[GitHub - eswaribala/boamsbatch1jul2023](https://github.com/eswaribala/boamsbatch1jul2023)

[GitHub - eswaribala/boamstrainingrepo](https://github.com/eswaribala/boamstrainingrepo)

Starting Vault Server –

C:\Users\Administrator\Desktop\training workspace\vault\_1.14.0\_windows\_amd64>vault server --config=vault.conf

Vault Server – <http://localhost:8200>

RabbitMQ for cloud bus –

docker run -d -t -it --hostname my-rabbitmq --name rabbitmq3-server -p 15672:15672 -p 5672:5672 -e RABBITMQ\_DEFAULT\_USER=guest -e RABBITMQ\_DEFAULT\_PASS=guest rabbitmq:3-management

Rabbit MQ - <http://localhost:15672/>

Eureka Server (Service Registry) –

docker run --publish 8761:8761 steeltoeoss/eureka-server

Eureka Console - <http://localhost:8761/>

Config Server project – configserver project

Config Server - <http://localhost:8888/boa-common.properties>

Zuul Gateway project – apigateway project

Zuul Gateway Console - <http://localhost:8765/api/customer>

Customer API - customerapiexternal project

Customer API console - <http://localhost:7074/>, <http://localhost:7076/>

Resilience4j demo project - resilience4jdemo project

Resilience4j demo console – <http://localhost:7077>

C:\Users\Administrator\eclipse-workspace

Eric Evans – Domain Driven development (DDD)

Domain – E-Commerce

Sub Domain – Cart, Order management, Delivery, Inventory etc.

Domain – Banking

Sub Domain – CRM, Forex, Lending, Insurance, Mortgage, Trading, Investments

Our POC deals with CRM.

Customer and Bank Accounts have different services.

Bank Account Services – deposit, withdraw, view balance, fund transfer, overdraft etc.

Customer Account Services – Create, update, view bank accounts

These services being offered are called contexts. The contexts are bounded by the Single responsibility-based principle.

SOLID Design Principles

* S - Single Responsibility
* O – Open for extension, closed for modification
* L – Liskov Substitution
* I – Interface Segregation
* D – Dependency Inversion

Contexts will interact with each other like withdrawal needs to check if OD is available, there has been previous deposits. Etc. This is called a Context Map.

Every context has models like deposit context will have models like account, account features etc. These models will have attributes.

Install Lombok plugin into eclipse.

[localhost:7074/individuals/v1.0/?mediaType=xml](http://localhost:7074/individuals/v1.0/?mediaType=json)

[localhost:7074/individuals/v1.0/?mediaType=json](http://localhost:7074/individuals/v1.0/?mediaType=json)

query{

getAllIndividuals {

email,

name {

firstName

middleName

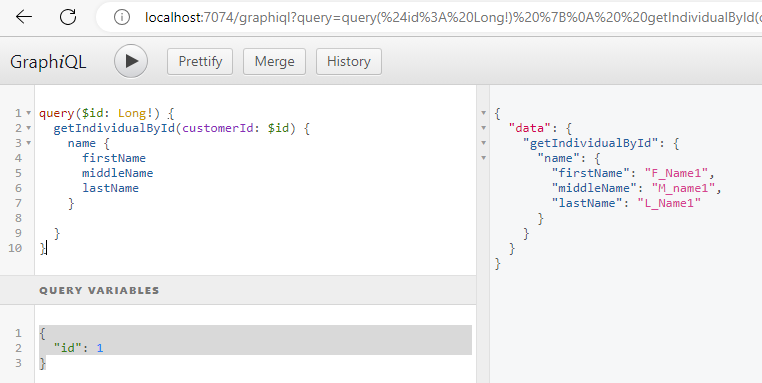
lastName

},

password

}

}

! Indicates that this parameter is mandatory.

GraphQL APIs are different than RESTful APIs. Traditional endpoint concept is alleviated as part of the graphQL query. For GraphQL API, the endpoint is always /graphiql.

While working with GraphQL, the Controller does not come into picture at all.

When resolving using graphql, code looks up

graphql.tools.schema-location-pattern: \*\*/\*.graphqls

from the properties file and then parses all the .graphqls files to get the required schema and queries. Now, it looks for the GraphQLQueryResolver implementations in the project which in our case are IndividualGLQuery.java and CustomerGLQuery.java files as they are auto scanned having been annotated as Component.

mutation($individualInput: IndividualInput!) {

addIndividual(individualInput: $individualInput){

customerId

}

}

{

"individualInput": {

"customerId": 0,

"email": "abc@3083.com",

"password": "pass123",

"contactNo": 9876652812,

"name": {

"firstName": "dfljdsklf",

"middleName": "fdssdfs",

"lastName": "dfssd"

},

"dob": "1970-02-02",

"gender": "FEMALE"

}

}

In application.properties of customerapiexternal project, we define that we want to refer the boa-common and boa-app peropeties.

spring.application.name=Customer-API,boa-common,boa-app

driver and url for db come from config server. Username and password are coming from vault server.

<!-- vault client -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-vault-config</artifactId>

</dependency>

The above starter config will look for a vault server at localhost and port of 8200. If these are to be changed, the same needs to be mentioned in bootstrap.properties file.

