Apache Pulsar ハンズオン #2

2022/03/11

Pulsar Functions / Pulsar IO

自己紹介

- 名前
  - 古田(ふるた) 悠人(ゆうと)
- 経歴
  - 2018/04 ヤフー株式会社 新卒入社
  - 2018/10 Apache Pulsarを使った社内向けメッセージングプラットフォーム 開発/運用
  - 2021/05 Apache Pulsar Committer

アジェンダ

1. Pulsar Functionsとは
2. Pulsar Functionsを使ってみる
3. Pulsar IOとは
4. Pulsar IOを使ってみる

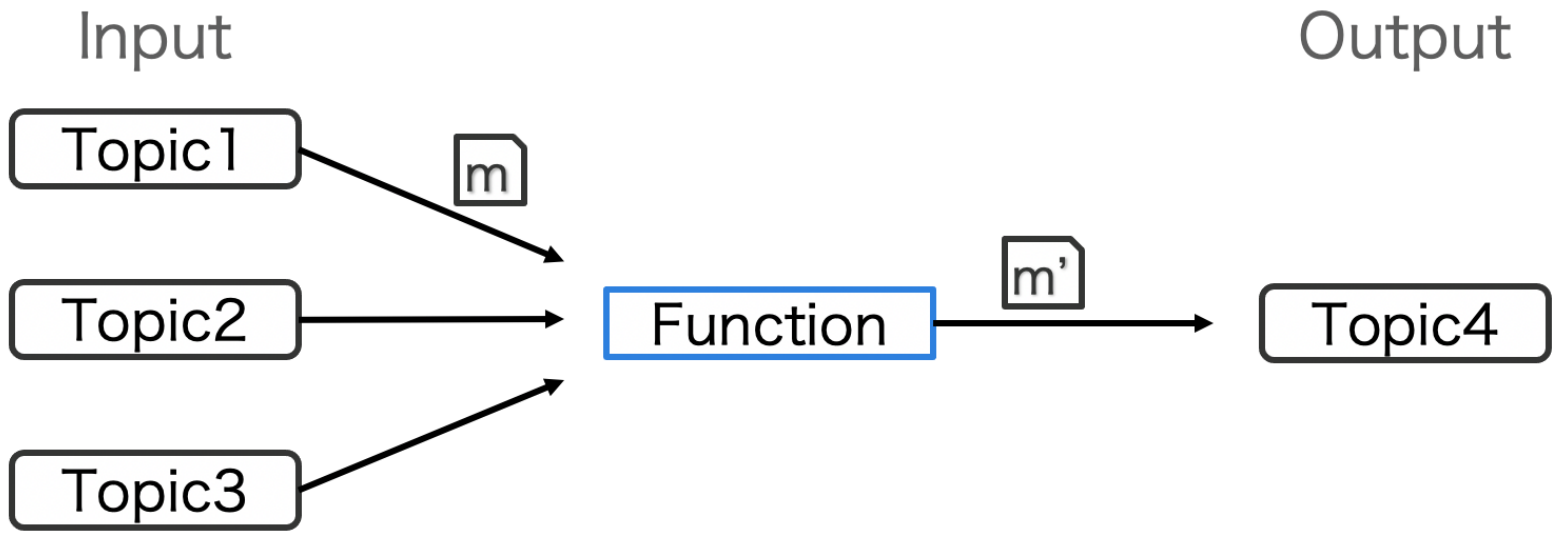
事前準備

- READMEの「Pulsar IO Cassandra Connectorの導入」にて準備をお願いいたします。
- pulsarコンテナと別に、ターミナルを5つ立ち上げてください。
  - pulsar-admin
  - consumer
  - producer
  - cassandra
  - cqlsh

Pulsar Functionsとは

Pulsar Functionsとは

- 流れてきたメッセージに任意の処理を適用し、他のトピックに送信する機能
- 処理をFunctionとして実装し、サーバに登録することで利用可能
- Functionで使用できる言語(v2.9.1)
    - Java, Python, Golang



Functionの例(Python)

