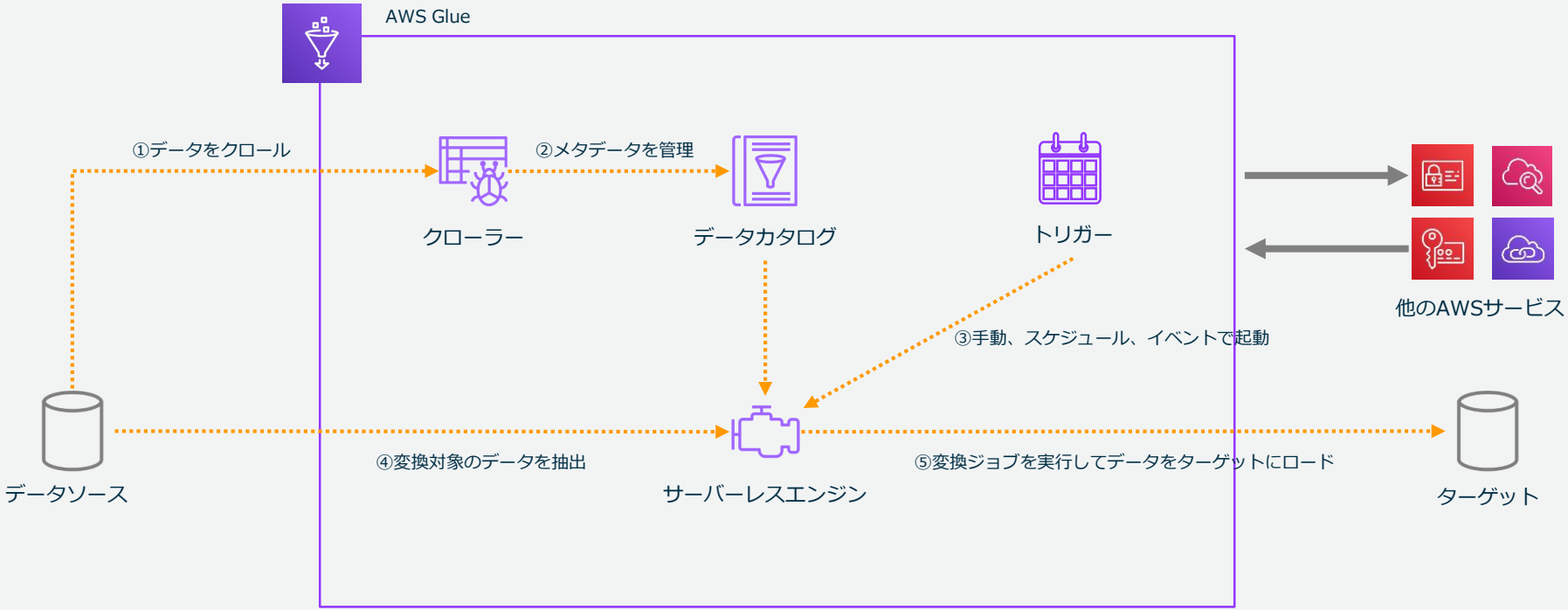
他のAWSサービスと
容易に連携

セキュア

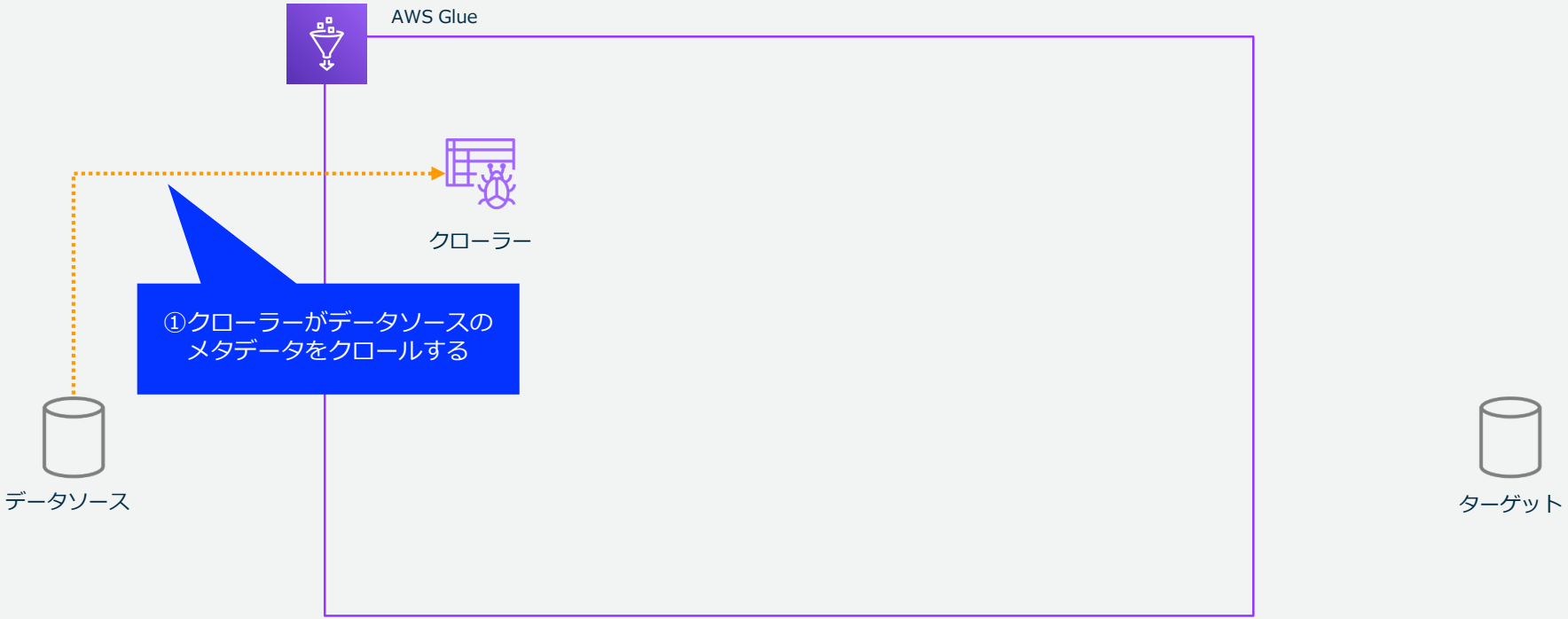
Notebookでの開発

# AWS Glueの機能

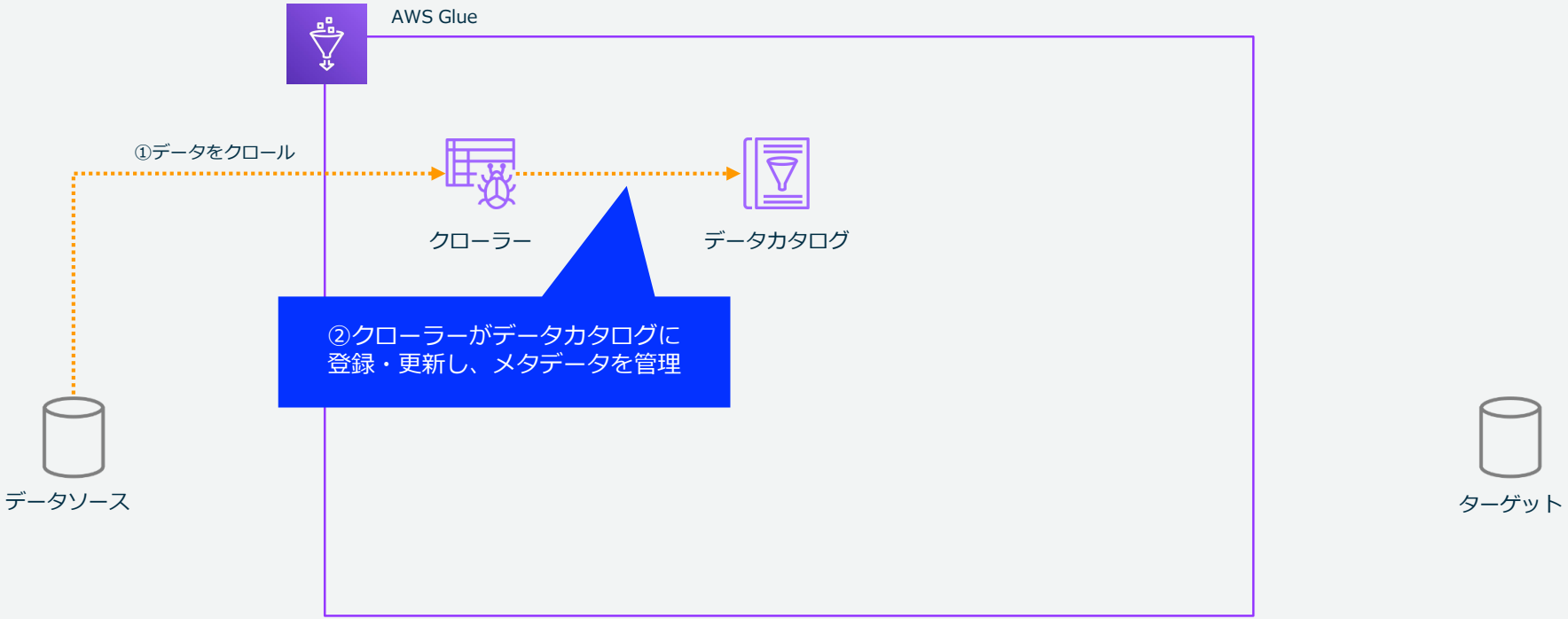
# AWS Glueの全体像



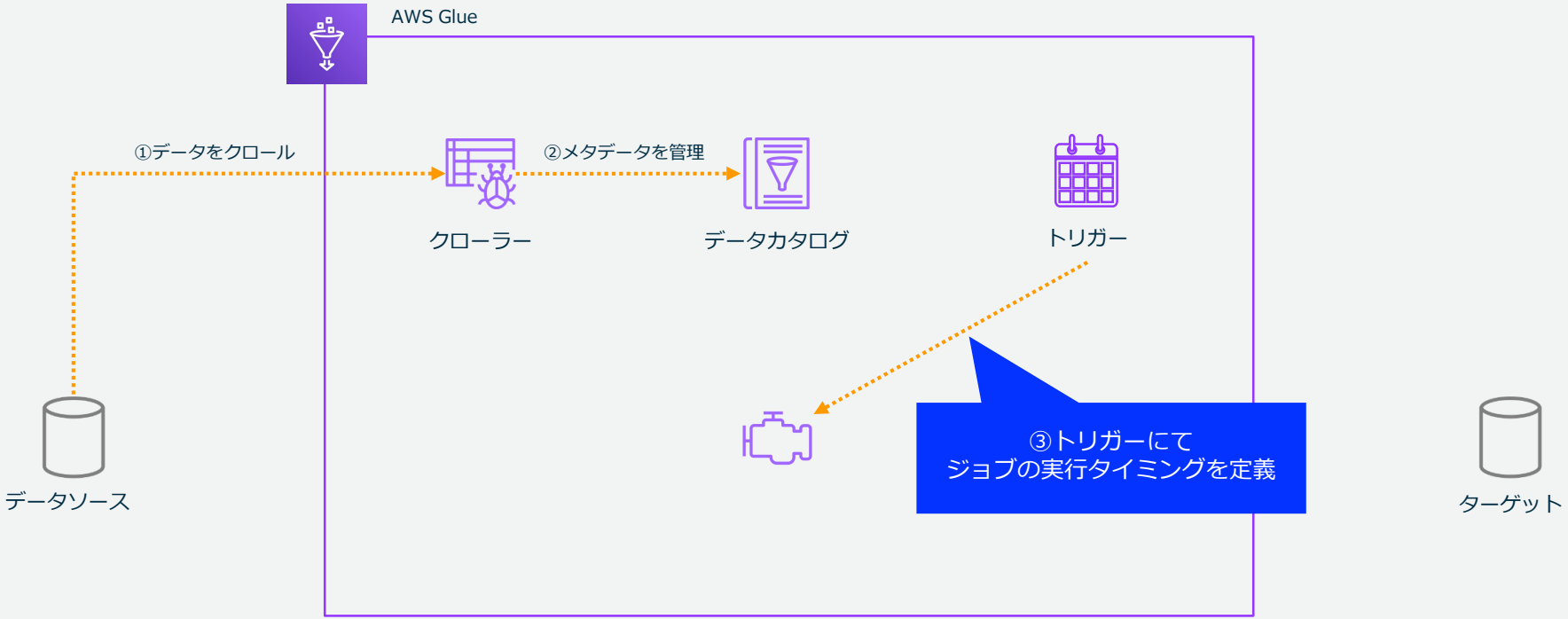
# AWS Glueの全体像



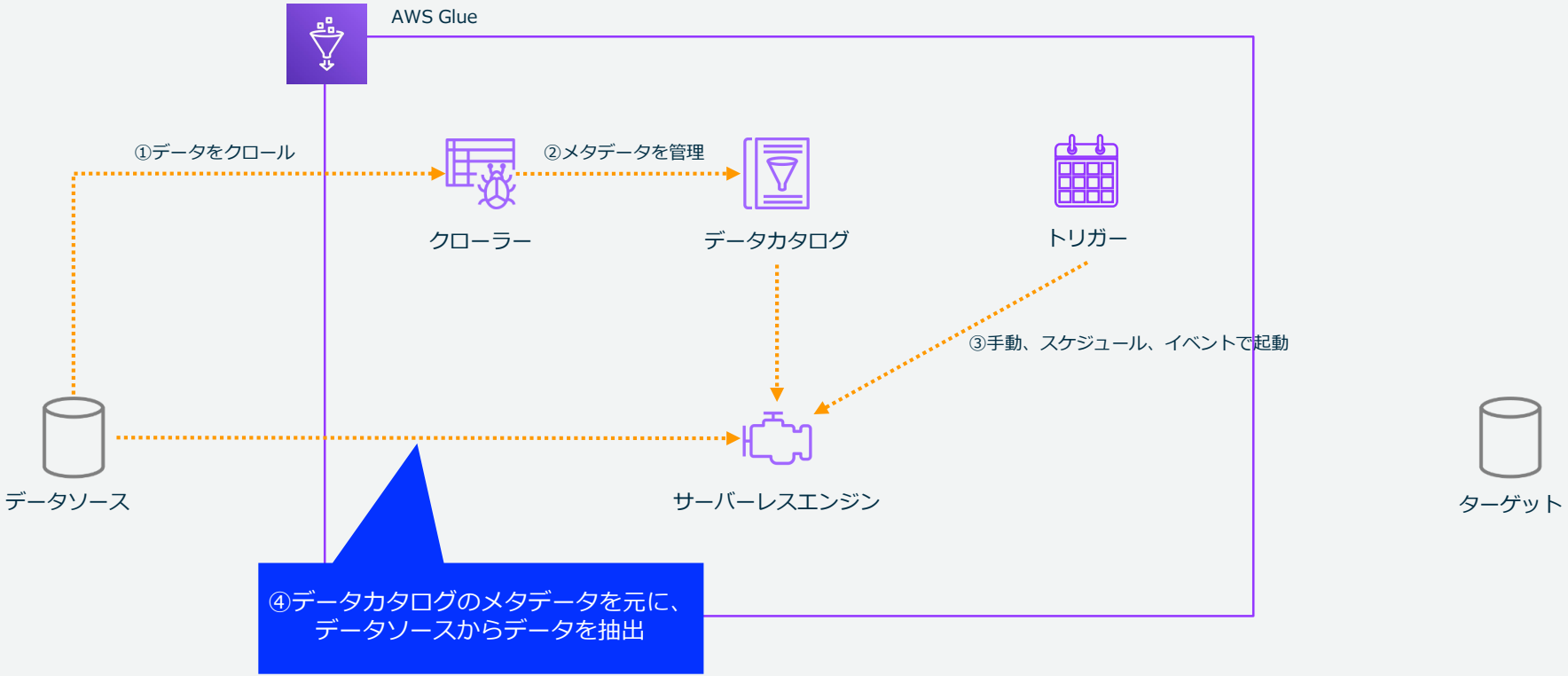
# AWS Glueの全体像



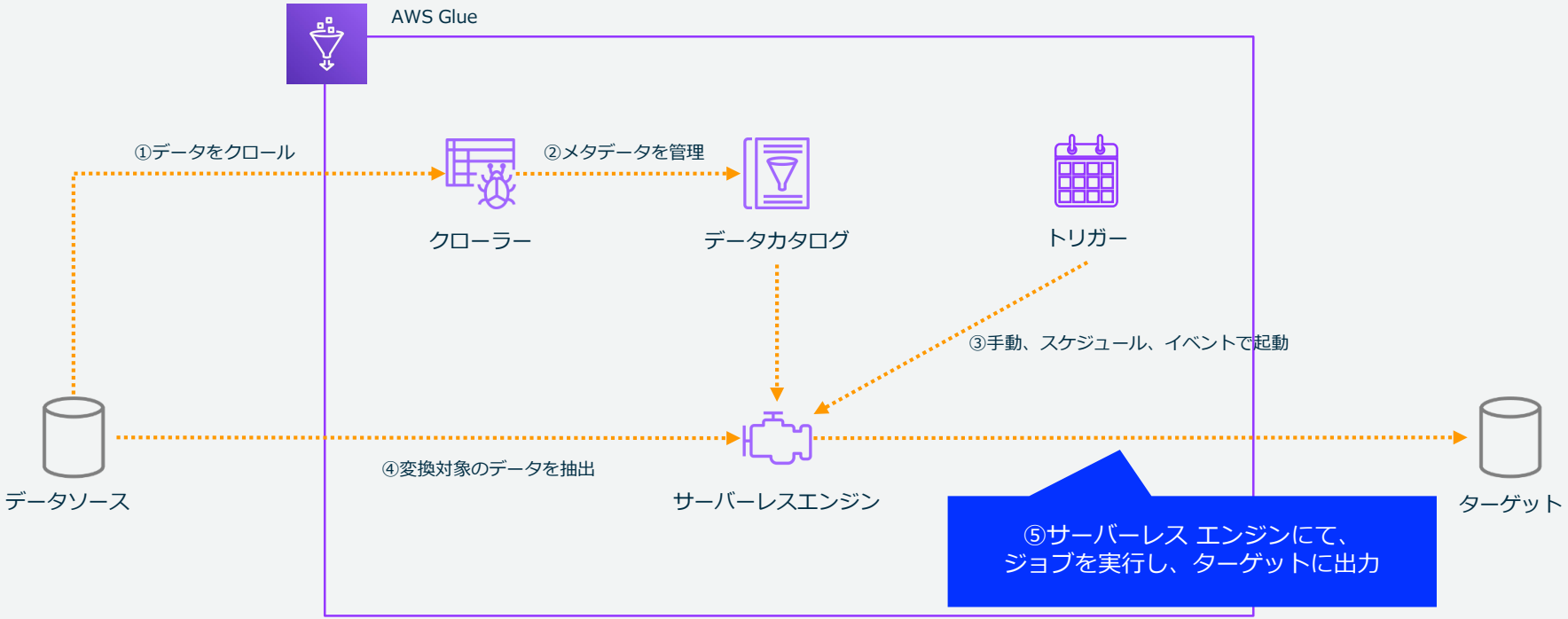
# AWS Glueの全体像



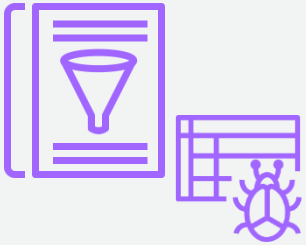
# AWS Glueの全体像



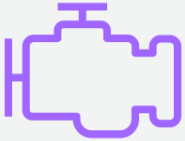
# AWS Glueの全体像



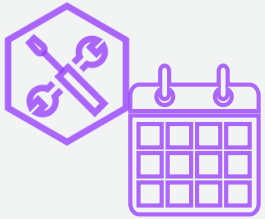
# AWS Glueの構成要素



データカタログ

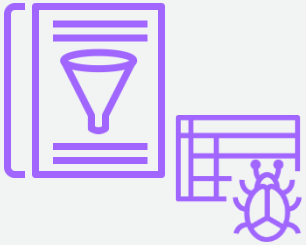


サーバーレスエンジン



オーケストレーション

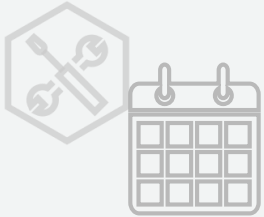
# AWS Glueの構成要素



**データカタログ**



サーバーレスエンジン



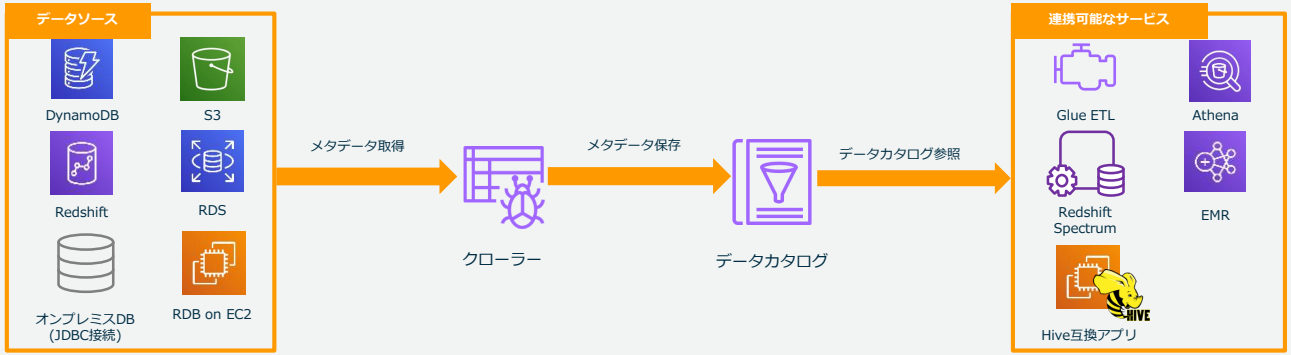
オーケストレーション

# データカタログ

## Apache Hiveメタストア互換のメタデータリポジトリ

- データカタログにメタデータを作成するにはクローラー、GlueのAPI、Hive DDL(Athena／EMR／Redshift Spectrum)の3つの方法が利用可能。
- テーブル、テーブルバージョン、パーティション、データベースのことをオブジェクトという(料金単位に関連する)
- データソースとして、Amazon DynamoDB、Amazon S3、Amazon Redshift、Amazon RDS、Amazon VPC内のRDB on Amazon EC2(Oracle、Microsoft SQL Server、MySQL、PostgreSQL)、JDBC接続可能なオンプレミスDBが指定可能
- メタデータをAmazon Redshift Spectrum、Amazon Athena、Amazon EMRに連携可能
- メタストアの管理が不要の為、運用負荷を低減できる

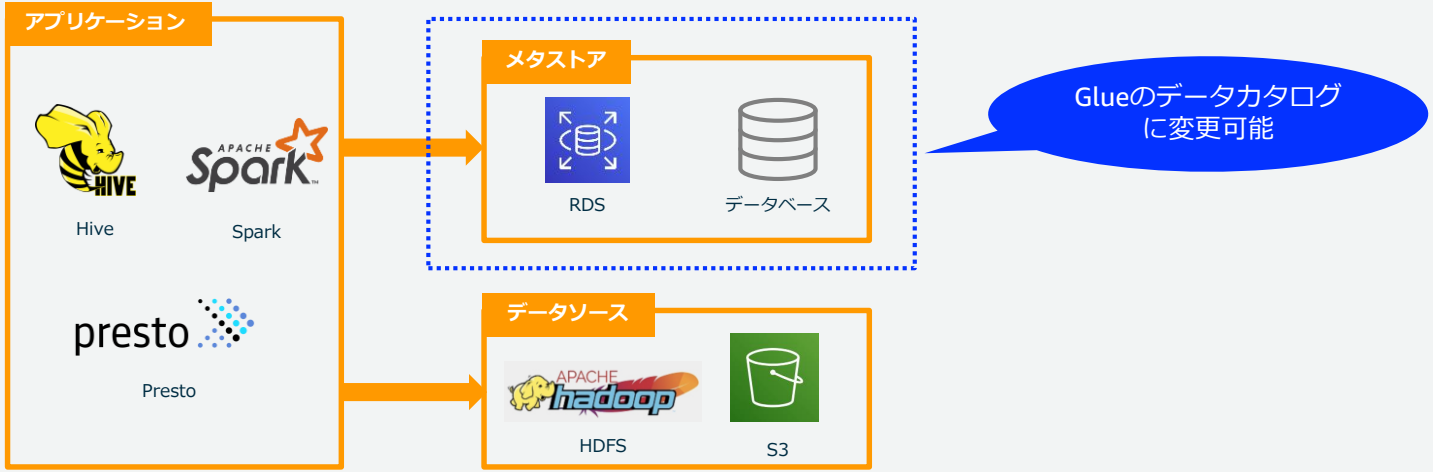
### データカタログの連携イメージ



# Apache Hiveメタストアとは

## Apache Hiveで実データとは別に表の定義だけ格納する仕組み

- 実データはHDFSやS3などに保存する
- EMRではデフォルトではマスターノード上のMySQL、外部メタストアを利用する際はRDSがHiveメタストアとして利用されていた
- その他Big Data関連のミドルウェアも参照することが可能



# クローラー

## Glueのデータカタログにメタデータを作成するプログラム

- 分類子の優先度に従って、スキーマ情報を自動で判断する
- 分類子：データのスキーマを決定するGlueの機能。分類子がデータ形式を認識するとスキーマを形成する
- クローラーを使わずにテーブル定義をAPI経由で登録することも可能
- 実行結果のログはAmazon CloudWatch Logsに出力される
- Grok・XML・JSON・CSVを用いて、分類子をカスタマイズ可能(=カスタム分類子)
- 指定したパス(S3)およびテーブル(JDBC接続)をクローラーの読込み対象外とするエクスクルードパターンも設定することが可能(※DynamoDBテーブルは未サポート)

参考URL：カスタム分類子の記述形式：https://docs.aws.amazon.com/ja_jp/glue/latest/dg/custom-classifier.html

# メタデータの構成例

| | |
|---|---|
| 名前 | sample_test_out01 |
| 説明 | |
| データベース | sample-test |
| 分類 | parquet |
| 場所 | s3://test-sample-glue-bucket-ijgowerfvwigoev542637/test_out/out01/ |
| 接続 | |
| 廃止 | いいえ |
| 最終更新日 | Thu Jun 27 10:35:27 GMT+900 2019 |
| 入力形式 | org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat |
| 出力形式 | org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat |
| Serde シリアル化ライブラリ | org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe |

テーブル情報

| | | | | | | |
|---|---|---|---|---|---|---|
| Serde パラメータ | serialization.format | 1 | | | | |
| テーブルのプロパティ | sizeKey | 18183 | objectCount | 63 | UPDATED_BY_CRAWLER | sample-test-data |
| | CrawlerSchemaSerializerVersion | 1.0 | recordCount | 30 | averageRecordSize | 143 |
| | CrawlerSchemaDeserializerVersion | 1.0 | compressionType | none | typeOfData | file |

テーブルプロパティ

スキーマ

表示中: 1 - 7 of 7

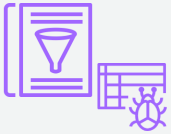
| | 列名 | データ型 | パーティションキー | Comment |
|---|---|---|---|---|
| 1 | deviceid | string | | |
| 2 | platform | string | | |
| 3 | email | string | | |
| 4 | location | string | Partition (0) | |
| 5 | year | string | Partition (1) | |
| 6 | month | string | Partition (2) | |
| 7 | day | string | Partition (3) | |

テーブルスキーマ

テーブルパーティション

クローラーがHiveパーティションを自動認識する
「sample-data/location=US/year=2019/month=08/day=06・・・」

# スキーマ管理



## データカタログに登録したテーブルのスキーマをバージョン管理することが可能

- テーブルのスキーマおよびスキーマのバージョンを一覧・比較することが可能
- 手動でスキーマ項目を追加、削除、型の変更が可能

### スキーマバージョン比較

旧バージョン

| 変更 | 列名 | データ型 | キー |
|---|---|---|---|
| | deviceid | bigint | |
| | location | string | |
| | platform | string | |
| | | | |
| | year | bigint | |
| | month | bigint | |
| | day | bigint | |

差分を比較

新バージョン

| 変更 | 列名 | データ型 | キー |
|---|---|---|---|
| | deviceid | bigint | |
| | location | string | |
| | platform | string | |
| Added | email | string | |
| | year | bigint | |
| | month | bigint | |
| | day | bigint | |

スキーマのバージョンを管理

最終更新日 27 6月 2019　テーブル　バージョン (現行バージョン)

表示中: 1 - 2

| バージョン | 作成日: | 作成者: |
|---|---|---|
| 1 | 27 6月 2019 10:1... | |
| 0 | 27 6月 2019 9:57... | [redacted] |

BigInt タイプ

列のタイプ

bigint

interval
smallint
string
timestamp

更新

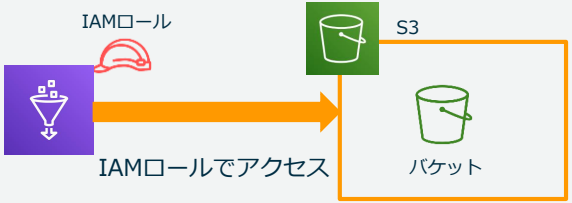
スキーマ編集画面でデータ型が変更可能

# 接続管理

## S3・DynamoDB・JDBC接続のアクセス方法

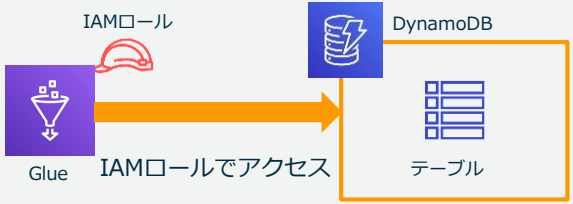
### S3

- AWS IAMでアクセスを行う
- S3バケットを指定する



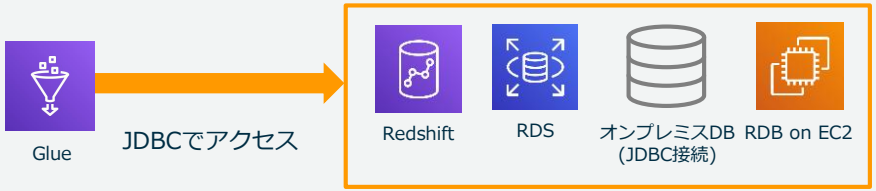

### DynamoDB

- AWS IAMでアクセス制御する
- テーブル名を指定する




### JDBC接続

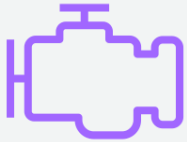
- 事前に接続設定を追加する
  (インスタンス名・データベース名・ユーザー名・パスワードを設定)
- 自己参照型のセキュリティグループでアクセス制御



# AWS Glueの構成要素



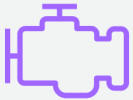
データカタログ



**サーバーレスエンジン**

オーケストレーション

# ジョブ作成



- ETLの処理単位をジョブといい、ジョブの種類にApache SparkとPython Shellがある
  (Python Shell：Pythonスクリプトを実行する機能)
- Glueが自動生成したコード、自身で作成するスクリプト、既存のコード(オンプレミスで動作していたものも可)が実行可能
- ジョブの状態を追跡(=チェックポイント)できるブックマーク機能がある
- SparkとPython Shellは下記バージョンをサポート

| Glueのバージョン | Spark | Python Shell |
|---|---|---|
| Glue 0.9 | Spark 2.2.1<br>(Python 2) | — |
| Glue 1.0 | Spark 2.4.3<br>(Python 2、Python 3) | Python 2.7<br>Python 3.6 |

