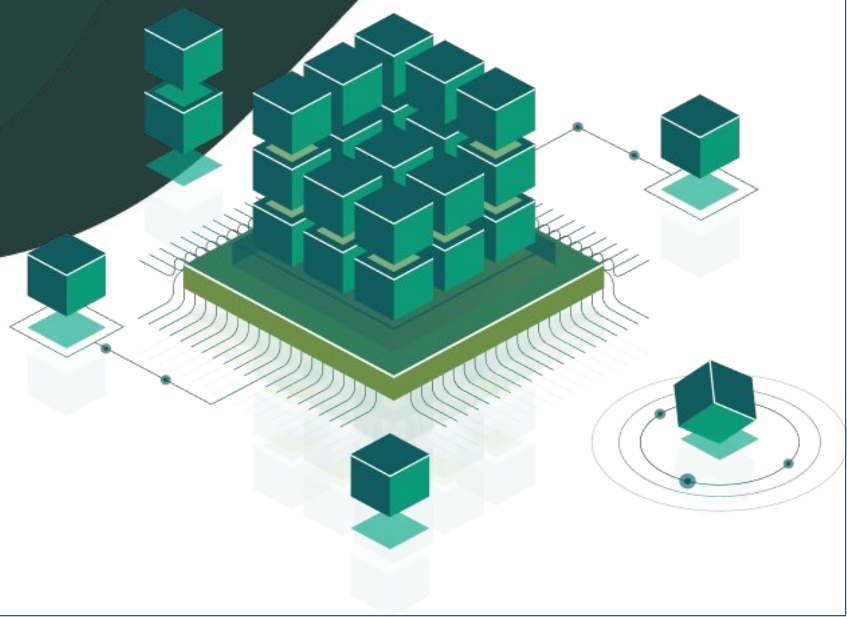
Message Queuing/ Event Bus, Distributed Memory Cache



Speaker:

Ivan Galas

SDE at Magnise

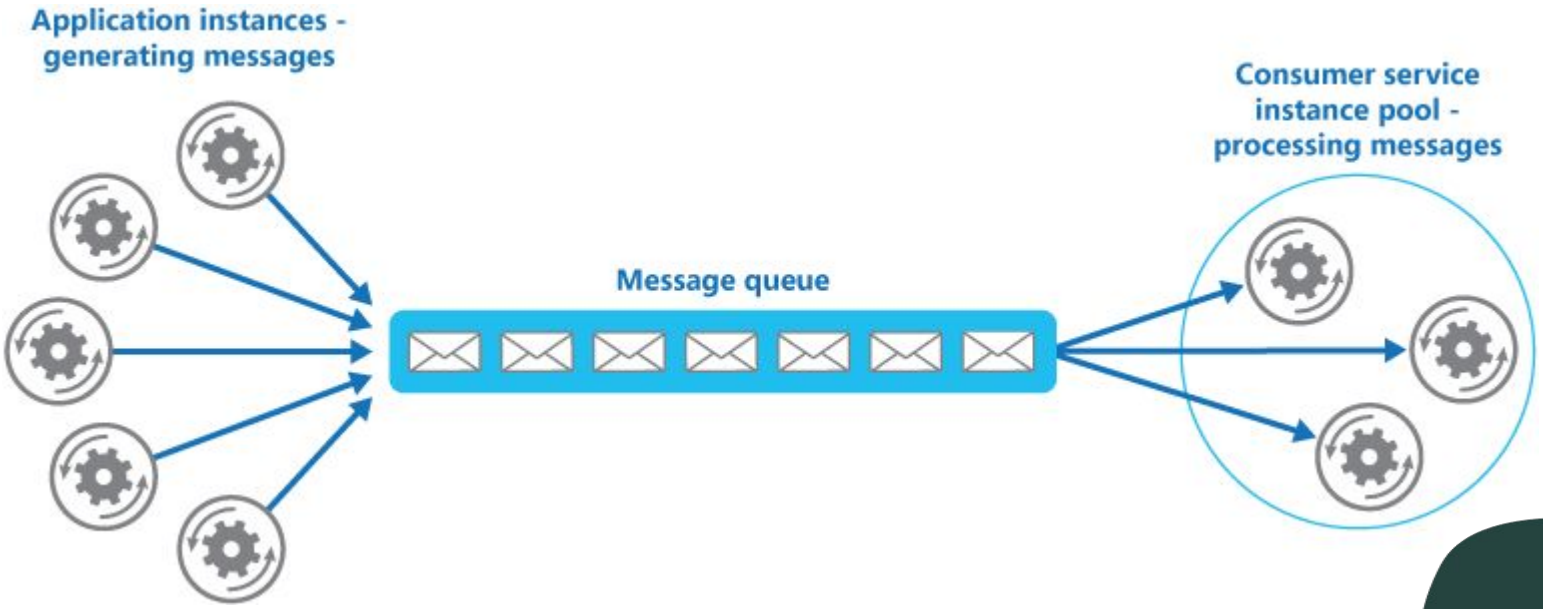
Message Queue

A **message queue** is a software engineering component used for communication between processes or between threads within the same process.

A **message queue** is a form of asynchronous service-to-service communication used in serverless and microservices architectures.

A **message queue** provides a temporary message storage when the destination program is busy or not connected.

Message Queue



Message Queue

Properties:

- Durability (messages may be kept in memory, written to disk, or committed to DB)
- Security policies (access control to messages)
- Message purging policies (queues or messages may have a TTL)
- Message filtering (filtering data on pre-specified by subscriber criteria)
- Delivery policies (message delivering guarantee, exactly one, at-least one)
- Batching policies (group and deliver)
- Queuing criteria (when should a message be considered "enqueued")
- Receipt notification(a publisher may need to know about message receiving)

Message Broker

Properties:

- Route messages to one or more destinations
- Transform messages to an alternative representation
- Perform message aggregation, composing or decomposing
- Interact with an external repository
- Invoke web services to retrieve data
- Respond to events or errors

Event Bus

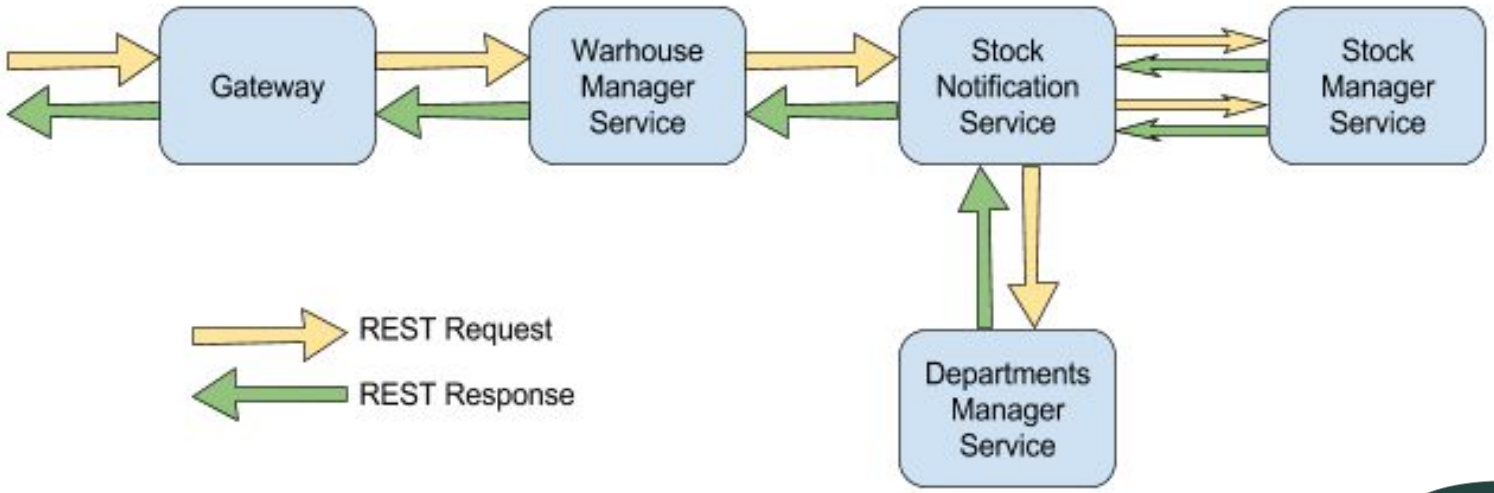
An **Event Bus** is an approach that allows different components to communicate with each other without knowing about each other.

A component can send an event to the event bus without knowing who will pick it up or how many others will pick it up.

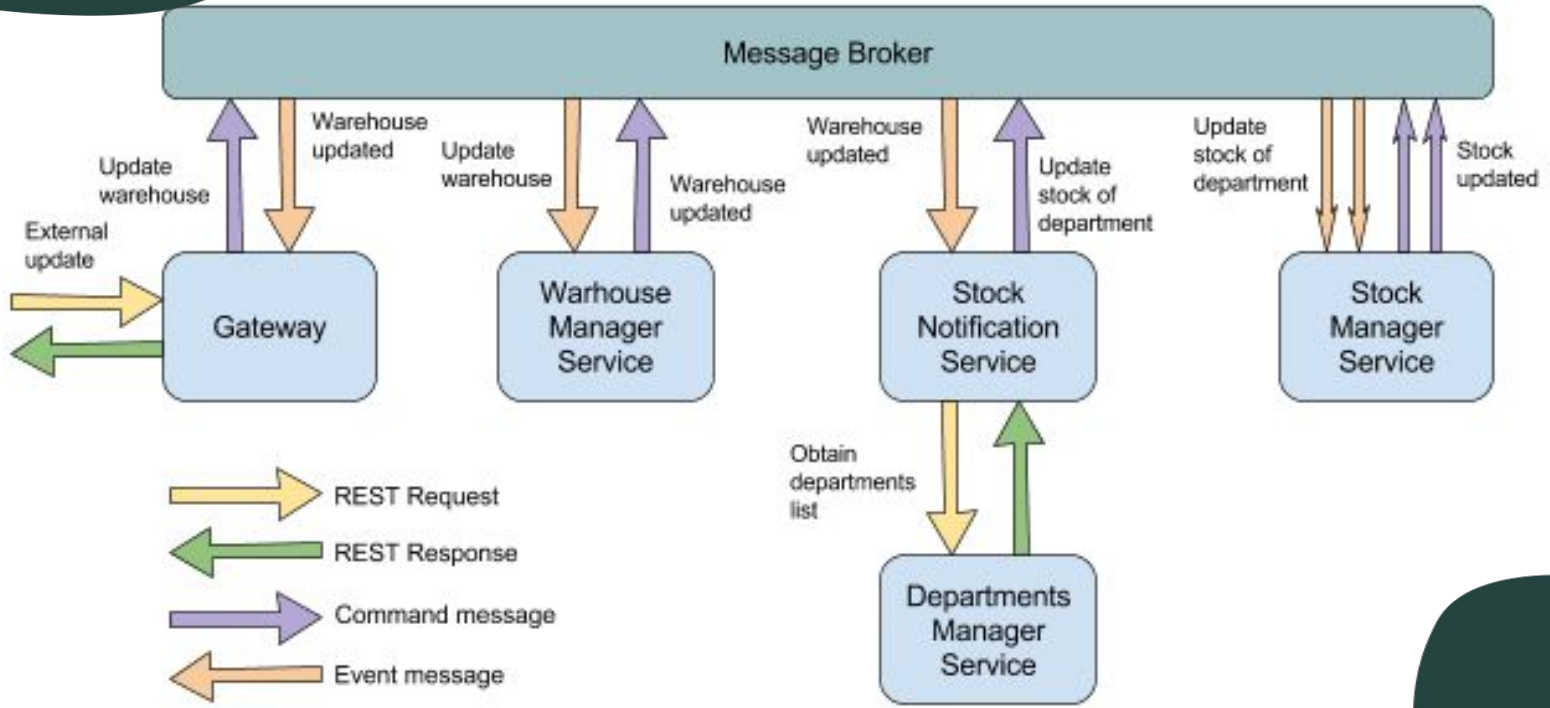
Components can also listen to events on an event bus, without knowing who sent the events.

- components can communicate without depending on each other.
- it is very easy to substitute a component.

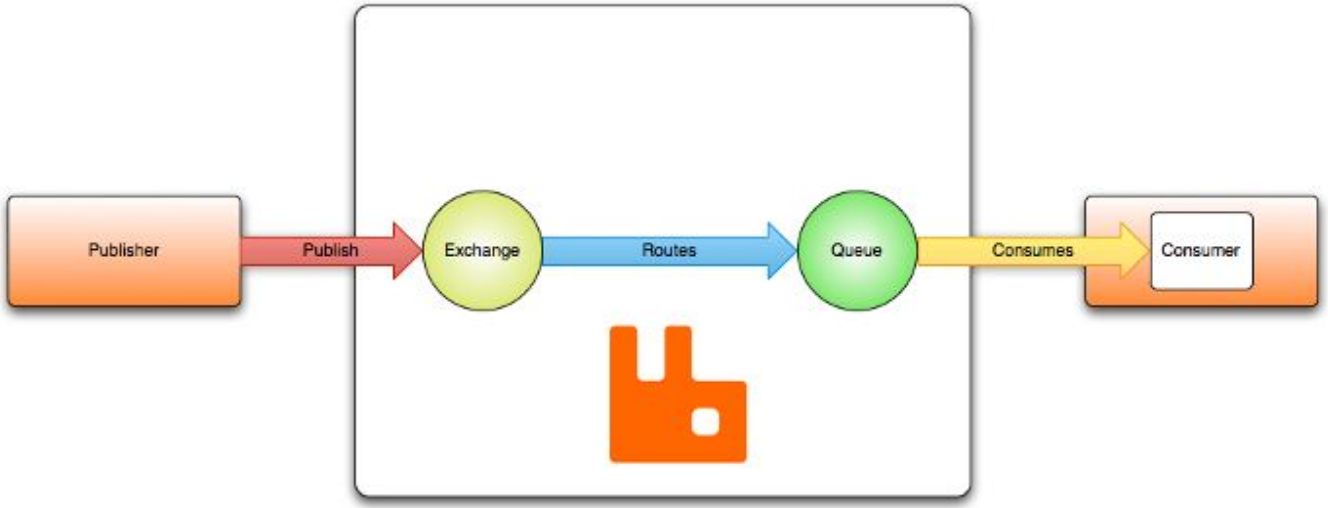
Synchronous architecture



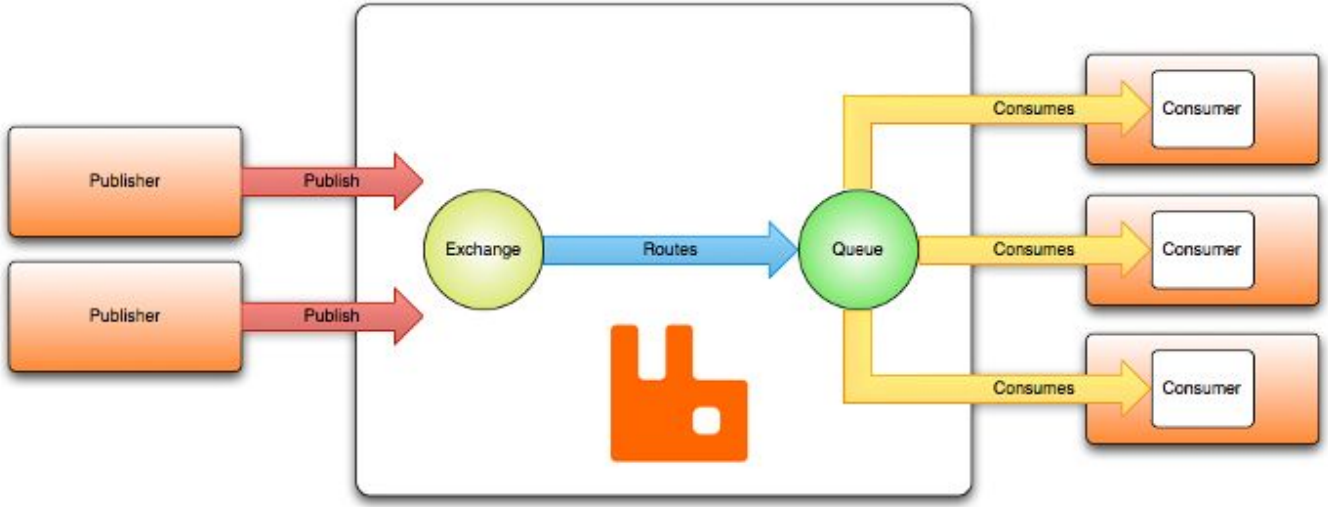
Asynchronous architecture



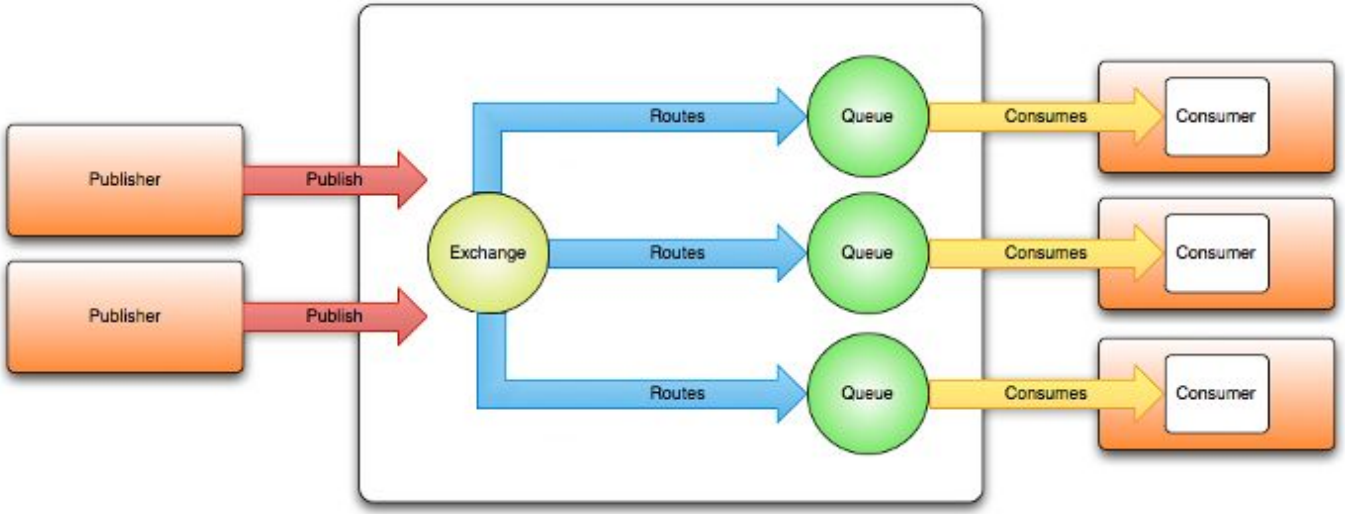
RabbitMQ


Publisher
Publish
Exchange
Routes
Queue
Consumes
Consumer

RabbitMQ



RabbitMQ



RabbitMQ

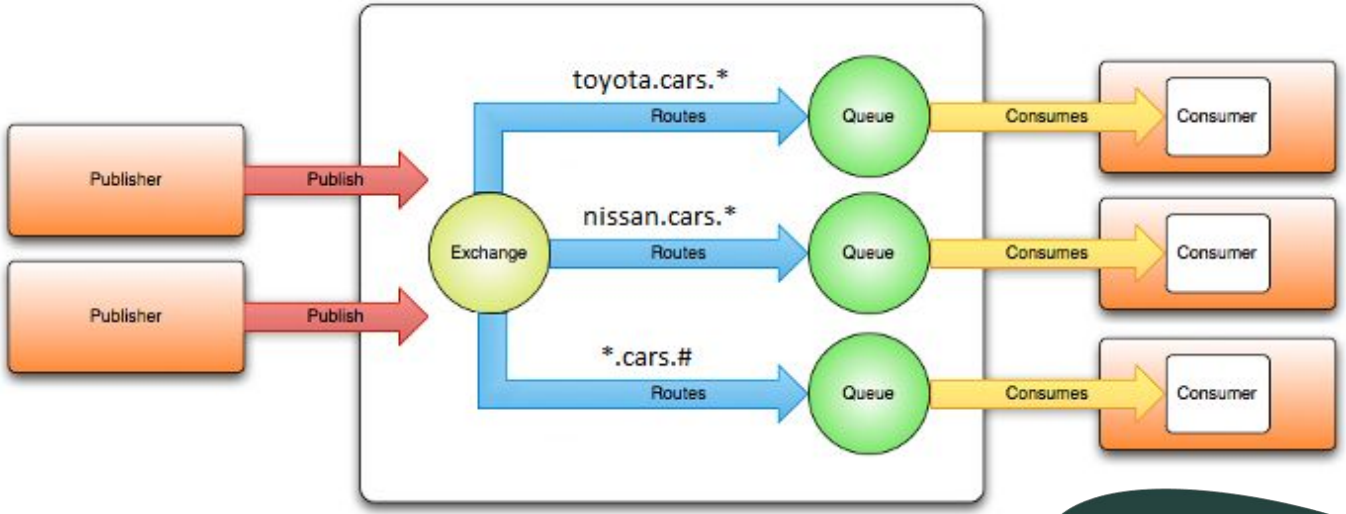
Exchange Types:

- **Direct exchange** (routes messages with a routing key equal to the routing key declared by the binding queue)
- **Fanout Exchange** (routes messages to all bound queues indiscriminately)
- **Topic Exchange**(routes messages to queues whose routing key matches all, or a portion of a routing key)
- **Headers Exchange**(routes messages based upon a matching of message headers to the expected headers specified by the binding queue)

RabbitMQ

Topic Exchange

nissan.cars.japan
nissan.cars
toyota.cars.japan.manufactured
japan.toyota.cars
import.nissan.cars.from.japan
toyota.cars.manufactured
no.latest.cars.toyota

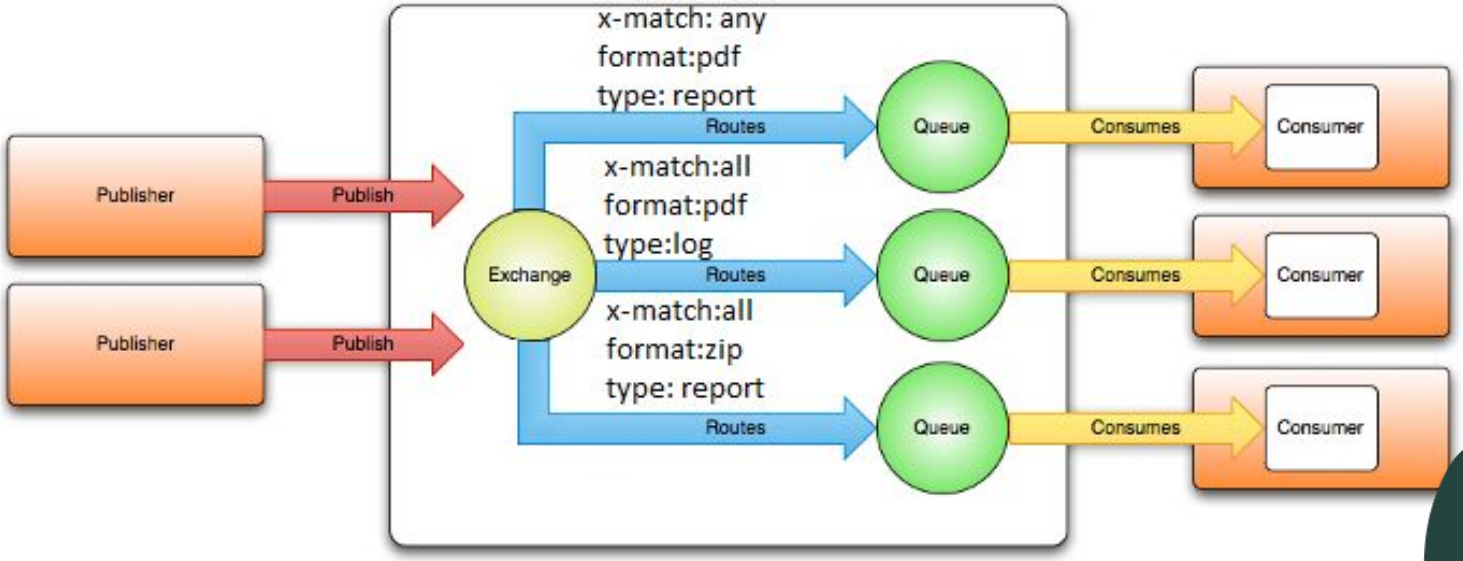


* exactly one word
# zero or more words

RabbitMQ

Header Exchange

'x-match' header:
- all
- any



RabbitMQ

Demo

RabbitMQ and Apache Kafka

The Queue
(RabbitMQ)

