Management of API Gateway Based on Micro-service Architecture

IOP Conf. Series: Journal of Physics: Conf. Series **1087** (2018) 032032 doi:10.1088/1742-6596/1087/3/032032

Management of API Gateway Based on Micro-service Architecture

J T ZHAO[1] S Y JING[1] and L Z Jiang[1]

[1] Institute of Network and Information Systems, School of Control and Computer Engineering, North China Electric Power University, Beijing, 102206, PR China

E-mail: Jingshuyang@eplian.com

Abstract. Micro-services are activities that run in your own programs and communicate with HTTP APIS through lightweight devices. Under the Micro-service architecture, the API gateway is an important component of the overall architecture. It abstracts the common functions that are needed in the Micro-services. As the only entry for a Micro-service, the API gateway encapsulates the specific internal implementation and interface of the system. Based on the analysis and comparison of the traditional framework and the Micro-service framework, this paper mainly analyzes the realization of the functions：load balancing, automatic service blowing, and Gray release, and gives the implement scheme of the key technology of the API gateway under the Micro-service architecture. The scheme discusses the case study of the technology selection and application architecture under micro service. And it also provides a new solution for the difficulties in manage API gateway under micro service by giving a detailed design for the authentication of the API gateway ,reverse proxy function and flow control function. By using API gateway, the problem of how a caller can call an independent service can be solved, thus the development efficiency can be greatly improved.

1. Introduction

With the rapid development of Internet technology, service-oriented architecture (SOA) has evolved in many ways. The popularity of the microservices architecture has replaced some of the SOA. One of the advantages of microservices over traditional monolithic architectures is that the separation of services brings isolation of updates, deployments, and management, allowing some individual services to innovate and experiment. This supports the continuous upgrading of the user experience and provides the support of the technical architecture for the process of digital transformation of the enterprise. The API gateway is a very common mode in the microservice architecture. Using the API Gateway can solve the problem of how callers call independent microservices. In order to manage the complex and numerous API well, we ought to use the API gateway to manage the service API in the construction of the microservice system. To put it simply, API Gateway is a special server, which is the only entrance entire micro-services. API gateway encapsulates the internal aspect of the system and the specific implementation of the interface, on the other hand, it has functions such as permission verification, load balancing, caching, and monitoring.

This article will analyze the API management from the following aspects : the background advantages of the API gateway under the microservice architecture, the analysis of its functional technical points, and the related technical solutions.

2. Microservices API Gateway Background and Benefits

The granularity of APIs provided by microservices is usually different from that of clients. Microservices generally provide fine-grained APIs, which means that clients need to interact with multiple services. Different clients require different data, and different types of clients have different network performance. The division of services may change over time, so it is necessary to hide details from clients.

API Gateway is an API-oriented, serially centralized strong management and control service that appears on the system boundary. Prior to the popularity of the microservices concept, the API gateway entity was born. The main application scenario at this time is OpenAPI, which is an open platform for external partners.

When the concept of microservices became popular, the API gateway seemed to be the standard component for integration at the upper application layer. Using the API Gateway service to micro service has many advantages. It can make the client not affected by the location of the service instance and undetectable how the application is split into multiple microservices. It provides the optimal API for each client to reduce the request. One of the benefits of API Gateway is to encapsulate the internal structure of the application. Compared to calling the specified service, the client interacts with the gateway more simply. API Gateway provides each client with a specific API, which reduces the number of client-server communications and simplifies client code.

3. Analysis of Functional Elements of Gateway Mode under Micro-services

In the scenario of using micro-service architecture, when the client calls the background micro-services, you need to perform login authentication, identity authentication authority, traffic control, load balancing, call log file, flow control and reverse proxy, health checks and other operations to call every microservice. For service managers, they should have functions such as service permissions, system monitoring, service flow control configuration, API URL routing rules configuration, and call setup. Therefore, the operation needs to be handed over to a high-performance intermediate layer for processing, so as to reduce the coupling between the systems and make the micro-service more focused on the business logic processing and reduce the overall system response time.

As shown in the figure1, requests from various terminals reach the gateway after one-layer load balancing. After the gateway performs unified login authentication, permission authentication, flow control, load balancing, and health check, the request is forwarded to the background microservices.

