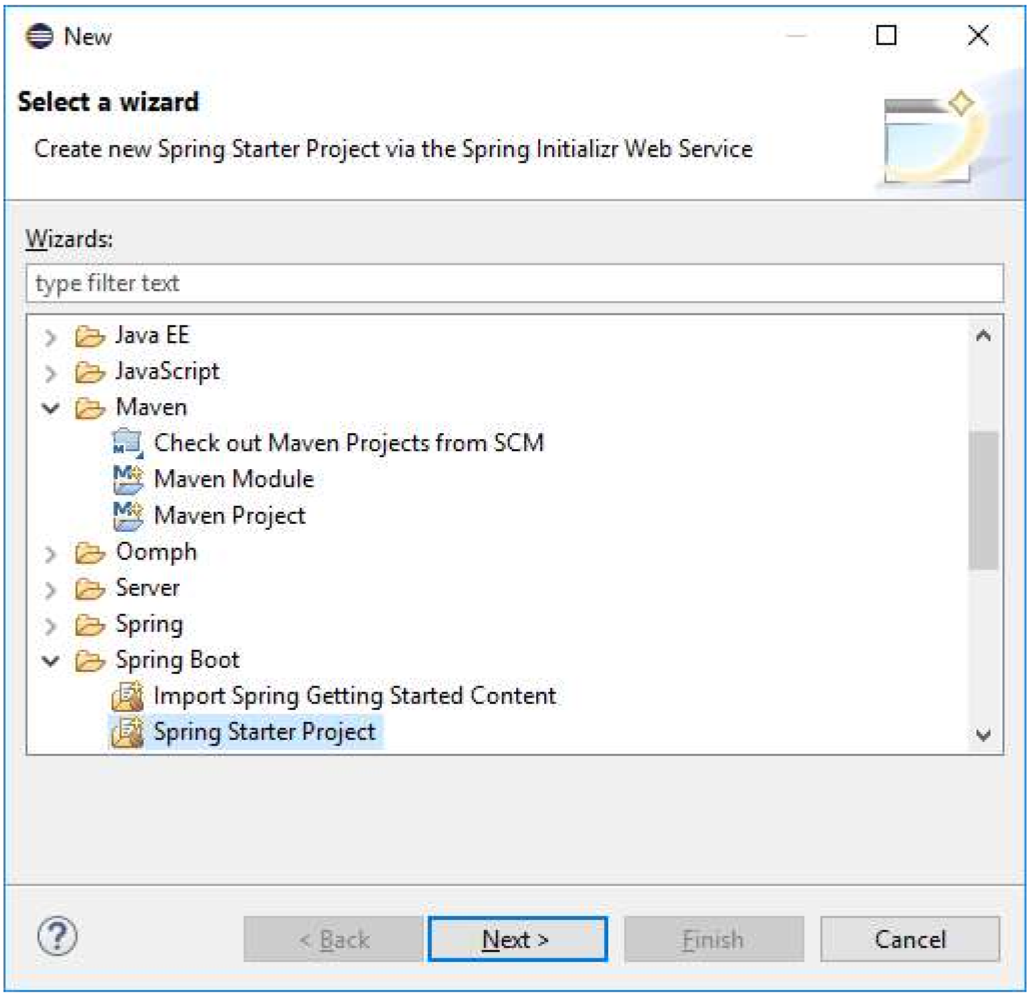
