version: '3.8'

Define la versión de la sintaxis del archivo docker-compose que estás utilizando. 3.8 es una versión moderna y estable.

services:

Es el bloque principal que agrupa la definición de todos los contenedores (llamados "servicios") que Docker Compose va a crear y gestionar.

kafka:

Define el primer servicio, al que llamaremos **kafka**. Este nombre es importante, ya que se convierte en el **hostname** que los otros contenedores usarán para conectarse a este (kafka:9092).

image: confluentinc/cp-kafka:latest

Especifica la imagen de Docker que se usará para construir este contenedor. En este caso, es la imagen oficial de Kafka de Confluent. (Nota: Es mejor usar una versión específica como 7.6.1 en lugar de latest para evitar que futuras actualizaciones rompan tu configuración).

environment:

Define una lista de variables de entorno que se inyectarán dentro del contenedor de Kafka. Así es como configuramos el software de Kafka.

CLUSTER\_ID: 'ULPlA8IRSzaJ3fEL32Ss3g'

Un ID único para tu clúster de Kafka. Es **obligatorio** para el modo KRaft (sin ZooKeeper).

KAFKA\_NODE\_ID: 1

Asigna un ID único a este nodo/bróker de Kafka dentro del clúster. Como es un solo nodo, le asignamos el 1.

KAFKA\_PROCESS\_ROLES: 'broker,controller'

Define los roles de este nodo. Al poner broker,controller, le decimos que actúe simultáneamente como **bróker** (manejando los datos) y como **controlador** (gestionando el clúster). Esto elimina la necesidad de ZooKeeper.

KAFKA\_CONTROLLER\_QUORUM\_VOTERS: '1@kafka:9093'

Indica quiénes son los "votantes" que gestionan el clúster. En este caso, es solo el nodo 1 (él mismo), que puede ser contactado en el hostname kafka en el puerto 9093.

# --- Configuración de Red para Comunicación Interna ---

KAFKA\_LISTENERS: 'INTERNAL://0.0.0.0:9092,CONTROLLER://kafka:9093'

Le dice al proceso de Kafka en qué puertos debe "escuchar" *dentro* de su propio contenedor.

* INTERNAL://0.0.0.0:9092: Define un canal de comunicación llamado INTERNAL para los datos, que escucha en todas las interfaces de red (0.0.0.0) en el puerto 9092.
* CONTROLLER://kafka:9093: Define un canal CONTROLLER para la gestión del clúster en el puerto 9093.

KAFKA\_ADVERTISED\_LISTENERS: 'INTERNAL://kafka:9092'

Esta es la dirección que Kafka le **anuncia** a los clientes (como tus aplicaciones de Node.js). Les dice: "Si quieres conectarte a mí usando el canal INTERNAL, búscame en el hostname kafka en el puerto 9092".

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: 'CONTROLLER:PLAINTEXT,INTERNAL:PLAINTEXT'

Mapea los nombres de los canales (CONTROLLER, INTERNAL) al protocolo de seguridad que deben usar. PLAINTEXT significa sin encriptación (ideal para desarrollo interno).

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: 'INTERNAL'

Define qué canal deben usar los brókeres de Kafka para hablar entre sí. (En este caso, solo hay uno).

KAFKA\_CONTROLLER\_LISTENER\_NAMES: 'CONTROLLER'

Le dice al nodo qué canal debe usar para sus tareas de controller.

# --- Configuraciones para un solo nodo ---

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

Para un clúster de un solo nodo, el "factor de replicación" (cuántas copias de los datos) debe ser 1.

KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 3000

Una configuración de optimización. Le dice a Kafka que espere 3 segundos (3000ms) cuando un consumidor se une a un grupo antes de reasignar las particiones. Esto ayuda a evitar errores de conexión al inicio.

# --- Capa 2: Crear los topics al iniciar ---

KAFKA\_CREATE\_TOPICS: "weather-data:1:1,weather-alerts:1:1"

Una instrucción muy útil. Le ordena a Kafka que cree automáticamente estos topics al arrancar, evitando errores de "topic no encontrado".

* weather-data:1:1 -> (Nombre:Particiones:Replicación)
* weather-alerts:1:1 -> (Nombre:Particiones:Replicación)

healthcheck:

Define una prueba de salud que Docker ejecutará para saber si el contenedor está realmente funcionando bien.

test: ["CMD-SHELL", "kafka-topics --bootstrap-server localhost:9092 --list"]

El comando que se ejecutará *dentro* del contenedor. Intenta listar los topics. Si el comando tiene éxito, el contenedor se marca como "saludable".

interval: 10s

Ejecuta la prueba de salud cada 10 segundos.

timeout: 5s

Si la prueba tarda más de 5 segundos, se considera fallida.

retries: 5

Si la prueba falla, la reintenta 5 veces antes de marcar el contenedor como "no saludable".

start\_period: 20s

No empieces a ejecutar la prueba de salud hasta 20 segundos después de que el contenedor haya arrancado, dándole tiempo a Kafka para iniciarse.

producer:

Define el segundo servicio, llamado **producer**.

build: .

Le dice a Docker Compose que no descargue una imagen, sino que **construya** una imagen usando el archivo Dockerfile que se encuentra en el mismo directorio (representado por .).

command: node producer.js

El comando que se ejecutará dentro del contenedor una vez que se inicie. En este caso, inicia tu script del productor.

depends\_on:

kafka:

condition: service\_healthy

Esto es crucial. Le dice al servicio producer que **no debe iniciarse** hasta que el servicio kafka haya pasado su healthcheck (es decir, esté en estado service\_healthy).

dashboard-backend:

Define el tercer servicio, llamado **dashboard-backend**.

build: .

Construye la imagen desde el Dockerfile local.

command: node dashboard-backend.js

Ejecuta el script del backend del panel principal.

ports:

- "8080:8080"

Mapea un puerto. Expone el puerto 8080 del contenedor al puerto 8080 de tu máquina local (host). Esto es lo que permite que tu index.html se conecte a ws://localhost:8080.

depends\_on:

kafka:

condition: service\_healthy

Igual que el productor, espera a que Kafka esté 100% saludable antes de iniciarse.

alerts-processor:

Define el cuarto servicio, **alerts-processor**.

build: .

Construye la imagen desde el Dockerfile local.

command: node alerts-processor.js

Ejecuta el script del procesador de alertas.

depends\_on:

kafka:

condition: service\_healthy

Espera a que Kafka esté saludable. (Este servicio no necesita ports porque es un trabajador interno que no se conecta con el exterior).

alerts-backend:

Define el quinto y último servicio, **alerts-backend**.

build: .

Construye la imagen desde el Dockerfile local.

command: node alerts-backend.js

Ejecuta el script del backend de alertas.

ports:

- "8081:8081"

Expone el puerto 8081 del contenedor al puerto 8081 de tu máquina local, permitiendo que alerts.html se conecte a ws://localhost:8081.

depends\_on:

kafka:

condition: service\_healthy

Espera a que Kafka esté saludable antes de iniciarse.