Message Queue CKafka

Best Practices

Product Introduction

Contents

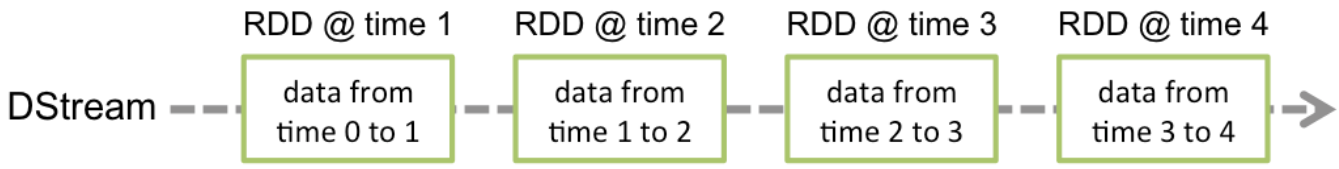
Best Practices
Spark Streaming with CKafka

Last updated：2018-02-08 11:18:53

Introduction to Spark Streaming

Spark Streaming is an extension of Spark Core, which is used for high-throughput and fault-tolerant processing of the continuous data. The external inputs now supported include Kafka, Flume, HDFS/S3, Kinesis, Twitter, and TCP socket.

Spark Streaming abstracts the continuous data into a DStream (Discretized Streams), while the DStream consists of a range of continuous RDDs (Resilient Distributed Datasets), and each RDD is the data generated within a certain time interval. The processing of DStream using functions is actually the processing of these RDDs.



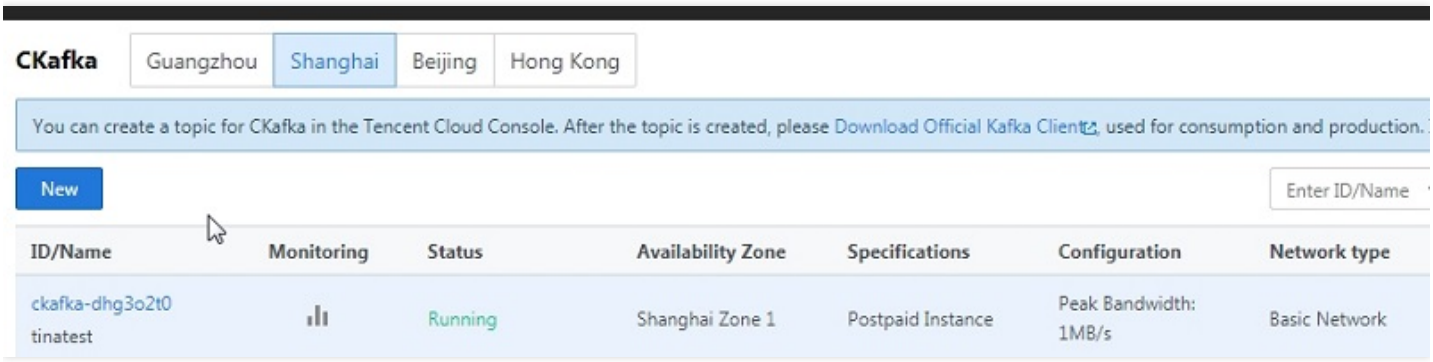
Now, Spark Streaming's support for Kafka as data input is divided into stable version and experimental version:

Kafka Version	spark-streaming-kafka-0.8	spark-streaming-kafka-0.10
Broker Version	0.8.2.1 or higher	0.10.0 or higher
Api Stability	Stable	Experimental
Language Support	Scala, Java, Python	Scala, Java
Receiver DStream	Yes	No
Direct DStream	Yes	Yes
SSL/TLS Support	No	Yes
Offset Commit Api	No	Yes
Dynamic Topic Subscription	No	Yes

Now, the following versions of CKafka are supported: 0.9.0.x, 0.10.0.x, 0.10.1.x, and 0.10.2.x. The Kafka dependency of version 0.10.2.1 is used in this practice scenario.

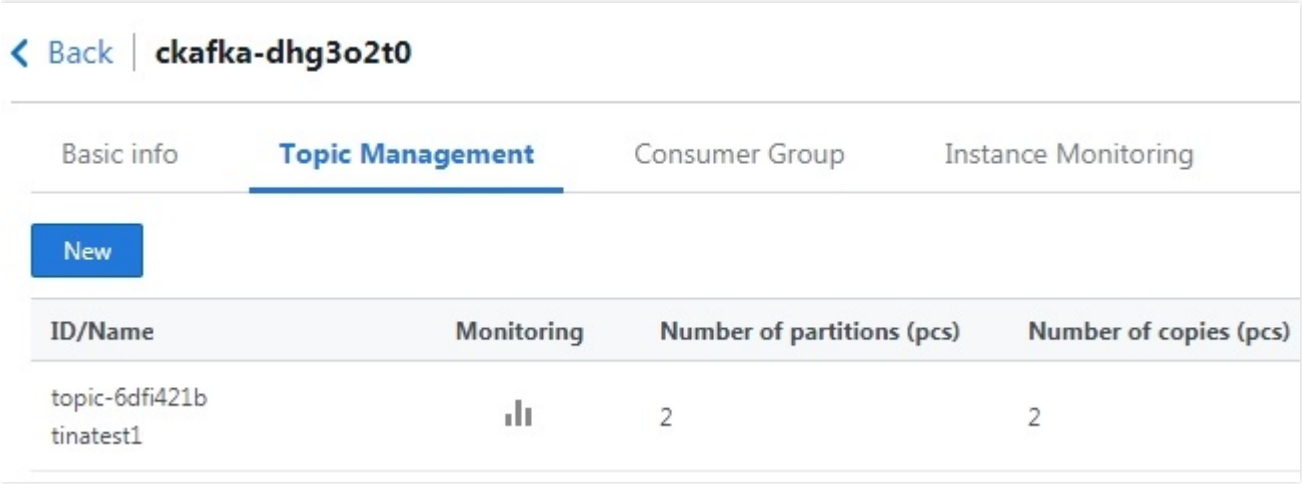
Connecting Spark Streaming to CKafka

Applying for Ckafka Instance



Confirm whether the network type matches the network that is now used

Creating Topic



We create a topic named spark_test here, and take this topic as an example to show how to produce and consume messages
[Private IP and Port] is the bootstrap-server to be used for production and consumption

CVM Environment

Centos 6.8 System

Package	Version
sbt	0.13.16
hadoop	2.7.3
spark	2.1.0
protobuf	2.5.0
ssh	CentOS installed by default
Java	1.8

Production in Ckafka

Now, the following versions of CKafka are supported: 0.9.0.x, 0.10.0.x, 0.10.1.x, and 0.10.2.x.
The kafka dependency of version 0.10.2.1 is used here.
build.sbt

```
name := "Producer Example"
version := "1.0"
```

```
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % "0.10.2.1"
```

producer_example.scala

```
import java.util.Properties
import org.apache.kafka.clients.producer._

object ProducerExample extends App {
   val  props = new Properties()
   props.put("bootstrap.servers", "172.16.16.12:9092") //Private IP and port in the instance information

   props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
   props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

   val producer = new KafkaProducer[String, String](props)
   val TOPIC="test"  //Specify the topic to be produced
   for(i<- 1 to 50){
         val record = new ProducerRecord(TOPIC, "key", s"hello $i") //Produce a message with key being
         producer.send(record)
   }
   val record = new ProducerRecord(TOPIC, "key", "the end "+new java.util.Date)
   producer.send(record)
   producer.close() //To be disconnected at last.
}
```

For more usages of ProducerRecord, please see
https://kafka.apache.org/0100/javadoc/org/apache/kafka/clients/producer/ProducerRecord.html

Consuming from CKafka

DirectStream

Add dependency to build.sbt

```
name := "Consumer Example"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-10" % "2.1.0"
```

DirectStream_example.scala

```
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.kafka.common.TopicPartition
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010.OffsetRange
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import collection.JavaConversions._
import Array._

object Kafka {
  def main(args: Array[String]) {
    val kafkaParams = Map[String, Object](
      "bootstrap.servers" -> "172.16.16.12:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[StringDeserializer],
      "group.id" -> "spark_stream_test1",
      "auto.offset.reset" -> "earliest",
      "enable.auto.commit" -> "false"
    )

    val sparkConf = new SparkConf()
    sparkConf.setMaster("local")
    sparkConf.setAppName("Kafka")
    val ssc = new StreamingContext(sparkConf, Seconds(5))
    val topics = Array("spark_test")

    val offsets : Map[TopicPartition, Long] = Map()

    for (i <- 0 until 3){
      val tp = new TopicPartition("spark_test", i)
      offsets.updated(tp , 0L)
    }
    val stream = KafkaUtils.createDirectStream[String, String](
      ssc,
      PreferConsistent,
      Subscribe[String, String](topics, kafkaParams)
    )
    println("directStream")
    stream.foreachRDD{ rdd=>
      //Output the obtained messages
```

```
            rdd.foreach{iter =>
                val i = iter.value
                println(s"${i}")
            }
            //Obtain the offset
            val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
            rdd.foreachPartition { iter =>
                val o: OffsetRange = offsetRanges(TaskContext.get.partitionId)
                println(s"${o.topic} ${o.partition} ${o.fromOffset} ${o.untilOffset}")
            }
        }

        // Start the computation
        ssc.start()
        ssc.awaitTermination()
    }
}
```

