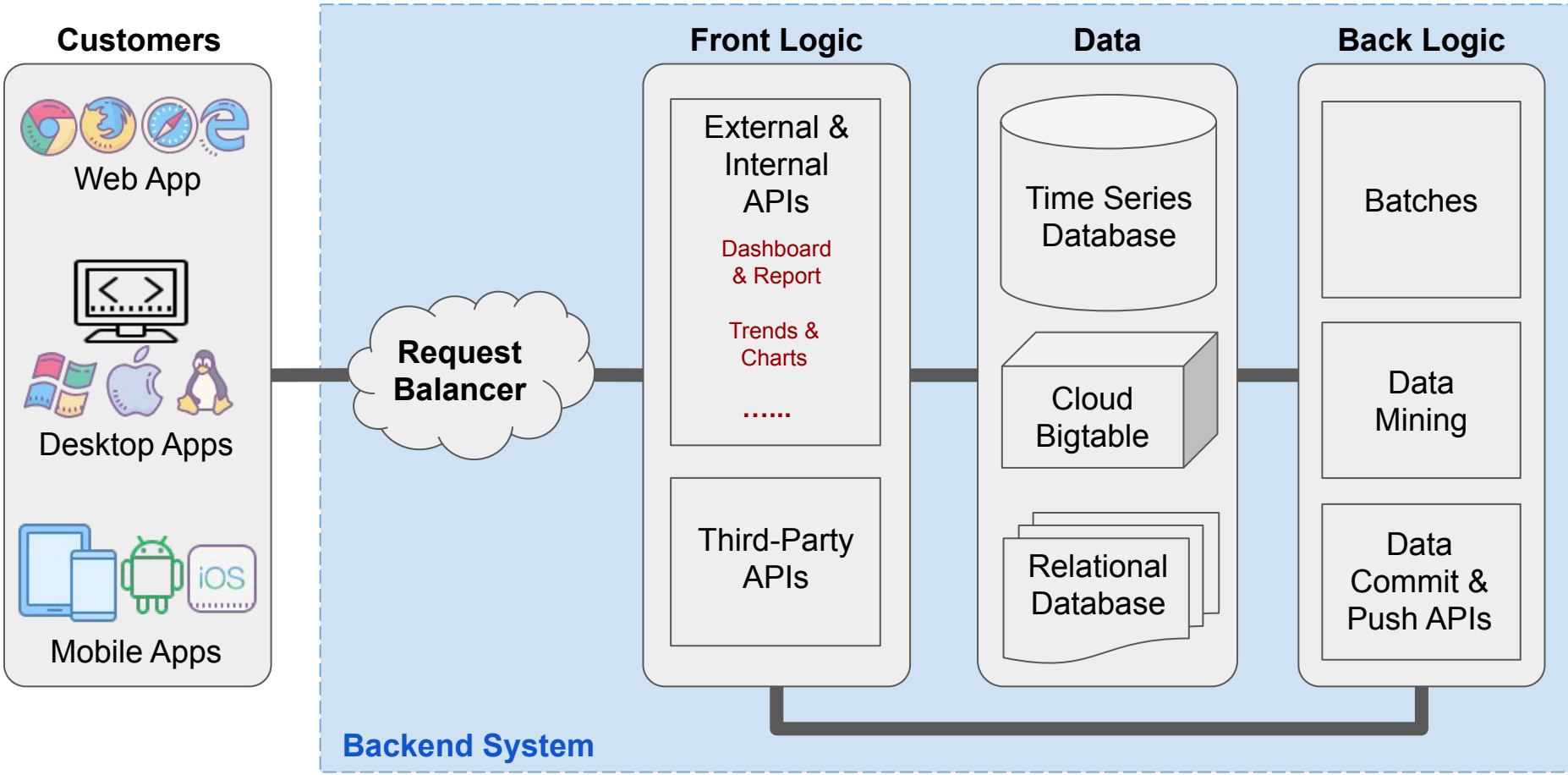
Software Engineer Challenge

Design a Google Analytic like Backend System

Author: Tran Xuan Hoang
Emails: hoang.tran@rakuten.com
hoangtx.social@gmail.com
Version: 1.0