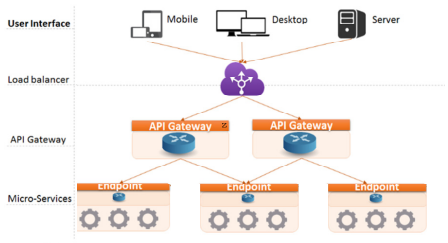

Figure 1. The client calls the micro-service process.

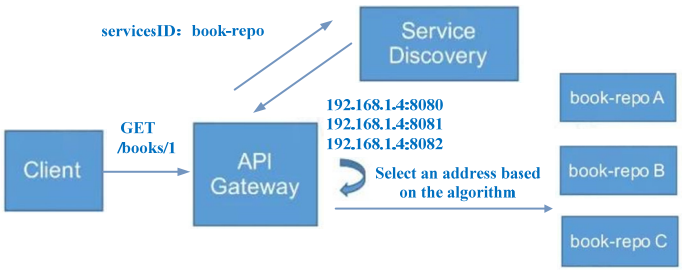

Figure 2. Load balancing process.

3.1. Implement load balancing

In actual deployment, when the application system is facing a large number of visits and the load is too high, the number of services is usually increased to scale out horizontally, and the cluster is used to improve the processing capability of the system. At this point, multiple services share the pressure of the system through a load algorithm, which is called load balancing. Using the API gateway, load balancing can be easily implemented. Service discovery is used to know the addresses and locations of all services. Load balancing algorithms are implemented in the API gateway to achieve load balancing.

3.2. Implement service blowing

In actual production, some services may fail for some reason. If you do not take some measures, it will cause the entire system to "avalanche." Or the number of service visits is will be limited due to the overall system load. Service blowing and service degradation are the main ways to solve the above problems. API Gateway can help to achieve these functions better. For the limitation of the number of calls for service ,when a service reach the limit, the API gateway will automatically stop the service from sending a request to the upstream, and perform service downgrade like the error page returned by the client or a unified response. For services that require a temporary failure, the API Gateway can automatically open the circuit breaker for the corresponding service and perform service blowing to prevent the entire system from "avalanche."

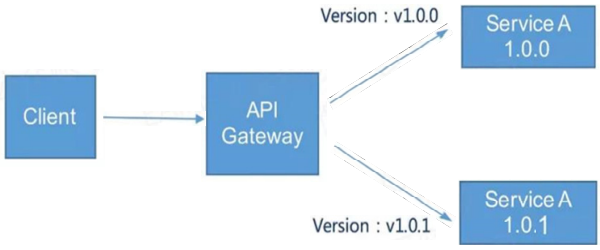

Figure 3. Gray release process diagram.

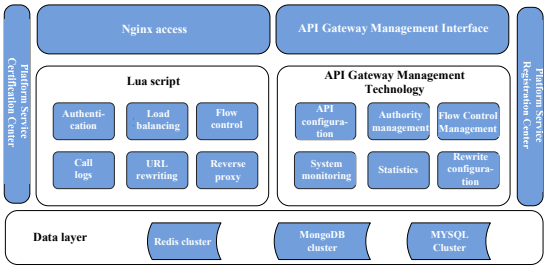

Figure 4. Architecture Diagram of Platformization Service API Gateway.

4. Micro-service gateway mode technology solution

4.1. Technical Selection

There are mainly five types of API Gateway WebApp, Mobile App, Partner Open API, Partner External API, IoT Smart Device. For the first three categories or the top four API gateways, except for guaranteeing the exchange of data, it is also necessary to implement identity authentication for access clients, anti-message playback and data tamper resistance, service authentication service calls, response data desensitization, traffic and concurrency control, and even API-based Called metering or billing.

The API gateway serves as the entrance of the background micro service request, and it must be required to have features such as high performance and expansibility. Therefore, it is preferred that the Ngnix-based lua language is used as an extended API gateway technology.

4.2. Application architecture

The design of the API gateway includes three elements: the API gateway itself, the API gateway client, and the supporting self-service platform. An important role played by the API gateway is that all clients and consumers access the microservices through a unified gateway and handle all non-business functions at the gateway layer. Typically, the gateway also provides REST/HTTP access APIs. The server registers and manages services through the API Gateway.

