APACHE KAFKA

By Fabrizio Amorelli

Summary

INTRODUCTION 2

REQUIREMENTS 2

INSTALL 3

Microsoft Windows 3

Ubuntu Linux 3

Java 3

Kafka 3

CLI – COMMAND LINE INTERFACE 6

Windows 6

Linux 6

SSL / TLS 7

Generate new Authority 7

Create Java Truststore and Keystore 7

Broker configuration 8

CLI SSL – Command Line Interface SSL 9

Script for generate certificates 10

SASL SSL 10

CLI SASL SSL - Command Line Interface SASL SSL 12

TOOLS 13

Offset Explorer 13

Kafdrop – Kafka Web UI 18

Requirements 18

Installation 18

Kafdrop SSL Connection 21

Kafdrop SASL SSL Connection 22

Basic Authentication for Kafka UI 23

Clients for Developers 26

.NET Confluent.Kafka 26

Pyhton Confluent.Kafka 31

INTRODUCTION

Apache Kafka is an Open Source stream processing platform written in Java and Scala and developed by the Apache Software Foundation. The project aims to create a low-latency and high-speed platform for managing real-time data feeds.

This project is use principally for all real-time data stream processing applications, It's Free and Multi-Platform. This document will be explain how configure the message brokering system with Apache Kafka on Windows and Linux, step-by-step.

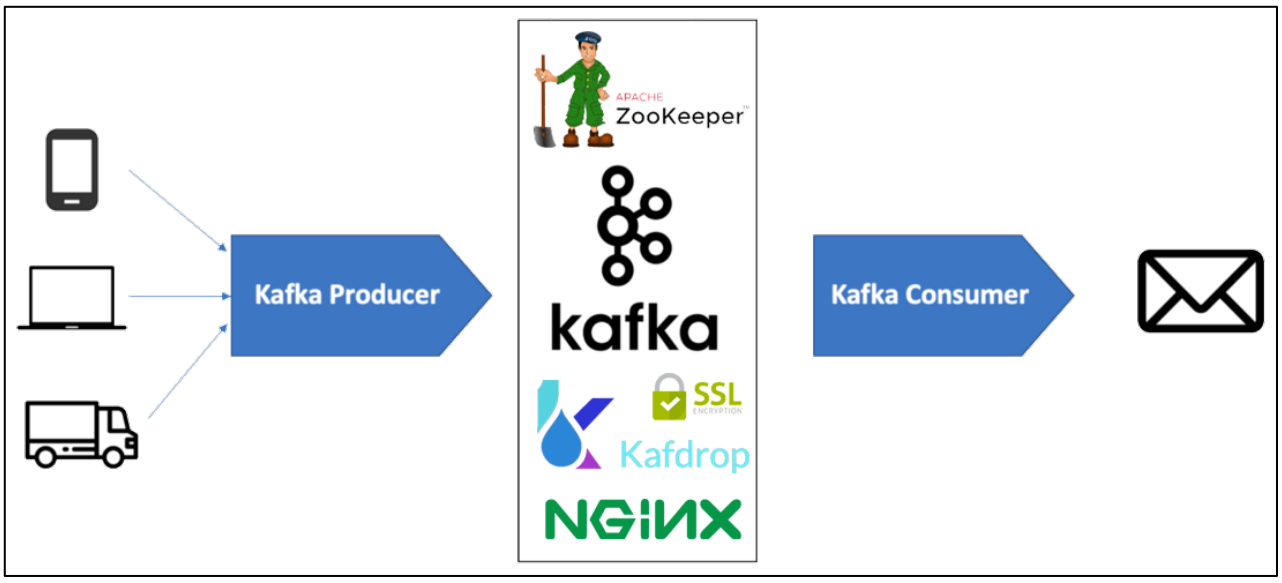
On this guide, I will use a single cluster with one Broker Server.
To encrypt and secure broker connection, I will enable also SSL/TLS and SASL SSL.

I will use the CLI to test Producer and Consumer connection, after some tools like Offset Explorer and I will install Kafdrop Web UI with NGINX Basic Authentication and SSL for HTTPS connections.

In the end, I will test Kafka also with some program written with Confluent.Kafka in .NET and Python.

NOTE: this guide is update to 07/2022 with Kakfa 2.13-3.2.0
In a future release, Apache Zookeeper will no longer be necessary.

DISCLAIMER: this document is a guide based on various documentations and examples find on the Internet about Open Source and Free Software. Some of the procedures shown in this document may be incorrect or not ideal for production environments. It's advisable always to contact an Expert.



REQUIREMENTS

OS: Windows, Linux, MacOS

Java ≥ 11

Minimum requirements

Memory	8 GB RAM
CPU	4 cores
Disk	500 GB
LAN	1 GbE-10 Gbe

INSTALL

Microsoft Windows

1. Download and install Java (https://www.oracle.com/java/technologies/downloads/)
2. Download latest Kafka version (https://kafka.apache.org/downloads) from Binary downloads
3. Extract and copy kafka_n.nn_n.nn folder in C:\
4. Edit *config/server.properties* and set logs path (where partitions files will be write)

```
log.dirs=c:/kafka/kafka-logs
```

5. Edit *config/zookeeper.properties* and set dataDir path

```
dataDir=c:/kafka/zookeeper-data
```

Ubuntu Linux

Java

1. Check update

```
sudo apt update
```

2. Install java

```
sudo apt install default-jre
```

3. Check Java version

```
java –version
```

4. Install Java JDK (optional)

```
sudo apt install default-jdk
```

5. Check JDK version

```
javac -version
```

Kafka

1. Download latest package from website

```
wget https://dlcdn.apache.org/kafka/3.2.0/kafka_2.13-3.2.0.tgz
```

2. Extract folder

```
tar -xvzf kafka_2.13-3.2.0.tgz
```

3. Install folder

```
sudo mv kafka_2.13-3.2.0 /usr/local/kafka
```

4. Create Zookeeper Service

```
sudo nano /etc/systemd/system/zookeeper.service
```

Insert

```
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
Type=simple
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

5. Create Kafka Service

```
sudo nano /etc/systemd/system/kafka.service
```

Insert

```
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service
After=zookeeper.service

[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh

[Install]
WantedBy=multi-user.target
```

6. Reload daemons

```
sudo systemctl daemon-reload
```

7. Change logs path in */usr/local/kafka/config/server.properties*
   (by default Kafka use system temp folder, on reboot all data will be removed)

```
log.dirs=/var/kafka-logs
```

8. Change snapshot dataDir path in */usr/local/kafka/config/zookeeper.properties*
   (by default Zookeeper use system temp folder, on reboot all data will be removed)

```
dataDir=/var/zookeeper
```

9. Enable services

```
sudo systemctl enable zookeeper
sudo systemctl enable kafka
```

10. Start services

```
sudo systemctl start zookeeper
sudo systemctl start kafka
```

11. Check services

```
sudo systemctl status zookeeper
sudo systemctl status kafka
```

NOTE: if there are some issue or errors, check paths on configurations files.
Check also that kafka folder has correct permissions for execute.

CLI – COMMAND LINE INTERFACE

Here are some commands line to start and use Kafka.
NOTE: for broker connection with SSL go to P. 9
For broker connection with SASL SSL go to P. 12

Windows

Start Zookeeper and Kafka

```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

New TOPIC

```
.\bin\windows\kafka-topics.bat --create --topic <topicname> --bootstrap-server <host>:<port>
```

Write messages on TOPIC

```
.\bin\windows\kafka-console-producer.bat --topic <topicname> --bootstrap-server <host>:<port>
[message1]
[message2]
```

Consume TOPIC messages

```
.\bin\windows\kafka-console-consumer.bat --topic <topicname> --bootstrap-server <host>:<port>
[message1]
[message2]
```

Linux

Start Zookeeper and Kafka

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

New TOPIC

```
./bin/kafka-topics.sh --create --topic <topicname> --bootstrap-server <host>:<port>
```

Write messages on TOPIC

```
./bin/kafka-console-producer.sh --topic <topicname> --bootstrap-server <host>:<port>
[message1]
[message2]
```

Consume TOPIC messages

```
./bin/kafka-console-consumer.sh--topic <topicname> --bootstrap-server <host>:<port>
[message1]
[message2]
```

SSL / TLS

Here are the procedure to enable SSL protocol for bootstrap server Kafka.
All commands are the same for Linux and Windows.
You can find OpenSSL for Windows here: https://slproweb.com/products/Win32OpenSSL.html
NOTE: on windows if It doesn't have env variable you can call programs with full path, for example:

```
"C:\Program Files\OpenSSL-Win64\bin\openssl.exe"
"%JAVA_HOME%\bin\keytool.exe"
```

This part is based on this guide: https://github.com/LGouellec/kafka-dotnet-ssl

Generate new Authority

Generate new private key

```
openssl genrsa -out root.key
```

Generate certificate authority

```
openssl req -new -x509 -key root.key -out root.crt
```

Create Java Truststore and Keystore

Generate truststore

```
keytool -keystore kafka.truststore.jks -alias CARoot -import -file root.crt
```

Generate keystore

```
keytool -keystore kafka01.keystore.jks -alias localhost -validity 365 -genkey -keyalg RSA -ext SAN=DNS:kafkahostname
```