RDD

The configuration of `build.sbt` is the same as above `RDD_example`

```
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010.OffsetRange
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import collection.JavaConversions._
import Array._


object Kafka {
    def main(args: Array[String]) {
        val kafkaParams = Map[String, Object](
            "bootstrap.servers" -> "172.16.16.12:9092",
            "key.deserializer" -> classOf[StringDeserializer],
            "value.deserializer" -> classOf[StringDeserializer],
            "group.id" -> "spark_stream",
            "auto.offset.reset" -> "earliest",
```

```
            "enable.auto.commit" -> (false: java.lang.Boolean)
        )
        val sc = new SparkContext("local", "Kafka", new SparkConf())
        val java_kafkaParams : java.util.Map[String, Object] = kafkaParams
        //Pull the messages in the corresponding offset range to partition in order. The request is blocked i
        val offsetRanges = Array[OffsetRange](
            OffsetRange("spark_test", 0, 0, 5),
            OffsetRange("spark_test", 1, 0, 5),
            OffsetRange("spark_test", 2, 0, 5)
        )
        val range = KafkaUtils.createRDD[String, String](
            sc,
            java_kafkaParams,
            offsetRanges,
            PreferConsistent
        )
        range.foreach(rdd=>println(rdd.value))
        sc.stop()
    }
}
```

For more usages of `kafkaParams`, please see
http://kafka.apache.org/documentation.html#newconsumerconfigs

Configuration of Environment

Installing "sbt"

1. Download sbt package from sbt's official website

2. Create a sbt_run.sh script in the sbt directory after decompression, and add the execution permission
   The content of script is as follows:

```
#!/bin/bash
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256
java $SBT_OPTS -jar `dirname $0`/bin/sbt-launch.jar "$@"
```

```
chmod u+x ./sbt_run.sh
```

3. Execute

```
./sbt-run.sh sbt-version
```

The display of sbt version indicates a normal operation

Installing "protobuf"

1. Download an appropriate version of protobuf
2. Decompress to enter the directory

```
./configure
make && make install
```

You need to pre-install gcc-g++, and the root permission may be required during installation

3. Log in again, and enter the following in the command line

```
protoc --version
```

4. The display of protobuf version indicates a normal operation

Installing "hadoop"

1. Go to hadoop's official website to download the required version
2. Add hadoop users

```
useradd -m hadoop -s /bin/bash
```

3. Add admin permission

```
visudo
```

4. Add the following in a new line under `root ALL=(ALL) ALL`
   `hadoop ALL=(ALL) ALL`
   Save and exit
5. Use hadoop to operate

```
su hadoop
```

6. Configure ssh for password-less login

```
cd ~/.ssh/                     # If the directory does not exists, execute ssh localhost first
ssh-keygen -t rsa              # Just press Enter when prompted
cat id_rsa.pub >> authorized_keys  # Add authorization
chmod 600 ./authorized_keys    # Modify file permission
```

7. Install java

```
sudo yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

8. Configure ${JAVA_HOME}

```
vim /etc/profile
```

Add the following at the end of text

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el6_8.x86_64/jre
export PATH=$PATH:$JAVA_HOME
```

Modify the corresponding path according to the installation

9. Decompress hadoop, and go to the directory

```
./bin/hadoop version
```

The display of version information indicates a normal operation

0. Configure a pseudo-distribution stand-alone cluster (you can build different types of clusters as needed)

```
vim /etc/profile
```

Add the following at the end of text

```
export HADOOP_HOME=/usr/local/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

Modify the corresponding path according to the installation

1. Modify `/etc/hadoop/core-site.xml`

```
<configuration>
<property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>Abase for other temporary directories.</description>
</property>
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

2. Modify `/etc/hadoop/hdfs-site.xml`

```
<configuration>
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
```

```
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
</property>
</configuration>
```

3. Change JAVA_HOME in `/etc/hadoop/hadoop-env.sh` to java path

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el6_8.x86_64/jre
```

4. Format NameNode

```
./bin/hdfs namenode -format
```

The appearance of 'Exitting with status 0' indicates a successful installation

5. Start hadoop

```
./sbin/start-dfs.sh
```

`NameNode`, `DataNode` and `SecondaryNameNode` processes exist upon a successful startup

Installing "spark"

Go to spark's official website to download the required version

Since hadoop has been installed, use *Pre-build with user-provided Apache Hadoop* here

Use `hadoop` user as well to operate here

1. Decompress to enter the directory
2. Modify the configuration file

```
cp ./conf/spark-env.sh.template ./conf/spark-env.sh
vim ./conf/spark-env.sh
```

Add the following in the first line

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

Modify the path according to the installation of hadoop

3. Run the example

```
bin/run-example SparkPi
```

The display of an approximate value of π output by the program indicates a successful installation

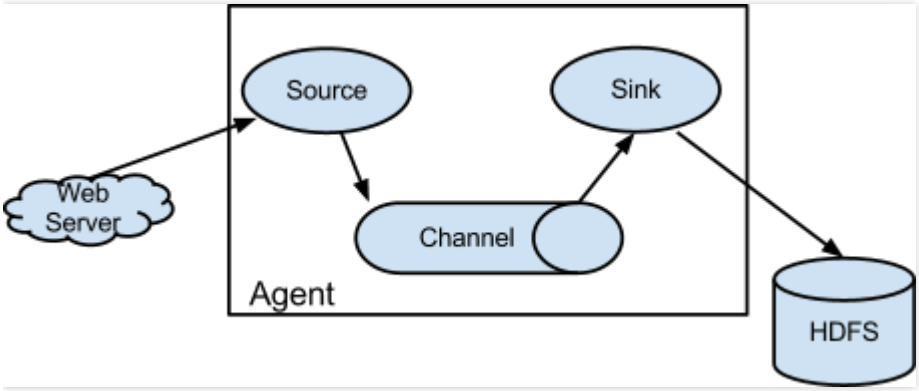
Flume with CKafka

Last updated：2018-02-08 11:19:16

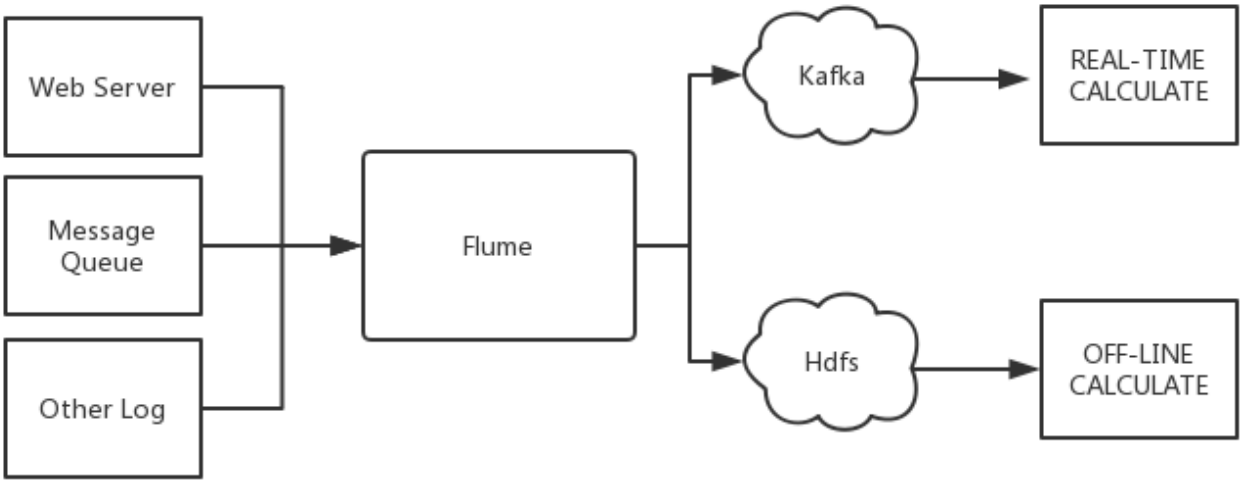
Introduction to Apache Flume

Apache Flume is a tool/service or a data aggregation mechanism, which collects data resources such as logs and events and centrally stores large amounts of data from various data sources. As a highly available and distributed configuration tool, Flume is designed by means of aggregating data streams, such as log data, from many different website servers to a centralized storage like HDFS and HBase.

The basic structure of Flume is shown below:



Flume uses agent as the minimum unit of an independent operation. An agent is a JVM. A single agent consists of three major components: Source, Sink and Channel.



Why Flume + Kafka

When you store data in a downstream storage module or a computing module such as HDFS or HBase, a variety of complex scenarios must be considered, for example, the amount of concurrent writes, system-loading pressure, network delay, etc. Flume is designed as a flexible distributed system with multiple APIs, and also provides customizable pipelines.
During production processing, the processing speeds are generally different. In this case, Kafka can be used as a cache. With "partition" structure and "append" for data addition, Kafka has an excellent throughput performance. Besides, "replication" also makes it highly fault-tolerant.
Therefore, most of the requirements in the production environment can be met by combining these two tools.

How to Connect to Open-source Kafka

Supported Version

1. The current version of Flume - Apache Flume 1.7.0 Released (released on October 17, 2016, and compatible with Kafka since version 1.6).
2. Kafka 0.9.x series are supported, while 0.8 is not supported now.

Preparations

1. Apache Flume (1.6.0 or later)
2. Kafka (0.9.x or later)
3. Flume's Kafka - Source, Sink components (confirmed to be used in Flume)

Flume and Kafka

Kafka can be used as "Source" or "Sink" to import or export messages.
1. Kafka Source
By configuring Kafka as a message source, you can pull the data from Kafka to the specified Sink as a consumer.
Main configuration options:

Configuration Item	Description
channels	Self-configured channel
type	Must be: org.apache.flume.source.kafka.KafkaSource

Configuration Item	Description
kafka.bootstrap.servers	The server of Kafka broker
kafka.consumer.group.id	The group id on the Kafka consumer side
kafka.topics	The topic of data source in Kafka
batchSize	The size of channel for each write
batchDurationMillis	The maximum time interval for each write

Example:

```
tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
tier1.sources.source1.channels = channel1
tier1.sources.source1.batchSize = 5000
tier1.sources.source1.batchDurationMillis = 2000
tier1.sources.source1.kafka.bootstrap.servers = localhost:9092
tier1.sources.source1.kafka.topics = test1, test2
tier1.sources.source1.kafka.consumer.group.id = custom.g.id
```

2. Kafka Sink

By configuring Kafka as the content receiver, you can push the data to Kafka Server for subsequent operations as a producer.

Main configuration options:

Configuration Item	Description
channel	Self-configured channel
type	Must be: org.apache.flume.sink.kafka.KafkaSink
kafka.bootstrap.servers	The server of Kafka broker
kafka.topics	The topic of data source in Kafka
flumeBatchSize	The size of Batch for each write
kafka.producer.acks	The production policy of Kafka producer

```
a1.sinks.k1.channel = c1
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = mytopic
```

```
a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
```

For more information, please see official website:
https://flume.apache.org/FlumeUserGuide.html

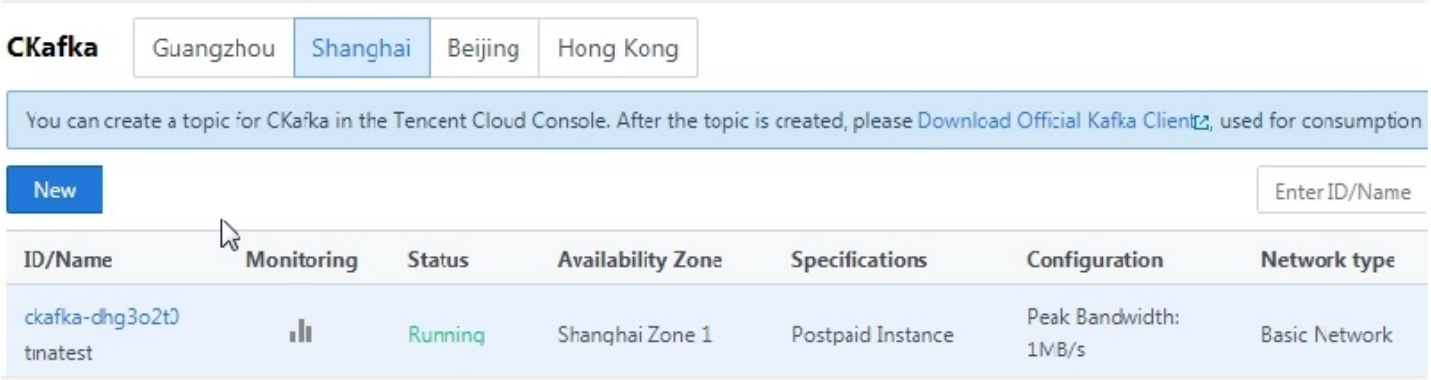
Connecting Flume to CKafka

Preparations

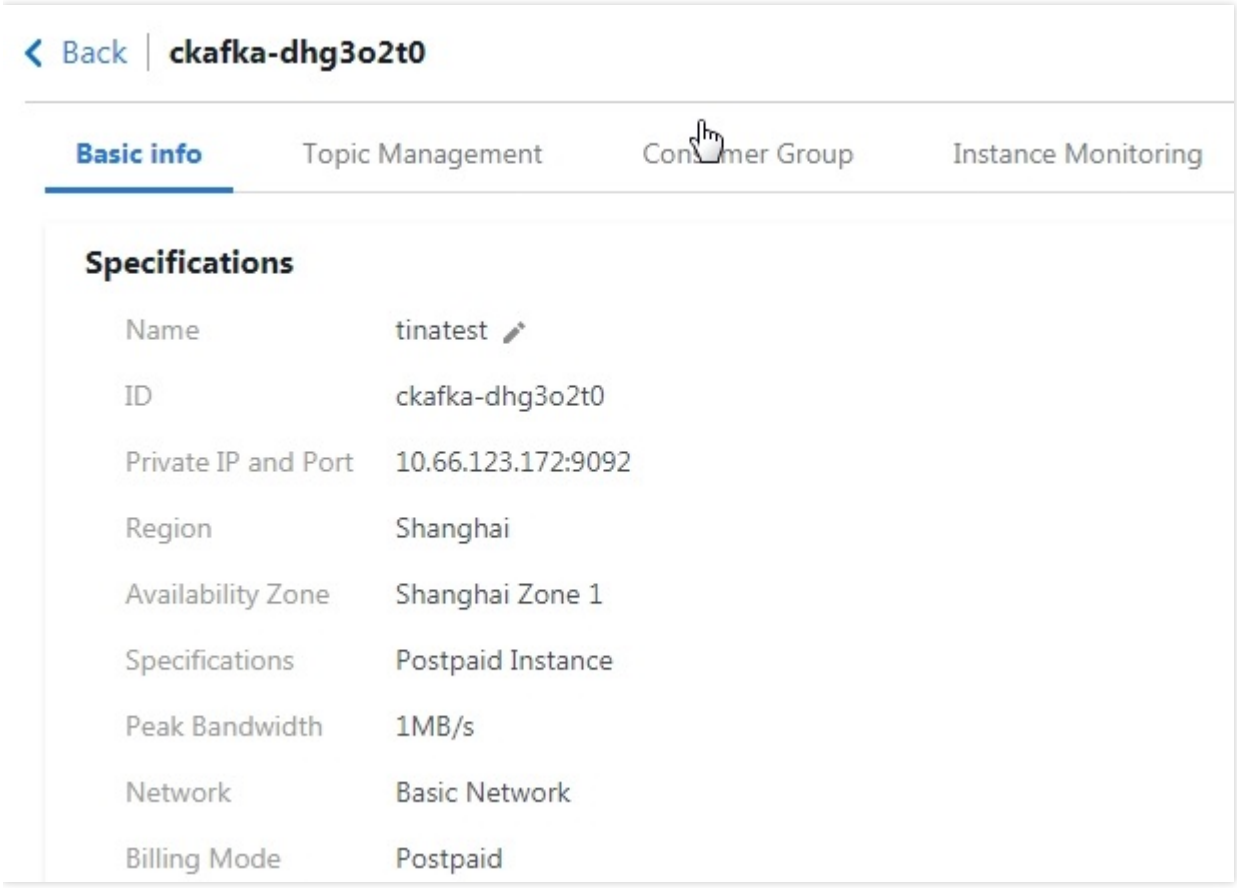
- Apply for an instance in the Ckafka application page, and create a corresponding Topic.
- For apache flume, this tutorial uses the latest flume 1.7.0 (http://flume.apache.org/download.html).

Creation of Ckafka

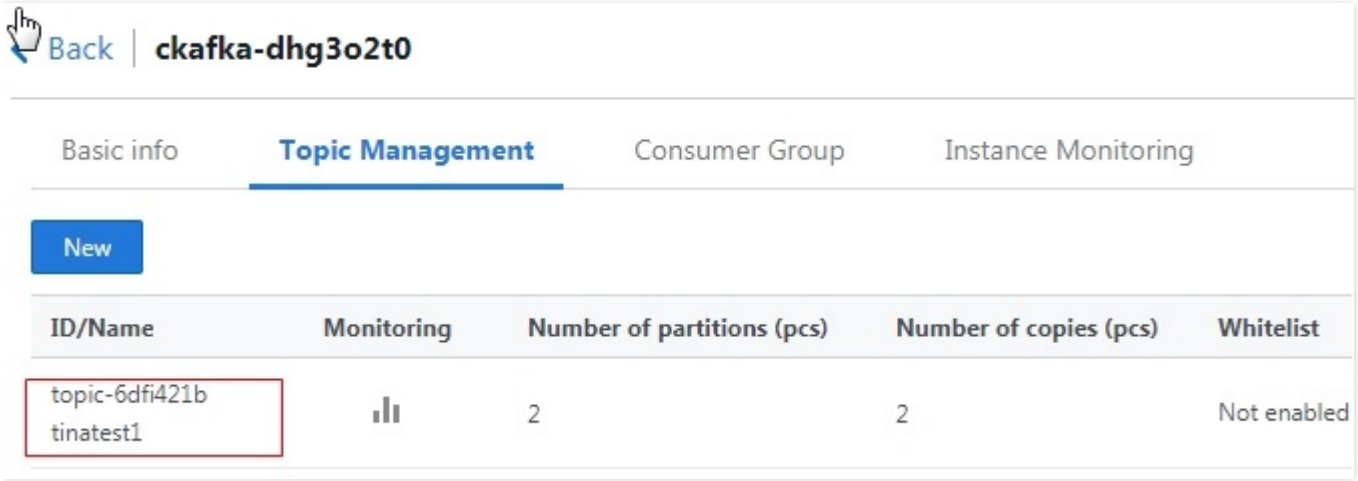
1) After the application for instance is approved, you can see the information of your instance from the console.



2) By clicking the instance name, you can see the specific information of assigned instance:



3) Click "Topic Management" to create a topic, where the name is flume_test.



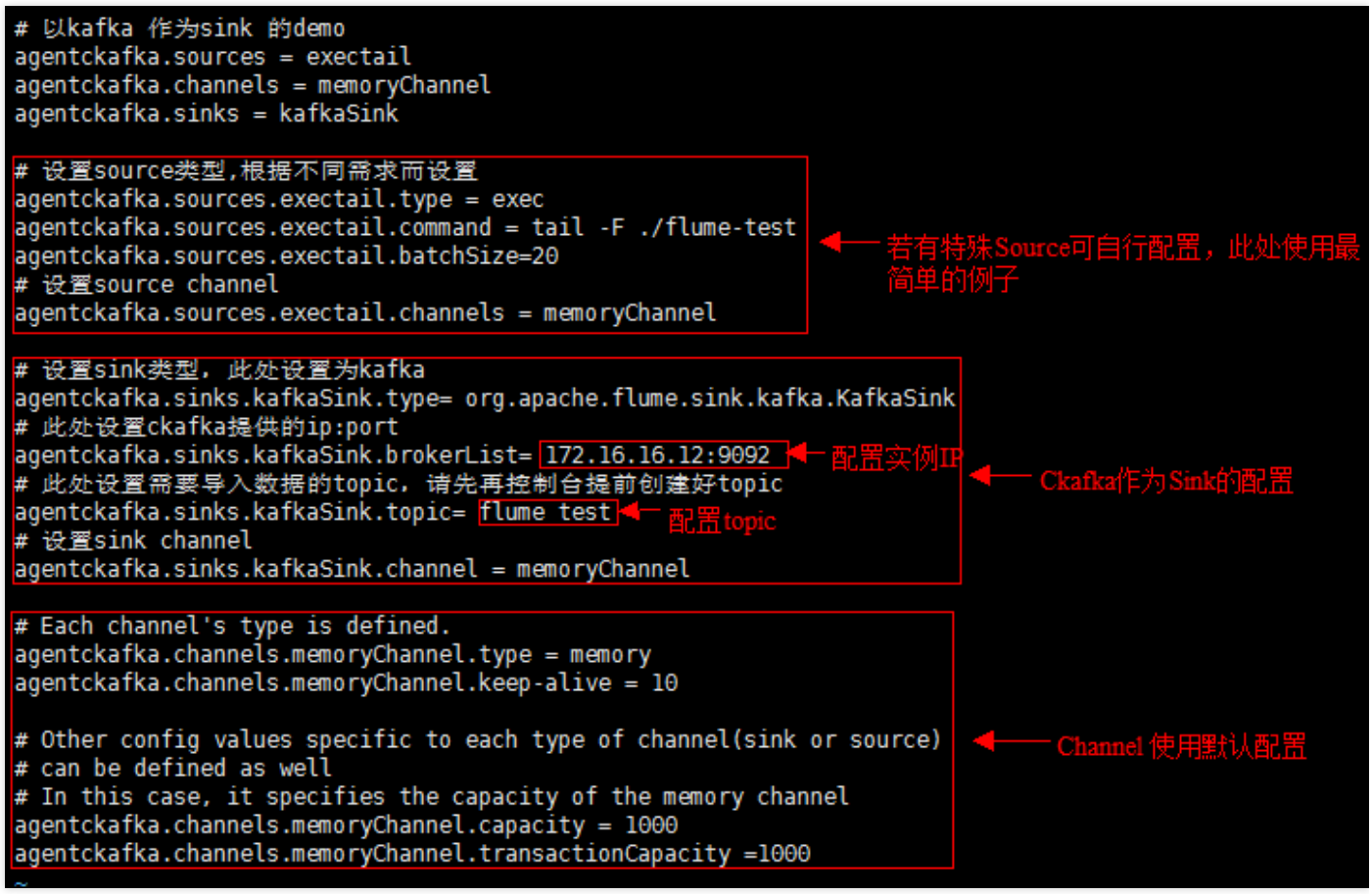
Now, the operating environment of Ckafka has been created.

Flume

1) Download the Apache flume package from the official website, and then decompress it.
2) Configure Flume options.

- Use Ckafka as Sink

  a) Write a configuration file, and focus on the combination of Flume and CKafka as Sink. Therefore, the default configuration is used for Source and Channel, which is not discussed here. The following is a simple demo (configured in the conf folder of the decompressed directory). Please note that, unless specified otherwise, use your own instance IP and topic to replace those in the configuration file:




```
# 以kafka 作为sink 的demo
agentckafka.sources = exectail
agentckafka.channels = memoryChannel
agentckafka.sinks = kafkaSink

# 设置source类型,根据不同需求而设置
agentckafka.sources.exectail.type = exec
agentckafka.sources.exectail.command = tail -F ./flume-test
agentckafka.sources.exectail.batchSize=20
# 设置source channel
agentckafka.sources.exectail.channels = memoryChannel

# 设置sink类型，此处设置为kafka
agentckafka.sinks.kafkaSink.type= org.apache.flume.sink.kafka.KafkaSink
# 此处设置ckafka提供的ip:port
agentckafka.sinks.kafkaSink.brokerList= 172.16.16.12:9092
# 此处设置需要导入数据的topic，请先再控制台提前创建好topic
agentckafka.sinks.kafkaSink.topic= flume_test
# 设置sink channel
agentckafka.sinks.kafkaSink.channel = memoryChannel

# Each channel's type is defined.
agentckafka.channels.memoryChannel.type = memory
agentckafka.channels.memoryChannel.keep-alive = 10

# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel
agentckafka.channels.memoryChannel.capacity = 1000
agentckafka.channels.memoryChannel.transactionCapacity =1000
```

  b) The source used here is tail -F flume-test, which is the new information in the file

  c) Launch Flume:

```
./bin/flume-ng agent -n agentckafka -c conf -f conf/flume-kafka-sink.properties
```

  d) Write a message to the flume-test file, and then the message is written into CKafka by Flume

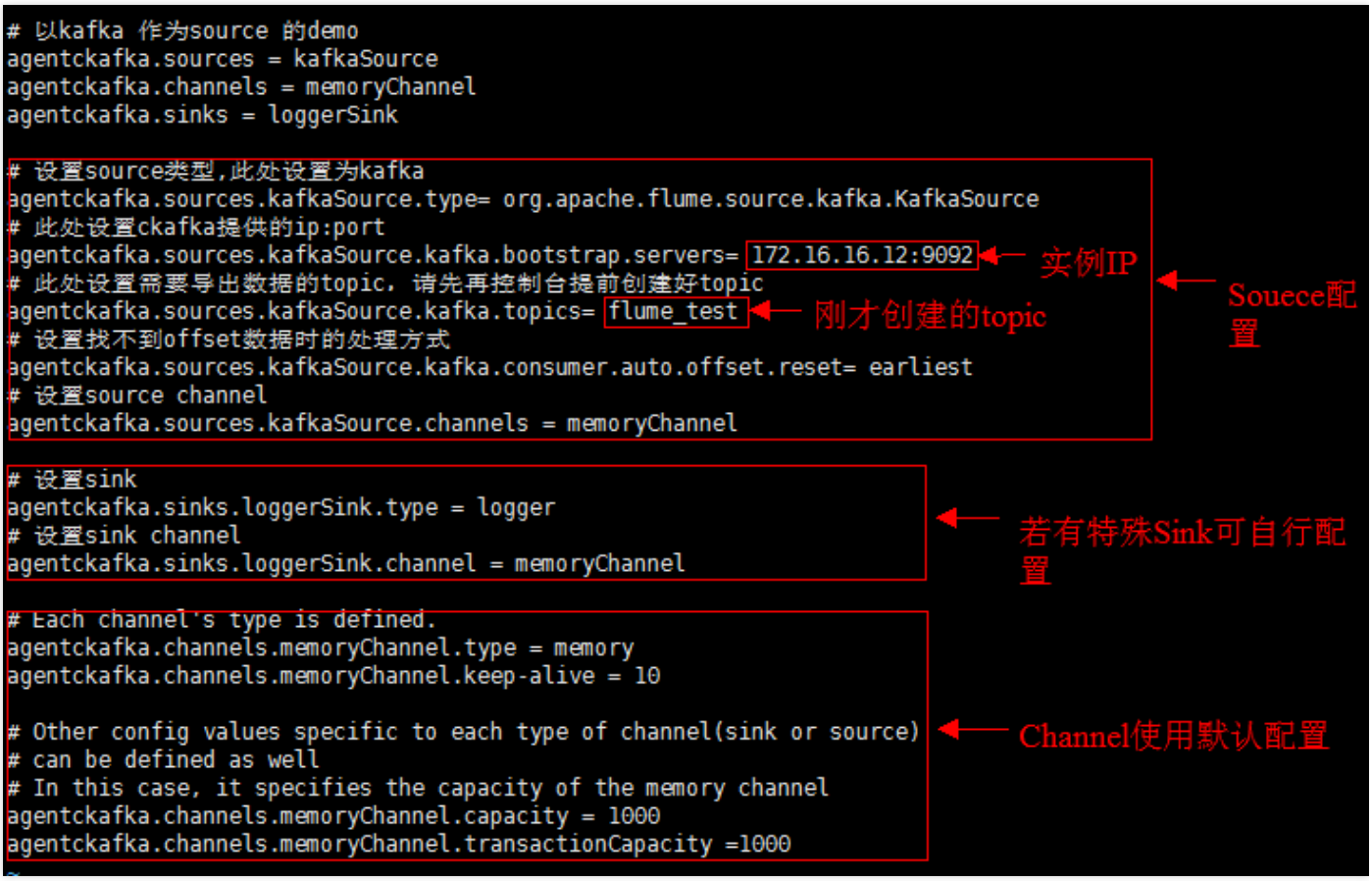
```
[root@VM_16_17_centos apache-flume-1.7.0-bin]# cat flume-test
ckafka
```

  e) Launch the CKafka client for consumption:

```
./kafka-console-consumer.sh --bootstrap-server 172.16.16.12:9092 --topic flume_test --from-beginnin
```

You can see that the above message has been consumed out

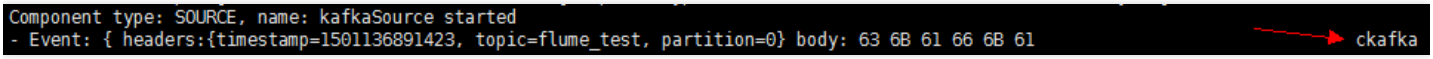
- Use Ckafka as Source
  a) Write a configuration file, and focus on the combination of Flume and CKafka as Source. Therefore, the default configuration is used for Sink and Channel, which is not discussed here. The following is a simple demo (configured in the conf folder of the decompressed directory). Please note that, unless specified otherwise, use your own instance IP and topic to replace those in the configuration file:



b) The sink used here is logger
c) Launch Flume:

```
./bin/flume-ng agent -n agentckafka -c conf -f conf/flume-kafka-source.properties
```

d) View the output information of logger (default path is logs/flume.log)



Storm with CKafka

Last updated：2018-02-08 11:20:46

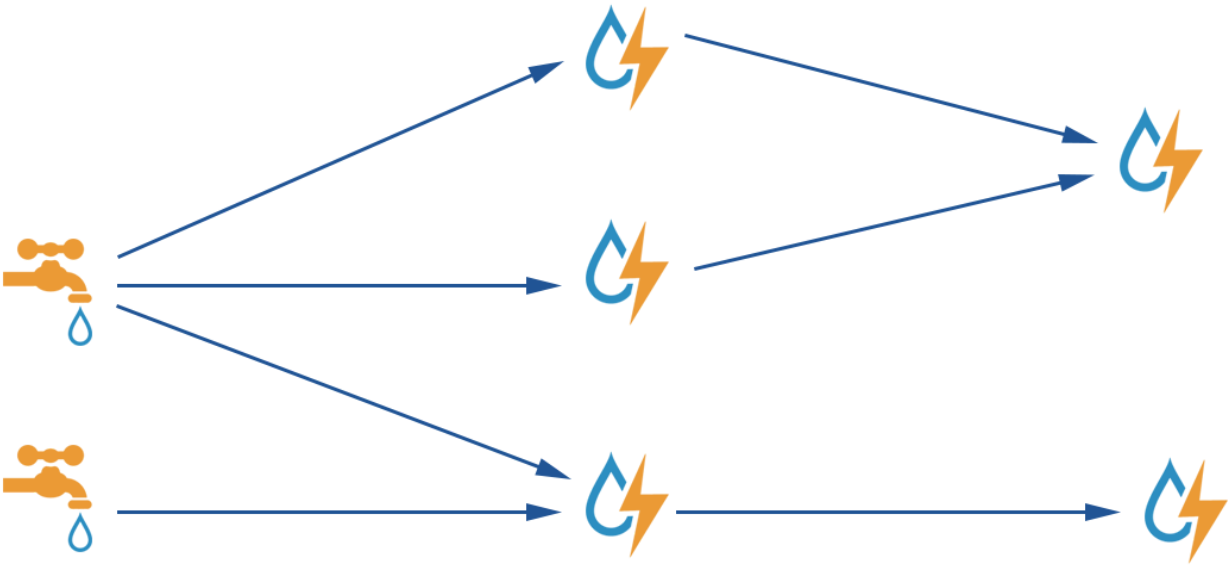
Introduction to Storm

Storm is a real-time distributed computing framework, which can perform stream-based data processing and provide common distributed RPC calling, so as to minimize the delay of event processing to sub-seconds. It is suitable for the real-time data processing scenarios with high delay requirements.

How Does Storm Work

Two types of nodes are supported in Storm clusters: the control node `Master Node` and the work node `Worker Node` . The `Nimbus` process runs on the `Master Node` for resource assignment and status monitoring. The `Supervisor` process runs on the `Worker Node` for listening on work tasks and starting `executor` . The entire Storm cluster relies on `zookeeper` for public data storage, cluster status listening, task assignment, etc.

The data processing program submitted to Storm by users is called `topology` . The minimum message unit it processes is `tuple` - an array of arbitrary objects. `topology` consists of `spout` and `bolt` , in which `spout` is the source of `tuple` , and `bolt` can subscribe any `tuple` issued by `spout` or `bolt` for processing.



Storm with ckafka

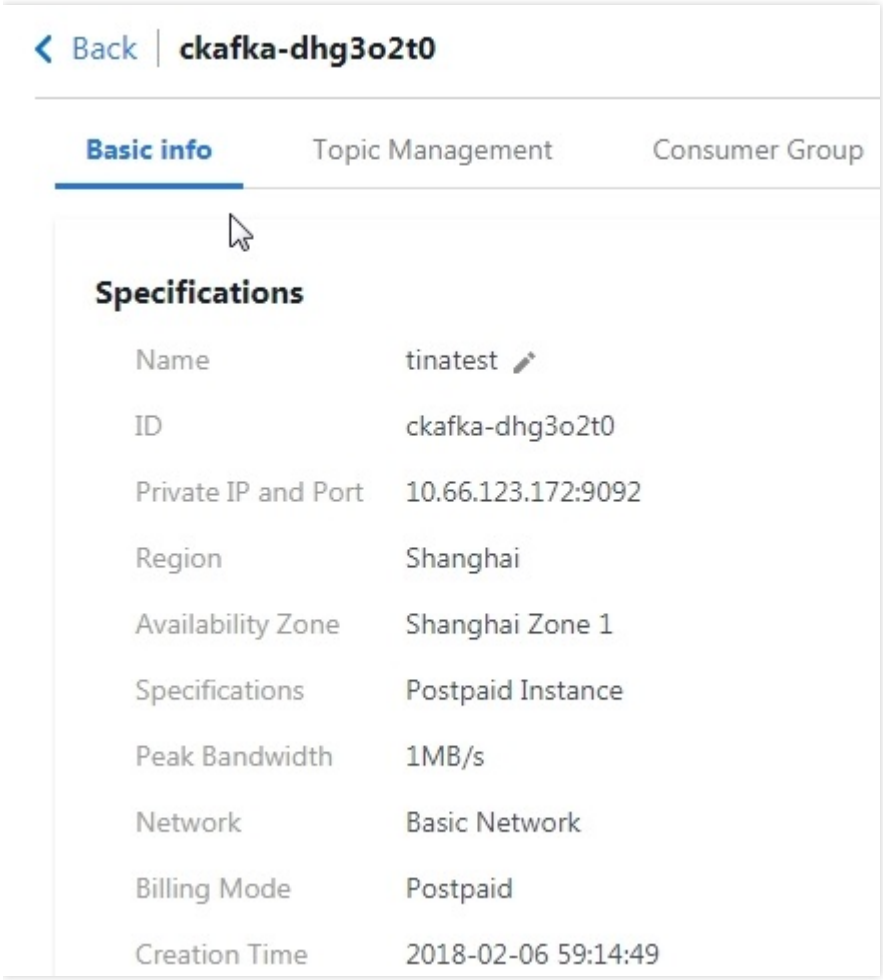
Storm can use CKafka as `spout` to process consumed data, or as `bolt` to store the processed data and provide the data for other components to consume.

Testing Environment

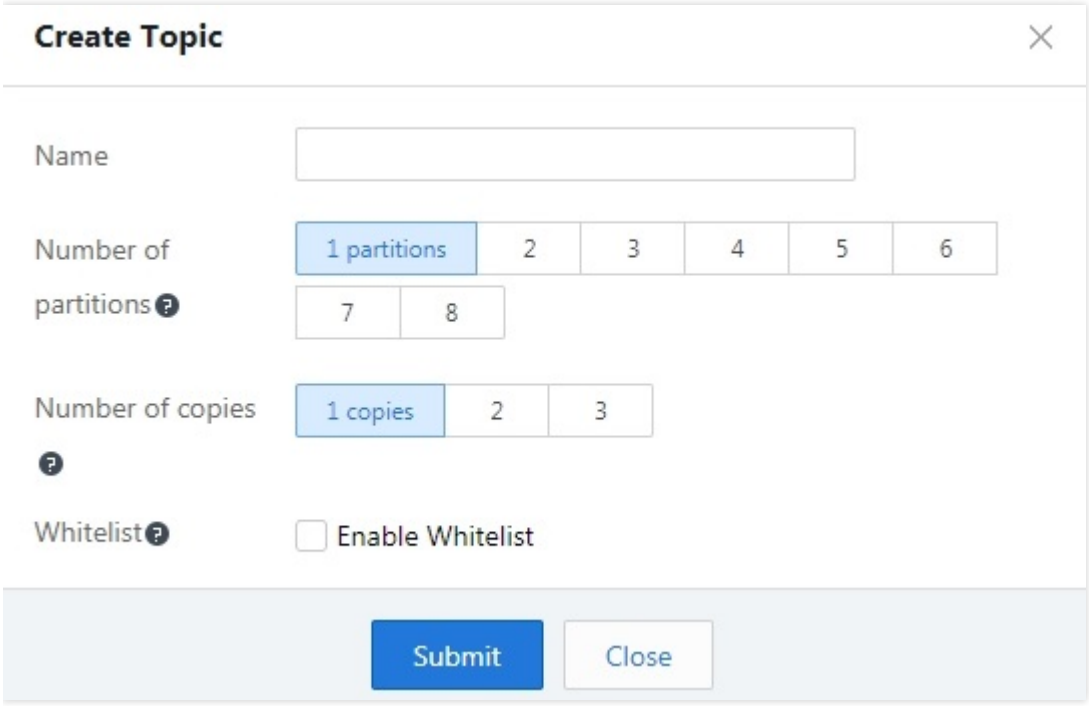
Centos 6.8 System

Package	Version
maven	3.5.0
storm	1.1.0
ssh	5.3
Java	1.8

Applying for the Creation of Ckafka Instance



Creating Topic



"maven" Dependency

"pom.xml" is configured as follows

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <modelVersion>4.0.0</modelVersion>
  <groupId>storm</groupId>
  <artifactId>storm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>storm</name>
    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.apache.storm</groupId>
         <artifactId>storm-core</artifactId>
         <version>1.1.0</version>
         <scope>provided</scope>
      </dependency>

      <dependency>
         <groupId>org.apache.storm</groupId>
         <artifactId>storm-kafka</artifactId>
         <version>1.1.0</version>
         <exclusions>
```

```
            <exclusion>
              <groupId>org.apache.kafka</groupId>
              <artifactId>kafka-clients</artifactId>
            </exclusion>
          </exclusions>
        </dependency>
        <dependency>
          <groupId>org.apache.storm</groupId>
          <artifactId>storm-kafka-client</artifactId>
          <version>1.1.1</version>
        </dependency>
        <dependency>
          <groupId>org.apache.kafka</groupId>
          <artifactId>kafka_2.11</artifactId>
          <version>0.10.2.1</version>
          <exclusions>
            <exclusion>
              <groupId>org.slf4j</groupId>
              <artifactId>slf4j-log4j12</artifactId>
            </exclusion>
          </exclusions>
        </dependency>
        <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>3.8.1</version>
        </dependency>
      </dependencies>

      <build>
        <plugins>
          <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
              <descriptorRefs>
                <descriptorRef>jar-with-dependencies</descriptorRef>
              </descriptorRefs>
              <archive>
                <manifest>
                  <mainClass>ExclamationTopology</mainClass>
                </manifest>
              </archive>
            </configuration>
            <executions>
              <execution>
```

```
                <id>make-assembly</id>
                <phase>package</phase>
                <goals>
                  <goal>single</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
              <source>1.8</source>
              <target>1.8</target>
            </configuration>
          </plugin>
        </plugins>
    </build>
</project>
```

