Spring Cloud

* Overview
* Externalized Configuration
* Microservices
* Service Discovery
* Discovery Client
* Security
* Load Balancing
* Circuit Breaker: Resilience4j
* Zuul

Spring Cloud:

It helps in providing the tools to quickly build the distributed applications with common design patterns like:-

Service Discovery  
Discovery Client  
Circuit Breaker  
Load Balacing

Using this coordination between the distributed applications can be simplified.

With spring cloud you can simplify the development of distributed applications with simple annotations & configurations.

Spring Cloud uses the Spring Boot project to quickly develop the applications, we must use the compatible spring boot projects to work spring cloud.

Spring cloud uses many spring boot features:

* Starter Projects
* Compatible version of Spring Boot for Spring Cloud
* Creating an executable jar/war
* Property configuration
* Passing the configurations from command line at runtime
* Deploying the war on external server
* Actuators endpoints: health, metrics, beans
* MVC implementation: Service, DAO
* JPA Repository
* REST based services
* Curl commands
* Working on GIT

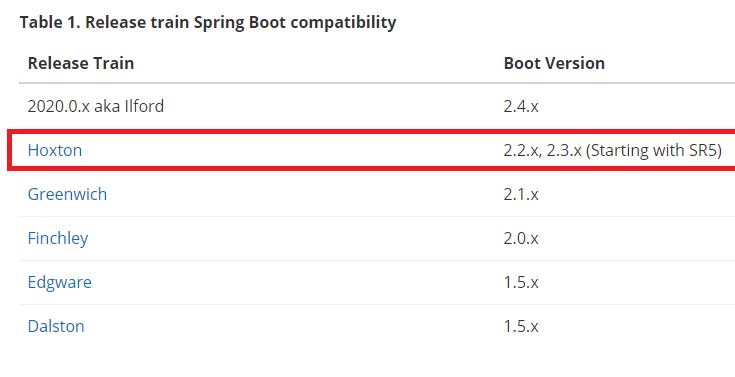
In spring cloud we are going to deal with:

* Externalized Configuration: This makes a configuration which is remotely available for multiple applications, you can apply security like Encryption by using JCE (Java Cryptographic Extension), You can also force applications to pass the credentials while accessing the remote location
* Microservices: Service Discovery, Discovery Client, Load-Balancing, we need to know some design patterns, like Development patterns, resilience pattern, routing pattern.
* Circuit Breaker: Hystrix (Deprecated), Resilience4j
* Zuul: Common door for all the incoming request to your service discovery, dynamic routing
* Secuirty

Getting Started

1. New Spring Boot project
2. Add spring cloud version to the existing project

Spring Boot compatible release train for spring cloud

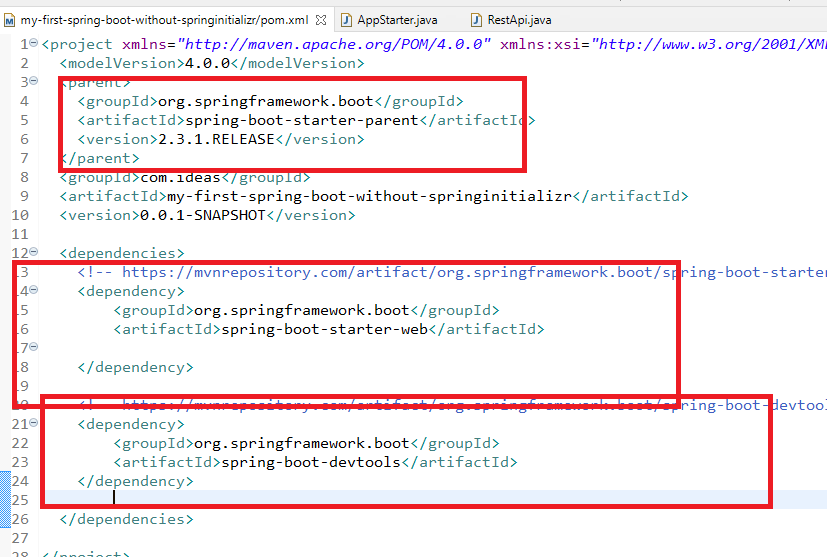


Spring Boot essentials:

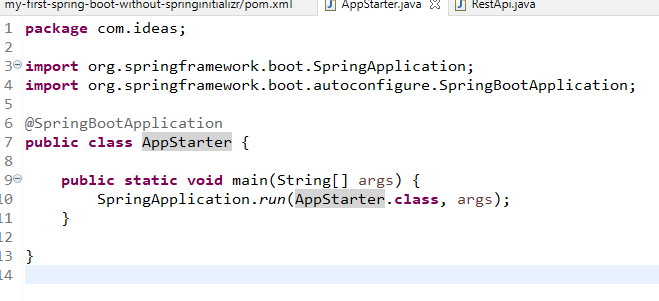
Starting from creating project to Creating rest services with JPA (in-memory database h2)

Spring Boot with Maven without spring initializr

pom.xml



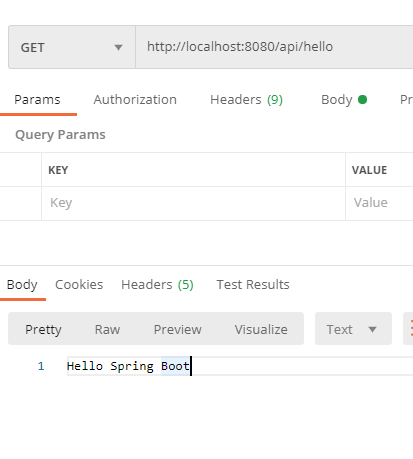
AppStarter



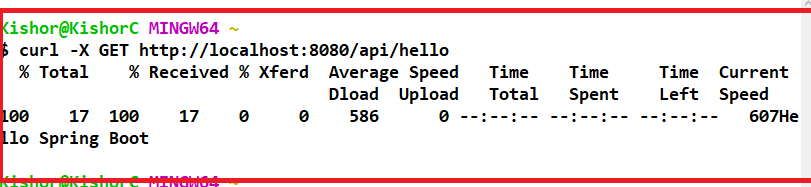
RestApi



Output:



Through cURL you can pass the GET request

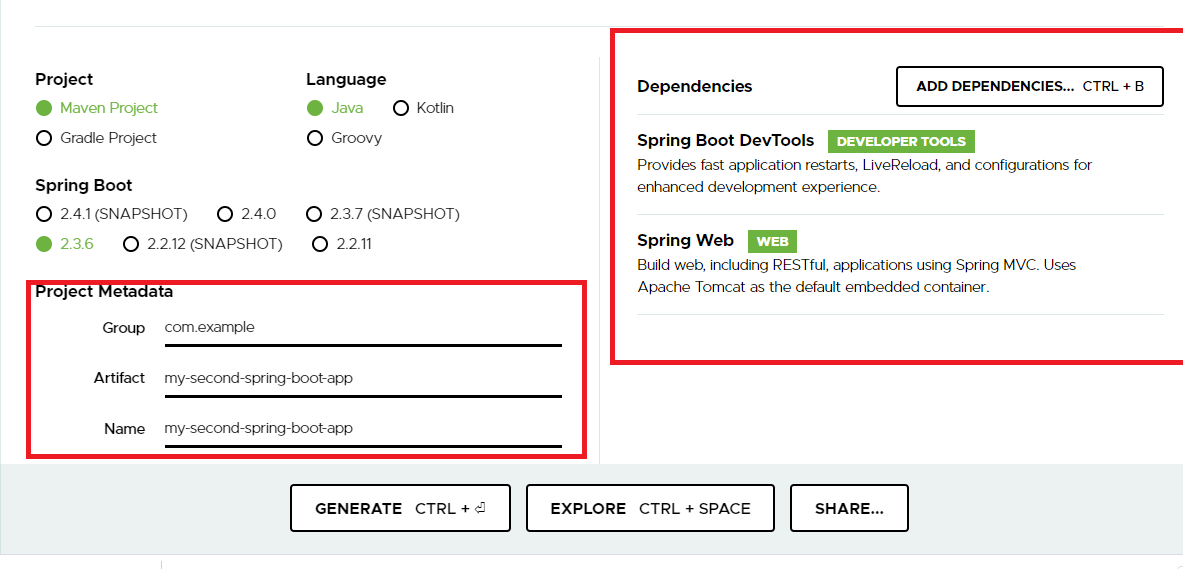


Some of the useful curl commands:

POST request with JSON data:   
curl -X POST url -H “Content-Type:application/json” -d “{...}”

PUT request:  
curl -X PUT url

Spring boot project with Spring Initializr



RestApi

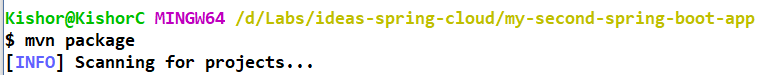


Benefits

* You will get UI based dependency configuration
* You will get pre-defined code to run/launch spring application
* You will get a plugin to create executable Jar/War
* You will get application.properties
* You will get compatible version of spring boot

How to create an executable jar file

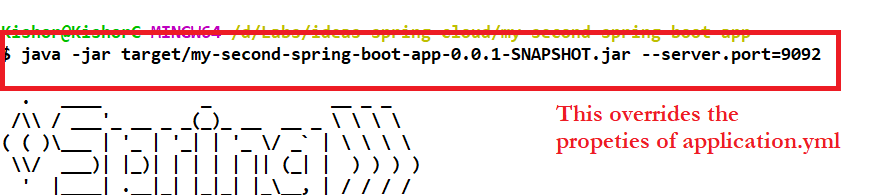
You should use *mvn package* command



You will get a jar file inside target folder

How to execute the jar file

You should use *java -jar <<file.jar>>*



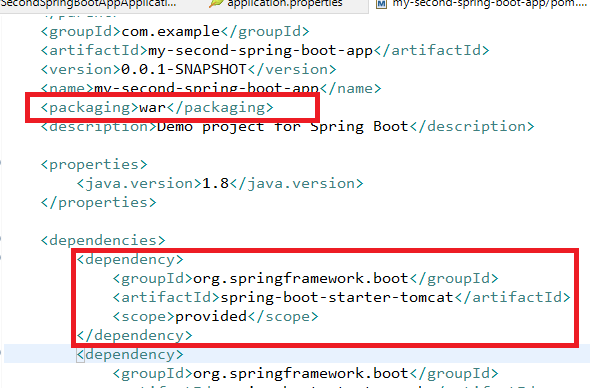
Deploying the war in an external server

* You need to perform some changes
* Main method doesn’t work hence you must use a filter called SpringBootServletInitializer which has a configure() method
* Once any class extends SpringBootServletInitializer configure() method will be executed you can write the launching of spring boot application inside the configure
* You must create WAR file instead of JAR file

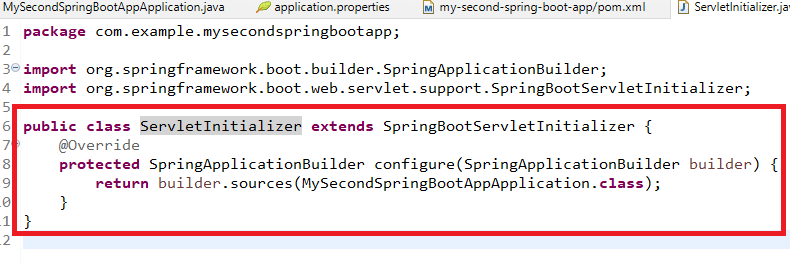
Steps:

* Change the packaging in pom.xml to war
* Embedded server must not be considered when war is built, you must use <scope>provided in pom.xml
* You must create a class that extends SpringBootServletInitializer

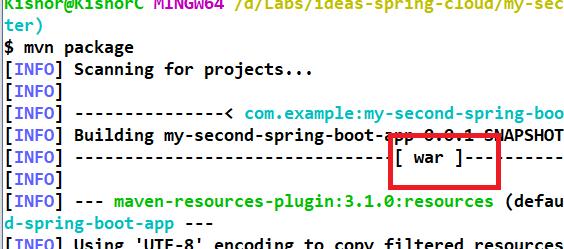
pom.xml



ServletInitializer

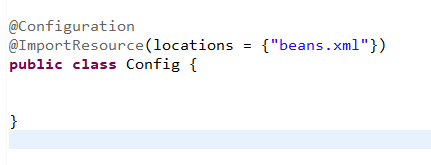


Now you can use mvn package



How to load the existing XML configuration in spring boot

You must use @Configuration class with @ImportResource annotation



Spring Boot best practices

1. Build systems - Maven, Gradle
2. Structuring your code - keep @SpringBootApplication in the root package
3. Avoid default package
4. Try to use @Configuration class instead of XML files
5. Don’t enable devtools in the production
6. Refer the migration guides when you want to migrate from old version to new version

Spring Actuator

You can monitor and manage applications in the production

you can see beans, health, metrics

health:

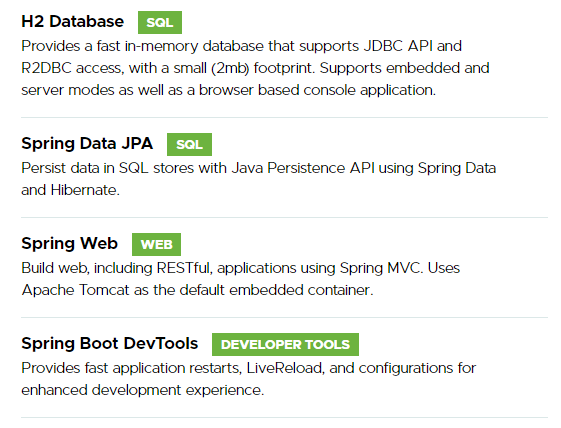
beans:

env:

metrics:

Spring Boot Starter JPA

You can use JpaRepository and CrudRepository which has inbuilt implementation as per the generics of your entity.



JpaRepository gives you lot of methods like

save() : it is used store & update both

deleteById():

findById()

findAll()

application.properties

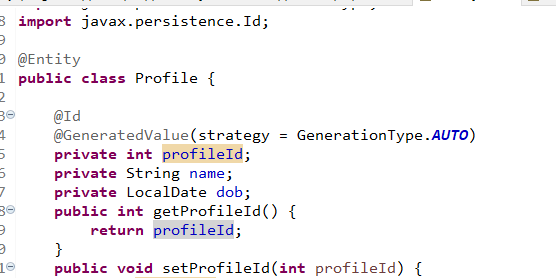
spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.url=jdbc:h2:mem:myDB

spring.datasource.username=username

spring.datasource.password=password

Profile.java



ProfileDaoRepository.java

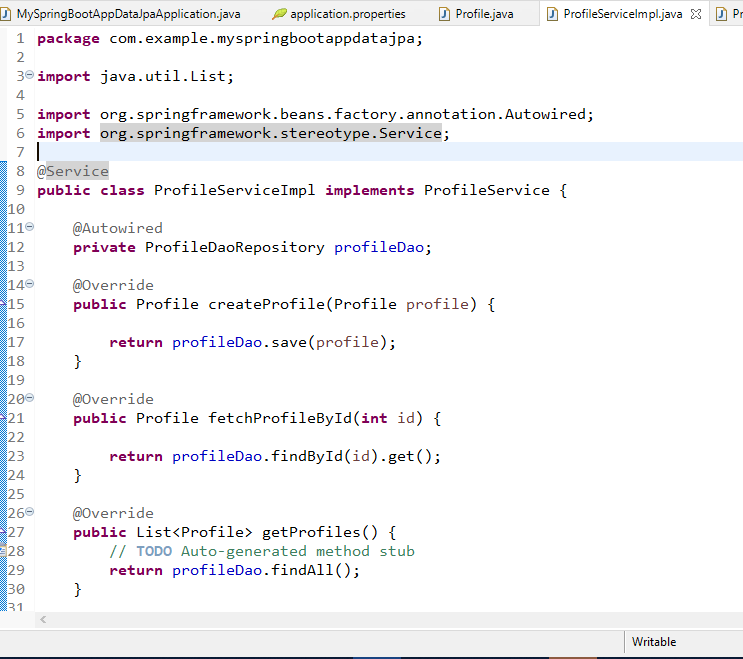
**package** com.example.myspringbootappdatajpa;

**import** org.springframework.data.jpa.repository.JpaRepository;

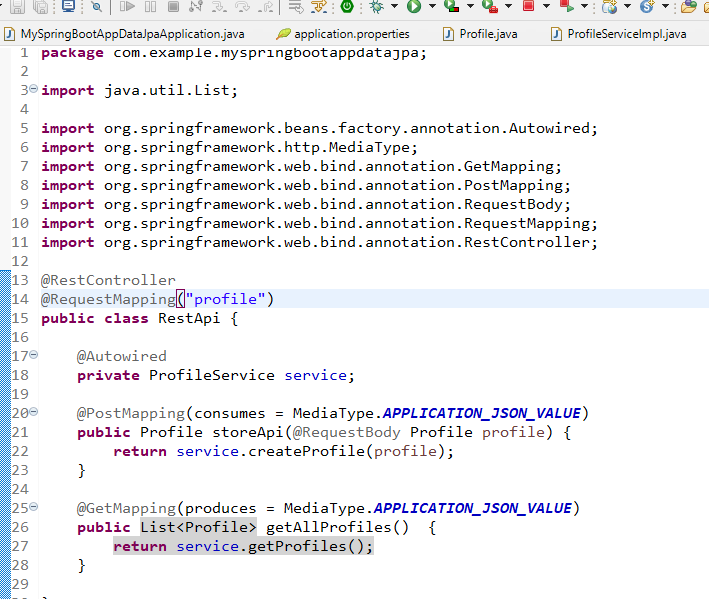
**public** **interface** ProfileDaoRepository **extends** JpaRepository<Profile, Integer> {

}

Service class



Rest



Spring Cloud main projects

Spring Cloud Config

Spring Cloud Netflix

Spring Cloud Security

Spring Cloud Configuration Server / Externalized Configuration:

Here the client & server can access the externalized configuration in a distributed system.

The externalized configuration location can be a filesystem / GIT

Features:

There are two things

1. Spring Cloud Server Features
   1. This takes care of locating the original location of the centralized repository/system
   2. Encrypt & Decrypt the sensitive information
2. Config Client Features
   1. This binds to the config server & spring environment will load the remote resource properties

You need to create 2 projects

1. Configuration Server
2. Configuration Client

Dependencies required for Configuration server:

Config-Server

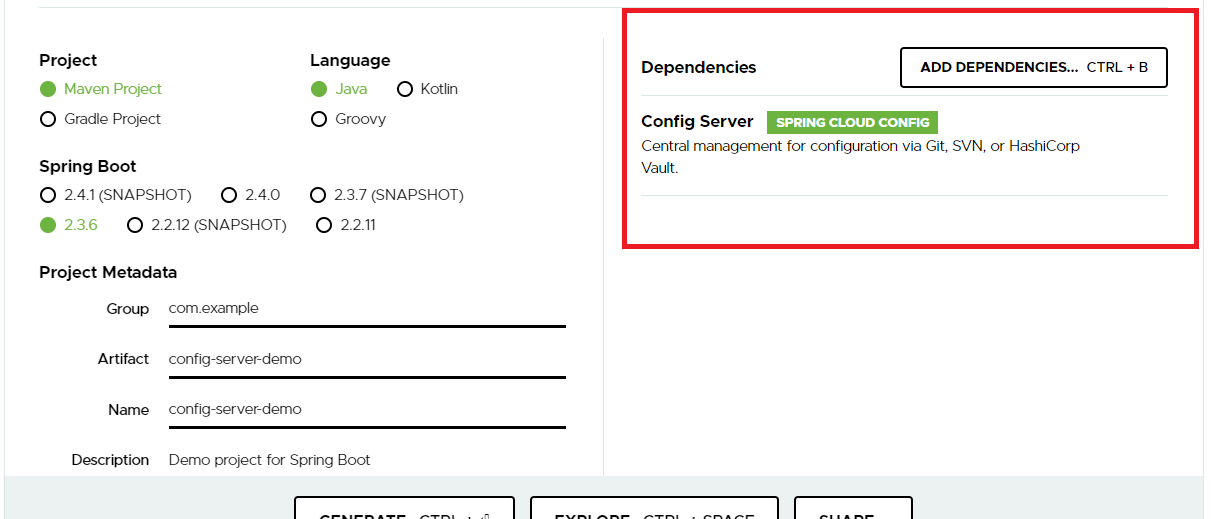
Dependencies required for Configuration client:

Web

Config-Client

(Optionals: Jpa, Devtools, Actuator)

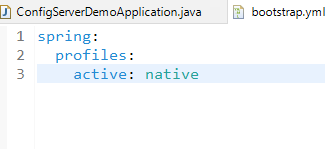
Config Server



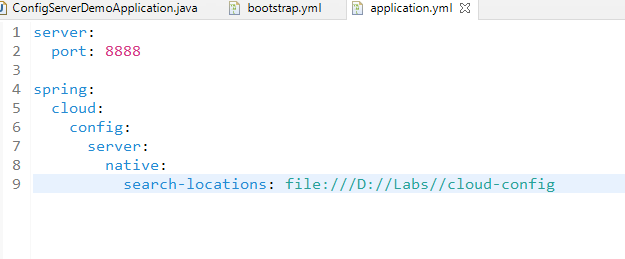
bootstrap.properties or bootstrap.yml: it is loaded before any property files, it is loaded at the time boostraping the application.

Create 2 yml files inside the configuration server project

bootstrap.yml

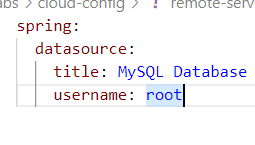


application.yml

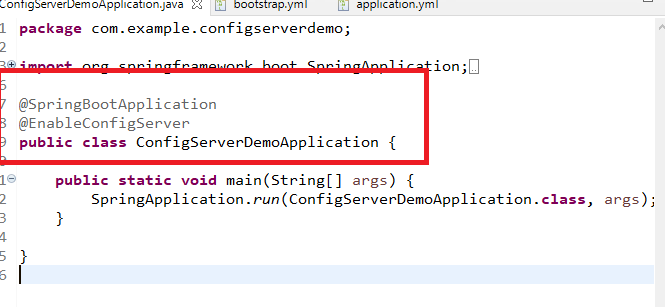


You can create one externalized configuration file in some directory.

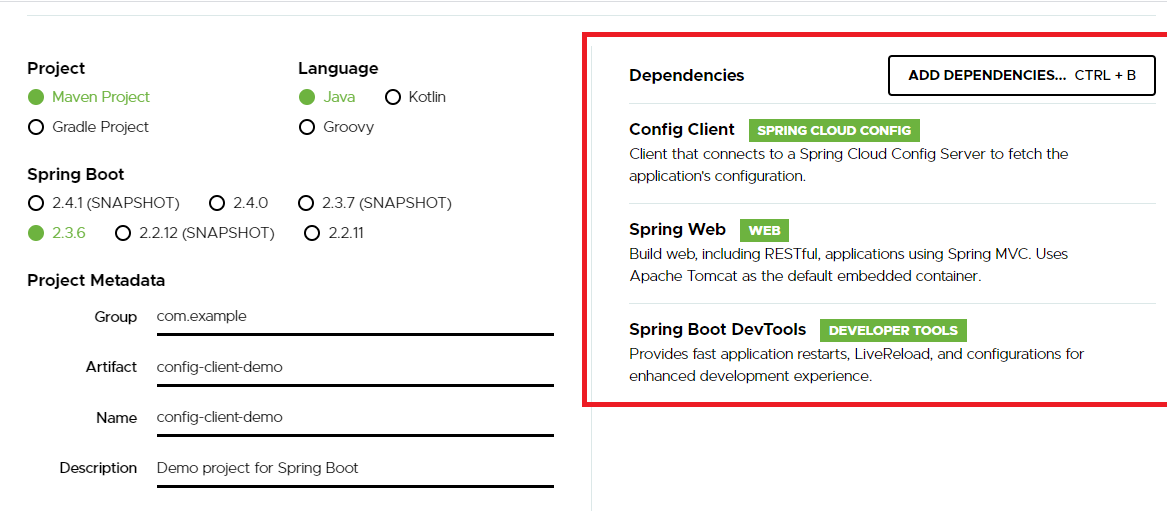
remote-server.yml



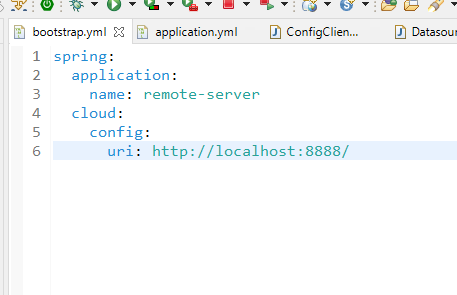
Now you can run the configuration server



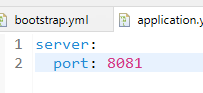
Client Program / Configuration Client



bootstrap.yml

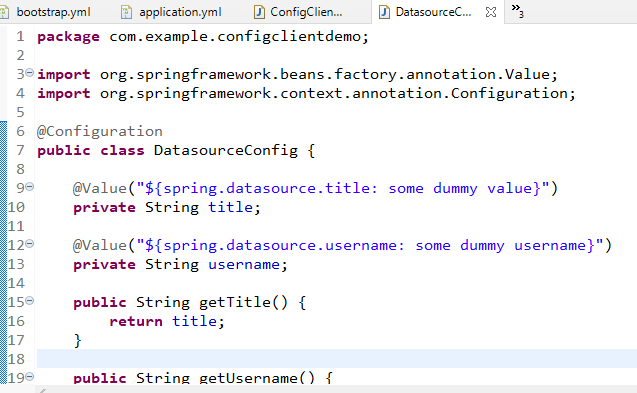


application.yml

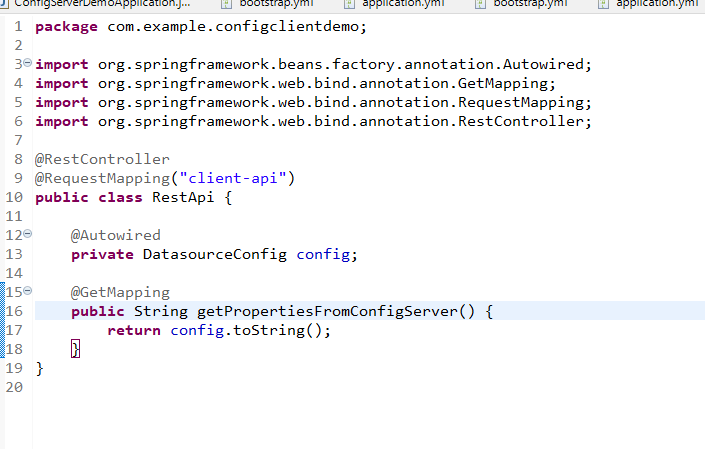


DatasourceConfig.java

Note: Here we are binding the external properties of configuration server to the DatasourceConfig variables

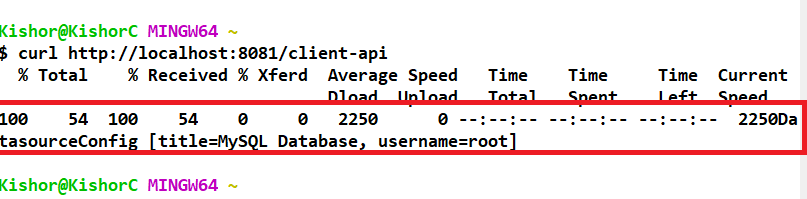


RestApi.java



Now you can launch the client application

Output:

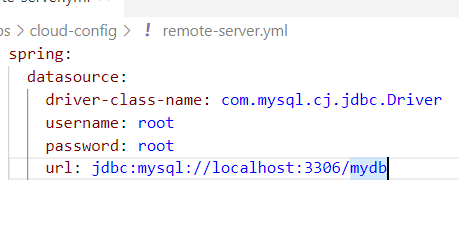


Note: If configuration server properties are modified it wouldn’t be affected to the client, because the client loads the configuration at the time startup, hence you must use some kind of endpoints given by actuator where you can get the modified configurations or you can use pub/sub mechanism through the events automatically changes will be affected

How to use the datasource information from the configuration server

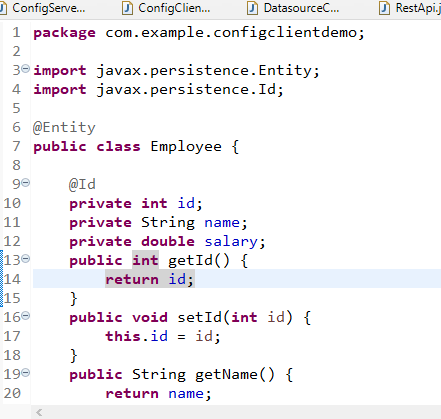
1. You must add spring-boot-starter-jpa & database drviver in the client pom.xml file
2. you must add datasource properties in the remote configuration server

remote-server.yml

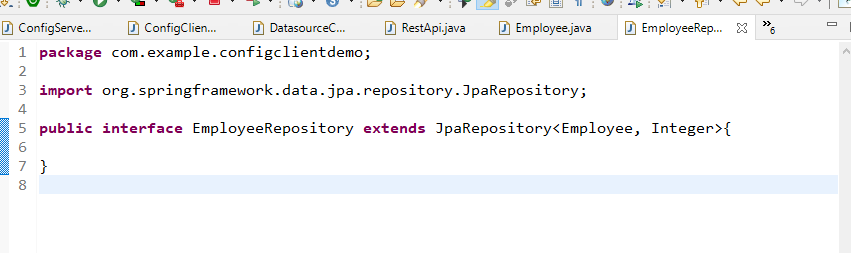


You can have Employee entity and JpaRepository interface type in the client application

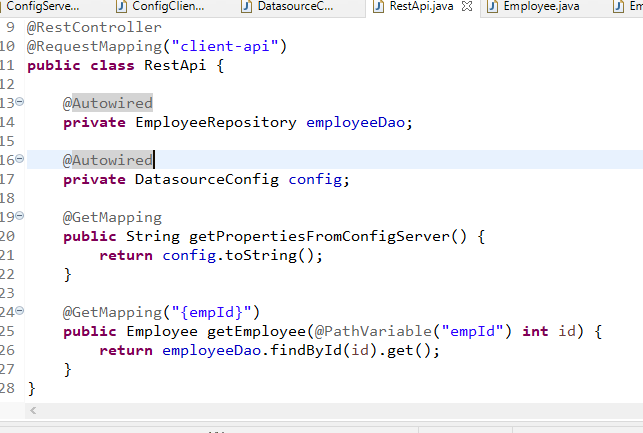
Employee.java



EmployeeRepository.java



RestApi.java



Output:

