**Spring Boot/microservices projects**

Contents

[ **car-rental-service** 1](#_Toc181113508)

[ **Spring Boot Microservice Project Full Course in 6 Hours** 3](#_Toc181113509)

[ **springboot-microservices-project-master** 22](#_Toc181113510)

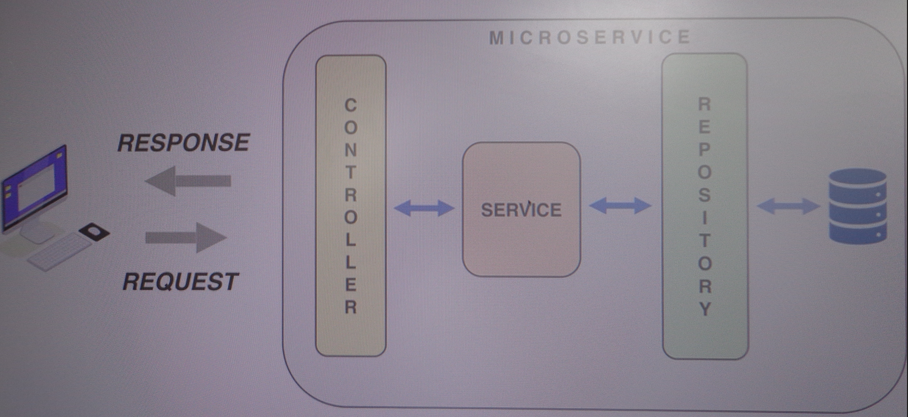
[ **Free Spring Boot Microservices Projects on GitHub | Download | Spring Boot + Spring Cloud: rameshmf projects** 22](#_Toc181113511)

[ **References** 25](#_Toc181113512)

* **car-rental-service**
* **Design**

PC <-> Microservice #1 (user-service)

<-> Microservice #2 (car-rental service)



* **Steps**
* Create a Spring Boot project:

Open Spring Tool Suite 4, New Project, Sprint Boot/Spring Starter Project.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

* Create a RESTful web service

Endpoints:

<http://localhost:8080/hello>

com.example.demo.service: to contain the classes in the business layer

Store all the data in the memory

Data Binding

POST:

A screenshot of a computer

Description automatically generated with medium confidence

PATCH:

A screenshot of a computer

Description automatically generated with medium confidence

* @RequestParam vs @PathVariable

// http://localhost:8080/test/1/Ana

@GetMapping("/test/{id}/{name}")

**public** **void** test(@PathVariable **int** id, @PathVariable String name) {

System.***out***.println(id + " " + name);

}

// http://localhost:8080/test?id=2&name=Ana

@GetMapping("/test")

**public** String test2(@RequestParam **int** id, @RequestParam String name) {

**return** (id + " " + name);

}

Get Header:

A screenshot of a computer

Description automatically generated with medium confidence

refs:

Java EE with Vaadin, JPA, Microservices, Spring and Maven

<https://tcsglobal.udemy.com/course/java-ee-with-vaadin-spring-boot-and-maven/learn/lecture/35581070#overview>

Section 16

* **Spring Boot Microservice Project Full Course in 6 Hours**

<https://www.youtube.com/watch?v=mPPhcU7oWDU>

Simple Online Shopping Application

Spring Boot Microservices Project Tutorial

Spring Boot, Spring Cloud

repository: <https://github.com/SaiUpadhyayula/spring-boot-microservices>

fork: <https://github.com/Zarbio2019/spring-boot-microservices>

**spring-boot-microservices-master-base**: code initial downloaded

**spring-boot-microservices-master**: code with my comments

* **spring-boot-microservices-master-videos:**

Spring Boot Microservices Project Example - Part 1 | Building Services

<https://www.youtube.com/watch?v=lh1oQHXVSc0>

Spring Boot Microservices Project Example - Part 2 \_ Inter Service Communication

<https://www.youtube.com/watch?v=D_XxZU72yMw>

Spring Boot Microservices Project Example - Part 3 | Service Discovery

<https://www.youtube.com/watch?v=0TQliqoX6Kc>

Spring Boot Microservices Project Example - Part 4 | API Gateway

<https://www.youtube.com/watch?v=8PJUnnhLXiI>

Spring Boot Microservices Project Example - Part 5 | Security

<https://www.youtube.com/watch?v=rbKzR6QWKLI>

Spring Boot Microservices Project Example - Part 6 | Resilience4J Circuit Breaker

