Deloitte.

Spring Cloud

Deloitte Technology Academy (DTA)

< / >

Agenda

Topics	Descriptions	Duration
Spring Cloud and Service Discovery	Features, Main Components, Spring Cloud Config, Service Discovery, and Server-side Service Discovery vs. Client-side Service Discovery	X hours XX minutes
API Guidelines	Best Practices (Verbs Naming Convention, Versioning), Exposing End Points, and Application Programming Interface (API) Documentation	X hours XX minutes
API Gateway	API Gateway	X hours XX minutes
Config Server	Usage and Implementation	X hours XX minutes
API Security	Different Ways of Securing the APIs	X hours XX minutes
Monitoring and Tracing	IAPI Tracing Using Zipkin, and Monitoring with Spring Boot Admin	X hours XX minutes
Microservice Ecosystem and Tools—Eureka	Eureka Server and Eureka Clients	X hours XX minutes
Zuul	Ribbon, Registering Eureka Clients, Zuul API Gateway, Zuul API Gateway–Filters, and Zuul API Gateway Server	X hours XX minutes
Postman—Usage, and Installation	Introduction to Postman, Installation Steps, and Postman in Action	X hours XX minutes

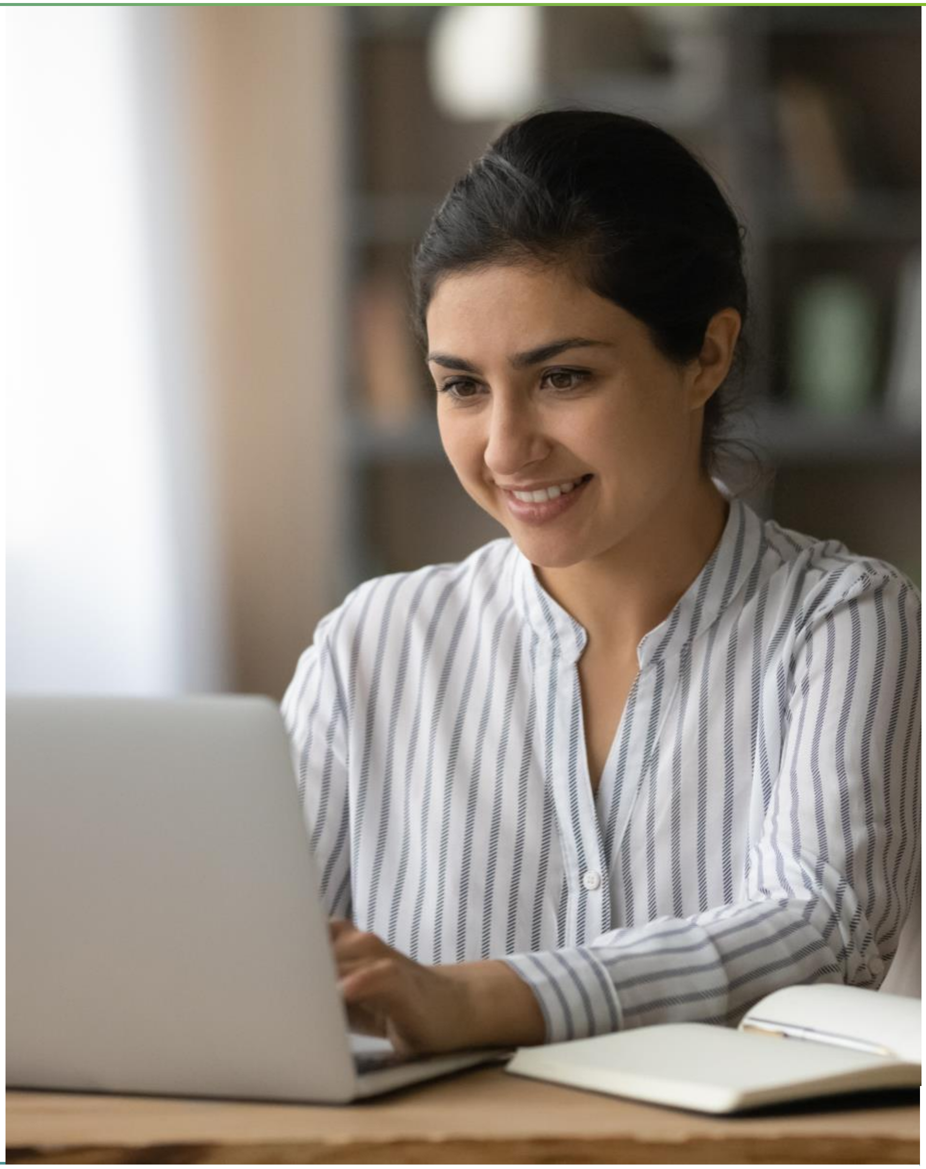
Learning Objectives

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

- Explain what is Spring Cloud, its features, and components
- Implement Service Discovery Eureka
- Explain Application Programming Interface (API) Gateway and its core features
- Describe how to externalize configuration using config server
- Explain API Tracing using Zipkin and Monitoring with Spring Boot Admin
- Identify API documentation, best practices of exposing Representational State Transfer (REST) end points
- Explain Microservices ecosystem and its tools

</>

Spring Cloud and Service Discovery



Spring Cloud—Features

- Distributed/versioned configuration
- Service registration and discovery
- Routing
- Service-to-service calls
- Load balancing
- Circuit Breakers
- Global locks
- Leadership election and cluster state
- Distributed messaging

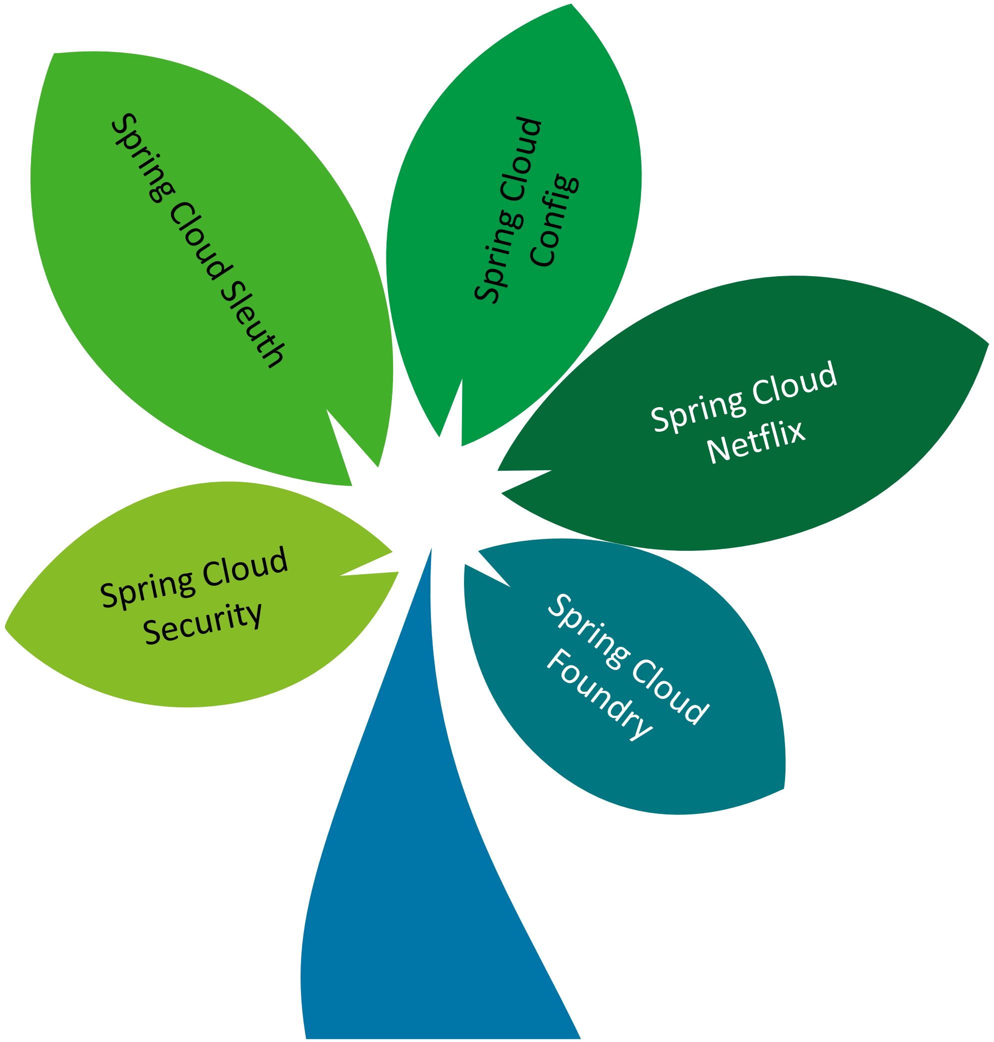
Spring Cloud—Features(Cont.)

- Implementing a set of common patterns required by distributed systems the Spring Cloud project is an easy-to-use umbrella project from the Spring team, located in the Java Spring libraries.

- Spring cloud is not a cloud solution, despite its name, but it does provide an essential number of capabilities for developing applications targeted at cloud deployments.

- Out of the box, Spring Cloud offers developers building business capabilities through Spring Boot, and leveraging the fault-tolerant, distributed, and self-healing capabilites that should be the primary focus.

Spring Cloud—Main Components



Spring Cloud Security
Provides support for load-balanced Open Authorization2 (OAuth2) rest-client, and authentication header relays in a Zuul proxy

Spring Cloud Sleuth
Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace, and log-based (Example: ELK) tracing

Spring Cloud Config

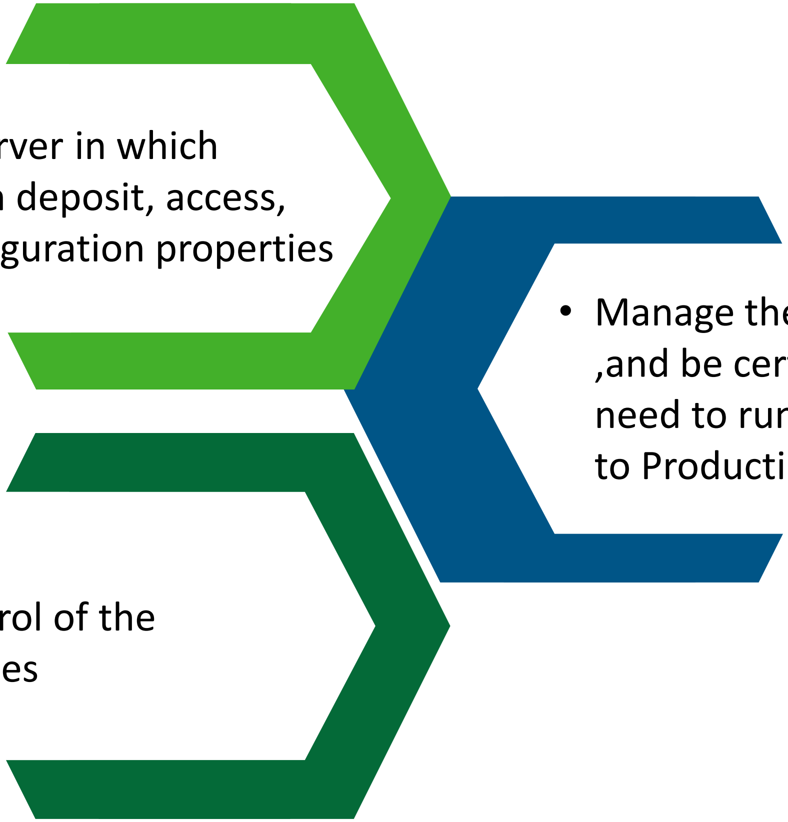
- Centralized external configuration management backed by Git repository
- Configuration resources map directly to Spring Environment but could be used by non-Spring applications, if desired

Spring Cloud Netflix
Integration with various Netflix Operations Support System (OSS) components (Eureka, Hystrix, Zuul, and Archaius, etc.)

Spring Cloud Foundry

- Integrates your application with Pivotal Cloud Foundry (PCF)
- Provides a service discovery implementation, and also makes it easy to implement Single Sign-On (SSO), and OAuth2 protected resources.

Spring Cloud—Spring Cloud Config



- Externalized configuration server in which applications, and services can deposit, access, and manage all runtime configuration properties

- Manage the configuration between different environments ,and be certain that applications have everything they need to run when they migrate from Dev to Test, and finally to Production

- Supports version control of the configuration properties

Spring Cloud—Spring Cloud Config (Cont.)

Setting up Config Server

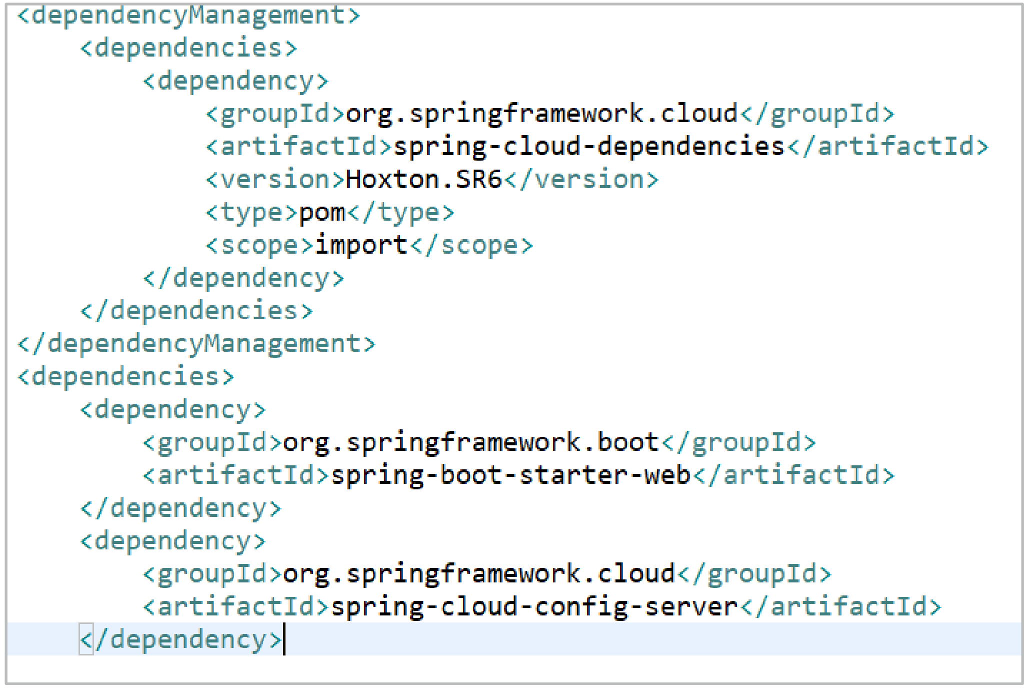
- As a dependency to the project add config server
- Add the `@EnableConfigServer and @SpringBootApplication` annotations
- In your resource folder configure your application.yml file. Your cloud-config server’s access to Github will be set up in this file.

```
cloud:
   config:
        server:
               git:
                 uri: <Credentials URI>
                username:
                Password:
```

- Create a bootstrap.yml file in your src/main/resources folder that will live alongside your application.yml, and it will contain information pointing to where the config server’s location is being hosted.

