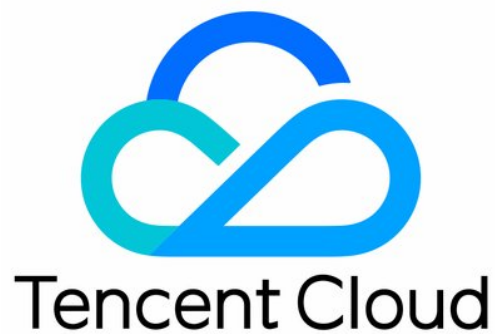


# **Message Queue CKafka**

## **Best Practices**

### **Product Documentation**



## Copyright Notice

©2013-2022 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Best Practices

Connecting Flink to CKafka

Connecting Schema Registry to CKafka

Connecting Spark Streaming to CKafka

Connecting Flume to CKafka

Connecting Kafka Connect to CKafka

Connecting Storm to CKafka

Connecting Logstash to CKafka

Connecting Filebeat to CKafka

Multi-AZ Deployment

Production and Consumption

Log Access

Connecting CLS to CKafka

Replacing Supportive Route (Old)

DataHub Best Practices

Connection to Kafka over HTTP

Unified Data Reporting

Simple Data Cleansing

Analysis of Change Logs Tracked by MongoDB Change Streams

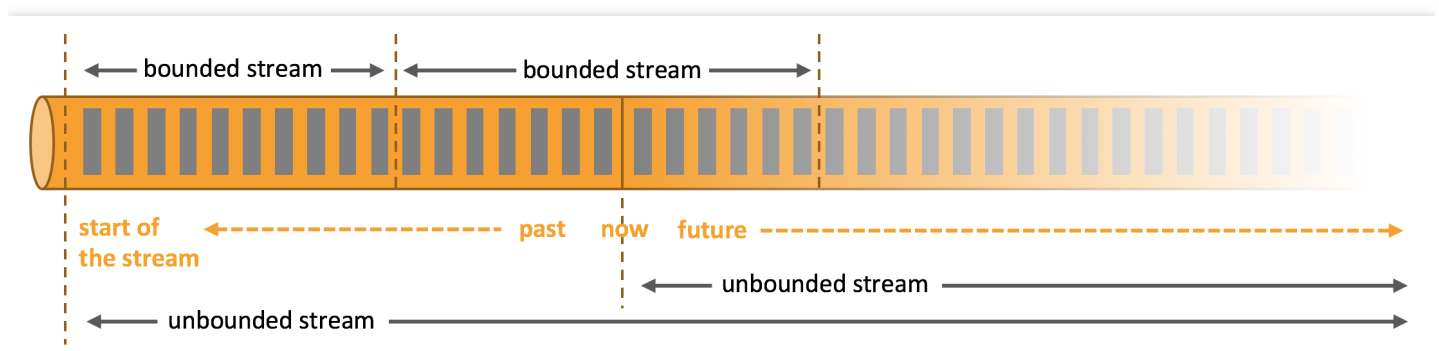
# Best Practices

## Connecting Flink to CKafka

Last updated : 2022-06-17 15:56:31

### Overview

Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.



Apache Flink excels at processing unbounded and bounded data sets. Precise control of time and state enable Flink's runtime to run any kind of application on unbounded streams. Bounded streams are internally processed by algorithms and data structures that are specifically designed for fixed sized data sets, yielding excellent performance.

Apache Flink requires real-time data from various sources (such as Apache Kafka or Kinesis) in order to execute applications. Flink provides special Kafka Connectors for reading and writing data from/to Kafka topics, which offers exactly-once processing semantics.

### Directions

#### Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.
3. On the instance's basic information page, get the instance access address in the **Access Mode** module, which is the `bootstrap-server` required by production and consumption.

**Access Mode?**

VPC Network PLAINTEXT

10.10.10.10 3092 [Delete](#)

## Step 2. Create a topic

1. On the instance's basic information page, select the **Topic Management** tab at the top.
2. On the topic management page, click **Create** to create a topic named `test`. This topic is used as an example below to describe how to consume messages.

ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-lt-...o6 storm_test		1	2	Disabled		2021-08-06 18:12:38	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a> ▾

## Step 3. Add Maven dependencies

Configure `pom.xml` as follows:

```

<!--?xml version="1.0" encoding="UTF-8"?-->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemalocation="http://maven.apache.org/POM/4.0.0 h
ttp://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelversion>4.0.0</modelversion>

<groupid>org.example</groupid>
<artifactid>Test-CKafka</artifactid>
<version>1.0-SNAPSHOT</version>
<dependencies>
<dependency>
<groupid>org.apache.kafka</groupid>
<artifactid>kafka-clients</artifactid>
<version>0.10.2.2</version>
</dependency>
<dependency>
<groupid>org.slf4j</groupid>
<artifactid>slf4j-simple</artifactid>
<version>1.7.25</version>
<scope>compile</scope>
</dependency>
<dependency>
<groupid>org.apache.flink</groupid>

```

```
<artifactid>flink-java</artifactid>
<version>1.6.1</version>
</dependency>
<dependency>
<groupid>org.apache.flink</groupid>
<artifactid>flink-streaming-java_2.11</artifactid>
<version>1.6.1</version>
</dependency>
<dependency>
<groupid>org.apache.flink</groupid>
<artifactid>flink-connector-kafka_2.11</artifactid>
<version>1.7.0</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupid>org.apache.maven.plugins</groupid>
<artifactid>maven-compiler-plugin</artifactid>
<version>3.3</version>
<configuration>
<source>1.8
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

## Step 4. Consume CKafka messages

You can click the tabs below to view the two methods of message consumption and view consumption results in the console or through printed logs.

- Consume via VPC
- Consume via public domain name

```
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import java.util.Properties;

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

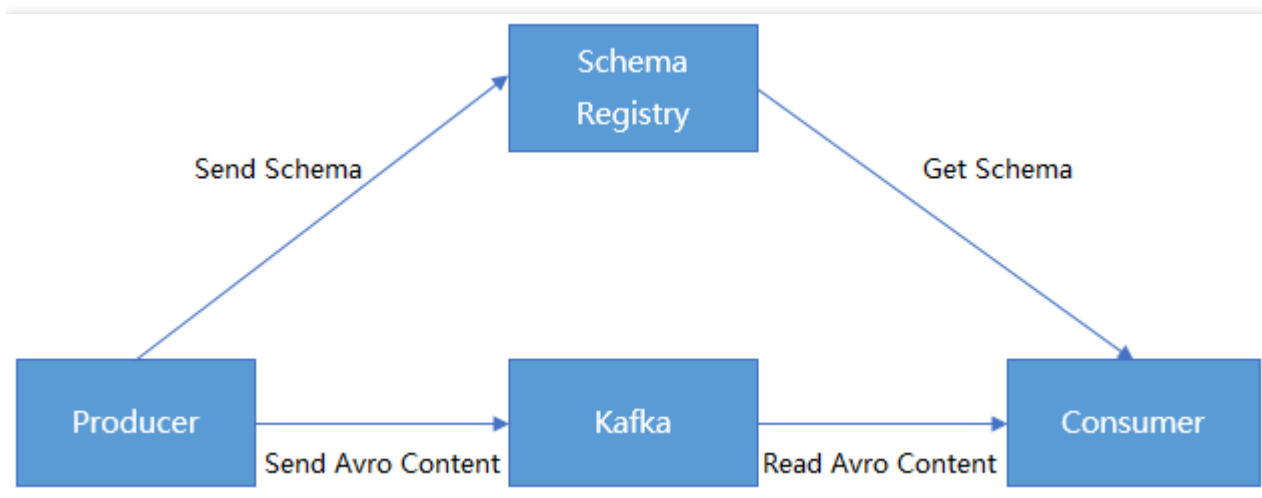
```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
Properties properties = new Properties();
//Domain name address for public network access, i.e., public routing address, which can be obtained in the access mode module of the instance details page.
properties.setProperty("bootstrap.servers", "IP:PORT");
//Consumer group ID.
properties.setProperty("group.id", "testConsumerGroup");
DataStream<string> stream = env
    .addSource(new FlinkKafkaConsumer<>("topicName", new SimpleStringSchema(), properties));
stream.print();
env.execute();
}
```

# Connecting Schema Registry to CKafka

Last updated : 2022-08-11 17:20:49

We can serialize/deserialize classes by using Avro APIs or the Twitter Bijection class library, but the disadvantage of the two methods is that the Kafka record size will multiply as each record must be embedded with a schema. However, the schema is required for reading the records.

CKafka makes it possible for data to share one schema by registering the content of the schema in Confluent Schema Registry. Kafka producers and consumers can implement serialization/deserialization by identifying the schema content in Confluent Schema Registry.



## Prerequisites

- You have downloaded [JDK 8](#).
- You have downloaded [Confluent OSS 4.1.1](#).
- You have created an instance as instructed in [Creating Instance](#).

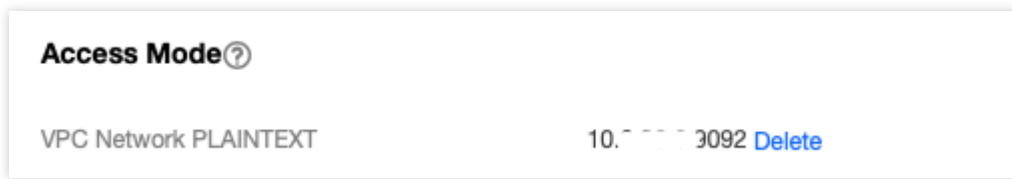
## Directions

### Step 1. Obtain the instance access address and enable automatic topic creation

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.



- On the instance's basic information page, get the instance access address in the **Access Mode** module.



- Enable automatic topic creation in the **Auto-Create Topic** module.

Note :

Automatic topic creation must be enabled as a topic named `schemas` will be automatically created when OSS is started.

## Step 2. Prepare Confluent configurations

- Modify the server address and other information in the OSS configuration file.

The configuration information is as follows:

```
kafkastore.bootstrap.servers=PLAINTEXT://xxxx
kafkastore.topic=schemas
debug=true
```

```
<blockquote class="rno-document-tips rno-document-tips-explain"> <div class="rno-
document-tips-body"> <i class="rno-document-tip-icon"></i> <div class="rno-docume
nt-tip-title">Note</div> <div class="rno-document-tip-desc"><p><code>bootstrap.se
rvers</code>: Access network, which can be copied in the <strong>Network</strong>
column in the <strong>Access Mode</strong> module on the instance details page in
the <a href="https://consoleintl.cloud.tencent.com.cn/ckafka">CKafka console</a>
.<br> </p></div> </div></blockquote>
```

- Run the following command to start Schema Registry.

```
bin/schema-registry-start etc/schema-registry/schema-registry.properties
```

The execution result is as follows:

```
kafkastore.init.timeout.ms = 60000
(io.confluent.kafka.schemaregistry.rest.SchemaRegistryConfig:179)
[2019-07-09 16:33:24,889] INFO Logging initialized @523ms (org.eclipse.jetty.util.log:186)
[2019-07-09 16:33:25,607] INFO Initializing KafkaStore with broker endpoints: PLAINTEXT://172.17.0.1:9092 (io.confluent.kafka.schemaregistry.storage.KafkaStore:103)
[2019-07-09 16:33:26,052] INFO Creating schemas topic _schemas (io.confluent.kafka.schemaregistry.storage.KafkaStore:186)
[2019-07-09 16:33:26,424] INFO Initialized last consumed offset to -1 (io.confluent.kafka.schemaregistry.storage.KafkaStoreReaderThread:138)
[2019-07-09 16:33:26,437] INFO [kafka-store-reader-thread-_schemas]: Starting (io.confluent.kafka.schemaregistry.storage.KafkaStoreReaderThread:66)
[2019-07-09 16:33:27,152] INFO Wait to catch up until the offset of the last message at 0 (io.confluent.kafka.schemaregistry.storage.KafkaStore:277)
[2019-07-09 16:33:27,221] INFO Joining schema registry with Kafka-based coordination (io.confluent.kafka.schemaregistry.storage.KafkaSchemaRegistry:209)
[2019-07-09 16:33:27,316] INFO Finished rebalance with master election result: Assignment{version=1, error=0, master='sr-1-/172.26.0.11-2019-07-09 16:33:27:278-f5aa186c-a6c2-4c93-95dd-172.17.0.1', masterIdentity=version=1,host=VM_0_11-centos,port=8081,scheme=http,masterEligibility=true} (io.confluent.kafka.schemaregistry.masterelector.kafka.KafkaGroupMasterElector:232)
[2019-07-09 16:33:27,336] INFO Wait to catch up until the offset of the last message at 1 (io.confluent.kafka.schemaregistry.storage.KafkaStore:277)
[2019-07-09 16:33:27,458] INFO Adding listener: http://0.0.0.0:8081 (io.confluent.rest.Application:190)
```

### Step 3. Receive/Send messages

Below is the content of the schema file:

```
{
  "type": "record",
  "name": "User",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "age", "type": "int"}
  ]
}
```

#### 1. Register the schema in the topic named `test`.

The script below is an example of registering a schema by calling an API with the `curl` command in the environment deployed in Schema Registry.

```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
--data '{"schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"id\", \"type\": \"int\"}, {\"name\": \"name\", \"type\": \"string\"}, {\"name\": \"age\", \"type\": \"int\"}]}"' \
http://127.0.0.1:8081/subjects/test/versions
```

#### 2. The Kafka producer sends messages.

```
package schemaTest;
import java.util.Properties;
import java.util.Random;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
public class SchemaProduce {
public static final String USER_SCHEMA = "{\"type\": \"record\", \"name\": \"User\", \" +
  \"fields\": [{\"name\": \"id\", \"type\": \"int\"}, \" +
  {\"name\": \"name\", \"type\": \"string\"}, {\"name\": \"age\", \"type\": \"int\"}]}\";
public static void main(String[] args) throws Exception {
Properties props = new Properties();
// Add the access address of the CKafka instance
props.put(\"bootstrap.servers\", \"xx.xx.xx.xx:xxxx\");
props.put(\"key.serializer\", \"org.apache.kafka.common.serialization.StringSerializer\");
// Use the Confluent `KafkaAvroSerializer`
props.put(\"value.serializer\", \"io.confluent.kafka.serializers.KafkaAvroSerializer\");
// Add the schema service address to obtain the schema
props.put(\"schema.registry.url\", \"http://127.0.0.1:8081\");
Producer<string, genericrecord=\"\"> producer = new KafkaProducer<>(props);
Schema.Parser parser = new Schema.Parser();
Schema schema = parser.parse(USER_SCHEMA);
Random rand = new Random();
int id = 0;
while(id < 100) {
id++;
String name = \"name\" + id;
int age = rand.nextInt(40) + 1;
GenericRecord user = new GenericData.Record(schema);
user.put(\"id\", id);
user.put(\"name\", name);
user.put(\"age\", age);
ProducerRecord<string, genericrecord=\"\"> record = new ProducerRecord<>(\"test\", user);
producer.send(record);
Thread.sleep(1000);
}
producer.close();
}
```

```
}
}
```

After running the script for a while, go to the [CKafka console](#), select the **Topic Management** tab on the instance details page, select the topic, and click **More > Message Query** to view the message just sent.

Instance:

Topic:

Query Type:

Partition ID:

Time:

Partition ID	Offset	Timestamp	Operation
0	0	2021-03-02 11:32:15	<a href="#">View Message Details</a>
0	1	2021-03-02 11:33:13	<a href="#">View Message Details</a>

### 3. The Kafka consumer consumes messages.

```
package schemaTest;
import java.util.Collections;
import java.util.Properties;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
public class SchemaProduce {
public static void main(String[] args) throws Exception {
Properties props = new Properties();
props.put("bootstrap.servers", "xx.xx.xx.xx:xxxx"); // Access address of the C
Kafka instance
props.put("group.id", "schema");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDes
erializer");
// Use the Confluent `KafkaAvroDeserializer`
props.put("value.deserializer", "io.confluent.kafka.serializers.KafkaAvroDeser
ializer");
// Add the schema service address to obtain the schema
props.put("schema.registry.url", "http://127.0.0.1:8081");
```

```
KafkaConsumer<string, genericrecord=""> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Collections.singletonList("test"));

try {
while (true) {
ConsumerRecords<string, genericrecord=""> records = consumer.poll(10);
for (ConsumerRecord<string, genericrecord=""> record : records) {
GenericRecord user = record.value();
System.out.println("value = [user.id = " + user.get("id") + ", " + "user.name
= "
+ user.get("name") + ", " + "user.age = " + user.get("age") + "], "
+ "partition = " + record.partition() + ", " + "offset = " + record.offset());
}
}
} finally {
consumer.close();
}
}
}
```

On the **Consumer Group** tab page in the [CKafka console](#), select the consumer group named `schema`, enter the topic name, and click **View Consumer Details** to view the consumption details.

