Lab 8.3: Kafka Security SASL PLAIN

Welcome to the session 8 lab 3. The work for this lab is done in `~/kafka-training/lab8.3`. In this lab, you are going to Kafka SASL PLAIN.

Please refer to the Kafka course notes for any updates or changes to this lab.

Find the latest version of this lab here.

Kafka and SASL PLAIN

You should use ***SASL/PLAIN*** with ***SSL*** only as ***transport layer*** to ensure no clear text passwords are not transmitted.
The Kafka default implementation of ***SASL/PLAIN*** specifies *usernames* and *passwords* in ***JAAS*** config files.
To avoid storing passwords on disk, you could define and use your own implementation of `javax.security.auth.spi.LoginModule`, or use disk encryption and Unix permissions to protect the username and passwords.

In production systems, external authentication servers may implement password authentication. Kafka brokers can be integrated to work with these servers by adding your own implementation of `javax.security.sasl.SaslServer`. The default implementation included in Kafka in the package `org.apache.kafka.common.security.plain` can be used as an example.

Create JAAS config for ZooKeeper add admin user

To log into ZooKeeper, you would need user *admin* and and a password (*kafka-123*).
You would configure this via JAAS file called `zookeeper_jass.conf`.
which will live under `/opt/kafka/config/security/zookeeper_jass.conf`.

ZooKeeper JAAS file

~kafka-training/labs/lab8.3/resources/opt/kafka/conf/security/kafka_broker_jaas.conf

```
// Zookeeper server authentication
Server {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="admin"
  password="kafka-123"
  user_admin="kafka-123";
};
```

Note we are using PlainLoginModule from Kafka.

ACTION EDIT resources/opt/kafka/conf/security/kafka_broker_jaas.conf and follow instructions in file

Modify ZooKeeper properties file add SASL config

We need ZooKeeper to use `org.apache.zookeeper.server.auth.SASLAuthenticationProvider` as its authProvider. This `authProvider` requires JaaS login via SASL config/zookeeper.properties.

Use Kafka SASLAuthenticationProvider from ZooKeeper

~kafka-training/labs/lab8.3/config/zookeeper.properties

```
dataDir=/tmp/zookeeper-secure2
clientPort=2181
maxClientCnxns=0

authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
requireClientAuthScheme=sasl
jaasLoginRenew=3600000
```

Note we are using SASLAuthenticationProvider from Kafka.

ACTION EDIT config/zookeeper.properties and follow instructions in file

Modify ZooKeeper startup script add JAAS config location

We need to copy JAAS config files to `/opt/kafka/config/security ( cp -R resources/opt/kafka/conf/security /opt/kafka/conf/ )` . ***KAFKA_OPTS*** used by kafka startup scripts to pass extra args to JVM.

Make ZooKeeper use JAAS config file

~kafka-training/labs/lab8.3/bin/run-zookeeper.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

export KAFKA_JAAS_FILE="/opt/kafka/conf/security/zookeeper_jaas.conf"
export KAFKA_OPTS="-Djava.security.auth.login.config=$KAFKA_JAAS_FILE"

## Run ZooKeeper
kafka/bin/zookeeper-server-start.sh \
   "$CONFIG/zookeeper.properties"
```

ACTION EDIT bin/run-zookeeper.sh and follow instructions in file

Create JAAS config for Kafka Brokers add users (admin, consumer, producer)

We will also need a JAAS config file for the broker which will live under `/opt/kafka/conf/security/kafka_broker_jaas.conf` . This JAAS config file will set up users for admin for zookeeper, and for inter-broker communication, as well as set up users for consumers and producers.

JAAS config file for Broker

~kafka-training/labs/lab8.3/resources/opt/kafka/conf/security/kafka_broker_jaas.conf

```
KafkaServer {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="admin"
  password="kafka-123"
  user_admin="kafka-123"
  user_stocks_consumer="consumer123"
  user_stocks_producer="producer123";
};

// Zookeeper client authentication
Client {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="admin"
  password="kafka-123";
};
```

ACTION - EDIT `resources/opt/kafka/conf/security/kafka_broker_jaas.conf` and follow instructions in file

Modify Kafka Brokers Config properties file add SASL config

We will need to edit config files `config/server-0.properties` , `config/server-1.properties` , `config/server-2.properties` .
Enabled ***SASL*** support to use ***PLAIN SASL***.

Inter-broker communication is using ***SASL_SSL*** and config producers and consumers to use ***10092, 10093, 10094*** with ***SASL_SSL*** protocol.

~kafka-training/labs/lab8.3/config/server-0.properties

```
broker.id=0

listeners=PLAINTEXT://localhost:9092,SASL_SSL://localhost:10092
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
security.inter.broker.protocol=SASL_SSL

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-0.properties and follow directions**

~kafka-training/labs/lab8.3/config/server-1.properties

```
broker.id=1
listeners=PLAINTEXT://localhost:9093,SASL_SSL://localhost:10093
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
security.inter.broker.protocol=SASL_SSL

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
```

```
min.insync.replicas=1
auto.create.topics.enable=false
zookeeper.connect=localhost:2181
num.partitions=1
delete.topic.enable=true
broker.rack=rack1
auto.leader.rebalance.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 directions

~kafka-training/labs/lab8.3/config/server-2.properties

```
broker.id=2

listeners=PLAINTEXT://localhost:9094,SASL_SSL://localhost:10094
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
security.inter.broker.protocol=SASL_SSL

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
min.insync.replicas=1
auto.create.topics.enable=true
zookeeper.connect=localhost:2181
num.partitions=1
delete.topic.enable=true
broker.rack=rack2
auto.leader.rebalance.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 directions

Modify Kafka Broker startup script add JAAS config location

~kafka-training/labs/lab8.3/bin/start-1st-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

export KAFKA_JAAS_FILE="/opt/kafka/conf/security/kafka_broker_jaas.conf"
export KAFKA_OPTS="-Djava.security.auth.login.config=$KAFKA_JAAS_FILE"

## Run Kafka for 1st Server
kafka/bin/kafka-server-start.sh \
   "$CONFIG/server-0.properties"
```

ACTION - EDIT bin/start-1st-server.sh and follow directions

~kafka-training/labs/lab8.3/bin/start-1st-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

export KAFKA_JAAS_FILE="/opt/kafka/conf/security/kafka_broker_jaas.conf"
export KAFKA_OPTS="-Djava.security.auth.login.config=$KAFKA_JAAS_FILE"

## Run Kafka for 1st Server
kafka/bin/kafka-server-start.sh \
   "$CONFIG/server-0.properties"
```

ACTION - EDIT bin/start-2nd-server.sh and follow directions

~kafka-training/labs/lab8.3/bin/start-1st-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

export KAFKA_JAAS_FILE="/opt/kafka/conf/security/kafka_broker_jaas.conf"
export KAFKA_OPTS="-Djava.security.auth.login.config=$KAFKA_JAAS_FILE"

## Run Kafka for 1st Server
kafka/bin/kafka-server-start.sh \
   "$CONFIG/server-0.properties"
```

ACTION - EDIT bin/start-3rd-server.sh and follow directions

Create JAAS config for Consumer add user

We will need to configure username and password in JAAS file to log into Kafka Brokers.

~kafka-training/labs/lab8.3/resources/opt/kafka/conf/security/kafka_consumer_stocks_jaas.conf

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="stocks_consumer"
  password="consumer123";
};
```

ACTION - EDIT resources/opt/kafka/conf/security/kafka_consumer_stocks_jaas.conf and follow directions

Modify Consumer createConsumer() add SASL config and JAAS config location

~kafka-training/labs/lab8.3/src/main/java/com/cloudurable/kafka/consumer/ConsumerUtil.java

```
package com.cloudurable.kafka.consumer;

import com.cloudurable.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";

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

        System.setProperty("java.security.auth.login.config",
                "/opt/kafka/conf/security/kafka_consumer_stocks_jaas.conf");

        final Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                bootstrapServers);
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");
        props.put("ssl.keystore.location",
                "/opt/kafka/conf/certs/kafka.keystore");
        props.put("ssl.keystore.password", "kafka123");
        props.put("ssl.truststore.location",
                "/opt/kafka/conf/certs/kafka.truststore");
        props.put("ssl.truststore.password", "kafka123");
        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());
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                StockDeserializer.class.getName());
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500);
        return new KafkaConsumer<>(props);
    }
...
}
```

ACTION - EDIT src/main/java/com/cloudurable/kafka/consumer/ConsumerUtil.java and follow directions

Create JAAS config for Producer add user

We will need to configure username and password in JAAS file to log into Kafka Brokers.

~kafka-training/labs/lab8.3/resources/opt/kafka/conf/security/kafka_producer_stocks_jaas.conf

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="stocks_producer"
  password="producer123";
};
```

ACTION - EDIT resources/opt/kafka/conf/security/kafka_producer_stocks_jaas.conf and follow directions

Modify Producer createProducer() add SASL config and JAAS config location

~kafka-training/labs/lab8.3/src/main/java/com/cloudurable/kafka/producer/support/StockPriceProducerUtils.java

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

import com.cloudurable.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() {

        System.setProperty("java.security.auth.login.config",
                "/opt/kafka/conf/security/kafka_producer_stocks_jaas.conf");

        final Properties props = new Properties();

        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "localhost:10092,localhost:10093");
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");

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

```
        props.put("ssl.truststore.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);
    }
...
}
```

ACTION - EDIT src/main/java/com/cloudurable/kafka/producer/support/StockPriceProducerUtils.java and follow directions

ACTION - COPY JAAS files `cp -R resources/opt/kafka/conf/security /opt/kafka/conf/`

Run the lab

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

ACTION - RUN ConsumerBlueMain from the IDE

ACTION - RUN StockPriceProducer from the IDE

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 SASL PLAIN auth.

Kafka Tutorial

This comprehensive *Kafka tutorial* covers Kafka architecture and design. The *Kafka tutorial* has example Java Kafka producers and Kafka consumers. The *Kafka tutorial* also covers Avro and Schema Registry.

Complete Kafka Tutorial: Architecture, Design, DevOps and Java Examples.

- Kafka Tutorial Part 1: What is Kafka?
- Kafka Tutorial Part 2: Kafka Architecture
- Kafka Tutorial Part 3: Kafka Topic Architecture
- Kafka Tutorial Part 4: Kafka Consumer Architecture
- Kafka Tutorial Part 5: Kafka Producer Architecture
- Kafka Tutorial Part 6: Using Kafka from the command line
- Kafka Tutorial Part 7: Kafka Broker Failover and Consumer Failover
- Kafka Tutorial Part 8: Kafka Ecosystem
- Kafka Tutorial Part 9: Kafka Low-Level Design
- Kafka Tutorial Part 10: Kafka Log Compaction Architecture
- Kafka Tutorial Part 11: Writing a Kafka Producer example in Java
- Kafka Tutorial Part 12: Writing a Kafka Consumer example in Java
- Kafka Tutorial Part 13: Writing Advanced Kafka Producer Java examples
- Kafka Tutorial 14: Writing Advanced Kafka Consumer Java examples

- Kafka Tutorial Part 15: Kafka and Avro
- Kafka Tutorial Part 16: Kafka and Schema Registry
- Kafka Tutorial

About Cloudurable

We hope you enjoyed this article. Please provide feedback. Cloudurable provides Kafka training, Kafka consulting, Kafka support and helps setting up Kafka clusters in AWS.