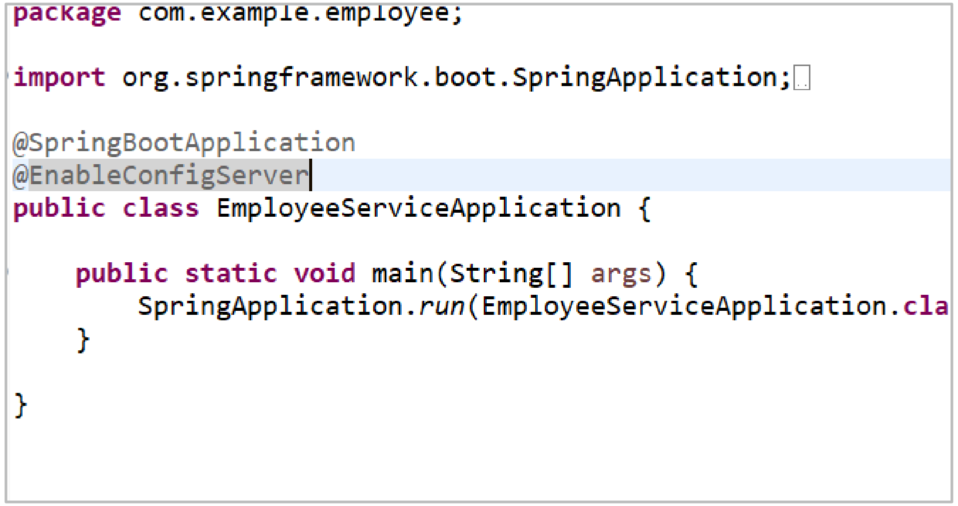
Spring Cloud—Illustration

Add dependency in pom.xml

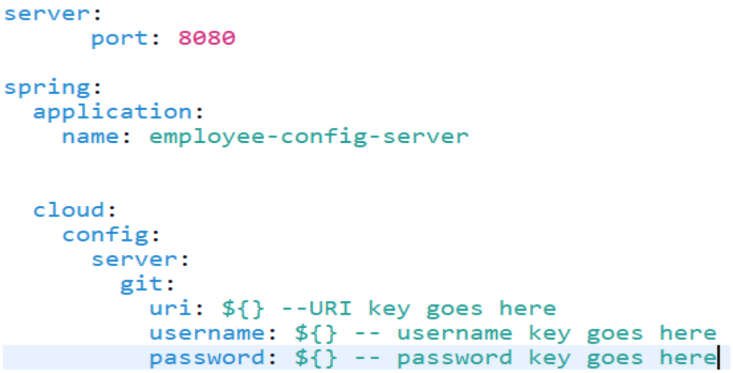


```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Hoxton.SR6</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
```

Add @EnableConfigServer annotation and application.yml



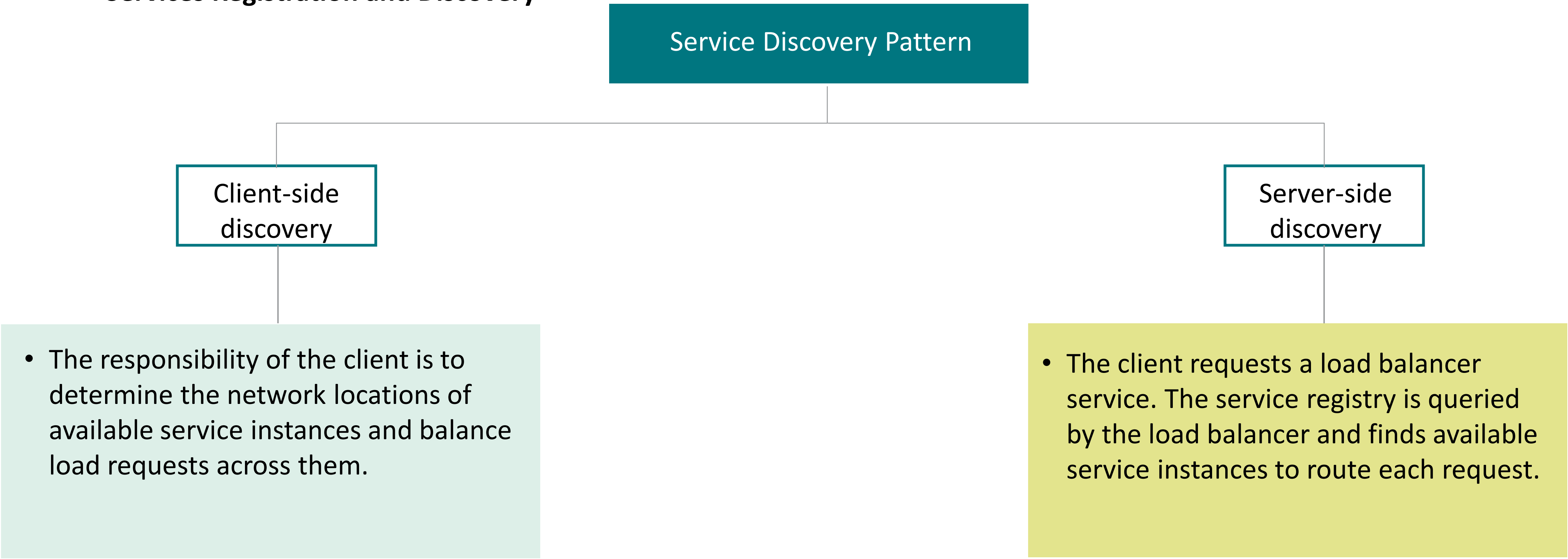
```java
package com.example.employee;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableConfigServer
public class EmployeeServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeServiceApplication.cla
    }

}
```



```yaml
server:
      port: 8080

spring:
  application:
    name: employee-config-server


  cloud:
    config:
      server:
        git:
          uri: ${} --URI key goes here
          username: ${} -- username key goes here
          password: ${} -- password key goes here
```

Spring Cloud—Service Discovery

Services Registration and Discovery



Service Discovery Pattern

Client-side discovery

- The responsibility of the client is to determine the network locations of available service instances and balance load requests across them.

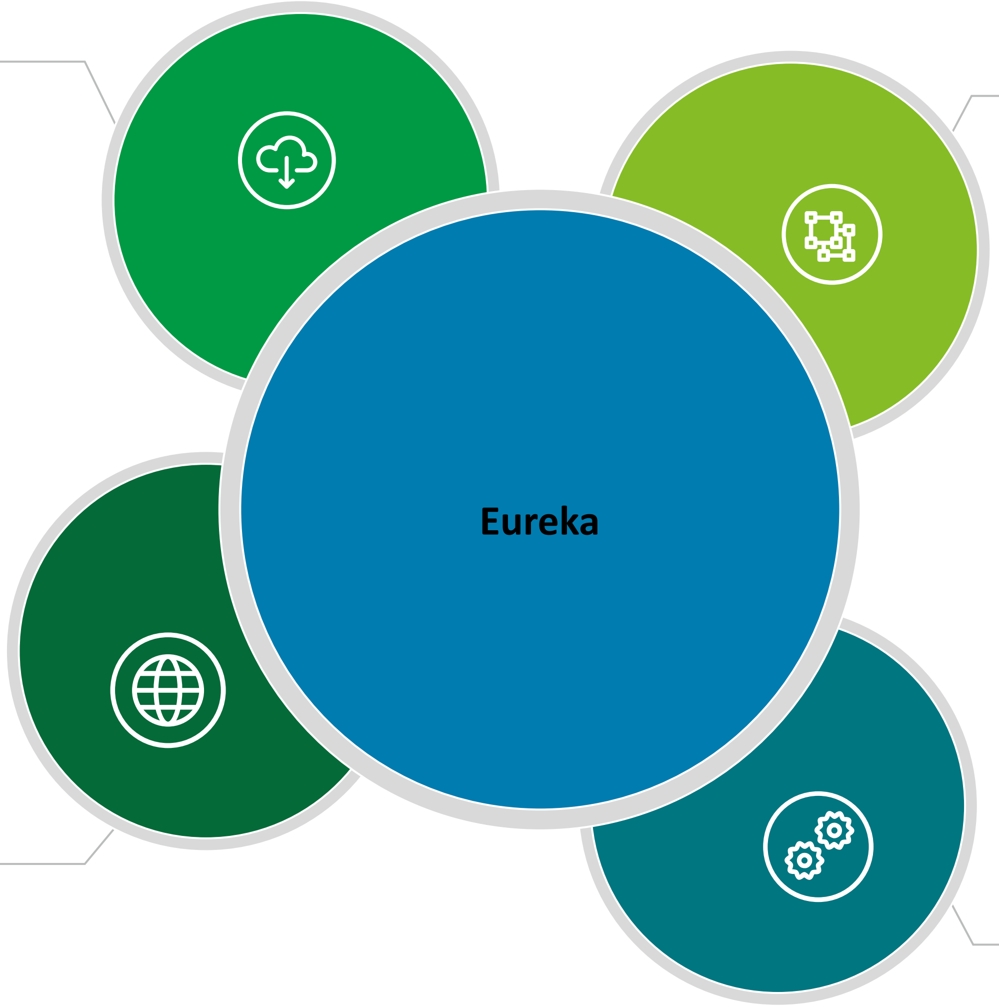
Server-side discovery

- The client requests a load balancer service. The service registry is queried by the load balancer and finds available service instances to route each request.

Spring Cloud—Service Discovery (Cont.)

- All client-service applications' information is held on the Eureka Server.

- The Eureka server will register Microservice allowing the Eureka server to know all the client IP addresses and applications running on each port.



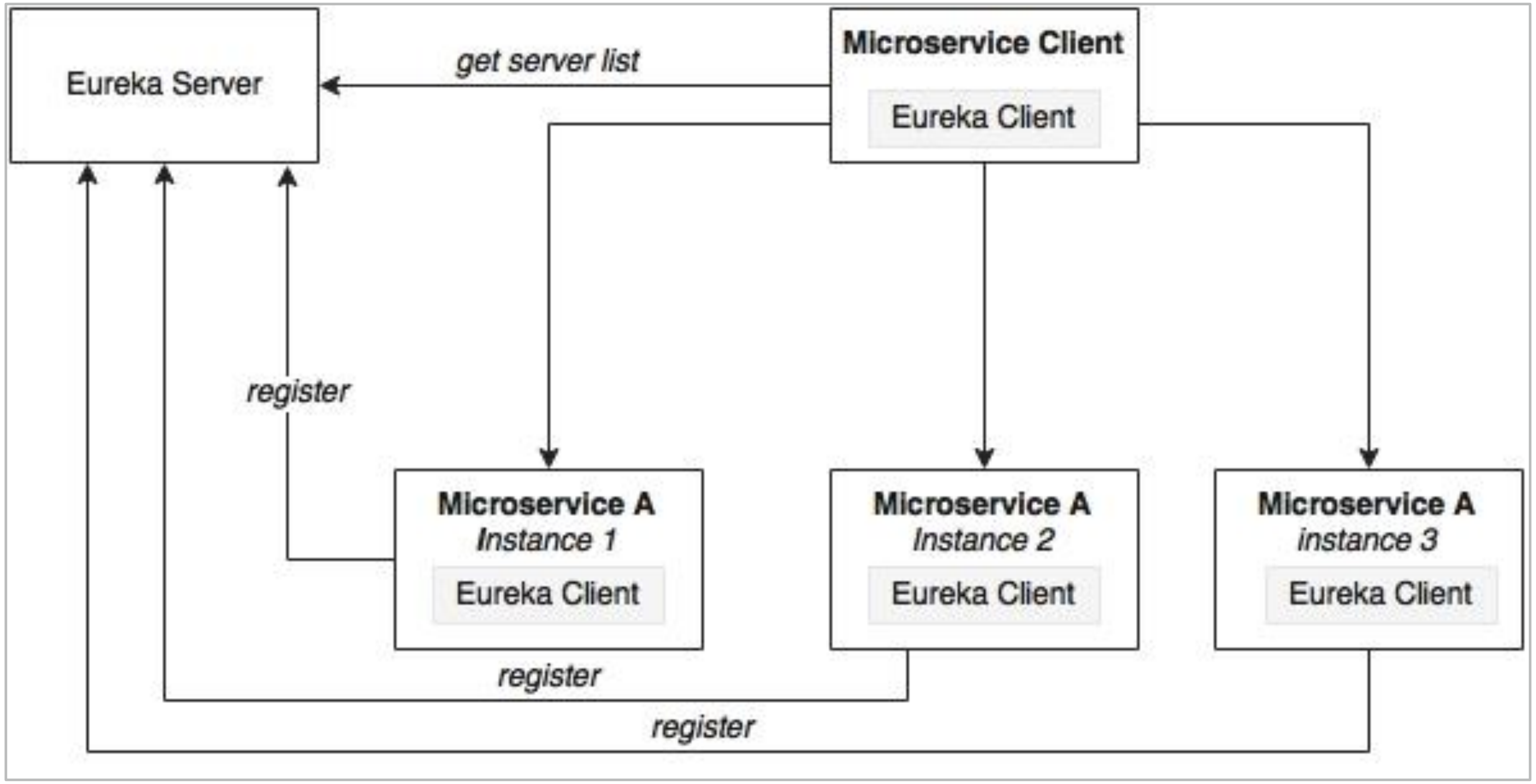
- Discovery Server is referenced as another name for Eureka Server.

- Eureka uses Ribbon for load balancing internally, and client-side service discovery pattern

Spring Cloud—Service Discovery (Cont.)

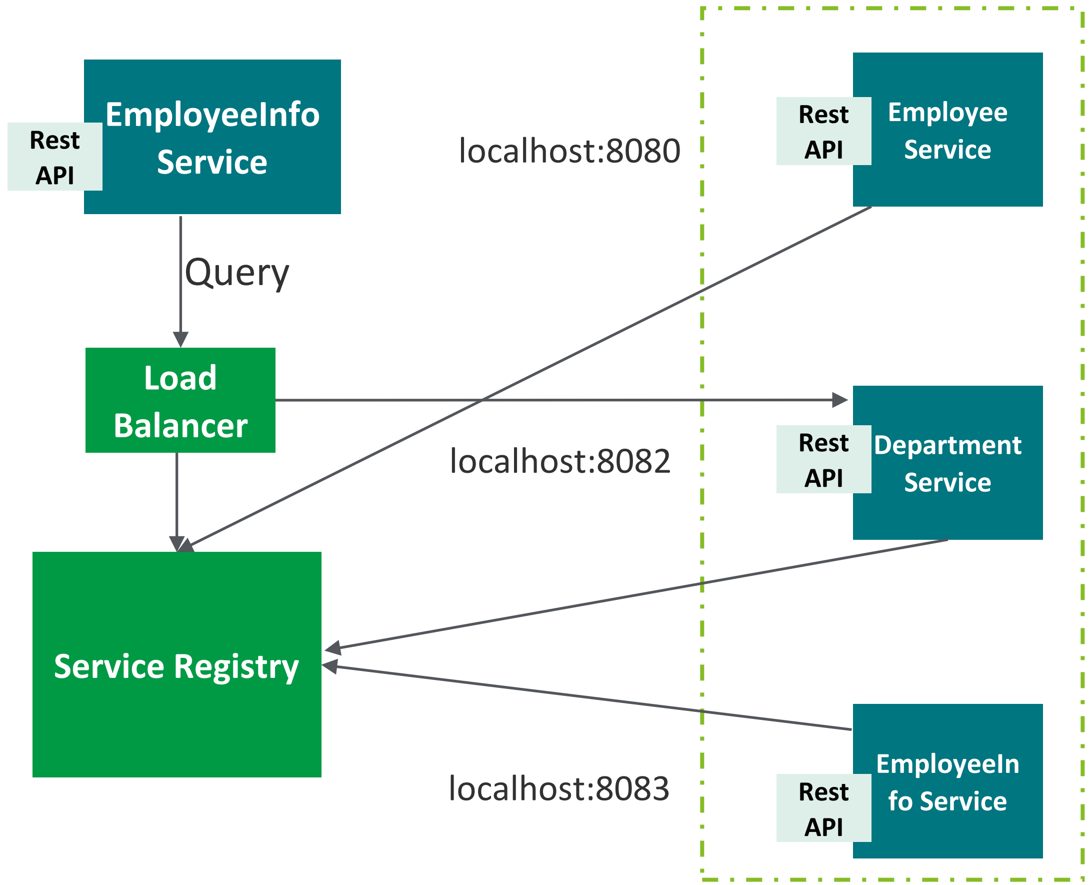
Illustration:

```
@EnableEurekaServer
@EnableDiscoveryClient
```

