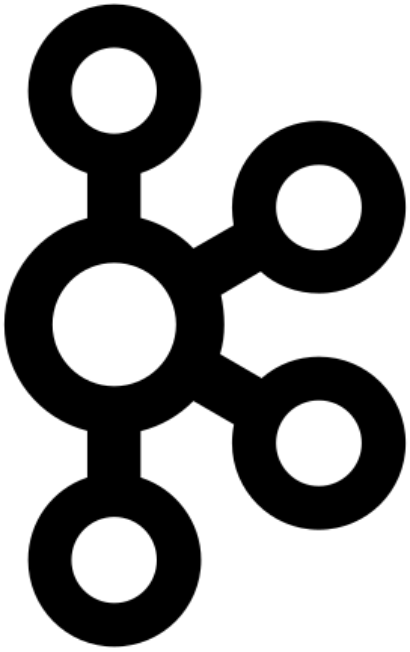
# Eventos y colas

Sprint 2 Lab 5



Felipe Izquierdo Romero

# Índice

# 1. Descripción del problema

Documentar la gestión de inventario de un ecommerce de venta de zapatos definiendo sus eventos siguiendo la plantilla EDA. De manera opcional implementar el sistema con Docker.

# 2. Desarrollo

Este es el esquema de la empresa:

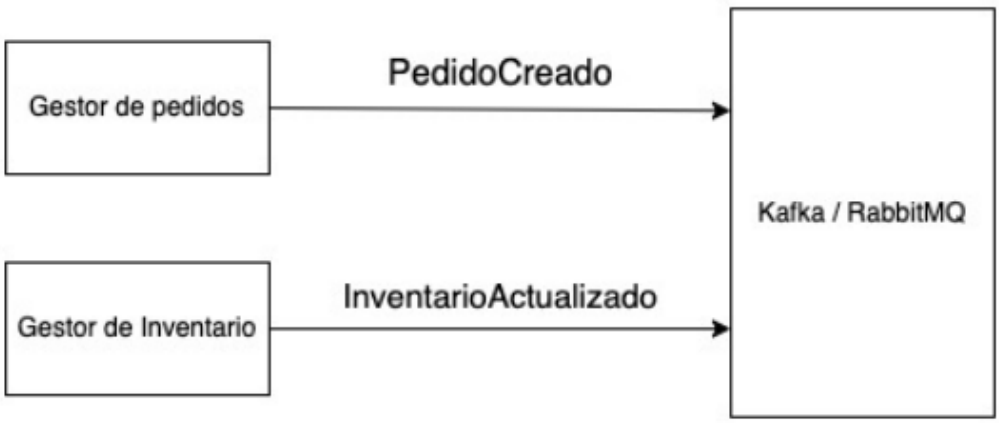



*Ilustración 1- Sistema de microservicios ecommerce*

Los dos microservicios contienen dos métodos que se comunicaran entre ellos a través de Kafka.

## 2.1. Descripción de eventos

### Pedido creado

Se encarga de generar el pedido con los datos de cada producto seleccionados por el cliente.

### Detalles del evento

**ID del Evento:** será un UID generado automáticamente por cada pedido

**Tipo de Evento:** evento de dominio

**Origen del Evento:** gestor de pedidos

**Consumidores del evento:** órquestador

**Datos del Evento:** el evento contendrá el identificador del producto, la cantidad y el precio de cada producto.

### Especificaciones Técnicas

**Formato:** JSON

**Esquema del mensaje:**

{

“id”:1,

”cantidad”:1,

”precio”:1.0

}

**id:**

Tipo de dato: Integer

Descripción: Identificador único del evento o entidad.

**cantidad:**

Tipo de dato: Integer

Descripción: La cantidad del producto o ítem en cuestión.

**precio:**

Tipo de dato: Float

Descripción: El precio unitario del producto o ítem.

**Validaciones:**

Campo id:

Debe ser un entero positivo.

Validación:

id > 0

Campo cantidad:

Debe ser un entero no negativo.

