複数のデータストアを統一的に扱うAPIシステムの実装

2014/11/21

(株) ネットワーク応用通信研究所
原　悠

NaCl

本日のテーマ

- データストアへのクエリAPIを提供する部分の設計と実装
- agenda
  - 1. システムの構成について
  - 2. クエリAPIの設計について
  - 3. イベント駆動モデルについて

1. システムの構成

- 要件
  - 複数のデータストアに対し、(ある程度)統一的にアクセスしたい
    - 現在はInfluxDB/BigQuery/Impalaを使おうとしている
  - スケールアウトできるようにしておきたい

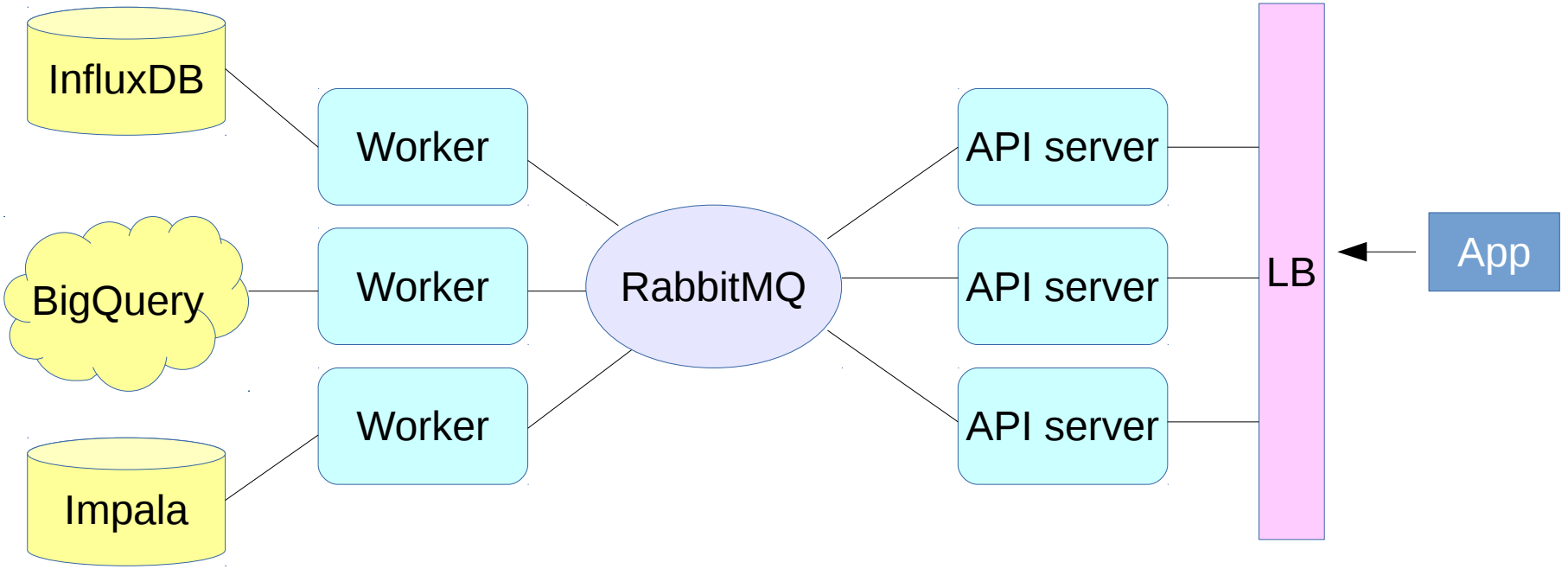
1. システムの構成

- Web APIを提供
  - データストアを指定してクエリ文字列を渡すと、クエリを実行してくれる
  - クエリ文字列の仕様はデータストアに依存
  - クエリ実行結果のフォーマットはある程度共通化
    - カラム名、データ

```
$ curl -d '{"queryEngine":"InfluxDB", \
             "query":"SELECT value FROM log_data LIMIT 1"}' \
       http://api_server/v1/QueryAndWait
{"statusCode":200,
 "result":{
   "header":{ ... },
   "data":{ ... }, ...
```

