Web Services – Service which we are providing using or through internet. We use Web services in order to communicate or interact with machine to machine or application to application. It should be interoperable means it should accept the files of irrespective of their technologies like java, php and c etc or platform independent.

XML stands for Extensible Markup Language.

JSON stands for JavaScript Object Notation.

Every web service offers a Service Definition. Which consists of Request/Response Format, Request Structure, Response Structure, Endpoint.

Key terminologoy:-

Message Exchange Format is the format of the Request and the Response.

Service Definition defines the contract between the service provider and service consumer.

Transport defines how a service is called, whether it is Http (over the web) or MQ used to communicate over a queue.

Transport indicates whether the services is exposed over a queue or whether it is exposed over the web.

REST defines an architectural approach whereas SOAP poses restrictions on the format of the XML.

SOAP stands for Simple Object Access Protocol. It defines a specific way of building web services. We use XML as the request exchange format. SOAP define a specific XML request and response structure.

If we are creating SOAP then we have to use this structure

1.SOAP-ENV : Envelope

SOAP-ENV : Header.

SOAP-ENV : Body.

The header contains meta information like authentication, authorization, signatures.

Body is where we put the real content.

SOAP

Format – SOAP XML Request, SOAP XML Response

WSDL (Web Service Definition Language) defines the endpoint basically our service is exposed at.

REST is a term which is coined by roy fielding, he is the guy who also developed HTTP protocol.

RESTful web services try to define services using the different concepts that are already present in Http.

Data Exchange Format – No restriction, JSON is popular

Transport – only HTTP

Service Definition – No standard, WADL/Swagger/…..(Web Application Description Language – machine readable XML description of HTTP – based web applications.

@responseBody will return the bean as is and we return as is, there is message conversion which would happen. The default conversion which is set up by spring boot auto configuration is using Jackson HTTP message converters.

Our beans are converted into JSON with the help of @ResponseBody+ JacksonHttpMessageConvertersConfiguration.

Error page is also the configuration of Auto Configuration(ErrorMvcAutoConfiguration)

PATCH – update part of a resource.

Response Status for REST API

404 – Resource is not found

500 – Server exception

400 – Validation error

200 – Success

201 – created

204 – No Content

401 – Unauthorized

400 – Bad Request

Whenever we want to return the URL of a created resource there is a specific HTTP header we need to make use of that header is called location header.

REST API Documentation :-

1. Manually Maintain Documentation.
2. Generated from code.

Our REST API consumers need to understand our REST API in terms of it resources, Actions, Request/Response Structure.

Generated from code:-

1. Swagger

2. Open API.

Swagger Specification and Swagger Tools were introduced on 2011.

Open API specification created based on Swagger Spec on 2016. Earlier called Swagger Specification.

Versioning of API:-

1.URL

2.Request Parameter

3.Header

4.Media Type

@GetMapping(path = “/person”, params = “version=1”)

Public PersonV1 getfirst()

{

Return new PersonV1(“Lakshmana Rao”);

}

@GetMapping(“/v2/person”)

Public PersonV2 getSecond()

{

Return new PersonV2(new Name(“Lakshmana”, “Rao”));

}

When we want to use Hateoas we need to wrap the user in EntityModel.

EntityModel<User>

WebMvcLinkBuilder

Serialization – Converting object to stream ex- JSON.

Static Filtering – Same filtering for a bean across different REST API.

@JsonIgnoreProperties, @JsonIgnore

Dynamic Filtering – Customize filtering for a bean for specific REST API.

@JsonFilter with FilterProvider

FilterProvider will allows us to define a number of filters.

**Bean Scopes** refers to the lifecycle of Bean that means when the object of Bean will be instantiated, how long does that object live, and how many objects will be created for that bean throughout.

Singleton: Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.

Prototype: A new instance will be created for a single bean definition every time a request is made for that bean.

Request: A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But Only valid in the context of a web-aware Spring ApplicationContext.

Session: Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.

Global-Session: Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.

Actuator provides us with a number of production-ready features, which helps us to monitor and manage our application in production. By default the actuator exposes the health of the application.

HAL – (JSON Hypertext Application Language) it is a simple format that gives a consistent and easy way to hyperlink between resources in our API.

HAL Explorer – An API explorer for RESTful hypermedia APIs using HAL.

MicroServices are the small autonomous services that work together.

It’s a style to develop a single application as a suite of small services, each running its own process, communicating with very lightweight mechanisms, things like HTTP.

Challenges with Microservices:-

1.Bounded context – setting up the boundaries is really big challenge for microservices.