Export certificate broker

```
keytool -keystore kafka01.keystore.jks -alias localhost -certreq -file kafka01.unsigned.crt
```

Sign certificate with CA

```
openssl x509 -req -CA root.crt -CAkey root.key -in kafka01.unsigned.crt -out kafka01.signed.crt -days 365 -CAcreateserial
```

NOTE: "-days 365" set validity of certificate in this case for 1 year

Import CA certificate on Broker Keystore

```
keytool -keystore kafka01.keystore.jks -alias CARoot -import -file root.crt
```

Import certificate on Broker Keystore

```
keytool -keystore kafka01.keystore.jks -alias localhost -import -file kafka01.signed.crt
```

Broker configuration

When certificates are correctly available, you must configure Kafka Broker.
If you have multiple Broker the same procedure must be applied on all configurations server files.
This example is based as explained in the introduction (P. 2) on single cluster and one broker server.
NOTE: the procedure is made on Linux and It's comparable to Windows

1. Stop Kafka

```
sudo systemctl stop kafka
```

2. Edit server configurations file, for example

```
sudo nano /usr/local/kafka/config/server.properties
```

3. Set the parameters

```
listeners=SSL://:9093
advertised.listeners=SSL://:9093
security.inter.broker.protocol=SSL
ssl.truststore.location=/home/fabri/kafka.truststore.jks
ssl.truststore.password=123456789
ssl.keystore.location=/home/fabri/kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.key.password=123456789
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
ssl.client.auth=required
ssl.endpoint.identification.algorithm=
ssl.keystore.type=JKS
ssl.truststore.type=JKS
```

4. Windows version

```
listeners=SSL://:9093
advertised.listeners=SSL://:9093
security.inter.broker.protocol=SSL
ssl.truststore.location= C://kafka//ssl//kafka.truststore.jks
ssl.truststore.password=123456789
ssl.keystore.location= C://kafka//ssl//kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.key.password=123456789
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
ssl.client.auth=required
ssl.endpoint.identification.algorithm=
ssl.keystore.type=JKS
ssl.truststore.type=JKS
```

5. Start Kafka

```
sudo systemctl start kafka
```

After with this configuration, broker will accept only secure connection on 9093 port.
If you want It's possible enable another instance without SSL on different port, for example

```
listeners=PLAINTEXT://:9092,SSL://:9093
advertised.listeners=PLAINTEXT://:9092,SSL://:9093
```

CLI SSL – Command Line Interface SSL

Here are some examples for CLI connections with SSL
Create a new file for example *client-ssl.properties* and set parameters

```
bootstrap.servers=192.168.1.61:9093
security.protocol=SSL
ssl.truststore.location=/home/fabri/ssl3/kafka.truststore.jks
ssl.truststore.password=123456
ssl.keystore.location=/home/fabri/ssl4/client.keystore.jks
ssl.keystore.password=123456
ssl.key.password=123456
ssl.enabled.protocols=TLSv1.2
ssl.client.auth=required
ssl.endpoint.identification.algorithm=
ssl.keystore.type=JKS
ssl.truststore.type=JKS
```

NOTE: on windows certificate paths must be set with escape, for example

```
bootstrap.servers=192.168.1.61:9093
security.protocol=SSL
ssl.truststore.location= C:\\Users\\Fabrizio\\Desktop\\client.truststore.jks
ssl.truststore.password=123456
ssl.keystore.location= C:\\Users\\Fabrizio\\Desktop\\client.keystore.jks
ssl.keystore.password=123456
ssl.key.password=123456
ssl.enabled.protocols=TLSv1.2
ssl.client.auth=required
ssl.endpoint.identification.algorithm=
ssl.keystore.type=JKS
ssl.truststore.type=JKS
```

Producer

```
bin/kafka-console-producer --broker-list kafka1:9093 --topic test --producer.config client-ssl.properties
```

Consumer

```
bin/kafka-console-consumer --bootstrap-server kafka1:9093 --topic test --consumer.config client-ssl.properties --from-beginning
```

Script for generate certificates

You can find on the shared folder two scripts for generate certificates with the same commands shown.

Script for Windows: https://github.com/Fabrizio04/Kafka/tree/main/SSL_Generate_Windows

Script for Linux: https://github.com/Fabrizio04/Kafka/tree/main/SSL_Generate_Linux

SASL SSL

Here are the procedure to enable SASL Authentication with SSL.
The procedure for generate SSL certificates is the same of previous chapter.
You can also enable only SASL without SSL (SASL PLAINTEXT).
This part is based on this guide: https://bit.ly/3cSYFub
NOTE: the procedure is made on Linux and It's comparable to Windows.

1. Stop Kafka and Zookeeper

```
sudo systemctl stop kafka
sudo systemctl stop zookeeper
```

2. Edit *zookeeper.properties* file and set

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
requireClientAuthScheme=sasl
```

3. Create on config folder a new file *zookeeper_jaas.conf* and set

```
Server {
      org.apache.zookeeper.server.auth.DigestLoginModule required
      user_super="admin-secret"
      user_kafka="kafka-secret";
};
```

4. Create on config folder a new file *kafka_server_jaas.conf* and set

```
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret";
};

Client {
   org.apache.zookeeper.server.auth.DigestLoginModule required
   username="kafka"
   password="kafka-secret";
};
```

5. Edit zookeeper service and set environment variable KAFKA_OPTS

```
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
Type=simple
Environment="KAFKA_OPTS=-Djava.security.auth.login.config=/usr/local/kafka/config/zookeeper_jaas.conf"
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

6. Edit kafka service and set environment variable KAFKA_OPTS

```
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service
After=zookeeper.service

[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
Environment="KAFKA_OPTS=-Djava.security.auth.login.config=/usr/local/kafka/config/kafka_server_jaas.conf"
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh

[Install]
WantedBy=multi-user.target
```

NOTE: if you run manually zookeeper and kafka, you must export the two variables before execute script, for example on Linux:

```
export KAFKA_OPTS="-Djava.security.auth.login.config=/KAFKA_HOME/config/zookeeper_jaas.conf"
zookeeper-server-start.sh
export KAFKA_OPTS="-Djava.security.auth.login.config=/KAFKA_HOME/config/kafka_server_jaas.conf"
kafka-server-start.sh
```

On Windows:

```
set "KAFKA_OPTS=-Djava.security.auth.login.config=c:\kafka\config\zookeeper_jaas.conf"
zookeeper-server-start.bat
set "KAFKA_OPTS=-Djava.security.auth.login.config=c:\kafka\config\kafka_server_jaas.conf"
kafka-server-start.bat
```

7. Reload daemons

```
sudo systemctl daemon-reload
```

8. Edits server properties configuration file and set

```
listeners=PLAINTEXT://:9092,SASL_PLAINTEXT://:9093,SASL_SSL://:9094
advertised.listeners=PLAINTEXT://:9092,SASL_PLAINTEXT://:9093,SASL_SSL://:9094

security.inter.broker.protocol=SASL_SSL

sasl.enabled.mechanisms=PLAIN
sasl.mechanism.inter.broker.protocol=PLAIN

authorizer.class.name=kafka.security.authorizer.AclAuthorizer
allow.everyone.if.no.acl.found=true
auto.create.topics.enable=false
```

9. Start Zookeeper and Kafka

```
sudo systemctl start zookeeper
sudo systemctl start kafka
```

CLI SASL SSL - Command Line Interface SASL SSL

Here an example for CLI connections with SASL SSL.
Remember to set environment variable as explained previously.

Create a new file for example *client-ssl-sasl.properties* and set parameters

```
bootstrap.servers=192.168.1.61:9094

sasl.mechanism=PLAIN
security.protocol=SASL_SSL

sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
  username="admin" \
  password="admin-secret";

ssl.truststore.location=/home/fabri/ssl3/kafka.truststore.jks
ssl.truststore.password=123456
ssl.keystore.location=/home/fabri/ssl4/client.keystore.jks
ssl.keystore.password=123456
ssl.key.password=123456
ssl.enabled.protocols=TLSv1.2
ssl.client.auth=required
ssl.endpoint.identification.algorithm=
ssl.keystore.type=JKS
ssl.truststore.type=JKS
```

Create a new file *kafka_client_jaas.conf* and set

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="admin"
  password="admin-secret";
};
Client {
org.apache.zookeeper.server.auth.DigestLoginModule required
  username="kafka"
  password="kafka-secret";
};
```

Producer

```
export KAFKA_OPTS="-Djava.security.auth.login.config=/usr/local/kafka/config/kafka_client_jaas.conf"
bin/kafka-console-producer --broker-list localhost:9094 --topic test --producer.config client-ssl-sasl.properties
```

Consumer

