Microservices

prerequisites

------------

The basic knowledge of computer programming and Service-Oriented Architecture

Microservices Martin Fowler and used at as Software architect.2011

Advantages

==========

It Increases modularity,easy to deployment, is easy to test , easy to maintain

Netflix,Paypal,Amazon,eBay and Twitter

Mono li thic(mänəˈliTHik)

What is Monolithic Architecture?

----------------------------------

Monolith, which normally means an object made from a single large piece of material. In Monolith Architecture an application is built as a single unit and has a single logical executable.

A standard enterprise application has a multi-layered architecture,generally presentation layer ,service layer and data layer sometimes , the integration layer to communicate with another application via messaging or REST API .

Q) Benefits of Monolithic Architecture?

1. Simple to develop (dividing it into multiple layers like Presentation layer,Service Layer, Data Access Layer)
2. Easy To test, Testing a single application is always easy
3. Easy Deploy we just need to copy the packaged application (jar, war etc) to the server
4. Simple to scale (we can perform horizontal scaling by running multiple instances behind a load balancer )

Q) Drawbacks of Monolithic Architecture ?

1. Even for small changes in the code,the entire application needs to be rebuilt and re-deployed.
2. This Type of application sometimes may affect complete service down or the application may not work as expected . Hence small problems may affect the entire application
3. Adding new concept/technology/new integration may become very complex
4. As the number of modules increases,the application size increases, downtime for re-deploy may also increase.
5. The code becomes complex which is hard to maintain

Q) What is Microservices Architecture?

Microservices is an application style that organizes an application as a group of smaller services instead of a single bulk service.

or

Microservice is an architecture style of development, Microservices is all about distributing or breaking a large application into small pieces (services) or a single individual application.

or

a particular way of designing software applications as suites of independently deployable service

Each small service have the following behavior

deployable independently as a single entity, Highly maintainable and testable.

Note :In Microservices Architecture each service has its own database schema. having a database schema per service is necessary if you can take really advantage of microservices

Q) How do Internal services communicate with each other in Microservices Architecture ?

Few Microservices expose a REST,RPC (Remote Procedure Call ) pr message-based API to be consumed by other microservices.

Some microservices may also implement UI to communicate with each other.

Note: The communication type depends on what you want to use. if you decide to use asynchronous request/response HTTP REST and for asynchronous communication message-based AMQP (Advanced Message Queuing Protocol) or Kafka

Synchronous communication: request expects response immediately

Asynchronous communication: request may not expect response immediately it has some time log

Q) Benefits of Microservices Architecture

1. Microservices offer high flexibility for changes.if we make any change in one service we do not need to stop other services
2. if one microservices is not working properly then it may not affect all other services.for example, let's assume Amazon shopping portal If payment service is not working then it may not stop other services such as Product search Add item to cart
3. Modifications, Upgrade, and Enhancements to new technology are not so complex. If the number of microservices increases the code will never become complex.It will not be mandatory to make other services down.
4. scaling service is easy and memory will be used in efficient(依飞选) way
5. It makes continuous deployment possible for complex applications as MS offers each service to be deployed independently
6. There is always a possibility to replace small parts of the system easily,in case if any part is unreliable or the client demands it to be replaced.
7. Easy to integrate with new systems and external sources via API
8. Simplicity in debugging & maintenance.
9. Better communication between developers and business users.
10. Development teams of a smaller size

Q) Drawbacks of Microservices Architecture?

1. Microservices Architecture includes complexity, when we need to implement inter-process communication mechanism including the additional implementation to handle it may partial failures
2. Testing Microservices Architecture also become complex as compared to Monolithic Architecture .for example a service you would need to test that depends on another service which may be down
3. Sometimes the result of one microservices may be used as the input of another microservice.
4. A minor change becomes very difficult to implement. suppose a change impacts multiple services, then you need to plan it carefully.
5. Deployment of MS applications is also much more complex.Each service may have multiple runtime instances.we need to configure, deploy, scale and monitor each instance separately
6. Even Runtime environment needs very high configuration and maintenance very complex and cost-effective
7. Expensive compared to monoliths.
8. Maintaining the network is more difficult.

Difference between monolithic architecture and Microservice Architecture?

Monolithic Architecture: It is "like a big container" where all the software components of an application are bundled together tightly. It is usually built as one large system and is one code-base.

Microservice Architecture: It involves structuring an application in the form of a cluster of small, autonomous services modeled around a business domain. The functional modules can be deployed independently, are scalable, are aimed at achieving specific business goals, and communicate with each other over standard protocols.