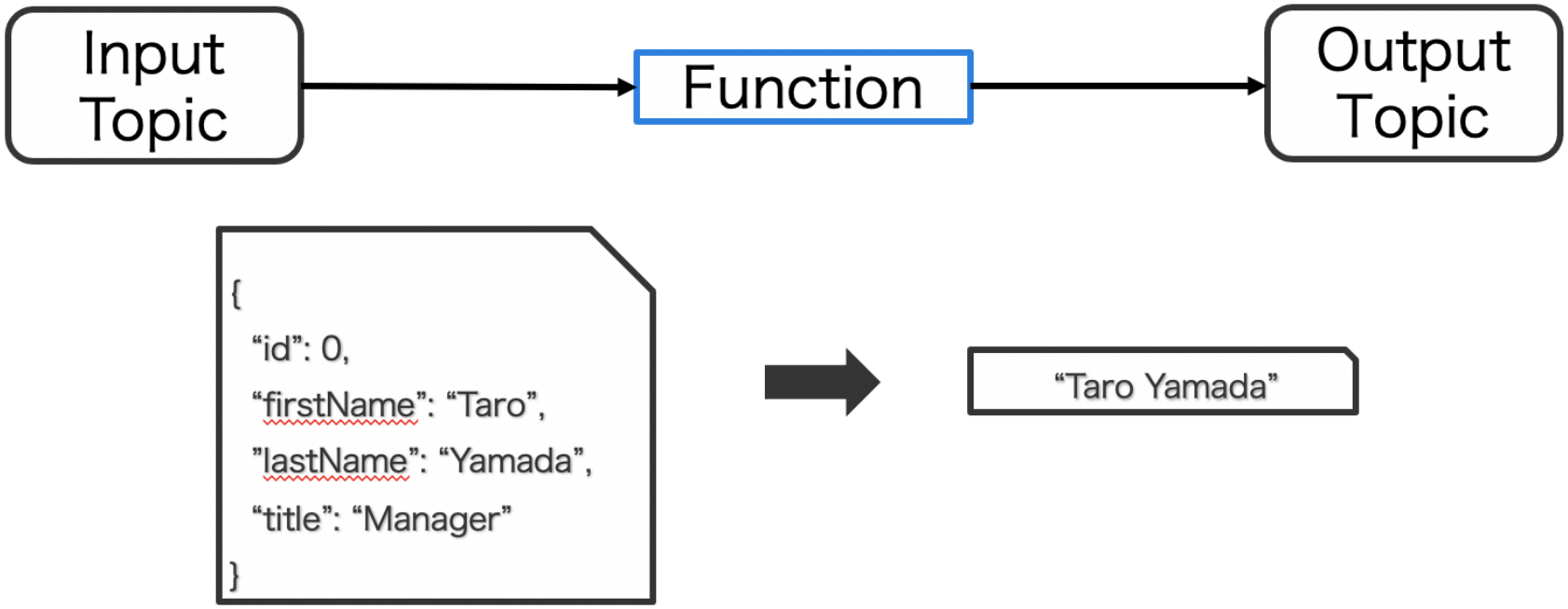
`function/exclamation_function.py`

```
from pulsar import Function

class ExclamationFunction(Function):
  def __init__(self):
    pass

  """
  * __init__ => processの順で実行される
  * input: 流れてきたメッセージの内容(byte)
  * context: Functionに関するメタ情報や機能
  * returnの引数がoutputトピックに流れる
  """
  def process(self, input, context):
    return input + '!!'
```

Pulsar Functionsを使ってみる

Pulsar Functionsを使ってみる

苗字と名前をつなげるFunction



Pulsar Functionsを使ってみる 手順

1. Functionの実装
2. Functionの登録
3. Inputトピックにメッセージを送信して確認

実装に入る前に

- ファイル中の `FIXME` という文字列を、
  スライドや正解用のファイル(`*_answer.py`)を参考にして修正してください。
- 修正したファイルで想定した挙動にならない場合は、
  正解用のファイルをコピペしてお試しください。

Functionの実装

`function/username_function.py`

```
from pulsar import Function

import json

class UsernameFunction(Function):
  def __init__(self):
    pass

  def process(self, input, context):
    input_json = json.loads(input)
    return ("%s %s" % (input_json["firstName"], input_json["lastName"]))
```

Functionの登録 1/3

```
## pulsar-adminコンテナを起動
$ docker-compose run pulsar-admin bash

## Functionの登録
root@pulsar-admin:/pulsar# ./bin/pulsar-admin functions create \
--py ./function/username_function.py \
--classname username_function.UsernameFunction \
--fqfn my-tenant/my-namespace/username_function \
--inputs persistent://my-tenant/my-namespace/input \
--output persistent://my-tenant/my-namespace/output

"Created successfully"
```

Functionの登録 2/3

`pulsar-admin functions create`

パラメータ	説明
--py	Functionのファイルパス
--classname	他Pythonファイルから参照するための名前 `<ファイル名>.<Functionのクラス名>`
--fqfn	Functionの登録名
--inputs	1つまたは複数のinputトピック
--output	1つのoutputトピック