```
export KAFKA_OPTS="-Djava.security.auth.login.config=/usr/local/kafka/config/kafka_client_jaas.conf"
bin/kafka-console-consumer --bootstrap-server localhost:9094 --topic test --consumer.config client-ssl-sasl.properties --from-beginning
```

TOOLS

Offset Explorer

Offset Explorer (formerly Kafka Tool) is a GUI application for managing and using Apache Kafka clusters. It provides an intuitive UI that allows one to quickly view objects within a Kafka cluster as well as the messages stored in the topics of the cluster. It contains features geared towards both developers and administrators. Some of the key features include:

- Quickly view all your Kafka clusters, including their brokers, topics and consumers
- View contents of messages in your partitions and add new messages
- View offsets of the consumers, including Apache Storm Kafka spout consumers
- Show JSON, XML and Avro messages in a pretty-printed format
- Add and drop topics plus other management features
- Save individual messages from your partitions to local hard drive
- Write your own plugins that allow you to view custom data formats
- Offset Explorer runs on Windows, Linux and Mac OS

Website: https://www.kafkatool.com/

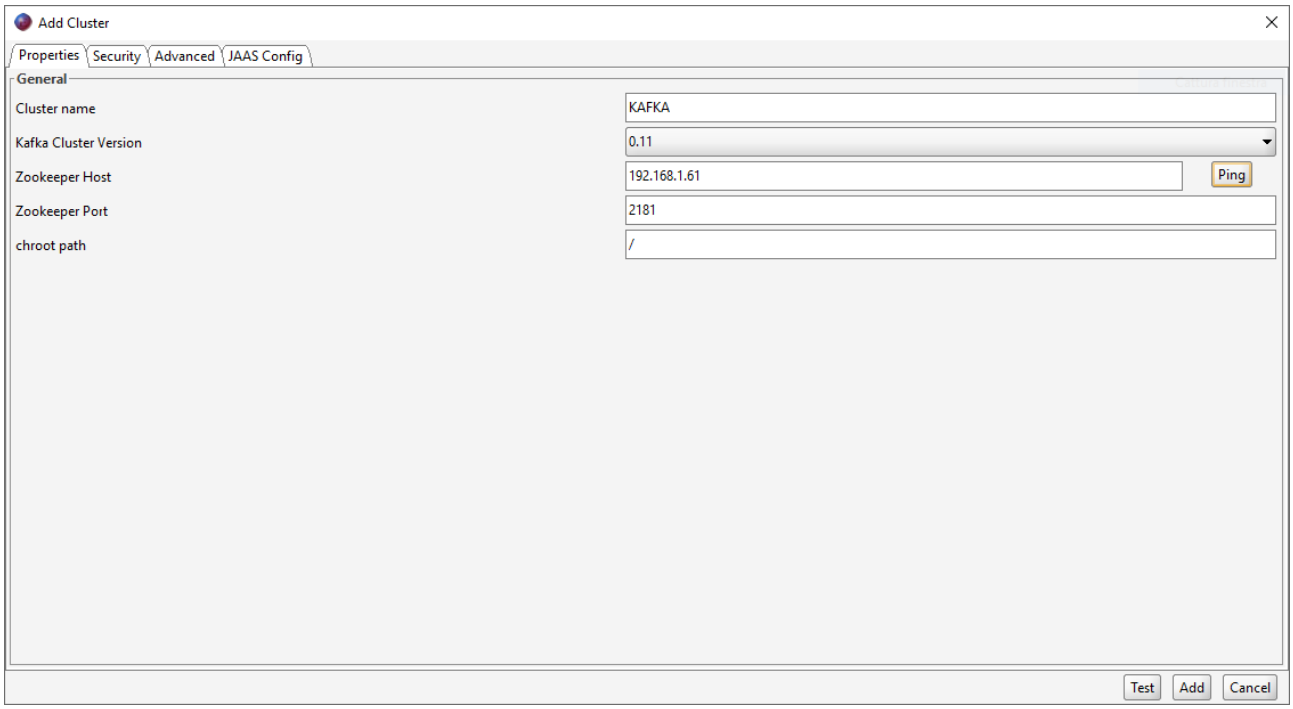
Download: https://www.kafkatool.com/download.html

Documentation: https://www.kafkatool.com/documentation/connecting.html

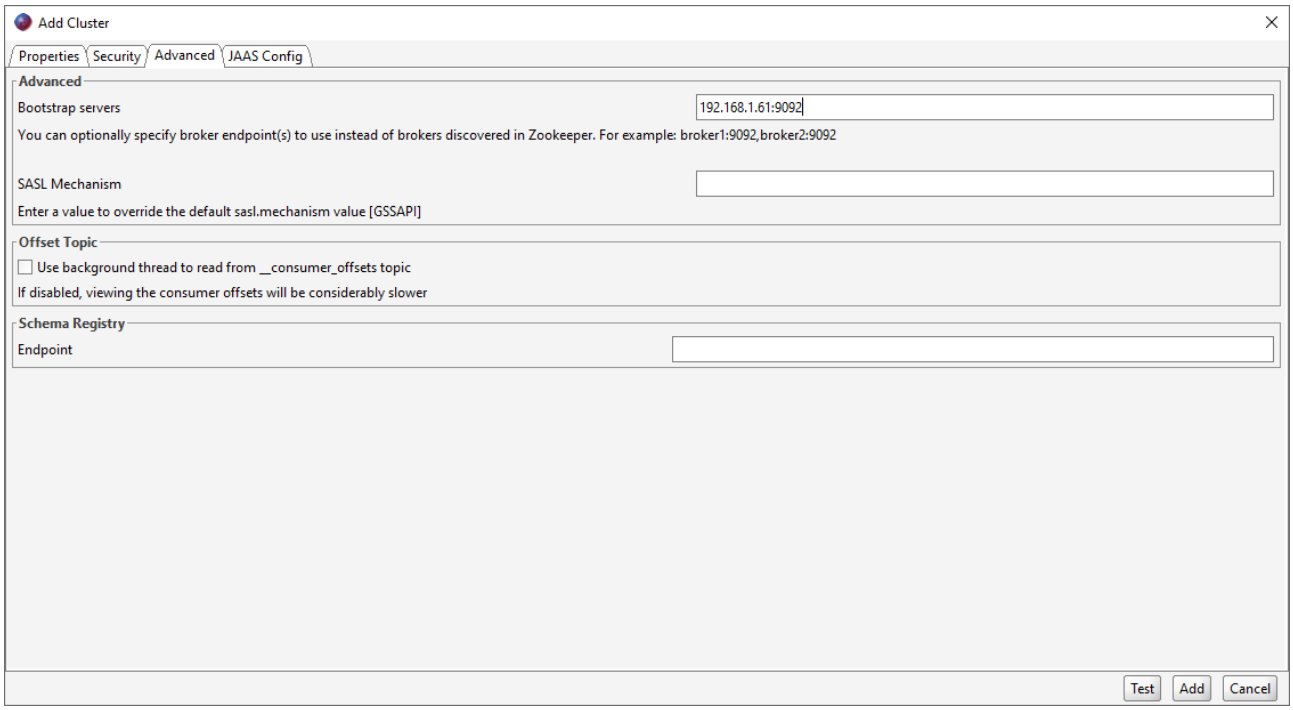
NOTE: if you have some issue with broker connection, check if DNS can resolve hostname.
If DNS cannot resolving name, for example You can manually add the resolve on configuration hosts file

