Pivotal

Workshop Hands On – Spring Cloud Services

Contents

1. Clone and build projects

1. Create a directory and navigate to the directory by executing the following command:

   mkdir ~/scs-ws && cd ~/scs-ws

```
1. bash
Derrick-Chuas-MacBook-Pro:~ tmchua$ mkdir ~/scs-ws && cd ~/scs-ws
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$ pwd
/Users/tmchua/scs-ws
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$
```

2. Clone the Spring Cloud Config project by running the following command:

   git clone https://github.com/derrick81/scs-config-master.git && \
   cd scs-config-master && \
   ./mvnw -DskipTests package && \
   cd ..

```
1. bash
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$ git clone https://github.com/derrick81/scs-config-master.git && \
> cd scs-config-master && \
> ./mvnw -DskipTests package && \
> cd ..
Cloning into 'scs-config-master'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 39 (delta 7), reused 32 (delta 3), pack-reused 0
Unpacking objects: 100% (39/39), done.
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< com.example:scs-config >-----------------------
[INFO] Building scs-config 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ scs-config ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
```

3. Clone the Spring Cloud Eureka project by running the following command:

   git clone https://github.com/derrick81/scs-eureka-master.git && \
   cd scs-eureka-master && \
   ./mvnw -DskipTests package && \
   cd ..

```
1. bash
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$ git clone https://github.com/derrick81/scs-eureka-master.git && \
> cd scs-eureka-master && \
> ./mvnw -DskipTests package && \
> cd ..
Cloning into 'scs-eureka-master'...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 33 (delta 4), reused 27 (delta 1), pack-reused 0
Unpacking objects: 100% (33/33), done.
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< com.example:scs-eureka >-----------------------
[INFO] Building scs-eureka 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ scs-eureka ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
```

4. Clone the 4 microservices and client projects by running the following command:

```
git clone https://github.com/derrick81/scs-clients-master.git && \
cd scs-clients-master/employee-service && \
./mvnw -DskipTests package && \
cd ../employee-feign-client
./mvnw -DskipTests package && \
cd ../bookstore-service
./mvnw -DskipTests package && \
cd ../bookstore-ui-feign-hystrix
./mvnw -DskipTests package && \
cd ../..
```

```
1. bash
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$ git clone https://github.com/derrick81/scs-clients-master.git && \
> cd scs-clients-master/employee-service && \
> ./mvnw -DskipTests package && \
> cd ../employee-feign-client
Cloning into 'scs-clients-master'...
remote: Enumerating objects: 138, done.
remote: Counting objects: 100% (138/138), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 138 (delta 29), reused 133 (delta 26), pack-reused 0
Receiving objects: 100% (138/138), 69.73 KiB | 157.00 KiB/s, done.
Resolving deltas: 100% (29/29), done.
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< com.example:employee-service >--------------------
[INFO] Building employee-service 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ employee-service ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
```

Pivotal

2. Run Spring Cloud Services locally

1. Open a **new terminal** and execute the following commands to run Spring Cloud Config:

```
cd ~/scs-ws/scs-config-master && \
./mvnw spring-boot:run
```

```
1. java
Derrick-Chuas-MacBook-Pro:scs-ws tmchua$ cd ~/scs-ws/scs-config-master && \
> ./mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< com.example:scs-config >-----------------------
[INFO] Building scs-config 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.1.0.RELEASE:run (default-cli) > test-compile @ scs-config >>>
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ scs-config ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
```

2. You should observe the following lines in the console output:

```
2019-04-19 15:50:17.377  INFO 7788 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebSe
rver  : Tomcat started on port(s): 8888 (http) with context path ''
2019-04-19 15:50:17.379  INFO 7788 --- [           main] c.e.scsconfig.ScsConfigApplication
      : Started ScsConfigApplication in 13.023 seconds (JVM running for 15.224)
```

3. Open a **new terminal** and execute the following commands to run Spring Cloud Eureka:

```
cd ~/scs-ws/scs-eureka-master && \
./mvnw spring-boot:run
```

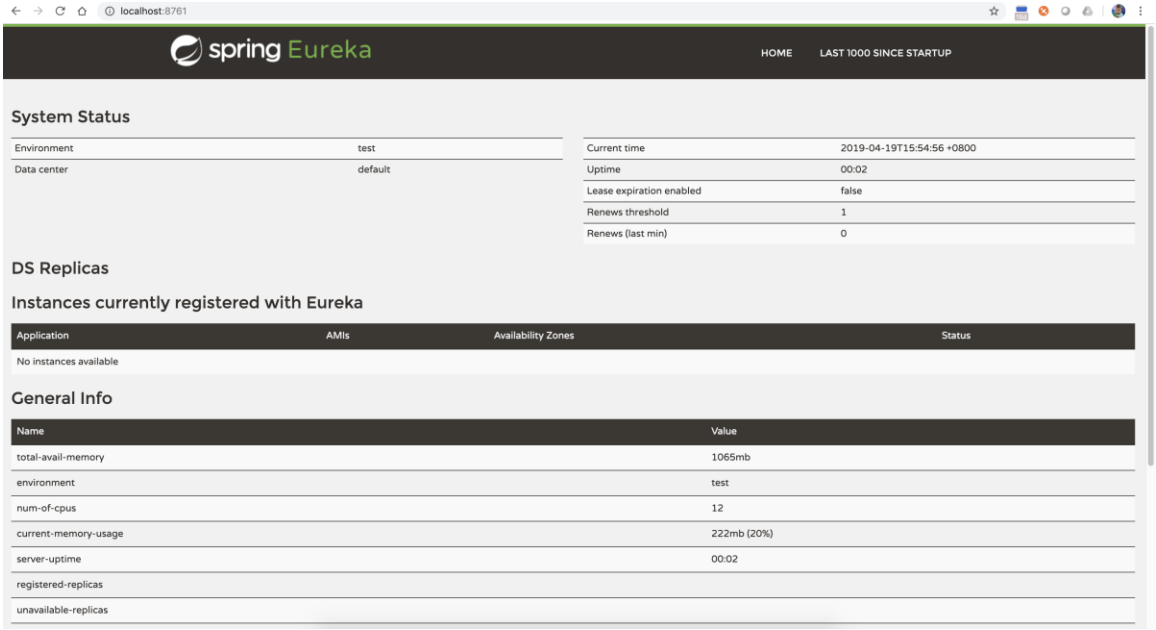
```
Derrick-Chuas-MacBook-Pro:~ tmchua$ cd ~/scs-ws/scs-eureka-master && \
> ./mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< com.example:scs-eureka >-----------------------
[INFO] Building scs-eureka 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.1.0.RELEASE:run (default-cli) > test-compile @ scs-eureka >>>
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ scs-eureka ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
```

4. You should observe the following lines in the console output:

```
2019-04-19 15:53:04.397  INFO 7820 --- [      Thread-14] e.s.EurekaServerInitializerConfiguration : Sta
rted Eureka Server
2019-04-19 15:53:04.414  INFO 7820 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port(s): 8761 (http) with context path ''
2019-04-19 15:53:04.415  INFO 7820 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Upd
ating port to 8761
2019-04-19 15:53:04.418  INFO 7820 --- [           main] c.e.scseureka.ScsEurekaApplication       : Sta
rted ScsEurekaApplication in 13.865 seconds (JVM running for 16.179)
```

5. Using a browser, navigate to the Spring Cloud Eureka web UI via the following URL:

http://localhost:8761



3. Run microservices and clients locally

1. Open a **new terminal** and execute the following commands to run Employee microservice:

```
cd ~/scs-ws/scs-clients-master/employee-service && \
./mvnw spring-boot:run
```

```
2019-04-19 15:56:10.286  INFO 7972 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_EMPLOYEE-SERVICE/192.168.2.90:employee-service:8081: registering service...
2019-04-19 15:56:10.335  INFO 7972 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_EMPLOYEE-SERVICE/192.168.2.90:employee-service:8081 - registration status: 204
2019-04-19 15:56:10.348  INFO 7972 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port(s): 8081 (http) with context path ''
2019-04-19 15:56:10.350  INFO 7972 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Upd
ating port to 8081
2019-04-19 15:56:10.356  INFO 7972 --- [           main] c.e.e.EmployeeServiceApplication         : Sta
rted EmployeeServiceApplication in 30.952 seconds (JVM running for 33.58)
```

2. Open a **new terminal** and execute the following commands to run Employee feign client:

```
cd ~/scs-ws/scs-clients-master/employee-feign-client && \
./mvnw spring-boot:run
```

```
2019-04-19 15:56:05.850  INFO 7981 --- [           main] com.netflix.discovery.DiscoveryClient    : Saw
 local status change event StatusChangeEvent [timestamp=1555660565850, current=UP, previous=STARTING]
2019-04-19 15:56:05.852  INFO 7981 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_EMPLOYEE-FEIGN-CLIENT/192.168.2.90:employee-feign-client:8082: registering service...
2019-04-19 15:56:05.934  INFO 7981 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port(s): 8082 (http) with context path ''
2019-04-19 15:56:05.939  INFO 7981 --- [           main] c.e.e.EmployeeFeignClientApplication     : Sta
rted EmployeeFeignClientApplication in 19.282 seconds (JVM running for 22.039)
2019-04-19 15:56:06.052  INFO 7981 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_EMPLOYEE-FEIGN-CLIENT/192.168.2.90:employee-feign-client:8082 - registration status: 204
```

3. Open a **new terminal** and execute the following commands to run Bookstore microservice:

```
cd ~/scs-ws/scs-clients-master/bookstore-service && \
./mvnw spring-boot:run
```

```
2019-04-19 15:56:12.818  INFO 7994 --- [           main] com.netflix.discovery.DiscoveryClient    : Saw
 local status change event StatusChangeEvent [timestamp=1555660572818, current=UP, previous=STARTING]
2019-04-19 15:56:12.819  INFO 7994 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_BOOKSTORE-SERVICE/192.168.2.90:bookstore-service:8083: registering service...
2019-04-19 15:56:12.861  INFO 7994 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_BOOKSTORE-SERVICE/192.168.2.90:bookstore-service:8083 - registration status: 204
2019-04-19 15:56:12.868  INFO 7994 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port(s): 8083 (http) with context path ''
2019-04-19 15:56:12.872  INFO 7994 --- [           main] c.e.b.BookstoreServiceApplication        : Sta
rted BookstoreServiceApplication in 19.052 seconds (JVM running for 21.835)
```

