Contents

[1 Theory 2](#_Toc94188514)

[1.1 Analogy 3](#_Toc94188515)

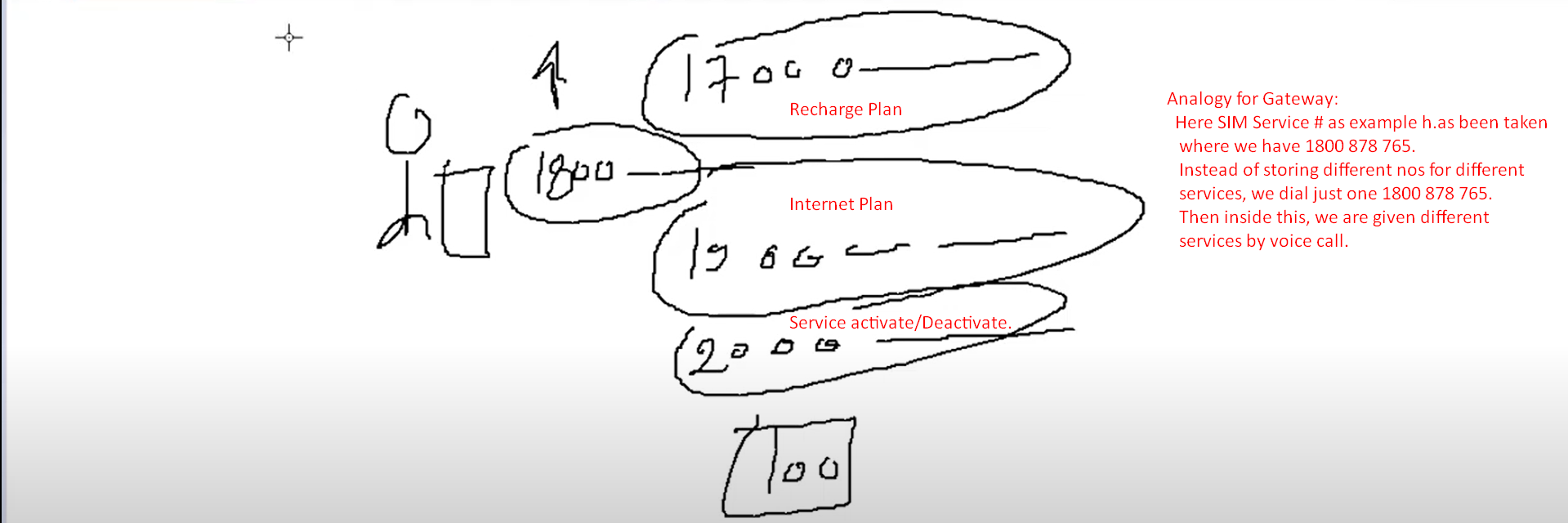
[1.2 Why 3](#_Toc94188516)

[1.3 API Gateway supports 3](#_Toc94188517)

[2 How to implement API Gateway 2](#_Toc94188518)

# Theory

## Analogy

1. 