Functionの登録 3/3

```
## 登録されたFunctionの確認
root@pulsar-admin:/pulsar# ./bin/pulsar-admin functions status \
--fqfn my-tenant/my-namespace/username_function
{
  "numInstances" : 1,
  "numRunning" : 1,
  "instances" : [ {
    "instanceId" : 0,
    "status" : {
      "running" : true, # trueならOK
...
```

Inputトピックにメッセージを送信して確認 1/5

Consumerを起動してメッセージを待つ

`client/username_consumer.py`

```
import pulsar
import logging

logger = logging.getLogger('pulsar')
logger.setLevel('INFO')
client = pulsar.Client('pulsar://pulsar:6650', logger=logger)

topic = "persistent://my-tenant/my-namespace/output"
consumer = client.subscribe(
    topic=topic, subscription_name="my-subscription")
```

(次のページに続く)

```
message = consumer.receive()
print(message.data().decode())
consumer.acknowledge(message)

consumer.close()
client.close()
```

Inputトピックにメッセージを送信して確認 2/5

Consumerを起動してメッセージを待つ

```
## consumerコンテナを起動
$ docker-compose run consumer bash

## consumerを起動
root@consumer:/pulsar# python ./client/username_consumer.py
```

Inputトピックにメッセージを送信して確認 3/5

Producerを起動して、メッセージを送信する

`client/username_producer.py`

```
import pulsar
import logging

class Employee(pulsar.schema.Record):
  id = pulsar.schema.Integer()
  firstName = pulsar.schema.String()
  lastName = pulsar.schema.String()
  title = pulsar.schema.String()

logger = logging.getLogger('pulsar')
logger.setLevel('INFO')
client = pulsar.Client('pulsar://pulsar:6650', logger=logger)
```

(次ページに続く)

```
topic = "persistent://my-tenant/my-namespace/input"
schema = pulsar.schema.JsonSchema(Employee)

producer = client.create_producer(topic=topic, schema=schema)
employee = Employee(id=100, firstName="Taro",
                    lastName="Yamada", title="Manager")
producer.send(employee)

producer.close()
client.close()
```

Inputトピックにメッセージを送信して確認 4/5

Producerを起動して、メッセージを送信する

```
## producerコンテナを起動
$ docker-compose run producer bash

## Producerを起動
root@producer:/pulsar# python ./client/username_producer.py
```

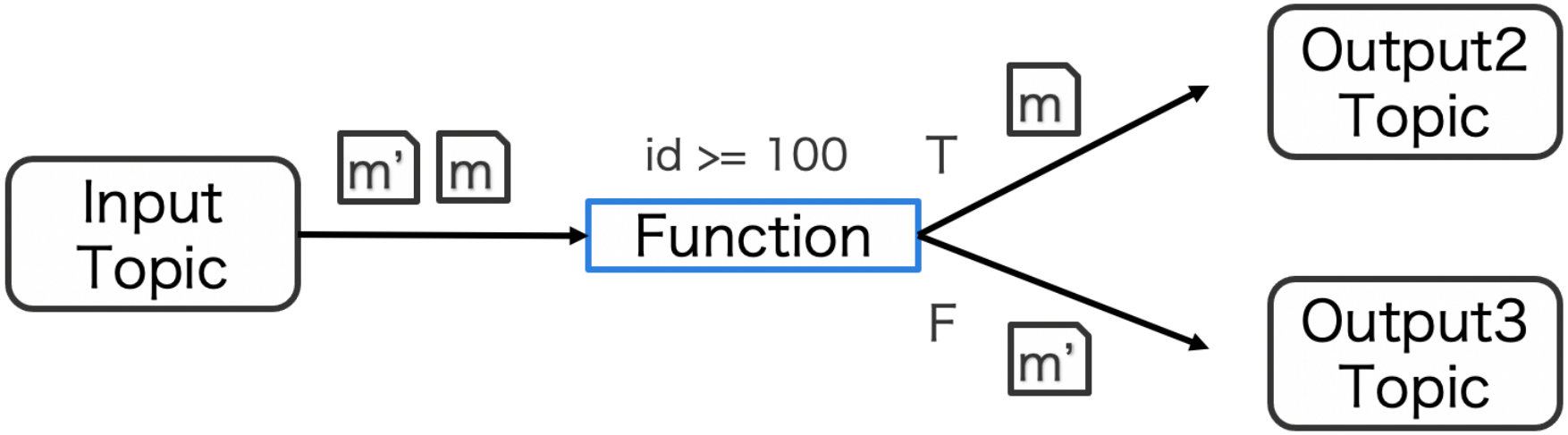
Inputトピックにメッセージを送信して確認 5/5

Consumerの画面で、以下のような文字列が出ればOK

```
Taro Yamada
```

時間が余った方は

idによってメッセージを振り分けるFunction



Functionの実装

`function/id_routing_function.py`

```
from pulsar import Function

import json

class IdRoutingFunction(Function):
  def __init__(self):
    self.output2_topic = "persistent://my-tenant/my-namespace/output2"
    self.output3_topic = "persistent://my-tenant/my-namespace/output3"
```