4. Open a **new terminal** and execute the following commands to run Bookstore feign hystrix client:

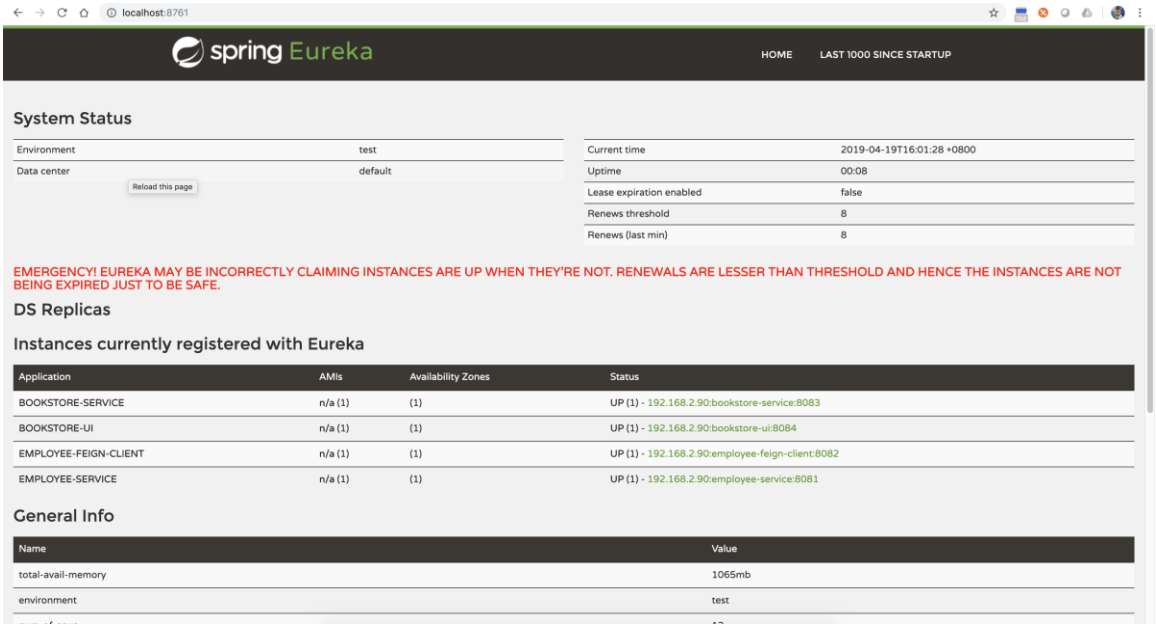
```
cd ~/scs-ws/scs-clients-master/bookstore-ui-feign-hystrix && \
./mvnw spring-boot:run
```

```
2019-04-19 15:56:22.256  INFO 8003 --- [           main] com.netflix.discovery.DiscoveryClient    : Saw
 local status change event StatusChangeEvent [timestamp=1555660582256, current=UP, previous=STARTING]
2019-04-19 15:56:22.258  INFO 8003 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_BOOKSTORE-UI/192.168.2.90:bookstore-ui:8084: registering service...
2019-04-19 15:56:22.295  INFO 8003 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : Dis
coveryClient_BOOKSTORE-UI/192.168.2.90:bookstore-ui:8084 - registration status: 204
2019-04-19 15:56:22.303  INFO 8003 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port(s): 8084 (http) with context path ''
2019-04-19 15:56:22.307  INFO 8003 --- [           main] ookstoreUiResttemplateHystrixApplication : Sta
rted BookstoreUiResttemplateHystrixApplication in 16.121 seconds (JVM running for 21.24)
```

5. Observe the Spring Cloud Config terminal console output and you should see the following:

```
2019-04-19 15:55:58.940  INFO 7788 --- [nio-8888-exec-1] o.s.c.c.s.e.NativeEnvironmentReposi
tory  : Adding property source: file:/var/folders/nw/dwd3y25930nggcm6yh3q947h0000gp/T/config
-repo-3141964626437442138/employee-service.properties
2019-04-19 15:55:59.800  INFO 7788 --- [nio-8888-exec-2] o.s.c.c.s.e.NativeEnvironmentReposi
tory  : Adding property source: file:/var/folders/nw/dwd3y25930nggcm6yh3q947h0000gp/T/config
-repo-3141964626437442138/employee-feign-client.properties
2019-04-19 15:56:18.455  INFO 7788 --- [nio-8888-exec-7] o.s.c.c.s.e.NativeEnvironmentReposi
tory  : Adding property source: file:/var/folders/nw/dwd3y25930nggcm6yh3q947h0000gp/T/config
-repo-3141964626437442138/bookstore-ui.properties
```

6. Refresh the Spring Cloud Eureka web UI and you should see the following:



7. You should also observe the registration from Spring Cloud Eureka's terminal console output:

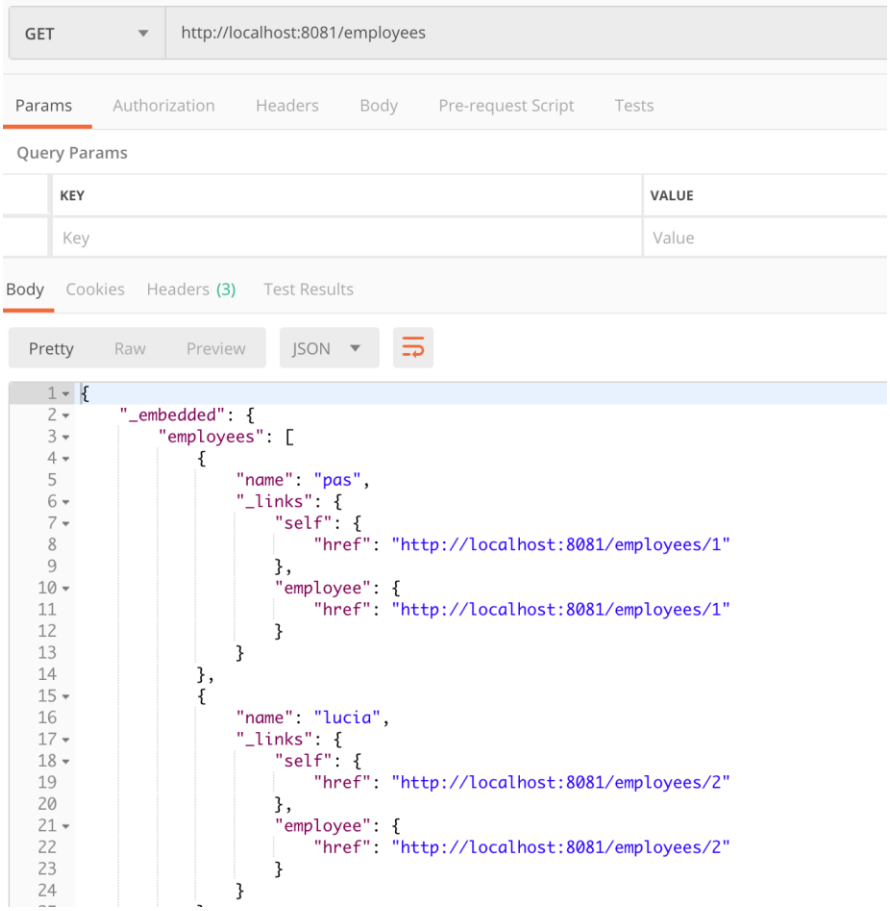
```
2019-04-19 15:56:06.049  INFO 7820 --- [nio-8761-exec-1] c.n.e.registry.AbstractInstanceRegistry  : Reg
istered instance EMPLOYEE-FEIGN-CLIENT/192.168.2.90:employee-feign-client:8082 with status UP (replicat
ion=false)
2019-04-19 15:56:10.334  INFO 7820 --- [nio-8761-exec-2] c.n.e.registry.AbstractInstanceRegistry  : Reg
istered instance EMPLOYEE-SERVICE/192.168.2.90:employee-service:8081 with status UP (replication=false)
2019-04-19 15:56:12.860  INFO 7820 --- [nio-8761-exec-6] c.n.e.registry.AbstractInstanceRegistry  : Reg
istered instance BOOKSTORE-SERVICE/192.168.2.90:bookstore-service:8083 with status UP (replication=fals
e)
2019-04-19 15:56:22.294  INFO 7820 --- [nio-8761-exec-9] c.n.e.registry.AbstractInstanceRegistry  : Reg
istered instance BOOKSTORE-UI/192.168.2.90:bookstore-ui:8084 with status UP (replication=false)
```

4. Test and observe locally

1. Invoke the employee microservice “employees” endpoint directly by executing the following in Postman.

   GET http://localhost:8081/employees

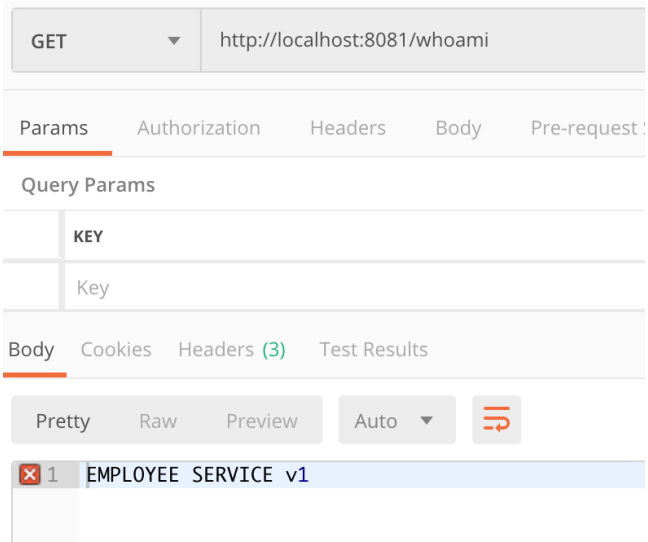
   You should observe the following if it is running successfully.



2. Invoke the employees microservice “whoami” endpoint directly by executing the following in Postman.

   GET http://localhost:8081/whoami

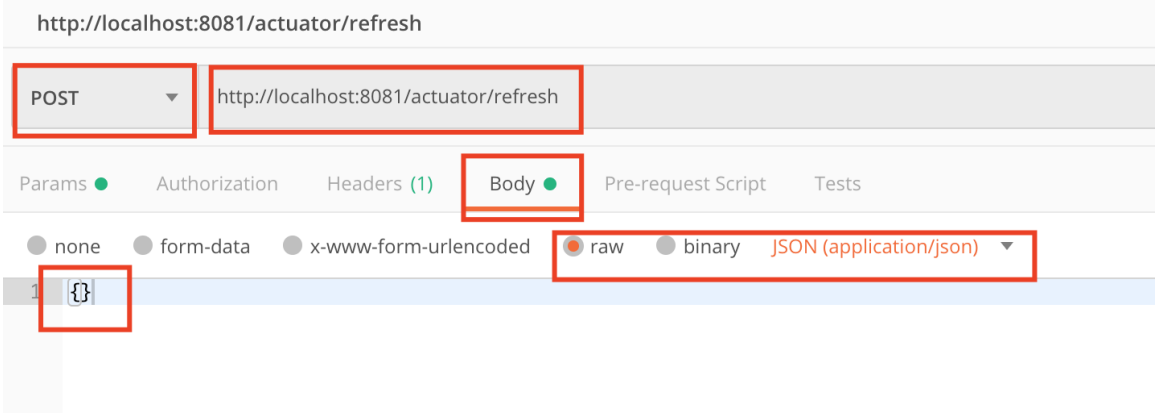
   You should observe the following if it is running successfully.



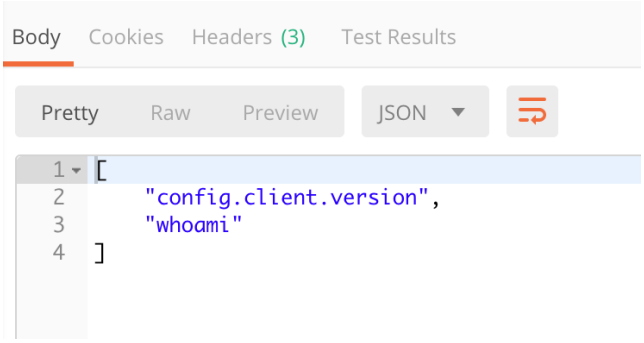
3. Wait for the instructor to update employee microservice config in the github config store to reflect "v2".

4. Refresh the employee microservice's copy of the configuration by executing the following command in Postman.

POST http://localhost:8081/actuator/refresh
Content-Type: application/json
Body: {}



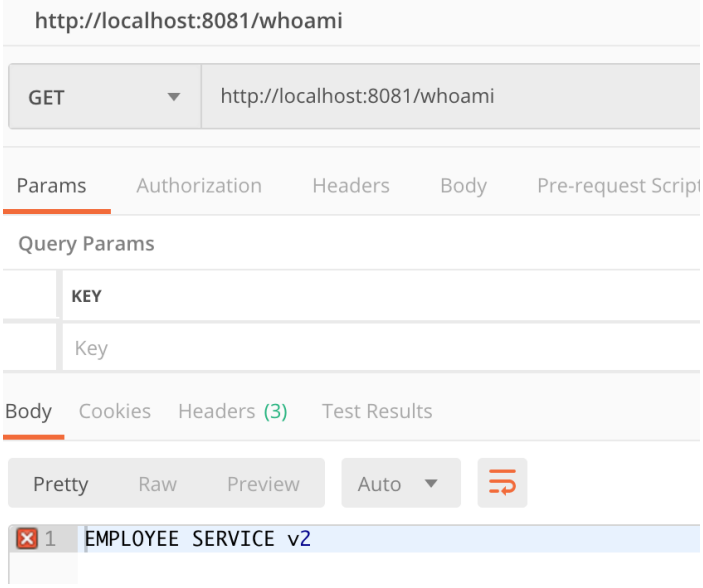
You should observe the following if executed successfully.