<!-- config client -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bootstrap</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

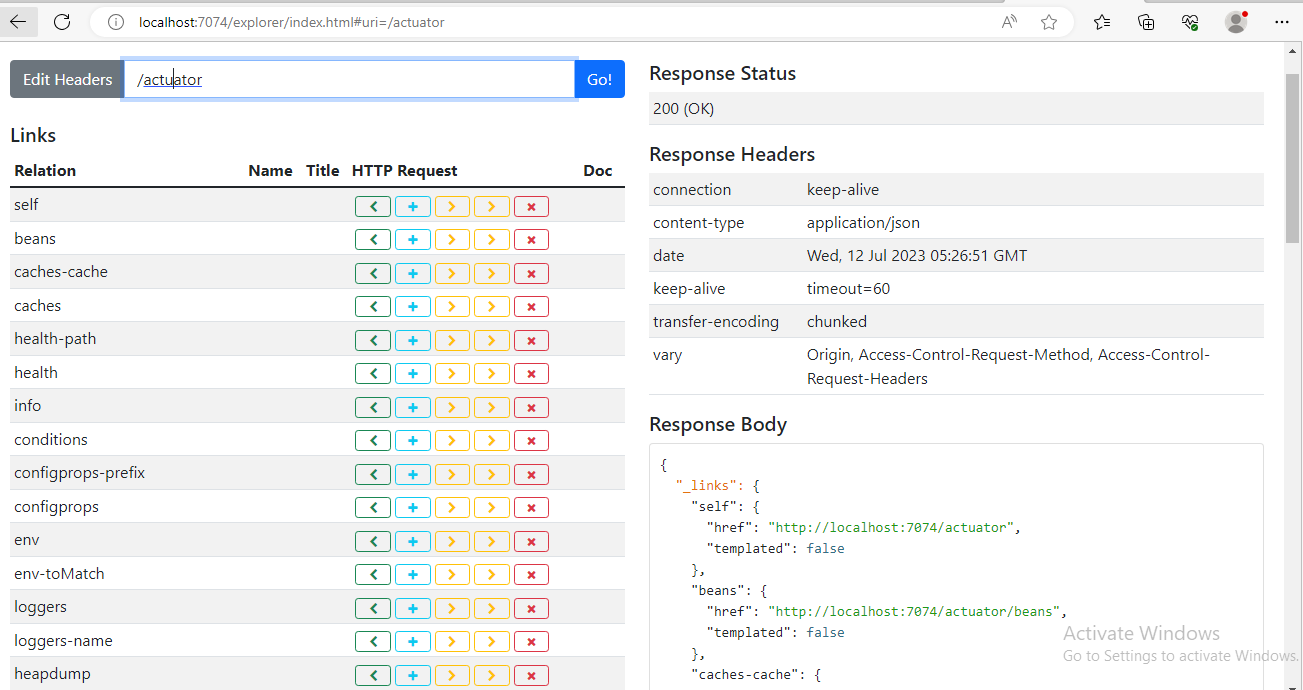
</dependency>

Similarly, the config server starter client looks up the configuration property files form config server at localhost 8888 as mentioned in the application.properties. in bootstrap properties file, we have mentioned the names of the property files like boa-common, boa-app. Otherwise, it will parse all the property files in the github repo of config server.

The below property

management.endpoints.web.exposure.include=\*

exposes the actuator endpoints for monitoring.



When the services are up and running and a property change has happened in the git repository of the config server, the services need to be triggered using /actuator/bus-refresh endpoint.

Bus refresh feature has a bug with spring 2.7 so we are having to move back to spring boot 2.3

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.12</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.boa</groupId>

<artifactId>customerapiexternal</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>customerapiexternal</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.8</spring-cloud.version>

</properties>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bootstrap</artifactId>

</dependency>

To

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.12.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.boa</groupId>

<artifactId>customerapiexternal</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>customerapiexternal</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>Hoxton.SR6</spring-cloud.version>

</properties>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!--<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bootstrap</artifactId>

</dependency>-->

For cloud Refresh, we need a cloud bus and need Rabbit MQ for it. We are pulling RabbitMQ image from docker for now.

There is a rabbitmq.txt file under boamssbatchijuly2023 master branch.

docker run -d -t -it --hostname my-rabbitmq --name rabbitmq3-server -p 15672:15672 -p 5672:5672 -e RABBITMQ\_DEFAULT\_USER=guest -e RABBITMQ\_DEFAULT\_PASS=guest rabbitmq:3-management

In this command, 15672 is the http port while 5672 is he TCP port.

The username and password are guest/guest.

Check from <http://localhost:15672> and login using guest/guest.

Docker un creates the container for which hostname, servername, http port, tcp port, username/password are all given as part of the command.

We can also see that the docker desktop shows the rabbitmq container in the containers menu.

Now we need to add cloud bus library nto the pom.xml

The properties for rabbit mq are mentioned in the boacommon.properties present in config server.

spring.rabbitmq.host: localhost

spring.rabbitmq.port: 5672

spring.rabbitmq.username: guest

spring.rabbitmq.password: guest

The console on startup gives the following in console saying that a queue has started.