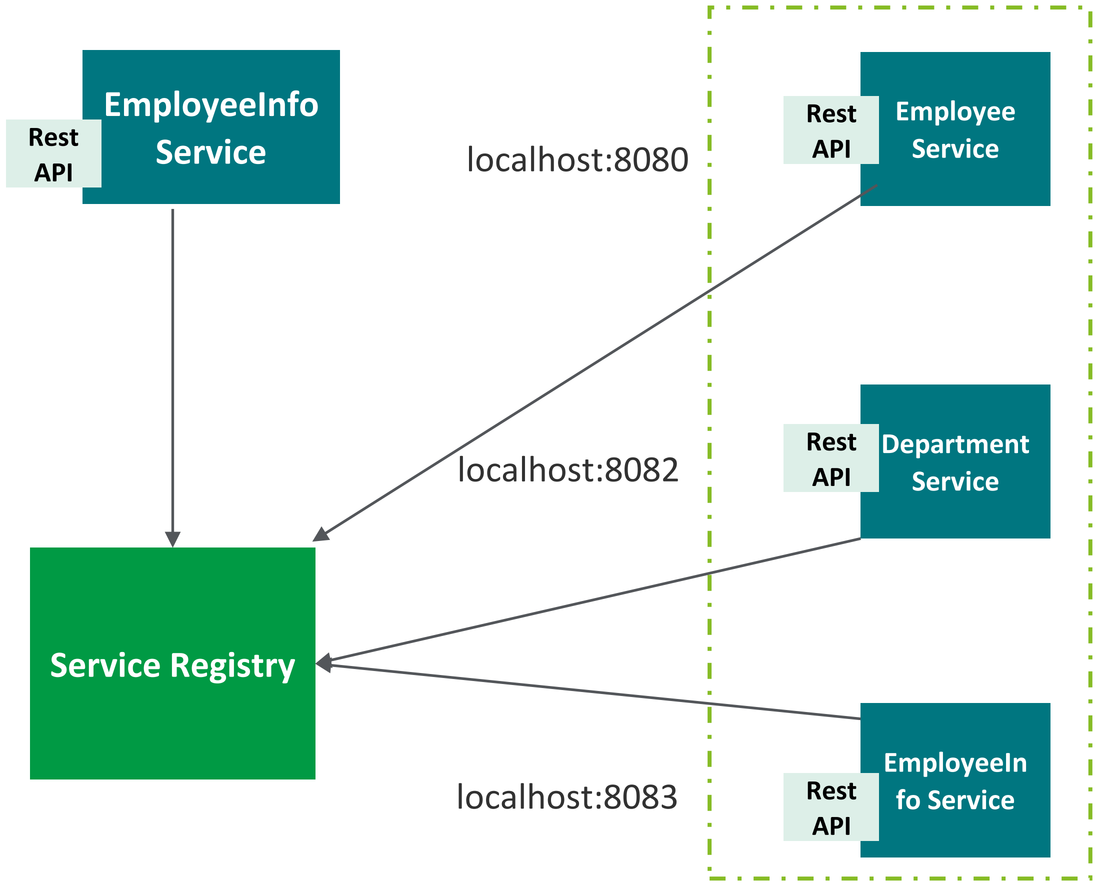


Spring Cloud—Server-Side Service Discovery vs. Client-Side Service Discovery

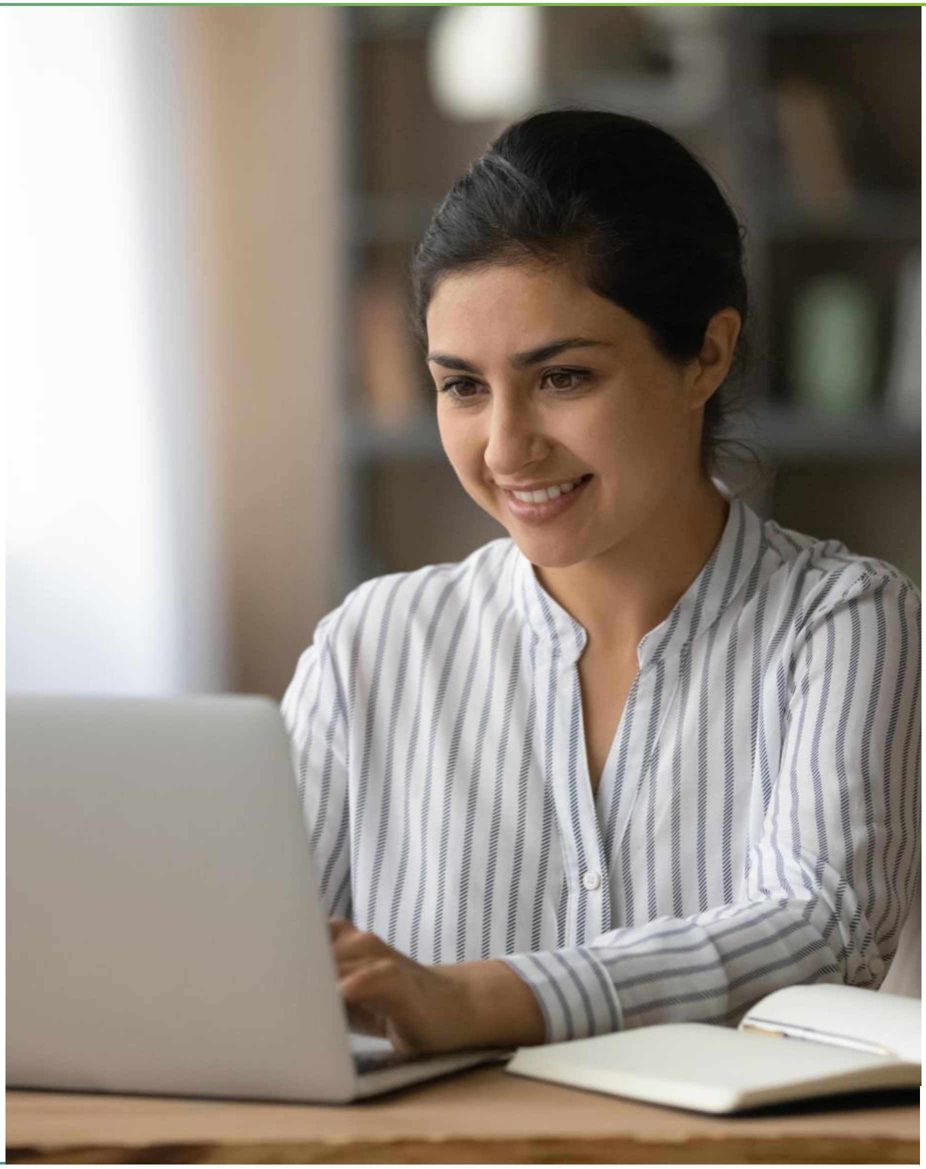
Server-Side Service Discovery



Client-Side Service Discovery



API Guidelines



Best Practices (Verbs Naming Convention, Versioning)

1. Design REST APIs optimally. For example, you should group all related APIs in single controller oriented around a use-case
2. Use nouns instead of verbs in the endpoint paths, which represent entities/resources to fetch or manipulate and use consistently plural nouns such as /orders/{id}/products over /order/{id}/product
3. The operation must be represented by the HTTP request, for example GET retrieves resources. POST creates a new data record
4. Controllers are not supposed to perform any business logic apart from routing and delegating the action to the proper services
5. In a nutshell, it is important to design REST APIs properly while at least considering the performance and ease of use for the API clients.

Exposing End Points

- Exposing directly the JPA/database entities representation in the REST endpoints to send/receive data from the client is not a good practice.
  - Because it creates a high coupling between the persistence models and the API models.
  - Also, it exposes the implementation details of the application
- The best approach of exposing endpoints is to use a separate data transfer object (DTO) that represents the API resource class which is mapped from a database entity or multiple entities.

API Documentation

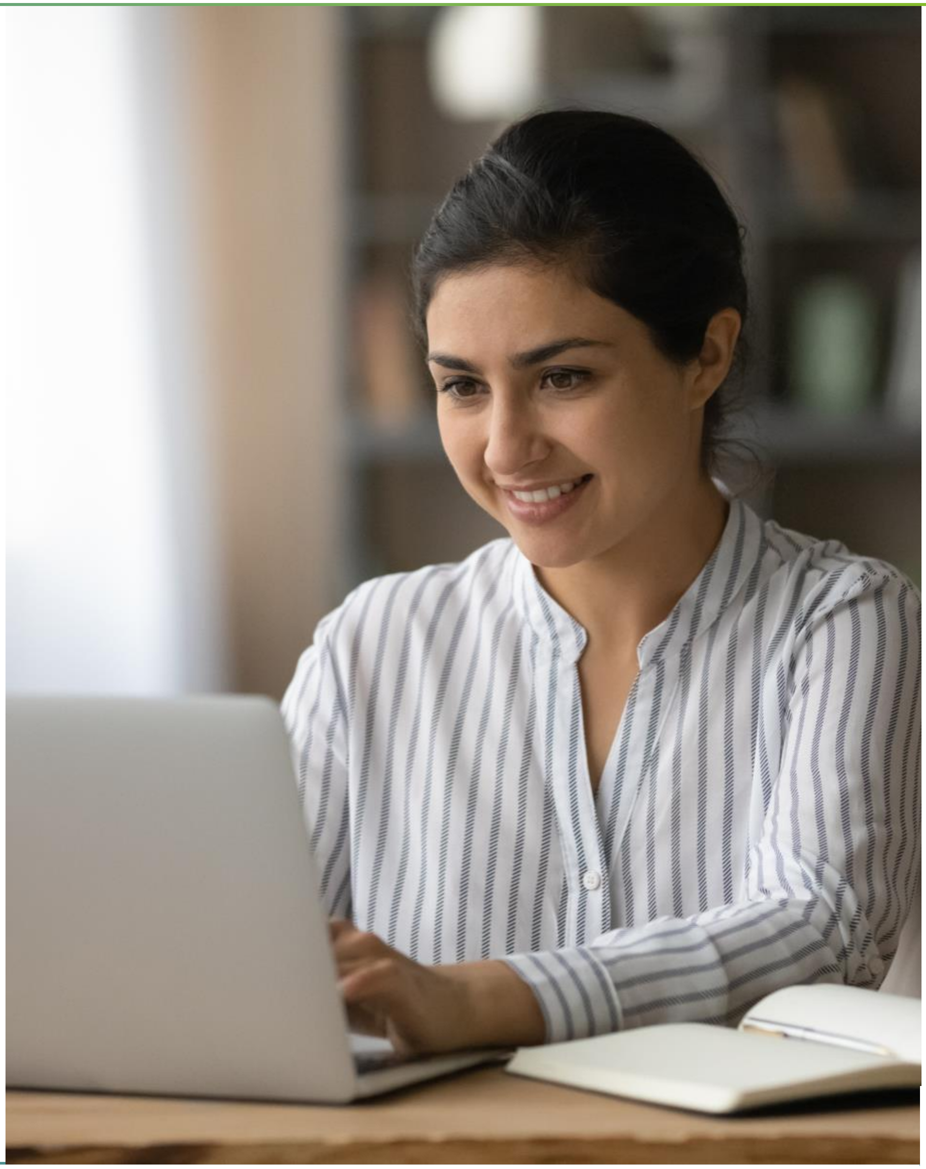
< / >

Use OPEN API 3.0 specification for API documentation

	Swagger (Open API)
Approach	Bottom-up Specification
Adaptability	Adapted by most of the API vendors
Document format	JavaScript Object Notation (JSON) and yet another markup language (YAML)
Editors	Swagger Tool
UI Experience	Good
Reusability	Not supported within same specification
Structured	Doesn’t support folder structuring
Advantages	A large open-source community using the following: • High Adoption rate • Large documentation available • Strong framework support

< / >

API Gateway



Spring Cloud—API Gateway

API Gateway

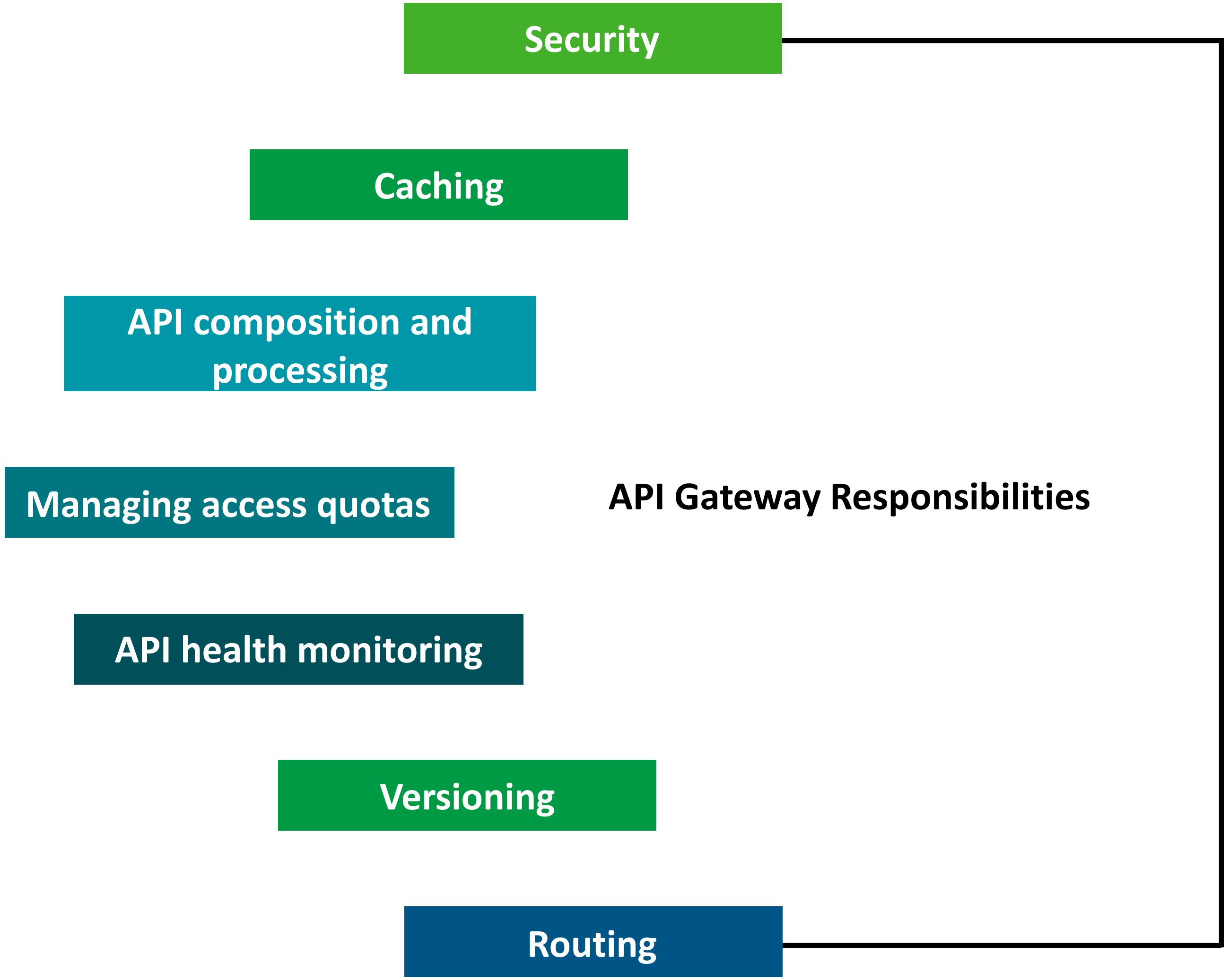
- The API Gateway is a server.

- It is a single-entry point into a system.

- API Gateway encapsulates the internal system architecture. It provides an API that is tailored for each client.

- The API Gateway handles all client requests. Then the API Gateway sends the requests to the appropriate microservice.

Spring Cloud—API Gateway (Cont.)



Security

Caching

API composition and processing

Managing access quotas

API health monitoring

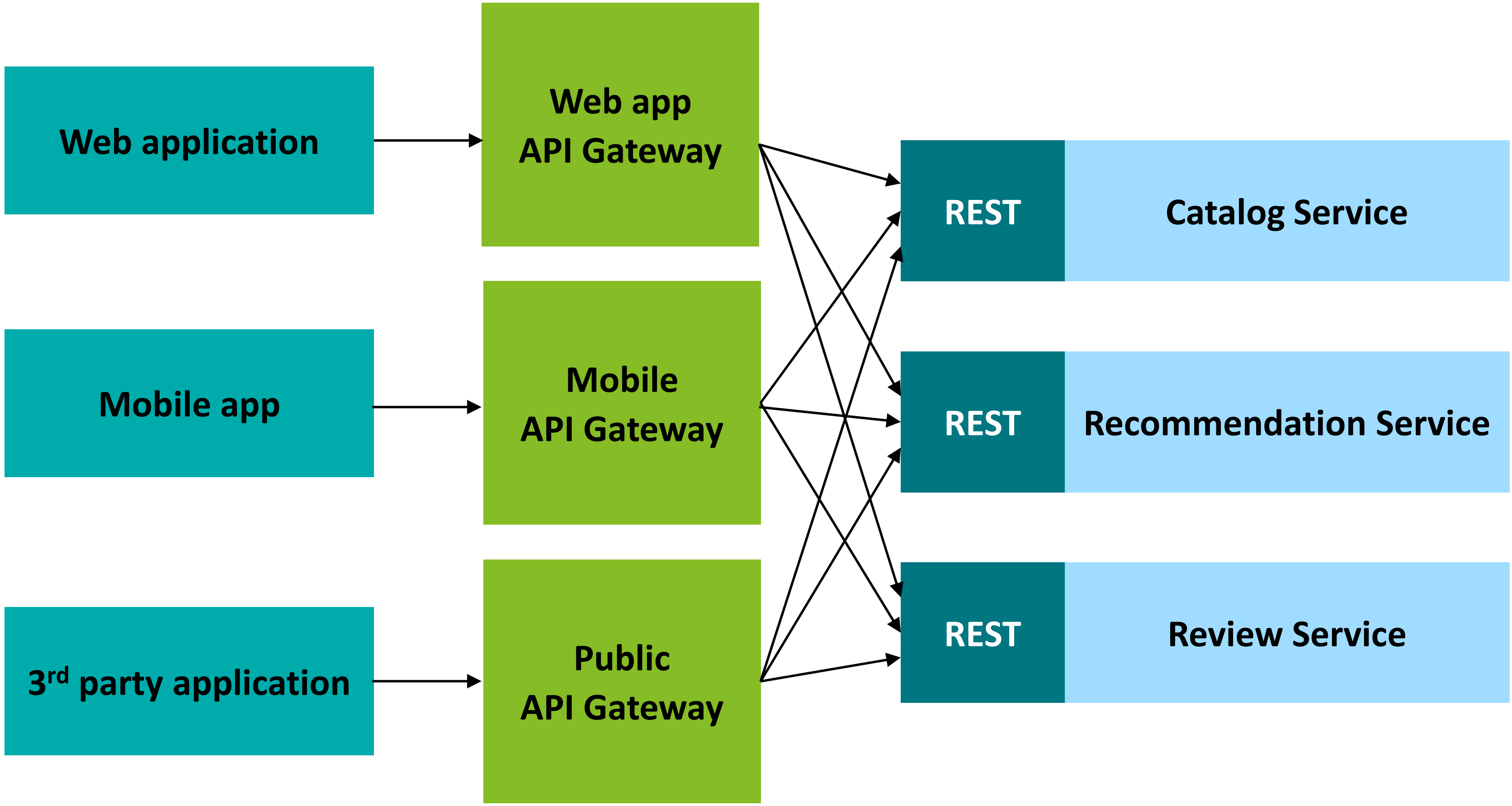
Versioning

Routing

API Gateway Responsibilities

Spring Cloud—API Gateway (Cont.)

