Taller de Modularización con Virtualización e Introducción a Docker

Requisitos de Finalización

Descripción

En este taller se profundiza en los conceptos de **modularización mediante virtualización**, usando **Docker** y **AWS**.

Los estudiantes deberán:

- Crear una aplicación web pequeña usando **Spring Boot**.
- Construir una **imagen Docker** para dicha aplicación.
- Ejecutar y configurar el contenedor en su máquina local.
- Publicar la imagen en **DockerHub**.
- Crear una **máquina virtual en AWS EC2**, instalar Docker y desplegar allí la imagen publicada.

Pre-requisitos

- Conocer **Java** y **Maven**.
- Saber desarrollar aplicaciones web en Java.
- Tener **Docker instalado** localmente.

Primera Parte: Crear la Aplicación Web

1. Crear un proyecto Java con Maven

2. Crear una aplicación Spring mínima

Controlador

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloRestController {

    private static final String template = "Hello, %s!";

    @GetMapping("/greeting")
    public String greeting(@RequestParam(value = "name", defaultValue = "World")
String name) {
        return String.format(template, name);
```

```
    }
}
```

Clase principal para iniciar Spring

```
@SpringBootApplication
public class RestServiceApplication {

    public static void main(String[] args) {
        SpringApplication app = new
SpringApplication(RestServiceApplication.class);
        app.setDefaultProperties(Collections.singletonMap("server.port",
getPort()));
        app.run(args);
    }

    private static int getPort() {
        if (System.getenv("PORT") != null) {
            return Integer.parseInt(System.getenv("PORT"));
        }
        return 5000;
    }
}
```

---

3. Agregar dependencias de Spring en el `pom.xml`

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>3.3.3</version>
    </dependency>
</dependencies>
```

---

4. Asegurar compatibilidad con Java 17+

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>
```

---

5. Copiar dependencias al directorio `target` con el plugin `maven-dependency-plugin`

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.0.1</version>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

---

6. Compilar el proyecto

```
mvn clean install
```

---

7. Ejecutar la aplicación desde consola

```
java -cp "target/classes:target/dependency/*" \
     co.edu.escuelaing.sparkdockerdemolive.RestServiceApplication
```

Acceder en el navegador a:

```
http://localhost:4567/hello
```

Segunda Parte: Crear la Imagen Docker y Subirla

1. Crear un `Dockerfile` en la raíz del proyecto

```

```

```
FROM openjdk:8

WORKDIR /usrapp/bin

ENV PORT 6000

COPY /target/classes /usrapp/bin/classes
COPY /target/dependency /usrapp/bin/dependency

CMD ["java","-
cp","./classes:./dependency/*","co.edu.escuelaing.sparkdockerdemolive.SparkWebServ
er"]
```

---

2. Construir la imagen

```
docker build --tag dockersparkprimer .
```

---

3. Verificar imágenes

```
docker images
```

---

4. Crear tres contenedores

```
docker run -d -p 34000:6000 --name firstdockercontainer dockersparkprimer
docker run -d -p 34001:6000 --name firstdockercontainer2 dockersparkprimer
docker run -d -p 34002:6000 --name firstdockercontainer3 dockersparkprimer
```

---

5. Verificar contenedores

```
docker ps
```

---

6. Acceder a los servicios

```
http://localhost:34000/hello
http://localhost:34001/hello
http://localhost:34002/hello
```

7. Crear archivo `docker-compose.yml`

```
version: '2'

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: web
    ports:
      - "8087:6000"

  db:
    image: mongo:3.6.1
    container_name: db
    volumes:
      - mongodb:/data/db
      - mongodb_config:/data/configdb
    ports:
      - 27017:27017
    command: mongod

volumes:
  mongodb:
  mongodb_config:
```

8. Ejecutar docker-compose

```
docker-compose up -d
```

9. Verificar contenedores

```
docker ps
```

Tercera Parte: Subir la Imagen a DockerHub

1. Crear cuenta en DockerHub y repositorio

2. Crear un tag apuntando a tu repositorio

```
docker tag dockersparkprimer usuario/repo
```

Ejemplo:

```
docker tag dockersparkprimer dnielben/firstsprkwebapprepo
```

---

3. Verificar que existe el tag

```
docker images
```

---

4. Autenticarse

```
docker login
```

---

5. Subir la imagen

```
docker push usuario/repo:latest
```

Cuarta Parte: AWS

1. Acceder a la VM en EC2

2. Instalar Docker

```
sudo yum update -y
sudo yum install docker
```

3. Iniciar Docker

```
sudo service docker start
```

4. Evitar usar sudo en cada comando

```
sudo usermod -a -G docker ec2-user
```

(Cerrar sesión y volver a entrar)

---

5. Correr el contenedor desde la imagen de DockerHub

```
docker run -d -p 42000:6000 --name firstdockerimageaws usuario/repo
```

---

6. Abrir el puerto en el security group de EC2

---

7. Acceder al servicio

Ejemplo:

```
http://ec2-35-175-205-168.compute-1.amazonaws.com:42000/hello
```

TAREA

Para la tarea el estudiante debe:

- Construir una aplicación web **usando SU PROPIO FRAMEWORK** (no Spring).
- Hacer su framework **concurrente**.
- Implementar apagado **elegante** del servidor.
- Crear una imagen Docker.
- Desplegar la aplicación en **AWS EC2** usando Docker.

Entregables

1. Código del proyecto en un **repositorio GitHub**.
2. **README profesional** que incluya:
    - Resumen del proyecto.
    - Arquitectura.
    - Diseño de clases.
    - Cómo generar las imágenes Docker.
    - Evidencias del despliegue (capturas).
3. **Video** mostrando los despliegues funcionando en AWS.