FLUJO APROBADOR DE PEDIDOS

Este ejemplo demuestra el uso de los siguientes patrones de la arquitectura microservicios:

- Config server (Spring Cloud Config)
- Service discovery (Spring Cloud Netflix EurekaServer)
- Reverse Proxy, Filtering, Routing (Spring Cloud Netflix ZuulProxy)
- Security (ZuulProxy, Spring Security)
- LoadBalancing (ZuulProxy, Ribbon)
- Circuit Breaker (ZuulProxy, Hystrix)
- Event-driven:
  - ✓ Streams/ Messaging (Spring Cloud Streams)
  - ✓ Streams processor (Apache Kafka)
- Datasource aislado dentro de cada servicio
- Log centralizado (Elasticsearch, Logstash, Kibana)

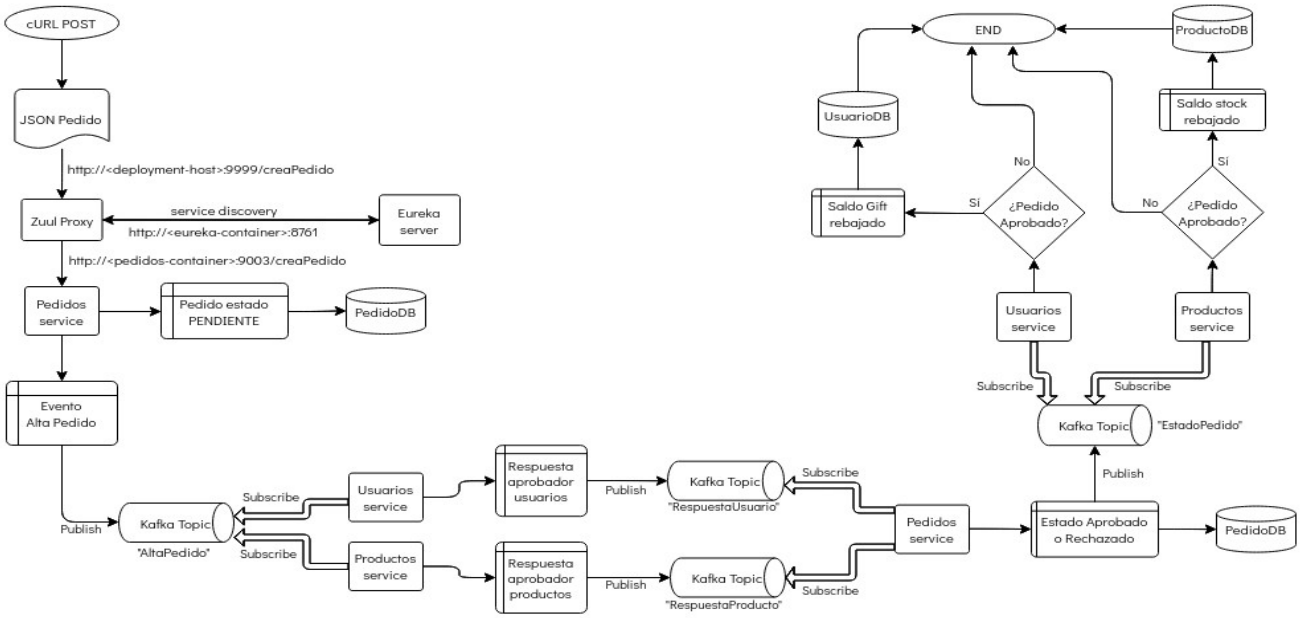
Despliegues:

- Despliegue en contenedores (docker, docker-compose)
- Despliegue en Amazon Web Services EC2
- Despliegue en Amazon Web Services ECS (Elastic Container Service)

Descripción del flujo:

- Estamos simulando un backend (muy simplificado) de un portal e-commerce, donde el usuario puede realizar un pedido de distintos productos y pagar parte del pedido (o el total) con su saldo de tarjeta de regalo o gift card.
- Un cliente http envía una petición POST al proxy o API gateway con el siguiente formato: *http://<deployment-host>:9999/creaPedido*
- El body del request contiene un documento JSON con los datos del pedido que se debe dar de alta
- El puerto 9999 es el que ha configurado ZuulProxy para recibir peticiones
- Mediante su configuración de rutas, Zuul asocia el path */creaPedido* con el servicio *pedidos*
- Acto seguido invoca el service discovery de EurekaServer para obtener la ubicación de una instancia del servicio *pedidos*
- Tras obtener la ubicación, se invoca por petición POST el servicio *pedidos*, operación */creaPedido*
- Este servicio procede a dar de alta el pedido en la base de datos (H2 in-memory database) con estado PENDIENTE, y publica un evento o mensaje en el Topic de Kafka *altaPedido*
- A este Topic de Kafka se subscriben los servicios de *usuarios* y *productos*, que proceden a revisar si existe saldo suficiente en la tarjeta Gift del usuario y en el nivel de stock de los productos solicitados
- Si alguno de ellos detectase saldo insuficiente, enviará una respuesta de rechazo al Topic de Kafka correspondiente: *respuestaUsuario* y *respuestaProducto*. En caso contrario envían una respuesta de aprobación
- A ambos Topics se subscribe el servicio de *pedidos*. El estado del pedido solamente se actualizará a Aprobado si el evento en ambos Topics contiene la respuesta de aprobación, en caso contrario el pedido quedará rechazado.
- El servicio *pedidos* envia un evento con el estado final del pedido al Topic *estadoPedido*, al cual se subscriben los servicios de *usuarios* y *productos*, que procederán a rebajar los saldos correspondientes solamente si el pedido está aprobado.

FLUJO APROBADOR DE PEDIDOS



Estructura:

```
drwxr-xr-x  5 rag rag 4096 ene 26 19:33 Build-deploy-test
drwxr-xr-x 10 rag rag 4096 ene 22 22:19 ConfigServer
drwxr-xr-x  9 rag rag 4096 ene 20 23:14 EurekaServer
drwxr-xr-x  9 rag rag 4096 ene 22 22:35 pedidos
drwxr-xr-x  9 rag rag 4096 ene 22 22:21 productos
drwxr-xr-x  9 rag rag 4096 ene 22 22:21 usuarios
drwxr-xr-x  9 rag rag 4096 ene 22 22:18 ZuulProxy
```

Notas sobre Datasource:

- Cada uno de los 3 servicios, pedidos, productos y usuarios, contiene un fichero data.sql dentro de src/main/resources. Dicho fichero se ejecuta en el arranque de cada servicio; Spring Boot se encarga de ello al tener el build.gradle del proyecto la dependencia con H2.
- En cada caso el script sql crea y alimenta con datos las tablas necesarias.
- El datasource está configurado como *in-memory database*.
- El acceso a datos es a través de Spring Data JPA.

Notas sobre Build:

- En cada directorio de proyecto existe un subdirectorio *dockerContext* con el Dockerfile para el build.
- Dentro del directorio Build-deploy-test:

```
-rwxr-xr-x 1 rag rag 1397 ene 26 19:14 build.sh
drwxr-xr-x 2 rag rag 4096 ene 26 21:12 despliegue-Amazon-EC2
drwxr-xr-x 2 rag rag 4096 ene 27 12:00 despliegue-Amazon-ECS
drwxr-xr-x 2 rag rag 4096 ene 23 11:32 kafka-standalone
```

- build.sh: recibe por parámetro la lista de servicios a compilar (solo pasamos los que hayamos modificado), y realiza un build (gradlew bootJar) de los proyectos que correspondan, después reconstruye las imágenes docker y las sube con push al docker hub.

Notas de despliegue Amazon AWS EC2:

- Dentro del directorio despliegue-Amazon-EC2:

```
-rwxr-xr-x 1 rag rag  388 ene 25 15:54 cURL-test-Amazon-EC2.sh
-rwxr-xr-x 1 rag rag  327 ene 26 21:12 despliega-compose.sh
-rw-r--r-- 1 rag rag 1847 ene 25 12:50 docker-compose.yml
-rwxr-xr-x 1 rag rag  200 ene 25 15:51 stop-containers.sh
```

- Levantamos una instancia EC2, idealmente de tipo t2.xlarge, con 4 vCPU y 16 GB de RAM. Instalamos java11, docker y docker-compose si no vienen ya instalados. Instalamos el ELK stack para el log centralizado: elasticsearch, logstash y kibana. Configuramos los puertos en estos tres componentes.
- Configuramos logstah: En logstash.conf cambiamos el path del log central a:

```
/home/ec2-user/pedidos-dockerized/log-central.log
```

- En el docker-compose.yml se ha configurado, para cada servicio, un volumen que mapea el directorio del container donde se guarda el log hacia el directorio ${HOME}/pedidos-dockerized. Es en este directorio de la instancia donde quedará el log centralizado.
- Configuramos kibana.yml: Para poder acceder a kibana con el navegador desde fuera de la instancia EC2, agregar la línea:

```
server.host: 0.0.0.0
```

- Arrancamos el ELK stack, cada comando en su respectivo directorio:

```
./elasticsearch
./logstash -f conf
./kibana
```

- Apache Kafka: docker-compose se encargará de configurar y levantar el servicio de Kafka. Para ello hace un pull de las imagenes de Confluent Inc., tanto del servicio Zookeeper como el de Kafka propiamente.
- Una vez levantada la instancia, ejecutamos:

```
./despliega-compose.sh <ec2-hostname>
```

  Ejemplo de *ec2-hostname*: ec2-18-203-234-80.eu-west-1.compute.amazonaws.com
- Se copiará el docker-compose.yml con scp, y luego se conectará por ssh para levantar los servicios del backend. Esperar un minuto a que levanten completamente todos los servicios, y ejecutamos:

```
./cURL-test-Amazon-EC2.sh <ec2-hostname>
```

  Con esto realizamos una prueba con un JSON de ejemplo.
- Conectamos a la base de datos H2 dentro de *pedidos* con el navegador, y vemos que el estado del pedido ha cambiado de PENDIENTE a APROBADO:

```
http://<ec2-hostname>:9003/h2-console
select * from pedido;
```

- Para revisar los logs de la aplicación conectamos con kibana en el navegador, creamos un index pattern “logstash-*” y vamos a “discover”:

```
http://<ec2-hostname>:5601
```

- En los logs se puede ver que en los servicios de *usuarios* y *productos* se han rebajado los saldos correspondientes ya que el pedido quedó con estado aprobado.
- Si ejecutamos el test una segunda vez, veremos que el pedido queda RECHAZADO (no hay suficiente stock de uno de los productos) y que en los logs ya no aparecen las rebajas de saldo.

Notas de despliegue Amazon AWS ECS:

- Amazon dispone de su propio orquestador de contenedores: AWS Elastic Container Service (ECS). La ventaja que aporta es que no tenemos que levantar manualmente una instancia ni los recursos necesarios. Solamente tenemos que tener un compose.yml y lanzar los comandos (ecs-cli) para que Amazon levante un cluster con todos los recursos necesarios.
- Dentro del directorio despliegue-Amazon-ECS:

```
-rw-r--r-- 1 rag rag 1587 ene 25 12:07 aws-compose.yml
-rwxr-xr-x 1 rag rag   55 ene 27 12:00 bajar-cluster-ECS.sh
-rwxr-xr-x 1 rag rag  389 ene 25 15:56 cURL-test-Amazon-ECS.sh
-rw-r--r-- 1 rag rag  642 ene 25 09:41 ecs-params.yml
-rwxr-xr-x 1 rag rag  851 ene 27 11:59 levantar-cluster-ECS.sh
```

- Levantamos el cluster ECS ejecutando (tarda unos minutos):

```
./levantar-cluster-ECS.sh
```

- Una vez que termina, debe listar los contenedores en estado RUNNING. Esperar un minuto más a que se levanten todos los servicios.
- El cluster levanta una instancia EC2, que tiene todo lo necesario preinstalado excepto el stack ELK. Deberemos instalarlo y configurarlo en la máquina igual que antes (despliegue EC2)
- Realizamos el mismo test que antes:

```
./cURL-test-Amazon-ECS.sh <ec2-hostname>
```

- Y podemos revisar los resultados en base de datos y en logs de la misma forma que para el despliegue EC2.