Super Doge

Yunqi, Li	Yu, Cao	Yinjie, Liu	Paolo, Rosettani
21201601	21205065	20211091	21204524

Synopsis:

Description of system:

Our project is Super Doge Benefit System, which is a points reward system. These reward points can be used to exchange for special products. In the design of our system, we assume Super Doge is a big enterprise. After becoming member of Super Doge, customers can get points in their accounts, and they can use these points to buy Nike products. There are two delivery companies as the partners of Super Doge. They will provide the delivery plans, and the system will choose the better one for customers.

Application domain:

Points reward system can be widely used to increase customer engagement. Our project implemented a points redeem system to provide benefit to customers. We provide a web page for customers to redeem points for products and review their orders.

Technology Stack

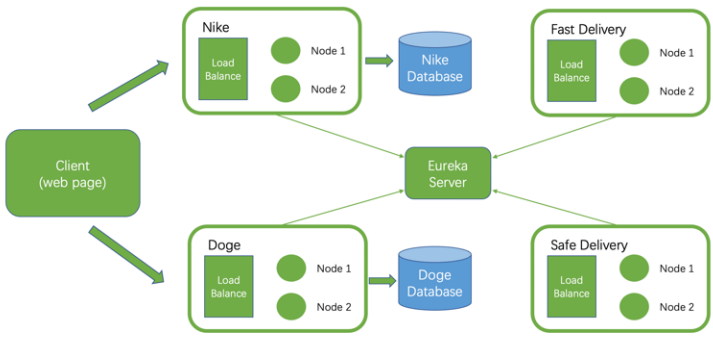
Main technologies:

- Eureka: providing service discovery for all the modules.
- Docker: deploying system, creating multi-instances for each module.
- Spring Load Balancer: providing load balance between multi-instances by adding spring annotation for rest template bean.
- Restful service: providing interactions between modules.
- MyBatis: a framework can simplify the implementation of database access in Java applications.
- MySQL: a relational database.
- Vue.js: a lightweight framework to implement front-end project.

System Overview

Project structure:

There are three layers including client, services and database. Services layer includes five modules: Eureka server, Doge, Nike, Fast Delivery and Safe Delivery. Below is the architecture diagram:



Client:

Client layer includes a Vue.js project, which can display the system. Users can login in the system, use points to buy Nike products and review their orders. It can invoke Nike module and Doge module for user operation.

Eureka Server:

Eureka Server implemented service discovery. It holds the information about all the module services. Services will register into the Eureka server, and Eureka server knows ports and IP addresses of each module service. So all the requests will be based on service name, without knowing hostname or port.

Doge:

Doge module stands for Super Doge, it maintains all the member information. So Doge module has its own database. There are two restful apis, one is for getting all the member information, the other is for updating left points of member.

Nike:

Nike module stands for a partner to provide products for points exchange. Nike module maintains products and orders with its database. It provides restful apis to query all the products and orders, as well as posting new order. After receiving new order, Nike module will invoke Doge module to update left points, and get delivery plan from Fast Delivery and Safe Delivery.

Fast Delivery:

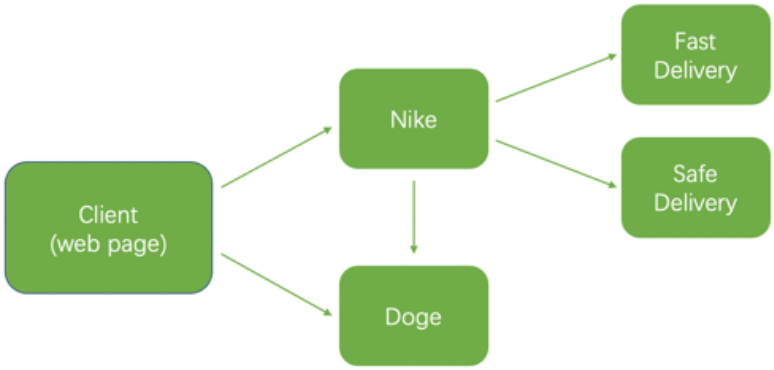
Fast Delivery can provide delivery plan for orders. As default, the delivery time of order will be three days later.

Safe Delivery:

Safe delivery can also provide delivery plan, but it has different way to set the time. It will call a public weather api to get current weather condition. It will set delivery time based on weather condition.

Business process of system:

Below is the process diagram:



Firstly, the client side will query all the products from Nike module and member information from Doge module. Customers can login by selecting a member. So the client side will query relevant information such like member profile and previous orders.

When customers want to redeem points for products, they can click the button to post a new order. The order request will be sent to Nike module, then Nike module will send a request to Doge module to update the left points of customer. Besides, Nike module will call Fast Delivery and Safe Delivery to get delivery time. Nike module will compare the time and choose the better one. At last, Nike module will add a new record in order database.

Scalability and fault tolerance:

In our system, we leveraged eureka, spring load balancer and docker to implement scalability and fault tolerance. We created multi-containers for each module, and all the containers registered in eureka server. In system design, Nike module works like the entry point, it has interactions with other three modules. So we added spring load balance configuration in Nike module. As a result, There is a load balance before each interaction. By default, we set two containers for each module, these two servers can receive request by turns. If one server is down, the other one will take all the requests. The system can still work, which provides fault tolerance. Besides, when we deploy the system, we can add or remove containers in docker compose file, so this is a way to provide scalability.

Contributions

Yunqi, Li:

Completing Nike module, eureka server and client side. Deploying system on docker.

Yu, Cao:

Completing Doge module. Integrating MyBatis.

Yinjie, Liu:

Completing Fast Delivery module.

Paolo, Rosettani:

Completing Safe Delivery module.

Reflections

Key challenges:

The most challenging part should be the solution of fault tolerance.

We started with considering multi-instances. So we created multi-containers based on each module image. But we were stuck at the configuration of these containers. At last, we added environment configuration in docker compose file, defining multi-containers port mapping and updating endpoints of eureka registering.

After deploying multi-containers, problem occurred in the restful calling between modules. Since for each module service, there are multi-instances registered in eureka server. Eureka server can't determine which instance to call. Then we added load balance to fix this problem. We added load balance annotation for rest template bean in Nike module. Since all other modules are only consumed by Nike module. There will be load balance before all the restful requests, which solved the problem.

What would you have done differently if you could start again?

We provided fault tolerance and scalability in project, but we know it may not be a perfect design.

Actually, our project is not auto-scaling, it means we need to add or remove instances manually for each module in docker compose file. If we can do it again, we will consider Kubernetes for container orchestration.

Besides, we implemented fault tolerance between all the module services, but not between client side and module services. Client side is pointing to fixed Nike and doge module container. In the future, we may add a web server before service layer, such as Nginx. So it can provide load balance for service layer.

Besides, there are more distributed technologies we can explore, for example, other components of spring cloud, such as ribbon and hystrix. Hopefully we will explore them in the future.

What have you learnt about the technologies you have used? Limitations? Benefits?

After completing our project, we have better understanding of distributed system and architecture. Eureka really provides a lot benefit on service discovery, we don't need to care about the service hostname and port. But we also learned single distributed technology may not work well, it always brings more benefit when applying with other technologies. For example, service discovery and load balance may be a better combination.