Spring Boot

simplilearn

TECHNOLOGY

Spring Cloud

Learning Objectives

By the end of this lesson, you will be able to:

- Identify and describe Spring Cloud's components and features
- Differentiate Spring Cloud from Spring Boot
- Configure a Spring Cloud server
- Evaluate Feign REST client and Eureka Naming Server in Spring Cloud

A Day in the Life of a Full Stack Developer

As part of an enterprise application project, you are utilizing Spring Boot as the framework. However, you seek to enable swift application development and establish cloud-based deployment, prompting you to opt for Spring Cloud.

Additionally, externalized configuration is required for both the server and client aspects.

To achieve these objectives, you will research and gain more insights into Spring Cloud, configure it appropriately, and familiarize yourself with its components.

Overview of Spring Cloud

Spring Cloud

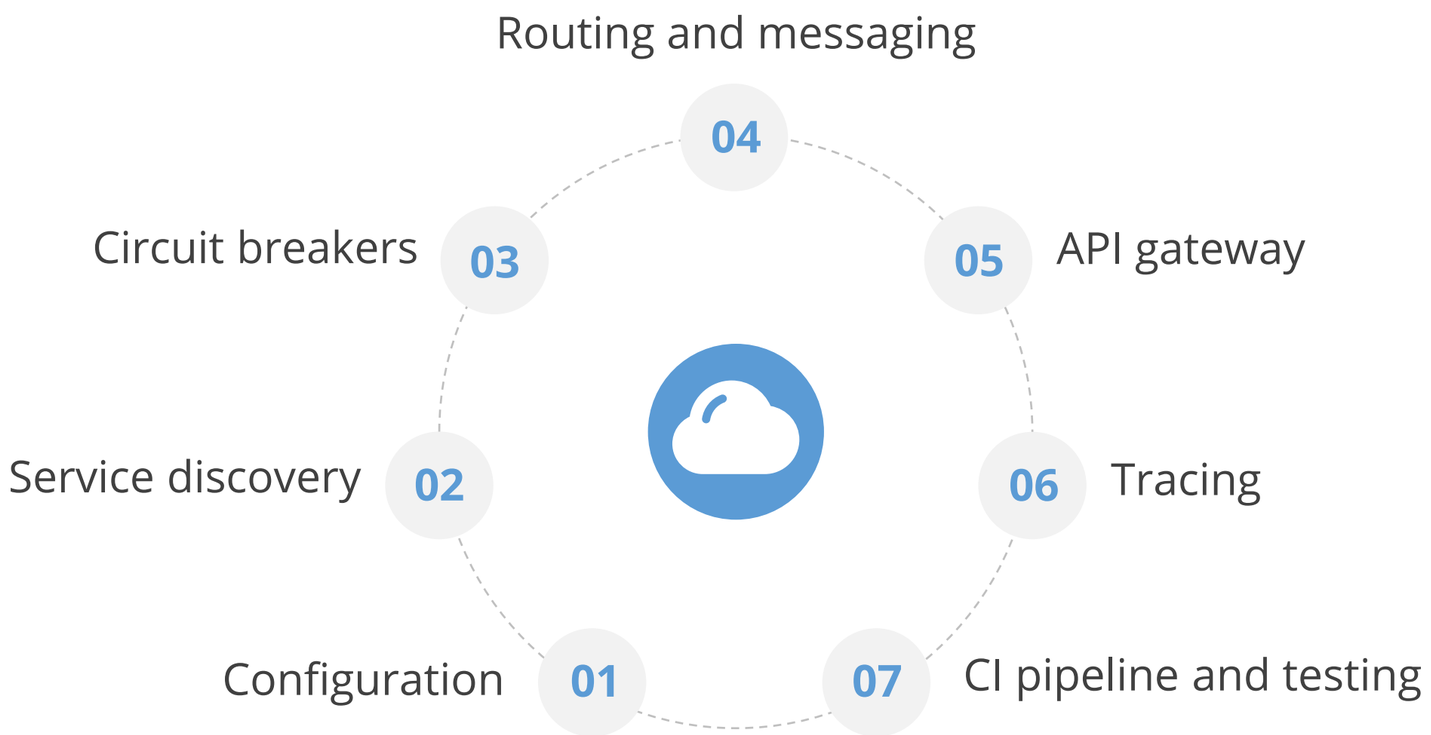
It is a Spring module that gives the Spring framework the RAD (Rapid Application Development) capability.

Spring Cloud framework helps to construct cloud-based allocation.

Spring Cloud

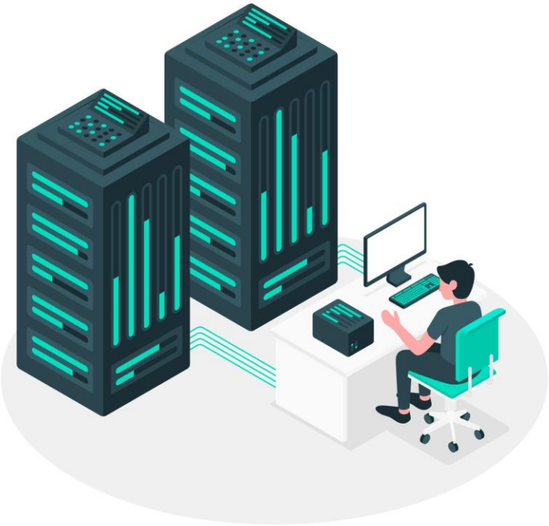
It gives developers the tools needed to quickly design patterns in distributed systems.

Components of Spring Cloud:



Configuration

It provides an externalized configuration in a distributed system on both the server and client sides.



External properties can be managed with a config server for the applications.

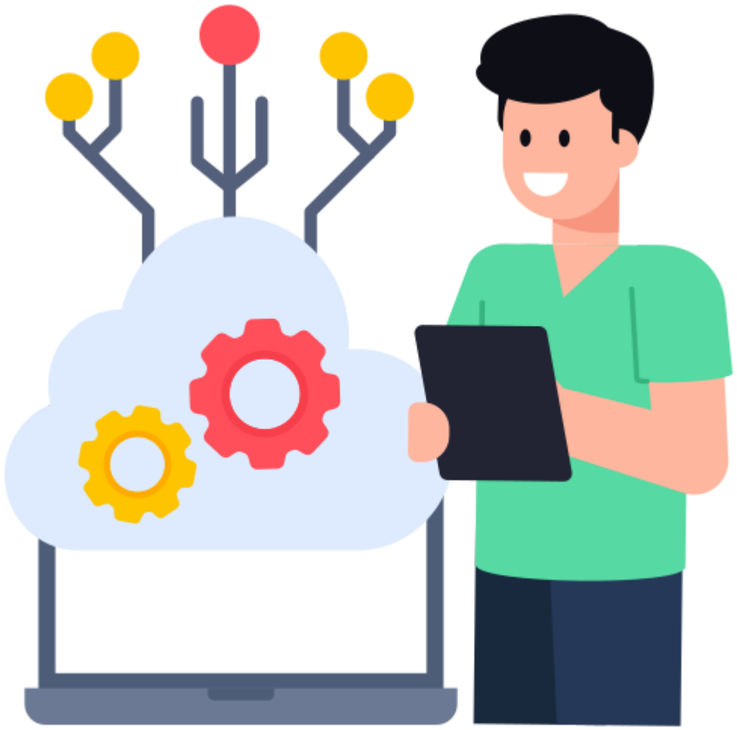
Configuration

The Spring Cloud can use the following to store config:

- Git
- SVN file system
- Vault

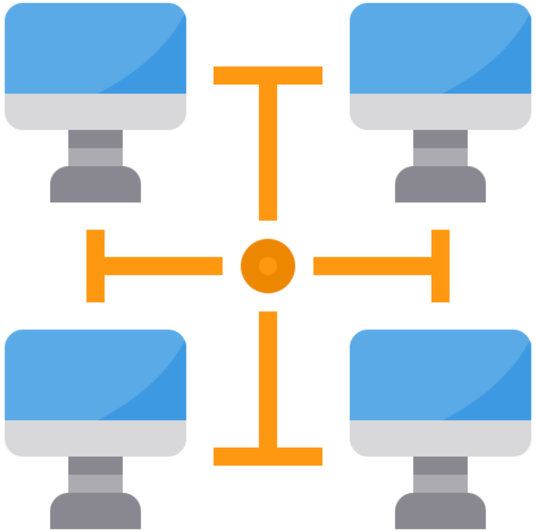
Configuration

The Config clients retrieve the configuration client from the server on the startup.



Service Discovery

It is the automatic detection of devices and services over a network.



It is the process of connecting an application and a microservice in a distributed context.

Service Discovery

In service discovery implementation:

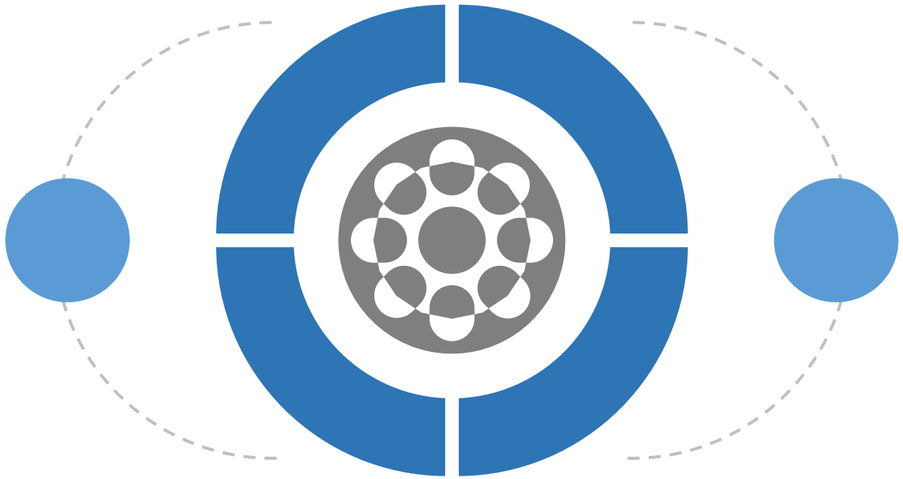
Central server maintains the address's global view

Clients connected to the main server can update and retrieve the address

Service Discovery

Two discovery patterns:

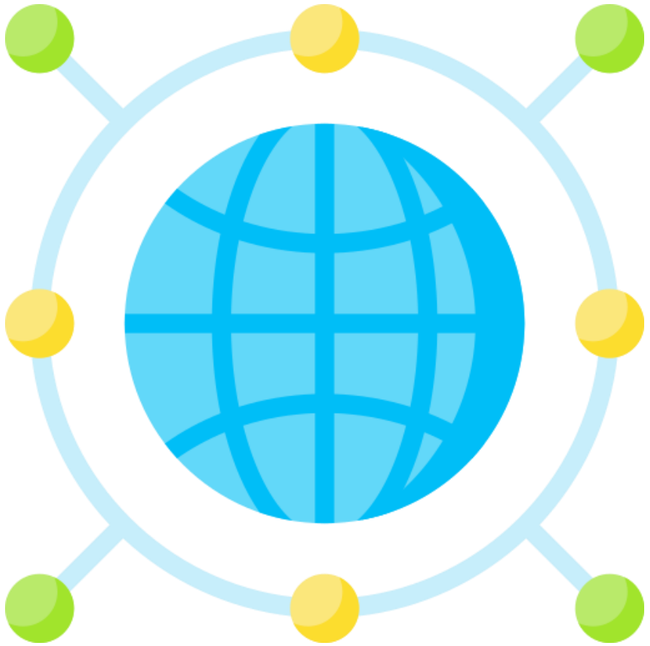
Client-side discovery



Server-side discovery

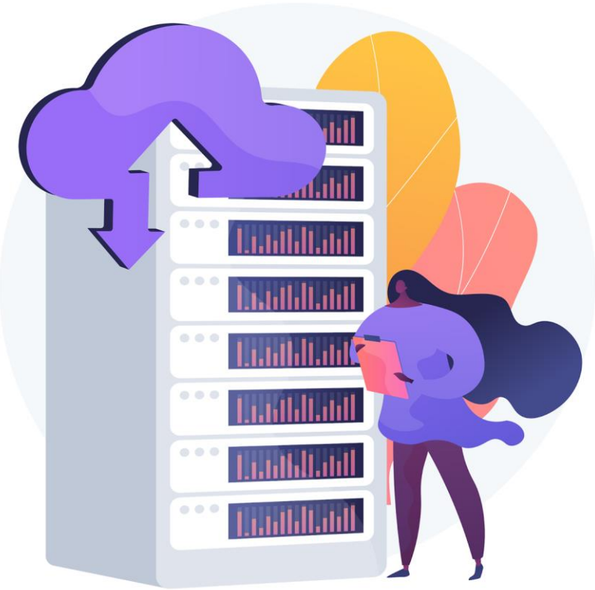
Client-Side Discovery

It takes care of determining the available network locations.



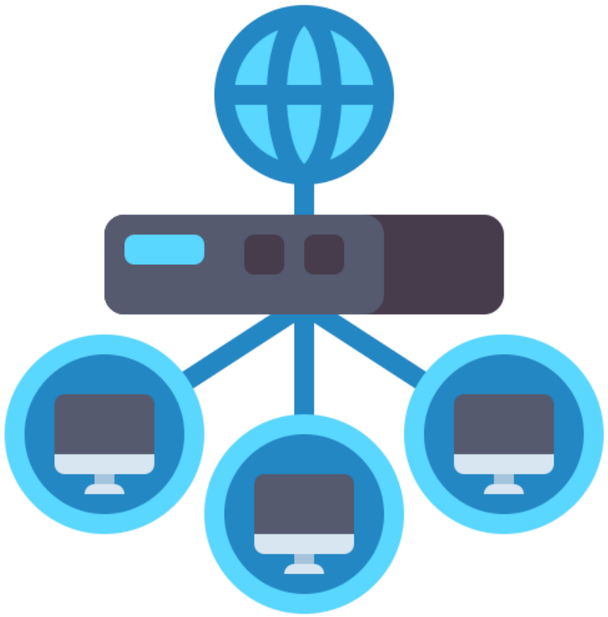
Client-Side Discovery

The client chooses one of the available services and submits the request using the load-balancing method.



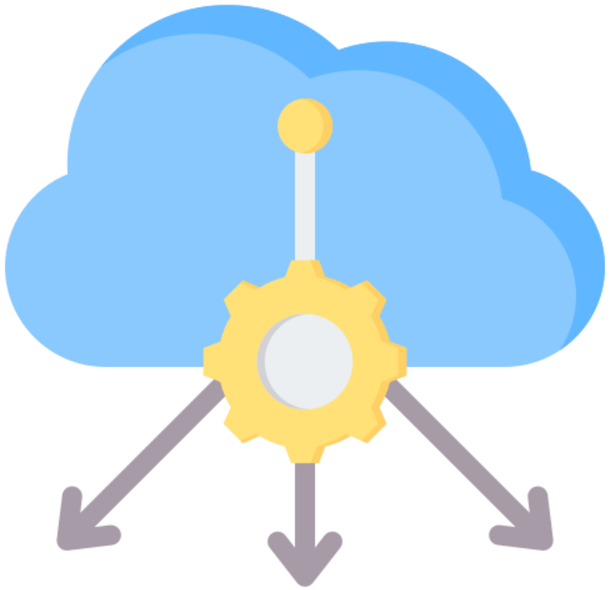
Server-Side Discovery

Here, the client makes an HTTP request to the service through the load balancer.



Server-Side Discovery

The load balancer contacts the service registry and routes each request to an available service instance.

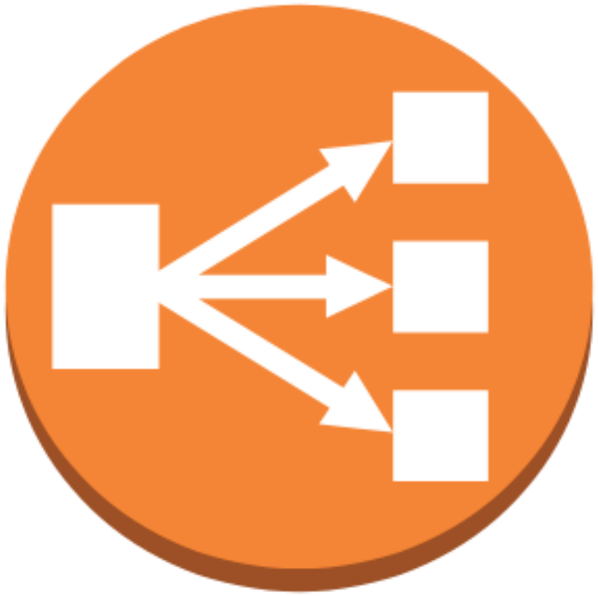


Server-Side Discovery

The service instances are registered and deregistered with the service registry.

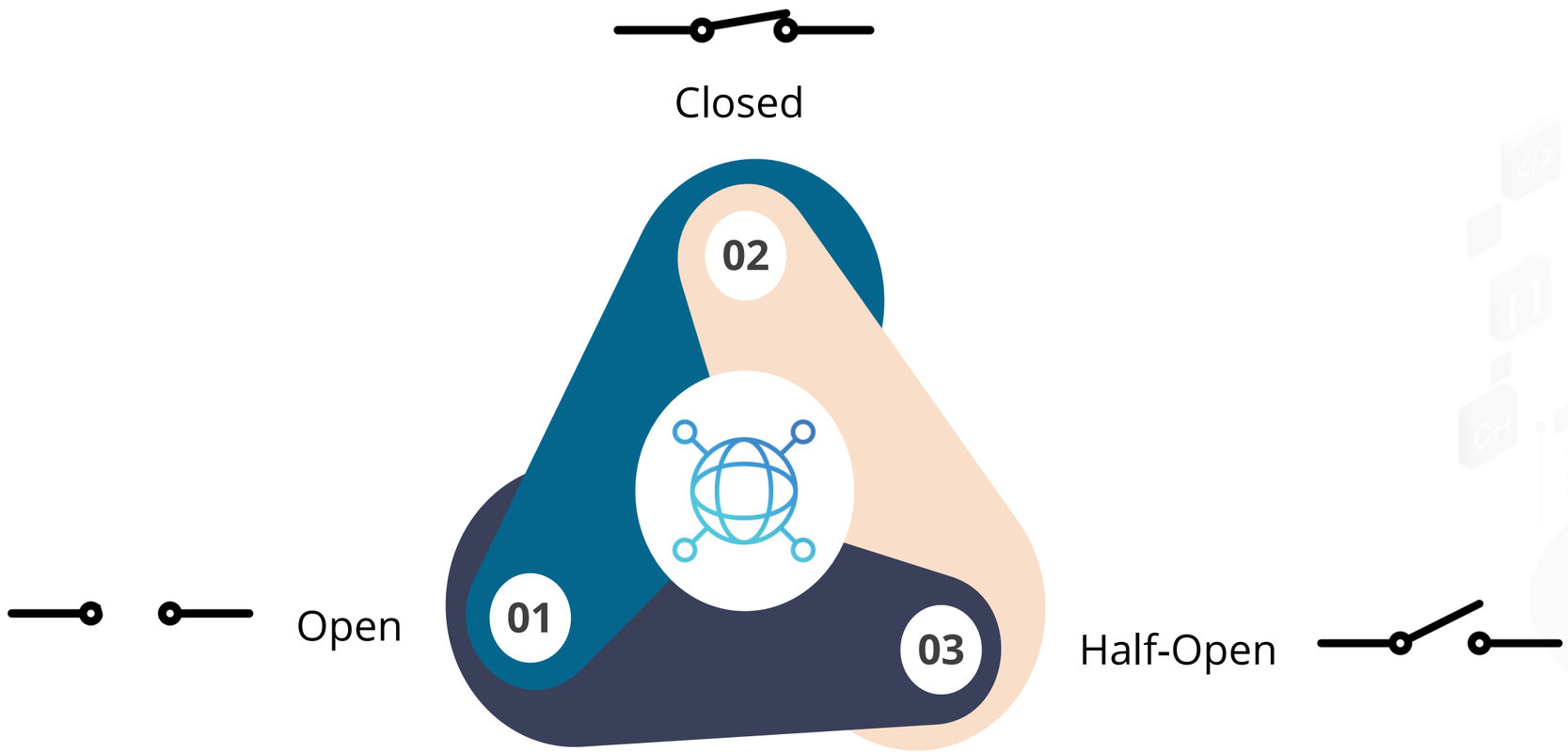
Server-Side Discovery

AWS ELB is an example of server-side discovery that balances external traffic from the internet.



Circuit Breakers

When all the services fail, the circuit breakers handle the situation. It has three states:



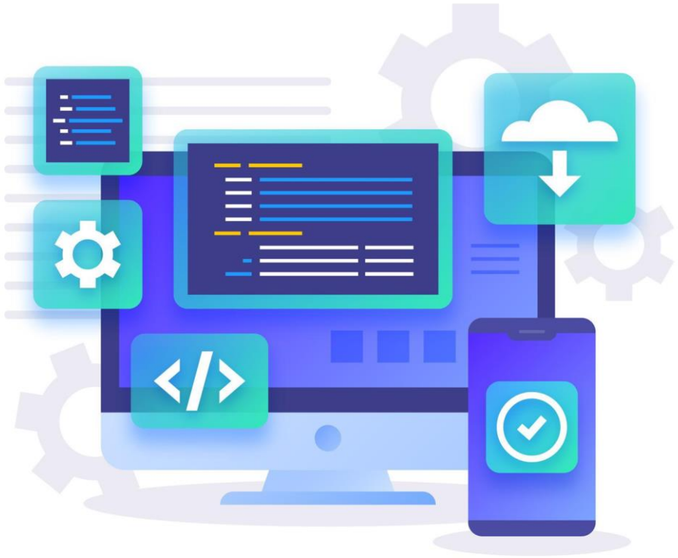
Circuit Breakers

Calls pass through to the supplier microservices and respond without any latency.

