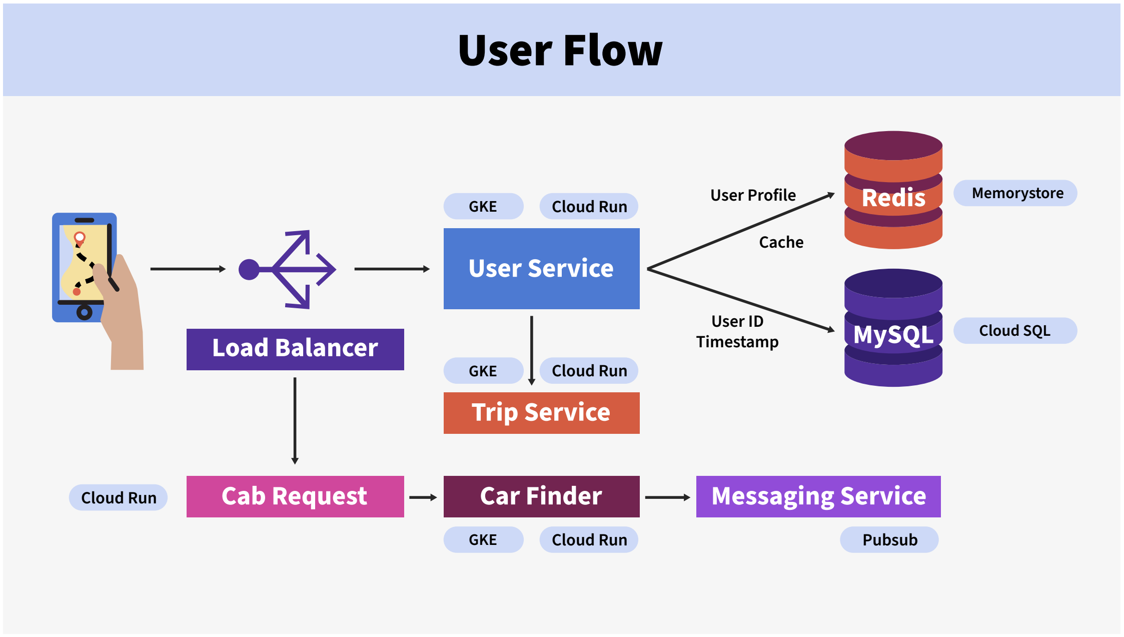
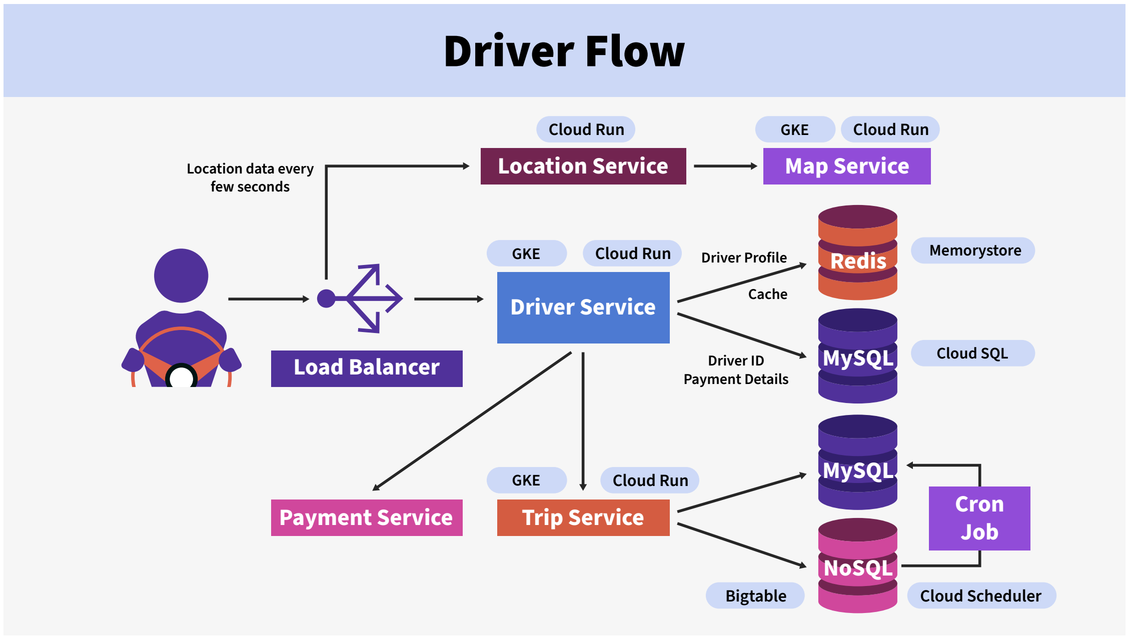
Ride Service Architecture

User Flow



- All users connect through the user app, which talks to a load balancer, which talks to the user service. The user service is where all user information is stored.
- Let's say a user wants to see their profile or update it; that request would be handled by the user service.
- If a user wants to see their trips, then the user service will talk to the trip service, fetch all the trips for that user, and send them back to the user.
- The user service uses MySQL and Redis to store and cache user information. It queries Redis first and then MySQL, if necessary.
- Now, when the user tries to book a cab, that experience is powered by the cab request service. It makes a WebSocket connection with the user's app, which displays a few cabs that are around them in the UI.
- It also places a request with the cab finder service to locate a cab for the trip. It sends a response through this WebSocket connection, saying that the cab is booked and these are the details.
- Cab finder notifies the messaging service if it finds a driver. If not, the information is sent to the messaging service for analytics (for example, directing drivers to areas with more customers).

Driver Flow



- Drivers use the driver app to communicate with the system. There's a driver service that's like the user service, but it's for drivers. Drivers use the driver service to get and update their information.
- For example, let's say that a driver wants to see their payment information or update it. The driver service will then call the payment service to get the information and send it back to the driver app.
- The driver service can also call the trip service to get the trips of all the drivers.
- All the UI data is powered by the driver service, which sits on top of another MySQL database that has all the driver information. It uses Redis for caching, just like the user service.
- The driver app also talks to location service via a WebSocket connection. As the driver moves through the city, their location is sent to the location service every 5 or 10 seconds.
- Now, you might be wondering why we don't just have the drivers create a new connection each time they want to send a location ping. Well, that would be a pretty heavy operation, so it's better to keep the connection open. Instead of WebSockets, we can also choose to use Google Remote Procedure Call, or gRPC, which is another bidirectional connection.
- Location service stores the driver's location information in a NoSQL key value data store that scales. We'll receive thousands, or even millions, of drivers all over the world who are sending their location updates every 5 seconds, so there's a lot of data being generated.

- There are two types of information that are stored in this data store: the driver’s live location, which is the last known location, and the route that the driver took during a trip. This information is needed for auditing and billing purposes. For example, if a customer wants to know how far they were driven, we can trace the driver’s route and calculate the distance.
- The location service then queries the map service to find out which area/segment the driver belongs to. It also gives us the estimated time of arrival (ETA), as well as the distance and route from point A to point B.
- The trip service is the source of truth for all trip information. It sits on top of two databases, MySQL and NoSQL. MySQL is used for all live information, like trips that are about to happen or are in progress. Once a trip is completed, a scheduled job moves to NoSQL for further storage and analysis.
- Why don’t we store all the information in MySQL? Because over time, this will become a massive volume of data. And the NoSQL database is good for low latency read queries, and we need that for scale.