1. システムの構成

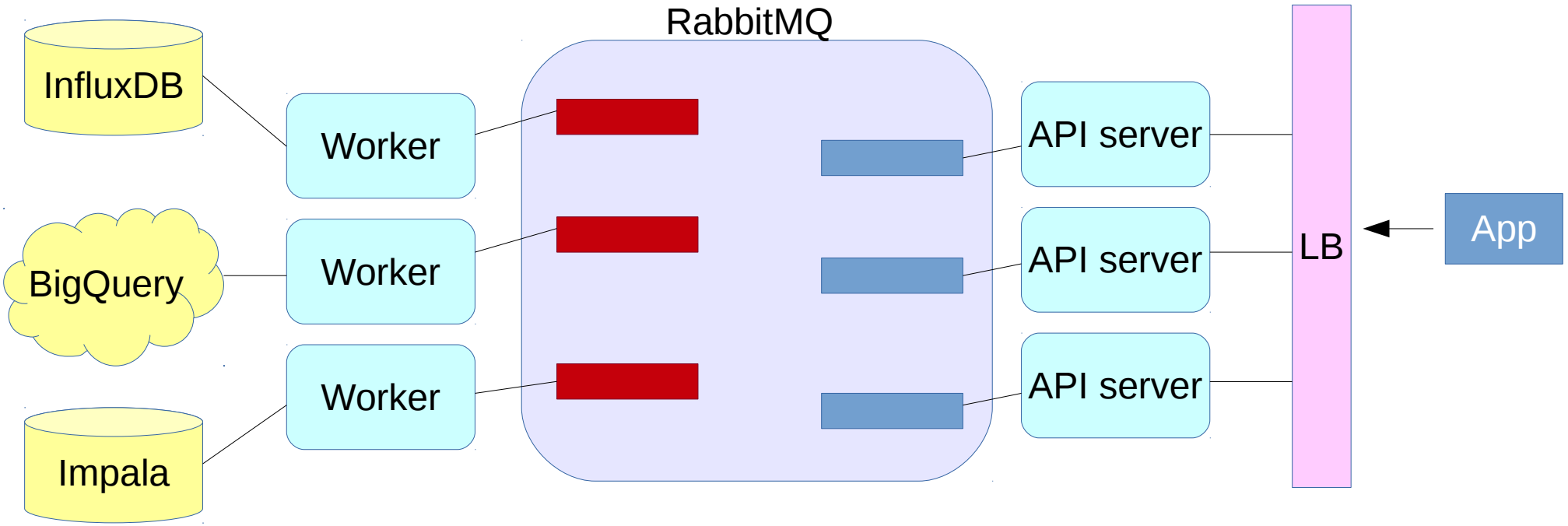
- メッセージキュー(RabbitMQ)を使用し、APIサーバとワーカーを分離
  - サービスを停止することなくスケールアウト可能
  - APIサーバ、ワーカはRubyで実装