Validación:

cantidad >= 0

Campo precio:

Debe ser un número flotante positivo o cero.

Validación:

precio >= 0.0

**Versionado:** v1.0.

## Reglas de Negocio Asociadas

Para que el pedido se realice la cantidad de stock debe de ser mayor o igual a la del pedido.

Tras realizar un pedido se actualizará el stock automáticamente.

## Casos de uso

**Producción del evento:** cada vez que el usuario pulse el botón de tramitar pedido se ejecutara este evento y esperara a finalizar con o sin éxito el proceso de transacción.

**Consumo del evento:** el orquestador será el encargado de recibir el evento para validar la transacción.

## Seguridad y privacidad

**Consideraciones de seguridad:** los UIDs de cada pedido serán generados con la librería uuid de Python que proporcionará identificadores únicos e inmutables.

**Consideraciones de privacidad:** este evento no será el encargado de tramitar datos del usuario para salvaguardar la integridad de los datos.

## Historial de cambios

**Registro de versiones:** v1.0. (2024-05-26): creación del evento.

## Inventario actualizado

Se encarga de comprobar el stock/inventario y actualizarlo si hay suficiente.

## Detalles del evento

**ID del Evento:** será un UID generado automáticamente por cada evento generado

**Tipo de Evento:** evento de log

**Origen del Evento:** gestor de Inventario

**Consumidores del evento:** órquestador

**Datos del Evento:** el evento contendrá un identificador por cada evento generado, un código de transacción y un mensaje de transacción.

## Especificaciones Técnicas

**Formato:** JSON

**Esquema del mensaje:**

{

“id”: a81e9b2a-e4d7-11ed-a6b2-0242ac120002,

”código_transaccion”:0,

”mensaje”:”Realizada con exito”

}

**id:**

Tipo de dato: String

Descripción: Identificador único del evento.

**código:**

Tipo de dato: Integer

Descripción: código que clasifica la descripción de la transacción.

**mensaje:**

Tipo de dato: String

Descripción: Describe en una frase el estado de la transacción.

**Validaciones:**

Campo codigo:

Debe ser un entero no negativo de rango de 0 a 2.

Validación:

codigo >= 0, código<=2

**Versionado:** v1.0.

## Reglas de Negocio Asociadas

Para que el stock se actualice la cantidad de stock debe de ser mayor o igual a la del pedido generado.

Tras la comprobación y la actualización de inventario si es posible se informará con un mensaje de éxito o error.

## Casos de uso

**Producción del evento:** cada vez que el usuario genere un pedido este microservicio comprobará el stock y generara un mensaje.

**Consumo del evento:** el orquestador será el encargado de recibir el evento para validar la transacción.

## Seguridad y privacidad

**Consideraciones de seguridad:** los UIDs de cada pedido serán generados con la librería uuid de Python que proporcionará identificadores únicos e inmutables.

**Consideraciones de privacidad:** este evento no será el encargado de tramitar datos del usuario para salvaguardar la integridad de los datos.

### Historial de cambios

**Registro de versiones:** v1.0. (2024-05-26): creación del evento.

## 3. Opcional

De manera opcional he montado el sistema de microservicios que se comunican con Kafka para gestionar los eventos, pero como en el anterior laboratorio he tenido algunos problemas al usar Docker y los microservicios de Kafka y zookeper en Docker me funcionan correctamente pero al iniciar los servicios gestor_inventario, gestor_pedidos y orquestador usando Python con flask no encuentran al bróker de Kafka y son incapaces de escuchar ni enviar mensajes.

De manera local si funciona, pero no he llegado a completar la parte opcional entera puesto que no he podido replicar la documentación que he descrito con anterioridad al 100%.

Los inconvenientes de no poder haber replicado todo son debido al tiempo y a los problemas que he encontrado durante el aprendizaje de esta tecnología.

Aquí muestro el funcionamiento del sistema de microservicios resumiendo la actividad opcional.

Video del funcionamiento de los microservicios