Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Discord Presentation Report

Semester: 7

Module: Distributed Systems Course

Elaborated:
Scripca Lina

Verified:
Gavrilița Mihail

Chișinău, 2023

Introduction

Discord is a free app that targets the gamer community and provides an easy way for friends to connect and communicate within text and voice chats. It was first launched in 2015, and as of 2021 has garnered over 350 million users with 150 monthly active users.

Thus, it is a system that has had to adapt fast to a growing community, which was achieved by means of distributing its components and employing different techniques to ensure its availability under a considerable load.

Functional Requirements

In order to ensure proper functionality and customer satisfaction, the developers had to focus on the main functional requirements within the app, including:

- Allowing users to create/join communities(guilds) through direct links;
- Allow users to communicate via voice, video, and text channels;
- Save text messages indefinitely in a database;
- Maintain a continuous connection with the client in order to send notifications;
- Extract the data of the system usage in order to facilitate further improvement.

There may be additional functionality required, such as game streaming and other user and bot related functionality, but the described above has been mentioned within the developer's blog often as warranting frequent architectural changes.

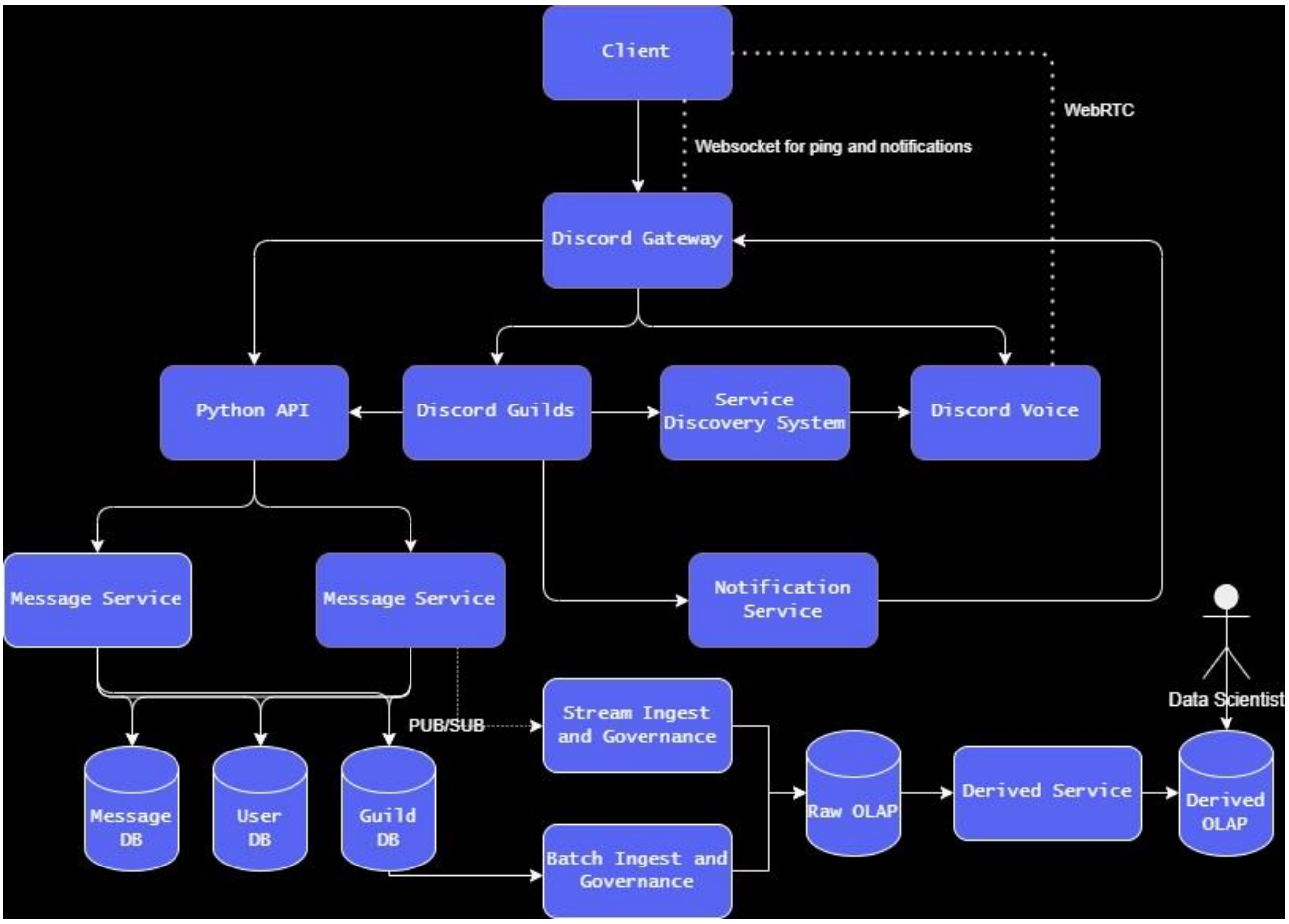
Non-Functional Requirements

Non-functional requirements represent the circumstances and the attributes that the system should work under, in order to be considered to be performing appropriately according to the specifications. In the case of the Discord system, the following requirements should be satisfied:

- Allow support for up to 11 million concurrent users app-wide;
- Allow text communication for up to 200.000 people within a single text channel;
- Allow communication for up to 2.6 million concurrent voice users with egress traffic of more than 220 Gbps (bits-per-second);
- Security against malicious actors and especially against DDoS attacks
- Scalability of the system, so further horizontal scalability can be employed to support the growth of the number of users.

Architecture Diagram

The following architecture diagram describes the way the Discord system may be structured. Some segments have been extracted from the official developer blog, such as the Guild and Voice System, the data processing system and the Database writing system, however, the are no details on how those segments interact with each other, so the connection between them is based on speculation.



User Clients are perpetually connected to the discord gateway via a WebSocket in order to be able to receive notifications from the Notification Service and check their connection status.