New

※Python Shellについては、Glue1.0のみ指定可能

参考URL：Glueバージョン(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/release-notes.html)
Spark Overview(https://spark.apache.org/docs/latest/)



aws

# Worker Type New

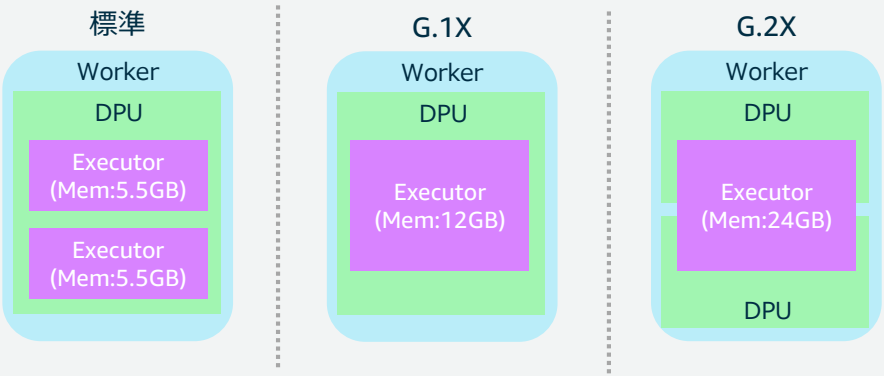
## Glue内のSparkジョブにメモリ大量使用ワークロード向けのWorker Typeが指定可能に

- ジョブ実行時に割り当てる処理能力をDPU(Data Processing Unit)という
  1DPU = 4vCPU、16GBメモリ
- これまでの標準に加えて、G.1xとG.2Xが選択可能に。

### Worker Type一覧

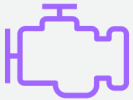
| Worker Type | DPU数 /1Worker | Executor数 /1Worker | メモリ数 /1Executor |
|---|---|---|---|
| 標準 | 1 | 2 | 5.5GB |
| G.1X | 1 | 1 | 12GB |
| G.2X | 2 | 1 | 24GB |

### Worker Type構成イメージ




参考URL：Spark Components(https://spark.apache.org/docs/latest/cluster-overview.html)

# SparkでETL実行した際に起きうる課題



DataFrame：データをテーブル構造で扱えるSparkの機能。SparkSQLを用いて、DataFrameを操作する

■テーブル例(特定カラムで複数の型が存在する場合)

| Col_a | Col_b | Col_c |
|---|---|---|
| | | 1 |
| | | 2 |
| | | 3 |
| | | 4 |
| | | ・・・ |
| | | ・・・ |
| | | 1,000,000 |
| | | “1000001” |
| | | “1000002” |

bigint(数値型)

string(文字列)

複数の型が存在した場合、処理が中断し、再処理しなければならない可能性がある

**DataFrame処理前にデータの中身を調査して、事前に複数の型が混じることを想定したETLコードを記述する必要がある**



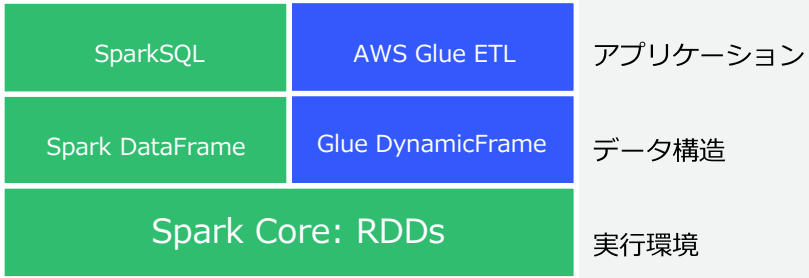
aws

# DynamicFrameとは

## SparkSQL DataFrameと似たGlue特有の抽象化の概念

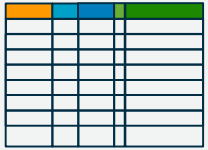
- SparkSQL DataFrameとの違いはETLに特化しているかどうか
  (DynamicFrameはスキーマの不一致を明示的にエンコードする”Schema on the Fly”を採用)
- 複数の型の可能性を残して、後で決定できるようにする (Choice型)
- DynamicFrameはデータ全体を表し、DynamicRecordはデータ１行を指す
- DataFrameとDynamicFrame間でそれぞれ変換することができる(fromDF関数・toDF関数)
- Pythonライブラリ PandasのDataFrameとは異なるので注意

**アーキテクチャ：SparkおよびGlueライブラリ**



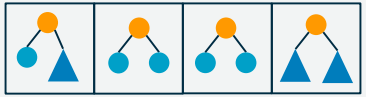

**データ構造イメージ**

SparkSQL DataFrame



構造テーブルに類似

DynamicFrame



半構造テーブルに類似

# Choice型

## DynamicFrameの列で複数の型を発見した場合に両方の型を持つことができる

- ResolveChoiceメソッドで型を解決することが可能

### choice型のデータ構造例

```
root
  |-- uuid: string
  |
  |-- device id: choice
  |        |-- long
  |        |-- string
```

device id列はlongとstringの両方のデータを持っている
(例：device idカラムに数字の1234と文字列の”1234”が混同する)

### ResolveChoiceの実行例

# ブックマーク機能

## ジョブの実行状態を保持する機能

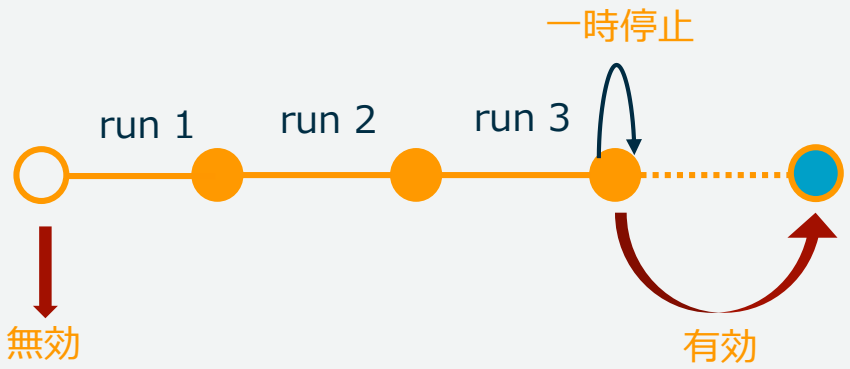
- 定常的にETL処理が必要な場合において有効

例：処理済みデータを再度処理しないように回避
処理結果のデータをターゲットに重複出力しないように回避

**設定内容**

| 設定 | 内容 |
|---|---|
| 有効 | 中断した場所から実行する |
| 無効 | 最初からジョブを実行する |
| 一時停止 | ブックマークの進行を一時的に無効にする |

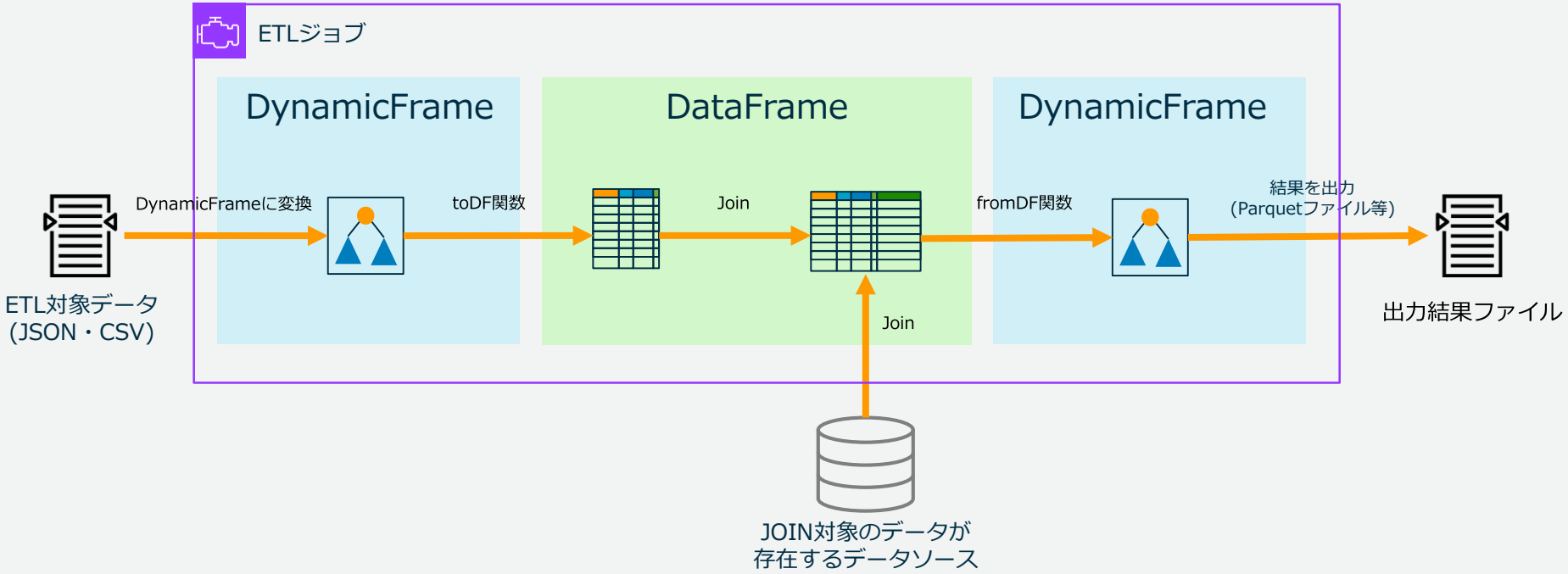
**実行イメージ**



# DynamicFrameとDataFrameの特性を生かしたETL処理

## ETLジョブの例

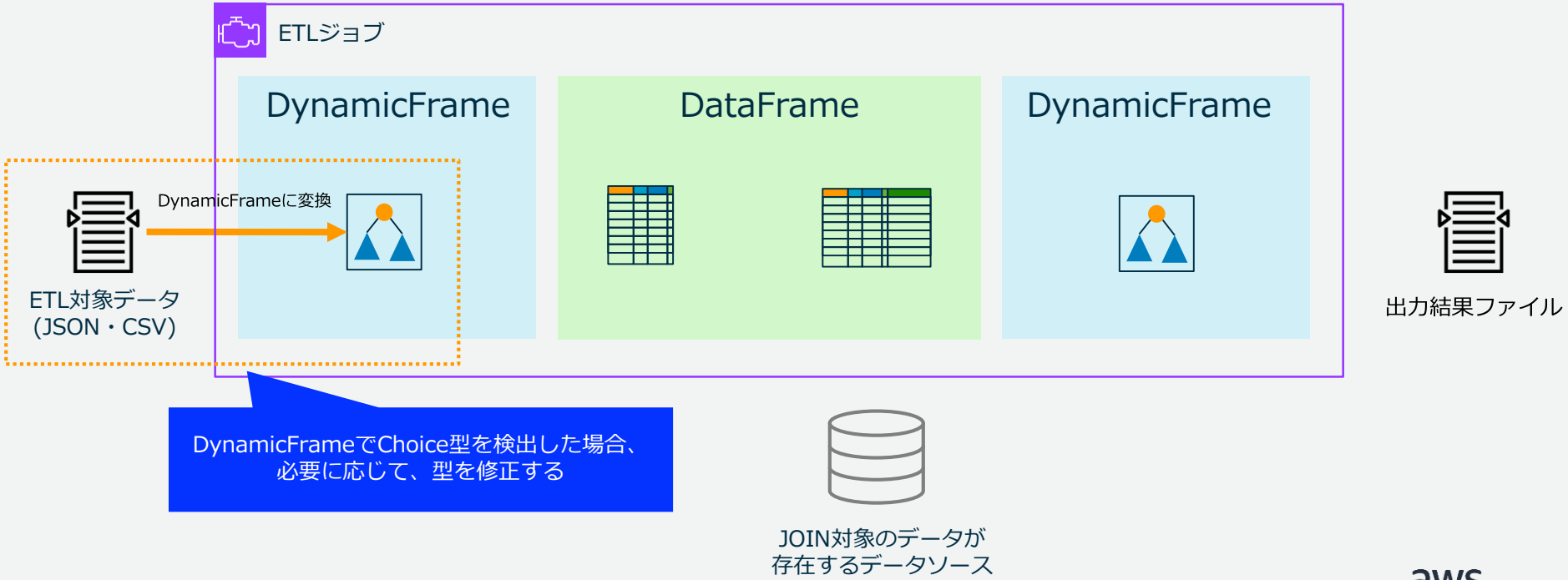
- JSONやCSVデータに対して、他のデータソースとJOIN、ファイル形式を変換して出力するジョブ



# DynamicFrameとDataFrameの特性を生かしたETL処理

## ETLジョブの例

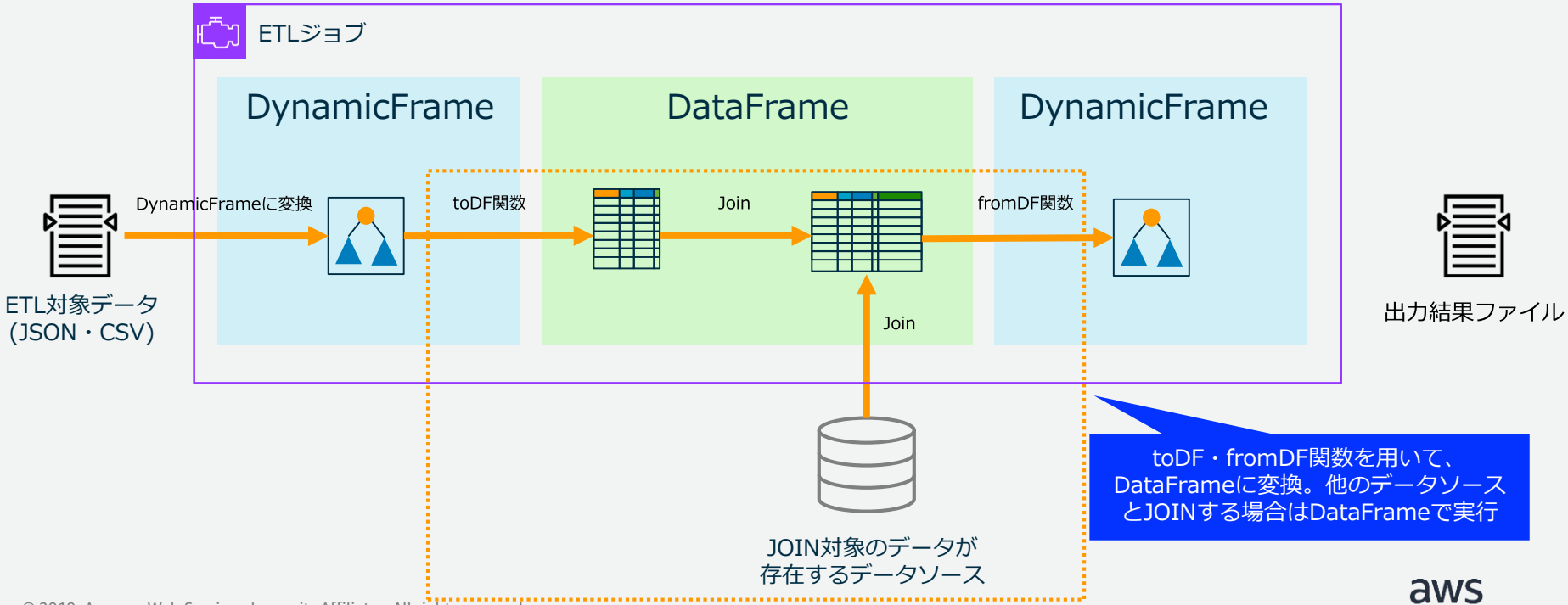
- JSONやCSVデータに対して、他のデータソースとJOIN、ファイル形式を変換して出力するジョブ