Table of Contents

System Architecture Overview

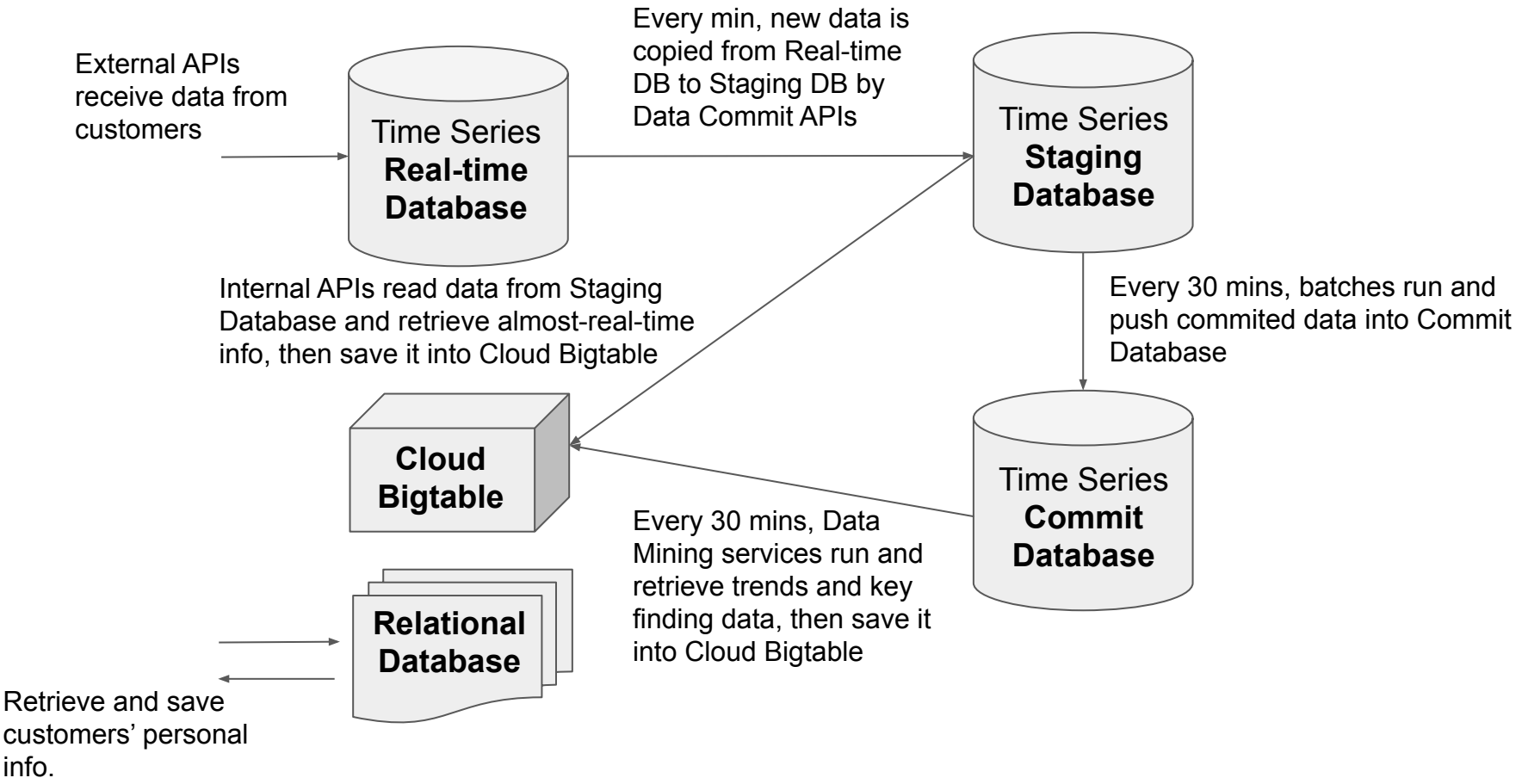


System Architecture Overview

Explanation

- The **backend system** consists of 4 main components (also called layers).
- The **request balancer** coordinates the incoming requests and outgoing responses so that network traffic is stable and the logic and data servers get a balanced amount of requests and/or queries.
- The **front logic** provides an interface between outside clients and the underlying databases. It is the first layer filtering incoming bad data and passing usable data to the data layer. It also provide APIs providing processed data that can be used to show on dashboard and reports, to infer trends, generate charts, etc.
- The **data** layer is where all data is saved. This design uses time series databases to save incoming data, and cloud bigtable to hold data that is ready to present to outside users.
- The **back logic** provides batches, data mining services, data related operations that all are used to process and maintain data internally.