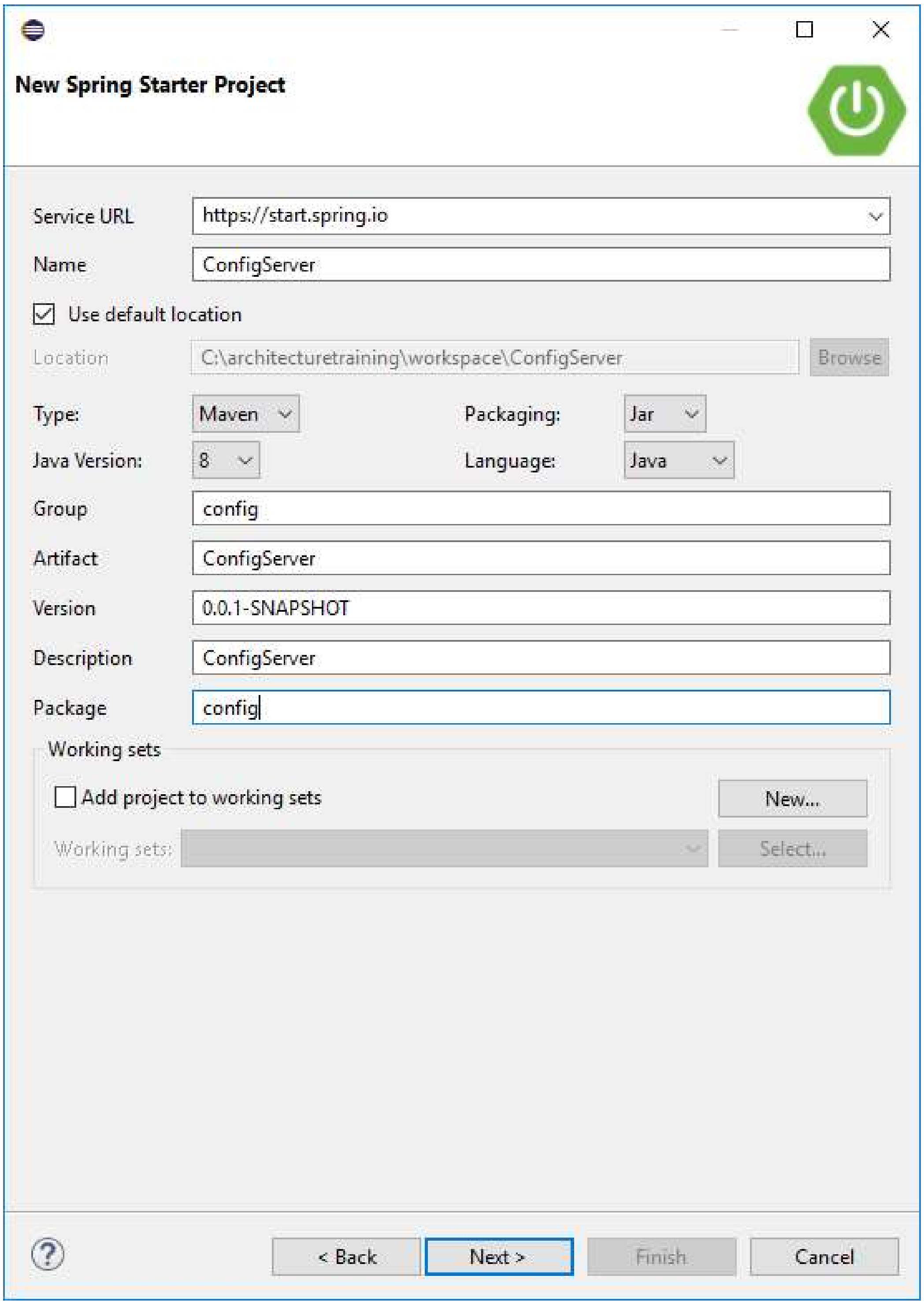
Exercise 1: Spring Boot Config Server