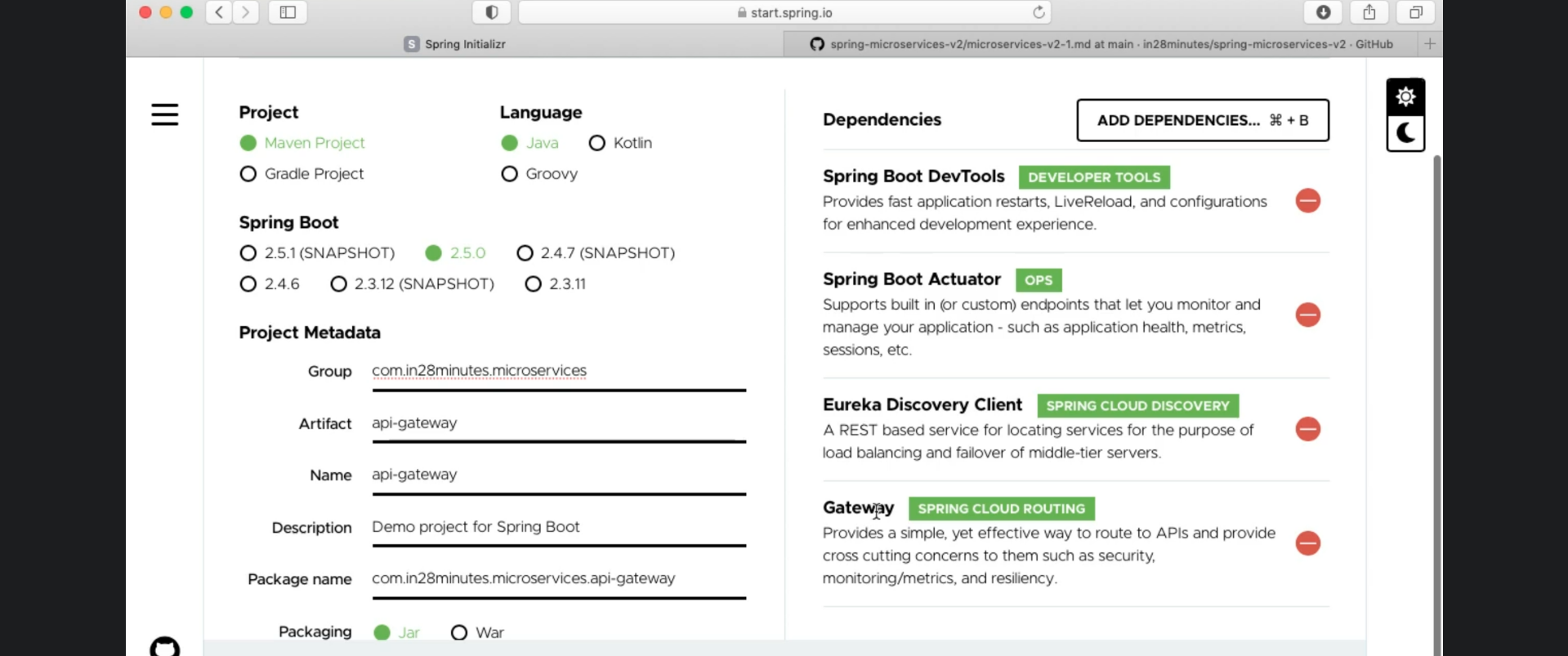
## Why

* 1. Common features across all microservices in microservice architecture:
     1. Routing to APIs.
     2. Cross Cutting Concerns:
        1. Security:
           1. Authentication.
           2. Authorization.
        2. Logging.
        3. Rate Limiting.
        4. Monitoring/Metrics.
        5. Resiliency.
        6. Auditing.
        7. Tracing.
     3. Central Policy Enforcement Point (PEP) for all calls.
  2. **Routing to APIs**:
     1. Hundred of ms along with many instances. Some are exposed to outside world (UI, Vendors).Now you have routing requirement such as if receiving so and so path, then route to a particular microservice.
     2. Based on request parameters, you want to do dynamic routing.   
        **For example**: if receiving beta version as request parameter, redirect to a new beta ms which you’re trying to evaluate otherwise redirect to stable ms.
  3. **Cross Cutting Concerns**:
     1. Why handling such concerns in API Gateway in centralized manner rather than asking corresponding developers of a microservice?
        1. 1st each developer has its own way to implement them. So no consistency in implementation.
        2. In worst case, someone developer may not implement security correctly thus insure ms.
     2. You may think that why don’t we create a **common library (Jar)** having crosscutting logic for all MSs.
        1. This way, you’re tightly coupling your MSs with this common library.  
           Such as if you want to introduce a new change in the common library, you have to consider the followings:
           1. What would be the impact on all microservices?
           2. How will my ms react?
           3. There is no **regression issue/bug**.  
              Thus all this will result in monolithic architecture style which we’re trying to avoid with ms.  
              Thus we should have a separate entity/ms handling all these crosscutting concerns.
  4. **A Single Gatekeeper**:

## API Gateway supports

* 1. **Zuul**: Was used in older versions of Spring Cloud.
     1. As Netflix is not supporting Zuul and is now in maintance mode.
  2. **Spring Cloud Gateway** project: By Spring Cloud.

# How to implement API Gateway

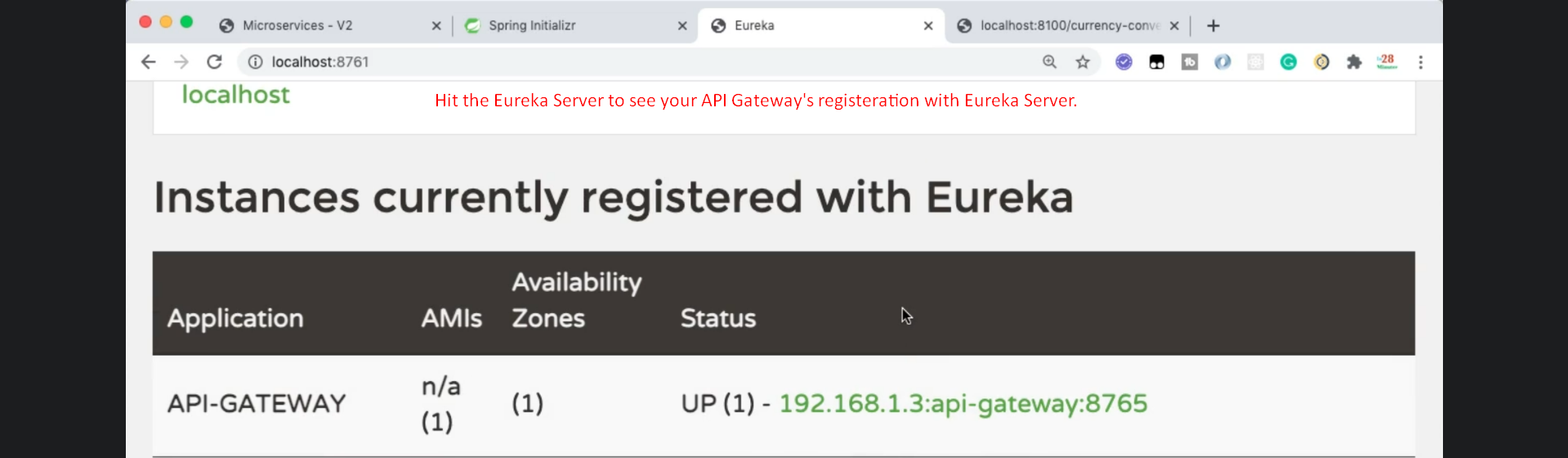
1. **Steps**:
   1. **Step 01**: Add dependencies
      1. 
         1. Let’s talk about dependencies:
            1. **Gateway**:

Spring Cloud Routing Project.

To convert your microservice project into Spring Cloud API Gateway.

* + - * 1. **Eureka Discovery Client**:

Gateway needs registry info about other microservices so that it can route to APIs and do load balancing. So Gateway needs to connect with Eureka Server to fetch registry info.

* 1. **Step 02:** Make configurations in application.properties file.
     1. 
        1. eureka.client.serviceUrl.defaultZone:
           1. Optional if Eureka server is running on the same machine as Gateway and on default port (8761).
  2. **Step 03**: Check if Gateway MS is properly registered with Eureka Server.
     1. 
  3. **Step 04**: How to access a microservice using Spring Cloud API Gateway
     1. http://localhost:8765/<app-name>/path