Práctica 3: Computación en la Nube

Máster en Tecnologías Web, Computación en la Nube y Aplicaciones Móviles.

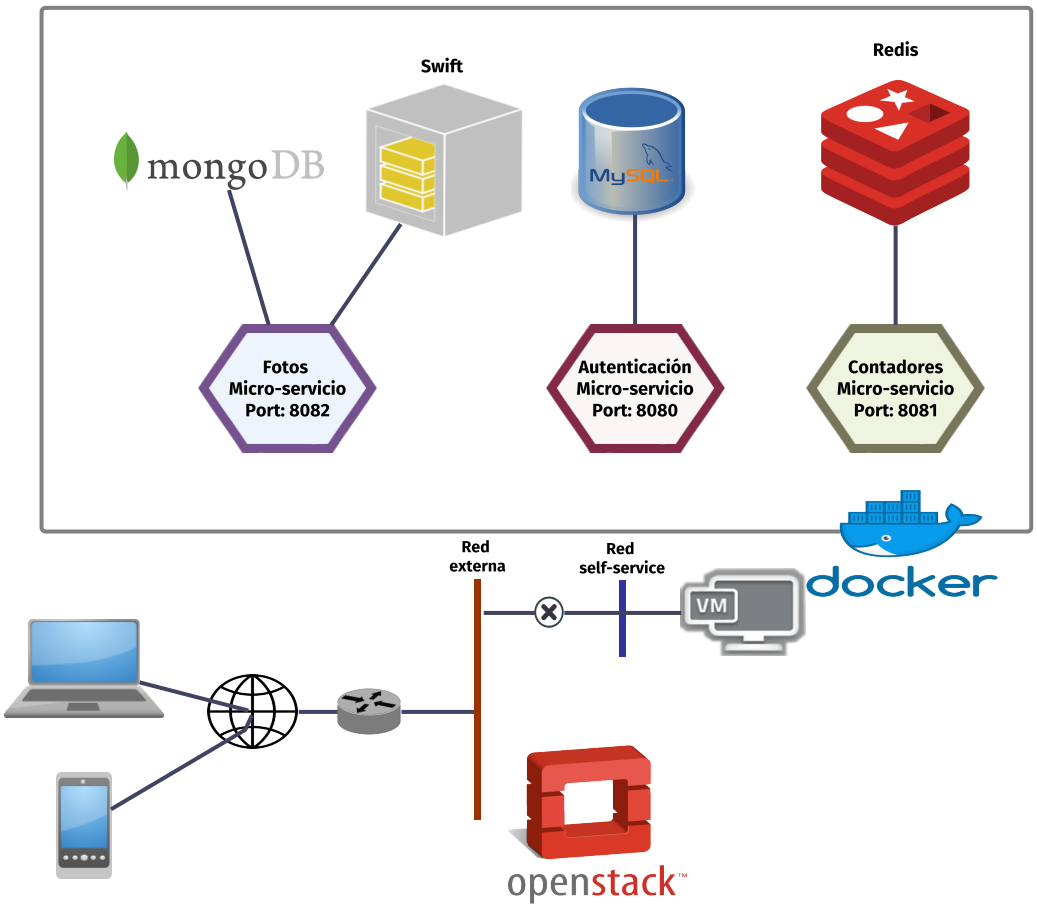
Juan Gutiérrez

Versión: 2018/06/01 - 11:54:37

1. Objetivos

- Realizar una aplicación con una arquitectura basada en micro-servicios.
- Encapsular los micro-servicios en contenedores
- Abastecer una máquina virtual en una infraestructura de computación en la nube
- Definir los servicios que componen la aplicación en un fichero YAML
- Realizar el despliegue de los servicios en la máquina virtual

La arquitectura se muestra en la siguiente figura:



2. Creación de una máquina virtual en OpenStack con docker-machine

Se debe crear una máquina virtual en OpenStack usando `docker-machine` con el driver `openstack`. Supondremos que hay una red creada, con una subred y la red está conectada mediante un *router* a la red externa. El *flavor* a utilizar será `m1.large` y la imagen `ubuntu-16-aufs` que tiene el usuario `ubuntu`.

Se debe realizar un ssh a la máquina y se debe iniciar un *swarm*. En este caso tendremos un clúster de una única máquina que es el *manager*.

El nombre de la imagen será: `docker-swarm-manager-gX`, donde `X` es el número de grupo.

3. Validación y obtención de información del JWT (se usará en el servicio de contadores y de fotos)

```
@Component
public class JWTChecker{
  @Value("${secret.token:provideoneatruntime}")
  private String secret;

  public String getToken(HttpServletRequest req) {
    String bearerToken = req.getHeader("Authorization");
    if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
      return bearerToken.substring(7, bearerToken.length());
    }
    return null;
  }

  public boolean validateToken(String token) {
    try {
      Jwts.parser().setSigningKey(secret.getBytes()).parseClaimsJws(token);
      return true;
    } catch (JwtException | IllegalArgumentException e) {
      throw new RuntimeException("Token not valid");
    }
  }

  public String getUsername(String token) {
    return Jwts.parser().setSigningKey(secret.getBytes()).parseClaimsJws(token).getBody
        ().getSubject();
  }
}
```

En los endpoints solicitaremos la inyección de una instancia de este tipo y realizaremos la siguiente secuencia:

```
Obtención del token
Validación del token
Obtención del usuario
```

Hay otras alternativas:

1. Hacer que la clase implemente a la interfaz `Filter` y al realizar la comprobación ponga en `request` un atributo con el nombre del usuario. Se puede encontrar un ejemplo en: https://aboullaite.me/spring-boot-token-authentication-using-jwt/

2. Integrar la validación de JWT en la autenticación y autorización de Spring Security. Se puede encontrar un ejemplo en:
   https://dzone.com/articles/implementing-jwt-authentication-on-spring-boot-api

4. Modificación e imagen del micro-servicio de contadores para validar JWT

Se debe modificar el micro-servicio de contadores para que se valide el JWT. Una vez probado y modificado se debe crear una imagen, se debe etiquetar como `twcammaster.uv.es/contadores_gX` y se debe subir al repositorio (usando `docker push`). Este servicio escuchará en el puerto 8081.