Illustration:



Web application

Web app API Gateway

Mobile app

Mobile API Gateway

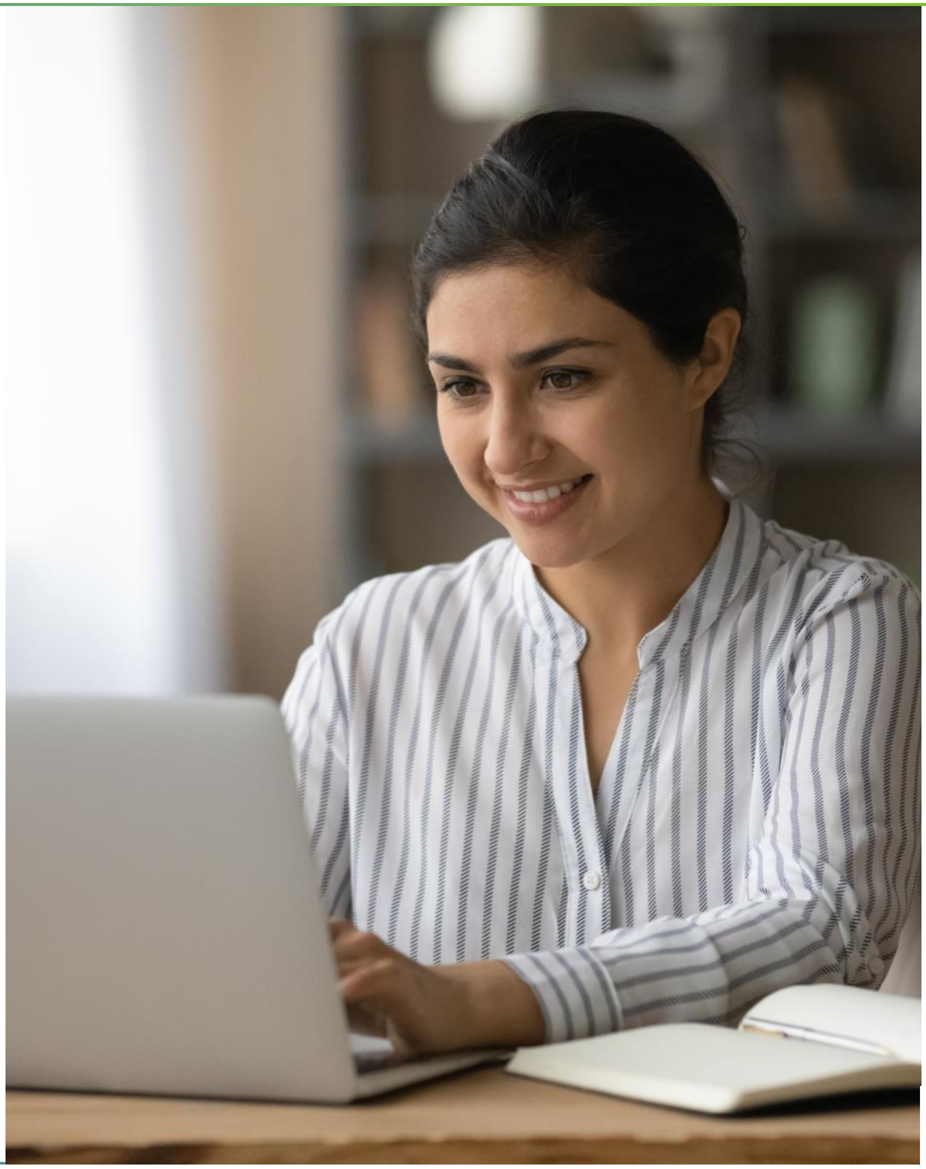
3rd party application

Public API Gateway

REST	Catalog Service
REST	Recommendation Service
REST	Review Service

</>

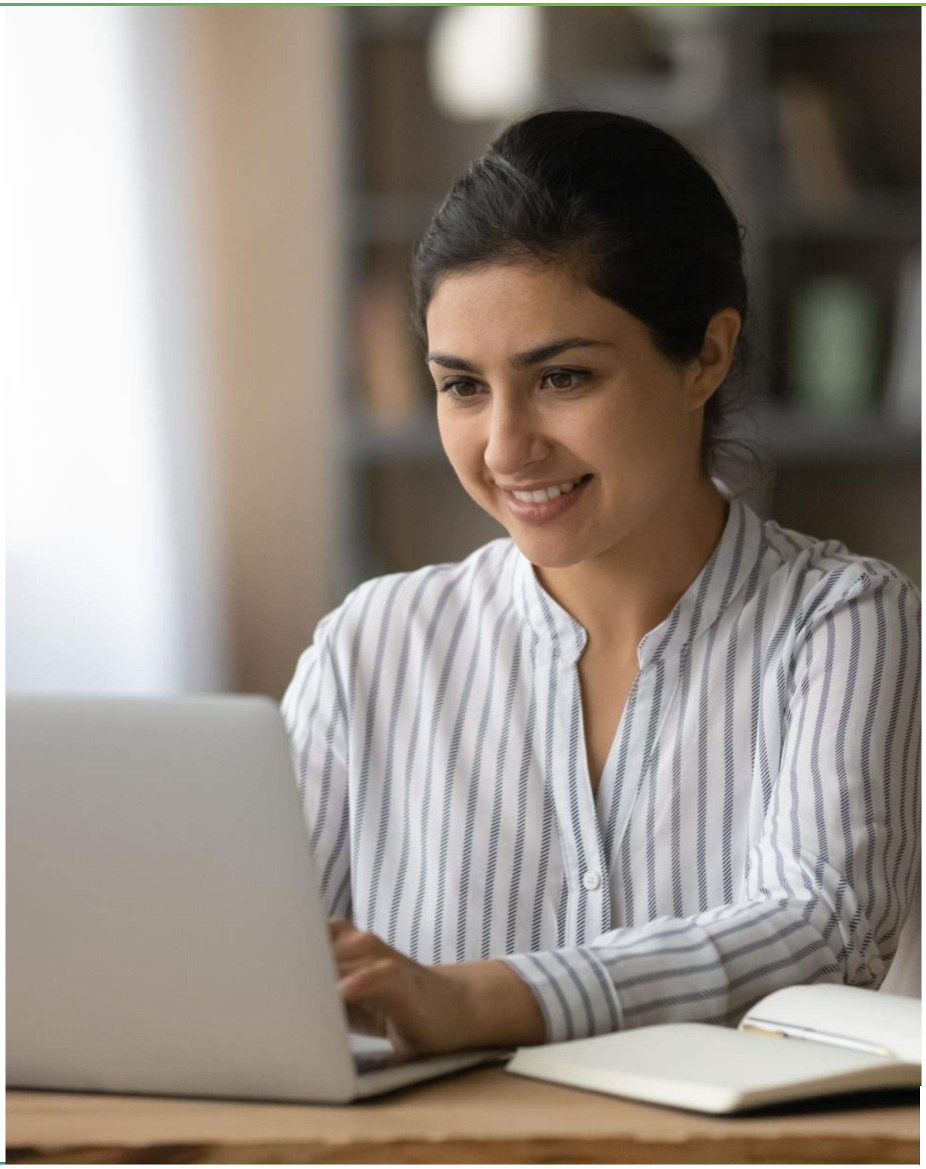
Config Server



Usage and Implementation

- Config server is an externalized configuration server in which applications and services can deposit, access, and manage all runtime configuration properties
- Also supports version control of the configuration properties
- Manage the configuration between different environments and be certain that applications have everything they need to run when they migrate from Dev to Test and finally to Production
- Add config server as a dependency to the project to set up config server

API Security

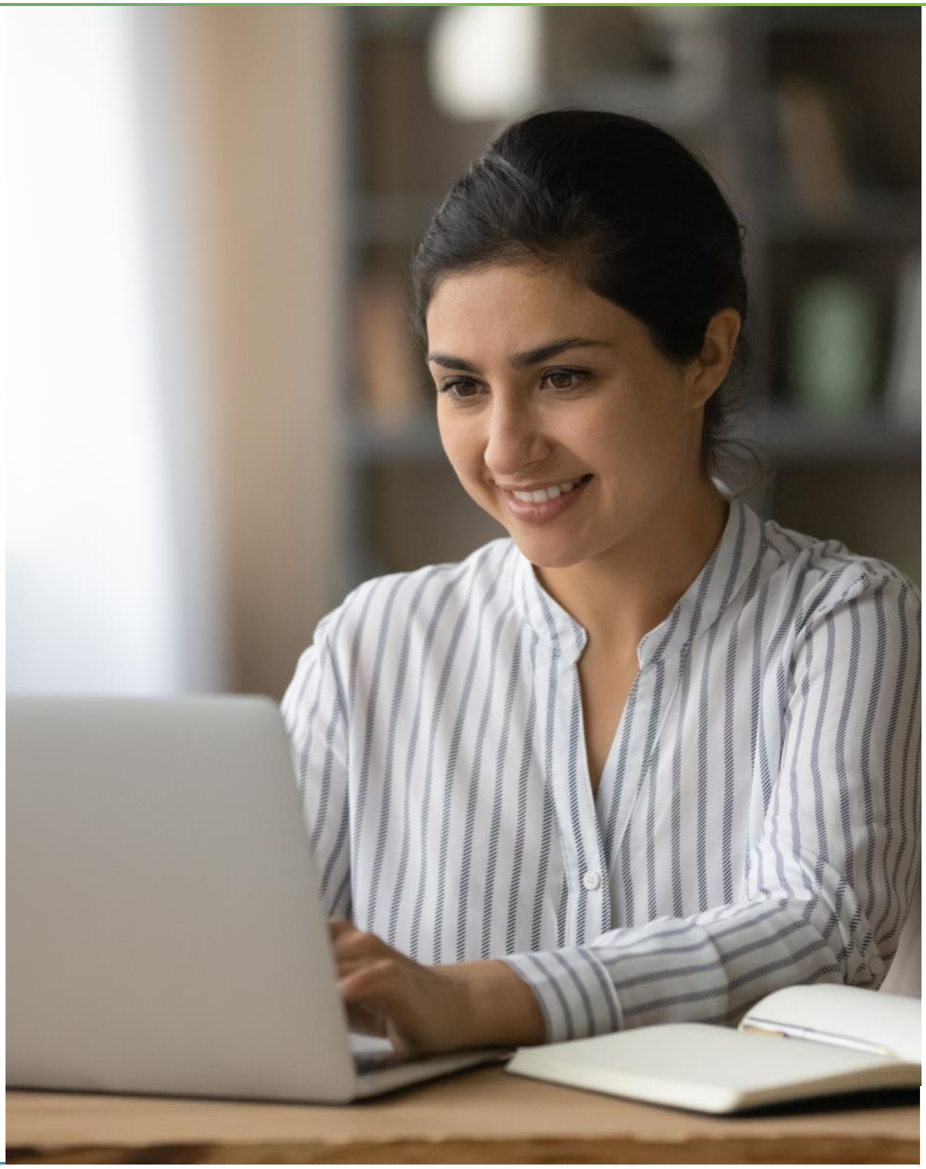


Different Ways of Securing the APIs

- Encrypt database and passwords(ie. Encrypt all data at rest)
- Use User identity and access tokens such as JWT, OAuth which can be trusted and authenticated by each service
- API gateways are the most commonly used solution and create one
- Use appropriate tools to monitor internal systems and services

</ >

Monitoring and Tracing



API Tracing Using Zipkin and Monitoring with Spring Boot Admin

Zipkin

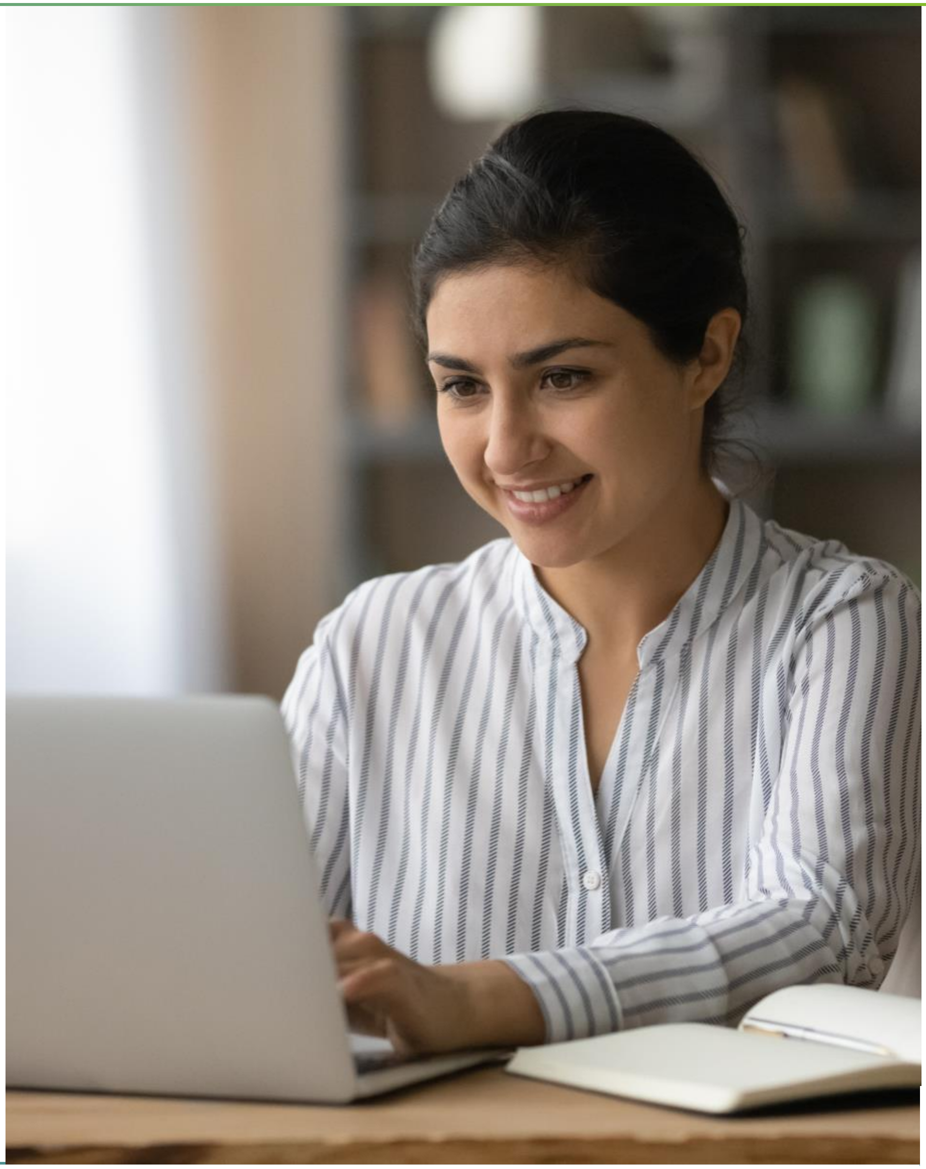
- Zipkin is an efficient tool for distributed tracing in the microservices ecosystem.
- Zipkin is an open-source project that provides mechanisms for sending, receiving, storing, and visualizing traces.
- Add Zipkin dependency, enable the zipkin server, do the required configuration and run it.

Spring Boot Admin

- Spring Boot Admin is a tool for visualizing endpoints exposed by Spring Boot Actuator with health checks and application details.
- This tool only allows monitoring and does not have such capabilities as creating new instances or restarting them.
- It has easy integration with Spring Cloud and can group all running instances of microservice by its name taken from Eureka registry.
- Add the required dependency and run the server to see the dashboard.

