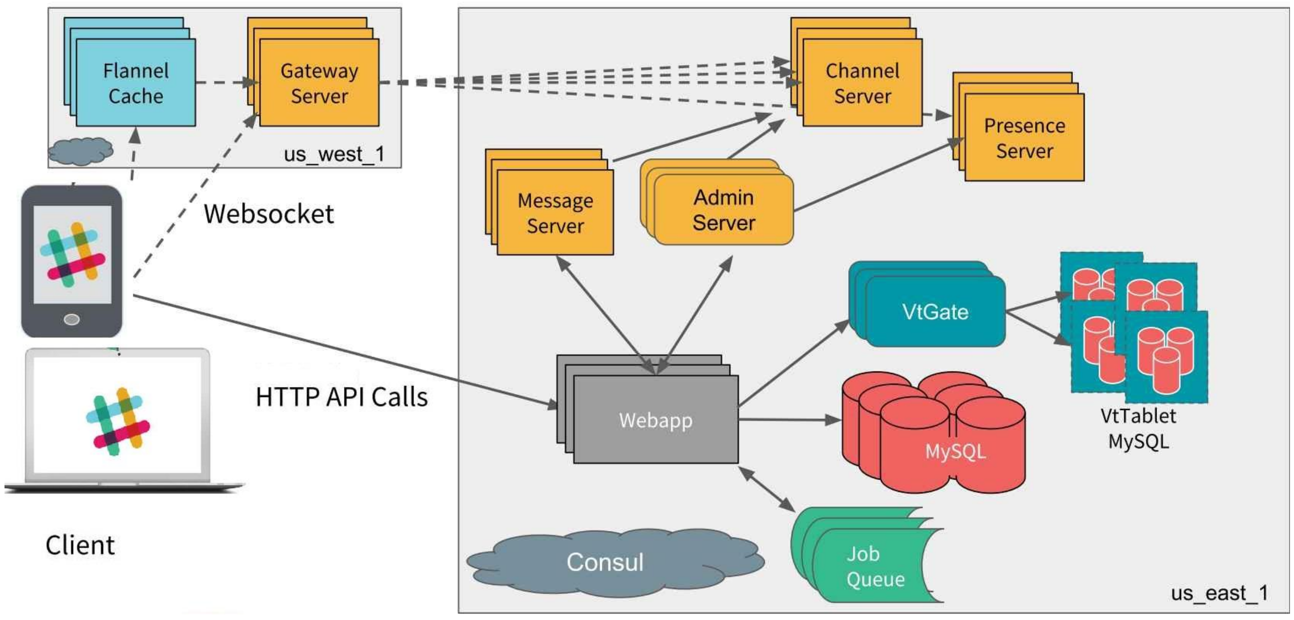
Slack implements a client-server architecture:



Clients - mobile, desktop, web and apps. Clients talk backend systems webapp servers, which handle HTTP request/response cycles and communicate with other systems like the main databases and job queues. The client sends a request and the server returns a response. The language and rules of communication are defined in the communication protocol. All protocols of the client-server model work at the application layer. Server implements API.

Flatten cache is used for lazy loading of the client. This service answers client queries on demand. It is backed by cache and on the agile network, so it provides fast data access. Users establish a web socket connection to Flannel first. If the user is the first user on his or her team, then Flannel does not have the team data in cache. It loads the data from DB, and then it establishes the web socket connection on behalf of the user. If the user is not the first user on the team, then Flannel has a cache, it escapes step two, and it goes to an established web socket connection directly for the user. Flannel caches relevant data of users, channels, bots, and more. It then provides query APIs for clients to fetch upon demand. It foresees data that will be requested next and pushes data proactively to clients.

Gateway Server is listening for an HTTP request. Upon receipt of a message, the gateway creates the given Slack channel if it doesn't exist.

Web application. It implements all the business logic of Slack. It is backed by MySQL, and a job queue system that exports asynchronous jobs. Webapp servers handle HTTP request/response cycles and communicate with other systems like the main databases and job queues It sends real time events to users. It uses web socket protocol, which is a photo plex communication protocol over HTTP, it is bi-directional, and both the server and the client can push data to the other end. For example, a new message posted in a channel is sent via an API call to the webapp servers, where it is sent to the message servers and is persisted in the database. The real-time message servers receive these new messages and send them to connected clients over web sockets, based on who is in the channel.

Messaging Server is a big router. It routes events to users. When the event happens on the team, a message is created, a reaction is made, a file is uploaded, and messaging server found this event to the list of users who are supposed to receive them.

Channel Server is responsible what channels are on the team and who is connected to which channels.

Admin Server is responsible for administering the workspace. For example, assigning system roles to members, changing a member's role.

Presence Server is basically a stand alone Presence Agent (PA) which accepts and stores information about users. It then sends notifications to subscribers when the user’s presence state changes. A user can have multiple devices, such as a PDA, laptop or cell phone, contributing his or her presence information. For each device, a PUA pushes presence information to the Server. When an end user wants to determine another user's availability (his presence information), he or she requests that information.

MySQL was used as the storage engine for all our data. Slack operated MySQL servers in an active-active configuration. Every message sent in Slack is persisted before it’s sent across the real-time websocket stack and shown to other members of the channel.

Vitess — a horizontal scaling system for MySQL. Migration to Vitess began in 2017 and Vitess now serves 99% of our overall query load. Vitess provides a database clustering system for horizontal scaling of MySQL. Vitess automatically handles functions like primary failovers and backups. It uses a lock server to track and administer servers, letting your application be blissfully ignorant of database topology. Vitess keeps track of all of the metadata about your cluster configuration so that the cluster view is always up-to-date and consistent for different clients.

Slack uses a **job queue system** for business logic that is too time-consuming to run in the context of a web request. Job queue is used for every slack message post, push notification, URL unfurl, calendar reminder, and billing calculation. The web app can sustain its high enqueue rate as the jobs are written to durable storage (Kafka).