(次のページに続く)

```
    """
    * context.publishを使うと、引数に指定したトピックにメッセージを流すことができる
    """
    def process(self, input, context):
      input_json = json.loads(input)
      name = ("%s %s" % (input_json["firstName"], input_json["lastName"]))
      if(input_json["id"] >= 100):
        context.publish(self.output2_topic, name)
      else:
        context.publish(self.output3_topic, name)
```

Functionの登録 1/2

```
## pulsar-adminコンテナからFunctionを登録
root@pulsar-admin:/pulsar# ./bin/pulsar-admin functions create \
--py ./function/id_routing_function.py \
--classname id_routing_function.IdRoutingFunction \
--fqfn my-tenant/my-namespace/id_routing_function \
--inputs persistent://my-tenant/my-namespace/input

"Created successfully"
```

Functionの登録 2/2

```
## 登録されたFunctionの確認
root@# ./bin/pulsar-admin functions status --fqfn my-tenant/my-namespace/id_routing_function
{
  "numInstances" : 1,
  "numRunning" : 1,
  "instances" : [ {
    "instanceId" : 0,
    "status" : {
      "running" : true, # trueならOK
...
```

Inputトピックにメッセージを送信して確認

- `client/username_producer.py` のidを変更して
  `persistent://my-tenant/my-namespace/output2` と
  `persistent://my-tenant/my-namespace/output3` に対して
  メッセージの振り分けがされるか確認
  - `client/username_consumer_output2.py` と
    `client/username_consumer_output3.py` をConsumerとしてご利用ください

Pulsar Functions まとめ

- 流れてきたメッセージに任意の処理を適用し、他のトピックに送信する機能
- 利用手順
    i. Functionを実装
    ii. Functionをサーバに登録
    iii. Inputトピックにメッセージを送信

公式ドキュメント
https://pulsar.apache.org/docs/en/functions-overview/

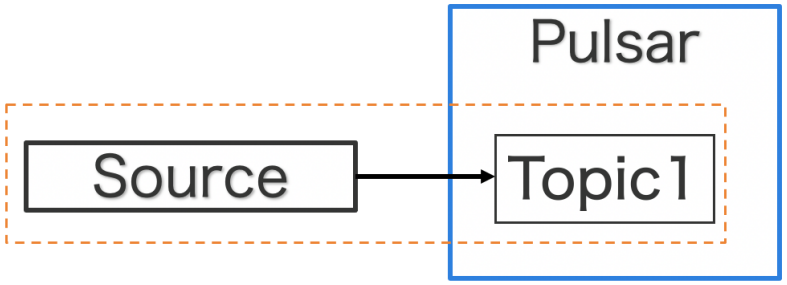
Pulsar IOとは

Pulsar IOとは

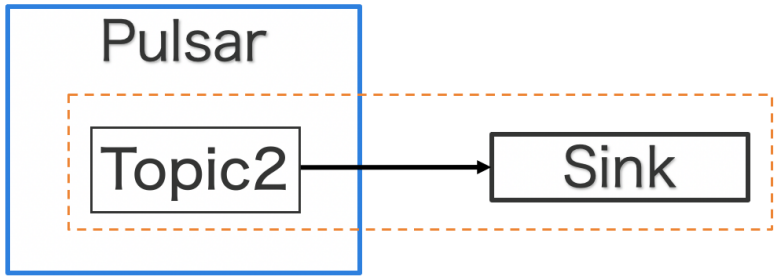
- データベースや他のメッセージングシステムなどの
  外部システムとPulsarをつなげるためのConnector
    - 外部システムとトピックを紐づける
- 公式で提供されているConnector: https://pulsar.apache.org/en/download/
- Connectorを自作することもできる

Source/Sink

- Source
  - 外部システムに書き込まれたデータがトピックに自動で流れる



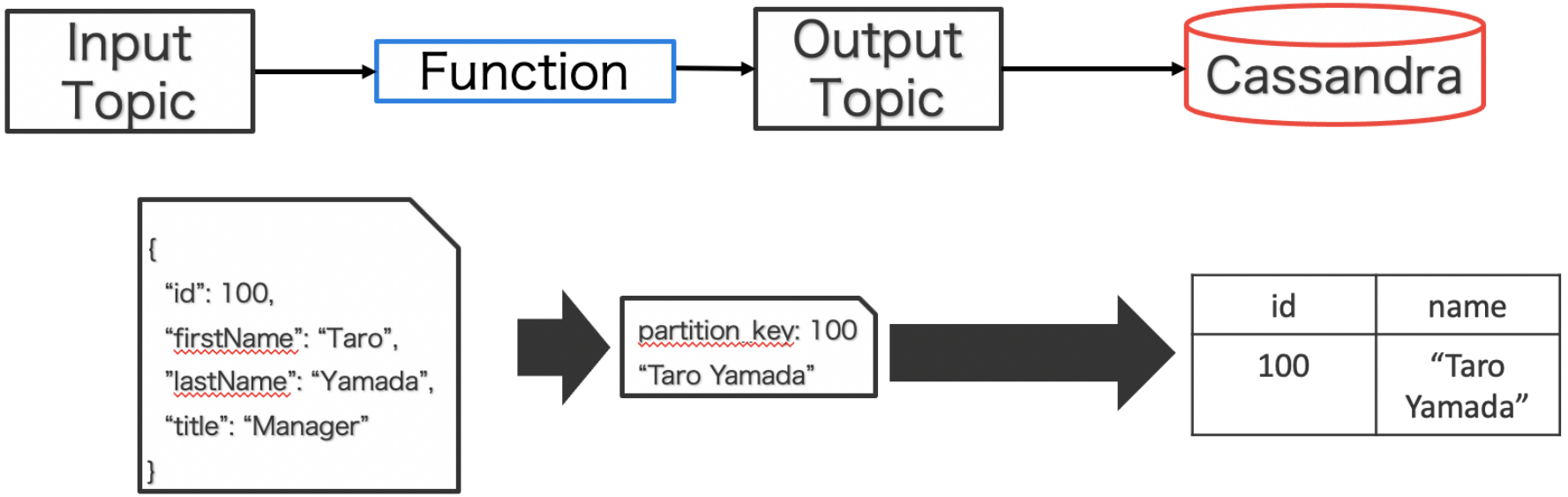
- Sink
  - トピックに書き込まれたデータが外部システムに自動で流れる



Pulsar IOを使ってみる

Pulsar IOを使ってみる

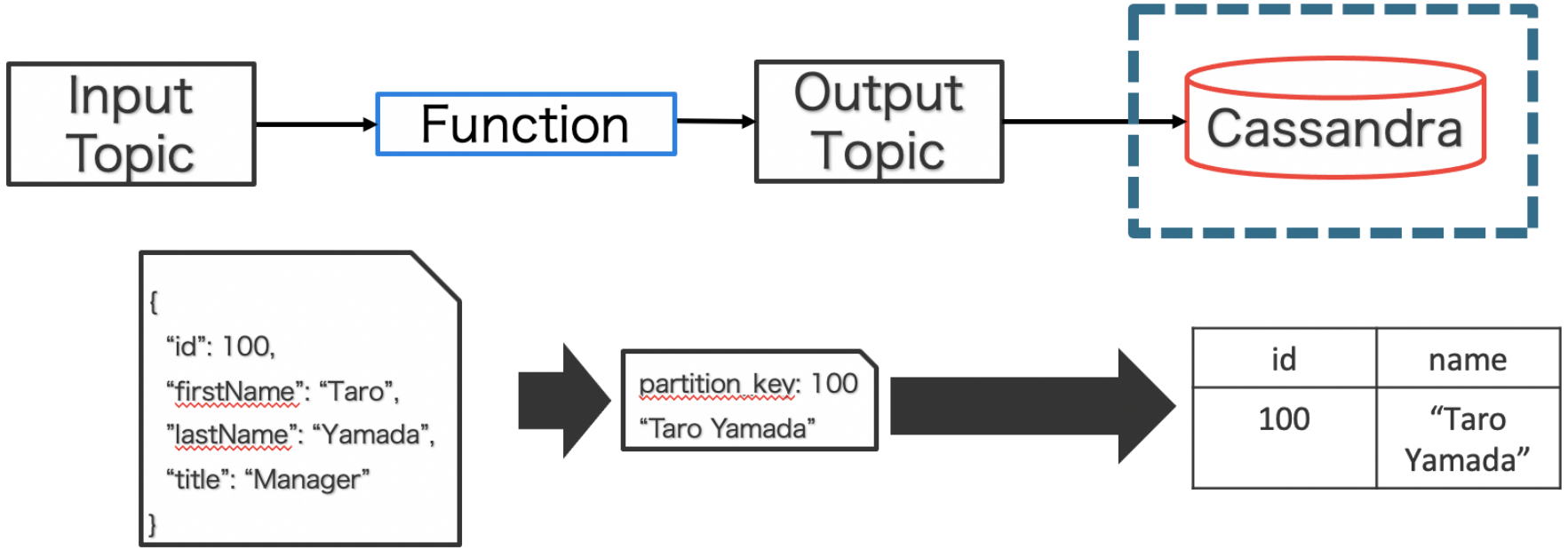
- Pulsarサーバに送信されたメッセージをCassandraに流す。
  - Cassandraとトピックを紐づけておき、Functionで作ったメッセージをそこに送る