Databases System

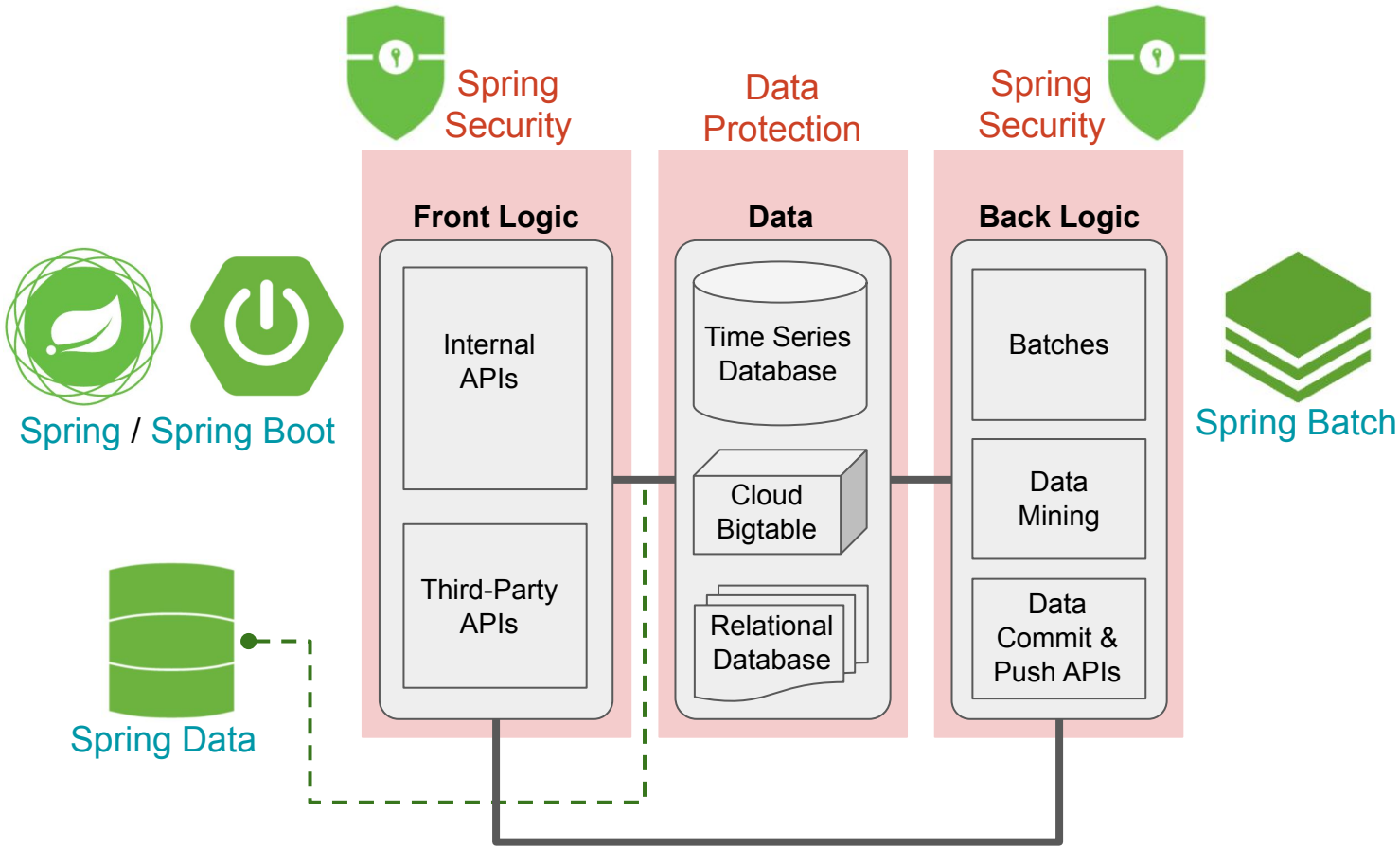


Databases System

Explanation

- The core of the backend system is how data is processed and saved into the database system, and how data is retrieved from that. This design uses three time series databases **real-time database** saving incoming data in real-time, **staging database** containing filtered healthy data with 1 minute of latency, and **commit database** holding committed data in a 30+1 minutes latency compared with the real-time data but is updated in batch.
- By creating 2 levels of latency, we let APIs and batches to do their job in analyzing data and extracting useful information that will be delivered to customers after it is saved in the **cloud bigtable**.
- The time series database and cloud bigtable provide low latency and massively scalable data storage and management systems that will be the key point to deal with the requirements of billions write requests per day and read/query patterns are time-series related metrics.