If a user were to want to join a voice server, they would send a request through the gateway and then either towards the Discord Guilds Service or directly towards a specific Discord Voice server, if there is already an existing voice channel with active participants. If no active voice channel has been set yet, the Discord Guilds service will employ a Service Discovery System in order to find the closest and emptiest Discord Voice service instance. After a Discord voice service has been allocated and accessed, a direct WebRTC connection is created between the client and the server, through which media information will be shared with minimal overhead.

Besides the Guild and Voice services which have been written in elixir, there exists a separate Python server for processing HTTP requests. It is not known whether all the user requests are routed towards it directly, or whether a series of requests related to the guild activity may be passed through the Discord Guilds server, so both cases have been considered within the diagram.

After a request is received, it will be routed towards one of the multiple Rust written Message Services, before requests to one of the databases can be made. Initially Discord used Cassandra databases, but due to the lag that they created when multiple writes were scheduled, a switch to ScyllaDB has been made, fairly recently at that.

Regarding the extraction of data, two types of processing services have been employed, a streaming service, which it is speculated to be directly connected to the messaging services via a PUB/Sub employing connection, and a batching service, which queries the databases once a set amount of time. The information from both types of services is then written within a database that contains Raw data, which is denoted as RawOLAP, after which the data from that OLAP is queried by several derivative services, which performs transformations and aggregates data within a Derived OLAP, the data within which may then be used by the employed Data Scientists to figure out ways of improving the services offered by Discord.

Technologies

The discord team mentions having employed the following technologies in their stack:

- Elixir – for signalling Servers
- WebRTC – for voice/video communication
- ScyllaDB (previously Cassandra)
- Rust – for Message Service
- Python – Load Balancer API for Message Service Clusters
- C++ - for Selection Forwarding Unit within Discord Voice Server
- WebSockets – for notification and signalling purposes
- Google Cloud – hosting the Discord Gateway and Discord Guilds (while the 850 voice servers in 13 regions are hosted physically in more than 30 data centres across the world)

Bibliography

https://discord.com/blog/how-discord-stores-trillions-of-messages

https://discord.com/blog/using-rust-to-scale-elixir-for-11-million-concurrent-users

https://discord.com/blog/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc

https://www.reddit.com/r/discordapp/comments/7osqy9/comment/dsc46ka/