Pulsar IOを使ってみる 手順

1. Cassandraの設定
    i. Connectorの配置
    ii. Cassandraの起動
    iii. keyspace/tableの作成
2. Sinkの設定
    i. Sinkの作成
    ii. Sinkの登録
3. Functionの設定
    i. Functionの実装
    ii. Functionの登録
4. トピックにメッセージを送信

Cassandraの設定



Connectorの配置

外部システムに対応するConnectorをPulsarサーバに置く

- READMEの「Pulsar IO Cassandra Connectorの導入」に相当

```
# pulsar-adminコンテナで実行
## Pulsarサーバに設定されているConnectorを取得
root@pulsar-admin:/pulsar# curl -s http://pulsar:8080/admin/v2/functions/connectors

[{"name":"cassandra",...}]
```

上記のコマンドの実行結果で何も出てこない場合は

次のページの手順を行う。

前のページでうまくいかなかったときの手順

1. READMEの「Pulsar IO Cassandra Connectorの導入」を行う
2. pulsarコンテナを再起動

```
## pulsarコンテナを停止
$ docker-compose stop pulsar

## ホストマシンのコマンドプロンプトが出るまでしばらく待つ
## その後、pulsarコンテナを起動
$ docker-compose up pulsar
```

3. 前のページのコマンドを再度実行

Cassandraの起動

```
## cassandraコンテナの起動
$ docker-compose up cassandra
```

しばらく待ち、以下を実行する

```
## Cassandraのstatusを確認
$ docker-compose exec cassandra nodetool status

Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens       Owns (effective)  Host ID                               Rack
UN  172.17.0.2  103.67 KiB  256          100.0%            af0e4b2f-84e0-4f0b-bb14-bd5f9070ff26  rack1
```

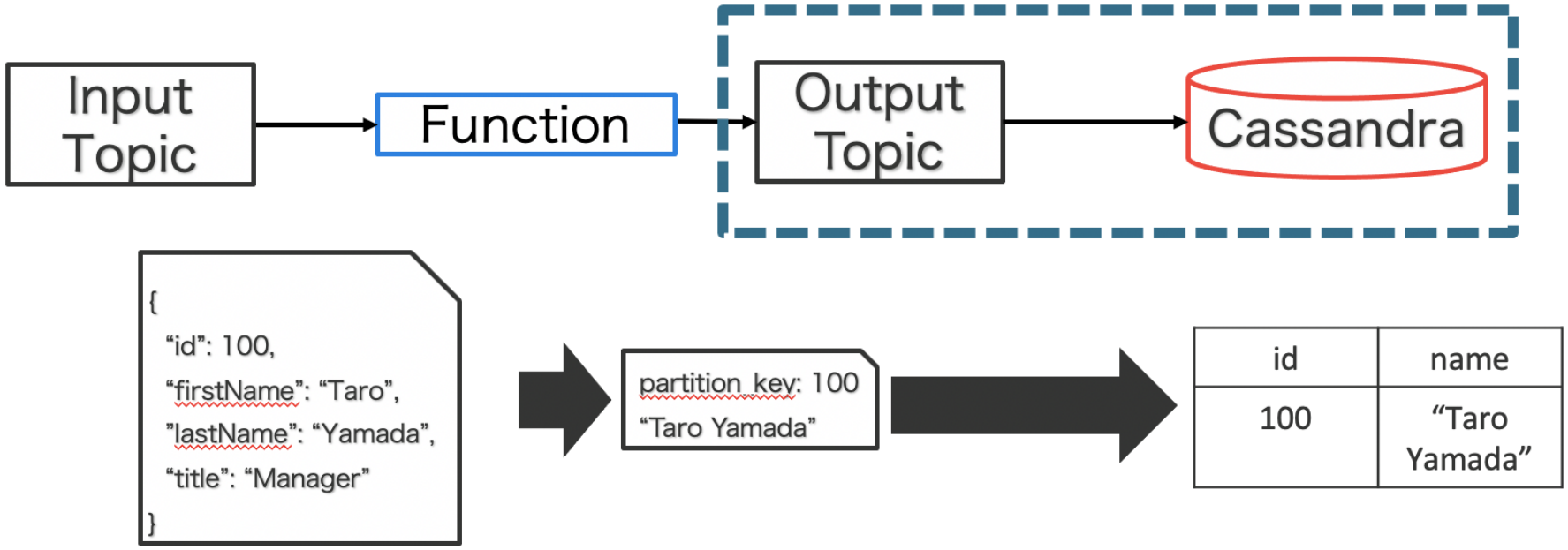
keyspace/tableの作成 1/2

```
$ docker-compose exec cassandra cqlsh localhost

## keysapceの作成
cqlsh> CREATE KEYSPACE pulsar_handson_keyspace
WITH replication = {'class':'SimpleStrategy','replication_factor':1};

## keyspaceの指定
cqlsh> USE pulsar_handson_keyspace;

## tableの作成
cqlsh:pulsar_handson_keyspace> CREATE TABLE pulsar_handson_table
(id text PRIMARY KEY, name text);
```

keyspace/tableの作成 2/2

```
## tableに何も保存されていないことを確認
cqlsh:pulsar_handson_keyspace> SELECT * FROM pulsar_handson_table;

 id | name
----+------

(0 rows)
```

Sinkの設定



Sinkの作成

`sink/cassandra-sink.yml`

```
configs:
  # 接続先の外部システム
  roots: "cassandra:9042"
  # keyspace
  keyspace: "pulsar_handson_keyspace"
  # columnFamily
  columnFamily: "pulsar_handson_table"
  # key
  keyname: "id"
  # column
  columnName: "name"
```

Sinkの登録 1/3

```
## Sinkの登録
root@pulsar-admin:/pulsar# ./bin/pulsar-admin sinks create \
--tenant my-tenant \
--namespace my-namespace \
--name cassandra-sink \
--sink-type cassandra \
--sink-config-file sink/cassandra-sink.yml \
--inputs persistent://my-tenant/my-namespace/output

"Created successfully"
```

Sinkの登録 2/3

`pulsar-admin sinks create`

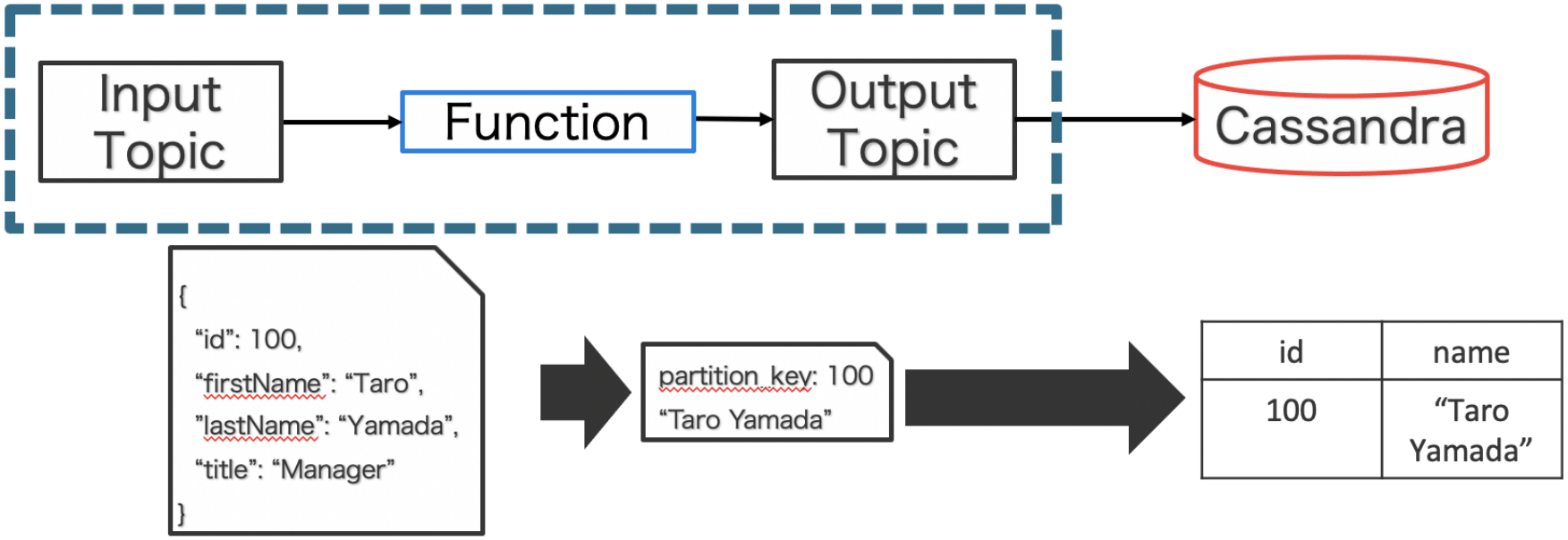
パラメータ	説明
--tenant	登録するSinkのTenantの名前
--namespace	登録するSinkのNamespaceの名前
--name	登録するSinkの名前
--sink-type	外部システムの種類
--sink-config-file	sinkの設定ファイルのパス
--inputs	1つまたは複数のinputトピック

Sinkの登録 3/3

登録されたSinkの確認

```
root@pulsar-admin:/pulsar# ./bin/pulsar-admin sinks status \
--tenant my-tenant \
--namespace my-namespace \
--name cassandra-sink

{
  "numInstances" : 1,
  "numRunning" : 1,
  "instances" : [ {
    "instanceId" : 0,
    "status" : {
      "running" : true, # trueならOK
....
```

Functionの設定



Functionの実装

以下の`function/to_cassandra_function.py`を参考に`function/username_function.py`を修正してください。

`function/to_cassandra_function.py`

```
from pulsar import Function

import json

class UsernameFunction(Function):
  def __init__(self):
    pass
```

(次ページに続く)

Functionの実装

```
    """
    * context.publishを使うと、引数に指定したトピックにメッセージを流すことができる。
    * partition_keyに入れた値がcassandraのkeyに入る
    """
    def process(self, input, context):
      input_json = json.loads(input)
      name = ("%s %s" % (input_json["firstName"], input_json["lastName"]))
      message_conf = {}
      message_conf["partition_key"] = str(input_json["id"])
      output_topic = "persistent://my-tenant/my-namespace/output"
      context.publish(output_topic, name, message_conf=message_conf)
```

Functionの登録 1/2

```
## Functionの更新
root@pulsar-admin:/pulsar# ./bin/pulsar-admin functions update \
--py ./function/username_function.py \
--classname username_function.UsernameFunction \
--fqfn my-tenant/my-namespace/username_function \
--inputs persistent://my-tenant/my-namespace/input \
--output persistent://my-tenant/my-namespace/output

"Updated successfully"
```

Functionの登録 2/2

```
## 登録されたFunctionの確認
root@pulsar-admin:/pulsar# ./bin/pulsar-admin functions status \
--fqfn my-tenant/my-namespace/username_function
{
  "numInstances" : 1,
  "numRunning" : 1,
  "instances" : [ {
    "instanceId" : 0,
    "status" : {
      "running" : true, # trueならOK
...
```

トピックにメッセージを送信

```
## Producerの起動
root@producer:/pulsar# python ./client/username_producer.py
```

Cassandraにメッセージが保存されているか確認

```
## Cassandraに保存されているか確認
cqlsh:pulsar_handson_keyspace> SELECT * FROM pulsar_handson_table;
```

以下のように表示されていればOK

```
 id  | name
-----+--------------
 100 | Taro Yamada

(1 rows)
```

時間が余った方は

- `client/username_producer.py` のIDや名前を別のものに修正して Cassandraに書き込まれるか確認してみましょう

Pulsar IO まとめ

- データベースや他のメッセージングシステムなどの
  外部システムとPulsarをつなげるためのコネクタ
- 利用手順
  i. 外部システムに対応するConnectorをPulsarサーバに配置
  ii. Pulsarサーバの再起動
  iii. 外部システムの起動
  iv. Source/Sinkの設定
  v. 外部システム/Topicにメッセージを書き込む
- 公式ドキュメント
  - https://pulsar.apache.org/docs/en/io-overview/

Function/Source/Sinkのログ確認

```
## pulsarコンテナにログイン
$ docker-compose exec pulsar bash

## ログファイルがあるディレクトリ一覧
root@# ls logs/functions/my-tenant/my-namespace/
cassandra-sink  username_function

## ログファイルを見る
root@# less logs/functions/my-tenant/my-namespace/username_function/username_function-0.log
```