云存储应用技术

第八章：消息队列

丁烨
dingye@dgut.edu.cn
网络空间安全学院

2019-11-28

東莞理工學院
DONGGUAN UNIVERSITY OF TECHNOLOGY

目录

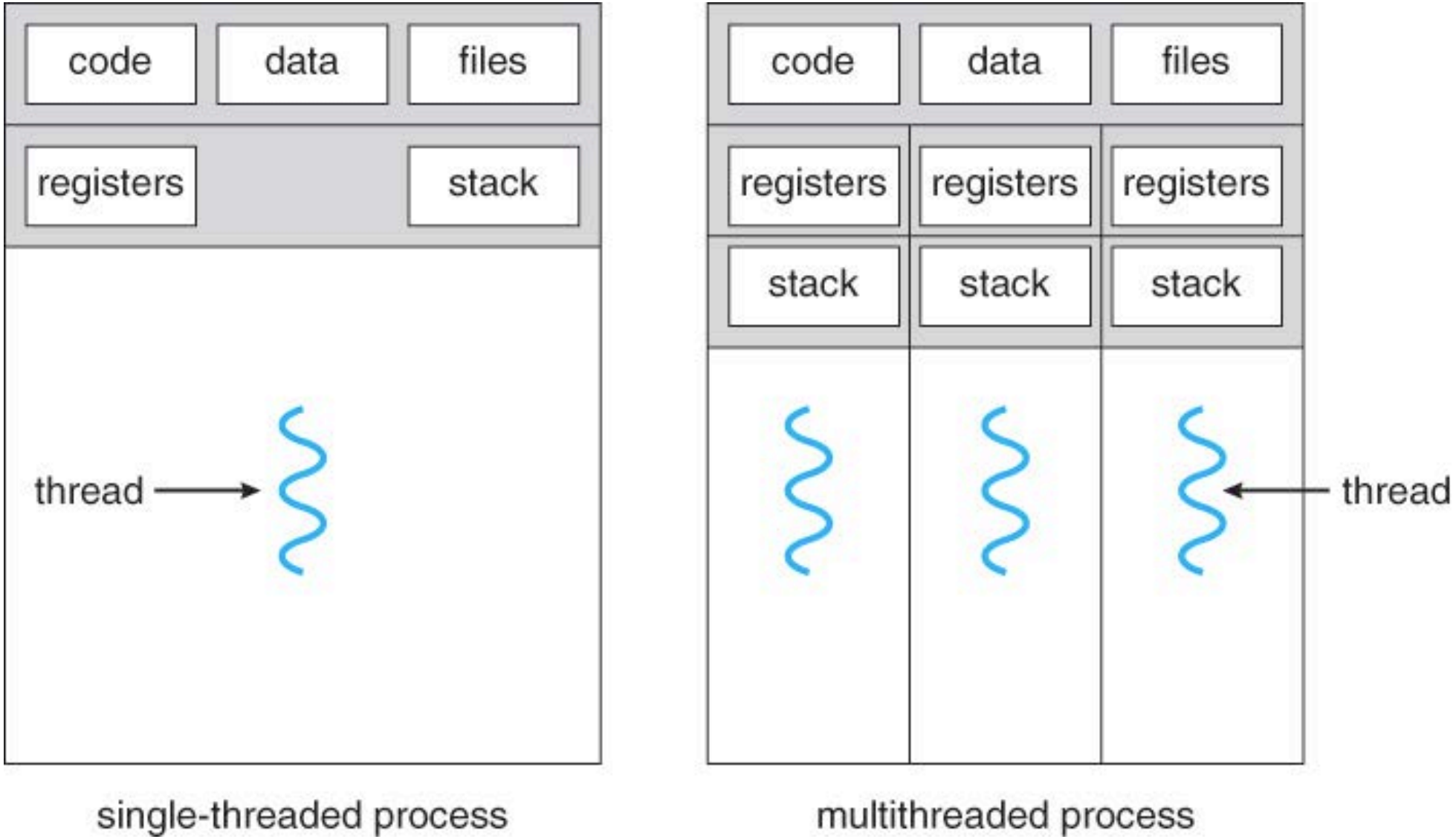
消息队列概述

消息队列的定义

- 消息队列（Message Queue）
- 一种进程（Process）间通信或同一进程的不同线程（Thread）间的通信方式
- 消息队列的使用者分为消息的发送者（Producer / Publisher）和接收者（Consumer / Subscriber）
- 发送者将消息发送到消息队列
- 接收者从消息队列中接收消息
- 消息队列提供了异步（Asynchronous）的通信协议，也就是说：消息的发送者和接收者不需要同时与消息队列交互
- 消息会保存在队列中，直到接收者取回它
- （大部分）消息队列会维护接收者接收的进度和状态

消息队列概述

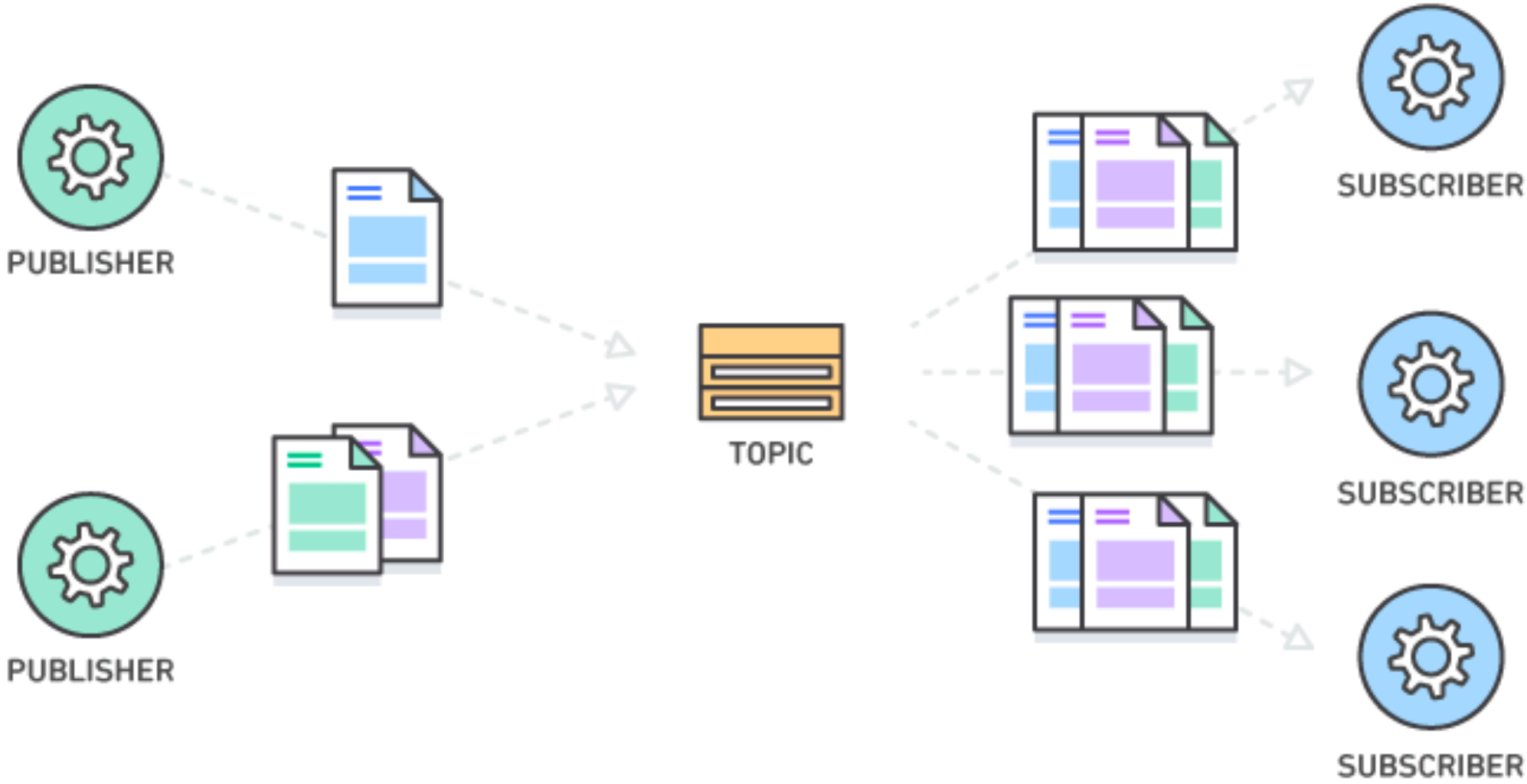
消息队列的定义



single-threaded process　　multithreaded process

消息队列概述

消息队列的定义

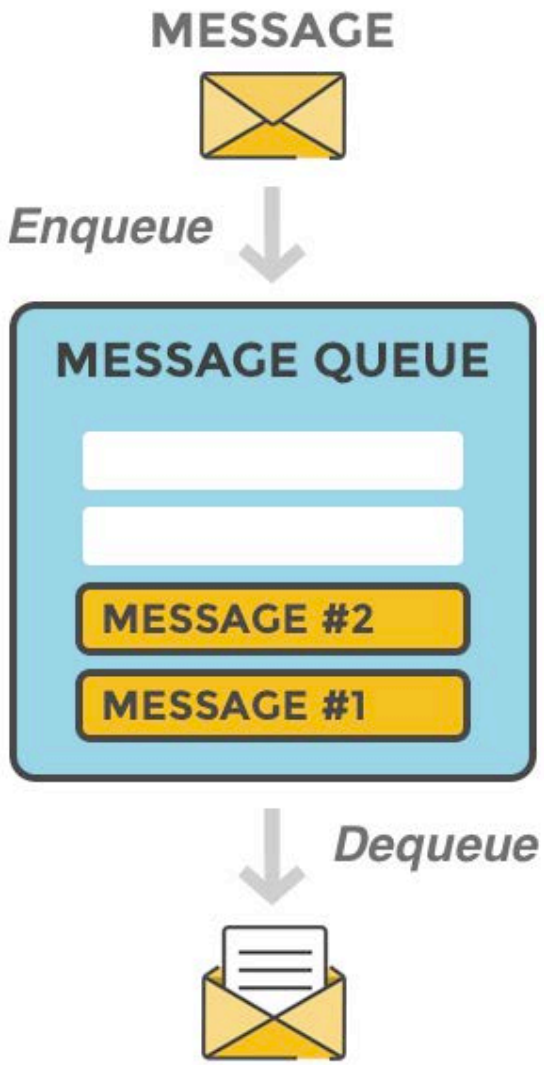


消息队列概述

消息队列的特点

- 消息队列中，通常有两部分参与方：
- 发送者（Producer / Publisher）
- 产生数据，并将数据存放在指定的话题（Topic）中
- 接收者（Consumer / Subscriber）
- 从消息队列中订阅（Subscribe）并接收消息

