PRACTICAL 2

**Develop Spring Boot API**

**Introduction :-**

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

In this example, we showcase that it is possible to host a non-Microsoft stack application to a Service Fabric cluster. We will not create a reliable service; instead, we will host a Java Spring Boot–based API as a guest executable and as a container. We will use VS Code to develop a simple Spring Boot application.

**Setting up the Development Environment**

Let’s set up the development environment.

1. Install Visual Studio 2017.

2. Install the Microsoft Azure Service Fabric SDK.

3. Install Visual Studio Code.

a. Install Spring Boot Extensions Pack.

b. Install Java Extensions Pack.

c. Install Maven for Java.

4. Make sure that the Service Fabric Local cluster is in a running state.

5. Install Docker Desktop.

6. Access the Azure container registry.

**Develop a Spring Boot API**

Now it’s time to get started on the application.

1. Launch Visual Studio Code as an administrator.

2. Press Ctrl+Shift+P to open the command palette.

3. Enter spring in the command palette, and choose Spring Initializr: Generate a maven A screenshot of a computer program

Description automatically generatedproject

4. Choose Java for Specify Project Language.

5. Enter com.microservices in the input group ID for your project.

6. Enter employeespringservice in the input artifact ID for your project.

7. Choose the latest Spring boot version.

8. Choose the following dependencies.

a. DevTools

b. Lombok

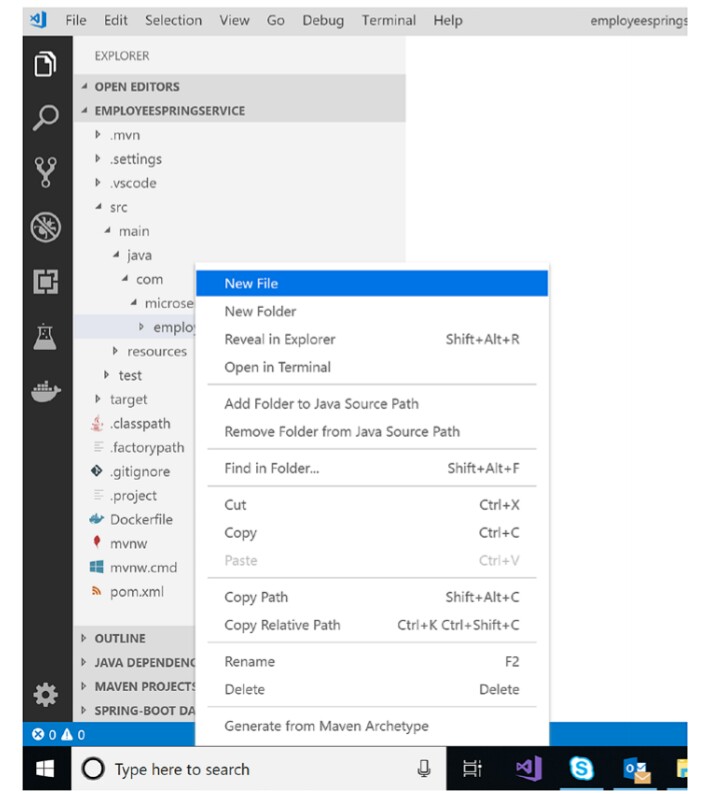
c. Web

d. Actuator

9. Choose the path where you want to save the solution.

10. Right-click the EMPLOYEESPRINGSERVICE folder under src ➤ main ➤ java ➤ com ➤ microservices, as shown in Figure 3-15, and click Add File.

Add a new file

11. Name the file Employee.Java and add the following code. (This is the definition of the employee object; we kept it simple by having only three properties to represent an employee.)

package com.microservices.employeespringservice;

import lombok.AllArgsConstructor;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

/\*\*\* Employee \*\*\*/

@Getter

@Setter

@EqualsAndHashCode

@AllArgsConstructor

public class Employee {

private String firstName;

private String lastName;

private String ipAddress; }

12. Now let’s create an employee service that returns an employee’s information. Right-click the employeespringservice folder and add a file named EmployeeService.java. Add the following code to it.