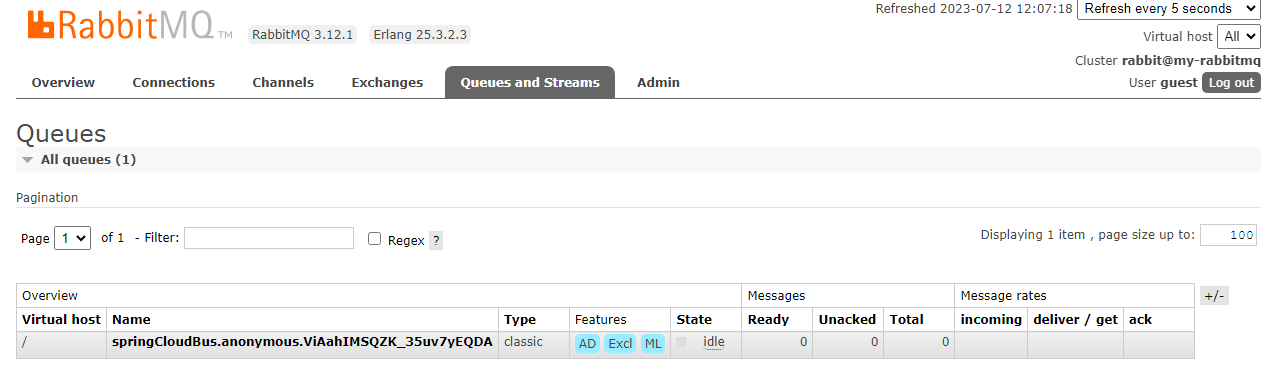
2023-07-12 11:56:33.277 INFO 5820 --- [ restartedMain] o.s.i.endpoint.EventDrivenConsumer : Adding {logging-channel-adapter:\_org.springframework.integration.errorLogger} as a subscriber to the 'errorChannel' channel

2023-07-12 11:56:33.277 INFO 5820 --- [ restartedMain] o.s.i.channel.PublishSubscribeChannel : Channel 'mysqlsecret-1.errorChannel' has 1 subscriber(s).

2023-07-12 11:56:33.277 INFO 5820 --- [ restartedMain] o.s.i.endpoint.EventDrivenConsumer : started bean '\_org.springframework.integration.errorLogger'

2023-07-12 11:56:33.279 INFO 5820 --- [ restartedMain] o.s.c.s.binder.DefaultBinderFactory : Creating binder: rabbit

It can be seen from the rabbit mq console –



Now we check by updating the “message” property of bo-common.properties file in config server and reading and logging ti from IndividualController.

message: Team checking bus refresh eventv1800

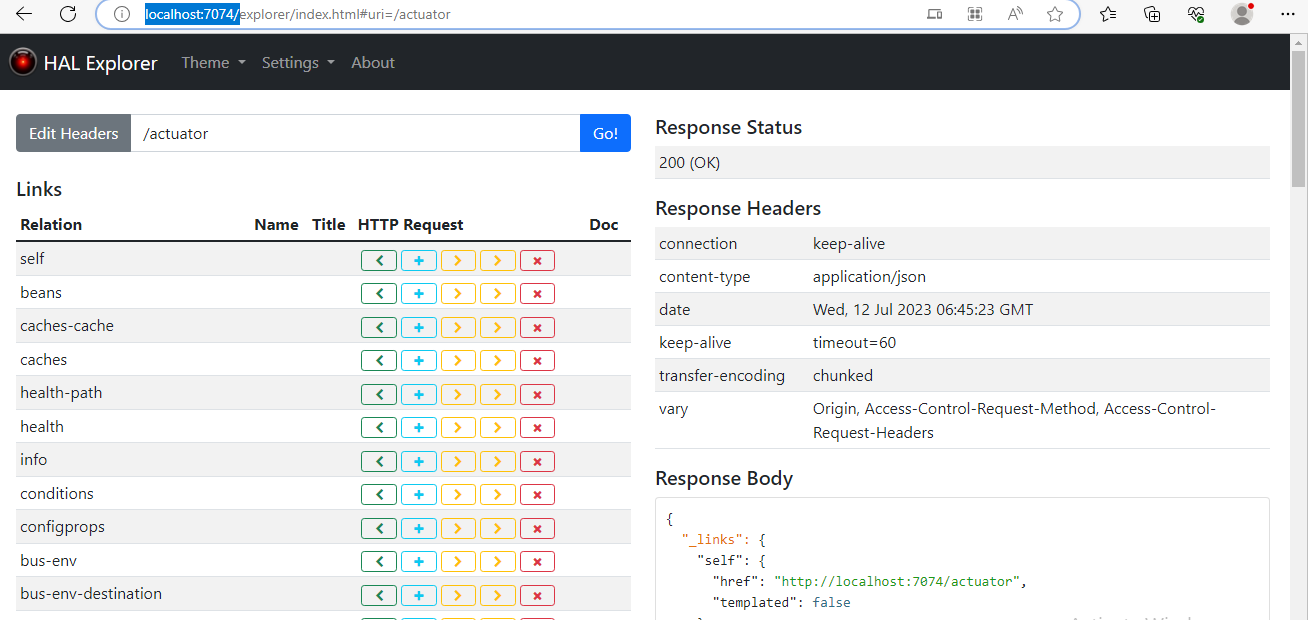
First, we see that this gets printed when we call getAllIndividuals method from swagger-ui.

