arm TREASURE DATA

KAFKA GUIDE

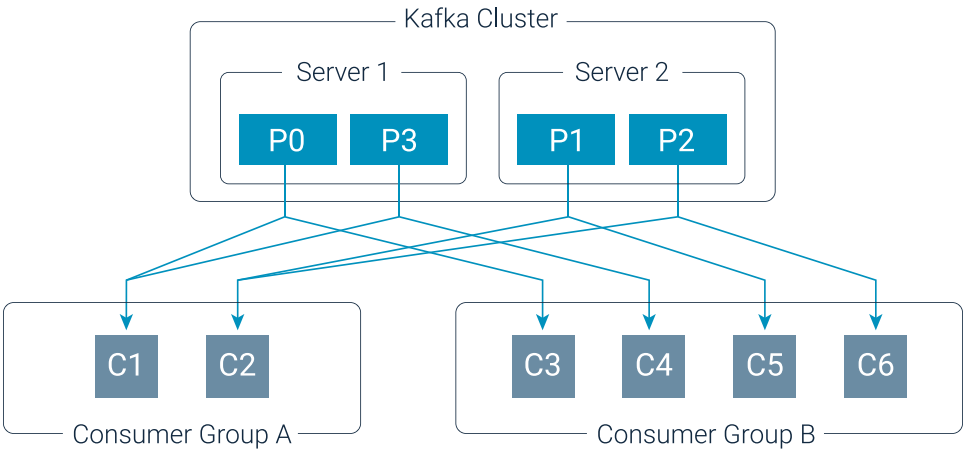
About Apache Kafka

Apache Kafka is a high-throughput distributed messaging system, based on a distributed, partitioned and replicated commit log service. Apache Kafka aims to serve as a unified, high-throughput and low latency platform for real time data feeds.

Why do you need it?

Given the enormous amount of data to be collected and analyzed, there must be a way to connect applications that **produce** data from applications that **consume** it.

In order to seamlessly integrate these often disparate and inaccessible parts of your data pipeline without expensive rewrites of parts of your applications on either end, a mechanism is required.



Typically, data in an analytics pipeline of the sort Kafka might handle could include the following:

- User behavior data
- Application performance tracing
- Application and/or activity logs
- Event messages

In addition to setting up the messaging service itself, we'll include a few extra tricks to build out your data analytics pipeline. With **Fluentd**, you can, with minimal system resources, collect and route data from Kafka to the data destination of your choice.

Treasure Data is a cloud service that superpowers and simplifies your data analytics pipeline with built-in data collection, storage, and analytics.

This guide will show you how to:

- Install Kafka
- Set up a single node/single broker cluster
- Set up a multiple broker cluster
- Import and Export Data
- Set up Fluentd
- Set up Fluentd Consumer for Kafka
- Query your data on Treasure Data

Note: writing custom Kafka producers and consumers is out of the scope of this guide. The producer and consumer scripts used in this guide are built-in example scripts that come with the Kafka distribution. In a real scenario, it would be your applications that act as producers and consumers.

Your mileage with this guide may vary: you may need to deviate a bit from these instructions to get everything working on your system of choice.

Prerequisites

Create a user for Kafka

```
$ sudo useradd kafka -m
$ sudo passwd kafka
$ sudo adduser kafka sudo
#log into that kafka user
$ su - kafka
```

Install Java and Zookeeper

```
$ sudo apt-get update
$ sudo apt-get install default-jre
$ sudo apt-get install zookeeperd
```

