

## Using @Value annotation

* Can access values from application.properties file by key
* Here the my.fullGreetDesc is injected into greetMsg
* Code
* *@RestController*
* public class Controller {
* *@Value*("${my.fullGreetDesc}")
* private String greetMsg;
* *@GetMapping*("/greet")
* public String getGreeting() {
* return greetMsg;
* }
* }

## Tricks while using @Value

1. If suppose key -value is not present in application.properties the which try to access using @Value, then in that case application run into error.
   * To avoid that @Value(“${my.greet:default value}”) => in this it will print default value instead of error
2. Suppose we got list as comma separated in app.prop file

my.list=one, two, three

In order to access as List<String> directly we can make it as

@Value(my.list)

List<String> list; //=> one,two, three populate as list

## Using @ConfigurationProperties

Suppose I want get value of key which associate with db from app.properties

db.connection={connectionString:'http://....', userName:'foo', password:'pass'}

db.host=127.0.0.1

db.port=1200

*@Configuration*

*@ConfigurationProperties*("db")

public class DbConnection {

private String connection;

private String host;

private int port;

public String getConnection() {

return connection;

}

public void setConnection(String connection) {

this.connection = connection;

}

public String getHost() {

return host;

}

public void setHost(String host) {

this.host = host;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

*@Autowired*

private DbConnection dbConnection;

*@GetMapping*("/db")

public String getDbDetails() {

return dbConnection.getConnection()+dbConnection.getHost()+dbConnection.getPort();

}

## Running jar file from cmd line

* mvn -v => check if maven installed
* mvn install
  + in target folder jar file gets created
* java -jar <name of jar file>

## **Externalize the properties file**

* Scenario
  + Suppose we create a jar file of application, we no longer able access application.properties file of application
  + In that case what we create a property file at target file. Now we can change value of any key externally and the application automatically pick the value from external property file instead of one which is within jar file
  + We can also pass value to key through command line as well
* In target folder run
  + Method 1
    - *vi application.properties* => in linux
    - *notepad application.properties* => in windows
    - it create application.properties file and opens it
    - Now if we run jar file it detects application.properties file from target file and use it in application instead of one which id within application
  + Method 2
    - Pass it through cmd itself
    - *java -jar <jar file name> --<key>={value we want to pass}*
      * ex :- java -jar spring-boot-config-0.0.1-SNAPSHOT.jar --my.fullGreetDesc="from cmd"

\*\* **What if we use both methods together?**

* It will first search for application.properties file from jar, them from target file. If it is there it overrides. And further if it is passed through cmd it overrides with cmd value.

## Using YAML file

* Useful when we nesting
* Extension is .yml
* Replace = with : => if we want to replace .properties with .yml
* Tab is not allowed, instead use space
* Comparison

spring.application.name: spring-boot-config

my:

greeting: Hello again

fullGreetDesc: Description:${my.greeting}

list:

values: one,two,three

dbValues: {connectionString:'http://....', userName:'foo', password:'pass'}

db:

connection: "{connectionString:'http://....', userName:'foo', password:'pass'}"

host: 127.0.0.1

port: 1200

#----------yml without nesting-------------------

#spring.application.name: spring-boot-config

#my.greeting: Hello again

#my.fullGreetDesc: Description:${my.greeting}

#my.list.values: one,two,three

#dbValues: "{connectionString:'http://....', userName:'foo', password:'pass'}"

#db.connection: {connectionString:'http://....', userName:'foo', password:'pass'}

#db.host: 127.0.0.1

#db.port: 1200

#----------application.properties---------------------

#spring.application.name: spring-boot-config

#my.greeting: Hello again

#my.fullGreetDesc: Description:${my.greeting}

#my.list.values: one,two,three

#dbValues: {connectionString:'http://....', userName:'foo', password:'pass'}

#db.connection: {connectionString:'http://....', userName:'foo', password:'pass'}

#db.host: 127.0.0.1

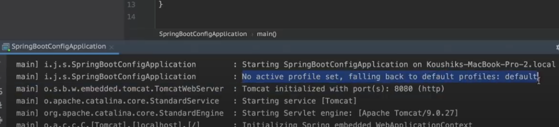
#db.port: 1200

**## Environment specific**

* So far we are able to achieve the externalise the property
* But now want to make it environment specific i.e. in dev environment it should pick application.extn file of dev, similarly for QA and production environment



* This becomes possible with **Spring Profiles**
* **Spring profile**
  + If we run our application without specifying profile it run with default. In case of default it takes values from application.properties



* + Create file with application-<profileName>.extn
    - Ex:- application-test.extn
  + In application.extn file mention the profileName
    - Spring.profiles.active: test
    - Can mention for multiple profiles
      * spring:

profiles:

active:

- test

- production

* + What if we want run .jar of specific environment after creating jar file
    - Run :- *java –jar <.jar> --spring.profiles.active=test*
      * Ex:- *java -jar spring-boot-config-0.0.1-SNAPSHOT.jar --spring.profiles.active=production*

(it overrides value from app.properties file)

* Selecting beans by profile



* **Environment object**

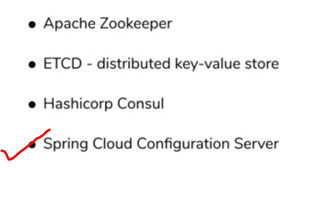
**## Spring cloud config server**

* Why we should use it for microservices ?

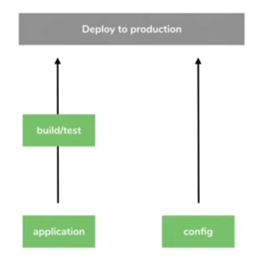
=> The earlier approach was fine when we were using single service. But not suitable with microservices. Not **Consistent**

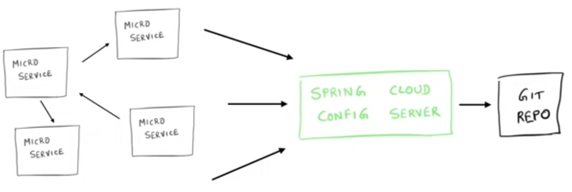


* + For each service we need to make change every time if we wish to change any property which is common to all. **Not consistent**
* Options to resolve the consistency



* Architecture
  + Scenario – If suppose we made change in config and don’t have other changes. So in this case we might need to redeploy our app.
    - So how can we resolve?
      * Why don’t we have that file git repo? So that app will look for config file in git repo so that we need not need to redeploy.





## Setting up spring cloud configuration server

* Create a new project
* Dependencies
  + Config server
* In main add @EnableConfigServer
* Go to application.properties
  + spring.cloud.config.server.git.uri=file:///D:/Common\_data/Microservice\_tut/javaBrains/Microservice\_javabrains/Step03/Config\_file
  + server.port=8888
* Run it on local:8888/application/default => details of application

## Setting up spring cloud config client