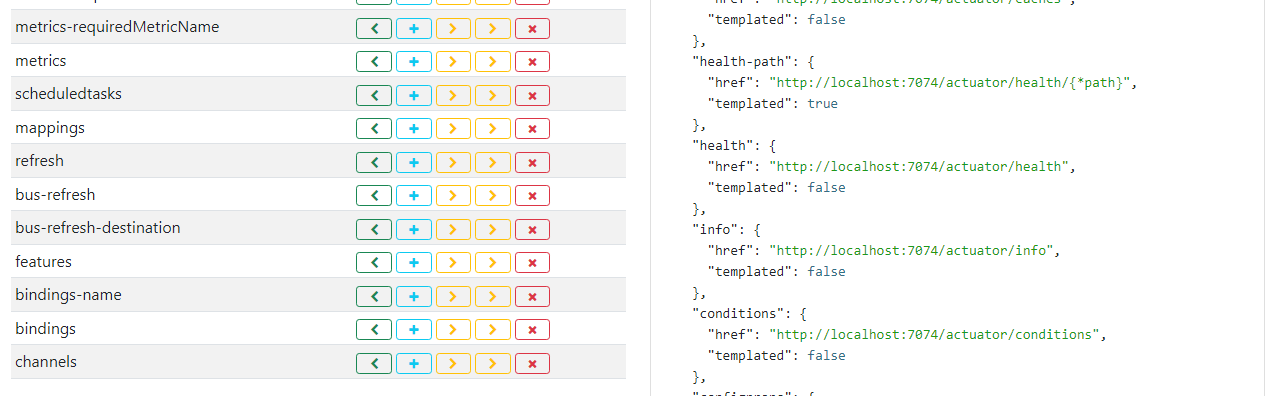
The message gets printed in the console as –

2023-07-12 11:59:12.782 INFO 5820 --- [nio-7074-exec-4] c.b.c.controllers.IndividualController : Message received from githubTeam checking bus refresh eventv1800

On changing the message, the change is not reflected because there has not been any bus refresh trigger yet.

We need to trigger it by sending a POST request to http:/localhost:7074/actuator/bus-refresh without any payload. This can also be done using the actuator endpoints from HAL browser by pressing the + sign beside the bus-refresh endpoint.





Once this is done, we see the below messages in the console that shows that the message is refreshed.

2023-07-12 12:12:45.389 INFO 5820 --- [nio-7074-exec-6] c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at : http://localhost:8888

2023-07-12 12:12:46.331 INFO 5820 --- [nio-7074-exec-6] c.c.c.ConfigServicePropertySourceLocator : Located environment: name=Customer-API,boa-common,boa-app, profiles=[dev], label=null, version=a731ad8c9187f87c9ad6a9e387df464e087f6854, state=null

2023-07-12 12:12:46.331 INFO 5820 --- [nio-7074-exec-6] b.c.PropertySourceBootstrapConfiguration : Located property source: [BootstrapPropertySource {name='bootstrapProperties-configClient'}, BootstrapPropertySource {name='bootstrapProperties-https://github.com/eswaribala/boamstrainingrepo.git/boa-app-dev.properties'}, BootstrapPropertySource {name='bootstrapProperties-https://github.com/eswaribala/boamstrainingrepo.git/boa-common.properties'}]

2023-07-12 12:12:46.333 INFO 5820 --- [nio-7074-exec-6] o.s.boot.SpringApplication : The following profiles are active: dev

2023-07-12 12:12:46.340 INFO 5820 --- [nio-7074-exec-6] o.s.boot.SpringApplication : Started application in 1.349 seconds (JVM running for 990.538)

2023-07-12 12:12:46.345 INFO 5820 --- [nio-7074-exec-6] o.s.s.c.ThreadPoolTaskScheduler : Shutting down ExecutorService

2023-07-12 12:12:46.668 INFO 5820 --- [nio-7074-exec-6] o.s.cloud.bus.event.RefreshListener : Keys refreshed [config.client.version, message]

Note that here we notified the service running in port 7074 about the property change via the trigger. Once this is done, the property change event will be broadcasted to all the services which are subscribed to the service bus. For this, we need to have all the services listening to the same queue in Rabbit MQ. We can specify the name of the queue in boa-common-properties where we had configured the host and port.

The triggering can also be done manually instead of the bus-refresh method.

Docker command for setting up a eurke server –

docker run --publish 8761:8761 steeltoeoss/eureka-server

Check at <http://localhost:8761>

Now the C:\Windows\System32\drivers\etc\hosts file needs to be referred to check the docker ip.

# Added by Docker Desktop

172.20.0.53 host.docker.internal

172.20.0.53 gateway.docker.internal

# To allow the same kube context to work on the host and the container:

127.0.0.1 kubernetes.docker.internal

# End of section

We changed the ip of eureka serviceurl to distinguish that it is a part of the docker container. This could have been done for the RabbitMQ instance as well.

eureka.client.serviceUrl.defaultZone: http://host.docker.internal:8761/eureka

eureka.client.fetch-registry: false

eureka.client.healthcheck.enabled: true

eureka.client.register-with-eureka: true

eureka.instance.lease-expiration-duration-in-seconds: 2

eureka.instance.lease-renewal-interval-in-seconds: 1

message: Team checking bus refresh eventv2000

The Application, in our case CustomerAPI needs to contact Eureka server. It pings the eureka server after 1 second duration. If the ping does not reach eureka server for 2 seconds, the application is considered as down by eureka server.

So the next step is to configure the CustomerApi to communicate with Eureka Server.

eureka client needs to be added as a dependency and annotation of @EnableEurekaClient

in the CustomerApiApplication.

2023-07-12 14:27:24.922 INFO 9652 --- [nfoReplicator-0] c.c.c.ConfigServicePropertySourceLocator : Located environment: name=Customer-API,boa-common,boa-app, profiles=[dev], label=null, version=50c955fbb5cf20d5583910e53f51ef11a6c0b861, state=null

2023-07-12 14:27:24.957 INFO 9652 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_CUSTOMER-API,BOA-COMMON,BOA-APP/172.20.0.81:Customer-API,boa-common,boa-app:7074: registering service...

