
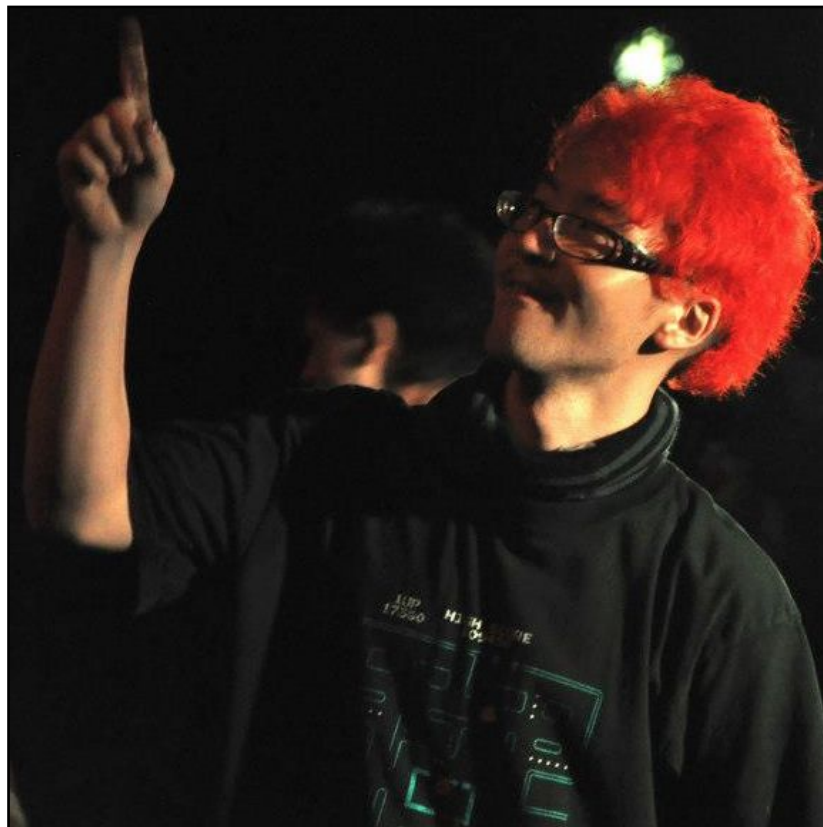


# Gunosy

## Digdag と Embulk と Athena で作る Gunosy の ELT基盤



株式会社 Gunosy  
Gunosy Tech Lab Data Reliability & MLOps Group  
中山貴博  
2019年7月31日



- 中山貴博 (@Civitaspo)
- Gunosy Tech Lab  
Data Reliability & MLOps Group  
Manager
- 経歴
  - DeNA -> Gunosy (2017/10 ~)
  - Hadoop の運用や ETL全般
- Embulk/Digdag などの古橋ウェアが大好き
  - Digdag Plugin公開数7個(総合1位)
  - Embulk Plugin公開数13個(総合3位)

ギリシャ語で「知識」を意味する「Gnosis(グノーシス)」+「u(“you”)」

「**”Gnosis” for “you”**」あなたのための知識

＝情報を届けるサービスを提供し続ける、という意味



- 2012年11月創業
- 2015年4月東証マザーズ上場
- 2017年12月東証第一部に市場変更
- 従業員数 215名  
(2019年5月末現在 連結ベース)
- 事業内容
  - － 情報キュレーションサービスその他メディアの開発及び運営
- 提供サービス  
グノシー、ニュースパス、LUCRA(ルクラ)

企業理念「情報を世界中の人に最適に届ける」

# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

はじめに

## Gunosy における ETL 基盤の目的と変遷、そして現在の ETL 基盤のお手軽さ

- Gunosy で ETL 基盤を構築するに至った経緯
- いかに利用者/運用者双方のコストを下げてきたか
- 現在の ETL 基盤のお手軽さ
  - AWS 上で Digdag/Embulk を運用してるなら明日から導入できるレベルの導入コストの低さ
  - 複雑な処理は Plugin が解決してくれる
  - 主な処理を Athena に任せる運用レスな世界

- 資料は SpeakerDeck に公開済みです
  - <https://speakerdeck.com/civitaspo/>
- タイトルも構築した基盤もELT基盤なのですがETLかELTの違いが発表の重要なポイントというわけではないのでETLに表記統一しています
- Digdag/Embulk/Athena の話を中心に行うので、その他の部分は意図的に薄くしか書いていません。気になる点があれば懇親会などでお話させてください。
- この発表で説明する ETL 基盤で使われている Plugin は全て公開していますので興味を持っていただけたら利用してフィードバックいただけると嬉しそうです
  - [https://github.com/civitaspo/embulk-output-s3\\_parquet](https://github.com/civitaspo/embulk-output-s3_parquet)
  - <https://github.com/civitaspo/digdag-operator-athena>

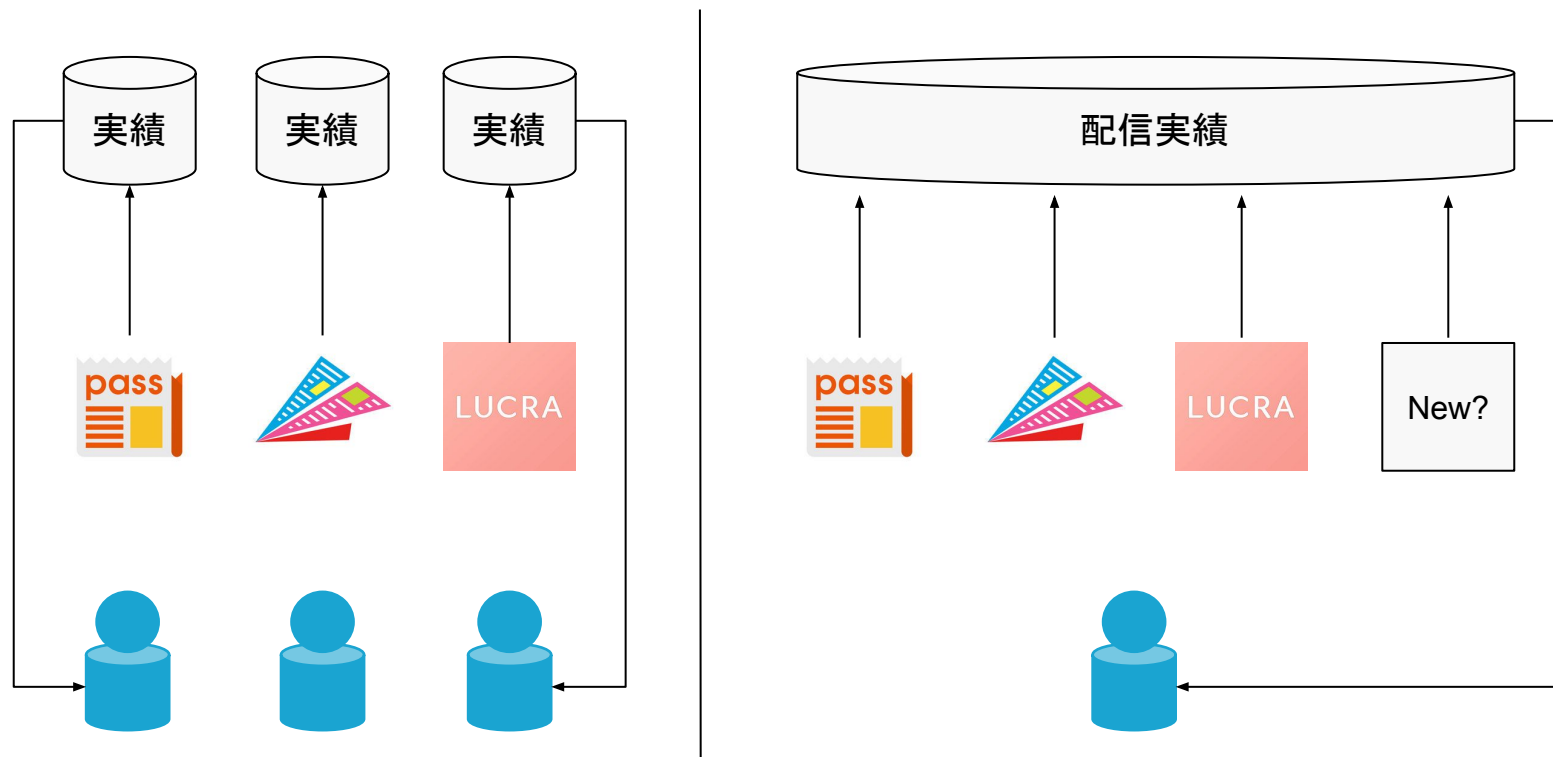


# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

# Gunosy の ETL 基盤事情

- 歴史的な事情からアプリ個別にデータ基盤・ETL基盤を持っていた
- 新アプリ毎に都度ロジック開発する負荷やコールドスタートを回避したい
- 都度チームを作って負荷が高まることやサイロ化を防ぎたい



Gunosy 横断のデータ基盤 / ETL 基盤の必要性

- DRE = Data Reliability Engineering
- Gunosy のデータを集中管理し信頼性を担保する組織
  - Gunosy Way “数字は神より正しい”
  - この“**数字**”の信頼性を担保する
- 具体的には
  - データ基盤・ETL 基盤の統合
  - 個別サービスでは必要のなかったデータの収集・生成
  - サービス間のデータ組み合わせによるデータ資産価値向上
  - 各サービスへのデータ配信
  - ...

今日の発表ではこのETL基盤部分について取り扱います



- 安定していること
- データデリバリーが高速であること
- 複数 AWS Account からのアクセス可能であること
- 様々なデータソース・フォーマットを扱えること
- データを必要する人が主体的に ETL Workflow を構築できること
  - “DRE だけがデータを作れる” とするとスケールしない
  - データを必要とする人 = 主にデータサイエンティスト
    - SQL がめっちゃめっちゃ書ける
  - DRE がレビューできること
- ...

- ★ ETL Workflow は UI で作るよりもコード管理できることが重要
- ★ SQL を中心とした ETL 開発ができると開発速度向上が見込める
- ★ 入出力は様々なデータソース・フォーマットへ対応可能である必要がある

# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

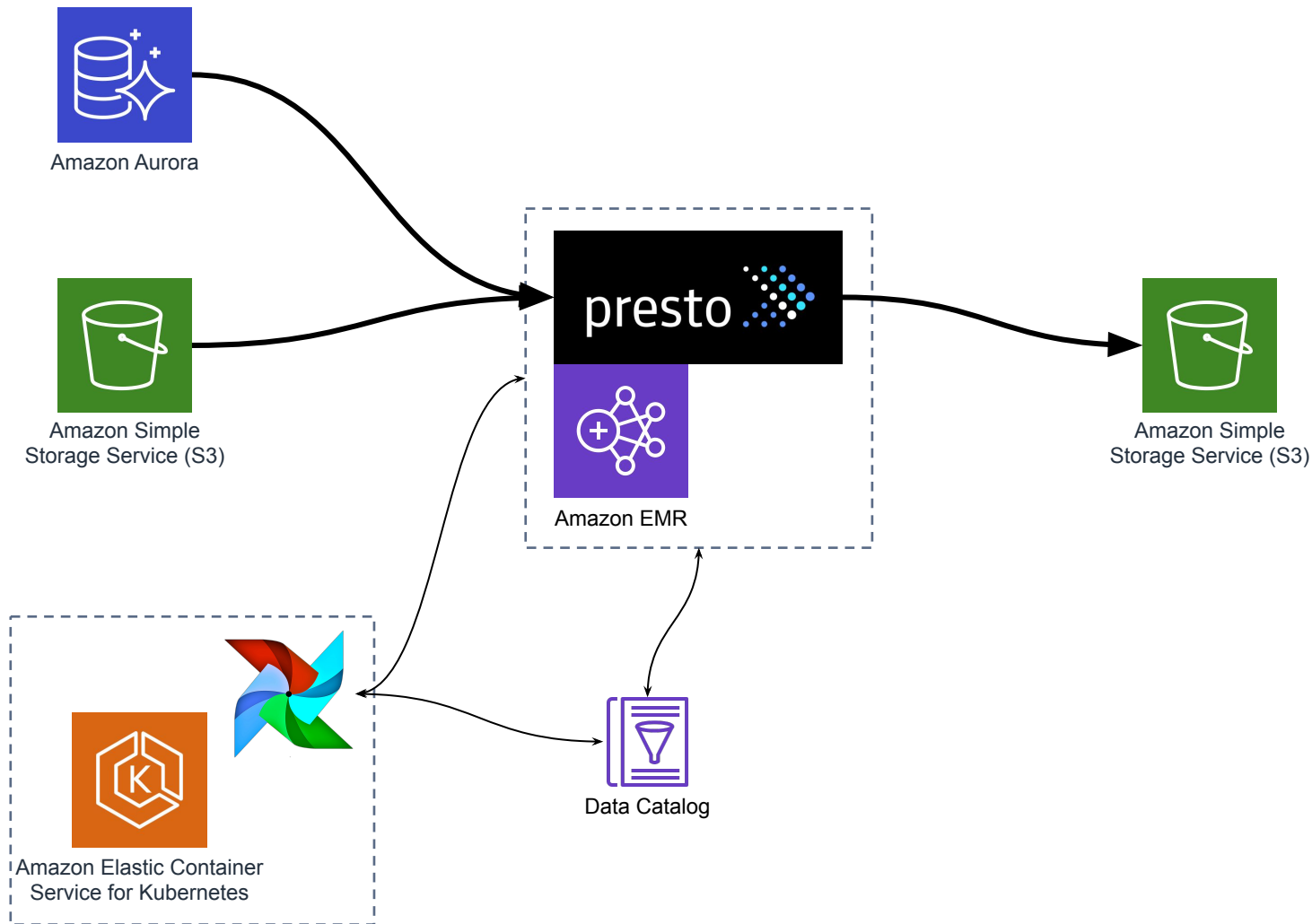
## 構築したETL基盤の変遷

- これからお話する ETL 基盤では様々なデータソースからデータを抽出し、S3 へデータを格納した後 Athena のテーブルとして利用できる状態にするところまでにしか触れません。
  - データ活用部分にまで触れちゃうと時間ガガガ...
  - すべてのデータを安定的に S3 に格納するまでの ETL 基盤をどう構築したかという観点でご覧になってください 
- あと Airflow を少し悪く書きますが、Airflow の運用に知見のある人と構築していたら違う結果になったかもしれません。単なる1事例としてご覧になってください 



v1 ETL 基盤

# ETL基盤の変遷: v1



Airflow と Presto を採用し、SQL のみで ETL できる基盤を構築した

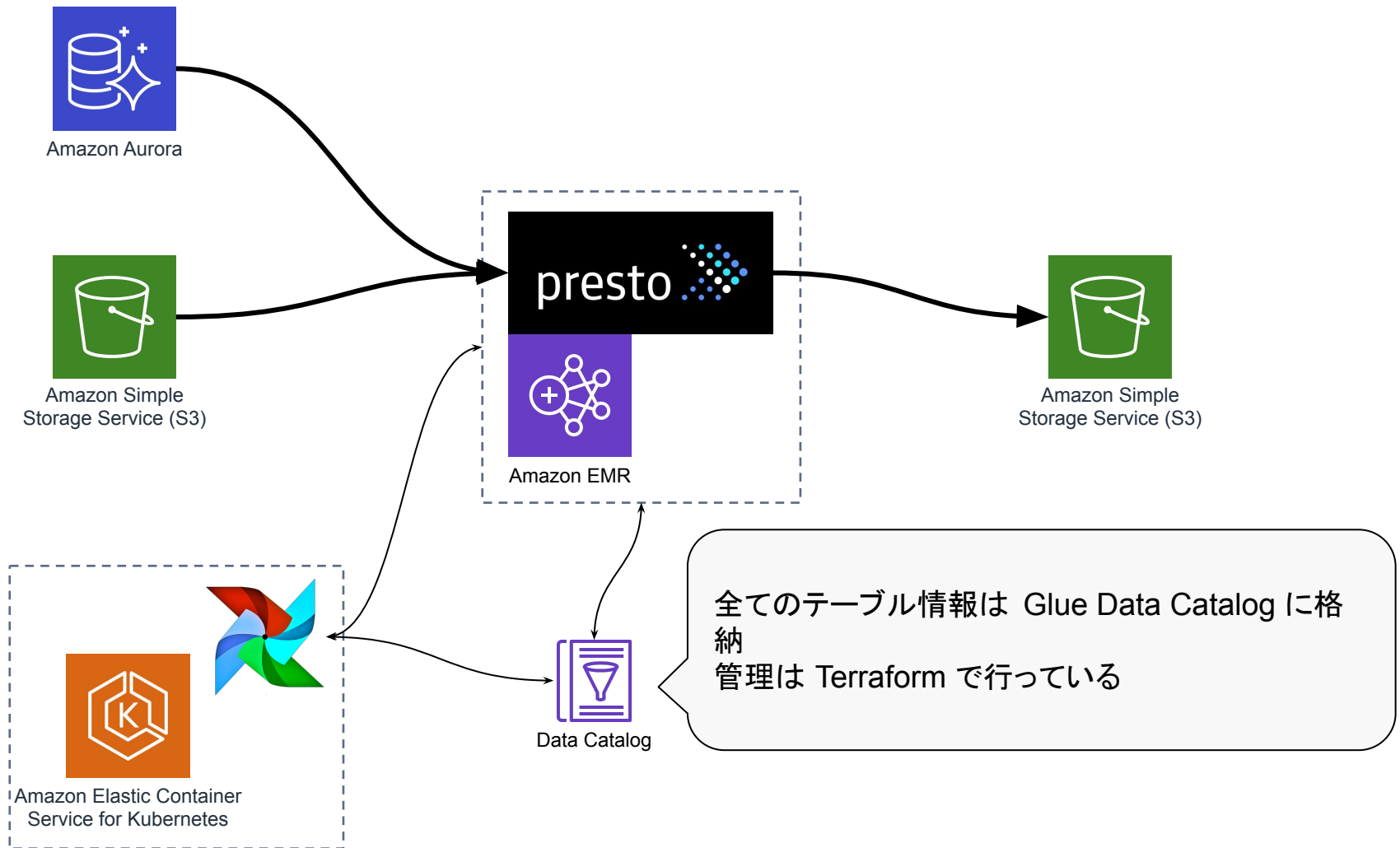
## Airflow の選定理由

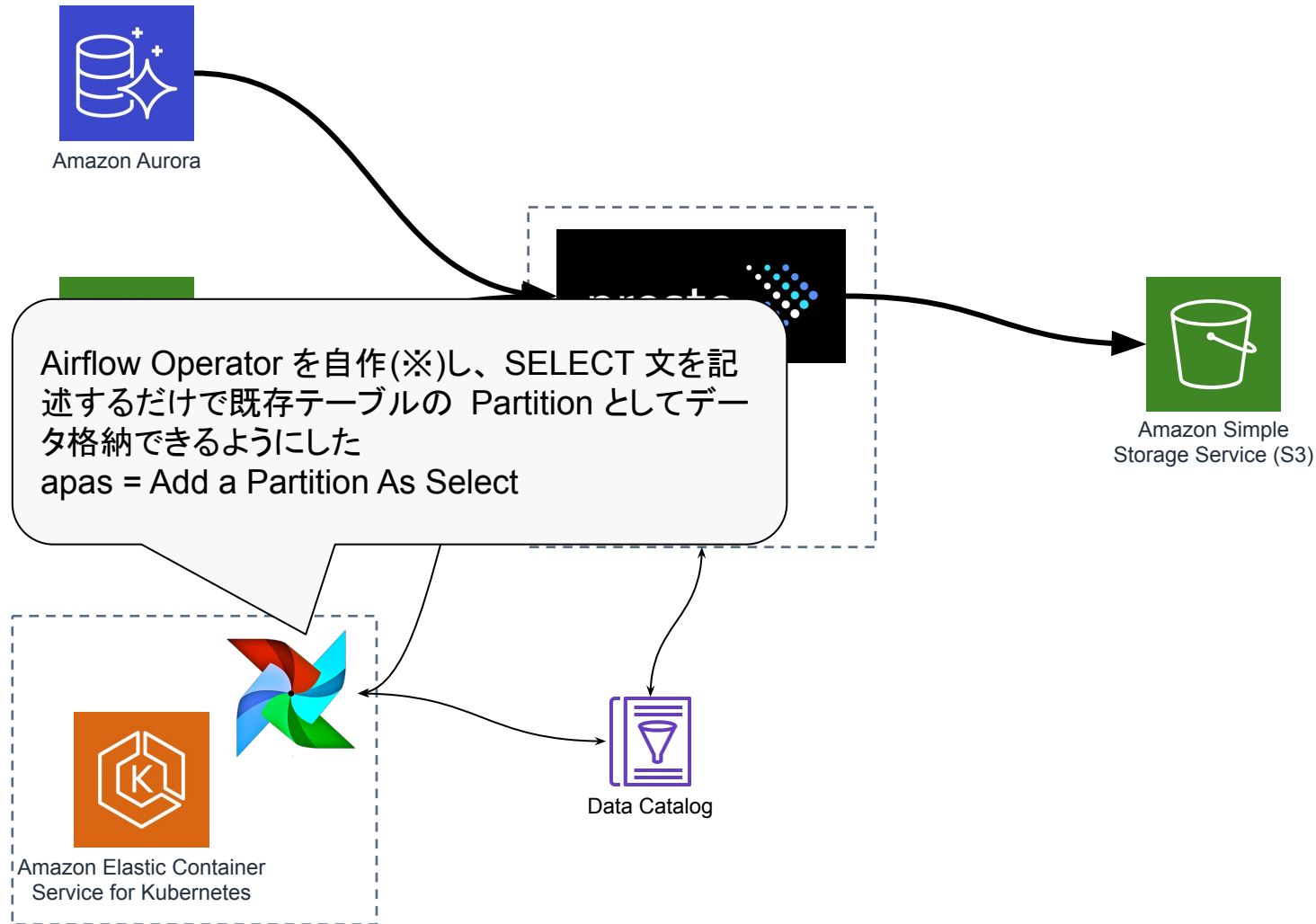
- Gunosy で利用される主要言語の1つである Python で Workflow を記述できる
- 開発コミュニティが大きい
- GCP や AWS で Managed Service として提供されるなど大企業での採用事例が多いため、スケーラビリティも高く、これからのスタンダードになっていくだろうことが見込めた

## Presto の選定理由

- 分散SQL実行エンジン
- データ処理で主にメモリを利用するため高速
- スケーラビリティが高い
- Presto は様々なデータソースへ接続可能な Connector が存在
  - 必要であれば Connector の自作もできる

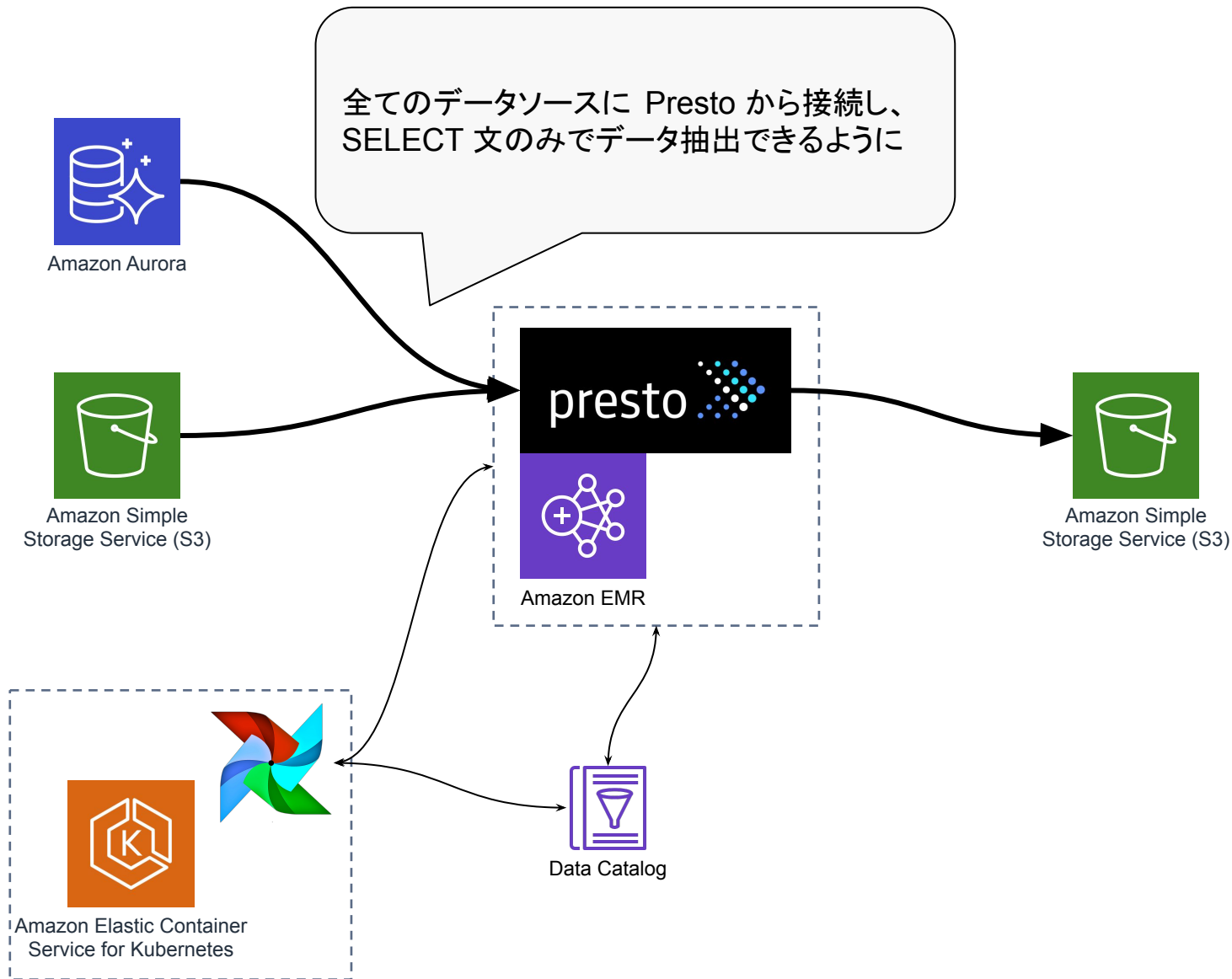
# ETL基盤の変遷: v1





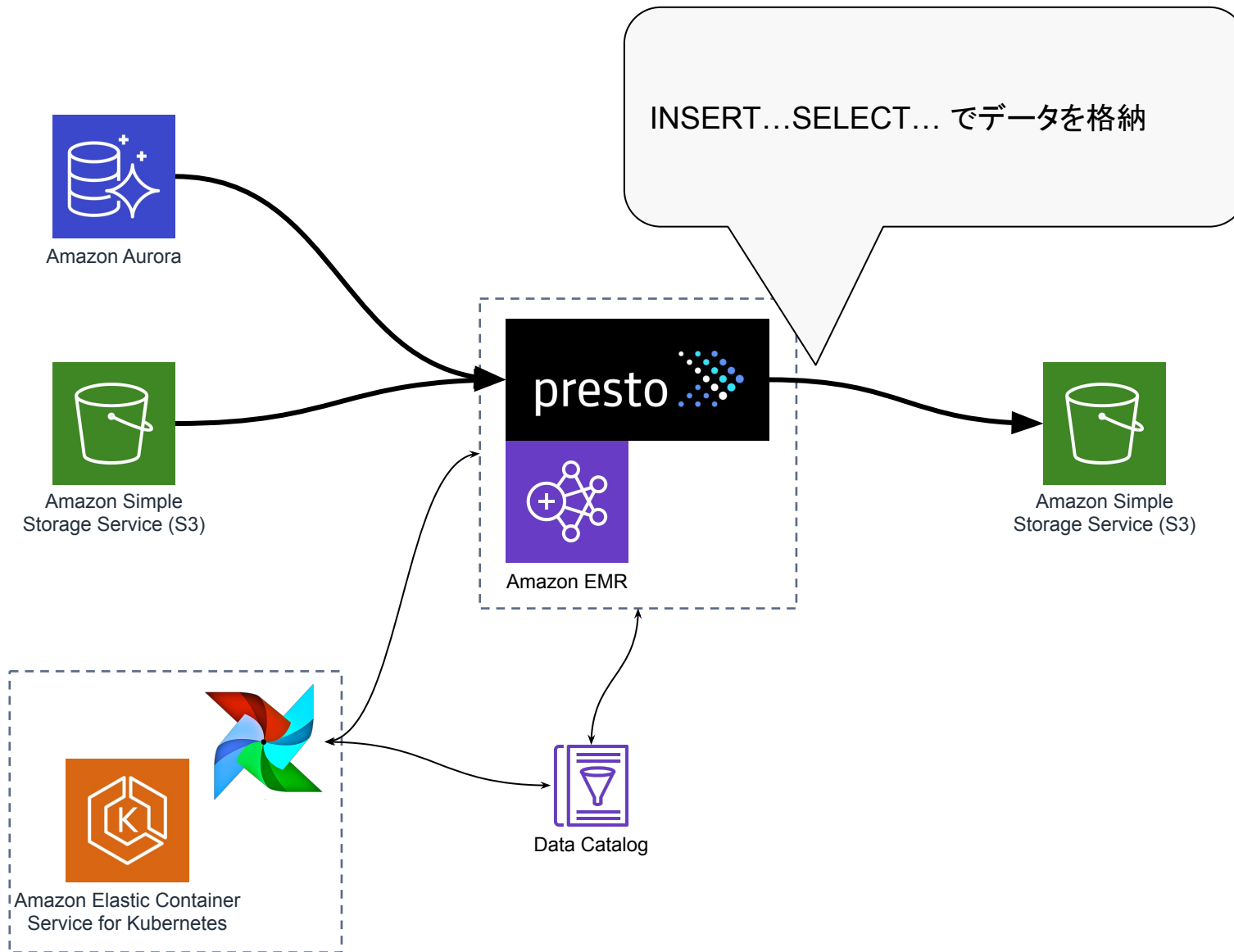
※ [https://github.com/civitaspo/airflow-plugin-glue\\_presto\\_apas](https://github.com/civitaspo/airflow-plugin-glue_presto_apas)

# ETL基盤の変遷: v1



# ETL基盤の変遷: v1

Gunosy



## v1 ETL 基盤の限界



Airflow を安定運用するコストが高い

## Poison Pill 問題

- <https://issues.apache.org/jira/browse/AIRFLOW-4514>
- Taking the poison pill. とログに出たとき、Airflow 上では Task が成功しているが実際には何も実行されていない状態になる

## Scheduler Stuck 問題

- <https://issues.apache.org/jira/browse/AIRFLOW-401>
- 大量に再実行を走らせたときに発生
- Scheduler が全く Task を Schedule しなくなる

Airflow を利用してもらうコストが高い

execution\_date の仕様が直感的ではない問題

- execution\_date は実際に schedule される時間の1つ前の時間が格納されている
  - “0 1,13 \* \* \*” という schedule だと 13時に実行される Task の execution\_date には 1時が入っている
- レビューでひたすら指摘する必要がある...

Python で自由度高く Workflow を記述できる反面、全体像を把握しにくい

- UI で rendering されないと Dag が理解できない
  - 複数の branching や join があると読むのがかなり厳しくなる
  - `op1 >> op2 >> op3`, `op2 >> op4 >> op5`, `op4 >> op3`, ...

基盤の安定性・利用者の開発容易性の観点から  
Airflow を運用しつづけるのは妥当ではないと判断

Presto にも少し辛みが...

INSERT...SELECT...のエラーハンドリング難しい問題

- 公式が提供する presto-python-client は INSERT...SELECT...の結果を SELECT が成功した時点で返すため、実際に S3 にデータが格納されたことを保証しない
  - <https://github.com/prestodb/presto-python-client/issues/58>
- S3 上の Object 生成を確認したり SELECT COUNT(1) などでデータ件数の確認をする必要がある

EMR 費用が高い問題

- Spot Fleet を使えばある程度 Instance 代は抑えられるが...
- Spot Fleet を利用すると代わりに Auto Scaling が出来ないので自前で実装する必要がある

Presto にも少し辛みが...

EMR で複数の Connector を扱うのが難しい

- EMR で追加できる Connector は 1種類につき1つのみ
  - 複数の MySQL Connector を使いたい場合は自前の bootstrap action を書く必要がある
  - MySQL を読みたい利用者と EMR の運用者の作業が密結合に

MySQL 上で enum として定義したカラムを CAST AS VARCHAR すると全ての値が enum の最大長までスペースで埋められる

- a, bbb の2択だとすると取得結果は “a ”, “bbb” となる
- SELECT 文で TRIM を入れてもらうなどデータソースごとに挙動の違いを意識する必要がある

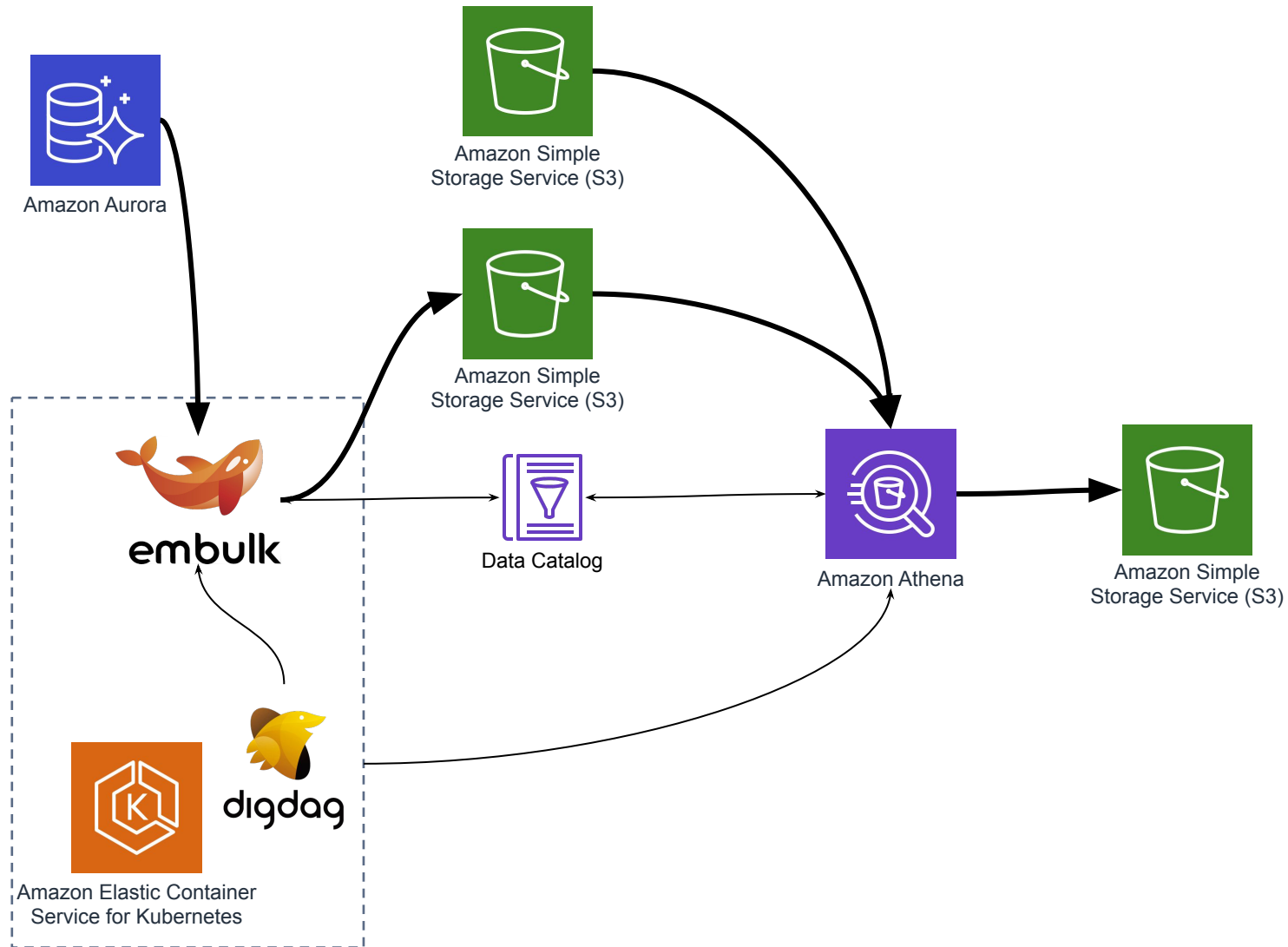
Presto にも運用コスト・利用者の開発容易性に課題が出てきた

Airflow/Presto で ETL 基盤を構築したが改善すべき点が多く出てきた

- Airflow は基盤の安定性・利用者の開発容易性の観点から別の Workflow Engine に変える判断になった
- Presto も運用コスト・利用者の開発容易性の面で課題解決が必要になった

v2 ETL 基盤

# ETL基盤の変遷: v2



## 新 Workflow Engine に Digdag を採用



- Treasure Data が開発する Workflow Engine
- YAML で Workflow を記述する
  - プログラミング言語で Workflow を記述できないので多少冗長になる一方、Workflow から生成される処理量が比較的予測しやすい
- Plugin 機構や ScriptingOperators による拡張性が高い
- 社内で大規模利用実績があり安定運用のノウハウがある



<https://tech.gunosy.io/entry/introduce-digdag-plugins>



Presto でクエリを実行するデータソースを統一するため  
S3 以外のデータソースは Embulk で一度 S3 に格納する方針に



- Treasure Data が開発する Bulkload Tool
- Plugin 機構により様々なデータソース・フォーマットのデータを Load することができる
  - 既に100以上の Plugin が存在するため大抵のケースでは Plugin 開発無しにデータの Load が可能
- 前職で大規模利用の経験があり安定運用のノウハウがある

Embulkに足りない

5つのこと

2015-12-15 Embulk Meetup Tokyo #2  
@Civitaspo

<https://speakerdeck.com/civitaspo/embulknizu-rinai5tufalsekoto>

データソースを S3 のみとしたので Presto は Amazon Athena に置き換え



- Amazon の Managed Query Service
  - Hive Connector で S3 に対してのみアクセスできる Managed Presto だと思って良い
- データスキャン量によってのみ課金される
  - \$5 / 1TB
  - CREATE TABLE などの DDLクエリは無料
  - 正常に完了しなかったクエリは無料
- 同時実行数は最大50まで上げられる
  - 50以上は AWS と相談が必要
- 2018-10-11 から CTAS を Support し、データ生成にも使えるようになった
  - v2 ETL 基盤の肝となる機能

## Digdag/Embulk/Athena で ETL 基盤を再構築した

- Workflow Engine は運用ノウハウのある Digdag となり安定した
- Dag の記述では実行開始時間と同値の session\_time を使えるようになり、また Digdag の DSL で記述するようになり、運用者/利用者ともに直感的に記述できるようになった(少なくとも僕たちにとっては)
- Presto on EMR を Athena という Full-Managed Service に変更したため EMR の運用コスト(金銭的/人的)が大幅に下がった
- S3 以外のストレージにあるデータは Embulk で事前に S3 に格納するため利用者はストレージの種別を意識しながら SQL を書かなくて良くなった

# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

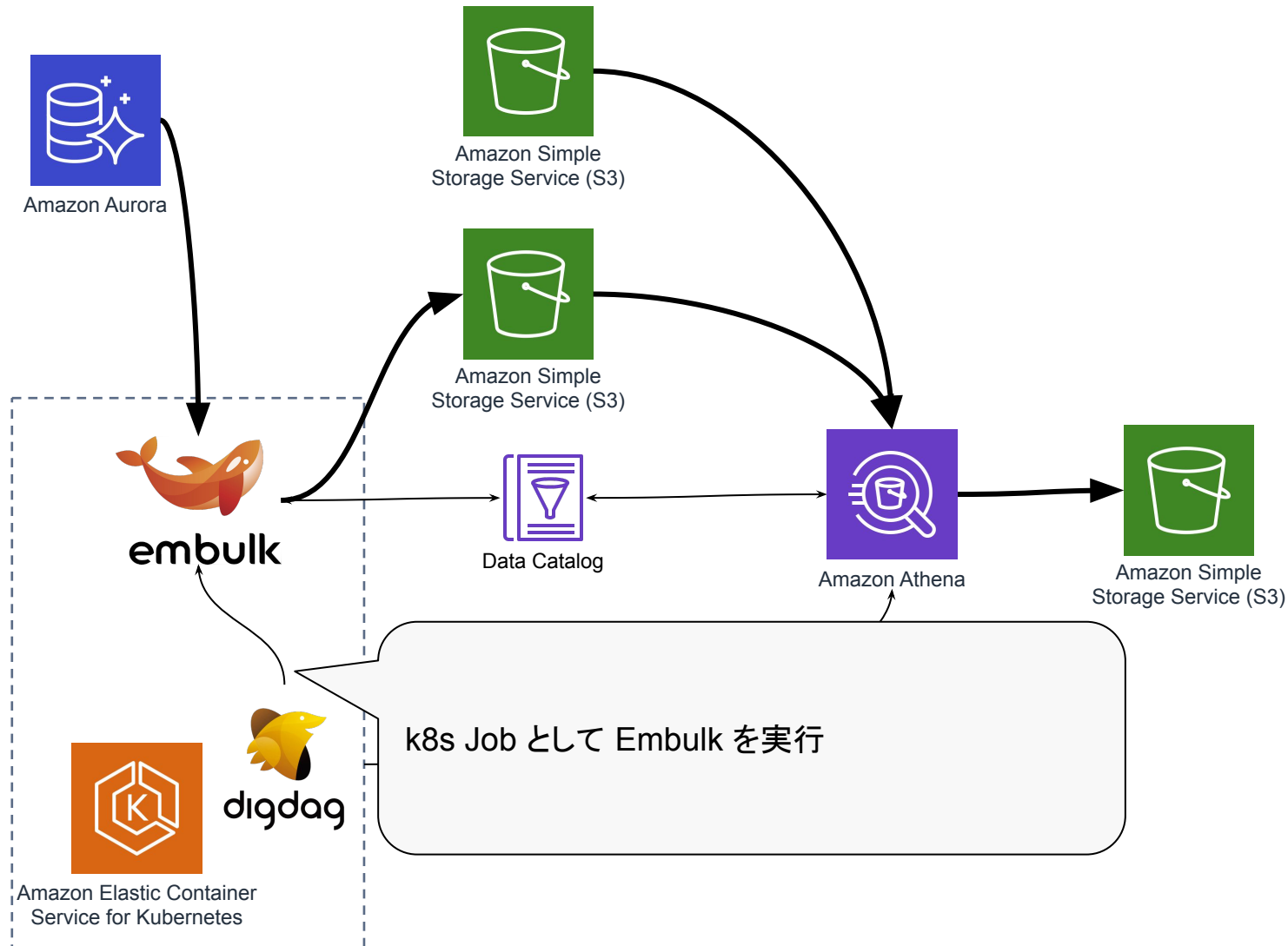
# Digdag/Embulk/Athenaで構築するお手軽ETL基盤

このパートでは構築した v2 ETL 基盤の詳細とお手軽ポイントについて話します

- データ格納までの流れ
- embulk-output-s3\_parquet
- digdag-operator-athena

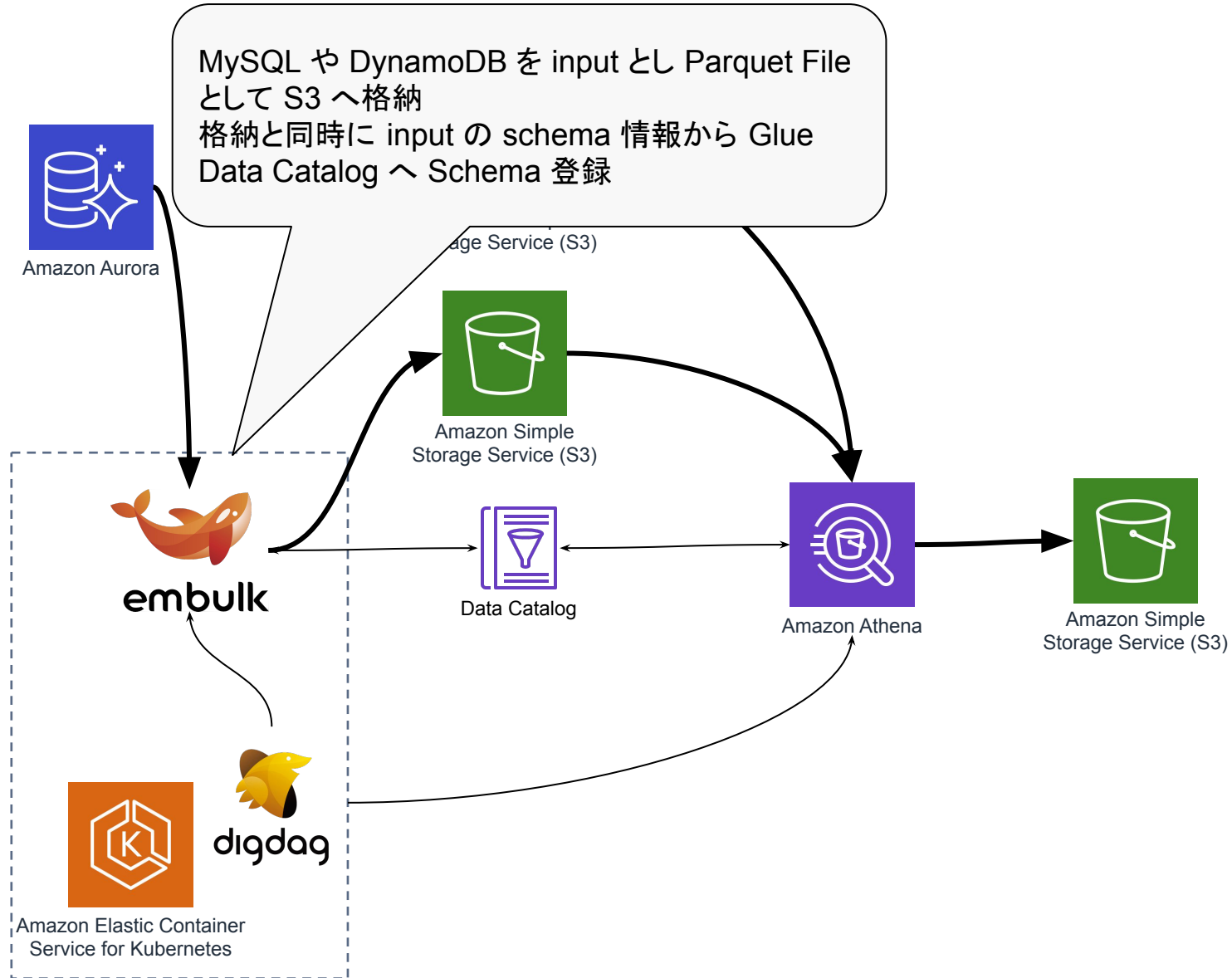
## データ格納までの流れ

# データ格納までの流れ

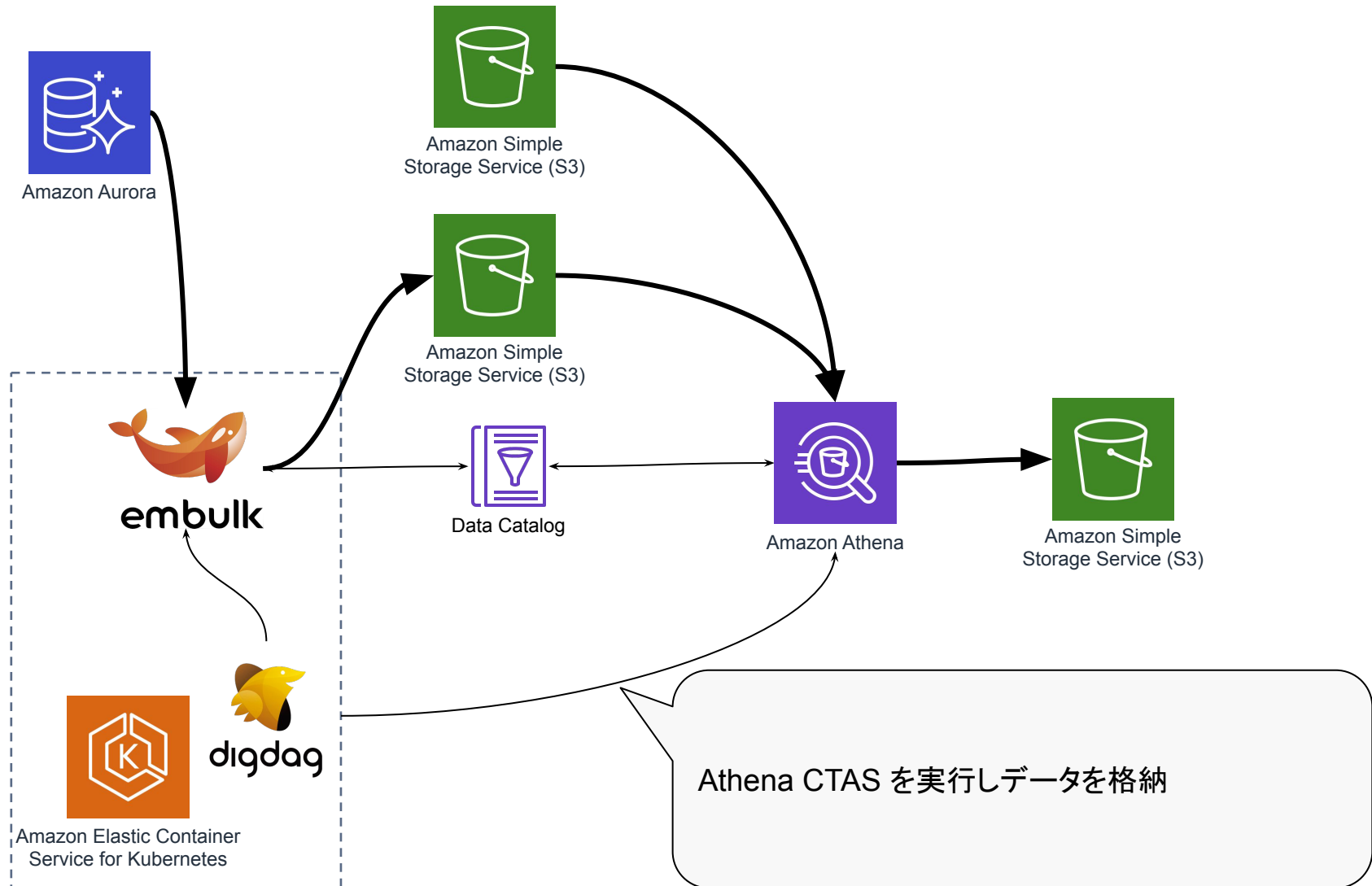




# データ格納までの流れ



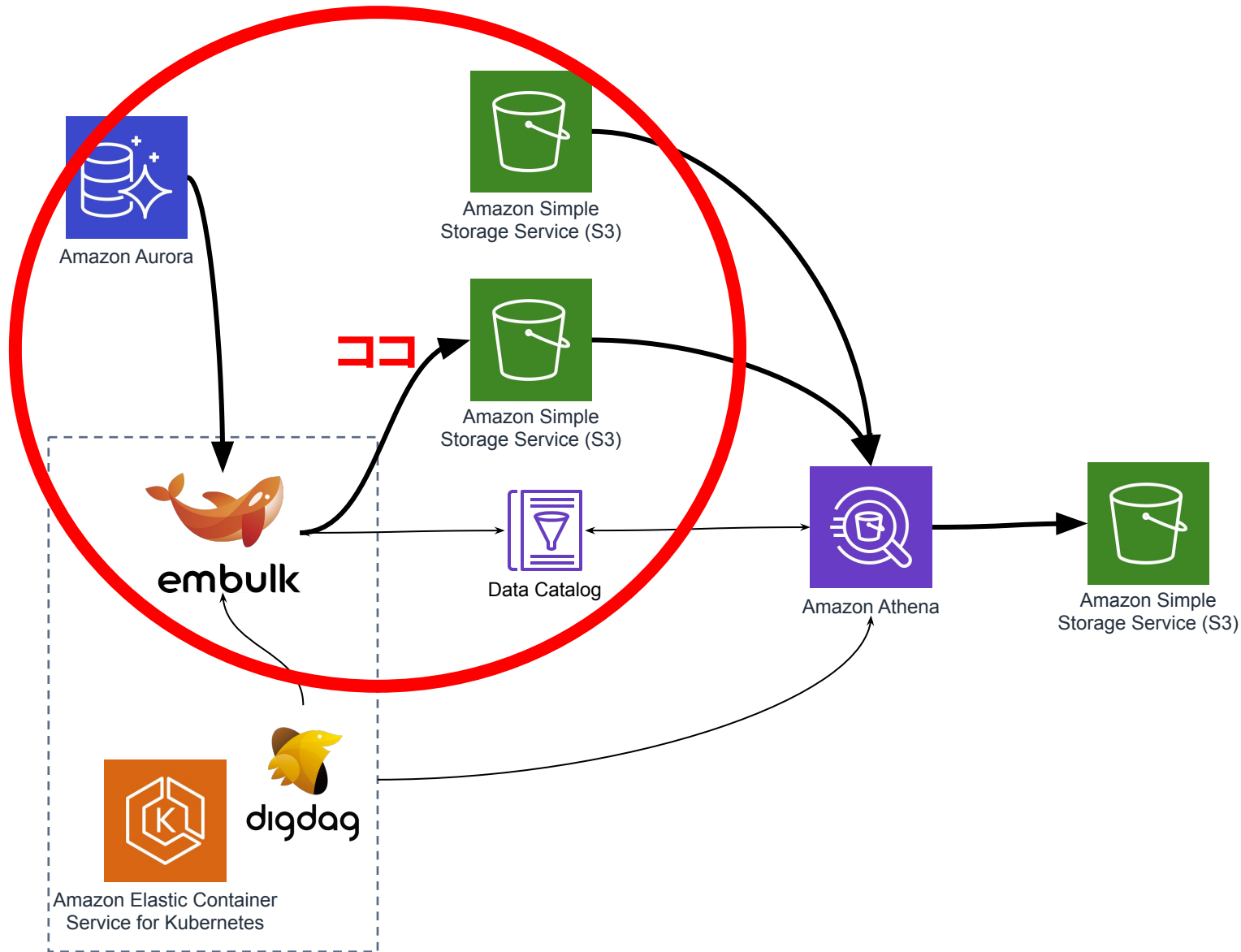
# データ格納までの流れ



**embulk-output-s3\_parquet**

# Digdag/Embulk/Athenaで作るお手軽ETL基盤

Gunosy



- S3 上に Parquet File を出力する Embulk Output Plugin
  - [https://github.com/civitaspo/embulk-output-s3\\_parquet](https://github.com/civitaspo/embulk-output-s3_parquet)
- 既に embulk-output-parquet という Plugin があるが AWS に特化して作られている点で異なる
  - AWS の全ての認証に対応している
    - embulk-output-parquet は hadoop-aws というライブラリに依存しているため ECS Task Role に対応していない
  - S3 固有の ACL 設定に対応している
  - catalog option がある

```
in:
  type: mysql
  host: example.rds.aws.com
  user: example_admin
  password: xxxxxxxx
  database: example_db
  table: example_table

out:
  type: s3_parquet
  bucket: my-bucket
  path_prefix: path/to/my-obj.
  file_ext: snappy.parquet
  compression_codec: snappy
  catalog:
    database: rds_example_db
    table: example_table
```

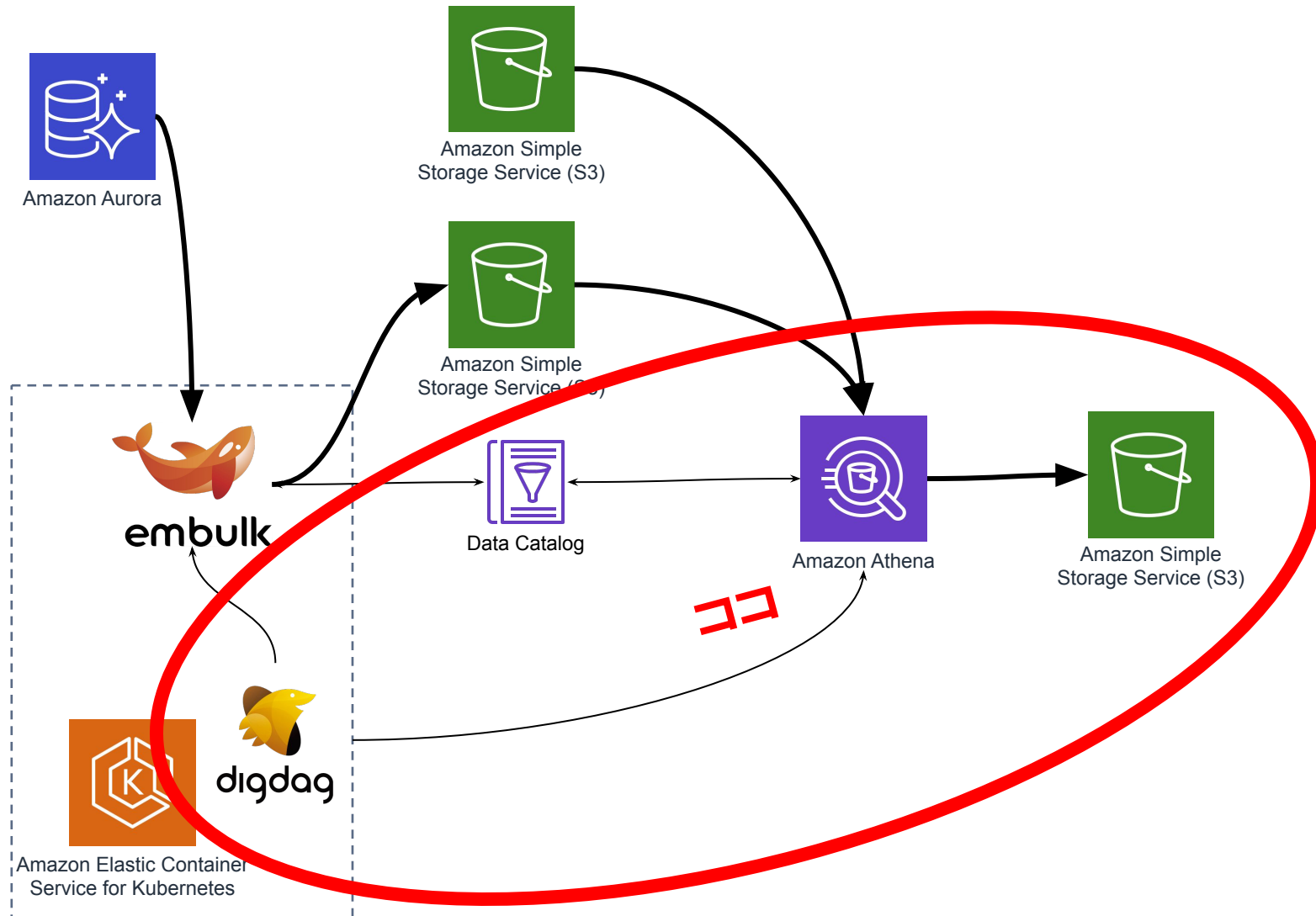
- 指定されると Glue Data Catalog に Table を作成する option
- Parquet File の作成に利用した Schema を元に Table 定義を自動生成してくれる
- MySQL から Full-dump するだけなら一切カラム定義を書かずに Athena にテーブルが作られる

どんなデータソース、どんなフォーマットでも  
Embulk を1度実行するだけで  
圧縮の効いた Athena Table としてアクセスできるようになった! お手軽!

**digdag-operator-athena**

# Digdag/Embulk/Athenaで作るお手軽ETL基盤

Gunosy





- digdag で athena を操作する Plugin
  - <https://github.com/civitaspo/digdag-operator-athena>
- 2019-07-30 時点で 10個の Operator をサポートしている
  - その中でもお手軽ポイントである2つを紹介する
    - **athena.ctas>**
    - **athena.apas>**

**athena.ctas> Operator**

- Athena の CTAS (Create Table As Select) 機能を便利に使う Operator
- SELECT 文と出力先情報を記述するだけで CTAS 対応の CREATE TABLE 文を自動生成して実行してくれる
  - 多くの利用者は SELECT 文には慣れているが CREATE TABLE 文には慣れていないという前提
  - CTASに必要な Format や Compression を事前に運用者側で定義してあげることで利用者は出力したいデータと出力先に集中できる

```
_export:  
  athena:  
    ctas:  
      format: parquet  
      compression: snappy  
      save_mode: error_if_exists  
  
+etl:  
  athena.ctas>: |  
    SELECT  
      a  
    , b  
    , c  
    , ROW_NUMBER() OVER (  
      ORDER BY d DESC  
    ) as rn  
  FROM  
    rds_example_db.example_table  
  database: etl_stage1  
  table: example_table_rn  
  location: s3://bucket/path/to/.../
```

- `_export` や `param file` などでフォーマットや圧縮形式、データ作成時の挙動など運用者側で事前に定義しておく
- SQL は `dig` ファイルに直接定義しても良いし、SQL ファイルの Path を記述することもできる

利用者は `database / table / location` といった出力先とデータを出力するための `SELECT` 文だけを気にすれば良い! お手軽!

CTAS を ETL として利用するために乗り越えたこと

- CTAS にはいくつかの制約がある
  - 既に Table が存在している場合は実行されないかエラーとなる
  - 出力先の Location が存在している場合エラーとなる
  - 必ず Table を作成しなければならない
- athena.ctas> Operator は save\_mode / table\_mode という2つの Option から事前事後処理を自動生成し上記制約を乗り越えている
  - save\_mode: Table/Location が存在していた場合の挙動
    - 上書きや存在していたら処理しないといった処理
  - table\_mode: Table/Location 作成時の挙動
    - テーブル定義のみ取得したい場合に空テーブルを作ったり
    - 逆にデータしか作成しないということもできる

```
_export:
dt_from: ${moment(session_time).add(-1, 'days').hours(0).format('YYYYMMDDHH')}
dt_to: ${moment(session_time).hours(0).format('YYYYMMDDHH')}
+prepare-etl:
+prepare-tables:
[parallel: true
+ad_imp:
athena.ctas>: |
SELECT DISTINCT
  abtest_name
, COUNT(DISTINCT bid_id) AS cnt
FROM
  adnwimps
WHERE
  dt >= '${dt_from}'
  AND dt < '${dt_to}'
  AND bid_id IS NOT NULL
GROUP BY 1
database: workspace
table: work_ad_imp
+ad_click:
athena.ctas>: |
SELECT
  abtest_name
, COUNT(DISTINCT bid_id) AS cnt
FROM
  gunosy.article_clk_log
WHERE
  dt >= '${dt_from}'
  AND dt < '${dt_to}'
  AND bid_id IS NOT NULL
GROUP BY 1
database: workspace
table: work_ad_click
+drop-table-before:
athena.drop_table>:
database: ad_kpi
table: yesterday_ctr
+kpi-etl:
athena.ctas>: |
SELECT
  imp.abtest_name AS abtest_name
, imp.cnt AS numimps
, click.cnt AS numclicks
, ROUND(100.0 * click.cnt / imp.cnt, 3) AS ctr
FROM
  workspace.work_ad_imp imp
, workspace.work_ad_click click
WHERE
  imp.abtest_name = click.abtest_name
database: ad_kpi
table: yesterday_ctr
location: s3://bucket/path/to/kpi/ctr/dt=${dt_from}/
```

- 必要なテーブルを複数作成後に最終的な出力を得る技
- 後で捨てるなら location を省略できる
- Query Timeout に引かかるような重いQueryを実行するときなどに使える

**athena.apas> Operator**

- Athena の CTAS (Create Table As Select) 機能を乱用して APAS (Add A Partition As Select) を実現する Operator
- SELECT 文と出力先情報を記述するだけで APAS に必要な処理を全て自動生成してくれる
  - Locaiton や Format / Compression も Table 情報から自動で生成するため SELECT 文と Partitioning の値のみ設定すればよい
    - Glue Data Catalog に正しく Table が定義されている必要はある
    - Locaiton や Format / Compression を明示することも可能
  - SELECT 文によって生成されるカラム定義と Table の定義に差分がないかの確認も行ってくれる



```
+etl:
  athena.apas>: sql/sample.sql
  database: etl_stage2
  table: sample_table
  partition_kv:
    p_date: '2019-07-31'
    p_hour: '19'
```

- Location/format/compression は Table 情報から自動生成
- SQL は dig ファイルに直接定義しても良いし、SQL ファイルの Path を記述することもできる

利用者は database / table / partition 情報といった出力先とデータを出力するための SELECT 文だけを気にすれば良い! お手軽!

athena.apas> が自動生成する Workflow について

1. save\_mode を元に事前処理実行: Partition/Location の存在を確認してエラーを発火したり削除を行ったり。
2. athena.ctas> operator の table\_mode: empty で空テーブルを作成
  - a. CREATE TABLE AS SELECT WITH NO DATA 構文
  - b. Glue Data Catalog 上にテーブル定義が作られる
3. 作成した空テーブルと Partition 追加するテーブルのカラム情報を比較し diff があればエラーを発火
4. 作成した空テーブルを削除
5. athena.ctas> operator で partition location に table\_mode: data\_only でデータのみ出力する
6. athena.add\_partition> operator で location とテーブルを紐付け

このパートでは構築したETL基盤のお手軽ポイントについて話しました

embulk-output-s3\_parquet

- AWS に特化して作ることによって AWS 特有の機能を実装できた
- catalog option は Athena 利用者にとって非常に便利

digdag-operator-athena

- CTAS 機能を ETL として使うための Workflow 自動生成機能は複雑な Workflow を考慮する必要がなくなり ETL 構築に強力なサポートを提供してくれる
- 特に APAS 機能は時系列データの蓄積を Athena 上で行う際に非常に楽に Workflow を構築することができる

ETL 基盤 = Extract / Transform / Load することに集中できる基盤

- 複雑な前処理・後処理は意識したくない・させたくない
- 処理を定型化して Plugin に隠蔽することで ETL Workflow 構築に集中できるようにした

# もくじ

- ① はじめに
- ② Gunosy の ETL 基盤事情
- ③ 構築したETL基盤の変遷
- ④ Digdag/Embulk/Athenaで構築するお手軽ETL基盤
- ⑤ おわりに

おわりに

おわりに

Gunosy

一緒に Gunosy で働こ  
う！！！！



# Gunosy

情報を世界中の人に最適に届ける