Back To Module Home

Basic Building Blocks for Modern System Design

0% completed

Search Module

Introduction to Building Blocks

Domain Name System

Load balancers

Cache

Databases

Key-value Store

Content Delivery Network (CDN)

Sequencer

Distributed Monitoring

Distributed Cache

Distributed Messaging Queue

System Design: The Distributed Messaging Queue

Requirements of a Distributed Messaging Queue’s Design

Considerations of a Distributed Messaging Queue’s Design

Design of a Distributed Messaging Queue: Part 1

Design of a Distributed Messaging Queue: Part 2

Evaluation of a Distributed Messaging Queue’s Design

Quiz on the Distributed Messaging Queue’s Design

Pub-sub

Rate Limiter

Design of a Distributed Messaging Queue: Part 1

Learn about the high-level design of a messaging queue and how to scale the metadata of queues.

We'll cover the following

So far, we've discussed the requirements and design considerations of a distributed messaging queue. Now, let's begin with learning about the high-level design of a distributed messaging queue.

Distributed messaging queue#

Unlike a single-server messaging queue, a distributed messaging queue resides on multiple servers. A distributed messaging queue has its own challenges. However, it resolves the drawbacks of a single-server messaging queue if designed properly.

The following sections focus on the scalability, availability, and durability issues of designing a distributed messaging queue by introducing us to a more fault-tolerant architecture of a messaging queue.

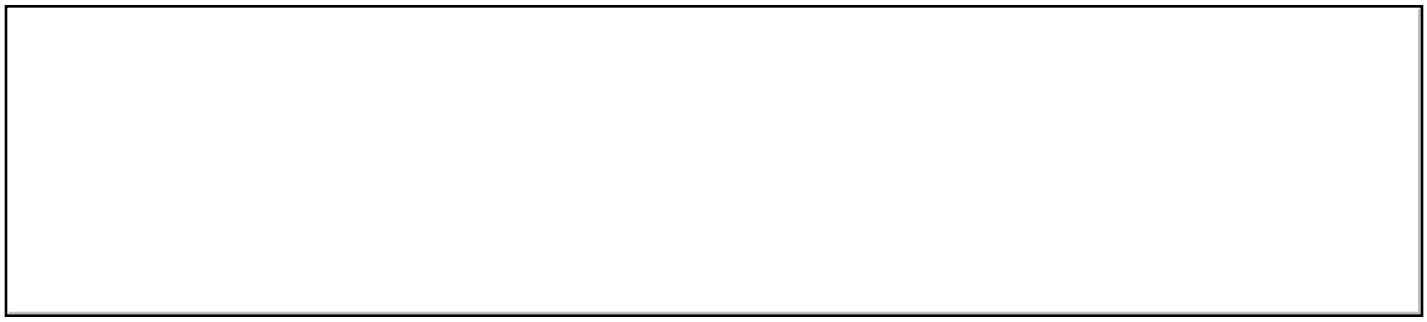
High-level design#

Before diving deep into the design, let's assume the following points to make the discussion more simple and easy to understand. In the upcoming material, we discuss how the following assumptions enable us to eliminate the problems in a single-server solution to the messaging queue.

1. Queue data is replicated using either a primary-secondary or quorum-like system inside a cluster (read through the Data Replication lesson for more details). Our service can use data partitioning if the queue gets too long to fit on a server. We can use a consistent hashing-like scheme for this purpose, or we may use a key-value store where the key might be the sequence numbers of the messages. In that case, each shard is appropriately replicated (refer to the Partition lesson for more details on this).

2. We also assume that our system can auto-expand and auto-shrink the resources as per the need to optimally utilize resources.

The following figure demonstrates a high-level design of a distributed messaging queue that's composed of several components.



High-level architecture of distributed messaging queue

The essential components of our design are described in detail below.

Load balancer#

The load balancer layer receives requests from producers and consumers, which are forwarded to one of the front-end servers. This layer consists of numerous load balancers. Therefore, requests are accepted with minimal latency and offer high availability.

Front-end service#

The front-end service comprises stateless machines distributed across data centers. The frontend provides the following services:

- **Request validation:** This ensures the validity of a request and checks if it contains all the necessary information.
- **Authentication and authorization:** This service checks if the requester is a valid user and if these services are authorized for use to a requester.
- **Caching:** In the front-end cache, metadata information is stored related to the frequently used queues. Along with this, user-related data is also cached here to reduce request time to authentication and authorization services.
- **Request dispatching:** The frondend is also responsible for calling two other services, the backend and the metadata store. Differentiating calls to both of these services is one of the responsibilities of the frontend.

Request deduplication: The frontend also tracks information related to all the requests, therefore, it also prevents **identical requests** from being put in a queue. Deciding what to do about duplicates might be as easy as searching a hash key in a store. If something is found in the store, this implies a duplicate and the message can be rejected.

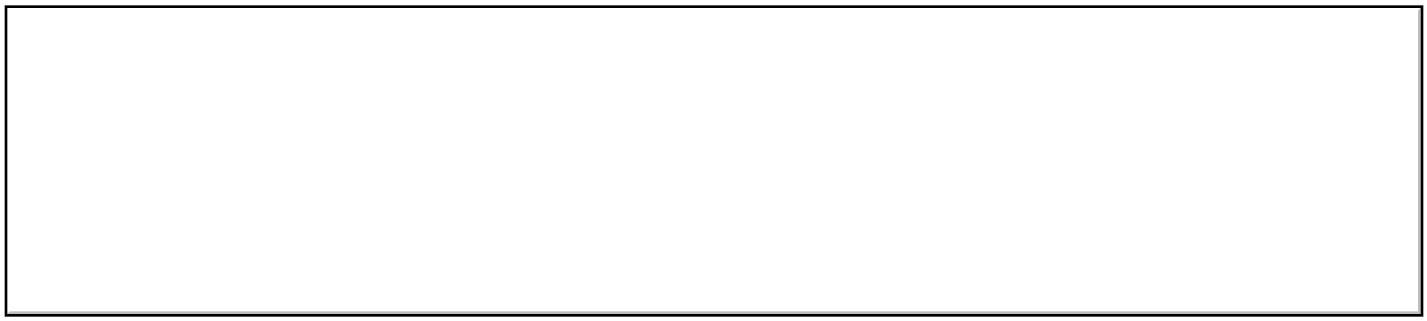
- **Usage data collection:** This refers to the collection of real-time data that can be used for audit purposes.

Metadata service#

This component is responsible for storing, retrieving, and updating the metadata of queues in the metadata store and cache. Whenever a queue is created or deleted, the metadata store and cache are updated accordingly. The metadata service acts as a middleware between the front-end servers and the data layer. Since the metadata of the queues is kept in the cache, the cache is checked first by the front-end servers for any relevant information related to the receipt of the request. If a cache miss occurs, the information is retrieved from the metadata store and the cache is updated accordingly.

There are two different approaches to organizing the metadata cache clusters:

1. If the metadata that needs to be stored is small and can reside on a single machine, then it's replicated on each cluster server. Subsequently, the request can be served from any random server. In this approach, a load balancer can also be introduced between the front-end servers and metadata services.

