

# Spring Cloud Configuration

-Shubham Ashtaputre

## A] What is Spring Cloud Config?

Spring Cloud Config is a framework within the Spring ecosystem that provides support for externalized configuration in distributed systems. It allows you to manage and store configuration data for multiple microservices or applications in a centralized location, making it easier to maintain and modify configurations.

With Spring Cloud Config, you can store configuration files in various backends, such as Git, SVN, and local file system. You can also use different file formats, including YAML, JSON, and properties files. Spring Cloud Config provides a RESTful API for accessing configuration properties and also supports dynamic refresh of configuration changes without restarting the microservices or applications.

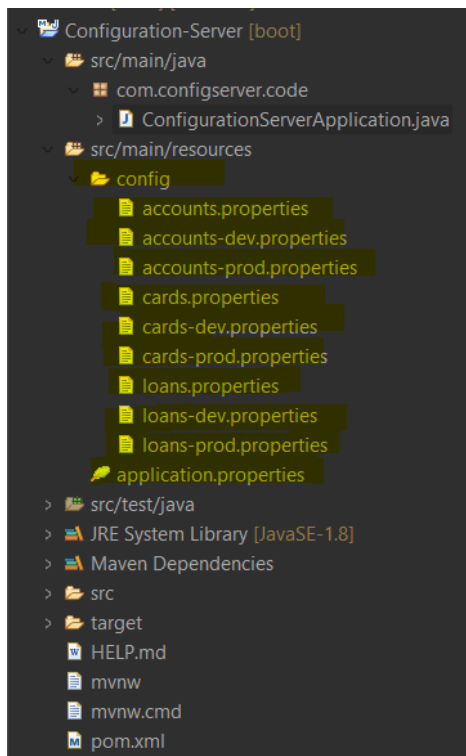
By using Spring Cloud Config, you can avoid hard-coding configuration values in your code and reduce the risk of configuration errors. It also enables you to manage configuration changes in a version control system, track configuration history, and perform audits. Overall, Spring Cloud Config provides a powerful and flexible way to manage configuration data in distributed systems.

## B] How to setup Spring Cloud Config and read properties value from classpath or any location:

1] add @EnableConfigServer

```
1 package com.configserver.code;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableConfigServer
9 public class ConfigurationServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(ConfigurationServerApplication.class, args);
13     }
14
15 }
16
```

2] create a config folder in which we will maintain all our properties related to all our microservices



3] inside **application.properties** file:

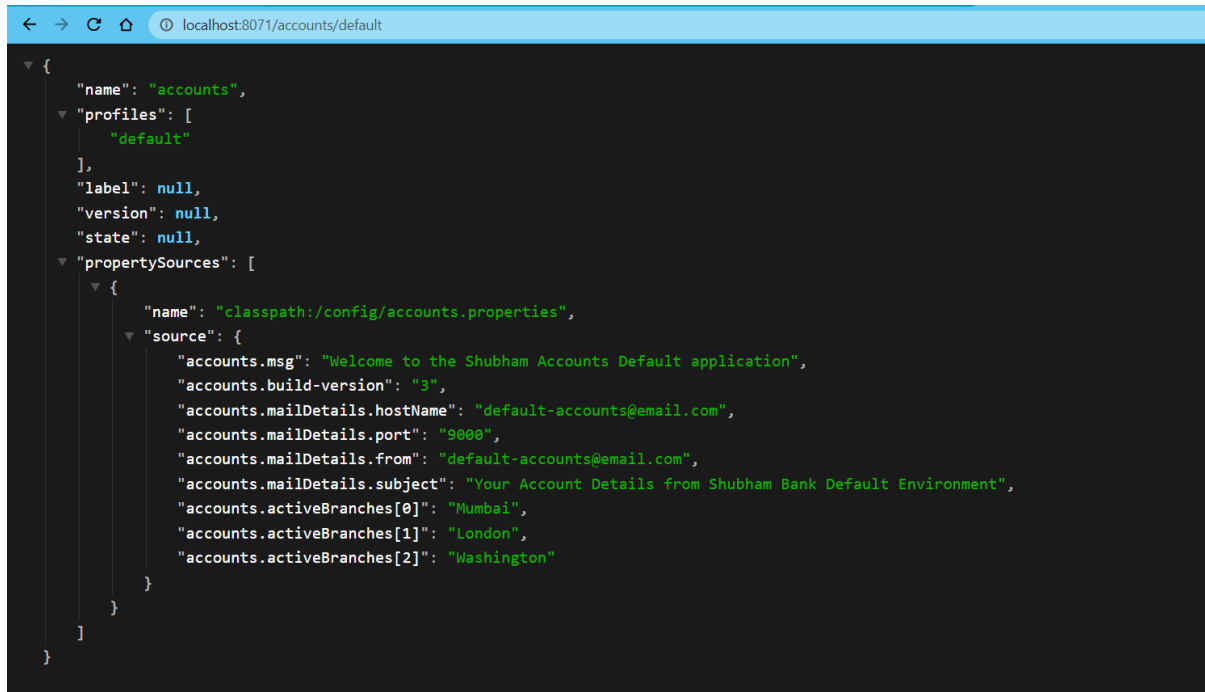
```
1 spring.application.name=configserver
2
3 spring.profiles.active=native
4 #spring.profiles.active=git
5
6 spring.cloud.config.server.native.search-locations=classpath:/config
7 #spring.cloud.config.server.native.search-locations=file:///C://config
8 #spring.cloud.config.server.git.uri=
9 #spring.cloud.config.server.git.clone-on-start=true
10 #spring.cloud.config.server.git.default-label=main
11
12 server.port=8071
13
14 #encrypt.key=shubhama
15
```

Here, I had set active profile as native i.e. **spring.profiles.active=native**. Now, this **native** means that we are going to use the configuration present either from the classpath or the properties file

And, **spring.cloud.config.server.native.search-locations=classpath:/config** means that my configurations files are present inside location **classpath:/config** and I'm telling spring cloud to search properties from this location

4] now to access this properties values start the application and access the url as below:

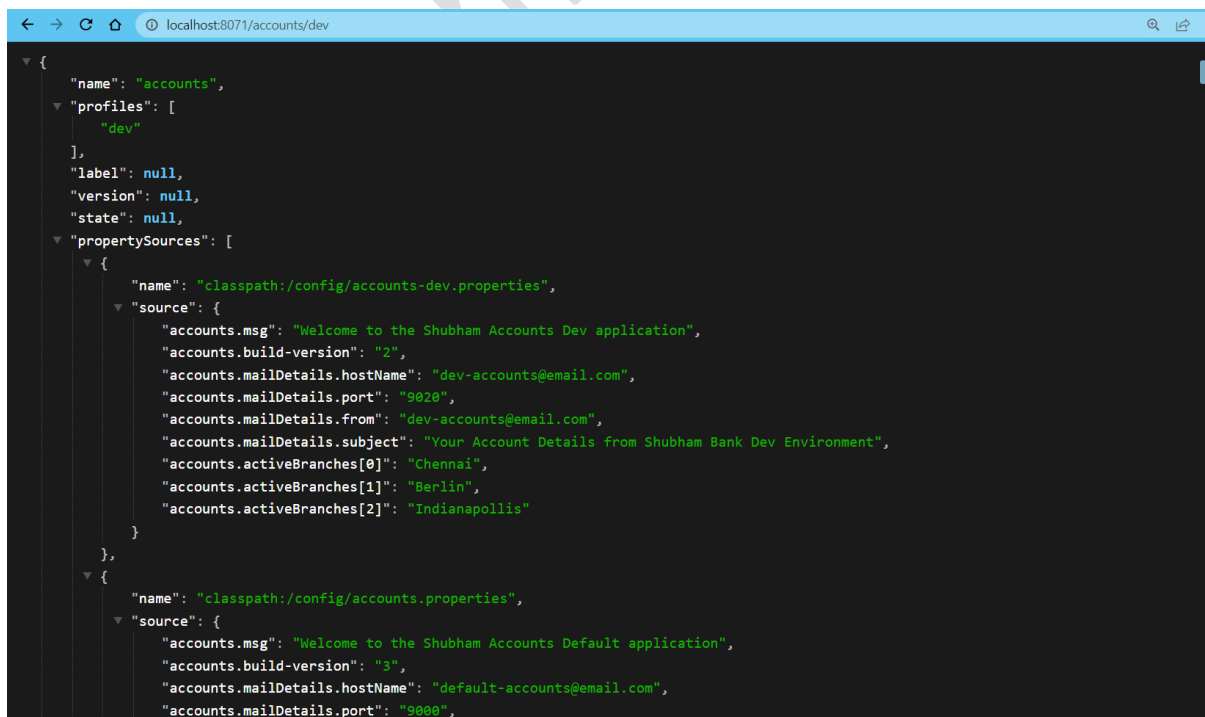
<http://localhost:8071/accounts/default>



```
{
  "name": "accounts",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "classpath:/config/accounts.properties",
      "source": {
        "accounts.msg": "Welcome to the Shubham Accounts Default application",
        "accounts.build-version": "3",
        "accounts.mailDetails.hostName": "default-accounts@email.com",
        "accounts.mailDetails.port": "9000",
        "accounts.mailDetails.from": "default-accounts@email.com",
        "accounts.mailDetails.subject": "Your Account Details from Shubham Bank Default Environment",
        "accounts.activeBranches[0]": "Mumbai",
        "accounts.activeBranches[1]": "London",
        "accounts.activeBranches[2]": "Washington"
      }
    }
  ]
}
```

To access dev environment:

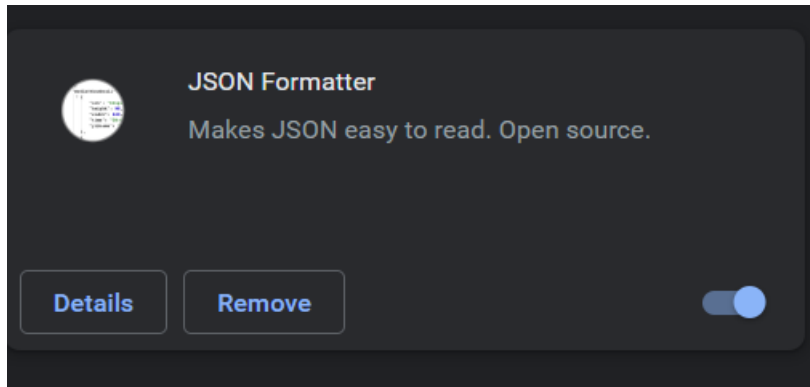
<http://localhost:8071/accounts/dev>



```
{
  "name": "accounts",
  "profiles": [
    "dev"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "classpath:/config/accounts-dev.properties",
      "source": {
        "accounts.msg": "Welcome to the Shubham Accounts Dev application",
        "accounts.build-version": "2",
        "accounts.mailDetails.hostName": "dev-accounts@email.com",
        "accounts.mailDetails.port": "9020",
        "accounts.mailDetails.from": "dev-accounts@email.com",
        "accounts.mailDetails.subject": "Your Account Details from Shubham Bank Dev Environment",
        "accounts.activeBranches[0]": "Chennai",
        "accounts.activeBranches[1]": "Berlin",
        "accounts.activeBranches[2]": "Indianapolis"
      }
    },
    {
      "name": "classpath:/config/accounts.properties",
      "source": {
        "accounts.msg": "Welcome to the Shubham Accounts Default application",
        "accounts.build-version": "3",
        "accounts.mailDetails.hostName": "default-accounts@email.com",
        "accounts.mailDetails.port": "9000",
        "accounts.mailDetails.from": "default-accounts@email.com",
        "accounts.mailDetails.subject": "Your Account Details from Shubham Bank Default Environment",
        "accounts.activeBranches[0]": "Mumbai",
        "accounts.activeBranches[1]": "London",
        "accounts.activeBranches[2]": "Washington"
      }
    }
  ]
}
```

Note:

Here, I'm using the extension JSON formatter extension to view JSON data



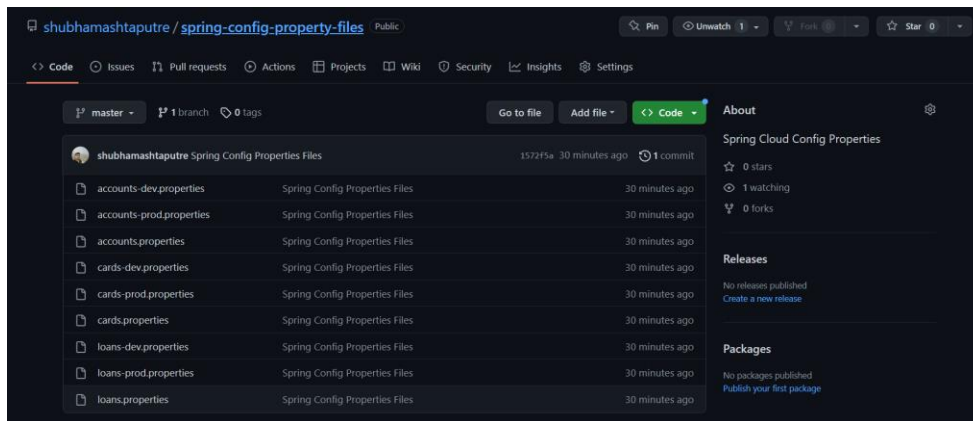
### C] How to read the properties values from the GIT location:

C1] make changes inside the application.properties file as below:

```
1 spring.application.name=configserver
2 #spring.profiles.active=native
3 spring.profiles.active=git
4 #spring.cloud.config.server.native.search-locations=classpath:/config
5 #spring.cloud.config.server.native.search-locations=file:///C:/config
6 spring.cloud.config.server.git.uri= https://github.com/shubhamashtaputre/spring-config-property-files.git
7 spring.cloud.config.server.git.clone-on-start=true
8 spring.cloud.config.server.git.default-label=master
9 server.port=8071
10 #encrypt.key=shubhama
```

**spring.cloud.config.server.git.uri** is our github location in which I'm maintaining my all property files, **spring.cloud.config.server.git.default-label** is the branch name in which my property files are present

C2] Git repository:



## D] Configuring application to read properties from Config server

In this case I'm making the Account microservice to read properties values from the config server accounts properties files, below are the steps to achieve it.

1] Add the below dependency in account pom.xml file:

```
26• <dependency>
27• <!-- Add this dependency while doing spring cloud config too
28• whenever we want our microservice app to act as a client for spring cloud config server-->
29• <groupId>org.springframework.cloud</groupId>
30• <artifactId>spring-cloud-starter-config</artifactId>
31• </dependency>
```

```
78 <!-- Adding the spring cloud config dependency in the project -->
79• <dependencyManagement>
80• <dependencies>
81• <dependency>
82• <groupId>org.springframework.cloud</groupId>
83• <artifactId>spring-cloud-dependencies</artifactId>
84• <version>${spring-cloud.version}</version>
85• <type>pom</type>
86• <scope>import</scope>
87• </dependency>
88• </dependencies>
89• </dependencyManagement>
```

2] Inside the **application.yml** file for Account microservice application add below changes

```
20 application:
21   name: accounts
22 profiles:
23   active: dev
24 config:
25   import: optional:configserver:http://localhost:8071
26
```

2.1] Here, accounts is the name of the properties files present inside the Configuration-Server application whose properties I want to read

2.2] Here, **active: dev** means I want to read properties of dev environment of this accounts properties file

2.3] Now this inside import is the address from where I will get my required properties files present at remote location

3] Create a '**AccountServiceConfig**' service file:

```
1 package com.account.service;
2
3 import java.util.List;
4
5 @Configuration
6 @ConfigurationProperties(prefix="accounts")
7 @Getter
8 @Setter
9 @ToString
10 public class AccountServiceConfig {
11
12     private String msg;
13     private String buildVersion;
14     private Map<String, String> mailDetails;
15     private List<String> activeBranches;
16
17 }
18
```

Here, if you see I had used annotation as **@ConfigurationProperties(prefix="accounts")**

Which describes as, **@ConfigurationProperties** is an annotation in the Spring Framework that is used to bind external configuration properties to a Java bean. It is commonly used in Spring Boot applications to externalize the configuration and separate it from the code.

4] Create Property POJO file:

```

1 package com.account.bean;
2
3 import java.util.List;
4
5 @Getter
6 @Setter
7 public class Property {
8
9     private String msg;
10    private String buildVersion;
11    private Map<String, String> mailDetails;
12    private List<String> activeBranches;
13
14    public Property(String msg, String buildVersion, Map<String, String> mailDetails,
15        List<String> activeBranches) {
16        super();
17        this.msg = msg;
18        this.buildVersion = buildVersion;
19        this.mailDetails = mailDetails;
20        this.activeBranches = activeBranches;
21    }
22 }
23
24
25

```

In this POJO class we will pass the properties file values from **AccountServiceConfig** using getters and setters

5] In controller class create the getter method as:

```

37
38 @GetMapping(path="/accounts/properties")
39 public String getPropertyDetails() throws JsonProcessingException {
40
41     ObjectMapper oW = new ObjectMapper().writer().withDefaultPrettyPrinter();
42
43     Property properties = new Property(accountsConfig.getMsg(), accountsConfig.getBuildVersion(),
44         accountsConfig.getMailDetails(), accountsConfig.getActiveBranches());
45
46     String jsonString = oW.writeValueAsString(properties);
47
48     System.out.println("----> "+jsonString);
49
50     return jsonString;
51 }
52
53

```

## E] Spring Cloud Configuration set the properties dynamically in the Docker-Compose file

1] Now, suppose you have deployed your microservice application into docker environment and after sometime you realized you need to change the environment from dev to prod for a particular microservice, at that time you again need to make changes into **application.yml** file of that microservice and again deploy it's image into docker



2] Now, to solve this issue what you can do is create a docker compose file and set the properties file values for that microservice from there itself and avoid further complications as below:

```
version: "3.8"

services:

  configuration-server:
    image: ssadocl23/configuration-server:2.0
    mem_limit: 700m
    ports:
      - "8072:8071"
    networks:
      - ssa-network

  account:
    image: ssadocl23/account:2.0
    mem_limit: 700m
    ports:
      - "8081:8080"
    networks:
      - ssa-network
    depends_on:
      - configuration-server
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
    environment:
      SPRING_PROFILES_ACTIVE: default
      SPRING_CONFIG_IMPORT: configserver:http://configuration-server:8071/
```

```
  loan:
    image: ssadocl23/loan:2.0
    mem_limit: 700m
    ports:
      - "8082:8080"
    networks:
      - ssa-network
    depends_on:
      - configuration-server
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
    environment:
      SPRING_PROFILES_ACTIVE: default
      SPRING_CONFIG_IMPORT: configserver:http://configuration-server:8071/

  card:
    image: ssadocl23/card:2.0
    mem_limit: 700m
    ports:
      - "8083:8080"
    networks:
      - ssa-network
    depends_on:
      - configuration-server
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
    environment:
      SPRING_PROFILES_ACTIVE: default
      SPRING_CONFIG_IMPORT: configserver:http://configuration-server:8071/

networks:
  ssa-network: {}
```

3] Now, if you see in this docker-compose file what I have done is I had deployed 4 microservices including spring cloud config server too from which my rest 3 microservices will read their environment properties

4] Here, if you see in account, loan and card microservice its says that these apps depends on **configuration-server** image so unless this **configuration-server** container is not running the rest of these 3 microservice will not start, and further I had also set the **restart\_policy** which will restart my 3 microservices the number 3 times as defined, if my **configuration-server** has not started in first 2 attempts else even during 3 times these 3 microservices are restarting and still at that time **configuration-server** has not started then my I should get error

5] Further if you see al these 4 microservices comes under same network so that they will start and stop simultaneously, they form a group

6] Further in **environment** I had also set which active environment I want to use for my particular microservice and the **configuration-server** URL too to read the property values

7] After deployment of this docker compose file in docker environment the docker container looks as below:

<input type="checkbox"/>	default	-	Exited	3 days ago	▶	⋮	🗑️
<input type="checkbox"/>	account-1 71918c00b3e4	<a href="#">ssadoc123/account:2.0</a>	Exited (143)	8081:8080	🔗	3 days ago	▶ ⋮ 🗑️
<input type="checkbox"/>	loan-1 3b3f3bc6252b	<a href="#">ssadoc123/loan:2.0</a>	Exited (143)	8082:8080	🔗	3 days ago	▶ ⋮ 🗑️
<input type="checkbox"/>	card-1 458e8e7e061d	<a href="#">ssadoc123/card:2.0</a>	Exited (143)	8083:8080	🔗	3 days ago	▶ ⋮ 🗑️
<input type="checkbox"/>	configuration-server-1 8a532c08dbab	<a href="#">ssadoc123/configuration-server:2.0</a>	Exited (143)	8072:8071	🔗	3 days ago	▶ ⋮ 🗑️

F] Using **@RefreshScope** annotation:

1] Now, suppose my microservices are running into the docker container and suppose I want change property values of some microservices, in that case I have to stop my all microservices and then change the property file values and again restart the microservices,

2] But this thing is not possible in real time as in production environment my live application is hosted and I had to again and again change my property files values and restart the docker server then I would affect the end user

3] So in that case I will simply use the **@RefreshScope** annotation which will refresh the updated property files values in background without need to restart the server

4] We assign this annotation as below in main file:

```
1 package com.account.start;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7
8
9
10 @SpringBootApplication
11 @RefreshScope
12 @ComponentScan(basePackages = {"com.account.*", "com.account.service"})
13 @EntityScan(basePackages = {"com.account.bean"})
14 @EnableJpaRepositories(basePackages = "com.account.dao")
15 public class AccountApplication {
16
17     public static void main(String[] args) {
18         SpringApplication.run(AccountApplication.class, args);
19     }
20
21 }
22
```

That's it

5] Follow the below steps to see this annotation usage:

5.1] Start the **ConfigurationServerApplication** application

5.2] For example start the **AccountApplication** application

5.3] If you can see my 'dev' environment is active in the account microservice as below:

GET http://localhost:8080/bank/accounts/properties

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DE
Key	Value	De

Body Cookies Headers (8) Test Results Status

Pretty Raw Preview Visualize Text

```

1  {
2    "msg" : "Welcome to the Shubham Accounts Dev application",
3    "buildVersion" : "2",
4    "mailDetails" : {
5      "hostName" : "dev-accounts@email.com",
6      "port" : "9020",
7      "from" : "dev-accounts@email.com",
8      "subject" : "Your Account Details from Shubham Bank Dev Environment"
9    },
10   "activeBranches" : [ "Chennai", "Berlin", "Indianapolis" ]
11 }

```

5.4] now, I will change the property file value of this account dev environment from the github location as my config server is read property values from github

master spring-config-property-files / accounts-dev.properties Go to file

shubhamashtaputre Spring Config Properties Files Latest commit 1572f5a last week History

1 contributor

11 lines (9 sloc) 415 Bytes Raw Blame

```

1 accounts.msg=Welcome to the Shubham Accounts Dev application
2 accounts.build-version=2
3
4 accounts.mailDetails.hostName=dev-accounts@email.com
5 accounts.mailDetails.port=9020
6 accounts.mailDetails.from=dev-accounts@email.com
7 accounts.mailDetails.subject=Your Account Details from Shubham Bank Dev Environment
8
9 accounts.activeBranches[0]=Chennai
10 accounts.activeBranches[1]=Berlin
11 accounts.activeBranches[2]=Indianapolis

```

Give feedback

Original values

master spring-config-property-files / accounts-dev.properties

shubhamashtaputre Update properties Latest commit af08476 now History

1 contributor

11 lines (9 sloc) 413 Bytes

```
1 accounts.msg =Welcome to the Shubham Accounts Dev application
2 accounts.build-version=3
3
4 accounts.mailDetails.hostName=dev-accounts@email.com
5 accounts.mailDetails.port=9030
6 accounts.mailDetails.from=dev-accounts@email.com
7 accounts.mailDetails.subject=Your Account Details from Shubham Bank Dev Environment
8
9 accounts.activeBranches[0]=Pune
10 accounts.activeBranches[1]=Berlin
11 accounts.activeBranches[2]=Indianapolis
```

Give feedback

## Updated Values

5.5] Check the actuator in the Accounts url as below:

New Collection GET New Request GET http://localhost:8080/2 + ... No E

http://localhost:8080/actuator

GET http://localhost:8080/actuator

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (5) Test Results Status: 200 OK Time: 27 ms Size: 1

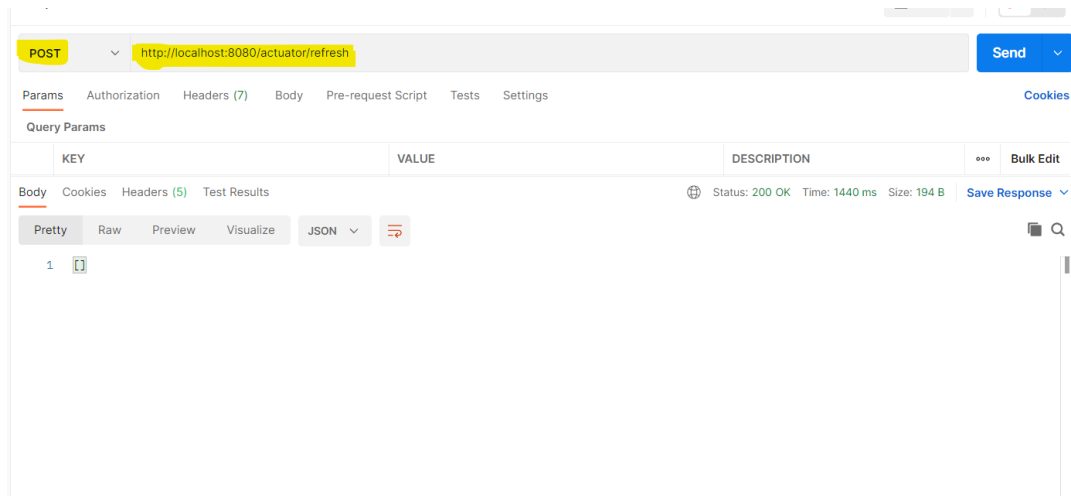
Pretty Raw Preview Visualize JSON

```
73     "templated": false
74   },
75   "scheduledtasks": {
76     "href": "http://localhost:8080/actuator/scheduledtasks",
77     "templated": false
78   },
79   "mappings": {
80     "href": "http://localhost:8080/actuator/mappings",
81     "templated": false
82   },
83   "refresh": {
84     "href": "http://localhost:8080/actuator/refresh",
85     "templated": false
86   },
87   "features": {
88     "href": "http://localhost:8080/actuator/features",
89     "templated": false
90   }
91 }
92 }
```

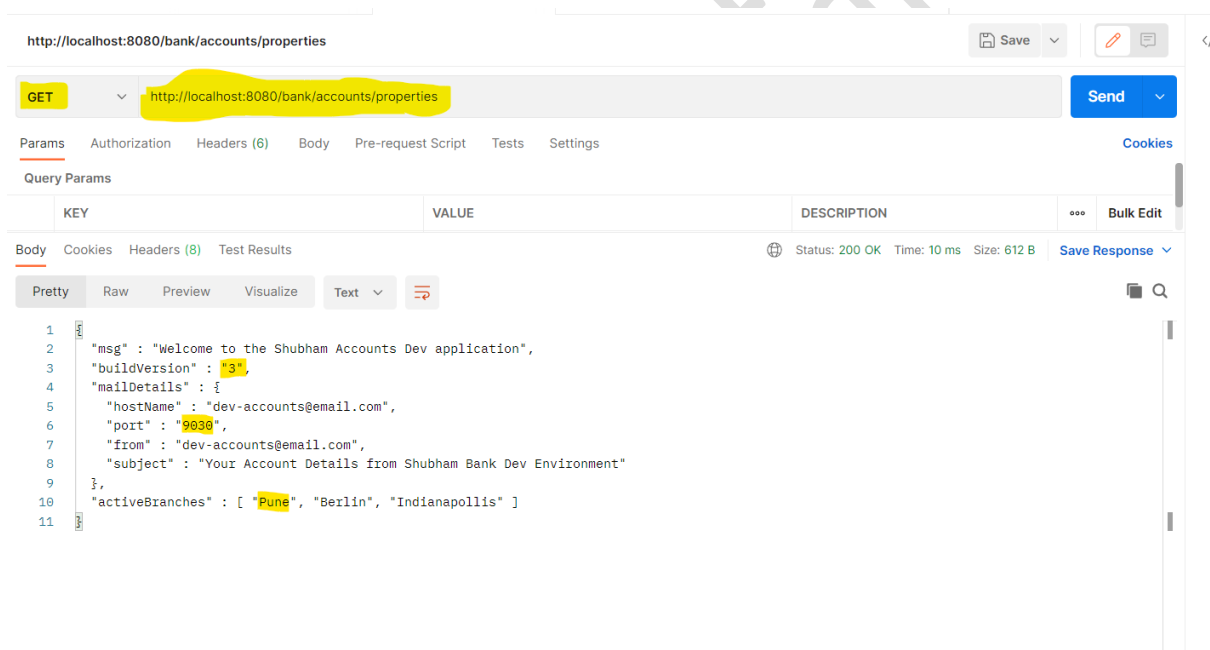
You will see this refresh url as below:

"http://localhost:8080/actuator/refresh"

Hit this refresh URL using POST method as below:



And then call you, account property **URL** you will see updated values:



Without restarting the server these values are updated

Shubham Ashtaputre

Shubham Ashtaputre