Workshop #1 - Daniel Ramirez Gomez

miércoles, 7 de febrero de 2024 1:07 a. m.

Git clone was done from the repository

```
root@ubuntu:/home/danigomez/Documents/Workshop-1# git clone https://github.com/
icesi-ops/training_microservices.git
```

We will proceed first with the app-config server.

First, access its Dockerfile to change its listening port to 8888, which is the common port for a configuration server in Spring Boot.
Also, verify that in its application.properties, it is also listening on that port.

```
# Server
spring.application.name=app-config
server.port=8888

# Config
spring.cloud.config.server.git.uri=https://github.com/icesi-ops/training_micros
ervices.git
spring.cloud.config.server.default-label=master
spring.cloud.config.server.git.search-paths=pay-app-spring-microservices/config
spring.cloud.config.server.git.skip-ssl-validation=true

# Consul
spring.cloud.consul.host=consul
spring.cloud.consul.port=8500
spring.cloud.consul.discovery.health-check-interval=5s
spring.cloud.consul.discovery.prefer-ip-address=trueroot@ubuntu:/home/danigomez
/Documents/Workshop-1/training_microservices/pay-app-spring-microservices/app-c
```

Now the network creation
docker network create microservicenetwork

```
danigomez@ubuntu:~/Documents/pr1/sd-workshop1$ docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
4437b54fd4e7   bridge                 bridge    local
ae2c872c25ff   distribuidos           bridge    local
38e3620d1c21   host                   host      local
d50b1af80e23   microservicenetwork    bridge    local
a9da5aa925db   none                   null      local
```

In the build.gradle, it is noticed that a service called consul is used. Therefore, a container must be created to keep the consul server running.
Starting Consul:
docker run -d -p 8500:8500 -p 8600:8600/udp --network microservicenetwork --name consul consul:1.15 agent -server -bootstrap-expect 1 -ui -data-dir /tmp -client=0.0.0.0

Now, build the image of the config server service.
Specify the platform that the image will use and also provide a tag. Use the Dockerfile from the current directory with ..
docker build -t ventana1901/app-config --platform=linux/amd64 .

Now, build his container
docker run -dit -p 8888:8888 --network microservicenetwork --name app-config ventana1901/app-config
Verify that the service is up.
curl --location --request GET 'http://localhost:8888/app-invoice/dev'

```
{"name":"app-invoice","profiles":["dev"],"label":null,"version":"76ca227bc376a79cb1321ffcea2dab90a0f00bd9","state":null,"propertySources":[{"name":"https://github.com/ices
i-ops/training_microservices.git/pay-app-spring-microservices/config/app-invoice-dev.properties","source":{"spring.application.name":"app-invoice","server.port":"8006","sp
ring.kafka.consumer.bootstrap-servers":"servicekafka:9092","spring.kafka.admin.properties.bootstrap.servers":"servicekafka:9092","spring.kafka.consumer.key-deserializer":"
org.apache.kafka.common.serialization.IntegerDeserializer","spring.kafka.consumer.value-deserializer":"org.apache.kafka.common.serialization.StringDeserializer","spring.ka
fka.consumer.group-id":"invoice-events-listener-group","logging.level.org.hibernate.SQL":"debug","spring.jpa.properties.hibernate.enable_lazy_load_no_trans":"true","spring
.jpa.hibernate.ddl-auto":"create","spring.datasource.url":"jdbc:postgresql://postgres:5432/db_invoice","spring.datasource.username":"postgres","spring.datasource.password"
:"postgres","spring.datasource.driver-class-name":"org.postgresql.Driver","spring.jpa.database-platform":"org.hibernate.dialect.PostgreSQL95Dialect","spring.cloud.consul.h
ost":"consul","spring.cloud.consul.port":"8500","spring.cloud.consul.discovery.health-check-interval":"5s","spring.cloud.consul.discovery.prefer-ip-address":"true"}}]}root
@ubuntu:/home/danigomez/Documents/Workshop-1/training_microservices/pay-app-spring-microservices/app-config#
```

Now, move to the app-invoice folder to start the microservice.
Check that in the app-invoice's curl, the port on which the service listens is 8006. Therefore, it needs to be modified in the Dockerfile in the EXPOSE section.

Then two dependencies must be added to the app-invoice's build.gradle.
implementation 'org.springframework.cloud:spring-cloud-starter-consul-discovery'
implementation 'org.springframework.boot:spring-boot-starter-actuator'

```
danigomez@ubuntu:~/Documents/Workshop-1/training_microservices/pay-app-spring-microservices/app-invoice$ cat build.gradle
plugins {
    id 'org.springframework.boot' version '2.3.10.RELEASE'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
}


group = 'com.aforo'
version = '1.0.0'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

ext {
    set('springCloudVersion', "Hoxton.SR11")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.kafka:spring-kafka'
    implementation 'org.springframework.cloud:spring-cloud-starter-config'
    implementation 'org.springframework.cloud:spring-cloud-starter-consul-discovery'
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.kafka:spring-kafka-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
    }
}
```

Verify in the config folder that in the app-invoice properties, consul is configured as variables

```
# Consul
spring.cloud.consul.host=consul
spring.cloud.consul.port=8500
spring.cloud.consul.discovery.health-check-interval=5s
spring.cloud.consul.discovery.prefer-ip-address=true
```

Build the image located in the folder containing the Dockerfile app-invoice
docker build -t ventana1901/app-invoice --platform=linux/amd64 .

Before starting its container, check that the service depends on its database and Kafka, so these microservices that provide operability to app-invoice must be started first.

To start Postgres, go to the folder containing its Dockerfile and execute the build.
docker build -t ventana1901/postgres --platform=linux/amd64 .

Now run the container
docker run -p 5434:5432 --name postgres --network distribuidos -e POSTGRES_PASSWORD=postgres -e POSTGRES_DB=db_invoice -d postgres:12-alpine

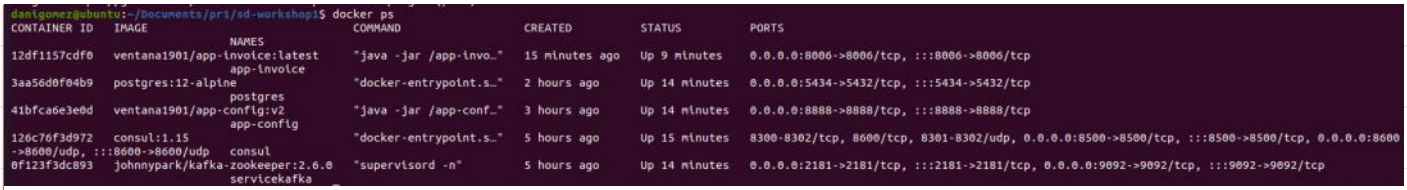
Now execute the Kafka service
docker run -p 2181:2181 -d -p 9092:9092 --name servicekafka --network microservice -e ADVERTISED_HOST=servicekafka -e NUM_PARTITIONS=3 johnnypark/kafka-zookeeper:2.6.0

Now app-invoice can be executed
docker run -dit -p 8006:8006 --name app-invoice --network microservicenetwork ventana1901/app-invoice

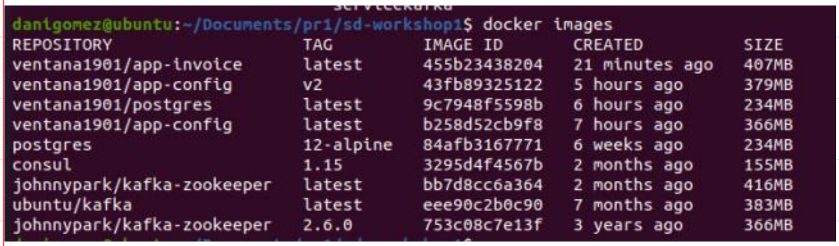
Once all the containers are ready, they should be executed in the correct order. First, stop them
Docker stop $(docker ps -aq)

Now start them in the following order with the docker start command (Container ID):
1)Consul
2)Kafka
3)Postgres
4)app-config
5)app-invoice

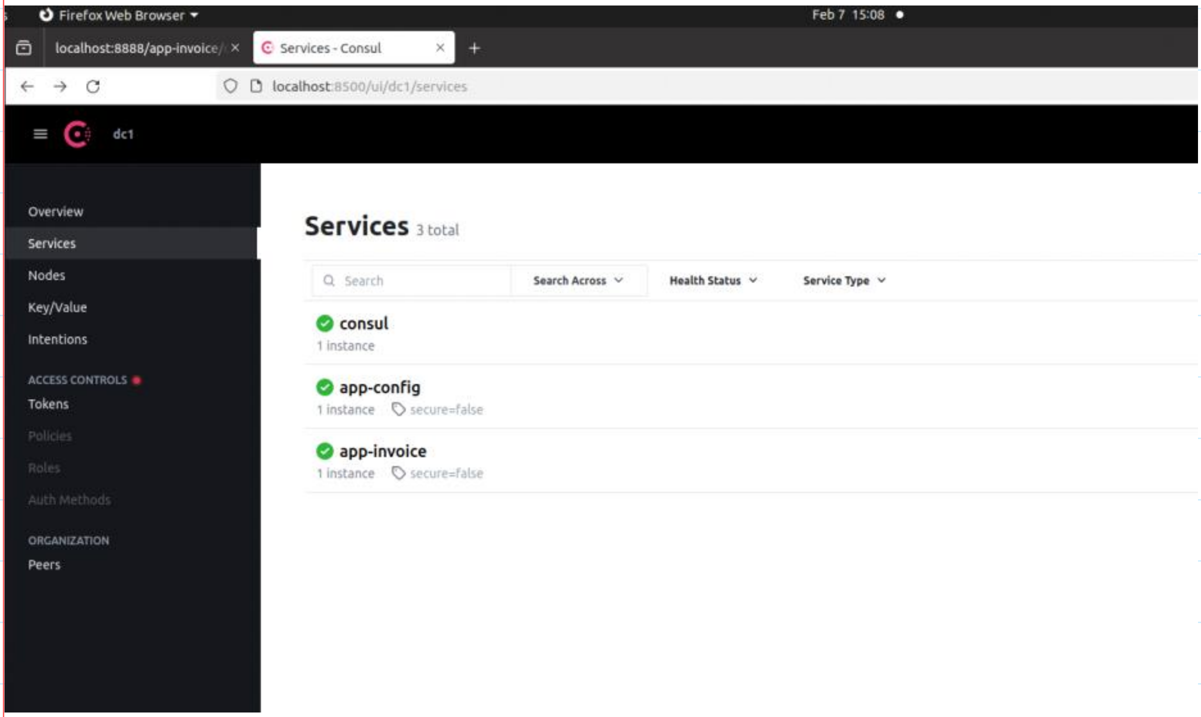
And i get the following



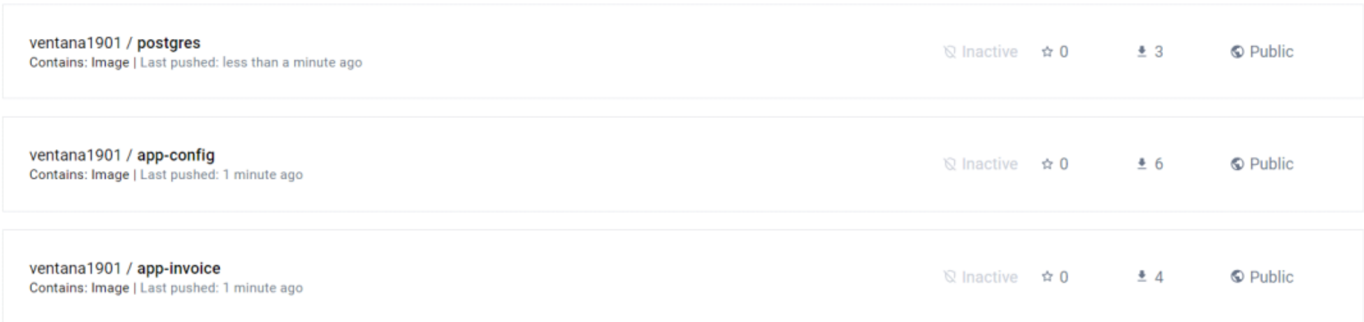
Here, you can also see all the created images



Now, you can verify that various microservices are in Consul



Finally, push the created images.



To now add data with PostgreSQL

First, you must execute this command in the container folder for the data, data.sql

PGPASSWORD=postgres psql -h localhost -d db_invoice -U postgres -f data.sql

Then, the data are added

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

Now, if we go to the URL http://localhost:8006/all, we can see the following.