# DynamicFrameとDataFrameの特性を生かしたETL処理

## ETLジョブの例

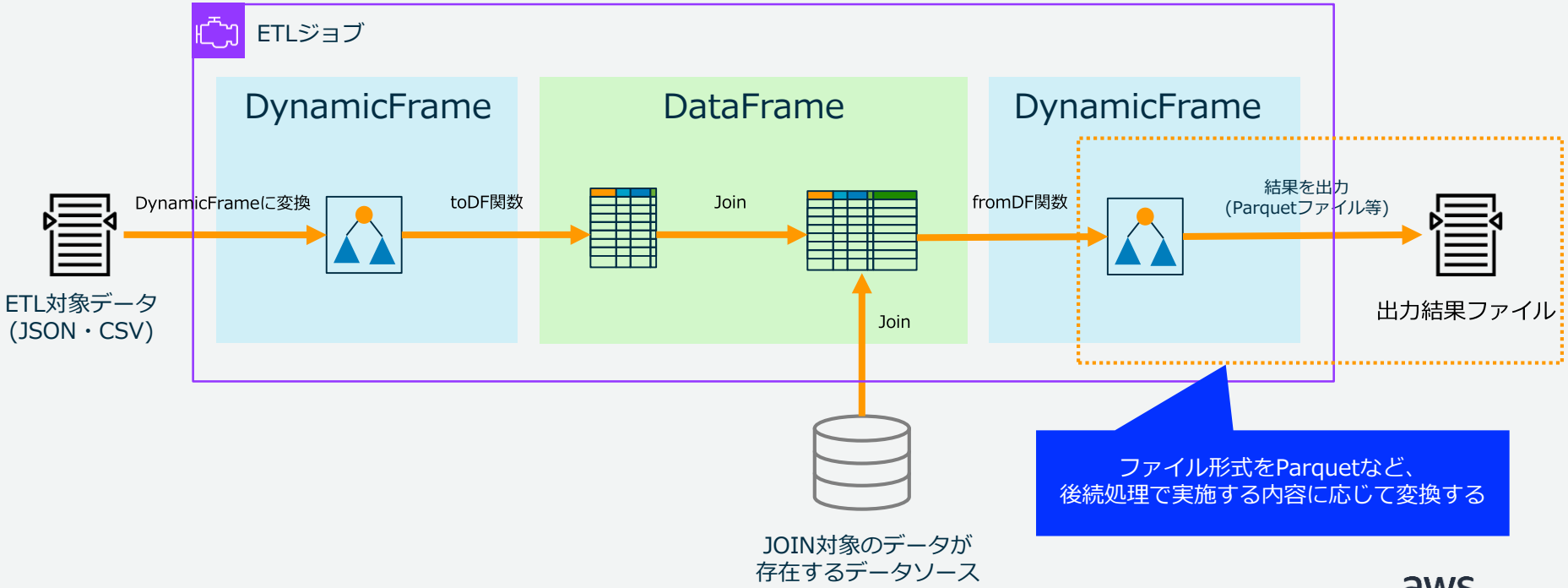
- JSONやCSVデータに対して、他のデータソースとJOIN、ファイル形式を変換して出力するジョブ



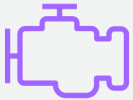
# DynamicFrameとDataFrameの特性を生かしたETL処理

## ETLジョブの例

- JSONやCSVデータに対して、他のデータソースとJOIN、ファイル形式を変換して出力するジョブ



# PySparkコードサンプル



## 初期化処理からデータソースへのアクセス、ジョブコミットまで

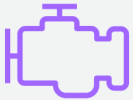
```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(arg['JOB_NAME'], arg)

memberships =
glueContext.create_dynamic_frame.from_catalog(
                        database="legislators",
                        table_name="memberships_json")
//省略
job.commit()
```

- GlueContextはSpark SQLContextを継承して、独自拡張したクラス
- create_dynamic_frame.from_catalogでデータカタログ経由でDynamicFrameを作成する
- create_dynamic_frame.from_RDDでSparkのRDDからDynamicFrameを作成可能
- create_dynamic_frame_from_optionsでデータカタログを経由せず、直接データソースにアクセスし、DynamicFrameを作成することも可能
- DynamicFrameだけでなく、SparkSQL DataFrameを記述、実行することも可能



aws

# PySparkコードサンプル



## push_down_predicateオプション

- DynamicFrame生成前にPre-Filteringすることでデータの読み込みを削減可能

```
my_partition_predicate = "(country=='JP' and year=='2019' and month=='08' and day=='06')"
glue_context.create_dynamic_frame.from_catalog(database = "my_S3_data_set",
   table_name = "catalog_data_table",
   push_down_predicate = my_partition_predicate)
```

## groupFiles、groupSizeオプション

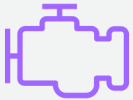
- データソースからファイルを読み取る際、グループ化する
- S3パーティション内のデータをグループ化する際はgroupFiles、読み取るグループのサイズをgroupSizeオプションで指定する
- スモールファイルをまとめて処理する際に、処理効率がよくなる

```
df =  glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://s3path/"],
 'recurse':True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```



aws

# PySparkコードサンプル



## ApplyMapping

- ETLのターゲットとなる列をDynamicFrameに合わせる処理

```
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("deviceid", "string", "deviceid", "string"), 【省略】,
("day", "long", "day", "long")], transformation_ctx = "applymapping1")
```

## ターゲットへの書き込み

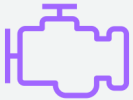
- write_dynamic_frame_catalogで出力。例ではParquet形式で出力

```
datasink = glueContext.write_dynamic_frame.from_options(
        frame = dropnullfields3, connection_type = "s3",
        connection_options = {
                "path": "s3://xxxxxxxxx/xxxxx/xxxxx"
        }, format = "parquet", transformation_ctx = "datasink")
```

・"parquet"、"orc"などのファイル形式を指定することが可能
・より Glue に最適化された "glueparquet"を利用することで、出力ファイルのスキーマを動的に計算し、
高速に"parquet"ファイルに書き込むことが可能 (通常の parquet ファイルとして読み込み可能)

参考URL：ETL 入力・出力形式オプション(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/aws-glue-programming-etl-format.html)

# Python Shellコードサンプル



## Redshiftに接続し、クエリを実行する例

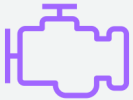
```
def get_connection(host):
    rs_conn_string = "host=%s port=%s dbname=%s
user=%s password=%s" % (
        host, port, db_name, user, password_for_user)

    rs_conn = pg.connect(dbname=rs_conn_string)
    rs_conn.query("set statement_timeout = 1200000")
    return rs_conn