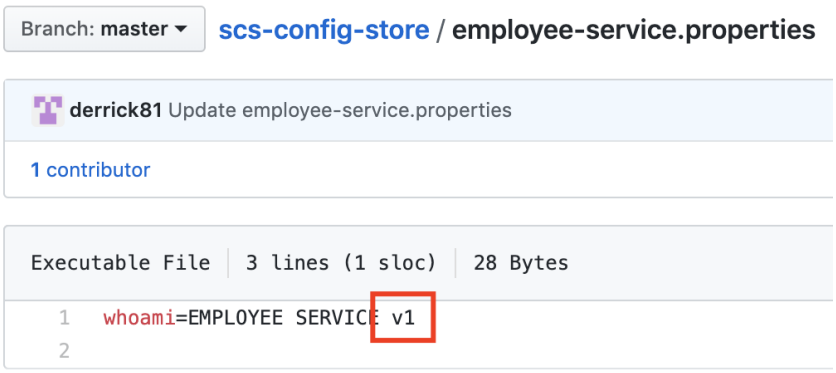
5. Invoke the employees microservice “whoami” endpoint directly by executing the following in Postman.

   GET http://localhost:8081/whoami

   You should observe the following (a change in the version number) if it is running successfully.



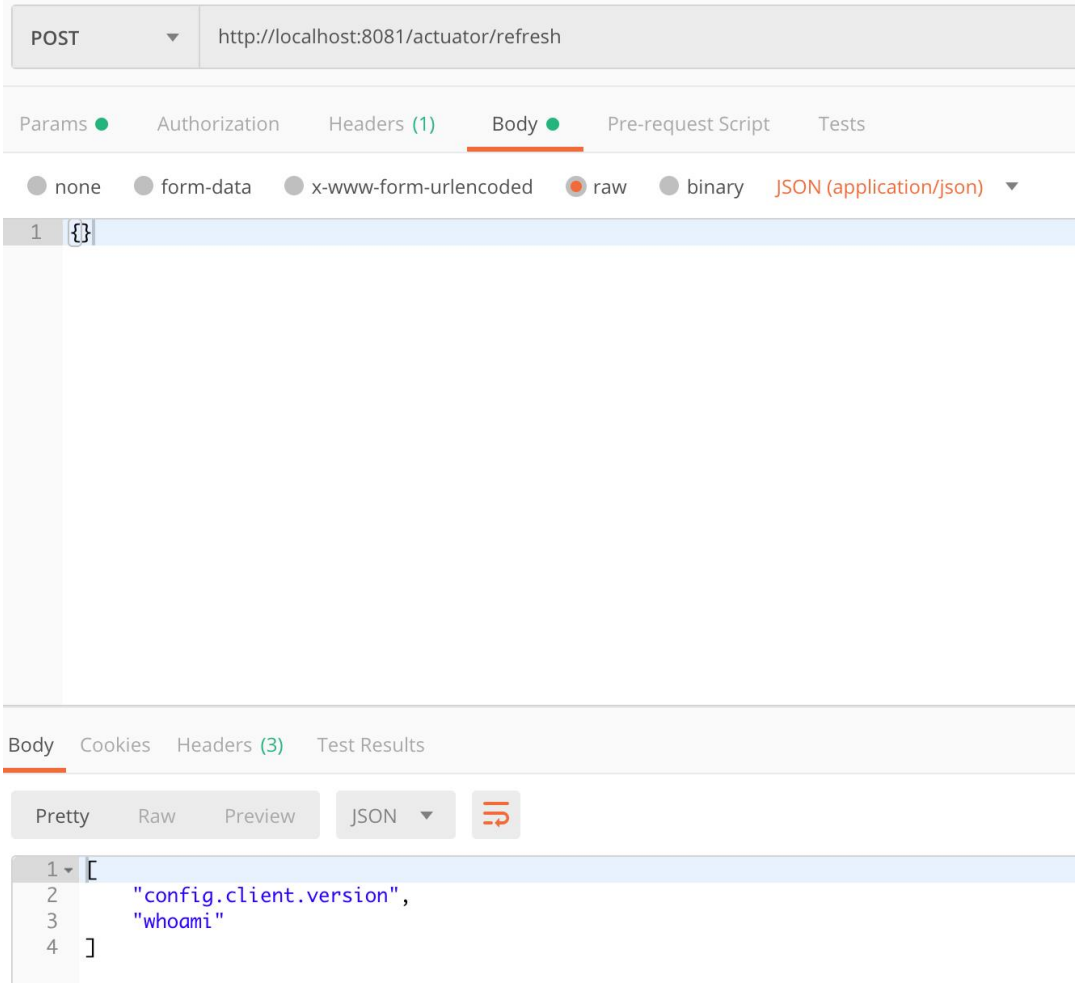
6. Wait for the instructor to revert employee microservice config in the github config store to reflect “v1”.



7. Refresh the employee microservice's copy of the configuration by executing the following command in Postman.

POST http://localhost:8081/actuator/refresh
Content-Type: application/json

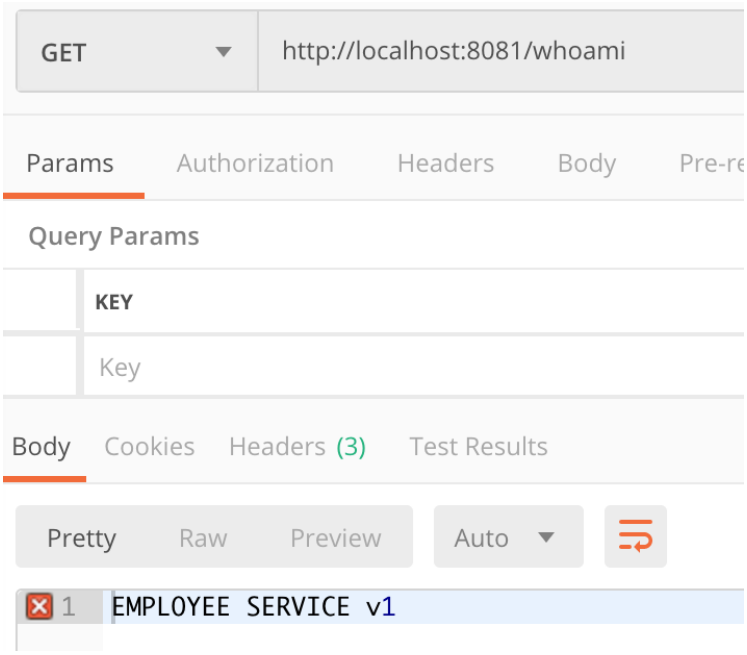
You should observe the following if executed successfully.



8. Invoke the employees microservice “whoami” endpoint directly by executing the following in Postman.

GET http://localhost:8081/whoami

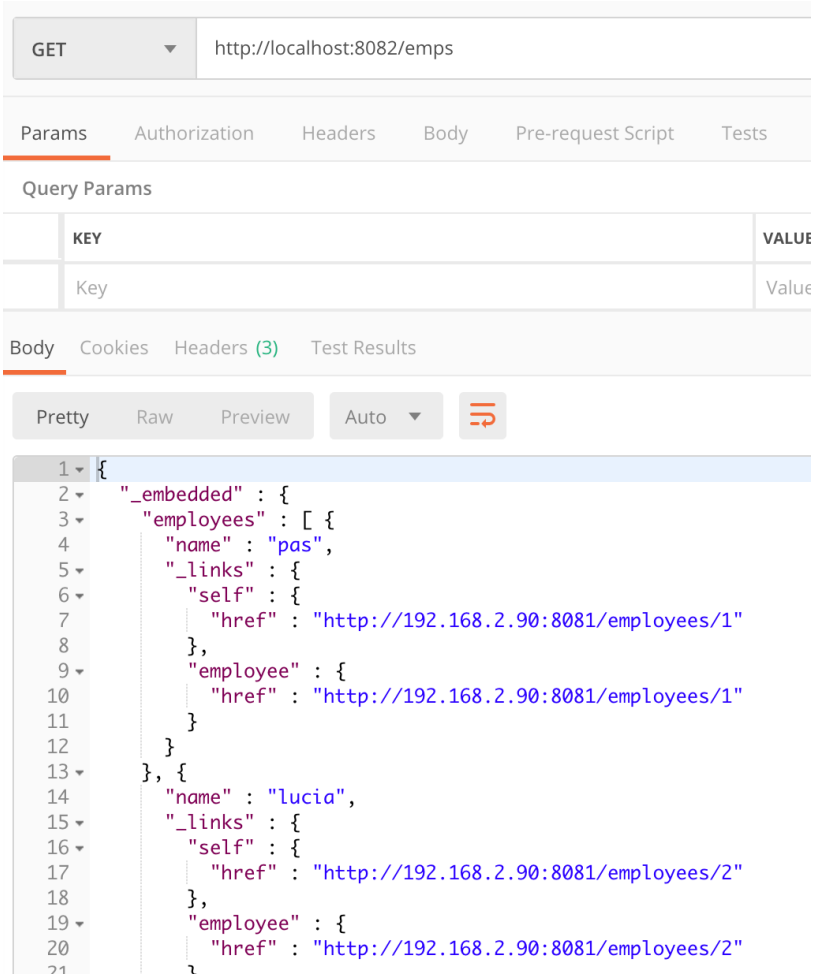
You should observe the following (a change in the version number) if it is running successfully.



9. Now consume the employee microservice through a feign client.
Invoke the client’s “employees” endpoint directly by executing the following in Postman.

GET http://localhost:**8082/emps**

You should observe the following if it is running successfully.



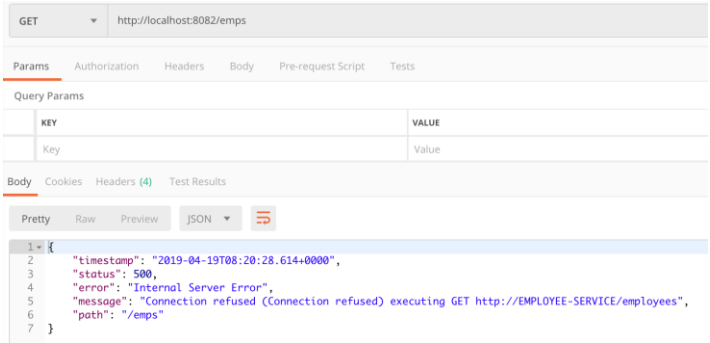
10. Now turn off the employee microservice to observe the clients response when the backing microservice is unavailable.
    Turn off the employee microservice by issuing the keyboard “Control+C” keystrokes on the terminal running the employee microservice.

```
2019-04-19 16:19:46.016  INFO 7972 --- [       Thread-9] com.zaxxer.hikari.HikariDataSource       : Hik
ariPool-2 - Shutdown initiated...
2019-04-19 16:19:46.018  INFO 7972 --- [       Thread-9] com.zaxxer.hikari.HikariDataSource       : Hik
ariPool-2 - Shutdown completed.
```

11. Invoke the client’s “employees” endpoint directly by executing the following in Postman.

    GET http://localhost:8082/emps

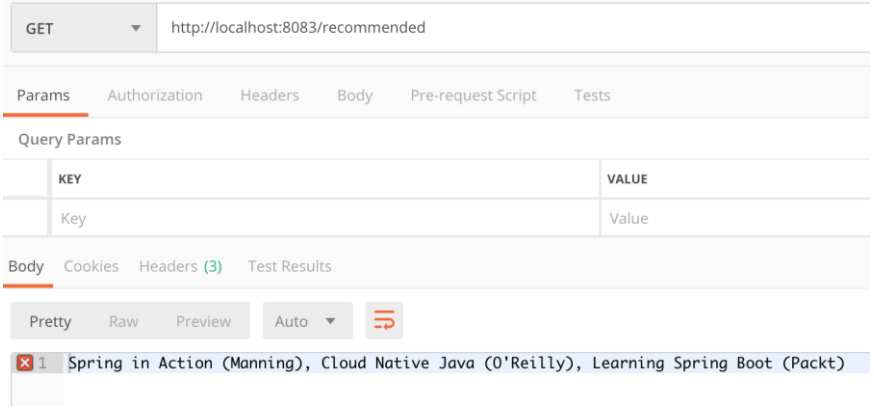
    You should observe the following failure.



12. Invoke the bookstore microservice “recommended” endpoint directly by executing the following in Postman.

    GET http://localhost:**8083/recommended**

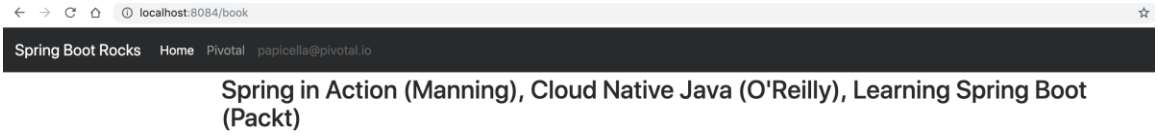
    You should observe the following if it is running successfully.



13. Invoke the bookstore web UI which will consume the bookstore microservice. Visit the following URL in the browser:

    http://localhost:**8084/book**

    You should see the following page.



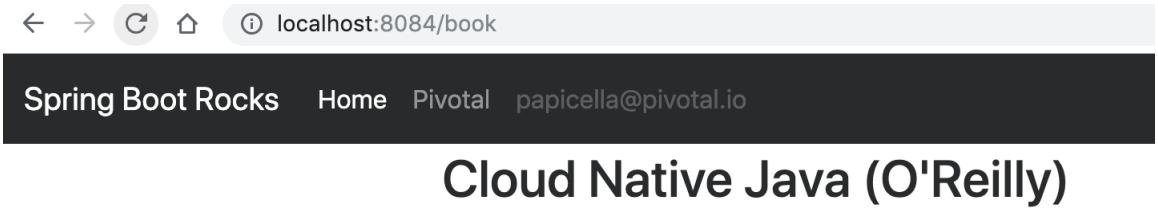
14. Now turn off the bookstore microservice to observe the clients response when the backing microservice is unavailable.
    Turn off the bookstore microservice by issuing the keyboard “Control+C” keystrokes on the terminal running the bookstore microservice.

```
2019-04-19 16:22:12.508  WARN 7994 --- [       Thread-9] .s.c.a.CommonAnnotationBeanPostProcessor : Des
troy method on bean with name 'scopedTarget.eurekaClient' threw an exception: org.springframework.beans
.factory.BeanCreationNotAllowedException: Error creating bean with name 'eurekaInstanceConfigBean': Sin
gleton bean creation not allowed while singletons of this factory are in destruction (Do not request a
bean from a BeanFactory in a destroy method implementation!)
[INFO]errick-Chuas-MacBook-Pro:bookstore-service tmchua$
```

15. Invoke the bookstore web UI which will consume the bookstore microservice. Visit the following URL in the browser:

    http://localhost:**8084/book**

    You should see the following page.



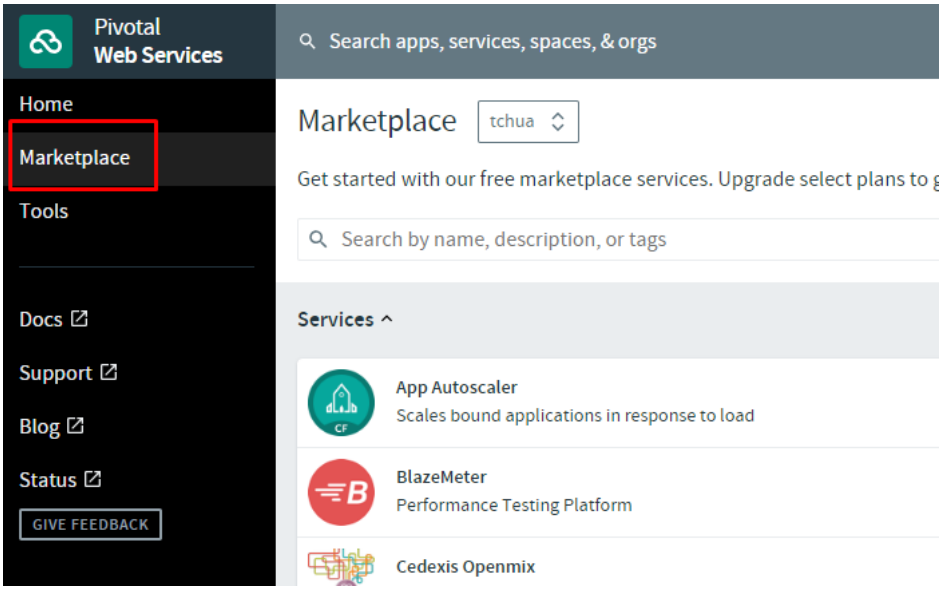
Cloud Native Java (O'Reilly)

5. Create Spring Cloud Services on PAS

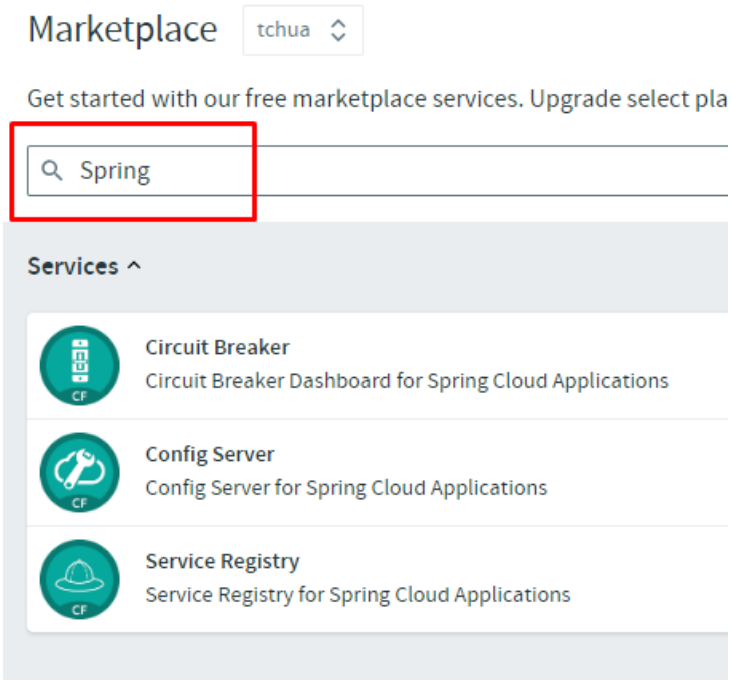
1. Navigate your browser to the following URL:

   https://console.run.pivotal.io

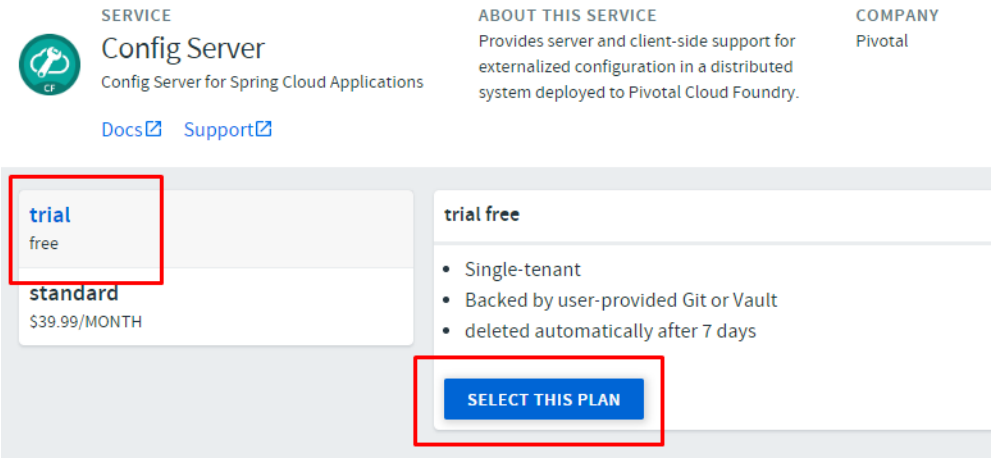
   Click on the “Marketplace” link.



2. Search using the keyword “Spring” and you should find the list of services narrowing to just the three Spring Cloud Services.



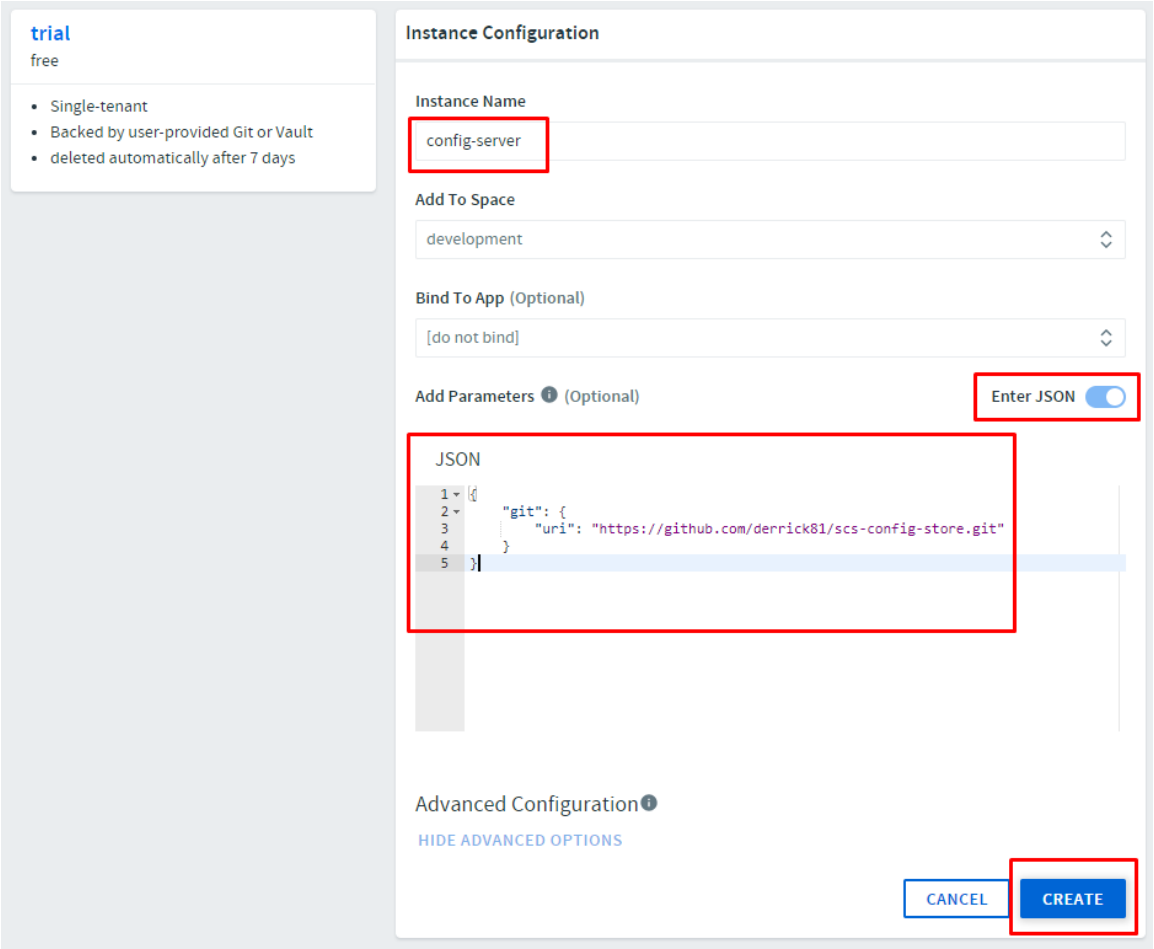
3. Click on “Config Server”. In the following screen, select the “trial” plan and then click on the “Select This Plan” button.



