lff l
9 Followers

May 11, 2020 · 4 min read

Build a kafka cluster with SSL with docker / docker-compose and Java Client

Requirements:

1. A kafka cluster with 3 brokers
2. The communication between 3 brokers should be encrypted (SSL)
3. The communication between brokers and Java Client should be encrypted.

After some test, I finally did that. The following docs online give me great help:

1. https://www.cnblogs.com/huxi2b/p/7427815.html

NOTE: This should be used only for evaluation. Do not use this in PROD. SSL communication between kafka instances are poor in performance.

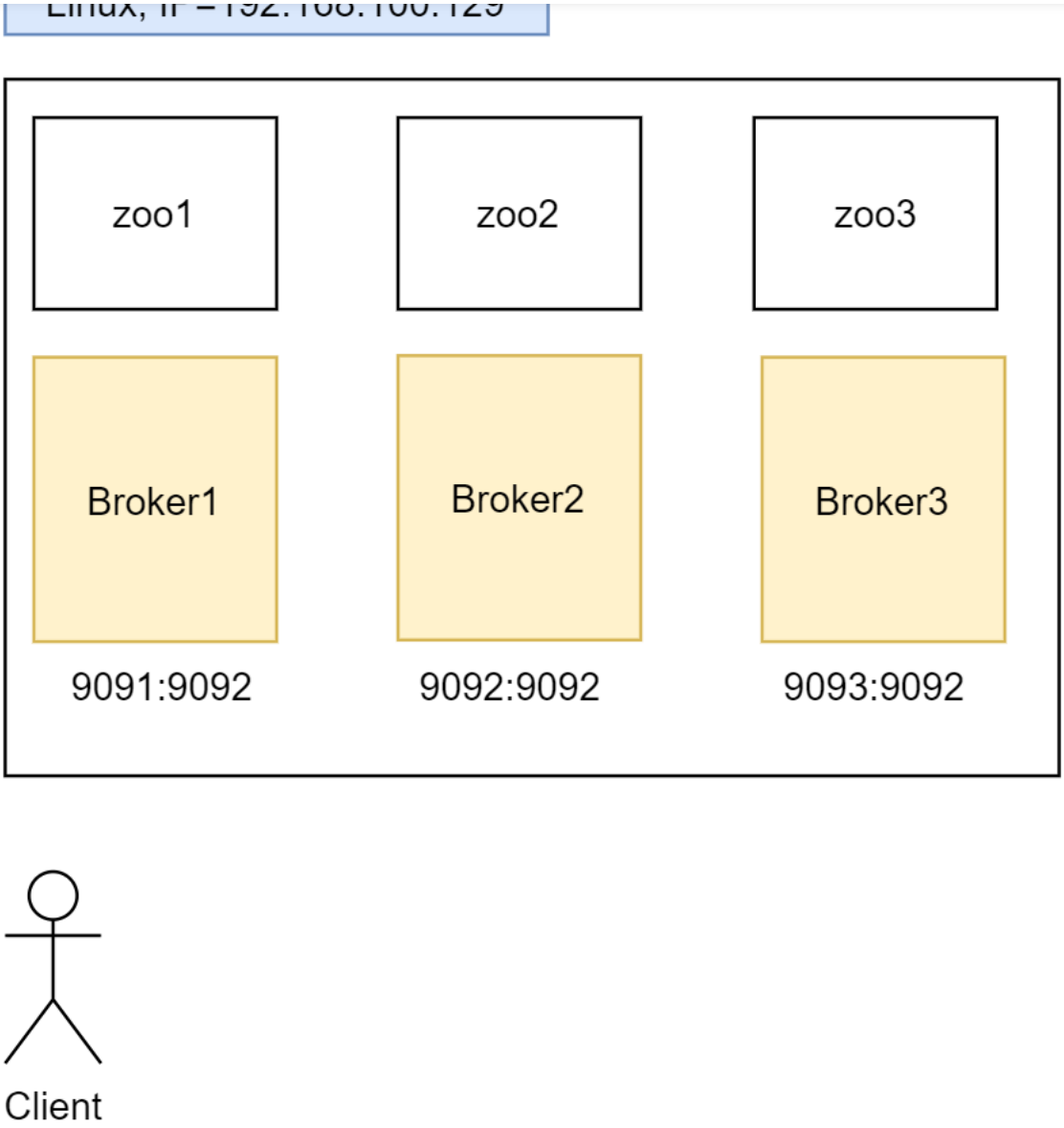
Software needed:

1. openssl (From CentOS)
2. keytool (From jdk)
3. docker/docker-compose

System Requirement. On a linux machine (192.168.100.129) the following containers are running (Zookeepers, kafka brokers). The 9091 port on linux is mapped to the port 9092 of container kafka broker 1. The 9092 port on linux is mapped to the port 9092 of container kafka broker 2. The 9093 port on linux is mapped to the port 9092 of container kafka broker 3.

So client will connect to this cluster as

192.168.100.129:9091,192.168.100.129:9092,192.168.100.129:9093



Step.1 Prepare the kafka container.

Because the default container does not support setting these parameters in ENV, we must build our own docker container for later usage.

Dockerfile:

```
FROM  wurstmeister/kafka
RUN echo "ssl.keystore.location=/certificates/kafka.keystore" >> /opt/kafka/config/server.properties && \
    echo "ssl.keystore.password=Aa123456!" >> /opt/kafka/config/server.properties && \
    echo "ssl.truststore.location=/certificates/kafka.truststore"  >> /opt/kafka/config/server.properties && \
    echo "ssl.truststore.password=Aa123456!"  >> /opt/kafka/config/server.properties && \
    echo "ssl.key.password=Aa123456!"  >> /opt/kafka/config/server.properties && \
    echo "ssl.client.auth=required"  >> /opt/kafka/config/server.properties
```

Dockerfile hosted with ❤ by **GitHub** **view raw**

Step.2 Create a docker-compose.yml file and add zookeeper support

Public docker-hub zookeeper images can be used.

```
version: '2'
services:
    zoo1:
        image: zookeeper
        restart: always
        hostname: zoo1
        ports:
            - "2181:2181"
        environment:
            ZOO_MY_ID: 1
            ZOO_SERVERS: server.1=zoo1:2888:3888;2181 server.2=zoo2:2888:3888;2181 server.3=zoo3:2888:3888;2181
    zoo2:
        image: zookeeper
        restart: always
        hostname: zoo2
        ports:
            - "2182:2181"
        environment:
            ZOO_MY_ID: 2
            ZOO_SERVERS: server.1=zoo1:2888:3888;2181 server.2=zoo2:2888:3888;2181 server.3=zoo3:2888:3888;2181

    zoo3:
        image: zookeeper
        restart: always
        hostname: zoo3
        ports:
            - "2183:2181"
        environment:
            ZOO_MY_ID: 3
            ZOO_SERVERS: server.1=zoo1:2888:3888;2181 server.2=zoo2:2888:3888;2181 server.3=zoo3:2888:3888;2181
```

docker-compose.yml hosted with ❤ by **GitHub** **view raw**

Step.3 Create the keystore, truststore, CA, Certificate Sign Request file,and sign the certificates of the 3 brokers with CA.

First, we need to modify `openssl` configuration to enable SAN names and add our broker name and host ip. The updated openssl configuration file (openssl.cf)

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

```

```
# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir             = /etc/pki/CA           # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options

# Extension copying option: use with caution.
copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = sha256                # use SHA-256 by default
preserve        = no                    # keep passed DN ordering
```

```
82
83  # For the CA policy
84  [ policy_match ]
85  countryName             = match
86  stateOrProvinceName     = match
87  organizationName        = match
88  organizationalUnitName  = optional
89  commonName              = supplied
90  emailAddress            = optional
91
92  # For the 'anything' policy
93  # At this point in time, you must list all acceptable 'object'
94  # types.
95  [ policy_anything ]
96  countryName             = optional
97  stateOrProvinceName     = optional
98  localityName            = optional
99  organizationName        = optional
100 organizationalUnitName  = optional
101 commonName              = supplied
102 emailAddress            = optional
103
104 ####################################################################
105 [ req ]
106 default_bits            = 2048
107 default_md              = sha256
108 default_keyfile         = privkey.pem
109 distinguished_name      = req_distinguished_name
110 attributes              = req_attributes
111 x509_extensions = v3_ca # The extentions to add to the self signed cert
112
113 # Passwords for private keys if not present they will be prompted for
114 # input_password = secret
115 # output_password = secret
116
117 # This sets a mask for permitted string types. There are several options.
118 # default: PrintableString, T61String, BMPString.
119 # pkix   : PrintableString, BMPString (PKIX recommendation before 2004)
120 # utf8only: only UTF8Strings (PKIX recommendation after 2004).
121 # nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
122 # MASK:XXXX a literal mask value.
123 # WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
124 string_mask = utf8only
125
126 req_extensions = v3_req # The extensions to add to a certificate request
127
128 [ req_distinguished_name ]
129 countryName                     = Country Name (2 letter code)
130 countryName_default             = XX
131 countryName_min                 = 2
132 countryName_max                 = 2
133
134 stateOrProvinceName             = State or Province Name (full name)
135 #stateOrProvinceName_default    = Default Province
136
137 localityName                    = Locality Name (eg, city)
138 localityName_default            = Default City
139
140 0.organizationName              = Organization Name (eg, company)
141 0.organizationName_default      = Default Company Ltd
142
143 # we can do this but it is not needed normally :-)
144 #1.organizationName             = Second Organization Name (eg, company)
145 #1.organizationName_default     = World Wide Web Pty Ltd
146
147 organizationalUnitName          = Organizational Unit Name (eg, section)
```

```
emailAddress                    = Email Address
emailAddress_max                = 64

# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName                = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]
```

```
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email
```

```
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]

default_tsa = tsa_config1       # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir             = ./demoCA              # TSA root directory
serial          = $dir/tsaserial        # The current serial number (mandatory)
crypto_device   = builtin               # OpenSSL engine to use for signing
signer_cert     = $dir/tsacert.pem      # The TSA signing certificate
                                        # (optional)
certs           = $dir/cacert.pem       # Certificate chain to include in reply
                                        # (optional)
signer_key      = $dir/private/tsakey.pem # The TSA private key (optional)

default_policy  = tsa_policy1           # Policy if request did not specify it
                                        # (optional)
other_policies  = tsa_policy2, tsa_policy3      # acceptable policies (optional)
digests         = sha1, sha256, sha384, sha512  # Acceptable message digests (mandatory)
accuracy        = secs:1, millisecs:500, microsecs:100  # (optional)
clock_precision_digits  = 0     # number of digits after dot. (optional)
ordering                = yes   # Is ordering defined for timestamps?
                                # (optional, default: no)
tsa_name                = yes   # Must the TSA name be included in the reply?
                                # (optional, default: no)
ess_cert_id_chain       = no    # Must the ESS cert id chain be included?
                                # (optional, default: no)
[ alt_names ]
DNS.1 = broker1
DNS.2 = broker2
DNS.3 = broker3
IP.1 = 192.168.100.129
```

openssl.cnf hosted with ❤ by **GitHub** **view raw**

At the end part of the file, 3 broker names (broker1, broker2, broker3) and the public ip addresss (192.168.100.129) is set in [alt_names] section. Make sure to update the ip address if yours is not the same.

Second, run the following bash script to generate the keystore and truststore.

1. This script file need to be placed under the same directory of the `openssl.cnf` in above step;

2. `keytool` and `openssl` must be on `PATH`

Remember to change the `CLUSTER_IP` and `PASSWORD` if your IP is different.

```
#!/bin/bash

################################## Set Variables #############################
CLUSTER_NAME_1=broker1  # Alias name (3 private name of the 3 brokers)
CLUSTER_NAME_2=broker2
CLUSTER_NAME_3=broker3
CLUSTER_IP=192.168.100.129 # Public IP, Change if needed
BASE_DIR=/tmp/kafka # Path
CERT_OUTPUT_PATH="$BASE_DIR/certificates" # certificates path
PASSWORD=Aa123456! # password
KEY_STORE_1="$CERT_OUTPUT_PATH/kafka.keystore"   # Kafka keystore path
TRUST_STORE="$CERT_OUTPUT_PATH/kafka.truststore" # Kafka truststore path
KEY_PASSWORD=$PASSWORD # keystore password
STORE_PASSWORD=$PASSWORD # keystore password
TRUST_KEY_PASSWORD=$PASSWORD  # truststore key password
TRUST_STORE_PASSWORD=$PASSWORD # truststore store password
CERT_AUTH_FILE="$CERT_OUTPUT_PATH/ca-cert" # CA path
CLUSTER_CERT_FILE="$CERT_OUTPUT_PATH/kafka-cert" # Cluster certificate path
DAYS_VALID=730 # Key vaild days
DNAME_1="CN=broker1, OU=Company, O=Company, L=Singapore, ST=Singapore, C=SG"
DNAME_2="CN=broker2, OU=Company, O=Company, L=Singapore, ST=Singapore, C=SG"
DNAME_3="CN=broker3, OU=Company, O=Company, L=Singapore, ST=Singapore, C=SG" # distinguished name
##############################################################################

mkdir -p $CERT_OUTPUT_PATH

```

```
keytool -keystore $KEY_STORE_1 -alias $CLUSTER_NAME_3 -validity $DAYS_VALID -genkey -keyalg RSA \
-storepass $STORE_PASSWORD -keypass $KEY_PASSWORD -dname "$DNAME_3"

echo "2. Creating CA......"
openssl req -new -x509 -keyout $CERT_OUTPUT_PATH/ca-key -out "$CERT_AUTH_FILE" -days "$DAYS_VALID" \
-passin pass:"$PASSWORD" -passout pass:"$PASSWORD" \
-subj "/C=SG/ST=Singapore/L=Singapore/O=Company/CN=Company"
echo "3. Import CA to truststore......"
keytool -keystore "$TRUST_STORE" -alias CARoot \
-import -file "$CERT_AUTH_FILE" -storepass "$TRUST_STORE_PASSWORD" -keypass "$TRUST_KEY_PASS" -noprompt

echo "4. Export Certificate Request file from keystore......"
keytool -keystore "$KEY_STORE_1" -alias "$CLUSTER_NAME_1" -certreq -file "$CLUSTER_CERT_FILE"_1 -storepass "$STORE_PASSWORD" -keypass "$

keytool -keystore "$KEY_STORE_1" -alias "$CLUSTER_NAME_2" -certreq -file "$CLUSTER_CERT_FILE"_2 -storepass "$STORE_PASSWORD" -keypass "$

keytool -keystore "$KEY_STORE_1" -alias "$CLUSTER_NAME_3" -certreq -file "$CLUSTER_CERT_FILE"_3 -storepass "$STORE_PASSWORD" -keypass "$

echo "5. Sign Certifiate by CA......"
openssl x509 -req -CA "$CERT_AUTH_FILE" -CAkey $CERT_OUTPUT_PATH/ca-key -in "$CLUSTER_CERT_FILE"_1 \
-out "${CLUSTER_CERT_FILE}_1-signed" \
-CAcreateserial \
-extfile ./openssl.cnf -extensions v3_req \
-days "$DAYS_VALID" -CAcreateserial -passin pass:"$PASSWORD"

openssl x509 -req -CA "$CERT_AUTH_FILE" -CAkey $CERT_OUTPUT_PATH/ca-key -in "$CLUSTER_CERT_FILE"_2 \
-out "${CLUSTER_CERT_FILE}_2-signed" \
-CAcreateserial \
-extfile ./openssl.cnf -extensions v3_req \
-days "$DAYS_VALID" -CAcreateserial -passin pass:"$PASSWORD"

openssl x509 -req -CA "$CERT_AUTH_FILE" -CAkey $CERT_OUTPUT_PATH/ca-key -in "$CLUSTER_CERT_FILE"_3 \
-out "${CLUSTER_CERT_FILE}_3-signed" \
-CAcreateserial \
-extfile ./openssl.cnf -extensions v3_req \
-days "$DAYS_VALID" -CAcreateserial -passin pass:"$PASSWORD"

echo "6. Import CA to keystore......"
keytool -keystore "$KEY_STORE_1" -alias CARoot_1 -import -file "$CERT_AUTH_FILE" -storepass "$STORE_PASSWORD" \
 -keypass "$KEY_PASSWORD" -noprompt

echo "7. Import Signed Certificates to keystore......"
keytool -keystore "$KEY_STORE_1" -alias "${CLUSTER_NAME_1}" -import -file "${CLUSTER_CERT_FILE}_1-signed" \
 -storepass "$STORE_PASSWORD" -keypass "$KEY_PASSWORD" -noprompt

keytool -keystore "$KEY_STORE_1" -alias "${CLUSTER_NAME_2}" -import -file "${CLUSTER_CERT_FILE}_2-signed" \
 -storepass "$STORE_PASSWORD" -keypass "$KEY_PASSWORD" -noprompt

keytool -keystore "$KEY_STORE_1" -alias "${CLUSTER_NAME_3}" -import -file "${CLUSTER_CERT_FILE}_3-signed" \
 -storepass "$STORE_PASSWORD" -keypass "$KEY_PASSWORD" -noprompt
```

setup.sh hosted with ❤ by **GitHub** **view raw**

Run the script, and if everything is fine, you will get the files under `BASE_DIR/certificates` . For default it is `/tmp/kafka/certificates` :

```
-rw-r--r-- 1 root root 1.3K May  7 10:30 ca-cert
-rw-r--r-- 1 root root   41 May  7 10:30 ca-cert.srl
-rw------- 1 root root 1.9K May  7 10:30 ca-key
-rw-r--r-- 1 root root 1.2K May  7 10:30 kafka-cert_1
-rw-r--r-- 1 root root 1.3K May  7 10:30 kafka-cert_1-signed
-rw-r--r-- 1 root root 1.2K May  7 10:30 kafka-cert_2
-rw-r--r-- 1 root root 1.3K May  7 10:30 kafka-cert_2-signed
-rw-r--r-- 1 root root 1.2K May  7 10:30 kafka-cert_3
-rw-r--r-- 1 root root 1.3K May  7 10:30 kafka-cert_3-signed
-rw-r--r-- 1 root root  11K May  7 10:30 kafka.keystore
-rw-r--r-- 1 root root  982 May  7 10:30 kafka.truststore
```

Only `kafka.keystore` and `kafka.truststore` is useful. Other files should be kept as a secret.

Step.4, Add `kafka` service configurations to the docker-compose.yml in Step.2

```
    broker1:
        image: my_kafka:latest
        restart: always
        hostname: broker1
#       command: ["sleep", "6000s"]
        ports:
          - "9091:9092"
        depends_on:
          - zoo1
          - zoo2
          - zoo3
        environment:
          KAFKA_BROKER_ID: 1
          KAFKA_ADVERTISED_HOST_NAME: 192.168.100.129
          KAFKA_ADVERTISED_PORT: 9091
          KAFKA_HOST_NAME: broker1
          KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
          KAFKA_LISTENERS: SSL://broker1:9092
          KAFKA_ADVERTISED_LISTENERS: SSL://192.168.100.129:9091
          KAFKA_HEAP_OPTS: "-Xmx256M -Xms128M"
          KAFKA_INTER_BROKER_LISTENER_NAME: SSL
        volumes:
          - /tmp/kafka/certificates/kafka.truststore:/certificates/kafka.truststore
```

```
        hostname: broker2
        ports:
          - "9092:9092"
        depends_on:
          - zoo1
          - zoo2
          - zoo3
        environment:
          KAFKA_BROKER_ID: 2
          KAFKA_ADVERTISED_HOST_NAME: 192.168.100.129
          KAFKA_ADVERTISED_PORT: 9092
          KAFKA_HOST_NAME: broker2
          KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
          KAFKA_LISTENERS: SSL://broker2:9092
          KAFKA_ADVERTISED_LISTENERS: SSL://192.168.100.129:9092
          KAFKA_HEAP_OPTS: "-Xmx256M -Xms128M"
          KAFKA_INTER_BROKER_LISTENER_NAME: SSL
        volumes:
          - /tmp/kafka/certificates/kafka.truststore:/certificates/kafka.truststore
          - /tmp/kafka/certificates/kafka.keystore:/certificates/kafka.keystore
    broker3:
        image: my_kafka:latest
        restart: always
       # command: ["sleep", "6000s"]
        hostname: broker3
        ports:
          - "9093:9092"
        depends_on:
          - zoo1
          - zoo2
          - zoo3
        environment:
          KAFKA_BROKER_ID: 3
          KAFKA_ADVERTISED_HOST_NAME: 192.168.100.129
          KAFKA_ADVERTISED_PORT: 9093
          KAFKA_HOST_NAME: broker3
          KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181
          KAFKA_LISTENERS: SSL://broker3:9092
          KAFKA_ADVERTISED_LISTENERS: SSL://192.168.100.129:9093
          KAFKA_HEAP_OPTS: "-Xmx256M -Xms128M"
          KAFKA_INTER_BROKER_LISTENER_NAME: SSL
        logging:
          options:
            max-file: "5"
            max-size: "10m"
        volumes:
          - /tmp/kafka/certificates/kafka.truststore:/certificates/kafka.truststore
          - /tmp/kafka/certificates/kafka.keystore:/certificates/kafka.keystore
    kafka-manager:
        image: sheepkiller/kafka-manager
        environment:
          ZK_HOSTS: zoo1:2181,zoo2:2181,zoo3:2181
        logging:
          options:
            max-file: "5"
            max-size: "10m"
        ports:
           - "9000:9000"
```

kafka.yml hosted with ❤ by **GitHub** **view raw**

For the configuration

```
KAFKA_ADVERTISED_HOST_NAME: 192.168.100.129      KAFKA_ADVERTISED_PORT: 9091
KAFKA_HOST_NAME: broker1
KAFKA_ZOOKEEPER_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181          KAFKA_LISTENERS: SSL://broker1:9092
KAFKA_ADVERTISED_LISTENERS: SSL://192.168.100.129:9091          KAFKA_HEAP_OPTS: "-Xmx256M -Xms128M"
KAFKA_INTER_BROKER_LISTENER_NAME: SSL
```

Change all the ip address `192.168.100.129` to your public ip, if needed. No need to change the `broker1` because its an internal hostname and can be resolved by docker-compose.

And for each `volume` section, the truststore / keystore file is mounted to `/certificates` folder of the kafka container. This is because in Step1. we put the keystore/truststore settings in `server.properties` (server.keystore.location, server,truststore.location). You also need to change the location if the `BASE_DIR/certificates` in the script was updated.

```
volumes:          - /tmp/kafka/certificates/kafka.truststore:/certificates/kafka.truststore          -
/tmp/kafka/certificates/kafka.keystore:/certificates/kafka.keystore
```

Step.5, start the services

run `docker-compose up -d` under the same directory with the `docker-compose.yml` to start zookeepers, and kafka clusters. If everything is fine, we will get a kafka cluster with SSL enabled.

Step.6, test with Client (Java).

Before running, need to copy the keystore/truststore to the machine on which Java Class will be running.

Producer Class:

```
    private static final String CONTENT = "TestMessage";

    public static void main(String[] argu) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "192.168.100.129:9091,192.168.100.129:9092,192.168.100.129:9093");
        //props.put("acks", "all")
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("security.protocol", "SSL");
        props.put("ssl.truststore.location", "C:\\dev\\test\\kafka\\src\\main\\resources\\kafka.truststore");
        props.put("ssl.truststore.password", "Aa123456!");

        props.put("ssl.keystore.location", "C:\\dev\\test\\kafka\\src\\main\\resources\\kafka.keystore");
        props.put("ssl.keystore.password", "Aa123456!");
        props.put("ssl.key.password", "Aa123456!");
        props.put("acks", "1");
        props.put("retries", 0);
        // Controls how much bytes sender would wait to batch up before publishing to Kafka.
        props.put("batch.size", 10);
        props.put("linger.ms", 10);
        final Producer<String, String> producer = new KafkaProducer<String, String>(props);
        System.out.println("Producer created");
        for (int i = 0; i < 1000; i++) {
            Integer times = 0;
            try {
                RecordMetadata o = producer
                    .send(new ProducerRecord<String, String>(TOPIC_NAME, Integer.toString(i), CONTENT + i)).get();
                System.out.println(o.topic() + " " + o.partition() + " " + o.offset() + " " + o.timestamp());
            } catch (Exception e) {
                e.printStackTrace();
            }
            System.out.println("Send one message " + i);
        }
    }
}
```

Producer.java hosted with ❤ by **GitHub** **view raw**

Please pay attention to the password settings, keystore/truststore setting. If needed update accordingly.

Consumer Class (Also change the password settings, keystore/truststore setting if needed)

```
public class ConsumerTest {
    public static void main(String[] argu) {
        KafkaConsumer<String, String> consumer = createConsumer("192.168.100.129:9091,192.168.100.129:9092,192.168.100.129:9093");
```

```
                System.out.printf("Received Message topic =%s, partition =%s, offset = %d, key = %s, value = %s\n",
                    record.topic(), record.partition(), record.offset(), record.key(), record.value());
            }
        }

    }

    private static KafkaConsumer<String, String> createConsumer(String brokers) {
        Properties props = new Properties();
        String consumeGroup = UUID.randomUUID().toString();
        props.put("group.id", consumeGroup);
        props.put("auto.offset.reset", "latest");
        // Set this property, if auto commit should happen.
        props.put("enable.auto.commit", "true");
        // Auto commit interval, kafka would commit offset at this interval.
        props.put("auto.commit.interval.ms", "10000");
        // This is how to control number of records being read in each poll
        props.put("max.partition.fetch.bytes", "10240");
        // Set this if you want to always read from beginning.
        // props.put("auto.offset.reset", "earliest");
        props.put("heartbeat.interval.ms", "3000");
        props.put("session.timeout.ms", "6001");
        props.put("key.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

        props.put("bootstrap.servers", brokers);
        //props.put("acks", "all")
        props.put("security.protocol", "SSL");
        props.put("ssl.truststore.location", "C:\\dev\\test\\kafka\\src\\main\\resources\\kafka.truststore");
        props.put("ssl.truststore.password", "Aa123456!");

        props.put("ssl.keystore.location", "C:\\dev\\test\\kafka\\src\\main\\resources\\kafka_1.keystore");
        props.put("ssl.keystore.password", "Aa123456!");
        props.put("ssl.key.password", "Aa123456!");
        return new KafkaConsumer<String, String>(props);
    }

}
```

Consumer.java hosted with ❤ by **GitHub** **view raw**