def query(con):
    statement = "Select * from table_name;"
    res = con.query(statement)
    return res
```

デフォルトで利用可能ライブラリ例

- Boto3
- CSV
- gzip
- Numpy
- pandas
- Scipy
- sklearn
- zipfile

参考URL：GlueでのPythonシェルジョブの追加(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/add-job-python.html

# サーバーレスETL処理の使い分け



## 小規模処理

### AWS Lambda

- 15分以内に完了できる処理
- 豊富なトリガー(S3に配置されたタイミングで逐次処理)
- Pandasなどのライブラリが利用可能

## 中規模処理

### AWS Glue
Python Shell

- 実行時間の制限なし
- Lambdaに比べてメモリ量が多い(1GBまたは16GB(※1))
- Pandasなどのライブラリが利用可能
- RedshiftやEMR、Athenaに対するSQLベースの分析

## 大規模処理

### AWS Glue
Spark

- 実行時間の制限なし
- 並列分散処理が得意
- 大量データの処理

**データの規模やETL処理の中でやりたいことによって使い分ける**

(※1)Python Shellでは0.0625(1/16)DPUと1DPUが選択でき、その計算結果を記載。

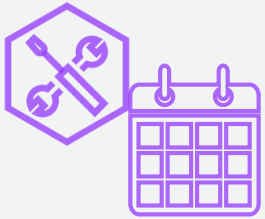
# AWS Glueの構成要素



データカタログ



サーバーレスエンジン



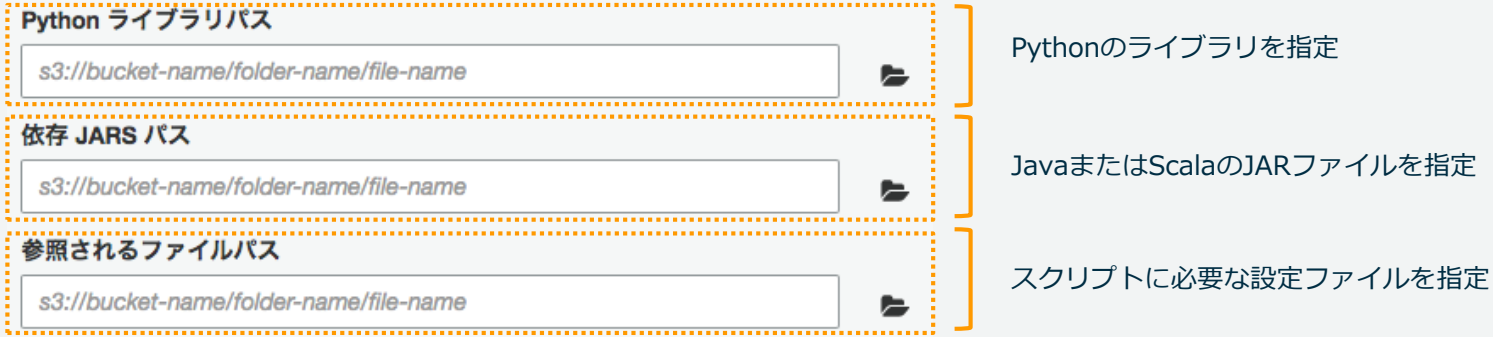
**オーケストレーション**

# 独自ライブラリの利用

## PythonおよびScalaの独自ライブラリが利用可能

- Spark(PySpark、Scala)、Python Shellともに独自のライブラリを利用することが可能
- S3にPythonのライブラリ、ScalaのJARファイルをアップロードし、パスを指定する
- S3のURLをカンマ区切りで記述することで複数のライブラリを指定可能
- PySpark利用時、C言語に依存するPandasなどのライブラリは利用できない。Pandasを利用したい場合は、Python Shellを推奨
- Python Shell：Python2.7または3.6互換のライブラリを指定可能

**例：Sparkのライブラリ指定場所**



# トリガー

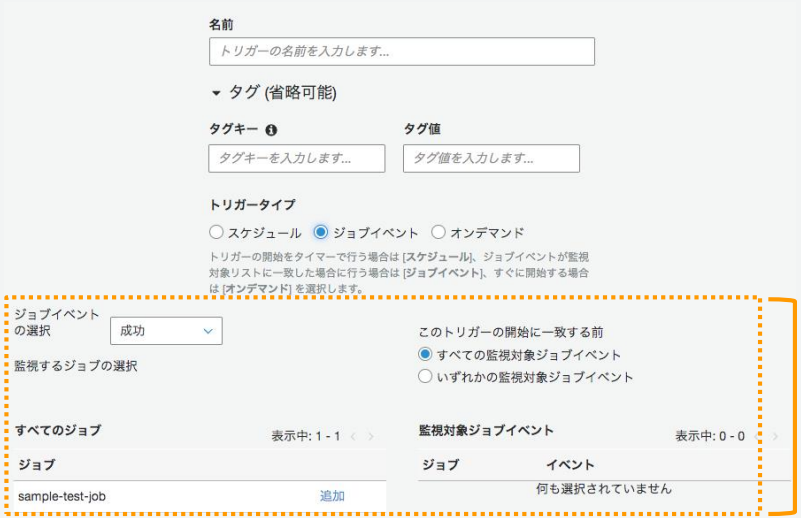
## ジョブを開始するための定義を設定できる機能

- スケジュール(日時・曜日・cron)、ジョブイベント、手動(即時実行)で指定可能

### スケジュール起動


ジョブの起動スケジュールを指定

### ジョブイベント起動




先行ジョブを指定

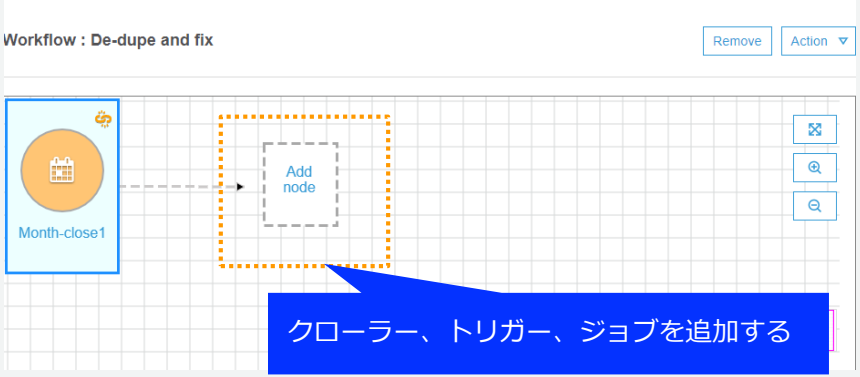
参考URL：トリガーの指定方法(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/trigger-job.html)

# ワークフロー機能 New

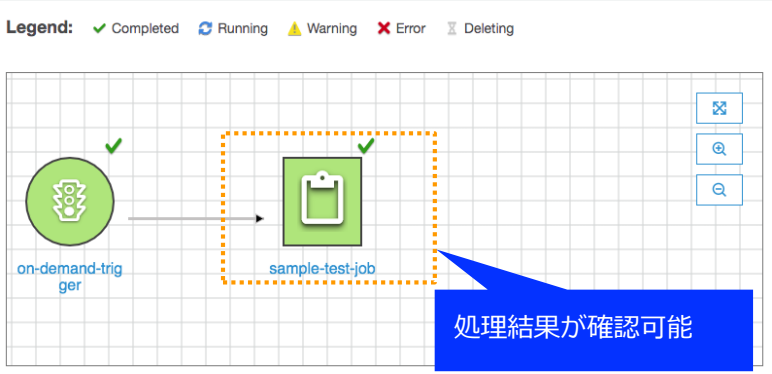
## クローラー、トリガー、ジョブのDAGを生成するワークフロー機能

- DAG(有向非巡回グラフ)：ある頂点からある頂点の方向が決まっており(有向)、同じ辺を通らない(非巡回)
- ワークフローの状況をモニタリングすることや、エラー時のトラブルシューティングを視覚的に確認可能
- boto3を利用したPython Shellを実行することで、他のAWSサービスと連携することが可能

### ワークフロー作成画面




### 処理結果確認画面




参考URL：ワークフロー機能(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/orchestrate-using-workflows.html)

# 開発

# 開発

## ETLジョブのコードを開発／実行するための環境

### 開発エンドポイント

- ジョブ実行環境に直接アクセスするためのアクセスポイント
- Glueでジョブを実行するために開発したコードを動かす実行環境
- 開発エンドポイントでG.1XとG.2XのWorker Typeを選択することが可能 New

### Notebookサーバー

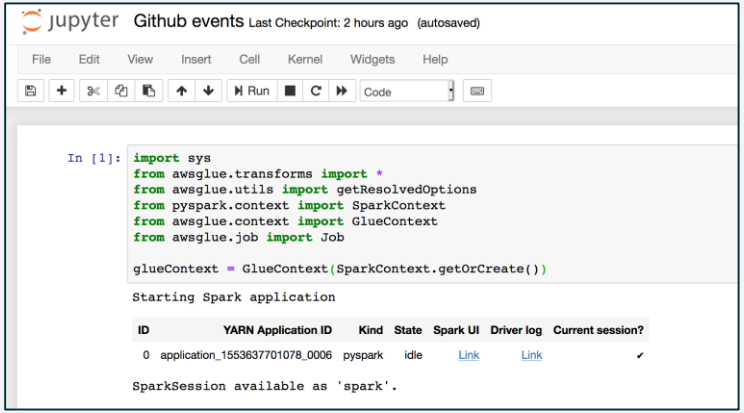
- Amazon SageMaker Notebook(Jupyter Notebook)もしくはApache Zeppelin Notebookが利用可能
- SageMaker Notebook、Zepplin NotebookともにVPC内にNotebookインスタンスを起動することが可能
- プログラムそのものの記述と実行結果を表示する環境
- 開発エンドポイントを削除しても、Notebookインスタンスは削除されない為、手動削除が必要

# SageMaker Notebook

- Glueのコンソール上でSageMaker Notebookサーバーを起動する
- SageMakerでSparkライブラリを利用することが可能
- ジョブ・開発エンドポイントでデータカタログが指定できるようになった為、データカタログに保存されているテーブルに対して、SageMaker Notebookから直接SparkSQLが実行可能に

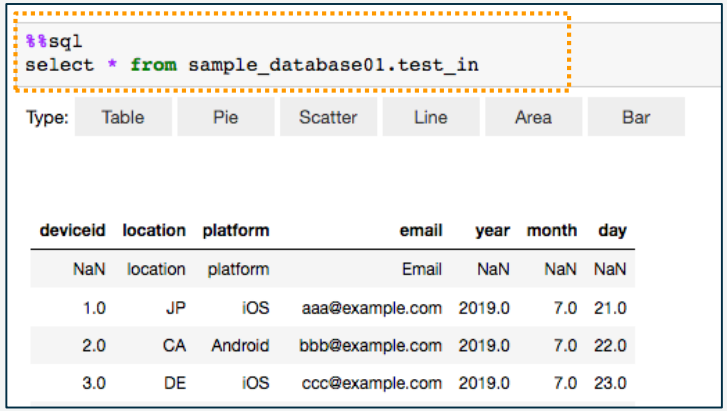
New

**SageMaker Notebook イメージ**




**IPythonファイルを作成時にPySparkが指定可能**

**Notebook内でのSparkSQLの実行**

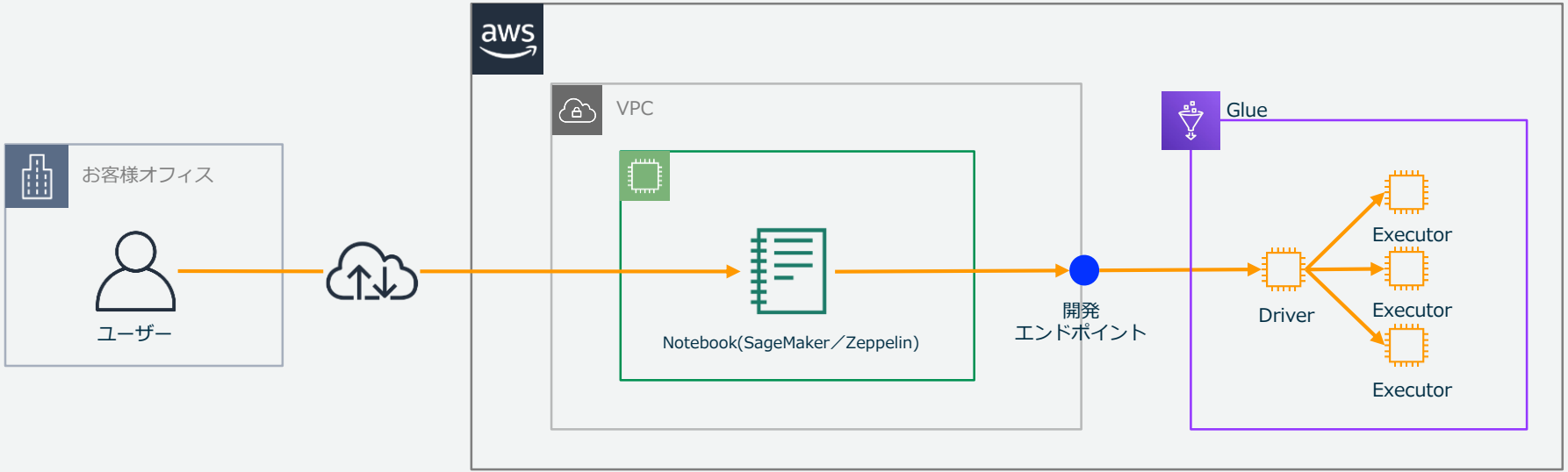



