



Microservices

Spring Cloud

Summary:

Now you will master the skills of creating microservice applications using Spring framework

Version: 1.00

Contents

I	Preamble	2
II	General Rules	3
III	Rules of the project	4
IV	Exercice 00: So many services	5

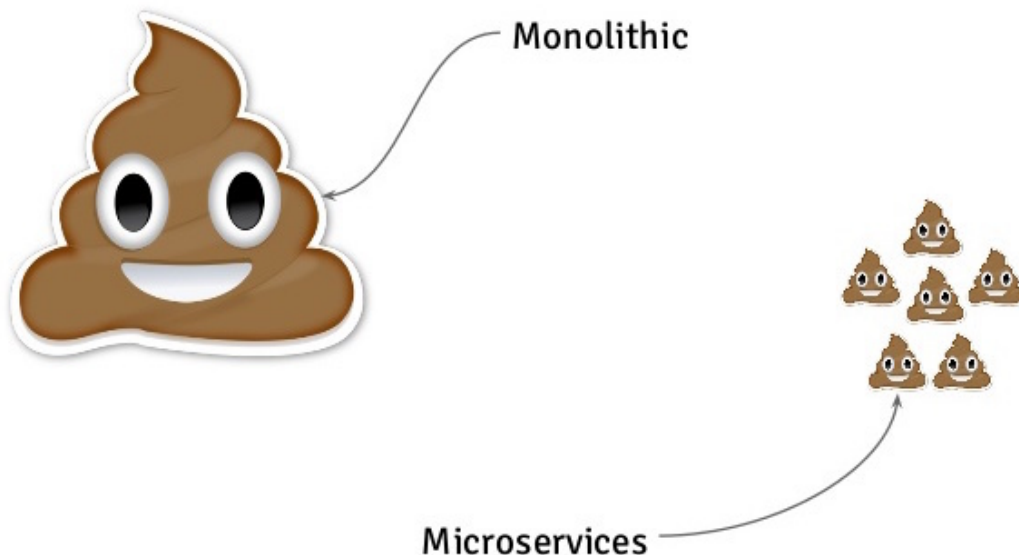
Chapter I

Preamble

Two important components of **Spring Cloud** infrastructure are:

- **Service Discovery**—a service that registers all other modules of the system. In **Spring Cloud**, this is implemented using **Netflix Eureka**. Each microservice in the system is an **Eureka** client and knows its location. Thus, regardless of the total number and location of service instances, each of them can access the other through **Eureka**.
- **Api Gateway** is a service that provides a single entry point to the functionality of all other applications in the system. One implementation in **Spring Cloud** is **Netflix Zuul**.

Monolithic vs Microservices



Chapter II

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- You must use the latest LTS version of Java. Make sure that compiler and interpreter of this version are installed on your machine.
- You must use GraalVM to run your code.
- You can use IDE to write and debug the source code (we recommend IntelliJ Idea).
- The code is read more often than written. Read carefully the [document](#) where code formatting rules are given. When performing each exercise, make sure you follow the generally accepted [Oracle standards](#)
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output
- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)


Chapter III

Rules of the project

- The solution for each exercise is a standalone **Maven** project implemented on the basis of **Spring Boot**.
- Project structure is at a developer's discretion.
- Each project shall contain a `data.sql` file with a set of test data.

Chapter IV

Exercise 00: So many services

	Exercise 00
So many services	
Turn-in directory : <i>ex00/</i>	
Files to turn in : InformationServices-folder	
Allowed functions : n/a	

You need to create a few standalone microservices with the following functionality:

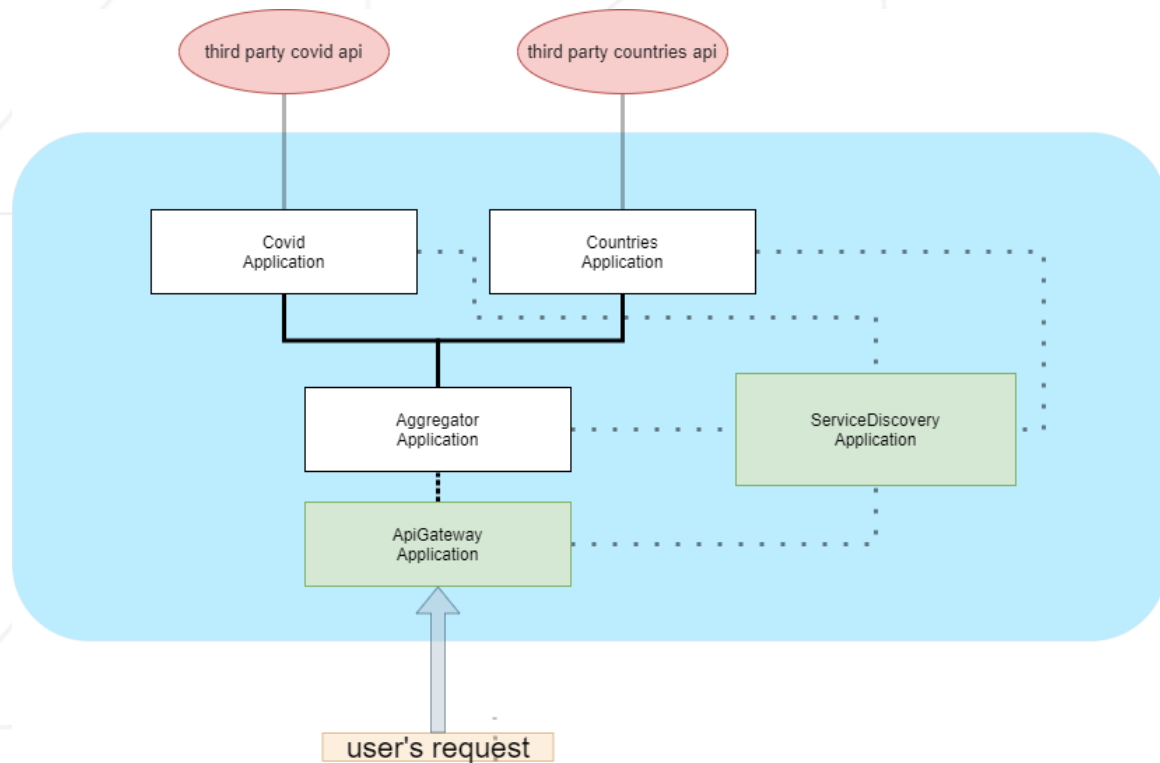
- **CountriesApplication** service that provides data about a country by its name. The information may include population, capital city, area code, etc. Corresponding URI: `/countries-management/countries/{country-name}`
- **CovidApplication** service, which provides information on statistics on COVID-19 for any country. Corresponding URI: `/covid-management/countries/{country-name}`
- **AggregatorApplication** service that provides complex information containing data obtained from **CountriesApplication** and **CovidApplication**. Corresponding URI: `/information-management/countries/{country-name}`

To obtain data by both country and COVID-19 statistics, you need to use third-party resources, for example: <https://covid-api.com/api> and <https://restcountries.com>

Thus, to handle a user's GET request to the above URL, **AggregatorApplication** should request data from two other services. The latter, in turn, will receive data from third-party APIs and then provide information to **AggregatorApplication**.

To handle user requests, you need to implement **ApiGatewayApplication** providing a single URI from **AggregatorApplication**.

General architecture of the system is shown below:



Requirements:

- Running a suite of applications shall be possible inside Docker environment.
- There shall be a **README** file that contains instructions for running applications inside **Docker**, as well as documentation of third-party **API** methods used in the application.
- **ApiGatewayApplication** shall be deployed on port 80
- **ServiceDiscoveryApplication** shall be available on the standard port of **Eureka**
- All other applications shall be launched on random ports each time they start.
- Thus, **AggregatorApplication** shall not have direct links (hostname + port) to **CovidApplication** and **CountriesApplication** services. These services shall only be accessed by their names.
- None of the applications shall have their own database to store information.