Closed state

Open state

Half-open state



Circuit Breakers

It returns an error call without executing the function.

Closed state

Open state

Half-open state

Circuit Breakers

The circuit turns to a half-open state when:

Closed state

Open state

Half-open state

The function execution is timed out

The function determines whether the underlying problem still exists or not

Circuit Breakers

The following are the characteristics of circuit breakers:

It returns all external calls to the service with an error.

It uses monitoring and the feedback mechanism and makes the trial call.

If the call returns successful, the circuit switches to the CLOSED state.

If the call to the supplier is timed out, the circuit remains in the OPEN state.

Routing and Messaging

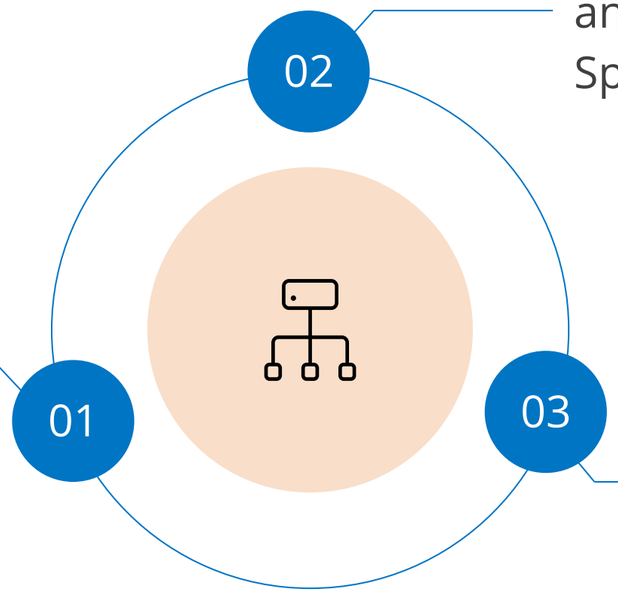
The following are the characteristics of routing and messaging:

Comprises of many microservices

Supports communication via messaging or the HTTP request

API Gateway

The following are the characteristics of API gateway:



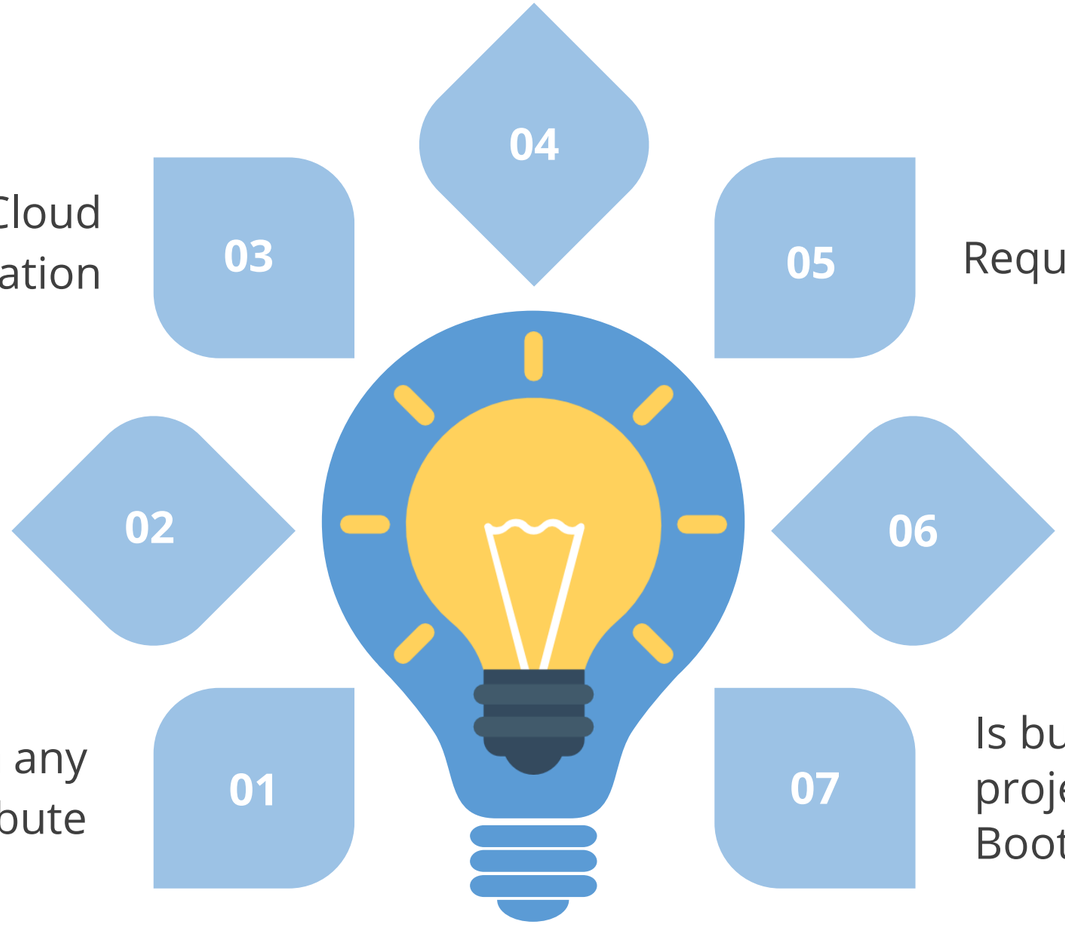
02 Provides the library for building an API gateway on the top of Spring MVC

01 Allows to route API requests to connect the services

03 Provides cross-cutting concerns

API Gateway: Features

The following are the features of API gateway:



01 Matches routes on any requested attribute

02 Predicates and filters are specific to the routes

03 Provides Spring Cloud discovery client integration

04 Facilitates writing the predicates and the filters

05 Requests rate limiting

06 Helps to rewrite the path

07 Is built on Spring framework 5, project reactor and the Spring Boot 2.0

Tracing

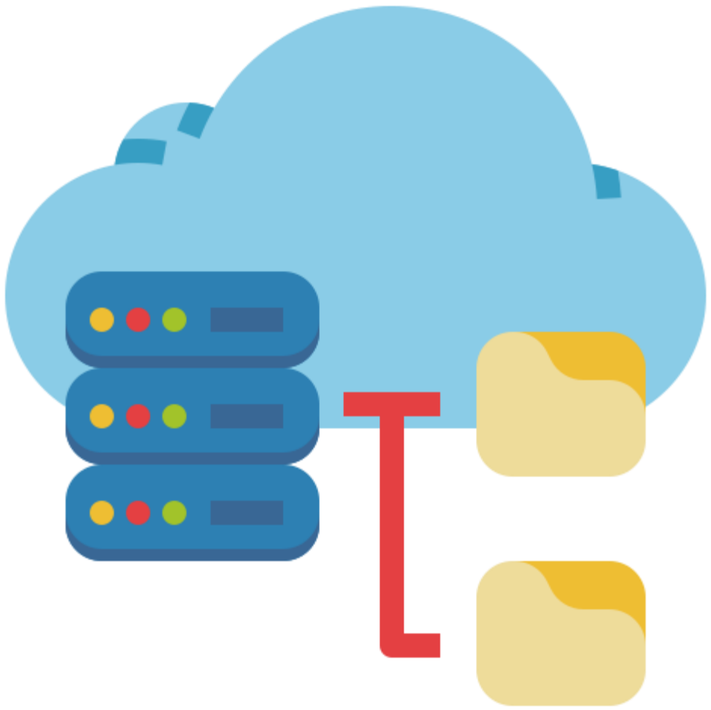
Spring Cloud's tracing feature enables obtaining application data with a single request.

Tracing

Tracing increases requests to various microservices.

Tracing

To enable tracing, add the Spring Cloud Sleuth library to the project.



Sleuth

Sleuth is responsible for recording the timing used for latency analysis.

Sleuth

Zipkin

It is a distributed tracing tool designed to analyze latency problems inside the microservice architecture.

Zipkin

Zipkin

Start by adding the spring-cloud-starter-Zipkin dependency

Zipkin

In microservices, the input traffic volume is very high, so a small amount of data cannot be collected.

The Spring Cloud sleuth provides the sampling policy and allows the input traffic to be sent to Zipkin for analysis.

Spring Cloud

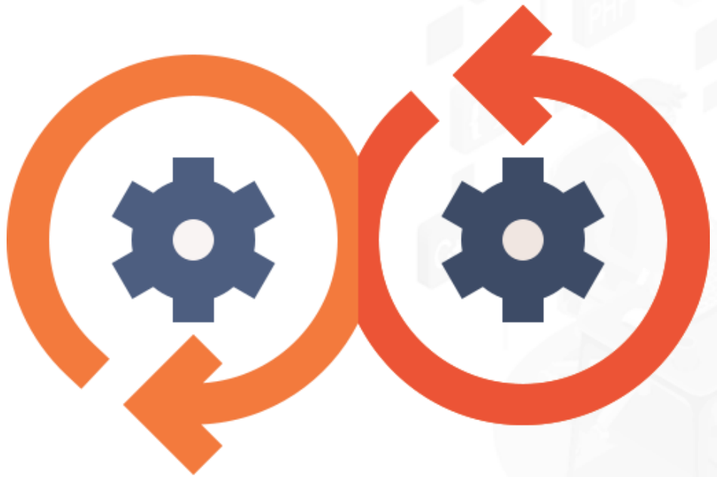
Add the spring-cloud-sleuth-stream dependency

CI Pipeline and Testing

Spring Cloud pipeline helps to build the application pipeline automatically.