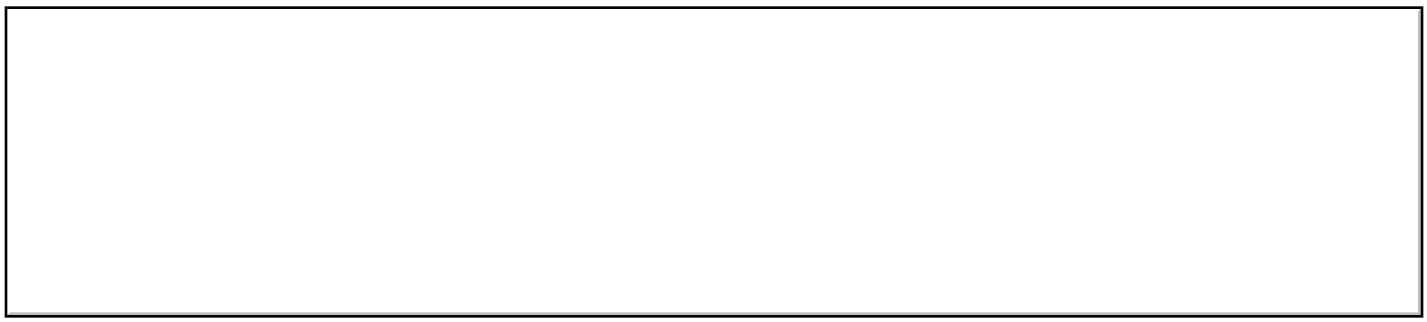


A load balancer is introduced between the front-end servers and metadata services

2. If the metadata that needs to be stored is too large, then one of the following modes

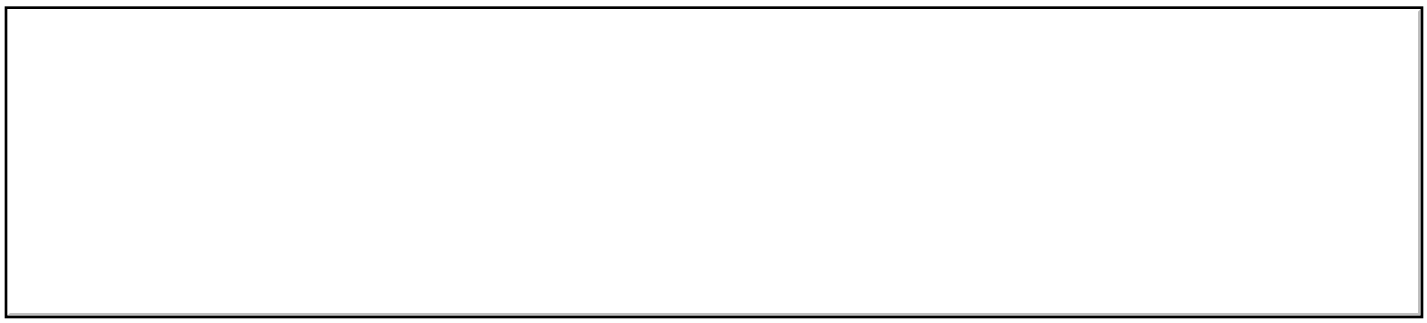
can be followed:

- The first strategy is to use the sharding approach to divide data into different shards. **Sharding** can be performed based on some partition key or hashing techniques, as was discussed in the lesson on database partitioning. Each shard is stored on a different host in the cluster. Moreover, each shard is also replicated on different hosts to enhance availability. In this cluster-organization approach, the front-end server has a mapping table between shards and the hosts. Therefore, the front-end server is responsible for redirecting requests to the host where the data is stored.



Mapping table resides on the front-end servers

- The second approach is similar to the first one. However, the mapping table in this approach is stored on each host instead of just on the front-end servers. Because of this, any random host can receive a request and forward it to the host where the data resides. This technique is suitable for read-intensive applications.



Mapping table resides on each metadata server

In our discussion on distributed messaging queues, we focused on the high-level design of this type of queue. Furthermore, we explored each component in the high-level design, including the following:

- Front-end servers and the services required of them for smooth operations.
- Load balancers.
- Metadata services.
- Metadata clusters and their organization.

A vital part of the design is the organization of servers at the backend for queue creation, deletion, and other such operations. The next lesson focuses on the organization of back-end servers and the management of queues, along with other important operations related to message delivery and retrieval.

Considerations of a Distributed Mess…

Next

Design of a Distributed Messaging Qu…

Mark as Completed

Report an Issue