=====================================================================

Service Discovery (Eureka)

Service Discovery is a hosting application where all different services register with each other. If service A wants to communicate with service B, A will send a request to the host application, the host application redirects the request to service B, whose host application provides a dynamic location for service B.

Service discovery is a design pattern which only gives out a way to sync up all applications in one platform.

A service register is a platform where all services are going to register.

The Service Register maintains details of all the applications whether it is run or not. Service registers store all the applications as key-value pairs, the key is the application name, value is the location provided by the application register only.

@EnableEurekaService: this is a class-level annotation that enables this class as a eureka server.

@EnableEurekaClient: class-level annotation enables this class as a eureka client.

url in yml file to register clients with servers.

RestTemplate: class provided by spring Rest api.

FeignClient - provided by spring cloud Netflix API “open Fign”

Feign **provides an abstraction over REST-based calls via annotation**, by which microservices can use to communicate with each other without writing detailed REST client code

@FeignClient interface level annotation. (name=” service name”)

RestTemplate and FeignClient both can call HTTP rest client.

@EnableFeignClient: class-level annotation on client service.

What is Circuit Breaker

As the number of interactions over the network increases there is a possibility that an abnormality of service may also increase. In that case, we need to have some handy techniques that can prevent or minimize this abnormal behavior. We can call it to fault tolerance. The circuit breaker is this technique, where we stop executing an error service or the service that throw any exception and redirect every request to a custom method(Fallback Method)

Circuit Breaker is a fault tolerance technique that monitors and detects when a service is behaving abnormally. it temporarily rejects incoming request calls until the service becomes healthy. Hence we have Netflix Hystrix is an open-source library that provides this solution

Circuit Breaker implemented by hystrix(@Hystrix())

Generally, we stop the execution of a particular method if it is continuously throwing an error.

Q)What is the Fallback method?

If the actual method of Microservice throws an exception continuously, then we avoid the execution of the actual logic for some time. instead, we redirect the request to the Dummy method that provides the response back to the client request. such a dummy method is called a fallback method. This method provides responses such as “Service not Working” or “Unable to Process”, “try after some time”.

What are the states of Circuit Breaker?

Closed, open, and half-open.

Closed

if the client request is sent to the actual service method, then it is called a close circuit.

Open

If a client request is redirected to fallback methods then such a case is an open circuit.

Half Open

Once the state becomes open, we wait for some time in the open state. After a certain period of time, the states become half-open during this period.

Microservice Design Pattern

1)Aggregator

2)API gateway

3)Chained or chain of Responsibility

4)Asynchronous Messages

5)Database or Shared Data

6)Event Sourcing

7)Branch

8)Command Query Responsibility segregator

9)Circuit Breaker

10)Decomposition

11) Event Driven

Aggregator

Aggregator is a web page(service) which invokes various services to get the required information or achieve the required functionality.

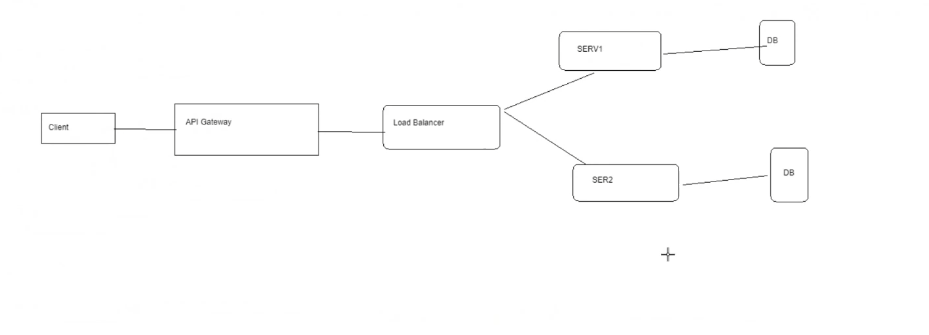
API Gateway

1.How can I request information from multiple microservices?

2.Different UI requires different data to respond.

3.How to handle multiple protocol requests?

The Solution could be the API Gateway Design Patterns.



This pattern is a service that provides a single-entry point in the help of load balancers for certain groups of microservices.

It sits between the client apps and the microservices. It acts as a reverse proxy, routing requests from clients to appropriate services. It can also provide other cross-cutting features such as authentication, authorization, SSL termination, load balancer and cache.

synchronized implementation

feign client

Service discovery is a design pattern which only gives out a way to sync up all applications in one platform.