Este servicio usa el contenedor Redis, un ejemplo de instrucción para ejecutarlo es:

```
docker run -d --rm -p 6379:6379 --name counters redis
```

5. Desarrollo del micro-servicio de fotos

Se debe desarrollar un micro-servicio para la descarga de fotos (en formato JSON) y para la subida de una nueva foto. Las fotos se deben almacenar en Swift y la información sobre la foto se debe almacenar en un MongoDB. Tanto Swift como MongoDB serán contenedores. El contenedor Swift debe escuchar en el puerto 8083 ya que se deben poder recuperar las fotos subidas. Se debe validar y obtener el usuario del JWT.

El micro-servicio de fotos escuchará en el puerto 8082.

Ayuda para la implementación:

1. En el enlace http://joss.javaswift.org/authentication.html#basic se muestra cómo se puede usar las clases de la librería JOSS para realizar la conexión con Swift. El usuario para realizar la conexión es: `test:tester`, su contraseña es `testing` y la URL http://host:8083/auth/v1.0 donde se tendrá que substituir `host` por el nombre del servicio donde se ejecuta Swift. Toda esta información necesaria para la ejecución del micro-servicio se debe proporcionar como configuración.

2. En el enlace https://github.com/javaswift/joss se muestran ejemplos de como usar la clase `Container` (cuya API se puede consultar en http://javadocs.joss.javaswift.org/org/javaswift/joss/model/Container.html) para: hacer público el contenido del contenedor y para subir objetos (en este caso serán las fotos).

3. Se debe añadir la dependencia `spring-boot-starter-data-mongodb` en el fichero `pom.xml`.

4. Para realizar la persistencia en MongoDB se pueden crear la clase: `Foto` y la interfaz `FotoRepository`.
   La primera estará anotada con la notación `@Document` y tendrá como atributos `id`, `user`, `titulo`, `descripcion`, y `url`.
   La segunda extenderá a la interfaz `MongoRepository<Foto,String>` y añadirá un método para obtener todas las fotos de un usuario.

5. El controller, ofrecerá el *end-point* fotos y aceptará peticiones GET y POST. En la petición GET obtendremos el usuario del JWT (que puede haber sido añadido al ámbito de petición como un atributo) y se devolverá un JSON con todas las fotos del usuario. En la petición POST procesaremos una petición `multipart/form-data` en la que nos pasarán: el fichero (`file`), el título (`title`) y la descripción (`description`). Con esta información subiremos la imagen a Swift (y obtendremos la URL de acceso pública) y añadiremos la información a MongoDB.

Una vez probado el micro-servicio se debe crear una imagen, se debe etiquetar como `twcammaster.uv.es/fotos_gX` y se debe subir al repositorio.

6. Creación de una imagen con el micro-servicio de autenticación

Se debe crear una imagen con el micro-servicio de autenticación que se encuentra en Aula Virtual. La imagen se debe etiquetar como `twcammaster.uv.es/auth_gX` y se debe subir al repositorio. Este servicio escuchará en el puerto 8080.

Este servicio usa el contenedor MySQL, un ejemplo de instrucción para ejecutarlo es:

```
docker run --name mysql -e MYSQL_DATABASE=users -e MYSQL_USER=users -e MYSQL_PASSWORD=
    userspwd -e MYSQL_ALLOW_EMPTY_PASSWORD=no -p 3306:3306 mysql
```

7. Definición del fichero YAML con todos los servicios

Se debe realizar un fichero YAML que defina el *stack* de servicios que forman la aplicación:

- Micro-servicio de autenticación
- MySQL
- Micro-servicio de contadorers
- Redis
- Micro-servicio de fotos
- MongoDB
- Swift

Se debe usar una red overlay que de denomine: `network_gX`.
La configuración que necesitan los micro-servicios se debe proporcionar usando `config`.
Para ejecutar el stack:

1. Cargar las variables de entorno de la máquina creada
2. Ejecutar `docker stack`

8. Entrega y evaluación

1. Un documento en el que se aparezcan y se describan todas las instrucciones realizadas y el fichero con la definición del stack.
2. El código fuente de todos los micro-servicios.
3. Las imágenes de los contenedores en el repositorio.

La fecha de la evaluación de forma individual se deberá realizar la ejecución de la aplicación partiendo de la instancia ya creada y detenida.

Las acciones que se deberán realizar serán:

1. Reiniciar la máquina virtual

2. Ejecutar el stack
3. Realizar peticiones al servicio
4. Escalar el micro-servicio de contadores (de una a dos instancias)
5. Detener el stack
6. Detener la máquina virtual

Se **aconseja** traer las instrucciones en uno o varios scripts probados y comentados para no tener sorpresas.

También se podrán hacer preguntas sobre el código fuente y sobre los dos trabajos anteriores.

9. Enlaces

- https://github.com/javaswift/joss
- http://joss.javaswift.org/authentication.html#basic
- http://javadocs.joss.javaswift.org/org/javaswift/joss/model/Container.html
- https://spring.io/guides/gs/accessing-data-mongodb/
- https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html