RabbitMQのキュー構成

リクエスト用キュー：データストアごとに1つ

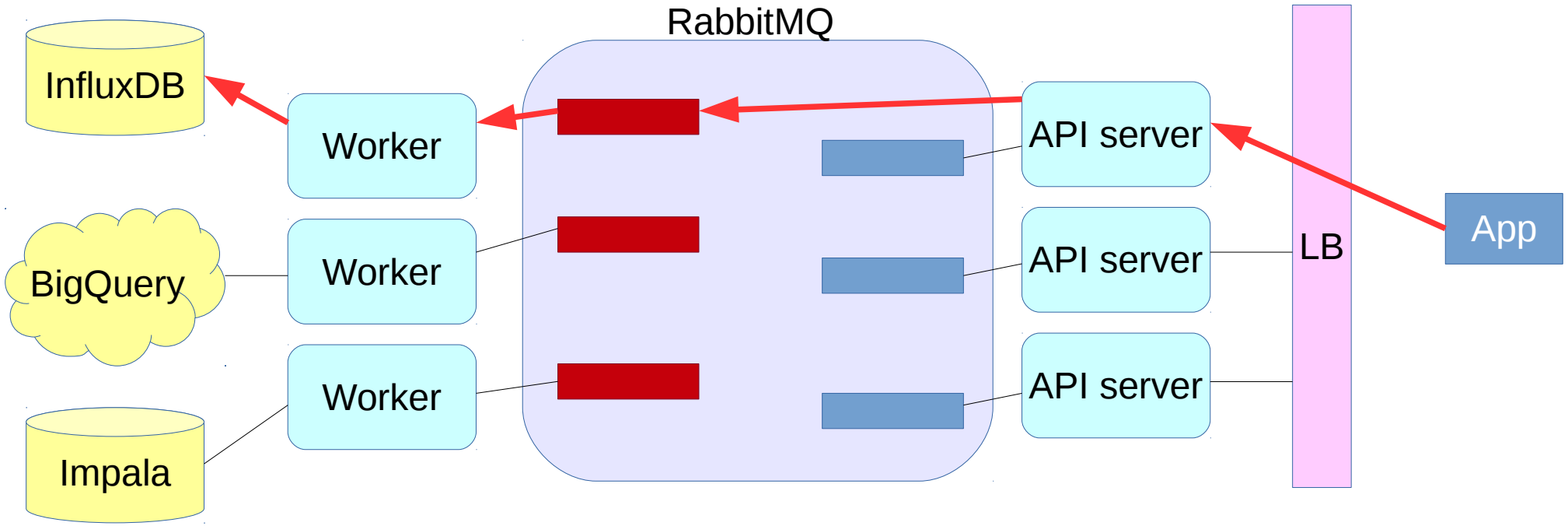
レスポンス用キュー：APIサーバごとに1つ



RabbitMQのキュー構成

リクエスト用キュー：データストアごとに1つ

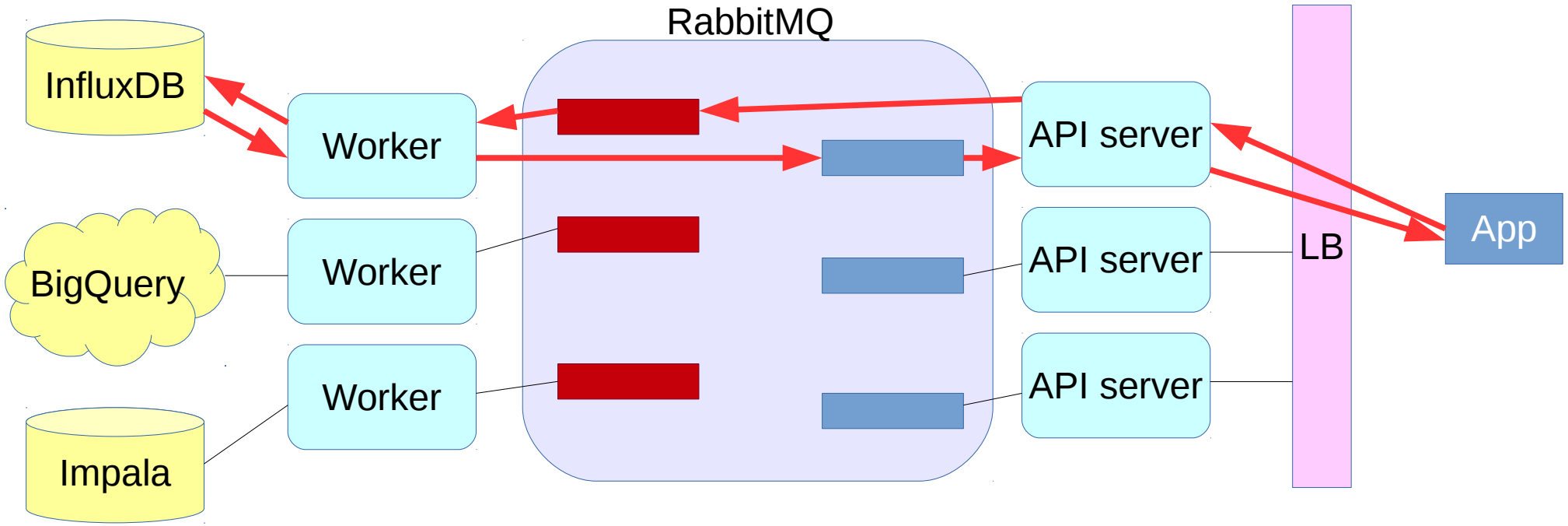
レスポンス用キュー：APIサーバごとに1つ



RabbitMQのキュー構成

リクエスト用キュー：データストアごとに1つ

レスポンス用キュー：APIサーバごとに1つ



2. クエリAPIの設計

- 同期(ブロックAPI)

```
result = QueryAndWait("SELECT ...")
```

  - データが小さい場合に使用
- 非同期(ノンブロックAPI)

```
queryId = Query("SELECT ...")
result = GetQueryResult(queryId)
```

  - データが大きい場合・実行時間が長い場合に使用
- コールバック

```
Query("SELECT ...",
      callbackUrl: "http://localhost:3000/")
```

  - 指定したURLに結果がPOSTされる
  - 実行時間がとても長い場合に使用

2.クエリAPIの設計

- クエリの実行結果がとても大きい場合どうするか
  - 全部が送られてくると困る
    ⇒先頭部分のみ返し、あとは別のAPI呼び出しで取得
- 続きをどう取得するか
  - a) APIを呼ぶたびに「続き」が順々に落ちてくる
    - Impala, BigQuery
  - b) limitとoffsetを指定して任意の箇所を取れる
    - BigQuery
    - Impalaでも、INSERT INTO ... SELECT ...を使えば可能