```
Windows   "C:\Windows\System32\drivers\etc\hosts"
Linux     /etc/hosts
```

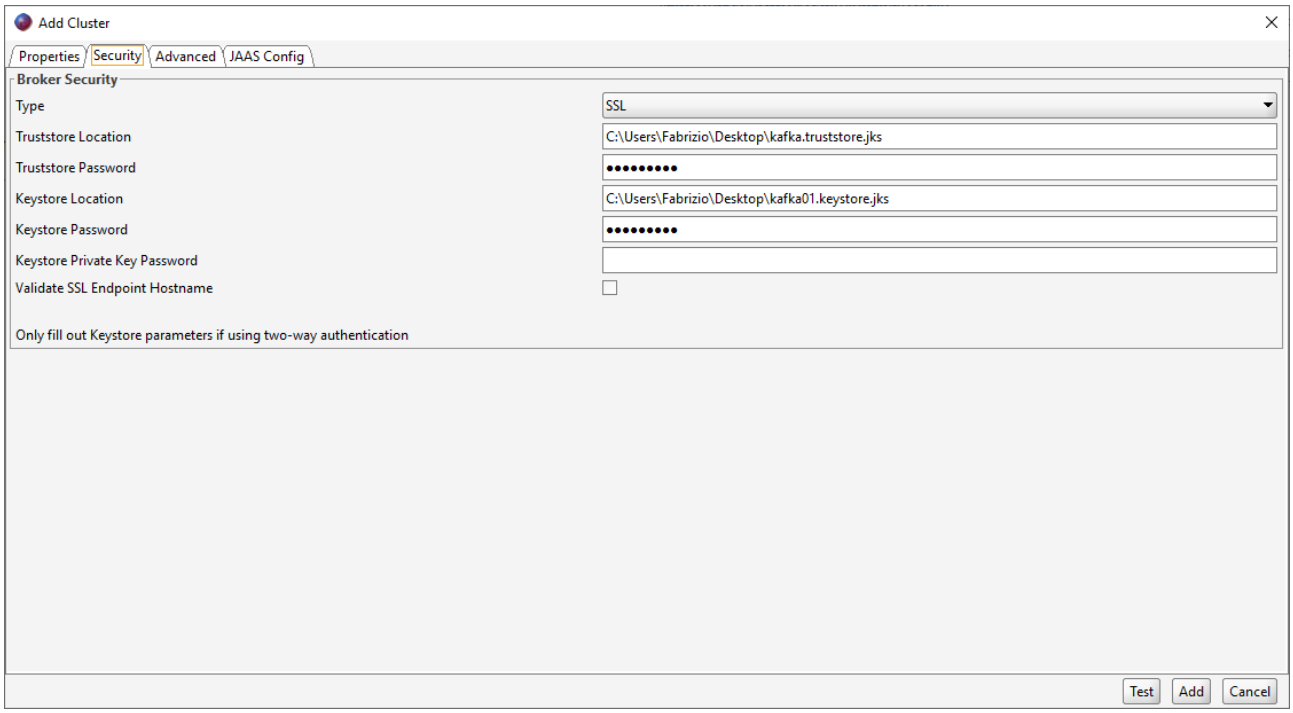
After installation, add a new Cluster for example



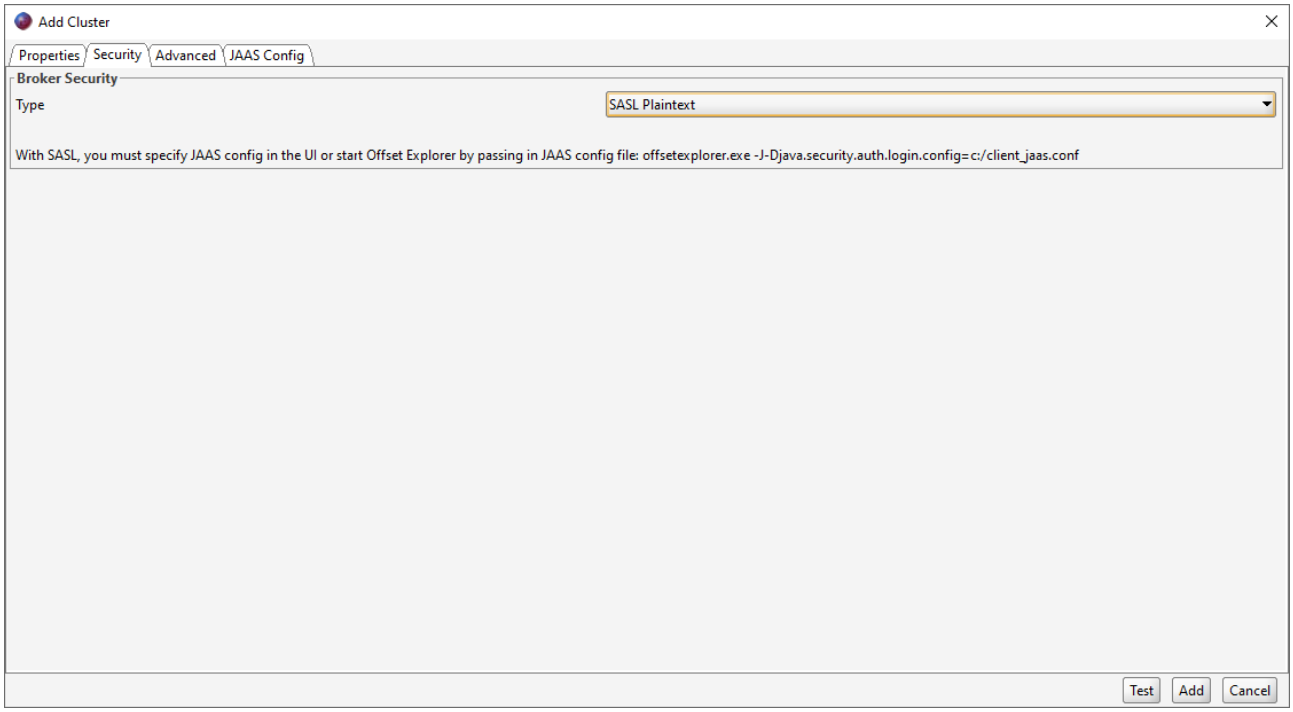
Specific the bootstrap server of the Cluster

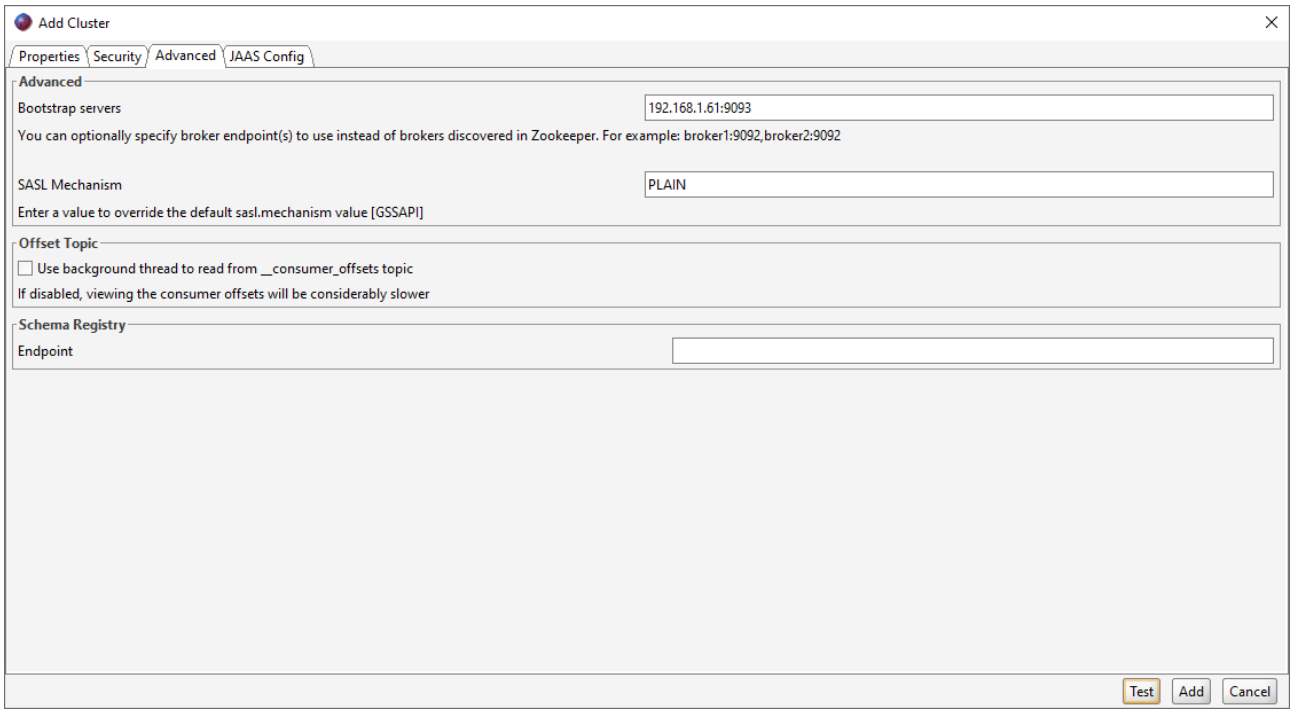


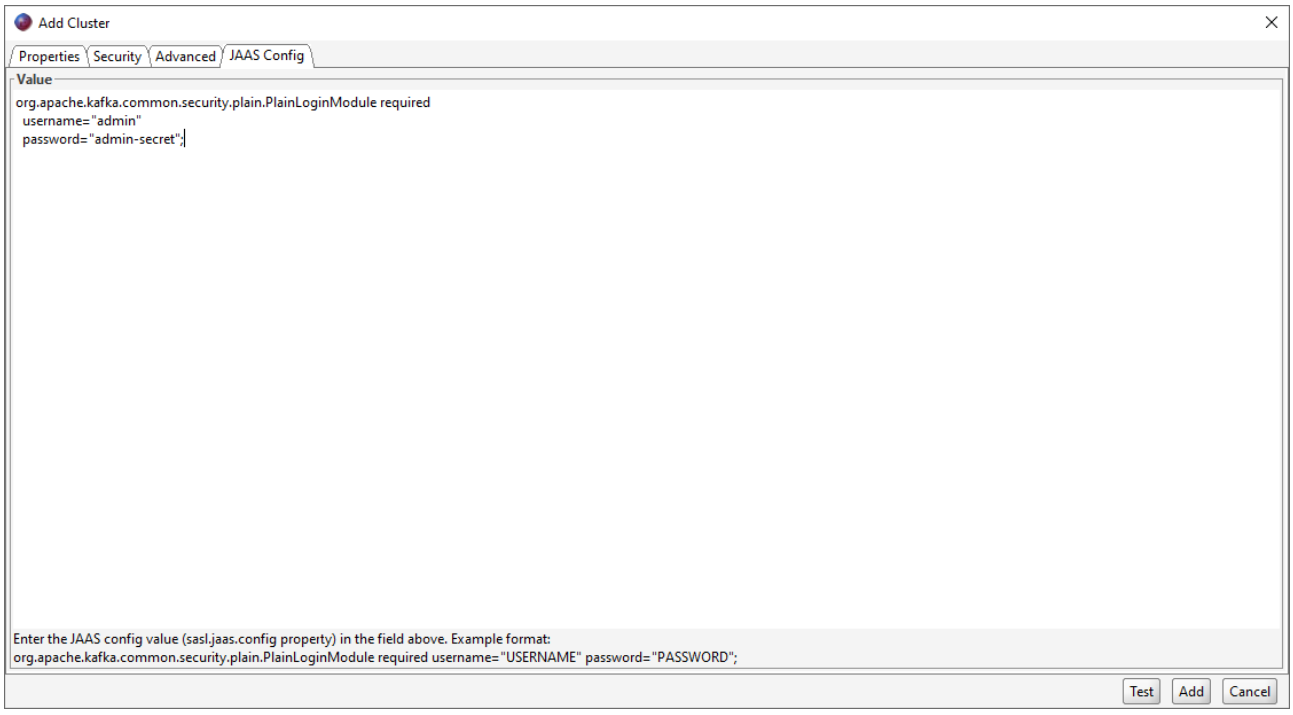
SSL Connection



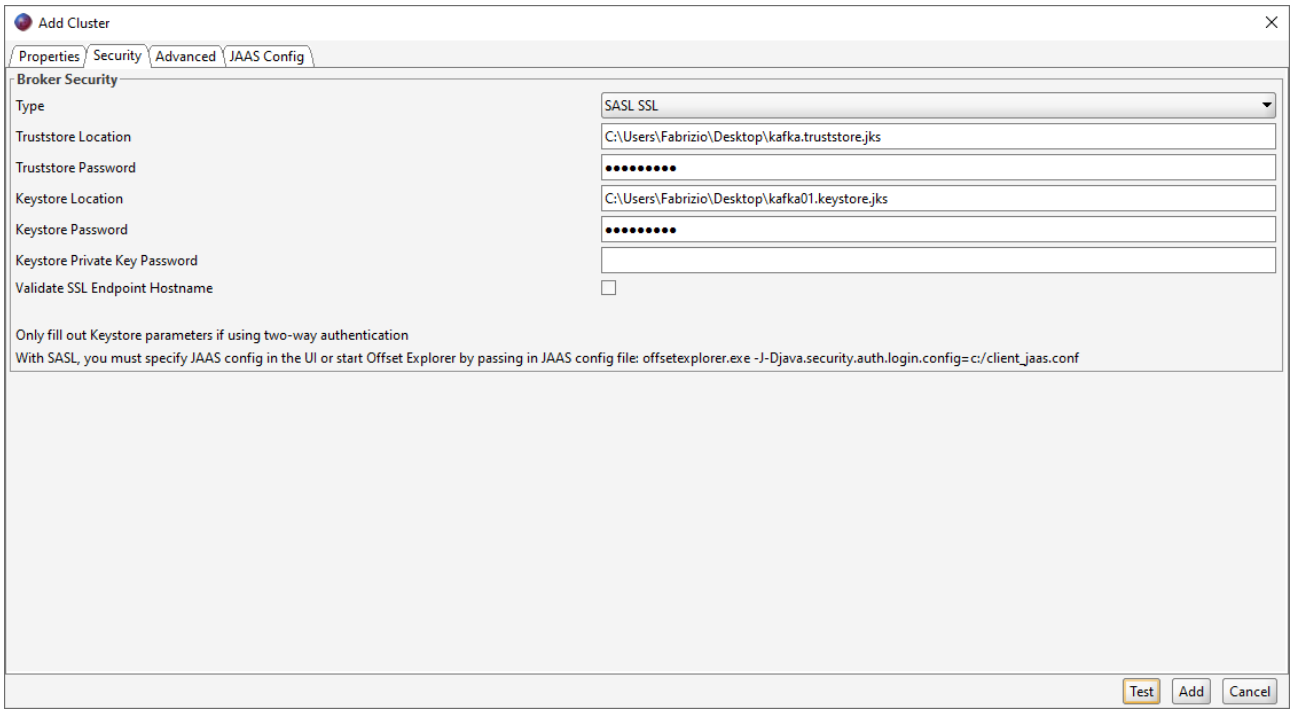
SASL Plaintext

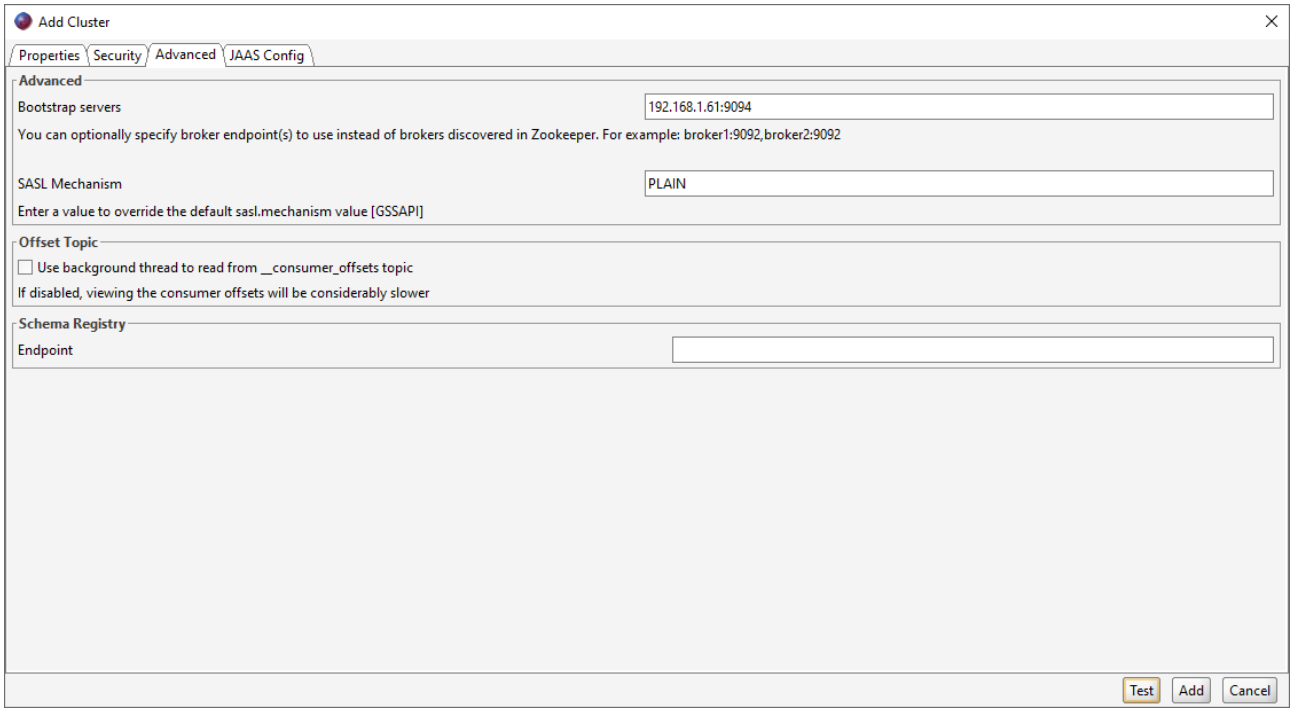


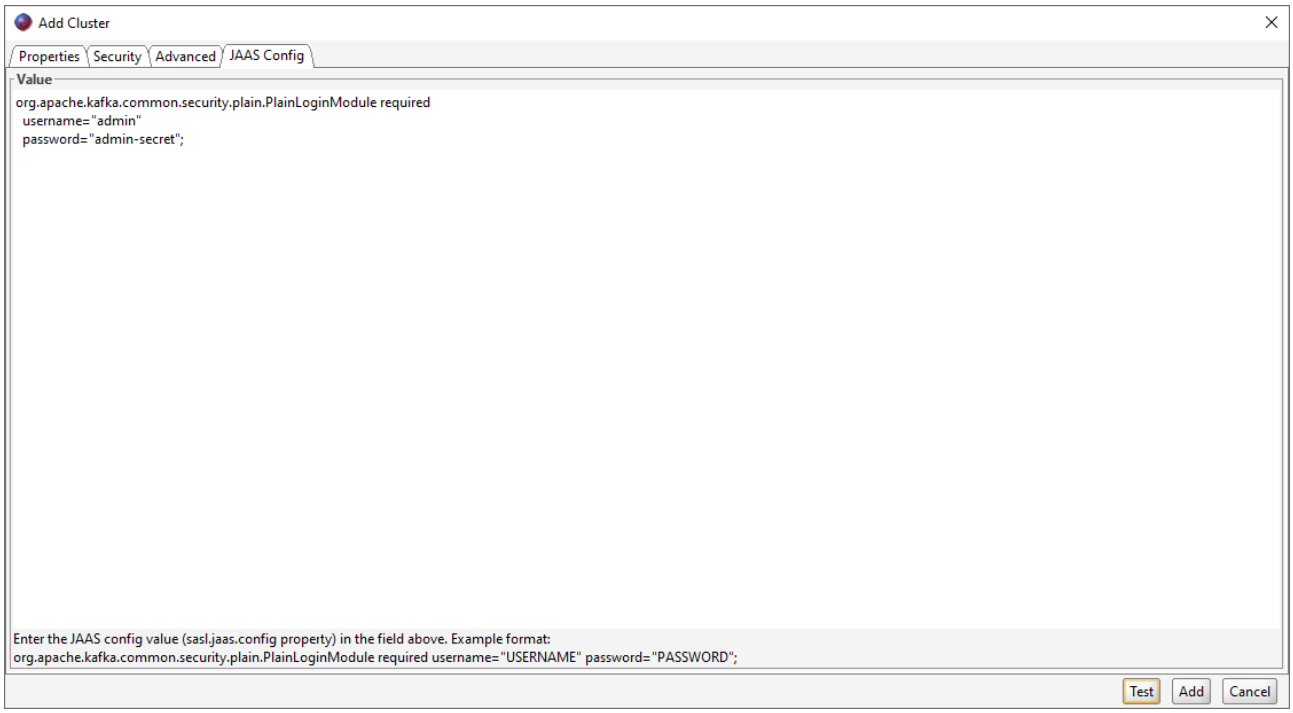

Add Cluster
Properties
Security
Advanced
JAAS Config
Advanced
Bootstrap servers
192.168.1.61:9093
You can optionally specify broker endpoint(s) to use instead of brokers discovered in Zookeeper. For example: broker1:9092,broker2:9092
SASL Mechanism
PLAIN
Enter a value to override the default sasl.mechanism value [GSSAPI]
Offset Topic
Use background thread to read from __consumer_offsets topic
If disabled, viewing the consumer offsets will be considerably slower
Schema Registry
Endpoint
Test
Add
Cancel


Add Cluster
Properties
Security
Advanced
JAAS Config
Value
org.apache.kafka.common.security.plain.PlainLoginModule required
username="admin"
password="admin-secret";
Enter the JAAS config value (sasl.jaas.config property) in the field above. Example format:
org.apache.kafka.common.security.plain.PlainLoginModule required username="USERNAME" password="PASSWORD";
Test
Add
Cancel

SASL SSL


Add Cluster
Properties
Security
Advanced
JAAS Config
Broker Security
Type
SASL SSL
Truststore Location
C:\Users\Fabrizio\Desktop\kafka.truststore.jks
Truststore Password
Keystore Location
C:\Users\Fabrizio\Desktop\kafka01.keystore.jks
Keystore Password
Keystore Private Key Password
Validate SSL Endpoint Hostname
Only fill out Keystore parameters if using two-way authentication
With SASL, you must specify JAAS config in the UI or start Offset Explorer by passing in JAAS config file: offsetexplorer.exe -J-Djava.security.auth.login.config=c:/client_jaas.conf
Test
Add
Cancel


Add Cluster
Properties
Security
Advanced
JAAS Config
Advanced
Bootstrap servers
192.168.1.61:9094
You can optionally specify broker endpoint(s) to use instead of brokers discovered in Zookeeper. For example: broker1:9092,broker2:9092
SASL Mechanism
PLAIN
Enter a value to override the default sasl.mechanism value [GSSAPI]
Offset Topic
Use background thread to read from __consumer_offsets topic
If disabled, viewing the consumer offsets will be considerably slower
Schema Registry
Endpoint
Test
Add
Cancel



Kafdrop – Kafka Web UI

Kafdrop is a Web Interface for Topics view and Consumers Groups.

This tool show info about brokers, topics, partitions, consumers and messages.

Website: https://github.com/obsidiandynamics/kafdrop

Requirements

Java ≥ 11

Kafka ≥ 0.11.0

Installation

Windows

1. Download latest version from https://github.com/obsidiandynamics/kafdrop/releases

2. Open commands prompt and run