Writing in CKafka

Using "spout/bolt"

"topology" Code

```
//KafkaProduceTopology.java
import org.apache.storm.Config;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.storm.generated.StormTopology;
import org.apache.storm.kafka.bolt.KafkaBolt;
import org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.bolt.selector.KafkaTopicSelector;
import org.apache.storm.kafka.bolt.selector.DefaultTopicSelector;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.io.Serializable;
import java.util.Arrays;
import java.util.List;
```

```
import java.util.concurrent.TimeUnit;

import java.util.Properties;

public class KafkaProduceTopology {
    public static void main(String[] args) throws Exception {
        //"ip:port" of the CKafka instance you just applied for
        String bootstrapServers = "111.230.216.45:9092";
        //Specify the topic into which the message is written
        String topic = "storm-topology-test";
          //Set producer attribute
          //Function reference:https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients
          //Attribute reference: http://kafka.apache.org/0102/documentation.html

          Properties props = new Properties();
          props.put("bootstrap.servers", bootstrapServers);
          props.put("acks", "1");
          props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
          props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        //Create a bolt to be written into Kafka. "fields" ("key", "message") is used as the key and message f
          KafkaBolt kafkaBolt = new KafkaBolt()
           .withProducerProperties(props)
           .withTopicSelector(new DefaultTopicSelector(topic))
           .withTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper());

          TopologyBuilder builder = new TopologyBuilder();
          //A spout class that generates messages in order, and its output field is sentence
        SerialSentenceSpout spout = new SerialSentenceSpout();
        AddMessageKeyBolt bolt = new AddMessageKeyBolt();
         builder.setSpout("spout", spout, 1);
          //Add the fields required to produce messages to CKafka for tuple
        builder.setBolt("add-key", bolt, 1).shuffleGrouping("spout");
        //Write the following into CKafka
          builder.setBolt("forwardToKafka", kafkaBolt, 8).shuffleGrouping("add-key");

        Config conf = new Config();
        //conf.setDebug(true);
        if (args != null && args.length > 0) {
            conf.setNumWorkers(1);

            StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
        }
        else {
            LocalCluster cluster = new LocalCluster();
             cluster.submitTopology("test", conf, builder.createTopology());
```

```
            Utils.sleep(10000);
            cluster.killTopology("test");
            cluster.shutdown();
        }
    }
}
```

Add "key" and "message" fields to `tuple`. If key is null, the produced messages are evenly assigned to each partition. When key is specified, messages are hashed to a specific partition based on the key value.

```
//AddMessageKeyBolt.java
public class AddMessageKeyBolt extends BaseBasicBolt {
    public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
    }
    //Add key for message
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String message = tuple.getString(0);
        collector.emit(new Values(null, message));
    }
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("key", "message"));
    }
}
```

Using "trident"

Use the trident class to generate topology

```
//TridentKafkaProduceTopology.java
import org.apache.storm.Config;
import org.apache.storm.kafka.trident.TridentKafkaState;
import org.apache.storm.kafka.trident.TridentKafkaStateFactory;
import org.apache.storm.kafka.trident.TridentKafkaUpdater;
import org.apache.storm.kafka.trident.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.trident.selector.DefaultTopicSelector;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.Stream;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.tuple.TridentTuple;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;
```

```
import java.io.Serializable;
import java.util.Arrays;
import java.util.concurrent.TimeUnit;
import java.util.List;
import java.util.Properties;

public class TridentKafkaProduceTopology {
    //Add the fields required to produce messages to CKafka for tuple
    public static class AddMessageKey extends BaseFunction {
        public void execute(TridentTuple tuple, TridentCollector collector)
        {
            String value = tuple.getString(0);
            int key = value.hashCode();
            //collector.emit(new Values(Integer.toString(key), tuple.getString(0)));
            collector.emit(new Values(null, tuple.getString(0)));
        }
    }
    public static void main(String[] args) throws Exception {
        //"ip:port" of the CKafka instance you just applied for
        String bootstrapServers = "111.230.216.45:9092";
        //Specify the topic into which the message is written
        String topic = "storm-trident-test";

            //Set producer attribute
            //Function reference:https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients
            //Attribute reference: http://kafka.apache.org/0102/documentation.html
            Properties props = new Properties();
            props.put("bootstrap.servers", bootstrapServers);
            props.put("acks", "1");
            props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
            props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        //A spout that generates messages in batches, and its output field is sentence
        TridentSerialSentenceSpout spout = new TridentSerialSentenceSpout(5);
        //Set trident
        TridentKafkaStateFactory stateFactory = new TridentKafkaStateFactory()
            .withProducerProperties(props)
            .withKafkaTopicSelector(new DefaultTopicSelector(topic))
            //Set fields ("key", "value") to be used for writing messages
            .withTridentTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper("key", "value"));

        TridentTopology builder = new TridentTopology();
        Stream stream = builder.newStream("spout", spout)
            .each(new Fields("sentence"), new AddMessageKey(), new Fields("key", "value"))
```

```
        ;
        stream.partitionPersist(stateFactory, new Fields("key", "value"), new TridentKafkaUpdater(), new Fiel

          Config conf = new Config();
        //conf.setDebug(true);
        if (args != null && args.length > 0) {
            conf.setNumWorkers(1);
            StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.build());
        }
        else {
            LocalCluster cluster = new LocalCluster();
             cluster.submitTopology("test", conf, builder.build());
             Utils.sleep(5000);
             cluster.killTopology("test");
             cluster.shutdown();
        }
    }
}
```

Consuming from CKafka

Using "spout/bolt"

```
//KafkaConsumeTopology.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.kafka.spout.KafkaSpout;
import org.apache.storm.kafka.spout.Func;
import org.apache.storm.kafka.spout.KafkaSpoutConfig;
import org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy;
import org.apache.storm.kafka.spout.KafkaSpoutRetryExponentialBackoff;
import org.apache.storm.kafka.spout.KafkaSpoutRetryExponentialBackoff.TimeInterval;
import org.apache.storm.kafka.spout.KafkaSpoutRetryService;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutOpaque;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.io.Serializable;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.TimeUnit;
```

```
import static org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy.LATEST;
import static org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy.EARLIEST;

public class KafkaConsumeTopology {
    public static void main(String[]  args) throws Exception {
        //"ip:port" of the CKafka instance you just applied for
        String bootstrapServers = "111.230.216.45:9092";
        //Specify the topic into which the message is written
        String topic = "storm-topology-test";
        Config conf = new Config();
        //conf.setDebug(true);
        conf.setMaxSpoutPending(20);
        conf.setNumWorkers(1);
        //Set retry policy
        KafkaSpoutRetryService kafkaSpoutRetryService =  new KafkaSpoutRetryExponentialBackoff(
            KafkaSpoutRetryExponentialBackoff.TimeInterval.microSeconds(500),
            KafkaSpoutRetryExponentialBackoff.TimeInterval.milliSeconds(2),
            Integer.MAX_VALUE,
            KafkaSpoutRetryExponentialBackoff.TimeInterval.seconds(10));
        //Set consumer parameter
        //Function reference:http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spou
        //Attribute reference: http://kafka.apache.org/0102/documentation.html
        KafkaSpoutConfig spoutConf =  KafkaSpoutConfig.builder(bootstrapServers, topic)
            .setGroupId("test-group1")    //Set consumption groupId
            .setProp(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true")    //Set automatic confirmatic
            .setProp(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000") //Set session timeout
            .setProp(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000") //Set request timeout
            .setOffsetCommitPeriodMs(10_000)    //Set automatic confirmation time (in ms)
            .setFirstPollOffsetStrategy(LATEST)    //Set to pull the latest messages
            .setRetry(kafkaSpoutRetryService)
            .build();

        final TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("kafka-spout", new KafkaSpout(spoutConf), 1);

        if (args != null && args.length > 0) {
            conf.setNumWorkers(3);
            StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
        }
        else {
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", conf, builder.createTopology());
            Utils.sleep(200000);
            cluster.killTopology("test");
```

```
            cluster.shutdown();
        }
    }
}
```