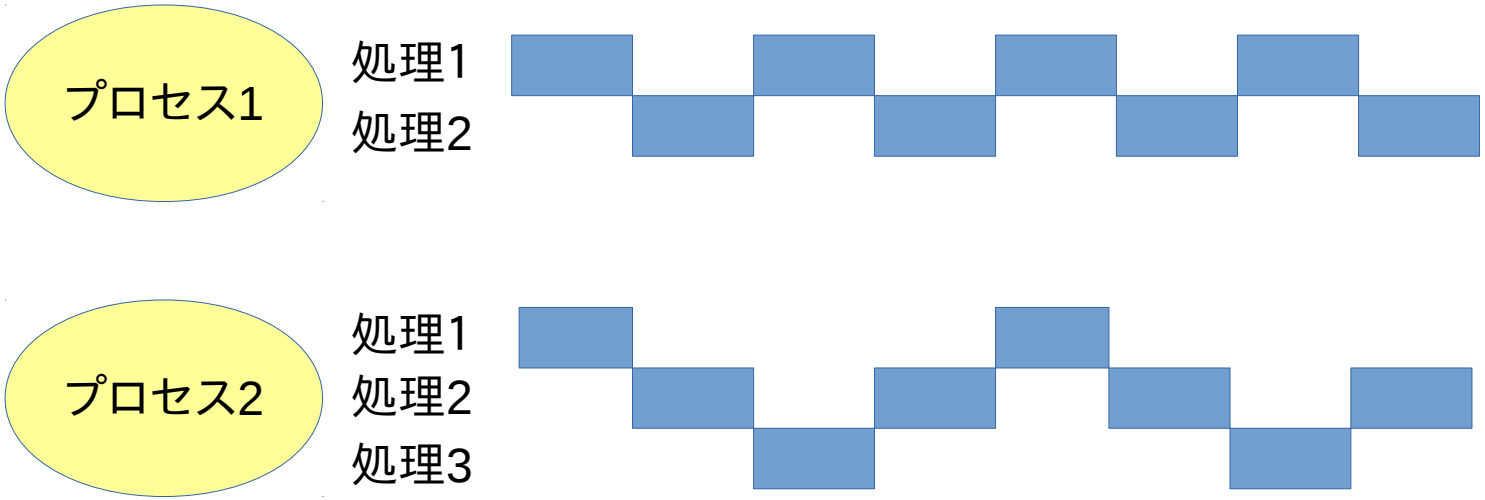
3. サーバ・ワーカーの実装

- どちらも待ち時間が多い
  - APIサーバ：ネットワーク待ち、実行結果待ち
  - ワーカー： DB待ち

  ⇒ スレッド、イベント駆動方式等
- 今回はイベント駆動方式を採用
  - (スレッドはRubyではあまり使われていない
    →スレッドに対応したライブラリが少ない)
  - eventmachine gemを使用

イベント駆動モデル

- シングルスレッド内で処理を切り替えていく
  - ネットワークやIO待ちのタイミングで別の処理をする
  - プロセスを複数立ち上げることでCPUコアを効率よく使う



イベント駆動モデルの注意点(1)

- 時間のかかる処理を直接書いてはいけない
  - イベント駆動モデルを前提としたライブラリを探す必要がある
  - 同期I/Oを行わないよう注意が必要(例:ログ出力)
- 例

```
# httpclient gem(同期API)
res = HttpClient.new.get(url)

# em-http-request gem(非同期API)
d = EM::HttpRequest.new(url).get
d.callback{ ... }
```

イベント駆動モデルの注意点(2)

- エラー処理に注意が必要
  - コールバック内でエラーが起きたらどうする？
  - 通常のRubyプログラムのように、例外は使えない

    エラーバックを持ちまわる必要がある
  - コールバック内で予期せぬ例外が発生した場合は、キャッチしてエラーバックを起動する必要がある
  - 手作業でやると煩雑になるので、何らかの仕組みが欲しい

    (例：EM::Deferrable)

イベント駆動モデルの注意点(2)

- エラー処理に注意が必要

```
01: def process_QueryAndWait(param)
02:   backend = select_backend(param)
03:   d = backend.exec_query(param[:query])
04:   d.callback{|result|
05:     begin
06:       send_response(200, result.data)
07:     rescue Exception => ex
08:       # エラー処理
09:     end
10:   }
11:   d.errback{|ex| # エラー処理 }
12: rescue Exception => ex
13:   # エラー処理
14: end
```

今後の予定

- BigQuery, Impala対応の実装
- 認証･アクセス制限
  - ユーザによっては一部のデータのみ閲覧可能とする