2023-07-12 14:27:25.515 INFO 9652 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_CUSTOMER-API,BOA-COMMON,BOA-APP/172.20.0.81:Customer-API,boa-common,boa-app:7074 - Re-registering apps/CUSTOMER-API,BOA-COMMON,BOA-APP

2023-07-12 14:27:25.515 INFO 9652 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_CUSTOMER-API,BOA-COMMON,BOA-APP/172.20.0.81:Customer-API,boa-common,boa-app:7074: registering service...

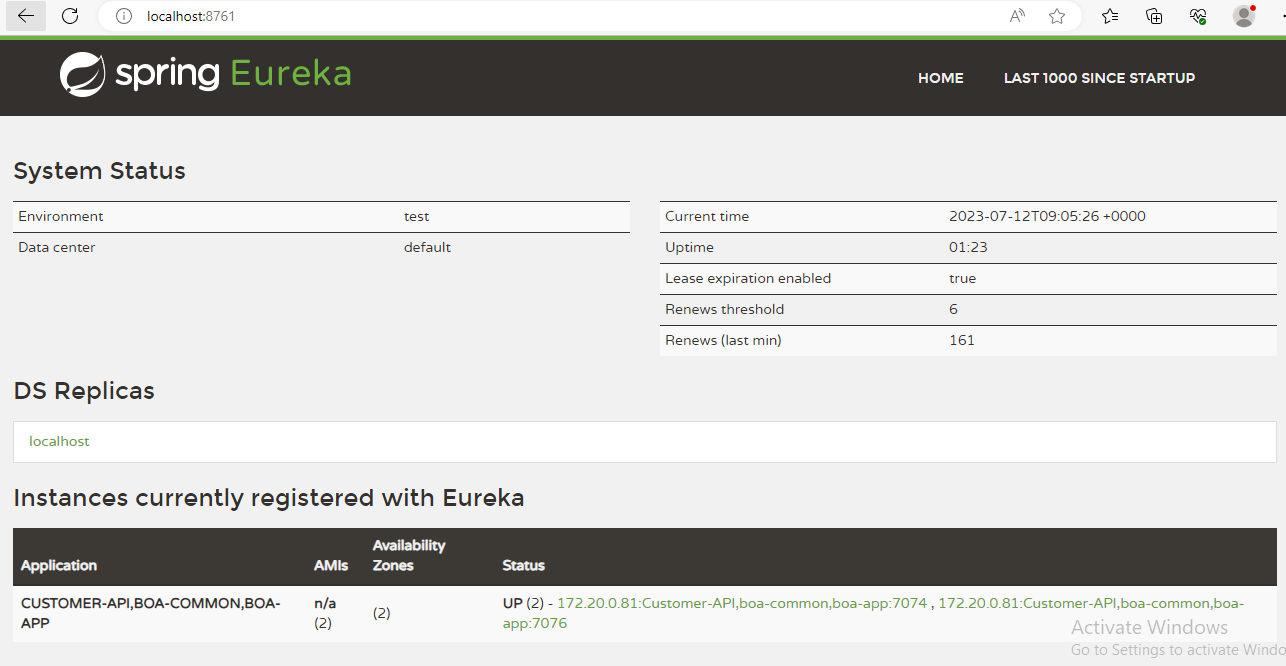
2023-07-12 14:27:25.623 INFO 9652 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_CUSTOMER-API,BOA-COMMON,BOA-APP/172.20.0.81:Customer-API,boa-common,boa-app:7074 - registration status: 204

2023-07-12 14:27:25.623 INFO 9652 --- [tbeatExecutor-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient\_CUSTOMER-API,BOA-COMMON,BOA-APP/172.20.0.81:Customer-API,boa-common,boa-app:7074 - registration status: 204

Now we will have another instance running for the same CustomerApiApplication. Just duplicate the Run Configuration for CustomerApplication and add another VM argument

-Dspring.profiles.active=dev -Dserver.port=7076.

Now check eureka server to see that 2 instances are showing up at ports 7074 and 7076.



If the instance at 7076 is stopped now, the eureka service will report only 1 instance after waiting for 2 secs and not getting anything from the instance from port 7076.

**Gateway**

Spring cloud has 2 gateways – Zuul and Spring gateway.

The application.properties f the applicationgateway project contains mapping as follows

#client

zuul.routes.customer.path=/api/customer/\*\*

#eureka application name

zuul.routes.customer.serviceId=CUSTOMER-API

It means that all the url patterns which follow a pattern of /api/customer/.. will be mapped to the CUSTOMER-API application registered with the Eureka service registry.

So if we open <http://localhost:8765/api/customer> it will be redirected to <http://172.20.0.81:7076/explorer/index.html#uri=/>

This means that the 7076 instance is contacted now.

**JWT Security**

We create a secret in vault called jwtsecret which will be used as secret for the signature portion of the jwt token that we would be using.

He bootstrap.properties

**public** **class** ApiSecurityConfig **extends** WebSecurityConfigurerAdapter

class has below methods implemented.

@Override

**public** **void** configure(WebSecurity web) **throws** Exception {

web.ignoring().antMatchers("/signin", "/signup");

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.csrf().disable().authorizeRequests().antMatchers("/signin", "/signup").permitAll().anyRequest()

.authenticated().and().exceptionHandling().authenticationEntryPoint(authenticationEntryPoint).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***);

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.**class**);

}

WebSecurity configure method handles generic web security while the endpoint level policies are applied using the HTTPSecurity configure method.

{

"userName":"eswaribala",

"password":"Test@123",

"roles":[{

"roleName":"ROLE\_ADMIN"

},

{

"roleName":"ROLE\_USER"

}]

}

-------------------------

{

"userName":"eswaribala",

"userPwd":"Test@123"

}

POST <http://localhost:7078/signup>

{

    "userName":"mani",

    "password":"Test@123",

    "roles":[{

     "roleName":"ROLE\_ADMIN"

    },

    {

        "roleName":"ROLE\_USER"

    }]

}

POST <http://localhost:7078/signin>

{

    "userName":"mani",

    "userPwd":"Test@123"

}

GET <http://localhost:7078/tests/v1.0/user/>

Authorization Header -

Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJtYW5pIiwicm9sZXMiOlsiUk9MRV9BRE1JTiIsIlJPTEVfVVNFUiJdLCJpYXQiOjE2ODkyMzI0NzYsImV4cCI6MTY4OTIzMjY1Nn0.bVjdQ\_P\_TAQiX27cqv5v1bc6qbNO9gaFRzm6ygZNSYsRKzPRZAtatXt1N7u8XzntCsqPa7FZiKRT1hbTz-9Q7A

docker pull mongo

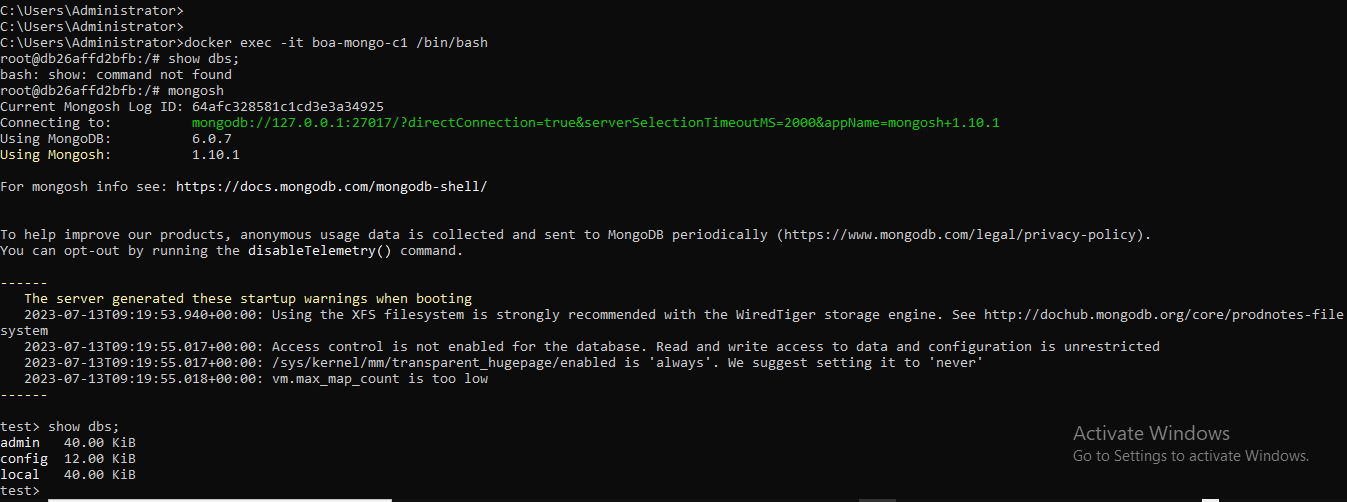
docker run --name boa-mongo-c1 -p 27017:27017 -d mongo

Once the mongo server is up and running, create the mongo client.

docker exec -it boa-mongo-c1 /bin/bash

This takes us to the container’s bash

root@db26affd2bfb:/#



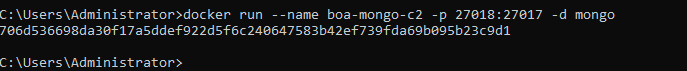
Get into the mongo bash using

>#mongosh

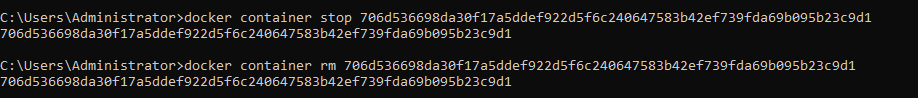
Now you can get the dbs using

>show dbs;

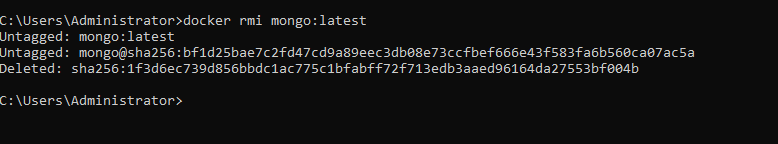
Create another docker container. By changing the container name and the port. Do not change the default port which is to the right. The left side port is the one on which the container runs.



We can stop the container using the container id.



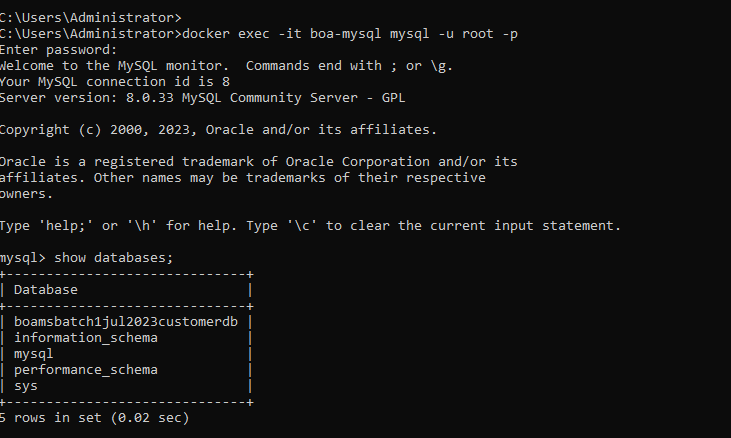
Once both the container instances are stopped.



**Create a my-sql container and check if the database is created by logging into mysql client console.**

docker run --name boa-mysql -e MYSQL\_ROOT\_PASSWORD=password -e MYSQL\_DATABASE=boamsbatch1jul2023customerdb -e MYSQL\_USER=demo\_user -e MYSQL\_PASSWORD=demo\_pass -d mysql

docker exec -it boa-mysql mysql -u root -p



**Password is password.**

**Build the customer api image –**

Go to the project folder and run the docker build

>> cd C:\Users\Administrator\Desktop\training workspace\Project workspace 7 - customerapidocker\boamsbatch1jul2023\customerapidocker\customerapidocker

>>docker build -f dockerfile -t customerapp .

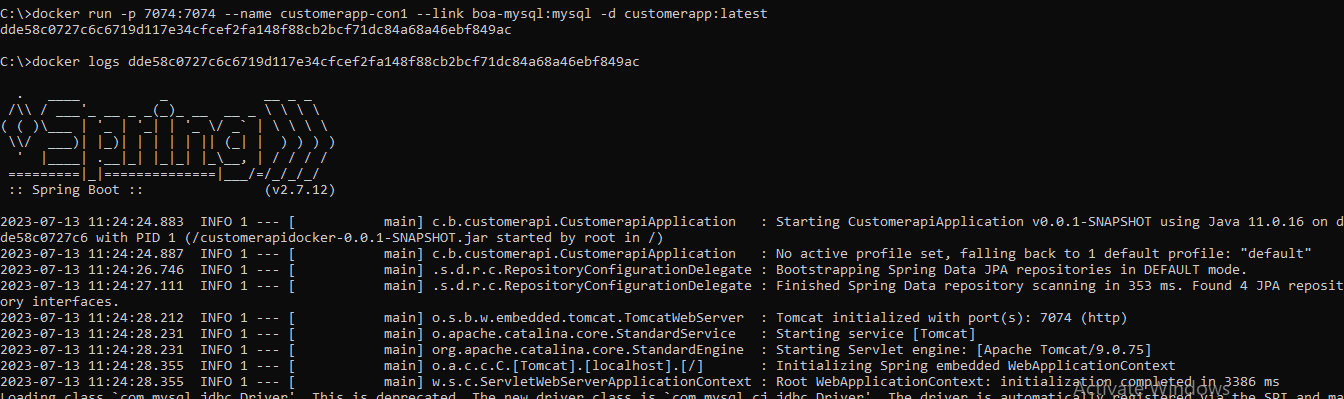
>> docker inspect customerapp

Now se this image to spawn a container and link that container to the mysql container

>>docker run -p 7074:7074 --name customerapp-c1 --link boa-mysql:mysql -d customerapp:latest

Check the logs with

>> docker logs <container-id>

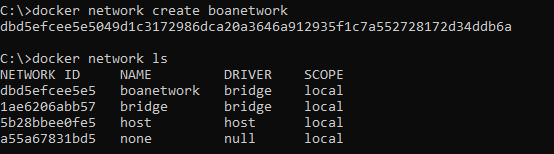


This is very tough as there a lot of steps involved.

SO we try to automate using docker compose.

Docker 24.0 is mapped to docker compose file.

Create the docker network manually



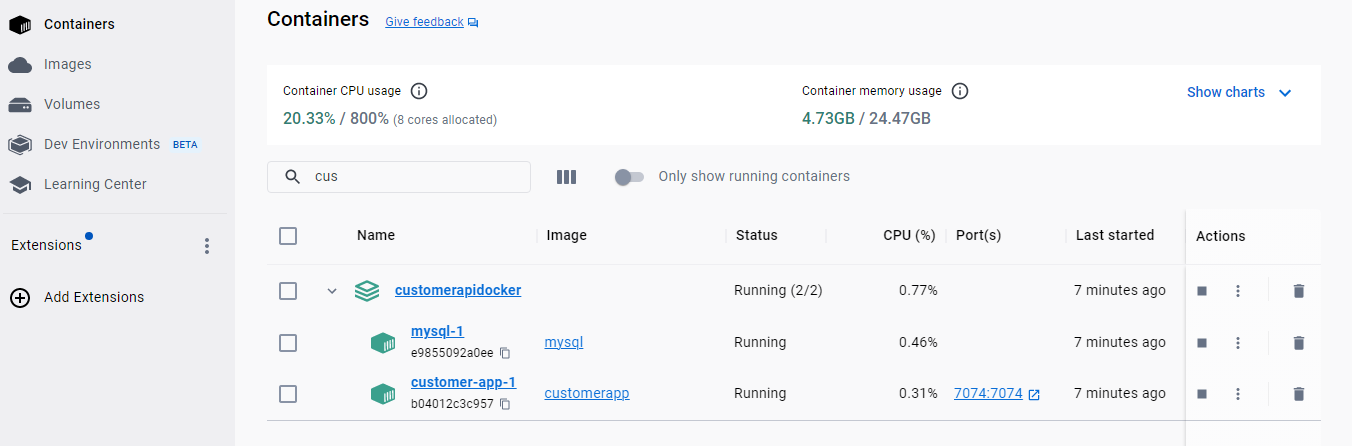
Then run the docker compose file

>> docker-compose up

Now the spring boot application will fail multiple times till the time the mysql service is up.