**ckafka-topic-demo / partition-0Monitoring Details** ✕

Real Time
Last 24 hours
Last 7 days
Select Date

Data Comparison

Period: 1 minute(s) ▾

[Refresh](#)

① Note: Max, Min, and Avg are the maximum, minimum, and average values of all points in the current line chart respectively. [Export Data](#)

<b>Current consumption offset</b>		Max: 100	Min: 100	Avg: 100	 
<b>Max offset for current partition</b>		Max: 216	Min: 137	Avg: 145.812	 
<b>Number of unconsumed messages</b>		Max: 116	Min: 37	Avg: 45.812	 
<b>Consumption Speed</b> messages/min		Max: 0 messages/min	Min: 0 messages/min	Avg: 0 messages/min	 

Start the consumer for consumption. Below is a screenshot of the consumption log:

```
value.subject.name.strategy = class io.confluent.kafka.serializers.subject.TopicNameStrategy
key.subject.name.strategy = class io.confluent.kafka.serializers.subject.TopicNameStrategy
2019-07-09 22:07:32.547 INFO [org.apache.kafka.common.utils.AppInfoParser] - Kafka version : 1.1.1
2019-07-09 22:07:32.549 INFO [org.apache.kafka.common.utils.AppInfoParser] - Kafka commitId : 8e67427ffb493498
2019-07-09 22:07:33.357 INFO [org.apache.kafka.clients.Metadata] - Cluster ID: rcrvUkeuStKYE1bbfM1xg
2019-07-09 22:07:33.364 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Discovered group coordinator 172.26.0.8:9095 (id: 2147473628 rack: null)
2019-07-09 22:07:33.366 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Revoking previously assigned partitions []
2019-07-09 22:07:33.366 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] (Re-)joining group
2019-07-09 22:07:33.405 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Successfully joined group with generation 5
2019-07-09 22:07:33.407 INFO [org.apache.kafka.clients.consumer.internals.ConsumerCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Setting newly assigned partitions [test-1, test-0, test-2]
value = [user.id = 2, user.name = name2, user.age = 10], partition = 1, offset = 11384
value = [user.id = 5, user.name = name5, user.age = 29], partition = 1, offset = 11385
value = [user.id = 8, user.name = name8, user.age = 19], partition = 1, offset = 11386
value = [user.id = 11, user.name = name11, user.age = 17], partition = 1, offset = 11387
value = [user.id = 14, user.name = name14, user.age = 20], partition = 1, offset = 11388
value = [user.id = 17, user.name = name17, user.age = 17], partition = 1, offset = 11389
value = [user.id = 20, user.name = name20, user.age = 40], partition = 1, offset = 11390
value = [user.id = 23, user.name = name23, user.age = 29], partition = 1, offset = 11391
value = [user.id = 26, user.name = name26, user.age = 6], partition = 1, offset = 11392
value = [user.id = 29, user.name = name29, user.age = 31], partition = 1, offset = 11393
value = [user.id = 32, user.name = name32, user.age = 1], partition = 1, offset = 11394
value = [user.id = 35, user.name = name35, user.age = 29], partition = 1, offset = 11395
value = [user.id = 38, user.name = name38, user.age = 24], partition = 1, offset = 11396
value = [user.id = 41, user.name = name41, user.age = 2], partition = 1, offset = 11397
value = [user.id = 44, user.name = name44, user.age = 14], partition = 1, offset = 11398
value = [user.id = 47, user.name = name47, user.age = 13], partition = 1, offset = 11399
value = [user.id = 50, user.name = name50, user.age = 29], partition = 1, offset = 11400
value = [user.id = 53, user.name = name53, user.age = 14], partition = 1, offset = 11401
value = [user.id = 56, user.name = name56, user.age = 27], partition = 1, offset = 11402
value = [user.id = 59, user.name = name59, user.age = 26], partition = 1, offset = 11403
value = [user.id = 62, user.name = name62, user.age = 11], partition = 1, offset = 11404
value = [user.id = 65, user.name = name65, user.age = 37], partition = 1, offset = 11405
value = [user.id = 68, user.name = name68, user.age = 17], partition = 1, offset = 11406
value = [user.id = 71, user.name = name71, user.age = 29], partition = 1, offset = 11407
value = [user.id = 74, user.name = name74, user.age = 23], partition = 1, offset = 11408
value = [user.id = 77, user.name = name77, user.age = 14], partition = 1, offset = 11409
value = [user.id = 80, user.name = name80, user.age = 21], partition = 1, offset = 11410
value = [user.id = 83, user.name = name83, user.age = 19], partition = 1, offset = 11411
value = [user.id = 86, user.name = name86, user.age = 20], partition = 1, offset = 11412
value = [user.id = 89, user.name = name89, user.age = 9], partition = 1, offset = 11413
value = [user.id = 92, user.name = name92, user.age = 27], partition = 1, offset = 11414
value = [user.id = 95, user.name = name95, user.age = 17], partition = 1, offset = 11415
value = [user.id = 98, user.name = name98, user.age = 25], partition = 1, offset = 11416
value = [user.id = 3, user.name = name3, user.age = 2], partition = 0, offset = 11446
value = [user.id = 6, user.name = name6, user.age = 9], partition = 0, offset = 11447
```

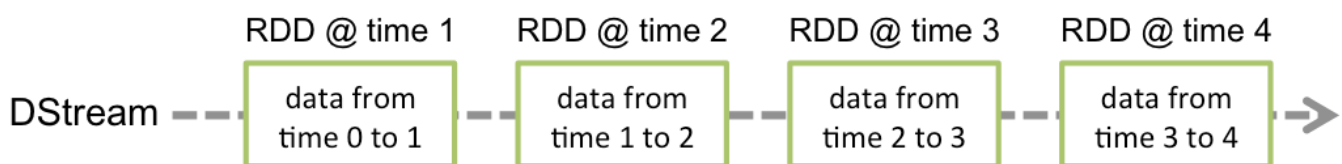
# Connecting Spark Streaming to CKafka

Last updated : 2022-07-06 16:54:26

As an extension of Spark Core, Spark Streaming is used for high-throughput and fault-tolerant processing of continuous data. Currently supported external input sources include Kafka, Flume, HDFS/S3, Kinesis, Twitter, and TCP socket.



Spark Streaming abstracts continuous data into a Discretized Stream (DStream), which consists of a series of continuous resilient distributed datasets (RDDs). Each RDD contains data generated at a certain time interval. Processing DStream with functions is actually processing these RDDs.



When Spark Streaming is used as data input for Kafka, the following stable and experimental Kafka versions are supported:

Kafka Version	spark-streaming-kafka-0.8	spark-streaming-kafka-0.10
Broker Version	0.8.2.1 or later	0.10.0 or later
API Maturity	Deprecated	Stable

Kafka Version	spark-streaming-kafka-0.8	spark-streaming-kafka-0.10
Language Support	Scala, Java, and Python	Scala and Java
Receiver DStream	Yes	No
Direct DStream	Yes	Yes
SSL / TLS Support	No	Yes
Offset Commit API	No	Yes
Dynamic Topic Subscription	No	Yes

Currently, CKafka is compatible with version above 0.9. The Kafka dependency of v0.10.2.1 is used in this practice scenario.

In addition, Spark Streaming in EMR also supports direct connection to CKafka. For more information, see [Connecting Spark Streaming to CKafka](#).

## Directions

### Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.
3. On the instance's basic information page, get the instance access address in the **Access Mode** module, which is the `bootstrap-server` required by production and consumption.

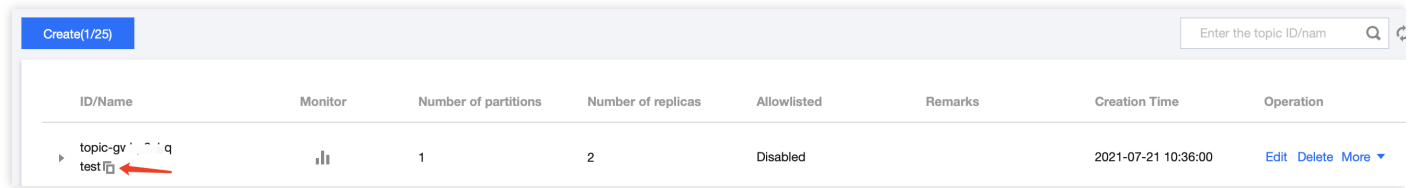
Access Mode <span>?</span>		<a href="#">Add a routing policy</a>	
Access Type	Access Mode	Internet	Operation
Supporting Environment	PLAINTEXT	9.10.10.10:9900	<a href="#">Delete</a>
Public domain name a...	PLAINTEXT	ckafka-lke75x...	<a href="#">Delete</a>


### Step 2. Create a topic

1. On the instance's basic information page, select the **Topic Management** tab at the top.



2. On the topic management page, click **Create** to create a topic named `test`. This topic is used as an example below to describe how to produce and consume messages.



ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-gv test		1	2	Disabled		2021-07-21 10:36:00	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a>

### Step 3. Prepare the CVM environment

#### CentOS 6.8

Package	Version
sbt	0.13.16
Hadoop	2.7.3
Spark	2.1.0
Protobuf	2.5.0
SSH	Installed on CentOS by default
Java	1.8

For specific installation steps, see [Configuring environment](#Configuring environment).

### Step 4. Connect to CKafka

- Producing Messages to CKafka
- Consuming Messages from CKafka

The Kafka dependency of v0.10.2.1 is used here.

1. Add dependencies to `build.sbt` :

```
name := "Producer Example"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % "0.10.2.1"
```

2. Configure `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.StringSeria
  lizer")
  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSer
  ializer")
  val producer = new KafkaProducer[String, String]
  val TOPIC="test" // Specify the topic to produce to
  for(i<- 1 to 50){
    val record = new ProducerRecord(TOPIC, "key", s"hello $i") // Produce a messag
    e whose key is "key" and value is "hello i"
    producer.send(record)
  }
  val record = new ProducerRecord(TOPIC, "key", "the end "+new java.util.Date)
  producer.send(record)
  producer.close() // Disconnect at the end
}
```

For more information on how to use `ProducerRecord`, see [ProducerRecord](#).

## Configuring environment[(id:Configuring environment)]

### Installing sbt

1. Download the sbt package from [sbt's official website](#).
2. After decompression, create an `sbt_run.sh` script with the following content in the sbt directory and add executable permissions:

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

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

3. Run the following command:

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

The display of sbt version indicates a successful installation.

## Installing Protobuf

1. Download an appropriate version of [Protobuf](#).
2. Decompress and enter the directory.

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

You should install gcc-g++ in advance, and the root permission may be required during installation.

3. Log in again and enter the following on the command line:

```
protoc --version
```

4. The display of Protobuf version indicates a successful installation.

## Installing Hadoop

1. Download the required version at [Hadoop's official website](#).
2. Add a Hadoop user.

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

3. Grant admin permissions.

```
visudo
```

4. Add the following in a new line under `root ALL=(ALL) ALL :`

```
hadoop ALL=(ALL) ALL
```

Save and exit.

5. Use Hadoop for operations.

```
su hadoop
```

6. Configure SSH password-free login.

```
cd ~/.ssh/ # If there is no such directory, run `ssh localhost` first  
ssh-keygen -t rsa # There will be prompts. Simply press Enter  
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:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.e16_8.x86_64/
jre
export PATH=$PATH:$JAVA_HOME
```

Modify the corresponding path based on the installation information.

9. Decompress Hadoop and enter the directory.

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

The display of version information indicates a successful installation.

0. Configure the pseudo-distributed mode (so that you can build different forms of clusters as needed).

```
vim /etc/profile
```

Add the following at the end:

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

Modify the corresponding path based on the installation information.

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 the Java path.

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.e16_8.x86_64/
jre
```

4. Format the NameNode.

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

The display of `Exiting with status 0` indicates a success.

5. Start Hadoop.

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

`NameNode`, `DataNode`, and `SecondaryNameNode` processes will exist upon successful startup.

## Installing Spark

Download the required version at [Spark's official website](#).

As Hadoop has already been installed, select `Pre-build with user-provided Apache Hadoop` here.

Note :

This example also uses the `hadoop` user for operations.

1. Decompress and 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 based on the Hadoop installation information.

3. Run the example.

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

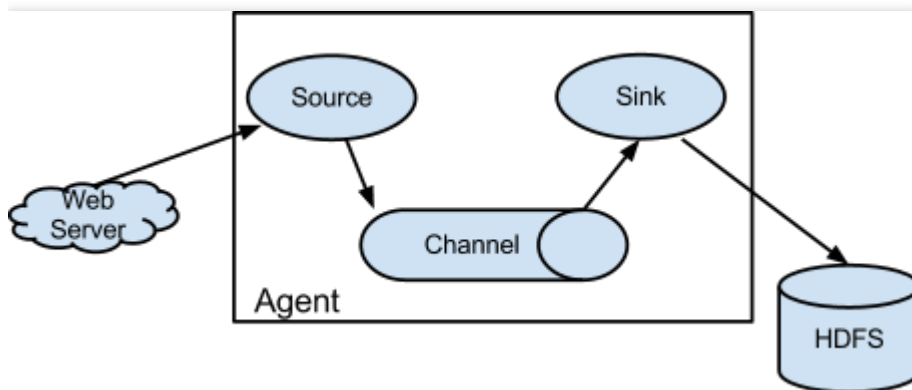
The display of an approximate value of  $\pi$  output by the program indicates a successful installation.

# Connecting Flume to CKafka

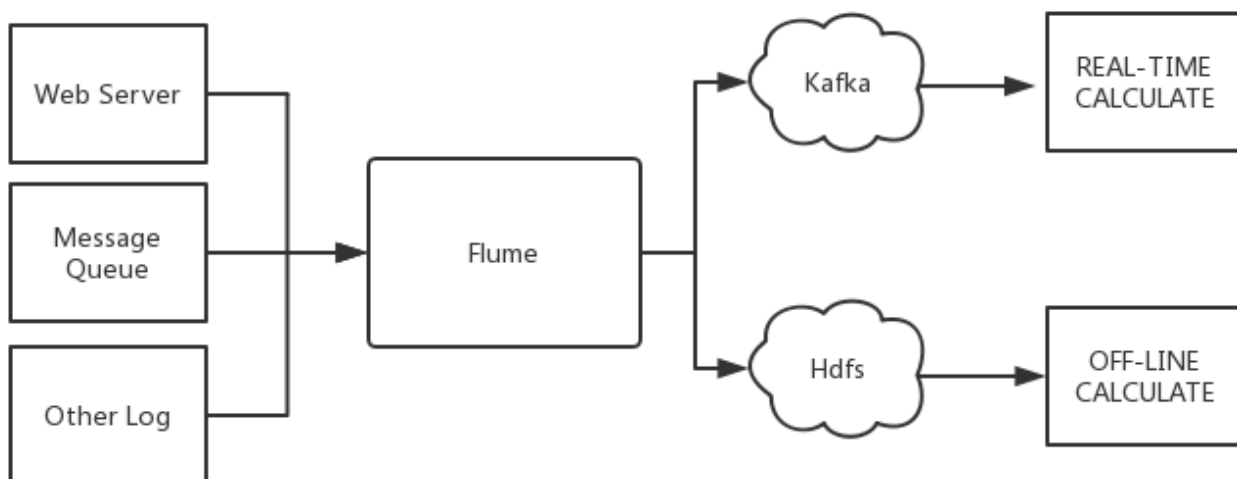
Last updated : 2021-08-10 15:42:04

Apache Flume is a distributed, reliable, and highly available log collection system that supports a wide variety of data sources such as HTTP, log file, JMS, and listening port. It can efficiently collect, aggregate, move, and store massive amounts of log data to a specified storage system like Kafka, distributed file system, and Solr search server.

The structure of Flume is as follows:



Flume uses agents as the smallest independent unit of operation. An agent is a JVM composed of three main components: source, sink, and channel.



## Flume and Kafka

When you store data in a downstream storage module or a computing module such as HDFS or HBase, a lot of complex factors need to be taken into account, such as the number of concurrent writes, system load, and network

delay. As a flexible distributed system, Flume has various APIs and provides customizable pipelines.

In the production process, Kafka can act as a cache when the production and consumption are at different paces. It has a partition structure and uses `append` to append data, which makes it have an excellent throughput. In addition, it has a replication structure, which makes it highly fault-tolerant.

Therefore, Flume and Kafka together can satisfy most requirements in production environments.

## Flume Connection to Open-Source Kafka

### Preparations

- Download [Apache Flume](#) (v1.6.0 or higher is compatible with Kafka).
- Download [Kafka](#) (v0.9.x or higher is required as v0.8 is no longer supported).
- Confirm that Kafka's source and sink components are already in Flume.

### Connection method

Kafka can be used as source or sink to import or export messages.

- Kafka Source
- Kafka Sink

Configure Kafka as the message source, i.e., pulling data from Kafka into a specified sink as a consumer. The main configuration items are as follows:

Configuration Item	Description
channels	Configured channel
type	This must be <code>org.apache.flume.source.kafka.KafkaSource</code>
kafka.bootstrap.servers	Kafka broker server address
kafka.consumer.group.id	ID of the group as Kafka consumer
kafka.topics	Kafka topic as data source
batchSize	Size of each write into channel
batchDurationMillis	Maximum time interval between writes

Sample:



```
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
```

For more information, please visit [Apache Flume's official website](#).