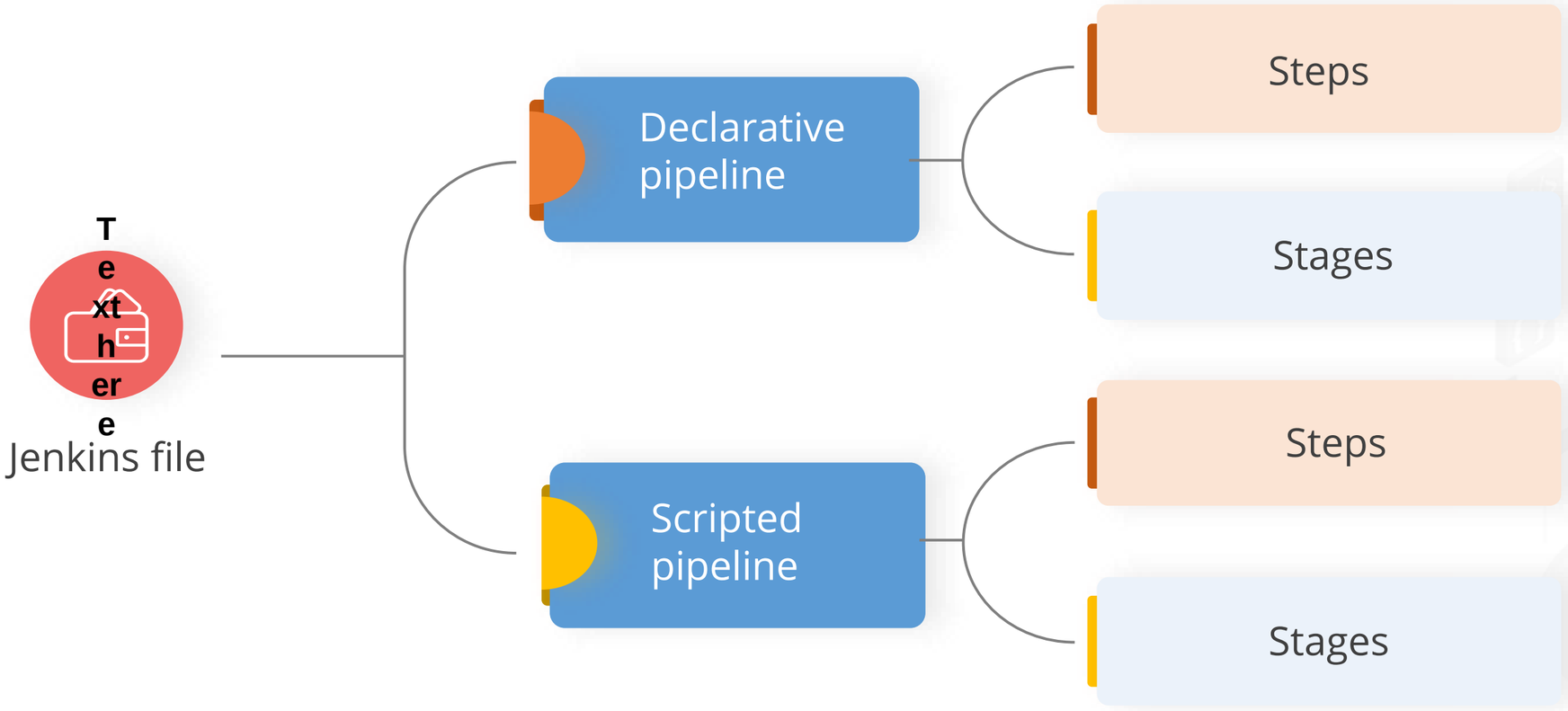
It needs building, testing, and deploying cloud-native applications.

Jenkins pipeline is a collection of tools for modeling complex pipelines as code.



CI Pipeline and Testing

The pipeline is written into the text file called Jenkins file.

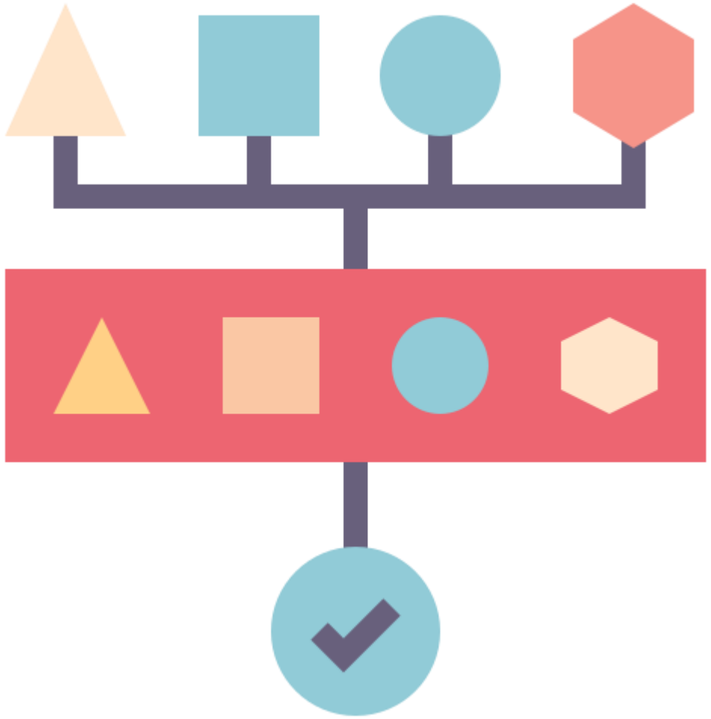


Note

The steps are the fundamental part of the pipeline as they tell the Jenkins server what to do.

CI Pipeline and Testing

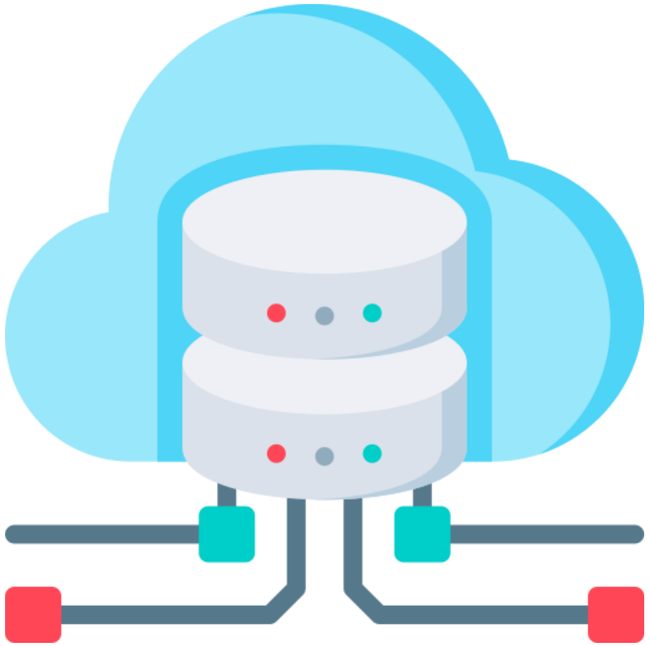
Stages are a significant part of the pipeline as it logically groups the steps.



Spring Cloud: Features

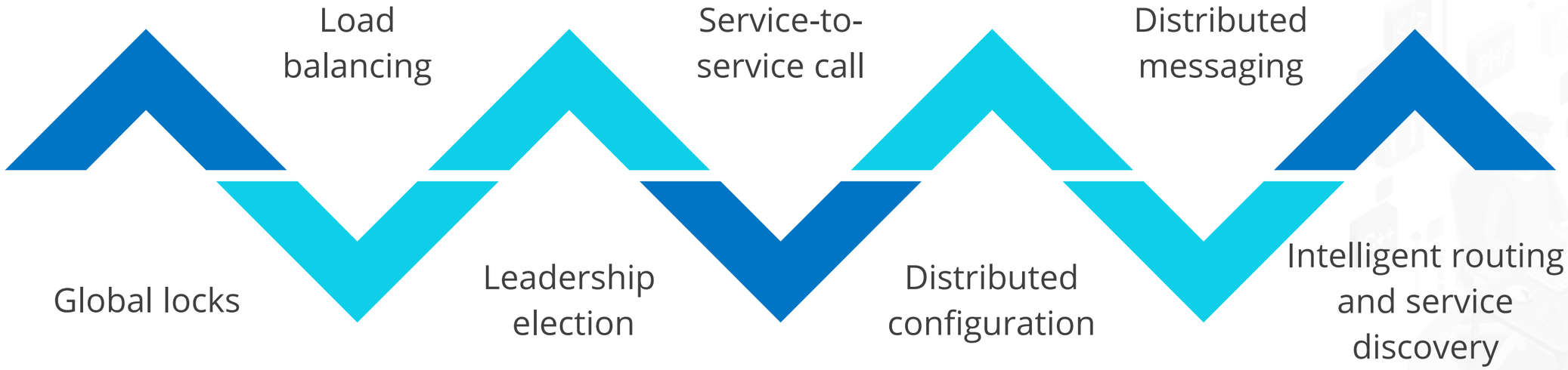
Spring Cloud: Features

Spring Cloud builds the concept of Spring Boot.



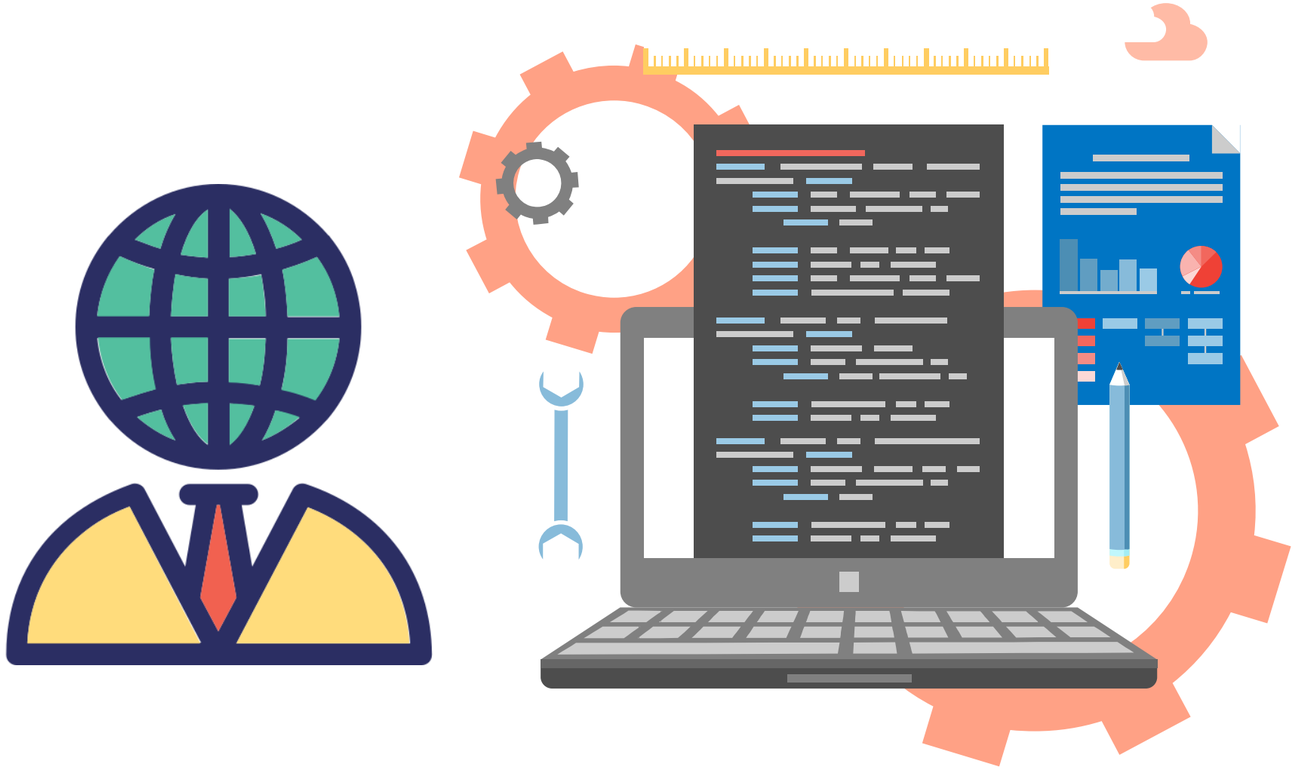
Spring Cloud: Features

Spring Cloud is built on some of the standard building blocks of the Spring framework:



Global Locks

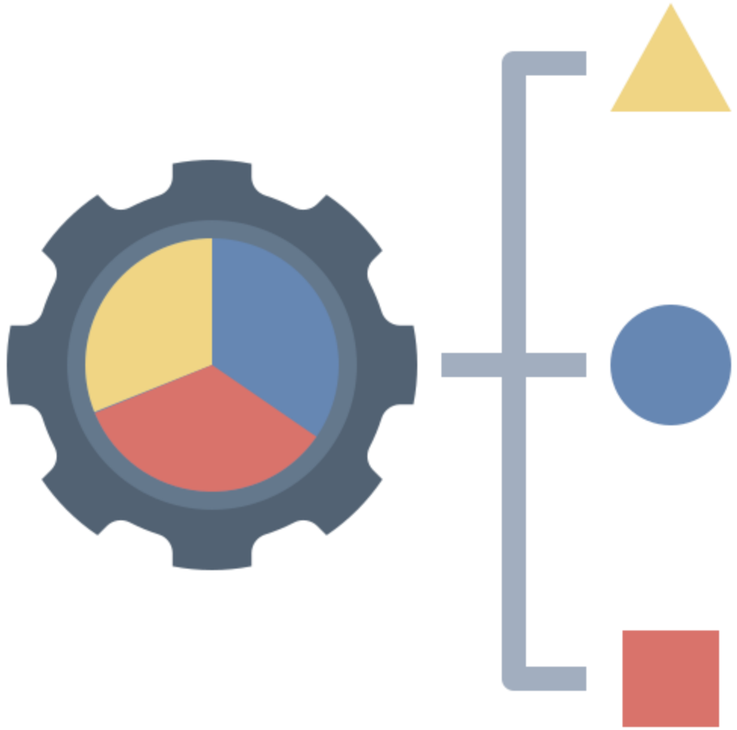
It is used to ensure that no two threads simultaneously access the same resource at the same time.



The programmer uses the mechanism to remove such situations.

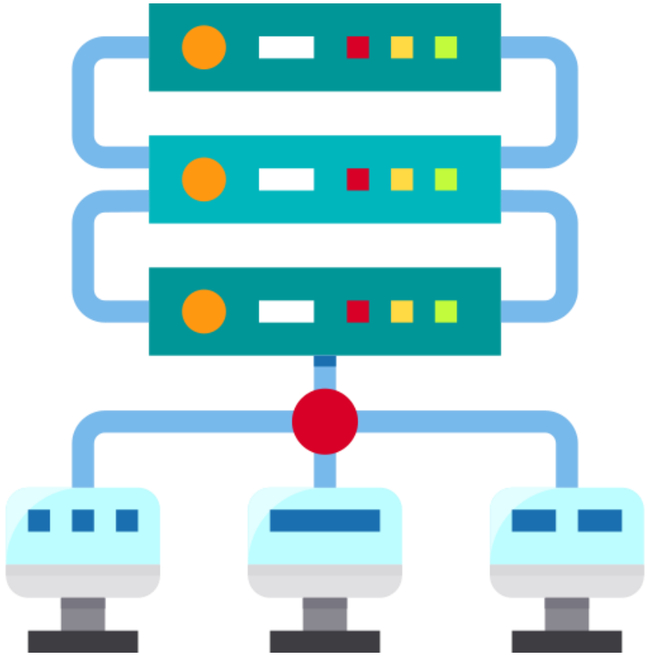
Global Locks

Each thread acquires the lock and operates on the resources.



Load Balancing

It distributes the network traffic to multiple backend servers or the server pool.



Load Balancing

Minimizes response time

Increases efficiency

Maximizes throughput

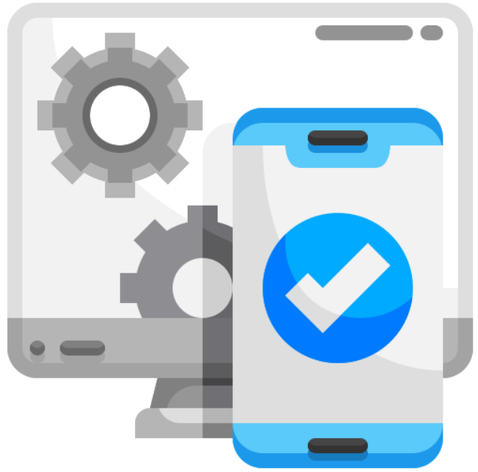
Optimizes resource uses

Note

It avoids the overload of any of the single resources.

Leadership Election

It allows the application to work with the other application through the third-party system.



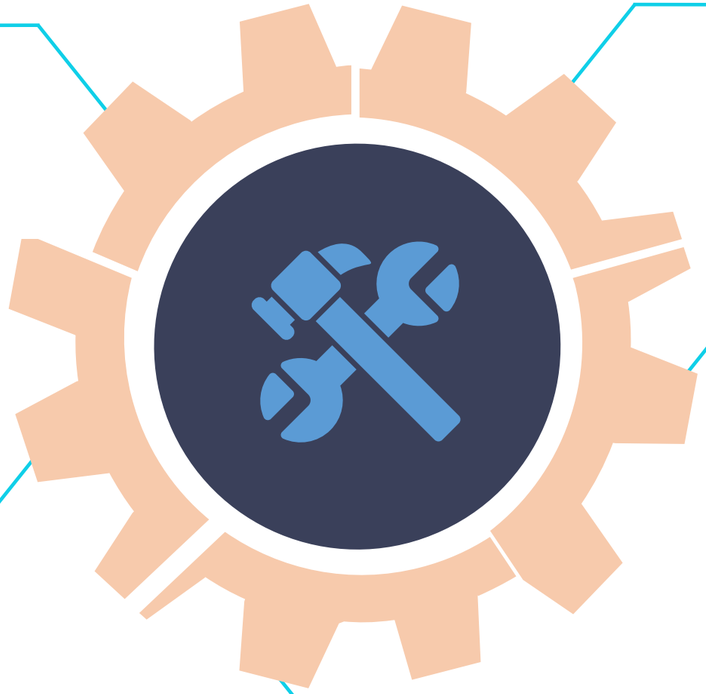
It provides the global state or global ordering without sacrificing availability.

Service-to-Service Spring Call

It explains the method through which a microservice interacts with other dependent microservices via the service registry.

Service-to-Service Spring Call

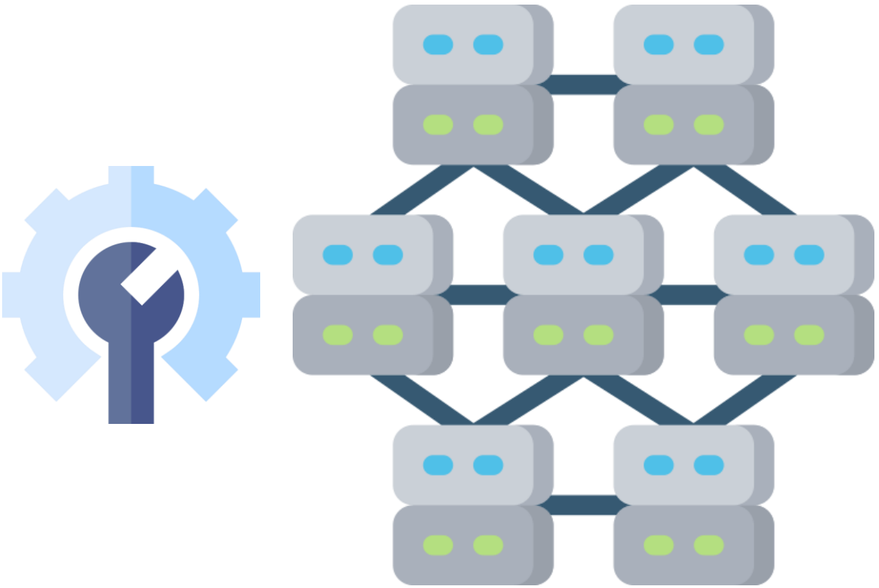
The sequence followed in a service-to-service call:



- Calling the rest endpoint
- Registering the service
- Fetching the registry
- Resolving the underlying IP address
- Finding the downstream service

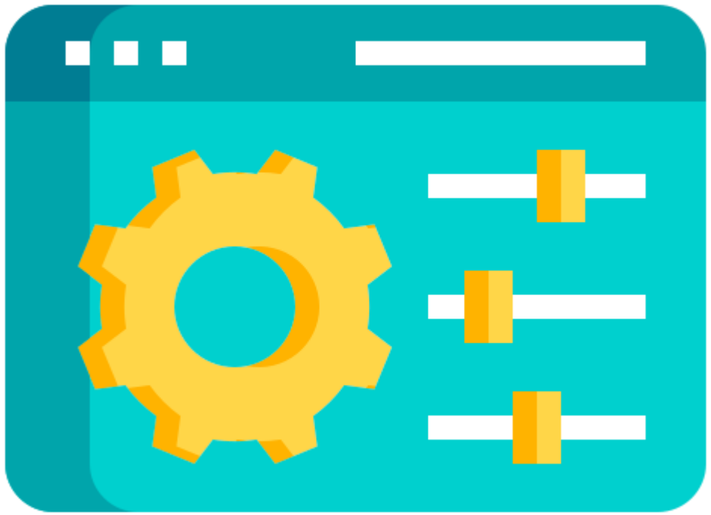
Distributed Configuration

It is used to configure every instance of the microservices.



Distributed Configuration

The Spring Cloud config server provides client-side support for externalized configuration in a distributed system.



Distributed Configuration

Distributed systems provide access to a central place to manage the external properties of the applications.

Distributed Messaging

A distributed messaging system benefits:

01 Reliability

02 Scalability

03 Persistence

Distributed Messaging

Messaging pattern follows the publish-subscribe model.

The sender is called the publisher, and the receiver of the message is called the subscriber.

Distributed Messaging

Popular high-throughput messaging systems are Apache Kafka and RabbitMQ.

