

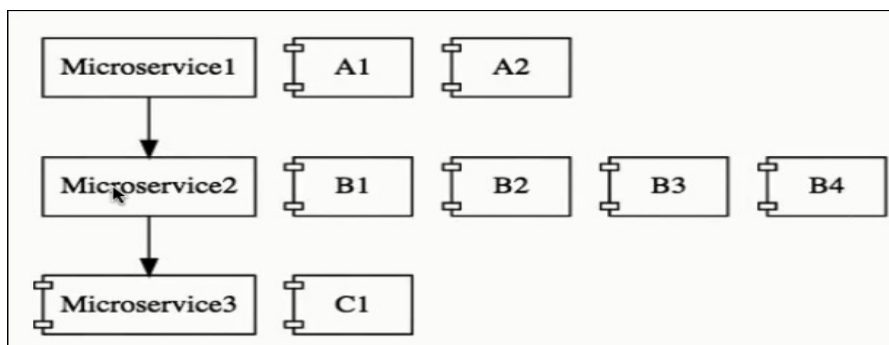
Lecture 50 – How to use the course



Lecture 51 – Introduction to Micro services.

MICROSERVICES

- REST
- & Small Well Chosen Deployable Units
- & Cloud Enabled



*There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies -
James Lewis and Martin Fowler*

Lecture 52: Challenges with MicroServices

1. BOUNDED CONTEXT – How to separate boundary for each service
2. CONFIGURATION MGMT- how to keep different env & respective variables separate
3. DYNAMIC SCALING(scale up & down) – ramp up/down & load balance between available instance .
4. VISIBILITY- Debugging . If there is a but where is it , by tracing the request propagation we can know the micro service which is having issue.
Monitoring the micro services.
5. PACK OF CARDS – Critical service going down will create ripple effect & crash entire system.
Thus fault tolerance is required in your micro services.

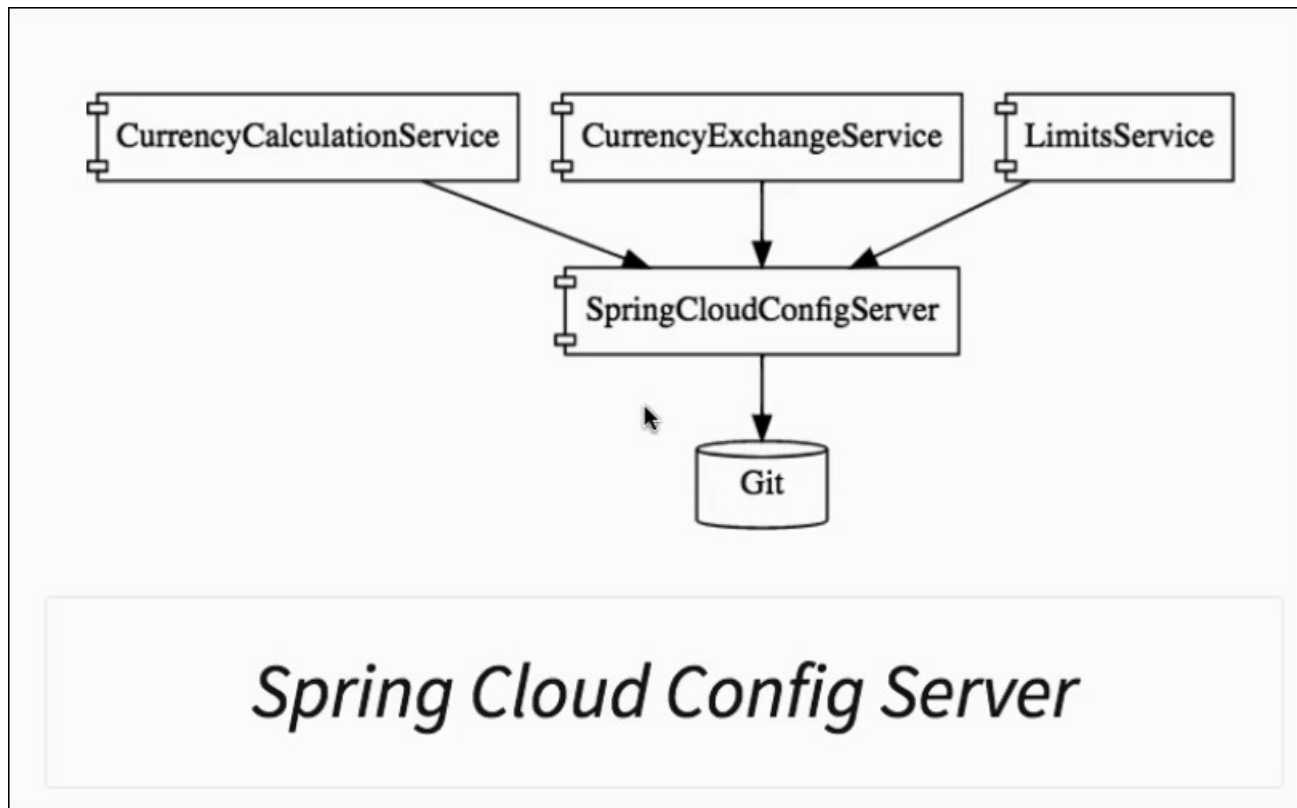
Lecture 53: Introduction to Spring Cloud

Ref: <https://projects.spring.io/spring-cloud/>

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Its not one framework , its a conglomerate of various frameworks to provide cloude based solution.

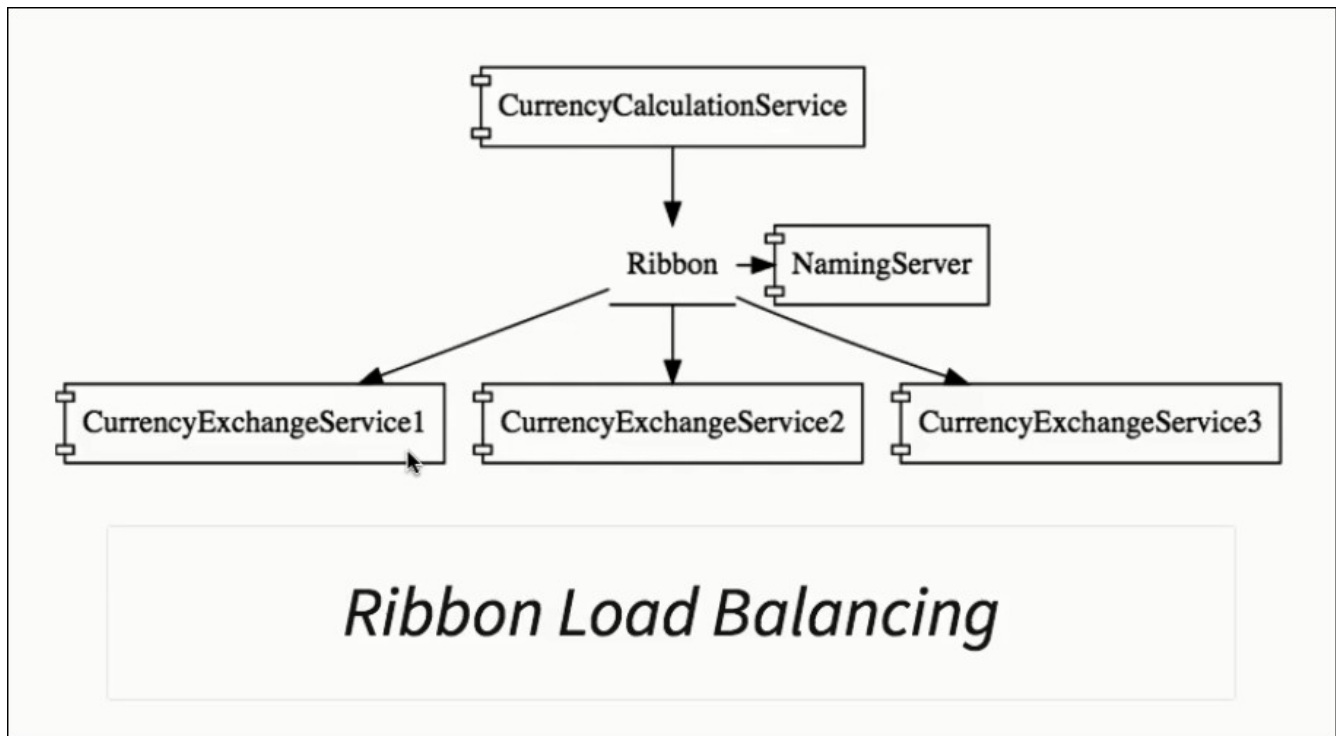
Spring Cloud Config Server :Configuration of different services of the distributed system in one centralized location on Git & SpringCloud helps to expose these configs to various service centrally



Dynamic Scaleup & down:

DYNAMIC SCALE UP AND DOWN

- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)



Visibility & Monitoring: Health/logs/console

VISIBILITY AND MONITORING

- Zipkin Distributed Tracing
- Netflix API Gateway (ZUVL)

Robustness: Make system tolerate fault

FAULT TOLERANCE

- Hystrix

Lecture 54: Micro services Advantages:

1. ADAPTION OF NEW TECHNOLOGY
2. DYNAMIC SCALING
3. FASTER RELEASE CYCLES

Lecture 55: USE A STANDARDISED SET OF PORTS as described below:

Ports

Application	Port
Limits Service	8080, 8081, ...
Spring Cloud Config Server	8888
Currency Exchange Service	8000, 8001, 8002, ..
Currency Conversion Service	8100, 8101, 8102, ...
Netflix Eureka Naming Server	8761
Netflix Zuul API Gateway Server	8765
Zipkin Distributed Tracing Server	9411

URLs

Application	URL
Limits Service	http://localhost:8080/limits POST -> http://localhost:8080/application/refresh
Spring Cloud Config Server	http://localhost:8888/limits-service/default http://localhost:8888/limits-service/dev
Currency Converter Service - Direct Call	http://localhost:8100/currency-converter/from/USD/to/INR/quantity/10
Currency Converter Service - Feign	http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000
Currency Exchange Service	http://localhost:8000/currency-exchange/from/EUR/to/INR http://localhost:8001/currency-exchange/from/USD/to/INR
Eureka	http://localhost:8761/
Zuul - Currency Exchange & Exchange Services	http://localhost:8765/currency-exchange-service/currency-exchange/from/EUR/to/INR http://localhost:8765/currency-conversion-service/currency-converter-feign/from/USD/to/INR/quantity/10
Zipkin	http://localhost:9411/zipkin/
Spring Cloud Bus Refresh	http://localhost:8080/bus/refresh

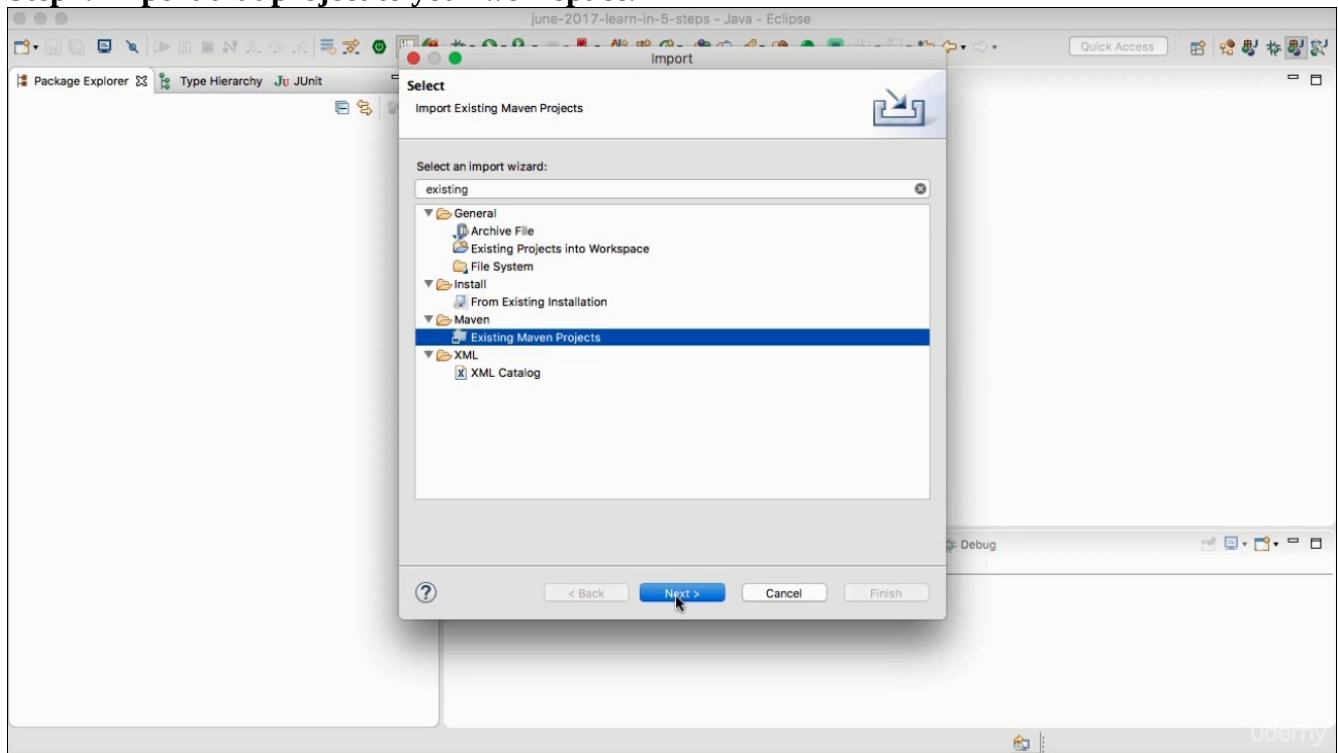
Lecture 56,57,58,59,60: Limits Service & Spring Cloud Config Server setup

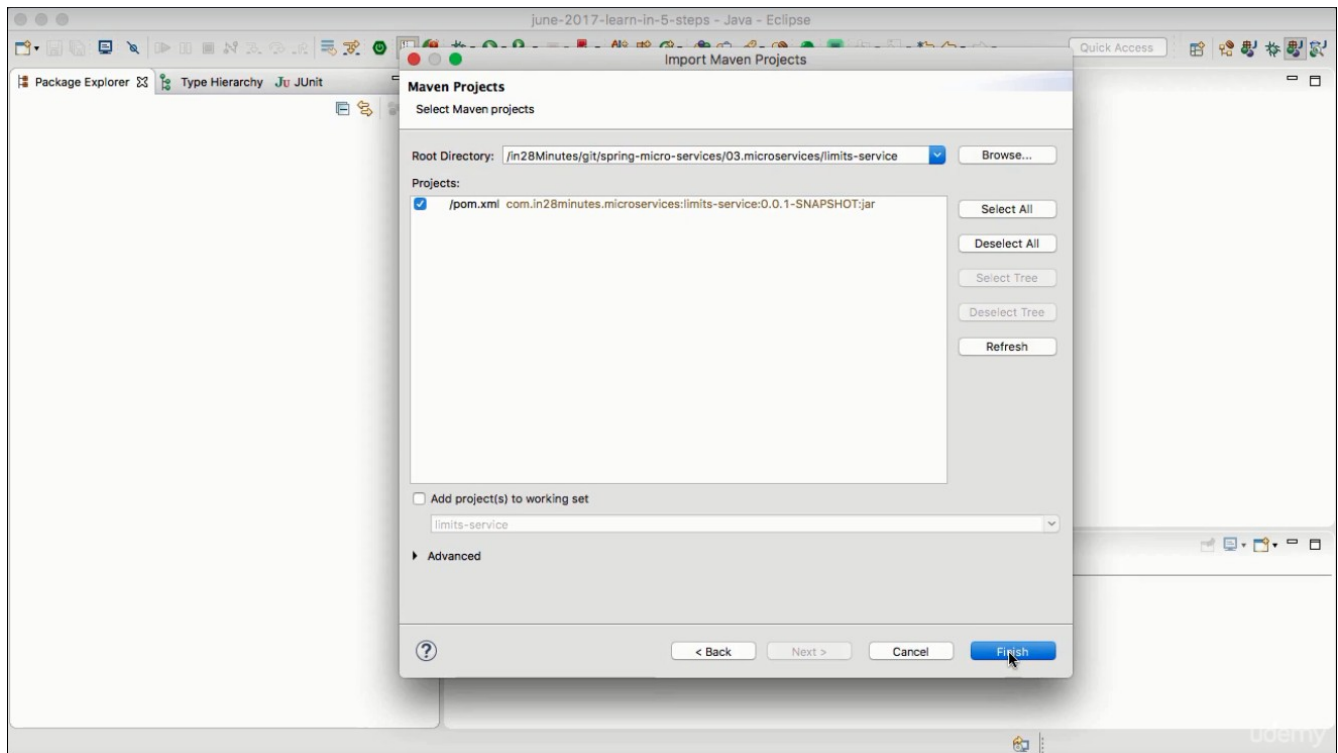
Action1: Create Limit Service

Step1: enter the below configuration & hit **Generate Project**

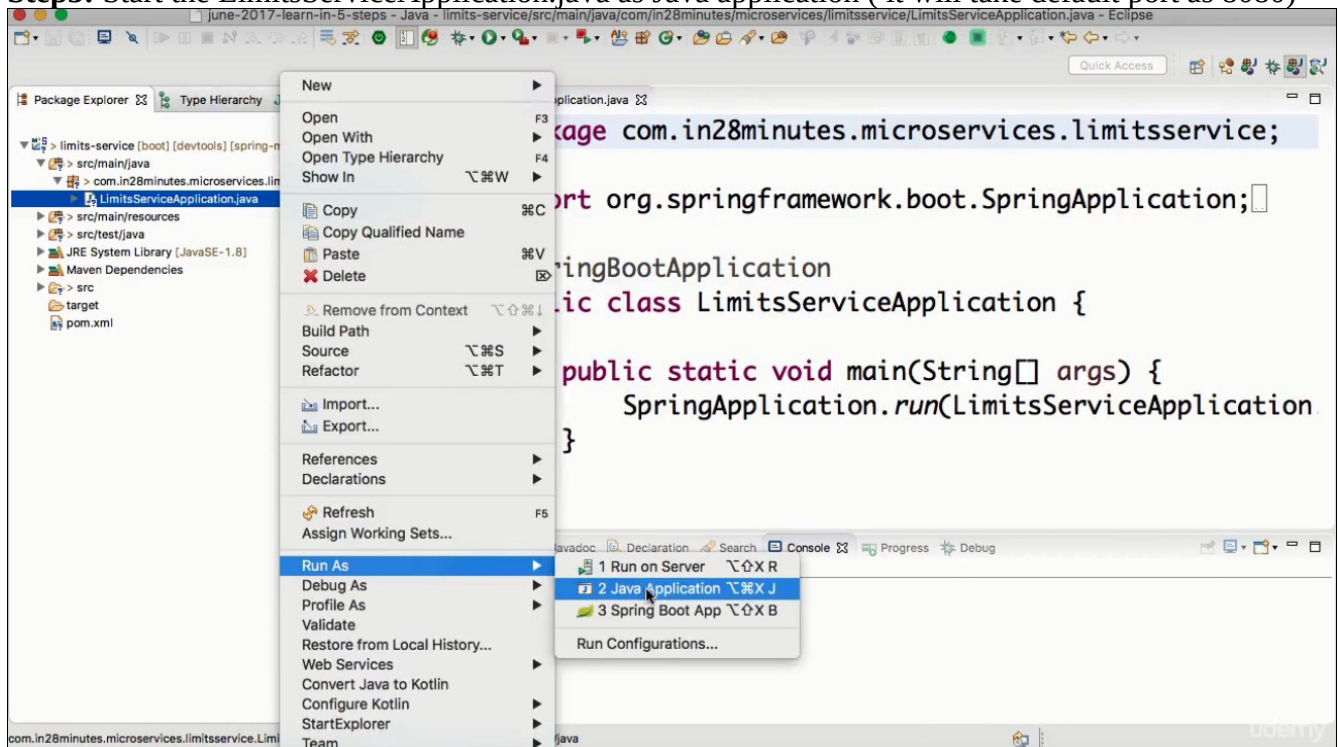
The screenshot shows the Spring Initializr web application at start.spring.io. The page is titled "SPRING INITIALIZR bootstrap your application now". The main heading is "Generate a Maven Project with Java and Spring Boot 2.0.0 M3". The "Project Metadata" section has "Group" set to `com.in28minutes.microservices` and "Artifact" set to `limits-service`. The "Dependencies" section has "Search for dependencies" set to "Web, Security, JPA, Actuator, Devtools...". The "Selected Dependencies" section shows "Web", "DevTools", "Actuator", and "Config Client". A green "Generate Project" button is at the bottom. A footer note says "start.spring.io is powered by Spring Initializr and Pivotal Web Services".

Step2: Import that project to your workspace.



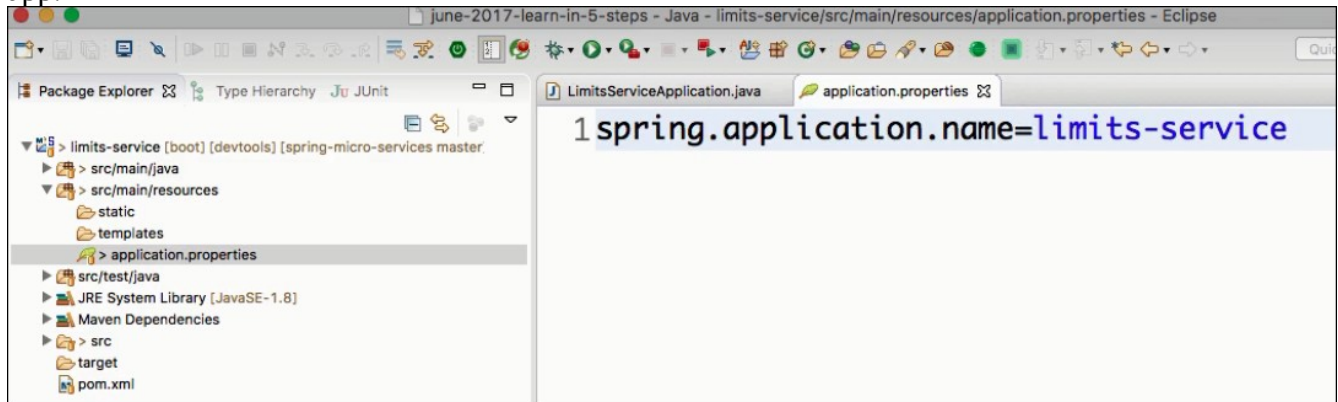


Step3: Start the LimitsServiceApplication.java as Java application (it will take default port as 8080)

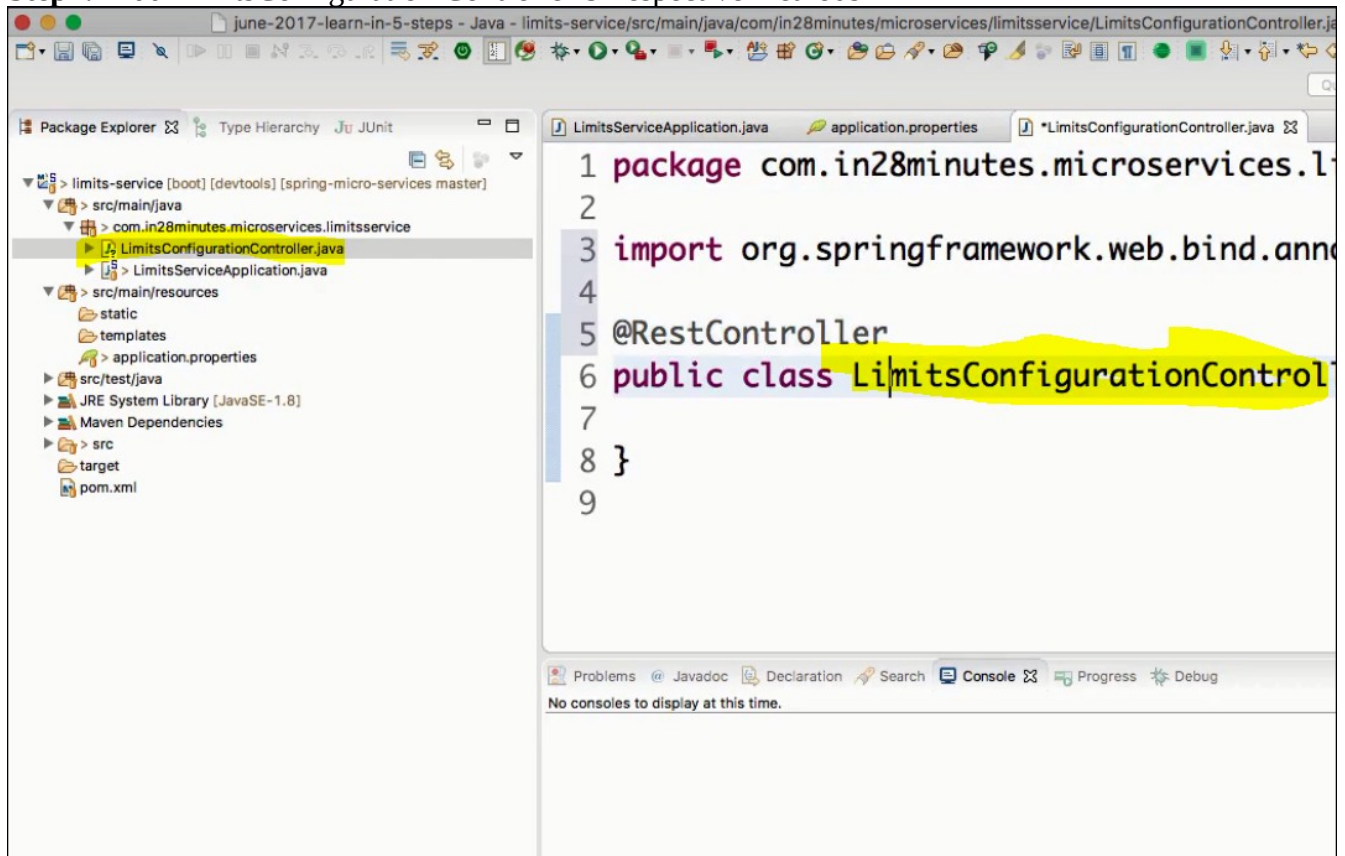


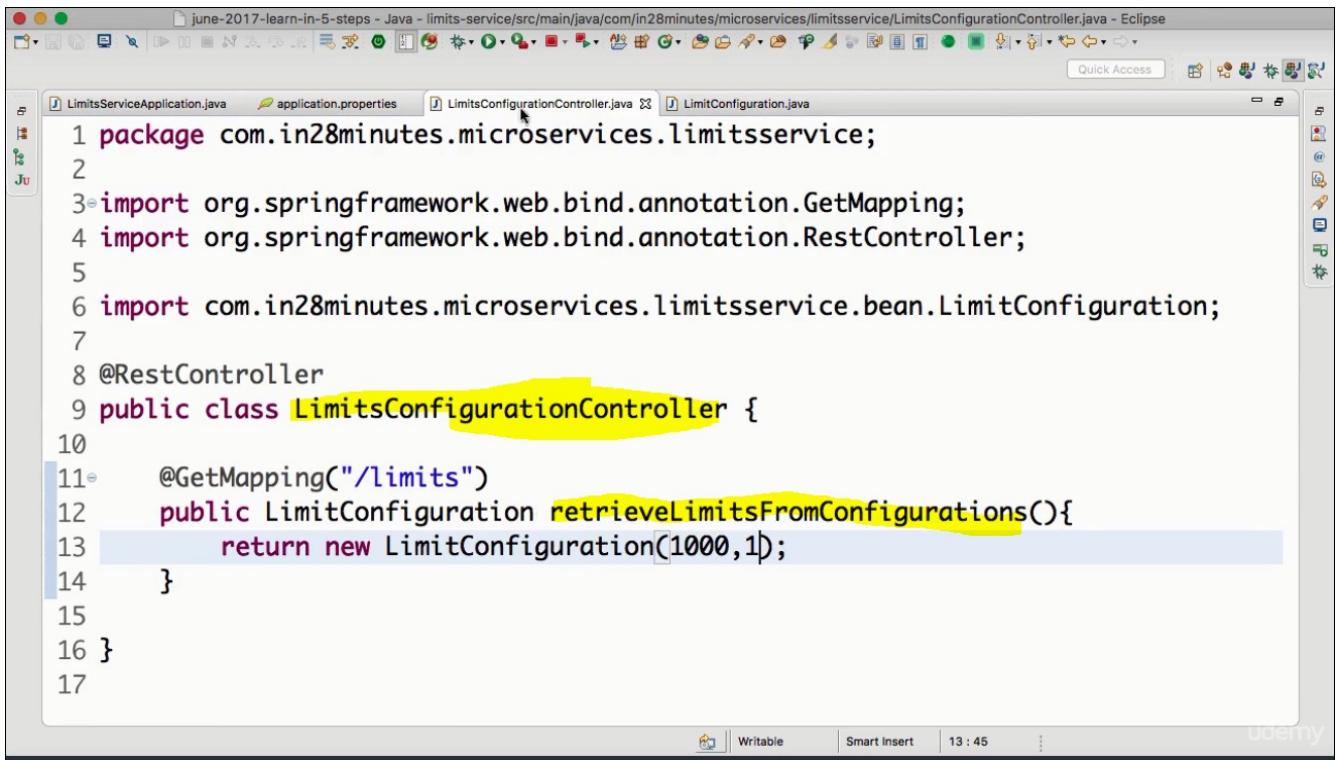
Approach1: get the limit service by hardcoding it

Step1: Now let's add application name to the **application.properties**. This will help in identifying the app.



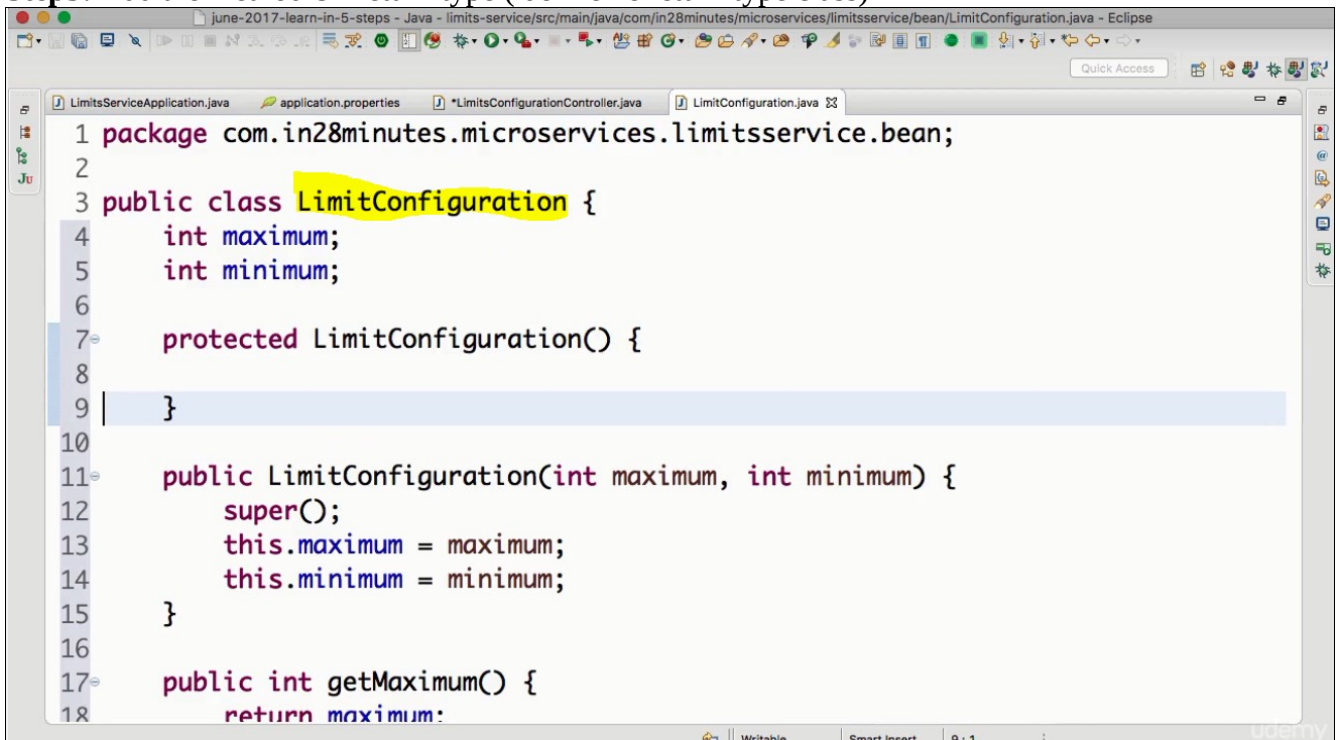
Step2: Add LimitsConfiguration Controller & respective methods





```
1 package com.in28minutes.microservices.limitsservice;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 import com.in28minutes.microservices.limitsservice.bean.LimitConfiguration;
7
8 @RestController
9 public class LimitsConfigurationController {
10
11     @GetMapping("/limits")
12     public LimitConfiguration retrieveLimitsFromConfigurations(){
13         return new LimitConfiguration(1000,1);
14     }
15
16 }
17
```

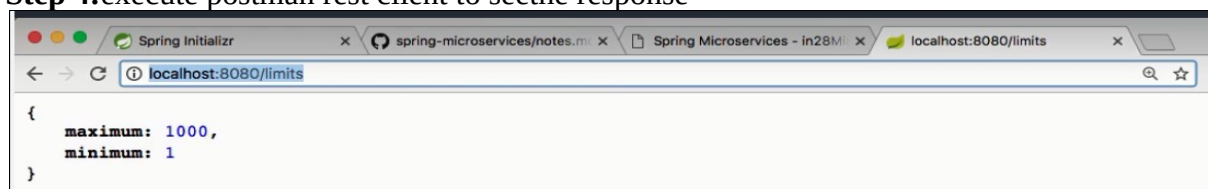
Step3: Add the method & Return type (define the return type class)



```
1 package com.in28minutes.microservices.limitsservice.bean;
2
3 public class LimitConfiguration {
4     int maximum;
5     int minimum;
6
7     protected LimitConfiguration() {
8
9     }
10
11     public LimitConfiguration(int maximum, int minimum) {
12         super();
13         this.maximum = maximum;
14         this.minimum = minimum;
15     }
16
17     public int getMaximum() {
18         return maximum;
19     }
20 }

```

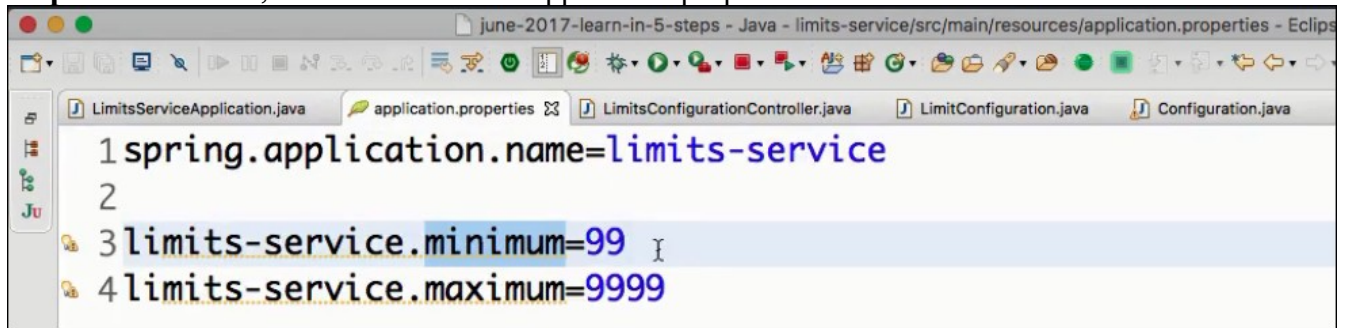
Step 4: execute postman rest client to see the response



```
{
  maximum: 1000,
  minimum: 1
}
```

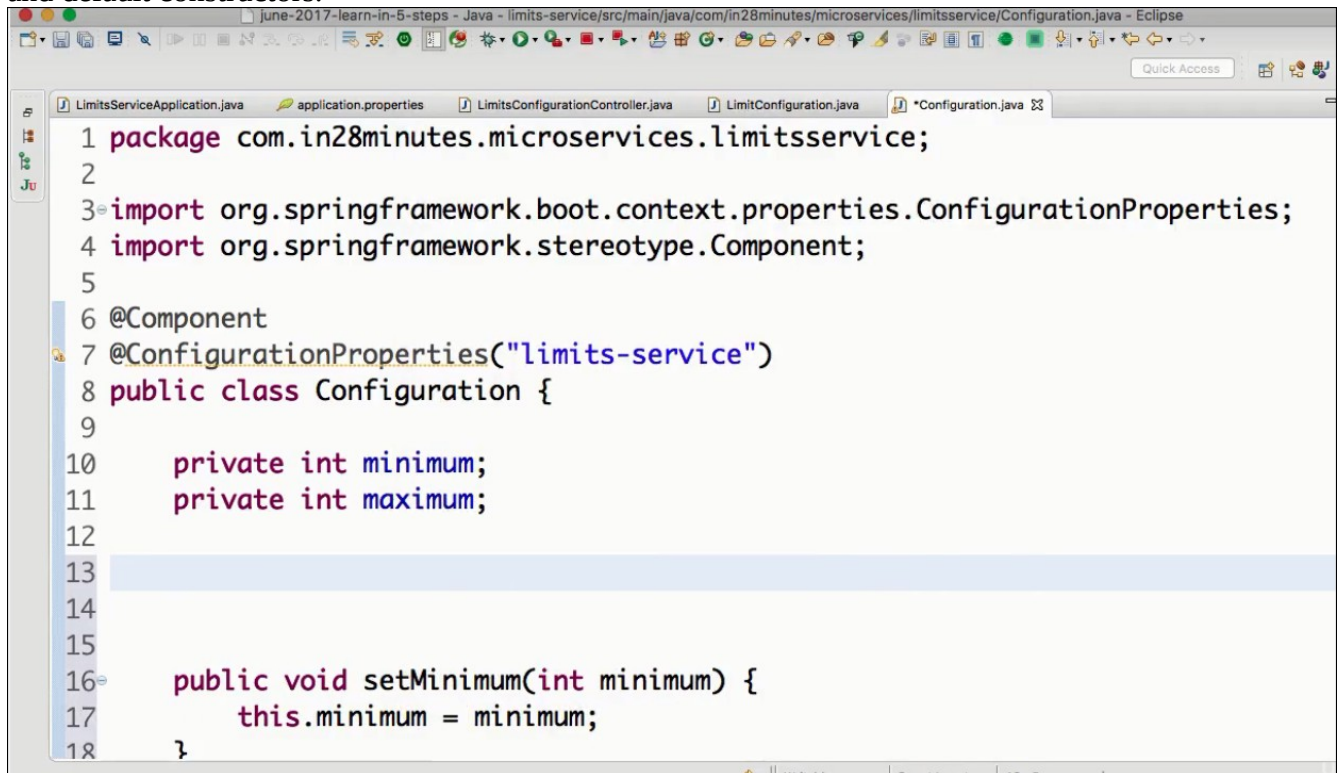
Approach 2: Get the value from application.properties file

Step1: Add the min, max limits value in application.properties file

A screenshot of the Eclipse IDE showing the 'application.properties' file. The file contains four lines of configuration: 1. 'spring.application.name=limits-service', 2. an empty line, 3. 'limits-service.minimum=99', and 4. 'limits-service.maximum=9999'. The third line is currently selected with the mouse cursor at the end of the value '99'.

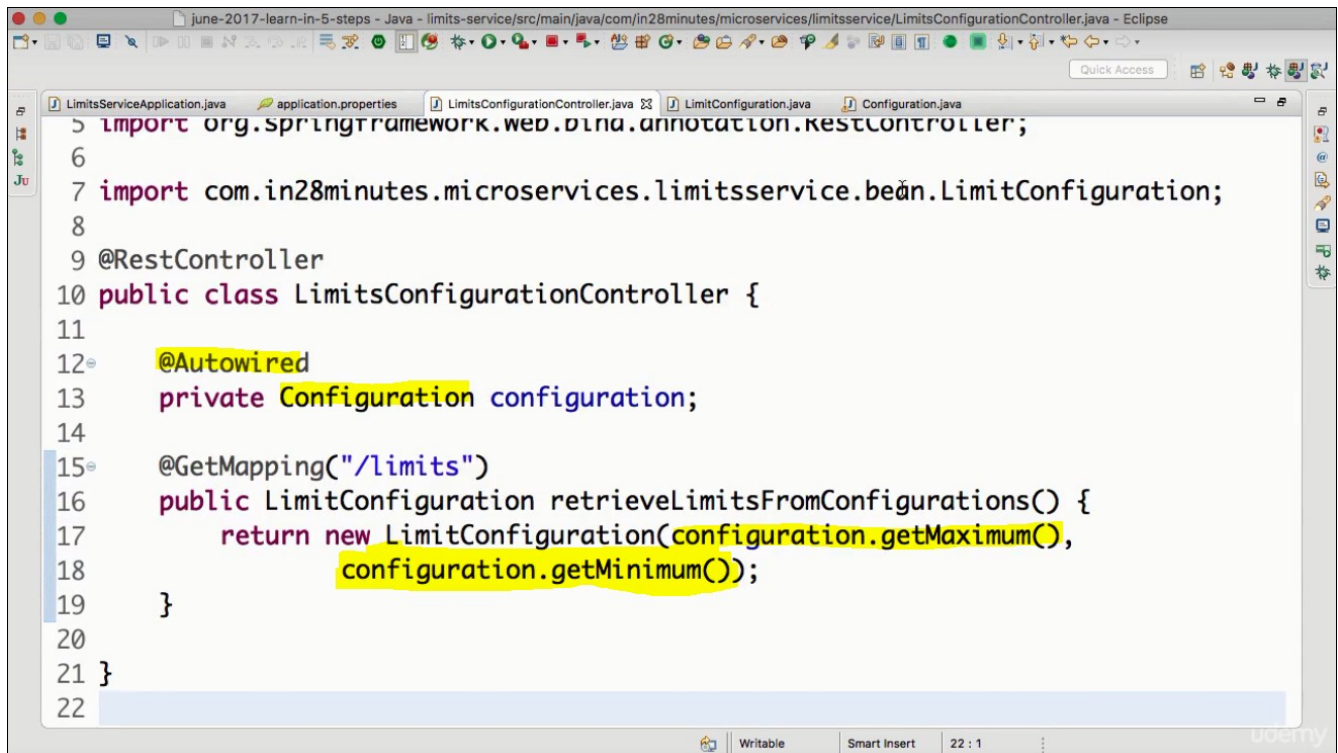
```
1 spring.application.name=limits-service
2
3 limits-service.minimum=99
4 limits-service.maximum=9999
```

Step2: Add the class **Configuration** (to read the values from properties). Do generate getters/setters and default constructors.

A screenshot of the Eclipse IDE showing the 'Configuration.java' file. The code defines a Spring configuration class. It includes package and import statements for Spring Boot's ConfigurationProperties and Component. The class is annotated with @Component and @ConfigurationProperties("limits-service"). It has two private integer fields, 'minimum' and 'maximum', and a public void method 'setMinimum' that updates the 'minimum' field.

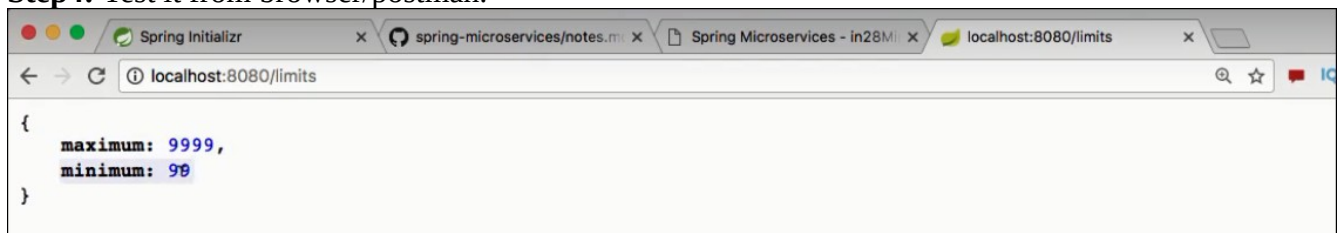
```
1 package com.in28minutes.microservices.limitsservice;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 @ConfigurationProperties("limits-service")
8 public class Configuration {
9
10     private int minimum;
11     private int maximum;
12
13
14
15
16     public void setMinimum(int minimum) {
17         this.minimum = minimum;
18     }
19 }
```

Step3: Autowire this class **Configuration** in **LimitServiceController**



```
1 > import org.springframework.web.bind.annotation.RestController;
2
3
4
5
6
7 import com.in28minutes.microservices.limitsservice.bean.LimitConfiguration;
8
9 @RestController
10 public class LimitsConfigurationController {
11
12     @Autowired
13     private Configuration configuration;
14
15     @GetMapping("/limits")
16     public LimitConfiguration retrieveLimitsFromConfigurations() {
17         return new LimitConfiguration(configuration.getMaximum(),
18             configuration.getMinimum());
19     }
20
21 }
22
```

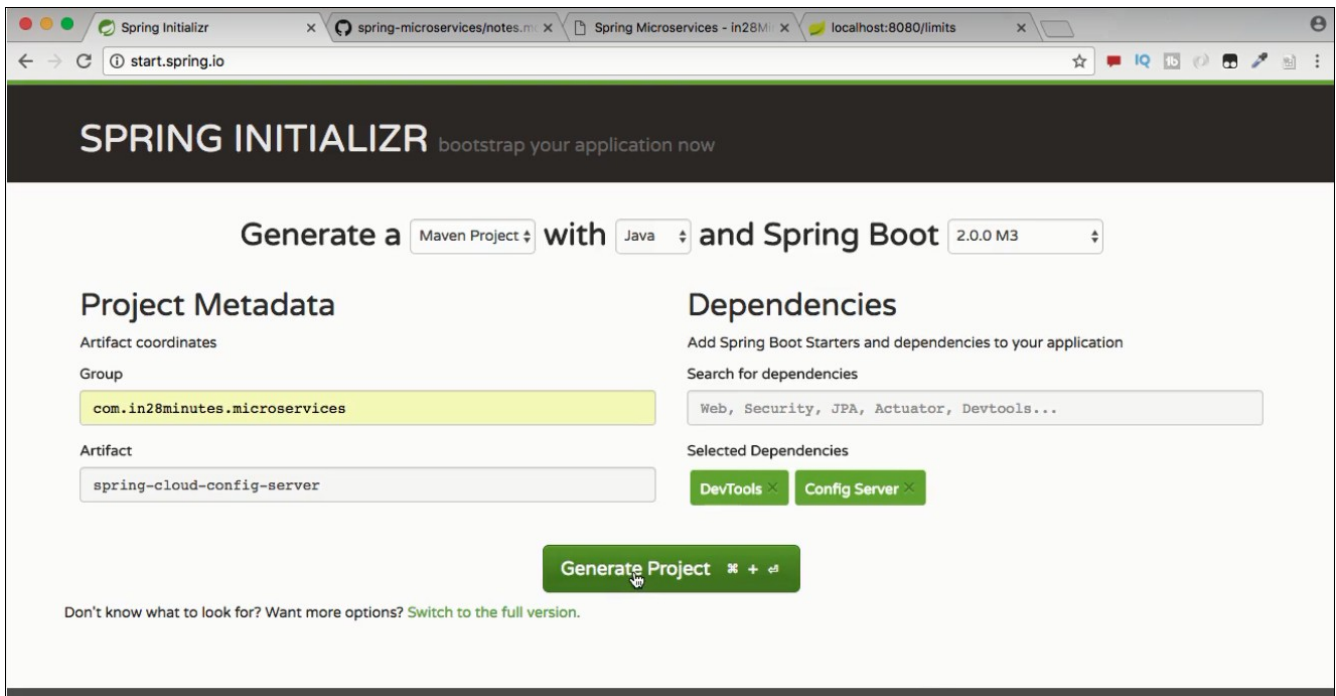
Step4: Test it from browser/postman.



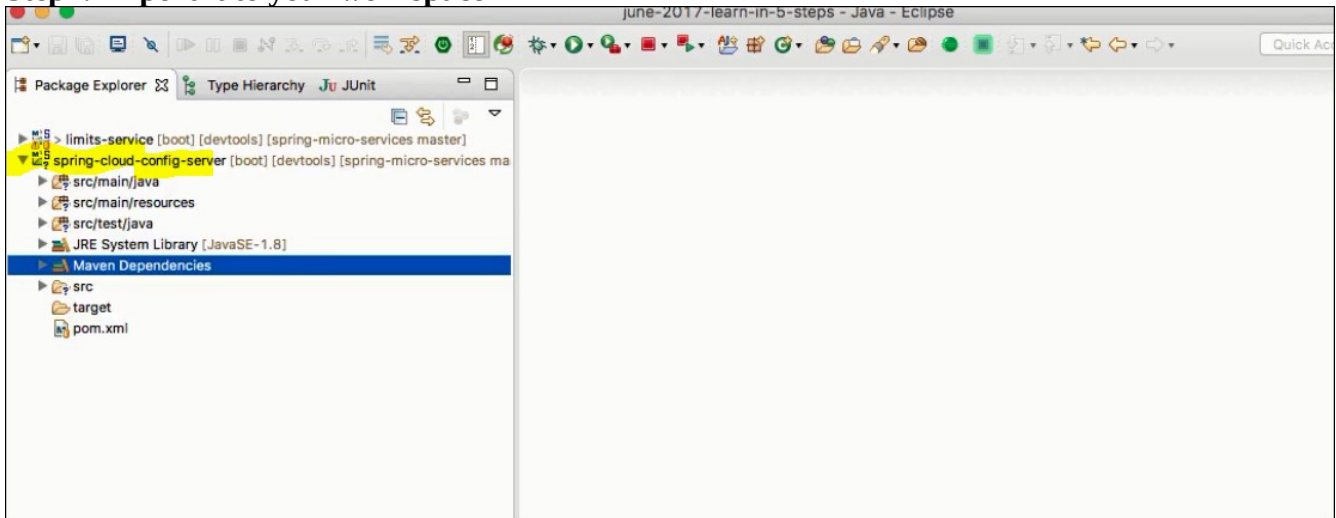
```
{
  maximum: 9999,
  minimum: 99
}
```

Action2: Create The Config Server:

Step1: create the project from Spring initializer



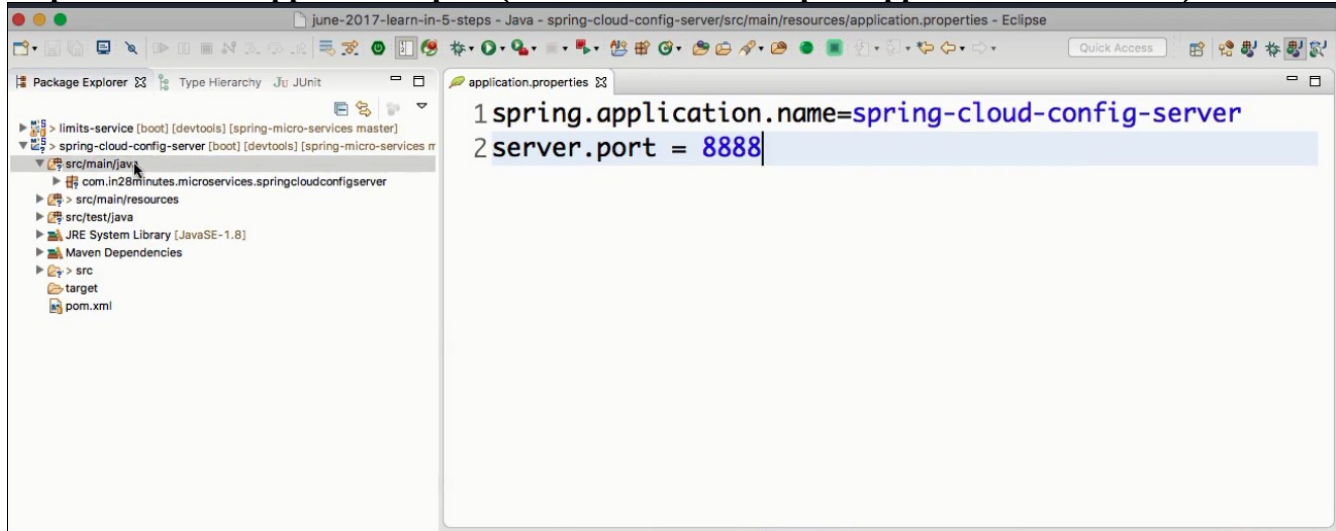
Step2: Import it to your workspace



Note : the below dependency in the pom is for config-server

```
28<dependencies>
29  <dependency>
30    <groupId>org.springframework.cloud</groupId>
31    <artifactId>spring-cloud-config-server</artifactId>
32  </dependency>
```


Step3: Define the App name & port (ref the standardized port/app name matrix chart)



Step 4: Install Git

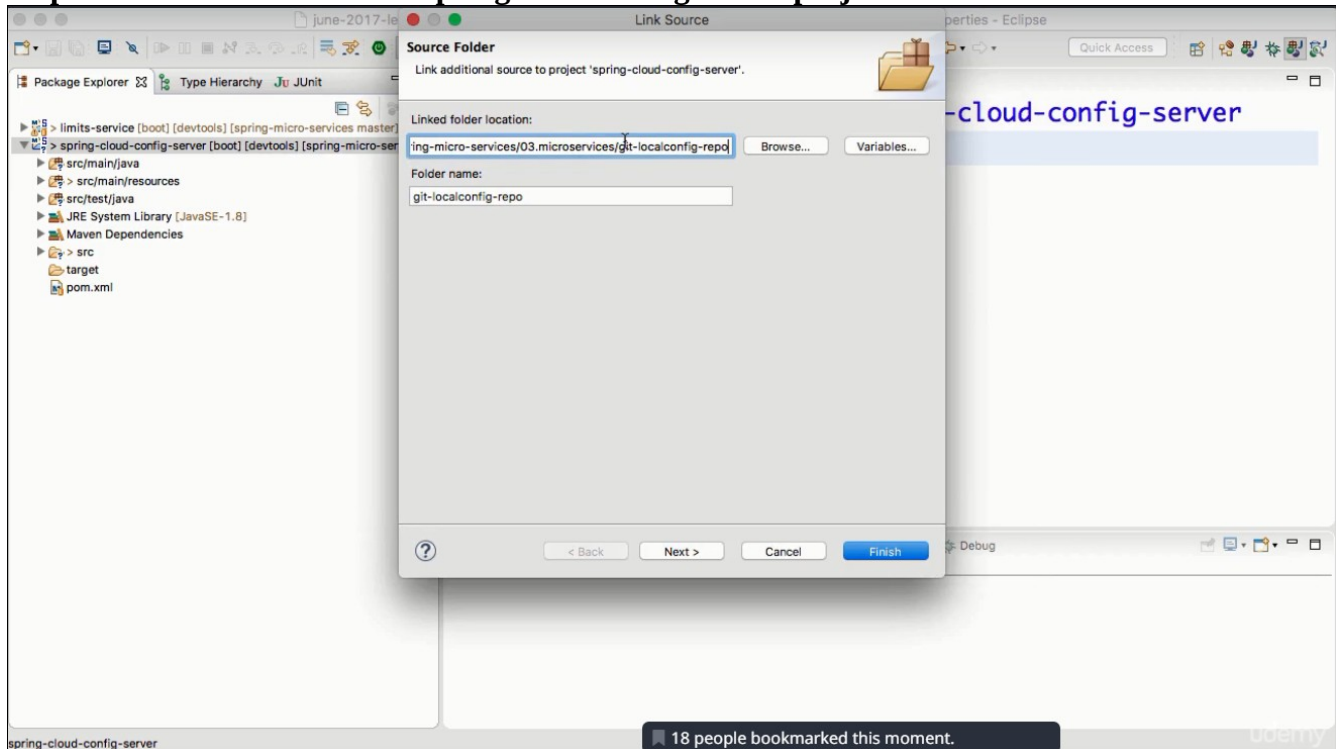
In this step , we create a local Git repository , add files to it & Commit.

```
cmdline$> cd in28Minutesgit/spring-micro-services/03.microservices
```

```
cmdline$> mkdir git-localconfig-repo
```

```
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ mkdir git-localconfig-repo
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ cd git-localconfig-repo/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git init
Initialized empty Git repository in /in28Minutes/git/spring-micro-services/03.microservices/git-localconfig-repo/.git/
```

Step5 : Link the Git Source to spring-cloud-config-server project to be used as a src



Step 6:Add the following files to the folder git-localconfig-repo using eclipse editor:
limit-service.properties

Commit the files in the Git repo:

```
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git add -A
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git commit -m "first commit"
[master (root-commit) 0898c54] first commit
Committer: Ranga Rao Karanam <rangaraokaranam@Rangas-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

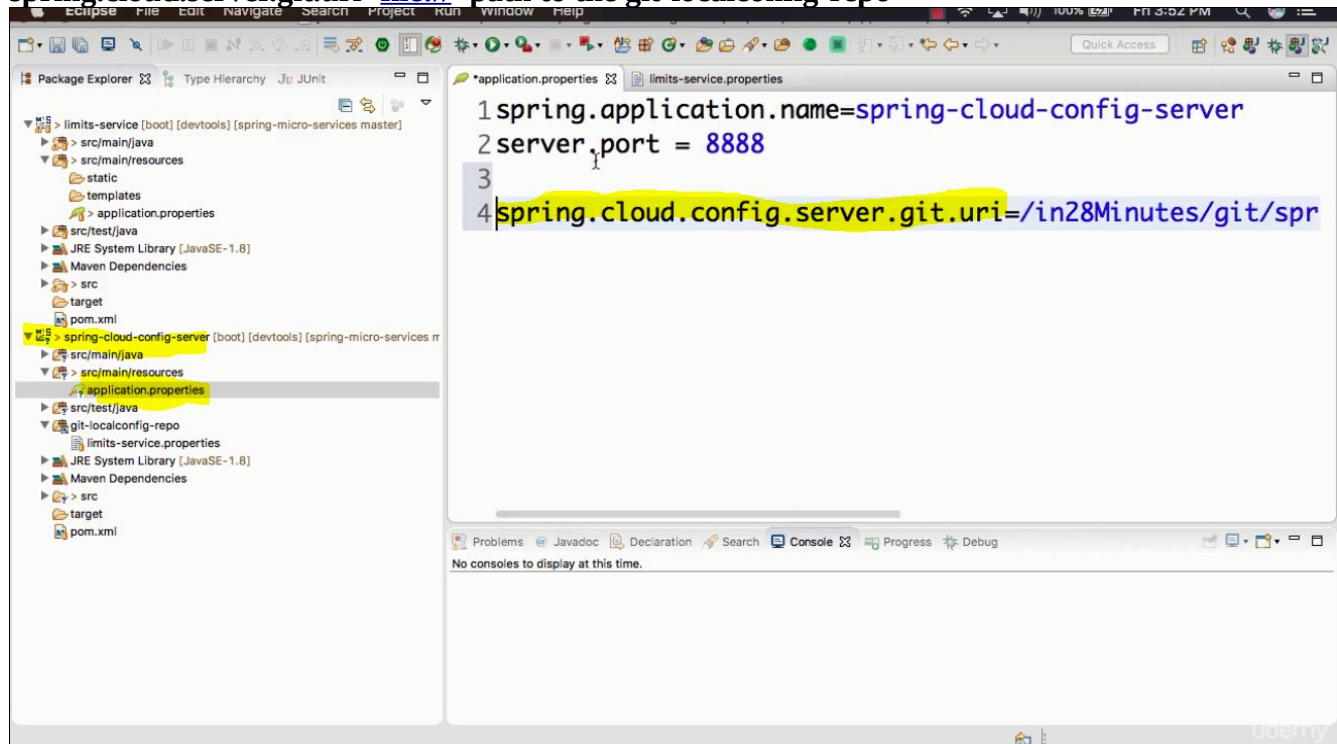
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

After doing this, you may fix the identity used for this commit with:

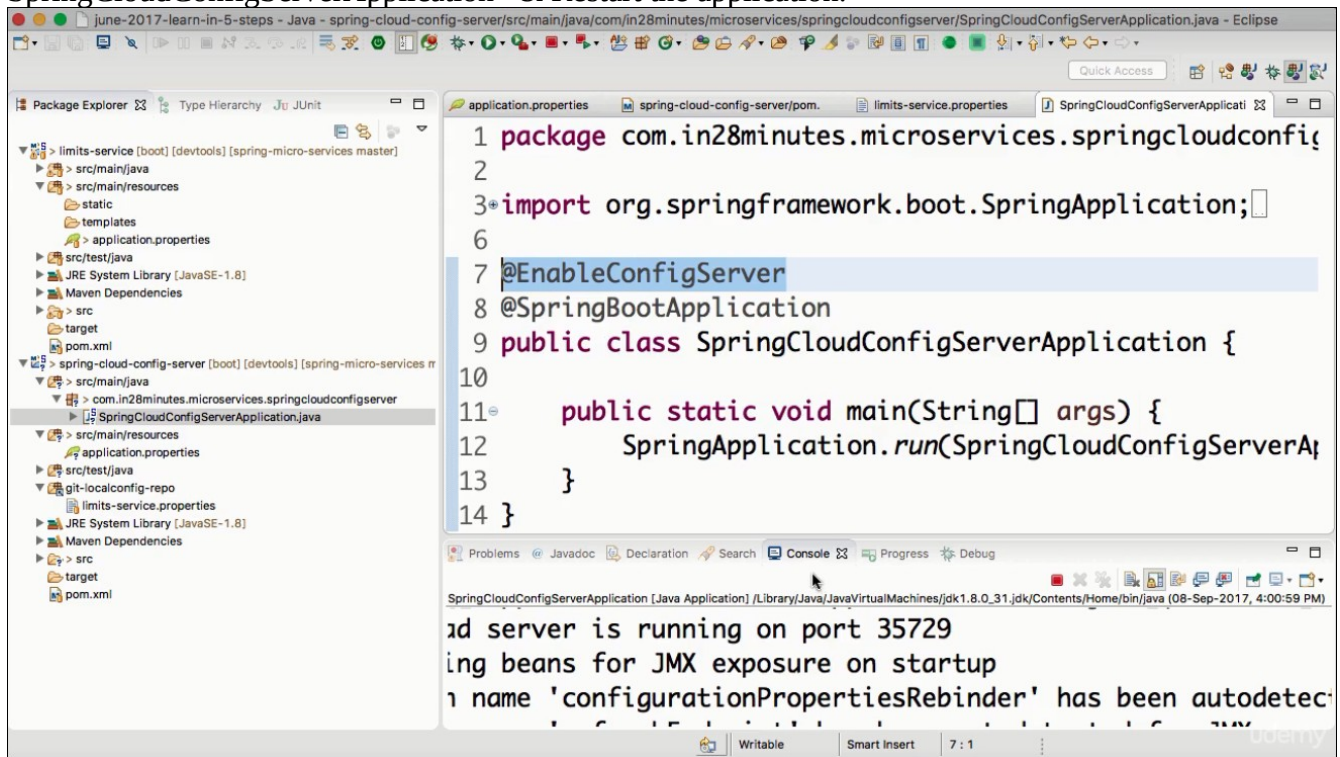
```
git commit --amend --reset-author

1 file changed, 2 insertions(+)
create mode 100644 limits-service.properties
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$
```

Step 7: Point the spring-cloud-config-server to read from the Git URL:
spring.cloud.config.server.git.uri=file://<path to the git-localconfig-repo>



Step 8: Enable Spring-cloud-config-server- Add the `@EnableConfigServer` annotation to the `SpringCloudConfigServerApplication` & Restart the application.

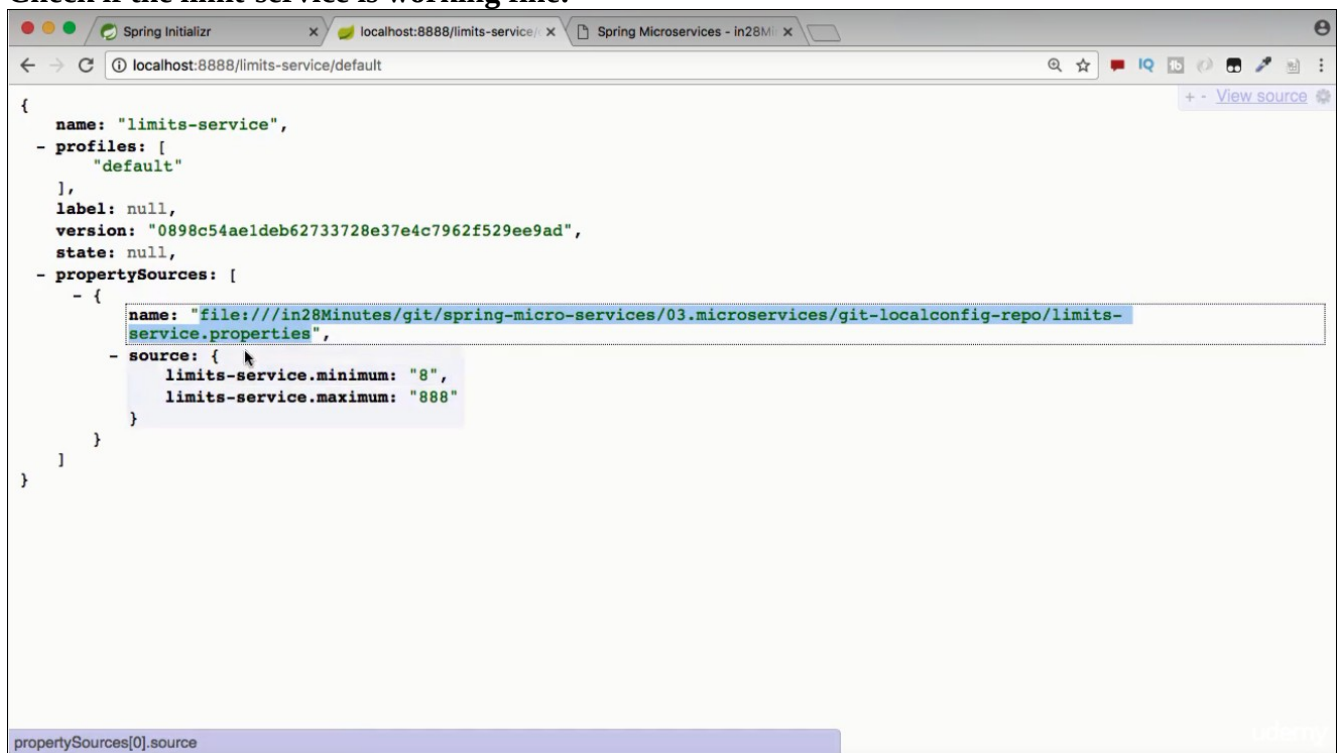


```
1 package com.in28minutes.microservices.springcloudconfigserver;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @EnableConfigServer
8 @SpringBootApplication
9 public class SpringCloudConfigServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(SpringCloudConfigServerApplication.class, args);
13     }
14 }
```

SpringCloudConfigServerApplication [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java (08-Sep-2017, 4:00:59 PM)

ad server is running on port 35729
ing beans for JMX exposure on startup
name 'configurationPropertiesRebinder' has been autodetected

Check if the limit-service is working fine:

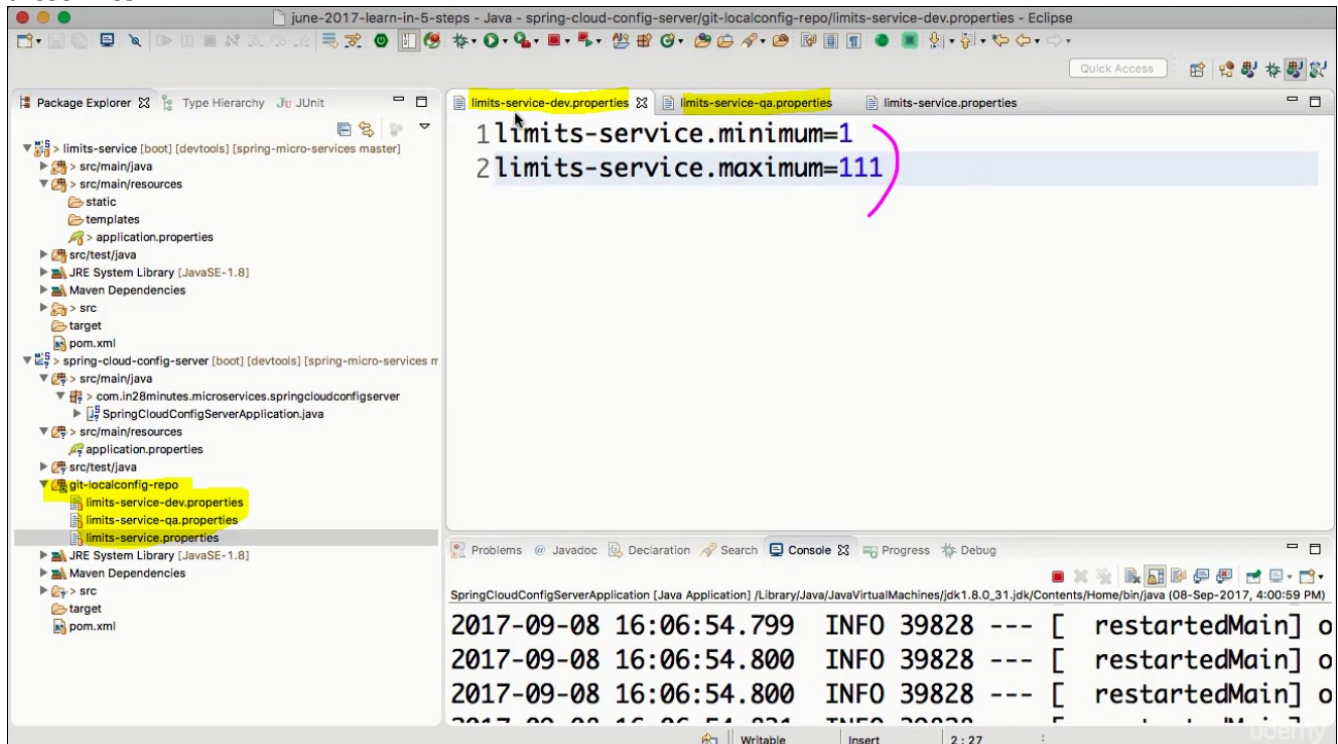


```
{
  "name": "limits-service",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "0898c54a1deb62733728e37e4c7962f529ee9ad",
  "state": null,
  "propertySources": [
    {
      "name": "file:///in28Minutes/git/spring-micro-services/03.microservices/git-localconfig-repo/limits-service.properties",
      "source": {
        "limits-service.minimum": "8",
        "limits-service.maximum": "888"
      }
    }
  ]
}
```

propertySources[0].source

Action3: Make it usable by multiple environment

Step1: Add multiple env specific files in the git-localconfig-repo , update configuration content in these files



Step2: Commit in Git



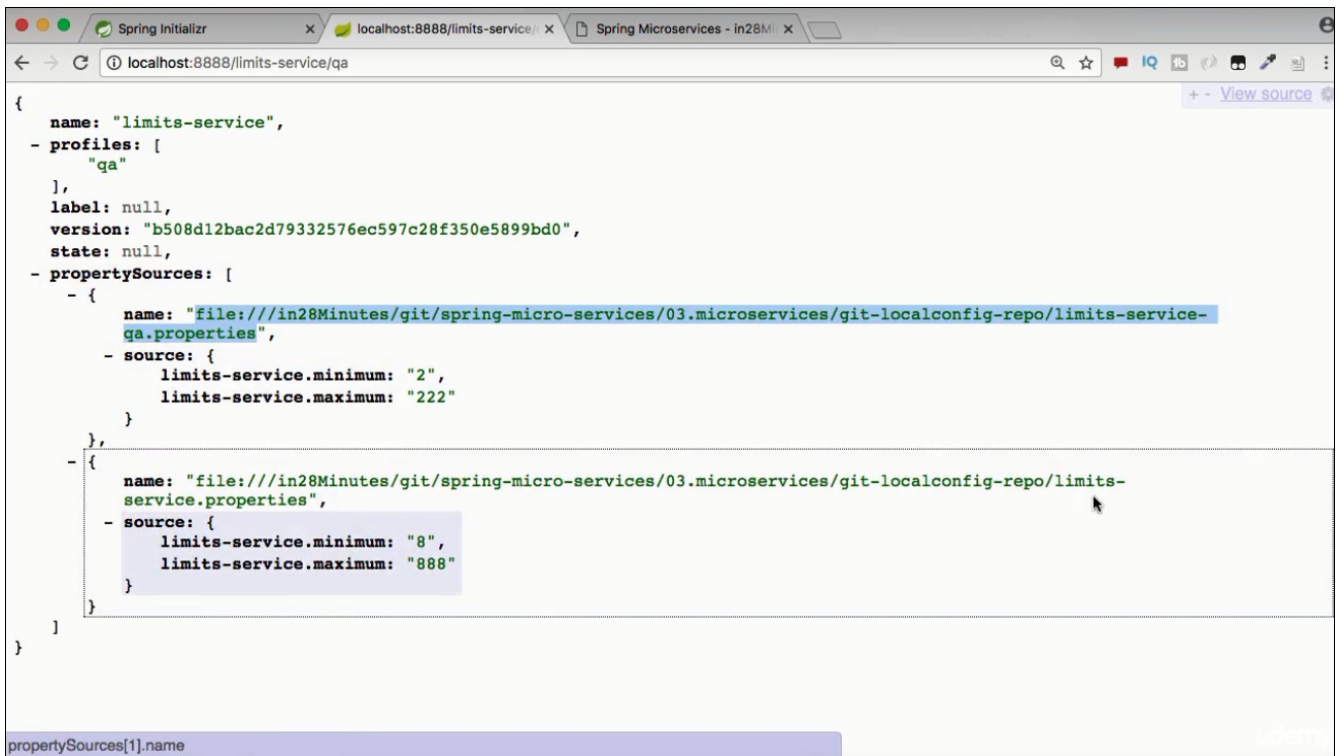
Step3: Check the env specific response from limit-service:

URL: <http://localhost:8888/limit-service/dev>

URL: <http://localhost:8888/limit-service/qa>

Look at the properties sources priorities:

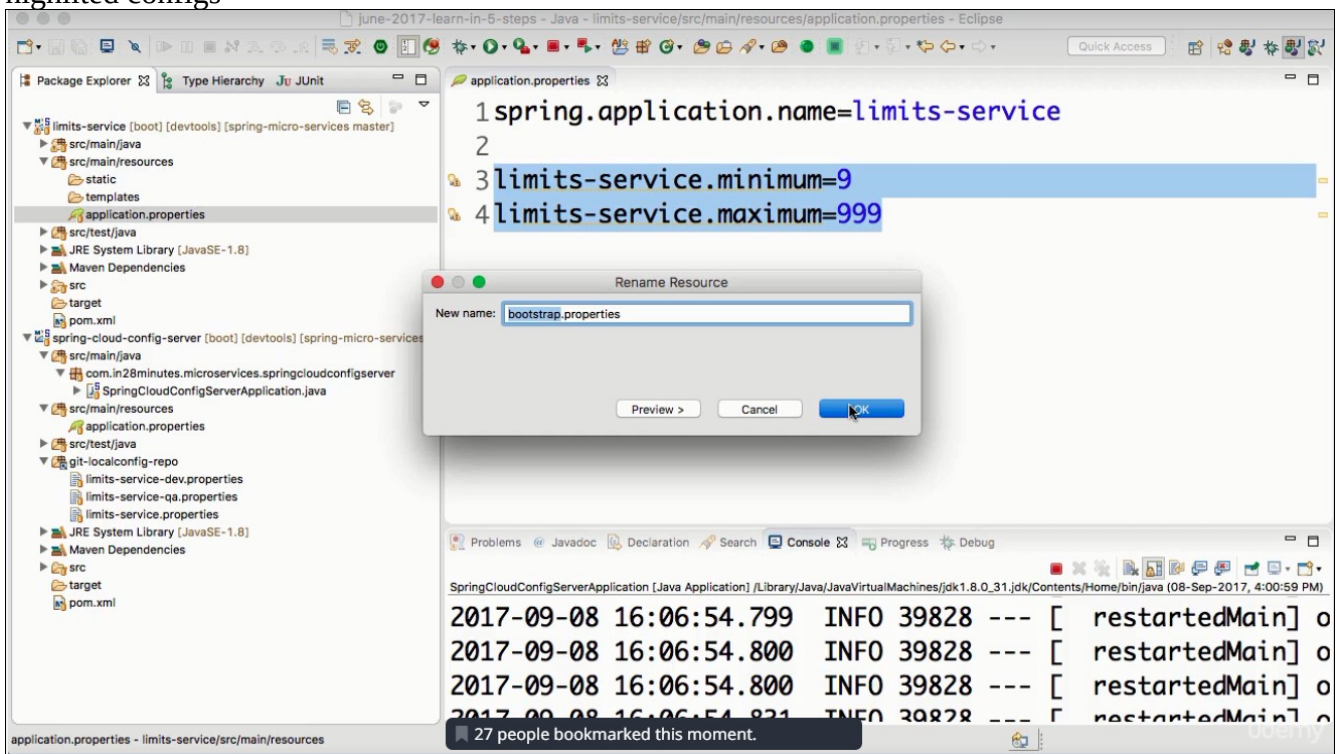
env specific properties will be taken first , if they are not found in a specific env , then default properties value will be taken.



```
{
  name: "limits-service",
  profiles: [
    "qa"
  ],
  label: null,
  version: "b508d12bac2d79332576ec597c28f350e5899bd0",
  state: null,
  propertySources: [
    {
      name: "file:///in28Minutes/git/spring-micro-services/03.microservices/git-localconfig-repo/limits-service-qa.properties",
      source: {
        limits-service.minimum: "2",
        limits-service.maximum: "222"
      }
    },
    {
      name: "file:///in28Minutes/git/spring-micro-services/03.microservices/git-localconfig-repo/limits-service.properties",
      source: {
        limits-service.minimum: "8",
        limits-service.maximum: "888"
      }
    }
  ]
}
```

Action4 : Link limit-service to point to spring-cloud-config-server

Step1: In limit-service rename application.properties → bootstrap.properties & remove all other highlighted configs



```
1 spring.application.name=limits-service
2
3 limits-service.minimum=9
4 limits-service.maximum=999
```

Rename Resource

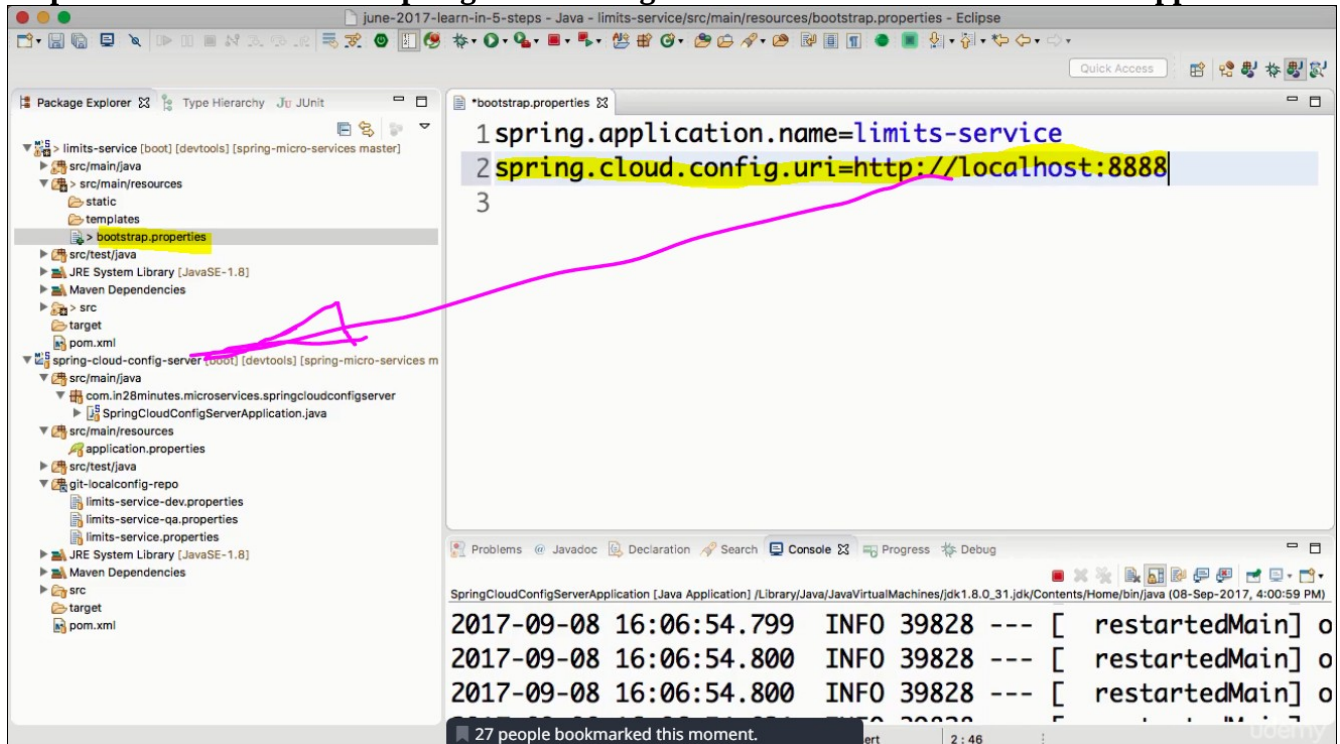
New name: bootstrap.properties

Preview > Cancel OK

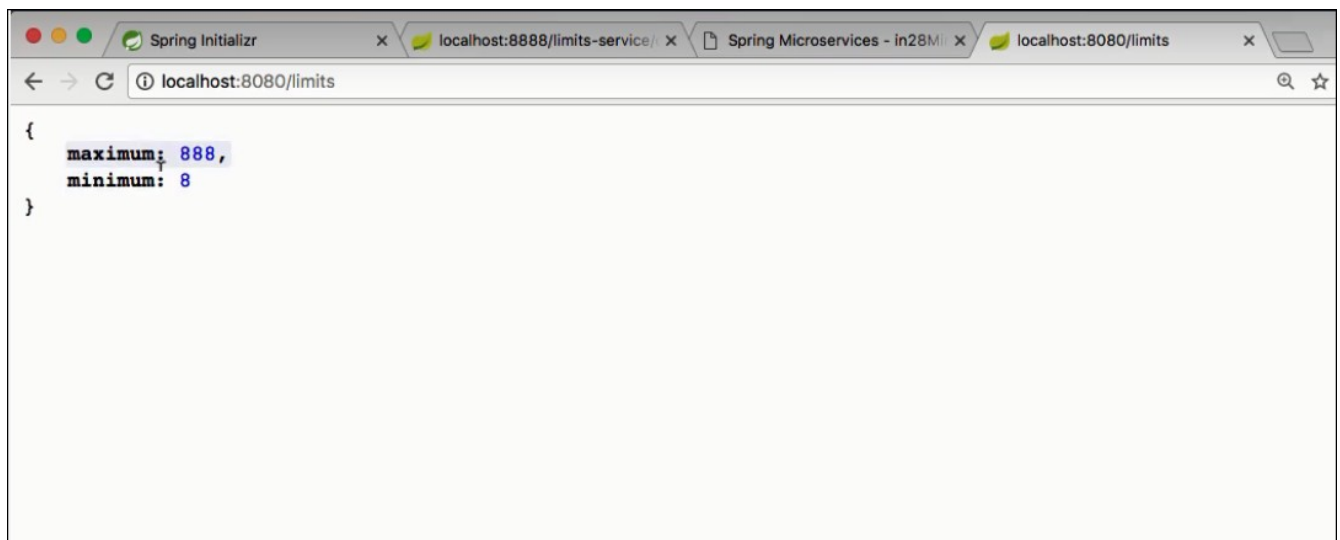
SpringCloudConfigServerApplication [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java (08-Sep-2017, 4:00:59 PM)

```
2017-09-08 16:06:54.799 INFO 39828 --- [ restartedMain] o
2017-09-08 16:06:54.800 INFO 39828 --- [ restartedMain] o
2017-09-08 16:06:54.800 INFO 39828 --- [ restartedMain] o
2017-09-08 16:06:54.821 INFO 39828 --- [ restartedMain] o
```

Step2: Provide the URL for spring-cloud-config-server .Save. Restart LimitServiceApplication.

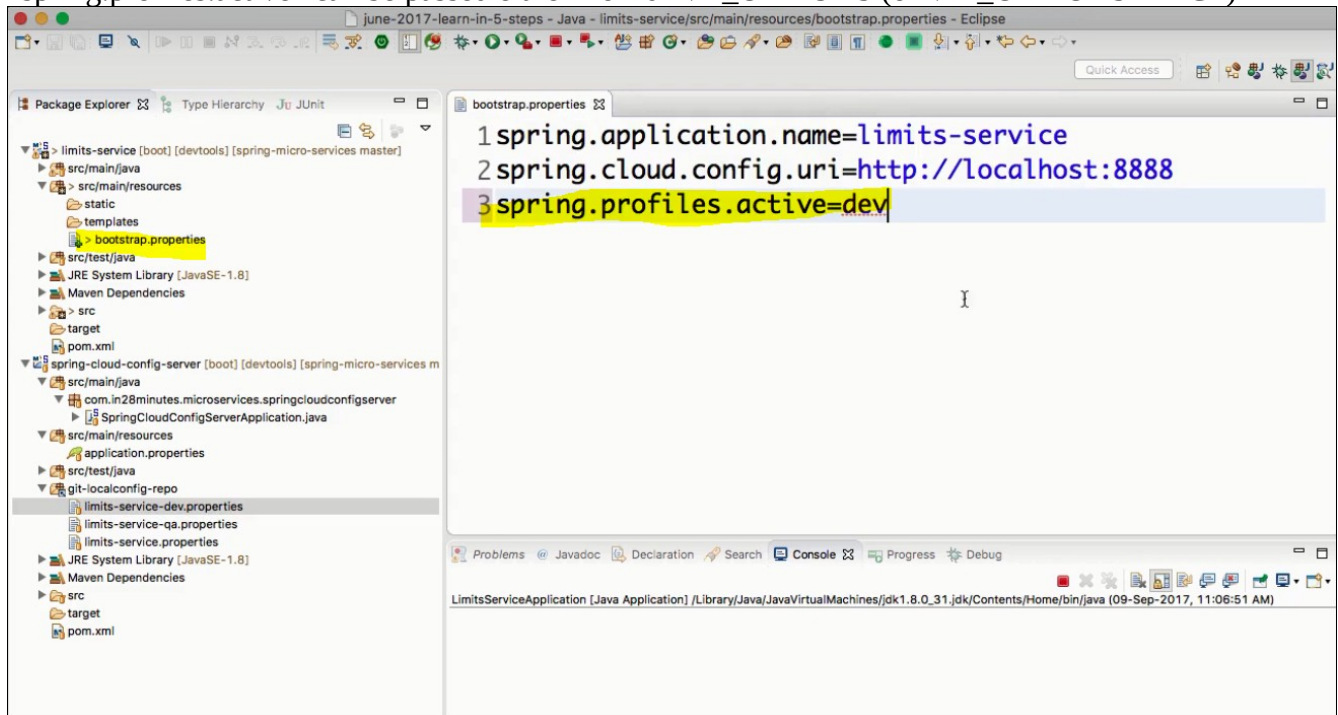


Now if u provide the url: <http://localhost:8080/limits> this will read the default values because profiling is not activated.



Step3 : Configure profiling (in Limit-service)

Add the below properties to point point which is the active profile. In actual deployment the value to “spring.profiles.active” can be passed either from JAVA_OPTIONS (or VM_OPTIONS in PCF)



Step 5 : Restart Limit-service and test