Load Balancer(only synchronized communication we need)

In load Balancer for example, if we receive 100 requests from clients, it will evenly divide requests into different services. Make sure maximum use of all the services reduces loading time and prevents all requests from going into a single service.

is to distributed load into different service

Client side load balancer: how the load handle

3)Chained or chain of Responsibility?(多个微服务合并给出一个output)

Chained or Chain of Responsibility Design Patterns Produces single output which is a combination of multiple chained services. so if you have 3 services lined up in a chain then the request is received by service A, then Service B, and Service C to generate consolidated(合并 )output. all these services use synchronous HTTP requests.

4)Asynchronous Messages(异步信息交流)

If the client gets blocked or has to wait for a long time in synchronous messaging. and if you do not want to wait for the client for a long time we can use asynchronous messaging. In this type of microservice design pattern, all the services can communicate with each other but do not have to communicate with other services sequentially(依次 ).

5)Database or Shared Data Pattern(共享数据保证每个数据库有足够数据支持)

For every application, there is a huge amount of data in the DB. So when we break into MicroService it is very important to know that each service has a sufficient(充足）{死飞选} amount of data to process a request. so each service can have a separate database or it can have a shared database pre-service.

6) Event Sourcing(sequence of 时间记录资源库那个服务器变动)

The event sourcing design pattern creates events regarding the change in the application state. Also, these events are stored as a sequence of events to help developers track which changes were made and when.

7) Branch pattern(向多个微服务发或者要数据在同一时间)

A microservice may need to get data from multiple sources including other microservice.

Branch Pattern is a combination of Aggregator and Chain of Responsibility design pattern and it allows simultaneous requests/responses. processing from two or more microservices.

8)Command Query Responsibility segregator(实现数据之间交流多个microservice 都有自己数据库)

Every Microservice design pattern has a database per each service or shared database. we cannot implement a query as the data access if it is only limited to one database of service. so in this case we can use the CQRS pattern

According to the CQRS design pattern, the application will be divided into two parts: Command and Query， command party will handle all the requests related to CREATE, UPDATE, DELETE

The query part will handle Materialized views.

9)cricuit breaker

10)Decomposition(拆分monolitith 用有逻辑的方式去拆分成microservice)

The main idea of Microservice is for developers to create small services. Each service has its own functionality but breaking an application into small pieces has to be done logically. so to decompose big applications into small services you can use a Decomposition pattern.

12 Factor approach of microservice

1.codebase

2.dependecies

3.config

4.backing service

5.build, release, run

6.processes

7.port binding.

8.concurrency

9.disposability.一次性

10.dev/prod parity(perədē)

11.logs

12admin process.

1.codebase :

----------

codebase tracked in revision control like Git, Subversion (SVN), bitbucket we can call it as code repo

2. Dependencies

Most programming languages offer a packaging system for distributing supporting libraries.

1) Hystrix,Fallback method.

[Hystrix](https://www.baeldung.com/introduction-to-hystrix) is watching methods for failing calls to related services. If there is such a failure, it will open the circuit and forward the call to a fallback method circuit breaker.

2) Zuul. API gateway

3) Zipkin + Sleuth **monitor microservice application.**

**Trace id is added to service header**

Spring Cloud Sleuth is used to generate and attach the trace id, span id to the logs so that these can then be used by tools like Zipkin and ELK for storage and analysis. Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in service architectures.

**Difference between span id and trace id?**

trace id is uniquely defined as a single request, job, or action.unique for each request

Something like each unique user-initiated web request will have its own traceId.

span id has tracked a unit of work. uniquely defined for each interprocess request.

Think of a request that consists of multiple steps. Each step could have its own spanId(different inter-process requests).

Advantage of the Zipkin?

Zipkin allows you to quickly visualize complex requests, and monitor our applications.

4) Ribbon/ Feign **dynamic load balance**

enables you to achieve fault tolerance in your applications, it balancing out the capacity needed to route application traffic. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.

**Ribbon: Load Balancer Without Eureka**

The ribbon is a client-side load balancer, which gives you a lot of control over the behavior of HTTP and TCP clients. Ribbon's Client component offers a good set of configuration options such as connection timeouts, retries, retry algorithm (exponential, bounded back off) etc

**Feign - Load Balancer using Eureka**

Feign is a declarative web service client, or declarative REST client. It makes writing web service clients easier.

5) Eureka server to get the registered applications from the Eureka server