Taking the platform service API gateway as an example, it can be divided into two parts: the service request agent subsystem and the gateway management subsystem. At the same time, it interacts with the platform service authentication center to handle service request authentication. On the other hand, the service instance information and interface information are loaded from the platform service registration center. The architecture of the platform service API gateway is shown in the figure.

4.2.1. Platform Service API Gateway Request Broker Subsystem. The platform-based gateway management subsystem encapsulates the main business logic into microservices and deploys and maintenance independently. The management interface of the API gateway is integrated into the

integrated management platform for management, combined with Spring Web, to implement API management, rights management, flow control, system monitoring and other functions through a three-tier architecture of Service, Internal Service, and DAO. That is, the service registration and discovery center can synchronize the service information and interface information of the micro service in real time, and also support the management personnel to manually add the API interface. The architecture of platform service API gateway subsystem business logic and portal interface separation is also the architecture approach of microservices and services combined in a microservice scenario. A business portal freely combines multiple microservices to implement certain business functions, achieves the decoupling of interfaces and complex business logic, and ensures the reasonable size and scope of a single project. This is also a difference between the microservice architecture approach and the monolithic architecture.

4.3. Detailed design

For API gateway authentication, monitoring, load balancing, caching, request slicing and management, static response processing, etc. It is not difficult for API gateway technology to access. However, for the best practice, the design of a key function Implementation is important. For example, the gateway authentication mode selection, rights verification function, flow control function, URL rewriting function, service API gateway request proxy forwarding, API gateway background management system, configuration service interface user level flow control function specific implementation.

4.3.1 API gateway authentication. Currently in microservices, the protection API needs to be invoked only by customers who have agreed to authorize. At present, most of the methods used are of the three types: AppKeys, OAuth2, and OAuth2+JWT. These authentication methods have their own characteristics and advantages.

a. AppKeys. There are many public cloud API gateways and data open platforms currently that use AppKeys Auth authentication, such as Alibaba API gateways and aggregation data. This authentication mode is issued by the API gateway with a key, or appkey+appsecret+ some kind of complicated encryption algorithm to generate AppKey. The caller directly calls the API after getting the key. netWhen receiving the API request, the API gateway first checks the validity of the key, including whether the key is invalid, whether the current calling API is subscribed, and so on. If the verification is successful, the API gateway requests the upstream service and returns the result. Here, the upstream service no longer checks the request and returns the result directly. AppKeys authentication mode is more suitable for Open Service scenarios, which does not involve user information, rights information.

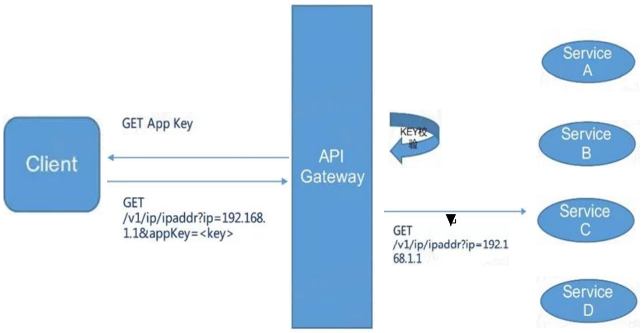

Figure 5. AppKeys authentication process.

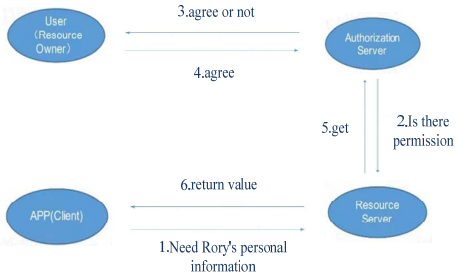

Figure 6. OAUTH / OAuth2 + JWT agreement certification .

b. OAuth2(Open Authorization). The OAUTH protocol provides a secure, open and simple standard for the authorization of user resources, but OAUTH's authorization will not allow third parties to touch

the user's account information (such as user name and password). In OAuth2, there are several roles: Resource Owner, Client, Authorization Server, and Resource Sever. The figure above is a simple OAUh2 process to illustrate the relationship between each role. Eventually, the app gets Rory's personal information.

c. OAuth2+JWT. The OAuth2 + JWT process is exactly the same as OAuth2. OAuth2 will eventually issue an Access Token to the caller. OAuth2+JWT actually replaces the Access Token with the JWT. The benefit of doing so is simply to reduce the number of queries to the DB at the time of the Token check.After the addition of the API Gateway, each of our services may require user information to determine if the current interface or feature is available to the current user. We can implement it by putting unified authentication on the API gateway. The API gateway performs unified interception and authentication. In combination with the authentication methods described above, the OAuth2 protocol can carry the characteristics of user information, so more OAuth2 authentication is used the way.

4.3.2. API Gateway Reverse Proxy Function. The API gateway processing request includes a series of main processes, such as receiving an API request, performing parameter verification, verifying the validity of the AccessToken, acquiring the APP information, and obtaining the API information according to the URL to perform flow control verification, request packet encapsulation, URL rewriting, and reverse proxy request, configuration request cache, storage call log, etc. The specific process flow is described as follows:

a. AccessToken validation. The generation of the AccessToken is based on the Oauth2.0 authorization protocol. First, it is verified whether the format of the AccessToken meets the requirements. An AccessToken conforming to the format requirement can be split into the application ID, the generation time, and the authentication signature. If the verification generation time is more than half an hour from the current time, the AccessToken expires. Then it obtains the application information from MongoDB through the application ID. And it compares the authentication signature by using the same encryption method to encrypt the AccessKey and the Security of the application information. If they are different, the AccessToken is illegal and cannot be called again.

b. Information loading. Through the verification of the AccessToken, the application information has been loaded, including the application-associated user information. According to the URL requested by the request, the API corresponding to the URL is loaded from the Nginx local cache. If there is no URL information in the local cache, the API information is queried from MongoDB. If it is still not, then the URL is an illegal request and is directly rejected. If the API information is queried, it includes the rights required for the URL, frequency control, IP limit, URL rewriting, and instance address.

This information is cached in the Nginx local cache. The Nginx cache uses server memory directly and reads very quickly, which can greatly reduce the time spent reading data remotely in high concurrency situations. However, the problem is that the local cache of each machine in the cluster cannot be cleared dynamically. Therefore, the API configuration needs to wait for a valid time of 20 minutes.

c. Flow control. After obtaining the information of the application information and the API, according to the requirements of the API, it is judged whether the application includes the right required for calling the API.If there is no right, the information of the insufficient right is returned. If the privilege verification passes, according to the configuration of the API, it is judged whether the call is restricted in the IP address.

If the IP restriction list is returned, an illegal IP address error message is returned. After the IP restriction is completed, it is determined that the API needs to perform application-level, user-level, and cluster-level traffic control. Flow control uses Redis's incr operation to determine if the number of

URLs in a unified second exceeds the frequency control threshold. If the threshold is exceeded, an error message that requests too frequently is returned.

Application-level traffic control uses URLs and application IDs as Redis stored keys, user-level controls use URLs and user IDs as Redis stored keys, and cluster-level flow holes use URLs and cluster instance names as Redis stored keys.

d. Request packaging. In the process of requesting reverse proxy, the background service may need to adjust the parameters. For example, you need to increase the default parameters, and add the authentication information, application information, and other data after requesting HTTP. In the process of requesting packaging, application information and verification information can be added to the HTTP Header according to configuration requirements.

e. URL rewriting. First, before the reverse proxy is performed, an instance is obtained by polling based on the instance information of the API. In addition, according to the rewriting rules of the URL, the original URL is converted into the URL of the actual internal service, and the call address of the reverse proxy is generated in conjunction with polling the selected instance address.

f. Reverse proxy. The reverse proxy uses Lua's Socket TCP library to establish a TCP connection with the remote service, encapsulate the HTTP packet according to the HTTP protocol, and send the HTTP packet over this TCP connection. In this process, the connection time and timeout time of the request can be controlled. In this TCP connection, the returned result is again encapsulated into JSON format and returned to the caller.

g. Result Cache. According to the API configuration information, you can configure whether the request result is cached and cached time . The request cache can avoid frequent establishment of connections with the same request, reduce latency, and improve service performance. The cache of API calls results in using the Redis cluster cache.

h. Store call log. In order to monitor the API gateway system, statistic API usage data, and analyze the performance issues of the API gateway, each application information, API information, time consumed by various processes, request parameters, and return parameters will be recorded through the log. Next, the big data platform is collected by the log collection system in real time, and logs are split and stored. You can then use Big Data Analytics Statistics to perform statistics and exception monitoring.

4.3.3. API gateway flow control configuration. This section describes the process of configuring the application of QPS (queries per second) traffic control services in a case. First, the service requester performs service processing, then encapsulates the request parameters, and uses the service client to invoke the service. After the microservice interface receives the call request, it first verifies the parameters according to the interface's business process, then stores the data in the database, and finally returns the processing result.

a. Service client initiates call. After data is saved at the service end, the user firstly processes the service according to the service, performing data access in its own service system, or checking whether the information entered by the user is legal. Then he encapsulates the request parameters and request management of the application flow control micro services through the platform service client.

b. Service processing on the server. After the flow control management micro service receives the request call, it first verifies whether the requested parameter is valid. If it is not legal, it returns an error message. Then the microservice performs business processing. For the operation of configuring the QPS value, first it checks whether the application exists. If the application does not exist, the

configuration cannot be performed. If the application exists, then it determines whether the QPS configuration exists. If the QPS configuration exists,it will perform data update. If it does not exist, the application QPS configuration is re-created.

c. Cross-cluster synchronization of data. For the application flow control configuration of the API gateway, the data is stored in the Mysql database cluster, but since the API gateway requests the proxy subsystem and reads the API configuration data directly from MongoDB, the value of the change in the Mysql database needs to be updated to the three MongoDB database clusters in Beijing,Nanjing and Shanghai. The main reason to use these three MongoDB clusters is that the API Gateway Request Broker subsystem is a cluster of three different regions. To reduce cross-region data access, three MongoDB clusters are configured. After updating the data of the APIs on the three clusters, clearly understand the QPS configuration of the API in Redis. If an update fails during this process, all operations on this database are rolled back.

5. conclusion

In the microservices architecture, each microservice exposes a set of fine-grained service providers. API gateways play an essential role in the microservice architecture. The API gateway ,serving as the gateway to each request initiated by the application, provides public functions such as load balancing, service blowing, and Gray release. It also integrates various micro-services and shields the complexity and diversity of the system, clearly simplifies the implementations of the communication between client and microservice applications.

Through the authentication method in the microservice architecture, the advantages and disadvantages of the microservice architecture and the challenges faced in implementing the microservice architecture are analyzed and summarized.

This paper analyzes the authentication scheme combined with the API gateway from the perspective of deployment, explores the key functional technology design of high-performance API gateways, uses API gateways, organizes and manages open microservice interfaces, and neither disrupts the micro-services architecture nor ensures the security of microservices. The main business difficulty of implementing an API gateway lies in its ability to handle high concurrent requests and performance requirements.

BY using the OpenResty platform, rights verification, flow control, URL rewriting, reverse proxy and other functions based on Ngnix and Lua languages, the high performance requirement of API gateway can be satisfied. However, there are still some issues that need to be continuously updated to meet the needs of continuous updating. Continuous optimization, better management of the API gateway can make the development more concise and efficient, and can achieve high-quality interaction with micro services.

References

[1] Tan Yiming. Design and Implementation of Platform Service Framework Based on Microservice Architecture [D]. *Beijing Jiaotong University* , 2017.

[2] Balalaie，Armin, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables DevOps: migration to a cloud-native architecture. *IEEE Software*, 2016. **33(3)** 42-52.

[3] Hane,Oskar. Build your own PaaS with Docker. *Packt Publishing Ltd* , 2015.

[4] Zuo W, BENHANKAT A, AMGHAR Y. Change-certric model for Web service evolution[C]. *Proceedings of International Conference on Web Services. Washington,D.C. ,USA: IEEE*, 2014:712-713.

[5] Newman S. Building Microservices[M]. *O'Reilly Media,Inc* , 2015.

[6] Gao Shihao. Correct posture for API management—API Gateway [DB/OL]. *https://mp.weixin.qq.com/s/Q9ZgUQIlGcBS5WPW6vwPhg*, 2018.

[7] Micro Service API Gateway [DB/OL]. *https://blog.csdn.net/zdp072/article/details/76473383*, 2017.

[8] He Zhuofan, Macro. Microservices and API Gateway (I): Why do I need an API gateway? [DB/OL]. *https://mp.weixin.qq.com/s/XTzRr0eR6ybpNFGJ57cVkA* , 2017.