Using "trident"

```
//TridentKafkaTopology.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.kafka.spout.KafkaSpout;
import org.apache.storm.kafka.spout.Func;
import org.apache.storm.kafka.spout.KafkaSpoutConfig;
import org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy;
import org.apache.storm.kafka.spout.KafkaSpoutRetryExponentialBackoff;
import org.apache.storm.kafka.spout.KafkaSpoutRetryExponentialBackoff.TimeInterval;
import org.apache.storm.kafka.spout.KafkaSpoutRetryService;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutOpaque;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.trident.Stream;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.io.Serializable;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.TimeUnit;

import static org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy.EARLIEST;
import static org.apache.storm.kafka.spout.KafkaSpoutConfig.FirstPollOffsetStrategy.LATEST;
import com.william.storm.trident.TridentPrinter;

public class TridentKafkaConsumeTopology {
    public static void main(String[]  args) throws Exception {
        //"ip:port" of the CKafka instance you just applied for
        String bootstrapServers = "111.230.216.45:9092";
        //Specify the topic into which the message is written
        String topic = "storm-trident-test";
        Config conf = new Config();

        conf.setMaxSpoutPending(20);
        conf.setNumWorkers(1);
```

```
        //Set retry policy
        KafkaSpoutRetryService kafkaSpoutRetryService =  new KafkaSpoutRetryExponentialBackoff(
            KafkaSpoutRetryExponentialBackoff.TimeInterval.microSeconds(500),
            KafkaSpoutRetryExponentialBackoff.TimeInterval.milliSeconds(2),
            Integer.MAX_VALUE,
            KafkaSpoutRetryExponentialBackoff.TimeInterval.seconds(10));
        //Set consumer parameter
        //Function reference:http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spou
        //Attribute reference: http://kafka.apache.org/0102/documentation.html
        KafkaSpoutConfig spoutConf =  KafkaSpoutConfig.builder(bootstrapServers, topic)
            .setGroupId("test-group1")    //Set consumption groupId
            .setProp(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true")    //Set automatic confirmatic
            .setProp(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000") //Set session timeout
            .setProp(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000") //Set request timeout
            .setOffsetCommitPeriodMs(10_000)    //Set automatic confirmation time (in ms)
            .setFirstPollOffsetStrategy(LATEST)    //Set to pull the latest messages
            .setRetry(kafkaSpoutRetryService)
            .build();

        TridentTopology builder = new TridentTopology();
        final Stream stream = builder.newStream("kafka-spout",
            new KafkaTridentSpoutOpaque(spoutConf))

        if (args != null && args.length > 0) {
            conf.setNumWorkers(3);
            StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.build());
        }
        else {
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", conf, builder.build());
            Utils.sleep(100000);
            cluster.killTopology("test");
            cluster.shutdown();
        }
    }
}
```

Submitting "Storm"

After being compiled with mvn package, Storm can be submitted to the local cluster for debug testing, or submitted to the formal cluster for running.

```
storm jar your_jar_name.jar topology_name
```

```
storm jar your_jar_name.jar topology_name tast_name
```

Logstash with CKafka

Last updated：2018-02-08 11:34:51

Introduction to Logstash

Logstash, an open-source log processing tool, is used to collect data from multiple sources, filter the collected data, and store the data for other purposes.

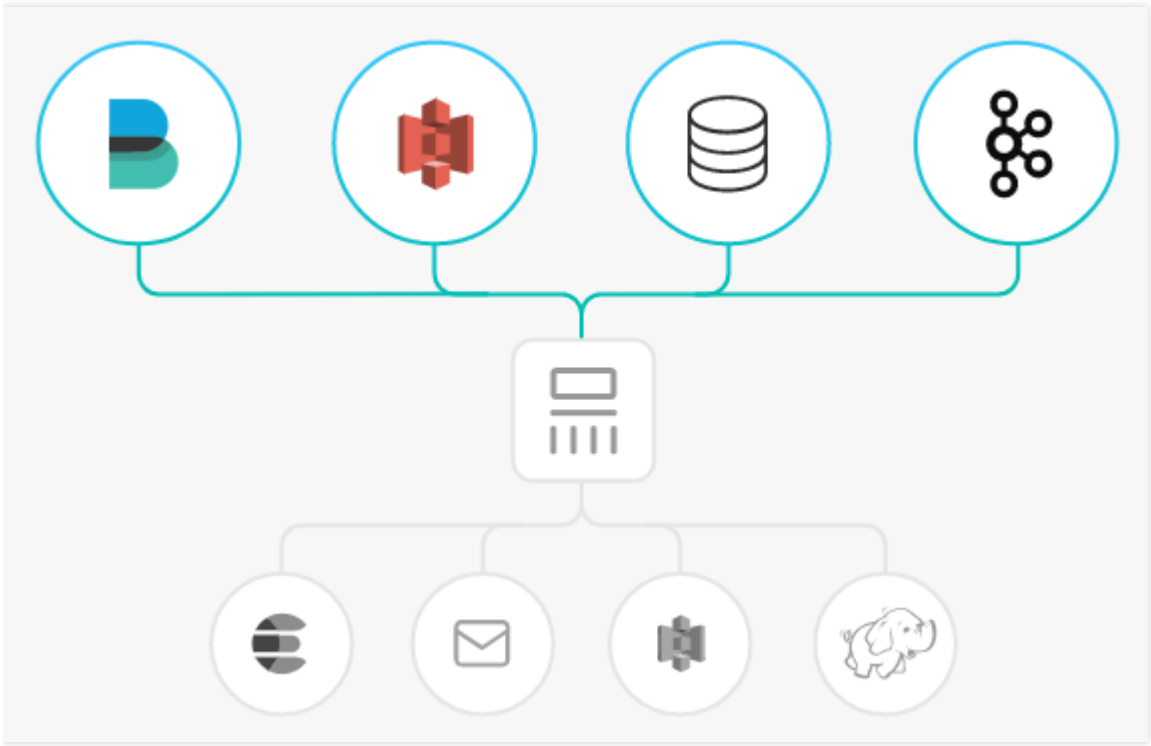
Logstash is highly flexible and features a powerful syntax analysis capability with a variety of plugs-ins. It also supports multiple input/output sources. In addition, as a horizontally scalable data pipeline, it is powerful in log collection and retrieval by coordinating with Elasticsearch and Kibana.

How Does Logstash Work

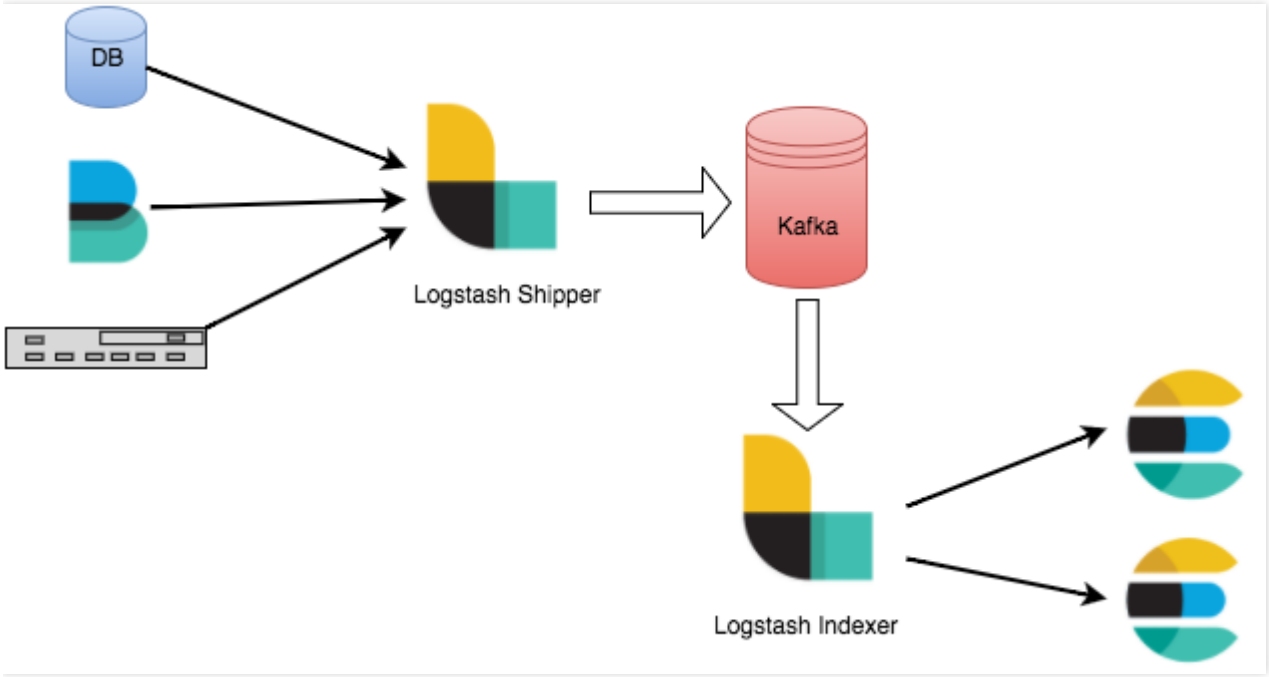
Logstash data processing can be divided into three stages: inputs → filters → outputs.

1. inputs - Generate data sources, such as file, syslog, redis, and beats.
2. filter - Modify and filter data, which belongs to the intermediate process in Logstash data pipeline. You can change events based on specific conditions. Some common filters include grok, mutate, drop and clone.
3. outputs - Transfer data to other locations. An event can be transferred to multiple outputs, and it ends when the transfer is completed. Elasticsearch is the most common output.

Besides, Logstash supports encoding/decoding, and you can specify the format on the inputs/outputs side.



Why Logstash + Kafka



1. You can asynchronously process data to prevent burst traffic.

2. In the process of decoupling, when an exception occurs in Elasticsearch, the upstream work is not affected.
3. Filtering data using Logstash consumes resources. The performance of Logstash may be affected if it is deployed on the production server.

Connecting CKafka

Supported Version

inputs

The compatibility of official version is described below:

Kafka Client Version	Logstash Version	Plugin Version	Why?
0.8	2.0.0 - 2.x.x	<3.0.0	Legacy, 0.8 is still popular
0.9	2.0.0 - 2.3.x	3.x.x	Works with the old Ruby Event API (`event['product']['price'] = 10`)
0.9	2.4.x - 5.x.x	4.x.x	Works with the new getter/setter APIs (`event.set('[product][price]', 10)`)
0.10.0.x	2.4.x - 5.x.x	5.x.x	Not compatible with the ⇐ 0.9 broker

The latest version for now is v5.1.8, which uses 0.10 Consumer API to read data.

For more information on parameter configuration, please see https://www.elastic.co/guide/en/logstash/current/plugins-inputs-kafka.html.

outputs

The compatibility of official version is described below:

Kafka Client Version	Logstash Version	Plugin Version	Why?
0.8	2.0.0 - 2.x.x	<3.0.0	Legacy, 0.8 is still popular
0.9	2.0.0 - 2.3.x	3.x.x	Works with the old Ruby Event API (`event['product']['price'] = 10`)
0.9	2.4.x - 5.x.x	4.x.x	Works with the new getter/setter APIs (`event.set('[product][price]', 10)`)
0.10.0.x	2.4.x - 5.x.x	5.x.x	Not compatible with the ⇐ 0.9 broker

The latest version for now is v5.1.7, which uses 0.10 Consumer API to produce data.

