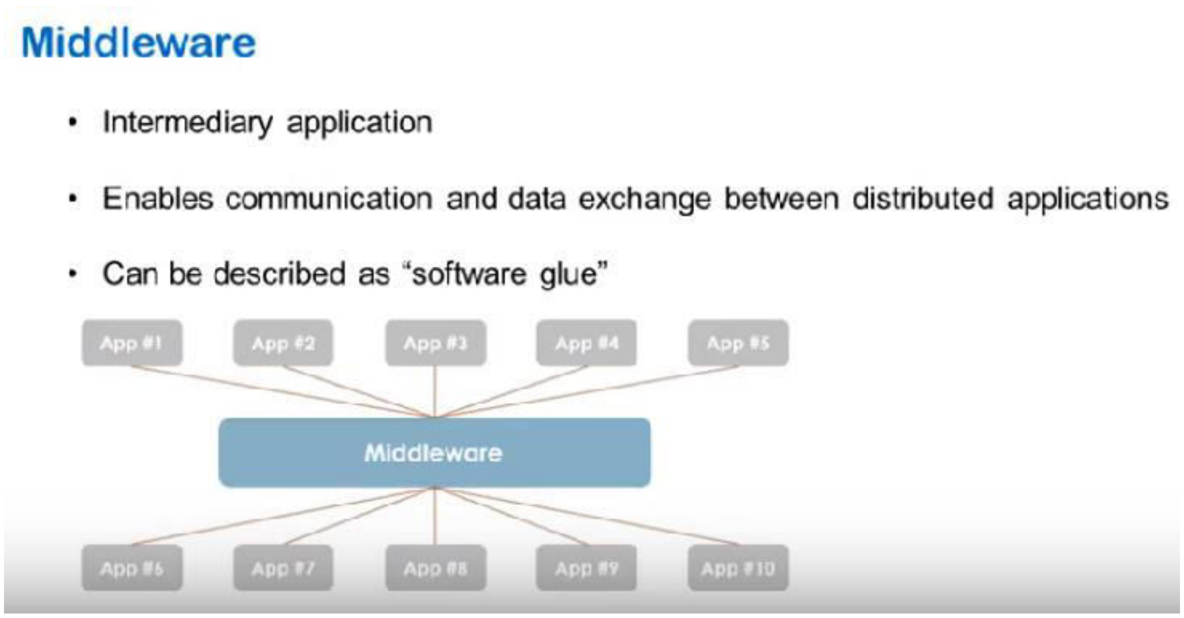
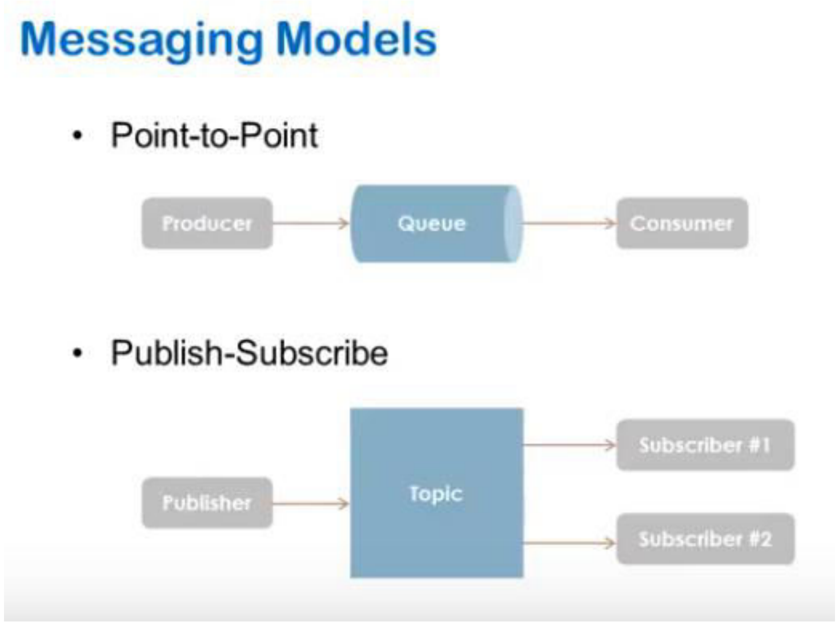
@ Middleware and Message Broker



Middleware

- Intermediary application
- Enables communication and data exchange between distributed applications
- Can be described as "software glue"

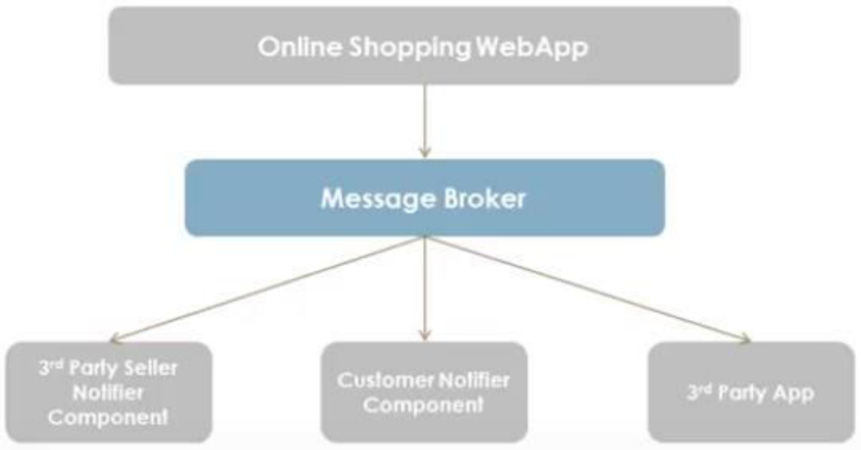


Messaging Models

- Point-to-Point

- Publish-Subscribe

Real-world Use Cases


Online Shopping WebApp
Message Broker
3rd Party Seller Notifier Component
Customer Notifier Component
3rd Party App

Popular Message Brokers

- RabbitMQ
- Apache ActiveMQ
- ZeroMQ
- Kafka

AMQP, RabbitMQ and Celery

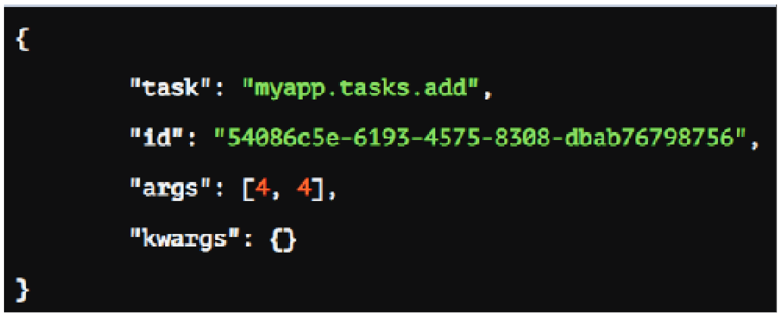
Celery is an asynchronous distributed task queue.
RabbitMQ is a message broker which implements the **A**dvanced **M**essage **Q**ueuing **P**rotocol (AMQP).

AMQP Key Terms

A **message** or **task** consists of attributes (**headers**) and payload (**body**). Some attributes are used by broker but most are used by consumers. Optional attributes are known as headers. Some common attributes are

1. Content type
2. Content encoding
3. Routing key

Massage payload or body contains data that goes to the consumer. Normally a serialisation format such as JSON is used for message payload. Content type and content encoding attributes are used to communicate the serialisation format. An example message payload serialised in JSON looks like,



```
{
        "task": "myapp.tasks.add",
        "id": "54086c5e-6193-4575-8308-dbab76798756",
        "args": [4, 4],
        "kwargs": {}
}
```

Producer

A **producer** is a user application that sends messages.

Broker

A **broker** receives messages from producer and router them to consumer. A broker consists an exchange and one more queues.

Exchange

A producer can send messages to queues only via exchange. **Exchange**s take a message from producer and route it into zero or more queues.
The routing algorithm used depends on the exchange type and rules called bindings.

Queue

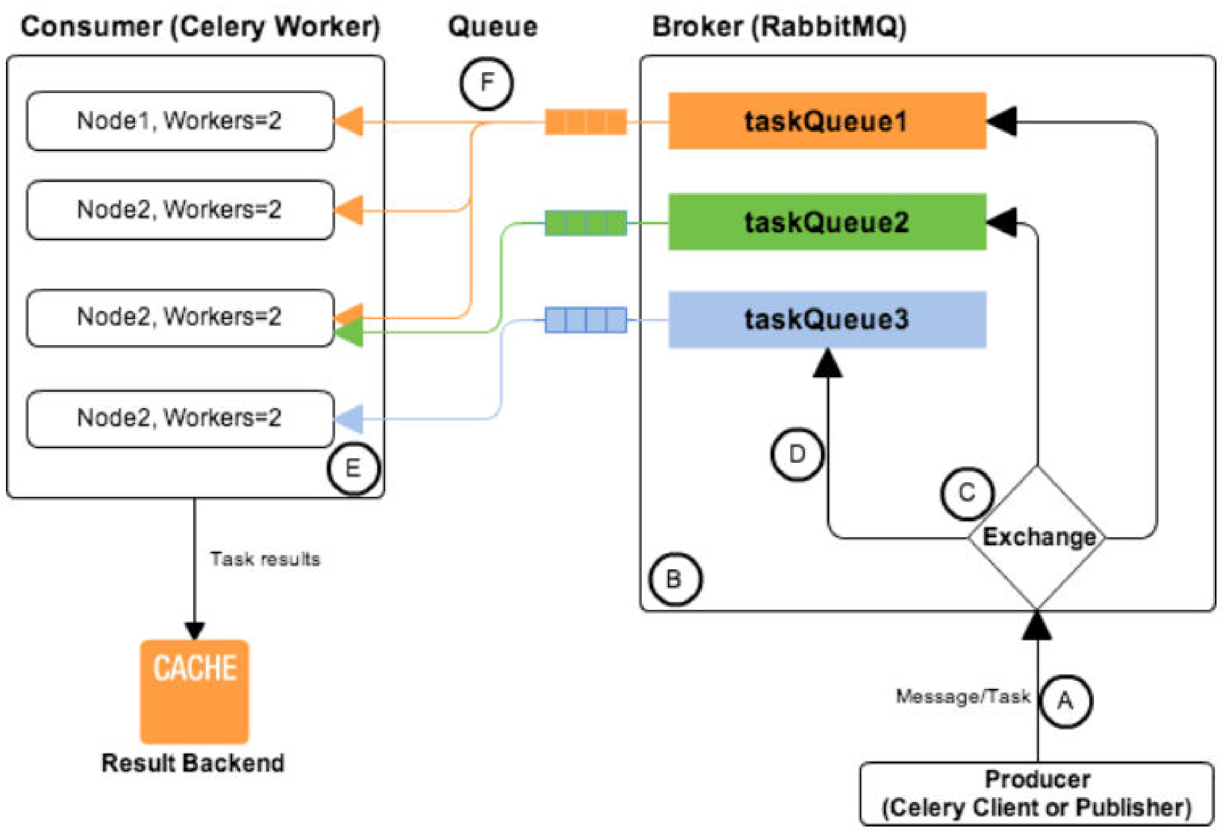
A message or task queue is a buffer that stores messages.

Bindings

Bindings are rules that the exchange uses to route messages to queues.

Prouting Keys

Bindings may have an optional routing key attribute. An exchange may use this field to route a message to the bound queue.



Celery and RabbitMQ

A **consumer** is an application that receives messages and process them.

Celery

Celery generally hides the complexity of AMQP protocols. Celery act as both the producer and consumer of RabbitMQ messages. In Celery, the producer is called client or publisher and consumers are called as workers. It is possible to use a different custom consumer (worker) or producer (client).

RabbitMQ or AMQP message queues are basically task queues.

A

Message originates from a Celery client. The message body contains

- name of the task to execute,
- task id (UUID)
- arguments to execute task with
- additional metadata – like the retries, eta, expires.

An example Celery message,

```
{"id": "4cc7438e-afd4-4f8f-a2f3-f46567e7ca77",
 "task": "celery.task.PingTask",
 "args": [],
 "kwargs": {},
 "retries": 0,
 "eta": "2009-11-17T12:30:56.527191"
}
```

B

RabbitMQ route messages/tasks to one or more queues. Routing of tasks requires:

1. Defining queues using `CELERY_QUEUES` setting. `CELERY_QUEUES` is a map of queue names and their exchange/exchange_type/binding_key.
2. Specifying task destination. The destination for a task is decided by the following (in order)
   a. The Routers defined in `CELERY_ROUTES` setting. `CELERY_ROUTES` is a map of task names and their queue/routing_key.
   b. The routing arguments to Task.apply_async().
   c. Routing related attributes defined in the Task itself.

In the AMQP both `routing_key` and `binding_key` are referred as the routing key.

C

The exchange type defines how the messages are routed through the exchange. The exchange types defined in the standard are direct, topic, fanout and headers.

D

The relationship between exchanges and a queue is called a binding.

E

Normally one server will have one node, but you can run multiple nodes on the same server.

Each node has multiple worker processes or threads. By default multiprocessing is used to perform concurrent execution of tasks.

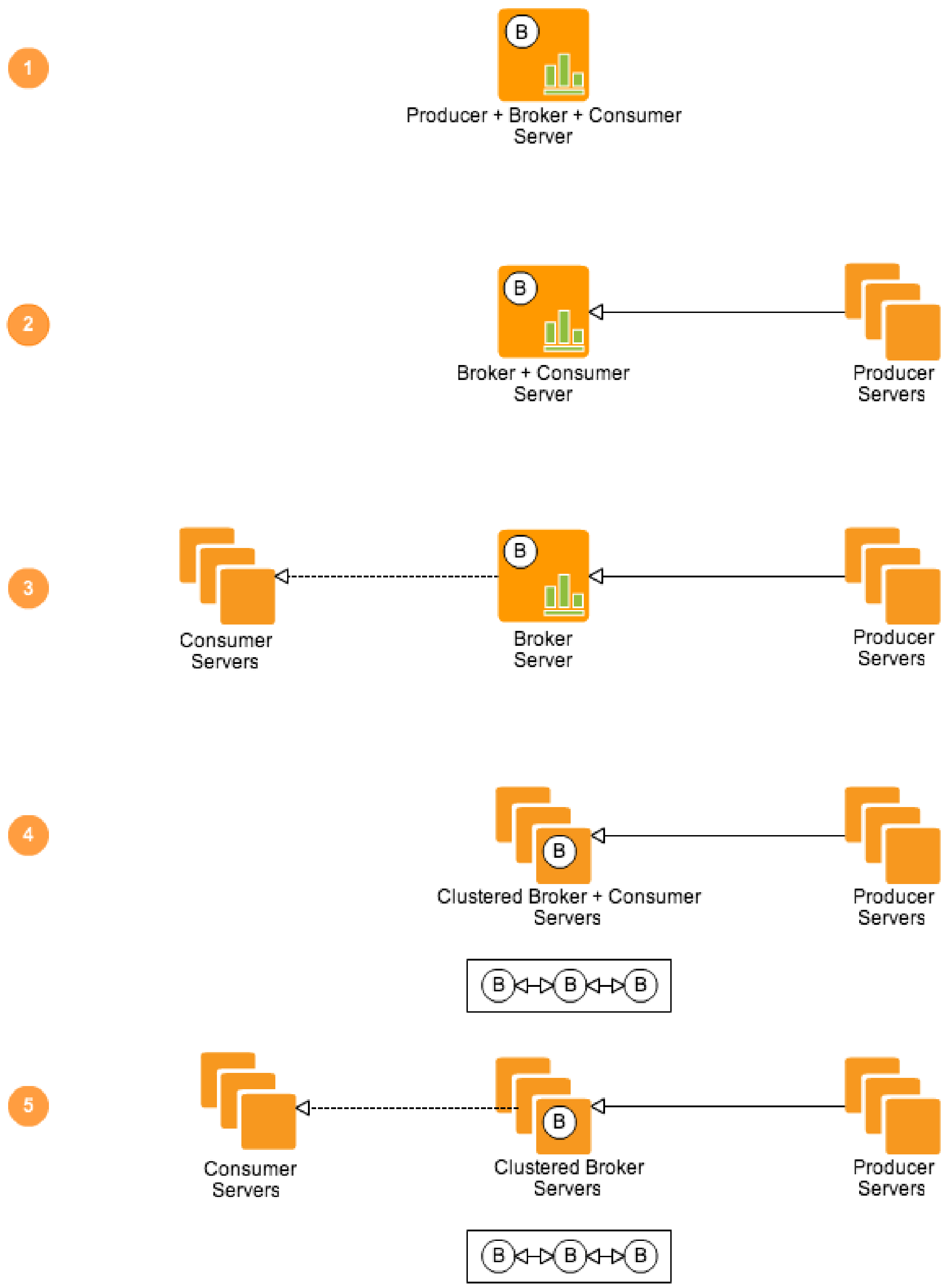
The number of worker processes/threads can be changed using the --concurrency argument and defaults to the number of CPUs available on the machine.

F

Various messaging scenarios are supported between Node to taskQueue -*1:M, M:M, M:1, 1:1, Round-robin etc.*

Producer, Broker, Consumer Arrangements

As AMQP is a network protocol, the producers, consumers and the broker can all reside on the same or different machines. Following are the possible arrangements for producer, broker and consumer,


1
B
Producer + Broker + Consumer Server
2
B
Broker + Consumer Server
Producer Servers
3
Consumer Servers
B
Broker Server
Producer Servers
4
B
Clustered Broker + Consumer Servers
Producer Servers
B B B
5
Consumer Servers
B
Clustered Broker Servers
Producer Servers
B B B