2.Configuration Management – we will be having number of microservices, these microservices will have multiple instances in each environment and there are multiple environments.

3.Dynamic scale up and scale down – Dynamically distribute the load among the active instances.

4.Visibility – What’s happening behind the scenes of these microservices

5.Pack of cards – In microservices architecture we have one microservice calling another, another calling another. So there would be the fundamental for the whole thing. And if that microservice goes down then the entire application might go down. So therefore it is very important to have fall tolerance in our microservices.

Spring cloud – It provides tools for developers to quickly build some of the common patterns in distributed systems.

If service is down hystrix or fault tolerance will helps us to configure a default response.

Advantages of microservices Architecture :-

1.Enables us to adapt new technology and process adaption.

2.Dynamic scaling.

3.Faster release cycles.

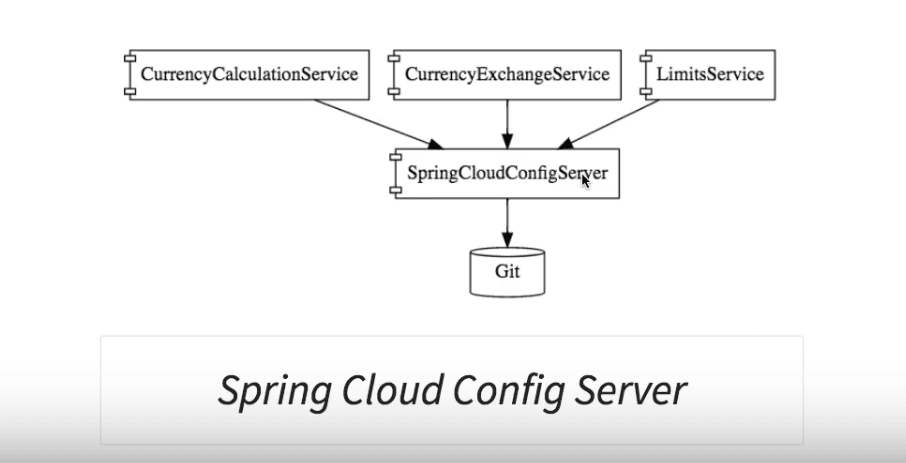
As part of the latest versions of spring boot and spring cloud we will be using spring cloud loadbalancer instead of ribbon.

Spring cloud gateway instead of Zuul.

Resilience4j instead of Hystrix.

We will use docker to containerize microservices.

Creating the project – Creating the project in spring initialzr with adding dependencies like spring web, devtools, Actuator – supports built in endpoints that let us monitor and manage our application such as application health, metrics, sessions etc and Config client – we want to limit service to connect to spring cloud config server.



The values in the application.properties is having less priority compared to the values which are present in our git repository.

Whenever we want to support multiple environments in spring application, we would go for the concept called profiles.

We can get the value of port by using keyword environment.

Creating instances to the application in run configurations -arguments -Dserver.port=8001

For seeing the details what are present in the database we use this command on application.properties

Spring.h2.console.enabled = true

By default of spring boot 2.4+ the load of data.sql is done before the tables are created. And that’s why we need configure to differ the execution of data.sql by

Spring.jpa.defer-datasource-initialization = true

The more resources we use more money we have to pay.

Monolithic is having large number of resources and

REST template can be used to make rest API calls.

New RestTemplate().getForEntity(null, null)

First parameter refers to the URL that we would want to work.

Second parameter refers to the response should be converted into the structure of particular class.

FEIGN is a framework used to call other microservices. In order to use this service we need to add the dependency on the pom file of currency conversion spring-cloud-openfeign

Once we add the dependency we need to enable it in spring boot application using annotation @EnableFeignClients

FEIGN makes us very very easy to call REST API and it also helps us to do load Balancing very easily.

If our url went down and a new instance was brought up on 8002, then we have to change the configuration of this application or the code of this application all the time and that’s the reason why we go for service registry or a naming server.

In a microservice architecture all the instances of all the microservices would register with a service registry.

In order to stop the eureka register by itself we need to add the properties like

Eureka.client.register-with-eureka = false

Eureka.client.fetch-registry = false

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.

Inside the currency conversion microservice there is load balancer component which is talking to the naming server, finding the instances and doing automatic load balances between them.

Microservices have lots of common features like authentication, authorization, logging, rate limiting

All the above feature we can implement using API gateway.

In the older version of spring cloud the popular API gateway to use was Zuul. Now recommended gateway is spring cloud gateway.

We have to enable the spring cloud application gateway in application.properties in order to work with gateway. By default it is false or disabled

Spring.cloud.gateway.discovery.locator.enabled = true

So this httpbin.org exposes a lot of API. We are hitting one of these APIs and it’s returning a response back with a few headers.

@Bean

public RouteLocator gatewayRouter(RouteLocatorBuilder builder)

{

Function<PredicateSpec, Buildable<Route>> routeFunction

// Function<PredicateSpec, Buildable<Route>> routeFunction

= p -> p.path("/get")

.filters(f -> f.addRequestParameter("MyHeader", "MyURI")

.addRequestParameter("Param", "MyValue"))

.uri("http://httpbin.org:80");

return builder.routes().route(routeFunction)

.build();

}

Using this builder we can customize the routes which we would want to use.

.route(p -> p.path("/currency-conversion-new/\*\*").filters(f -> f.rewritePath(null, null).uri("lb://currency-conversion"))

The first argument is what is the string to be replaced.

The second argument is what should it be replaced with.

.route(p -> p.path("/currency-conversion-new/\*\*")

.filters(f -> f.rewritePath("/currency-conversion-new/(?<segment>.\*)", "/currency-conversion-feign/${segment}"))

.uri("lb://currency-conversion"))

Spring cloud gateway is an awesome way to route our APIs and implement our crosscutting concerns like Security, Monitoring/metrics. This is built on top of Spring Webflux(Reactive Approach). We can match routes on any request attribute and we can define predicates and filters.

Predicate is a functional interface which represents a predicate (boolean-valued function) of one argument. It is defined in the java.util.function package and contains test() a functional method. Boolean test(T t)

For spring boot there is a circuit breaker framework which is available which is called Resilience4j

In the previous versions of spring boot and spring cloud Netflix Hystrix was the recommended circuit breaker framework.

Fault Tolerance

Consider a scenario in which six microservices are communicating with each other. The microservice-5 becomes down at some point, and all the other microservices are directly or indirectly depend on it, so all other services also go down.

The solution to this problem is to use a fallback in case of failure of a microservice. This aspect of a microservice is called fault tolerance.

**Fault tolerance** can be achieved with the help of a **circuit breaker**. It is a pattern that wraps requests to external services and detects when they fail. If a failure is detected, the circuit breaker opens. All the subsequent requests immediately return an error instead of making requests to the unhealthy service. It monitors and detects the service which is down and misbehaves with other services. It rejects calls until it becomes healthy again.

Resilience4j is a lightweight fault tolerance library designed for functional programming. Resilience4j provides higher-order functions (decorators) to enhance any functional interface, lambda expression or method reference with a Circuit Breaker, Rate Limiter, Retry or Bulkhead. You can stack more than one decorator on any functional interface, lambda expression or method reference. The advantage is that you have the choice to select the decorators you need and nothing else.

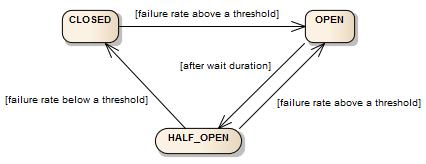
NOTE: Resilience4j 2 requires Java 17.

resilience4j.retry.instances.sample.maxAttempts = 5 - Bydefault @Retry annotation will make 3 retries, if we want to change it to 5 then we need to add the property in application.properties

resilience4j.retry.instances.sample.waitDuration = 1s - it is the wait time between the retries

resilience4j.retry.instances.sample.enableExponentialBackoff = true - by using this property we can increase the wait time exponentially

The CircuitBreaker is implemented via a finite state machine with three normal states: CLOSED, OPEN and HALF\_OPEN and two special states DISABLED and FORCED\_OPEN.



The CircuitBreaker uses a sliding window to store and aggregate the outcome of calls. You can choose between a count-based sliding window and a time-based sliding window. The count-based sliding window aggregrates the outcome of the last N calls. The time-based sliding window aggregrates the outcome of the calls of the last N seconds.

Rate Limiting – Basically rate limiting is all about saying in 10 seconds I want to only allow 10,000 calls to the sample api

Basically we are setting up the time and during that time period I only want to allow a specific number of calls

We can also configure how many calls are allowed that’s called bulkhead.

Patterns makes microservices work well together

Technologies are libraries and frameworks to solve common problems.

We are building service for reusability and don’t have the clear idea about where it is going to be used.

Rate limiting is a software engineering strategy that allows creators and maintainers of API infrastructures to control access to their APIs. The number of calls that any consumer can make is checked during a particular time.

Api gateway requires following dependencies – DevTools, Actuator, Eureka Disovery Client, Gateway (Spring cloud routing).