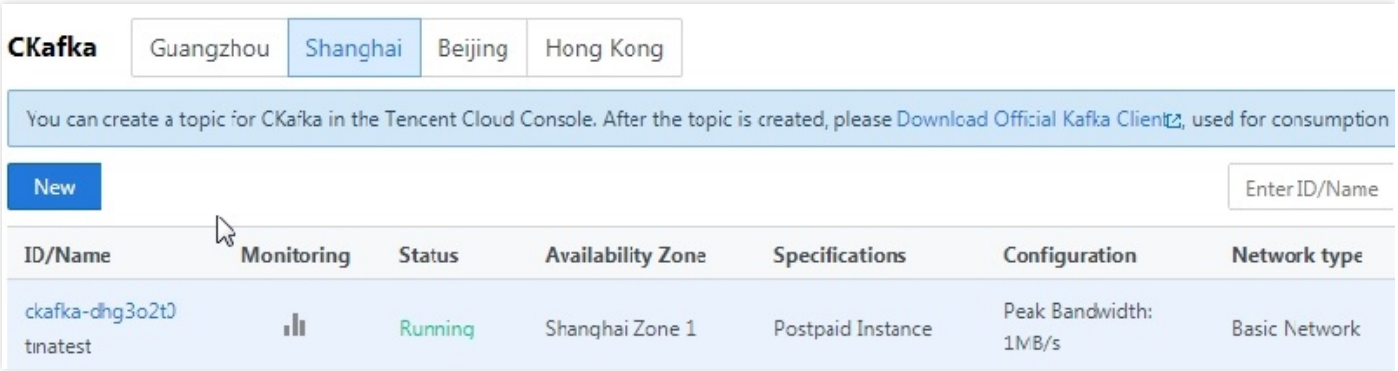
For more information on parameter configuration, please see https://www.elastic.co/guide/en/logstash/current/plugins-outputs-kafka.html.

Preparations

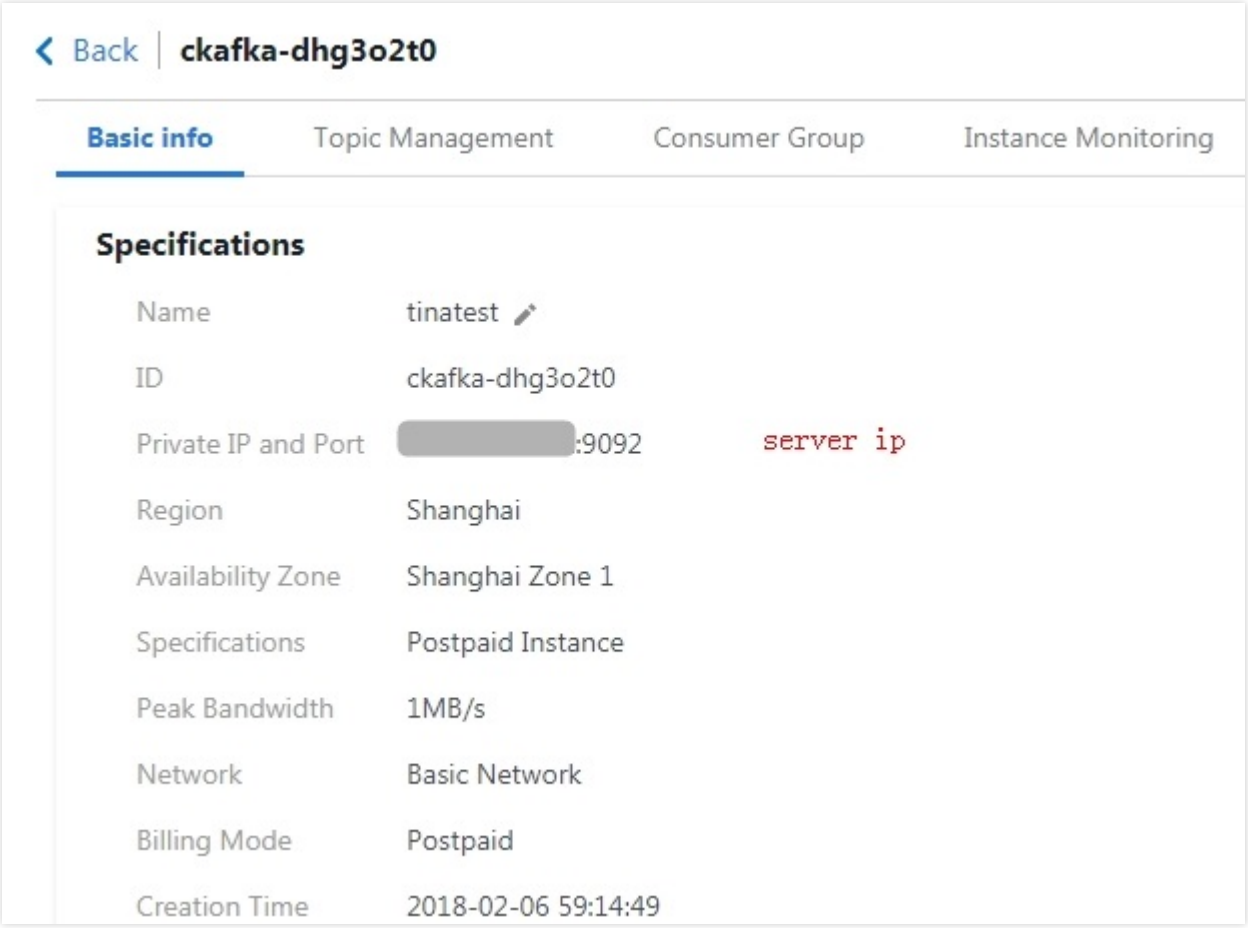
- Java version: java 8
- Logstash version: 5.5.2 (August 17, 2017)
- Apply for a Ckafka instance, and create a corresponding topic

Creation of CKafka

- After the application for instance is approved, you can see the information of your instance from the console.



- By clicking the instance name, you can see the specific information of assigned instance.

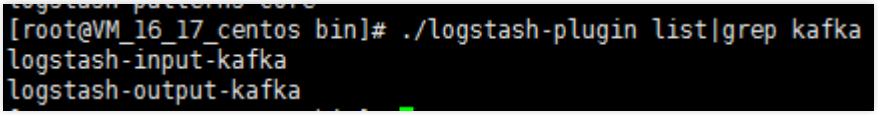


- Click "Topic Management" to create a topic, where the name is **logstash_test**.

Now, the operating environment of CKafka has been created.

Connecting CKafka as "inputs"

1. Run bin/logstash-plugin list to check whether logstash-input-kafka is contained in the supported plugs-in.



2. Write the configuration file "input.conf".
   Here, standard output is taken as the key data, and Kafka is used as the data source

```
input{
    kafka {
        bootstrap_servers => "172.16.16.12:9092"   <-- ckafka instance's ip
        group_id => "logstash_group"
        topics => ["logstash_test"]                <-- topic's name
        consumer_threads => 3
        auto_offset_reset => "earliest"
    }
}

output{
    stdout{codec=>rubydebug}
}
~
```

3. Launch Logstash to consume message.

```
[root@VM_16_17_centos bin]# ./logstash -f input.conf
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
Sending Logstash's logs to /data/ryan/logstash-5.5.2/logs which is now configured via log4j2.properties
[2017-09-06T18:07:41,926][INFO ][logstash.pipeline        ] Starting pipeline {"id"=>"main", "pipeline.workers"=>4, "pipe
[2017-09-06T18:07:41,943][INFO ][logstash.pipeline        ] Pipeline main started
[2017-09-06T18:07:41,999][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:24:23.039Z localhost ckafka"
}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:23:28.343Z localhost logstash"
}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:23:15.848Z localhost test"
}
```

You can see that the data in the topic above has been consumed out now.

For more information on parameter configuration when Kafka is used as output, please see https://www.elastic.co/guide/en/logstash/current/plugins-inputs-kafka.html#plugins-inputs-kafka-auto_offset_reset.

Connecting CKafka as "outputs"

1. Run bin/logstash-plugin list to check whether logstash-output-kafka is contained in the supported plugs-in.

```
[root@VM_16_17_centos bin]# ./logstash-plugin list|grep kafka
logstash-input-kafka
logstash-output-kafka
```

2. Write the configuration file "output.conf".
   Here, standard input is taken as the data source, and Kafka is used as the data destination.

```
input {
    stdin{}
}

output {
    kafka {
        bootstrap_servers => "172.16.16.12:9092"  <-- ckafka instance's name
        topic_id => "logstash_test"
    }                                              <-- topic's name
}
```

3. Launch Logstash to produce message.

```
[root@VM_16_17_centos bin]# ./logstash -f output.conf   <-- launch logstash
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
Sending Logstash's logs to /data/ryan/logstash-5.5.2/logs which is now configured via log4j2.properties
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[2017-09-06T17:22:56,185][INFO ][logstash.pipeline        ] Starting pipeline {"id"=>"main", "pipeline.workers"=>4, "pip
[2017-09-06T17:22:56,202][INFO ][logstash.pipeline        ] Pipeline main started
The stdin plugin is now waiting for input:
[2017-09-06T17:22:56,239][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600}
test
logstash   <-- produce message
ckafka
```

4. Verify the data produced just now.

```
[root@VM_16_17_centos bin]# ./kafka-console-consumer.sh --bootstrap-server 172.16.16.12:9092 --topic logstash_test --from-beginning --new-consumer
2017-09-06T09:23:28.343Z localhost logstash
2017-09-06T09:24:23.039Z localhost ckafka
2017-09-06T09:23:15.848Z localhost test
```

For more information on parameter configuration when Kafka is used as output, please see https://www.elastic.co/guide/en/logstash/current/plugins-outputs-kafka.html.