VS

The Log
(Apache Kafka)

The Queue

- First In First Out data structure

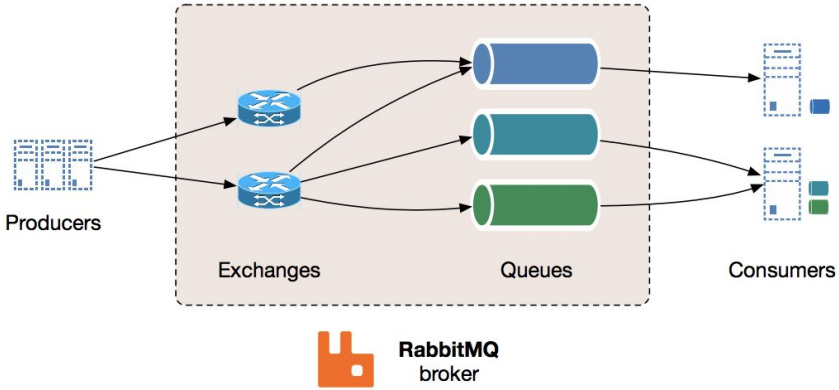
- Optimized for efficient write and read operations from either end of the sequence
- Stored data is transient in nature
- Not shared between applications

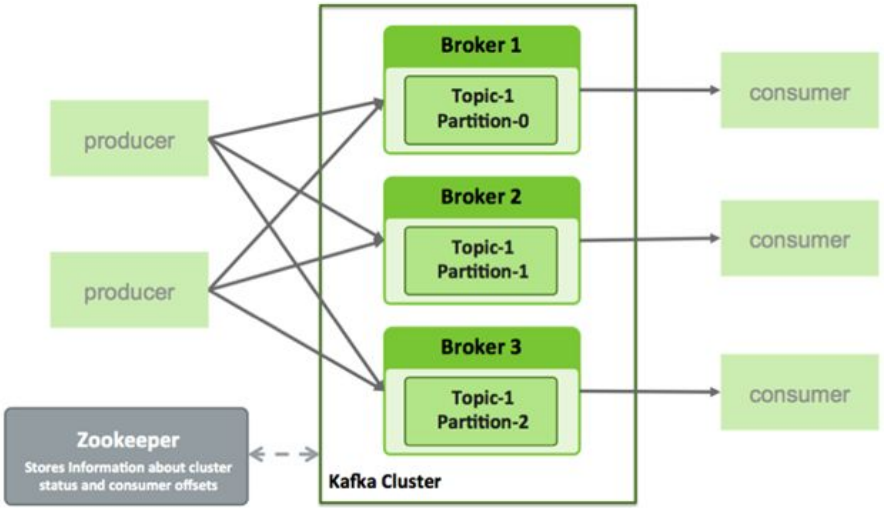
The Log

- Also known as Write-Ahead Logs, Transaction Logs and Commit Logs.
- Append-only. Ordered By Time.
- Unique log sequence numbers.
- Stored data is persistent in nature
- Shared between applications

magnitude of service
MAGNISE

RabbitMQ and Apache Kafka



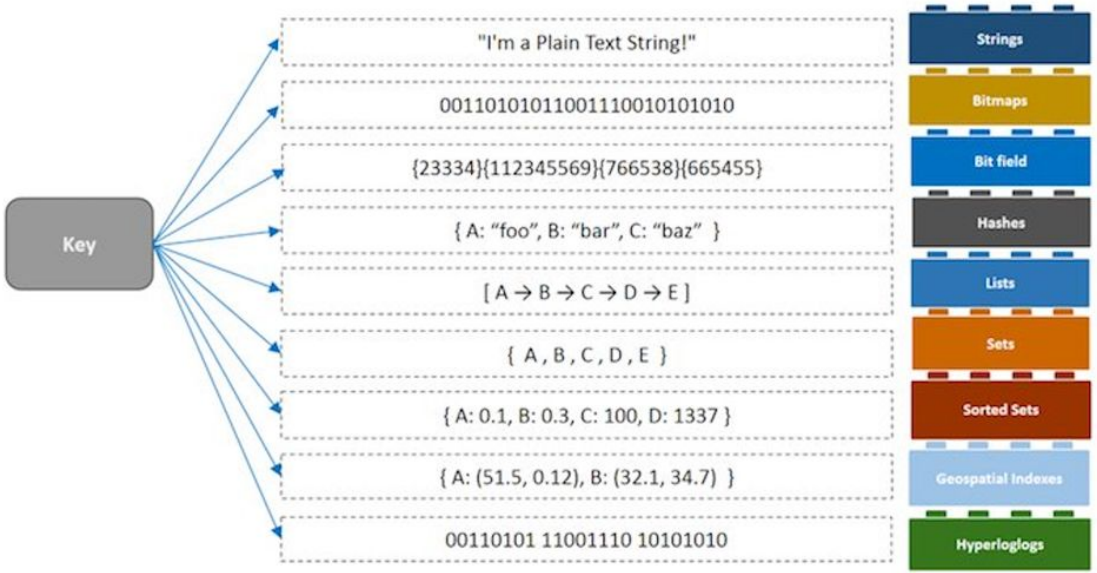


RabbitMQ and Apache Kafka

Property	Apache Kafka	RabbitMQ
Approach	Pull Based	Push Based
Message Ordering	Yes	No
Message LifeTime	Yes	Yes
Delivery Guarantees	Inside partition for whole batch of messages	At least once guarantee
Message Priorities	No	Yes

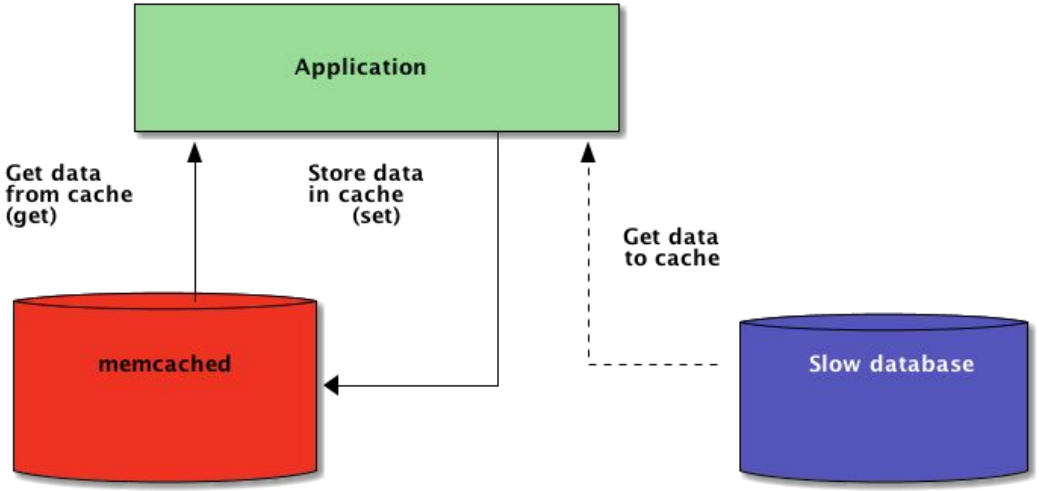
Distributed memory

Redis (remote dictionary server) is an in-memory data structure solution implementing a distributed, in-memory key-value database with optional durability.



Distributed memory

Memcached is a general-purpose distributed memory caching system. It is used for caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.



Distributed memory

Redis vs Memcached

	Redis	Memcached
In-memory	x	x
Virtual-memory	x	
Persist data to disk	x	
Dataset replication	x	
Authentication	x	x
Strong Authentication		x
Simple key-values	x	x
Key enumeration	x	
Data structures	x	
Channel pub/sub	x	
Atomic operations	x	x

Thank you

magnitude of service
MAGNISE