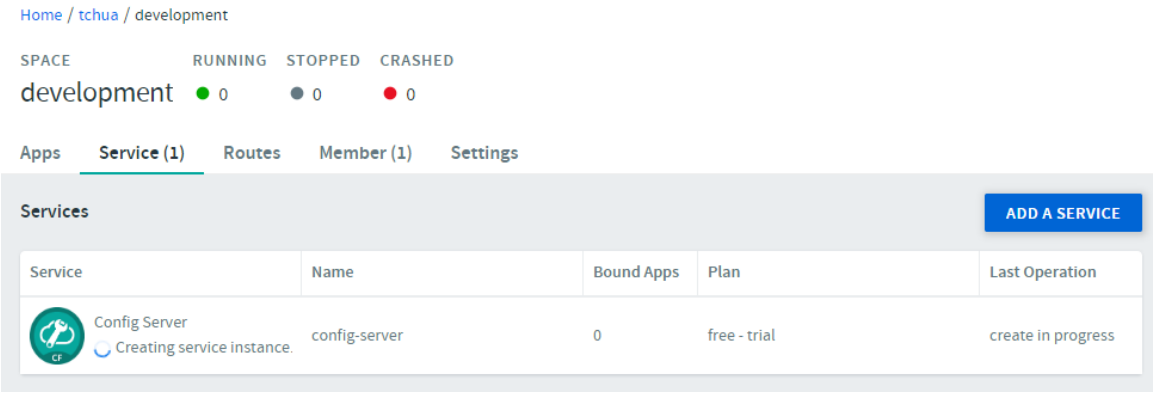
4. The instance configuration screen appears. Use the following values to create the service.

Instance name: config-server
Enter JSON: Enable
JSON value:

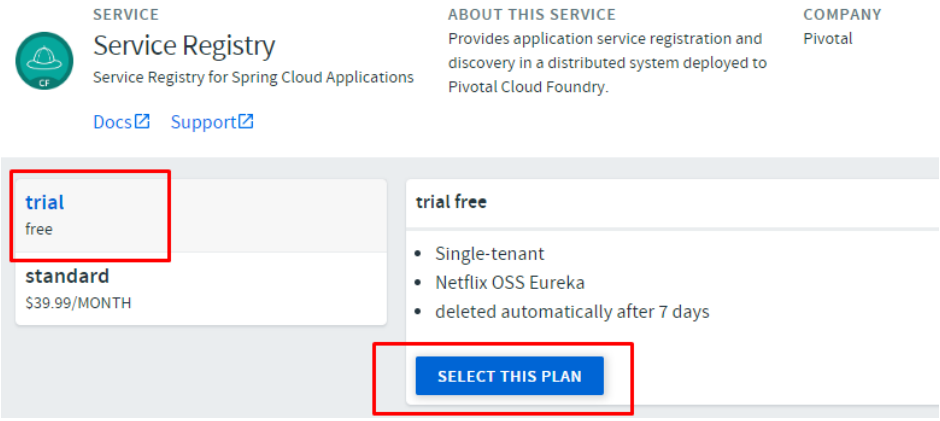
```
{
        "git": {
                "uri": "https://github.com/derrick81/scs-config-store.git"
        }
}
```



5. You should get redirected to the following screen.

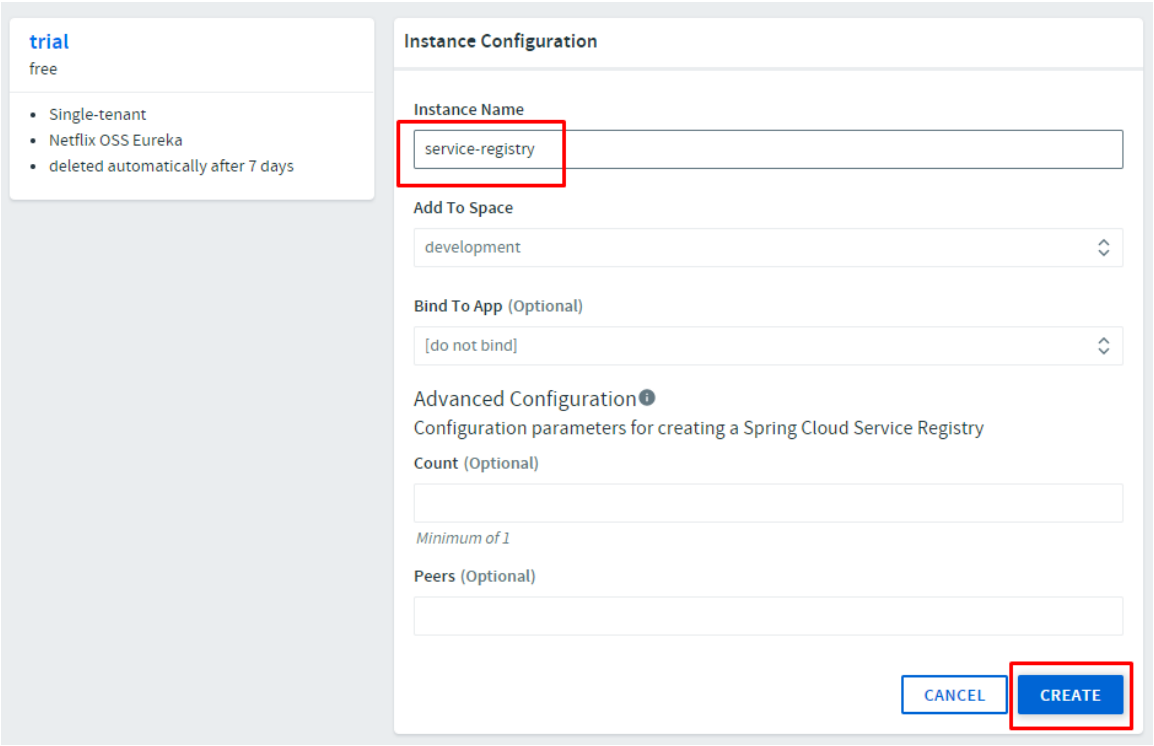


6. Go back to the Marketplace to create a Service Registry service. Similar to the Config Server, select the “trial” plan.

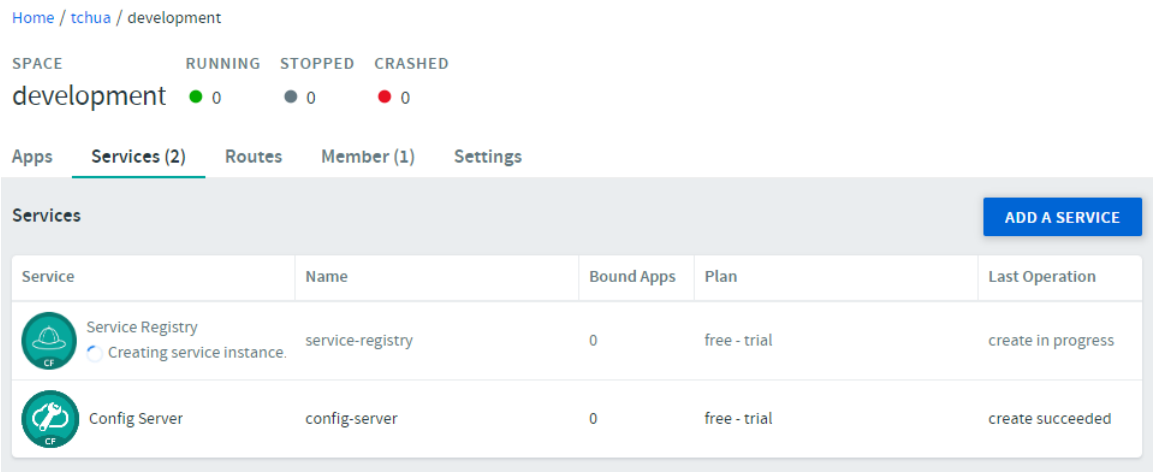


7. The instance configuration screen appears. Use the following values to create the service.

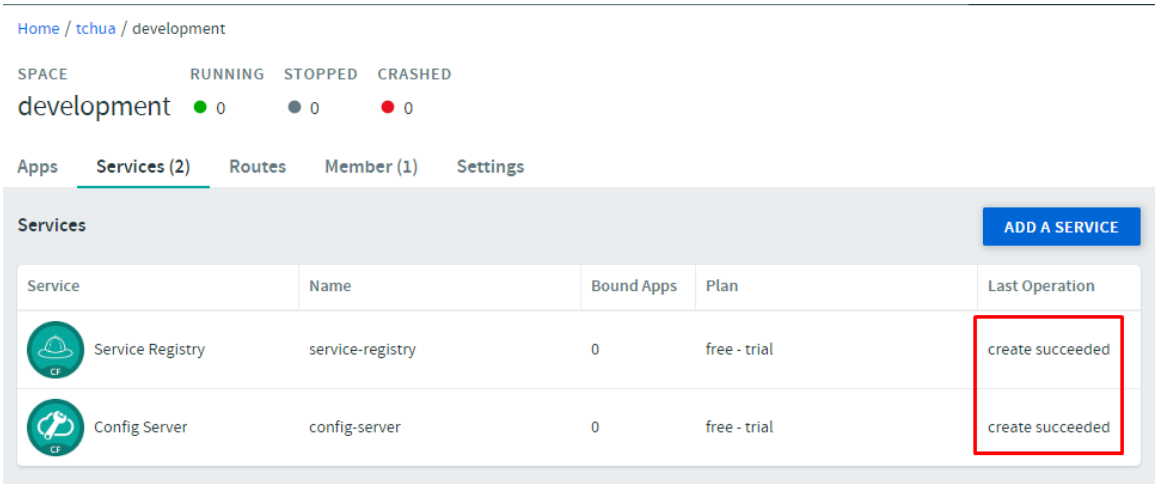
Instance name: service-registry



8. You will get redirected back to the “Services” tab of your space.



9. Wait for a couple of minutes and you should see both the Config Server and Service Registry created successfully.



6. Push microservices and clients to PAS

1. Log in to the cf cli using the following command, selecting the right Org and Space in the process.

   cf login -a https://api.run.pivotal.io

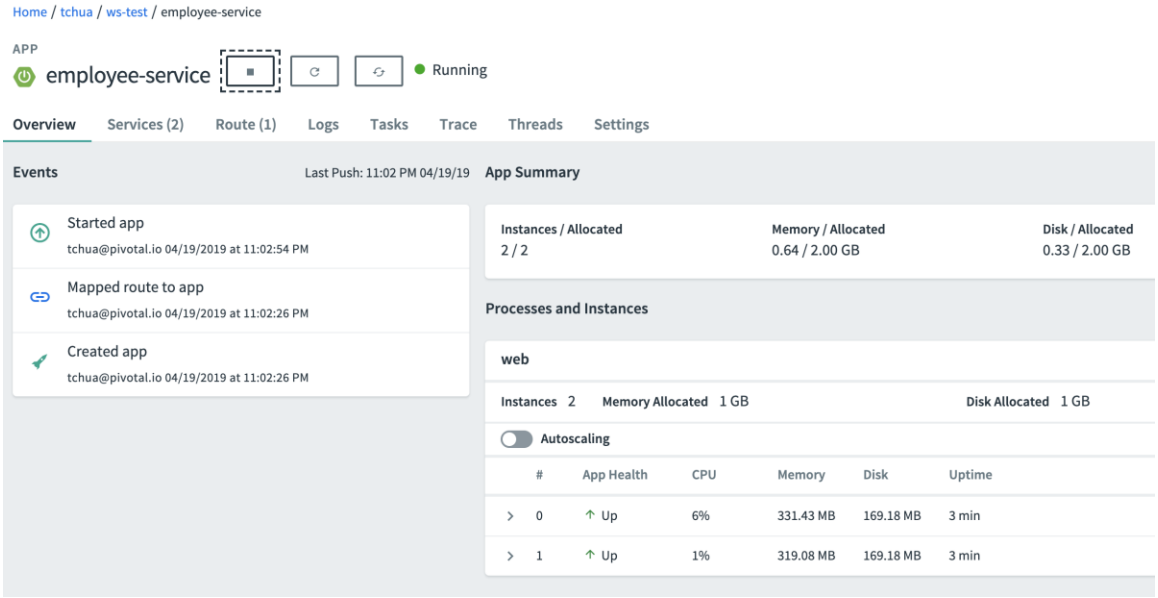
```
Derrick-Chuas-MacBook-Pro:employee-service tmchua$ cf login -a https://api.run.pivotal.io
API endpoint: https://api.run.pivotal.io
```

