LAB XX - AWS and Microservices

One of the capabilities mentioned in the microservices capability model is the use of the cloud infrastructure for microservices. Earlier in this chapter, we also explored the necessity of using the cloud for microservices deployments. So far, we have not deployed anything to the cloud. As we have eight microservices in total—Config-server, Eureka-server, Turbine, RabbitMQ, Elasticsearch, Kibana, and Logstash—in our overall BrownField PSS microservices ecosystem, it is hard to run all of them on the local machine.

In the rest of this book, we will operate using AWS as the cloud platform to deploy BrownField PSS microservices.

### Installing Docker on AWS EC2

In this section, we will install Docker on the EC2 instance.

This example assumes that readers are familiar with AWS and an account is already created on AWS.

Perform the following steps to set up Docker on EC2:

* Launch a new EC2 instance. In this case, if we have to run all the instances together, we may need a large instance. The example uses **t2.large**.
* In this example, the following Ubuntu AMI image is used: ubuntu-trusty-14.04-amd64-server-20160114.5 (ami-fce3c696).
* Connect to the EC2 instance and run the following commands:
* **sudo apt-get update**   
  **sudo apt-get install docker.io**
* The preceding command will install Docker on an EC2 instance. Verify the installation with the following command:
* **docker version**

## Running BrownField services on EC2

In this section, we will set up BrownField microservices on the EC2 instances created. In this case, the build is set up in the local desktop machine, and the binaries will be deployed to AWS.

Perform the following steps to set up services on an EC2 instance:

* Install Git via the following command:
* **sudo apt-get install git**
* Create a Git repository on any folder of your choice.
* Change the Config server's bootstrap.properties to point to the appropriate Git repository created for this example.
* Change the bootstrap.properties of all the microservices to point to the config-server using the private IP address of the EC2 instance.
* all \*.properties from the local Git repository to the EC2 Git repository and perform a commit.
* Change the Eureka server URLs and RabbitMQ URLs in the \*.properties file to match the EC2 private IP address. Commit the changes to Git once they have been completed.
* On the local machine, recompile all the projects and create Docker images for the search, search-apigateway, and website microservices. Push all of them to the Docker Hub registry.
* the config-server and the Eureka-server binaries from the local machine to the EC2 instance.
* Set up Java 8 on the EC2 instance.
* Then, execute the following commands in sequence:
* **java –jar config-server.jar**   
  **java –jar eureka-server.jar**   
  **docker run –net host rabbitmq:3**  
  **docker run --net host -p 8090:8090 rajeshrv/search:1.0**  
  **docker run --net host -p 8095:8095 rajeshrv/search-apigateway:1.0**  
  **docker run --net host -p 8001:8001 rajeshrv/website:1.0**
* Check whether all the services are working by opening the URL of the website and executing a search. Note that we will use the public IP address in this case: http://54.165.128.23:8001.