Get the code and untar it

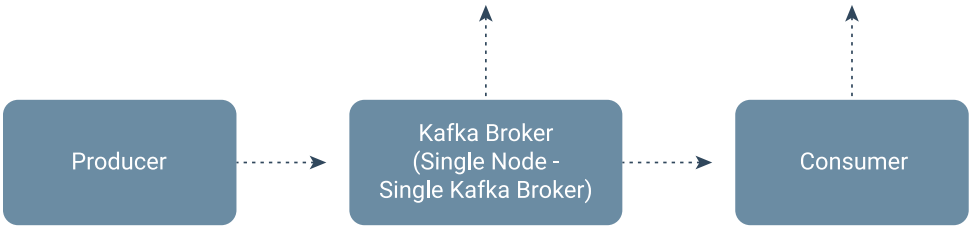
```
$ mkdir downloads  && cd downloads

$ wget “http://mirror.cc.columbia.edu/pub/software/
apache/kafka/0.8.2.1/kafka_2.11-0.8.2.1.tgz” -O ~/
downloads/kafka.tgz

$ tar -xvzf ~/downloads/kafka.tgz --strip 1
$ cd ../kafka/downloads/
```

Single node/single broker cluster

Note: you'll need to open multiple command tabs (we had as many as 6 open to run different producers, zookeeper, brokers and consumers) and each one should ssh separately into the virtual machine, local instances, AWS Instance or droplet you are running Kafka on.



Start Zookeeper

```
> bin/zookeeper-server-start.sh config/zookeeper.
properties
[2013-04-22 15:01:37,495] INFO Reading configuration
from: config/zookeeper.properties (org.apache.zookeeper.
server.quorum.QuorumPeerConfig)
...
```

Start Kafka Server

```
> bin/kafka-server-start.sh config/server.properties
[2013-04-22 15:01:47,028] INFO Verifying properties
(kafka.utils.VerifiableProperties)
[2013-04-22 15:01:47,051] INFO Property socket.send.
buffer.bytes is overridden to 1048576 (kafka.utils.
VerifiableProperties)
...
```

Create a topic

Let's create a topic named "test" with a single partition and only one replica:

```
> bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1
--topic test
```

We can now see that topic if we run the list topic command:

```
> bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

Send some messages from command line (start a producer & send messages)

Note: this step uses the built-in producer script that comes with the Kafka distribution. In a real scenario, it would be your application that acts as the producer.

```
> bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
Message!
Another message!
Moar messages! :v
```

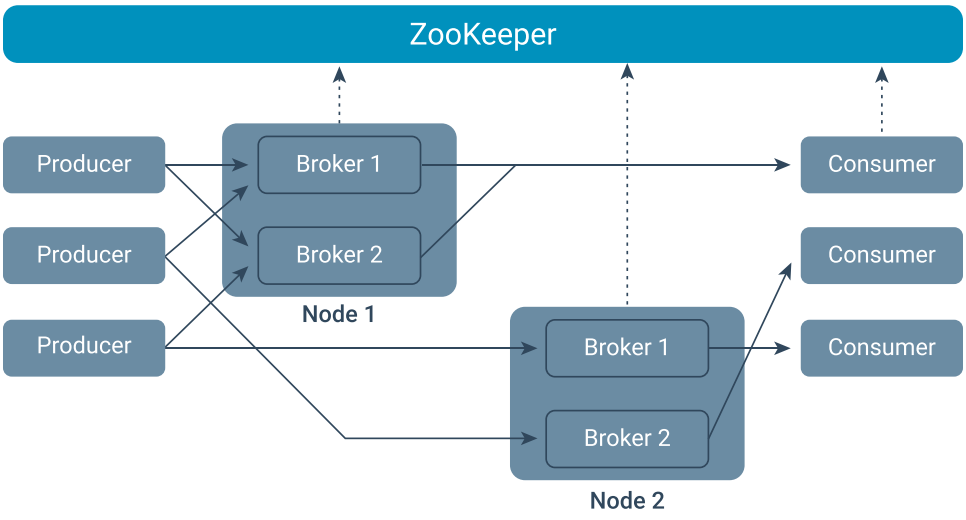
Receive some messages from command line (start a consumer & receive messages)

Note: this step uses the built-in consumer script that comes with the Kafka distribution. In a real scenario, it would be your application that acts as the consumer.