</>

Microservice Ecosystem and Tools—Eureka

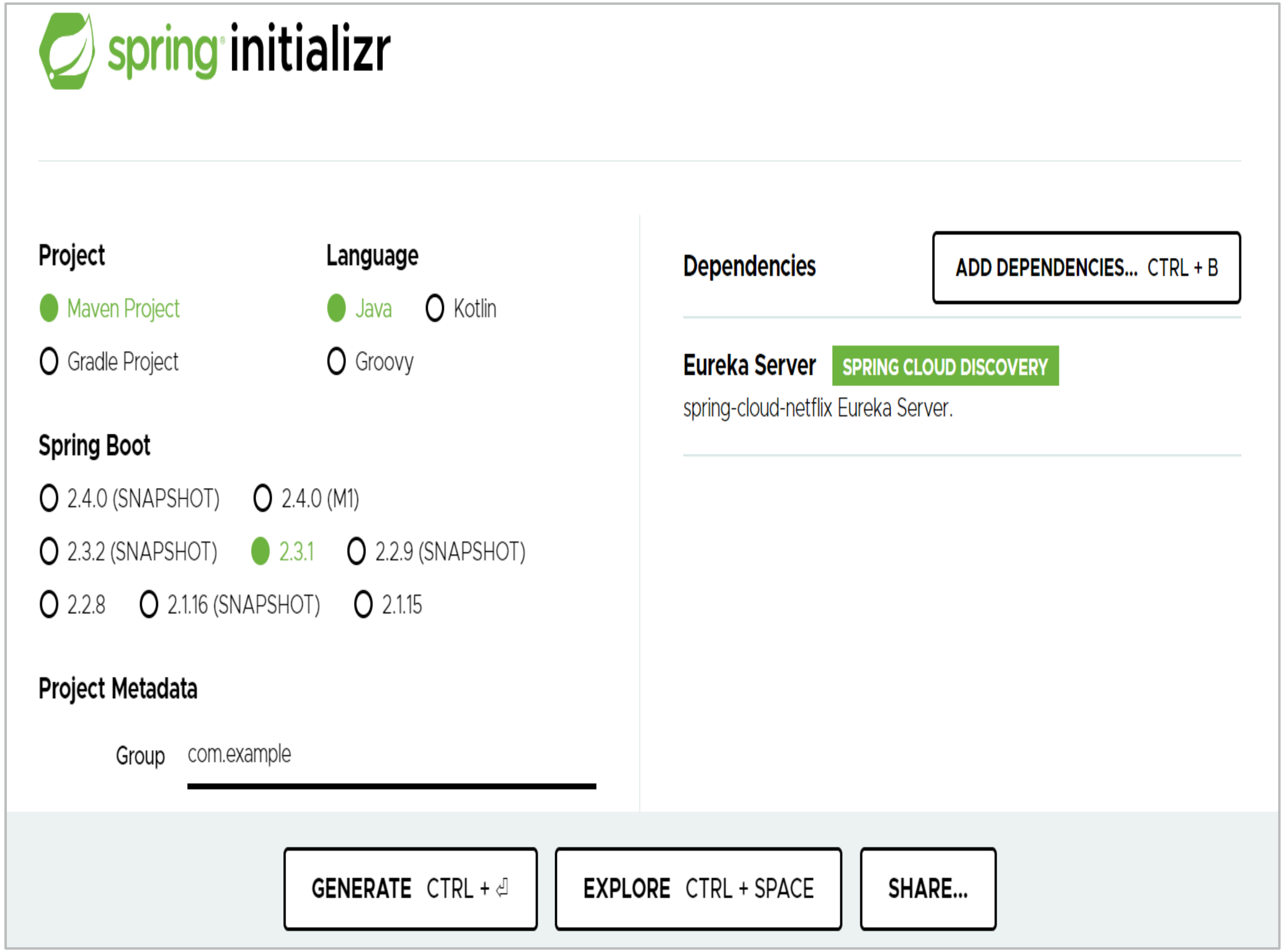


Microservices Ecosystem and Tools—Eureka Illustration

Creation of Discovery Server

- Go to start.spring.io
- Select the Spring Boot version
- Click on Add Dependencies, and search for Eureka server
- Click on GENERATE to create the Maven project
- Import the project to your workspace as Maven project
- Eureka server is nothing but a normal Spring boot application
- Change the port number to 8761 in application.properties/application.yml if applicable
- Add @EnableEurekaServer to the ---Application.java
- Start the application

Microservices Ecosystem and Tools—Eureka Illustration (Cont.)



Project

● Maven Project

○ Gradle Project

Language

● Java ○ Kotlin

○ Groovy

Spring Boot

○ 2.4.0 (SNAPSHOT) ○ 2.4.0 (M1)

○ 2.3.2 (SNAPSHOT) ● 2.3.1 ○ 2.2.9 (SNAPSHOT)

○ 2.2.8 ○ 2.1.16 (SNAPSHOT) ○ 2.1.15

Project Metadata

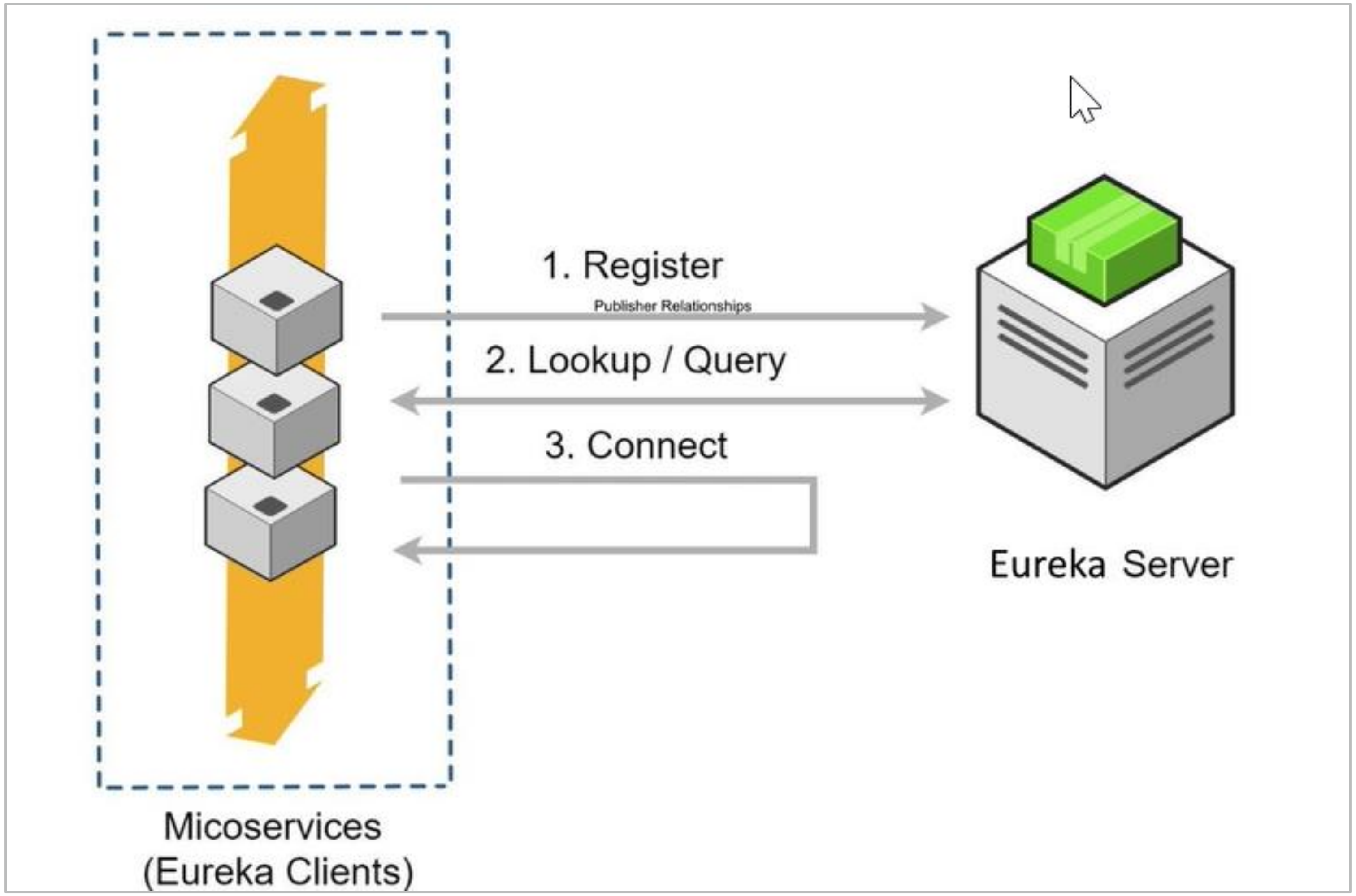
Group com.example

Dependencies

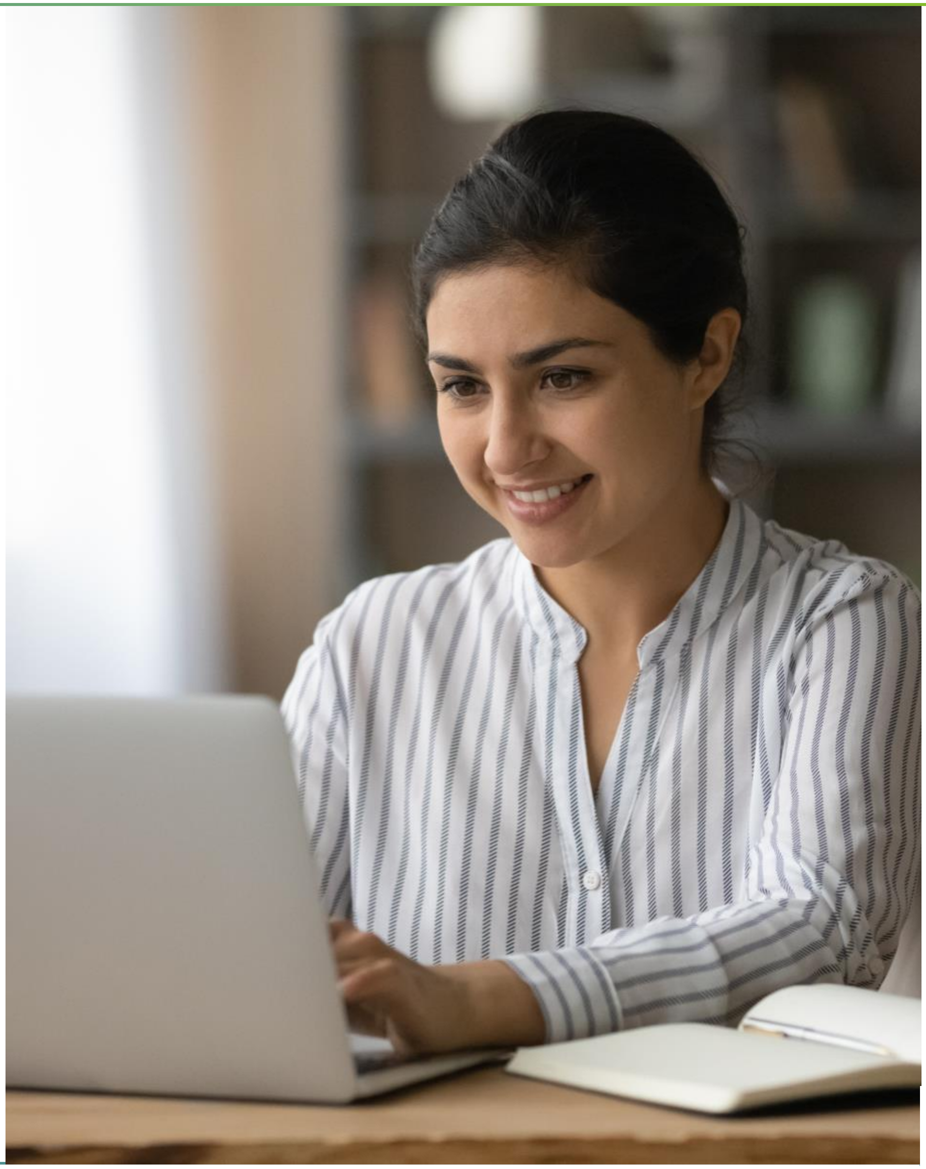
Eureka Server SPRING CLOUD DISCOVERY

spring-cloud-netflix Eureka Server.

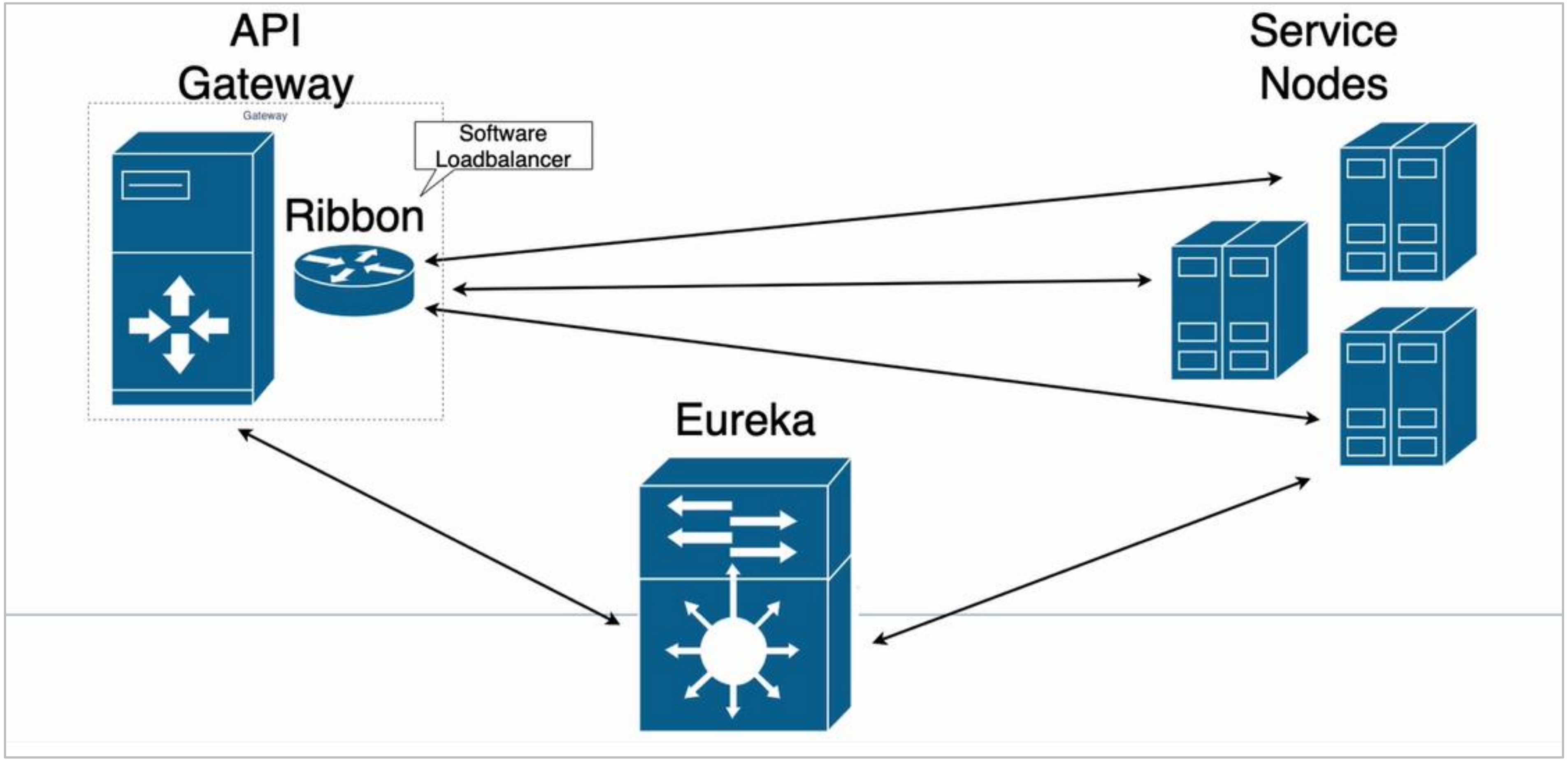
Microservices Ecosystem and Tools—Eureka Server and Eureka Clients



Zuul



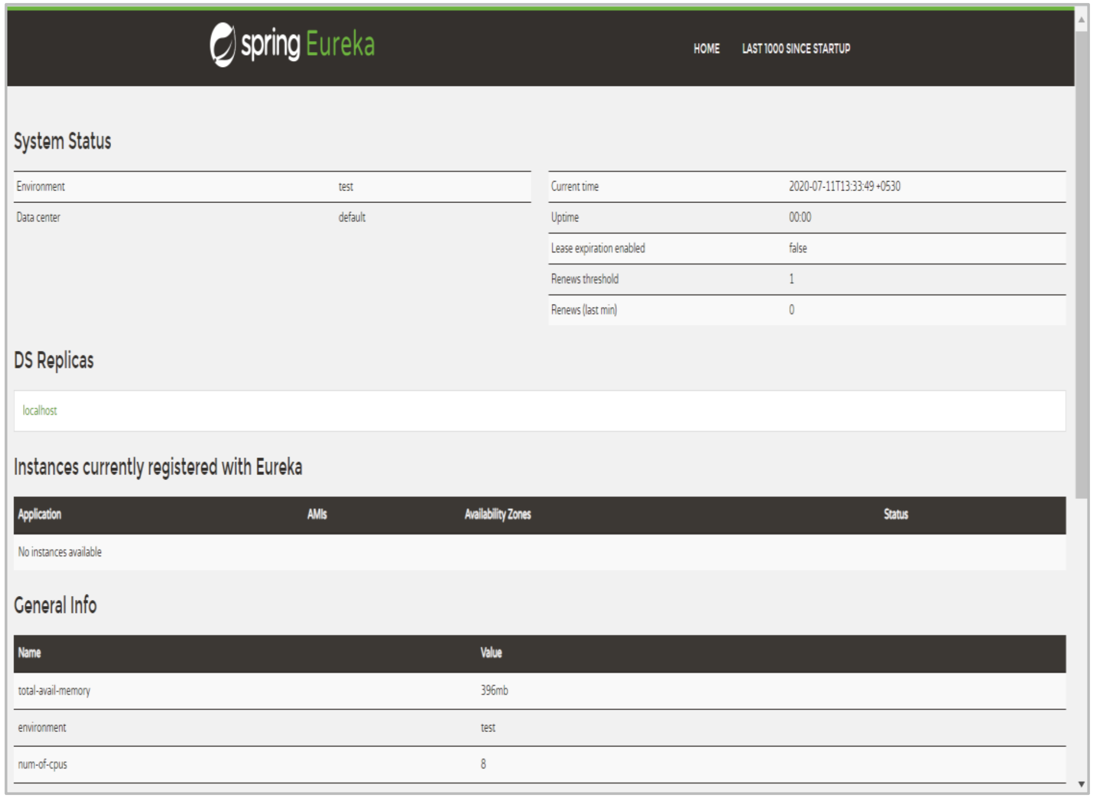
Microservices Ecosystem and Tools—Ribbon



Microservices Ecosystem and Tools—Eureka Illustration

Eureka Server Dashboard

- When a microservice is bootstrapped, it reaches out to the Eureka server, and advertises its existence with the binding information.
- Once registered, the service endpoint sends ping requests to the registry every 30 seconds to renew its lease.
- If a service endpoint cannot renew its lease in a few attempts, that service endpoint will be taken out of the service registry.
- When a client wants to contact a microservice endpoint, the Eureka client provides a list of currently available services based on the requested service ID.