**「%%sql」を記述することでSparkSQLが実行可能**

参考URL：SageMakerでSparkを利用する(https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/apache-spark.html)

# 開発エンドポイントとNotebookの関係

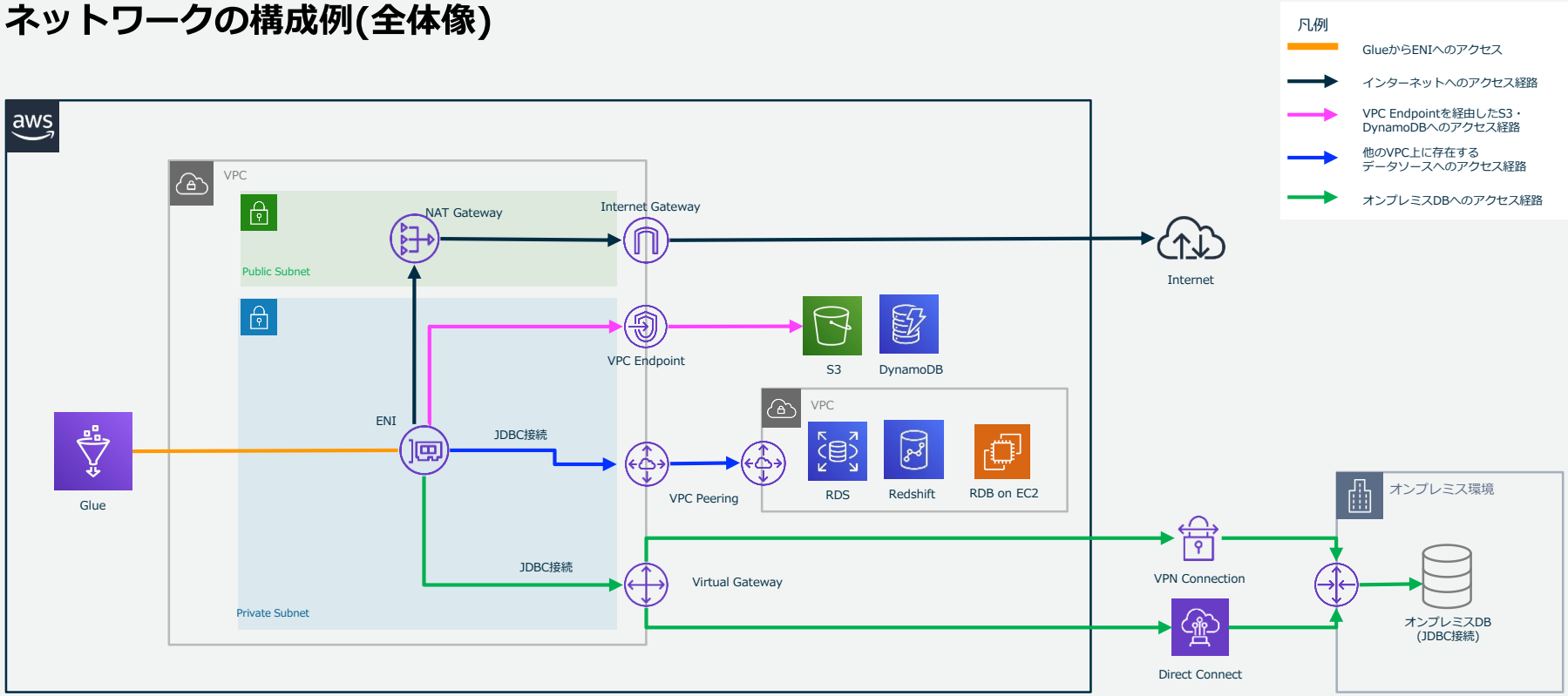
## ユーザーからNotebook、開発エンドポイント・Glue(PySpark)へのアクセスイメージ



# ネットワーク／セキュリティ／監視

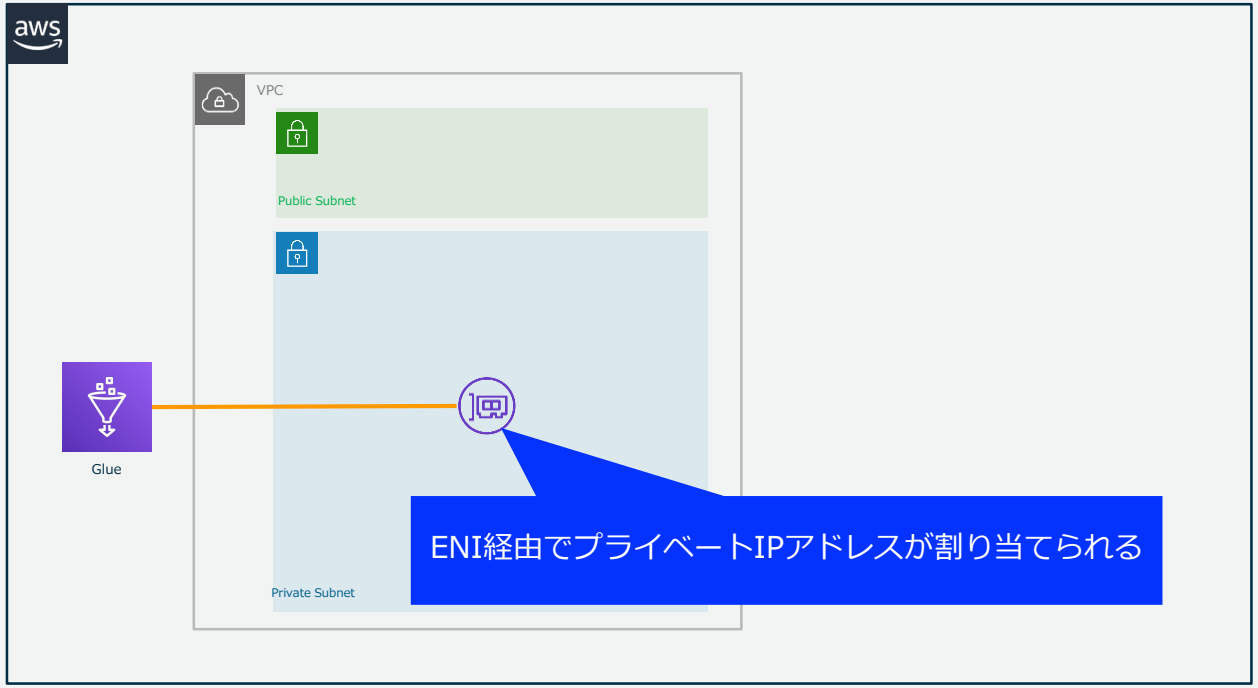
# GlueからVPCへのアクセス

## ネットワークの構成例(全体像)



# GlueからVPCへのアクセス

## ネットワークの構成例(GlueからVPCへのアクセス)

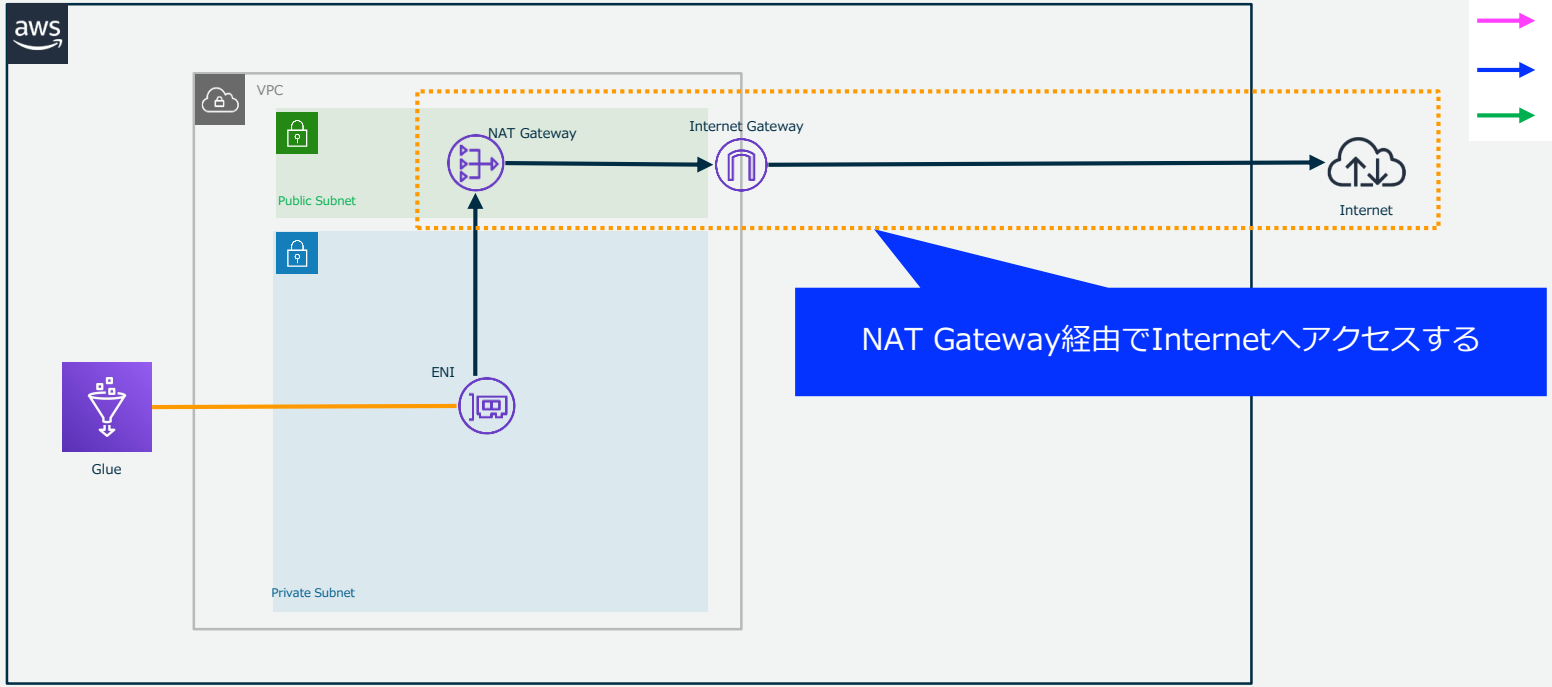



凡例

- GlueからENIへのアクセス
- インターネットへのアクセス経路
- VPC Endpointを経由したS3・DynamoDBへのアクセス経路
- 他のVPC上に存在するデータソースへのアクセス経路
- オンプレミスDBへのアクセス経路

# GlueからVPCへのアクセス

## ネットワークの構成例(インターネットへのアクセス)

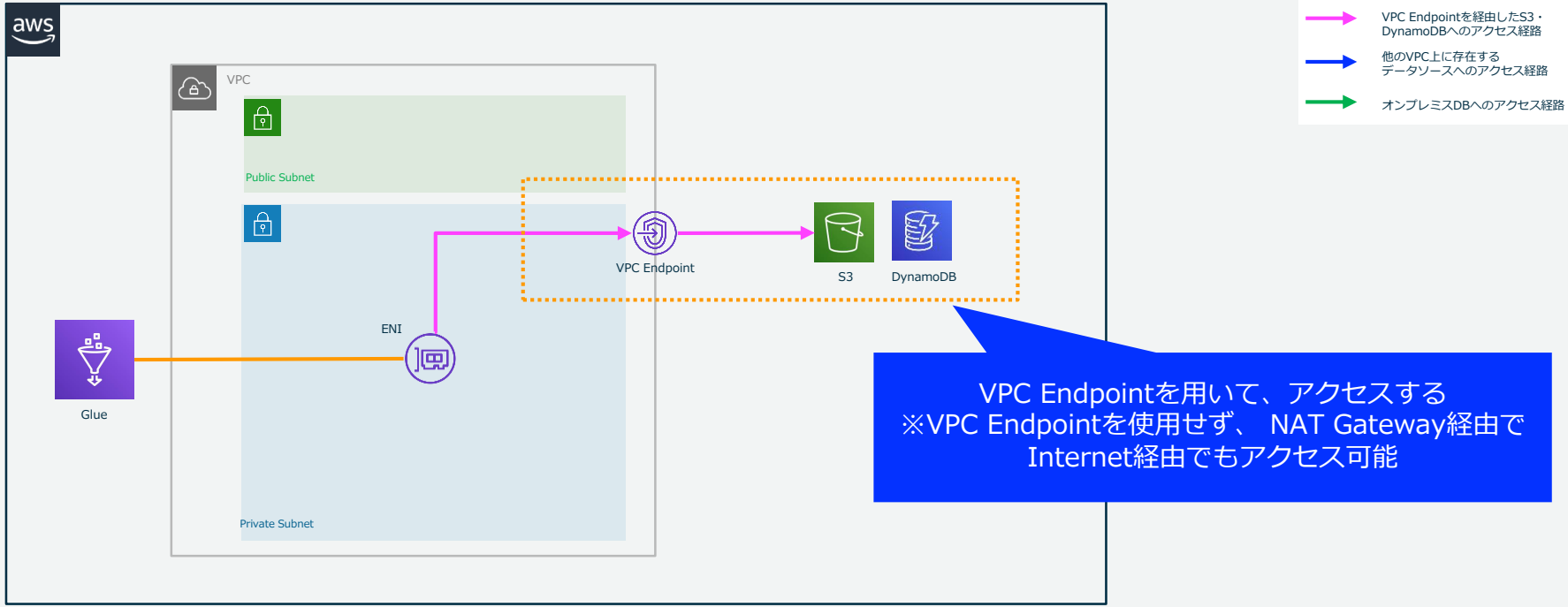



凡例

- GlueからENIへのアクセス
- インターネットへのアクセス経路
- VPC Endpointを経由したS3・DynamoDBへのアクセス経路
- 他のVPC上に存在するデータソースへのアクセス経路
- オンプレミスDBへのアクセス経路

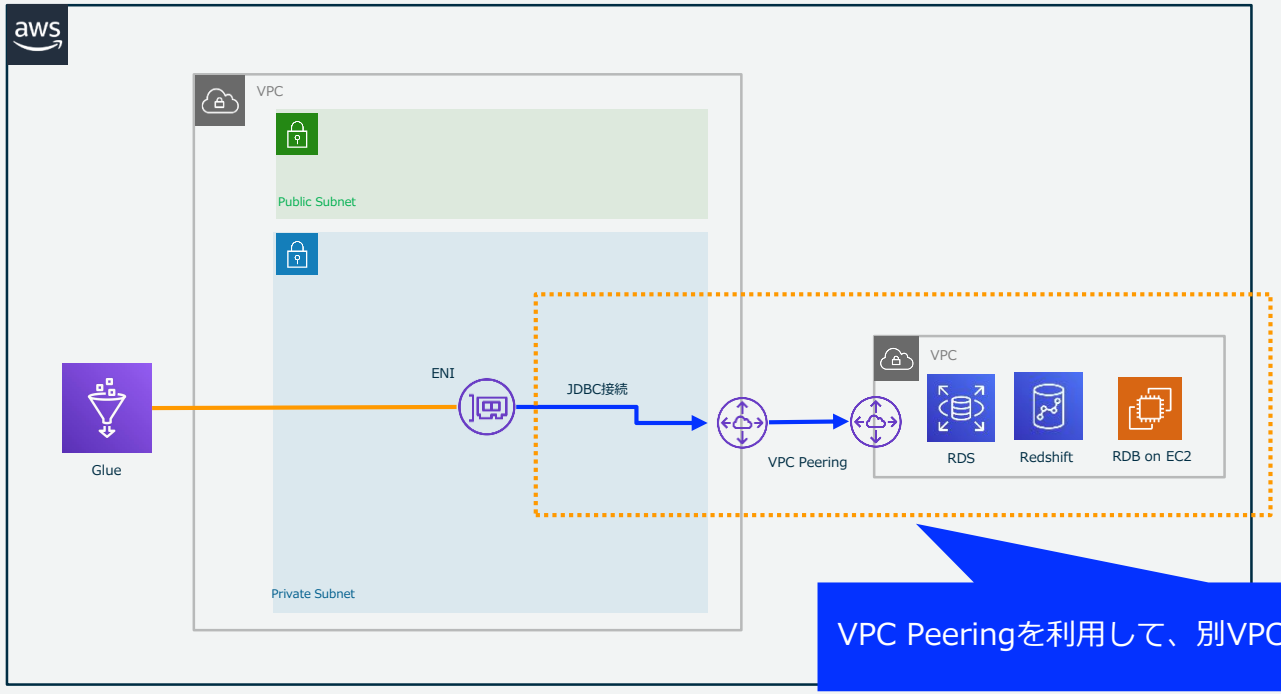
# GlueからVPCへのアクセス

## ネットワークの構成例(VPC Endpoint経由でS3・DynamoDBにアクセス)



# GlueからVPCへのアクセス

## ネットワークの構成例(別VPCにあるデータベースにJDBC接続)



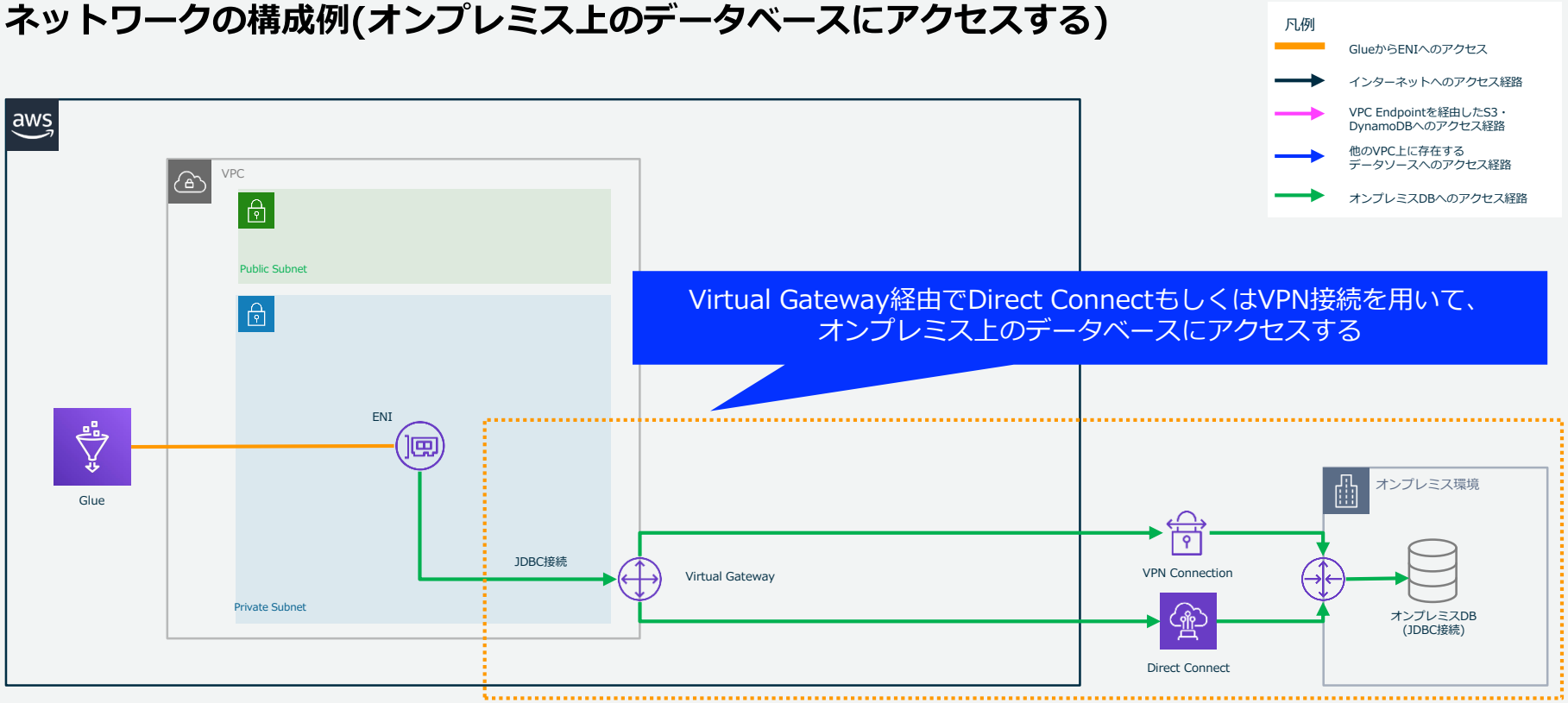

凡例

- GlueからENIへのアクセス
- インターネットへのアクセス経路
- VPC Endpointを経由したS3・DynamoDBへのアクセス経路
- 他のVPC上に存在するデータソースへのアクセス経路
- オンプレミスDBへのアクセス経路

VPC Peeringを利用して、別VPC内にあるデータベースにアクセス

# GlueからVPCへのアクセス

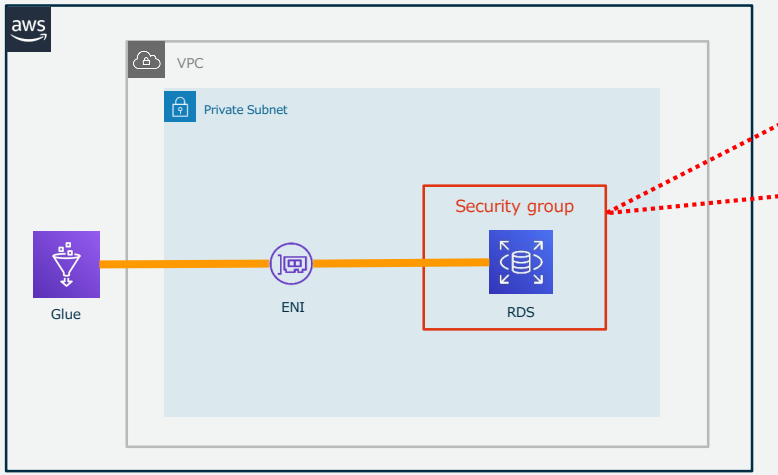
## ネットワークの構成例(オンプレミス上のデータベースにアクセスする)



# セキュリティグループ

- RDS・Redshiftなどインスタンス単位でアクセス制御を行う場合、Glueからアクセスできるようにセキュリティグループを設定する
- 自己参照型のセキュリティグループを設定する
  (自己参照型：自分のセキュリティグループからのアクセスをすべて許可する設定)

**セキュリティグループの設定例**




セキュリティグループ(RDS)

セキュリティグループID：sg-xxxxxxxx

| ポート番号 | IPアドレス |
|---|---|
| 3306 | 10.0.0.0/16 |
| ALL | sg-xxxxxxxx |

RDSのセキュリティグループIDを設定(=自己参照型)

参考URL：自己参照グループの設定(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/setup-vpc-for-glue-access.html)

# IAM設定

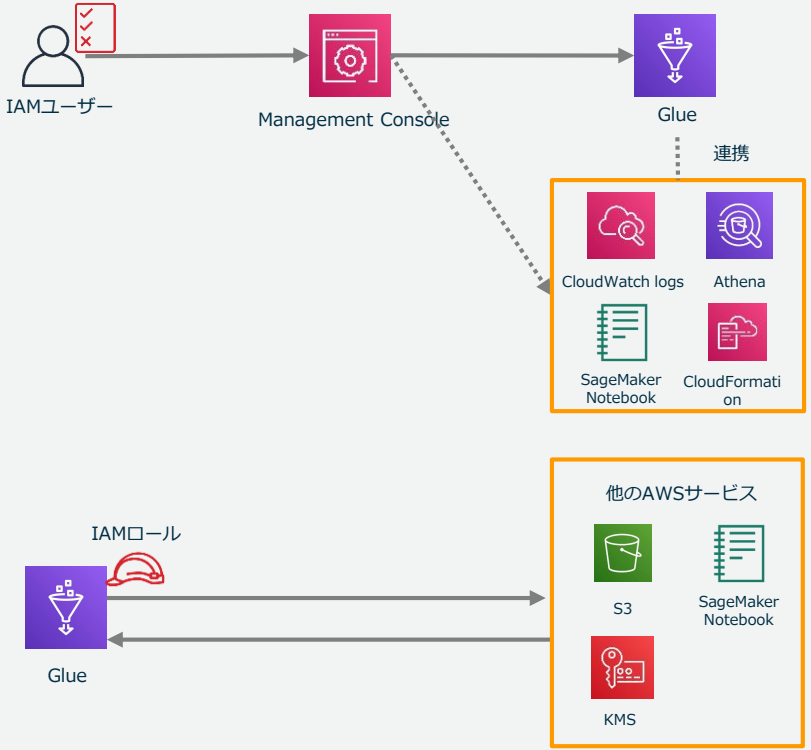
## IAMユーザー・IAMロールを用いて、Glueの権限管理を行う

### IAMユーザー

- AWSマネジメントコンソール上でGlueを利用する際に必要な権限(他サービスはバックグラウンドで使用)
- IAMユーザーで必要な権限(ノートブック利用を含む)
  - ・AWSGlueConsoleFullAccess
  - ・CloudWatchLogsReadOnlyAccess
  - ・AWSGlueConsoleSageMakerNotebookFullAccess
  - ・AWSCloudFormationReadOnlyAccess
  - ・AmazonAthenaFullAccess

### IAMロール

- ETLジョブ実行および開発エンドポイントに対してアクセスする際に、IAMロールを指定する
- ETLジョブは「AWSGlueServiceRole」、開発エンドポイントは「AWSGlueServiceNotebookRole」が基本ポリシーであり、他サービスと連携する(S3など)場合はそのサービスのIAMロールを設定する
- S3上のファイルが暗号化されている場合は復号可能なロールも必要



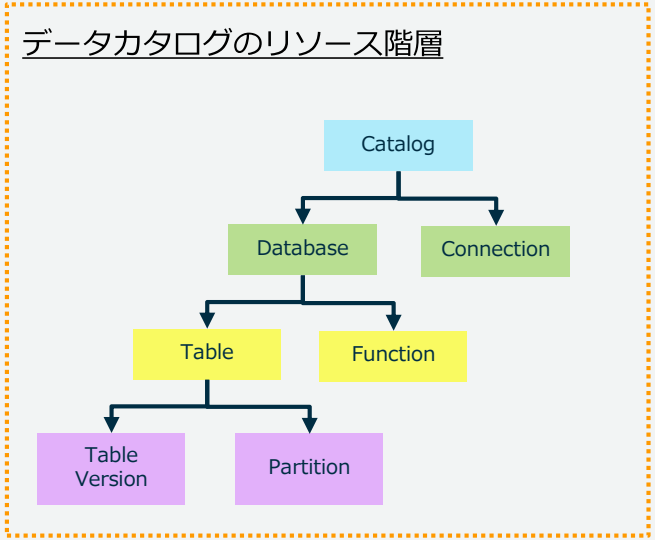
# リソースレベルによるポリシーとアクセス許可 New

## Glue内で管理するアクセスポリシー

- データカタログリソースへのアクセス制御を行う
- データカタログに対するクロスアカウント・クロスリージョンのアクセス制御が可能

**リソースポリシーの例**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam:: account-A-id:user/dev",
        "arn:aws:iam:: account-B-id:root"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:table/db1/*",
        "arn:aws:glue:us-east-1:account-A-id:database/db1",
        "arn:aws:glue:us-east-1:account-A-id:catalog"
      ]
    }
  ]
}
```