```
java -jar kafdrop-3.30.0.jar
```

3. Open commands prompt and run

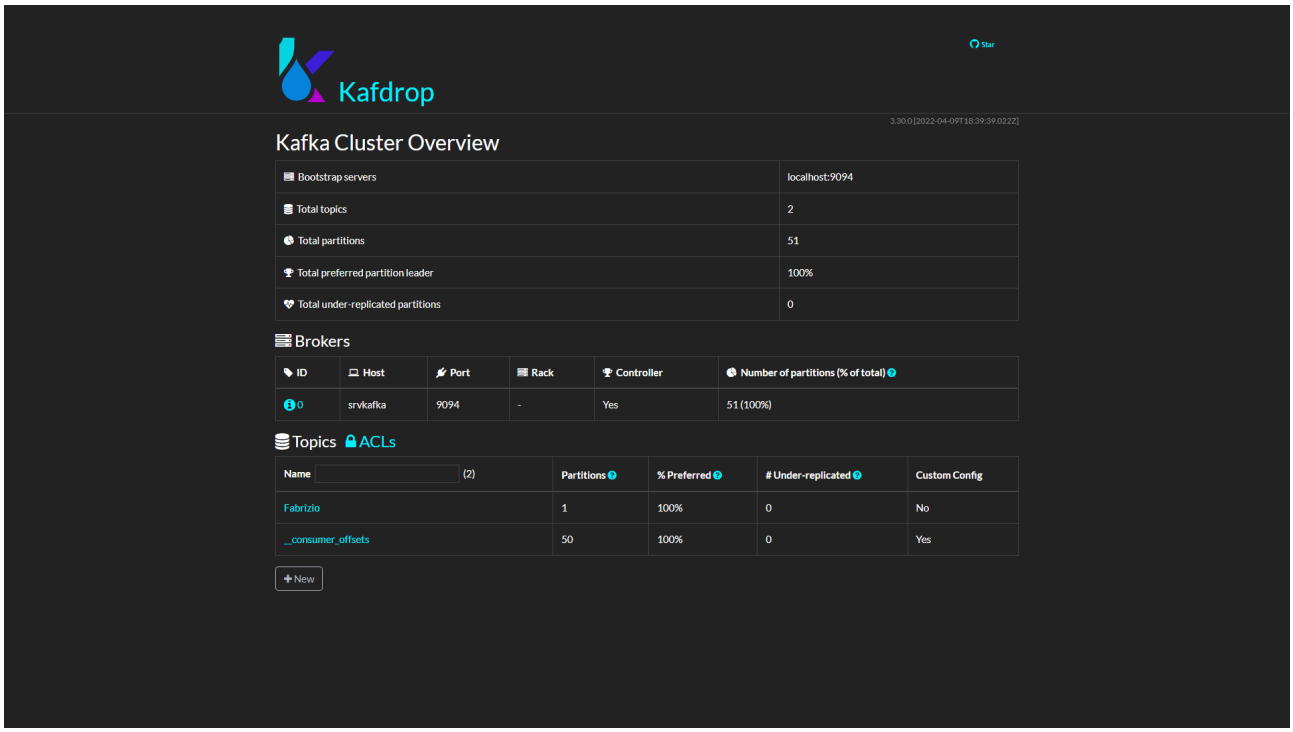
```
java -jar kafdrop-3.30.0.jar
```

By default Kafdrop try to connect with PLAINTEXT on localhost:9092
You can specific kafka servers with the parameter:

```
java -jar kafdrop-3.30.0.jar --kafka.brokerConnect=<host:port,host:port>
```

Open web interface on a Browser on the ip/host server at port 9000, for example

```
http://192.168.1.61:9000/
```



Linux

1. Download Kafdrop

```
wget https://github.com/obsidiandynamics/kafdrop/releases/download/3.30.0/kafdrop-3.30.0.jar
```

2. Copy Java Executable file in a folder and set execution privileges

```
sudo mv kafdrop-3.30.0.jar /usr/local/kafka/kafdrop
sudo chmod +x kafdrop-3.30.0.jar
```

3. Create a bash script *kafdrop.sh* and set start commands for example

```
#!/bin/sh
cd /usr/local/kafka/kafdrop/
sudo /usr/bin/java -jar kafdrop-3.30.0.jar --kafka.brokerConnect=localhost:9093
```

4. Create the Kafdrop Service

```
sudo nano /etc/systemd/system/kafdrop.service
```

5. Set into file service

```
[Unit]
Description=Kafdrop UI
Requires=kafka.service
After=kafka.service

[Service]
ExecStart=/usr/local/kafka/kafdrop/kafdrop.sh
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

6. Reload daemon

```
sudo systemctl daemon-reload
```

7. Enable service

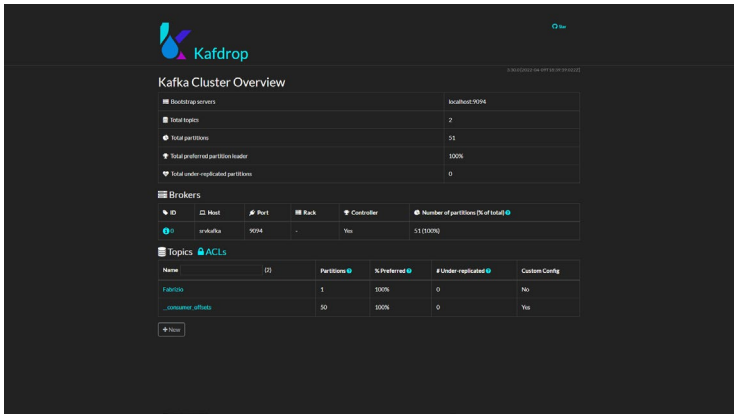
```
sudo systemctl enable kafdrop
```

8. Start service and check status

```
sudo systemctl start kafdrop
sudo systemctl status kafdrop
```

If You have some issue or error, please remember to set execution privileges on executable jar file and start bash script. When service is active, try to connect with Browser for example

```
http://192.168.1.61:9000/
```



Kafdrop SSL Connection

Here are the procedure to set secure connection for Kafdrop vs broker servers.
NOTE: the procedure is made on Linux and It's comparable to Windows.

