Interprocess Communication Patterns II - Example

Docker with ActiveMQ

Owerya RESOURCING

How to deploy an Async Messaging Service?

- In real life your service will run in containers.

- We will now look at how to package an ActiveMQ project using Docker and docker-compose.

Sender App

- Our sender app just sends three words to the ActiveMQ channel

MqSenderSpringBootAppApplication.java

```java
@SpringBootApplication
public class MqSenderSpringBootAppApplication {

  public static void main(String[] args) {
    ApplicationContext context = SpringApplication.run(MqSenderSpringBootAppApplication.class, args);

    MySender sender = context.getBean(MySender.class);

    sender.sendMessage("Huey");
    sender.sendMessage("Louis");
    sender.sendMessage("Dewey");
  }
}
```

MySender.java

```java
@Component
public class MySender {

  @Autowired
  JmsTemplate template;

  public void sendMessage(String message) {
    template.convertAndSend("myqueue", message);
  }
}
```

Receiver App

- Our receiver app prints those to stdout

MqReceiverSpringBootAppApplication.java

```
@SpringBootApplication
public class MqReceiverSpringBootAppApplication {

    public static void main(String[] args) {
        SpringApplication.run(MqReceiverSpringBootAppApplication.class, args);
    }

}
```

MyReceiver.java

```
@Component
public class MyReceiver {

    @JmsListener(destination = "myqueue")
    public void receiveMessage(String message) {
        System.out.println("Received message: " + message);
    }
}
```

Project Dependencies

- Both projects are configured to recognized the MQ service:

```
spring.activemq.broker-url=tcp://activemq:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

- And have identical project dependencies on activemq

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-activemq</artifactId>
   </dependency>
```

- Apart from this there is no special configuration.

Building the apps and the Dockerfile

- The java apps are built using Maven and the two target jar files are put into the target directory of each project.
- A Dockerfile to containerize the application (just change the jarfile name for the receiver app)

```
FROM openjdk:11.0
ADD target/MqSenderSpringBootApp-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Dockerfile-send-app

ActiveMQ Dockerfile

- Now get ActiveMQ running from a Dockerfile.

```
FROM rmohr/activemq
EXPOSE 61616
EXPOSE 8161
RUN find /opt/activemq/ -type d -exec chmod 777
{} \;
USER 999
```

Dockerfile-activemq

Docker-compose

```yaml
version: '3.8'
services:

  activemq:
    container_name: activemq
    build:
      context: .
      dockerfile: Dockerfile-activemq
    image: emps/activemq:1.0.0
    ports:
      - "61616:61616"
      - "8161:8161"
    volumes:
      - /docker/emps/activemq/data:/data/activemq
      - /docker/emps/activemq/log:/var/log/activemq
    restart: always

  appsend:
    container_name: appsendmq
    build:
      context: ./MqSenderSpringBootApp
      dockerfile: Dockerfile-send-app
    image: emps/appsendmq:1.0.0
    links:
      - activemq:activemq

  apprecv:
    container_name: apprecvmq
    build:
      context: ./MqReceiverSpringBootApp
      dockerfile: Dockerfile-recv-app
    image: emps/apprecvmq:1.0.0
    links:
      - activemq:activemq
```

docker-compose.yaml

To start it all:

```
$docker-compose up
```

- This will
  - Download and configure an ActiveMQ server listening on the right ports
  - Package the sender and receive jar files into docker containers and serve them
  - Configure the necessary network links between the components.

```
apprecvmq   | Received message: Huey
apprecvmq   | Received message: Louis
apprecvmq   | Received message: Dewey
```

- To pack it all away, CTRL-C followed by

```
$docker-compose down
```

Questions or Comments?

Owerya RESOURCING