Assignment 2: Micro service Architecture

Lu Wang-14302365, Tianshi Chen-14094606, Zheng Zhang-14270293
Assignment-groups 5

April 21, 2022

1 Design and Implementation

In the microservices architecture, each microservices is loosely coupled, allowing each microservice to be deployed independently. Each microservice only focuses on one task and completes the task well[1]. Based on this feature, we independently deployed a login authentication server and URL-shorter (based on Assignment 1).

1.1 Design

In this assignment, we develop a login-server to get access to the URL-shortrner server. We deployed login server and URL-shortener server independently on two different servers. Login-server can pass information of login status to URL-shortner server using Json Web Token[2]. Every time when a user logs in to the login-server, the server will return a Json Web Token to the URL-shortner server.

1.2 Implementation

Based on Assignment1's code, we created a new Flask program as the authorization server, running on a different port to the shortener server. The authorization server provides three services, create accounts, log in, and verify the login status. Besides, we updated the shortener services to adapt to these changes.

1.2.1 Login Server

For the authorization server, the creating account service receives a form-data of username and password from the user by post method and stores them. If the user attempts to create an existing username, it will return an error code and a message to inform that the username already exists. The login service receives the username and password from an HTTP form when the user logs in. After the service checks the correctness, it returns a JWT (JSON Web Token) for success or an error code for failure. For the verify service, it receives a JWT from the bearer token. The service will decode the token and verify if the token is correct. If the token is valid, it will send the username back. Otherwise, it will return an error code.

1.2.2 URL Shortener Server

In the shortener server, we updated the mapping between the short URL and the original URL. For now, it contains a new key as owner with a value of a list, which means we gave every original URL an attribute to indicate its owners. Also, if someone changes the long URL by the PUT method, the service will create a new mapping between the short URL and the one the user wants to modify. These could guarantee that one user's change to the longer URL will not affect the other users who own the same short URL. Besides, when users operate creating, changing, and deleting, the service will connect to the authorization server to verify the login status of the requester and then check their ownership. If everything is correct, the operation will finish successfully, and if error codes occur, the process will fail, and the service will return an error code. Before, when the user performed a get method without a short URL, the service would return all mappings, but now, it will check the login status and return all mappings owned by the user, which means the user could not see the others' owned URLs.

2 The Answer to question 3

In assignment2, we apply login-server and URL-shortner-server, so there are two different ports. If we want to extend the functions of assignment 2, there will be more and more ports. Therefore, it is very necessary to design a general port service. In our design, the general port service only open one port to receive all kinds of queries. In the port server, there is a mapping form which can classify the incoming parameters according to the type and correspond to the microservice of the specific request. In this case, the query will be related to specific microservice and the queries can be pointed to the correct microservice.

3 The Answer to question 4

In assignment 2, we only apply a simple design of the RESTful microservices[5] architecture, regardless of the problem of overload. However, in practical application, such a situation often occurs during peak traffic times[3]. During traffic congestion, the stagnation of a service can affect the users' sense of experience on client , so we need to scale up our services independently of each other[**villamizar2015evaluating**].

In our scenario, we suppose to establish a gate that can make intelligent judgments through AI technology. A client can send a request to any one of multiple microservices, and this intelligent gate is responsible for request forwarding, composition, and protocol conversion. All requests must first go through the intelligent gate, and then forward the request to the corresponding microservice. Through this method, our sercies can be more independently frome each other.

In addition to the previous method, we can also solve this problems through load balancing[4]. For the server-side, we can use a server list and maintain the list utilizing heartbeat detection to ensure that all service nodes in the list can be accessed normally. When the user sends a request, it will first reach the load balancer (also equivalent to a service). The load balancer takes out the address of a server from the list of available servers according to the load balancing algorithm (rotation training, random and weighted rotation training), and then forwards it to reduce the pressure on the system. For the client-side, the client can store a list of servers captured from the registry. Through academic search, we expect that there is a client load balancing tool called Spring Cloud Ribbon, which is a client-based load balancing tool. This tool can automatically convert service-oriented rest template requests into service calls for client load balancing. Ribbon maintains a list of services. If a service instance logs off or dies, the ribbon can eliminate it by itself. Ribbon uses the service information list read from the Eureka server (stored locally, i.e. in the client) to reasonably load and directly request specific microservices when calling service instances.

4 The Answer to question 5

If we want to track all the services like location and its health status, we can generate a unique ID at the beginning of each request and pass it to the whole call chain. We can use this ID to track the whole request and obtain the performance indicators of each call link. We found that there are many open source distributed tracking libraries to choose from, and the most popular libraries may be Zipkin and Jaeger.

In the tracking process, we use the HTTP header as the carrier to transmit the tracking information (traceid). For example, RESTful uses the HTTP protocol, so we put the tracking information into the HTTP message header for transmission, so that we can clearly know the location and health information.

References

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. "A systematic mapping study in microservice architecture". In: *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE. 2016, pp. 44–51.

[2] Michael Jones, Brain Campbell, and Chuck Mortimore. “Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants”. In: *May-2015.{Online}. Available: https://tools. ietf. org/html/rfc7523* (2015).

[3] Wubin Li et al. “Service mesh: Challenges, state of the art, and future research opportunities”. In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2019, pp. 122–1225.

[4] Sambit Kumar Mishra, Bibhudatta Sahoo, and Priti Paramita Parida. “Load balancing in cloud computing: a big picture”. In: *Journal of King Saud University-Computer and Information Sciences* 32.2 (2020), pp. 149–158.

[5] Cesare Pautasso. “RESTful web services: principles, patterns, emerging technologies”. In: *Web Services Foundations*. Springer, 2014, pp. 31–51.