- The traditional relational database is used to store customers' basic info.

Logic Implementation Framework



Logic Implementation Framework

Explanation

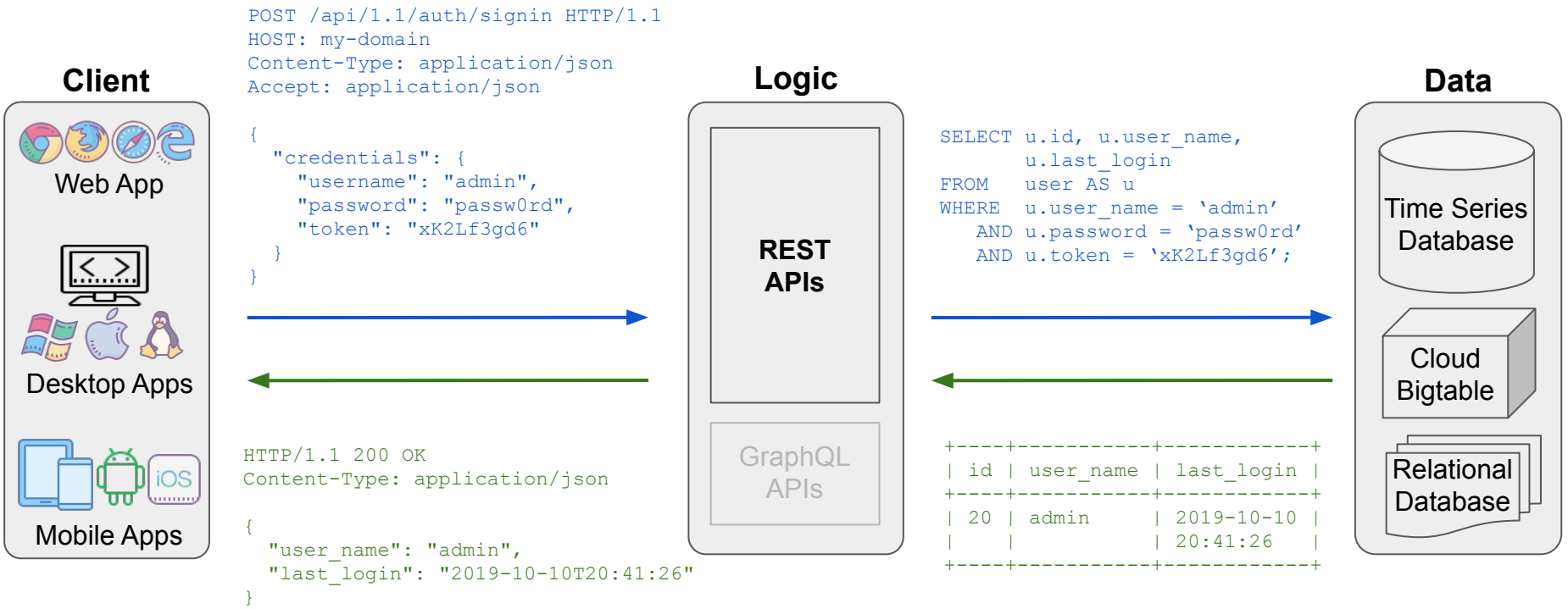
- This design proposes Spring/Spring Boot as the framework for the implementation of APIs and batches. Spring provides a quite diversified ecosystem of libraries and currently being developed to work with time-series data and high-frequent and big queries.
- Spring/Spring Boot make it easy to create robust and flexible microservice apps that later on will be the key for us to host and scale our backend system.
- Spring Security framework introduces libraries to employ security solutions. This is an advantage that helps us to quickly make our backend system safer.