<https://www.youtube.com/watch?v=kn5h-GTjDY0>

Spring Boot Microservices Project Example - Part 7 | Distributed Tracing

<https://www.youtube.com/watch?v=Cm75_MIo_aY>

Spring Boot Microservices Project Example - Part 8 | Event Driven Architecture using Kafka

<https://www.youtube.com/watch?v=-ebTPcHANnI>

Spring Boot Microservices Project Example - Part 9 | Dockerize the application

<https://www.youtube.com/watch?v=5_EXMJbhLY4>

Spring Boot Microservices Project Example - Part 10 | Monitoring using Prometheus & Grafana

<https://www.youtube.com/watch?v=5X3vKgwqolQ>

* Architectural patterns:

Service Discovery

Centralized Configuration

Distributed Tracing

Event Driven Architecture

Centralized Logging

Circuit Breaker

Secure Microservice using Keycloak

* Topics:

Introduction

High-Level Architecture

Logical Architecture

Build Services

Inter Process Communication

Service Discovery using Netflix Eureka

Implement API Gateway using Spring Cloud Gateway

Secure Microservices using Keycloak

Implement Circuit Breaker

Event Driven Architecture using Kafka

Dockerize the application

Monitoring Microservices using Prometheus and Grafana

* **Services**
* **Product Service**: Create and View Products, acts as Product Catalog.

Database: MongoDB

* **Order Service**: Can order Products.

Database: MySQL

* **Inventory Service**: Can check if product is in stock or not.

Database: MySQL

* **Notification Service**: Can send notifications, after order is placed.

Is a Stateless Service which it does not have any database. Responsible to send out notifications to users.

* **Order Service**, **Inventory Service** and **Notification** **Service** are going to interact with each other.
* Synchronous and Asynchronous Communication.
* **Solution Architecture Diagram**

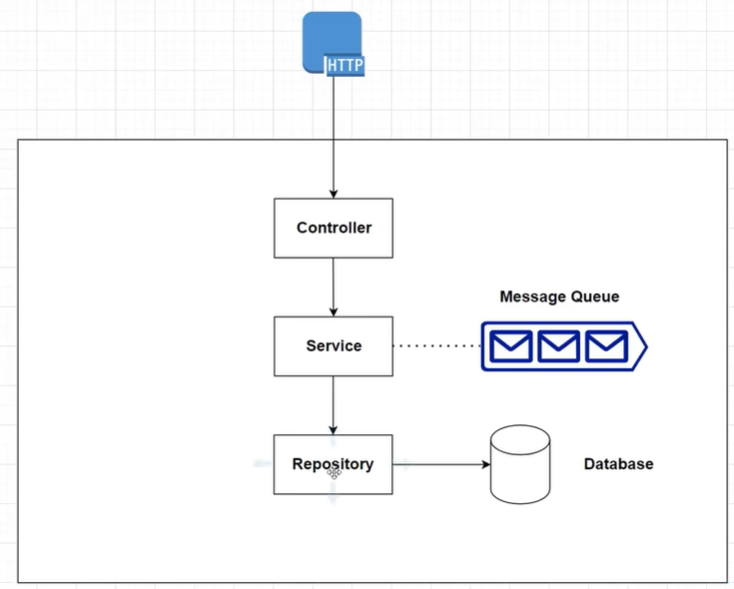
Asynchronous Communication: with Message Queues (Kafka, RabbitMQ)

Zipkin: to do the distributed tracing and also to do the customer to do the login to do the centralized working.

