Kafka

Contents

Script

Scaler has a huge number of learners who use the platform parallely, and do multiple activities like during live lectures they ask questions, send chat messages, give reactions, at other times they solve problems for assignment and homework, check dashboard, check leaderboard, order swags from Scaler store and all these activities for every student will need to be tracked and monitored. These actions served as inputs for an array of applications on the backend such as machine learning systems, search optimizers, and report generators which all play an important role in enriching the user experience.

Other information like device from which the student is logging in, probably location updates, and more such information, what would be a correct way to develop a system which can take all these information and store them for future reference. This monitoring and storing of data should not impact the performance of the application and students should not feel any lag or slowness in the system.

Scaler wants to increase the number of students on the platform, you have to design a system which should also work when the load becomes 10X the current load.

What to do now?

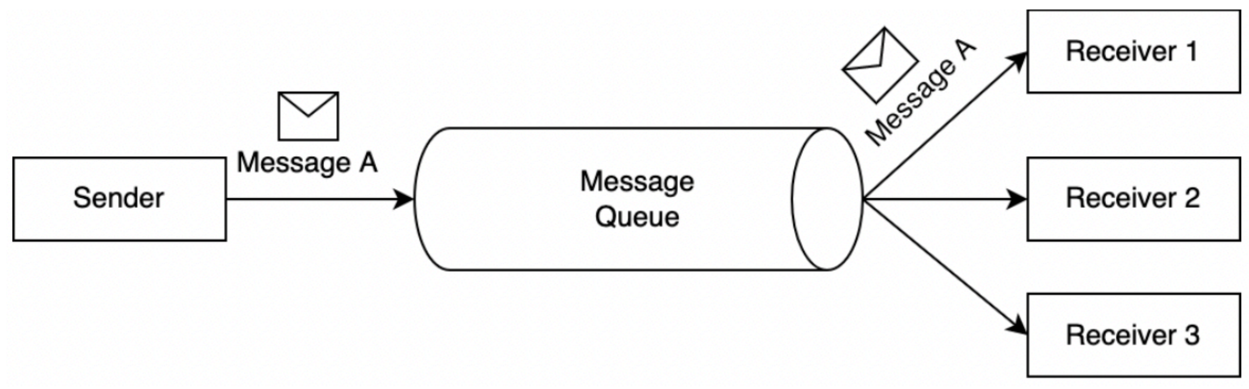
Of course, you talk to some of your fellow engineers or read something online to find the solution, i.e., Setup a Messaging Queue. It can help you create a process to consume all the activities done by students on the website and store them without impacting the performance of the actual system.

Messaging Queue

A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures. Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer. Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.

We can have single of multiple senders sending messages and multiple receivers accepting the message as well, messaging queue works asynchronously and helps in transferring the data between multiple services.

An example of a single sender and multiple receiver via messaging queue is shown below.



There are multiple messaging queues available to be used like :
1. Kafka
2. AWS SQS
3. RabbitMQ
4. Tibco

We would be using Kafka as a messaging queue going ahead.