package com.microservices.employeespringservice;

import java.net.InetAddress;

import java.net.UnknownHostException;

import org.springframework.stereotype.Service;

/\*\*

\* EmployeeService

\*/

@Service

public class EmployeeService {

public Employee GetEmployee(String firstName, String

lastName){

String ipAddress;

try {

getHostAddress().toString();

} catch (UnknownHostException e) {

ipAddress = e.getMessage();

}

Employee employee = new Employee(firstName,

lastName,ipAddress);

return employee;

}

}

13. Now let’s create an employee controller that invokes the employee service to return the details of an employee. Right-click the employeespringservice folder and add a file named EmployeeController.

package com.microservices.employeespringservice;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ResponseBody;

/\*\*

\* EmployeeController

\*/

@Controller

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping("/")

@ResponseBody

public Employee getEmployee(){

return employeeService.GetEmployee("Spring","Boot");

}

}

Now you are ready for a simple REST-based service that returns employee information. Visual Studio Code has some cool features to run Spring Boot applications, and we are going to use the same.

1. Open DemoApplication.java and once you open the file, you see the option to run or debug the application,Please note that this may take time as Visual Studio automatically downloads the dependencies.

A screenshot of a computer program

Description automatically generated

2. Click Run and open the Controller class. You see the URL where your controller service is hosted.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

3. Click the URL and you see the output show in your default browser.

4. Right-click employeespringservice under Maven Projects. Click package,This generates the JAR file.

A screenshot of a computer

Description automatically generated

Now you have a simple Spring Boot-based REST API, and you have generated a JAR file. We will now deploy it to Service Fabric as a guest executable. To deploy, we will use Visual Studio 2017.

**Deploy a Spring Boot Service as a Guest Executable**

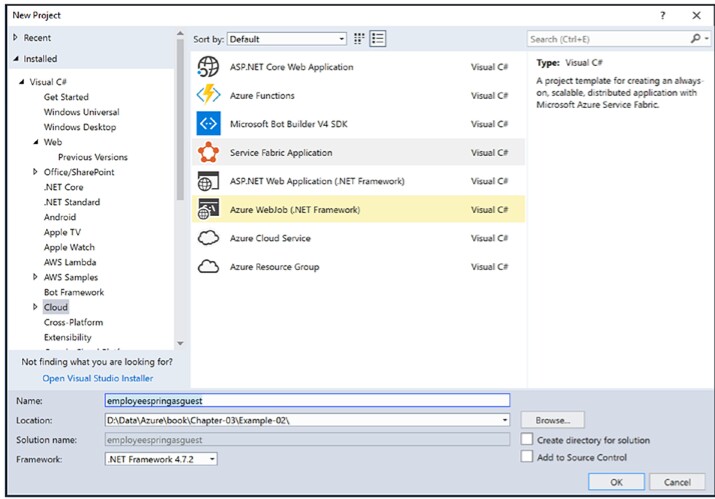
After executing all the steps in the previous section, your development is complete. Please follow

the steps in this section to deploy the developed Spring Boot application as a guest executable. This shows that it is possible to host a non-Microsoft stack application on a Service Fabric cluster by using a guest executable programming model. Service Fabric considers guest executables a stateless service.

1. Launch Visual Studio 2017 as an administrator.

1. Click File ➤ New Project ➤ Select Cloud ➤ Service Fabric Application.

2. Name the application employeespringasguest



3. In New Service Fabric Service, select the following

a. Service Name: employeeguestservice

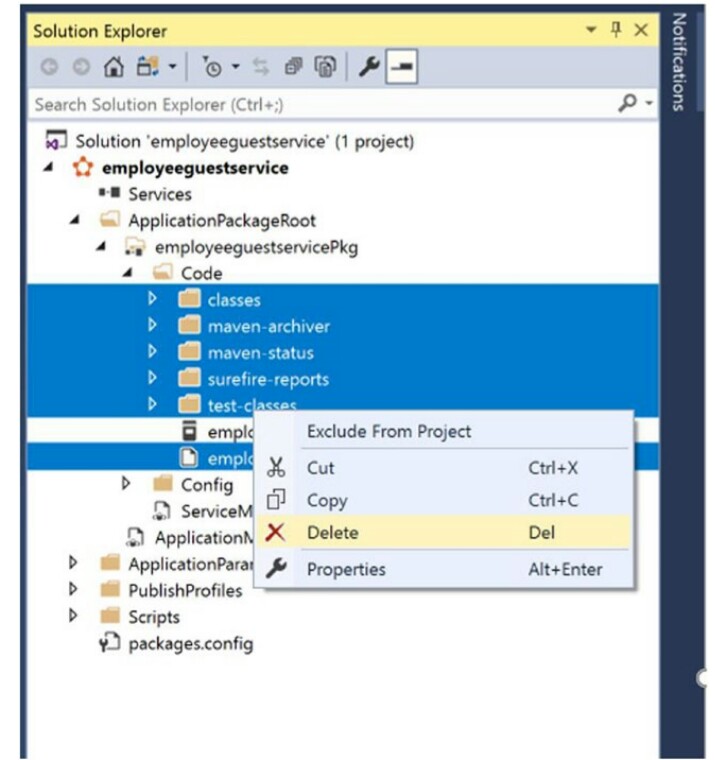
b. Code Package Folder: Point to the target folder in which Visual Studio Code generated the JAR file for the Spring Boot service.

c. Code Package Behavior: Copy folder contents to folder

d. WorA screenshot of a service

Description automatically generatedking Folder: CodeBase

4. Delete the selected files from the Code folder.



5. We also need to upload the runtime to run the JAR. Generally, it resides in the JDK installation folder (C:\ java-1.8.0-openjdk-1.8.0.191-1.b12.redhat.windows.x86\_64). Paste it in the Code folder.

A screenshot of a computer

Description automatically generated

6. Open ServiceManisfest.xml and set the following values.

<EntryPoint>

<ExeHost>

<Program>jre\bin\java.exe</Program>

<Arguments>-jar ..\..\employeespringservice-0.0.1-

SNAPSHOT.jar</Arguments>

<WorkingFolder>CodeBase</WorkingFolder>

<!-- Uncomment to log console output (both stdout and stderr) to one of the

service's working directories. -->

<!-- <ConsoleRedirection FileRetentionCount="5"

FileMaxSizeInKb="2048"/> -->

</ExeHost>

</EntryPoint>

</CodePackage>

<Resources>

<Endpoints>

<!-- This endpoint is used by the communication listener to obtain the port on which to

listen. Please note that if your service is partitioned, this port is shared withreplicas of different partitions that are placed in

your code. -->

<Endpoint Name="employeeguestserviceTypeEndpoint"

Protocol="http" Port="8080" Type="Input" />

</Endpoints>

</Resources>

7. Make sure that the local Service Fabric cluster is up and running. Click F5. Browse the Service Fabric dashboard. The default URL is http://localhost:19080/Explorer/index.html. You see that your service is deployed.

A screenshot of a computer

Description automatically generated

8. Browse http://localhost:8080 to access your service. In servicemanifest.xml, we specified the seA screenshot of a computer

Description automatically generatedrvice port as 8080; you can browse the same on 8080.

**Deploy a Spring Boot Service as a Container**

So far, we have deployed the service as a guest executable in Service Fabric. Now we will follow the steps to deploy the Spring service as a container in Service Fabric. This explains that in addition to creating stateful and stateless services, Service Fabric also orchestrates containers like any other orchestrator, even if the application wasn’t developed on a Microsoft stack.

1. Open Visual Studio Code. Open the folder where employeespringservice exists. Open the Docker file.

2. Make sure that the name of the JAR file is correct.

3. Select Switch to Windows container… in Docker Desktop.

A screenshot of a computer

Description automatically generated

4. Create the Azure Container Registry resource in the Azure portal. Enable the admin user.

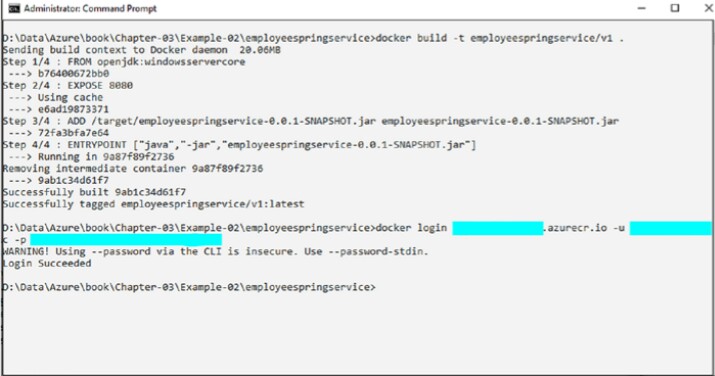
A screenshot of a computer

Description automatically generated

5. Open the command prompt in Administrative Mode and browse to the directory where the Docker file exists.

6. Fire the following command, including the period at the end. (This may take time because it downloads the Window Server core image from the Docker hub)

docker build -t employeespringservice/v1 .



A screenshot of a computer

Description automatically generatedNow the container image is available locally. You have to push the image to Azure Container

Registry.

1. Log in to Azure Container Registry using the admin username and password. Use the following command .

docker login youracr.azurecr.io -u yourusername -p yourpassword

A screenshot of a computer

Description automatically generated2. Fire the following commands to upload the image to ACR.

docker tag employeespringservice/v1 youracr.azurecr.io/book/

employeespringservice/v1

docker push myservicefabric.azurecr.io/book/

employeespringservice/v1

3. Log in to the Azure portal and check if you can see your image in Repositories.Since the container image is ready and uploaded in Azure Container Registry, let’s create a Service Fabric project to deploy the container to the local Service Fabric cluster.

1. Launch Visual Studio 2017 as an administrator.

2. Click File ➤ New Project ➤ Select Cloud ➤ Service Fabric Application.

3. Name the application employeespringascontainer.

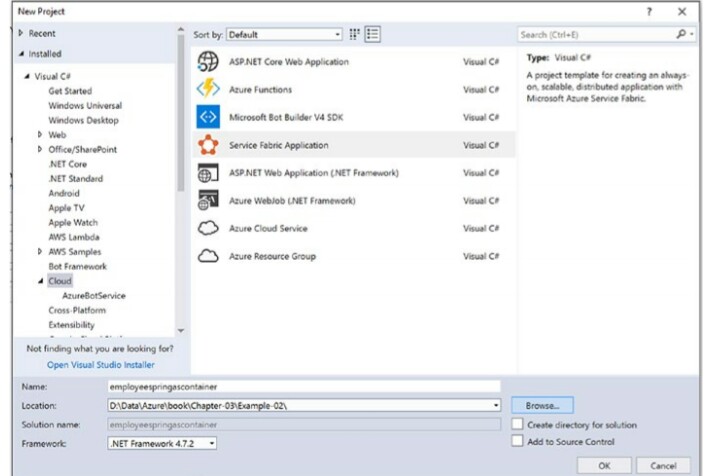
A screenshot of a computer

Description automatically generated

4. In New Service Fabric Service, select the following.

a. Service Name: employeecontainerservice

b. Image Name: youracr.azurecr.io/book/employeespringservice/v1

 c. User Name: Your username in the Azure Container Registry

d. Host Port: 8090

e. Container Port: 8080

5. Once the solution is created, open the ApplicationManifest.xml. Specify the right password for the admin user. (Since this is a sample, we kept the password unencrypted; for real-word applications you have to encrypt the password

A computer screen with a menu

Description automatically generated

Now we are ready to build and deploy the

container to the local Service Fabric cluster. Since we have given the user information to download the image from Azure Container Registry, Visual Studio downloads and deploy the container to the local services fabric cluster.

1. Right-click the Service Fabric project and publish.

2. Select the local cluster profile to publish to the local Service Fabric cluster. To deploy to Azure, select the cloud profile. Make sure that the Service Fabric cluster is up and ready in your subscription

A screenshot of a computer

Description automatically generated

.

3. Browse the Service Fabric dashboard. The default URL is http://localhost:19080/Explorer/index.html. Your service is deployed.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

4. Browse to http://localhost:8090/ to access your service. You get the response which is served from the container run by Service Fabric.

A screenshot of a computer

Description automatically generated