A picture containing text, screenshot, diagram, font

Description automatically generated

* **Logical Architecture**



* **Build Services**
  + **Code: product-service**

Java 17, Maven, Spring Boot 2.6.4

/product-service/src/main/resources/application.properties

* + - Dependencies

Lombok

Spring Web

Spring Data MongoDB

* + - Test with Postman

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

* + - Automated Test

Integrated tests (not write unit test), Junit 5

* Dependencies:

Testcontainers: Java Library for JUnit tests

used for module mongodb: <https://java.testcontainers.org/modules/databases/mongodb/>

Jupiter/Junit 5: <https://java.testcontainers.org/test_framework_integration/junit_5/>

* + **Code: order-service**

Java 17, Maven, Spring Boot 2.6.6

/order-service/src/main/resources/application.properties

server-port=8081 (start point from order-service)

* + - Dependencies

Lombok

Spring Web

Spring Data JPA

MySQL Driver

* + - Test with Postman

A screenshot of a computer

Description automatically generated with medium confidence

* + **Code: inventory-service**

Java 17, Maven, Spring Boot 2.6.6

/inventory-service/src/main/resources/application.properties

server-port=8081 (start point from order-service)

* + - Dependencies

Lombok

Spring Web

Spring Data JPA

MySQL Driver

* + - Test with Postman
  + **Code: microservices-new**

1:20:00 to 1:25:00

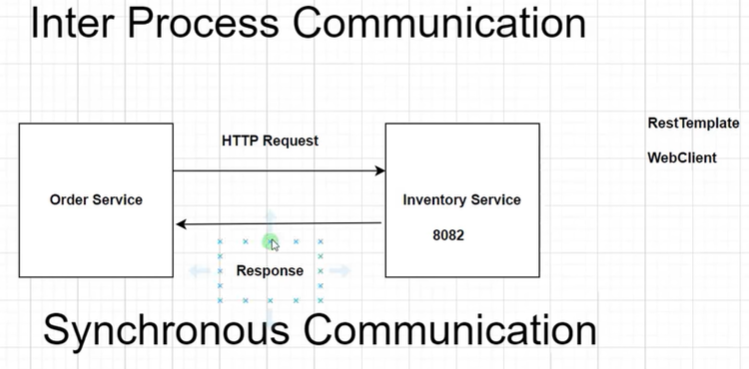
Maven project to create Maven **module** for each service. OpenJDK 17

* + - Commands

Build the project without install: mvn clean verify

* **Inter Process Communication**
  + **Order Service** communicate with **Inventory Service** to check whether the product is in stock or not.

Synchronous Communication when wait for the response.



A picture containing text, screenshot, font, line

Description automatically generated

For Synchronous request can use: RestTemplate or WebClient

In that case use **WebClient**, because RestTemplate is limited <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>

* **Code:**

/order-service/src/main/java/com/programmingtechie/orderservice/service/OrderService.java

/order-service/src/main/java/com/programmingtechie/orderservice/config/WebClientConfig.java

pom.xml: spring-boot-starter-webflux

/inventory-service/src/main/java/com/programmingtechie/inventoryservice/controller/InventoryController.java

* **Test: Place an order**

Inventory in stock: /inventory-service/src/main/java/com/programmingtechie/inventoryservice/InventoryServiceApplication.java

Place an order:

A screenshot of a computer

Description automatically generated with medium confidence

Return an exception: Product is not in stock, please try again later.

A screenshot of a computer

Description automatically generated with medium confidence

* **Service Discovery using Netflix Eureka**

Microservices are running on a cloud environment. For assign IP dynamically with the same port for each service we use Service Discovery.

A diagram of a service

Description automatically generated with medium confidence

* + **Service Discovery**

Create a server something called as a Discovery Server which store all the information about the services (service name, IP addresses) using requests from a Service Registry

A diagram of a service

Description automatically generated with low confidence

Avoid hard coding the URL of the inventory service:

A picture containing text, diagram, font, line

Description automatically generated

A diagram of a service

Description automatically generated with low confidence

Here there are 3 instances of inventory.

**The services are registered in the Discovery Server.**

Discovery Server registers all the instances.

While trying to make a call to the Discovery Server, the client will store a local copy of the registry.

Enable Client-Side LoadBalancing in our Eureka clients.