Spring Cloud Configuration

Problem Statement:

You have been asked to understand how to manage an application through Spring Cloud configuration.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a Spring Boot project with cloud configuration
2. Configuring the Spring Cloud in Eclipse IDE
3. Configuring the Git repository
4. Setting up the Git repository path for the Spring Cloud Configuration
5. Deploying the Spring Cloud Configuration project

Spring Cloud vs. Spring Boot

Spring Cloud vs. Spring Boot

Spring Cloud is the framework that provides the facilities to use the cloud services in our application, whereas Spring boot is a rapid application development platform.

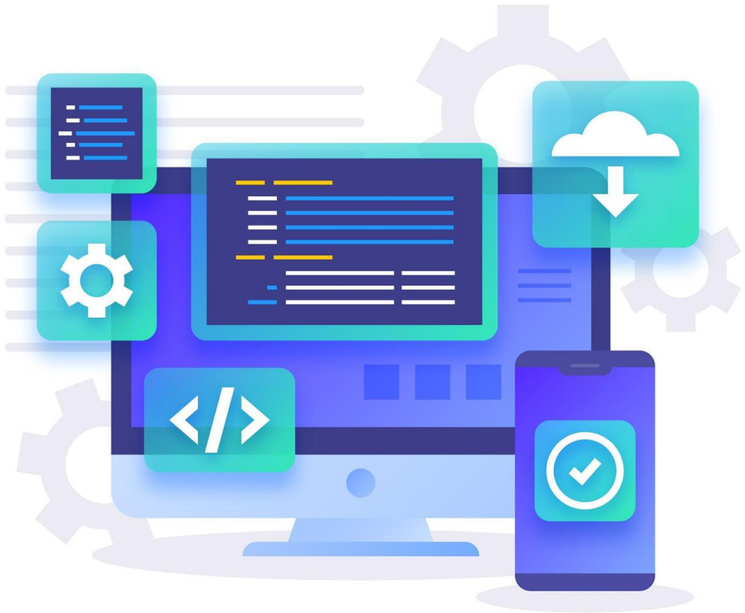
Spring Cloud vs. Spring Boot

Spring Cloud acts as the container Orchestration tool.

Spring Boot helps to develop Restful APIs.

Spring Cloud vs. Spring Boot

Spring Cloud gives a developer-friendly environment for developing and deploying microservices.

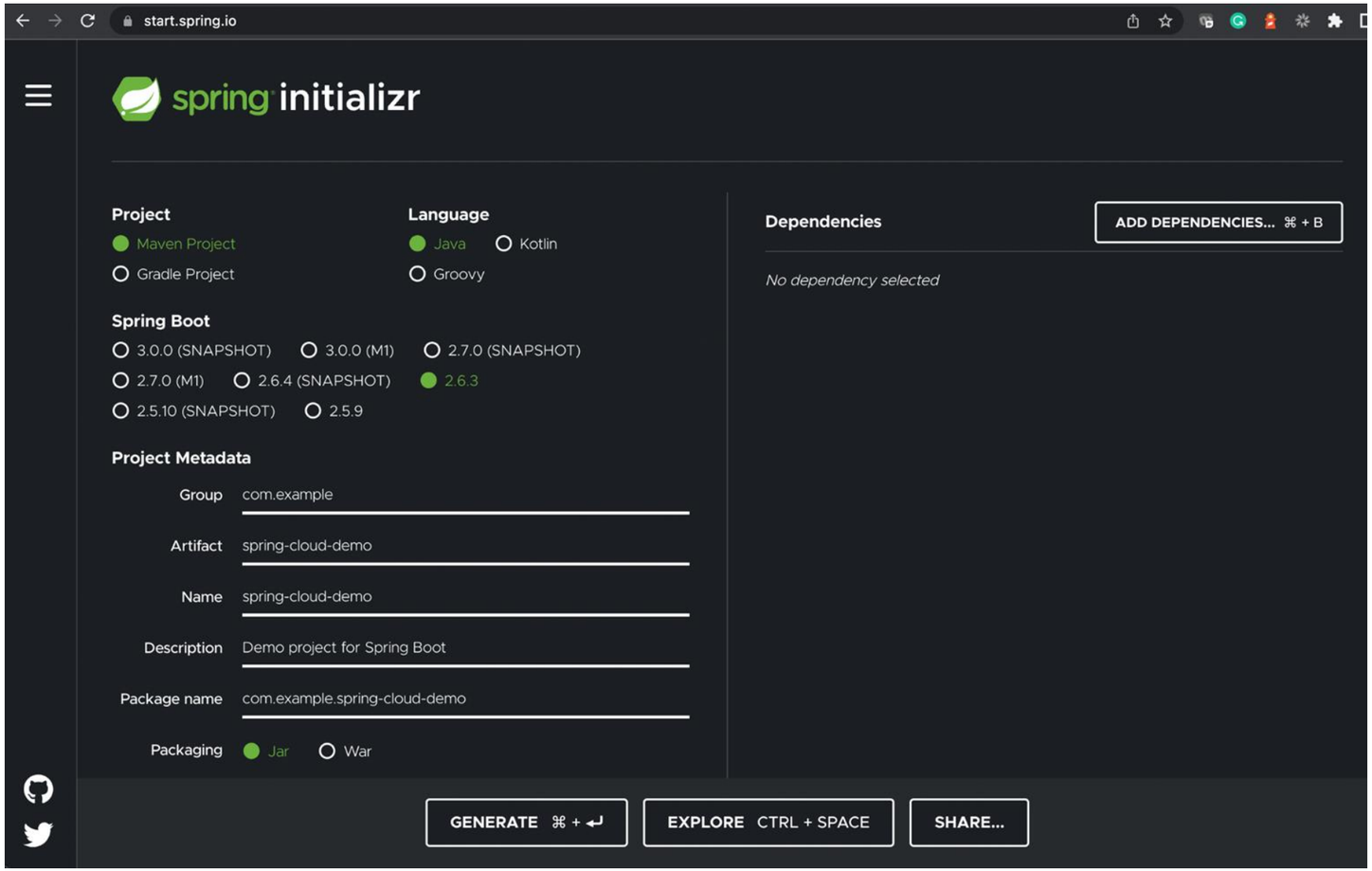


Spring Boot develops and runs standalone web applications and microservices in less time.

Setting Up Spring Cloud Config Server

Setting Up Spring Cloud Config Server

Step 1: Create the maven project using the Spring initialize https://..../



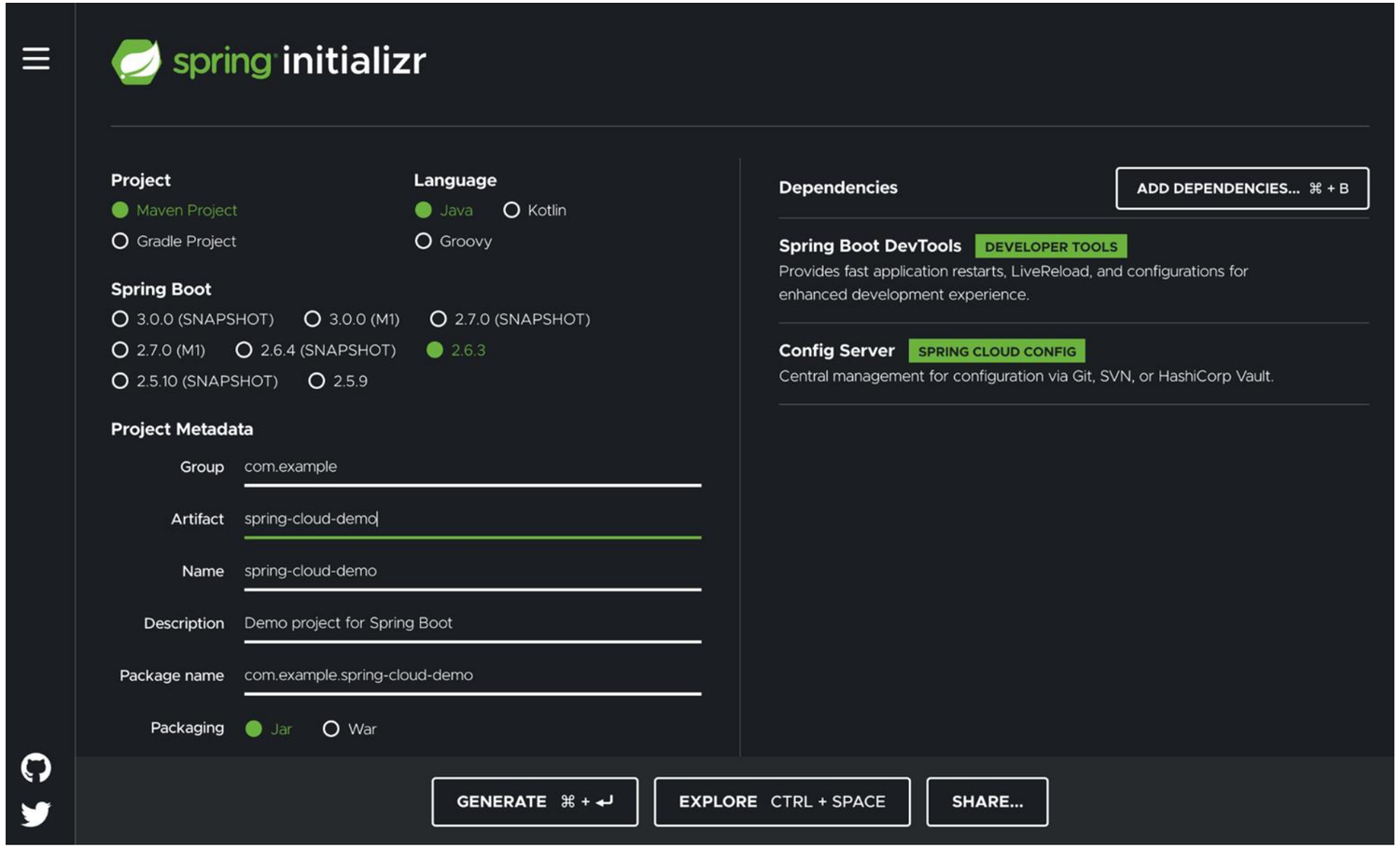
Setting Up Spring Cloud Config Server

The next steps to follow are:

2. Select spring boot version 2.x or a newer release, but avoid using snapshot versions
3. Provide the group name
4. Provide the artifact iID

Setting Up Spring Cloud Config Server

Step 5: Add the Spring Boot dev tools and the config server dependencies



Setting Up Spring Cloud Config Server

The below are the next steps to follow:

- Click on the GENERATE button
- Open Eclipse and import the downloaded Maven project

Setting Up Spring Cloud Config Server

Create a Git repository and configure the Spring Cloud config server.

Installing the Local Git

Step 1: Download and install Git, if not installed

Installing the Local Git

Step 2: Create the Git repository and store the required files

Installing the Local Git

Access the files from the Spring Cloud config server, open the Git bash, and type the following command:

```
mkdir  my-spring-cloud-repo
cd  my-spring-cloud-repo/
```

Installing the Local Git

Initializing the new Git repository:

```
git init
```

Note

It initializes an empty Git repository.