System Status

Environment	test
Data center	default

Current time	2020-07-11T13:33:49 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

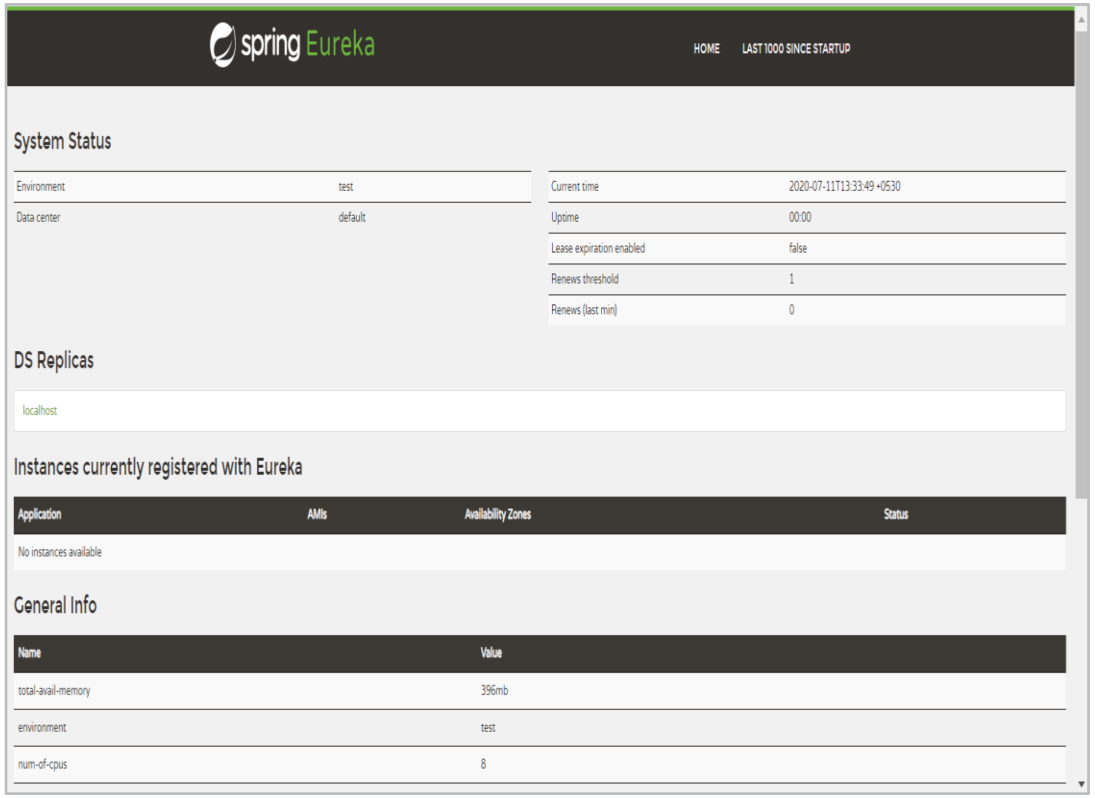
Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	396mb
environment	test
num-of-cpus	8

Microservices Ecosystem and Tools—Eureka Illustration(Cont.)

- The Eureka server is zone aware. Zone information can also be supplied when registering a service.
- When a client requests for a services instance, the Eureka service tries to find the service running in the same zone.
- The Ribbon client then load balances across these available service instances supplied by the Eureka client.
- The communication between the Eureka client and the server is done using REST, and JSON.



System Status

Environment	test
Data center	default

Current time	2020-07-11T13:33:49 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

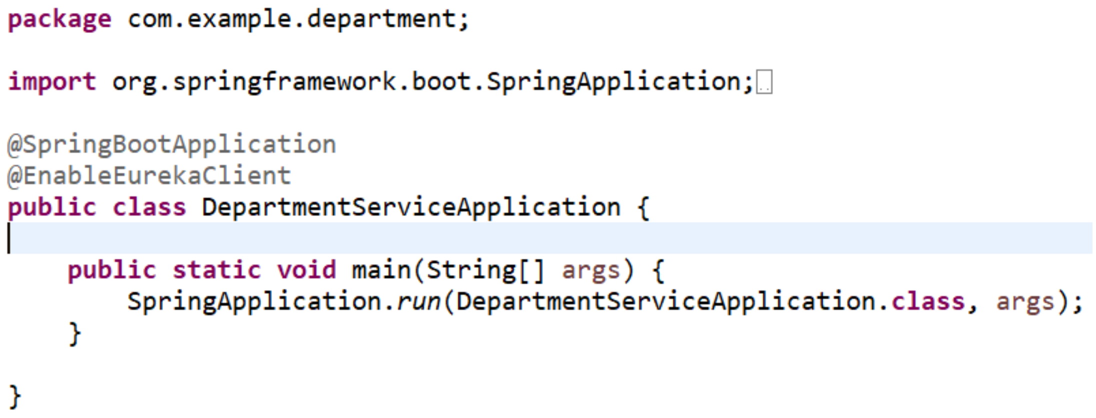
Name	Value
total-avail-memory	396mb
environment	test
num-of-cpus	8

Microservices Ecosystem and Tools—Registering Eureka Clients

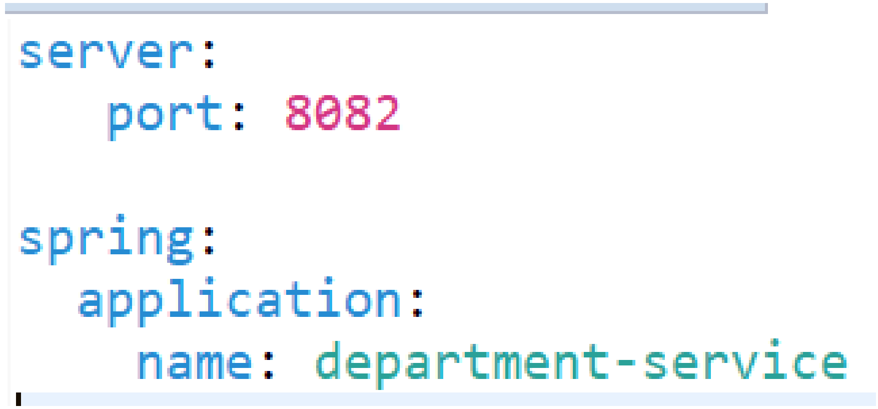
Add dependency in pom.xml

```
<properties> <java.version>1.8</java.version>
<spring-cloud.version>Greenwich.RELEASE</spring-
cloud.version> </properties>
<dependencies> <dependency>
<groupId>org.springframework.boot</groupId>
<artifactld>spring-boot-starter-web</artifactld>
</dependency> <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-abc-eureka-
client</artifactId>
</dependency> <dependency>
<groupId>org.springframework.boot</groupId>
<artifactld>spring-boot-starter-
test</artifactld> <scope>test</scope>
<exclusions)] <exclusion>
<groupId>org.junit.vintage</groupId>
<artifactld>junit-vintage-engine</artifactld>
</exclusion> </exclusions>
</dependency> </dependencies>
```

Add @EnableEurekaClient annotation



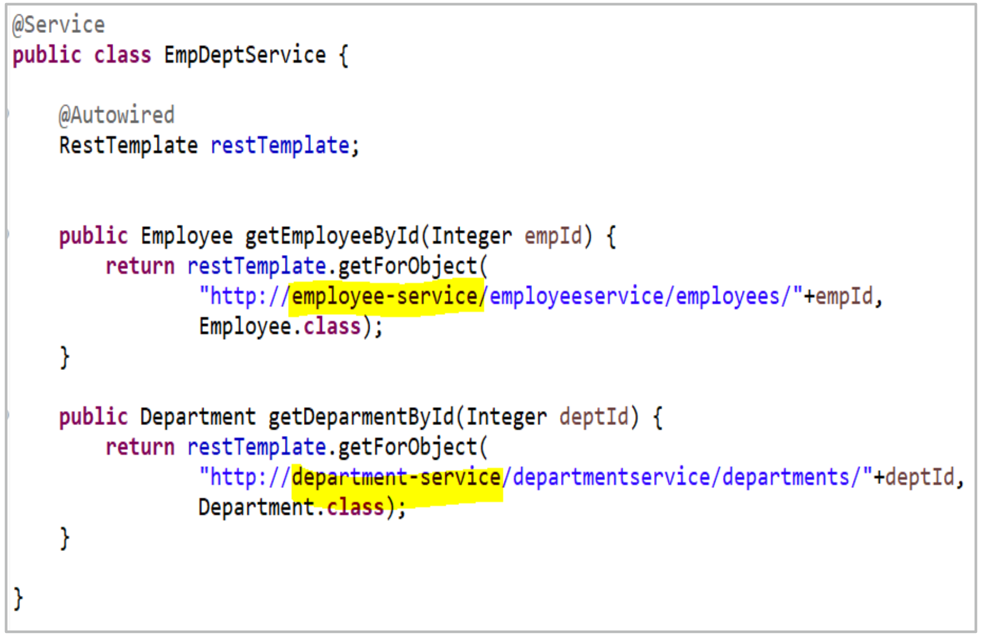
```java
package com.example.department;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableEurekaClient
public class DepartmentServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(DepartmentServiceApplication.class, args);
    }

}
```



```yaml
server:
    port: 8082

spring:
  application:
    name: department-service
```

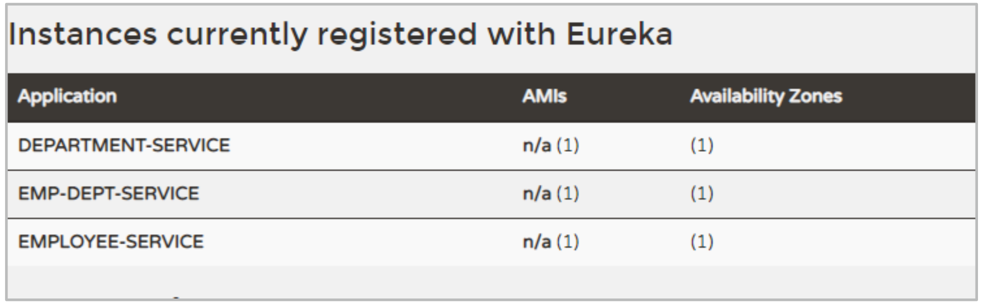
Microservices Ecosystem and Tools—Registering Eureka Clients(Cont.)

Change `Empdeptservice.Java` By Replacing Hard-coded Values



```
@Service
public class EmpDeptService {

    @Autowired
    RestTemplate restTemplate;


    public Employee getEmployeeById(Integer empId) {
        return restTemplate.getForObject(
                "http://employee-service/employeeservice/employees/"+empId,
                Employee.class);
    }

    public Department getDeparmentById(Integer deptId) {
        return restTemplate.getForObject(
                "http://department-service/departmentservice/departments/"+deptId,
                Department.class);
    }

}
```

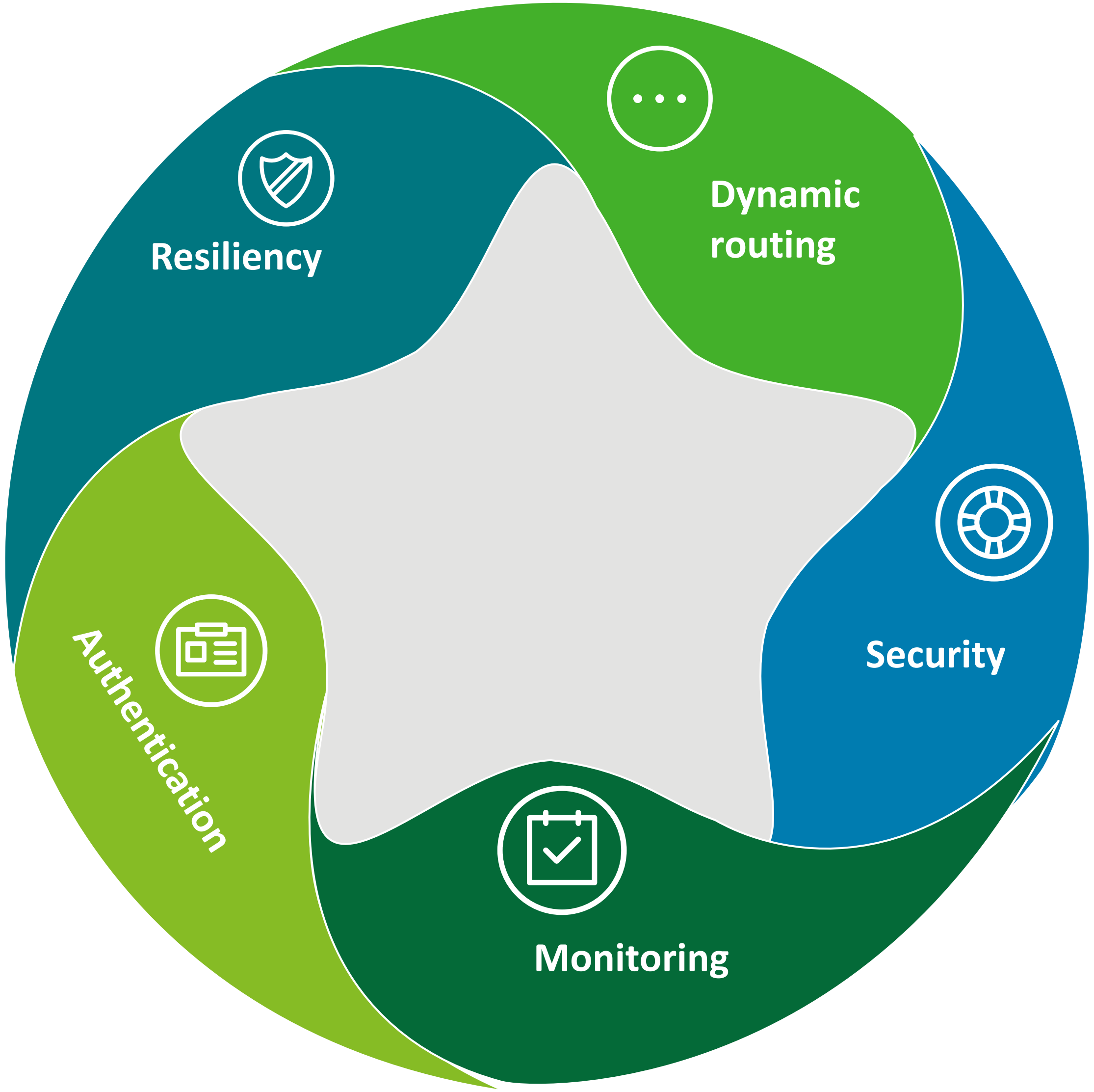
Three services are currently registered in the Eureka server which can be seen in the Eureka dashboard.



Instances currently registered with Eureka

Application	AMIs	Availability Zones
DEPARTMENT-SERVICE	n/a (1)	(1)
EMP-DEPT-SERVICE	n/a (1)	(1)
EMPLOYEE-SERVICE	n/a (1)	(1)

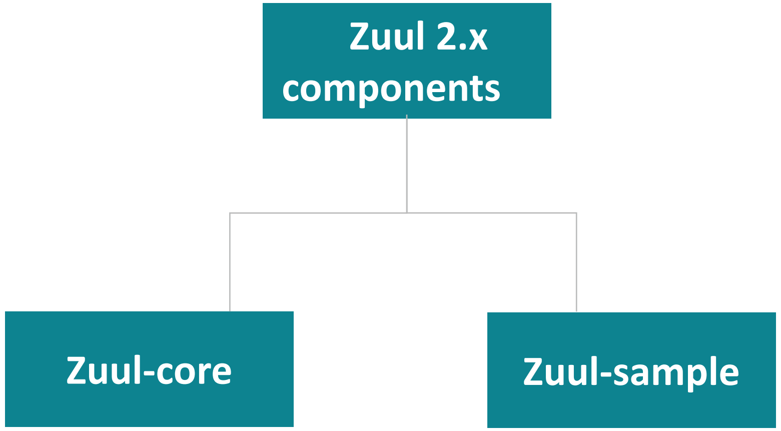
Microservices Ecosystem and Tools—Zuul API Gateway



- Zuul Server is an API Gateway application.
- Microservice applications' dynamic routing is performed by the Zuul Server, and it also handles all the requests.
- Known as the Edge Server it is the front door for all requests.

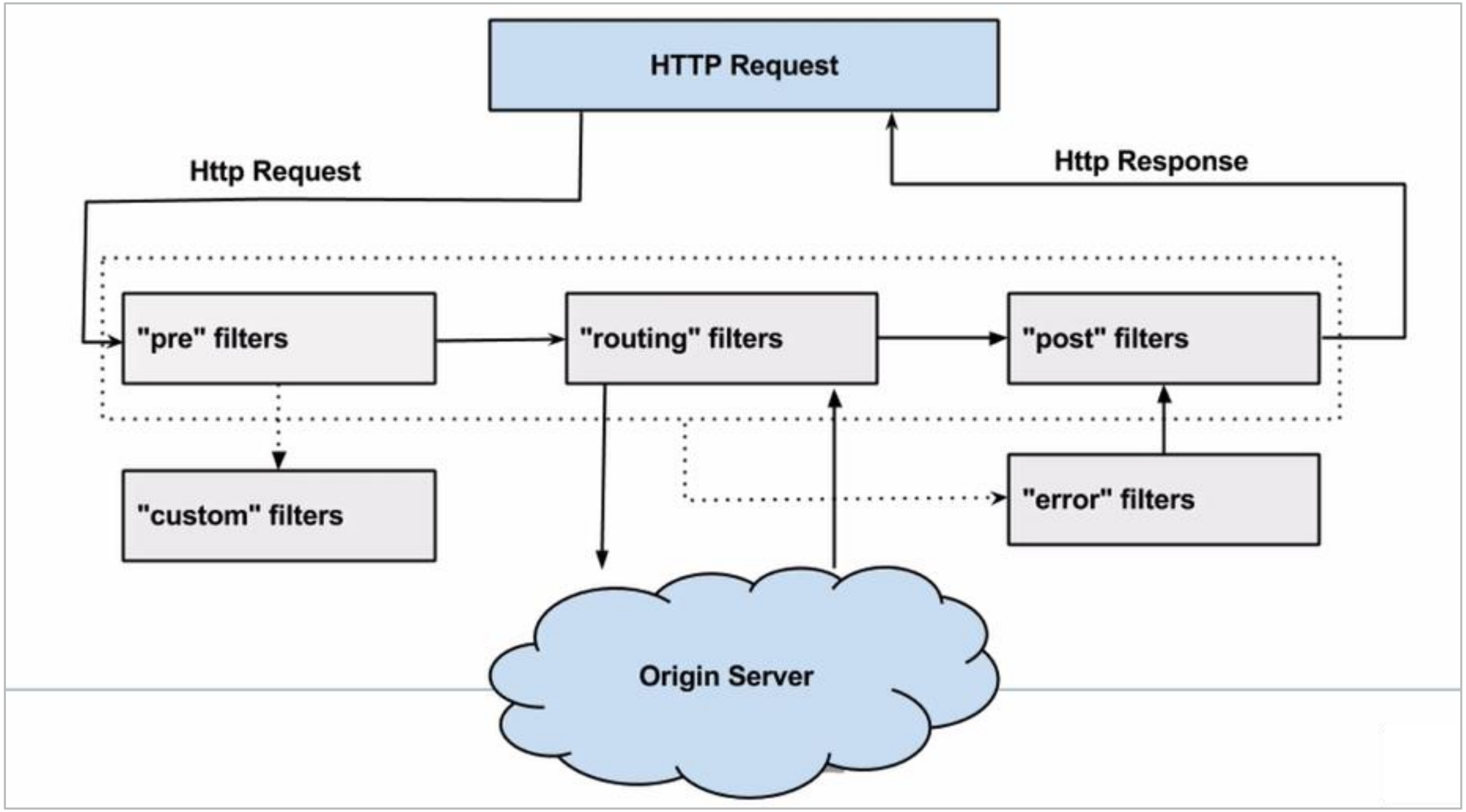
Microservices Ecosystem and Tools—Zuul API Gateway—Filters

Components of Zuul:

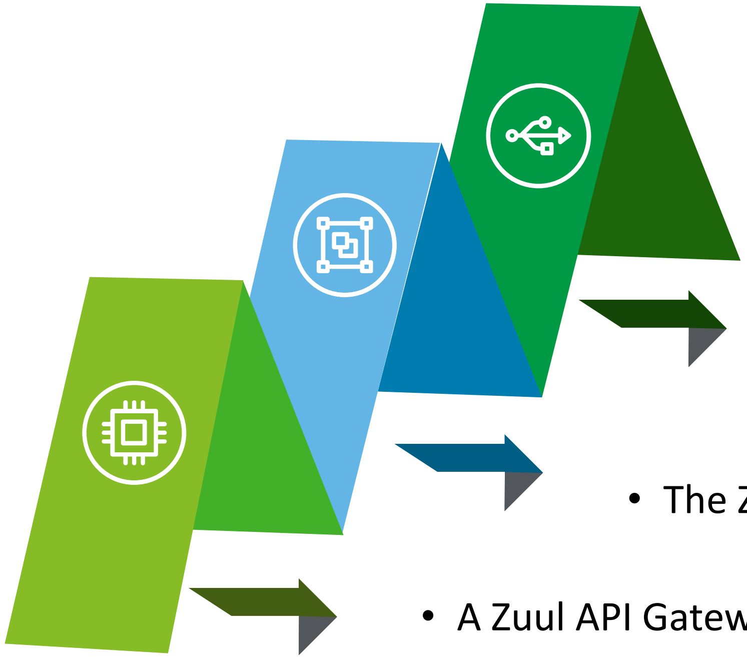


Authentication and Security	Each resource is provided authentic requirements.
Insights and Monitoring	It gives us an accurate view of production by tracking meaningful data and statistics.
Dynamic Routing	Requests are dynamically routed as needed to different backed clusters.
Stress Testing	In order to test performance traffic is increased to a cluster.
Load Shedding	A request that goes over the limit is dropped based on the allocation of capacity for each type of request.
Static Response Handling	Instead of being forwarded to an internal cluster, a response is built directly at the edge.
Multi-region Resiliency	In order to diversify our ELB usage routes are requested across the AWS regions.

Microservices Ecosystem and Tools—Netflix's Architecture of Zuul



Microservices Ecosystem and Tools—Zuul API Gateway Server

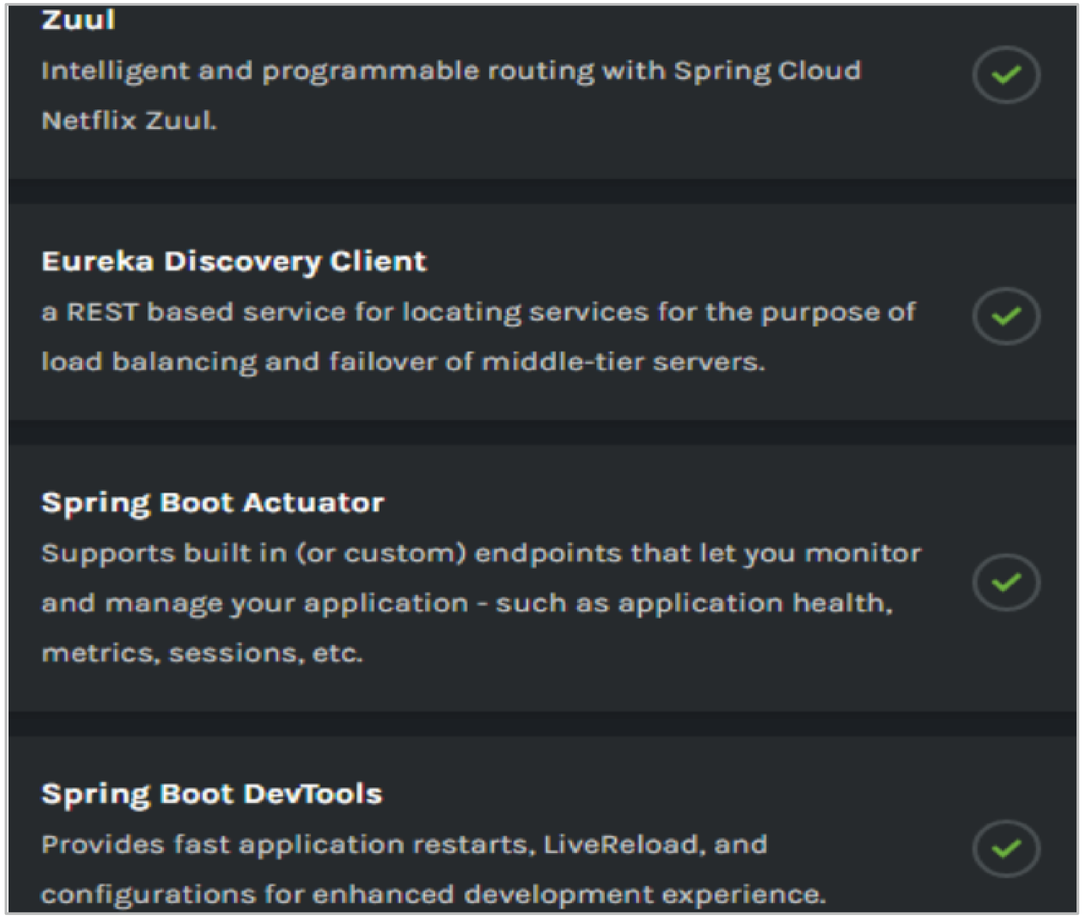


- It is configured that all important requests pass through the Zuul API Gateway.
- The Zuul API Gateway things it should do are decided.
- A Zuul API Gateway component is created.

Microservices Ecosystem and Tools—Zuul API Gateway Server (Cont.)

The Zuul API Gateway server set up steps:

- Open Spring Initializer https://start.spring.io.
- Provide the Group name. ex: com.deloitte.microservices.
- Provide the Artifact. We have provided netflix-zuul-api-gateway-server.
- Add the dependencies: Zuul, Eureka Discovery, Actuator, and dev ools.
- Click on the Generate button, import into STS or Eclipse.



Zuul
Intelligent and programmable routing with Spring Cloud Netflix Zuul. ✓

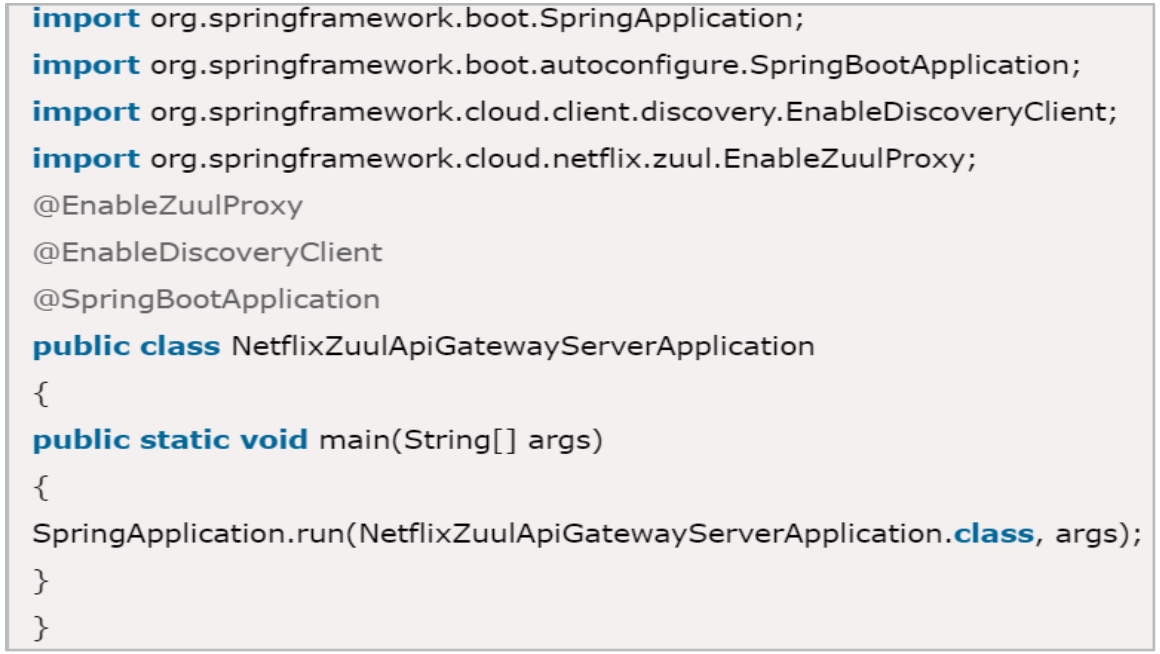
Eureka Discovery Client
a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers. ✓

Spring Boot Actuator
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc. ✓

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience. ✓

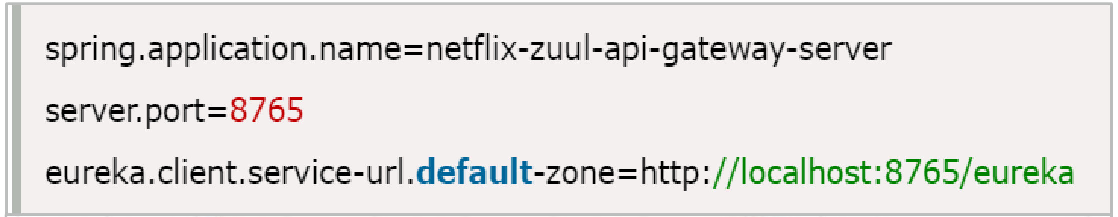
Microservices Ecosystem and Tools—Zuul API Gateway Server (Cont.)

Open the NetflixZuulApiGatewayServerApplication.java file and enable the Zuul proxy and discovery client by using the annotations `@EnableZuulProxy` and `@EnableDiscoveryClient`, respectively.

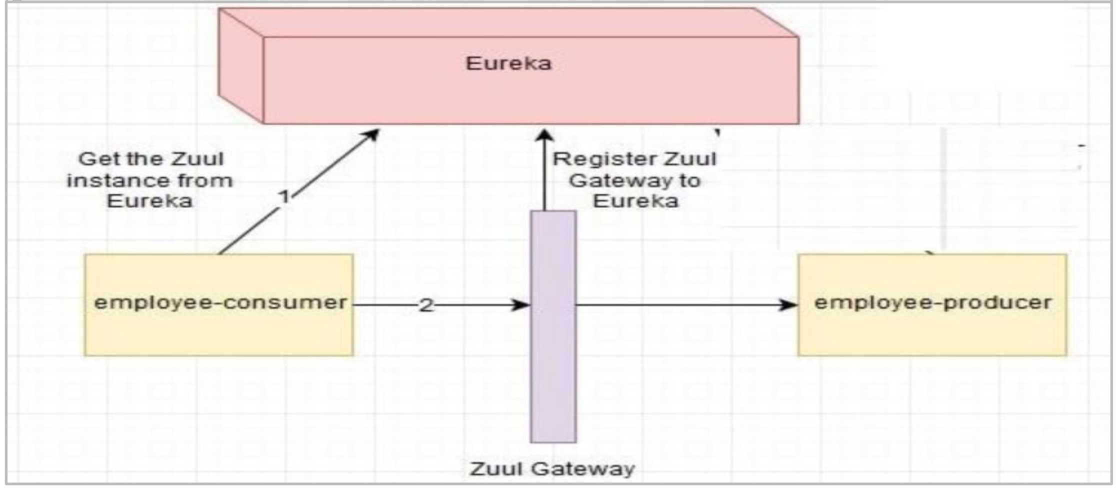


```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
@EnableZuulProxy
@EnableDiscoveryClient
@SpringBootApplication
public class NetflixZuulApiGatewayServerApplication
{
public static void main(String[] args)
{
SpringApplication.run(NetflixZuulApiGatewayServerApplication.class, args);
}
}
```

Open `application.properties` file and configure the application name, port, and eureka naming server.



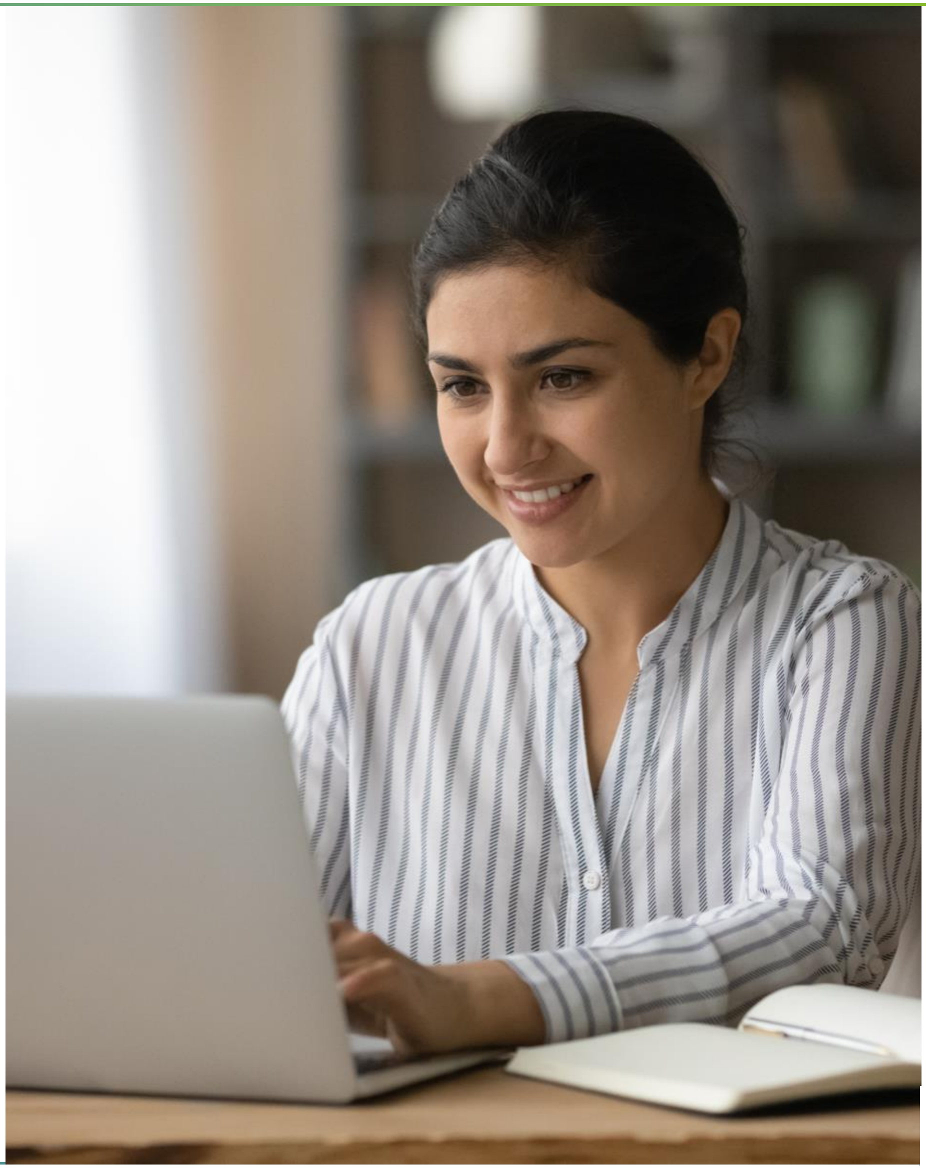
```
spring.application.name=netflix-zuul-api-gateway-server
server.port=8765
eureka.client.service-url.default-zone=http://localhost:8765/eureka
```



Eureka
Get the Zuul instance from Eureka
1
Register Zuul Gateway to Eureka
employee-consumer
2
employee-producer
Zuul Gateway

< / >

Postman—Usage, and Installation



Spring Cloud—What Is Postman?

Postman is a software development tool.

DEBUG

Test APIs, examine responses, add tests and scripts

DESIGN & MOCK

Design in Postman and use Postman’s mock service

DOCUMENT

Create beautiful web-viewable documentation

MONITOR

Create automated tests to monitor API responsiveness

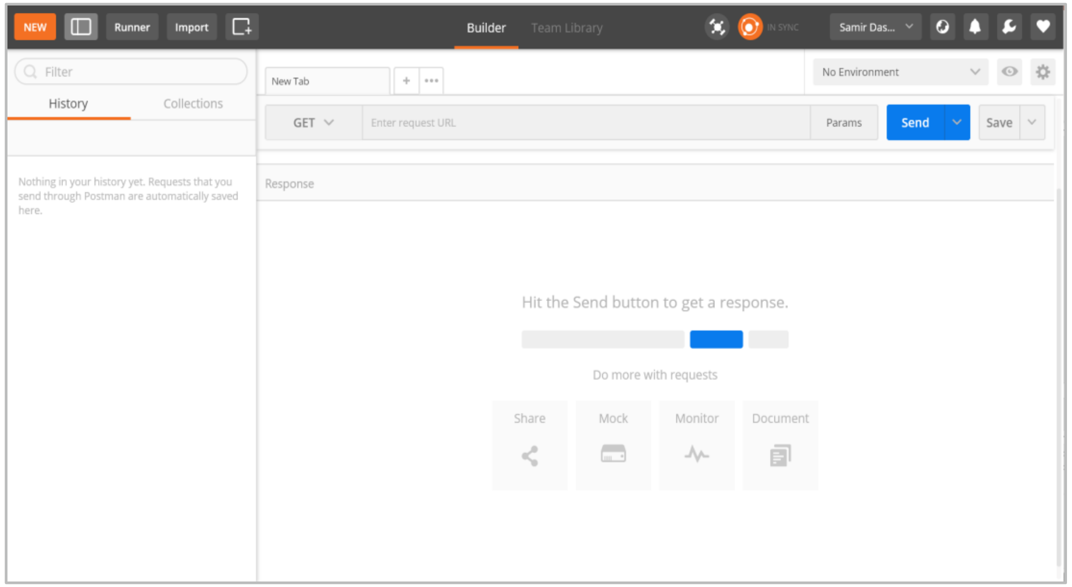
AUTOMATED TESTING

Run Automated tests using Postman collection runner

Spring Cloud—Postman Installation Steps

Installation steps

- The Postman API Client needs to be downloaded from the Postman website `(https://www.getpostman.com/product/api-client).`
- The downloaded installer should be double-clicked and prompts should be followed.
- Launch the application by clicking the Postman icon after the installer finishes.
- Once the application starts for the first time, users will be presented with a login window.
- Have a Postman account? Enter your Postman username and password.
- Have a Google account for university use? The "Sign in with Google" button should be clicked and prompts followed. Do not use a personal Google account.



Nothing in your history yet. Requests that you send through Postman are automatically saved here.

Hit the Send button to get a response.

Do more with requests

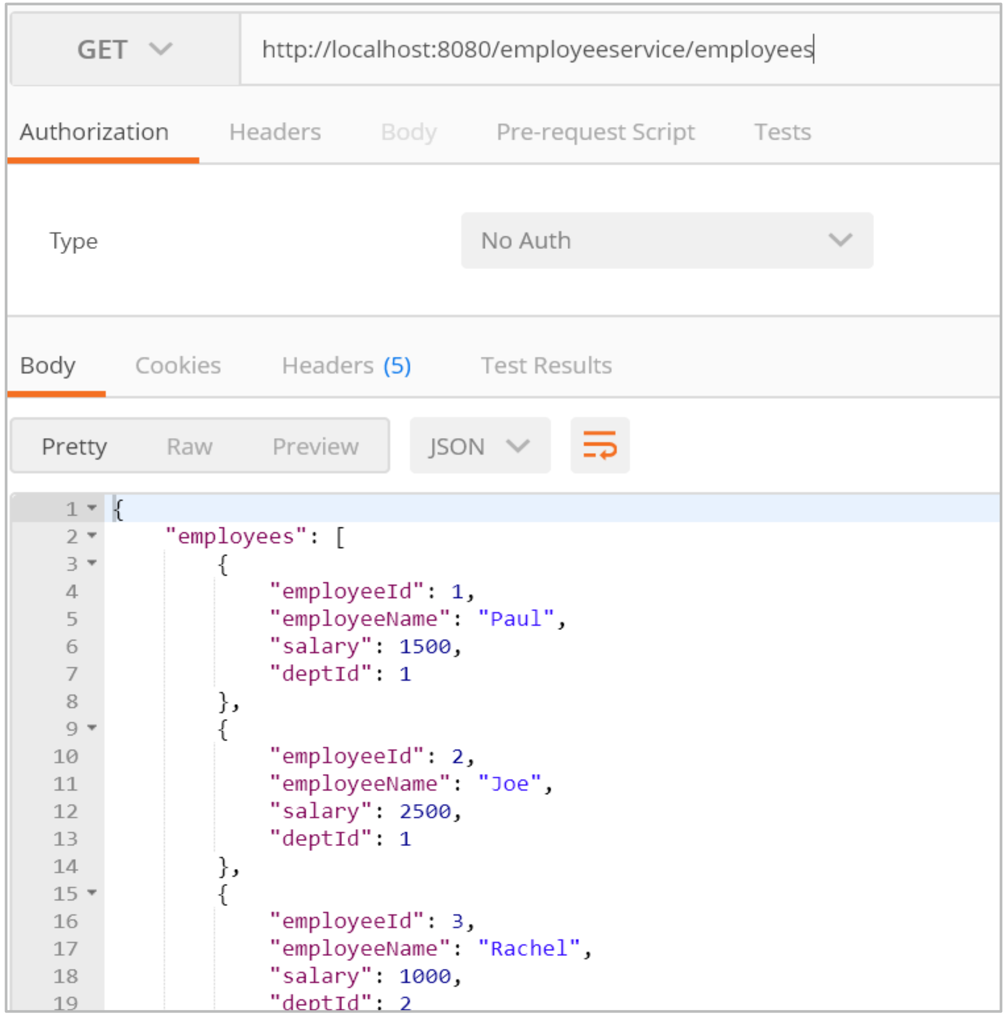
Spring Cloud—Postman in Action

Employee Microservice

- We have defined two REST APIs in Employee microservice in the earlier example.
  - For fetching the list of all employees

    `/employeeservice/employees`
  - For fetching employee by employee Id

    `/employeeservice/employees/{employeeid}`
- Here we see Postman in action where user has involved the `/employees` API and the employee list is returned by the API in the form of a JSON array containing three employees.



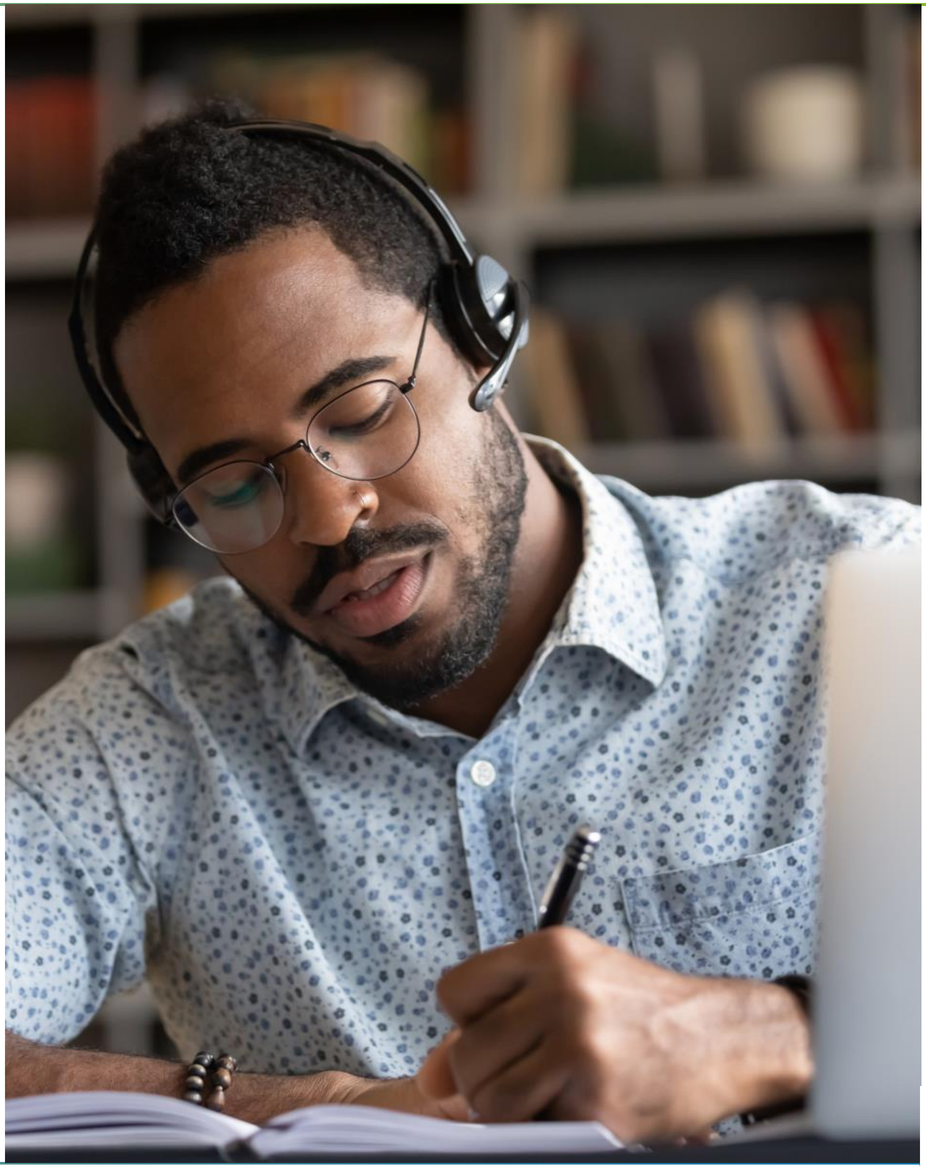
```
{
    "employees": [
        {
            "employeeId": 1,
            "employeeName": "Paul",
            "salary": 1500,
            "deptId": 1
        },
        {
            "employeeId": 2,
            "employeeName": "Joe",
            "salary": 2500,
            "deptId": 1
        },
        {
            "employeeId": 3,
            "employeeName": "Rachel",
            "salary": 1000,
            "deptId": 2
```

Summary

Here are the key learning points of the module.

- The Spring Cloud project is an umbrella project from the Spring team that implements a set of common patterns required by distributed systems, as a set of easy-to-use Java Spring libraries.
- API Gateway encapsulates the internal system architecture. It provides an API that is tailored for each client.
- Config server is an externalized configuration server in which applications and services can deposit, access, and manage all runtime configuration properties.
- Zipkin is an efficient tool for distributed tracing in the microservices ecosystem.
- Spring Boot Admin is a tool for visualizing endpoints exposed by Spring Boot Actuator with health checks and application details.
- Eureka Server is an application that holds the information about all client-service applications.

Hands-On Labs



Hands-On Activity 1

Activity Details	
Problem Statement	Write StudentRestController class to expose end points as, - finding all record on /students - find all students having marks greater than 80 :- /students/marks - finding record by rollNo :- /students/{rollNo} - adding new record /students - updating the name of a student :- /students/{rollNo}/{name} - deleteing the record :- /students/{rollNo} Test all developed endpoints using POSTMAN

Thank You

Deloitte.

About Deloitte
Deloitte refers to one or more of Deloitte Touche Tohmatsu Limited, a UK private company limited by guarantee (“DTTL”), its network of member firms, and their related entities. DTTL and each of its member firms are legally separate and independent entities. DTTL (also referred to as “Deloitte Global”) does not provide services to clients. In the United States, Deloitte refers to one or more of the US member firms of DTTL, their related entities that operate using the “Deloitte” name in the United States and their respective affiliates. Certain services may not be available to attest clients under the rules and regulations of public accounting. Please see www.deloitte.com/about to learn more about our global network of member firms.

This communication contains general information only, and none of Deloitte Touche Tohmatsu Limited (“DTTL”), its global network of member firms or their related entities (collectively, the “Deloitte organization”) is, by means of this communication, rendering professional advice or services. Before making any decision or taking any action that may affect your finances or your business, you should consult a qualified professional adviser.

No representations, warranties or undertakings (express or implied) are given as to the accuracy or completeness of the information in this communication, and none of DTTL, its member firms, related entities, employees or agents shall be liable or responsible for any loss or damage whatsoever arising directly or indirectly in connection with any person relying on this communication. DTTL and each of its member firms, and their related entities, are legally separate and independent entities.