First we will create a config server.
In Eclipse select the menu items **File->New->Other** and select **Spring Boot->Spring Starter Project**.



Click **Next**



Service URL: https://start.spring.io

Name: ConfigServer

[✓] Use default location

Location: C:\architecturetraining\workspace\ConfigServer

Type: Maven

Packaging: Jar

Java Version: 8

Language: Java

Group: config

Artifact: ConfigServer

Version: 0.0.1-SNAPSHOT

Description: ConfigServer

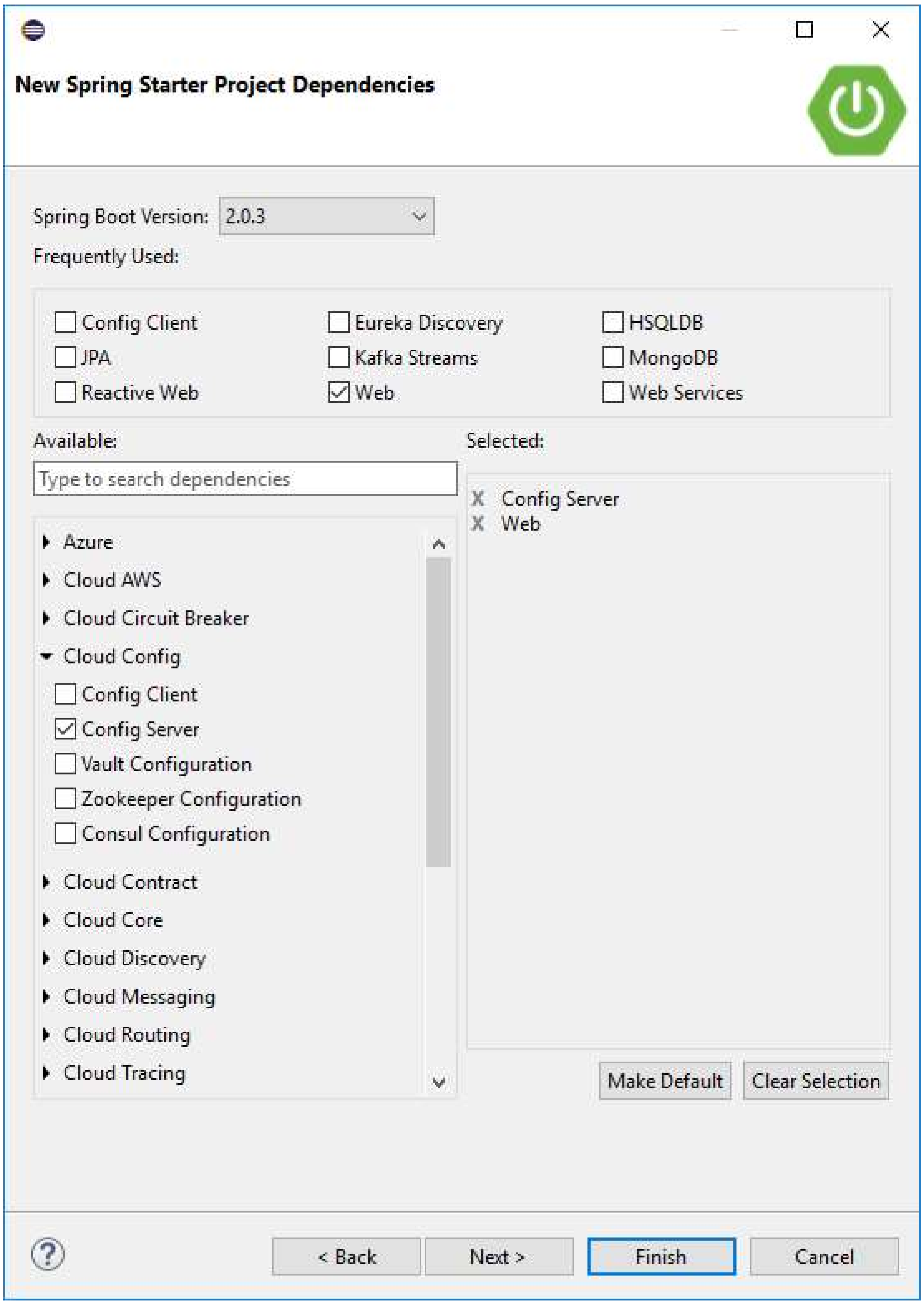
Package: config

Working sets

[ ] Add project to working sets

Working sets:

Fill in the details as given in the picture above and click **Next**.


New Spring Starter Project Dependencies

Spring Boot Version: 2.0.3

Frequently Used:

- [ ] Config Client
- [ ] Eureka Discovery
- [ ] HSQLDB
- [ ] JPA
- [ ] Kafka Streams
- [ ] MongoDB
- [ ] Reactive Web
- [✓] Web
- [ ] Web Services

Available:

Type to search dependencies

- Azure
- Cloud AWS
- Cloud Circuit Breaker
- Cloud Config
  - [ ] Config Client
  - [✓] Config Server
  - [ ] Vault Configuration
  - [ ] Zookeeper Configuration
  - [ ] Consul Configuration
- Cloud Contract
- Cloud Core
- Cloud Discovery
- Cloud Messaging
- Cloud Routing
- Cloud Tracing

Selected:

X Config Server
X Web

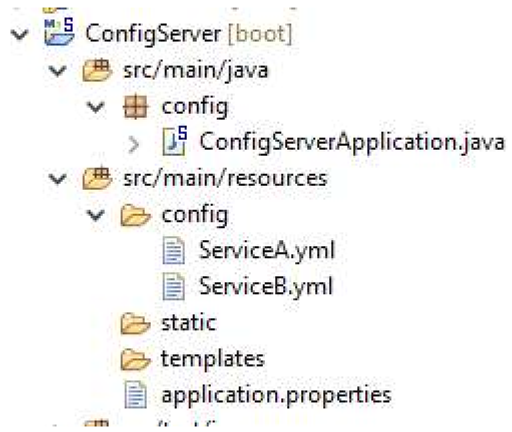
Select **Web** and **Config Server (from Cloud Config)** and click **Finish**.

Then add the **@EnableConfigServer** annotation to the application:

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

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

Then create a new folder with the name config in scr/main/resources and add 2 new files with the names ServiceA.yml and ServiceB.yml



```
ConfigServer [boot]
  src/main/java
    config
      ConfigServerApplication.java
  src/main/resources
    config
      ServiceA.yml
      ServiceB.yml
    static
    templates
    application.properties
```

Fill both config files with the following content:

config/ServiceA.yml

```
greeting: Hello from Service A
```

config/ServiceB.yml

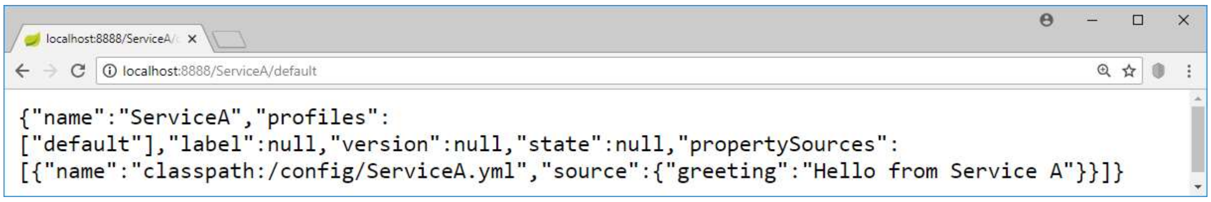
```
greeting: Hello from Service B
```

Then fill application.properties with the following content:

application.properties

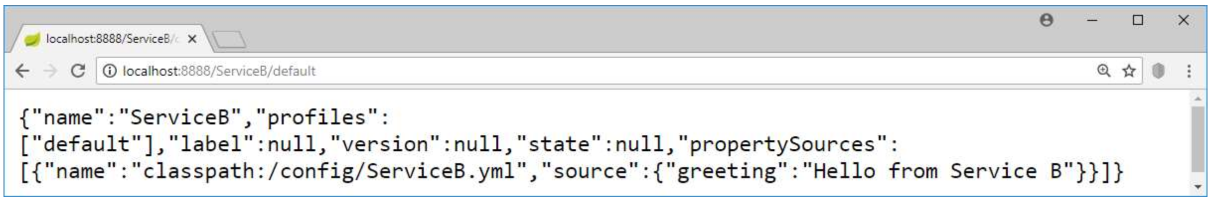
```
spring.profiles.active=native
server.port=8888
```

Now run the ConfigServer and check if it works:



localhost:8888/ServiceA/default

```
{"name":"ServiceA","profiles":
["default"],"label":null,"version":null,"state":null,"propertySources":
[{"name":"classpath:/config/ServiceA.yml","source":{"greeting":"Hello from Service A"}}]}
```

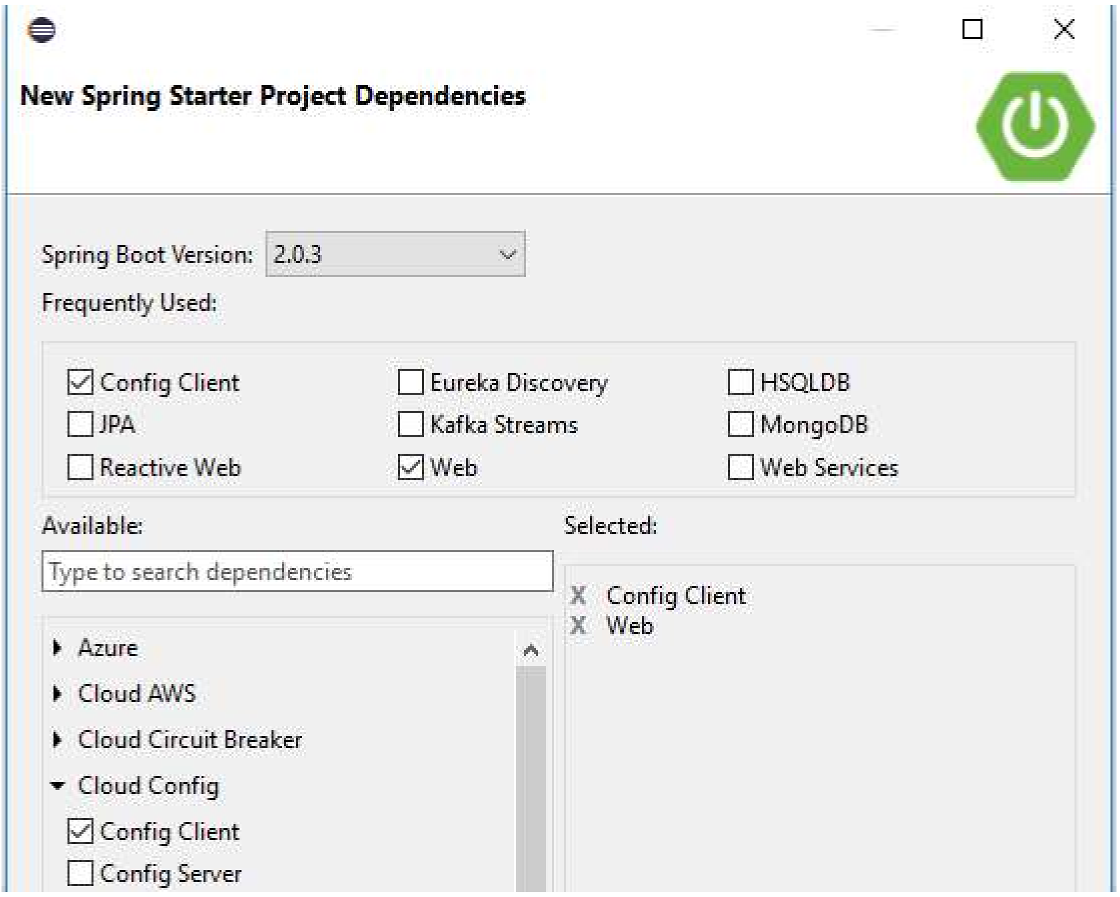


localhost:8888/ServiceB/default

```
{"name":"ServiceB","profiles":
["default"],"label":null,"version":null,"state":null,"propertySources":
[{"name":"classpath:/config/ServiceB.yml","source":{"greeting":"Hello from Service B"}}]}
```

Now we need to write ServiceA .

Create a new Spring Boot project called ServiceAApplication and give it the libraries web and Config Client



New Spring Starter Project Dependencies

Spring Boot Version: 2.0.3

Frequently Used:

☑ Config Client	☐ Eureka Discovery	☐ HSQLDB
☐ JPA	☐ Kafka Streams	☐ MongoDB
☐ Reactive Web	☑ Web	☐ Web Services

Available: Type to search dependencies

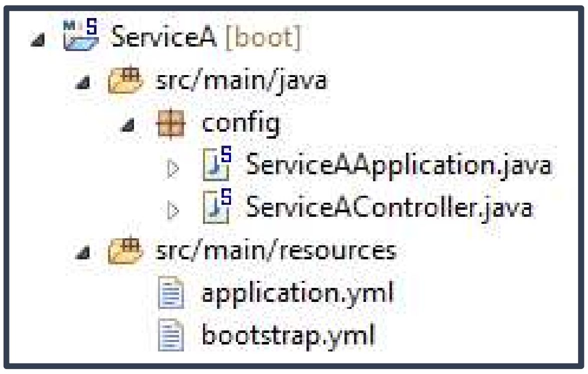
- Azure
- Cloud AWS
- Cloud Circuit Breaker
- Cloud Config
  - ☑ Config Client
  - ☐ Config Server

Selected:

- X Config Client
- X Web

Then implement ServiceAApplication as follows:

```java
@RestController
public class ServiceAController {
  @Value("${greeting}")
  private String message;
  @RequestMapping("/")
  public String getName() {
    return message;
  }
}
```



```
ServiceA [boot]
  src/main/java
    config
      ServiceAApplication.java
      ServiceAController.java
  src/main/resources
    application.yml
    bootstrap.yml
```

application.yml

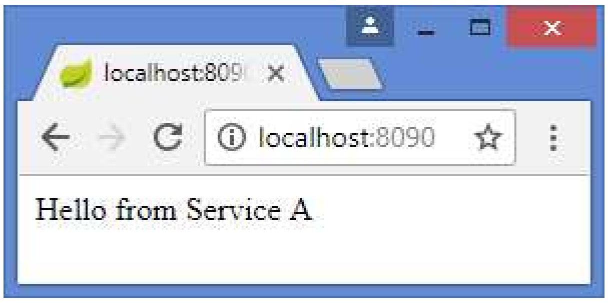
```yaml
server:
  port: 8090
```

bootstrap.yml

```yaml
spring:
  application:
    name: ServiceA
  cloud:
    config:
      url: http://localhost:8888
```

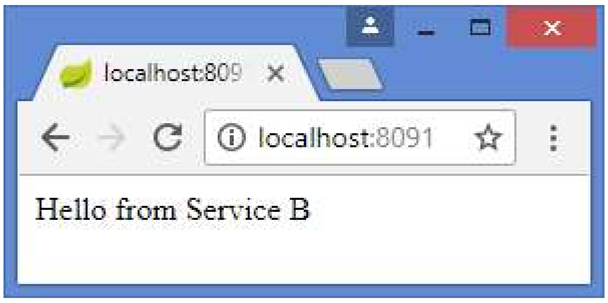
In the same way implement **ServiceBApplication**.

Then test if the applications work correctly:



localhost:8090

Hello from Service A



localhost:8091

Hello from Service B

<u>Exercise 2: Config server for our webshop</u>

Given is the solution of our webshop using components, which are basically microservices. I changed the JMS part back to REST.

Now convert all microservices to a spring cloud microservice by adding the library:

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

to all 4 services.

Then change application.properties to application.yml. Check if everything is still working.

Also give every service a bootstrap.yml file with the name of the application, and the location of the config service.

Then move some configuration from the services to the config server, and check if it still work.