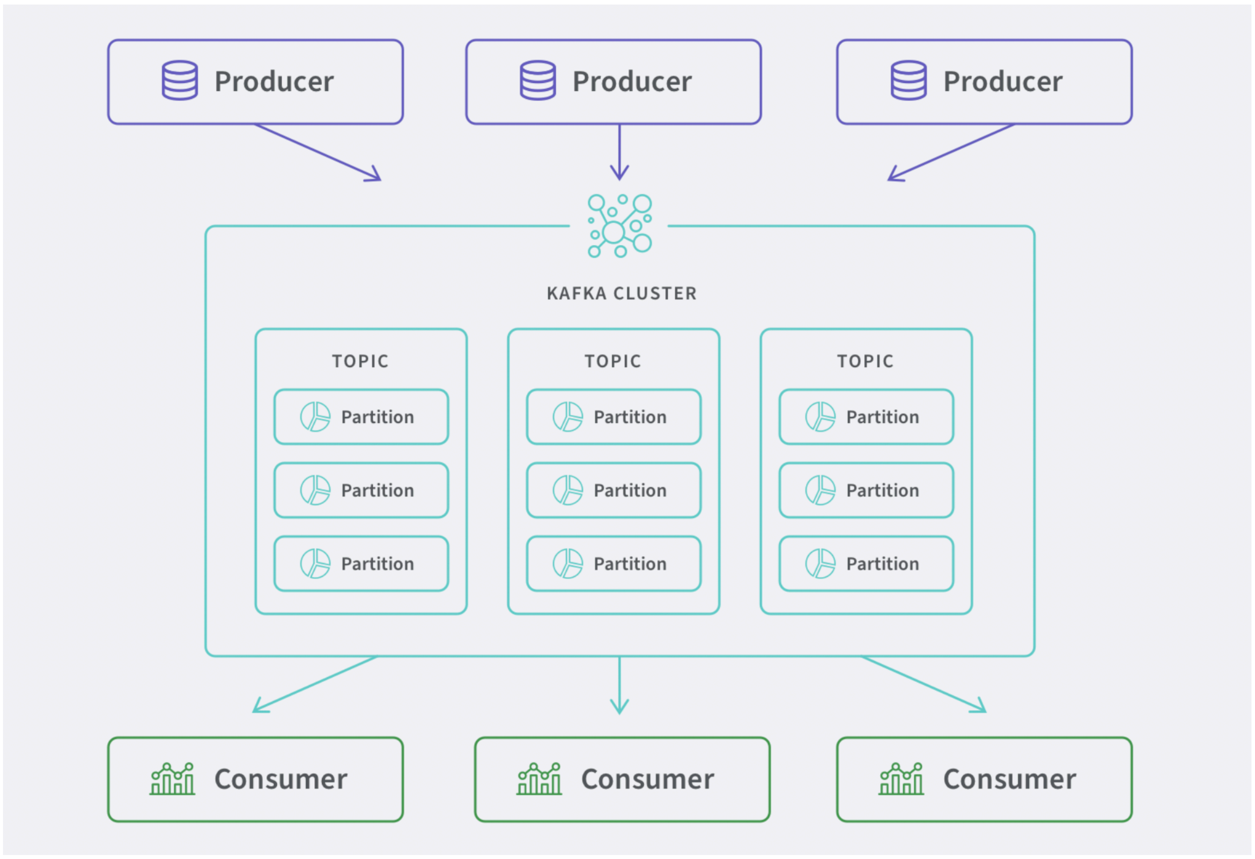
Kafka

Kafka is described by the official documentation as:

A distributed event streaming platform that lets you read, write, store, and process events (also called records or messages in the documentation) across many machines.

At a high level, Apache Kafka allows you to publish and subscribe to streams of records, store these streams in the order they were created, and process these streams in real time.

Now let's dig a bit deeper.



Producer

Client application that push events into topics

Cluster

One or more servers (called brokers) running Apache Kafka

Topic

The method to categorize and durably store events. There are two types of topics: compacted and regular. Records in compacted topics do not expire based on time or space bounds. Newer topic messages update older messages that possess the same key and Apache Kafka does not delete the latest message unless deleted by the user. For regular topics, records can be configured to expire, deleting old data to free storage space.

Partition

The mechanism to distribute data across multiple storage servers (brokers). Messages are indexed and stored together with a timestamp and ordered by the position of the message within a partition. Partitions are distributed across a node cluster and are replicated to multiple servers to ensure that Apache Kafka delivers message streams in a fault-tolerant manner

Consumers

Client applications which read and process the events from partitions. The Apache Kafka Streams API allows writing Java applications which pull data from Topics and write results back to Apache Kafka. External stream processing systems such as Apache Spark, Apache Apex, Apache Flink, Apache NiFi and Apache Storm can also be applied to these message streams

Why ?

Multiple Partitions

By dividing a topic into multiple partitions, Apache Kafka provides load balancing over a pool of servers. This allows you to scale production clusters up or down to fit your needs and to spread clusters across geographic regions or availability zones.

Low Latencies

By decoupling data streams, Apache Kafka is able to deliver messages at network limited throughput using a cluster of servers with extremely low latency (as low as 2ms).

Fault-tolerant cluster and Intra-cluster replication

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

The objective of creating a fault-tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems.

Intra-cluster replication means one cluster replicates its own set of storage with another cluster and its set of storage.

Apache Kafka makes the data highly fault-tolerant and durable in two main ways. First, it protects against server failure by distributing storage of data streams in a fault-tolerant cluster. Second, it provides intra-cluster replication because it persists the messages to disk.

Examples

We will build two different projects to understand the work and setup of Kafka.

1. Basic setup where we will have one producer and one consumer, both pushing and consuming messages from a single Kafka topic. We will build and run this from our terminal.

2. We will try to create basic design for a distributed system containing different micro-services, we will build a online test consultation service, which will have the following micro services :

- Dashboard for users to book a test -> BookingService
- Service which will accept bookings and allocate a lab for each test -> LabService
- Service which consume the user booking and store in DB for analytics purposes -> DataService

  BookingService will produce data and push messages to a Kafka topic, LabService and DataService will consume the messages from the kafka topic and do their respective tasks.