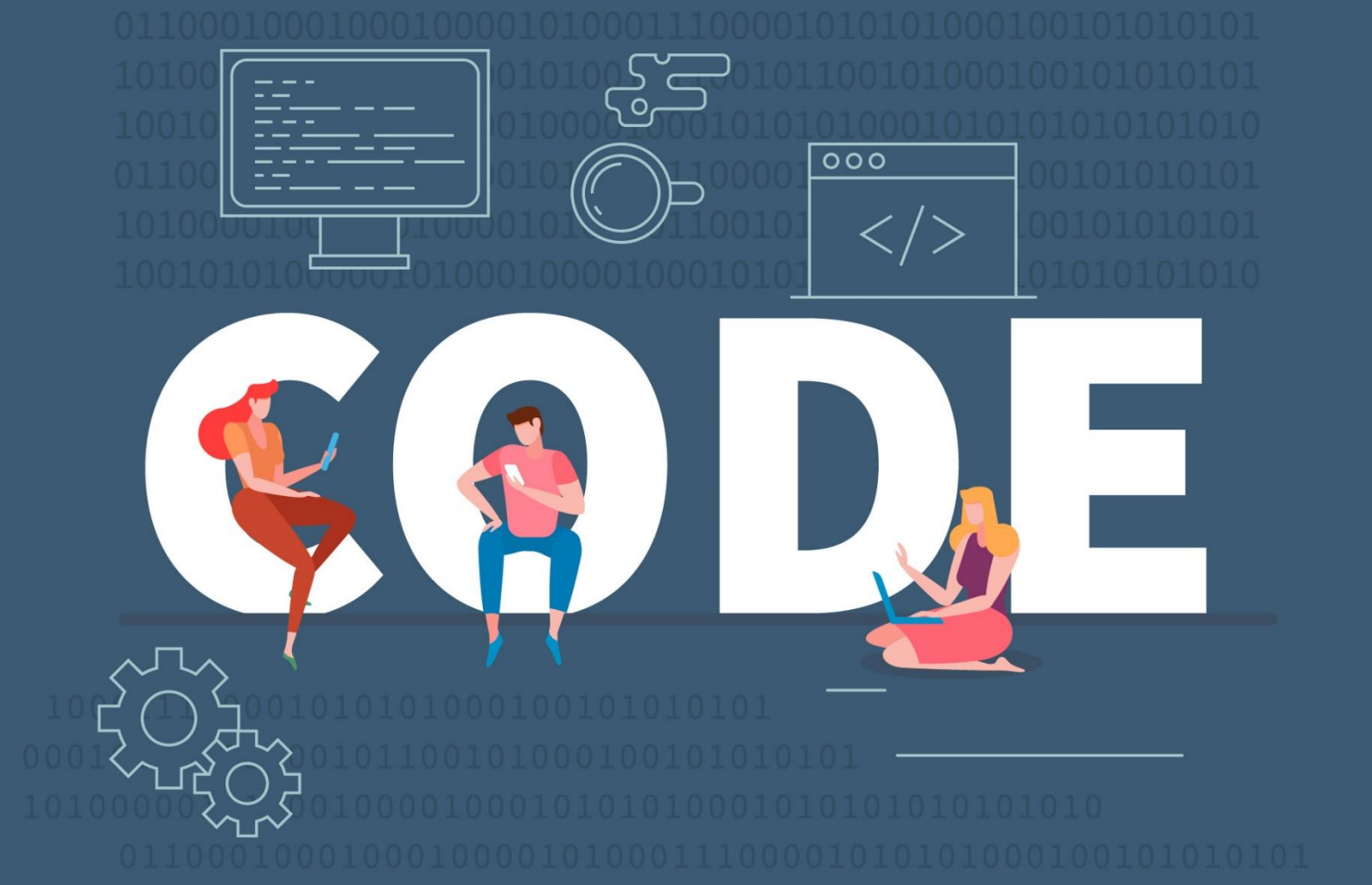
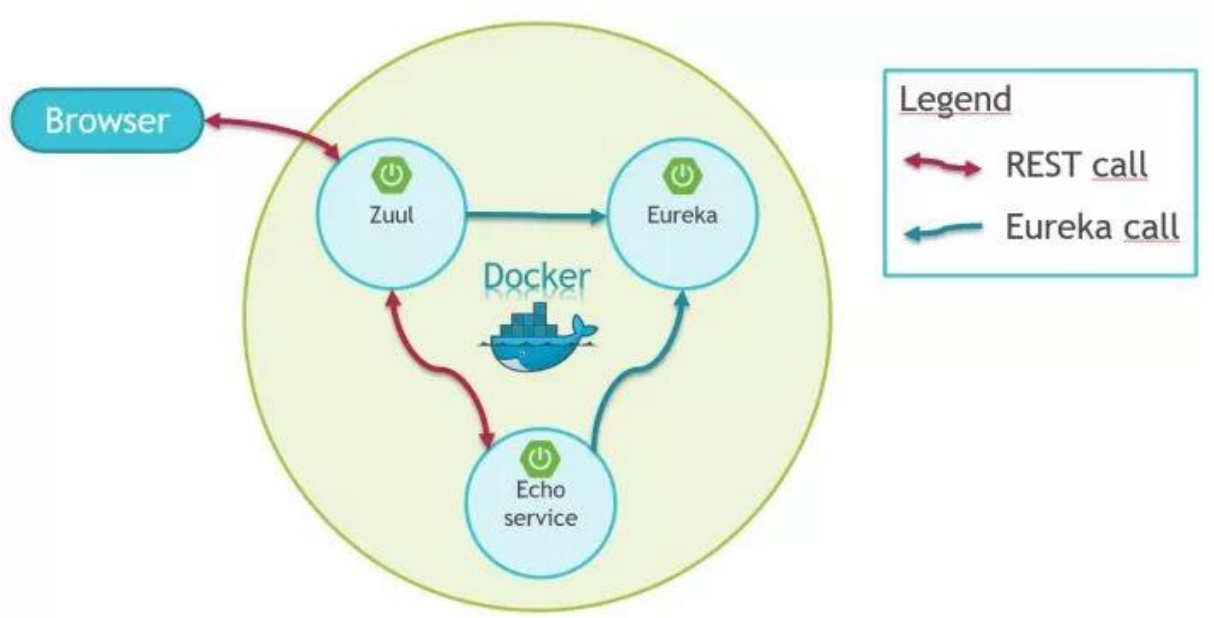


Microservices Tutorial Coding Thursday

Inhalt

Szenario

Das Bild unten hier beschreibt das Szenario der Vorlage, welche auf GitHub verfügbar ist. Es besteht aus einem Load-Balancer und einem Discovery Service mit dem Namen Eureka.



Es wird zu aller erst das Eureka Service gestartet, damit es für die anderen Services zur Verfügung steht. Beim initialisieren des Echo Services registriert sich dieses beim Eureka Service mit dem Namen seiner ArtefactId.

Währenddessen wird auch der Zuul Server hochgefahren die über vordefinierten Routen weiß, welche Services grundsätzlich verfügbar sind. Eureka kann dann auf Nachfrage sagen, wo die Services liegen, damit Zuul die Request weiterleiten bzw. verteilen kann.

Start

Holen (clone/pull) Sie die Vorlage von GitHub für dieses Tutorial mit

```
git clone https://github.com/FH-Tutorials/spring-boot-docker-example.git
```

und bauen Sie die Applikation mit vom Verzeichnis *spring-boot-docker-example* mit

```
mvn clean install
```

Wenn Sie danach `docker images` ausführen sollte sie diese Images gelistet bekommen

```
$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
echo-service                latest              0f0050d89093        22 hours ago        141MB
eureka-server               latest              cff611f1455a        22 hours ago        145MB
zuul-server                 latest              8390797aae92        22 hours ago        142MB
```

Mit dem Befehl `docker-compose up` können diese Images gestartet werden und sind dann als Container sichtbar mit `docker ps`:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
a24f43dc2980        echo-service        "java -jar /echo-ser…"   41 seconds ago      Up 39 sec
dcc69f178133        zuul-server         "java -jar /zuul-ser…"   41 seconds ago      Up 39 sec
750b6689cd39        eureka-server       "java -jar /eureka-s…"   7 minutes ago       Up 41 sec
```

Gemäß der README.md-Datei auf GitHub ist das Service jetzt auf

http://<<docker-host>>:9090/api/echo-service/echo
erreichbar. *<<docker-host>>* wird bei den meisten Installationen *localhost* entsprechen. Das Service sollte mit einem JSON String antworten der wie folgt aussieht:

```
{"address":"172.19.0.3","port":9098,"hostName":"a24f43dc2980"}
```

Nun kann mit

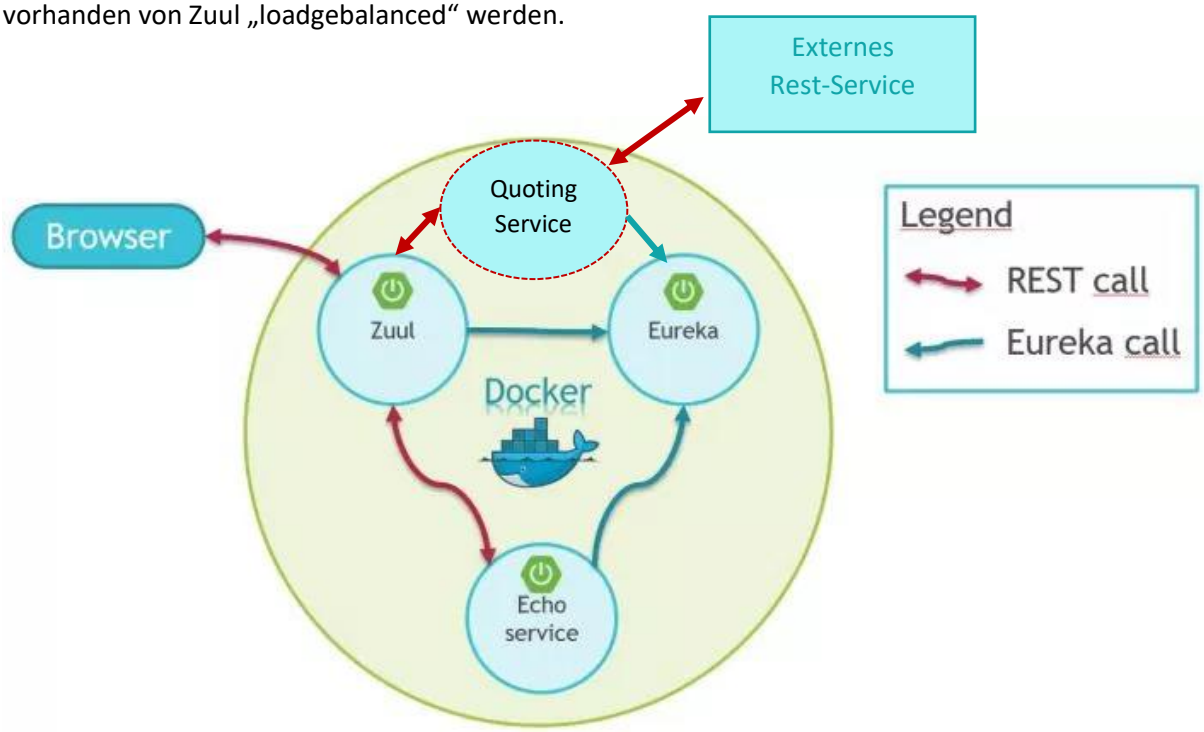
```
docker-compose scale echo-service=2
```

ein zweiter Echo-Service hochgefahren werden der, wenn er die Spring-Boot Scripts erfolgreich gestartet hat zur Verfügung steht und sich beim Zuul-Load Balancer sowie beim Eureka Discovery Service registriert hat.

Damit kann beim mehrmaligen Aufrufen des echo-service/echo festgestellt werden, dass sich die IP-Adresse alternierend ändert und damit natürlich auch der entsprechende Service der darauf geantwortet hat ein anderer ist.

Tutorial Ziel

Dieses Tutorial hat nun zum Ziel einen neuen Microservice zu erstellen, welcher ebenfalls in dieser Infrastruktur läuft. Dieses Service soll sich ebenfalls bei Eureka registrieren und falls mehrfach vorhanden von Zuul „loadgebalanced“ werden.



Bei Quoting-Service handelt es sich um eine Implementierung, die eine REST Anfrage an ein außenstehendes System stellen soll das JSON-Resultat verändern und anschließend ebenfalls an den Aufrufer zurückgeben.

Das externe Service ist öffentlich verfügbar unter http://gturnquist-quoters.cfapps.io/api/random und liefert, wie der Name schon vermuten lässt zufällige Zitate (Quotes).

Eine Beispielantwort könnte z.B. so aussehen:

```
{
    "type":"success",
    "value": {
        "id":7,
        "quote":"The real benefit of Boot, however, is that it's just Spring. That
    means any direction the code takes, regardless of complexity, I know it's a safe
    bet."

    }

}
```

Unser Server soll also nun diesen JSON-String parsen und in ein Objekt überführen. Damit wir sehen, dass diese wirklich verarbeitet wurde führen wir eine kleine Änderung an der Quote selbst durch. Also z.B. irgendeine Zeichenkette anhängen und schicken wie erwähnt diese zum Aufrufer retour.

Wie geht es weiter

Hier nun die Schritte zum Erstellen und integrieren des Service.

Erstellen eines Maven Moduls

Entweder manuell oder z.B. in Eclipse mit File->New Project->New Maven **Module** ein neues Mavenmodul erstellen. Mit

- Modul Name: quoting-service,
- Parent: spring-boot-docker-example
- Auswählen: „Create Simple Project (skip archetype selection)“

und anschließendem Klicken von **“FINISH”**. Damit wird ein neues Modul erstellt und es im Elternprojekt eingetragen.

Pom.xml & application.yml

Unser *quoting-service* benötigt die gleichen Abhängigkeiten wie das vorhandene *echo-service*, deswegen können wir die *pom.xml* des *echo-services* in die ***pom.xml*** des *echo-service* kopieren und anschließend die *artefactId* auf „quoting-service“ ändern. Ebenso der <name> ist sinnvollerweise mit „Quoting Service“ zu setzen.

Kurze Besprechung: pom.xml

Das gleich kann man nun mit der Datei „**application.yml**“ aus dem echo-service machen. Daher einfach die Datei von echo-service/src/main/java/main/resources in das gleichnamige Verzeichnis im quoting-service kopieren und dann wieder den Namen in Zeile 18 auf quoting service ändern.

Kurze Besprechung: application.yaml

Fügen Sie nun zusätzlich die URL die externen Services hinzu. Hier ist der Eintrag:

```
service:

  location: http://gturnquist-quoters.cfapps.io/api/random
```

Achtung: yaml Dateien müssen korrekt eingerückt sein damit sie richtig gelesen werden.

Application.java

Als nächstes ist nun die eigentliche Applikation dran. Legen sie nun im package „org.exampledriven.docker.quoting“ ein Klasse mit dem Namen „**Application**“ an, welche so aussieht:

```
package org.exampledriven.docker.quoting;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@EnableWebMvc
@SpringBootApplication
@EnableDiscoveryClient
public class Application {

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

}
```

Hier wird die Applikation mit drei Annotationen gestartet:

- `@EnableWebMvc` Erstellen einer MVC Applikation
- `@SpringBootApplication` Als SpringBoot Applikation annotieren
- `@EnableDiscoveryClient` Als DiscoveryClient agieren

Essentiell hierbei ist besonders die Annotation @EnableDiscoveryClient, welche dafür sorgt, dass der Client sich beim Eureka Service registriert. Die Lokation kann im application.yml file gefunden werden.

Die Model Files

Im Package „`org.exampledriven.docker.quoting.model`” sind nun zwei Container Objekte zu erstellen die den Status des externen Services halten können.

1. Objekt „Value“
   - mit Attribut “private Long id;”
   - und Attribut „private String quote;“
2. Objekt “Quote”
   - mit Attribut “private String type;”
   - und Attribut “private Value value;”

Lassen Sie sich beim Erstellen den Default-Konstruktor sowie getter und setter von Eclipse erstellen.

Annotieren Sie bitte ebenfalls die Klassen mit dem Attribut „@JsonIgnoreProperties(ignoreUnknown = true)“ damit es nicht zu Überraschungen kommt, wenn in dem JSON String Attribute vorkommen die nicht eingeplant wurden.

QuoteController

Der QuoteController ist der Teil der Applikation, der nun tatsächlich ein Request an das externe Quoting Service stellt und das JSON verarbeitet. Deswegen erstellen Sie nun im Package

„org.exampledriven.docker.quoting.rest“ die Klasse QuoteController und kopieren Sie diese Vorlage hinein:

```
package org.exampledriven.docker.quoting.rest;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;

import org.exampledriven.docker.quoting.model.Quote;

@RestController
public class QuoteController {
    @Autowired
    private RestTemplate restTemplate;

    @Value("${service.location}")
    private String serviceLocation;

    @RequestMapping(value = "/", method = RequestMethod.GET, produces =
"application/json")
    public Quote doQuote() {
      Quote quote = restTemplate.getForObject(serviceLocation, Quote.class);
      quote.getValue().setQuote( quote.getValue().getQuote() + " - attaching some
value.");
        return quote;
    }
}
```

Damit das *restTemplate* Attribut befüllt wird (Dependency Injection), muss natürlich noch ein „Bean“ definiert werden. Deswegen fügen wir in der Application.java noch folgende Methode hinzu:

```
@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
      return builder.build();
}
```

Route Services

Jetzt müssen wir noch, damit der Zuul-Server über unser Service bescheid weiß, eine Route dafür definieren. Deswegen fügen wir unter halb der Route des echo-services jetzt noch unser quoting-service ein. Der yaml code sollte so aussehen:

```
  routes:
    echo-by-dns:
      path: /echo-service-by-dns/**
      url: http://echo-service:9098
    quoting-service:
      path: /quoting-service/**
      url: http://quoting-service:9098
```

Nochmals zur Erinnerung: Yaml-Files müssen richtig eingerückt sein, damit sie interpretiert werden können.

docker-compose.yaml

Unsere Implementierung des Service ist jetzt fertig. Jedoch muss es noch korrekt gestartet werden. Damit docker-compose das starten für uns übernimmt muss jetzt nur noch ein Eintrag im docker-compose.yaml file erstellt werden. Der Eintrag ist bis auf den Namen der gleiche wie beim echo-service und sieht so aus:

```
  quoting-service:
    image: echo-service
    depends_on:
     - eureka-server
    deploy:
      mode: replicated
      replicas: 2
```

Bauen und starten

Die Applikation kann jetzt neu gebaut werden mit `mvn clean install`. Dabei wird durch das *com.spotify* plugin ebenfalls die Docker-Images gebaut.

Mit `docker-compose up` wird das Service gestartet und kann verwendet werden. Das neue Service ist nach dem erfolgreichen Starten mit

http://<<docker-host>>:9090/api/quoting-service/

erreichbar und sollte uns, nach dem Aufruf, ein Zitat liefern.