UIMP
Universidad Internacional Menéndez Pelayo

AEPIA
Spanish Association for Artificial Intelligence

Hortonworks Partner
ACADEMIC TRAINING

Pautas generales para las prácticas:

1. Las prácticas **deberán acompañarse de un breve documento README** que detalle cómo ejecutar la práctica y cómo configurar los parámetros si existen, así como cualquier otra particularidad que sea necesario conocer. Adicionalmente, si se desea explicar algún detalle de implementación, podrá adjuntarse opcionalmente un documento de diseño en el que se presente y justifique el diseño elegido.

2. El código debe seguir las **buenas prácticas** habituales: variables con nombres significativos, comentarios, eficiencia, etc.

Ejercicio de evaluación: Trident

La tienda online todocaro.com quiere construir un sistema de control en tiempo real de sus ventas. El equipo de dirección de ventas ha detectado irregularidades en las ventas y cancelaciones a lo largo del tiempo y desearían poder consultar en tiempo real los **clientes en cada país** con el mayor volumen (en $) **comprado y cancelado**. Esta consulta permitirá tomar medidas a tiempo en cuanto se presenten irregularidades.

El equipo de programación está especializado en Java, por lo que se decide utilizar Trident como herramienta de trabajo.

Se le pide que diseñe tanto la arquitectura como el software para cumplir con este requisito, justificando las decisiones.

Notas

Ficheros de datos: https://www.dropbox.com/s/5soif14s2muln3q/ficheros-practica-streaming.zip?dl=1

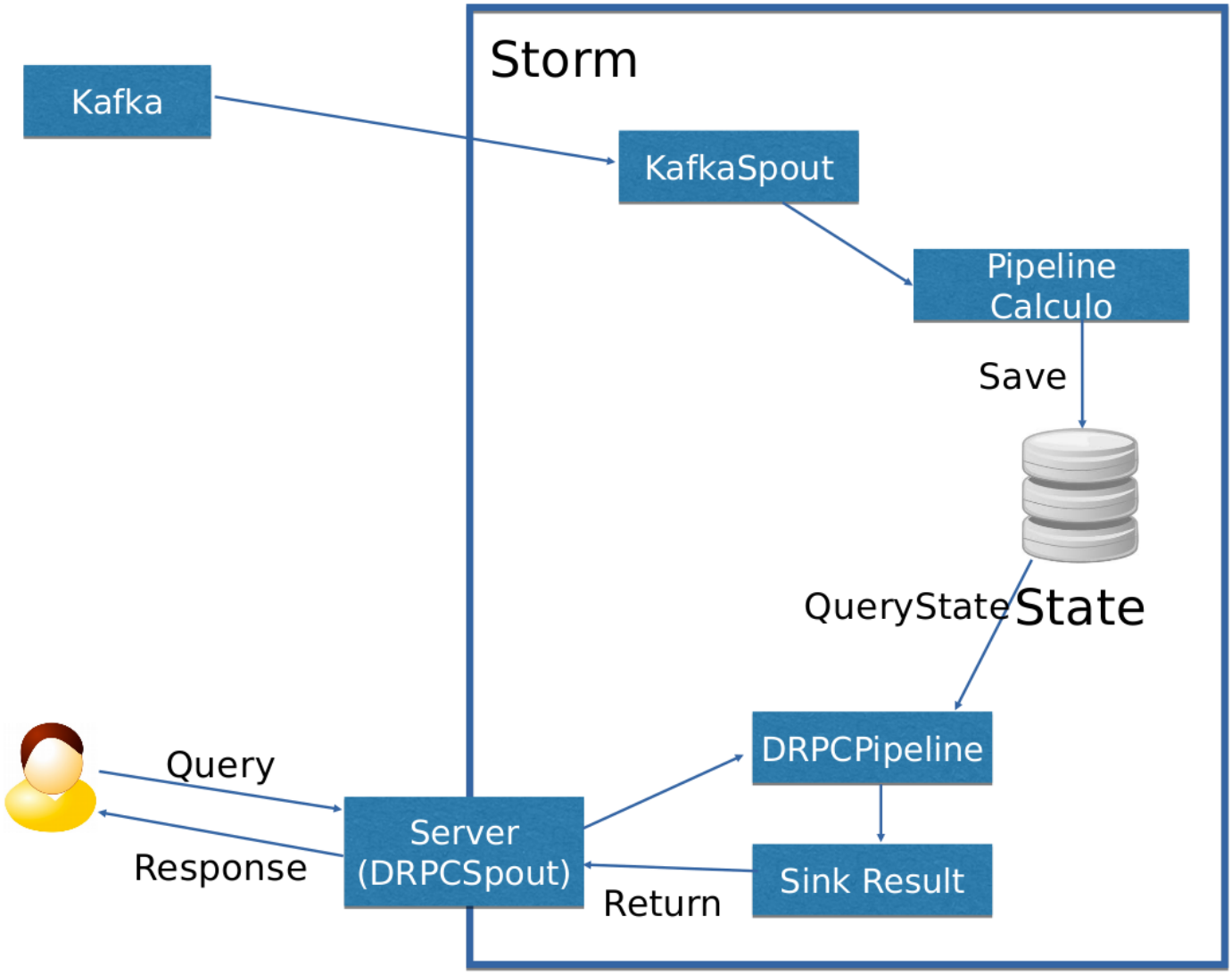
Consejos

- En el ejemplo disponible en https://github.com/eshioji/trident-tutorial/blob/master/src/main/java/tutorial/storm/trident/example/TopHashtagByCountry.java puede ver cómo se utiliza una arquitectura similar.
- En el caso de que utilice Kafka como entrada, utilice el emisor de compras que encontrará en la práctica de Spark Streaming para simular el stream de compras.

Requisitos

- La aplicación debe recibir una cadena de texto que contenga uno o varios nombres de países, que será de los que se muestre el resumen.
- Cada segundo, para cada país de los solicitados, se mostrarán los 5 clientes con un mayor montante en $ de compras y los 5 clientes con un mayor montante en $ de cancelaciones.
- En el caso de utilizar Kafka, deberá crearse un topic con el nombre “compras-trident” al cual emitirá el simulador de stream de compras y desde el que leerá la aplicación.

Propuesta de arquitectura a seguir:



```java
public static StormTopology buildTopology(LocalDRPC drpc) {
    //CREAMOS EL SPOUT PARA LEER DE KAFKA
    TridentKafkaConfig spoutConfig = new TridentKafkaConfig(new ZkHosts("namenode:2181"), "sentences");
    spoutConfig.scheme = new SchemeAsMultiScheme(new StringScheme());
    spoutConfig.forceFromStart = true;

    TransactionalTridentKafkaSpout kafkaSpout = new TransactionalTridentKafkaSpout(spoutConfig);

    //AHORA CREAMOS LA TOPOLOGÍA
    TridentTopology topology = new TridentTopology();

    //Creamos el contador de palabras, que recibirá las frases desde Kafka, las transformará en
    //palabras y hará el recuento de cada palabra y lo almacenará en el proveedor de estado.
    TridentState wordCounts = topology.newStream("spout", kafkaSpout).parallelismHint(16)
            .each(new Fields("str"), new Split(), new Fields("word"))
            .groupBy(new Fields("word"))
            .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
            .parallelismHint(16);

    //Por último creamos el stream que se "conecta" al "proceso" que está realizando el recuento
    //y consulta el estado. Paso a paso:
    //1. Creamos un nuevo stream de tipo DRPC. Le damos un nombre a la función y le pasamos el
    //objeto drpc.
    topology.newDRPCStream("word-count-func", drpc)
    //2. Las tuplas que tenemos contienen una sola cadena ("args") con el parámetros que hayamos
    //pasado a la llamada a execute del DRPC. Las separamos en palabras y le damos nombre al
    //campo
    .each(new Fields("args"), new Split(), new Fields("filter-word"))
    //3. Agrupamos por palabras.
    .groupBy(new Fields("filter-word"))
    //4. Consultamos el estado del "proceso" contador que declaramos antes indicando:
    //    - la variable TridentState a consultar
    //    - el campo que se utilizará como entrada de la función siguiente
    //    - la función de consulta. MapGet hace una consulta tipo diccionario: busca el
    //          valor asociado a la clave que recibe como entrada (en este caso el contenido del
    //          campo "filter-word".
    //    - cómo queremos llamar al nuevo campo de salida.
    .stateQuery(wordCounts, new Fields("filter-word"), new MapGet(), new Fields("count"))
    //5. Filtramos los valores que no estén en el recuento.
    .each(new Fields("count"), new FilterNull())
    //6. Sumamos todas las tuplas para tener un único valor. Si comentamos esta línea veremos la
    //tupla entera.
    .aggregate(new Fields("count"), new Sum(), new Fields("sum"));

    return topology.build();
}

public static void main(String[] args) throws Exception {

    Config conf = new Config();
    conf.setMaxSpoutPending(20);
    LocalDRPC drpc = new LocalDRPC();
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("trident-kafka-word-count", conf, buildTopology(drpc));
    for (int i = 0; i < 100; i++) {
        //En este bucle hacemos llamadas a la función que creamos con el DRPC al crear la topología.
        System.out.println("Good: " + drpc.execute("word-count-func", "good"));
        System.out.println("Happy: " + drpc.execute("word-count-func", "happy"));
        System.out.println("Other: " + drpc.execute("word-count-func", "are a"));
        Thread.sleep(1000);
    }
    drpc.shutdown();
    cluster.shutdown();
}
```