Installing the Local Git

Step 3: Move to the spring-cloud-demo project and add the link to the specific folder:

```
Build Path-> Configure Build Path
```

Installing the Local Git

Step 4: Select the source tab. Click on the link source and browse the folder:

```
my-spring-cloud-repo.
```

Step 5: Right click on the folder -> New -> Other-> File-> Next-> file name:limits-service-properties-> finish

Installing the Local Git

Step 6: Write this code in the properties file:

```
limits-service.minimum=1
limits-service.minimum=11
```

Installing the Local Git

Step 7: Add the files:

```
git add -A
```

Installing the Local Git

Step 8: Execute the command to commit the changes in the repository:

```
git commit -m “spring cloud initial commit”
```

Note

It records or snapshots the file permanently in the version history.

Setting up Spring Cloud Config Client

Problem Statement:

You have been asked to set up a Spring Cloud Config Client to retrieve application configuration from a distant Git repository.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating the Spring Starter project
2. Configuring db connection
3. Running the application

Feign REST Client

Feign REST Client

Feign is the declarative web service (HTTP client) developed by Netflix.

It helps to simplify API clients.

Feign REST Client

To use Feign, create the interface and annotate it. The Feign REST client:

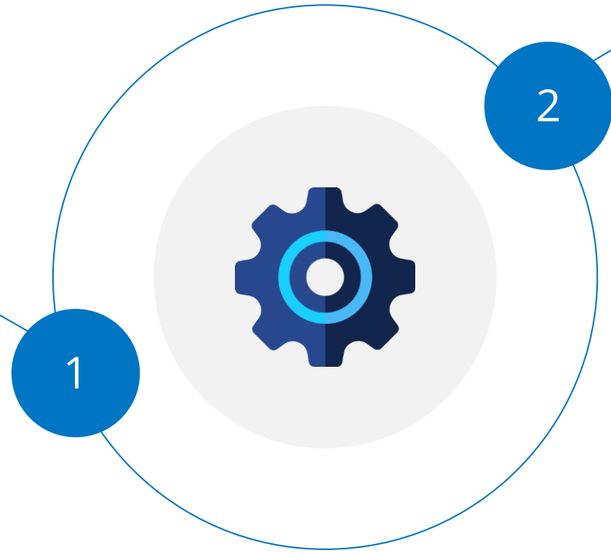
- Provides pluggable annotation support
- Creates the REST API clients

Feign REST Client

Developers can use declarative annotations to call the REST services.

Feign REST Client

Spring Cloud provides the OpenFeign integrations for Spring Boot apps through:



1. Auto-configuration
2. Binding to the Spring environment

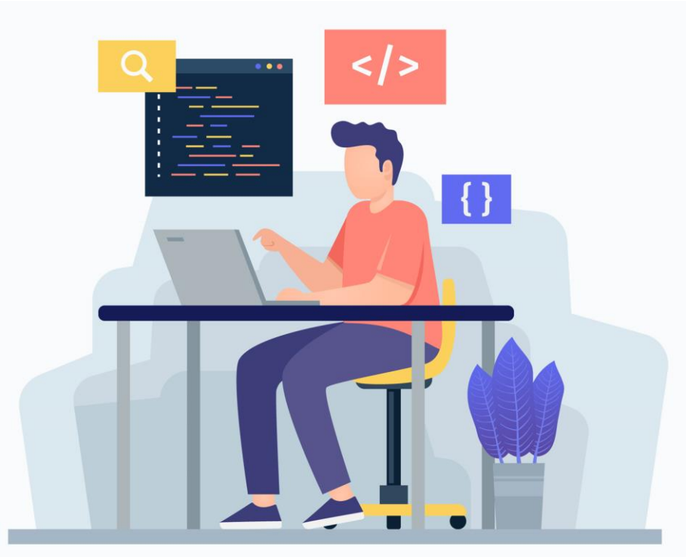
Feign REST Client

Spring Boot application uses the RestTemplate to call the user service without the Feign.

spring-cloud-starter-openfeign

Feign REST Client

Feign helps to avoid the repetition of coding.



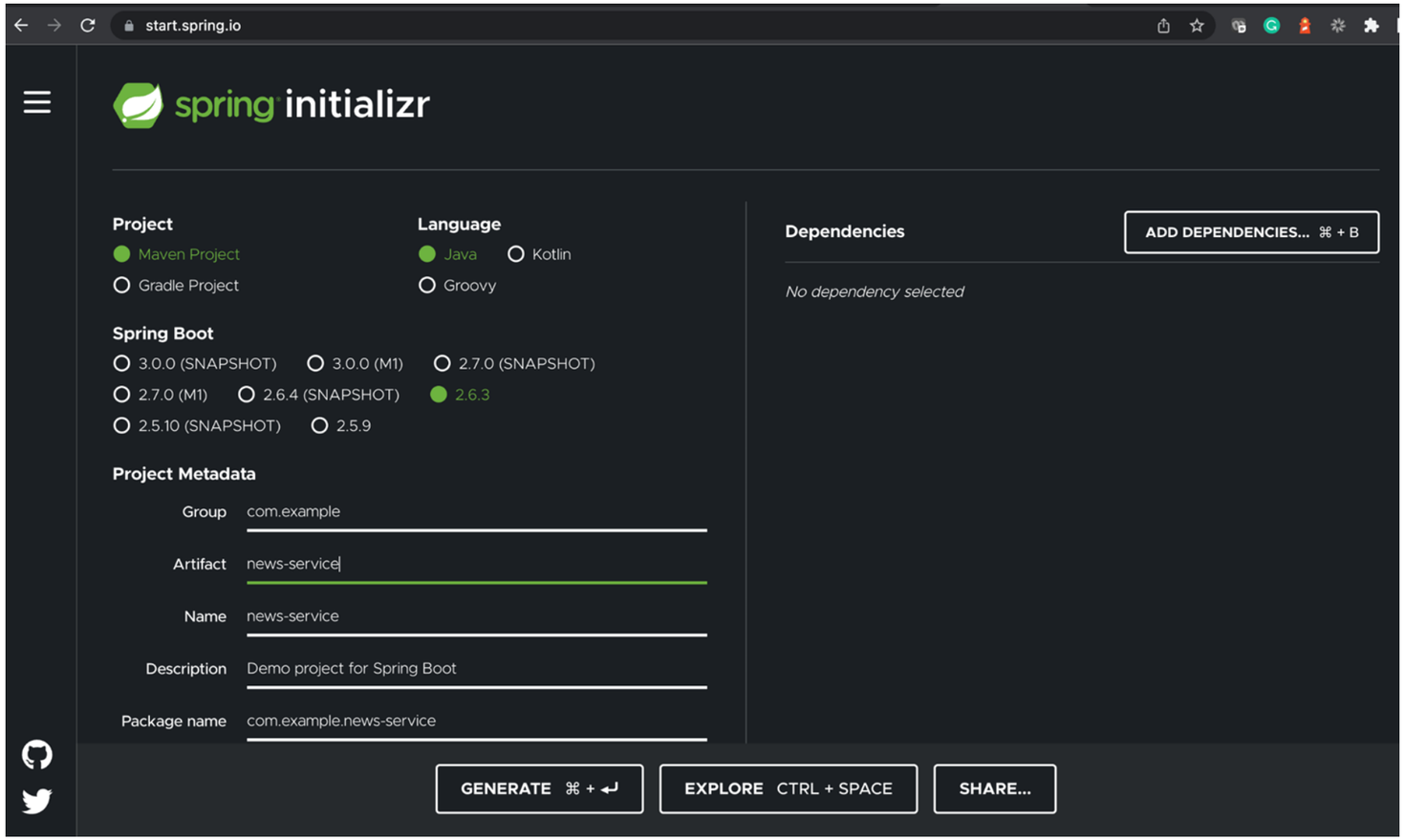
Note

Feign integrates with the ribbon, which is a client-side load-balancing framework.

Feign REST Client

Feign inherits from Netflix.

Step 1: Create the news-service project using Spring Initializr



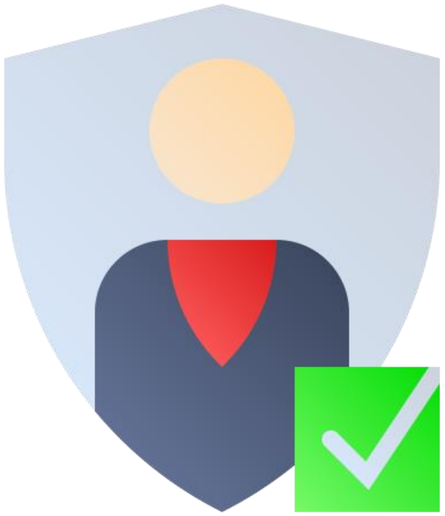
Feign REST Client

Step 2: Open the pom.xml and add the Feign dependency:

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-feign</artifactId>
<version>........</version>
</dependency>
```

Feign REST Client

Step 3: Once the dependency is added, enable Feign to scan the clients by adding the annotation



```
@EnableFeignClients
```

Feign REST Client

Step 4: Define the attribute in the @EnableFeignClients annotation

Example:

```
@SpringBootApplication
@EnableFeignClients("com.example.news-service")
public class NewsServiceApplication
{
public static void main (String[] args)
{
SpringApplication.run(NewsServiceApplication.class,args);
}
}
```

Feign REST Client

Step 5: Create the Feign proxy to talk to the external microservices

NewsServiceProxy

Feign REST Client

Step 6: Add the annotation @FeignClient and pass the attributes, the name, and the URL

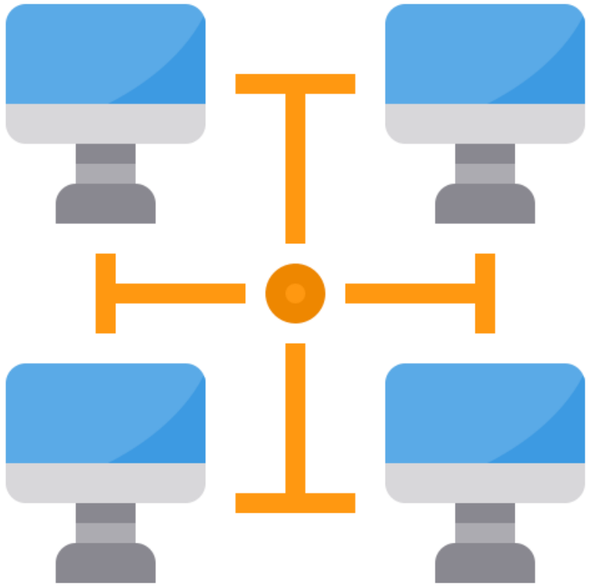
In the name attribute, write the name of the service that is required:

```
@FeignClient(name="news-service",
url="localhost:8000"”)
```

Eureka Naming Server

Eureka Naming Server

The naming server is the computer application that implements the network service to respond to queries.



Eureka Naming Server

Eureka naming server is a REST-based server used in the AWS Cloud services for load balancing.

Eureka Naming Server

The following are the characteristics of Eureka naming server:

Comes with a Java-based client component, the eureka client

Holds the information about all the client service applications

Gets microservice register itself with the Eureka naming server

Comes with the bundle of the Spring Cloud and runs on the default port 4400

Gets the naming server to register the client services with their port numbers and IP addresses

Eureka Naming Server

Need for a naming server:

Load balancing for the middle tier is not available due to the server's inherent instability.

Eureka naming server fills the gap between the client and the middle-tier load balancer.

Eureka Naming Server

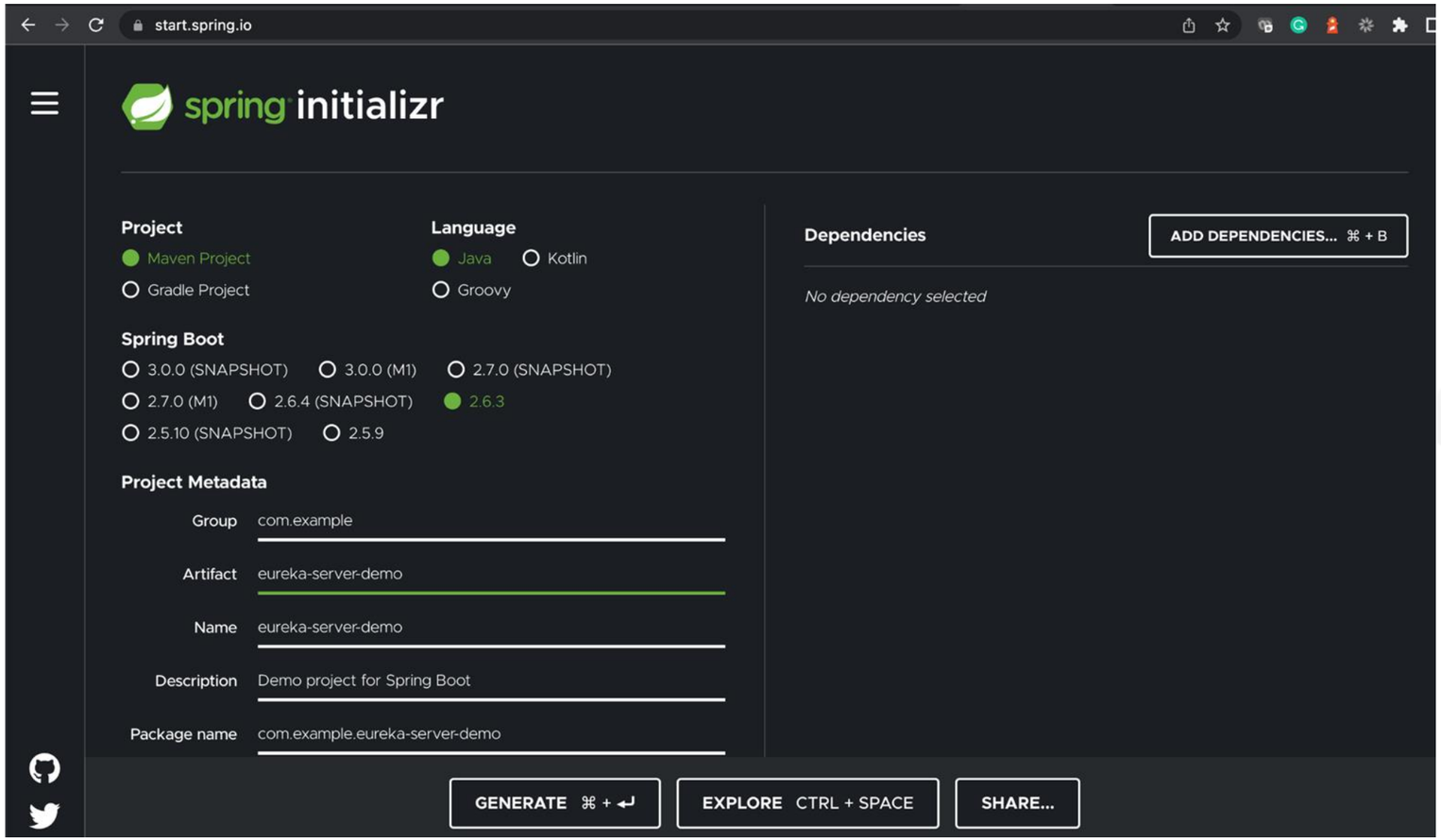
Creating the component of the Eureka Naming Server:

01 Open the Spring initializer https://start.spring.io

02 Add the group name

Eureka Naming Server

Step 3: Provide the artifact ID



Eureka Naming Server

Step 4: Add these dependencies: Eureka server, Config Client, actuator, and the development tools

Eureka Naming Server

Step 5: Click on the **GENERATE** button. It downloads a zip file.

Eureka Naming Server

Step 6: Extract the zip file, paste the folder in the Spring Tool Suite (STS) workspace, and import it

File -> Import -> Existing maven projects -> Next -> Eureka-server-demo project-> Finish

Eureka Naming Server

Step 7: Open the **EurekaServerDemoApplication.java** file and enable the eureka server using an annotation **@EnableEurekaServer**.

Eureka Naming Server

Example:

```
EurekaServerDemoApplication.java
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerDemoApplication
{
public static void main(String[] args){
SpringApplication.run(EurekaServerDemoApplication.class,
args);
}
}
```

Eureka Naming Server

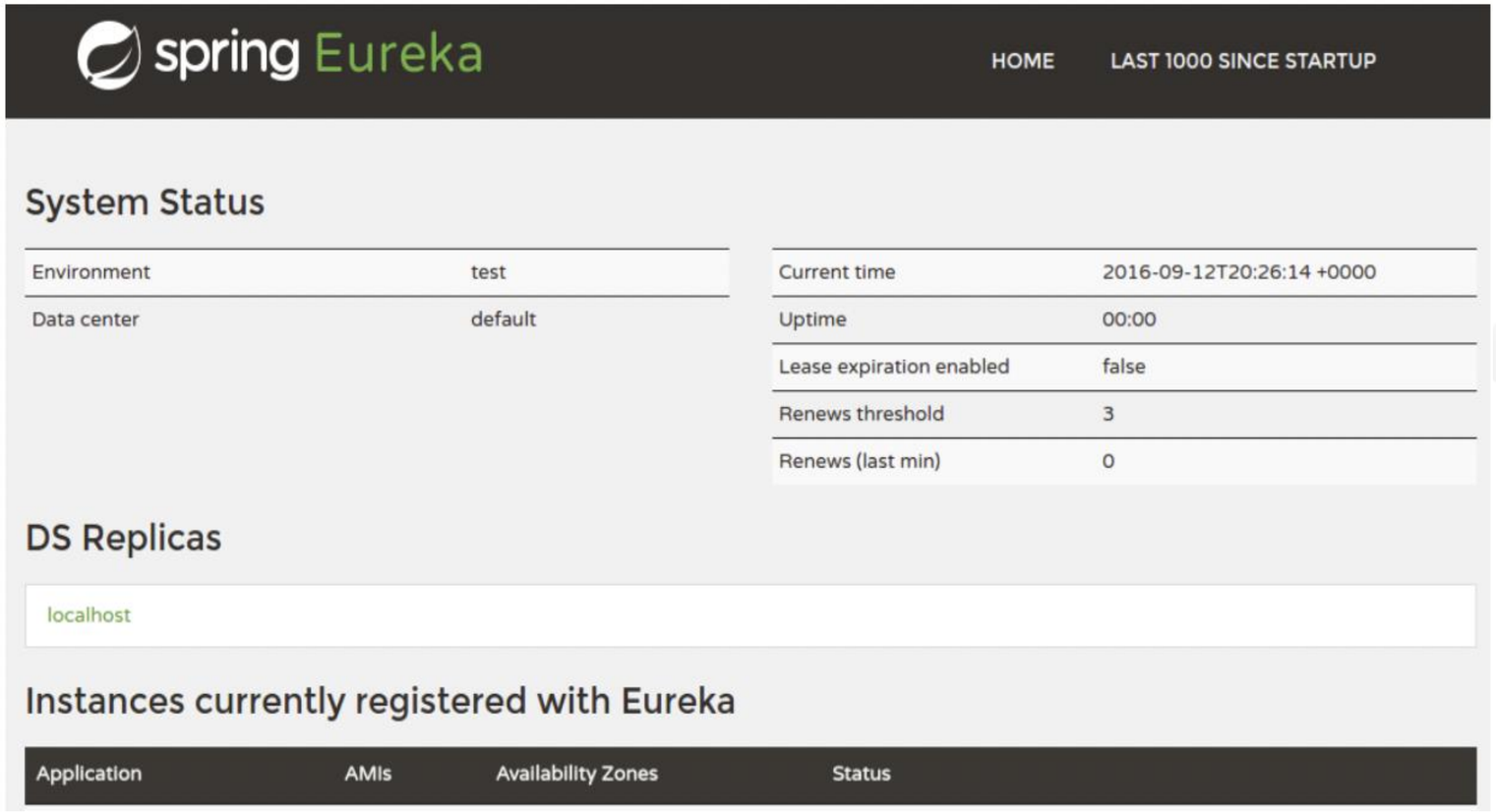
Step 8: Open the **application.properties** file and configure the application name, port, and the Eureka server, using this code:

```
spring.application.name = eureka-server-demo
server.port=4400
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Eureka Naming Server

Step 9: Run the **EurekaServerDemoApplication.java** file as the Java application

Step 10: Open the browser and type the URL http://localhost:4400



Setting up Eureka Server

Problem Statement:

You have been asked to set up a Eureka Server, which acts as a service registry and enables service discovery in a Spring Cloud application.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Creating a new Spring Starter project
2. Configuring the Eureka Server properties
3. Running the Eureka Server

Registering Microservice with Eureka Server

Problem Statement:

You have been asked to demonstrate how to register a microservice with Eureka Server using Spring Boot.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Configuring the Eureka server
2. Creating a Spring Starter project
3. Creating an AppController.java file
4. Testing the application on Eureka Server

Key Takeaways

- Spring Cloud is a Spring module that gives the Spring framework the RAD capability.
- Service Discovery is the process of connecting an application and a microservice in a distributed context.
- Spring Cloud's tracing feature enables obtaining application data with a single request.
- Distributed systems provide access to a central place to manage the external properties of the applications.
- Feign is the declarative web service (HTTP client) developed by Netflix.

TECHNOLOGY

Thank You

simplilearn