Use REST and GraphQL Architectures for APIs

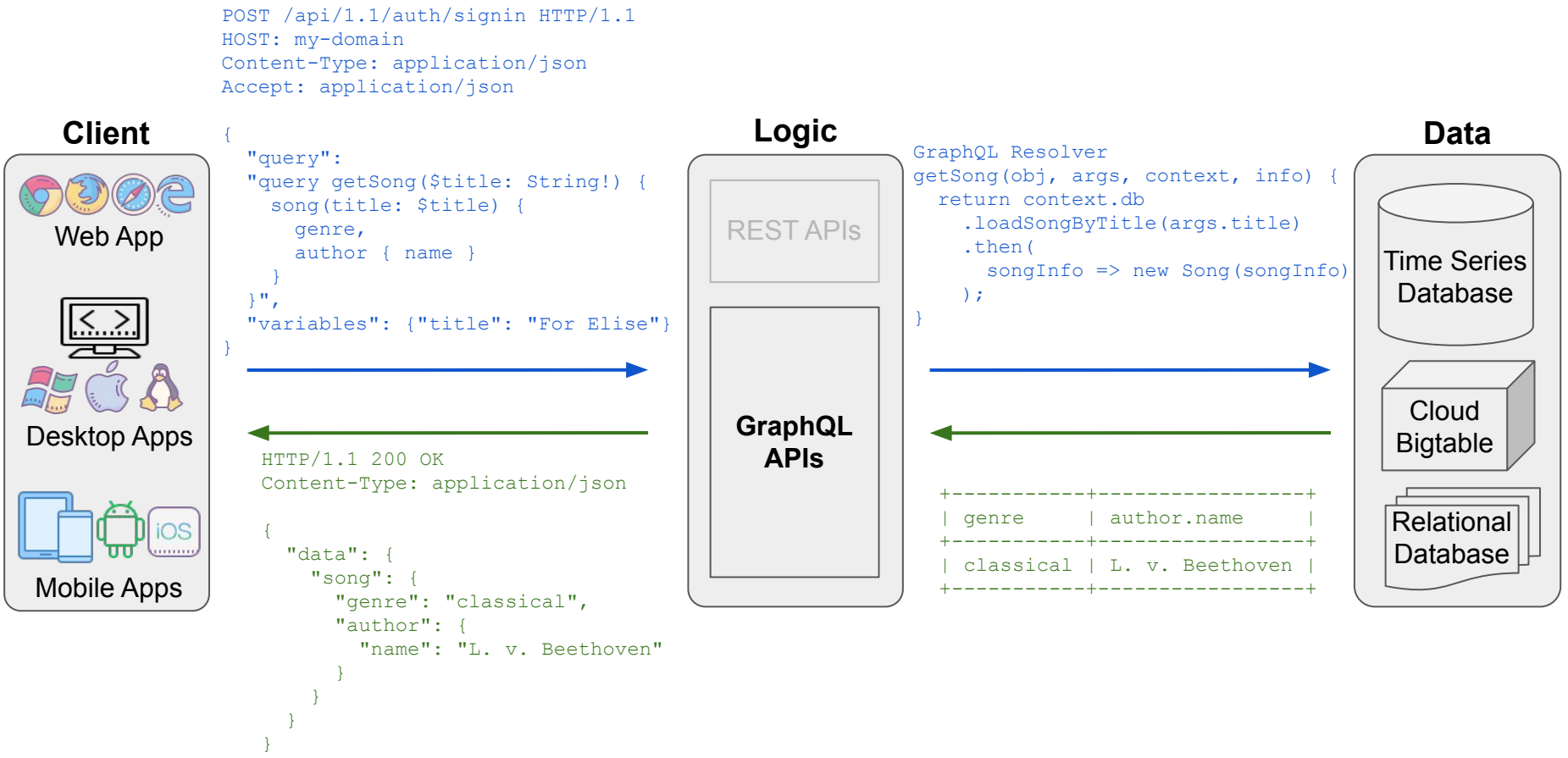
Explanation

- We design the longest latency to be around 30 minutes. As the backend system needs to serve a very large number of requests and responses, retrieving the right data and responding the really necessary data will optimize the traffic and reduces the risk of long delay. GraphQL is the right fit API architecture to make such robust requests and responses.
- In addition, REST is still a good choice for APIs implementing the logic of normal web based applications. Therefore, this design keeps the use of mixing both the REST and the GraphQL as the flexibility in architecting our API requests and responses communication.

Use REST API Architecture

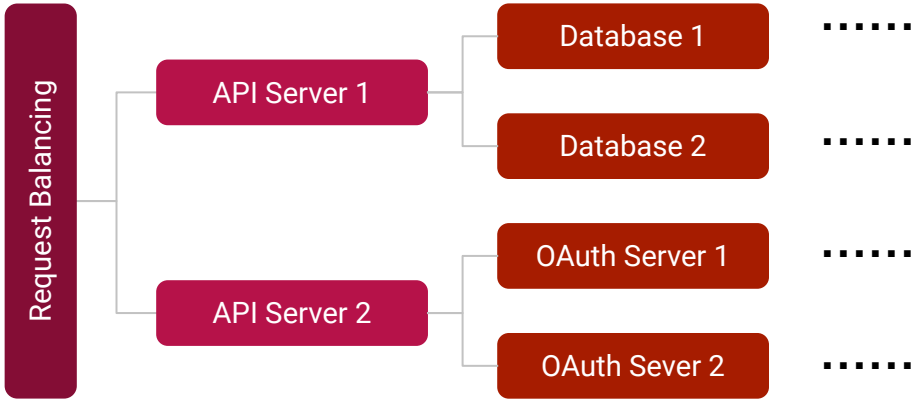


But Also Use GraphQL API Architecture



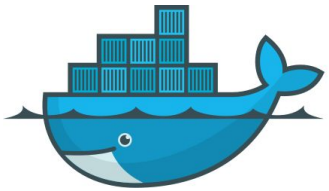
Hosting Strategy

Hosting the entire system as a set of microservice instances



Migrating, Scaling with Containerization

Kubernetes



Docker

Hosting Strategy

Explanation

- In order to have our backend system run with minimum downtime, it is critical to be able to scale up our app whenever we need. In addition, dividing our big backend system into smaller and lint microservices and host them under appropriate server resources will make our system much more stable and efficient.
- We also use containerization to wrap our microservices into containers. This help us to have simple and fast deployment, enhanced productivity, improved scalability, improved security, etc.

The End

Go to the top slide

Table of contents