※アカウントAのデータベースdb1に任意のテーブルを作成可能なアクセス権。アカウントAのdevユーザーとアカウントBのルートユーザーに同じアクセスを許可する。

参考URL： リソースベースのアクセス許可(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/using-identity-based-policies.html)

# 暗号化

## データカタログ、暗号化されたデータソースとも連携可能

### データカタログ

- KMSキーを指定して、データベースやテーブルなどデータカタログ全体の暗号化が設定可能
- クローラーまたはジョブで定義するIAMロールにはKMSを操作できる設定(Decrypt、Encrypt、GenerateDataKey権限)が必要

### 接続パスワード

- データカタログがGetConnectionおよびGetConnectionsのAPIを実行時に取得される
- 接続の作成・更新された際、パスワードが暗号化されているかチェックし、暗号化されている場合は指定されたKMSキーがチェックされる

### セキュリティ設定

- Glueが使用できるセキュリティのプロパティ
- S3、CloudWatch Logs、ジョブのブックマークが暗号化の対象
- セキュリティ設定はETLジョブのパラメータとして渡されるS3サーバーサイド暗号化(SSE-S3)の設定をすべてオーバーライドする。ジョブにセキュリティ設定とSSE-S3の両方が設定されている場合、SSE-S3の方は無視されるので、注意

参考URL：AWS Glueの暗号化と安全なアクセス(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/encryption-glue-resources.html)

# モニタリング

## クローラー・ジョブステータス・ジョブの実行状況が確認可能

### クローラー・ジョブステータスのハンドリング

- クローラー、ETLジョブのステータス変更やETLジョブ実行中のステータスをトリガーにAmazon CloudWatch Eventsを実行可能
  例えば、ETLジョブ失敗時にAmazon SNSやAWS Lambdaに連携することが可能

### ジョブの実行状況確認

- ETLジョブのジョブ実行状況は管理コンソールまたはCloudWatch Logsで確認可能
- 暗号化されている場合は指定されたKMSキーによって判断される
- [Monitoring options] - [Job Metrics]オプションを有効にすることでジョブ監視とデバッグが可能

  ジョブ監視の例

  ・Sparkのメモリが不足していないかどうかドライバーとエグゼキューターのメモリ使用率の確認

  ・ジョブ実行時のDPU(データ処理単位)数のモニタリング

  ・複数ジョブの進行

参考URL：ジョブ監視とデバッグ(https://docs.aws.amazon.com/ja_jp/glue/latest/dg/monitor-profile-glue-job-cloudwatch-metrics.html)