## Flume Connection to CKafka

- Using CKafka as Sink
- Using CKafka as Source

### Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of an instance to enter the instance basic information page.
3. On the instance basic information page, get the instance access address in the **Access Mode** module.

Access Mode <span>?</span>		<a href="#">Add a routing policy</a>	
Access Type	Access Mode	Internet	Operation
Supporting Environment	PLAINTEXT	9.1...:9900	<a href="#">Delete</a>
Public domain name a...	PLAINTEXT	ckafka-lke75x...	<a href="#">Delete</a>

### Step 2. Create a topic

1. On the instance basic information page, select the **Topic Management** tab on the top.

2. On the topic management page, click **Create** to create a topic named `flume_test` .

ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-gp5nb2bm flume_test		1	2	Disabled		2021-08-06 18:08:59	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a>

### Step 3. Configure Flume

1. Download the [Apache Flume toolkit and decompress it](#).
2. Write the configuration file `flume-kafka-sink.properties` . Below is a simple demo (configured in the `conf` folder in the extracted directory). If there is no special requirement, simply replace your own instance IP and topic in the configuration file. The source used in this example is `tail -F flume-test` , which represents the information newly added in the file.

```
# Take kafka as a sink
agentckafka.sources = exectail
agentckafka.channels = memoryChannel
agentckafka.sinks = kafkaSink

# Set source type as needed
agentckafka.sources.exectail.type = exec
agentckafka.sources.exectail.command = tail -F ./flume-test
agentckafka.sources.exectail.batchSize=20
# Set source type as needed
agentckafka.sources.exectail.channels = memoryChannel

# Set the sink type. It is set to kafka here
agentckafka.sinks.kafkaSink.type= org.apache.flume.sink.kafka.KafkaSink
# Set the ip:port provided by Ckafka
agentckafka.sinks.kafkaSink.brokerList= 172.16.16.12:9092
# Set the topic that needs to import data. Create the topic in advance in the console before proceeding here
agentckafka.sinks.kafkaSink.topic= flume test
# Set 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
```

If you have a special source, you can configure it by yourself. The simplest example is used here

Configuration for using CKafka as the sink

Configure instance IP

Configure topic

Configuration for using CKafka as the sink

3. Run the following command to start Flume:

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

4. Write messages to the `flume-test` file. At this time, the messages will be written by Flume to CKafka.

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

5. Start the CKafka client for consumption.

```
./kafka-console-consumer.sh --bootstrap-server xx.xx.xx.xx:xxxx --topic flume_test --from-beginning --new-consumer
```

#### Note

Enter the access address of the CKafka instance just created as `bootstrap-server` and the name of the topic just created as `topic`.

You can see that the messages have been consumed.

```
[root@VM_16_17_centos bin]# ./kafka-console-consumer.sh --bootstrap-server 172.16.16.12:9092 --topic flume_test --from-beginning --new-consumer
ckafka ←
```

# Connecting Kafka Connect to CKafka

Last updated : 2019-09-24 16:42:00

Kafka Connect currently supports two execution modes: standalone and distributed.

## Starting Connect in Standalone Mode

Start Connect in standalone mode by running the following commands:

```
bin/connect-standalone.sh config/connect-standalone.properties connector1.properties [connector2.properties ...]
```

Accessing CKafka is basically the same as accessing the open-source Kafka, except that you need to change `bootstrap.servers` to the IP assigned when the instance is applied for.

## Starting Connect in Distributed Mode

Start Connect in distributed mode by running the following commands:

```
bin/connect-distributed.sh config/connect-distributed.properties
```

In this mode, Kafka Connect stores offsets, configs, and task status information in Kafka topics, which are configured in the following fields in `connect-distributed`:

```
config.storage.topic  
offset.storage.topic  
status.storage.topic
```

These three topics need to be created manually to ensure that their attributes meet the requirements of Connect.

- `config.storage.topic` should have only one partition, multiple replicas, and be in compact mode.
- `offset.storage.topic` should have multiple partitions, multiple replicas, and be in compact mode.
- `status.storage.topic` should have multiple partitions, multiple replicas, and be in compact mode.

Configure `bootstrap.servers` to the IP assigned when the instance is applied for.

Configure `group.id` to identify the Connect cluster, which should be differentiated from the consumer group.

# Connecting Storm to CKafka

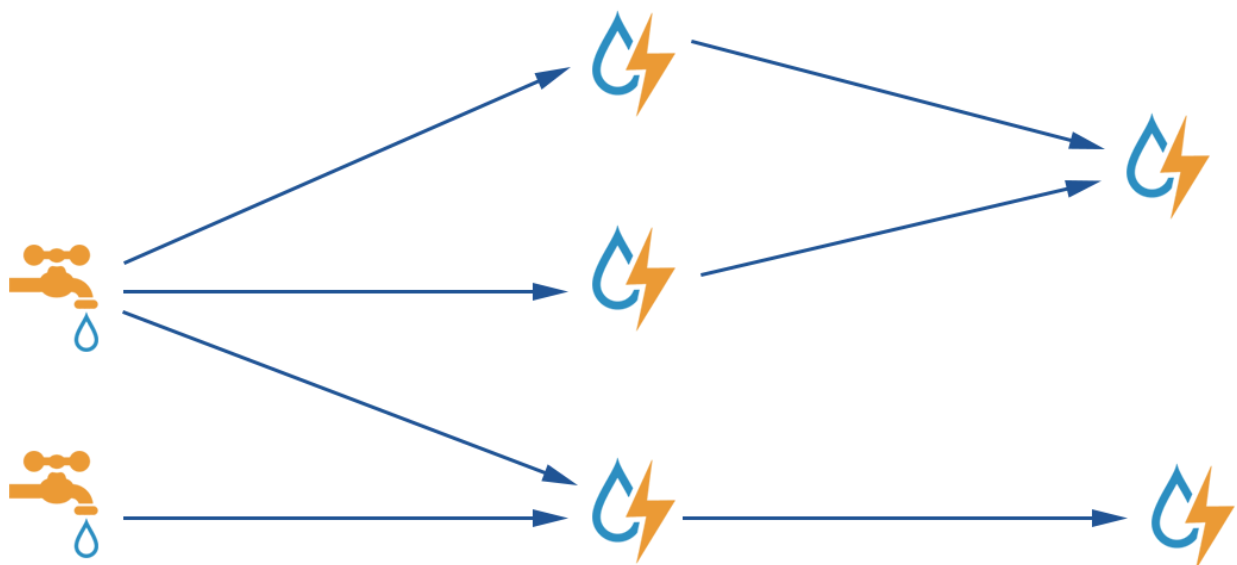
Last updated : 2022-07-06 16:54:27

Storm is a distributed real-time computing framework that can perform stream-based data processing and provide universal distributed RPC calling so as to reduce the delay of event processing down to sub-seconds. It is suitable for real-time data processing scenarios where low delay is required.

## How Storm Works

There are two types of nodes in a Storm cluster: `master node` and `worker node`. The `Nimbus` process runs on the `master node` for resource allocation and status monitoring, and the `Supervisor` process runs on the `worker node` for listening on work tasks and starting the `executor`. The entire Storm cluster relies on `ZooKeeper` for common data storage, cluster status listening, task assignment, etc.

A data processing program submitted to Storm is called a `topology`. The minimum message unit it processes is `tuple` (an array of arbitrary objects). A `topology` consists of `spout` and `bolt`, where `spout` is the source of `tuple`, while `bolt` can subscribe to any `tuple` issued by `spout` or `bolt` for processing.



## Storm with CKafka

Storm can use CKafka as a `spout` to consume data for processing or as a `bolt` to store the processed data for consumption by other components.

## Testing environment

### CentOS 6.8

Package	Version
Maven	3.5.0
Storm	2.1.0
SSH	5.3
Java	1.8

## Prerequisites

- Download and install JDK 8. For detailed directions, see [Java SE Development Kit 8 Downloads](#).
- Download and install Storm. For more information, see [Apache Storm downloads](#).
- You have [created a CKafka instance](#).

## Directions

### Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.
3. On the instance's basic information page, get the instance access address in the **Access Mode** module.

Access Mode <span>?</span>		<a href="#">Add a routing policy</a>	
Access Type	Access Mode	Internet	Operation
Supporting Environment	PLAINTEXT	9.1...:9900	<a href="#">Delete</a>
Public domain name a...	PLAINTEXT	ckafka-lke75x...	<a href="#">Delete</a>

### Step 2. Create a topic

1. On the instance's basic information page, select the **Topic Management** tab at the top.

2. On the topic management page, click **Create** to create a topic.

ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-ll...o6 storm_test		1	2	Disabled		2021-08-06 18:12:38	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a> ▾

### Step 3. Add Maven dependencies

Configure `pom.xml` as follows:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemalocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<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>2.1.0</version>
</dependency>
<dependency>
<groupid>org.apache.storm</groupid>
<artifactid>storm-kafka-client</artifactid>
<version>2.1.0</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>4.12</version>
<scope>test</scope>
</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
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

## Step 4. Produce a message

### Using spout/bolt

Topology code:



```
//TopologyKafkaProducerSpout.java
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.bolt.KafkaBolt;
import org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.bolt.selector.DefaultTopicSelector;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.utils.Utils;

import java.util.Properties;

public class TopologyKafkaProducerSpout {
    // `ip:port` of the CKafka instance applied for
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    // Specify the topic to which to write messages
    private final static String TOPIC = "storm_test";
    public static void main(String[] args) throws Exception {
        // Set producer attributes
        // For functions, visit https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html
        // For attributes, visit http://kafka.apache.org/0102/documentation.html
        Properties properties = new Properties();
        properties.put("bootstrap.servers", BOOTSTRAP_SERVERS);
        properties.put("acks", "1");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        // Create a bolt to be written to Kafka. `fields("key" "message")` is used as the key and message for the produced message by default, which can also be specified in `FieldNameBasedTupleToKafkaMapper()`
        KafkaBolt kafkaBolt = new KafkaBolt()
            .withProducerProperties(properties)
            .withTopicSelector(new DefaultTopicSelector(TOPIC))
            .withTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper());
        TopologyBuilder builder = new TopologyBuilder();
        // A spout class that generates messages in sequence with the output field being `sentence`
        SerialSentenceSpout spout = new SerialSentenceSpout();
        AddMessageKeyBolt bolt = new AddMessageKeyBolt();
        builder.setSpout("kafka-spout", spout, 1);
        // Add the fields required to produce messages to CKafka for the tuple
        builder.setBolt("add-key", bolt, 1).shuffleGrouping("kafka-spout");
        // Write to CKafka
    }
}
```

```
builder.setBolt("sendToKafka", kafkaBolt, 8).shuffleGrouping("add-key");

Config config = new Config();
if (args != null && args.length > 0) {
    // Cluster mode, which is used to package a jar file and run it in Storm
    config.setNumWorkers(1);
    StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.createTopology());
} else {
    // Local mode
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("test", config, builder.createTopology());
    Utils.sleep(10000);
    cluster.killTopology("test");
    cluster.shutdown();
}

}
```

Create a spout class that generates messages in sequence:

```
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.Map;
import java.util.UUID;

public class SerialSentenceSpout extends BaseRichSpout {

    private SpoutOutputCollector spoutOutputCollector;

    @Override
    public void open(Map map, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollector) {
        this.spoutOutputCollector = spoutOutputCollector;
    }

    @Override
    public void nextTuple() {
        Utils.sleep(1000);
    }
}
```

```
// Produce a `UUID` string and send it to the next component
spoutOutputCollector.emit(new Values(UUID.randomUUID().toString()));
}

@Override
public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
    outputFieldsDeclarer.declare(new Fields("sentence"));
}
}
```

Add `key` and `message` fields to the `tuple`. If `key` is null, the produced messages will be evenly allocated to each partition. If a key is specified, the messages will be hashed to specific partitions based on the key value:

```
//AddMessageKeyBolt.java
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class AddMessageKeyBolt extends BaseBasicBolt {

    @Override
    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        // Take out the first field value
        String messae = tuple.getString(0);
        // System.out.println(messae);
        // Send to the next component
        basicOutputCollector.emit(new Values(null, messae));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        // Create a schema to send to the next component
        outputFieldsDeclarer.declare(new Fields("key", "message"));
    }
}
```

## Using trident

Use the trident class to generate a topology

```
//TopologyKafkaProducerTrident.java
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
```

```
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.trident.TridentKafkaStateFactory;
import org.apache.storm.kafka.trident.TridentKafkaStateUpdater;
import org.apache.storm.kafka.trident.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.trident.selector.DefaultTopicSelector;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;
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.util.Properties;

public class TopologyKafkaProducerTrident {
    // `ip:port` of the CKafka instance applied for
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    // Specify the topic to which to write messages
    private final static String TOPIC = "storm_test";
    public static void main(String[] args) throws Exception {
        // Set producer attributes
        // For functions, visit https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html
        // For attributes, visit http://kafka.apache.org/0102/documentation.html
        Properties properties = new Properties();
        properties.put("bootstrap.servers", BOOTSTRAP_SERVERS);
        properties.put("acks", "1");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        // Set the trident
        TridentKafkaStateFactory stateFactory = new TridentKafkaStateFactory()
            .withProducerProperties(properties)
            .withKafkaTopicSelector(new DefaultTopicSelector(TOPIC))
        // Set to use `fields("key", "value")` as the written message, which doesn't have a default value as `FieldNameBasedTupleToKafkaMapper` does
            .withTridentTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper("key", "value"));
        TridentTopology builder = new TridentTopology();
        // A spout that generates messages in batches with the output field being `sentence`
        builder.newStream("kafka-spout", new TridentSerialSentenceSpout(5))
            .each(new Fields("sentence"), new AddMessageKey(), new Fields("key", "value"))
            .partitionPersist(stateFactory, new Fields("key", "value"), new TridentKafkaStateUpdater(), new Fields());
    }
}
```

```
Config config = new Config();
if (args != null && args.length > 0) {
    // Cluster mode, which is used to package a jar file and run it in Storm
    config.setNumWorkers(1);
    StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.build());
} else {
    // Local mode
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("test", config, builder.build());
    Utils.sleep(10000);
    cluster.killTopology("test");
    cluster.shutdown();
}

private static class AddMessageKey extends BaseFunction {

    @Override
    public void execute(TridentTuple tridentTuple, TridentCollector tridentCollector) {
        // Take out the first field value
        String message = tridentTuple.getString(0);
        //System.out.println(message);
        // Send to the next component
        //tridentCollector.emit(new Values(Integer.toString(message.hashCode()), message));
        tridentCollector.emit(new Values(null, message));
    }
}
}
```

Create a spout class that generates messages in batches:

```
//TridentSerialSentenceSpout.java
import org.apache.storm.Config;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.spout.IBatchSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.Map;
import java.util.UUID;
```

```
public class TridentSerialSentenceSpout implements IBatchSpout {

    private final int batchCount;

    public TridentSerialSentenceSpout(int batchCount) {
        this.batchCount = batchCount;
    }

    @Override
    public void open(Map map, TopologyContext topologyContext) {

    }

    @Override
    public void emitBatch(long l, TridentCollector tridentCollector) {
        Utils.sleep(1000);
        for(int i = 0; i < batchCount; i++){
            tridentCollector.emit(new Values(UUID.randomUUID().toString()));
        }
    }

    @Override
    public void ack(long l) {

    }

    @Override
    public void close() {

    }

    @Override
    public Map<string, object=""> getComponentConfiguration() {
        Config conf = new Config();
        conf.setMaxTaskParallelism(1);
        return conf;
    }

    @Override
    public Fields getOutputFields() {
        return new Fields("sentence");
    }
}
```

## Step 5. Consume the message

## Using spout/bolt

```
//TopologyKafkaConsumerSpout.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.spout.*;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.HashMap;
import java.util.Map;

import static org.apache.storm.kafka.spout.FirstPollOffsetStrategy.LATEST;

public class TopologyKafkaConsumerSpout {
    // `ip:port` of the CKafka instance applied for
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    // Specify the topic to which to write messages
    private final static String TOPIC = "storm_test";

    public static void main(String[] args) throws Exception {
        // Set a retry policy
        KafkaSpoutRetryService kafkaSpoutRetryService = new KafkaSpoutRetryExponentialBackoff(
            KafkaSpoutRetryExponentialBackoff.TimeInterval.microSeconds(500),
            KafkaSpoutRetryExponentialBackoff.TimeInterval.milliSeconds(2),
            Integer.MAX_VALUE,
            KafkaSpoutRetryExponentialBackoff.TimeInterval.seconds(10)
        );
        ByTopicRecordTranslator<string, string=""> trans = new ByTopicRecordTranslator<>
        (
            (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
            new Fields("topic", "partition", "offset", "key", "value"));
        // Set consumer parameters
        // For functions, visit http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spout/KafkaSpoutConfig.Builder.html
        // For parameters, visit http://kafka.apache.org/0102/documentation.html
        KafkaSpoutConfig spoutConfig = KafkaSpoutConfig.builder(BOOTSTRAP_SERVERS, TOPI
```

```
C)
.setProp(new HashMap<string, object="">(){
    put(ConsumerConfig.GROUP_ID_CONFIG, "test-group1"); // Set the group
    put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000"); // Set the session timeo
    ut period
    put(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000"); // Set the request timeo
    ut period
    })
.setOffsetCommitPeriodMs(10_000) // Set the automatic confirmation period
.setFirstPollOffsetStrategy(LATEST) // Set to pull the latest message
.setRetry(kafkaSpoutRetryService)
.setRecordTranslator(trans)
.build();

TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("kafka-spout", new KafkaSpout(spoutConfig), 1);
builder.setBolt("bolt", new BaseRichBolt(){
    private OutputCollector outputCollector;
    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

    }

    @Override
    public void prepare(Map map, TopologyContext topologyContext, OutputCollector ou
    tputCollector) {
        this.outputCollector = outputCollector;
    }

    @Override
    public void execute(Tuple tuple) {
        System.out.println(tuple.getStringByField("value"));
        outputCollector.ack(tuple);
    }
}, 1).shuffleGrouping("kafka-spout");

Config config = new Config();
config.setMaxSpoutPending(20);
if (args != null && args.length > 0) {
    config.setNumWorkers(3);
    StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.createTopo
    logy());
}
else {
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("test", config, builder.createTopology());
    Utils.sleep(20000);
}
```



```
cluster.killTopology("test");
cluster.shutdown();
}
}
}
```

