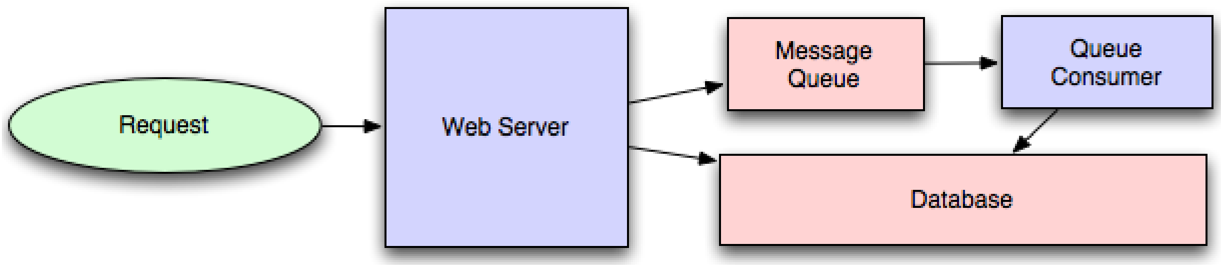
1. Concepts

- Message queuing makes it possible for applications to communicate asynchronously, by sending messages to each other via a queue. A message queue provides temporary storage between the sender and the receiver so that the sender can keep operating without interruption when the destination program is busy or not connected. ***Asynchronous processing*** allows a task to call a service, and *move on to the next task* while the service processes the request at its own pace.
- A **queue** is a line of things waiting to be handled — in sequential order starting at the beginning of the line. A **message queue** is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed.
- A **message** is the data transported between the sender and the receiver application; it's essentially a byte array with some headers on top. An example of a message could be an event. One application tells another application to start processing a specific task via the queue.
- The basic architecture of a **message queue** is simple: there are client applications called **producers** that create messages and deliver them to the message queue. Another application, called a **consumer**, connects to the queue and gets the messages to be processed. Messages placed onto the queue are stored until the consumer retrieves them.
- The queue can provide protection from service outages and failures.
- Examples of queues: Kafka, Heron, real-time streaming, Amazon SQS, and RabbitMQ.



2. Design user interface when MQ is involved

Dividing work between off-line work handled by a consumer and in-line work done by the web application depends entirely on the interface you are exposing to your users.

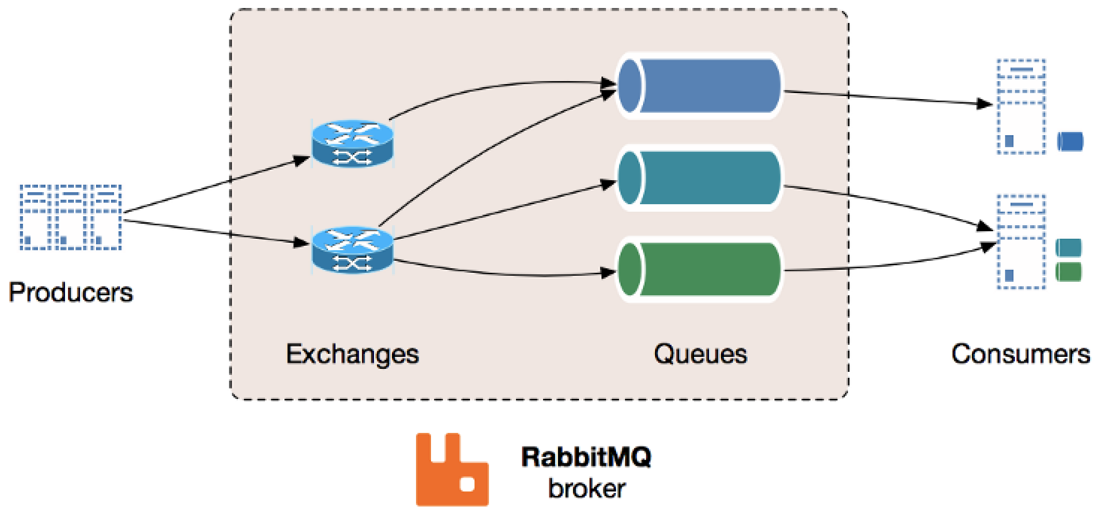
Generally, you'll either:

1. perform almost no work in the consumer (merely scheduling a task) and inform your user that the task will occur offline, usually with a polling mechanism to update the interface once the task is complete (for example, provisioning a new VM on Slicehost follows this pattern), or
2. perform enough work in-line to make it appear to the user that the task has completed, and tie up hanging ends afterward (posting a message on Twitter or Facebook likely follow this pattern by updating the tweet/message in your timeline but updating your followers' timelines out of the band; it's simple isn't feasible to update all the followers for a Scobleizer in real-time).