# Continuous Logging New

## Spark ETLジョブの進捗状況をリアルタイムに追跡できる機能

- 5秒間隔・各executor終了前までログが出力される
- CloudWatch Logsにてアプリケーション固有メッセージ、プログレスバー(進捗状況)の表示が可能
- デフォルトのフィルタを使用することにより、ログの詳細度を調整できる

**アプリケーション固有メッセージのスクリプト例**

Python

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

Scala

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
  def main(sysArgs: Array[String]) {
    val logger = new GlueLogger
    logger.info("info message")
    logger.warn("warn message")
    logger.error("error message")
  }
}
```

参考URL：Continuous Logging (https://docs.aws.amazon.com/ja_jp/glue/latest/dg/monitor-continuous-logging.html)

# その他 前回(2017/10/18)からの主なアップデート New

**・Glueがインターフェイス型VPCエンドポイントを実装**

- PrivateLinkでGlueのAPIが利用可能に

**・クローラー、トリガー、ジョブ、開発エンドポイントにタグづけが可能に**

- タグを活用してコスト計算、IAMポリシーを利用してリソースへのアクセス制御を行うことが可能

**・データカタログで利用しているHiveメタストア用のソースコードがダウンロード可能に**

- Hiveメタストアと互換性のあるアプリケーションを構築する実装例として利用可能
  参考URL(https://github.com/awslabs/aws-glue-data-catalog-client-for-apache-hive-metastore)

**・既存のデータカタログのテーブルをソースとしてサポート**

- 既存のテーブルに対して、スキーマの変更を検出してテーブル定義を更新する

**・Apache SparkのETLジョブメトリクスが追加**

- コードのデバッグ、データ問題の特定、CPU容量計算に利用可能

**・ブックマーク機能のサポートファイル形式としてParquetとORCが追加**

- 既存のJSON・CSV・Avro・XMLに加えて、ParquetとORCが追加(Glueバージョン1.0以降)



aws

# ユースケース

# ユースケース

1. データカタログを用いたメタデータ管理
2. ジョブによるSQLの定期実行
3. WorkFlow機能を用いたETLパイプライン
4. サーバーレスアナリティクス
5. データレイクを用いたログ分析基盤
6. GlueとSageMakerを用いた機械学習基盤

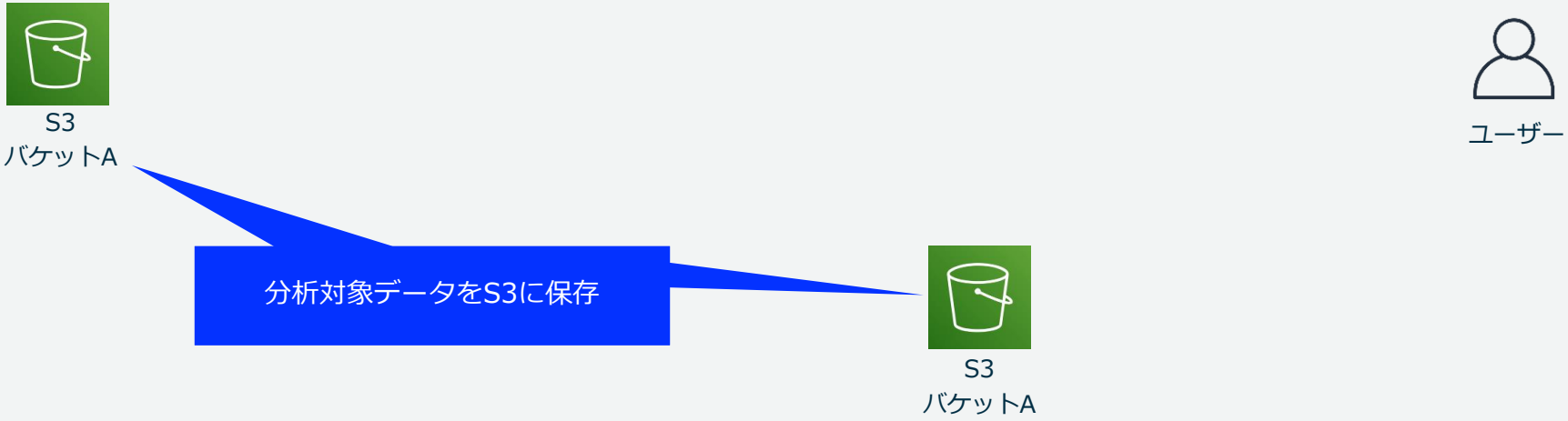
# 1.データカタログを用いたメタデータ管理

## EMR・Athena・Redshift Spectrumを利用する際のメタデータ管理に利用

- S3上にあるデータのメタデータ(スキーマ構造・パーティション等)をデータカタログに登録する
- Redshiftからクエリが実行できるように、Redshift Spectrumを利用する
- ユーザーがクエリを実行時、データカタログの情報を元にRedshift SpectrumがS3上にデータを取得する

# 1.データカタログを用いたメタデータ管理

## アーキテクチャ例



# 1.データカタログを用いたメタデータ管理

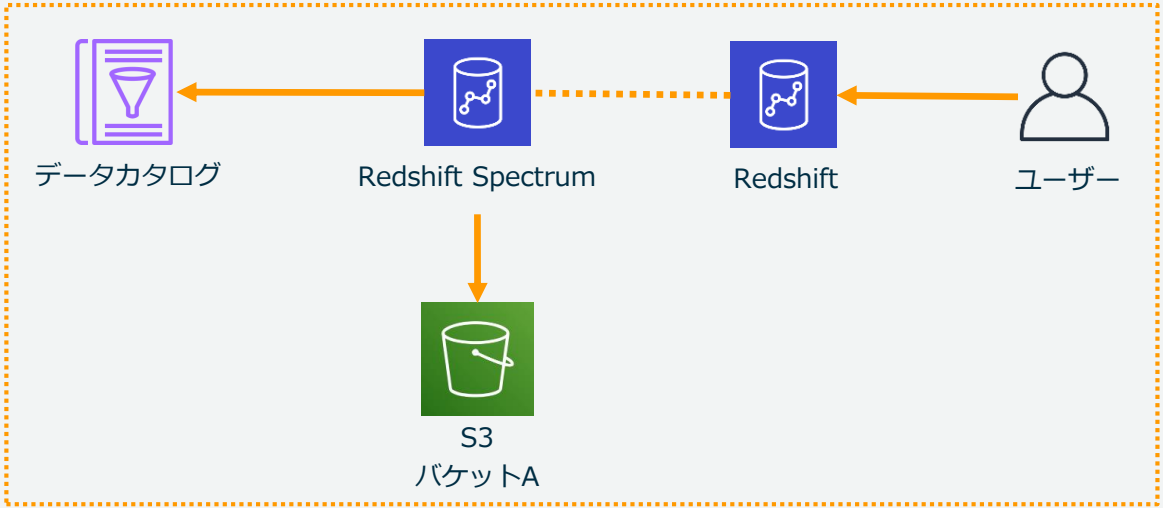
## アーキテクチャ例

# 1.データカタログを用いたメタデータ管理

## アーキテクチャ例

S3
バケットA

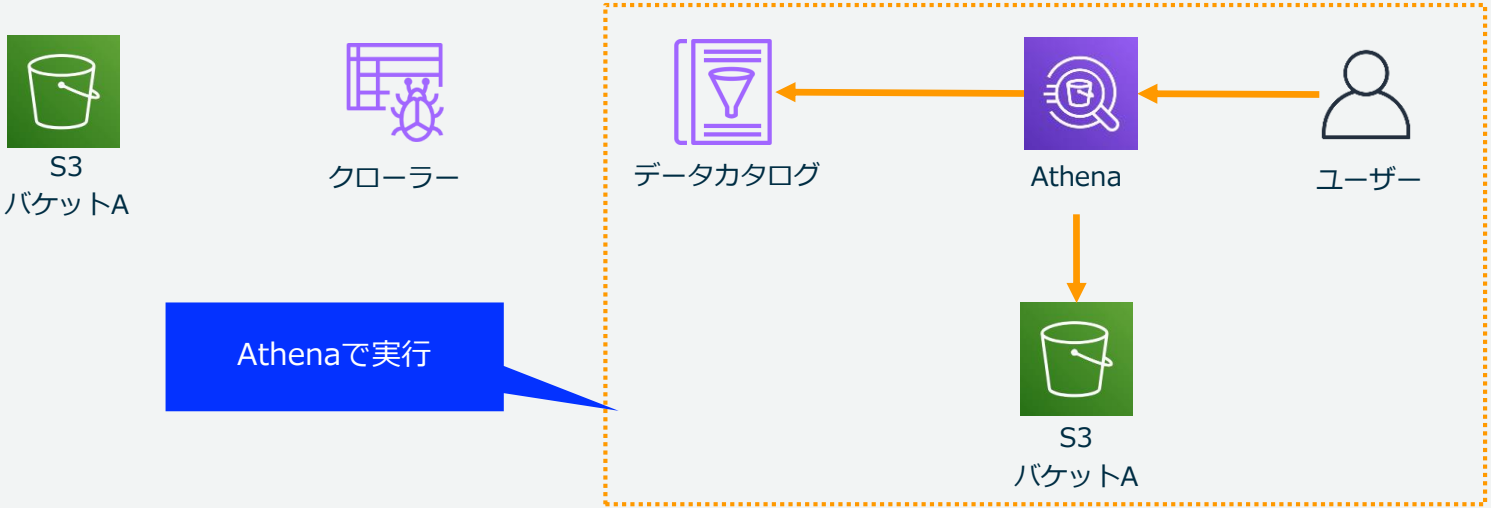
クローラー




Redshiftの外部テーブルとして
Redshift Spectrumを利用

# 1.データカタログを用いたメタデータ管理

## アーキテクチャ例



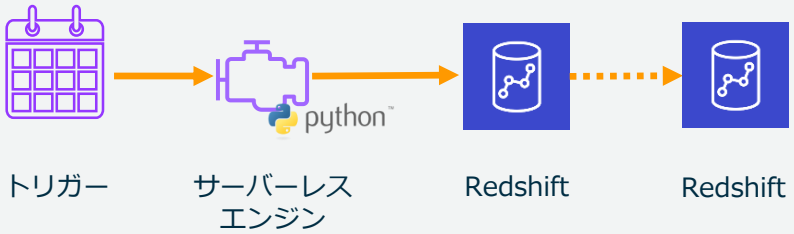
# 2.ジョブによるSQLの定期実行

## トリガー・Python Shellを用いてRedshiftに定期クエリを実行する

- 実行するPython Shellにタイムアウトを設定できる為、SQLを長時間実行し続けることやタイムアウト値を設定して途中で中断することが可能

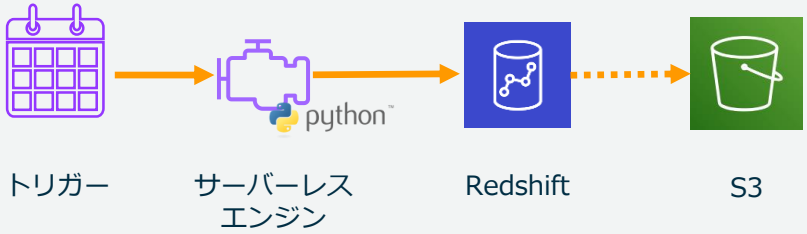
### アーキテクチャ例

①Redshiftのテーブルから別テーブルを作成する




・夜間バッチでデータマートを作成する場合に有効

②RedshiftのデータをS3にCSV出力する




・Redshift Spectrumの利用目的で古いデータを定期的に送信する
・S3を利用する他のサービスに連携する目的で送信する

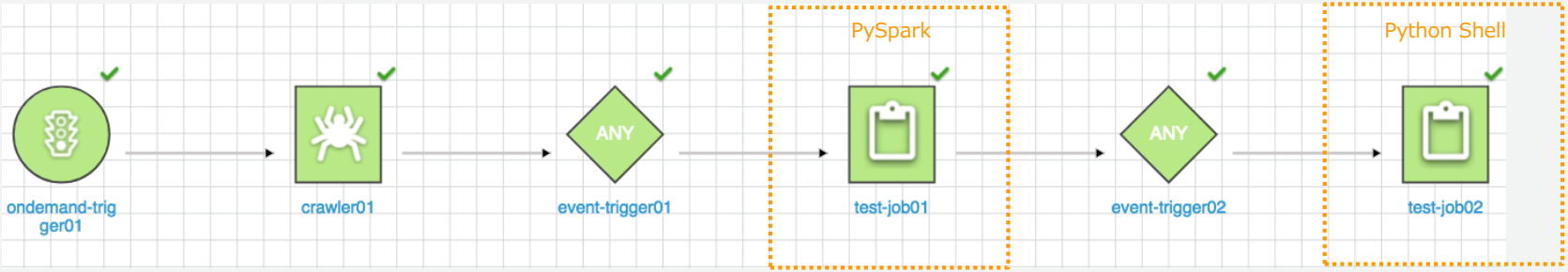
# 3.WorkFlow機能を用いたETLパイプライン

## WorkFlow機能を用いて単一ジョブではなく、複数ジョブを組み合わせて実行する

下記ジョブをGlueのWorkflow機能を使って、パイプライン処理として定義する

- S3上にあるデータをクロールし、データカタログに登録する
- PySparkでフォーマット変換・パーティション化をして、S3に出力するETL処理を実行する
- Python Shellでジョブの終了をSNSで通知する

### ワークフロー例



# 4.サーバーレスアナリティクス

## Glueの開発エンドポイント、SageMaker Notebookを用いて分析を行う

- 開発エンドポイント設定時にGlueのデータカタログを利用するように設定する
- SparkSQLを用いて、標準SQLでSageMaker Notebookから対話的にデータ分析が可能

### 分析例

```
%%sql
describe sample_database01.test_in
```

Type: Table Pie Scatter Line Area Bar

| col_name | data_type |
|---|---|
| device_id | bigint |
| location | string |
| platform | string |
| email | string |
| year | bigint |
| month | bigint |
| day | bigint |

