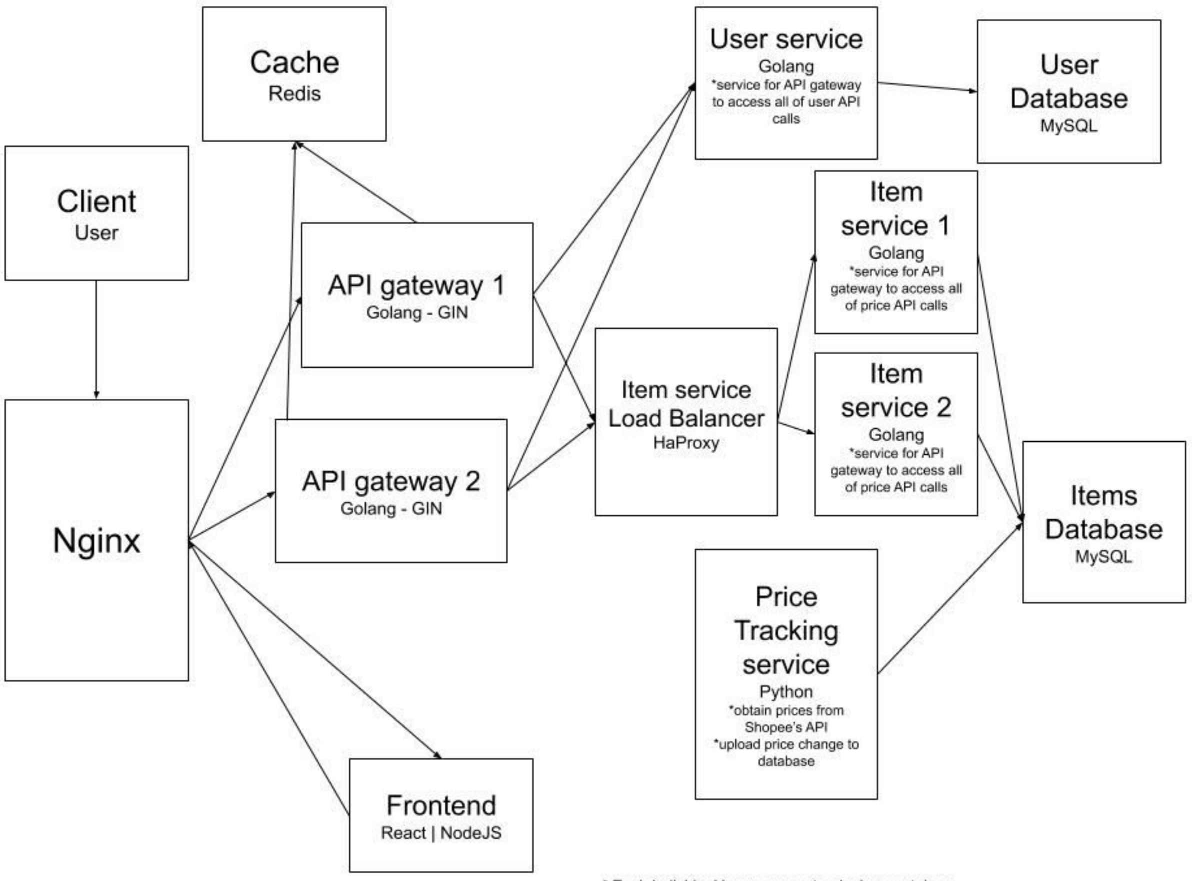
Shopee WSA Entry Task Concluding Report

Sim Zhi Qi

Software Architecture



Cache
Redis
User service
Golang
*service for API gateway to access all of user API calls
User Database
MySQL
Client
User
API gateway 1
Golang - GIN
Item service 1
Golang
*service for API gateway to access all of price API calls
Item service Load Balancer
HaProxy
Item service 2
Golang
*service for API gateway to access all of price API calls
Nginx
API gateway 2
Golang - GIN
Items Database
MySQL
Price Tracking service
Python
*obtain prices from Shopee's API
*upload price change to database
Frontend
React | NodeJS

* Each individual box represent a docker container
* Use of docker-compose to run all containers in the same network

Services

1. app

- Use as a router to route request to frontend and api-gateway
- Built with Nginx
- Frontend built with React
- Frontend accessed with http://localhost
- Api-gateway accessed with http://localhost/api
- Use as a load balancer for api-gateway to balance request between 2 api-gateway servers

2. Api-gateway

- Use as a gateway for all API endpoints
- Built in Golang & Gin
- 2 instance of api-gateway running for load to be balanced between
- Passes each request to their respective services
- Connected to respective services via gRPC acting as a gRPC client

3. Redis-user-sessions

- Use as a cache for api-gateway
- Store user sessions

4. User-service

- Service for user functionalities
- Built with Golang
- Connected to user-db via TCP
- Connected to api-gateway via gRPC acting as a gRPC server

5. Item-service

- Service for item functionalities
- Built with Python
- Connected to items-db via TCP
- There is a load balancer using HaProxy to balance load between 2 instances of item-service
- Connected to api-gateway via gRPC acting as a gRPC server

6. Price-service

- Service to retrieve data from Shopee API
- Built with Python
- Connected to items-db via TCP

7. User-db

- Database for user data
- Using MySQL as DBMS
- User table to store information of all user login credentials
- UserItems table to store information of all user’s watchlist items

8. Items-db

- Database for item data
- Using MySQL as DBMS
- Item table to store information of item details
- ItemPrice table to store price changelog of items

9. Prom

- Prometheus gateway for metrics tracking

10. Grafana

- Grafana dashboard for metrics tracking

11. Cadvisor

- Monitoring for docker containers

Timeline of Implementation

1. Design Doc + API Doc
2. Implement Nginx configuration with React
    a. Nginx will act as a reverse proxy which routes requests based on the path passed, with /api path being for the api endpoints and the rest being for the frontend.
    b. Docker double build where I built a image with nodejs to build the React app first, and then I re-build an Nginx image and copy the static build files from the react app to the nginx container and serve the static files from there
    c. Nginx proxy_pass to api-gateway service
3. Implement Go-Gin API interface for API gateway
    a. Use CORs middleware to set CORs policy for debugging localhost:3000
    b. Use groups to group different apis based on which service they call
4. Implement MySQL database with Docker (2 instances for 2 database)
5. Implement User service
    a. Link user service with api-gateway using gRPC
    b. Built proto file for both api-gateway and user service
    c. Link with user-db using Go mysql connector
    d. Use connection pool to maintain connection to database instance
6. Implement Item service
    a. Link item service with api-gateway using gRPC
    b. Built proto file for both api-gateway and item service and generate files for both python and Go
    c. Link with items-db using python mysql connector
    d. Use connection pool to maintain connection to database instance

7. Implement price service
    a. Used apscheduler library for scheduling jobs to retrieve data from Shopee API
    b. Run calls to Shopee's API every hour to update flash deals as well as track price of every item in the db
    c. Link with items-db using python mysql connector
    d. Use connection pool to maintain connection to database instance
8. Implement user session auth in API gateway
    a. Implemented another container REDIS to cache user session details
    b. Store user session key in cookie and use cookie to validate with redis cache to validate and authenticate user
    c. Add middleware to certain endpoints to restrict usage to only authenticated users
    d. Cookie use httpOnly flag to be more secure
9. Implement front-end
10. Implement Pagination
    a. Used infinite scrolling on front end
    b. Added params of offset and limit for each endpoints that needed pagination
    c. Used LIMIT & OFFSET in query for watchlist
    d. Used id > OFFSET & LIMIT in query for items
11. Setup Grafana & prometheus
    a. Track queries per second of each endpoint
    b. Track CPU usage of each container
    c. Track response latency of each endpoint
    d. Track response size of each endpoint
12. Setup Benchmark testing
    a. Used wrk library to generate multiple simultaneous connections and rapid calls to the api
    b. Track queries per second on different endpoints
13. Logging
    a. Log all logs into logs/app.log file with volumes attached so that it can be accessed
14. Load Balancing
    a. Load balance api-gateway from Nginx
    b. Load balance item-service with HaProxy as a new container instance

Optimisation

- Setup connection pool for all SQL connections so that time to connect to SQL database via TCP is saved
- Increase the number of workers for gRPC servers
- Tried to setup client side connection pool for gRPC but it did not improve benchmarks
- Setup load balancing for item-service so that there is a wider pool of SQL connections and load for querying is splitted
    - Item service is the most heavily used server based on queries due to the nature of the endpoints (get-item, price)
- Setup load balancing for api-gateway so that there is more gRPC connections to services and load for requests is splitted
- Bottleneck mainly in the services connecting to SQL and the api-gateway connecting to services via gRPC

Learnings

- Learnt to use gRPC as a way to connect between services as it is lighter weight and faster generally due to the nature of the data transfer
- Learnt to use Golang/Gin as API endpoint
- Learnt session management and comparison between different kinds of session management like JWT vs Sessions and usage of cookies
- Learnt pagination and the usefulness of it
- Learnt to use Prometheus to track various metrics
- Learnt to use Grafana to have a dashboard of all metrics
- Learnt about the usefulness of benchmarking in various ways to stress test system
- Learnt the benefits and cons of various optimisation techniques in microservice such as connection pooling and load balancing