1. Create a new file with this exactly name ***kafka.properties*** on kafdrop executable jar file folder path Set

```
security.protocol=SSL
ssl.endpoint.identification.algorithm=
ssl.protocol=TLS
ssl.key.password=123456789
ssl.keystore.location=/home/fabri/kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.keystore.type=JKS
ssl.truststore.location=/home/fabri/kafka.truststore.jks
ssl.truststore.password=123456789
ssl.truststore.type=JKS
```

   On windows paths will be with escape for example

```
security.protocol=SSL
ssl.endpoint.identification.algorithm=
ssl.protocol=TLS
ssl.key.password=123456789
ssl.keystore.location=C:\\Users\\Fabrizio\\Desktop\\Genera_Windows\\kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.keystore.type=JKS
ssl.truststore.location=C:\\Users\\Fabrizio\\Desktop\\Genera_Windows\\kafka.truststore.jks
ssl.truststore.password=123456789
ssl.truststore.type=JKS
```

2. Reload Kafdrop service

```
sudo systemctl stop kafdrop
sudo systemctl start kafdrop
```

After service is active, try to connect with Browser always on port 9000

Kafdrop SASL SSL Connection

Here are the procedure to set secure connection for Kafdrop vs broker servers.
NOTE: the procedure is made on Linux and It's comparable to Windows.

1. Create a new file with this exactly name ***kafka.properties*** on kafdrop executable jar file folder path Set

```
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
ssl.endpoint.identification.algorithm=
ssl.protocol=TLS
ssl.key.password=123456789
ssl.keystore.location=/home/fabri/kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.keystore.type=JKS
ssl.truststore.location=/home/fabri/kafka.truststore.jks
ssl.truststore.password=123456789
ssl.truststore.type=JKS
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="admin" password="admin-secret";
```

   On windows paths will be with escape for example

```
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
ssl.endpoint.identification.algorithm=
ssl.protocol=TLS
ssl.key.password=123456789
ssl.keystore.location=C:\\Users\\Fabrizio\\Desktop\\Genera_Windows\\kafka01.keystore.jks
ssl.keystore.password=123456789
ssl.keystore.type=JKS
ssl.truststore.location=C:\\Users\\Fabrizio\\Desktop\\Genera_Windows\\kafka.truststore.jks
ssl.truststore.password=123456789
ssl.truststore.type=JKS
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="admin" password="admin-secret";
```

2. Reload Kafdrop service

```
sudo systemctl stop kafdrop
sudo systemctl start kafdrop
```

After service is active, try to connect with Browser always on port 9000

Basic Authentication for Kafka UI

As described on the official documentation, Kafdrop doesn't support authentication for Web Interface.
To enable a Basic Authentication, I will install and set NGINX Web Server with Reverse Proxy.
Here are the procedure on Linux, on Windows it's slightly different but anyway easy.

1. Install packets

```
sudo apt update
sudo apt install nginx
sudo apt install apache2-utils
```

2. Create script for credentials for example

```
sudo htpasswd -c /usr/local/kafka/.htpasswd
```

3. Edit default configuration file

```
sudo nano /etc/nginx/conf.d/default.conf
```

4. Set for example this configuration

```
server {

    listen              80;
    server_name    192.168.1.61;

    auth_basic "Restricted Area";
    auth_basic_user_file /usr/local/kafka/.htpasswd;


    location / {
      proxy_pass http://127.0.0.1:9000;
    }

    location /logout {
     return 401;
    }

}
```

```
server {

        listen          80;
        server_name     srvkafka;

        auth_basic "Restricted Area";
        auth_basic_user_file /usr/local/kafka/.htpasswd;

        location / {
          proxy_pass http://127.0.0.1:9000;
        }

        location /logout {
         return 401;
        }

}
```

5. Reload NGINX

```
sudo systemctl reload nginx
```

After try to connect on the ip / host with credentials.

To log out just insert on the web address the path /logout

In the end, it's possible also enable HTTPS secure connection.

Here are a simple procedure to enable it (always with self-signed certificates)

1. Generate certificates with key

```
sudo su
mkdir /etc/nginx/certificate
cd /etc/nginx/certificate
openssl req -new -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out nginx-certificate.crt -keyout nginx.key
```

2. Edit default configuration file

```
sudo nano /etc/nginx/conf.d/default.conf
```

3. Insert custom virtual host configuration as You want, for example

```
[Configuration 1 with web path]
server {
        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;
        ssl_certificate /etc/nginx/certificate/nginx-certificate.crt;
        ssl_certificate_key /etc/nginx/certificate/nginx.key;
        ssl_protocols      TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers        HIGH:!aNULL:!MD5;

        location / {
          root   /var/www/html;
          index  index.html index.htm;
          autoindex on;
        }
}


[Configuration 2 with reverse proxy]
server {
        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;
        ssl_certificate /etc/nginx/certificate/nginx-certificate.crt;
        ssl_certificate_key /etc/nginx/certificate/nginx.key;
        ssl_protocols      TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers        HIGH:!aNULL:!MD5;

        auth_basic "Restricted Area";
        auth_basic_user_file /usr/local/kafka/.htpasswd;

        location / {
           proxy_pass http://127.0.0.1:9000;
         }

         location /logout {
          return 401;
        }
}


[Force redirect from http to https]
return 301 https://$host$request_uri;
```

NOTE: Kafdrop UI it's still available on default port 9000 without authentication or encryption.

A simple way to disable default port and unauthorized connections for example is enable Firewall.

Please note that Firewall Configuration changes from System to System, for example on Windows Server you should use Windows Firewall, otherwise your own Firewall System.

Here you can find some easy commands to enable Firewall for example on Ubuntu Linux.

Guide: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-18-04

Show Firewall status

```
sudo ufw status
```

Enable / Disable Firewall

```
sudo ufw enable
sudo ufw disable
```

PAY ATTENTION: if You are connected with SSH protocol and firewall doesn't allow secure shell connection, You could lost the session and for reconnect you must first allow the SSH on Firewall

Show app list profiles

```
sudo ufw app list
```

Enable app

```
sudo ufw allow OpenSSH
```

Enable port

```
sudo ufw allow 22
```

Clients for Developers

There are lot of many Clients library available for Developers, for example for C/C++, Python, GO, .NET, Node.js, PHP, Swift, etc…

Here you can find a full list of Clients library available:
https://cwiki.apache.org/confluence/display/KAFKA/Clients

In this document, I will use some example written in .NET and Python.

I will use Confluent.Kafka library, all examples shown are available on Github: https://bit.ly/3OJm4ve

.NET Confluent.Kafka

Here you can find some examples for Producer / Consumer connection with Confluent.Kafka for .NET
These examples are made with latest version available on .NET Core (≥ 5)

Confluent.Kafka Github: https://github.com/confluentinc/confluent-kafka-dotnet

Overview: https://docs.confluent.io/kafka-clients/dotnet/current/overview.html

Full Documentation: https://bit.ly/3PNWNBv

Installation

Package Manager Console

```
PM> Install-Package Confluent.Kafka
```

CMD

```
dotnet add package Confluent.Kafka
```

Producer Client Example

```
using Confluent.Kafka;
using System.Net;

string topic = "Fabrizio";
string keyValue = "Saluto";
string mesValue = "Ciao";

var config = new ProducerConfig
{
   BootstrapServers = $"192.168.1.61:9092",
   ClientId = Dns.GetHostName(),
};

Console.WriteLine($"Send message: [{keyValue} : {mesValue}]");

using (var producer = new ProducerBuilder<string, string>(config).Build())
{
   var t = producer.ProduceAsync(topic, new Message<string, string> {
      Key = keyValue,
      Value = mesValue
   });

   producer.Flush();

   t.ContinueWith(task =>
   {
      if (task.IsFaulted)
      {
         Console.WriteLine("ERRORE: message not send");
      }
      else
      {
         Console.WriteLine($"Inserted to offset: {task.Result.Offset}");
      }
   });
}
```

Output

```
Send message: [Saluto : Ciao]
Inserted to offset: 14
```

Consumer Client Example

```
using Confluent.Kafka;

var config = new ConsumerConfig
{
   BootstrapServers = "192.168.1.61:9092",
   GroupId = "fabry",
   AutoOffsetReset = AutoOffsetReset.Earliest,
};

using (var c = new ConsumerBuilder<string, string>(config).Build())
{

   List<string> myTopics = new List<string>() { "Fabrizio", "test" };

   // Automatic subscription for all topic specificated
   // All message will be consumed automatically from all partition if the GroupId not consumed the messages yet
   c.Subscribe(myTopics);

   // Automatic subscription for single topic
   //c.Subscribe("Fabrizio");

   // Automatic subscription for single topic to one partition
   //c.Assign(new TopicPartition("Fabrizio",0));

   // Automatic subscription for single topic to one partition, start consume from specificated offset (included)
   //c.Assign(new TopicPartitionOffset("Fabrizio",0,5));

   CancellationTokenSource cts = new CancellationTokenSource();

   Console.CancelKeyPress += (_, e) =>
   {
      e.Cancel = true; // Prevent process close
      cts.Cancel();
   };

   Console.Title = $"Kafka Consumer [{config.GroupId}]";
   Console.Clear();

   try
   {
      while (true)
      {
         try
         {
            var cr = c.Consume(cts.Token);

            Console.WriteLine($"Message consumed [{cr.Partition}]/[@{cr.Offset}][{cr.Message.Key} : {cr.Message.Value}] -
'{cr.TopicPartitionOffset}' - {cr.Message.Timestamp.UtcDateTime.ToLocalTime()}");

         }
         catch (ConsumeException e)
         {
            Console.WriteLine($"ERROR: {e.Error.Reason}");
         }

      }
   }
   catch (OperationCanceledException)
   {
      // Ensure that Consumer leaves the group safely and that the final offsets have been consumed.
      c.Close();
   }
}
```

Output

```
Message consumed [[0]]/[@15][Saluto : Ciao] - ' Fabrizio [[0]] @15' - 24/06/2022 16:15:48
Message consumed [[0]]/[@16][Saluto : Ciaooooooo] - 'Fabrizio [[0]] @16' - 24/06/2022 16:15:54
Message consumed [[0]]/[@4][Oggetto : Prova] - 'test [[0]] @4' - 24/06/2022 16:16:32
```

Custom elaboration examples

Read latest messages

```
using (var c = new ConsumerBuilder<string, string>(config)
    .SetPartitionsAssignedHandler((c, ps) =>
    {
        return ps.Select(tp => new TopicPartitionOffset(tp, c.QueryWatermarkOffsets(tp, TimeSpan.FromSeconds(10)).High - 1));
        //restart from latest message (included) from all partition on all specificated topic
        // only with subscribe
    })
    .Build())
{
    c.Subscribe("Fabrizio");

    ...

}
```

Read total Partitions number of a Topic

```
using (var adminClient = new AdminClientBuilder(new AdminClientConfig {

    BootstrapServers = "192.168.1.61:9092",
    ClientId = Dns.GetHostName()

}).Build())
{
    var meta = adminClient.GetMetadata(TimeSpan.FromSeconds(20));

    //var topic = meta.Topics.SingleOrDefault(t => t.Topic == "__consumer_offsets");
    var topic = meta.Topics.SingleOrDefault(t => t.Topic == "Fabrizio");

    var topicPartitions = topic.Partitions;

    for(int i=0;i<topicPartitions.Count;i++)
        Console.WriteLine(topicPartitions[i].PartitionId);
}
```

SSL Connection

Here are the procedure for secure connection to Kafka with SSL, based on this documentation
https://github.com/LGouellec/kafka-dotnet-ssl

NOTE: Confluent.Kafka .NET library use OpenSSL for SSL Connection with certificate crt and key files.
For this reason, it's necessary to generate new certificate with CA for the clients (producer or consumer).
In this example I will use also OpenSSL Win32

1. Generate a new private key and certificate request

```
openssl req -newkey rsa:2048 -nodes -keyout consumer_client.key -out consumer_client.csr
```

2. Generate certificate client signed by CA

```
openssl x509 -req -CA root.crt -CAkey root.key -in consumer_client.csr -out consumer_client.crt -days 365 -CAcreateserial
```

3. Set the properties to secure connection for
   ProducerConfig / ConsumerConfig / AdminClientBuilder:

```
var config = new ConsumerConfig
{
        BootstrapServers = "192.168.1.61:9093",
        GroupId = "fabry",
        AutoOffsetReset = AutoOffsetReset.Earliest,

        SecurityProtocol = SecurityProtocol.Ssl,
        SslEndpointIdentificationAlgorithm = SslEndpointIdentificationAlgorithm.None,
        SslCaLocation = @"C:\Users\Fabrizio\Desktop\root.crt",
        SslCertificateLocation = @"C:\Users\Fabrizio\Desktop\producer_client.crt",
        SslKeyLocation = @"C:\Users\Fabrizio\Desktop\producer_client.key"

};
```

SASL SSL Connection

Here are the procedure for secure connection to Kafka with SASL SSL, based on this documentation https://github.com/LGouellec/kafka-dotnet-ssl

The procedure for generating client certificate is the same as already explained previously.

Set the properties to secure connection for ProducerConfig / ConsumerConfig / AdminClientBuilder

```
var config = new ConsumerConfig
{
   BootstrapServers = "192.168.1.61:9094",
   GroupId = "fabrynew",
   AutoOffsetReset = AutoOffsetReset.Earliest,

   SaslMechanism = SaslMechanism.Plain,
   SecurityProtocol = SecurityProtocol.SaslSsl,
   SaslUsername = "admin",
   SaslPassword = "admin-secret",

   SslEndpointIdentificationAlgorithm = SslEndpointIdentificationAlgorithm.None,
   SslCaLocation = @"C:\Users\Fabrizio\Desktop\root.crt",
   SslCertificateLocation = @"C:\Users\Fabrizio\Desktop\consumer_client.crt",
   SslKeyLocation = @"C:\Users\Fabrizio\Desktop\consumer_client.key"

};
```

Pyhton Confluent.Kafka

Confluent.Kafka Github: https://github.com/confluentinc/confluent-kafka-python

Install the packet

```
pip install confluent-kafka
```

Producer

```
from confluent_kafka import Producer

p = Producer({'bootstrap.servers': '192.168.1.61:9092'})
kdata = "Saluto"
data = "Ciao da Python"

def delivery_report(err, msg):
    """ Called once for each message produced to indicate delivery result.
        Triggered by poll() or flush(). """
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}] {}'.format(msg.topic(), msg.partition(), msg.offset()))

# Trigger any available delivery report callbacks from previous produce() calls
p.poll(0)

# Asynchronously produce a message, the delivery report callback
# will be triggered from poll() above, or flush() below, when the message has
# been successfully delivered or failed permanently.
p.produce(topic='Fabrizio', key=kdata.encode('utf-8'), value=data.encode('utf-8'), callback=delivery_report)

# Wait for any outstanding messages to be delivered and delivery report
# callbacks to be triggered.
p.flush()
```

High-level Consumer

```
from types import NoneType
from confluent_kafka import Consumer


c = Consumer({
    'bootstrap.servers': '192.168.1.61:9092',
    'group.id': 'Fabry',
    'auto.offset.reset': 'earliest'
})

c.subscribe(['Fabrizio'])

while True:
    msg = c.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    if isinstance(msg.key(), NoneType):
        key="NO_KEY"
    else:
        key=msg.key().decode('utf-8')

    #key=str(msg.key())
    value=msg.value().decode('utf-8')

    print(f"Received message {key}:{value} [{msg.partition()}][{msg.offset()}]")

c.close()
```

Custom elaboration examples

```
from types import NoneType
import confluent_kafka
from confluent_kafka import Consumer


c = Consumer({
   'bootstrap.servers': '192.168.1.61:9092',
   'group.id': 'Fabry',
   'auto.offset.reset': 'earliest'
})

def my_assign (consumer, partitions):
      for p in partitions:
         p.offset = confluent_kafka.OFFSET_END
      print('assign', partitions)
      consumer.assign(partitions)


# Read only new messages, ignore all exists on topic
c.subscribe(['Fabrizio'], on_assign=my_assign)

# Automatic subscription for single topic to one partition
#c.assign([confluent_kafka.TopicPartition("Fabrizio", 0)])

# Automatic subscription for single topic to one partition, start consume from specificated offset (included)
#c.assign([confluent_kafka.TopicPartition("Fabrizio", 0, 10)])

while True:
   msg = c.poll(1.0)

   if msg is None:
      continue
   if msg.error():
      print("Consumer error: {}".format(msg.error()))
      continue

   if isinstance(msg.key(), NoneType):
      key="NO_KEY"
   else:
      key=msg.key().decode('utf-8')

   #key=str(msg.key())
   value=msg.value().decode('utf-8')

   print(f"Received message {key}:{value} [{msg.partition()}][{msg.offset()}]")

c.close()
```

Read total Partitions number of a Topic

```
from confluent_kafka.admin import AdminClient

c = AdminClient({
    'bootstrap.servers': '192.168.1.61:9092',
})

info = c.list_topics("Fabrizio").topics

print(f"{info['Fabrizio']}")

# To get exactly number, I need to change the __str__ method on TopicMedadata Class
def __str__(self):
        return str(len(self.partitions))
```

SSL Connection

NOTE: You must use certificates PEM format for secure connection.

You can convert crt and key with these following commands

```
openssl x509 -in mycert.crt -out mycert.pem -outform PEM
openssl rsa -in mykey.key -out mykey.pem -outform PEM
```

After, set these attribute on the dictionary configuration

```
'security.protocol' : 'SSL',
'ssl.endpoint.identification.algorithm' : 'none',
'ssl.ca.location' : 'C:\\Users\\Fabrizio\\Desktop\\root.pem',
'ssl.certificate.location' : 'C:\\Users\\Fabrizio\\Desktop\\consumer_client.pem',
'ssl.key.location' : 'C:\\Users\\Fabrizio\\Desktop\\key.pem'
```

SASL SSL Connection

```
'security.protocol' : 'SASL_SSL',
'sasl.mechanisms' : 'PLAIN',
'sasl.username' : 'admin',
'sasl.password' : 'admin-secret',

'ssl.endpoint.identification.algorithm' : 'none',
'ssl.ca.location' : 'C:\\Users\\Fabrizio\\Desktop\\root.pem',
'ssl.certificate.location' : 'C:\\Users\\Fabrizio\\Desktop\\consumer_client.pem',
'ssl.key.location' : 'C:\\Users\\Fabrizio\\Desktop\\key.pem'
```