```
%%sql
select location, count(platform) from sample_database01.test_in group by location
```

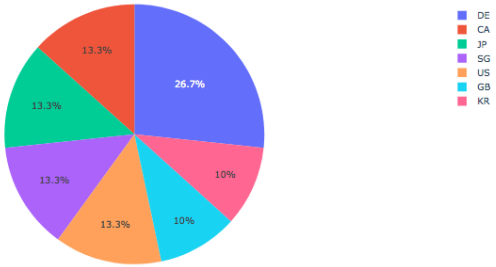
Type: Table Pie Scatter Line Area Bar

Encoding:

X location

Y count(platforr

Func. Max



# 5.データレイクを用いたログ分析基盤

## スピードレイヤ・バッチレイヤを活用したログ分析基盤

- ログデータをS3に保存し、Athena・Redshiftでクエリが実行できるようにGlueでETL処理を実施する

- Lambda Architecture(※1)に沿って、スピードレイヤとバッチレイヤを構築する

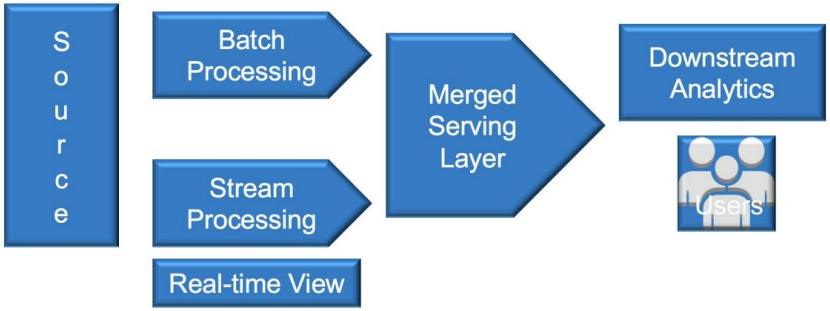



Figure 1: Lambda Architecture

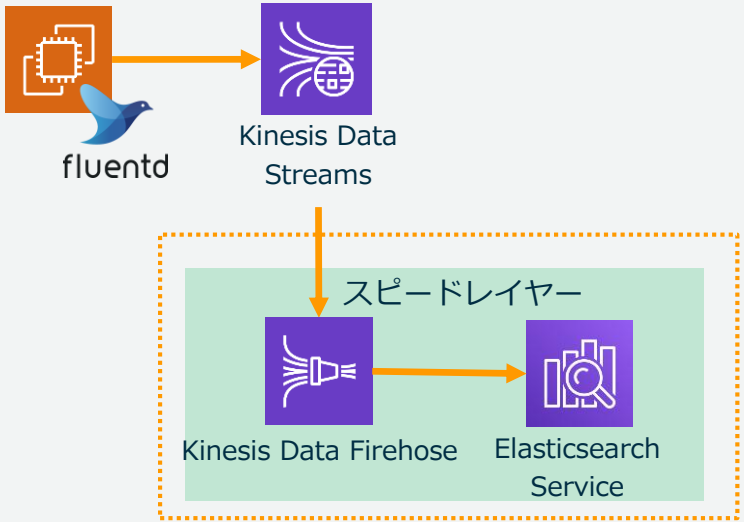
(※1)Lambda Architecture：参考URL(https://d1.awsstatic.com/whitepapers/lambda-architecure-on-for-batch-aws.pdf)。AWS Lambdaとは異なる。

# 5.データレイクを用いたログ分析基盤

## アーキテクチャ例

# 5.データレイクを用いたログ分析基盤

## アーキテクチャ例

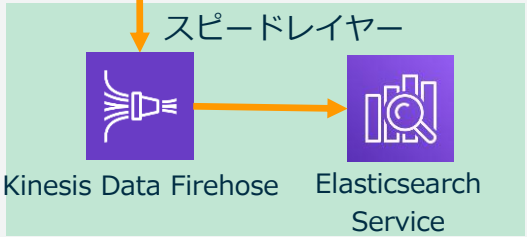


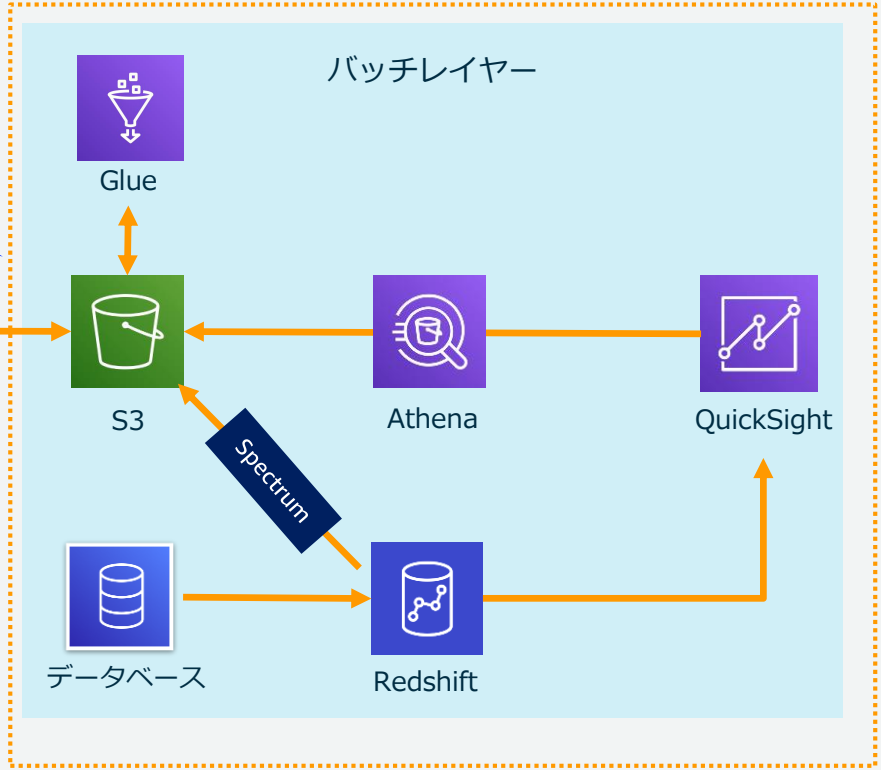

リアルタイム性の高いデータを可視化する

# 5.データレイクを用いたログ分析基盤

S3に溜まったデータをGlueのデータカタログ、サーバーレスエンジンを利用して、ETL処理を実行する。
実行処理した結果に対してクエリを実行し、可視化する



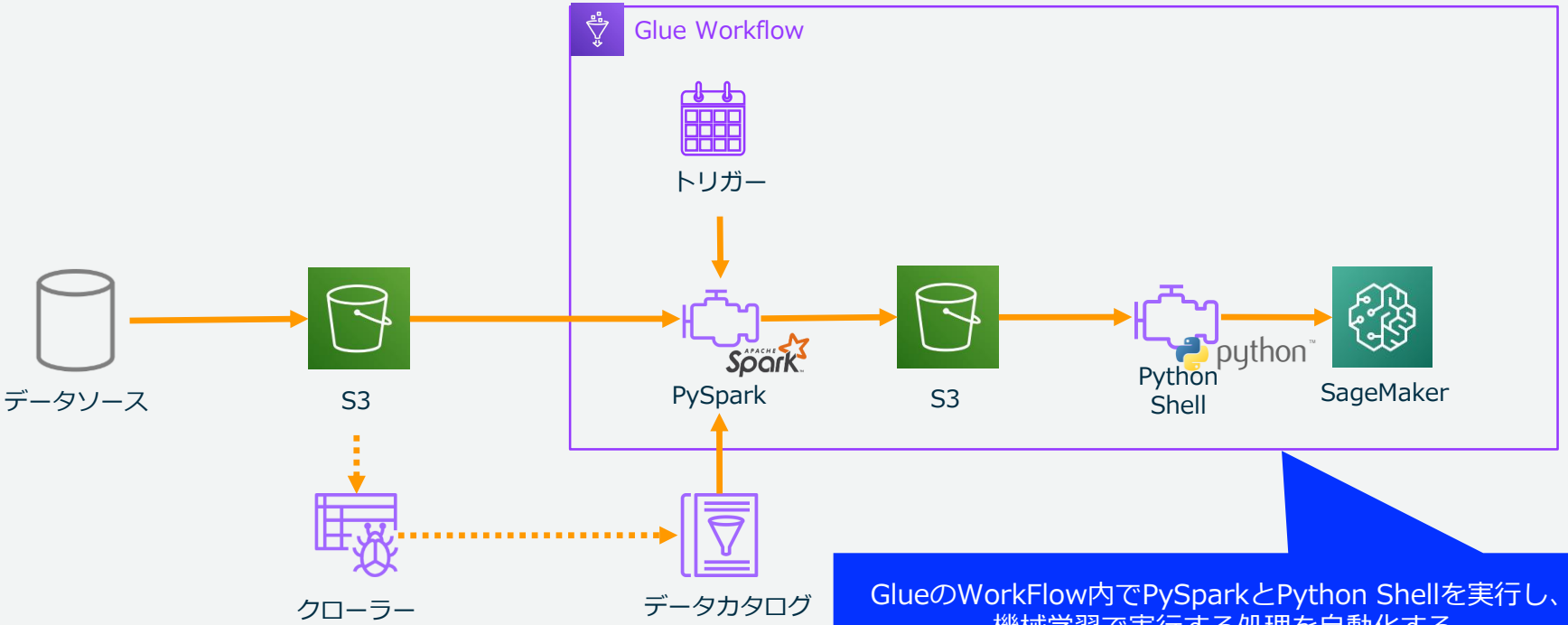
# 6.GlueとSageMakerを用いた機械学習基盤

## Glue WorkFlowを利用したETL・機械学習のワークフロー

- Glueにて学習用入力データを作成し、学習ジョブの実行とモデルのデプロイをSageMakerで実行する
- ワークフロー自体はGlueのWorkFlow機能、またはStep Functionsを利用する
  Glueのワークフローで実行する場合はPySparkとPython Shellを組み合わせる
  Step Functionsを利用する場合は、ワークフローの内容をJSONで定義する
  コードの実装、サービスのインテグレーション内容をもとに選択する

# 6.GlueとSageMakerを用いた機械学習基盤

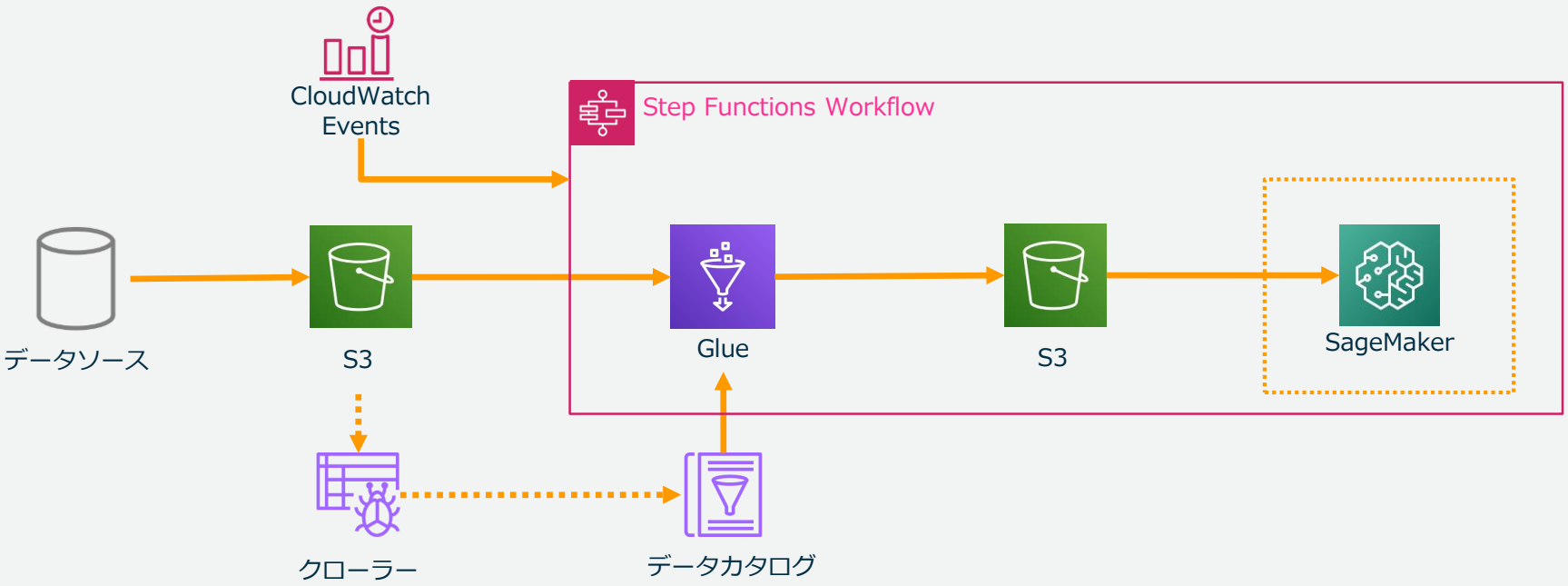
## アーキテクチャ例




GlueのWorkFlow内でPySparkとPython Shellを実行し、
機械学習で実行する処理を自動化する

# 6.GlueとSageMakerを用いた機械学習基盤

## アーキテクチャ例



# 料金

# 料金

## 1DPU(Data Processing Unit) = 4vCPU、16GB

### ・ETLジョブ

**Apache Spark** : $0.44 DPU/時 (課金は秒単位)
- 10分間未満の処理は10分としてカウントされる
- 2個以上のDPUが必要で、デフォルトでは各Sparkジョブに10個のDPUが割り当てられる

**Python Shell** : 1DPU : $0.44 DPU/時 (課金は秒単位)または1/16DPU : $0.0275 DPU/時(課金は秒単位)
- 1分間未満の処理は1分としてカウントされる
- 1個または1/16DPU個が選択可能で、デフォルトでは各Python Shell単位で1/16個のDPUが割り当てられる

### ・開発エンドポイント

エンドポイント作成から料金が発生。DPU単位で費用がかかる
2個以上のDPUが必要で、デフォルトでは5個のDPUが割り当てられる
Zeppenlin Serverの場合は通常のEC2の料金、SageMaker NotebookはNotebookインスタンスの料金が発生する

# 料金

## ・データカタログ

**ストレージ**：100万オブジェクトまで無料
(オブジェクト=テーブル、テーブルバージョン、パーティション、データベース)
100万以上保存された場合、10万オブジェクトあたり$1/月

**リクエスト**：100万リクエスト/月まで無料
100万以上保存された場合、10万オブジェクトあたり$1/月

## ・クローラー

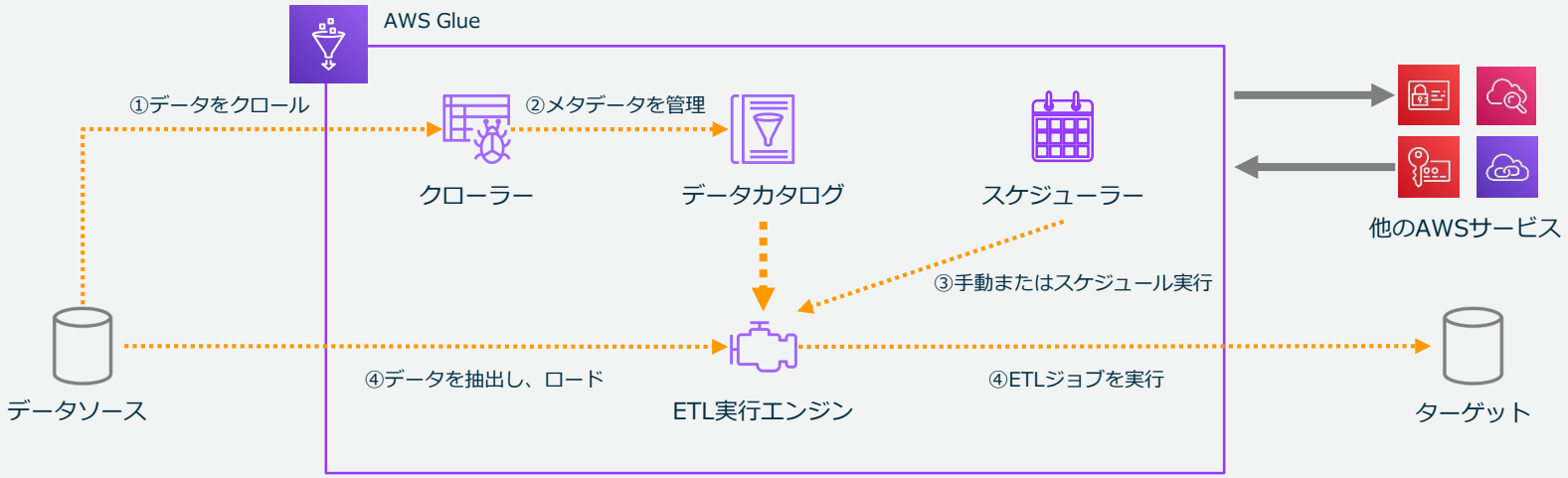
$0.44 DPU/時 (課金は秒単位)
- クローラーの実行に使用されたDPUの数に応じて時間あたりの課金が発生
- クローラーごとに10分の最小期間が設定される。10分未満の処理は10分として計算される

## ・その他

データ転送量やS3の保存データについては別途料金がかかる

# まとめ

# まとめ




- Glueはサーバーレスの ETLサービス
- クローラー・データカタログでメタデータを管理
- EMR／Athena／Redshift、SageMakerなど他のサービスとセキュアに連携

# 参考資料

AWS Glue ホームページ
https://aws.amazon.com/jp/glue/

AWS Glue 開発者ガイド（公式ドキュメント）
https://aws.amazon.com/jp/documentation/glue/

AWS Glue ETL Code Samples (サンプルコード)
https://github.com/awslabs/aws-glue-samples

AWS Glue用のPythonパッケージ（awsglue.*のソースコード）
https://github.com/awslabs/aws-glue-libs/

AWS Glueの料金
https://aws.amazon.com/jp/glue/pricing/

AWS Glueのサービス制限
https://docs.aws.amazon.com/ja_jp/general/latest/gr/aws_service_limits.html#limits_glue

# Q&A

お答えできなかったご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

# AWS の日本語資料の場所「AWS 資料」で検索


https://amzn.to/JPArchive

# AWS Well-Architected 個別技術相談会

## 毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

# ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive