Apache Pulsar ハンズオン #2

2022-03-11

自己紹介

- 木村（きむら） 武志（たけし）
- 来歴
  - 2020-04 ヤフー株式会社 入社
  - 2020-10 社内 messaging platform 運用の team に配属

本日の流れ

19:00	**Pulsar**: 概要を説明
19:15	**Pulsar standalone**: 単純な送受信
19:45	**Pulsar Schema**: 複雑な data の送受信
20:00	休憩
20:10	**Pulsar Functions**: data の処理
20:40	**Pulsar IO**: 外部 system と接続
21:20	終了

Pulsarの概要

Pulsar

- 出版-購読 (Pub-Sub) 方式で message を送受信するための platform
- Yahoo Inc. で開発された
- *The Apache Software Foundation* の top level project (2018/09-)

特徴:

- multi-tenancy
  - 複数の user, service が利用可能
- geo-replication
  - 複数の cluster に data を複製する
- 低遅延
  - publish 遅延 < 5ms
- scale可能
  - topic 数 > 1M

https://pulsar.apache.org/docs/en/concepts-overview/

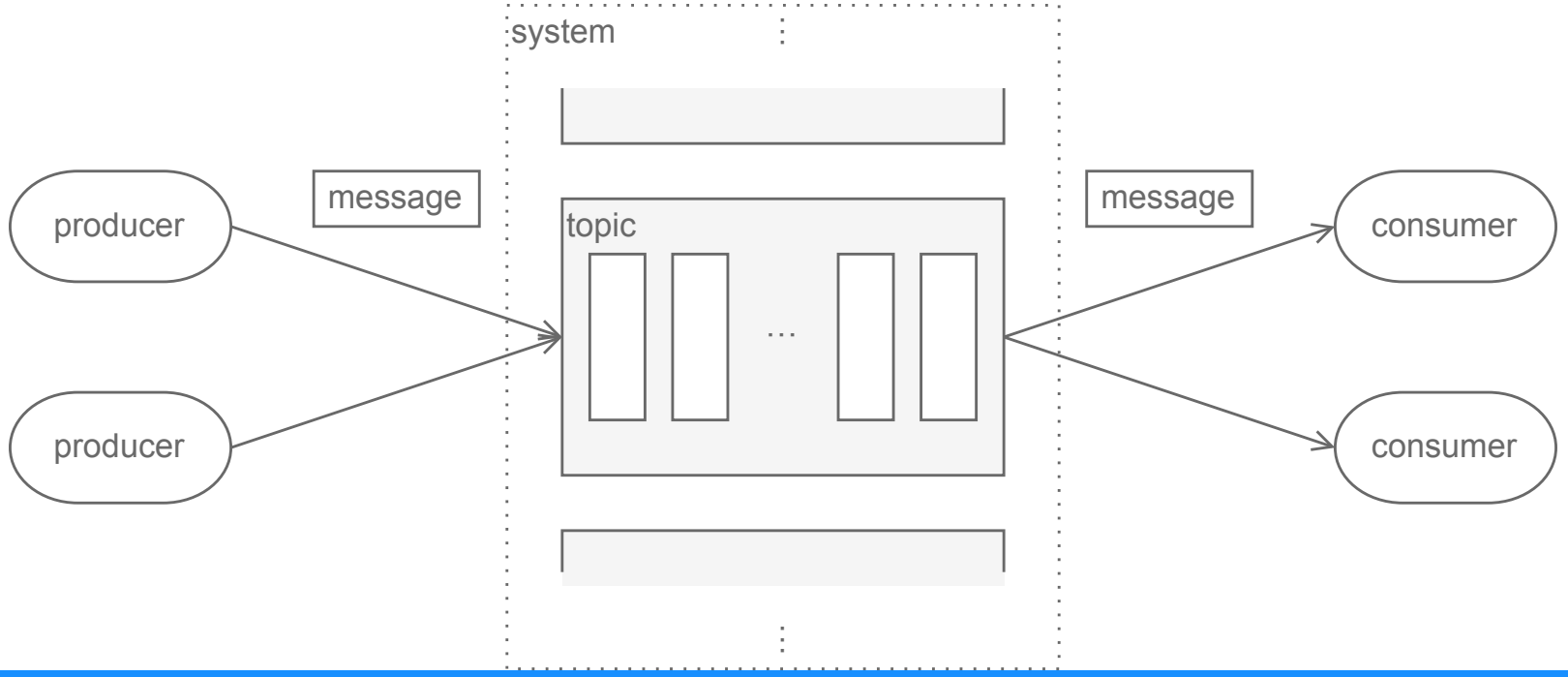
利用企業

- Yahoo Inc.
- Tencent
- Baidu
- Huawei
- and more*

* https://pulsar.apache.org/powered-by/

出版-購読 (pub-sub) 方式

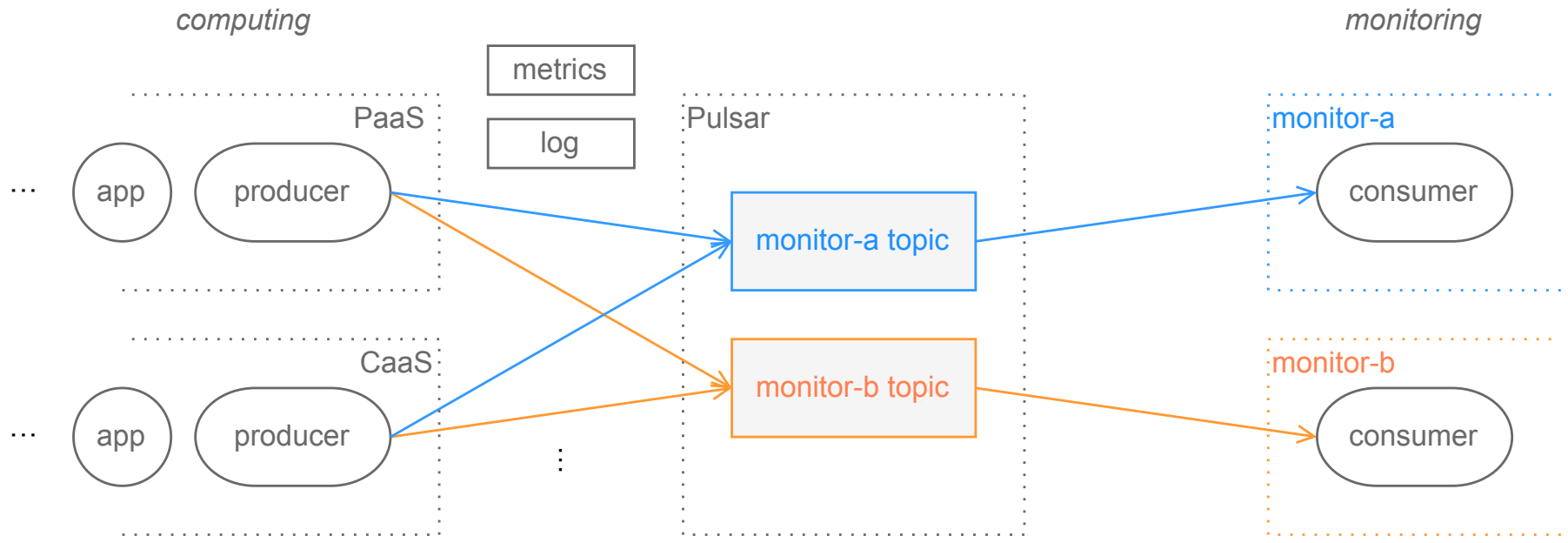
- producer: 送信側
- consumer: 受信側
- topic: producer, consumer が接続する endpoint
- producer, consumer は topic だけ知っていれば良い (疎結合)



利用例

- 社内:
  - 複数の service と 複数の監視 platform が存在
  - 各 service × 監視 platform 毎に agent を設定する必要が有った

- Pulsar を利用する事で利用が用意・scale 可能に



topic

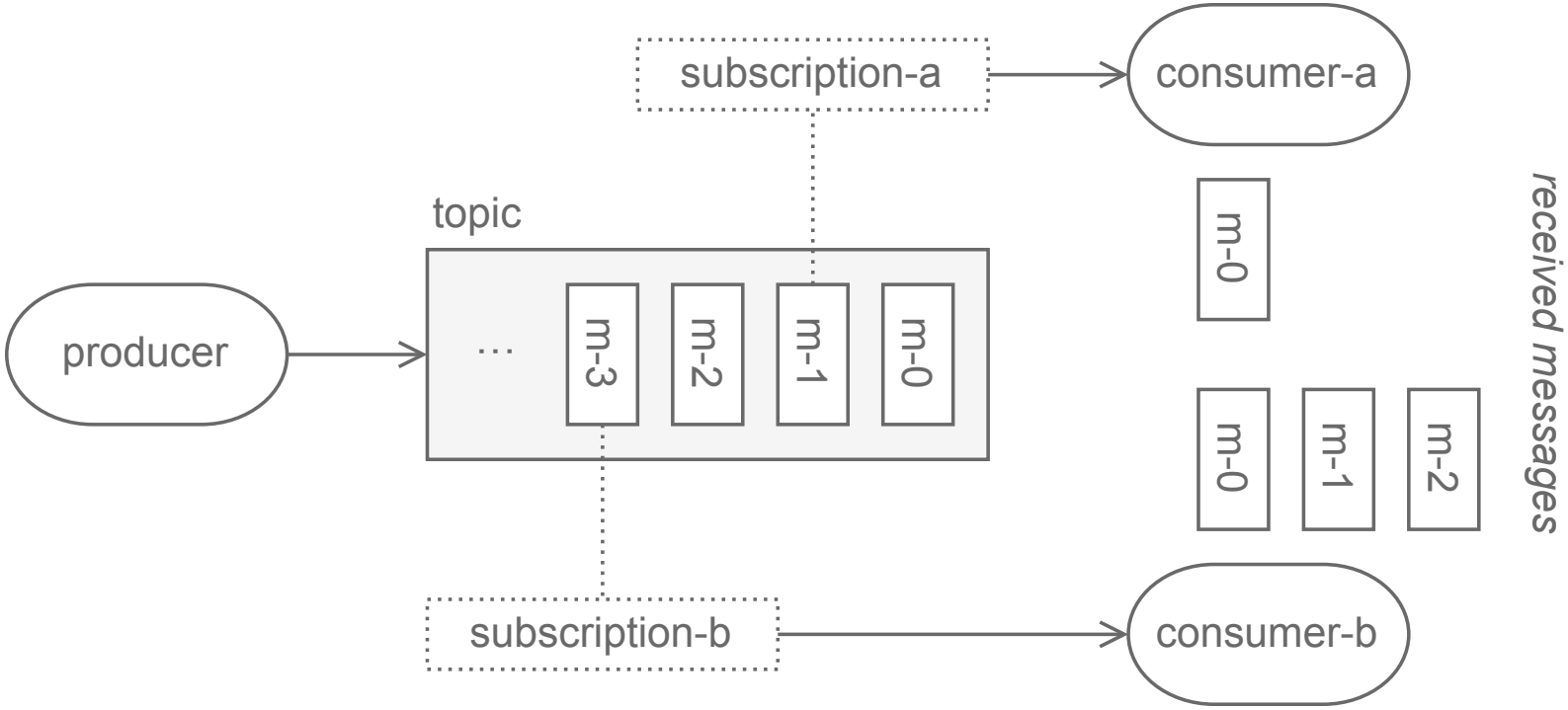
- topic 名は以下の形を取る

```
persistent://${tenant}/${namespace}/${topic}
```

- `persistent`: message が永続化される
  - 永続化されず memory だけに保持される `non-persistent` も有る
- namespace: topic の名前空間
- tenant: namespace の名前空間

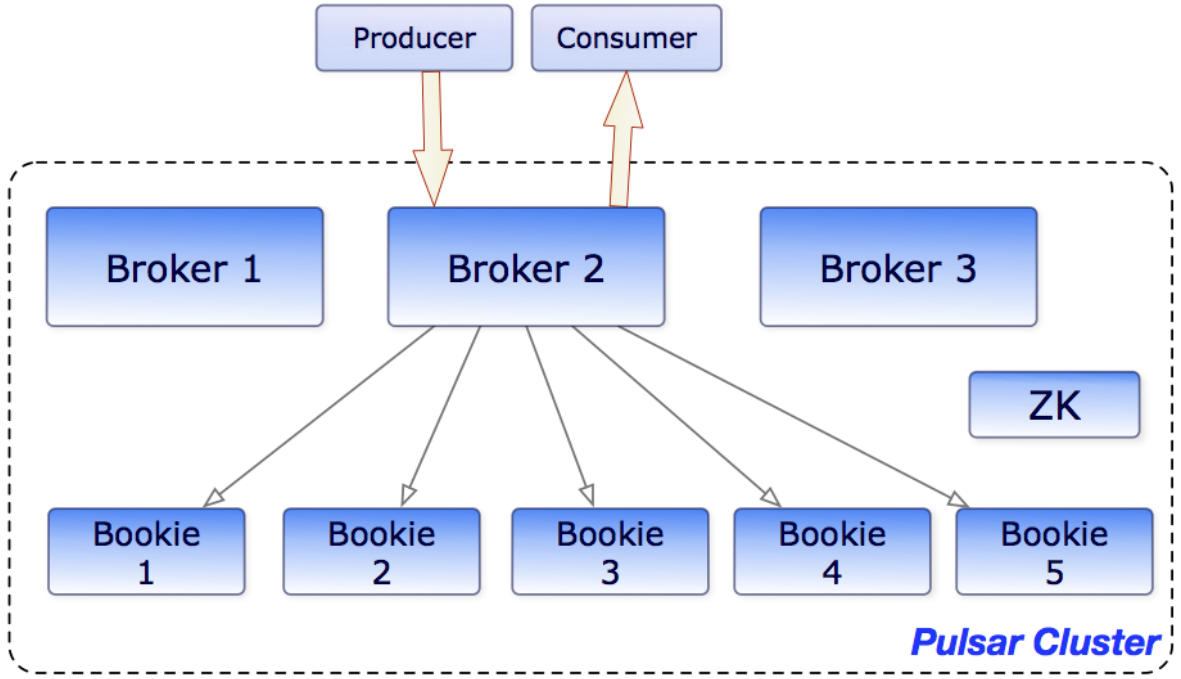
subscription

- consumer は topic を購読 (subscribe) する際に *subscription* を作る
- subscription は topic 内の message をどこまで受信したかを管理する



構成

- *broker*:
  - producer, consumer を仲介する
  - admin 用の REST API を提供する
- *bookie*:
  message を保持する
- *zookeeper* (右図 ZK):
  各種設定値 などの metadata を保持する



https://pulsar.apache.org/docs/en/concepts-architecture-overview/

Pulsar standalone

- 発表資料は配布されているので、適宜御確認ください
  - https://github.com/FIXME

standalone mode とは

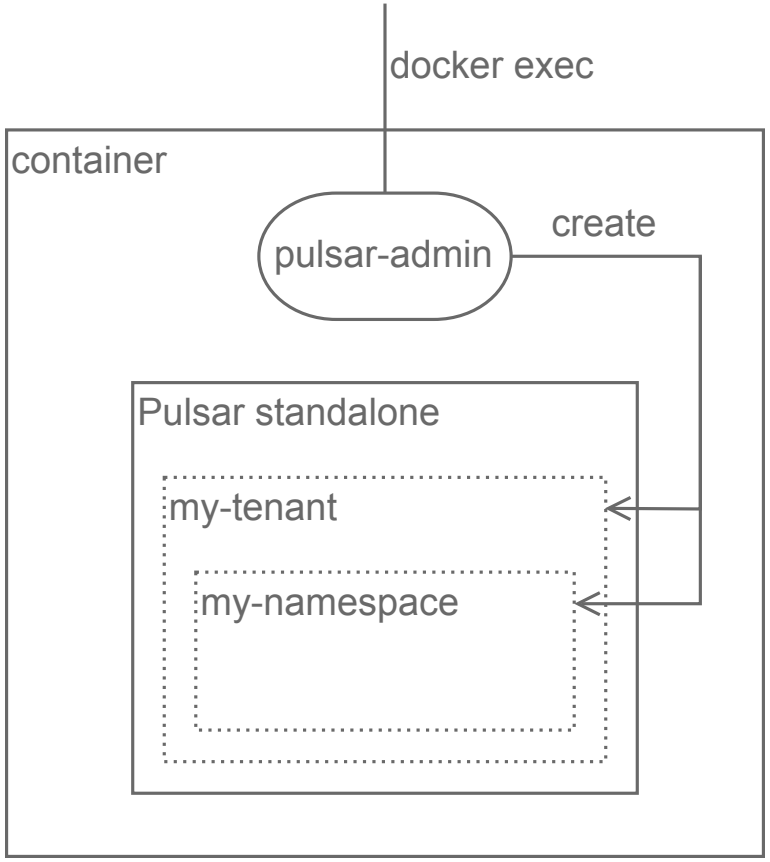
- *Pulsar standalone*
  - 単一の process で broker, bookie, zookeeper などを立てられる mode
  - local 環境での開発に便利 (production 用ではない)
- 本章の目標:
  Docker 上で Pulsar を standalone mode で起動する

Pulsar standalone を起動する

```
# docker を起動しておきます
# memory 割当を設定しておきます

# 配布資料の directory 内で進めます
$ cd handson

# 事前準備で用意した file を確認します
$ ls connectors/pulsar-io-cassandra-2.9.1.nar

# Pulsar standalone で起動する container を立てます
$ docker-compose --compatibility -p handson up pulsar

# (別 temrinal) container を確認します
$ docker container list

CONTAINER ID   IMAGE                              ...   NAMES
cd1a202c1859   apachepulsar/pulsar:2.9.1          ...   handson_pulsar_1
```

pulsar-admin で tenant と namespace を作る

- *pulsar-admin*: instance の tenant, namespace, topic と言った情報を管理する interface
- tenant と namespace を作成する

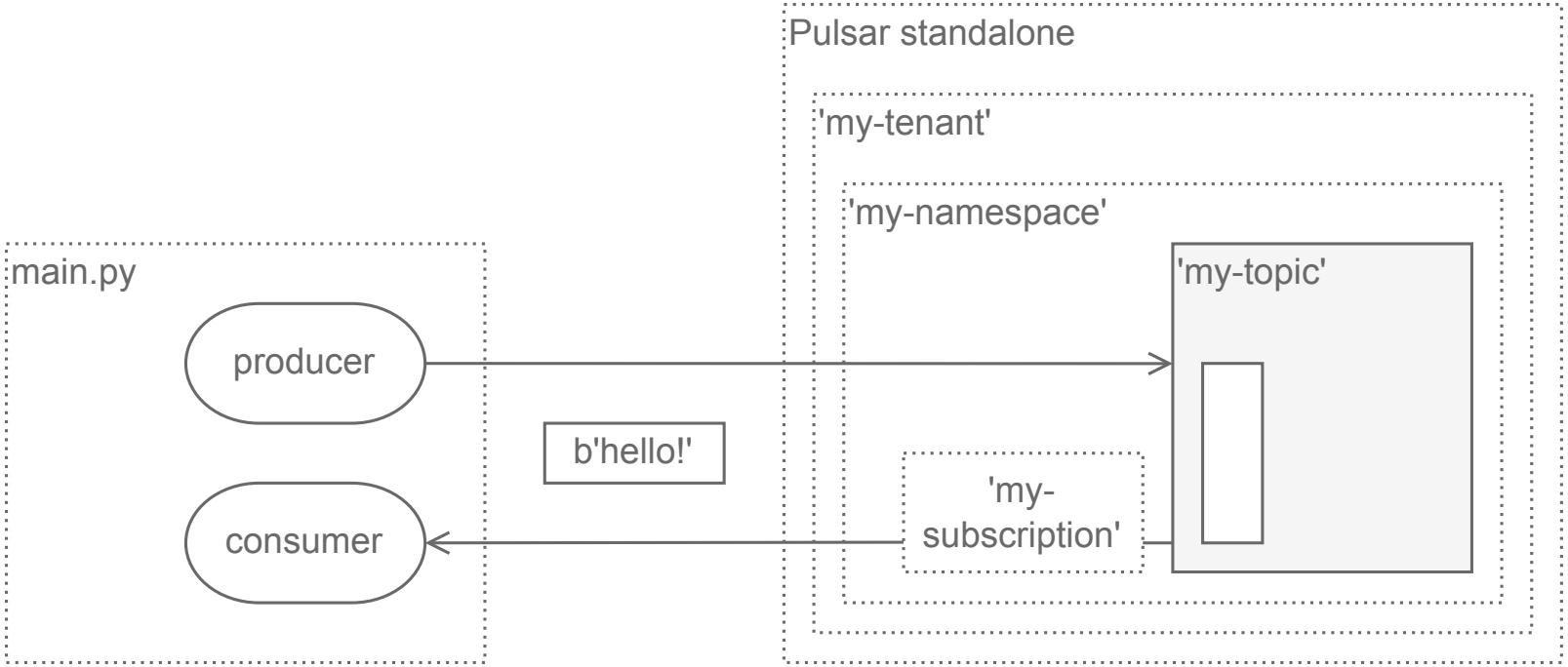


```
# 立てた container の terminal に入る
$ docker exec -it handson_pulsar_1 /bin/bash

# tenant を作成
$ bin/pulsar-admin tenants create my-tenant

# 'my-tenant' の存在を確認
$ bin/pulsar-admin tenants list

"my-tenant"
"public"
"pulsar"
"sample"
```

```
# namespace を作成
$ bin/pulsar-admin namespaces create my-tenant/my-namespace

# 確認
$ bin/pulsar-admin namespaces list my-tenant

"my-tenant/my-namespace"

$ exit
```

Python で produce, consumeを 試す

- client library を利用して produce, consume をする
- Java, C++, C#, Node, Go, Python 向けなどが存在
    - 今回は Python を採用



```
# 配布した`codes/main.py`を穴埋めしましょう
# editor は何でも構いません
$ vim codes/main.py
```

```
import pulsar
import logging

# log level を設定
# client を作成
logger = logging.getLogger('pulsar')
logger.setLevel('INFO')
client = pulsar.Client('pulsar://pulsar:6650', logger=logger)
```

consumer, producer 作成, 送信

```
# 先程作った tenant と namespace 上の topic に subscribe する consumer を作る
# topic は自動で作られる
topic = 'persistent://my-tenant/my-namespace/my-topic'
consumer = client.subscribe(topic, 'my-subscription')

# 同 topic を指定し producer を作る
producer = client.create_producer(topic)
# byte 列として message を送信する
producer.send('hello!'.encode())
```

受信

```
# message を受信する
message = consumer.receive()
print(f'id: {message.message_id()}, data: {message.value().decode()}')

# acknowledge（受信した事を broker に通知）する
# これをしないと subscription は次の message に進まない（設定によっては再送される）
consumer.acknowledge(message)

# 最後に client, producer, consumer を閉じる
# 閉じないと program 終了後も接続が残る
producer.close()
consumer.close()
client.close()
```

実行

```
$ cd handson

# client 用の container を起動
$ docker-compose run pulsar-client bash
$ pwd
/pulsar

# terminal に入ると、/pulsar/codes に今書いた code が有ります
# 実行
$ python codes/main.py
```

```
# consumer が受信した message を出力する
id: (1236,2,-1,-1), data: hello!
```

時間が余ったら

- code を書き換えてみる
  - 例: 非同期 receive

```
def listener(consumer, message):
    # do something

consumer = client.subscribe(topic, subscription, message_listener=listener)
```

Pulsar Schema

Pulsar Schema

- Pulsar で送受信するのは基本的に byte 列
- *Pulsar Schema*:

  複雑な data を 送受信できる
  - library が serialize, deserialize してくれる
- Python, Java, C++, Go client で利用可能

schema 定義

- schema 定義として `codes/employee.py` が用意してある

```
from pulsar import schema

# 送受信したい class を定義
class Employee(schema.Record):
    id_ = schema.Integer()
    firstName = schema.String()
    lastName = schema.String()
    title = schema.String()

# JSON schema 化
employeeSchema = schema.JsonSchema(Employee)
```

code 修正

```
 import pulsar
 import logging
+from employee import Employee, employeeSchema

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

# schema は topic に紐付くので、別の topic にする
-topic = 'persistent://my-tenant/my-namespace/my-topic'
+topic = 'persistent://my-tenant/my-namespace/schema-topic'

# subscribe 時に schema を指定
-consumer = client.subscribe(topic, 'my-subscription')
+consumer = client.subscribe(
+    topic=topic, subscription_name='my-subscription', schema=employeeSchema)
```

```
 print('sending message')

# producer 作成時に schema を指定
-producer = client.create_producer(topic)
+producer = client.create_producer(topic=topic, schema=employeeSchema)

-producer.send('hello!'.encode())
+producer.send(Employee(id_=100, firstName='Taro',
+                        lastName='Yamada', title='Manager'))

 print('receiving message')

 message = consumer.receive()
# .data() でなく .value()
-print(f'id: {message.message_id()}, data: {message.value().decode()}')
+print(message.value())
+print(f'firstName: {message.value().firstName}')
 consumer.acknowledge(message)
```

実行

```
# 先と同様に実行する
$ python codes/main.py

{'_required_default': False, '_default': None, '_required': False,
    'id_': 100, 'firstName': 'Taro', 'lastName': 'Yamada', 'title': 'Manager'}
firstName: Taro
```

時間が余ったら

```
$ docker exec -it handson_pulsar_1 /bin/bash

# 登録された schema を確認
$ bin/pulsar-admin schemas get persistent://my-tenant/my-namespace/schema

# 削除
$ bin/pulsar-admin schemas delete persistent://my-tenant/my-namespace/schema
# 別の schema を登録できる
```

または

```
# schema に要素を追加する
Boolean, Double, Bytes, Array, Map, ...
# 送受信を試す場合、別の topic にする
```

next

- Pulsar Functions
- Pulsar IO