**Apache Kafka**

Apache Kafka was originated at LinkedIn and later became an open sourced Apache project in 2011, then First-class Apache project in 2012. Kafka is written in Scala and Java. Apache Kafka is publish-subscribe based fault tolerant messaging system. It is fast, scalable and distributed by design.

What is Kafka?

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption.

Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss.

Kafka is built on top of the ZooKeeper synchronization service.

It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

### **Benefits**

Following are a few benefits of Kafka −

* **Reliability** − Kafka is distributed, partitioned, replicated and fault tolerance.
* **Scalability** − Kafka messaging system scales easily without down time..
* **Durability** − Kafka uses Distributed commit log which means messages persists on disk as fast as possible, hence it is durable..
* **Performance** − Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

**NOTE(Conclusion of Kafka)** By Apache Kafka site**:**

Kafka is a distributed streaming platform.

We think of a streaming platform as having three key capabilities:

1. It lets you publish and subscribe to streams of records. In this respect it is similar to a message queue or enterprise messaging system.
2. It lets you store streams of records in a fault-tolerant way.
3. It lets you process streams of records as they occur.



First a few concepts:

* Kafka is run as a cluster on one or more servers.
* The Kafka cluster stores streams of *records* in categories called *topics*.
* Each record consists of a key, a value, and a timestamp.

Kafka has four core APIs:

* The [Producer API](https://kafka.apache.org/documentation.html#producerapi) allows an application to publish a stream of records to one or more Kafka topics.
* The [Consumer API](https://kafka.apache.org/documentation.html#consumerapi) allows an application to subscribe to one or more topics and process the stream of records produced to them.
* The [Streams API](https://kafka.apache.org/documentation/streams) allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
* The [Connector API](https://kafka.apache.org/documentation.html#connect) allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

**Steps for installing and running kafka :**

Apache Kafka is a scalable and high-throughtput messaging system which is capable of efficiently handling a huge amount of data.

You can either deploy Kafka on one server or build a distributed Kafka cluster for greater performance. As a starter, this article explains how to install Apache Kafka on one single Vultr CentOS 7 server instance.

### **Prerequisites:**

Before moving on, you should:

* Deploy a Vultr CentOS 7 server instance. Depending on your needs, you may need to increase the available memory.
* Use a sudo user to log in from your SSH terminal.

### **Step 1: Update the CentOS 7 system**

Use the command below to update your system to the latest stable status:

sudo yum update -y && sudo reboot

After the reboot has finished, use the same sudo user to log in again.

### **Step 2: Install OpenJDK Runtime**

You need to setup a Java virtual machine on your system before you can run Apache Kafka properly. Here, you can install OpenJDK Runtime Environment 1.8.0 using YUM:

sudo yum install java-1.8.0-openjdk.x86\_64

Validate your installation with:

java -version

The output should resemble:

openjdk version "1.8.0\_91"

OpenJDK Runtime Environment (build 1.8.0\_91-b14)

OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)

You also need to setup the "JAVA\_HOME" and "JRE\_HOME" environment variables:

sudo vi /etc/profile

Append the following lines to the original content of the file:

export JAVA\_HOME=/usr/lib/jvm/jre-1.8.0-openjdk

export JRE\_HOME=/usr/lib/jvm/jre

Save and quit:

:wq

Reload the profile to put your changes into effect:

source /etc/profile

### **Step 3: Download Apache Kafka**

Download the latest stable version of Apache Kafka from the official website. At the time of writing, it's 0.9.0.1.

cd ~

wget http://www-us.apache.org/dist/kafka/0.9.0.1/kafka\_2.11-0.9.0.1.tgz

Unzip the archive to a preferred location, such as /opt:

tar -xvf kafka\_2.11-0.9.0.1.tgz

sudo mv kafka\_2.11-0.9.0.1 /opt

### **Step 4: Start and test Apache Kafka**

At this point, Apache Kafka is available on your system. Let's give it a test drive.

#### **4.1: Get into the Kafka directory**

cd /opt/kafka\_2.11-0.9.0.1

#### **4.2: Start the Zookeeper server**

bin/zookeeper-server-start.sh -daemon config/zookeeper.properties  
OR

bin/zookeeper-server-start.sh config/zookeeper.properties

#### **4.3: Modify the configuration of your Kafka server**

vi bin/kafka-server-start.sh

Adjust the memory usage according to your specific system parameters. For example, if you are using a Vultr server instance with 768MB memory in the test environment, you need to locate the following line:

export KAFKA\_HEAP\_OPTS="-Xmx1G -Xms1G"

Replace it with:

export KAFKA\_HEAP\_OPTS="-Xmx256M -Xms128M"

Save an quit:

:wq

#### **4.4: Start the Kafka server**

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

If everything went successfully, you will see several messages about the Kafka server's status, and the last one will read:

INFO [Kafka Server 0], started (kafka.server.KafkaServer)

This means that you have started the Kafka server.

#### **4.5: Create a topic "test" in a new SSH connection**

Open a new SSH connection, use the following commands to create a topic "test":

cd /opt/kafka\_2.11-0.9.0.1

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic testSandeep(topic name)

OR

bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic sandeepfirsttopic1 --partitions 2 --replication-factor 1

You can view your topics with the following command:

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

In our case, The output will read:

test

#### **4.6: Produce messages using the topic "test"**

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testsandeep

Using the command above, you can input any number of messages as you wish, such as:

Welcome aboard!

Bonjour!

If you receive an error similar to "WARN Error while fetching metadata with correlation id" while inputting a message, you'll need to update the server.properties file with the following info:

port = 9092

advertised.host.name = localhost

#### **4.7: Display messages**

Open a third SSH connection, and then run the following commands:

cd /opt/kafka\_2.11-0.9.0.1

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic testsandeep --from-beginning

OR(Imp will run use other cmd using process id of docker and go to existing kafka file “docker exec -it -t 72544edsc bash”)

Bin/kafka-console-consumer.sh –-topic testsandeep --from-beginning --zookeeper localhost:2181

Ta-da! The messages you produced earlier will display in the third SSH connection. Of course, if you input more messages from the second SSH connection now, you will immediately see them on the third SSH connection.

Finally, you can press Ctrl+C on each SSH connection to stop these scripts.