## Using trident

```
//TopologyKafkaConsumerTrident.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.generated.StormTopology;
import org.apache.storm.kafka.spout.ByTopicRecordTranslator;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutConfig;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutOpaque;
import org.apache.storm.trident.Stream;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;
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.util.HashMap;

import static org.apache.storm.kafka.spout.FirstPollOffsetStrategy.LATEST;

public class TopologyKafkaConsumerTrident {
    // `ip:port` of the CKafka instance applied for
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    // Specify the topic to which to write messages
    private final static String TOPIC = "storm_test";

    public static void main(String[] args) throws Exception {
        ByTopicRecordTranslator<string, string=""> trans = new ByTopicRecordTranslator<>
        (
            (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
            new Fields("topic", "partition", "offset", "key", "value"));
        // Set consumer parameters
        // For functions, visit http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spout/KafkaSpoutConfig.Builder.html
        // For parameters, visit http://kafka.apache.org/0102/documentation.html
    }
}
```

```
KafkaTridentSpoutConfig spoutConfig = KafkaTridentSpoutConfig.builder(BOOTSTRAP_
SERVERS, TOPIC)
.setProp(new HashMap<string, object="">()){{
put(ConsumerConfig.GROUP_ID_CONFIG, "test-group1"); // Set the group
put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true"); // Set automatic confirma
tion
put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000"); // Set the session timeo
ut period
put(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000"); // Set the request timeo
ut period
}})
.setFirstPollOffsetStrategy(LATEST) // Set to pull the latest message
.setRecordTranslator(trans)
.build();

TridentTopology builder = new TridentTopology();
// Stream spoutStream = builder.newStream("spout", new KafkaTridentSpoutTransact
ional(spoutConfig)); // Transaction type
Stream spoutStream = builder.newStream("spout", new KafkaTridentSpoutOpaque(spou
tConfig));
spoutStream.each(spoutStream.getOutputFields(), new BaseFunction(){
@Override
public void execute(TridentTuple tridentTuple, TridentCollector tridentCollecto
r) {
System.out.println(tridentTuple.getStringByField("value"));
tridentCollector.emit(new Values(tridentTuple.getStringByField("value")));
}
}, new Fields("message"));

Config conf = new Config();
conf.setMaxSpoutPending(20);conf.setNumWorkers(1);
if (args != null && args.length > 0) {
conf.setNumWorkers(3);
StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.build());
}
else {
StormTopology stormTopology = builder.build();
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("test", conf, stormTopology);
Utils.sleep(10000);
cluster.killTopology("test");
cluster.shutdown();stormTopology.clear();
}
}
}
```

## Step 6. Submit Storm

After being compiled with `mvn package` , Storm can be submitted to the local cluster for debugging or submitted to the production cluster for running.

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

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

# Connecting Logstash to CKafka

Last updated : 2022-08-11 17:23:38

Logstash is an open-source log processing tool that can be used to collect data from multiple sources, filters it, and then stores it for other uses.

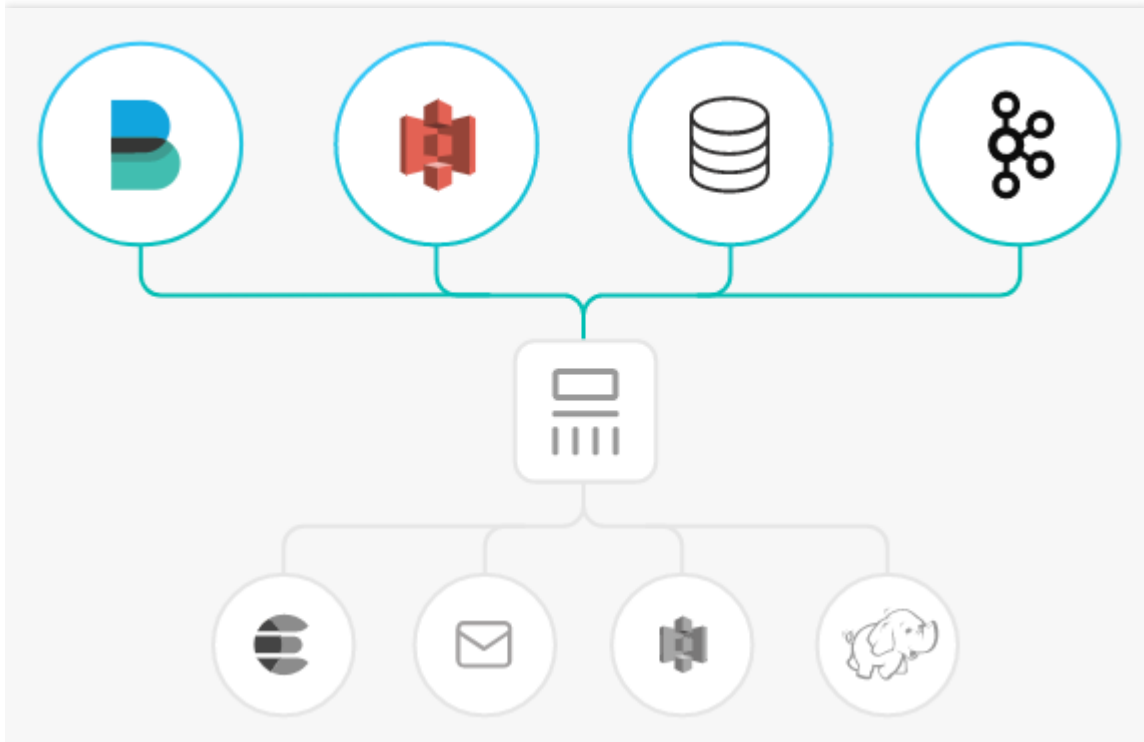
Logstash is highly flexible and has powerful syntax analysis capabilities. With a variety of plugins, it supports multiple types of inputs and outputs. In addition, as a horizontally scalable data pipeline, it has powerful log collection and retrieval features that work with Elasticsearch and Kibana.

## How Logstash Works

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

1. Inputs: Collect data from multiple sources like file, syslog, redis, and beats.
2. Filters: Modify and filter the collected data. Filters are intermediate processing components in the Logstash data pipeline. They can modify events based on specific conditions. Some commonly used filters are grok, mutate, drop, and clone.
3. Outputs: Transfer the processed data to other destinations. An event can be transferred to multiple outputs, and the event ends when the transfer is completed. Elasticsearch is the most commonly-used output.

In addition, Logstash supports encoding and decoding data, so you can specify data formats on the input and output ends.

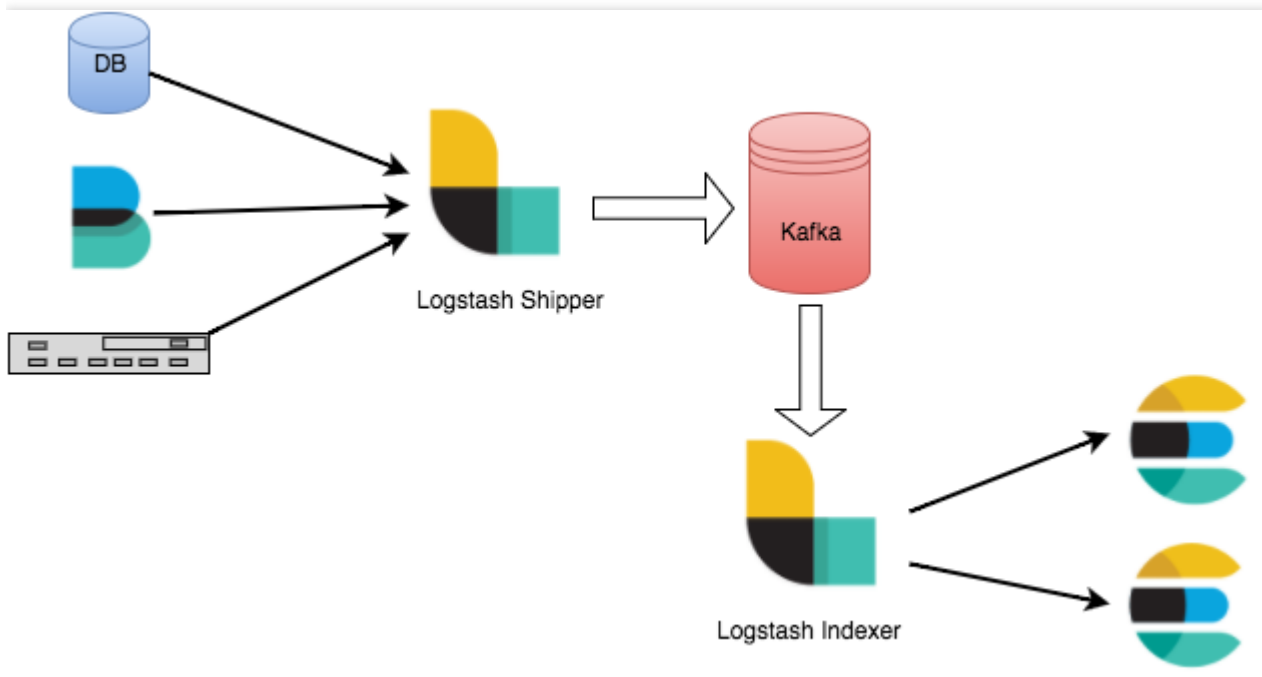


## Strengths of Connecting Logstash to Kafka

- Data can be asynchronously processed to prevent traffic spikes.
- Components are decoupled, so when an exception occurs in Elasticsearch, the upstream work will not be affected.

Note :

Logstash consumes resources when processing data. If you deploy Logstash on a production server, the performance of the server may be affected.



## Directions

### Preparations

- Download and install Logstash as instructed in [Installing Logstash](#).
- Download and install JDK 8 as instructed in [Java SE Development Kit 8u341](#).
- Create a CKafka instance as instructed in [Creating Instance](#).

### Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.

3. On the instance's basic information page, get the instance access address in the **Access Mode** module.

Access Mode <span>?</span>		<a href="#">Add a routing policy</a>	
Access Type	Access Mode	Internet	Operation
Supporting Environment	PLAINTEXT	9.1...:9900	<a href="#">Delete</a>
Public domain name a...	PLAINTEXT	ckafka-lke75x...	<a href="#">Delete</a>

## Step 2. Create a topic

1. On the instance's basic information page, select the **Topic Management** tab at the top.
2. On the topic management page, click **Create** to create a topic named `logstash_test`.

ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-9...c0 logstash_test		1	2	Disabled		2021-08-06 18:10:25	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a>

## Step 3. Connect to CKafka

Note :

You can click the following tabs to view the detailed directions for using CKafka as `inputs` or `outputs`.

- Connecting as inputs
- Connecting as outputs

1. Run `bin/logstash-plugin list` to check whether `logstash-input-kafka` is included in the supported plugins.

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

2. Write the configuration file `input.conf` in the `.bin/` directory.

In the following example, Kafka is used as the data source, and the standard output is taken as the data destination.

```
input {
  kafka {
    bootstrap_servers => "xx.xx.xx.xx:xxxx" // CKafka instance access address
    group_id => "logstash_group" // CKafka group ID
    topics => ["logstash_test"] // CKafka topic name
    consumer_threads => 3 // Number of consumer threads, which is generally the same as the number of CKafka partitions
    auto_offset_reset => "earliest"
  }
}

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

3. Run the following command to start Logstash and consume messages.

```
./logstash -f input.conf
```

The returned result is as follows:

```
[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"
}
```

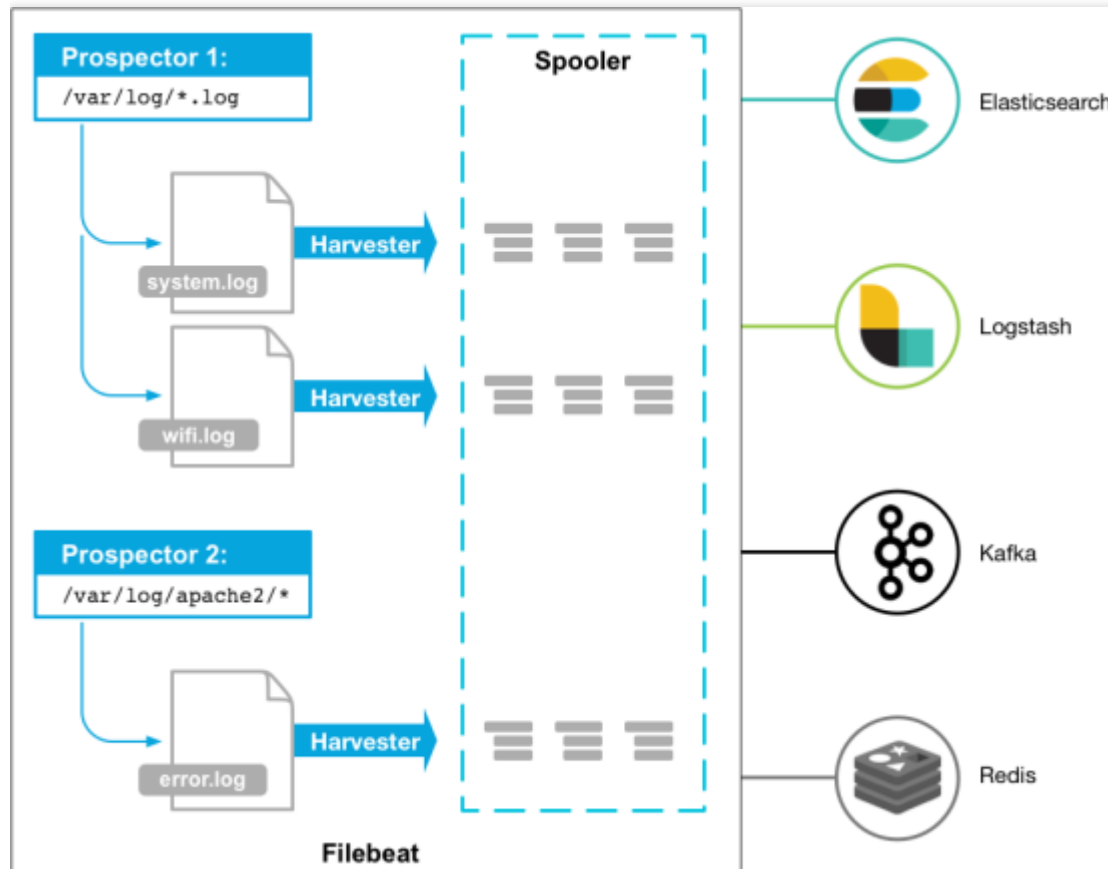
You can see that the data in the topic above has been consumed now.



# Connecting Filebeat to CKafka

Last updated : 2022-11-29 11:38:57

[Beats](#) platform hosts various single-purpose data shippers that can be used as lightweight agents after installation to send collected data from hundreds or thousands of machines to the target.



Beats offers a wide variety of shippers, and you can download the one which best suits your needs. This document uses Filebeat, a lightweight log shipper, as an example to describe how to connect Filebeat to CKafka and handle common problems that may occur after the connection.

## Prerequisites

- You have downloaded and installed Filebeat. For more information, see [Installing Logstash](#).
- You have downloaded and installed JDK 8. For more information, see [Java Downloads](#).
- You have created a CKafka instance. For more information, see [Creating Instance](#).

## Directions

## Step 1. Get the CKafka instance access address

1. Log in to the [CKafka console](#).
2. Select **Instance List** on the left sidebar and click the **ID** of the target instance to enter its basic information page.
3. On the instance's basic information page, get the instance access address in the **Access Mode** module.

Access Mode <span>?</span>		<a href="#">Add a routing policy</a>	
Access Type	Access Mode	Internet	Operation
Supporting Environment	PLAINTEXT	9.10.10.0:9900	<a href="#">Delete</a>
Public domain name a...	PLAINTEXT	ckafka-lke75x...	<a href="#">Delete</a>

## Step 2. Create a topic

1. On the instance's basic information page, select the **Topic Management** tab at the top.
2. On the topic management page, click **Create** to create a topic named `test`.

ID/Name	Monitor	Number of partitions	Number of replicas	Allowlisted	Remarks	Creation Time	Operation
topic-gv... test		1	2	Disabled		2021-07-21 10:36:00	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">More</a>

## Step 3. Prepare the configuration file

Enter the installation directory of Filebeat and create the configuration monitoring file `filebeat.yml`.

```

#===== For versions after Filebeat 7.x, change `filebeat.prospectors` to `file
beat.inputs` =====
filebeat.prospectors:

- input_type: log

# This is the path to the monitoring file.
paths:
- /var/log/messages

#===== Outputs =====

```

```
#----- kafka -----
output.kafka:
version:0.10.2 // Set the value to the open-source version of the CKafka instance.
# Set to the access address of the CKafka instance
hosts: ["xx.xx.xx.xx:xxxx"]
# Set the name of the target topic
topic: 'test'
partition.round_robin:
reachable_only: false

required_acks: 1
compression: none
max_message_bytes: 1000000

# The following parameters need to be configured for SASL. If SASL is not required, skip them.
username: "yourinstance#yourusername" // `username` should consist of the instance ID and username
password: "yourpassword"
```

## Step 4. Send a message in Filebeat

1. Run the following command to start the client:

```
sudo ./filebeat -e -c filebeat.yml
```

2. Add data to the monitoring file (for example: `testlog` ).

```
echo ckafka1 >> testlog
echo ckafka2 >> testlog
echo ckafka3 >> testlog
```

3. Start the consumer to consume the corresponding topic and get the following data.

```
{"@timestamp":"2017-09-29T10:01:27.936Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka1","offset":500,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
{"@timestamp":"2017-09-29T10:01:30.936Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka2","offset":508,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
```

```
{"@timestamp":"2017-09-29T10:01:33.937Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka3","offset":516,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
```

## SASL/PLAINTEXT mode

If you want to configure SASL/PLAINTEXT, you need to set the username and password under the Kafka configuration.

```
# The following parameters need to be configured for SASL. If SASL is not required, skip them.
username: "yourinstance#yourusername" // `username` should consist of the instance ID and username
password: "yourpassword"
```

## FAQs

The Filebeat log file (default path: `/var/log/filebeat/filebeat` ) contains a large number of INFO logs as follows:

```
2019-03-20T08:55:02.198+0800 INFO kafka/log.go:53 producer/broker/544 starting up
2019-03-20T08:55:02.198+0800 INFO kafka/log.go:53 producer/broker/544 state change to [open] on wp-news-filebeat/4
2019-03-20T08:55:02.198+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/4 selected broker 544
2019-03-20T08:55:02.198+0800 INFO kafka/log.go:53 producer/broker/478 state change to [closing] because EOF
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 Closed connection to broker bitar1d12:9092
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/5 state change to [retrying-3]
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/4 state change to [flushing-3]
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/5 abandoning broker 478
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/2 state change to [retrying-2]
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat/2 abandoning broker 541
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/leader/wp-news-filebeat
```

```
t/3 state change to [retrying-2]
2019-03-20T08:55:02.199+0800 INFO kafka/log.go:53 producer/broker/478 shut down
```

This problem may be related to the Filebeat version. Products in the Elastic family are updated frequently, and major version incompatibility problems often occur.

For example, v6.5.x supports Kafka v0.9, v0.10, v1.1.0, and v2.0.0 by default, while v5.6.x supports Kafka v0.8.2.0 by default.

Check the version configuration in the configuration file:

```
output.kafka:
  version:0.10.2 // Set the value to the open-source version of the CKafka instance.
```

## Notes and Precautions

- When data is sent to CKafka, `compression.codec` cannot be set.
- Gzip compression is not supported by default. To use it, [submit a ticket](#).  
As Gzip compression causes high CPU consumption, if it is used, all messages will become `InValid`.
- The program cannot run properly when the LZ4 compression method is used. Possible causes include:  
The message format is incorrect. The default message version of CKafka is v0.10.2. You need to use the message format v1.
- Setting method for SDK varies by Kafka client. You can query the setting method in the open-source community (e.g., [C/C++ Client Descriptions](#)) to set the version of the message format.

# Multi-AZ Deployment

Last updated : 2022-07-06 16:54:27

## Multi-AZ Deployment of CKafka

CKafka Pro Edition supports multi-AZ deployment. When you purchase a CKafka instance in a region that has three or more AZs, you can select up to three of them for multi-AZ deployment. Partition replicas of this instance will be forcibly distributed across the nodes in the three AZs, which enables your instance to provide services uninterruptedly when one AZ becomes unavailable.

Note :

Only the Pro Edition supports multi-AZ deployment.

### How multi-AZ deployment of CKafka works

The multi-AZ deployment feature of CKafka involves three layers: network, data, and control.

#### Network layer

CKafka exposes a VIP to the client. After the client is connected to the VIP, it will get the metadata information (which is generally addresses mapped one to one at different ports on the same VIP) of the topic partitions .

This VIP can fail over to another AZ at any time. When an AZ becomes unavailable, the VIP will automatically shift to another AZ in the same region, thus achieving multi-AZ disaster recovery.

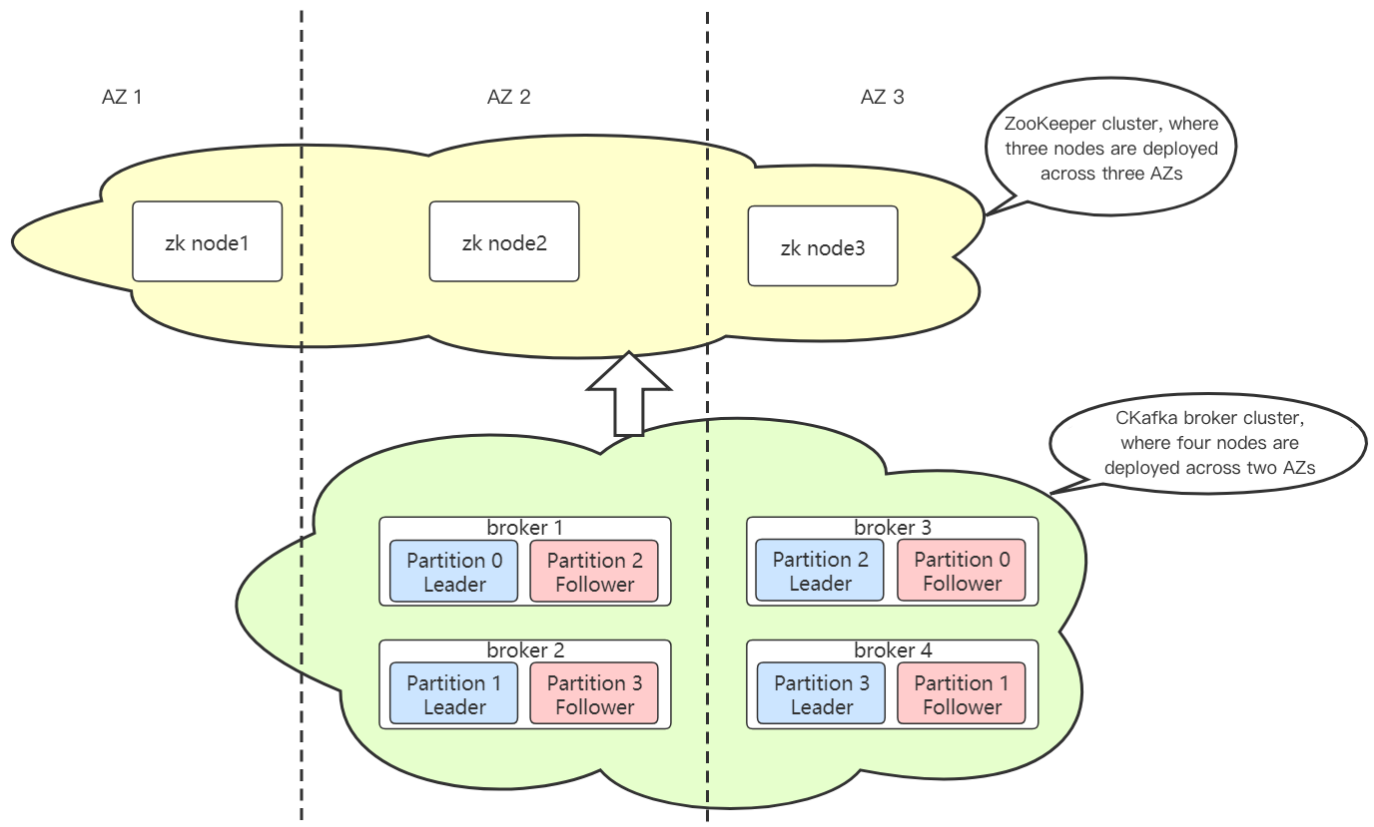
#### Data layer

The data layer of CKafka is deployed in the same distributed way as Kafka; in other words, multiple data replicas are distributed on different broker nodes which are deployed in different AZs. When a partition is processed, there will be a leader-follower relationship between such nodes. When the leader goes offline due to an exception, the cluster controller will elect a new partition leader to handle the requests for the partition.

For the client, after an AZ becomes unavailable due to an exception, if the leader of a topic partition is located on a broker node in the AZ, the originally established link will time out or be closed. After an exception occurs on the leader node of the partition, the controller will elect a new leader node to provide services, and if the controller node is also abnormal, the remaining nodes will run for a new controller node. The [leader switch](#) can be completed within seconds (proportional to the number of nodes in the cluster and the size of the metadata). The client will periodically refresh the metadata of the topic partitions and link the new leader node for production and consumption.

## Control layer

The control layer of CKafka applies the same technical scheme as Kafka. It relies on ZooKeeper to manage service discovery and cluster controller election for broker nodes. **The ZK nodes in the ZooKeeper cluster of a CKafka instance that supports multi-AZ deployment are deployed in three AZs (or data centers).** In this way, even if the ZK nodes in any AZ fail and disconnect, the entire ZooKeeper cluster can still provide services.



## Pros and Cons of Multi-AZ Deployment

### Pros

Multi-AZ deployment can significantly improve the disaster recovery capability of the cluster. When a single AZ experiences force majeure events such as network instability and restart after power outage, the client can resume message production and consumption after waiting a short while to reconnect.

### Cons

In multi-AZ deployment mode, as partition replicas are distributed in multiple AZs, message replication across AZs experiences an additional network latency compared with message replication in one single AZ, which directly affects

the client write time for production (the client `ACK` parameter is greater than 1 or equal to -1 or all). Currently, the network latency across AZs in major regions such as Guangzhou, Shanghai, and Beijing typically ranges between 10 ms and 40 ms.

## Multi-AZ Deployment Scenarios

### Unavailability of one single AZ

After a single AZ becomes unavailable, as explained above, the client will disconnect, reconnect, and then provide services normally after reconnection.

As multi-AZ deployment is currently not supported for the management of API services, after a single AZ becomes unavailable, you may not be able to create topics, configure ACL policies, or view monitoring data in the console, but this will not affect the production and consumption of existing businesses.

### Network isolation between two AZs

If network isolation occurs between two AZs (that is, they cannot communicate), a cluster "split-brain" event may occur; in other words, the nodes in both AZs provide services, but data written in one of the AZs will be treated as dirty data after the cluster is recovered.

Suppose network isolation occurs between the cluster controller node and a ZK node in the ZooKeeper cluster and other nodes, other nodes will run for a new controller (which can be successfully elected since most nodes in the ZooKeeper cluster communicate normally over the network), but the isolated controller still regards itself as the controller node. In this case, a "split-brain" event occurs in the cluster.

At this time, the client writes need to be considered on a case by case basis. For example, when the client's `ACK` is set to `-1` or `all` and the number of replicas is two, if the cluster contains three nodes, they will be distributed in a 2:1 ratio after the split-brain event occurs. In this case, an error will be reported when the original leader writes to the partitions in the AZ with one node, while writes in the other AZ will succeed. However, if the number of replicas is three and `ACK = -1` or `ACK = all` is configured, writes on neither AZ will succeed. Therefore, you need to take further actions based on the specific parameter configuration.

After the cluster network is back to normal, the client can resume production and consumption without performing any operations. As the server will normalize the data again, the data on one of the split nodes will be truncated directly, which, however, will not result in data loss in case of multi-replica multi-AZ data storage.

## Directions

### Selecting multiple AZs for instance



1. Log in to the [CKafka console](#).
2. Click **Instance List** on the left sidebar and click **Create** to enter the purchase page.
3. On the instance purchase page, set the configuration information for purchase based on your actual needs.
  - Billing Mode: Monthly subscription
  - Specs Type: Select the Standard or Pro edition based on your business needs.
  - Kafka Version: Select a Kafka version based on your business needs. For more information, see [Suggestions for CKafka Edition Selection](#).
  - Region: Select a region close to the resource for client deployment.
  - AZ: Select an AZ based on your actual needs.
    - Standard Edition: This edition does not support multi-AZ deployment.
    - Pro Edition: If the current region supports multi-AZ deployment, you can select up to three AZs for deployment. For more information on multi-AZ deployment, see [Multi-AZ Deployment](#).
  - Product Specification: Select a model based on the peak bandwidth and disk capacity.
  - Message Retention Period: Select a value between 24 and 2,160 hours.

When the disk capacity is insufficient (that is, the disk utilization reaches 90%), old messages will be deleted in advance to ensure the service availability.
  - VPC: If you need to access other VPCs, you can modify the routing access rules as instructed in [Adding Routing Policy](#).
  - Tag: It is optional. For more information, see [Tag Overview](#).
  - Instance Name: When purchasing multiple instances, you can use the features of automatically incrementing the instance suffix number and specifying a pattern string. For detailed directions, see [Naming with Consecutive Numeric Suffixes or Designated Pattern String](#).
4. Click **Buy Now** to complete the instance creation process.

# Production and Consumption

Last updated : 2022-03-28 17:47:17

This document describes the best practices of message production and consumption in CKafka to help you reduce errors in message consumption.

## Message Production

### Recommendations for topic use

- Configuration requirements: It is recommended to use 3 replicas and in-sync replication. The minimum number of in-sync replicas should be 2, and the number of in-sync replicas cannot be equal to the number of topic replicas; otherwise, the failure of 1 replica will lead to the inability to produce messages.
- Creation method: You can choose whether to enable **Auto-Create Topic** in CKafka. After it is enabled, a topic with 3 partitions and 2 replicas will be automatically created when a topic that has not been created is produced or consumed in.

### Retry upon failure

In a distributed environment, due to network or other issues, messages may occasionally fail to be sent. This is probably because the message actually has been sent successfully but the ACK mechanism failed, or because the message indeed has not been sent successfully.

You can set the following retry parameters based on your business needs:

Parameter	Description
<code>retries</code>	Number of retries, which is <code>3</code> by default. You can consider setting it to <code>Integer.MAX_VALUE</code> (valid and maximum) for applications that have zero tolerance for data loss.
<code>retry.backoff.ms</code>	Retry interval. We recommend you set it to <code>1000</code> .

By doing so, you will be able to deal with the issue where the broker's leader partition can't respond to producer requests immediately.

### Async sending

The sending API is async. If you want to receive the result of sending, you can call

```
metadataFuture.get(timeout, TimeUnit.MILLISECONDS) .
```

## One producer corresponding to one app

A producer is thread-safe and can send messages to any topics. Generally, we recommend that one application correspond to one producer.

### Acks

Kafka's ACK mechanism refers to the producer's mechanism for acknowledgment of message sending. It is set to `Acks` in Kafka 0.10.x or `request.required.acks` on version 0.8.x. The setting of `Acks` will directly affect the throughput of the Kafka cluster and the reliability of messages.

The `Acks` parameter is described as follows:

Parameter	Description
<code>acks=0</code>	No response from the server is required. In this case, the performance is high, but the risk of data loss is great.
<code>acks=1</code>	A response will be returned after the primary server node writes successfully. In this case, the performance is moderate, and the risk of data loss is also moderate. A failure of the primary node may cause data loss.
<code>acks=all</code>	A response will be returned only after the primary server node writes successfully and the nodes in ISR sync successfully. The performance is poor, but the risk of data loss is small. Only a failure of both the primary and secondary nodes will cause data loss.

We recommend you select `acks=1` generally and select `acks=all` for important services.

### Batch

A CKafka topic generally has multiple partitions. Before the producer client sends a message to the server, it needs to determine the topic and the partition to which the message should be sent. When sending multiple messages to the same partition, the producer client will package the messages into a batch and then send the batch to the server. Additional overheads will be incurred when the batch is processed. In general, a small batch will cause the producer client to generate a large number of requests, which will cause the messages to queue up on the client and the server, lead to an increase in the corresponding device's CPU utilization, and thus increase the delays in message sending and consumption. A suitable batch size can reduce the number of requests sent to the server by the client during message sending, thereby improving the throughput and reducing the delay in message sending.

The batch parameters are described as follows:

Parameter	Description
-----------	-------------

Parameter	Description
<code>batch.size</code>	Size of cached messages sent to each partition (which is the sum of bytes of the messages rather than the number of messages). Once the set value is reached, a network request will be triggered, and the producer client will send the messages to the server in batch.
<code>linger.ms</code>	Maximum amount of time each message is cached. After this time elapses, the producer client will ignore the limit of the <code>batch.size</code> and immediately send the messages to the server.
<code>buffer.memory</code>	After the total size of all cached messages exceeds this value, the messages will be sent to the server immediately, and the limits of <code>batch.size</code> and <code>linger.ms</code> will be ignored. The default value of <code>buffer.memory</code> is 32 MB, which can ensure sufficient performance for a single producer.

#### Note

If you start multiple producers in the same JVM, it is likely that each producer will use 32 MB of cache space. In this case, OOM errors may occur, and you need to consider the value of `buffer.memory` in order to avoid OOM errors.

You can adjust the values of the parameters based on your actual business needs.

## Key and value

Each message in CKafka has two fields: key (message identifier) and value (message content).

For ease of tracking, set a unique key for each message, which allows you to track a message and print its sending and consumption logs to learn about its production and consumption conditions.

If you want to send a large number of messages, we recommend you use the sticky partitioning strategy instead of setting keys.

## Sticky partitioning

Only messages sent to the same partition will be placed in the same batch, so one factor that determines how a batch will be formed is the partitioning strategy set by the Kafka producer. The Kafka producer allows you to choose a partition that suits your business by setting the partitioner implementation class. If a key is specified for a message, the default strategy of the Kafka producer is to hash the message key and then select a partition based on the result of hashing to ensure that messages with the same key are sent to the same partition.

If no key is specified for a message, the default strategy of Kafka below v2.4 is to use all partitions in the topic in loops and send the message to each partition in a round robin manner. However, this default strategy has a poor batch performance and may produce a large number of small batches, which increases actual delays. As it was inefficient in partitioning messages without a key, Kafka 2.4 introduced the sticky partitioning strategy.

The sticky partitioning strategy mainly addresses the problem of small batches caused by the distribution of messages without a key into different partitions. The main practice is to randomly select another partition and use it as much as possible for subsequent messages after the batch is completed for a partition. With this strategy, messages will be sent to the same partition in the short run, but from the perspective of the entire execution, messages will be evenly sent to different partitions, which helps avoid skewed partitions while reducing delays and improving the overall service performance.

If you use a Kafka producer client on v2.4 or later, the default partitioning strategy is sticky partitioning. If you use an older producer client, you can implement a partitioning strategy on your own based on how the sticky partitioning strategy works and then make it take effect through the `partitioner.class` parameter.

For more information on how to implement the sticky partitioning strategy, see the following implementation of Java code. The code is implemented by switching from one partition to another at certain time intervals.