**Eureka Server** and **Eureka Client** to implement the **Discovery Server**.

* **Dependencies:**

Eureka Server: Netflix Eureka <https://spring.io/projects/spring-cloud-netflix>

* **Code:**

**Eureka Server**: /microservices-new/discovery-server

Spring Cloud Version: 2021.0.2

/microservices-new/discovery-server/src/main/resources/application.properties

**Eureka Client**:

@EnableEurekaClient

* **Test:**

Check the status of our services in a Eureka Dashboard <http://localhost:8761>

A screenshot of a computer

Description automatically generated

* + **Create multiple instances of the Inventory Service**

Assign random port

In Intellij IDEA java you can run multiple instances in parallel

In Eureka Dashboard:

A screenshot of a computer

Description automatically generated with medium confidence

* **Implement API Gateway using Spring Cloud Gateway**

A picture containing diagram, line, plan, technical drawing

Description automatically generated

A diagram of a product service

Description automatically generated with low confidence

The microservices environment can have multiple instances and the application can run on different ports.

So for this reason we cannot rely on hard coding these values when we want to call the services.

**API Gateway** is responsible to route the requests from users to the corresponding services.

A diagram of a product service

Description automatically generated with medium confidence

Routing based on Request Headers

Authentication

Security

Load Balancing

SSL Termination

* + **API Gateway**

Spring Cloud Gateway:

<https://spring.io/projects/spring-cloud-gateway>

<https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/>

**Routes**: Is like a building block of Gateway, so each URL we want to route through the API Gateway we will create a corresponding route.

**Predicate**: lets you define the matching criterias for all the HTTP requests.

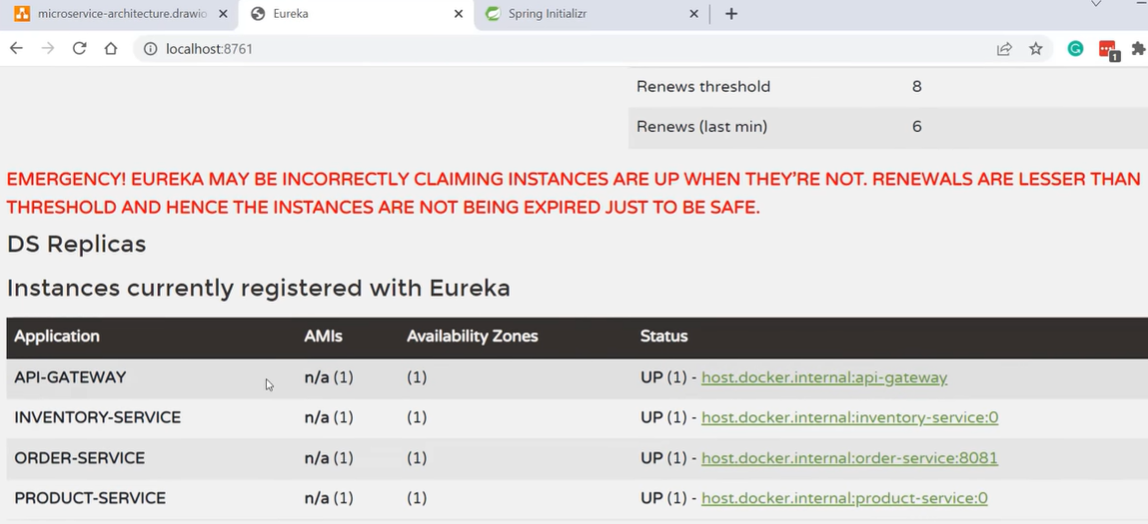
**Filters**: you can modify the requests which are coming into our real chart coming into the API Gateway. So we can modify the request as well as the response, that means you can add additional URL parameters and you can also add some additional headers to the response when we are trying to process the request and response information.

Dependencies: Gateway

* Code:

/microservices-new/api-gateway

/microservices-new/api-gateway/src/main/resources/application.properties



* Make a request to the product service through the API Gateway:

A screenshot of a computer

Description automatically generated with low confidence

* + **Access to Eureka Server through the API Gateway**

A screenshot of a computer

Description automatically generated

* **Secure Microservices using Keycloak**

Authorization Server: Keycloak <https://www.keycloak.org/>

Spring Security

* Code:

/microservices-new/api-gateway/src/main/java/com/programming/techie/apigateway/config/SecurityConfig.java

/microservices-new/api-gateway/src/main/resources/application.properties

* Test:

From postman:

Without authorization (token):

A screenshot of a computer

Description automatically generated with medium confidence

With authorization (token):

A screenshot of a computer

Description automatically generated with medium confidence

* From Keycloak to access to the Discovery Server:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

* **Implement Circuit Breaker**

**Circuit Breaker** is used when we want to have a resilient communication between our services. We don’t want the request to terminate abruptly. Is useful when exist slow network connection, not responding on time (timeout), facing a performance issue, we have some fallback mechanism.