2. Execute the following chained commands in your terminal to push the Employee microservice to PAS:

   cd ~/scs-ws/scs-clients-master/employee-service && \
   cf push

```
1. cf
Derrick-Chuas-MacBook-Pro:employee-service tmchua$ cd ~/scs-ws/scs-clients-master/employee-service && \
> cf push
Pushing from manifest to org tchua / space ws-test as tchua@pivotal.io...
Using manifest file /Users/tmchua/scs-ws/scs-clients-master/employee-service/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:        employee-service
  path:        /Users/tmchua/scs-ws/scs-clients-master/employee-service/target/employee-service-0.0.1-SNAPSHOT.jar
+ instances:   2
+ memory:      1G
  services:
+   config-server
+   service-registry
  env:
+   JAVA_OPTS
+   TRUST_CERTS
  routes:
+   employee-service-thankful-genet.cfapps.io
```

3. Verify in the Apps Manager that the Employee microservice has been deployed successfully.

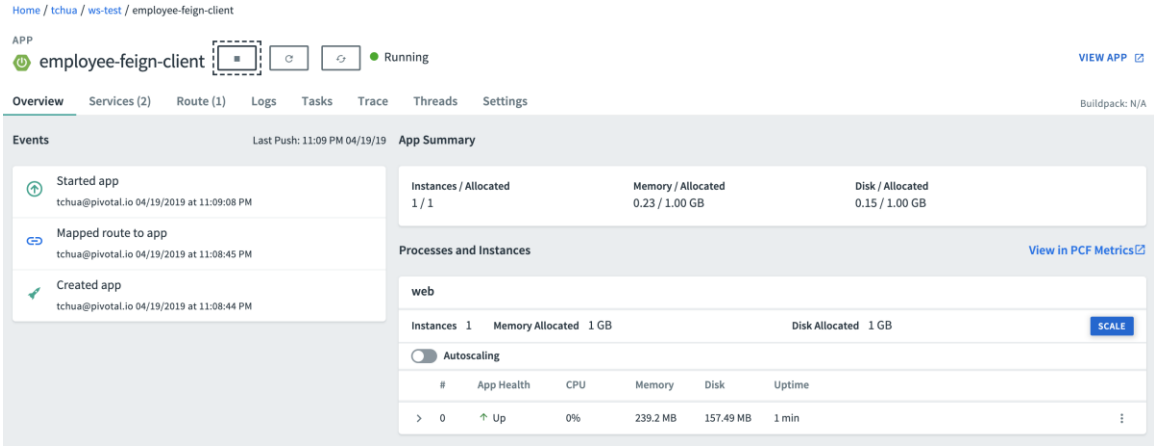


4. Execute the following chained commands in your terminal to push the Employee feign client to PAS:

```
cd ~/scs-ws/scs-clients-master/employee-feign-client && \
cf push
```

```
Derrick-Chuas-MacBook-Pro:employee-service tmchua$ cd ~/scs-ws/scs-clients-master/employee-feign-client && \
> cf push
Pushing from manifest to org tchua / space ws-test as tchua@pivotal.io...
Using manifest file /Users/tmchua/scs-ws/scs-clients-master/employee-feign-client/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:        employee-feign-client
  path:        /Users/tmchua/scs-ws/scs-clients-master/employee-feign-client/target/employee-feign-client-0.0.1-SNAPSHOT.jar
+ instances:   1
+ memory:      1G
  services:
+   config-server
+   service-registry
  env:
+   JAVA_OPTS
+   TRUST_CERTS
  routes:
+   employee-feign-client-wacky-chimpanzee.cfapps.io
```

5. Verify in the Apps Manager that the Employee feign client has been deployed successfully.

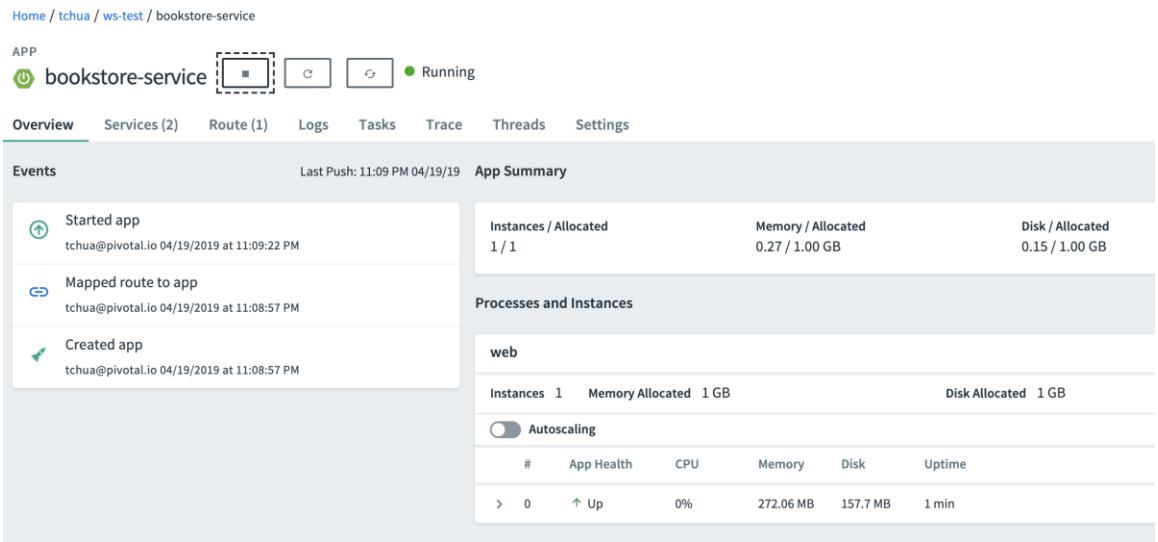


6. Execute the following chained commands in your terminal to push the Bookstore microservice to PAS:

```
cd ~/scs-ws/scs-clients-master/bookstore-service && \
cf push
```

```
Derrick-Chuas-MacBook-Pro:~ tmchua$ cd ~/scs-ws/scs-clients-master/bookstore-service && \
> cf push

Pushing from manifest to org tchua / space ws-test as tchua@pivotal.io...
Using manifest file /Users/tmchua/scs-ws/scs-clients-master/bookstore-service/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:        bookstore-service
  path:        /Users/tmchua/scs-ws/scs-clients-master/bookstore-service/target/bookstore-service-0.0.1-SNAPSHOT.jar
+ instances:   1
+ memory:      1G
  services:
+   config-server
+   service-registry
  env:
+   JAVA_OPTS
+   TRUST_CERTS
  routes:
+   bookstore-service-unexpected-baboon.cfapps.io
```

7. Verify in the Apps Manager that the Bookstore microservice has been deployed successfully.

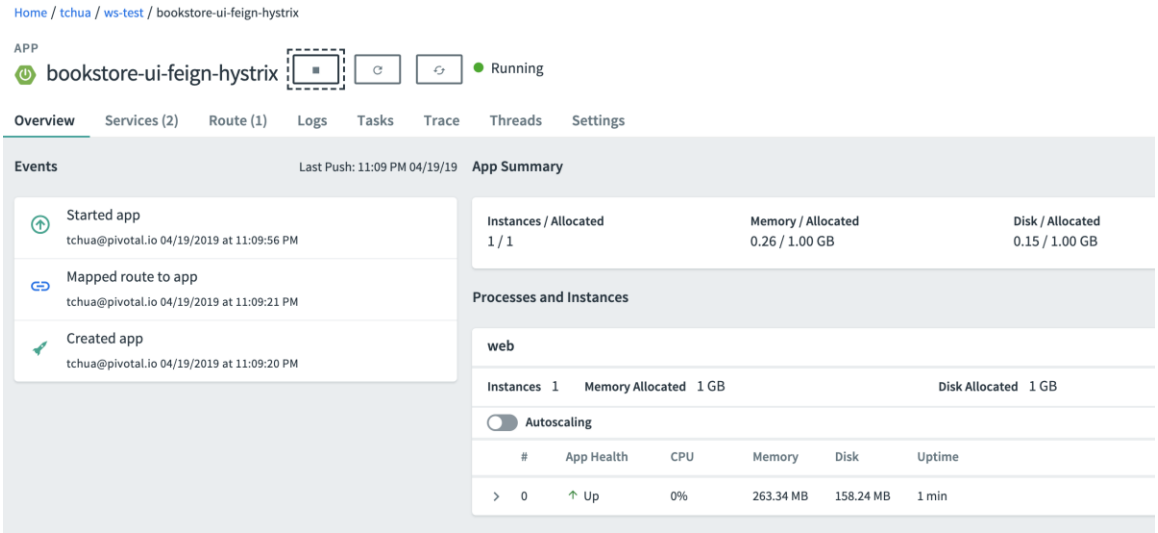


8. Execute the following chained commands in your terminal to push the Bookstore feign hystrix client to PAS:

cd ~/scs-ws/scs-clients-master/bookstore-ui-feign-hystrix && \
cf push

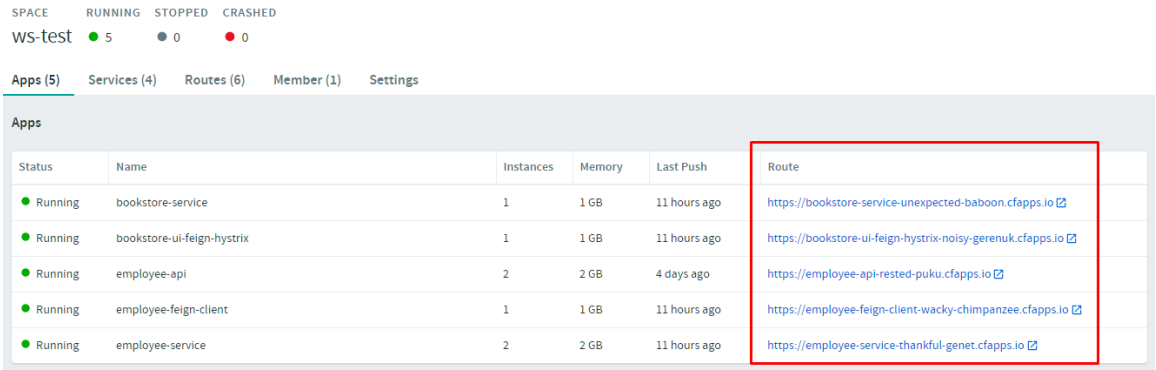
```
Derrick-Chuas-MacBook-Pro:~ tmchua$ cd ~/scs-ws/scs-clients-master/bookstore-ui-feign-hystrix && \
> cf push

Pushing from manifest to org tchua / space ws-test as tchua@pivotal.io...
Using manifest file /Users/tmchua/scs-ws/scs-clients-master/bookstore-ui-feign-hystrix/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:        bookstore-ui-feign-hystrix
  path:        /Users/tmchua/scs-ws/scs-clients-master/bookstore-ui-feign-hystrix/target/bookstore-ui-resttemplate-hystrix-0.0.1-SNAPSHOT.jar
+ instances:   1
+ memory:      1G
  services:
+   config-server
+   service-registry
  env:
+   JAVA_OPTS
+   TRUST_CERTS
  routes:
+   bookstore-ui-feign-hystrix-noisy-gerenuk.cfapps.io
```

9. Verify in the Apps Manager that the Bookstore feign hystrix client has been deployed successfully.



7. Test and observe on Cloud

1. Take note of the route for each of the four applications that were pushed to PAS by going to the Apps tab in your space. Note that there will be five apps because of the Employee API from the previous workshop exercise.



SPACE ws-test | RUNNING 5 | STOPPED 0 | CRASHED 0

Apps (5) Services (4) Routes (6) Member (1) Settings

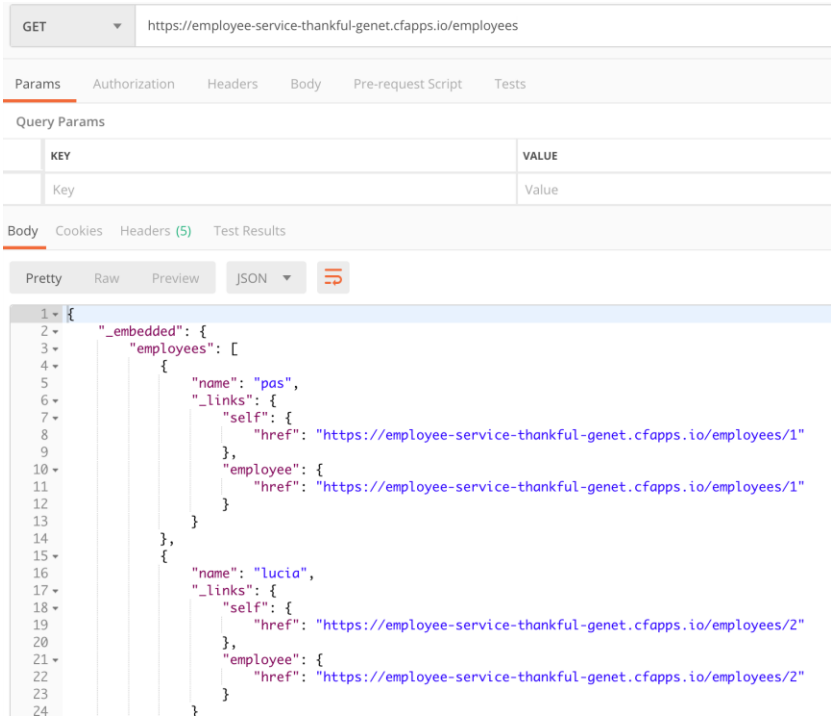
Apps

Status	Name	Instances	Memory	Last Push	Route
Running	bookstore-service	1	1 GB	11 hours ago	https://bookstore-service-unexpected-baboon.cfapps.io
Running	bookstore-ui-feign-hystrix	1	1 GB	11 hours ago	https://bookstore-ui-feign-hystrix-noisy-gerenuk.cfapps.io
Running	employee-api	2	2 GB	4 days ago	https://employee-api-rested-puku.cfapps.io
Running	employee-feign-client	1	1 GB	11 hours ago	https://employee-feign-client-wacky-chimpanzee.cfapps.io
Running	employee-service	2	2 GB	11 hours ago	https://employee-service-thankful-genet.cfapps.io

2. Invoke the employee microservice “employees” endpoint directly by executing the following in Postman.

GET https://{employee.microservice.route}/employees

You should observe the following if it is running successfully.



GET https://employee-service-thankful-genet.cfapps.io/employees

Params Authorization Headers Body Pre-request Script Tests

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

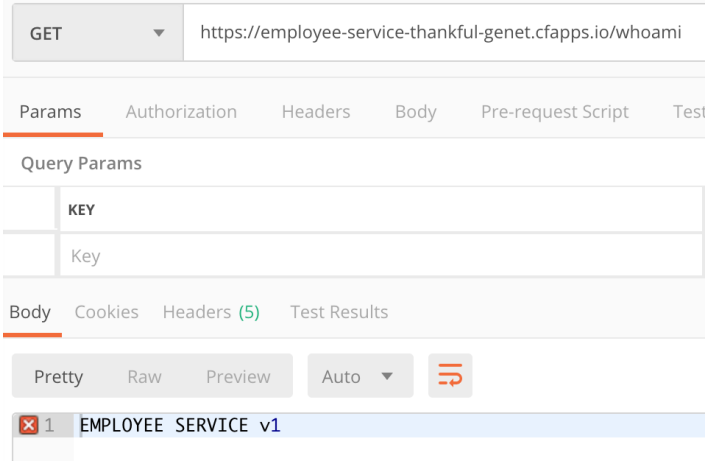
Pretty Raw Preview JSON


```
{
    "_embedded": {
        "employees": [
            {
                "name": "pas",
                "_links": {
                    "self": {
                        "href": "https://employee-service-thankful-genet.cfapps.io/employees/1"
                    },
                    "employee": {
                        "href": "https://employee-service-thankful-genet.cfapps.io/employees/1"
                    }
                }
            },
            {
                "name": "lucia",
                "_links": {
                    "self": {
                        "href": "https://employee-service-thankful-genet.cfapps.io/employees/2"
                    },
                    "employee": {
                        "href": "https://employee-service-thankful-genet.cfapps.io/employees/2"
                    }
                }
```

3. Invoke the employees microservice “whoami” endpoint directly by executing the following in Postman.

GET https://{employee.microservice.route}/whoami

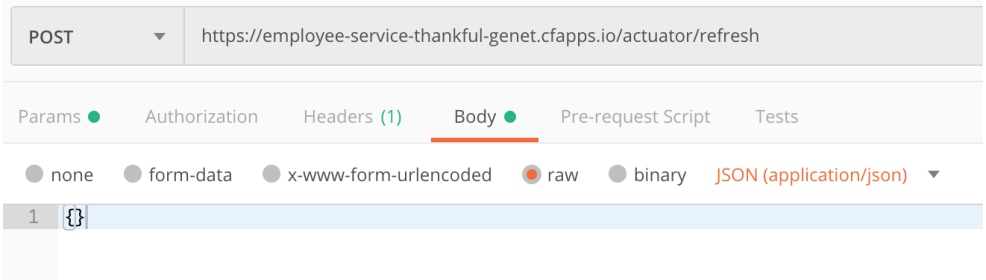
You should observe the following if it is running successfully.



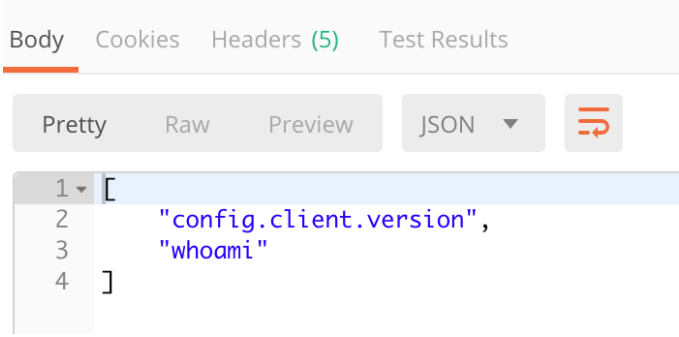
4. Wait for the instructor to update employee microservice config in the github config store to reflect “v2”.

5. Refresh the employee microservice’s copy of the configuration by executing the following command in Postman.

POST https://{employee.microservice.route}/actuator/refresh
Content-Type: application/json
Body: {}



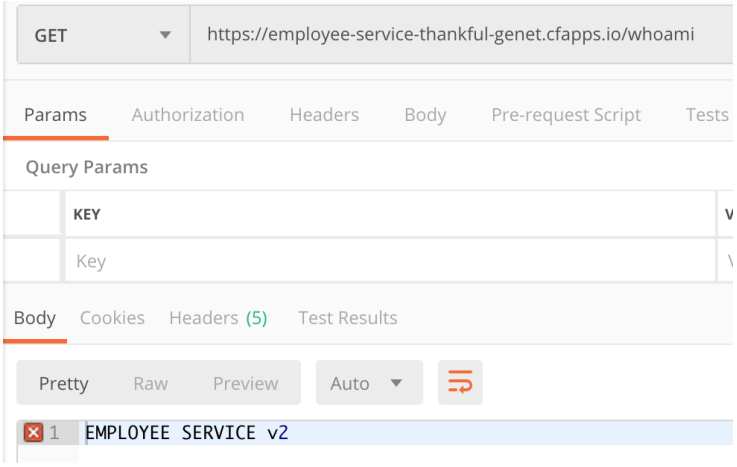
You should observe the following if executed successfully.



6. Invoke the employees microservice “whoami” endpoint directly by executing the following in Postman.

   GET https://{employee.microservice.route}/whoami

   You should observe the following (a change in the version number) if it is running successfully.

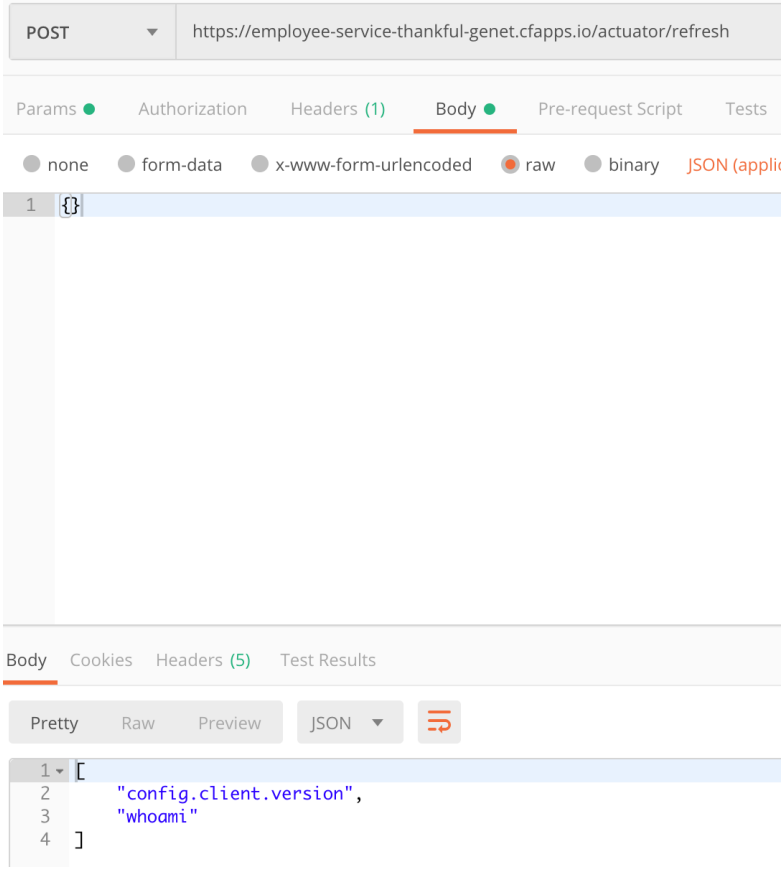


7. Wait for the instructor to revert employee microservice config in the github config store to reflect “v1”.

8. Refresh the employee microservice's copy of the configuration by executing the following command in Postman.

POST https://{employee.microservice.route}/actuator/refresh
Content-Type: application/json
Body: {}

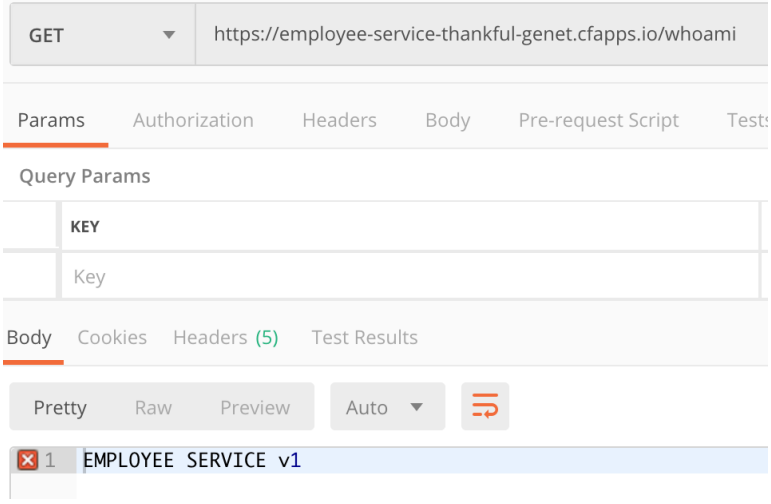
You should observe the following if executed successfully.



9. Invoke the employees microservice "whoami" endpoint directly by executing the following in Postman.

GET https://{employee.microservice.route}/whoami

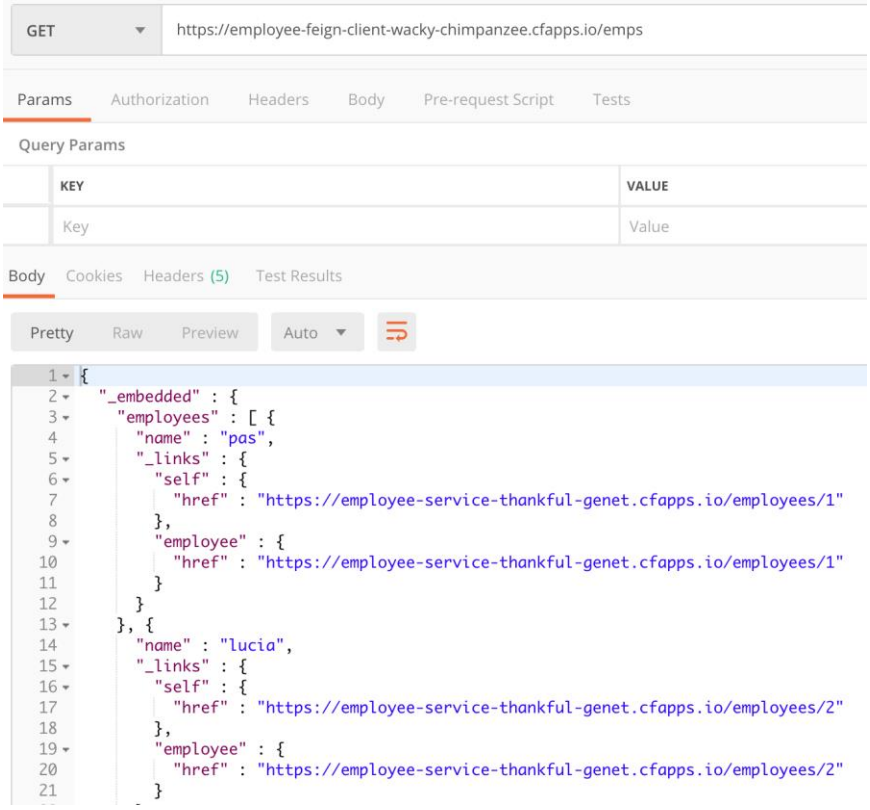
You should observe the following (a change in the version number) if it is running successfully.



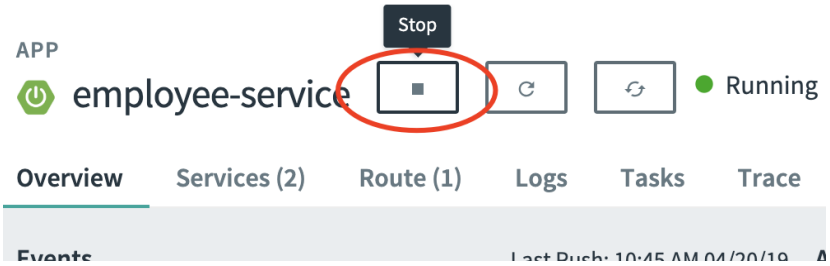
10. Now consume the employee microservice through a feign client.
Invoke the client's "employees" endpoint directly by executing the following in Postman.

GET https://{employee.feign.client.route}**/emps**

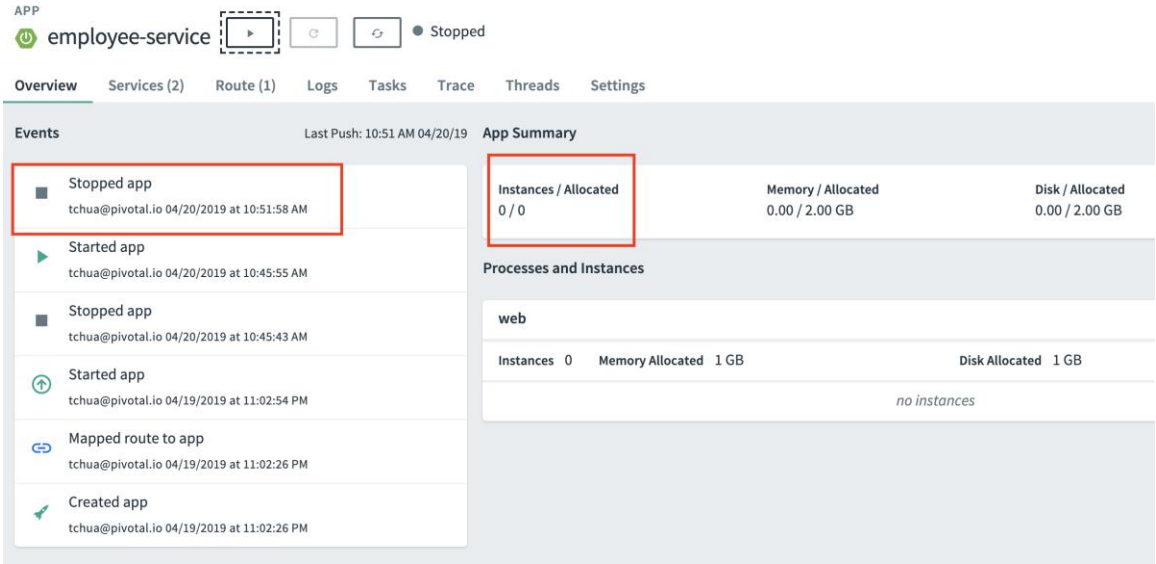
You should observe the following if it is running successfully.



11. Now turn off the employee microservice to observe the clients response when the backing microservice is unavailable.
Turn off the employee microservice by clicking on the “Stop” button on the employee microservice’s app page.



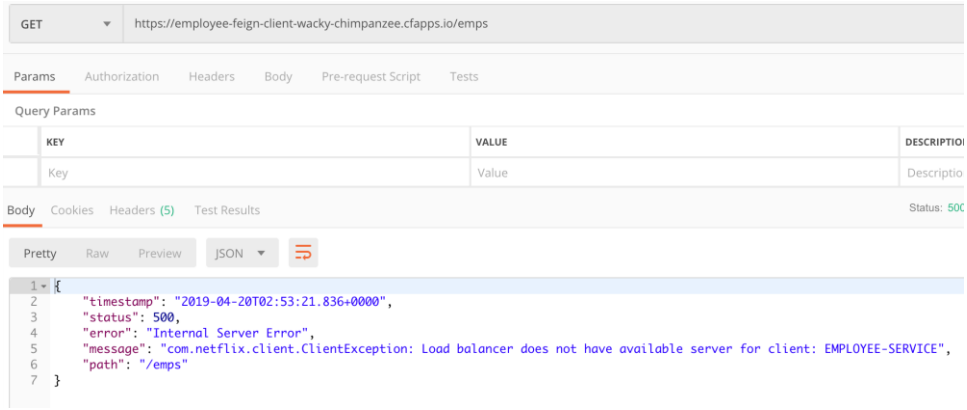
After confirming the stop, you should see that the app has indeed stopped running.



12. Invoke the client’s “employees” endpoint directly by executing the following in Postman.

GET https://{employee.feign.client.route}/emps

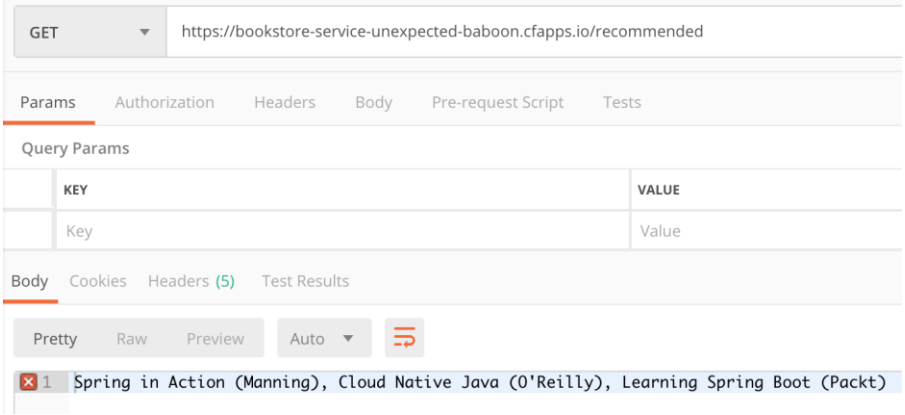
You should observe the following failure.



13. Invoke the bookstore microservice “recommended” endpoint directly by executing the following in Postman.

GET https://{bookstore.microservice.route}**/recommended**

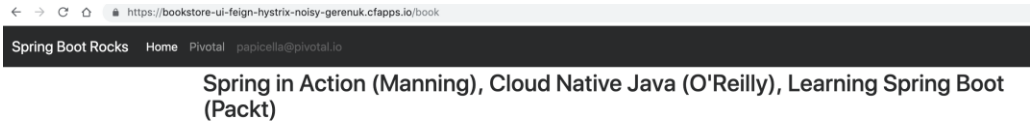
You should observe the following if it is running successfully.



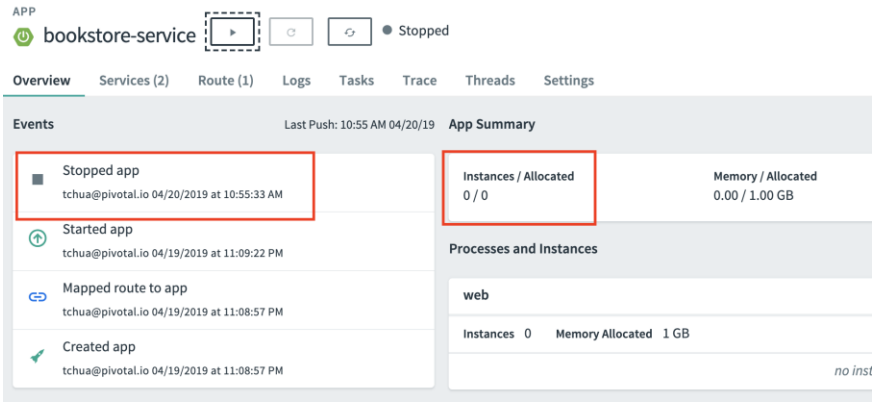
14. Invoke the bookstore web UI which will consume the bookstore microservice. Visit the following URL in the browser:

https://{bookstore.ui.feign.hystrix.route}**/book**

You should see the following page.



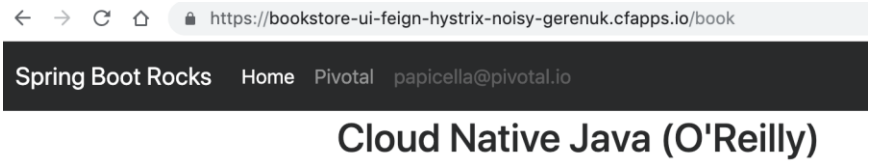
15. Now turn off the bookstore microservice to observe the clients response when the backing microservice is unavailable.
Turn off the employee microservice by clicking on the “Stop” button on the bookstore microservice’s app page.



16. Invoke the bookstore web UI which will consume the bookstore microservice. Visit the following URL in the browser:

https://{bookstore.ui.feign.hystrix.route}**/book**

You should see the following page.



Cloud Native Java (O'Reilly)