SYSTEM ARCHITECTURE DOCUMENT

TWITTER LIKE SERVICE APP

20, DEC 2022

TWITTER LIKE SERVICE SYSTEM ARCHITECTURE DOCUMENT

CATALOG

1. Introduction

Welcome to the system architecture document for our Twitter-like Application Service ! In this document, we will provide an overview of the various components that make up our service and how they interact with one another. We will also describe the technology stack that we have chosen to build our service and explain why we believe it is the best fit for our needs.

Our service is designed to allow users to post short messages, called "tweets," and interact with each other through likes, comments, and retweeting. It also provides a way for users to discover and follow other users, as well as search for tweets based on keywords or hashtags.

To support these features, we have implemented a robust, scalable, and highly available architecture that can handle the high volume of traffic and data that our service generates. We have also placed a strong emphasis on security, both in terms of protecting our users' data and ensuring the integrity of our service.

2. Overview

In this system architecture document, we will provide a high-level overview of the various components that make up our application and how they interact with one another. We will also describe the technology stack that we have chosen to build our application and explain why we believe it is the best fit for our needs.

Overall, our goal is to provide a comprehensive and detailed view of the inner workings of our application, so that you can understand how it is built and how it operates. We hope this document will give you a better understanding of the design decisions that went into building our application and how they contribute to its overall performance and reliability.

3. ARCHITECTURE COMPONENTS

The components of our Twitter-like application can be broadly divided into three categories: frontend, backend, and infrastructure.

3.1.Frontend Components:

3.1.1.Web Frontend: The web frontend is a component in a Twitter-like service system architecture that provides the user interface for the service. **:**The web frontend is a web application that users will interact with to view and post tweets. The web frontend will be implemented as a web application(Using React web framework) and a mobile app (Java / Kotlin) and is responsible for rendering the content and functionality of the service to the user.

Here is how the web frontend might work in a Twitter-like service system architecture:

i. When a user accesses the Twitter-like service using a web browser or a mobile app, the web frontend is loaded onto the user's device.

ii. The web frontend communicates with the API server to retrieve data and functionality from the backend services.

iii. The web frontend renders the content and functionality of the service to the user, allowing them to view their feed, post tweets, interact with other users, and perform other actions.

iv. When the user performs an action that requires access to data or functionality, the web frontend sends a request to the API server.

v. The API server processes the request and routes it to the appropriate backend service for processing.

vi. The backend service processes the request and sends a response back to the API server.

vii. The API server formats the response and sends it back to the web frontend.

The web frontend is an important component in a Twitter-like service system architecture because it provides the interface through which users interact with the service. The web frontend shall be designed for usability, performance, and scalability to ensure that the service can provide a seamless and enjoyable experience to users.

3.1.2.Load balancer: A load balancer is a component in a system architecture that distributes incoming traffic across multiple servers or resources to optimize resource utilization, maximize throughput, minimize response time, and ensure availability of the system.

In a Twitter-like service, the load balancer plays a crucial role in handling the large volume of incoming traffic and requests from users. It receives incoming requests from clients, and then forwards them to one of the available servers based on a load balancing algorithm. This helps to evenly distribute the load among the servers and prevent any single server from becoming overwhelmed.

The load balancer can also detect when a server is unavailable or experiencing high levels of traffic, and redirect requests to other servers to ensure continuous availability of the service.

Overall, the load balancer is an important component in a Twitter-like service system architecture as it helps to ensure reliable and efficient handling of incoming traffic and requests.

3.2.Backend Components:

3.2.1.API Server: The load balancer distributes incoming requests to multiple instances API server:
The API (Application Programming Interface) server is a core component in a Twitter-like service system architecture that exposes a set of APIs that clients can use to interact with the service. The API server acts as a gateway between the clients and the backend services, handling requests from clients and routing them to the appropriate backend services for processing. It handles requests to create and retrieve tweets and persists the tweets to a database. The API server also uses a cache to store the most recent tweets, allowing for faster retrieval of tweets.f the API server. This allows the system to scale horizontally and handle a large number of requests.

Here is how the API server would work in a Twitter-like service system architecture:

i. When a client (such as a mobile app or a web application) makes a request to the Twitter-like service, the request is sent to the API server.

ii. The API server receives the request and processes it according to the API specification. This may involve verifying the authenticity of the request, checking the permissions of the client, and validating the input data.

iii. If the request is valid, the API server routes the request to the appropriate backend service for processing. For example, if the request is to post a tweet, the API server might route the request to the tweet storage service.

iv. The backend service processes the request and sends a response back to the API server.

v. The API server formats the response according to the API specification and sends it back to the client.

vi. The API server is an important component in a Twitter-like service system architecture because it allows clients to access the service's functionality and data in a standardized and secure way. It also enables the service to decouple the frontend clients from the backend services, making it easier to scale and maintain the service.

It is important to note that the cache is a temporary storage location and the data stored in the cache may become stale over time. To ensure that the cache stays up-to-date, the service will implement a cache invalidation strategy that removes stale data from the cache on a regular basis.

Amazon API Gateway: Is a fully managed service that makes it easy to create, publish, maintain, monitor, and secure APIs at any scale. It supports a wide range of protocols, including HTTP, HTTPS, WebSocket, and HTTP/2, and it can be used to build a variety of API-based applications.

In the context of a Twitter-like service, Amazon API Gateway will be used to build real-time, bidirectional communication channels between the platform and its users. This will be achieved using the WebSocket protocol, which allows for the creation of a persistent, low-latency connection between a client and a server.

3.2.2.Database: The database stores user data and tweet data. The user data includes information about each user's followers, and the tweet data includes the text of each tweet as well as the user who posted it. The database is designed to be scalable and fault-tolerant. The database will be implemented using various types of database technologies, such as relational databases, NoSQL databases, and in-memory databases.

Here is how the database might work in a Twitter-like service system architecture:

i. When a client (such as a mobile app or a web application) makes a request to the Twitter-like service that requires access to data, the request is routed to the appropriate backend service.

ii. The backend service processes the request and generates a database query to retrieve the necessary data from the database.

iii. The database executes the query and returns the data to the backend service.

iv. The backend service processes the data and sends a response back to the client.

Amazon Neptune: Is a fully managed graph database service that makes it easy to build and run applications that use highly connected data. It is designed to be scalable, highly available, and durable, and it supports both property graphs and RDF graphs.

In the context of a Twitter-like service, Amazon Neptune will be used to store and manage data about users, their relationships, and their activity on the platform. This data could be used to build a variety of applications, such as recommendation engines, fraud detection systems, and customer segmentation models.

Amazon DynamoDB: Is a fully managed, NoSQL database service that provides fast and predictable performance with seamless scalability. It is designed to be highly available, flexible, and easy to use, and it supports a wide range of data models and access patterns.

In the context of a Twitter-like service, Amazon DynamoDB will be used to store data about users, their followers, and the tweets they have liked or shared. We could then use this data to build a recommendation engine that suggests content to users based on their interests and activity, or to perform real-time analytics on user behavior and engagement.

Amazon ElastiCache for Redis: Is a fully managed, in-memory data store that makes it easy to deploy, operate, and scale a Redis cache in the cloud. It is designed to be highly available, scalable, and fast, and it can be used to store and manage data that requires low latency and high performance.

In the context of a Twitter-like service, Amazon ElastiCache for Redis will be used to store and manage data that is accessed frequently and needs to be processed in real-time. This could include data such as user profiles, recent activity, or popular content.

Generally, the database is a critical component in a Twitter-like service system architecture because it stores and manages the data that is used by the service. The database should be highly available, scalable, and secure to ensure that the service can handle a high volume of requests and maintain the integrity of the data.

3.2.3.Cloud Platform: The cloud platform will provide the infrastructure for deploying and running the web frontend, API server, database, and cache. It will be a managed service, such as Amazon Elastic Container Service (ECS) or Google Kubernetes Engine (GKE), which will handle the deployment and scaling of the components. We shall proceed with ECS in this case.

Here is how the cloud platform might work in a Twitter-like service system architecture:

i. The cloud platform provides a set of services and infrastructure for hosting and deploying the service, such as virtual machines, container clusters, load balancers, and databases.

ii. The service is deployed on the cloud platform, using the provided infrastructure and services.

iii. The cloud platform manages the infrastructure and services needed by the service, including scaling, monitoring, and security.

iv. The cloud platform also provides tools and services for developing, testing, and deploying the service, such as continuous integration and delivery (CI/CD) pipelines, code repositories, and testing environments.

Amazon Kinesis Data Streams: Is a fully managed, real-time data streaming service that enables you to ingest, process, and analyze large streams of data quickly and at scale. It can be used to build a variety of data processing applications, including real-time analytics, data lakes, and data pipelines.

In the context of a Twitter-like service, Kinesis Data Streams will be used to ingest and process data in real-time as it is generated by users posting updates, interacting with other users, or generating other types of activity on the platform. This data could then be analyzed in real-time to provide insights, generate notifications, or trigger other actions.

Amazon Elastic MapReduce (EMR): Is a fully managed, cloud-based data processing service that makes it easy to process large amounts of data quickly and at scale. It is based on the Apache Hadoop open-source framework and allows you to run a range of data processing workloads, including batch processing, stream processing, and interactive analysis.

In the context of a Twitter-like service, Amazon EMR will be used to process and analyze large volumes of data generated by users posting updates, interacting with other users, or generating other types of activity on the platform. This data could be used to provide insights, generate notifications, or trigger other actions.

Amazon Elasticsearch Service: Is a fully managed, cloud-based search and analytics service that makes it easy to search, analyze, and visualize data in real-time. It is based on the open-source Elasticsearch engine and allows you to index, search, and analyze structured and unstructured data at scale.

In the context of a Twitter-like service, Amazon Elasticsearch Service will be used to build a search and analytics platform that allows users to search for tweets, users, and other types of content on the platform. It could also be used to perform real-time analytics on user behavior and engagement, or to build machine learning models that use data from the platform.

Amazon Simple Storage Service (S3): Is a fully managed, cloud-based storage service that makes it easy to store, retrieve, and manage large amounts of data. It is highly scalable, durable, and secure, and it can be used to store a wide range of data types, including text, audio, video, images, and more.

In the context of a Twitter-like service, Amazon S3 will be used to store, index and manage data generated by users posting updates, interacting with other users, or generating other types of activity on the platform. This data could be used for a variety of purposes, including analytics, machine learning, and data backup and recovery.

Generally, The cloud platform is an important component in a Twitter-like service system architecture because it provides the resources and tools needed to host and manage the service in the cloud. The cloud platform offers many benefits, such as scalability, reliability, security, and cost-effectiveness, which can help the service to handle a high volume of users and traffic.

3.3.Infrastructure Components:

3.3.1.Cache: A cache is a component in a Twitter-like service system architecture that stores frequently accessed data in a temporary storage location in order to reduce the number of requests made to the main data store and improve the performance of the service. This allows for faster retrieval of tweets and reduces the load on the database.. The cache is periodically flushed to the database to ensure that tweets are persisted.

Here is how the cache might work in a Twitter-like service system architecture:

i. When a client (such as a mobile app or a web application) makes a request to the Twitter-like service, the request is first directed to the cache.

ii. The cache checks to see if the requested data is stored in the cache. If the data is present, it is returned to the client.

iii. If the data is not present in the cache, the request is forwarded to the main data store (such as a database) to retrieve the data.

iv. The data store returns the data to the cache, which stores a copy of the data in its temporary storage.

v. The cache then returns the data to the client.

By storing frequently accessed data in the cache, the Twitter-like service can reduce the load on the main data store and improve the performance of the service. The cache can be implemented using various types of storage technologies, such as in-memory databases, disk-based storage, or cloud-based storage.

3.3.2.Message Queue Server: A cloud message queue server is a component in a Twitter-like service system architecture that provides a messaging service for decoupling and asynchronously exchanging messages between different components or services. The message queue server will be used to queue up tweets from users to be delivered to their followers' feeds. The cloud message queue server will be implemented using various types of messaging technologies, such as RabbitMQ, Apache Kafka, or Amazon Simple Queue Service (SQS).

Here is how the cloud message queue server might work in a Twitter-like service system architecture:

i. When a component or service needs to send a message to another component or service, it sends the message to the cloud message queue server.

ii. The cloud message queue server stores the message in a queue until it can be delivered to the recipient.

iii. When the recipient is ready to receive the message, it retrieves the message from the queue.

iv. The cloud message queue server delivers the message to the recipient and marks the message as delivered.

The cloud message queue server is an important component in a Twitter-like service system architecture because it enables the different components and services of the service to communicate and exchange data asynchronously. This can help to improve the performance, scalability, and reliability of the service by decoupling the components and allowing them to operate independently.The cloud message queue server will be used to send notifications to users, process tweets, or perform other tasks asynchronously. This can help to reduce the load on the main components and improve the overall performance of the service.

Overall, these components work together to deliver a seamless and efficient user experience, while also ensuring that our application can scale to meet the needs of our growing user base. In the following sections, we will delve deeper into the details of each component and how they fit into the overall architecture of our application.

3.4.System Architecture Diagram

The following diagram shows the system architecture, including the components and their relationships:

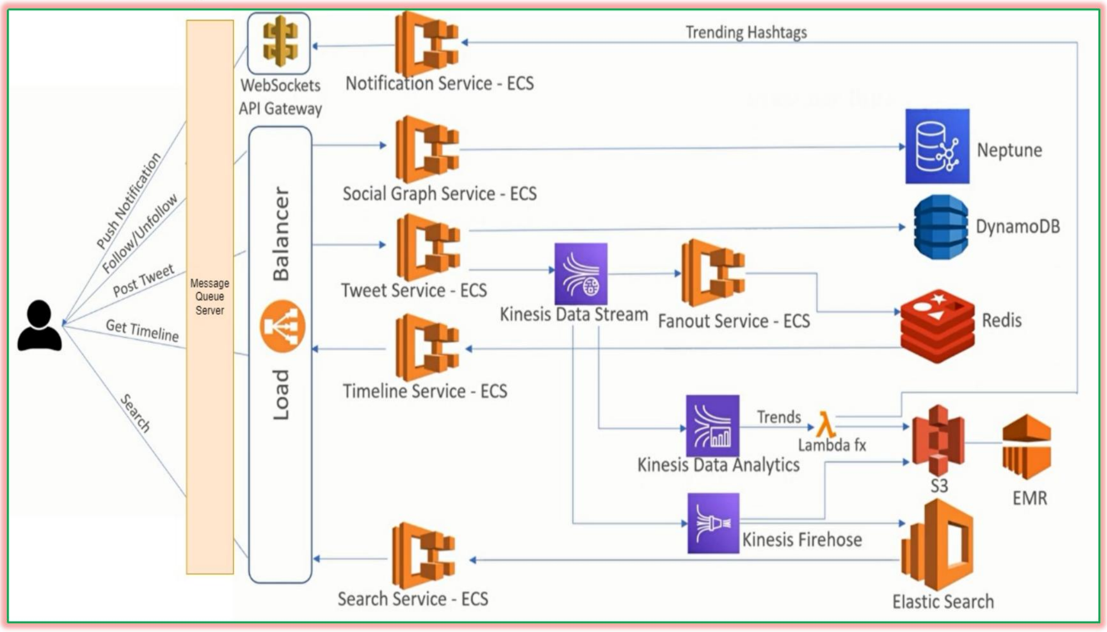

Figure 1: System Architecture Diagram

4.DEPLOYMENT

In this section, we will describe how our Twitter-like application is deployed in production.

Our application is deployed across a number of servers in a cloud computing environment, in this case we shall use Amazon Web Services (AWS) . The load balancer and API server instances will be scaled horizontally to handle a high volume of requests. The database and cache will be run on separate clusters of servers to ensure scalability and fault tolerance. This allows us to scale our application up or down as needed, depending on the volume of traffic and data that it needs to handle.

We use a combination of virtual machines and containerization to deploy our application. Virtual machines allow us to run multiple instances of our application on a single physical server, while containerization allows us to package and deploy our application more efficiently.

We use a configuration management tool, such as Ansible or Chef, to automate the process of deploying and updating our application. This allows us to make changes to our application quickly and consistently, without the need for manual intervention.

We also use a continuous integration and delivery (CI/CD) pipeline to automate the build, test, and deployment process for our application. This allows us to deploy new features and updates to our application quickly and safely, without the need for manual intervention.

Overall, our deployment strategy is designed to provide high availability and scalability for our application, while also allowing us to make changes and updates efficiently and safely

5.MONITORING AND ALERTING

In this section, we will describe how we monitor and alert on the performance and health of our Twitter-like application.

Monitoring and alerting are critical components of the system architecture for our Twitter-like application. They allow us to track the performance and health of our application in real-time, identify issues as they arise, and take corrective action to prevent them from impacting our users.

The system will be monitored using a combination of metrics and logs. Metrics will be collected for the performance of the load balancer, API server, database, and cache, and alerts will be configured to notify the operations team if any component is not performing within acceptable limits. Logs will be collected and aggregated to provide visibility into the system's behavior and to aid in debugging any issues.

We use a combination of monitoring tools and alerts to ensure that our application is operating correctly and performing at its best. These tools allow us to track a wide range of metrics, including system performance, application performance, and user engagement.

We use tools such as Grafana and Datadog to visualize and analyze our monitoring data. These tools provide real-time dashboards and alerts that allow us to identify and resolve issues as they arise.

We have also implemented a system for alerting on critical issues, such as server failures or application outages. We use tools such as PagerDuty to send notifications to our operations team when an issue arises, allowing us to respond quickly and minimize the impact on our users.

Overall, our monitoring and alerting system is designed to give us visibility into the performance and health of our application, allowing us to identify and resolve issues quickly and ensure the best possible user experience.

6.SECURITY

In this section, we will describe the security measures that we have implemented to protect our Twitter-like application and our users' data.

We take security very seriously and have implemented a number of measures to ensure the integrity and confidentiality of our application and our users' data. Some of the key security measures that we have implemented include:

6.1.Encryption: We use encryption to protect the confidentiality of our users' data, both in transit and at rest in the database. This includes SSL/TLS for data in transit and AES for data at rest.All data transmitted between our servers and users' browsers is encrypted using Secure Sockets Layer (SSL) or Transport Layer Security (TLS). This ensures that the data is secure and cannot be intercepted by third parties.

6.2.Authentication: We use a combination of passwords, two-factor authentication and OAuth to ensure that only authorized users can access our application.

6.3.Access controls: We use role-based access controls to ensure that users can only access the data and functions that they are authorized to use.We will also use access control lists (ACLs) to restrict access to sensitive data and resources to authorized users only. This helps to prevent unauthorized access and ensures that only authorized users can access sensitive data.

6.4.Firewalls: We use firewalls to protect our servers and networks from external threatsand unauthorized access. These firewalls are configured to block unauthorized access and traffic, helping to prevent attacks and breaches.

6.5.Audit and Penetration testing: We regularly perform security audits and penetration testing penetration testing to identify and address potential vulnerabilities.

Overall, our security measures are designed to protect the integrity and confidentiality of our application and our users' data, while also ensuring that we can respond quickly to any security threats that may arise.

7.BACKUPS AND DISASTER RECOVERY

In this section, we will describe the measures we have taken to ensure the reliability and availability of our Twitter-like application, including our backups and disaster recovery plan.

We understand the importance of keeping our application running smoothly and continuously, and have implemented a number of measures to ensure its reliability and availability. Some of the key measures we have taken include:

7.1.Regular backups: We shall regularly back up our databases and other critical data to ensure that we can recover from data loss or corruption. We store these backups in a secure, offsite location to protect against data loss due to natural disasters or other catastrophic events. We also use replication to keep multiple copies of our data, which helps to ensure that we always have access to a copy of our data in the event of a disaster.

7.2.Redundancy: We shall implement redundant systems for key components of our application, such as web servers and database servers. We have a plan in place to quickly and efficiently recover our application and restore service to our users. This includes identifying the root cause of the disaster, deploying a backup or replica of our application, and verifying that the application is functioning correctly. This allows us to continue to operate even if one of these systems fails, ensuring high availability for our users.

7.3.Disaster recovery plan: We shall have a comprehensive disaster recovery plan in place to ensure that we can recover from any unexpected events that may disrupt our service. This plan includes procedures for recovering from data loss, system failures, and other unexpected events. We also regularly test our disaster recovery plan to ensure that it is effective and that our team is prepared to respond to any potential disasters. This includes conducting drills and simulations to ensure that we are able to recover from a disaster in a timely and efficient manner.

Overall, our backups and disaster recovery plan are designed to ensure the reliability and availability of our application and protect against data loss and other unexpected events. We regularly review and update our plan to ensure that it remains effective in protecting our application and our users.

8.CONCLUSION

In conclusion, the Twitter-like application system architecture is a complex and highly scalable system that enables users to interact with and share content on the platform. The system is composed of various components and services, including the web frontend, API server, cache, database, cloud message queue server, load balancer, and cloud platform, which work together to provide the functionality and performance required by the service.

The system architecture is designed to be flexible and scalable, allowing the service to handle a high volume of users and traffic without any degradation in performance. It also enables the service to be deployed and managed efficiently in the cloud, leveraging the benefits of cloud technologies such as scalability, reliability, security, and cost-effectiveness.

Overall, the Twitter-like application system architecture is a key factor in the success of the service, providing the foundation on which the service is built and enabling it to deliver a seamless and enjoyable experience to users.