But it will retry and get up eventually.

The same can be seen using docker desktop.



To clean all the containers

>>docker-compose down

Tag the image and push to docker hub.

C:\>docker tag customer:latest manikandanrajaraman/boabatch1mstrainingjul2023repo:customerappv1

C:\>docker push manikandanrajaraman/boabatch1mstrainingjul2023repo:customerappv1

The push refers to repository [docker.io/manikandanrajaraman/boabatch1mstrainingjul2023repo]

274f21de93c3: Pushed

7b7f3078e1db: Mounted from library/openjdk

826c3ddbb29c: Mounted from library/openjdk

b626401ef603: Mounted from library/openjdk

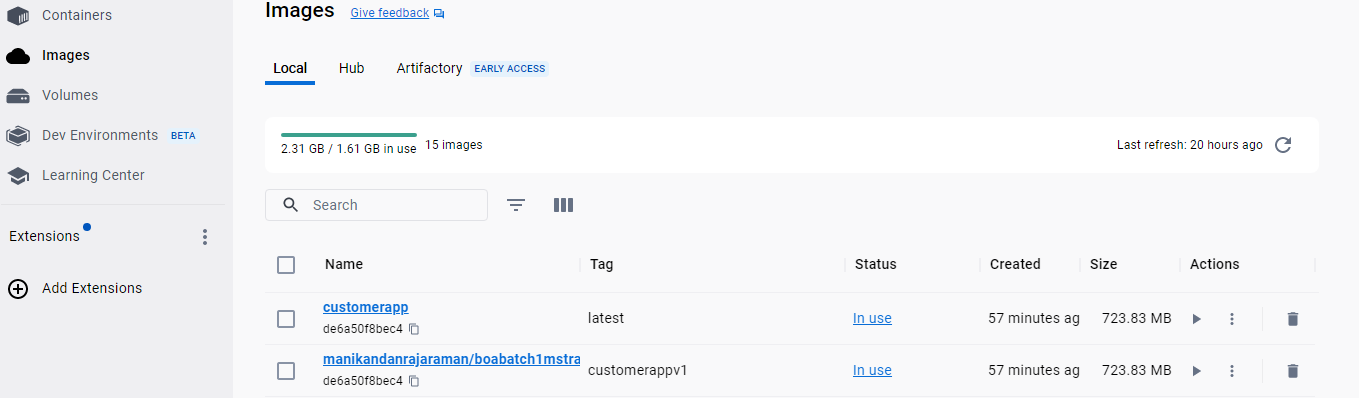
9b55156abf26: Mounted from library/openjdk

293d5db30c9f: Mounted from library/openjdk

03127cdb479b: Mounted from library/openjdk

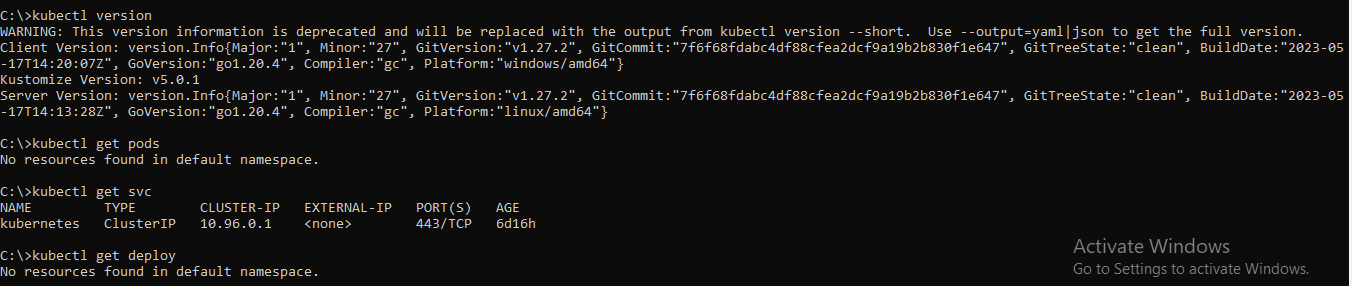
9c742cd6c7a5: Mounted from library/openjdk

customerappv1: digest: sha256:0a333a188db53810950826ade4a6851d82a1f3282d22a2c5d96277f4ea9e6304 size: 2007



**Kubernetes**

Some basic kubernetes commands



server.port=7075

spring.application.name=Customer-API

#data source

spring.datasource.url=jdbc:mysql://localhost:3306/boamsbatch1jul2023customerdb?useSSL=true

spring.datasource.username=root

spring.datasource.password=rps@12345

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

## JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database=default

spring.jpa.show-sql=true

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

#spring.datasource.continue-on-error=false

spring.datasource.generate-unique-name=false

spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImpl

spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

spring.jackson.serialization.fail-on-empty-beans=false

#graphql

graphql.tools.schema-location-pattern: \*\*/\*.graphqls