```
public class MyStickyPartitioner implements Partitioner {
    // Record the time of the last partition switch.
    private long lastPartitionChangeTimeMillis = 0L;
    // Record the current partition.
    private int currentPartition = -1;
    // Partition switch time interval, which can be selected based on your business needs.
    private long partitionChangeTimeGap = 100L;

    public void configure(Map<String, ?> configs) {}
    /**
     * Compute the partition for the given record.
     *
     * @param topic The topic name
     * @param key The key to partition on (or null if no key)
     * @param keyBytes serialized key to partition on (or null if no key)
     * @param value The value to partition on or null
     * @param valueBytes serialized value to partition on or null
     * @param cluster The current cluster metadata
     */
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster) {
        // Get the information of all partitions.
        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
```

```
if (keyBytes == null) {
    List<PartitionInfo> availablePartitions = cluster.availablePartitionsForTopic(topic);
    int availablePartitionSize = availablePartitions.size();
    // Determine the current available partitions.
    if (availablePartitionSize > 0) {
        handlePartitionChange(availablePartitionSize);
        return availablePartitions.get(currentPartition).partition();
    } else {
        handlePartitionChange(numPartitions);
        return currentPartition;
    }
} else {
    // For messages with a key, a partition will be selected based on the hash value of the key.
    return Utils.toPositive(Utils.murmur2(keyBytes)) % numPartitions;
}

private void handlePartitionChange(int partitionNum) {
    long currentTimeMillis = System.currentTimeMillis();
    // After the partition switch time interval elapses, the next partition will be switched to; otherwise, the original partition will still be used.
    if (currentTimeMillis - lastPartitionChangeTimeMillis >= partitionChangeTimeGap || currentPartition < 0 || currentPartition >= partitionNum) {
        lastPartitionChangeTimeMillis = currentTimeMillis;
        currentPartition = Utils.toPositive(ThreadLocalRandom.current().nextInt()) % partitionNum;
    }
}

public void close() {}
}
```

## Order in partition

Within a single partition, messages are stored in the order in which they are sent. Each topic is divided into a number of partitions. If messages are distributed to different partitions, the cross-partition message order cannot be ensured.

If you want messages to be consumed in the sending order, you can specify keys for such messages on the producer. If such messages are sent with the same key, CKafka will select a partition for their storage based on the hash of the key. As a partition can be listened on and consumed by only one consumer, messages will be consumed in the sending order.

## Message Consumption

## Basic message consumption process

1. Poll the data.
2. Execute the consumption logic.
3. Poll the data again.

## Load balancing

Each consumer group can contain multiple consumers with the same `group.id` value. In this way, consumers in the same consumer group consume the same subscribed topic.

For example, if consumer group A subscribes to topic A and enables three consumer instances C1, C2, and C3, then each message sent to topic A will be eventually delivered to only one of the three instances. By default, CKafka will evenly distribute messages to the consumer instances to achieve consumption load balancing.

Load balancing within CKafka works by allocating the partitions of the subscribed topic evenly to all consumers. Therefore, the number of consumers should not be greater than the number of partitions; otherwise, there will be consumer instances with no partitions. Except for the first start, load balancing will be triggered every time consumer instances are restarted, added, or removed.

## Subscription relationship

We recommend that all consumer instances in the same consumer group subscribe to the same topic so as to facilitate troubleshooting.

- **A consumer group subscribes to multiple topics.**

A consumer group can subscribe to multiple topics, and the messages in such topics will be consumed evenly by the consumers in the consumer group. For example, if consumer group A subscribes to topics A, B, and C, then messages in the three topics will be consumed evenly by the consumers in consumer group A.

Below is the sample code to make a consumer group subscribe to multiple topics:

```
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic: topics) {
    subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);
```

- **A topic is subscribed to by multiple consumer groups.**

A topic can be subscribed to by multiple consumer groups, and the consumers in such consumer groups independently consume all messages in the topic. For example, if both consumer groups A and B subscribe to topic A, each message sent to topic A will be delivered to both the consumer instances in consumer group A and the consumer instances in consumer group B, and the two processes are independent of each other.

## One consumer group corresponding to one application

We recommend that one consumer group correspond to one application. In other words, different applications correspond to different code snippets. If you need to write different code snippets in the same application, you should prepare different copies of `kafka.properties`, such as `kafka1.properties` and `kafka2.properties`.

## Consumer offset

Each topic has multiple partitions, and each partition counts the total number of current messages, which is called `MaxOffset`.

CKafka consumers consume messages in partitions in sequence and record the number of consumed messages, which is called `ConsumerOffset`.

Number of remaining messages (aka the "number of retained messages") = `MaxOffset` - `ConsumerOffset`.

## Offset commit

There are two parameters related to CKafka consumers:

- `enable.auto.commit`: The default value is `true`.
- `auto.commit.interval.ms`: The default value is 1000, i.e., 1s.

As a result of the combination of the two parameters, before the client polls the data, it will always check the time of the last committed offset first, and if the time defined by the `auto.commit.interval.ms` parameter has elapsed, the client will start an offset commit.

Therefore, if `enable.auto.commit` is set to `true`, it is always necessary to ensure that the data polled last time has been consumed before data polling; otherwise, the offset may be skipped.

If you want to control offset commits by yourself, set `enable.auto.commit` to `false` and call the `commit(offsets)` function.

## Offset reset

The `ConsumerOffset` will be reset under the following two circumstances:

- The server has no committed offsets (for example, when the client is started for the first time).



- A message is pulled from an invalid offset (for example, the `MaxOffset` in a partition is 10, but the client pulls a message from offset 11).

For a Java client, you can configure a resetting policy by using `auto.offset.reset`. There are three policies:

- `latest`: Consumption will start from the maximum offset.
- `earliest`: Consumption will start from the minimum offset.
- `none`: Resetting will not be performed.

Note :

- We recommend you set the resetting policy to `latest` instead of `earliest` so as to avoid starting consumption from the beginning when the offset is invalid, as that may cause a lot of repetitions.
- If you manage the offset by yourself, you can set the policy to `none`.

## Message pull

In the consumption process, the client pulls messages from the server. When the client pulls large messages, you should control the pulling speed and pay attention to the following parameters:

- `max.poll.records`: Set it to 1 if a single message exceeds 1 MB in size.
- `max.partition.fetch.bytes`: Set it to a value slightly greater than the size of a single message.
- `fetch.max.bytes`: Set it to a value slightly greater than the size of a single message.

When messages are consumed over the public network, a disconnection may often occur due to the bandwidth limit of the public network. In this case, you should control the pulling speed and pay attention to the following parameters:

- `fetch.max.bytes`: We recommend you set it to half of the public network bandwidth (note that the unit of this parameter is bytes, while the unit of the public network bandwidth is bits).
- `max.partition.fetch.bytes`: We recommend you set it to one third or one fourth of `fetch.max.bytes`.

## Duplicate messages and consumption idempotency

CKafka consumption uses the at-least-once semantics, that is, a message is delivered at least once, which guarantees that messages will never be lost. However, messages may be delivered more than once. Network issues and client restarts may cause a few duplicate messages. If the application consumer is sensitive to duplicate messages (such as orders and transactions), idempotency should be implemented for the messages.

Taking a database application as an example, the common practice is as follows:

- When sending a message, pass in the key as the unique ID.
- When consuming a message, determine whether the key has already been consumed; if so, ignore it; otherwise, consume it once.

Of course, if the application itself is not sensitive to a small number of duplicate messages, idempotency is not necessary.

## Consumption failures

In CKafka, messages are consumed from partitions one by one in sequence. If the consumer fails to execute the consumption logic after getting a message, for example, when dirty data is stored on the application server, the message will fail to be processed, and human intervention will be required. There are two methods for dealing with this situation:

- The consumer will keep trying to execute the consumption logic upon failure. This method may cause the consumer thread to be jammed by the current message and lead to message heap.
- As CKafka is not designed to process failed messages, in practice, it will typically print failed messages or store them in a service (such as a dedicated topic created for storing failed messages), so that you can regularly check failed messages, analyze the causes of failures, and process accordingly.

## Consumption delays

In the consumption process, the client pulls messages from the server. In general, if the client can consume messages timely, there will be no significant delays. If a high delay is detected, you should first check whether messages heap up and speed up consumption accordingly.

## Consumption heap

The common causes of message heap include the following:

- Consumption is slower than production. In this case, you should speed up consumption.
- Consumption is jammed.

After getting a message, a consumer will execute the consumption logic and generally make some remote calls. If it waits for the results at the same time, the consumption process may be jammed.

It should be ensured as much as possible that the consumer will not jam the consumption threads. If it needs to wait for the call results, we recommend you set a wait timeout period, so that the consumer will be treated as a failure after the timeout period elapses.

## Speeding up consumption

- Increase the number of consumer instances.

You can either increase the number of consumer instances directly in the process (you need to ensure that each

instance corresponds to one thread) or deploy multiple consumer instance processes.

#### Note

After the number of instances exceeds the number of partitions, you can't add more instances; otherwise, there will be idle consumer instances.

- Increase the number of consumer threads.
  - i. Define a thread pool.
  - ii. Poll the data.
  - iii. Commit the data into the thread pool for concurrent processing.
  - iv. Poll the data again after a successful concurrent processing result is returned.

## Socket buffers

The default value of the `receive.buffer.bytes` parameter in Kafka 0.10.x is 64 KB, but the default value of the `socket.receive.buffer.bytes` parameter in Kafka 0.8.x is 100 KB.

Both the default values are too small for high-throughput environments, especially when the bandwidth-delay product of the network between the broker and the consumer is greater than that of the local area network (LAN).

For networks with a delay of 1 ms or more and a high bandwidth (such as 10 Gbps or higher), we recommend you set the socket buffer size to 8 or 16 MB.

Even if you don't have enough memory, you should consider setting this parameter to at least 1 MB. You can also set it to -1, so that the underlying operating system will adjust the buffer size based on the actual network conditions.

However, for consumers that need to start "hot" partitions, automatic adjustment may not be that fast.

# Log Access

## Connecting CLS to CKafka

Last updated : 2022-11-01 16:23:38

### Overview

You can ship log topic data to CKafka for real-time stream computing and storage. If you haven't purchased a CKafka instance, you can consider using the [Consumption over Kafka](#) feature of CLS.

### Prerequisites

- You have activated the CKafka service.
- Make sure that the current account has the permission to enable shipping to CKafka. If your account is a sub-account, it needs to be authorized by the root account first. For more information, see [Examples of Custom Access Policies](#).

### Directions

1. Create a CKafka instance in the same region as the log topic. For more information, see [Creating Instance](#).
2. Configure the following parameters to create a topic in the same region as the log topic. For more information, see [Creating Topic](#).
  - **Preset ACL Policy:** Disable this option.
  - **Show advanced configuration:**
    - **CleanUp.policy:** Select **delete**; otherwise, shipping will fail.
    - **max.message.bytes:** Set this value to 8 MB or above. Otherwise, when the size of a single message in CLS exceeds the specified limit, the message cannot be written to the CKafka topic, and shipping will fail.
3. Go to the [CLS console](#) and enter the shipping task management page or log topic management page as needed.
  - On the left sidebar, click **Shipping Task Management** and select a region, logset, and log topic.
  - On the left sidebar, click **Log Topic** and select a log topic to be shipped to CKafka to enter the log topic management page.

4. Click the **Ship to CKafka** tab to enter the configuration page.
5. Click **Edit** on the right to enable shipping to CKafka. Then, select the target CKafka instance and topic as well as the log field to be shipped.
6. Click **OK** to start shipping to CKafka. If the task status is **Enabled**, the feature is enabled successfully.

Note :

To cleanse the log data before shipping to CKafka, see [Log Filtering and Distribution](#).

## FAQs

### What should I do if the log data cannot be shipped to CKafka?

If ACL authentication is enabled in CKafka, the log data cannot be shipped. In this case, you need to disable the ACL of the topic.

### What should I do if the system prompts that I have no permissions to read/write the CKafka topic?

If you directly use an API to ship data to CKafka, you may not have the read/write permissions of the CKafka topic. If you ship data in the console, the system will guide you through the authorization process, but if you directly call an API for shipping, you need to authorize manually. For more information, see [Viewing and Configuring Shipping Permissions](#).

# Replacing Supportive Route (Old)

Last updated : 2022-11-29 11:36:24

## Background

To enjoy more stable and reliable services, we recommend you switch to the new supportive route.

## Impact

Because it is necessary to update the bootstrap-server addresses of producers and consumers and restart the service, the switch will cause a momentary interruption in business production and consumption.

## Directions

1. Create a supportive route.
2. Switch the CKafka bootstrap-server addresses of all producers and consumers to the newly created supportive route. The switch order does not matter as long as they are all switched.
3. Restart producers and consumers based on the business conditions. The restart order does not matter.
4. Observe whether the business is stable for at least 3 hours (recommended).
5. Delete the old supportive route.

## Rollback

If an exception occurs during business verification, you need to roll back to the old supportive route in the following steps provided that it has not been deleted.

1. Change back to the old supportive route address on all producers and consumers.
2. Restart all producer and consumer services.
3. Verify the business.

# DataHub Best Practices

## Connection to Kafka over HTTP

Last updated : 2022-05-20 11:34:10

### Overview

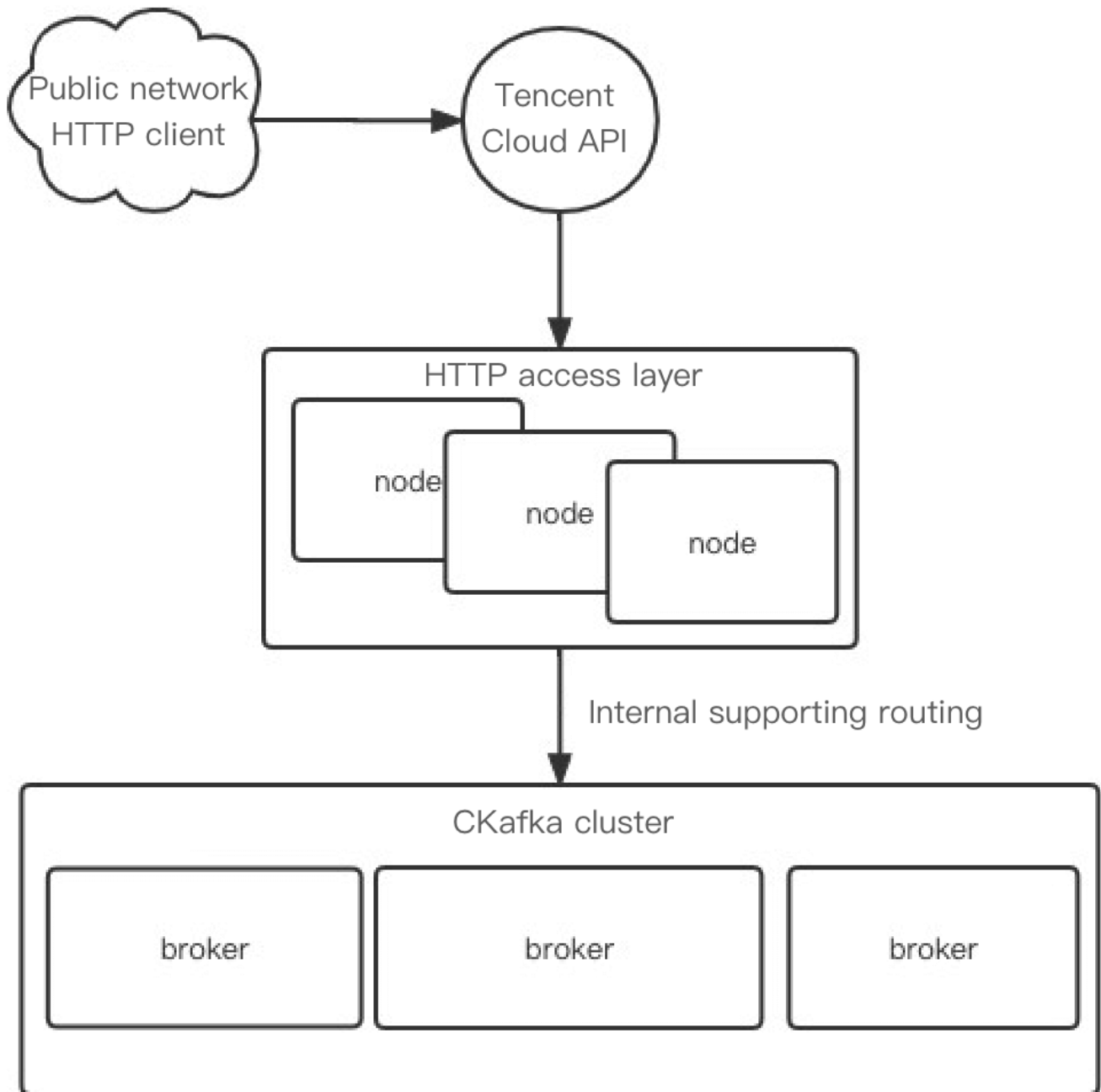
Apache Kafka, like any client-server application, offers access to its functionality through a well-defined set of APIs. These APIs are exposed via the Kafka wire protocol, a Kafka-specific binary protocol over TCP. The best way to interact with the Apache Kafka APIs is to make use of a client library that works with the Kafka wire protocol. The Apache Kafka project only officially supports a client library for Java, but in addition to that, Confluent officially supports client libraries for C/C++, C#, Go, and Python.

Unfortunately, some programming languages lack officially supported, production-grade client libraries for Kafka. However, HTTP is a widely available, universally supported protocol. For data access, DataHub exposes message sending APIs over the HTTP protocol to simplify client configurations.

This document describes message sending in the HTTP-based data access feature of DataHub and provides suggestions for real world cases.

### Architecture

After the HTTP data access layer is enabled, an HTTP client in the public network can directly send messages to a CKafka instance through TencentCloud API as shown below:



## Prerequisites

You have created the target CKafka instance and topic.

## Directions

### Creating data access task



For detailed directions, see [Reporting over HTTP](#).

## Sending message via SDK

1. Import the data reporting SDK through Maven or Gradle into the Java project. Below is the `pom.xml` file for project configuration:

```
<dependency>
<groupid>com.tencentcloudapi</groupid>
<artifactid>tencentcloud-sdk-java</artifactid>
<version>3.1.430</version>
</dependency>
```

2. Click **Task Details** in [Data Access](#) and copy the access point information to the SDK for data writes.
3. Enter the access point information. In the sample code, `generateMsgFromUserAccess` is used to assemble all messages to be sent.

```
List<batchcontent> batchContentList = generateMsgFromUserAccess(userId);
// Here, `ap-xxx` is the region abbreviation of the corresponding TencentCloud
API
CkafkaClient client = new CkafkaClient(
new Credential("yourSecretId", "yourSecretKey"), "ap-xxx");

SendMessageRequest messageRequest = new SendMessageRequest();
// Access point ID of the data access task
messageRequest.setDataHubId("datahub-lxxxxxx6");
messageRequest.setMessage(batchContentList.toArray(BatchContent[]:new));
try {
SendMessageResponse sendMessageResponse = client.SendMessage(messageRequest);
String[] messageId = sendMessageResponse.getMessageId();
for (String s : messageId) {
LOGGER.info(s)
}
} catch (TencentCloudSDKException e) {
LOGGER.error(e.getMessage());
}
```

4. Below is a sample returned value for message sending at the HTTP access layer:

```
{
  "Response": {
    "MessageId": [
      "datahub-lxxxxxx6:topicDev:4:2:1648185961342:1648185961398"
    ],
    "RequestId": "3fq3na5r-xxxx-xxxx-xxxx-b2fiv0se7ded"
  }
}
```

5. Here, **MessageId** consists of a series of metadata fields returned after the message is sent to the CKafka instance, as detailed below:

```
"[datahubId]:[topic name]:[topic partition number]:[topic offset]:[time when the HTTP access layer received the message]:[time when the message was sent to Kafka]"
```

## Querying message

You can query messages sent at the HTTP access layer in the [CKafka console](#). For detailed directions, see [Querying Message](#). In this example, messages at offset 2 in partition 4 in the `topicDev` topic are queried as shown below:

The screenshot shows the CKafka console query interface. The filters are set as follows:

- Instance: ckafka
- Topic: [redacted]
- Query Type: Query by offset (selected)
- Partition ID: 0
- Start Offset: 0

A blue "Query" button is visible below the filters.

The results table below shows two messages:

Partition ID	Offset	Timestamp	Operation
0	0	2021-03-02 11:32:15	<a href="#">View Message Details</a>
0	1	2021-03-02 11:33:13	<a href="#">View Message Details</a>

## Pausing task

If you find that the data access task affects the normal business, you can pause the task.

1. On the [Data Access](#) page, click **Pause** in the **Operation** column of the target task to pause the task.
2. If the prompt in the top-right corner in the following figure is displayed, the task was paused successfully.
3. At this time, if you send a message at the HTTP access layer, you will receive the following response:

```
{
  "Response": {
    "Error": {
      "Code": "FailedOperation",
      "Message": "task status suspended [datahub-lxxxxxxx6]"
    },
    "RequestId": "5f737a5b-xxxx-xxxx-xxxxx-b2fb703e7ded"
  }
}
```

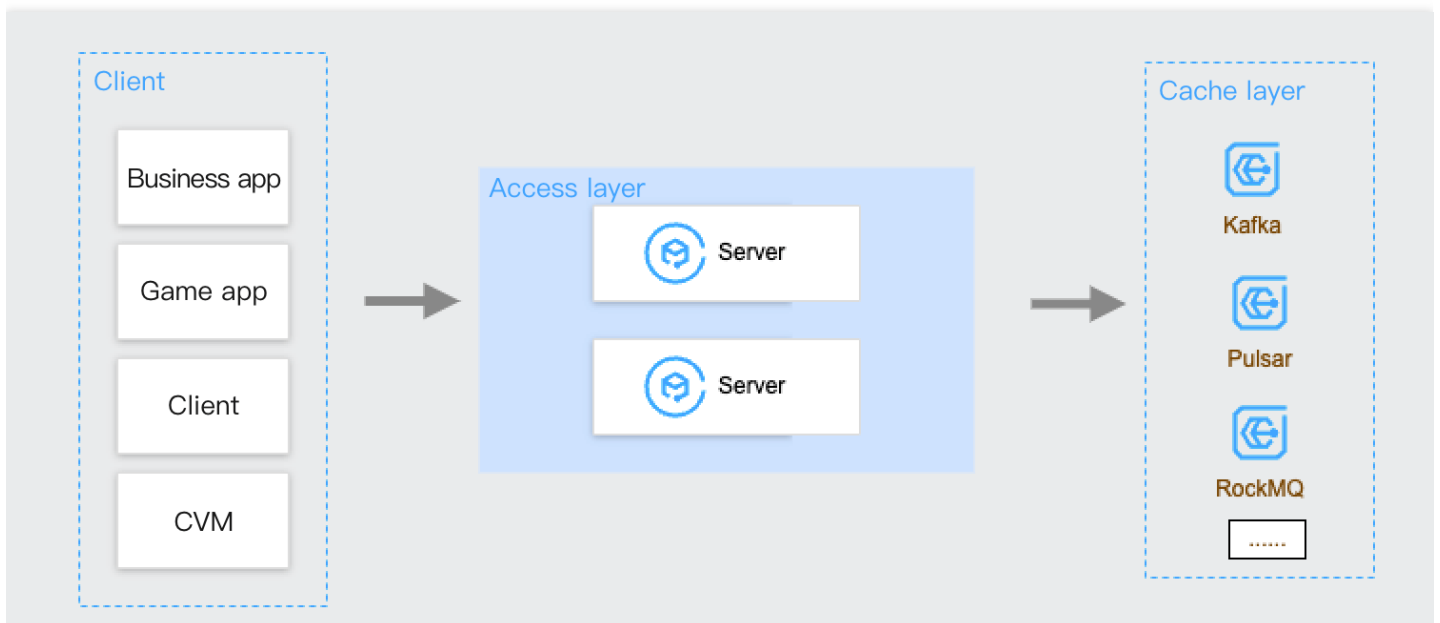
```
}  
}
```

# Unified Data Reporting

Last updated : 2022-05-20 11:40:21

## Overview

DataHub is a data access and processing platform in Tencent Cloud, which provides data access, processing, and distribution features at one stop. Data is vital in internet businesses, and data access and reporting serve as the bridge between data generation, computing, storage, and analysis throughout the entire linkage. Therefore, simple yet efficient data access is critical. A business usually has data to be reported to the backend for storage, analysis, computing, and search, such as business metrics, process information, and monitoring data. The general processing linkage is as shown below:



You can set up a classic data reporting architecture generally in the following steps:

1. Build/Purchase a storage engine to store reported data.
2. Develop and deploy a server to receive data, define APIs, and run the service.
3. Define information such as API protocol and authentication on the client and server.
4. Write code based on the protocol information for the client to report data.

In the above four steps, the development and deployment workload for the server is the highest, as you need to consider code logic development as well as the scalability and stability of the server and downstream storage. In addition, when the data volume gets high, problems on the server will become more obvious, and the server needs to be maintained with a lot of manpower and resources. As tasks involved in this regard are generally universal, DataHub

aims to meet the requirements in such scenario by offering a stable, elastic, high-reliability, and high-throughput data access service.

## How It Works

DataHub provides SDKs for various programming languages, including Java, Python, Go, PHP, Node.js, C++, and .NET, to help the client better report data. Data can be reported to a storage engine such as CKafka in three simple steps (more Tencent Cloud message queue services like TDMQ for RocketMQ, Pulsar, RabbitMQ, and CMQ will be supported in the future).

1. Create an access point in the DataHub console.
2. Report the data via the SDK.
3. Query the data.

## Directions

1. Create an access point in the DataHub console as instructed in [Reporting over HTTP](#).
2. Report the data via the SDK as instructed in [Data Reporting SDK](#).
3. Query the data. After the data is reported to DataHub, you can query the message content in real time as instructed in [Querying Message](#).

## Data Empowerment

After you complete data access easily and quickly through DataHub, how to make the data generate value becomes the most important thing. To address this, DataHub provides two core features:

### Data processing

DataHub offers a simple data ETL engine, which can cleanse most types of data in order to simply format and process data for subsequent use.

### Data distribution

After data processing is completed, DataHub can also meet the data distribution needs in various scenarios:

- Real-time search: When you need to search for data, you can export real-time data streams to a search service such as Elasticsearch and CLS.
- OLAP analysis: When you need to analyze data, you can export real-time data streams to an engine such as ClickHouse and TDW.

- Persistent storage: When you need to persistently store data, you can export real-time data streams to a persistent storage engine such as HDFS and COS.
- Stream computing: When you need to process data with custom code, you can use Flink, Spark, and code in different programming languages over the standard Kafka protocol for data processing.

After the data has gone through the four stages of reporting, access, processing, and distribution, the general data reporting and analysis needs can be easily and quickly satisfied, creating data value at ultra low costs.

# Simple Data Cleansing

Last updated : 2022-05-20 11:42:57

## Overview

When using the data access and distribution services in DataHub, you may encounter the following issues:

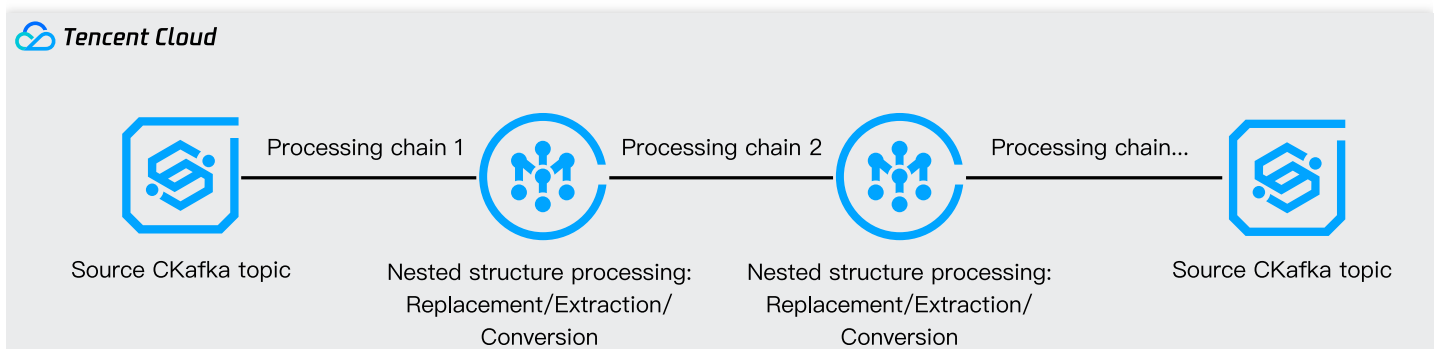
- You need to parse certain fields in messages to get the relevant information.
- You need to process a field in messages multiple times in an iterative manner.
- You need to convert messages to unstructured raw messages before you can use them.
- You need to process messages in a multi-level nested format.

Based on the long-term experience of technical experts in the CKafka team, the brand new data processing component v2.0 is launched. It has the following new features while remaining fully compatible with v1.0:

- Diverse plugins: Data processing supports various preconfigured processing plugins to help you quickly generate data in desired consumption formats.
- Chain processing: Data processing supports chain message processing; that is, the previous processing result can be used as the input parameters in the current processing. This greatly facilitates processing complex data structures.
- Visual preview: Data processing supports real-time structured JSON preview to help you quickly locate and troubleshoot problems.
- Type conversion: Data processing can convert data between different types to make data format check easier.

## How It Works

The overall data processing flow is as shown below, with each component structure detailed as follows:



- In the data processing component cluster, multiple workers form a consumer group to batch read messages from the source topic and process each message in sequence.
- In each message, the nested structure of message fields is expanded according to the processing chain sequence configured in the console, and operations such as replacement, extraction, data conversion, and time formatting are performed.
- The current processing chain reads the result of the previous processing chain, performs chain processing, and ships the processing result to the next chain.
- The result of the last processing chain is shipped to the configured target topic. At this time, the data processing of a message is completed.

## Prerequisites

- The CKafka service has been activated.
- The message format is JSON or string. Currently, other encoding protocols are not supported.

## Actual Application

### Processing string-type log

Below is a log in the default Nginx format (aka `combined` format), from which you can parse information such as requester, packet, and request status for further analysis.

```
66.249.65.159 - - [06/Nov/2014:19:10:38 +0600] "GET /news/53f8d72920ba2744fe873e
bc.html HTTP/1.1" 404 177 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac O
S X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8
536.25 (compatible; Googlebot/2.1; +http://www.google.com/bot.html) "
```

As the `combined` format of Nginx uses spaces and `-` to separate data, design the parsing process in the following steps:

1. First, use the `"` **separator** to initially parse the data. At this time, data processing automatically converts the log to the JSON structure.