A screen shot of a computer

Description automatically generated with medium confidence

* States of Circuit Breaker:

A picture containing line, diagram, circle, white

Description automatically generated

Spring Cloud Circuit Breaker: <https://spring.io/projects/spring-cloud-circuitbreaker>

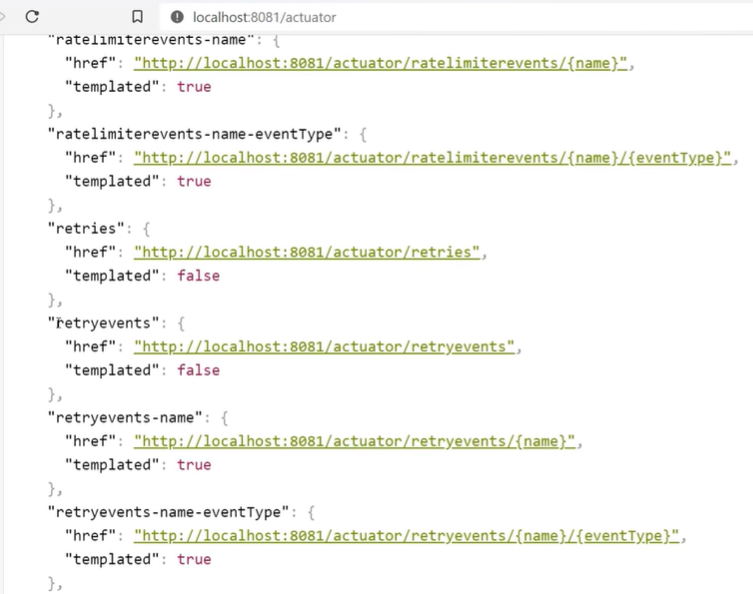
* Library:

Resilience4J: by Netflix <https://resilience4j.readme.io/docs>

* Code:

OrderController.java: fallbackMethod()

localhost:8081/actuator/health



A screenshot of a computer

Description automatically generated with medium confidence

* **Implement Distributed Tracing**
* **Distributed Tracing:** help us to trace the request from the start to the finish.

**A picture containing text, line, font, diagram

Description automatically generated**

* Spring Cloud Sleuth <https://spring.io/projects/spring-cloud-sleuth>, is a tracing framework which help us to generate the trace ID and the span ID whenever we receive a request.
* Zipkin <https://zipkin.io/> : to visualize the information.

A screenshot of a computer

Description automatically generated with medium confidence

* **Event Driven Architecture using Kafka**

**Even Driven Architecture** is used for Async Communication between Order Service and Notification Service in a form of Events.

Apache Kafta <https://spring.io/projects/spring-kafka>: for send messages

Zookeeper

/microservices-new/docker-compose.yml

/microservices-new/order-service/src/main/java/com/programmingtechie/orderservice/event/OrderPlacedEvent.java

* + **Notification Service**

/microservices-new/notification-service

Dependencies: Lombok, Spring Web, Spring for Apache Kafka, Eureka Discovery Client, Zipkin Client, Sleuth

* **Dockerize the application**

Using Docker compose

* api-gateway:

/microservices-new/api-gateway/Dockerfile

* **Monitoring Microservices using Prometheus and Grafana**

Grafana: to visualize

A picture containing text, line, font, number

Description automatically generated

* Dependencies: Spring Boot Actuator, Prometheus

/microservices-new/Prometheus

* Prometheus:

A screenshot of a computer

Description automatically generated with medium confidence

* Grafana:

A screenshot of a computer

Description automatically generated

* **springboot-microservices-project-master**

repository: <https://github.com/SaiUpadhyayula/springboot-microservices-project>

**springboot-microservices-project-master**: code initial downloaded

* **springboot-microservices-project-master-videos:**

Spring Boot Microservices Project - Part 1 - Spring Boot Config Server & Eureka Service Discovery

<https://www.youtube.com/watch?v=Fq85GschdLw>

[REVISED] Spring Boot Microservices Project - Part 2 - Spring Cloud Gateway Keycloak Example

<https://www.youtube.com/watch?v=nGcVzBMjEGg>

* **Free Spring Boot Microservices Projects on GitHub | Download | Spring Boot + Spring Cloud: rameshmf projects**

<https://www.youtube.com/watch?v=uorLNq0QeqQ&list=PLGRDMO4rOGcNSBOJOlrgQqGpIgo6_VZgR&index=64>

<https://www.javaguides.net/2020/06/free-spring-boot-microservices-open-source-projects-github.html>

* **PiggyMetrics**

A simple way to deal with personal finances.

repository: <https://github.com/RameshMF/PiggyMetrics>

* Resources:

Microservice Architecture

Spring Boot

Spring Cloud

Docker

MongoDB

RabbitMQ

* **Shopping Cart App**

Spring Boot Microservice Project - Shopping Cart App | GitHub

<https://www.sourcecodeexamples.net/2020/10/spring-boot-microservice-project.html>

repository: <https://github.com/sivaprasadreddy/spring-boot-microservices-series>

Spring Boot, Spring Cloud

* + **Architecture**
* catalog-service: It provides REST API to provide catalog information like products.
* inventory-service: It provides a REST API to manage product inventory.
* cart-service: It provides a REST API to hold the customer cart details.
* order-service: It provides a REST API to manage orders.
* customer-service: It provides a REST API to manage customer information.
* shoppingcart-ui: It is a customer-facing front-end web application.
* **Spring Petclinic App**

repository: <https://github.com/spring-petclinic/spring-petclinic-microservices>

* Architecture diagram:

A diagram of a computer

Description automatically generated with low confidence

* **Movie Recommendations - Spring Cloud Example Project**

repository: <https://github.com/kbastani/spring-cloud-microservice-example>

* Data Services:

A picture containing diagram, sketch, text, drawing

Description automatically generated

* **Microservices Demo Project**

repository: <https://github.com/piomin/sample-spring-microservices-new>

* + **Architecture**

This sample microservices-based system consists of the following modules:

* **gateway-service**: a module that Spring Cloud Netflix Zuul for running Spring Boot application that acts as a proxy/gateway in our architecture.
* **config-service**: a module that uses Spring Cloud Config Server for running configuration server in the native mode. The configuration files are placed on the classpath.
* **discovery-service**: a module that depending on the example it uses Spring Cloud Netflix Eureka or Spring Cloud Netflix Alibaba Nacos as an embedded discovery server.
* **employee-service:** a module containing the first of our sample microservices that allows us to perform CRUD operation on an in-memory repository of employees
* **department-service:** a module containing the second of our sample microservices that allows to perform CRUD operation on an in-memory repository of departments. It communicates with employee-service.
* **organization-service:** a module containing the third of sample microservices that allows us to perform CRUD operation on an in-memory repository of organizations. It communicates with both employee-service and organization-service.

A diagram of a service

Description automatically generated with low confidence

* Restful Web Service with Spring Boot V2
* Create project

<https://start.spring.io>

Maven, Java 17, Spring Boot 3.0.0 (M4), Jar

Dependencies: Spring Web, Spring Data JPA, H2 Database, Spring Boot DevTools

* **References**
* Helloworld Microservices Introduction

<https://www.youtube.com/watch?v=SPATMHP-xw8>

<https://github.com/redhat-helloworld-msa>

* Aprende Microservicios con Spring Boot

<https://www.youtube.com/watch?v=oD09MVpLZ-s>

* Java Microservices with Spring Boot and Spring Cloud

<https://developer.okta.com/blog/2019/05/22/java-microservices-spring-boot-spring-cloud>

A picture containing text, screenshot, diagram, design

Description automatically generated

Java Microservices in 20 Minutes with Spring Boot and Spring Cloud

<https://www.youtube.com/watch?v=rH-FnPC_xCA>

* Spring Boot Tutorial for Beginners - Learn Spring Boot in 10 Hours

<https://www.youtube.com/watch?v=_thI-4AF7M8>