Messaging with RabbitMQ

https://www.rabbitmq.com/#getstarted

About RabbitMQ

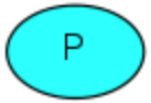
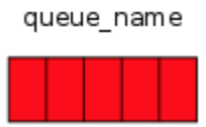
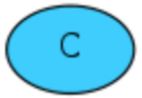
RabbitMQ is an open-source distributed message/streaming broker.

RabbitMQ supports AMQP (Advanced Messaging Queuing Protocol), MQTT (Message Queue Telemetry Transport protocol)

AMQP, MQTT are an open standard application layer protocols.

RabbitMQ accepts, stores, and forwards *messages*: blobs of data

Jargons

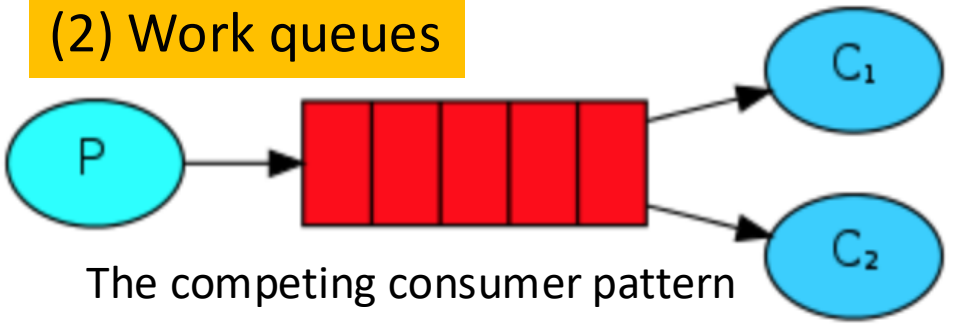
Producing A program which sends message is a *producer.*	
Queue : a post box living inside RabbitMQ.	
Consuming A *consumer* is a program that mostly waits to receive messages.	

RabbitMQ is a push model.

Queue sends message to consumers

(1) Hello world

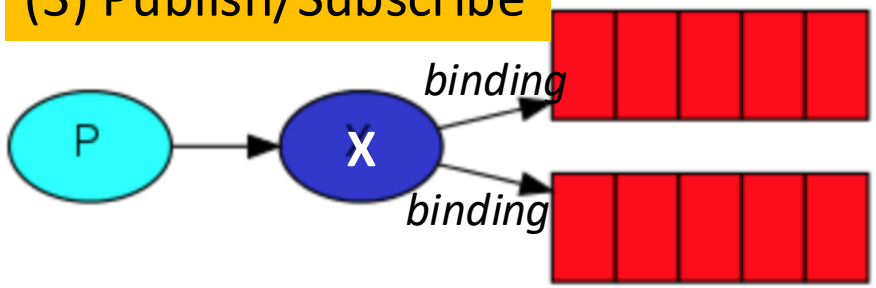
(2) Work queues



The competing consumer pattern

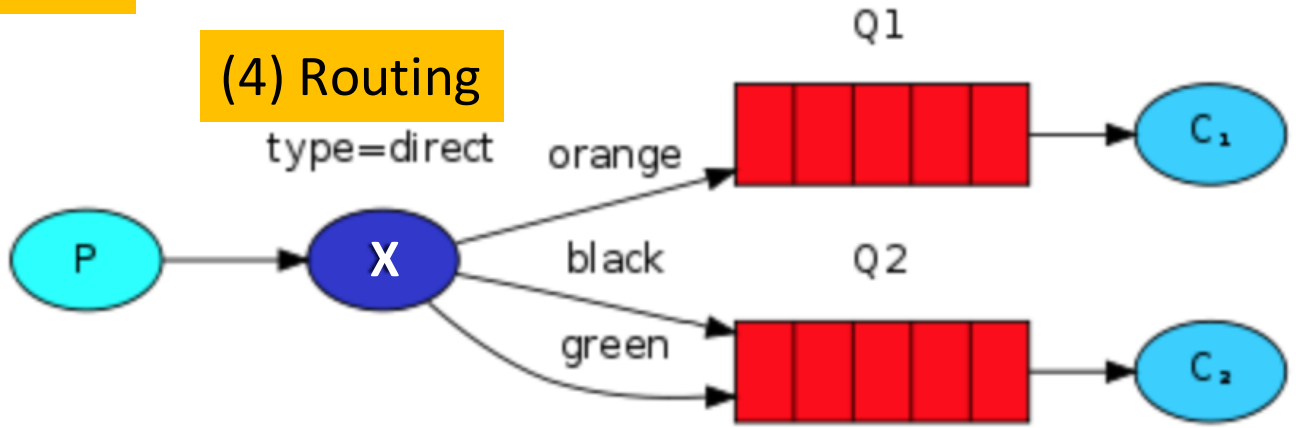
(7) Publisher confirms

(3) Publish/Subscribe



Exchange type: direct, topic, headers, fanout

(4) Routing



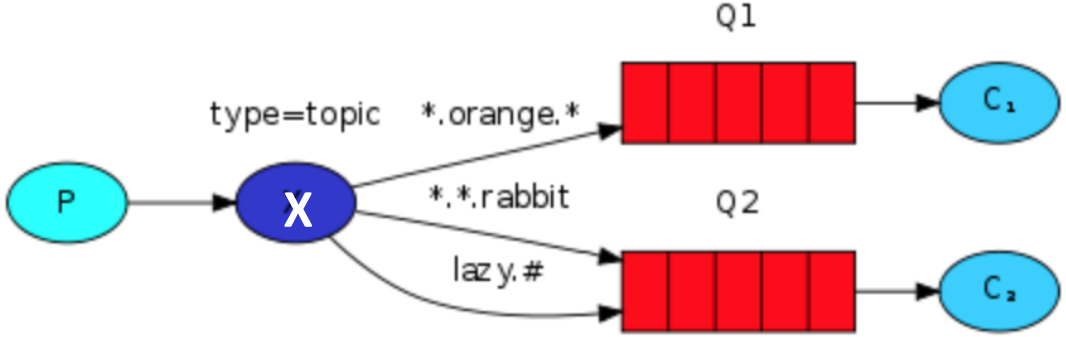
Exchange type: direct, topic, headers, fanout

binding key == routing key

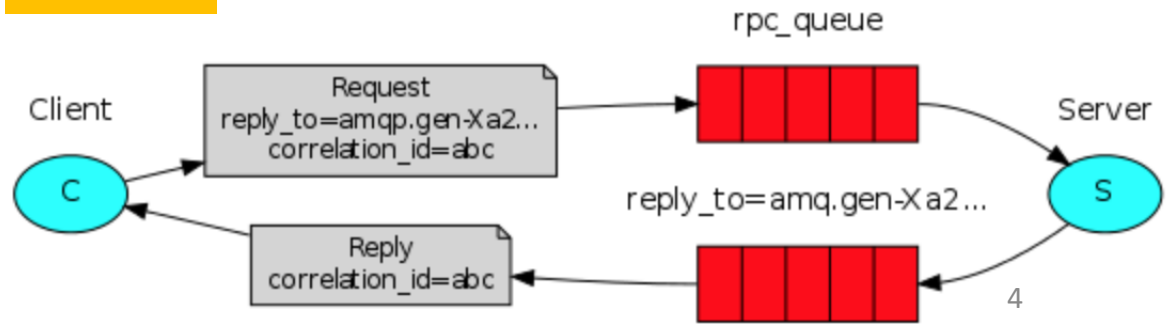
(5) Topics

* Can substitute for exactly one word

\# Can substitute for zero or more words



(6) RPC



Pre-requisite

Install RabbitMQ on local machine.

RabbitMQ server

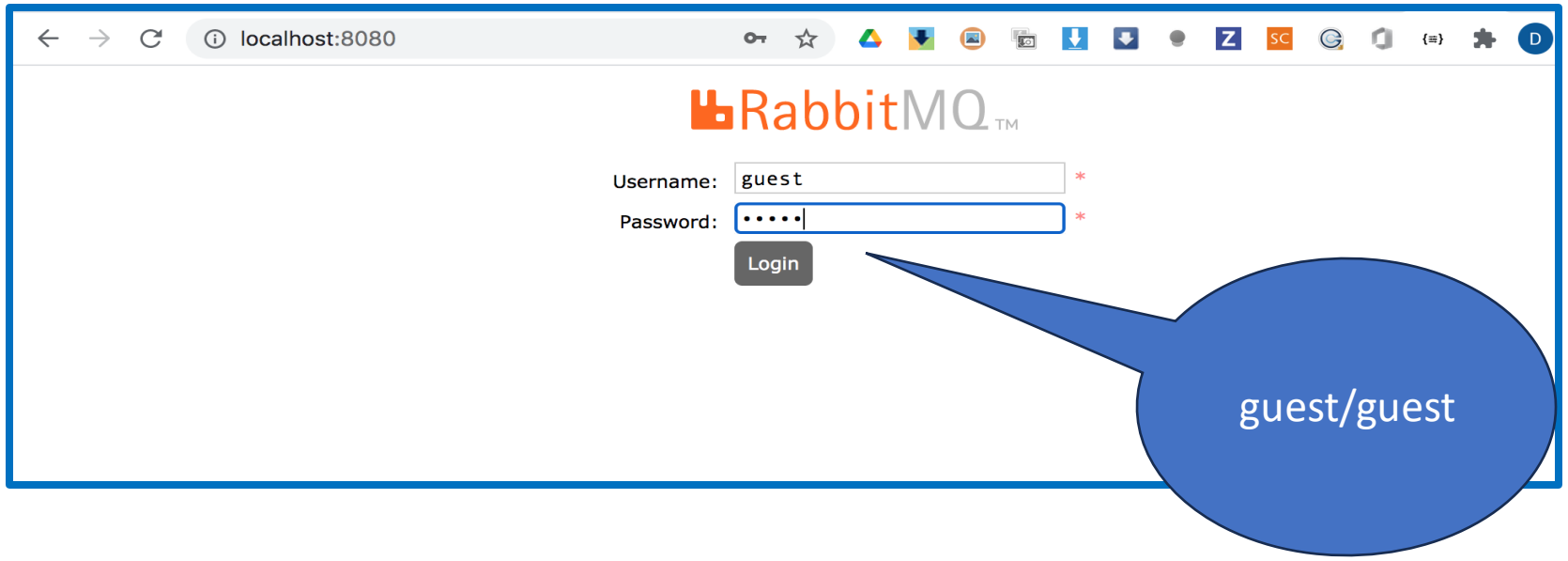
```
docker run --name my-rabbitmq -p 5672:5672 -d rabbitmq:4.1.4
```

RabbitMQ management

```
docker run --name my-rabbitmq-management -p 8080:15672 -d
rabbitmq:4.1.4-management
```

https://hub.docker.com/rabbitmq

Pre-requisite

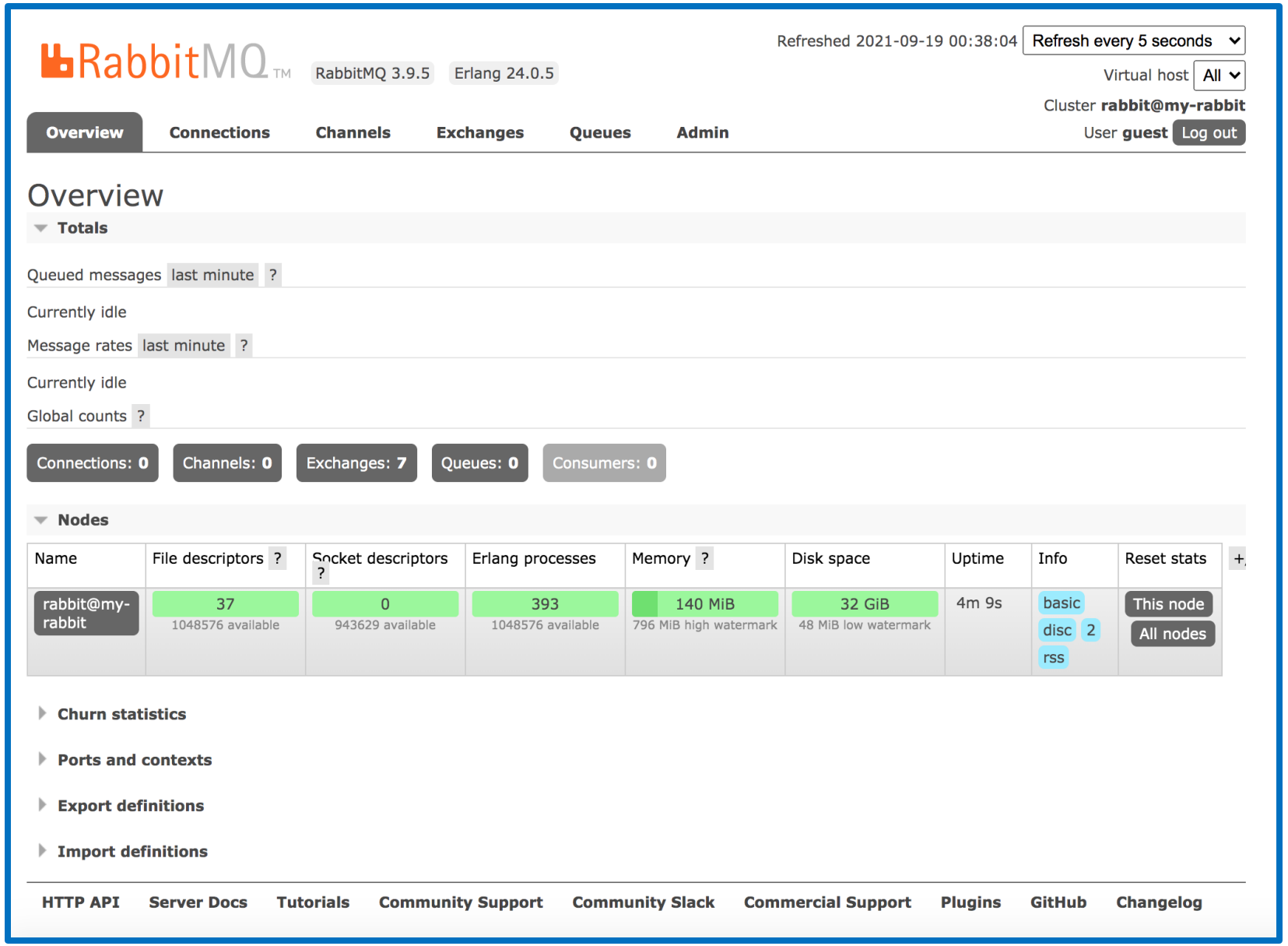


http://localhost:8080

```
docker exec my-rabbitmq-management rabbitmqctl status
docker exec my-rabbitmq-management rabbitmqctl
```

https://hub.docker.com/rabbitmq

RabbitMQ Management



Python library as RabbitMQ client

- python -m pip install pika --upgrade

(1)
Hello world

Producer (P)

```
1   import sys
2
3   #!/usr/bin/env python
4   import pika
5
6   connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
7   channel = connection.channel()
8   channel.queue_declare(queue='hello')
9
10  channel.basic_publish(exchange='',
11                        routing_key='hello',
12                        body='Hello World!')
13  print(" [x] Sent 'Hello World!'")
14  connection.close()
```

(1) Hello world


```
python 1_HelloReceive.py
python 1_HelloSent.py
```

(1)
Hello world

Consumer (C)

```python
#!/usr/bin/env python
import pika, sys, os

def main():
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='hello')

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)

    channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

(1) Hello world


```
python 1_HelloReceive.py
python 1_HelloSent.py
```

(1)
Hello world

DEMO

(1) Hello world

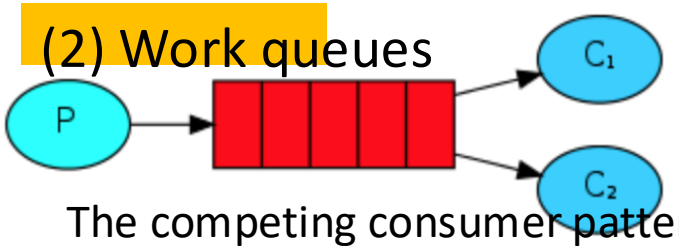

```
python 1_HelloReceive.py
python 1_HelloSent.py
```

(2) Work queue/ Task queue

The assumption behind a work queue is that each task is delivered to exactly one worker.

Round robin dispatching

(2) Work queues

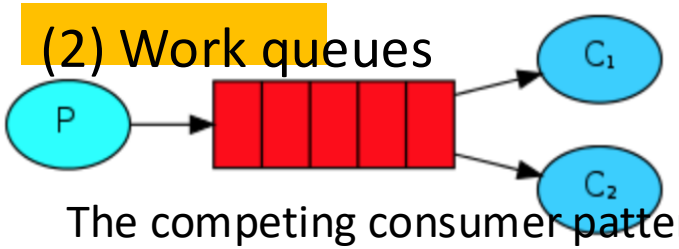


The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

(2) Work queue/ Task queue (cont.)

(2) Work queues



The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

Producer (P)

Message durability

```
1   #!/usr/bin/env python
2   import pika
3   import sys
4
5   connection = pika.BlockingConnection(
6       pika.ConnectionParameters(host='localhost'))
7   channel = connection.channel()
8
9   channel.queue_declare(queue='task_queue', durable=True)
10                                          #make queue persistent
11  message = ' '.join(sys.argv[1:]) or "Hello World!"
12  channel.basic_publish(
13      exchange='',
14      routing_key='task_queue',
15      body=message,
16      properties=pika.BasicProperties(
17          delivery_mode=2,  # make message persistent
18      ))
19  print(" [x] Sent %r" % message)
20  connection.close()
```

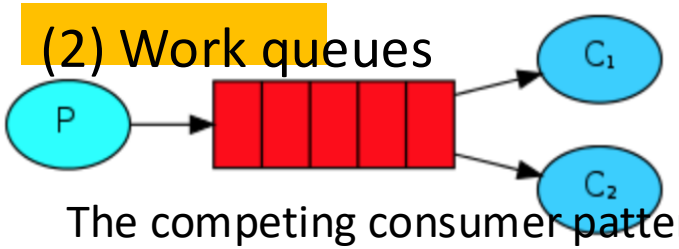
To ensure if RabbitMQ quits or crashes, it won't forget queues

Message will be stored to disk.

pika.spec.PERSISTENT_DELIVERY_MODE

(2) Work queue/ Task queue (cont.)

(2) Work queues



The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

Consumer (C)

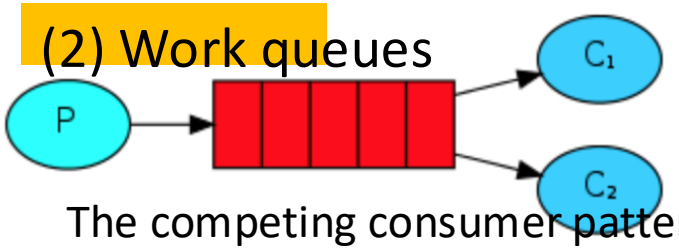
Message durability

```
1   #!/usr/bin/env python
2   import pika
3   import time
4
5   connection = pika.BlockingConnection(
6       pika.ConnectionParameters(host='localhost'))
7   channel = connection.channel()
8
9   channel.queue_declare(queue='task_queue', durable=True)
10  print(' [*] Waiting for messages. To exit press CTRL+C')
11
12
13  def callback(ch, method, properties, body):
14      print(" [x] Received %r" % body.decode())
15      time.sleep(body.count(b'.'))
16      print(" [x] Done")
17      ch.basic_ack(delivery_tag=method.delivery_tag)
18
19
20  channel.basic_qos(prefetch_count=1)
21  channel.basic_consume(queue='task_queue', on_message_callback=callback)
22
23  channel.start_consuming()
24
```

To ensure if RabbitMQ quits or crashes, it won't forget queues

(2) Work queue/ Task queue (cont.)

(2) Work queues



The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

Consumer (C)

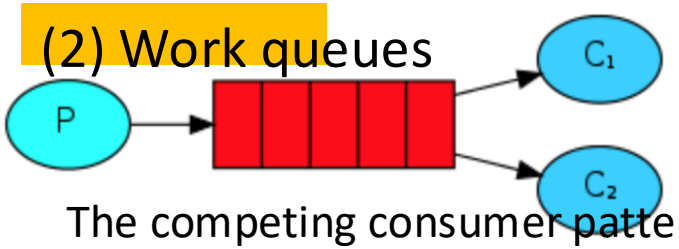
Manage acknowledge

```python
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode())
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()

```

To ensure that worker will send ack when completing the task.

(2) Work queue/ Task queue (cont.)

(2) Work queues



The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

Consumer (C)

What the difference between round robin and fair dispatch?

Fair dispatch

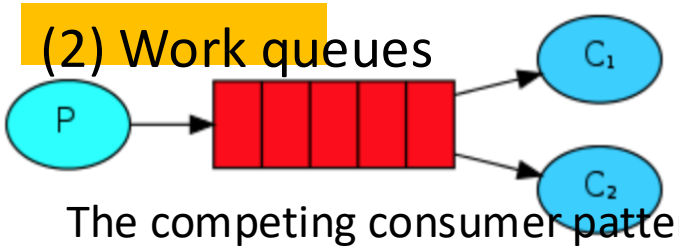
```python
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode())
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()

```

Tell the RabbitMQ not to dispatch a new task if the worker is busy.

(2) Work queue/ Task queue (cont.)

DEMO

(2) Work queues



The competing consumer pattern

```
python 2_Worker.py
python 2_Worker.py
…
python 2_TaskQueue.py
```

(3) Publish/ Subscribe (fanout)

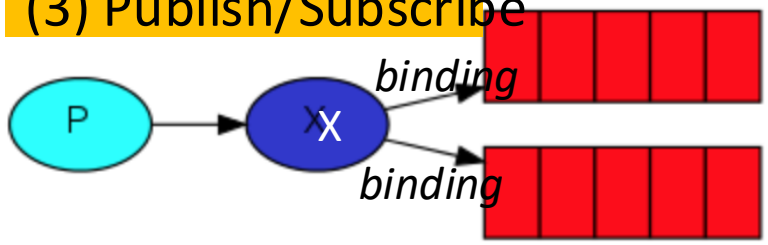
RabbitMQ will deliver a message to multiple consumers.

```
docker exec my-rabbitmq-management rabbitmqctl list_exchanges
docker exec my-rabbitmq-management rabbitmqctl list_bindings
```

Producer (P)

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='logs', exchange_type='fanout')

message = ' '.join(sys.argv[1:]) or "info: Hello World!"
channel.basic_publish(exchange='logs', routing_key='', body=message)
print(" [x] Sent %r" % message)
connection.close()
```

(3) Publish/Subscribe



Exchange type: direct, topic, headers, fanout

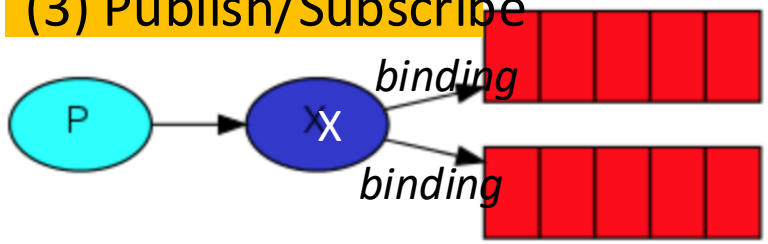
```
python 3_Publish_broadcast_log.py
python 3_Subscribe_log.py
```

The use of exchange introduces the full messaging model in Rabbit.

Core idea is that a producer never send any messages directly to a queue..

(3) Publish/ Subscribe (fanout) (cont.)

(3) Publish/Subscribe



Exchange type: direct, topic, headers, fanout

```
python 3_Publish_ broadcast_log
python 3_Subscribe_log.py
```

Consumer (C)

```
1  #!/usr/bin/env python
2  import pika
3
4  connection = pika.BlockingConnection(
5      pika.ConnectionParameters(host='localhost'))
6  channel = connection.channel()
7
8  channel.exchange_declare(exchange='logs', exchange_type='fanout')
9
10 result = channel.queue_declare(queue='', exclusive=True)
11 queue_name = result.method.queue
12
13 channel.queue_bind(exchange='logs', queue=queue_name)
14
15 print(' [*] Waiting for logs. To exit press CTRL+C')
16
17 def callback(ch, method, properties, body):
18     print(" [x] %r" % body)
19
20 channel.basic_consume(
21     queue=queue_name, on_message_callback=callback, auto_ack=True)
22
23 channel.start_consuming()
```

We want to hear all messages and only the current flowing messages not the older ones.
แปลว่า?

Let the server choose the random queue name and this queue will be deleted when the consumer's connection is closed.

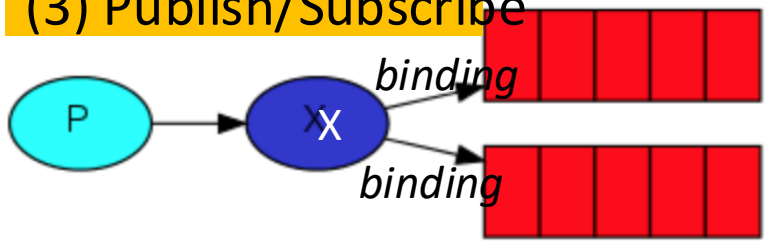
Bind exchange with queue

(3) Publish/ Subscribe (fanout) (cont.)

```
python 3_Subscribe_log.py
python 3_Subscribe_log.py
python 3_Publish_broadcast_log.py
```

DEMO

(3) Publish/Subscribe

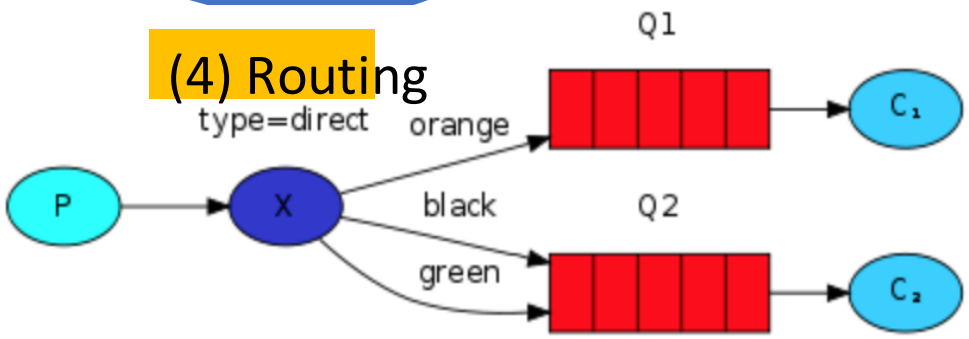


Exchange type: direct, topic, headers, fanout

```
python 3_Publish_broadcast_log.py
python 3_Subscribe_log.py
```

(4) Routing

(4) Routing



Producer (P)

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='direct_logs', exchange_type='direct')

severity = sys.argv[1] if len(sys.argv) > 1 else 'info'
message = ' '.join(sys.argv[2:]) or 'Hello World!'
channel.basic_publish(
    exchange='direct_logs', routing_key=severity, body=message)
print(" [x] Sent %r:%r" % (severity, message))
connection.close()
```

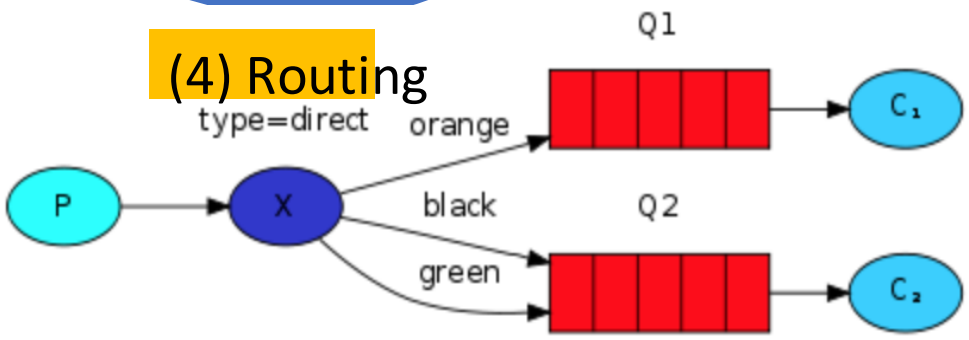
Exchange type: direct, topic, headers, fanout

binding key == routing key

```
python 4_Publish_direct_log.py
python 4_Subscrib_direct_log.py
```

(4) Routing

(4) Routing



Exchange type: direct, topic, headers, fano
binding key == routing key

```
python 4_Publish_direct_log.py
python 4_Subscrib_direct_log.py
```

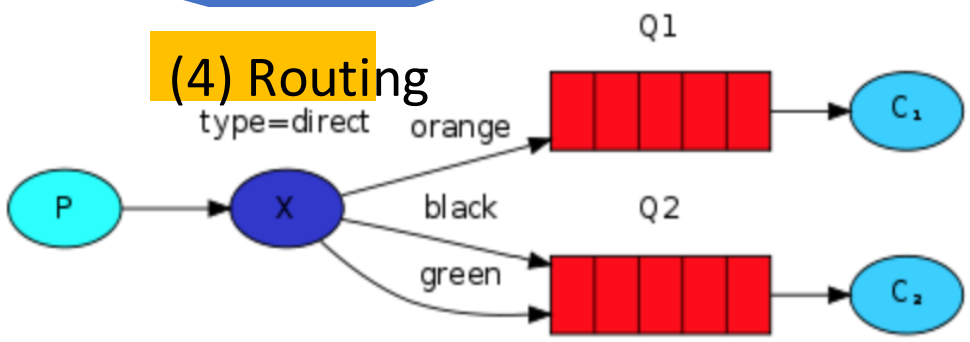
Consumer (C)

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='direct_logs', exchange_type='direct')

result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

severities = sys.argv[1:]
if not severities:
    sys.stderr.write("Usage: %s [info] [warning] [error]\n" % sys.argv[0])
    sys.exit(1)

for severity in severities:
    channel.queue_bind(
        exchange='direct_logs', queue=queue_name, routing_key=severity)

print(' [*] Waiting for logs. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] %r:%r" % (method.routing_key, body))


channel.basic_consume(
    queue=queue_name, on_message_callback=callback, auto_ack=True)

channel.start_consuming()
```

(4) Routing

```
python 4_Subscribe_direct_log.py warning error > logs_from_rabbit.log
python 4_Subscribe_direct_log.py info warning error
python 4_Publish_direct_log.py error "GPU card 0 is down!!"
python 4_Publish_direct_log.py info ”Add new GPU card!!
```

(4) Routing



Exchange type: direct, topic, headers, fanou

binding key == routing key

```
python 4_Publish_direct_log.py
python 4_Subscrib_direct_log.py
```

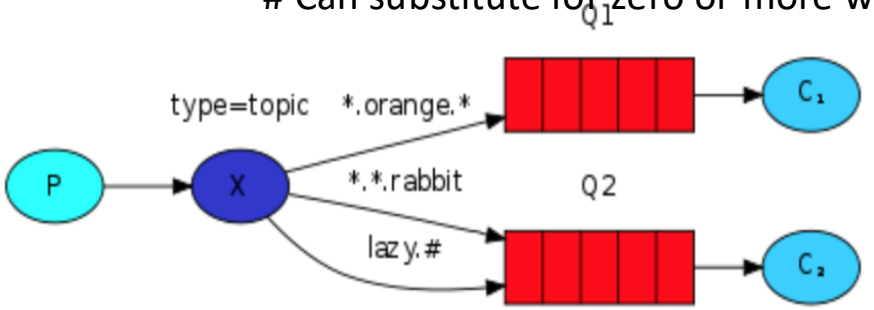
DEMO

(5) Topic

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='topic_logs', exchange_type='topic')

routing_key = sys.argv[1] if len(sys.argv) > 2 else 'anonymous.info'
message = ' '.join(sys.argv[2:]) or 'Hello World!'
channel.basic_publish(
    exchange='topic_logs', routing_key=routing_key, body=message)
print(" [x] Sent %r:%r" % (routing_key, message))
connection.close()
```

(5) Topics

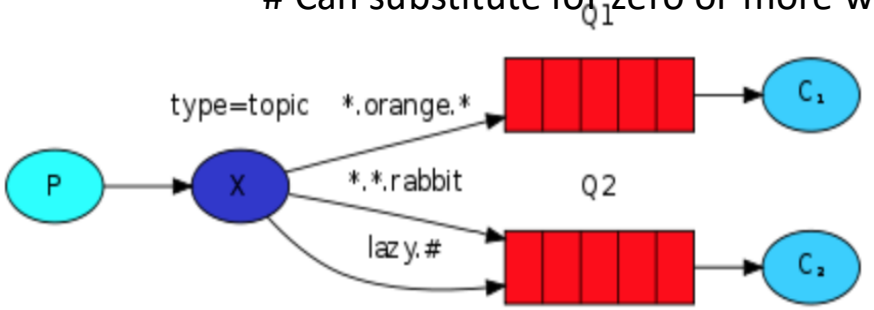
* Can substitute for exactly one word
# Can substitute for zero or more words




```
python 5_Publish_topic_log.py
python 5_Subscribe_topic_log.py
```

(5) Topic

(5) Topics * Can substitute for exactly one word
Can substitute for zero or more words




```
python 5_Publish_topic_log.py
python 5_Subscribe_topic_log.py
```

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.exchange_declare(exchange='topic_logs', exchange_type='topic')

result = channel.queue_declare('', exclusive=True)
queue_name = result.method.queue

binding_keys = sys.argv[1:]
if not binding_keys:
    sys.stderr.write("Usage: %s [binding_key]...\n" % sys.argv[0])
    sys.exit(1)

for binding_key in binding_keys:
    channel.queue_bind(
        exchange='topic_logs', queue=queue_name, routing_key=binding_key)

print(' [*] Waiting for logs. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] %r:%r" % (method.routing_key, body))


channel.basic_consume(
    queue=queue_name, on_message_callback=callback, auto_ack=True)

channel.start_consuming()
```

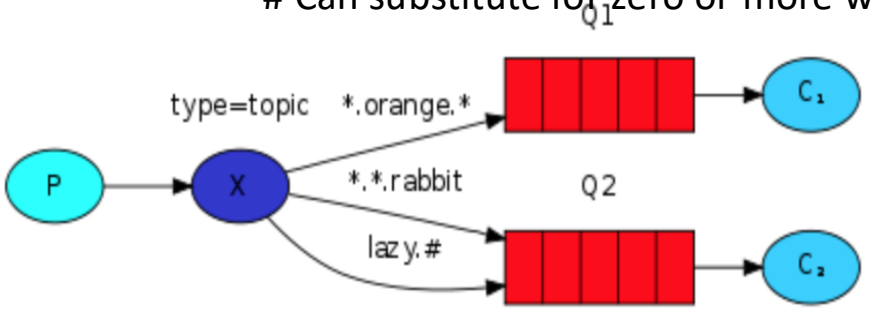
(5) Topic

```
Terminal 1: python 5_Subscribe_topic_log.py "#"
Terminal 2: python 5_Subscribe_topic_log.py "kern.*"
Terminal 3: python 5_Subscribe_topic_log.py "*.critical"
Terminal 4: python 5_Subscribe_topic_log.py "kern.*" "*.critical"
Terminal 5: python 5_Publish_topic_log.py "kern.critical" "kernel error"
```

DEMO

(5) Topics

* Can substitute for exactly one word
# Can substitute for zero or more words




```
python 5_Publish_topic_log.py
python 5_Subscribe_topic_log.py
```

RabbitMQ & Apache Kafka

RabbitMQ	Kafka
Direct messaging: users can set sophisticated rules for message delivery.	Kafka is ideal for handling large amounts of homogeneous messages, such as logs or metric, and it is the right choice for instances with high throughput.
RabbitMQ will be usually used with Cassandra for message playback.	Replay messages
Multiprotocol supports: AMQP, STOMP, MQTT, Web sockets and others.	Big data consideration with e.g., Elastic search, Hadoop
Flexibility: varied point-to-point, request/reply, publish/subscribe	Scaling capability; topics can be split into partitions
Communication: supports both async and sync	Batches: Kafka works best when messages are batched.
Security	Security
Mature platform	Mature platform
Slower than Kafka	Don’t come with user-friendly GUI but can be monitored via Kibana

https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case
https://dattell.com/data-architecture-blog/kafka-vs-rabbitmq-how-to-choose-an-open-source-message-broker/

Summary of differences: Kafka vs. RabbitMQ

	RabbitMQ	Kafka
Architecture	RabbitMQ's architecture is designed for complex message routing. It uses the push model. Producers send messages to consumers with different rules.	Kafka uses partition-based design for real-time, high-throughput stream processing. It uses the pull model. Producers publish messages to topics and partitions that consumers subscribe to.
Message handling	RabbitMQ brokers monitor message consumption. It deletes messages after they're consumed. It supports message priorities.	Consumers keep track of message retrieval with an offset tracker. Kafka retains messages according to the retention policy. There's no message priority.
Performance	RabbitMQ has low latency. It sends thousands of messages per second.	Kafka has real-time transmission of up to millions of messages per second.
Programming language and protocol	RabbitMQ supports a broad range of languages and legacy protocols.	Kafka has limited choices of programming languages. It uses binary protocol over TCP for data transmission.

https://aws.amazon.com/compare/the-difference-between-rabbitmq-and-kafka/

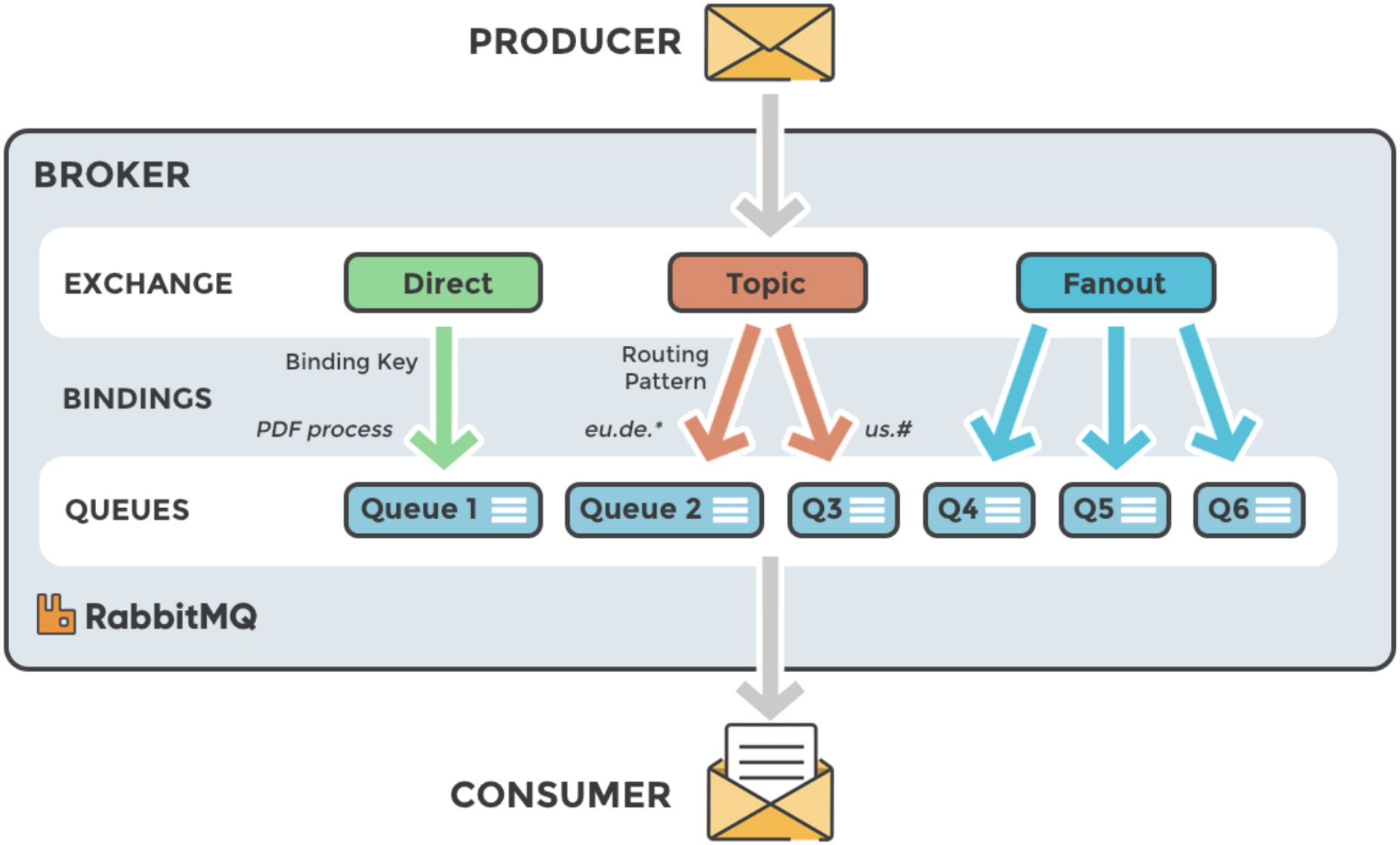
RabbitMQ & Apache Kafka

RabbitMQ: Used for high throughput and reliable background jobs, integration and intercommunication between and within applications, perform complex routing to consumers, integrate multiple applications and services with non-trivial routing logic.

Kafka: Best used for basic streaming without complex routing with maximum throughput, ideal for event-sourcing, stream processing, multi-stage pipelines, routinely audited systems, real-time processing and analyzing data

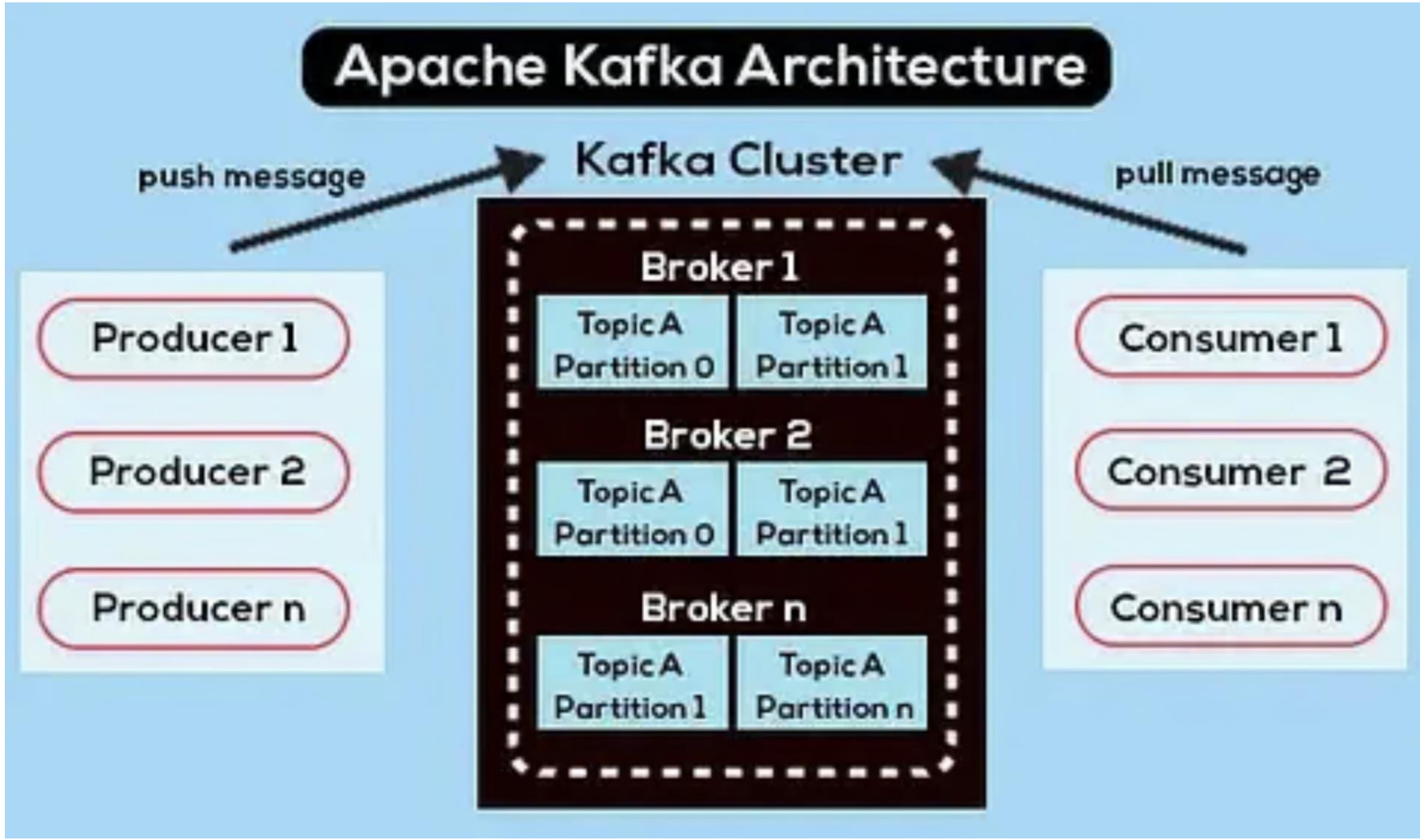
https://www.simplilearn.com/kafka-vs-rabbitmq-article#what_is_kafka_used_for