```
> bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --topic test --from-beginning
Message!
Another message!
Moar messages! :v
```

Multiple Broker cluster

For our purposes, we'll set up only one producer, multiple brokers, and a single consumer, although the relevant steps can be duplicated to set up multiple producers and consumers.



Create config file for each of the brokers

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

Edit each config file

```
config/server-1.properties:
    broker.id=1
    port=9093
    log.dir=/tmp/kafka-logs-1

config/server-2.properties:
    broker.id=2
    port=9094
    log.dir=/tmp/kafka-logs-2
```

Start Zookeeper

```
> bin/zookeeper-server-start.sh config/zookeeper.
properties
[2013-04-22 15:01:37,495] INFO Reading configuration
from: config/zookeeper.properties (org.apache.zookeeper.
server.quorum.QuorumPeerConfig)
```

Start Kafka Servers (multiple nodes)

```
> bin/kafka-server-start.sh config/server.properties
[2013-04-22 15:01:47,028] INFO Verifying properties
(kafka.utils.VerifiableProperties)
[2013-04-22 15:01:47,051] INFO Property socket.send.
buffer.bytes is overridden to 1048576 (kafka.utils.
VerifiableProperties)
> bin/kafka-server-start.sh config/server-1.properties &
...
> bin/kafka-server-start.sh config/server-2.properties &
```

Create new topic (replication factor of 3)

```
> bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 3 --partitions 1
--topic my-replicated-topic
```

Check which broker is doing what (“describe topics” command)

```
bin/kafka-topics.sh --describe --zookeeper
localhost:2181 --topic my-replicated-topic
Topic:my-replicated-topic PartitionCount:1
ReplicationFactor:3 Configs:
  Topic: my-replicated-topic    Partition: 0
Leader: 1     Replicas: 1,2,0   Isr: 1,2,0

bin/kafka-topics.sh --describe --zookeeper
localhost:2181 --topic test
Topic:test    PartitionCount:1  ReplicationFactor:1
Configs:
  Topic: test Partition: 0   Leader: 0   Replicas: 0
Isr: 0
```

Send - Publish a few messages to our new topic

```
> bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic my-replicated-topic
...
my test message 1
my test message 2
^C
```

Receive - Consume Published messages

```
> bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --from-beginning --topic my-replicated-
topic
...
my test message 1
my test message 2
^C
```

Test Fault tolerance - Kill broker 1

```
> ps | grep server-1.properties
7564 ttys002    0:15.91 /System/Library/Frameworks/
JavaVM.framework/Versions/1.6/Home/bin/java...
> kill -9 7564
```

Check what happened to the topic

```
> bin/kafka-topics.sh --describe --zookeeper
localhost:2181 --topic my-replicated-topic
Topic:my-replicated-topic PartitionCount:1
ReplicationFactor:3 Configs:
  Topic: my-replicated-topic    Partition: 0
Leader: 2     Replicas: 1,2,0   Isr: 2,0
```

Run a consumer and check the messages again

```
> bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --from-beginning --topic my-replicated-
topic
...
my test message 1
my test message 2
^C
```

Use Kafka Connect to import/export data

Create some Seed data

```
echo -e "foo\nbar" > test.txt
```

Start two connectors in "standalone" mode

```
> bin/connect-standalone.sh config/connect-standalone.
properties config/connect-file-source.properties config/
connect-file-sink.properties
```

Once the Kafka Connect process has started, the source connector should start reading lines from `test.txt` and producing them to the topic `connect-test`, and the sink connector should start reading messages from the topic `connect-test` and write them to the file `test.sink.txt`

Verify data has gone through pipeline by examining file

```
> cat test.sink.txt
foo
bar
```

Note that the data is being stored in the Kafka topic `connect-test` .

Run a console consumer to see the data in the topic

```
> bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --topic connect-test --from-beginning
{"schema":{"type":"string","optional":false},"payload":"foo"}
{"schema":{"type":"string","optional":false},"payload":"bar"}
...
```

Setting up Fluentd

Installing NTP Server

```
$sudo apt-get install ntp
```

Setup Restrict values in ntp.conf

Modify `/etc/ntp.conf` to ensure the following two restrict lines:

```
# Permit time synchronization with our time source, but do not
# Permit the source to query or modify the service on this
system.
restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery
```

Add local Clock as NTP backup and set NTP Log Parameters

Modify `/etc/ntp.conf` accordingly:

```
...
server  127.127.1.0 # local clock
fudge   127.127.1.0 stratum 10
…
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp.log
```

Restart the ntp service/daemon

```
$ service ntpd start
or
$ /etc/init.d/ntp start
```

Increase the Maximum number of File Descriptors

You can check the current number of file descriptors with the following command:

```
$ ulimit -n
65535
```

If your console shows 1024, it's insufficient. Update your `/etc/security/limits.conf` accordingly and reboot your machine:

```
root soft nofile 65536
root hard nofile 65536
* soft nofile 65536
* hard nofile 65536
```

Installing Fluentd on Debian

For Precise

```
$ curl -L
https://toolbelt.treasuredata.com/sh/install-ubuntu-
precise-td-agent2.sh | sh
```

Launch/stop/restart/ and check Fluentd daemon status

```
$ /etc/init.d/td-agent start
$ /etc/init.d/td-agent stop
$ /etc/init.d/td-agent restart
$ /etc/init.d/td-agent status
```

Understanding Fluentd configuration file

```
$ sudo nano /etc/td-agent/td-agent.conf
```

The configuration file consists of the following directives:

1. **source** directives determine the input sources.
2. **match** directives determine the output destinations.
3. **filter** directives determine the event processing pipelines.
4. **system** directives set system wide configuration.
5. **label** directives group the output and filter for internal routing
6. **@include** directives include other files.

More information can be found at **Fluentd documentation page.**

Setting up Fluentd consumer for Kafka

Upgrade Gradle on host

Surprise! You're probably not running the latest, so:

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:cwchien/gradle
$ sudo apt-get update
$ sudo apt-get install gradle
```

Upgrade Java on Host (at least to java 1.7)

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
$ sudo apt-get install oracle-java7-set-default

$ java -version
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) Client VM (build 24.80-b11, mixed
mode)
```

Clone kafka-fluentd-consumer and build it

```
$ sudo git clone
https://github.com/treasure-data/kafka-fluentd-consumer.
git

$ cd kafka-fluentd-consumer
$ gradle shadowJar
```

Configure Fluentd to send output to Treasure Data

Finally edit td-agent.conf to route output to Treasure Data (this step assumes you have saved your Treasure Data write only API key as an environment variable):

```
$ sudo nano /etc/td-agent/td-agent.conf

<source>
  type forward
</source>

<match td.*.*>
  type tdlog
  apikey "#{ENV['TD_API_KEY']}"

  auto_create_table
  buffer_type file
  buffer_path /var/log/td-agent/buffer/td
  flush_interval 5s

  <secondary>
  type file
  path /var/log/td-agent/failed_records
  </secondary>
</match>
```

Run Zookeeper, Kafka, Create a topic, send messages and run kafka-fluentd-consumer

```
# start zookeeper
$ bin/zookeeper-server-start.sh config/zookeeper.
properties

# start kafka
$ bin/kafka-server-start.sh config/server.properties

# create test topic
$ bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1
--topic test

# send multiple messages in JSON format (kafka-fluentd-
consumer
# requires this)
$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
{"a": 1}
{"a": 1, "b": 2}

# confirm the messages by starting a consumer
$ bin/kafka-console-consumer.sh --zookeeper
localhost:2181 --topic test --from-beginning
{"a": 1}
{"a": 1, "b": 2}
```

Modify `config/fluentd-consumer.properties` with an appropriate configuration. Don't forget to change to `fluentd.consumer.topics=test.` (or, accordingly to table name).

```
# Now run the kafka-fluentd-consumer.  Make sure that
paths are correct.
$ java -Dlog4j.configuration=file:///path/to/log4j.
properties -jar build/libs/kafka-fluentd-consumer-0.2.1-
all.jar config/fluentd-consumer.properties
```

This will forward logs to Fluentd (localhost:24224). This consumer uses log4j so you can change logging configurations via `-Dlog4j.configuration` argument.

Query your data on Treasure Data

Run the following query on Treasure Data console:

```
select * from test
```

You should see something like this: