Construyendo microservicios usando Spring Boot y Docker - Parte 2

por Shamik Mitra MVB · **25 y 18 de febrero · Zona de microservicios**

Los microservicios en contenedores requieren una nueva supervisión. Vea por qué se necesita un nuevo enfoque de APM para incluso ver aplicaciones en contenedores .

En la segunda mitad de este tutorial, crearemos la búsqueda de empleados y otros servicios, y luego implementaremos sus microservicios con Docker.

Crear el servicio de búsqueda de empleados

Ahora crearemos un pequeño microservicio que en realidad devuelve la información del empleado en función de la ID aprobada. Además, puede devolver toda la información del empleado. Expondré una API REST y registraré este Microservicio con el servidor Eureka para que otros microservicios puedan encontrarlo.

Elegimos EurekaClient de start.spring.io.

Veamos el pom.xml para esto:

```
<? xml  version = "1.0" encoding = "UTF-8"?>
< proyecto
xmlns = "http://maven.apache.org/POM/4.0.0"
xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi: schemaLocation = "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maver
< modelVersion > 4.0.0 </ modelVersion >
< ID de grupo > com.example </ groupId >
< artifactId > EmployeeSearchService </ artifactId >
< version > 0.0.1-SNAPSHOT </ version >
< packaging > jar </ packaging >
< name > EmployeeSearchService </ name >
< description > Proyecto de demostración para Spring Boot </ description >
< padre >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-parent </ artifactId >
< version > 1.5.4.RELEASE </ version >
< relativePath />
<! - buscar padre del repositorio ->
</ parent >
```

```
< propiedades >
< project.build.sourceEncoding > UTF-8 </ project.build.sourceEncoding >
< project.reporting.outputEncoding > UTF-8 </ project.reporting.outputEncoding >
< java.version > 1.8 </ java.version >
< spring-cloud.version > Dalston.SR1 </ spring-cloud.version >
</ propiedades >
< dependencias >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-actuator </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-config </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-jersey </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-web </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-test </ artifactId >
< scope > prueba </ scope >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-eureka </ artifactId >
</ dependency >
</ dependencias >
< dependencyManagement >
< dependencias >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-dependencies </ artifactId >
< version > $ {spring-cloud.version} </ version >
< tipo > pom </ type >
< scope > import </ scope >
</ dependency >
</ dependencias >
</ dependencyManagement >
< construir >
< plugins >
< plugin >
```

```
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-maven-plugin </ artifactId >
</ plugin >
</ plugins >
</ build >
</ project >
```

Veamos el bootstrap.properties:

```
spring.unpplication.nunme=EmploYeeSeunrch
spring.cloud.config.uri=http://localhost:9090
eureka.client.serviceUrl.defunaultZone:http://
server.port=8080
security.basic.enable:false
management.security.enabled:false
```

Aquí, doy el nombre lógico del servicio **EmployeeSerach** , todas las instancias de este servicio registradas con este nombre en el servidor Eureka, este es el nombre lógico común para todas las instancias del Servicio EmployeeSerach. también doy la URL del servidor de configuración (Tenga en cuenta que cuando lo implementemos en la ventana acoplable, deberíamos cambiar el host local al contenedor Docker IP del servidor de configuración para encontrar el servidor de configuración)

Además, menciono la ubicación del servidor Eureka (Tenga en cuenta que cuando lo implementemos en la ventana acoplable, deberíamos cambiar el host local al IP del contenedor Docker de Eureka para encontrar el servidor Eureka,

Ahora crea el controlador y el archivo de servicio

```
paquete  com . ejemplo . EmployeeSearchService . servicio ;

importar  java . util . Colección ;
importar  java . util . HashMap ;
importar  java . util . Mapa ;
importar  java . util . corriente . Coleccionistas ;
importar  java . util . corriente . Corriente ;

import  org . springframework . estereotipo . Servicio ;

importación  com . ejemplo . EmployeeSearchService . dominio . modelo . Empleado ;

@Servicio
public  class  EmployeeSearchService {
```

```
15
16  private  static  Map  <  Long , Employee  >  EmployeeRepsitory  =  null ;
17
18  static {
19
     Stream  <  String  >  employeeStream  =  Stream . de ( "1, Shamik Mitra, Java, arquitect
20
      "3, Swastika Mitra, AI, Sr. Arquitecto" );
21
22
23   EmployeeRepsitory  =  employeeStream . map ( employeeStr  -> {
24    String [] info  =  employeeStr . división ( "," );
      return  createEmployee ( nuevo  Long ( info [ 0 ]), info [ 1 ], info [ 2 ], info [ 3 ]
25
26   }). collect ( Colectores . toMap ( Employee :: getEmployeeId , emp  ->  emp ));
27
28  }
29
     private  static  Employee  createEmployee ( Identificación larga  , String name , String
30
31   Empleado  emp  =  nuevo  Empleado ();
32   emp . setEmployeeId ( id );
33   emp . setName ( nombre );
34   emp . setPracticeArea ( practiceArea );
35   emp . setDesignation ( designación );
36   emp . setCompanyInfo ( "Cognizant" );
37   devolver  emp ;
38  }
39
40   Empleado  público findById ( Identificación larga  ) {
41   return  EmployeeRepsitory . get ( id );
42  }
43
44   Colección  pública <  Empleado  >  findAll () {
45   return  EmployeeRepsitory . valores ();
46  }
47
48 }
```

Archivo controlador

```
1  paquete  com . ejemplo . EmployeeSearchService . controlador ;
2
3  importar  java . util . Colección ;
4
5  import  org . springframework . habas . fábrica . anotación . Autocableado ;
6  import  org . springframework . nube . contexto . config . anotación . RefreshScope ;
7  import  org . springframework . web . se unen . anotación . PathVariable ;
```

```
import  org . springframework . web . se unen . anotación . RequestMapping ;
import  org . springframework . web . se unen . anotación . RestController ;

importación  com . ejemplo . EmployeeSearchService . dominio . modelo . Empleado ;
importación  com . ejemplo . EmployeeSearchService . servicio . EmployeeSearchService ;


@RefreshScope
@RestController
 clase  pública EmployeeSearchController {

 @Autowired
 EmployeeSearchService  employeeSearchService ;

 @RequestMapping ( "/ employee / find / {id}" )
  empleado  público findById ( @PathVariable  Long  id ) {
  return  employeeSearchService . findById ( id );
 }

 @RequestMapping ( "/ employee / findall" )
  Colección  pública <  Empleado  >  findAll () {
  return  employeeSearchService . findAll ();
 }
}
```

```
paquete  com . ejemplo . EmployeeSearchService . dominio . modelo ;

 Empleado de clase  pública {
 private  Long  employeed ;
  nombre de cadena  privada ;
 private  String  practiceArea ;
  designación de cadena  privada ;
 private  String  companyInfo ;
 public  Long  getEmployeeId () {
  return  employeeId ;
 }
 public  void  setEmployeeId ( Long  employeeId ) {
  esta . employeeId  =  employeeId ;
 }
 public  String  getName () {
   nombre de regreso ;
 }
 public  void  setName ( String  name ) {
  esta . nombre  =  nombre ;
 }
 public  String  getPracticeArea () {
  return  practiceArea ;
 }
```

```
}
public void setPracticeArea ( String practiceArea ) {
 esta . practiceArea = practiceArea ;
}
public String getDesignation () {
  designación de regreso ;
}
public void setDesignation ( String designation ) {
 esta . designación = designación ;
}
public String getCompanyInfo () {
 devolver companyInfo ;
}
public void setCompanyInfo ( String companyInfo ) {
 esta . companyInfo = companyInfo ;
}
@Anular
public String toString () {
 return "Empleado [employeeId =" + employeeId + ", name =" + name + ", practiceA
}



}
```

No hay nada lujoso aquí, solo creo unos empleados y los asigno a la URL de reposo.

veamos el archivo Spring Boot ahora

```
paquete com . ejemplo . EmployeeSearchService ;

import org . springframework . arranque . SpringApplication ;
import org . springframework . arranque . autoconfiguración . SpringBootApplication ;
import org . springframework . nube . cliente . descubrimiento . EnableDiscoveryClient ;
import org . springframework . nube . netflix . Eureka . EnableEurekaClient ;

@EnableDiscoveryClient
@SpringBootApplication
public class EmployeeSearchServiceApplication {

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

```
}
```

Aquí, uso @ EnableDiscoveryClient para registrar este servicio como un cliente eureka.

Ahora si presiono este http: // localhost: 8080 / employee / find / 1 puedo ver la siguiente salida

```
{
   "employeeId" : 1 ,
   "nombre" : "Shamik Mitra" ,
   "practiceArea" : "Java" ,
   "designación" : "Arquitecto" ,
   "companyInfo" : "Cognizant"
}
```

Crear el servicio del tablero de instrumentos del empleado

Ahora crearé otro servicio que use el Servicio de búsqueda de empleados para obtener información del empleado para comunicarse con el servicio de búsqueda de empleados. Voy a utilizar el cliente Feign y también utilizar Hystrix como un interruptor de circuito, por lo que si el servicio de búsqueda de empleados se ha reducido, se puede dar datos por defecto.

Pom.xml:

```
<? xml  version = "1.0" encoding = "UTF-8"?>
< proyecto
xmlns = "http://maven.apache.org/POM/4.0.0"
xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi: schemaLocation = "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maver
< modelVersion > 4.0.0 </ modelVersion >
< ID de grupo > com.example </ groupId >
< artifactId > EmployeeDashBoardService </ artifactId >
< version > 0.0.1-SNAPSHOT </ version >
< packaging > jar </ packaging >
< name > EmployeeDashBoardService </ name >
< description > Proyecto de demostración para Spring Boot </ description >
< padre >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-parent </ artifactId >
< version > 1.5.4.RELEASE </ version >
< relativePath />
<! - buscar padre del repositorio ->
</ parent >
< propiedades >
< project.build.sourceEncoding > UTF-8 </ project.build.sourceEncoding >
```

```
< project.reporting.outputEncoding > UTF-8 </ project.reporting.outputEncoding >
< java.version > 1.8 </ java.version >
< spring-cloud.version > Dalston.SR1 </ spring-cloud.version >
</ propiedades >
< dependencias >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-actuator </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-config </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-eureka </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-jersey </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-web </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-starter-test </ artifactId >
< scope > prueba </ scope >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-fingir </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-ribbon </ artifactId >
</ dependency >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
< artifactId > spring-cloud-starter-hystrix </ artifactId >
</ dependency >
</ dependencias >
< dependencyManagement >
< dependencias >
< dependencia >
< groupId > org.springframework.cloud </ groupId >
```

```
< artifactId > spring-cloud-dependencies </ artifactId >
< version > $ {spring-cloud.version} </ version >
< tipo > pom </ type >
< scope > import </ scope >
</ dependency >
</ dependencias >
</ dependencyManagement >
< construir >
< plugins >
< plugin >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-maven-plugin </ artifactId >
</ plugin >
</ plugins >
</ build >
</ project >
```

bootstrap.properties:

```
spring.unpplication.nunme=EmployeeDunshBounrd
spring.cloud.config.uri=http://localhost:9090
eureka.client.serviceUrl.defunaultZone:http://
server.port=8081
security.basic.enable:false
management.security.enabled:false
```

Feign cliente:

```
paquete  com . ejemplo . EmployeeDashBoardService . controlador ;

importar  java . util . Colección ;

import  org . springframework . nube . netflix . fingir . FeignClient ;
import  org . springframework . nube . netflix . cinta . RibbonClient ;
import  org . springframework . web . se unen . anotación . PathVariable ;
import  org . springframework . web . se unen . anotación . RequestMapping ;

importación  com . ejemplo . EmployeeDashBoardService . dominio . modelo . EmployeeInfo ;




@FeignClient ( name  =  "EmployeeSearch" )
@RibbonClient ( name  =  "EmployeeSearch" )
```

```
@
 interfaz  pública EmployeeServiceProxy {

 @RequestMapping ( "/ employee / find / {id}" )
 public  EmployeeInfo  findById ( @PathVariable ( value  =  "id" ) Identificación larga  )

 @RequestMapping ( "/ employee / findall" )
  Colección  pública <  EmployeeInfo  >  findAll ();

}
```

Controlador:

```
paquete  com . ejemplo . EmployeeDashBoardService . controlador ;

importar  java . util . Colección ;

import  org . springframework . habas . fábrica . anotación . Autocableado ;
import  org . springframework . nube . contexto . config . anotación . RefreshScope ;
import  org . springframework . web . se unen . anotación . PathVariable ;
import  org . springframework . web . se unen . anotación . RequestMapping ;
import  org . springframework . web . se unen . anotación . RestController ;

importación  com . ejemplo . EmployeeDashBoardService . dominio . modelo . EmployeeInfo ;

@RefreshScope
@RestController
 clase  pública FeignEmployeeInfoController {

 @Autowired
 EmployeeServiceProxy  proxyService ;

 @RequestMapping ( "/ dashboard / fingir / {myself}" )
 public  EmployeeInfo  findme ( @PathVariable  Long  yo mismo ) {
  devolver  proxyService . findById ( yo mismo );

 }

 @RequestMapping ( "/ dashboard / fingir / peers" )
  Colección  pública <  EmployeeInfo  >  findPeers () {
  devolver  proxyService . findAll ();
 }

}
```

Servicio de arranque de arranque de primavera

Servicio de arranque de arranque de primavera

```
paquete  com . ejemplo . EmployeeDashBoardService ;

import  org . springframework . arranque . SpringApplication ;
import  org . springframework . arranque . autoconfiguración . SpringBootApplication ;
import  org . springframework . arranque . web . cliente . RestTemplateBuilder ;
import  org . springframework . nube . cliente . interruptor de circuito . EnableCircuitBr
import  org . springframework . nube . cliente . descubrimiento . EnableDiscoveryClient ;
import  org . springframework . nube . netflix . fingir . EnableFeignClients ;
import  org . springframework . contexto . anotación . Frijol ;
import  org . springframework . web . cliente . RestTemplate ;

@EnableDiscoveryClient
@EnableFeignClients
@SpringBootApplication
public  class  EmployeeDashBoardService {

public  static  void  main ( String [] args ) {
SpringApplication . ejecutar ( EmployeeDashBoardService . clase , args );
}

@Frijol
public  RestTemplate  restTemplate ( RestTemplateBuilder  builder ) {
volver  constructor . construir ();
}
}
```

Estamos listos ahora. Si presiono la URL http: // localhost: 8081 / dashboard / feign / 1, veré la siguiente respuesta:

```
{
   "employeeId" : 1 ,
   "nombre" : "Shamik Mitra" ,
   "practiceArea" : "Java" ,
   "designación" : "Arquitecto" ,
   "companyInfo" : "Cognizant"
}
```

Creando el servicio de puerta de enlace

pom.xml:

```
<? xml  version = "1.0" encoding = "UTF-8"?>
< proyecto
xmlns = "http://maven.apache.org/POM/4.0.0"
xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"  xsi: schemaLocation = "http://ma
< modelVersion > 4.0.0 </ modelVersion >
< padre >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-dependencies </ artifactId >
< version > 1.5.6.RELEASE </ version >
< relativePath > ../../spring-boot-dependencies </ relativePath >
</ parent >
< artifactId > spring-boot-starter-parent </ artifactId >
< packaging > pom </ packaging >
< nombre > Spring Boot Starter Parent </ name >
< description > Parent pom proporciona dependencia y administración de complementos para a
construido con Maven </ description >
< url > http://projects.spring.io/spring-boot/ </ url >
< organización >
< nombre > Pivotal Software, Inc. </ name >
< url > http://www.spring.io </ url >
</ organization >
< propiedades >
< java.version > 1.6 </ java.version >
< resource.delimiter > @ </ resource.delimiter >
<! - delimitador que no entra en conflicto con Spring $ {} marcadores de posición ->
< project.build.sourceEncoding > UTF-8 </ project.build.sourceEncoding >
< project.reporting.outputEncoding > UTF-8 </ project.reporting.outputEncoding >
< maven.compiler.source > $ {java.version} </ maven.compiler.source >
< maven.compiler.target > $ {java.version} </ maven.compiler.target >
</ propiedades >
< dependencyManagement >
< dependencias >
< dependencia >
< groupId > org.springframework </ groupId >
< artifactId > spring-core </ artifactId >
< version > $ {spring.version} </ version >
< exclusiones >
< exclusion >
< groupId > commons-logging </ groupId >
< artifactId > commons-logging </ artifactId >
</ exclusion >
</ exclusions >
</ dependency >
</ dependencias >
```

```
</ dependencyManagement >
< construir >
<! - Activar el filtrado por defecto para las propiedades de la aplicación ->
< recursos >
< recurso >
< directorio > $ {basedir} / src / main / resources </ directory >
< filtrado > verdadero </ filtro >
< incluye >
< include > ** / application * .yml </ include >
< include > ** / application * .yaml </ include >
< include > ** / application * .properties </ include >
</ includes >
</ resource >
< recurso >
< directorio > $ {basedir} / src / main / resources </ directory >
< excluye >
< exclude > ** / application * .yml </ exclude >
< exclude > ** / application * .yaml </ exclude >
< exclude > ** / application * .properties </ exclude >
</ excludes >
</ resource >
</ resources >
< pluginManagement >
< plugins >
<! - Aplicar más valores predeterminados razonables para proyectos de usuario ->
< plugin >
< groupId > org.apache.maven.plugins </ groupId >
< artifactId > maven-failsafe-plugin </ artifactId >
< ejecuciones >
< ejecución >
< objetivos >
< objetivo > integración-prueba </ objetivo >
< objetivo > verificar </ objetivo >
</ goals >
</ execution >
</ executions >
</ plugin >
< plugin >
< groupId > org.apache.maven.plugins </ groupId >
< artifactId > maven-jar-plugin </ artifactId >
< configuración >
< archivo >
< manifestar >
< clase principal > $ {clase de inicio} </ clase principal >
< addDefaultImplementationEntries > true </ addDefaultImplementationEntries >
</ manifestar >
</ archive >
```

```
</ configuration >
</ plugin >
< plugin >
< groupId > org.apache.maven.plugins </ groupId >
< artifactId > maven-surefire-plugin </ artifactId >
< configuración >
< incluye >
< include > ** / * Tests.java </ include >
< include > ** / * Test.java </ include >
</ includes >
< excluye >
< exclude > ** / Abstract * .java </ exclude >
</ excludes >
</ configuration >
</ plugin >
< plugin >
< groupId > org.apache.maven.plugins </ groupId >
< artifactId > maven-war-plugin </ artifactId >
< configuración >
< failOnMissingWebXml > falso </ failOnMissingWebXml >
< archivo >
< manifestar >
< clase principal > $ {clase de inicio} </ clase principal >
< addDefaultImplementationEntries > true </ addDefaultImplementationEntries >
</ manifestar >
</ archive >
</ configuration >
</ plugin >
< plugin >
< groupId > org.codehaus.mojo </ groupId >
< artifactId > exec-maven-plugin </ artifactId >
< configuración >
```

```
< clase principal > $ {clase de inicio} </ clase principal >

</ configuration >

</ plugin >

< plugin >

< groupId > org.apache.maven.plugins </ groupId >

< artifactId > maven-resources-plugin </ artifactId >

< version > 2.6 </ version >

< configuración >

< delimitadores >

< delimitador > $ {recurso.delimitador} </ delimitador >

</ delimitadores >

< useDefaultDelimiters > falso </ useDefaultDelimiters >

</ configuration >

</ plugin >

< plugin >

< groupId > pl.project13.maven </ groupId >

< artifactId > git-commit-id-plugin </ artifactId >

< ejecuciones >

< ejecución >

< objetivos >

< objetivo > revisión </ objetivo >

</ goals >

</ execution >

</ executions >

< configuración >

< verbose > true </ verbose >

< dateFormat > aaaa-MM-dd'T'HH: mm: ssZ </ dateFormat >

< generateGitPropertiesFile > true </ generateGitPropertiesFile >

< generateGitPropertiesFilename > $ {project.build.outputDirectory} /git.properties </ ger

</ configuration >
```

```
</ plugin >
<! - Admite nuestro propio complemento ->
< plugin >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-maven-plugin </ artifactId >
< ejecuciones >
< ejecución >
< objetivos >
< objetivo > reempaquetado </ objetivo >
</ goals >
</ execution >
</ executions >
< configuración >
< clase principal > $ {clase de inicio} </ clase principal >
</ configuration >
</ plugin >
<! - Soporte de empaque de color (si el usuario no desea usar nuestro complemento) ->
< plugin >
< groupId > org.apache.maven.plugins </ groupId >
< artifactId > maven-shade-plugin </ artifactId >
< dependencias >
< dependencia >
< groupId > org.springframework.boot </ groupId >
< artifactId > spring-boot-maven-plugin </ artifactId >
< version > 1.5.6.RELEASE </ version >
</ dependency >
</ dependencias >
< configuración >
< keepDependenciesWithProvidedScope > true </ keepDependenciesWithProvidedScope >
< createDependencyReducedPom > true </ createDependencyReducedPom >
```

```
< createDependencyReducedPom > true </ createDependencyReducedPom >

< filtros >

< filtro >

< artefacto > *: * </ artefacto >

< excluye >

< excluir > META-INF / *. SF </ exclude >

< excluir > META-INF / *. DSA </ exclude >

< excluir > META-INF / *. RSA </ exclude >

</ excludes >

</ filter >

</ filtros >

</ configuration >

< ejecuciones >

< ejecución >

< fase > paquete </ fase >

< objetivos >

< objetivo > sombra </ objetivo >

</ goals >

< configuración >

< transformadores >
< transformation  implementation = "org.apache.maven.plugins.shade.resource.AppendingTrans

< resource > META-INF / spring.handlers </ resource >

</ transformador >
< transformation  implementation = "org.springframework.boot.maven.PropertiesMergingResour

< resource > META-INF / spring.factories </ resource >

</ transformador >
< transformation  implementation = "org.apache.maven.plugins.shade.resource.AppendingTrans

< resource > META-INF / spring.schemas </ resource >

</ transformador >
< transformation  implementation = "org.apache.maven.plugins.shade.resource.ServicesResour
```

```
< transformation  implementation = "org.apache.maven.plugins.shade.resource.ManifestResour

< clase principal > $ {clase de inicio} </ clase principal >

</ transformador >

</ transformers >

</ configuration >

</ execution >

</ executions >

</ plugin >

</ plugins >

</ pluginmanagement >

</ build >

</ project >
```

bootstrap.properties:

```
spring.unpplication.nunme=EmploYeeAPIGuntewunay
eureka.client.serviceUrl.defunaultZone:http://
server.port=8084
security.basic.enable:false
management.security.enabled:false
zuul.routes.emploYeeUI.servIcEId=EmploYeeDuns
zuul.host.socket-timeout-millis=30000
```

Aquí, observe la propiedad **zuul.routes.employeeUI.serviceId = EmployeeDashBoard** . Con esto, instruimos a Zuul que cualquier URL que contenga la UI del empleado debe redirigirse al servicio EmployeeDashboard.

Spring Boot file:

```
paquete  com . ejemplo . EmployeeZuulService ;


import  org . springframework . arranque . SpringApplication ;
import  org . springframework . arranque . autoconfiguración . SpringBootApplication ;
import  org . springframework . nube . cliente . descubrimiento . EnableDiscoveryClient ;
import  org . springframework . nube . netflix . zuul . EnableZuulProxy ;
```

```
7
8  @EnableZuulProxy
9  @EnableDiscoveryClient
10 @SpringBootApplication
11 public  class  EmployeeZuulServiceApplication {
12
13  public  static  void  main ( String [] args ) {
14   SpringApplication . ejecutar ( EmployeeZuulServiceApplication . class , args );
15  }
16 }
```

Ahora si ejecuto el servicio y presiono http: // localhost: 8084 / employeeUI / dashboard / feign / 1, nos da esta respuesta:

```
1  {
2     "employeeId" : 1 ,
3     "nombre" : "Shamik Mitra" ,
4     "practiceArea" : "Java" ,
5     "designación" : "Arquitecto" ,
6     "companyInfo" : "Cognizant"
7  }
```

Despliegue en contenedores Docker

Ya basta de codificar. Veamos cómo se está ejecutando nuestra aplicación. En este momento, todos nuestros servicios están listos y funcionando perfectamente en la máquina local. Pero no queremos que nuestro código se ejecute solo en su configuración local. Más bien, queremos verlo funcionando con gran éxito en producción. (Amamos nuestro código como nuestro bebé y queremos verlo exitoso todo el tiempo). Pero a medida que enviamos a nuestros bebés a la escuela o los guiamos por el camino correcto hacia el éxito, también tenemos que guiar nuestra aplicación. Así que démonos un paseo por el mundo **DevOps** e intentemos darle a nuestro código fuente el camino correcto hacia la producción.

Bienvenido al Docker World

Docker no necesita presentación. Si sientes que todavía necesitas una guía, no dudes en echar un vistazo aquí https://docs.docker.com/get-started/ .

En el futuro, supongo que tiene instalado Docker CE en su máquina. Los conceptos que utilizaremos aquí para la implementación son los siguientes:

1. **Dockerfile** : este es un documento de texto que contiene todas las instrucciones necesarias para construir una imagen Docker. Usando el conjunto de instrucciones de un archivo Docker, podemos escribir pasos que copiarán archivos, harán la instalación, etc. Para obtener más referencias, visite https://docs.docker.com/engine/reference/builder/ .

2. **Docker Compose** : esta es una herramienta que puede crear y generar múltiples contenedores. Ayuda a construir el entorno requerido con un solo comando.

Como se muestra en el diagrama de arquitectura de microservicios, crearemos un contenedor individual para cada servicio. A continuación se muestra la lista de contenedores para nuestro ejemplo:

1. Servidor de configuración
2. EmployeeService
3. Servicio de la Junta de Empleados
4. Servicio de panel de empleados
5. Servicio Gateway

Configuración de Docker para el servidor de configuración

El contenedor debe contener el archivo jar del servidor de configuración. Aquí, seleccionaremos el archivo jar de la máquina local. En un escenario de la vida real, debemos empujar el archivo jar a un sistema de administrador de repositorio de artefactos, como Nexus o Artifactory, y el contenedor debe descargar el archivo del administrador de repositorio.

El servidor de configuración debe estar disponible en el puerto 8888 según bootstrap.properties.

Como se mencionó anteriormente, el servidor de configuración leerá la configuración desde una ubicación de archivo, por lo que nos aseguraremos de que esos archivos de propiedades puedan recuperarse incluso si el contenedor se cae.

Cree una carpeta llamada config-repo que contendrá el archivo de propiedades requerido. Nos aseguraremos de lo siguiente para el contenedor del servidor de configuración.

```
1  #  mkdir  config - repo
2  #  cd  config - repo
   #  echo  "service.employyesearch.serviceId = EmployeeSearch"  >  EmployeeDashBoard . prop
3
4  #  echo  "user.role = Dev"  >  EmployeeSearch . propiedades
```

Regrese a la carpeta principal y cree un archivo Docker llamado Dockerfile. Este Dockerfile creará nuestra imagen base, que contiene Java.

```
1  #  cd .. /
2  #  vi  Dockerfile
```

Coloque en el contenido a continuación:

```
DESDE  alpino : borde
MANTENIMIENTO  javaonfly
EJECUTAR  apk  add  - no - caché  openjdk8
```

DESDE : Esta palabra clave le dice a Docker que use una imagen determinada con su etiqueta como base de construcción.

MANTENEDOR : UN MANTENEDOR es el autor de una imagen

EJECUTAR : este comando instalará openjdk8 en el sistema.

Ejecute el siguiente comando para crear la imagen Docker base:

```
docker build --tag=alpine-jdk:base --rm=true .
```

Una vez que la imagen base se haya creado correctamente, es hora de crear la imagen Docker para el Servidor de configuración.

Cree una carpeta llamada archivos y coloque el archivo jar del servidor de configuración en el directorio. Luego, cree un archivo llamado Dockerfile-configserver con el siguiente contenido:

```
DESDE  alpine - jdk : base
MANTENIMIENTO  javaonfly
Copiar  archivos / MicroserviceConfigServer . jar  / opt / lib /
EJECUTAR  mkdir  / var / lib / config - repo
COPY  config - repo  / var / lib / config - repo
ENTRYPOINT [ "/ usr / bin / java" ]
CMD [ "-jar" , "/opt/lib/MicroserviceConfigServer.jar" ]
VOLUME  / var / lib / config - repo
EXPOSE  9090
```

Aquí, hemos mencionado la creación de la imagen a partir de la imagen alpine-jdk creada anteriormente. Copiaremos el archivo jar denominado employeeconfigserver.jar en la ubicación / opt / lib y también copiaremos el config-repo en el directorio / root. Cuando el contenedor se inicia, queremos que el servidor de configuración comience a funcionar, por lo tanto, ENTRYPOINT y CMD están configurados para ejecutar el comando de Java. Necesitamos montar un volumen para compartir los archivos de configuración desde fuera del contenedor; el comando VOLUME nos ayuda a lograr eso. El servidor de configuración debe ser accesible al mundo exterior con el puerto 9090; es por eso que EXPOSE 9090.

Ahora construyamos la imagen de Docker y etiquétela como servidor de configuración:

```
#  docker  build  - file = Dockerfile - configserver  - tag = config - server : latest  -
```

Ahora vamos a crear un volumen Docker:

```
# docker volume create - name = config - repo
# docker run - name = config - server - publish = 9090 : 9090 - volume = config - rep
```

¡Uf! Una vez que ejecutemos el comando anterior, deberíamos poder ver un contenedor Docker activo y en ejecución. Si vamos al navegador y pulsamos la URL http: // localhost: 9090 / config / default / , también deberíamos poder acceder a las propiedades.

EurekaServer

Del mismo modo, tenemos que crear un archivo Docker para EurekaServer, que se ejecutará en el puerto 9091. El Dockerfile para Eureka Server debería ser el siguiente:

```
DESDE alpine - jdk : base
MANTENIMIENTO javaonfly
COPY files / MicroserviceEurekaServer . jar / opt / lib /
ENTRYPOINT [ "/ usr / bin / java" ]
CMD [ "-jar" , "/opt/lib/MicroserviceEurekaServer.jar" ]
EXPOSE 9091
```

Para construir la imagen, use este comando:

```
estibador construir --file = Dockerfile-EurekaServer --tag = Eureka-servidor: últimas --rr
docker run --name = eureka-server --publish = 9091 : 9091 eureka-server: más reciente
```

Microservicios

Ahora es el momento de implementar nuestros microservicios reales. Los pasos deben ser similares; lo único que debemos recordar es que nuestros microservicios dependen de ConfigServer y EurekaServer, por lo que siempre debemos asegurarnos de que antes de comenzar nuestros microservicios, los dos anteriores estén en funcionamiento. Hay dependencias involucradas entre los contenedores, por lo que es hora de explorar Docker Compose. Es una forma hermosa de asegurarse de que los contenedores que se generan se mantengan en un cierto orden.

Para hacer eso, deberíamos escribir un Dockerfile para el resto de los contenedores. A continuación se muestra el archivo Dockerfile:

```
Dockerfile - EmployeeSearch .
==============================
DESDE alpine - jdk : base
MANTENIMIENTO javaonfly
RUN apk - no - cache add netcat - openbsd
COPY files / EmployeeSearchService . jar / opt / lib /
COPY EmployeeSearch - entrypoint . sh / opt / bin / EmployeeSearch - entrypoint . sh
```

```
EJECUTAR  chmod  755  / opt / bin / EmployeeSearch - entrypoint . sh
EXPOSE  8080

Dockerfile - EmployeeDashboard
================================
DESDE  alpine - jdk : base
MANTENIMIENTO  javaonfly
RUN  apk  - no - cache  add  netcat - openbsd
COPY  files / EmployeeDashBoardService . jar  / opt / lib /
COPY  EmployeeDashBoard - entrypoint . sh  / opt / bin / EmployeeDashBoard - entrypoint .
EJECUTAR  chmod  755  / opt / bin / EmployeeDashBoard - entrypoint . sh
EXPOSE  8080

Dockerfile - ZuulServer
=======================================
DESDE  alpine - jdk : base
MANTENIMIENTO  javaonfly
COPY  files / EmployeeZuulService . jar  / opt / lib /
ENTRYPOINT [ "/ usr / bin / java" ]
CMD [ "-jar" , "/opt/lib/EmployeeZuulService.jar" ]
EXPOSE  8084
```

Una cosa es notar aquí que he creado dos scripts de shell para los servicios de Employee and Employee Dashboard, indica a Dockercompose no iniciar el servicio Employee and Employee Dashboard hasta que el servidor de configuración y el servidor Eureka hayan comenzado.

```
Empleado  dashBoard  Script
================================
# ! / bin / sh

mientras  !  nc  - z  config - servidor  9090 ; hacer
    echo  "Esperando el servidor de configuración"
    dormir  3
hecho
mientras  !  nc  - z  eureka - servidor  9091 ; hacer
    echo  "Esperando el Servidor Eureka"
    dormir  3
hecho

java  - jar  / opt / lib / EmployeeDashBoardService . tarro
================================
 Servicio del  empleado Script
================================
# ! / bin / sh

mientras  !  nc  - z  config - servidor  9090 ; hacer
    echo  "Esperando el servidor de configuración"
```

```
    dormir  3
hecho
mientras  !  nc  - z  eureka - servidor  9091 ; hacer
    echo  "Esperando el Servidor Eureka"
    dormir  3
hecho

java  - jar  / opt / lib / EmployeeSearchService . tarro
```

Ahora vamos a crear un archivo llamado docker-compose.yml, que usará todos estos archivos Docker para generar nuestros entornos requeridos. También se asegurará de que los contenedores requeridos que se generen estén manteniendo el orden correcto y estén interrelacionados.

```
versión : '2.2'
servicios :
    config-server :
        container_name : config-server
        construir :
            contexto : .
            dockerfile : Dockerfile-configserver
        imagen : config-server : más reciente
        exponer :
            - 9090
        puertos :
            - 9090: 9090
        redes :
            - emp-network
        volúmenes :
            - config-repo: / var / lib / config-repo

    eureka-servidor :
        container_name : eureka-server
        construir :
            contexto : .
            dockerfile : Dockerfile-EurekaServer
        imagen : eureka-servidor : más reciente
        exponer :
            - 9091
        puertos :
            - 9091: 9091
        redes :
            - emp-network

    EmployeeSearchService :
        container_name : EmployeeSearch
        construir :
```

```
        construir :
            contexto : .
            dockerfile : Dockerfile-EmployeeSearch
        imagen : empleadosearch : latest
        ambiente :
            SPRING_APPLICATION_JSON : '{"spring": {"cloud": {"config": {"uri": "http: // c

        punto de entrada : /opt/bin/EmployeeSearch-entrypoint.sh
        exponer :
            - 8080
        puertos :
            - 8080: 8080
        redes :
            - emp-network
        enlaces :
            - config-server: config-server
            - eureka-servidor: eureka-servidor
        depends_on :
            - config-server
            - eureka-servidor
        el registro de :
            controlador : json-file
    EmployeeDashboardService :
        container_name : EmployeeDashboard
        construir :
            contexto : .
            dockerfile : Dockerfile-EmployeeDashboard
        imagen : employeedashboard : último
        ambiente :
            SPRING_APPLICATION_JSON : '{"spring": {"cloud": {"config": {"uri": "http: // c

        punto de entrada : /opt/bin/EmployeeDashBoard-entrypoint.sh
        exponer :
            - 8081
        puertos :
            - 8081: 8081
        redes :
            - emp-network
        enlaces :
            - config-server: config-server
            - eureka-servidor: eureka-servidor
        depends_on :
            - config-server
            - eureka-servidor
        el registro de :
            controlador : json-file
```

```
78              controlador : json-file
79      ZuulServer :
80          container_name : ZuulServer
81          construir :
82              contexto : .
83              dockerfile : Dockerfile-ZuulServer
84          imagen : zuulserver : más reciente
85          exponer :
86              - 8084
87          puertos :
88              - 8084: 8084
89          redes :
90              - emp-network
91          enlaces :
92              - eureka-servidor: eureka-servidor
93          depends_on :
94              - eureka-servidor
95          el registro de :
96              controlador : json-file
97  redes :
98      emp-network :
99          conductor : puente
10
0   volúmenes :
10
1       config-repo :
10
2           externo : verdadero
```

En el archivo de redacción Docker a continuación hay algunas entradas importantes:

1. **versión** : un campo obligatorio donde necesitamos mantener la versión del formato Docker Compose.

2. **servicios** : cada entrada define el contenedor que necesitamos generar.

    1. **compilación** : si se menciona, Docker Compose debería crear una imagen a partir del Dockerfile proporcionado.

    2. **imagen** : el nombre de la imagen que se creará.

    3. **redes** : el nombre de la red que se utilizará. Este nombre debe estar presente en la sección de redes.

    4. **enlaces** : esto creará un enlace interno entre el servicio y el servicio mencionado. Aquí, el servicio EmployeeSearch necesita acceder al servidor config y Eureka.

5. **depende** : esto es necesario para mantener el orden. El contenedor EmployeeSearch depende del Eureka y el Servidor de configuración. Por lo tanto, Docker se asegura de que los contenedores Eureka y Config Server se generen antes de que se genere el contenedor EmployeeSearch.

Después de crear el archivo, construyamos nuestras imágenes, creemos los contenedores necesarios y comencemos con un solo comando:

```
docker-compose up --build
```

Para detener el entorno completo, podemos utilizar este comando:

```
docker-compose down
```

La documentación completa de Docker Compose se puede encontrar en https://docs.docker.com/compose/ .

Para resumir, escribir el archivo Dockerfile y Docker Compose es una actividad que se realiza una sola vez, pero le permite generar un entorno completo bajo demanda en cualquier momento.

Conclusión

Esta es la guía completa sobre cómo construir diferentes componentes en microservicios y desplegarlos en Docker. En producción, debe haber CI / CD involucrados, por lo que no es necesario que conozca todos los comandos de Docker para crear imágenes, pero como desarrollador de pila completa, es importante aprender cómo puede crear y crear una imagen en Docker.

¿Tienes Cloud? Contenedores? Microservicios? Obtenga un mejor control y rendimiento. Comience su prueba de Instana APM hoy.

Me gusta este artículo? Leer más de DZone

Construyendo Microservicios Usando Spring Boot y Docker - Parte 1

Microservicios con Spring Boot, Axon CQRS / ES y Docker

Microservicios con arranque de resorte - Parte 3 - Creación de servicio de microversión de conversión de moneda

Gratis DZone Refcard
Microservicios en Java

Temas: MICROSERVICIOS, ARRANQUE POR RESORTE, ACOPLADOR, TUTORIAL

Las opiniones expresadas por los contribuidores de DZone son suyas.

Recursos para socios de **microservicios**

Asegurando las API de Microservice Nuevo eBook O'Reilly
CA Technologies

Gestión de la calidad del servicio en la era de los microservicios
Instana

The Five Principles of Monitoring Microservices - Best Practices.
Sysdig

Microservices Architecture eBook: Dowlnload Your Copy Here
CA Technologies