RabbitMQ Overall Architecture



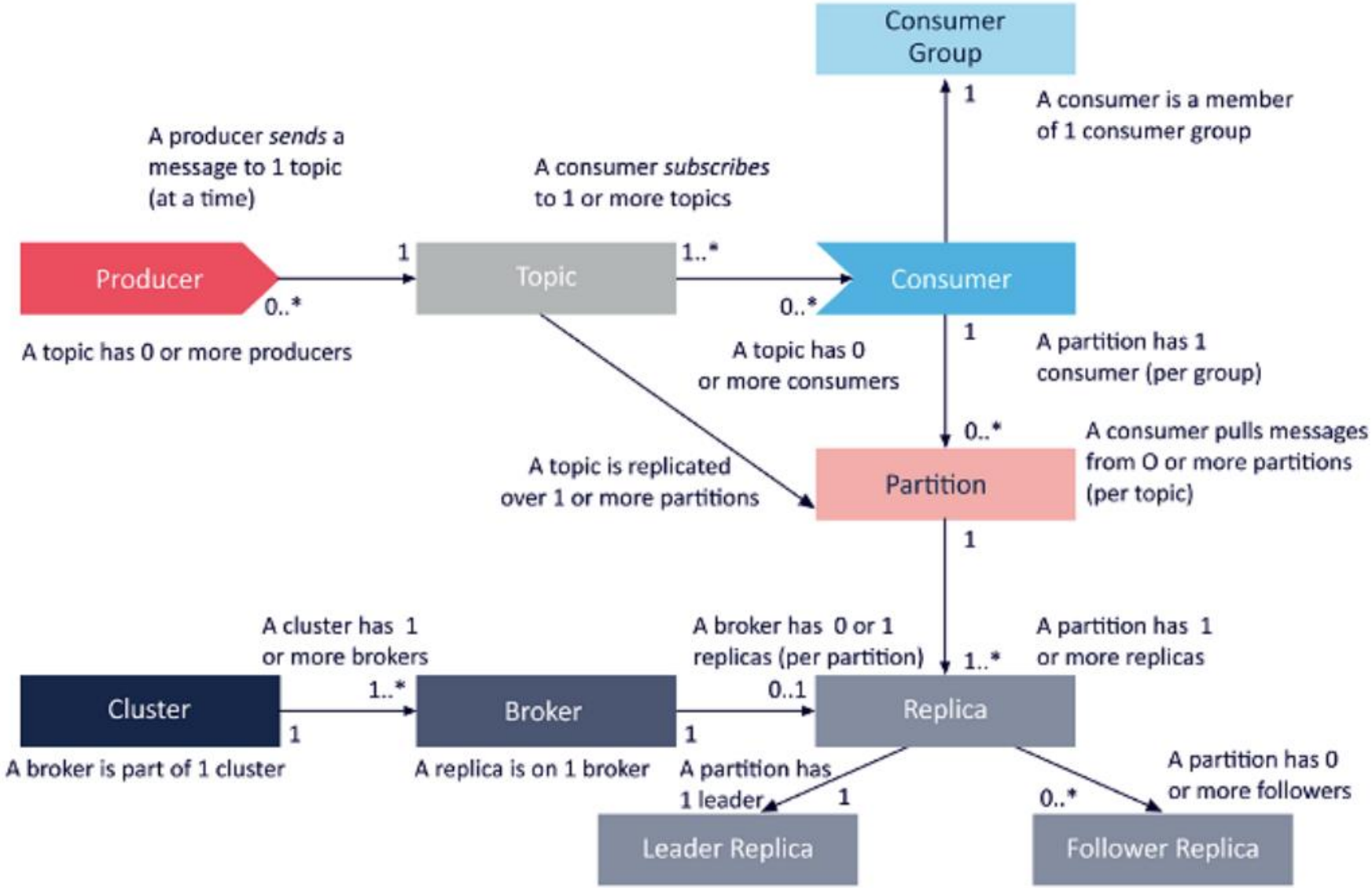
Source: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

Kafka Overall Architecture



Source: https://www.projectpro.io/article/apache-kafka-architecture-/442

Main Components in Kafka



https://www.instaclustr.com/education/apache-kafka/apache-kafka-architecture-a-complete-guide-2025/

Assignment

- Go to get starter source codes from

https://drive.google.com/drive/folders/1iRNlwWf8uSkfp3TriQm9QZCHjgx75CEk?usp=sharing

- Try running Kafka as shown in the starter kit (how_to_start_kafka.txt)
- Based on the given RabbitMQ and Kafka codes, compare the performance of RabbitMQ and Kafka based on round-robin work, where a producer produces 100K messages with the four consumers and the varied message sizes of 0.1, 0.5, and 1KB.
- Things to submit
  1) Your modified codes with the experiments
  2) Three plots (a message size per plot) comparing the throughputs between the two brokers.

RabbitMQ as a Service

CloudAMQP

Pricing Documentation Support Blog About us Login Sign Up

RabbitMQ as a Service

Managing the largest fleet of RabbitMQ clusters in the world

Get a managed RabbitMQ server today