Lab 8.1: Kafka SSL

Welcome to the session 8 lab 1. The work for this lab is done in `~/kafka-training/labs/lab8.1`. In this lab, you are going to setup Kafka SSL support.

Note: Do not run scripts inside `bin` directory. Run scripts from `~/kafka-training/labs/lab8.1/solution` directory

Note: Lab solution is available in following directory: `~/kafka-training/labs/lab8.1/solution`

Important!

Run following script first to stop any running kafka/zookeeper process and clear logs.

`~/kafka-training/kill-clean.sh`

Steps to use SSL for Consumers and Producers

Generate SSL key and certificate for each Kafka broker

ACTION - EDIT `bin/create-ssl-key-keystore.sh` and follow instructions

```
#!/usr/bin/env bash
set -e

# Common Variables for SSL Key Gen
CERT_OUTPUT_PATH="$PWD/resources/opt/kafka/conf/certs"
KEY_STORE="$CERT_OUTPUT_PATH/kafka.keystore"
TRUST_STORE="$CERT_OUTPUT_PATH/kafka.truststore"
PASSWORD=kafka123
KEY_KEY_PASS="$PASSWORD"
KEY_STORE_PASS="$PASSWORD"
TRUST_KEY_PASS="$PASSWORD"
TRUST_STORE_PASS="$PASSWORD"
CLUSTER_NAME=kafka
CERT_AUTH_FILE="$CERT_OUTPUT_PATH/ca-cert"
CLUSTER_CERT_FILE="$CERT_OUTPUT_PATH/${CLUSTER_NAME}-cert"
D_NAME="CN=CloudDurable Image $CLUSTER_NAME cluster, OU=Fenago, O=Fenago"
D_NAME="${D_NAME}, L=San Francisco, ST=CA, C=USA, DC=fenago, DC=com"
DAYS_VALID=365

mkdir -p "$CERT_OUTPUT_PATH"

echo "Create the cluster key for cluster communication."
keytool -genkey -keyalg RSA -alias "${CLUSTER_NAME}_cluster" \
-keystore "$KEY_STORE" -storepass "$KEY_STORE_PASS" \
-keypass "$KEY_KEY_PASS" -dname  "$D_NAME" -validity "$DAYS_VALID"

echo "Create the Certificate Authority (CA) file to sign keys."
openssl req -new -x509 -keyout ca-key -out "$CERT_AUTH_FILE" \
-days "$DAYS_VALID" \
-passin pass:"$PASSWORD" -passout pass:"$PASSWORD" \
-subj "/C=US/ST=CA/L=San Francisco/O=Engineering/CN=fenago.com"
```

```
echo "Import the Certificate Authority file into the trust store."
keytool -keystore "$TRUST_STORE" -alias CARoot \
-import -file "$CERT_AUTH_FILE" \
-storepass "$TRUST_STORE_PASS" -keypass "$TRUST_KEY_PASS" \
-noprompt

echo "Export the cluster certificate from the key store."
keytool -keystore "$KEY_STORE" -alias "${CLUSTER_NAME}_cluster" \
-certreq -file "$CLUSTER_CERT_FILE" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt

echo "Sign the cluster certificate with the CA."
openssl x509 -req -CA "$CERT_AUTH_FILE" -CAkey ca-key \
-in "$CLUSTER_CERT_FILE" -out "${CLUSTER_CERT_FILE}-signed" \
-days "$DAYS_VALID" -CAcreateserial -passin pass:"$PASSWORD"

echo "Import the Certificate Authority (CA) file into the key store."
keytool -keystore "$KEY_STORE" -alias CARoot -import -file "$CERT_AUTH_FILE" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt

echo "Import the Signed Cluster Certificate into the key store."
keytool -keystore "$KEY_STORE" -alias "${CLUSTER_NAME}_cluster" \
-import -file "${CLUSTER_CERT_FILE}-signed" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt
```

Generate cluster certificate into a keystore use keytool

```
echo "Create the cluster key for cluster communication."
keytool -genkey -keyalg RSA -alias "${CLUSTER_NAME}_cluster" \
-keystore "$KEY_STORE" -storepass "$KEY_STORE_PASS" \
-keypass "$KEY_KEY_PASS" -dname  "$D_NAME" -validity "$DAYS_VALID"
```

keytool ships with Java used for SSL/TLS

`-genkey (generate key)`

`-keystore (location of keystore to add the key)`

`-keyalg RSA (use the RSA algorithm for the key)`

`-alias (alias of the key we use this later to extract and sign key)`

`-storepass (password for the keystore)`

`-keypass (password for key)`

`-validity (how many days is this key valid)`

Generate or use CA (Certificate Authority) use openssl

```
echo "Create the Certificate Authority (CA) file to sign keys."
openssl req -new -x509 -keyout ca-key -out "$CERT_AUTH_FILE" \
-days "$DAYS_VALID" \
-passin pass:"$PASSWORD" -passout pass:"$PASSWORD" \
-subj "/C=US/ST=CA/L=San Francisco/O=Engineering/CN=fenago.com"
```

X.509 certificate contains a public key and an identity (is hostname, or an organization, or an individual and is either signed by a certificate authority or self-signed)

`-req -new -x509 (create contains a public key and an identity)`

`-days (how many days is this certificate valid)`

`-passin pass / -passout pass (passwords to access the certificate)`

`-subj (pass identity information about the certificate)`

Import CA into Kafka's truststore use keytool

```
echo "Import the Certificate Authority file into the trust store."
keytool -keystore "$TRUST_STORE" -alias CARoot \
-import -file "$CERT_AUTH_FILE" \
-storepass "$TRUST_STORE_PASS" -keypass "$TRUST_KEY_PASS" \
-noprompt
```

`-import -file (is CA file we generated in the last step)`

`-keystore (location of trust keystore file)`

`—storepass (password for the keystore)`

`—keypass (password for key)`

Export and Sign cluster certificate with CA use openssl

```
echo "Export the cluster certificate from the key store."
keytool -keystore "$KEY_STORE" -alias "${CLUSTER_NAME}_cluster" \
-certreq -file "$CLUSTER_CERT_FILE" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt

echo "Sign the cluster certificate with the CA."
openssl x509 -req -CA "$CERT_AUTH_FILE" -CAkey ca-key \
-in "$CLUSTER_CERT_FILE" -out "${CLUSTER_CERT_FILE}-signed" \
-days "$DAYS_VALID" -CAcreateserial -passin pass:"$PASSWORD"
```

Export the CLUSTER_CERT_FILE from the first step from the keystore, then sign the CLUSTER_CERT_FILE with the CA

Import CA and signed cluster certificate into Kafka's keystore use keytool

```
echo "Import the Certificate Authority (CA) file into the key store."
keytool -keystore "$KEY_STORE" -alias CARoot -import -file "$CERT_AUTH_FILE" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt

echo "Import the Signed Cluster Certificate into the key store."
keytool -keystore "$KEY_STORE" -alias "${CLUSTER_NAME}_cluster" \
-import -file "${CLUSTER_CERT_FILE}-signed" \
-storepass "$KEY_STORE_PASS" -keypass "$KEY_KEY_PASS" -noprompt
```

Import the CA file into keystore, it was already imported into the truststore. Import the signed version of the cluster certificate into the keystore. This was the file we create in the last step.

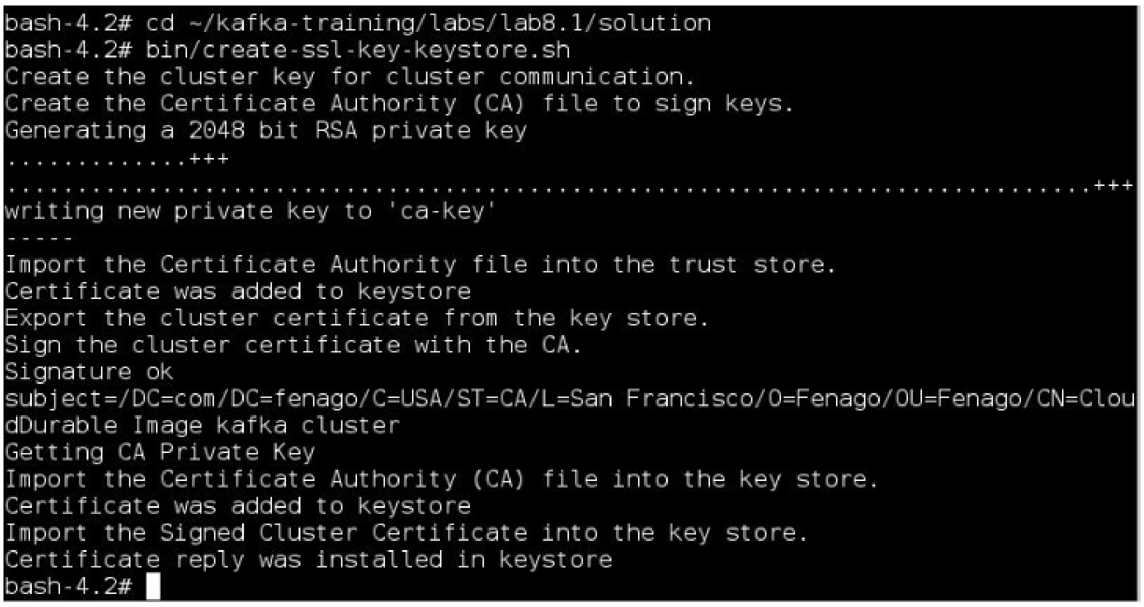
Run bin/create-ssl-key-keystore.sh and copy files to /opt/kafka

Running create-ssl-key-keystore.sh

You will want to run `create-ssl-key-keystore.sh` and then copy and/or move files so that each Broker, Producer or Consumer has access to `/opt/kafka/conf/certs/` .

Running create-ssl-key-keystore.sh

```
$ cd ~/kafka-training/labs/lab8.1/solution

$ bin/create-ssl-key-keystore.sh
```



```
bash-4.2# cd ~/kafka-training/labs/lab8.1/solution
bash-4.2# bin/create-ssl-key-keystore.sh
Create the cluster key for cluster communication.
Create the Certificate Authority (CA) file to sign keys.
Generating a 2048 bit RSA private key
.............+++
...........................................................................+++
writing new private key to 'ca-key'
-----
Import the Certificate Authority file into the trust store.
Certificate was added to keystore
Export the cluster certificate from the key store.
Sign the cluster certificate with the CA.
Signature ok
subject=/DC=com/DC=fenago/C=USA/ST=CA/L=San Francisco/O=Fenago/OU=Fenago/CN=Clou
dDurable Image kafka cluster
Getting CA Private Key
Import the Certificate Authority (CA) file into the key store.
Certificate was added to keystore
Import the Signed Cluster Certificate into the key store.
Certificate reply was installed in keystore
bash-4.2#
```

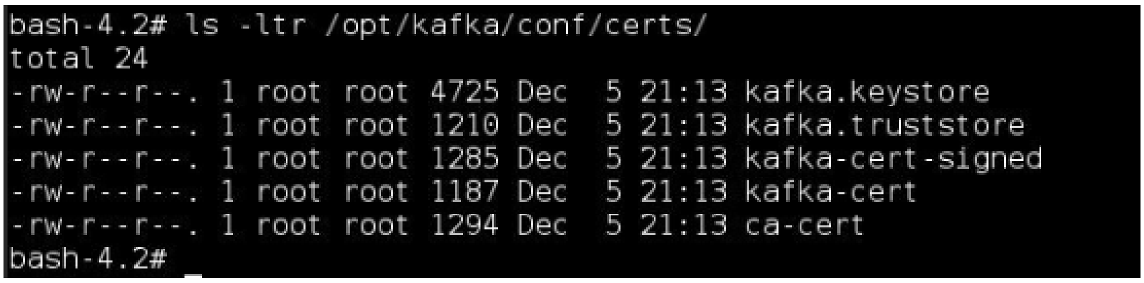
ACTION - RUN `bin/create-ssl-key-keystore.sh`

ACTION - COPY output of `bin/create-ssl-key-keystore.sh` to `/opt/kafka/`

```
$ sudo cp -R resources/opt/kafka/ /opt/
```

ACTION - See files generated `ls /opt/kafka/conf/certs/` (5 files)

ca-cert - Certificate Authority file - don't ship this around
kafka-cert - Kafka Certification File - public key and private key, don't ship this around
kafka-cert-signed - Kafka Certification File signed with CA - don't ship this around
kafka.keystore - needed on all clients and servers
kafka.truststore - needed on all clients and servers



```
bash-4.2# ls -ltr /opt/kafka/conf/certs/
total 24
-rw-r--r--. 1 root root 4725 Dec  5 21:13 kafka.keystore
-rw-r--r--. 1 root root 1210 Dec  5 21:13 kafka.truststore
-rw-r--r--. 1 root root 1285 Dec  5 21:13 kafka-cert-signed
-rw-r--r--. 1 root root 1187 Dec  5 21:13 kafka-cert
-rw-r--r--. 1 root root 1294 Dec  5 21:13 ca-cert
bash-4.2#
```

Configuring Kafka Servers

You will need to configure the listener's protocols for each server. In this example, we are using three servers. You will want to configure Kafka, so it is available on SSL and plaintext. The plaintext important for tools, and you could block Plaintext at firewalls or using routes.

You will need to pass in the truststore and keystore locations and passwords.

ACTION - EDIT `config/server-0.properties` and follow instructions

```
broker.id=0
listeners=PLAINTEXT://localhost:9092,SSL://localhost:10092
ssl.keystore.location=/opt/kafka/conf/certs/kafka.keystore
ssl.keystore.password=kafka123
ssl.key.password=kafka123
ssl.truststore.location=/opt/kafka/conf/certs/kafka.truststore
ssl.truststore.password=kafka123
ssl.client.auth=required

log.dirs=./logs/kafka-0
default.replication.factor=3
num.partitions=8
min.insync.replicas=2
auto.create.topics.enable=false
broker.rack=us-west2-a
queued.max.requests=1000
auto.leader.rebalance.enable=true

zookeeper.connect=localhost:2181
delete.topic.enable=true
compression.type=producer
message.max.bytes=65536
replica.lag.time.max.ms=5000
num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.recovery.threads.per.data.dir=1
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
zookeeper.connection.timeout.ms=6000
```

ACTION - EDIT `config/server-1.properties` and follow instructions

```
broker.id=1
listeners=PLAINTEXT://localhost:9093,SSL://localhost:10093
ssl.keystore.location=/opt/kafka/conf/certs/kafka.keystore
ssl.keystore.password=kafka123
ssl.key.password=kafka123
ssl.truststore.location=/opt/kafka/conf/certs/kafka.truststore
ssl.truststore.password=kafka123
ssl.client.auth=required
```

```
log.dirs=./logs/kafka-1
default.replication.factor=3
num.partitions=8
min.insync.replicas=2
auto.create.topics.enable=false
broker.rack=us-west2-a
queued.max.requests=1000
auto.leader.rebalance.enable=true

zookeeper.connect=localhost:2181
delete.topic.enable=true
compression.type=producer
message.max.bytes=65536
replica.lag.time.max.ms=5000
num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.recovery.threads.per.data.dir=1
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
zookeeper.connection.timeout.ms=6000
```

ACTION - EDIT `config/server-2.properties` and follow instructions

```
broker.id=2
listeners=PLAINTEXT://localhost:9094,SSL://localhost:10094
ssl.keystore.location=/opt/kafka/conf/certs/kafka.keystore
ssl.keystore.password=kafka123
ssl.key.password=kafka123
ssl.truststore.location=/opt/kafka/conf/certs/kafka.truststore
ssl.truststore.password=kafka123
ssl.client.auth=required

log.dirs=./logs/kafka-2
default.replication.factor=3
num.partitions=8
min.insync.replicas=2
auto.create.topics.enable=false
broker.rack=us-west2-a
queued.max.requests=1000
auto.leader.rebalance.enable=true

zookeeper.connect=localhost:2181
delete.topic.enable=true
compression.type=producer
message.max.bytes=65536
replica.lag.time.max.ms=5000
num.network.threads=3
num.io.threads=8
```

```
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.recovery.threads.per.data.dir=1
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
zookeeper.connection.timeout.ms=6000
```

Configure Kafka Consumer

You will need to pass in truststore and keystore locations and passwords to the consumer.

ACTION - EDIT `src/main/java/com/fenago/kafka/consumer/ConsumerUtil.java` and follow instructions in file.

```
package com.fenago.kafka.consumer;

import com.fenago.kafka.model.StockPrice;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;

import static java.util.concurrent.Executors.newFixedThreadPool;

public class ConsumerUtil {

    public static final String BROKERS =
"localhost:10092,localhost:10093,localhost:10094";

    private static Consumer<String, StockPrice> createConsumer(
            final String bootstrapServers, final String clientId ) {

        final Properties props = new Properties();

        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                BROKERS);

        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SSL");
        props.put("ssl.truststore.location",
"/opt/kafka/conf/certs/kafka.truststore");
```

```
        props.put("ssl.truststore.password", "kafka123");
        props.put("ssl.keystore.location", "/opt/kafka/conf/certs/kafka.keystore");
        props.put("ssl.keystore.password", "kafka123");

        //Turn off auto commit - "enable.auto.commit".
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);
        props.put(ConsumerConfig.CLIENT_ID_CONFIG, clientId);
        props.put(ConsumerConfig.GROUP_ID_CONFIG,
                "StockPriceConsumer");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class.getName());
        //Custom Deserializer
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                StockDeserializer.class.getName());
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500);

        // Create the consumer using props.
        return new KafkaConsumer<>(props);
    }
...
}
```

Configure Kafka Producer

You will need to pass in truststore and keystore locations and passwords to the producer.

ACTION - EDIT `src/main/java/com/fenago/kafka/producer/support/ProducerUtils.java` and follow instructions in file.

```
package com.fenago.kafka.producer.support;

import com.fenago.kafka.model.StockPrice;
import io.advantageous.boon.core.Lists;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;
import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class StockPriceProducerUtils {

    private static Producer<String, StockPrice> createProducer() {
        final Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "localhost:10092,localhost:10093");

        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SSL");
        props.put("ssl.truststore.location",
"/opt/kafka/conf/certs/kafka.truststore");
        props.put("ssl.truststore.password", "kafka123");
        props.put("ssl.keystore.location", "/opt/kafka/conf/certs/kafka.keystore");
        props.put("ssl.keystore.password", "kafka123");

        props.put(ProducerConfig.CLIENT_ID_CONFIG, "StockPriceProducerUtils");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                StockPriceSerializer.class.getName());
        props.put(ProducerConfig.LINGER_MS_CONFIG, 100);
        props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16_384 * 4);
        props.put(ProducerConfig.COMPRESSION_TYPE_CONFIG, "snappy");

        return new KafkaProducer<>(props);
    }
...
}
```

Run the lab

ACTION - RUN ZooKeeper and three Kafka Brokers (scripts are under bin for ZooKeeper and Kafka Brokers).

Note: Do not run scripts inside `bin` directory. Run scripts from `~/kafka-training/labs/lab8.1/solution` directory

Terminal 1

```
cd ~/kafka-training/labs/lab8.1/solution
bin/run-zookeeper.sh
```

Terminal 2

```
cd ~/kafka-training/labs/lab8.1/solution
bin/start-1st-server.sh
```

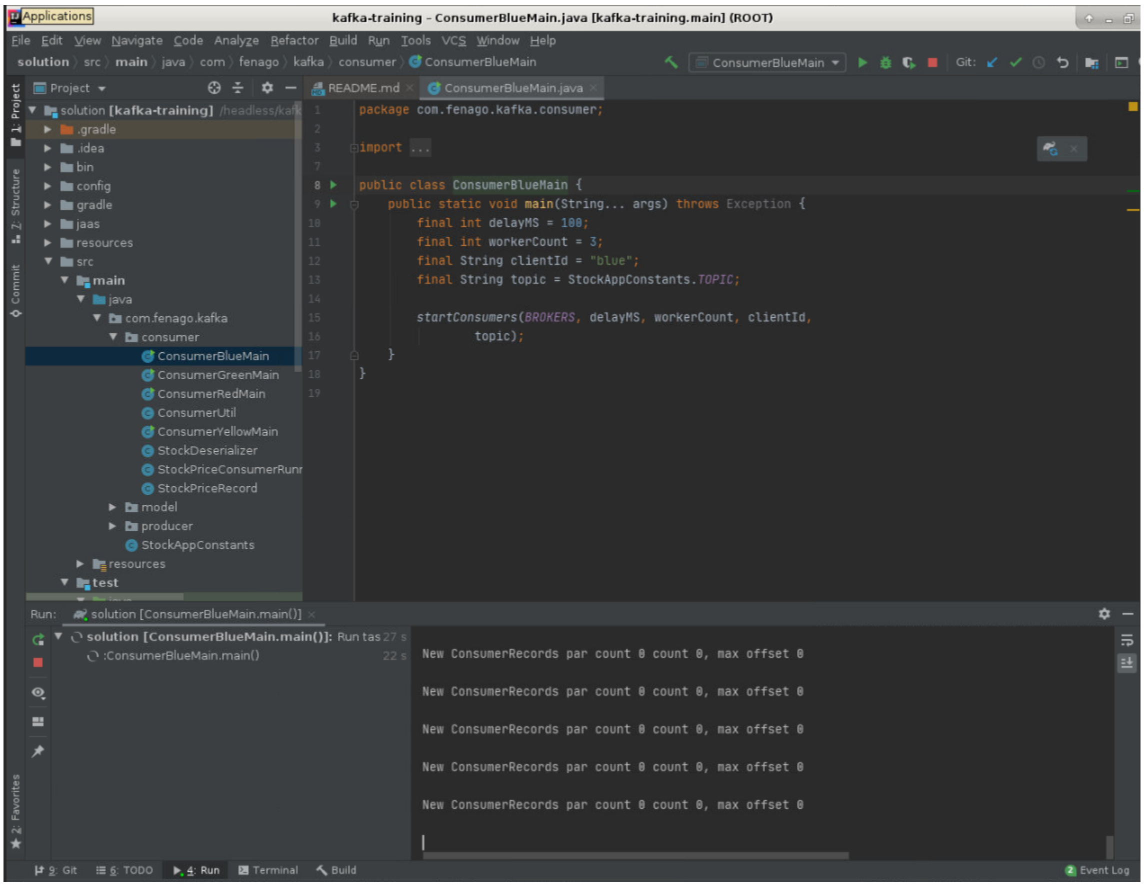
Terminal 3

```
cd ~/kafka-training/labs/lab8.1/solution
bin/start-2nd-server.sh
```

Terminal 4

```
cd ~/kafka-training/labs/lab8.1/solution
bin/start-3rd-server.sh
```

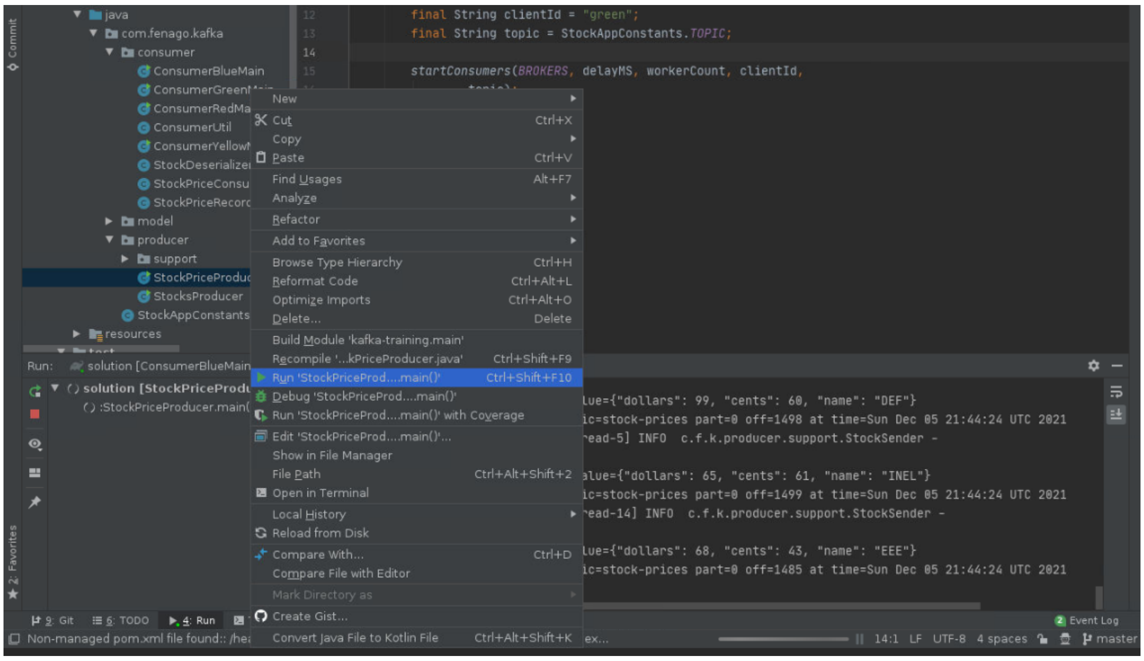
ACTION - RUN ConsumerBlueMain from the IDE



```java
package com.fenago.kafka.consumer;

import ...

public class ConsumerBlueMain {
    public static void main(String... args) throws Exception {
        final int delayMS = 100;
        final int workerCount = 3;
        final String clientId = "blue";
        final String topic = StockAppConstants.TOPIC;

        startConsumers(BROKERS, delayMS, workerCount, clientId,
                topic);
    }
}
```

```
New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0
```

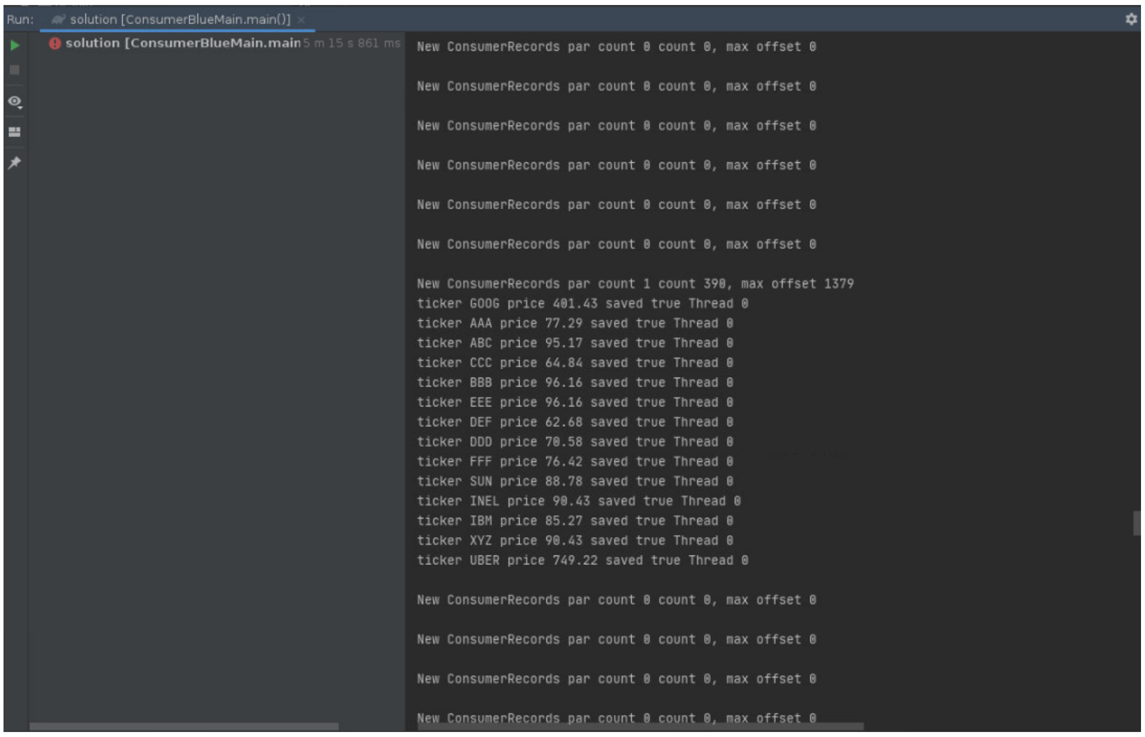
ACTION - RUN StockPriceProducer from the IDE



```java
        final String clientId = "green";
        final String topic = StockAppConstants.TOPIC;

        startConsumers(BROKERS, delayMS, workerCount, clientId,
```

```
lue={"dollars": 99, "cents": 60, "name": "DEF"}
ic=stock-prices part=0 off=1498 at time=Sun Dec 05 21:44:24 UTC 2021
read-5] INFO  c.f.k.producer.support.StockSender -

alue={"dollars": 65, "cents": 61, "name": "INEL"}
ic=stock-prices part=0 off=1499 at time=Sun Dec 05 21:44:24 UTC 2021
read-14] INFO  c.f.k.producer.support.StockSender -

lue={"dollars": 68, "cents": 43, "name": "EEE"}
ic=stock-prices part=0 off=1485 at time=Sun Dec 05 21:44:24 UTC 2021
```

Wait for some time and verify that messages are logged in consumer.


Run:
solution [ConsumerBlueMain.main()]
solution [ConsumerBlueMain.main 5 m 15 s 861 ms

```
New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 1 count 390, max offset 1379
ticker GOOG price 401.43 saved true Thread 0
ticker AAA price 77.29 saved true Thread 0
ticker ABC price 95.17 saved true Thread 0
ticker CCC price 64.84 saved true Thread 0
ticker BBB price 96.16 saved true Thread 0
ticker EEE price 96.16 saved true Thread 0
ticker DEF price 62.68 saved true Thread 0
ticker DDD price 70.58 saved true Thread 0
ticker FFF price 76.42 saved true Thread 0
ticker SUN price 88.78 saved true Thread 0
ticker INEL price 90.43 saved true Thread 0
ticker IBM price 85.27 saved true Thread 0
ticker XYZ price 90.43 saved true Thread 0
ticker UBER price 749.22 saved true Thread 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0

New ConsumerRecords par count 0 count 0, max offset 0
```

ProTip Scroll up to view complete consumer output.

Expected results

You should be able to send records from the producer to the broker and read records from the consumer to the broker using SSL.