```
{
  "0": "66.249.65.159 - - [06/Nov/2014:19:10:38 +0600] ",
  "1": "GET /news/53f8d72920ba2744fe873ebc.html HTTP/1.1",
  "2": " 404 177 ",
  "5": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26
(KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8536.25 (compatible; Go
```



```
oglegbot/2.1; +http://www.google.com/bot.html) "
```

2. As shown in the above JSON structure, there are still concatenated coupled data records in the fields of keys `0` and `2` as affected by `-` and spaces. Therefore, split the fields of the two keys with the space and `-` **separators**. The JSON result after splitting is as follows:

```
{
  "1": "GET /news/53f8d72920ba2744fe873ebc.html HTTP/1.1",
  "5": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8536.25 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)",
  "0.0": "66.249.65.159 ",
  "0.2": " [06/Nov/2014:19:10:38 +0600] ",
  "2.1": "404",
  "2.2": "177"
}
```

3. As the time format is enclosed by square brackets “[ ]”, use a **separator** again to extract the time information. The JSON structure after extraction is as follows:

```
{
  "1": "GET /news/53f8d72920ba2744fe873ebc.html HTTP/1.1",
  "5": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8536.25 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)",
  "0.0": "66.249.65.159 ",
  "0.2": "06/Nov/2014:19:10:38 +0600",
  "2.1": "404",
  "2.2": "177"
}
```

4. At this point, all fields have been split appropriately. Name the `keys` of the corresponding mapped fields based on the field attribute. The eventual modification result is as follows:

```
{
  "request": "GET /news/53f8d72920ba2744fe873ebc.html HTTP/1.1",
  "http_user_agent": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8536.25 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)",
  "remote_addr": "66.249.65.159 ",
  "dateTime": "06/Nov/2014:19:10:38 +0600",
  "status": "404",
  "body_bytes_sent": "177"
}
```

## Processing nested log

Below is a sample TKE collection format. The TKE collector will place the metadata into the `kubernetes` field in the JSON structure and place the collected log into the `log` field. The overall structure is as follows:

```
{
  "@timestamp": 1648803500.63659,
  "@filepath": "/var/log/tke-log-agent/test7/c816991f-adfe-4617-8cf3-9997aea90ded/c_tke-es-687995d557-n29jr_default_nginx-add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba/jgw_INFO_2022-02-10_15_4.log",
  "log": "15:00:00.000[4349811564226374227] [http-nio-8081-exec-64] INFO com.qcloud.jgw.gateway.server.topic.TopicService",
  "kubernetes": {
    "pod_name": "tke-es-687995d557-n29jr",
    "namespace_name": "default",
    "pod_id": "c816991f-adfe-4617-8cf3-9997aea90ded",
    "labels": {
      "k8s-app": "tke-es",
      "pod-template-hash": "687995d557",
      "qcloud-app": "tke-es"
    },
    "annotations": {
      "qcloud-redeploy-timestamp": "1648016531476",
      "tke.cloud.tencent.com/networks-status": "[{\n  \"name\": \"tke-bridge\", \n  \"interface\": \"eth0\", \n  \"ips\": [\n    \"172.16.0.31\" \n  ], \n  \"mac\": \"ae:61:12:4a:c2:ba\", \n  \"default\": true, \n  \"dns\": { }\n}]"
    },
    "host": "10.0.96.47",
    "container_name": "nginx",
    "docker_id": "add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba",
    "container_hash": "nginx@sha256:e1211ac17b29b585ed1aee166a17fad63d344bc973bc63849d74c6452d549b3e",
    "container_image": "nginx"
  }
}
```

When collected logs are shipped to Elasticsearch, as the nested JSON format is supported, you don't need to make major changes on the data. However, when logs are shipped to a database, you need to convert the data to an identifiable single-level JSON format. In this case, design the parsing process in the following steps:

1. Use the **JSONPATH** statement to process the `$.kubernetes` JSON structure at the first nested level and select `JSON` as the parsing mode. This converts the nested JSON structure to a single-level JSON structure.

The tested result is as follows:

```
{
  "@timestamp": 1.64880350063659E9,
```

```

"@filepath": "/var/log/tke-log-agent/test7/c816991f-adfe-4617-8cf3-9997aea90ded/c_tke-es-687995d557-n29jr_default_nginx-add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba/jgw_INFO_2022-02-10_15_4.log",
"log": "15:00:00.000[4349811564226374227] [http-nio-8081-exec-64] INFO com.qcloud.jgw.gateway.server.topic.TopicService",
"$kubernetes.pod_name": "tke-es-687995d557-n29jr",
"$kubernetes.namespace_name": "default",
"$kubernetes.pod_id": "c816991f-adfe-4617-8cf3-9997aea90ded",
"$kubernetes.labels": {
  "k8s-app": "tke-es",
  "pod-template-hash": "687995d557",
  "qcloud-app": "tke-es"
},
"$kubernetes.annotations": {
  "qcloud-redeploy-timestamp": "1648016531476",
  "tke.cloud.tencent.com/networks-status": "[{\n  \"name\": \"tke-bridge\", \n  \"interface\": \"eth0\", \n  \"ips\": [\n    \"172.16.0.31\" \n  ], \n  \"mac\": \"ae:61:12:4a:c2:ba\", \n  \"default\": true, \n  \"dns\": {} \n}]"
},
"$kubernetes.host": "10.0.96.47",
"$kubernetes.container_name": "nginx",
"$kubernetes.docker_id": "add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba",
"$kubernetes.container_hash": "nginx@sha256:e1211ac17b29b585ed1aee166a17fad63d344bc973bc63849d74c6452d549b3e",
"$kubernetes.container_image": "nginx"
}

```

2. Process the `$.kubernetes.annotations` and `$.kubernetes.labels` nested structures at the second level. Use `Map` to select the two names in the processing chain to convert the nested format into a single-level JSON format. The processing result is as follows:

```

{
  "@timestamp": 1648803500.63659,
  "@filepath": "/var/log/tke-log-agent/test7/c816991f-adfe-4617-8cf3-9997aea90ded/c_tke-es-687995d557-n29jr_default_nginx-add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba/jgw_INFO_2022-02-10_15_4.log",
  "log": "15:00:00.000[4349811564226374227] [http-nio-8081-exec-64] INFO com.qcloud.jgw.gateway.server.topic.TopicService",
  "$kubernetes.pod_name": "tke-es-687995d557-n29jr",
  "$kubernetes.namespace_name": "default",
  "$kubernetes.pod_id": "c816991f-adfe-4617-8cf3-9997aea90ded",
  "$kubernetes.host": "10.0.96.47",
  "$kubernetes.container_name": "nginx",
  "$kubernetes.docker_id": "add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba",

```

```

"$kubernetes.container_hash": "nginx@sha256:e1211ac17b29b585ed1aee166a17fad63
d344bc973bc63849d74c6452d549b3e",
"$kubernetes.container_image": "nginx",
"$kubernetes.labels.k8s-app": "tke-es",
"$kubernetes.labels.pod-template-hash": "687995d557",
"$kubernetes.labels.qcloud-app": "tke-es",
"$kubernetes.annotations.qcloud-redeploy-timestamp": "1648016531476",
"$kubernetes.annotations.tke.cloud.tencent.com/networks-status": "[{\n \name
\": \"tke-bridge\", \n \interface\": \"eth0\", \n \ips\": [\n \"172.16.0.31
\n ], \n \mac\": \"ae:61:12:4a:c2:ba\", \n \default\": true, \n \dns\": {}
\n}]"
}

```

3. Rename the `keys` of the corresponding mapped fields and delete unnecessary fields. Click **Add Processing Chain** and **Process All Upper-Level Results**. Below is a sample result after organization and optimization:

```

{
"@timestamp": 1.64880350063659E9,
"@filepath": "/var/log/tke-log-agent/test7/c816991f-adfe-4617-8cf3-9997aea90ded/c_tke-es-687995d557-n29jr_default_nginx-add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba/jgw_INFO_2022-02-10_15_4.log",
"log": "15:00:00.000[4349811564226374227] [http-nio-8081-exec-64] INFO com.qcloud.jgw.gateway.server.topic.TopicService",
"pod_name": "tke-es-687995d557-n29jr",
"namespace_name": "default",
"pod_id": "c816991f-adfe-4617-8cf3-9997aea90ded",
"host": "10.0.96.47",
"container_name": "nginx",
"docker_id": "add90ccf49626ef42d5615a636aae74d6380996043cf6f6560d8131f21a4d8ba"
}

```

#### Note :

If the `key` contains a period (.) when you use **JSONPath** to process a parameter, you need to add square brackets and single quotation marks to the path for isolation.

For example, to get the desired fields from `{"key1.key2": "value1"}`, you need to use `$.['key1.key2']` to get the corresponding key values.

## Processing serialized JSON string-type log

Sometimes, the JSON format needs to be escaped to the string format during data transfer in order to meet the format or performance requirements. This string format is called **raw JSON**, which must be deserialized to the JSON format

in data processing. The following uses the raw JSON format in MongoDB as an example, and the overall structure is as shown below:

```
{
  "key": " {\n \"categories\": [\"dev\"],\n \"created_at\": \"2020-01-05 13:42:19.324003\",\n \"icon_url\": \"https://assets.chucknorris.host/img/avatar/chuck-norris.png\",\n \"id\": \"elgv2wkv8ioag6xywykbq\",\n \"updated_at\": \"2020-01-05 13:42:19.324003\",\n \"url\": \"https://api.chucknorris.io/jokes/elgv2wkv8ioag6xywykbq\",\n \"value\": \"Chuck Norris's keyboard doesn't have a Ctrl key because nothing controls Chuck Norris.\"\n }\n"
```

After the raw JSON format is converted to the general JSON format during data processing, the message can be directly shipped to the target downstream service without changing the field format. The overall processing process is as follows:

1. Set **Parsing Mode** to JSON to pre-read the message into the internal MAP format. The parsing result is as follows:

```
{
  "key": " {\n \"categories\": [\"dev\"],\n \"created_at\": \"2020-01-05 13:42:19.324003\",\n \"icon_url\": \"https://assets.chucknorris.host/img/avatar/chuck-norris.png\",\n \"id\": \"elgv2wkv8ioag6xywykbq\",\n \"updated_at\": \"2020-01-05 13:42:19.324003\",\n \"url\": \"https://api.chucknorris.io/jokes/elgv2wkv8ioag6xywykbq\",\n \"value\": \"Chuck Norris's keyboard doesn't have a Ctrl key because nothing controls Chuck Norris.\"\n }\n"
```

2. Click **Add Processing Chain**, use MAP to select the `key`, and select `JSON` as the parsing mode. Then, data processing can automatically convert the raw JSON format to the JSON format. The parsing result is as follows:

```
{
  "key.categories": [
    "dev"
  ],
  "key.created_at": "2020-01-05 13:42:19.324003",
  "key.icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
  "key.id": "elgv2wkv8ioag6xywykbq",
  "key.updated_at": "2020-01-05 13:42:19.324003",
  "key.url": "https://api.chucknorris.io/jokes/elgv2wkv8ioag6xywykbq",
  "key.value": "Chuck Norris's keyboard doesn't have a Ctrl key because nothing controls Chuck Norris."
}
```

3. Click **Add Processing Chain** and **Process All Upper-Level Results**, rename the `keys` of the corresponding mapped fields, and delete unnecessary fields. Below is a sample result after organization and optimization:

```
{
  "categories": [
    "dev"
  ],
  "created_at": "2020-01-05 13:42:19.324003",
  "icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
  "id": "elgv2wkv8ioag6xywykbq",
  "updated_at": "2020-01-05 13:42:19.324003",
  "url": "https://api.chucknorris.io/jokes/elgv2wkv8ioag6xywykbq",
  "value": "Chuck Norris's keyboard doesn't have a Ctrl key because nothing controls Chuck Norris."
}
```

**Note :**

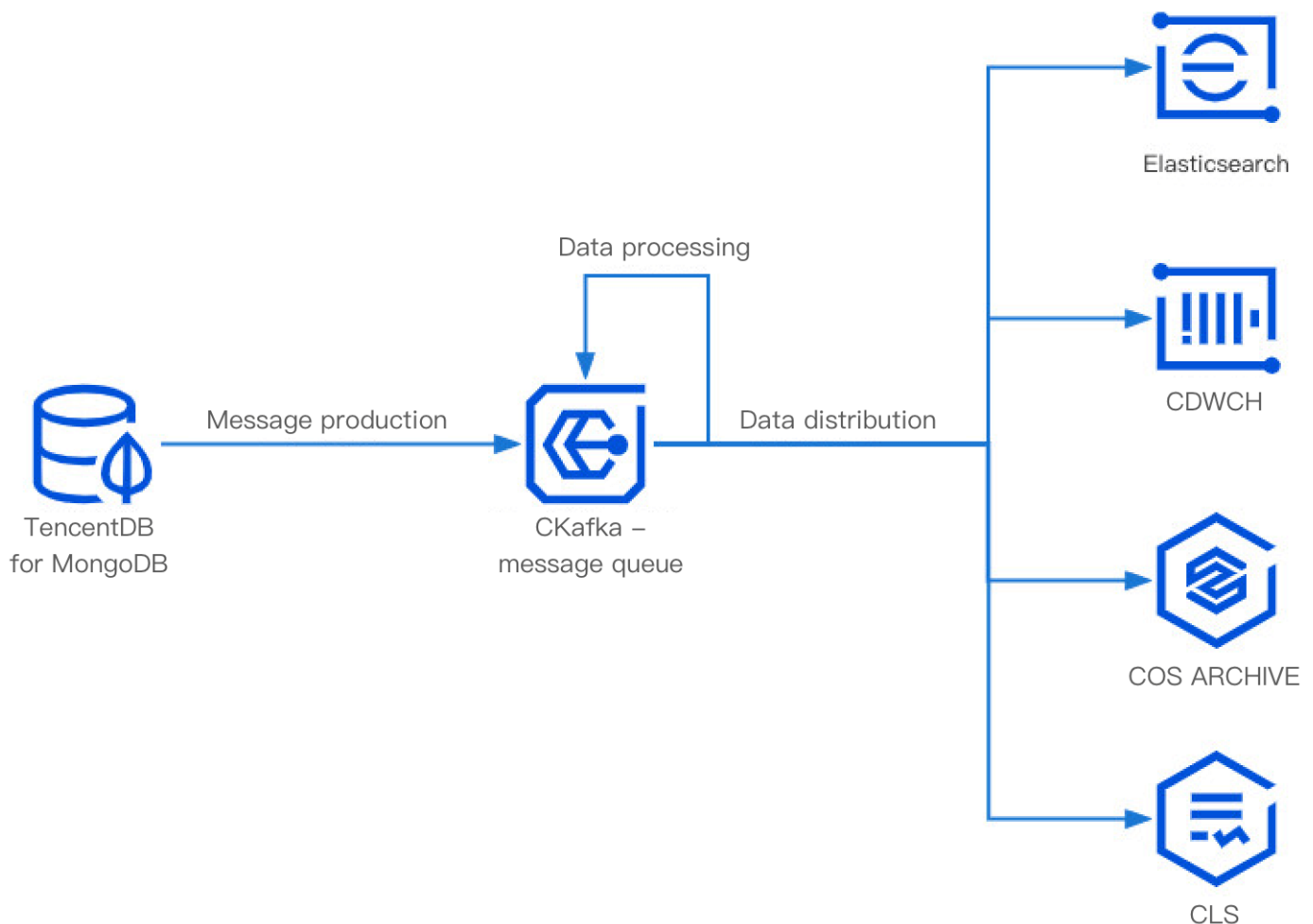
Currently, only MAP-type data in the raw JSON format can be parsed. If the first level is in List type, such as `"["test1","test2"]"` or `"[{"key":"value"}]"`, as it cannot be parsed into appropriate key values, a parsing failure will be reported.

# Analysis of Change Logs Tracked by MongoDB Change Streams

Last updated : 2022-06-06 14:39:21

## Overview

MongoDB uses change streams to track changes. However, to better search for change logs, you usually need to sync them to Elasticsearch or CLS.



This document uses connecting MongoDB to CKafka and distributing CKafka data to CLS as an example to describe how to use the data dump service of DataHub to analyze change logs tracked by change streams in MongoDB.

## How It Works

For more information on data access configuration items for MongoDB, see [Source Connector](#). You can set different configuration items to perform corresponding data processing tasks on the accessed change logs and then write them to a topic in the CKafka instance.

## Prerequisites

- TencentDB for MongoDB has been activated, or CLB is used to listen on the self-built MongoDB instance. The TCP:27017 port has been opened in the security group.
- The CKafka service has been activated.
- The CLS service has been activated.

## Directions

### Step 1. Create a data access task

1. Log in to the [CKafka console](#).
2. Click **Data Access** on the left sidebar, select the region, and click **Create Task**.
3. In the pop-up window, select **Asynchronously pulled data** > **MongoDB** for **Data Source Type**.
4. Click **Next** and enter the task details.
5. Click **Submit** and wait for the task status to become **Healthy**.
6. When MongoDB data changes, you can see that there will be incremental messages in the selected topic in the CKafka instance.

### Step 2. Create a data distribution task

1. Log in to the [CKafka console](#).
2. Click **Data Distribution** on the left sidebar, select the region, and click **Create Task**.
3. Select **Cloud Log Service (CLS)** as the **Target Type** and click **Next**.
4. Enter the task details and select the same CKafka instance and topic as those used in the data access task, so that produced messages can be directly consumed.
5. Click **Submit** and wait for the task status to become **Healthy**.

#### Note

When a task is in **Healthy** status and incremental messages are written to the topic, they will be directly consumed to the specified CLS log topic.



### Step 3. View the distributed data

1. Log in to the [CLS](#) console.
2. Select **Search and Analysis** on the left sidebar, select the logset ID and log topic ID entered during distribution task creation, and you can view the change logs of MongoDB.
3. You can search by keyword to directly get the required logs.