3. The role of message queuing in a microservice architecture

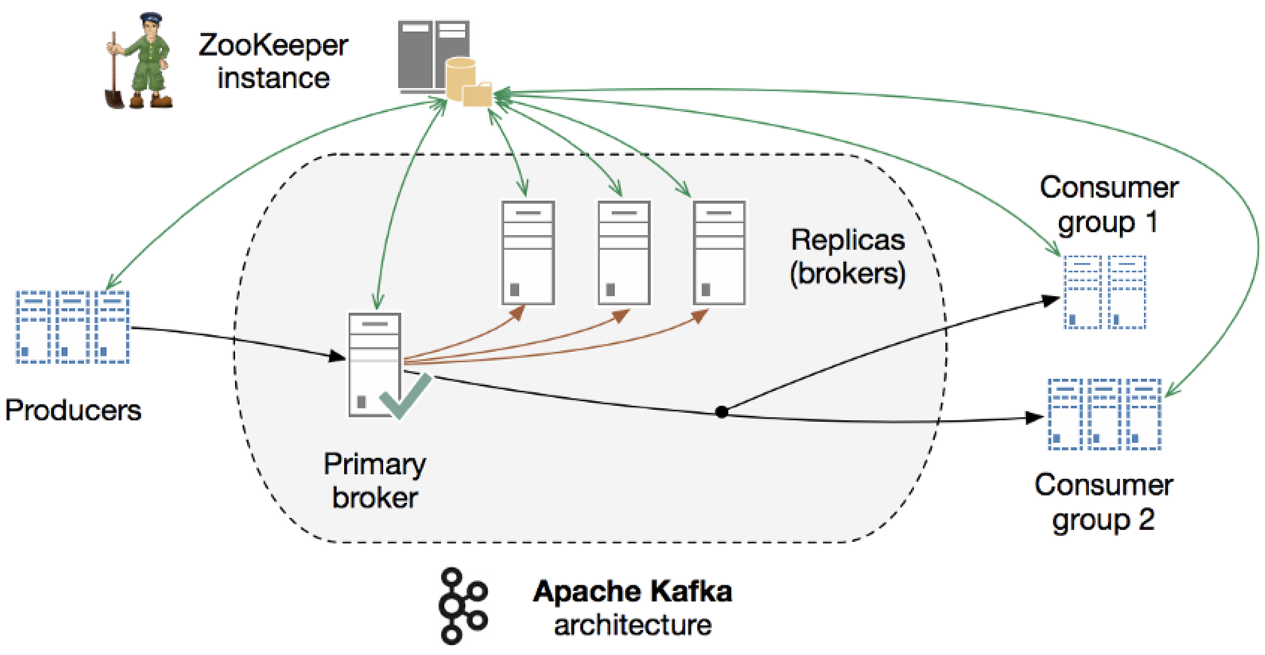
- In a microservice architecture, there are different functionalities divided across different services, that offer various functionalities. These services are coupled together to form a complete software application.
- Typically, in a microservice architecture, there are cross-dependencies, which entail that no single service can perform its functionalities without getting help from other services. **This is where it's crucial for your system to have a mechanism in place which allows services to keep in touch with each other without getting blocked by responses.**
- Message queuing fulfills this purpose by providing a means for services to push messages to a queue asynchronously and ensure that they get delivered to the correct destination. To implement a message queue between services, you need a **message broker, think of it as a mailman**, who takes mail from a sender and delivers it to the correct destination.

4. Message Broker — RabbitMQ



- RabbitMQ is one of the most widely used message brokers, it acts as the message broker, “the mailman”, a microservice architecture needs.
- RabbitMQ consists of:
  1. producer — the client that creates a message
  2. consumer — receives a message
  3. queue — stores messages
  4. exchange — enables to route messages and send them to queues
- The system functions in the following way:
  1. producer creates a message and sends it to an exchange
  2. exchange receives a message and routes it to queues subscribed to it
  3. consumer receives messages from those queues he/she is subscribed to
     One should note that messages are filtered and routed depending on the type of exchange.
- [Use RabbitMQ via CloudAMQP]

5. Apache Kafka



- When to use rabbitqm or apache kafka