- 发送者可以是多个进程或线程
- 接收者也可以是多个进程或线程
- 话题可能不止一个，且一个话题也可能采用分布式存储



消息队列概述

消息队列的特点

- 发送者可以是多个进程或线程
- 接收者也可以是多个进程或线程
- 话题可能不止一个，且一个话题也可能采用分布式存储

- 一个话题中包含不同发送者发送的消息
- 一个发送者也可以往不同的话题中发送消息

- 一个接收者可以同时订阅多个话题
- 一个进程或线程可能既是发送者，也是接收者

- 不同接收者看到的消息是一样的，但是同步进度可能不一样

消息队列概述

消息队列的应用场景

- 消息队列最常见的应用：聊天软件
- 数据实时性要求高
- 数据的发送者 / 接收者情况复杂
- 数据不需要持久保存

- 群聊（Group Chat）在很大程度上依赖消息队列技术来实现

消息队列概述

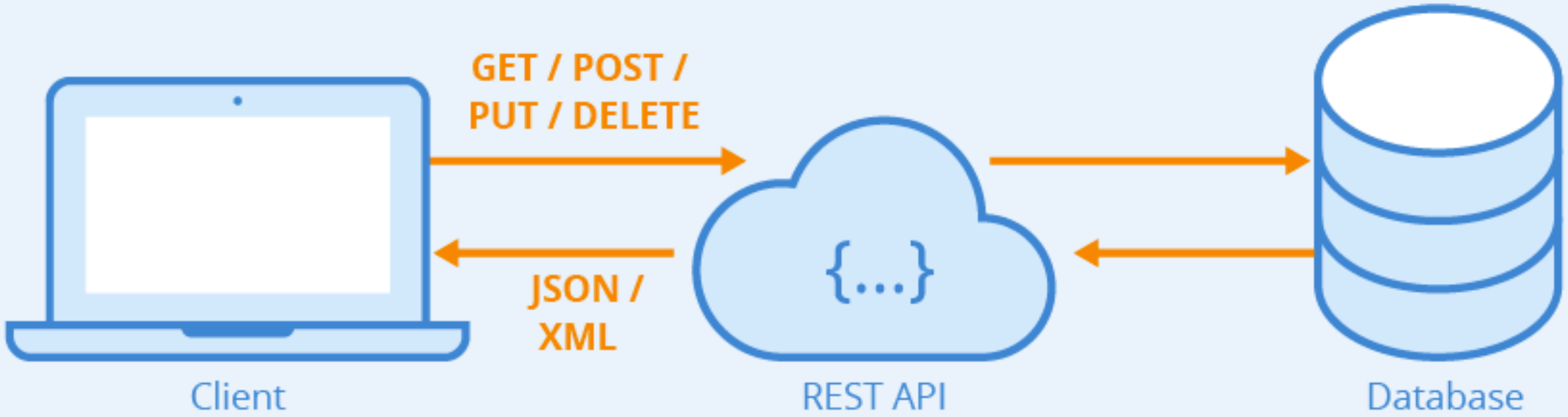
消息队列的应用场景

- 消息队列也常常用在异步任务管理中
- 尤其是基于 REST API 的互联网应用

- 表现层状态转换（Representational State Transfer，REST）
- 一种万维网软件架构风格，目的是便于不同软件 / 程序在网络（例如互联网）中互相传递信息
- 符合或兼容于这种架构风格（简称为 REST 或 RESTful）的网络服务，允许客户端发出以统一资源标识符访问和操作网络资源的请求，而与预先定义好的无状态操作集一致化
- 当前在三种主流的 Web 服务实现方案中，因为 REST 模式与复杂的 SOAP 和 XML-RPC 相比更加简洁，越来越多的 Web 服务开始采用 REST 风格设计和实现

消息队列概述

消息队列的应用场景



消息队列概述

消息队列的应用场景

- 由于 REST 的架构特点，导致后台开发和前端开发可以完全剥离开
- 且后台开发和前端开发可以采用完全不同的架构和开发语言
- 因此诞生了“API 工程师”、“前端工程师”、“iOS 工程师”、“Android 工程师”等独立的开发者职业

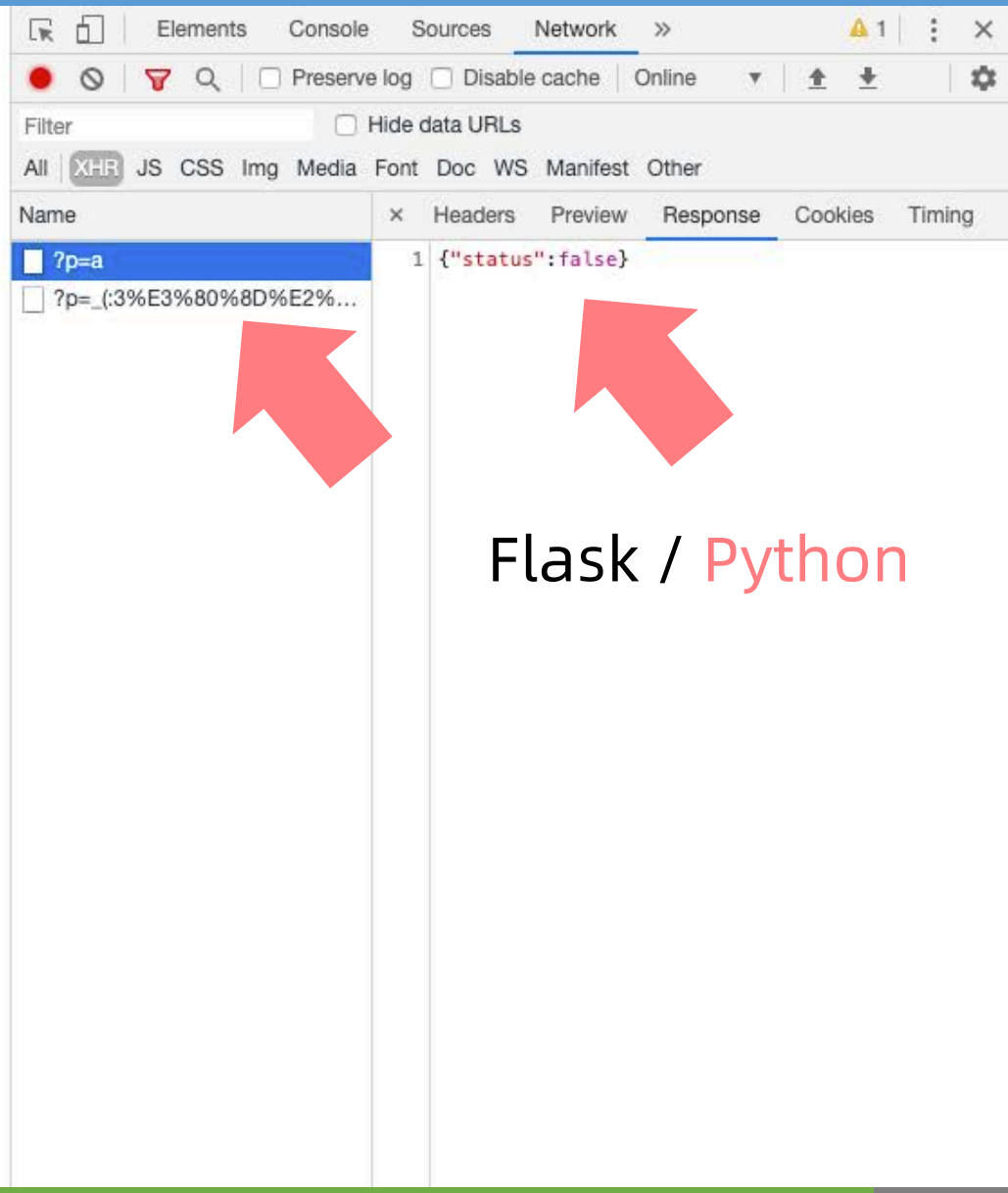
消息队列概述

消息队列的应用场景

https://www.avatarsys.org/deepella/login

(¯ェ¯;)

React.js / JavaScript



Flask / Python

```python
from flask import Flask
app = Flask(__name__)


@app.route("/")
def hello():
    return "Hello World!"


if __name__ == "__main__":
    app.run()
```

消息队列概述

消息队列的应用场景

- REST API 由于需要直接给用户返回数据，因此不能长时间无响应
- 但是，如果处理的过程过于复杂，则无法做到立刻返回
- 此时，应当委托另一个（或一群）服务器处理数据，并先返回“处理中”的返回值
- 等另一个（或一群）服务器处理数据完毕后，再返回“已完成”的返回值

- 这个过程中，需要使用消息队列
- 使得主服务器（Master）与从服务器（Slave）可以沟通

❖ 消息队列主从 REST 服务架构工作流程：

主服务器（Master），一般只有一个

1. 收到用户创建任务的 API 请求
2. 于消息队列中创建“任务详情及其参数”的消息
3. 将此任务状态设置为“已提交”
4. 返回任务状态

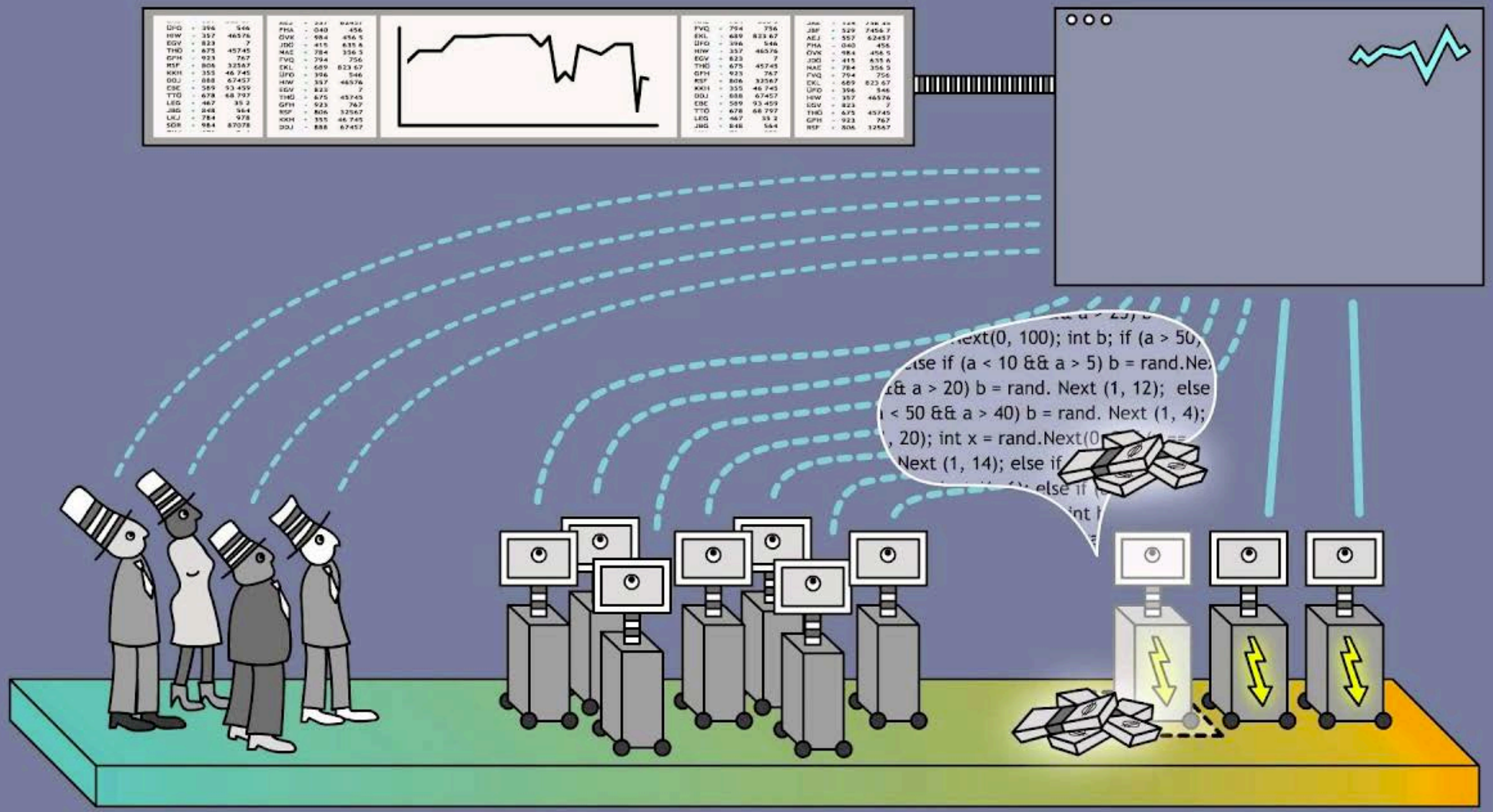
1. 收到用户查询任务状态的 API 请求
2. 返回任务状态

从服务器（Slave），一般有多个

1. 不断轮询消息队列中的任务列表，接收“任务详情及其参数”的消息
2. 确认该任务的状态是否为“已提交”，如果不是，则返回步骤 1，否则继续
3. 将此任务状态设置为“处理中”
4. 处理此任务
5. 处理完毕后，将此任务状态设置为“已完成”
6. 返回步骤 1

消息队列概述

消息队列的应用场景

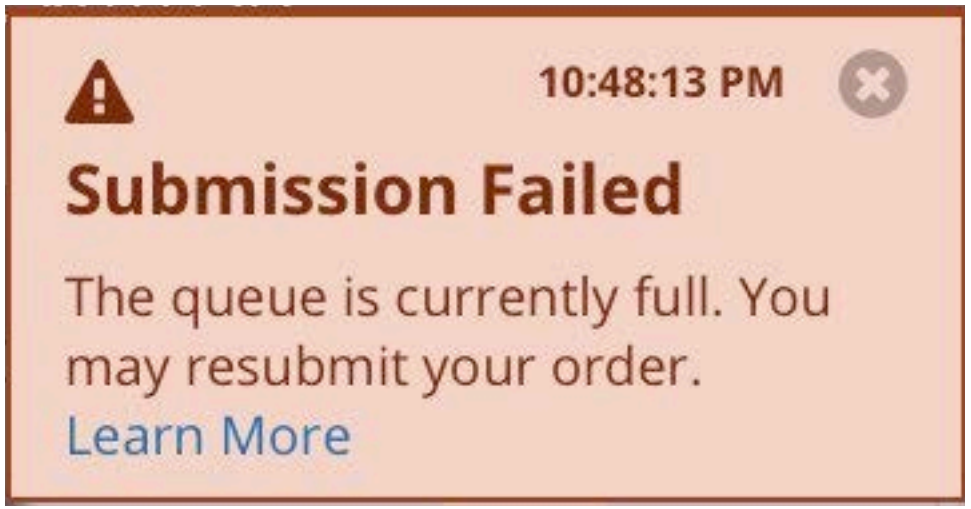


消息队列概述

消息队列的应用场景

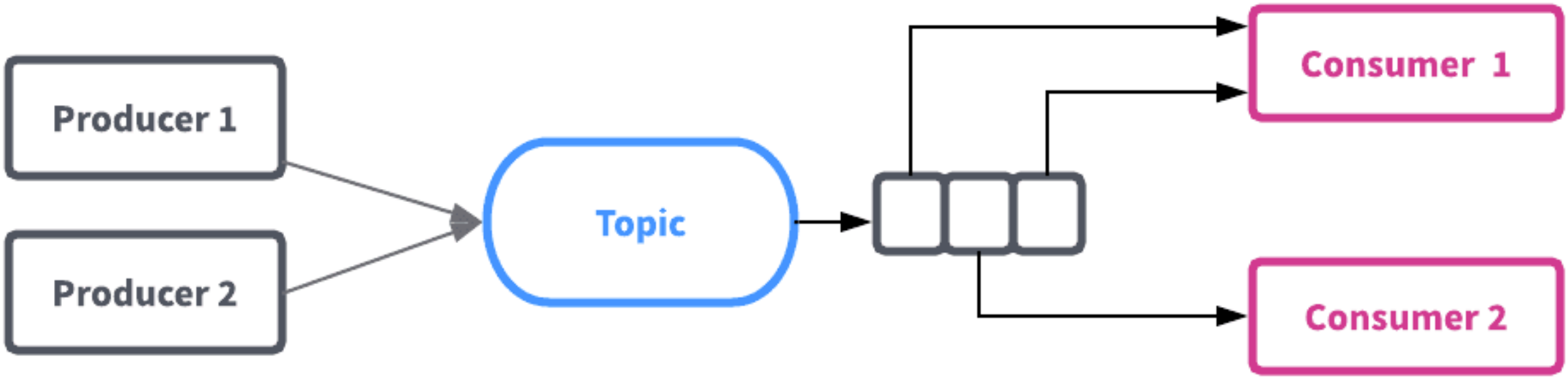
- 高频交易（High-frequency Trading，HFT）
- 是指从那些人们无法利用的、极为短暂的市场变化中寻求获利的自动化程序交易，例如某种证券买入价和卖出价差价的微小变化，或者某只股票在不同交易所之间的微小价差
- 这种交易的速度如此之快，以至于有些交易机构将自己的“服务器群组”安置到了离交易所的服务器很近的地方，以缩短交易指令通过光缆以光速传送的时间
- 一般是以电脑买卖盘程式进行非常高速的证券交易，从中赚取证券买卖价格的差价

- 高频交易会产生大量“买入（Bid）”、“卖出（Ask）”交易消息，因此需要性能极高的消息队列平台



消息队列概述

消息队列总结



消息队列概述

消息队列总结

- 消息队列系统或平台的基本要求
- 可靠（Durability）：不能无故丢失消息
- 安全（Security Policies）：需要保证消息的安全性，并提供安全机制
- 消息销毁机制（Message Purging Policies）：消息需要有有效期（Time-to-live）
- 消息过滤（Message Filtering）：需要有“话题”或类似的机制
- 消息同步进度（Delivery Policies）：需要记录接收者同步的进度
- 批量处理（Batching Policies）：消息可以被批量推送给接收者
- 排队机制（Queuing Criteria）：消息需要按照某种顺序保存
- 确认机制（Receipt Notification）：消息可以由接收者确认收到，而非消息队列确认

消息队列概述

消息队列总结

- 常见的消息队列系统或软件包（Library）
- RabbitMQ
- Apache Kafka
- Apache Pulsar
- Celery

- 大部分云计算平台都提供消息队列的服务
- Amazon Simple Queue Service
- Google Cloud Pub / Sub
- 阿里云消息队列 MQ

目录

RabbitMQ 简介

- RabbitMQ
- https://www.rabbitmq.com/

- RabbitMQ 是实现了高级消息队列协议（AMQP）的一个开源消息队列系统
- RabbitMQ 服务器端是用 Erlang 语言编写的
- 大部分主流编程语言均有 RabbitMQ 的客户端，包括：Python、Java、Ruby、PHP、C#、JavaScript、Go、Elixir、Objective-C、Swift 等

- RabbitMQ 是目前市场占有率最大的消息队列系统

安装 RabbitMQ

- Ubuntu 18.04
- 安装 RabbitMQ：
- `sudo apt install rabbitmq-server`

- 启动 RabbitMQ（服务模式）：
- `sudo service rabbitmq-server start`
- 或（用户模式）：
- `rabbitmq-server`

安装 RabbitMQ

- macOS Catalina
- 安装 RabbitMQ：
- `brew install rabbitmq`

- 启动 RabbitMQ：
- `rabbitmq-server`
- 如果需要 RabbitMQ 在后台运行，可以使用 screen

安装 RabbitMQ

- Windows 10（不推荐）
- 安装 RabbitMQ：
- https://github.com/rabbitmq/rabbitmq-server/releases

- 安装完毕后，RabbitMQ 的服务会在后台自动运行，如果需要关闭或重新启动，可以使用 Windows 的“系统服务”功能管理

- 大部分消息队列系统都不支持 Windows，RabbitMQ 是比较少见支持 Windows 平台的消息队列系统，但 Windows 下性能较差，不推荐使用

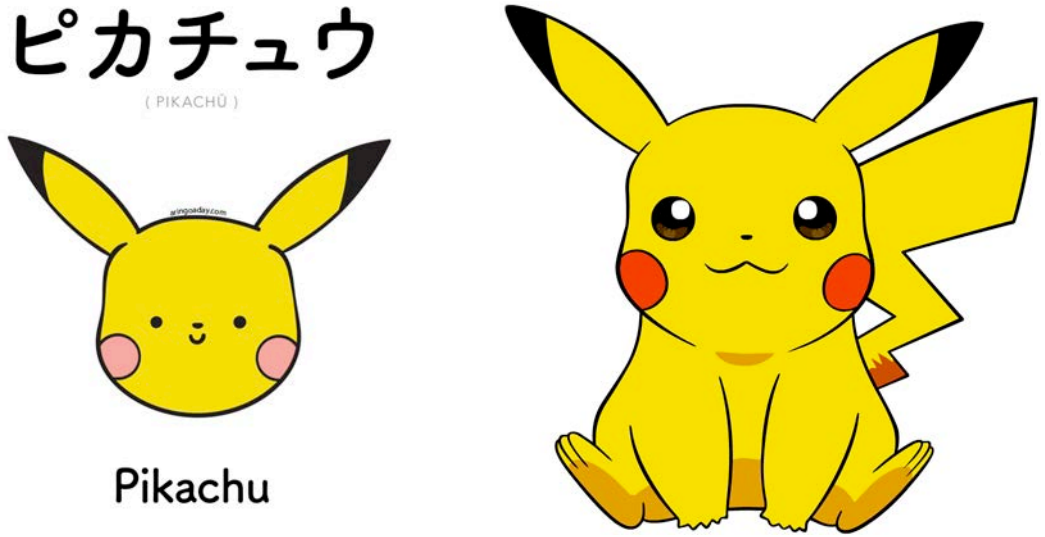
- RabbitMQ 客户端
- 大部分主流编程语言均有 RabbitMQ 的客户端，包括：Python、Java、Ruby、PHP、C#、JavaScript、Go、Elixir、Objective-C、Swift 等
- Python 客户端（Pika）：https://github.com/pika/pika
- Ruby 客户端（Bunny）：http://rubybunny.info/
- Go 客户端：http://godoc.org/github.com/streadway/amqp
- JavaScript 客户端：http://www.squaremobius.net/amqp.node/

- 本课程采用 Python 作为例子

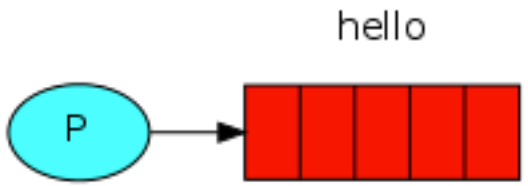
使用 RabbitMQ

- Pika（RabbitMQ Python 客户端）
- https://github.com/pika/pika

- `sudo apt install python3-pip`
- `pip3 install --user -U pika`

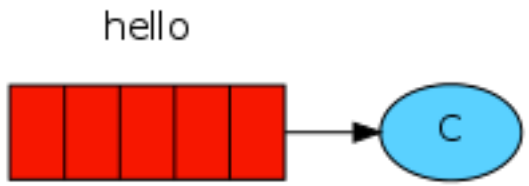


使用 RabbitMQ



- 发送端（Producer / Publisher）
- `import pika`
- `connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))`
- `channel = connection.channel()`
- `channel.queue_declare(queue='hello')`
- `channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')`
- `print(" [x] Sent 'Hello World!'")`
- `connection.close()`

使用 RabbitMQ



- 接收端（Consumer / Subscriber）

```
import pika
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
print(' [*] Waiting for messages. To exit press CTRL+C')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
channel.basic_consume(queue='hello', auto_ack=True, on_message_callback=callback)
channel.start_consuming()
```

使用 RabbitMQ

```
~ » python3 producer.py
 [x] Sent 'Hello World!'
```

```
~ » python3 subscriber.py
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello World!'
```

- RabbitMQ 教程
- https://www.rabbitmq.com/getstarted.html

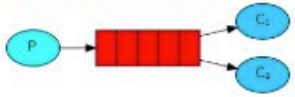
1 "Hello World!"

The simplest thing that does *something*

- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

2 Work queues

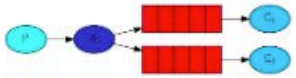
Distributing tasks among workers (the competing consumers pattern)



- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

3 Publish/Subscribe

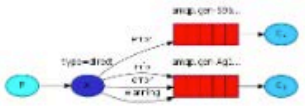
Sending messages to many consumers at once



- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

4 Routing

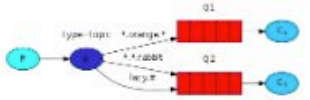
Receiving messages selectively



- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

5 Topics

Receiving messages based on a pattern (topics)



- Python
- Java
- Ruby
- PHP
- C#
- JavaScript
- Go
- Elixir
- Objective-C
- Swift
- Spring AMQP

- Understanding RabbitMQ Webinar
- https://content.pivotal.io/webinars/dec-12-understand-rabbitmq-for-developers-and-operators-webinar

目录

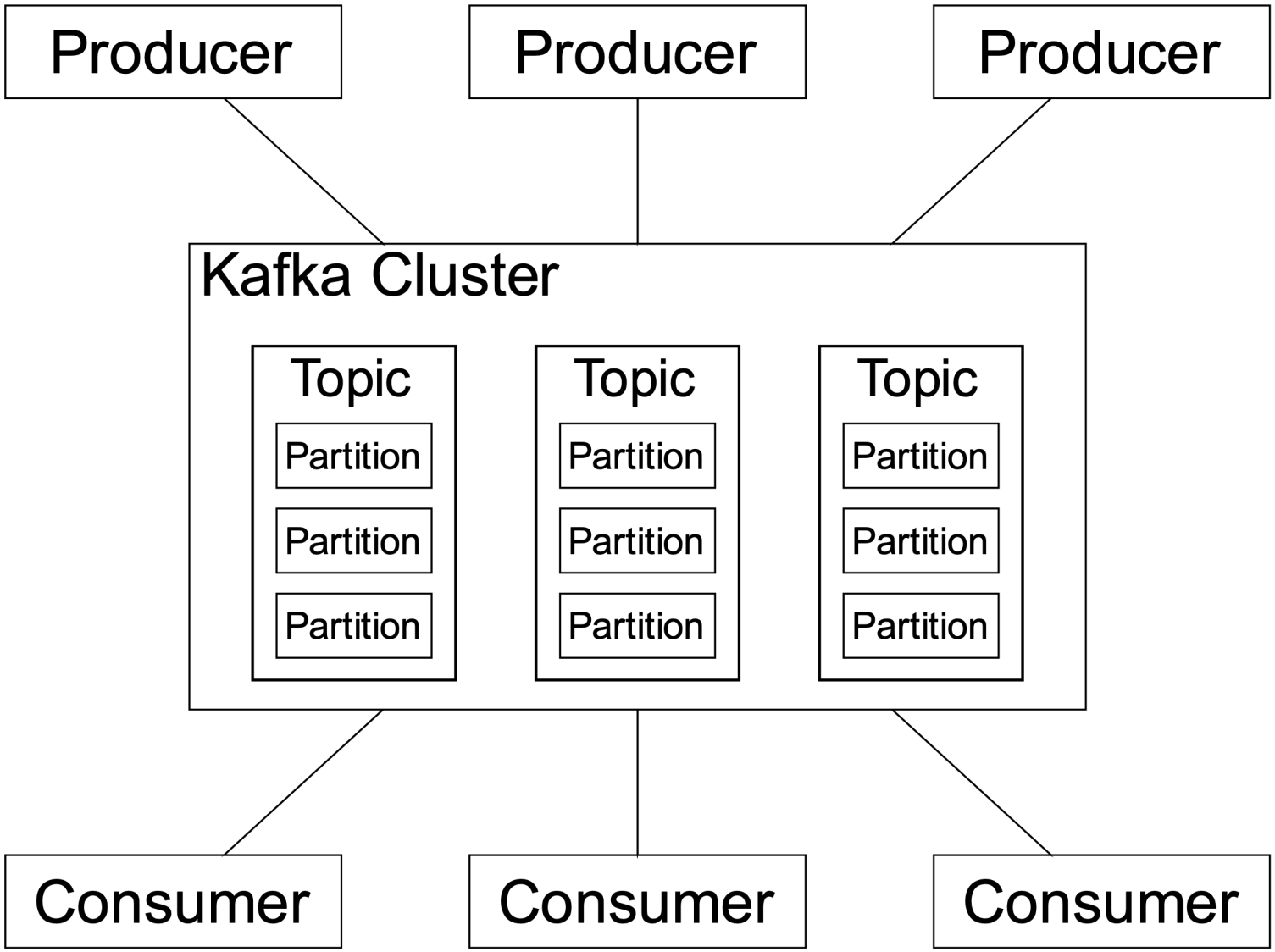
Apache Kafka 简介

- Apache Kafka
- http://kafka.apache.org/

- Kafka 是一个最初由领英（LinkedIn）开发，目前由 Apache 软件基金会管理的一个开源流处理平台，由 Scala 和 Java 编写
- 该项目的目标是为处理实时数据提供一个统一、高吞吐、低延迟的平台
- 其持久化层本质上是一个“按照分布式事务日志架构的大规模发布 / 订阅消息队列”
- 此外，Kafka 可以通过 Kafka Connect 连接到外部系统（用于数据输入 / 输出）

- Kafka 的性能远超 RabbitMQ
- 但是由于架构问题，目前 Kafka 已经基本上被 Pulsar 取代

Apache Kafka 简介



安装 Apache Kafka

- 下载 Kafka
- https://www.apache.org/dyn/closer.cgi?path=/kafka/2.3.0/kafka_2.12-2.3.0.tgz

- Kafka 基于 Scala 和 Java 开发，因此天然跨平台
- 由于使用了 Java 虚拟机，Kafka 无需编译，因此也无需安装

安装 Apache Kafka

- 启动 Kafka
- tar -xzf kafka_2.12-2.3.0.tgz
- cd kafka_2.12-2.3.0

- screen
- bin/zookeeper-server-start.sh config/zookeeper.properties

- screen
- bin/kafka-server-start.sh config/server.properties

- 注意：由于 Kafka 需要使用 Apache ZooKeeper，因此需要分别启动两个服务

安装 Apache Kafka

- Apache ZooKeeper
- http://zookeeper.apache.org/
- ZooKeeper 是 Apache 软件基金会的一个软件项目，它为大型分布式计算提供开源的分布式配置服务、同步服务和命名注册
- ZooKeeper 曾经是 Hadoop 的一个子项目，但现在是一个独立的顶级项目
- ZooKeeper 的架构通过冗余服务实现高可用性，因此，如果第一次无应答，客户端就可以询问另一台 ZooKeeper 主机，ZooKeeper 节点将它们的数据存储于一个分层的命名空间，非常类似于一个文件系统或一个前缀树结构
- 简单来讲，ZooKeeper 可以确保分布式服务有序，无故障的运转

使用 Apache Kafka

- Kafka 有着方便的命令行接口（CLI），使用方法如下：

- 创建话题：
- `bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test`

- 查看话题列表：
- `bin/kafka-topics.sh --list --bootstrap-server localhost:9092`

使用 Apache Kafka

- 发送端（Producer / Publisher）
- bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

- 使用该工具后，可以输入消息内容，以回车分隔多个消息
- 输入 EOF（Ctrl + D）可以结束消息的输入
- 例如，我们可以输入以下两条信息：
- This is a message
- This is another message

- 接收端（Consumer / Subscriber）
- bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

- 如果执行正确，那么可以接收到刚才输入的两条信息：
- This is a message
- This is another message

- Kafka 支持分布式架构，详情可参考：
- http://kafka.apache.org/quickstart
- http://kafka.apache.org/documentation/

- Kafka 的性能分析：
- https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

- Kafka 由于架构问题，不同的客户端性能并不一样
- 官方 Python 客户端（不推荐）：
- https://github.com/dpkp/kafka-python

- Confluent 版本的 Python 客户端（推荐）：
- https://github.com/confluentinc/confluent-kafka-python
- Confluent 版本的 Kafka 客户端性能要好很多
- 由于使用了部分可执行库，该版本在 Windows 中编译十分困难，因此不建议在 Windows 中使用

目录

Apache Pulsar 简介

- Apache Pulsar
- https://pulsar.apache.org/

- Pulsar 是一个最初由雅虎（Yahoo）开发，目前由 Apache 软件基金会管理的一个开源流处理平台，主要由 Java 编写
- Pulsar 和 Kafka 的定位比较相似，目标都是为处理实时数据提供一个统一、高吞吐、低延迟的平台

- Pulsar 的可扩展性（不需要 ZooKeeper）、性能都比 Kafka 略胜一筹
- 目前 Kafka 已经基本上被 Pulsar 取代

Apache Pulsar 简介

- Pulsar 的意思是“脉冲星”

YAHOO! JAPAN

Tencent 腾讯

安装 Apache Pulsar

- 下载 Pulsar
- https://archive.apache.org/dist/pulsar/pulsar-2.4.1/apache-pulsar-2.4.1-bin.tar.gz

- Pulsar 主要基于 Java 开发，因此天然跨平台
- 由于使用了 Java 虚拟机，Pulsar 无需编译，因此也无需安装

安装 Apache Pulsar

- 启动 Pulsar
- tar xvfz apache-pulsar-2.4.1-bin.tar.gz
- cd apache-pulsar-2.4.1
- bin/pulsar standalone

- 如果启动正确，会看到类似下面的输出：

```
2017-06-01 14:46:29,192 - INFO  - [main:WebSocketService@95] - Configuration Store cache started
2017-06-01 14:46:29,192 - INFO  - [main:AuthenticationService@61] - Authentication is disabled
2017-06-01 14:46:29,192 - INFO  - [main:WebSocketService@108] - Pulsar WebSocket Service started
```

使用 Apache Pulsar

- 类似 Kafka，Pulsar 也提供了 CLI，使用方法如下：

- 发送端（Producer / Publisher）
- `bin/pulsar-client produce my-topic --messages "hello-pulsar"`

- 接收端（Consumer / Subscriber）
- `bin/pulsar-client consume my-topic -s "first-subscription"`

使用 Apache Pulsar

- 类似 RabbitMQ，Pulsar 也提供了支持不同语言的客户端，主要包括：
- Java：https://pulsar.apache.org/docs/en/client-libraries-java
- Go：https://pulsar.apache.org/docs/en/client-libraries-go
- Python：https://pulsar.apache.org/docs/en/client-libraries-python
- C++：https://pulsar.apache.org/docs/en/client-libraries-cpp

- 本课程采用 Python 作为例子

- Apache Pulsar Python 客户端
- https://pypi.python.org/pypi/pulsar-client

- `sudo apt install python3-pip`
- `pip3 install --user -U pulsar-client`

使用 Apache Pulsar

- 发送端（Producer / Publisher）

```
import pulsar
client = pulsar.Client('pulsar://localhost:6650')

producer = client.create_producer('my-topic')
for i in range(10):
    producer.send(('Hello-%d' % i).encode('utf-8'))
client.close()
```

使用 Apache Pulsar

- 接收端（Consumer / Subscriber）

```
consumer = client.subscribe('my-topic', 'my-subscription')
while True:
    msg = consumer.receive()
    try:
        print("Received '{}' id='{}'".format(msg.data(), msg.message_id()))
        consumer.acknowledge(msg)
    except:
        consumer.negative_acknowledge(msg)
client.close()
```

使用 Apache Pulsar

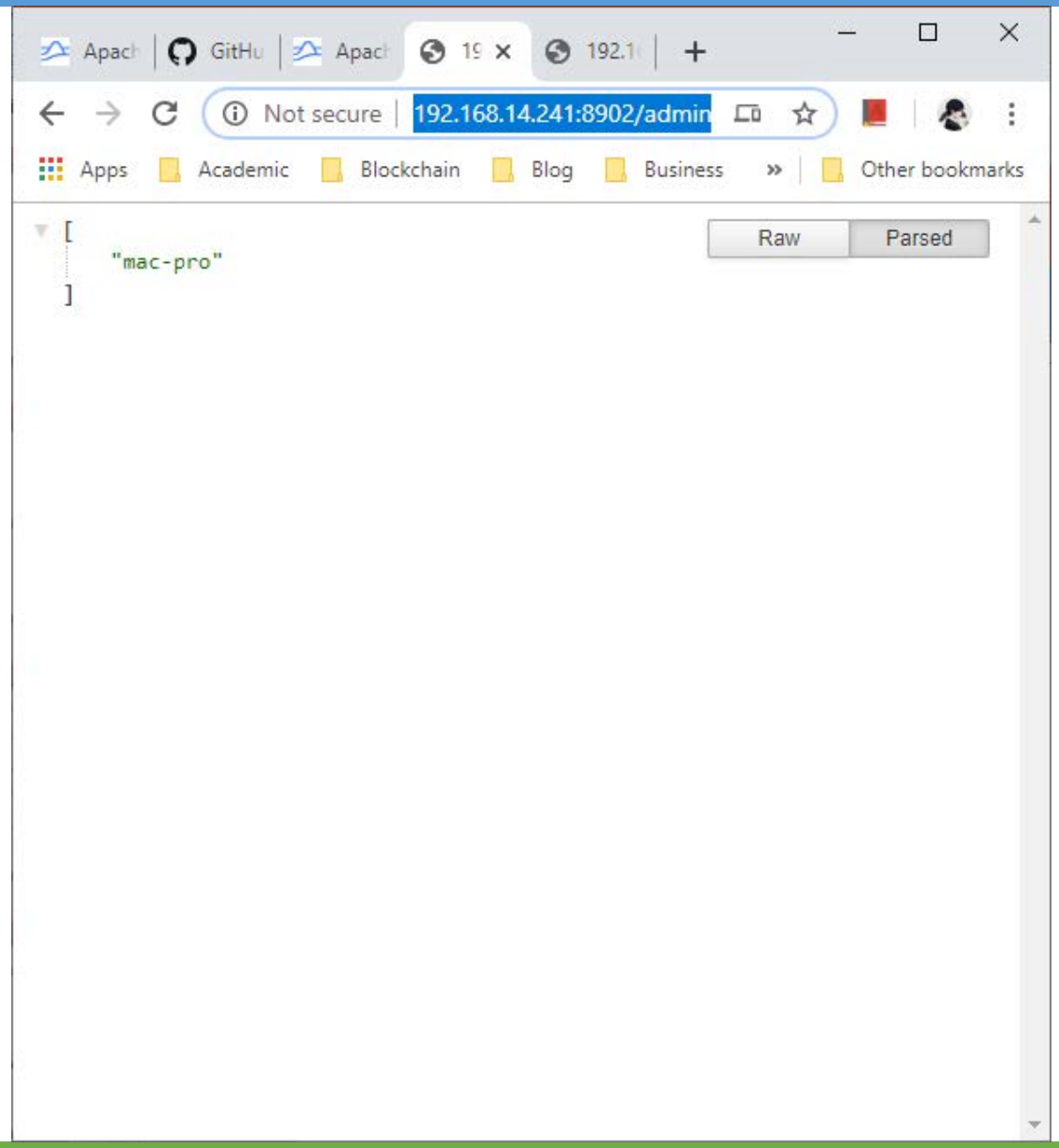
..orkspace/test

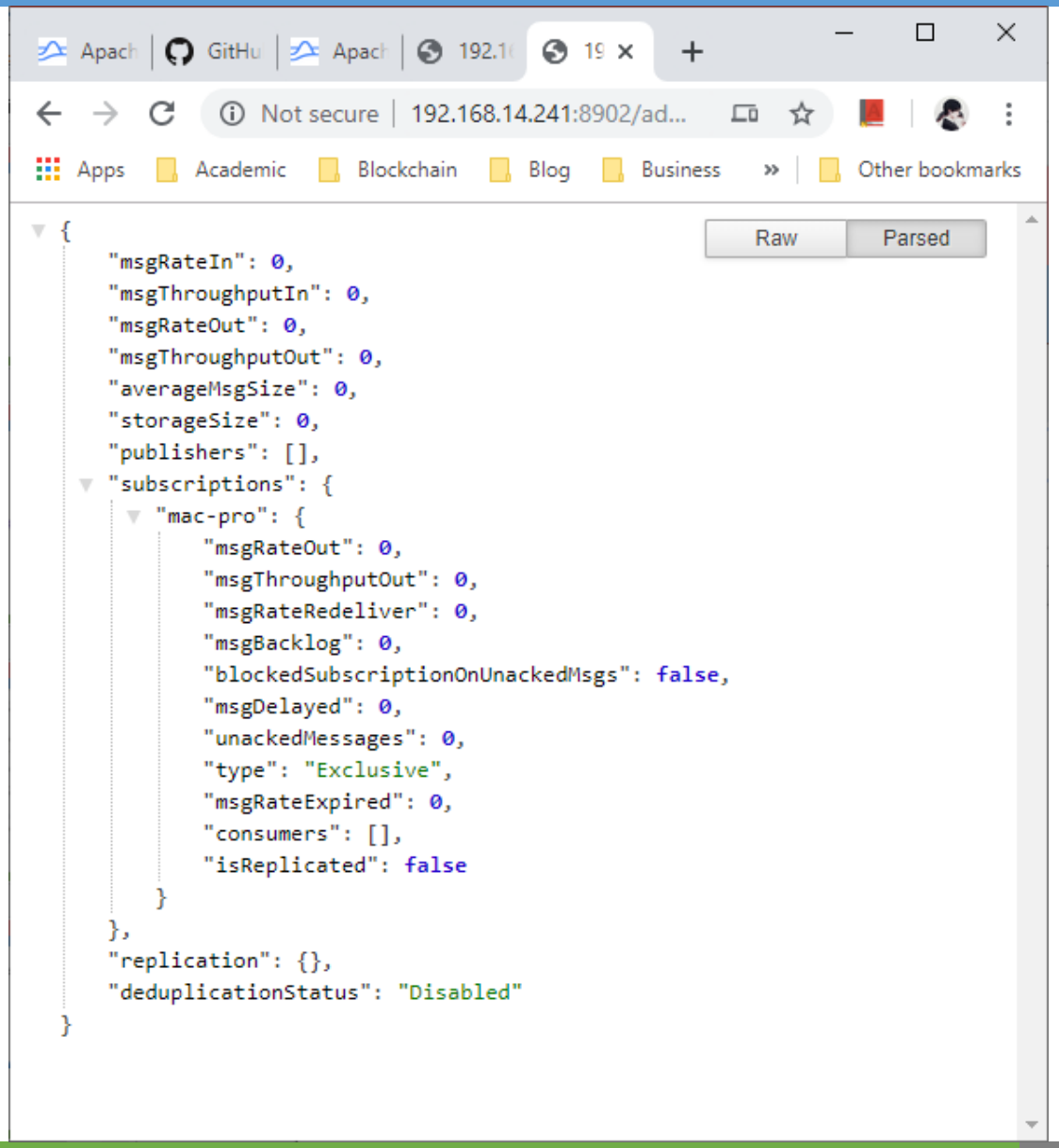
```
~/Workspace/test » python3 producer.py                                                    valency@ThinkPad-T450s
2019-11-25 22:27:53.428 INFO  ConnectionPool:72 | Created connection for pulsar://192.168.14.241:8901
2019-11-25 22:27:53.438 INFO  ClientConnection:324 | [10.8.0.102:8753 -> 192.168.14.241:8901] Connected to broker
2019-11-25 22:27:53.571 INFO  HandlerBase:52 | [persistent://public/default/my-topic, ] Getting connection from pool
2019-11-25 22:27:53.591 INFO  ConnectionPool:72 | Created connection for pulsar://pulsar:6650
2019-11-25 22:27:53.617 INFO  ClientConnection:326 | [10.8.0.102:8754 -> 192.168.14.241:8901] Connected to broker through proxy. Logical broker: pulsar:/
/pulsar:6650
2019-11-25 22:27:53.757 INFO  ProducerImpl:155 | [persistent://public/default/my-topic, ] Created producer on broker [10.8.0.102:8754 -> 192.168.14.241:8
901]
2019-11-25 22:27:53.922 INFO  ClientImpl:482 | Closing Pulsar client
2019-11-25 22:27:53.923 INFO  ProducerImpl:483 | [persistent://public/default/my-topic, standalone-2-38] Closing producer for topic persistent://public/d
efault/my-topic
2019-11-25 22:27:53.937 INFO  ProducerImpl:521 | [persistent://public/default/my-topic, standalone-2-38] Closed producer
~/Workspace/test » python3 consumer.py                                                    valency@ThinkPad-T450s
2019-11-25 22:27:56.928 INFO  Client:88 | Subscribing on Topic :my-topic
2019-11-25 22:27:56.930 INFO  ConnectionPool:72 | Created connection for pulsar://192.168.14.241:8901
2019-11-25 22:27:56.941 INFO  ClientConnection:324 | [10.8.0.102:8755 -> 192.168.14.241:8901] Connected to broker
2019-11-25 22:27:57.057 INFO  HandlerBase:52 | [persistent://public/default/my-topic, my-subscription, 0] Getting connection from pool
2019-11-25 22:27:57.072 INFO  ConnectionPool:72 | Created connection for pulsar://pulsar:6650
2019-11-25 22:27:57.083 INFO  ClientConnection:326 | [10.8.0.102:8756 -> 192.168.14.241:8901] Connected to broker through proxy. Logical broker: pulsar:/
/pulsar:6650
2019-11-25 22:27:57.202 INFO  ConsumerImpl:170 | [persistent://public/default/my-topic, my-subscription, 0] Created consumer on broker [10.8.0.102:8756 -
> 192.168.14.241:8901]
Received 'b'Hello-0'' id='(30581,10,-1,-1)'
Received 'b'Hello-1'' id='(30581,11,-1,-1)'
Received 'b'Hello-2'' id='(30581,12,-1,-1)'
```

❖ Apache Pulsar REST API

- ❖ 除了命令行客户端（CLI）和编程客户端（SDK），Pulsar 亦提供了 REST API，因此可扩展性极强
- ❖ 例如，查询某个话题的接收端列表：
- ❖ http://localhost:8080/admin/v2/persistent/public/default/my-topic/subscriptions
- ❖ 查询某个话题的状态：
- ❖ http://localhost:8080/admin/v2/persistent/public/default/my-topic/stats

使用 Apache Pulsar

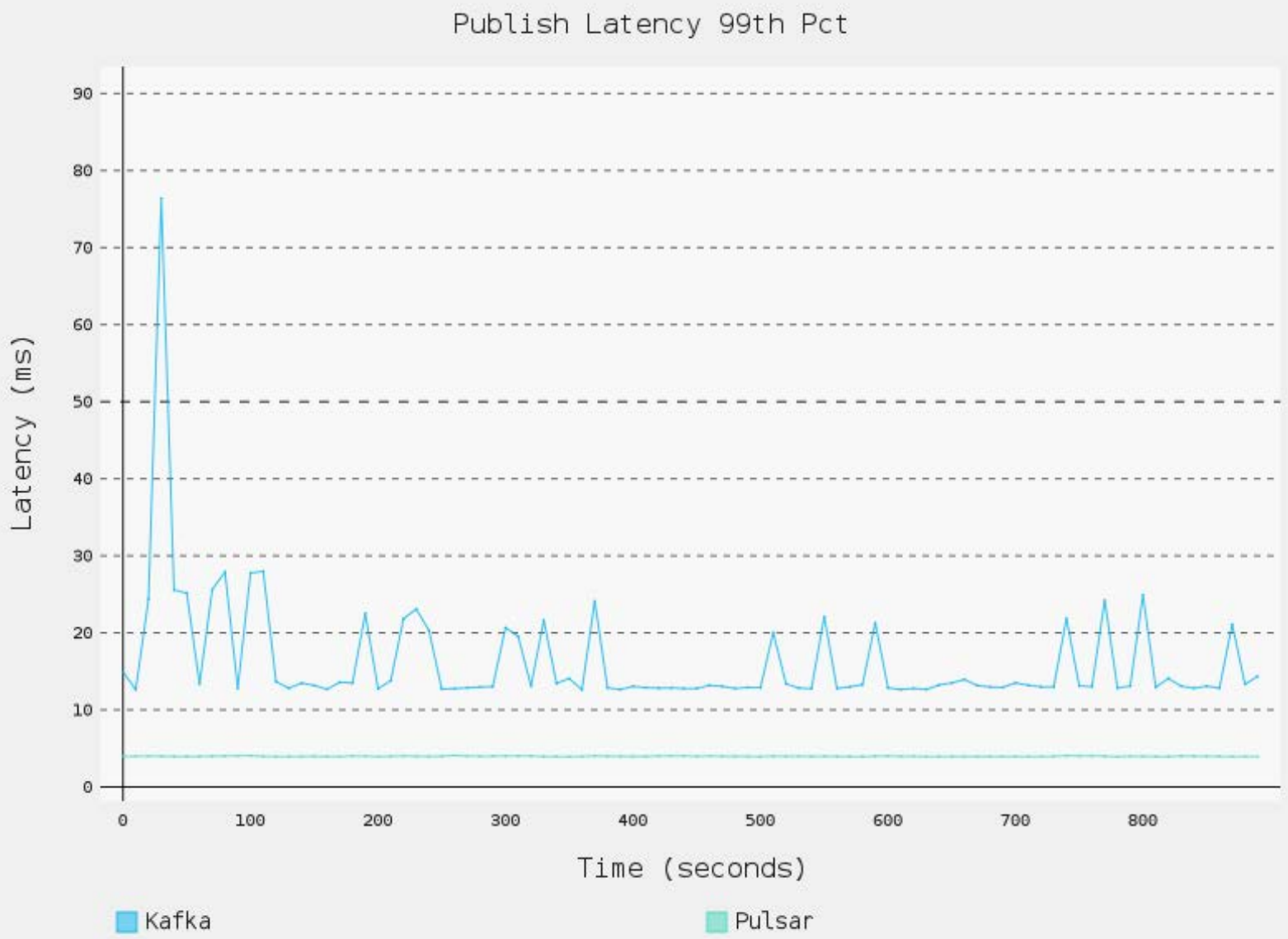




- Pulsar 的 Docker 版本
- https://pulsar.apache.org/docs/en/standalone-docker/

```
docker run -it \
  -p 6650:6650 \
  -p 8080:8080 \
  -v $PWD/data:/pulsar/data \
  apachepulsar/pulsar:2.4.1 \
  bin/pulsar standalone
```

- Kafka 和 Pulsar 的性能比较
- https://kafkaesque.io/performance-comparison-between-apache-pulsar-and-kafka-latency/



Apache Pulsar 进阶

- 分布式部署方法以及其他进阶内容
- https://pulsar.apache.org/docs/en/concepts-overview/

目录

Celery 简介

- Celery
- http://www.celeryproject.org/

- Celery 是一个开源免费的异步任务队列（Asynchronous Task Queue）系统
- Celery 自身采用 Python 编写，但任务队列协议（Webhooks）可以使用任意语言实现
- 官方提供的客户端包括：Ruby（Rcelery）、PHP、Go、JavaScript 等
- Celery 需要使用消息队列作为底层 Broker，且 Celery 支持大部分常见的消息队列或流数据库，包括：RabbitMQ（推荐）、Redis（推荐）、MongoDB、Amazon SQS、CouchDB、IronMQ 等

- Celery 是目前使用最广的异步任务队列系统

Celery 简介

- Celery 是为互联网应用的主从 REST 服务架构而设计的
- Celery 拥有一个主服务，通过 Broker（即消息队列）连接多个从服务
- 主服务通过消息队列创建任务，从服务通过消息队列认领并完成任务

- 安装 Celery
- `pip install celery`

- 本课程以 Python 为例介绍 Celery 的使用过程
- 在使用 Celery 之前，需要选择一个 Broker，本课程以 RabbitMQ 为例
- RabbitMQ 的安装和使用方法请参见上文

使用 Celery

- 从服务（Slave）
- tasks.py

```
from celery import Celery
app = Celery('tasks', broker='pyamqp://guest@localhost//')
@app.task
def add(x, y):
    return x + y
```

- 启动从服务：

```
celery -A tasks worker --loglevel=info
```

- 可以在不同的服务器上，启动多个从服务进程

使用 Celery

- 主服务（Master）
- `from tasks import add`
- `add.delay(4, 4)`

- 查看任务完成状态：result.ready()，会返回 True 或 False
- 获得任务执行结果：result.get()，此函数会阻塞代码执行，直到从服务执行完毕
- 如果希望阻塞代码执行，可以使用：result.get(timeout=1)
- 如果不希望将从服务的错误传送到主服务，可以使用：result.get(propagate=False)
- 此时如果需要查看从服务的错误可以使用：result.traceback

使用 Celery

Flower Dashboard Tasks Broker Monitor Logout Docs Code

Show 10 entries Search:

Name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
geonode.solr.tasks.sync_to_solr	3e194dc0-da55-4cd2-8e41-989686185b8d	FAILURE	(157,)	{}		2017-08-07 16:57:49.643	2017-08-07 16:57:49.646		celery@ubuntu-xenial
geonode.solr.tasks.sync_to_solr	044badf1-3782-446a-9eae-f05441e5fe15	SUCCESS	(157,)	{}	'None'	2017-08-07 16:56:38.424	2017-08-07 16:56:38.428	0.070	celery@ubuntu-xenial
geonode.solr.tasks.sync_to_solr	e479e00b-626a-4633-9d99-4a0df4032c33	SUCCESS	(157,)	{}	'None'	2017-08-07 16:56:13.236	2017-08-07 16:56:13.242	0.388	celery@ubuntu-xenial

Showing 1 to 3 of 3 entries Previous 1 Next

使用 Celery

Flower Dashboard Tasks Broker Monitor Logout Docs Code

app.tasks.celery_task 57382bfa-0566-4ec3-8957-1a2f76d85d5c

Basic task options

Name	app.tasks.celery_task
UUID	57382bfa-0566-4ec3-8957-1a2f76d85d5c
State	SUCCESS
args	(1,)
kwargs	{}
Result	'1 Done!'

Advanced task options

Received	2019-07-25 10:03:32.756258 UTC
Started	2019-07-25 10:03:32.760406 UTC
Succeeded	2019-07-25 10:04:02.787915 UTC
Retries	0
Worker	celery@hassanzadeh-ubuntu
Timestamp	2019-07-25 10:04:02.787915 UTC
Runtime	30.029780786999254
Clock	959
Root	<Task: app.tasks.celery_task(57382bfa-0566-4ec3-8957-1a2f76d85d5c) SUCCESS clock:959>
Root id	57382bfa-0566-4ec3-8957-1a2f76d85d5c
Children	<_weakrefset.WeakSet object at 0x7f2ba9fd85c0>

- Celery 详细使用教程
- http://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html

- Django
- https://www.djangoproject.com/

- Django Celery
- https://github.com/celery/django-celery

目录

其他消息队列系统

LMAX Disruptor

- 金融行业经常有以下的消息队列需求：
- 单一发送者，多个接收者（Single Producer & Multiple Consumer）
- 指一种特殊的消息队列，其中只有一个发送者，而有多个接收者，且：
- 接收者的进度不一样，以及
- 接收者如果长时间不接受新数据，可以容忍数据丢失

- 撮合引擎（Matching Engine）作为发送者，只有一个
- 行情系统（Ticker）作为接收者
- 不同的行情类型（如 OHLC、Ticker 等）都需要从撮合引擎订阅数据，且频率不一样

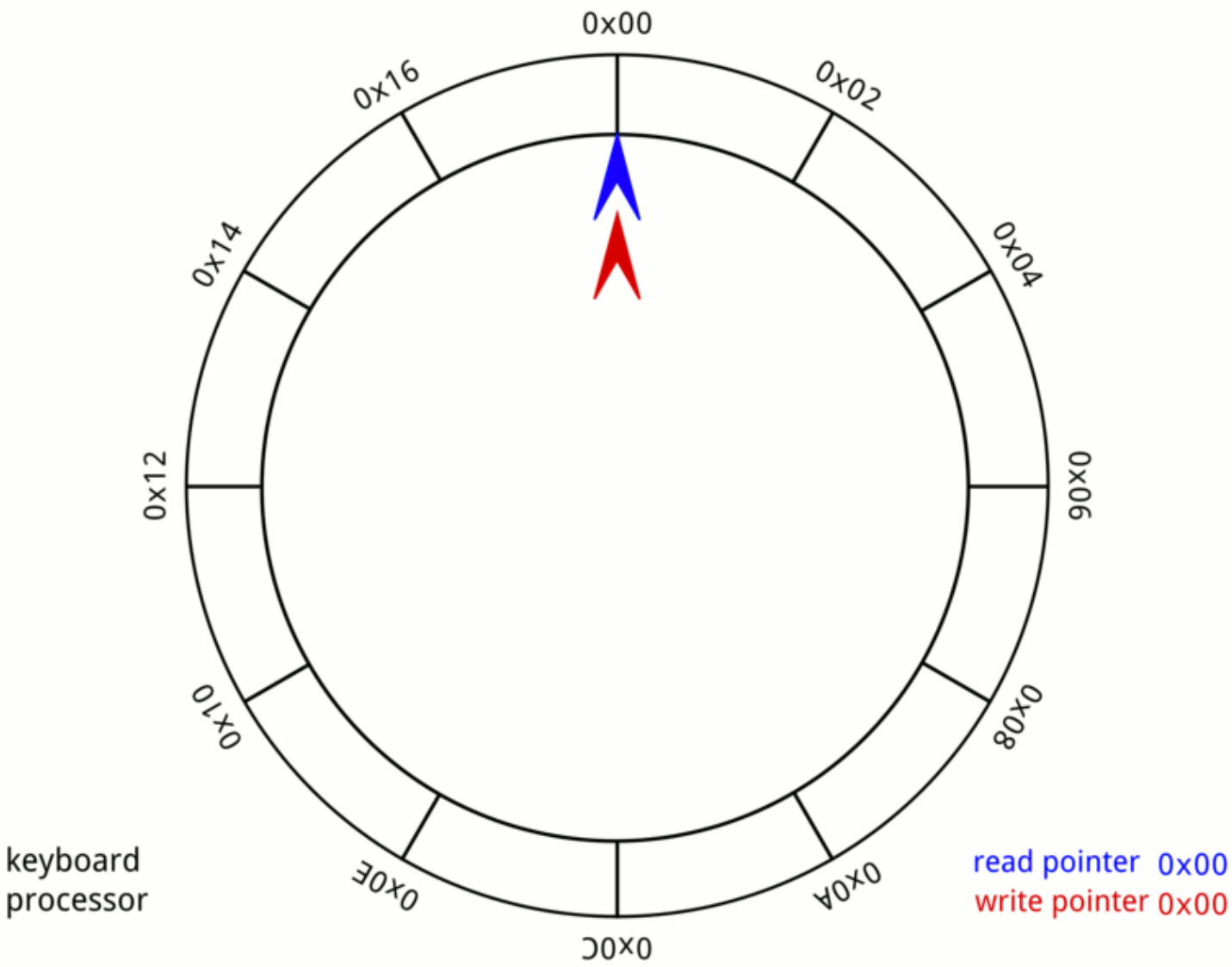
其他消息队列系统

LMAX Disruptor

- 环形缓冲（Ring Buffer，又称 Circular buffer）
- 是一种用于表示一个固定尺寸、头尾相连的缓冲区的数据结构，适合缓存数据流
- 当一个数据元素被用掉后，其余数据元素不需要移动其存储位置
- 相反，一个非环形缓冲（例如一个普通的队列）在用掉一个数据元素后，其余数据元素需要向前搬移

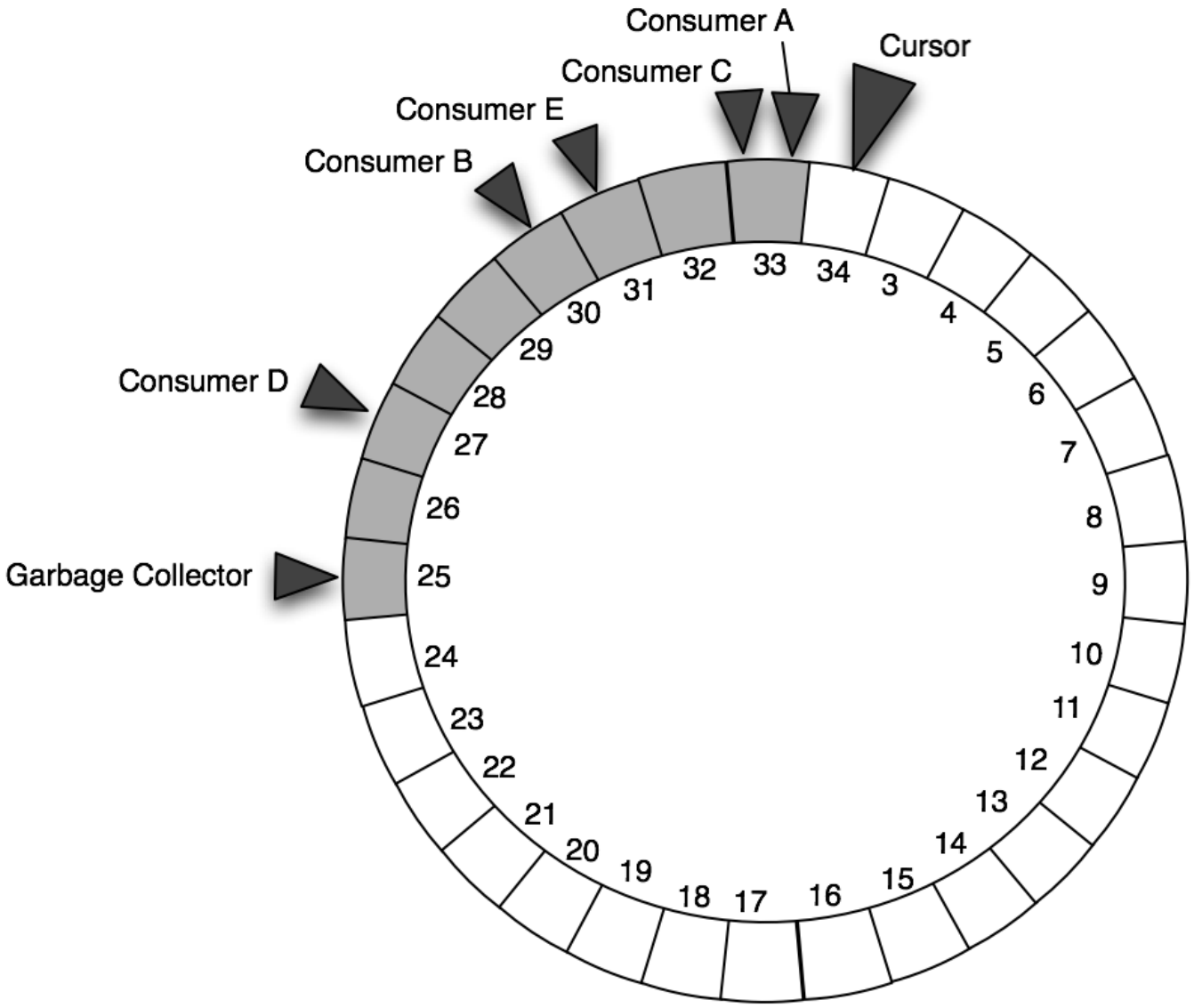
其他消息队列系统

LMAX Disruptor



其他消息队列系统

LMAX Disruptor



其他消息队列系统

LMAX Disruptor

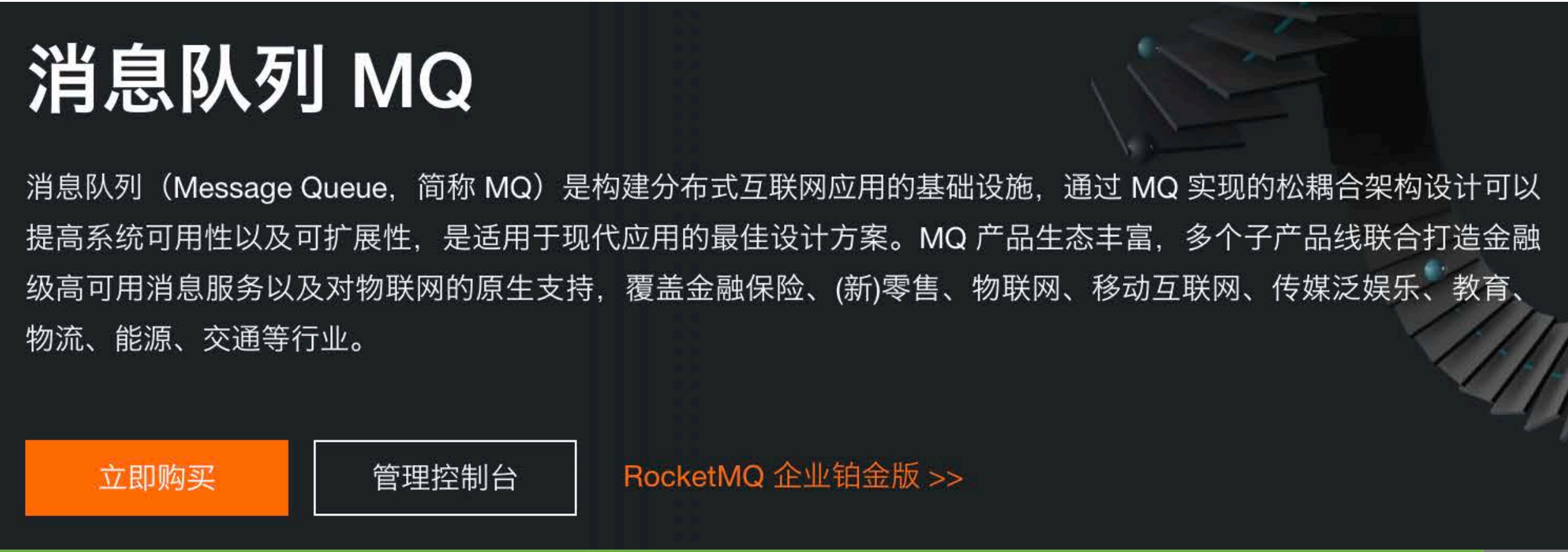
- LMAX Disruptor
- https://lmax-exchange.github.io/disruptor/

- 环形缓冲的一个 Java 版本
- LMAX 是一家数字货币（Cryptocurrency）交易所
- Disruptor 性能卓越，如果开启 CPU 的 Compare-and-swap 功能，每秒消息处理速度可达到 10 万次以上，远远超过标准的消息队列
- 芝加哥商品交易所（Chicago Mercantile Exchange，CME）也采用了类似 Disruptor 的架构

其他消息队列系统

云服务供应商的消息队列系统

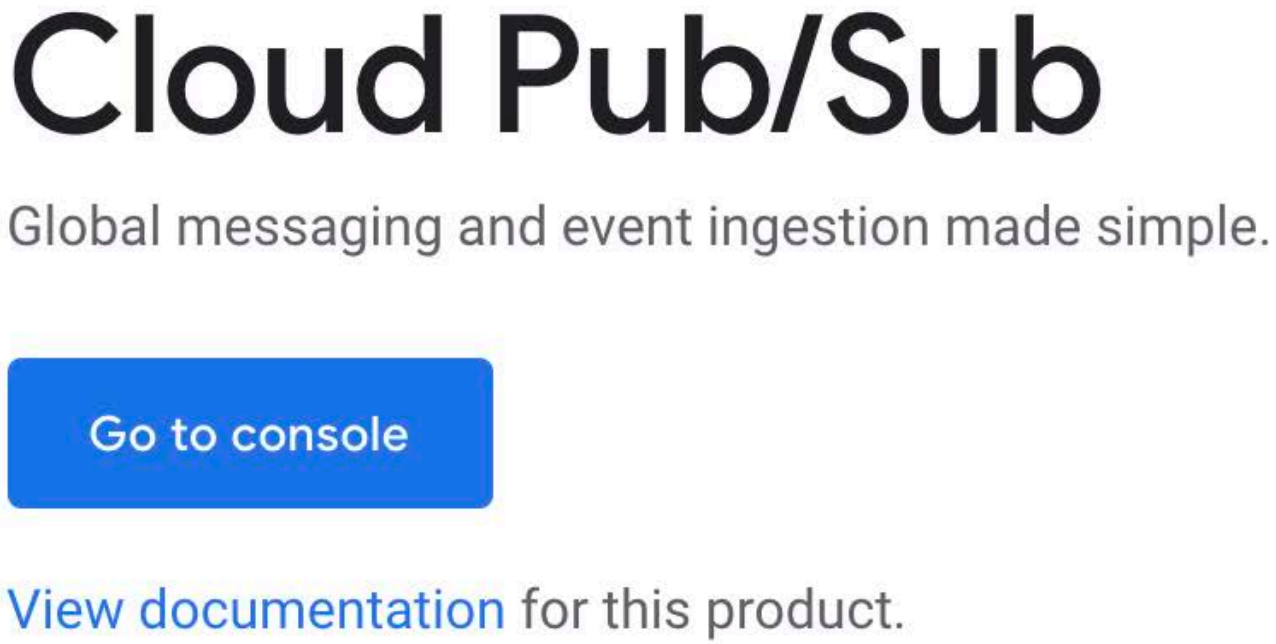
- 阿里云消息队列 MQ
- http://aliyun.com/product/ons



其他消息队列系统

云服务供应商的消息队列系统

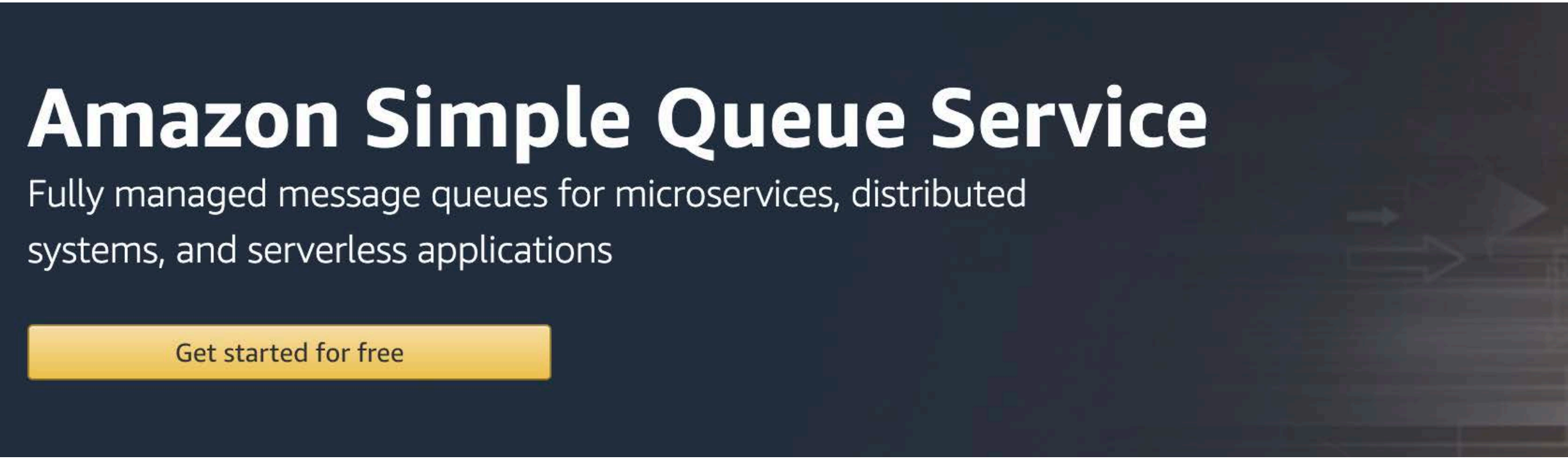
- Google Cloud Pub/Sub
- https://cloud.google.com/pubsub/



其他消息队列系统

云服务供应商的消息队列系统

- Amazon Simple Queue Service
- https://aws.amazon.com/sqs/



总结

- 三大云服务平台
- Google Cloud：https://cloud.google.com
- Amazon Web Services：https://aws.amazon.com
- 阿里云：https://www.aliyun.com

总结

- 课外阅读
- 《云存储技术——分析与实践》，刘洋著，经济管理出版社
- http://product.dangdang.com/24247525.html

- 《Ahead in the Cloud》，Stephen Orban（GM of AWS）
- https://www.amazon.com/Ahead-Cloud-Practices-Navigating-Enterprise/dp/1981924310/

- 《Cloud Computing: Concepts, Technology & Architecture》，Thomas Erl
- https://www.amazon.com/Cloud-Computing-Concepts-Technology-Architecture/dp/0133387526/

Thanks!