Kafka Admin Client and Security

Apache Kafka

Machine Learning + Big Data in Real Time + Cloud Technologies

=> The Future of Intelligent Systems

Where to Find The Code and Materials?

https://github.com/iproduct/course-apache-kafka

Java Security

Java Security: Basic Concepts

- Authentication
- Authorization
- Data integrity
- Confidentiality
- Non-repudiation
- Auditing
- Quality of Service
- Role
- Realm
- User
- Group
- Principal
- Access Control List (ACL)

Java Security: Key Java SE Technologies - I

- Java Authentication and Authorization Service (JAAS) – defines extensible model for adding new Pluggable Authentication Modules (PAM)
- JavaGeneric Security Services (Java GSS-API) – token-based API for secure message exchange, unifying access to a number of security technologies, including Kerberos
- Java Cryptography Extension (JCE) – provides ability to encrypt, generate and agree symmetric, asymmetric, block and stream cyphers

Java Security: Key Java SE Technologies - II

- Java Secure Sockets Extension (JSSE) – provides implementation of SSL and TLS protocols in Java, allowing encryption, server authentication (and optionally clients), and message integrity
- Simple Authentication and Security Layer (SASL) – standard defining authentication protocol and a security layer establishment between client and server (e.g. Kafka broker), providing a framework for implementation of concrete authentication mechanisms (semantics and content)

Java Authentication and Authorization Service (JAAS)

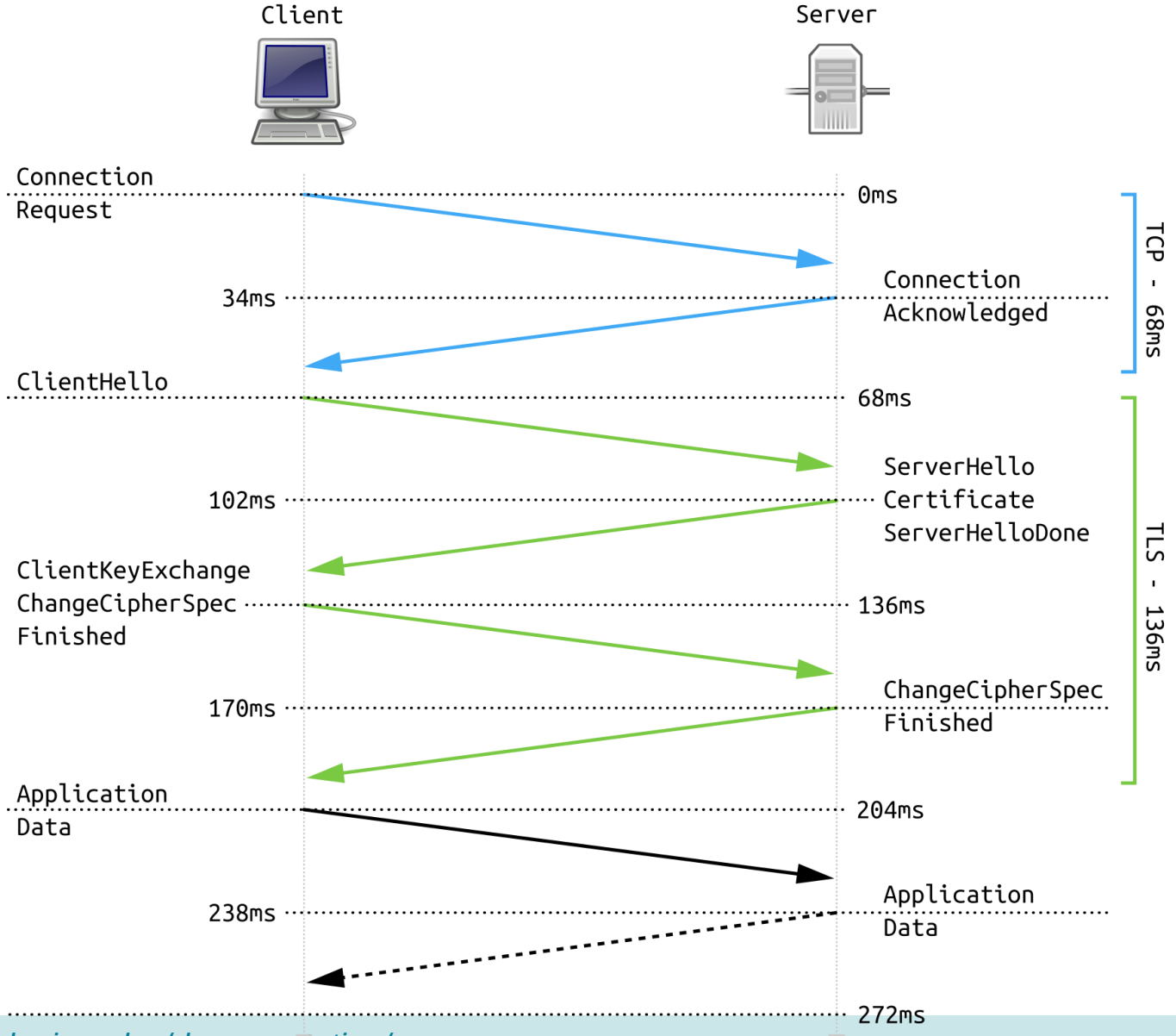
- JAAS can be used for two purposes:
  - Authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, bean, or a servlet;
  - Authorization of users to ensure they have the access control rights (permissions) required to do the actions performed.
- JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework, making login services independent from authentication technologies employed.
- JAAS provides a framework that enforces access controls based on who runs the code in the Java security architecture.
- Kafka uses the JAAS for SASL configuration.

Transport Layer Security (TLS)

- Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible.
- The TLS protocol aims primarily to provide cryptography, including privacy (confidentiality), integrity, and authenticity through the use of certificates, between two or more communicating computer applications. It runs in the application layer and is itself composed of two layers: the TLS record and the TLS handshake protocols.
- TLS is a proposed Internet Engineering Task Force (IETF) standard, first defined in 1999, and the current version is TLS 1.3, defined in August 2018.
- TLS is the successor of the now-deprecated Secure Sockets Layer (SSL).

https://www.selenium.dev/documentation/

Simplified TLS 1.2 Handshake



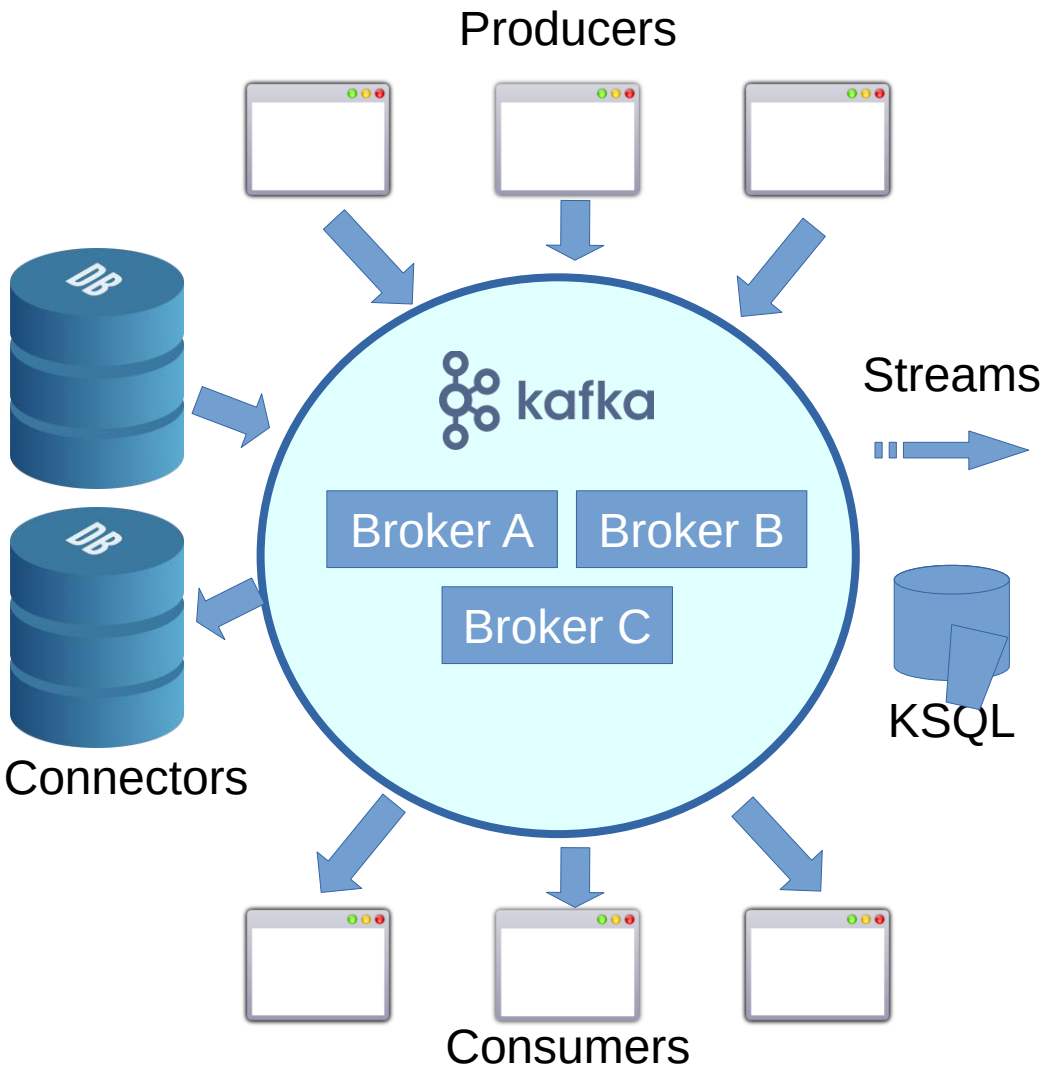
Using Self-Signed Certificates

- keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass changeit
- keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore springboot.p12 -validity 3650 -storepass changeit
- keytool -list -v -keystore springboot.jks
- keytool -list -v -keystore springboot.p12
- keytool -importkeystore -srckeystore springboot.jks -destkeystore springboot.p12 -deststoretype pkcs12
- keytool -import -alias springboot -file myCertificate.crt -keystore springboot.p12 -storepass password
- keytool -export -keystore springboot.p12 -alias springboot -file myCertificate.crt

Kafka Security

Kafka Core APIs

- The Producer API - publish a stream of records to one or more Kafka topics.
- The Consumer API - subscribe to one or more topics and process the stream of records produced to them.
- The Streams API - a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems – e.g. connector to a DB might capture every change in a table



Kafka Security Features - I

Kafka 0.9.0.0 added a number of security features, used either separately or together, increasing security in a Kafka cluster:

- Authentication of connections to brokers from clients (producers and consumers), other brokers and tools, using either SSL or SASL. Kafka supports the following SASL mechanisms:
  - SASL/GSSAPI (Kerberos) - starting at version 0.9.0.0
  - SASL/PLAIN - starting at version 0.10.0.0
  - SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512 - starting at version 0.10.2.0
  - SASL/OAUTHBEARER - starting at version 2.0
- Authentication of connections from brokers to ZooKeeper

https://kafka.apache.org/documentation.html#security

Kafka Security Features - II

- Encryption of data transferred between brokers and clients, between brokers, or between brokers and tools using SSL (Note that there is a performance degradation when SSL is enabled, the magnitude of which depends on the CPU type and the JVM implementation.)
- Authorization of read / write operations by clients
- Authorization is pluggable and integration with external authorization services is supported
- Security is optional - non-secured clusters are supported, as well as a mix of authenticated, unauthenticated, encrypted and non-encrypted clients. The guides below explain how to configure and use the security features in both clients and brokers: https://kafka.apache.org/documentation.html#security

https://kafka.apache.org/documentation.html#security

Kafka Security Using TLS

- Each broker needs its own private-key/certificate pair, and the client uses the certificate to authenticate the broker.
- Each logical client needs a private-key/certificate pair if client authentication is enabled, and the broker uses the certificate to authenticate the client.
- You can configure each broker and logical client with a truststore, which is used to determine which certificates (broker or logical client identities) to trust (authenticate). You can configure the truststore in many ways. Consider the following two examples:
  - The truststore contains one or many certificates: the broker or logical client will trust any certificate listed in the truststore.
  - The truststore contains a Certificate Authority (CA): the broker or logical client will trust any certificate that was signed by the CA in the truststore.
  - Using the CA method is more convenient, because adding a new broker or client doesn't require a change to the truststore.

https://kafka.apache.org/documentation.html#security

Kafka Security Using TLS

- Each broker needs its own private-key/certificate pair, and the client uses the certificate to authenticate the broker.
- Each logical client needs a private-key/certificate pair if client authentication is enabled, and the broker uses the certificate to authenticate the client.
- You can configure each broker and logical client with a truststore, which is used to determine which certificates (broker or logical client identities) to trust (authenticate). You can configure the truststore in many ways. Consider the following two examples:
  - The truststore contains one or many certificates: the broker or logical client will trust any certificate listed in the truststore.
  - The truststore contains a Certificate Authority (CA): the broker or logical client will trust any certificate that was signed by the CA in the truststore.
  - Using the CA method is more convenient, because adding a new broker or client doesn't require a change to the truststore: https://github.com/confluentinc/confluent-platform-security-tools/blob/master/single-trust-store-diagram.pdf

Kafka SSL Security – Prepare Keys

[https://kafka.apache.org/documentation/#security_ssl]

[https://docs.oracle.com/en/java/javase/11/tools/keytool.html]

keytool -genkeypair -alias localhost -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore server.keystore.p12 -validity 365 -storepass changeit -ext SAN=DNS:{FQDN},IP:{IPADDRESS1}

keytool -list -v -keystore server.keystore.p12

openssl req -x509 -config openssl-ca.cnf -newkey rsa:4096 -sha256 -nodes -out cacert.pem -outform PEM -days 365

keytool -keystore client.truststore.jks -alias CARoot -import -file cacert.pem

keytool -keystore server.truststore.jks -alias CARoot -import -file cacert.pem

keytool -keystore server.keystore.p12 -alias localhost -certreq -file cert-file.crt

openssl x509 -req -CA cacert.pem -CAkey cakey.pem -in cert-file.crt -out cert-signed.crt -days 365 -CAcreateserial -passin pass:changeit

keytool -keystore server.keystore.p12 -alias CARoot -import -file cacert.pem

keytool -keystore server.keystore.p12 -alias localhost -import -file cert-signed.crt

openssl x509 -in certificate.crt -text -noout

Kafka SSL Security - Brokers

[https://kafka.apache.org/documentation/#security]

```
############################## SSL Config
##############################################
listeners=PLAINTEXT://:9092,SSL://:8092

ssl.endpoint.identification.algorithm=
ssl.keystore.location=server.keystore.p12
ssl.keystore.password=changeit
ssl.key.password=changeit
ssl.truststore.location=server.truststore.jks
ssl.truststore.password=changeit
ssl.client.auth=none
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
ssl.keystore.type= PKCS12
ssl.truststore.type=JKS
```

Kafka SSL Security - Clients

[https://kafka.apache.org/documentation/#security]

```
############################## SSL Config
##############################
bootstrap.servers=localhost:8092

ssl.endpoint.identification.algorithm=
security.protocol=SSL
ssl.truststore.location=client.truststore.jks
ssl.truststore.password=changeit
ssl.truststore.type=JKS
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
```

```
openssl s_client -debug -connect localhost:8092 -tls1_2

kafka-console-producer --broker-list localhost:8092 --topic temperature --
producer.config config/producer-ssl.properties
kafka-console-consumer.bat --bootstrap-server localhost:8092 --topic
temperature --from-beginning --consumer.config config/consumer-ssl.properties
```

SASL PLAIN + ACL Configuration

[https://kafka.apache.org/documentation.html#security_sasl_plain]

```
################################ Enable SASL - PLAIN Authentication ########################
listeners=PLAINTEXT://:9092,CONTROLLER://:19092,SASL_SSL_INTERNAL://:8092,SASL_SSL://:8093

security.inter.broker.protocol=SASL_SSL
security.protocol=SASL_SSL
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.mechanism.controller.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN


################################ ACL Authorization ##################################
authorizer.class.name=org.apache.kafka.metadata.authorizer.StandardAuthorizer
allow.everyone.if.no.acl.found=true
super.users=User:admin
ssl.principal.mapping.rules=RULE:^CN=(.*?),OU=ServiceUsers.*$/$1/L
```

SASL PLAIN Client Configuration

[https://kafka.apache.org/documentation.html#security_sasl_plain]

```
props.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:8093");
// SSL security config
//        props.put("security.protocol", "SSL");
props.put("ssl.endpoint.identification.algorithm", "");
props.put("ssl.truststore.location", "D:\\CourseKafka\\kafka_2.13-3.2.0\\client.truststore.jks");
props.put("ssl.truststore.password", "changeit");
props.put("ssl.truststore.type", "JKS");
props.put("ssl.enabled.protocols", "TLSv1.2,TLSv1.1,TLSv1");
props.put("ssl.protocol", "TLSv1.2");

// SASL PLAIN Authentication
props.put("sasl.jaas.config", "org.apache.kafka.common.security.plain.PlainLoginModule required
username='admin' password='admin123';");
props.put("security.protocol", "SASL_SSL");
props.put("sasl.mechanism", "PLAIN");
```

Configuring ACLs Programmatically Using Admin Client

[https://docs.confluent.io/platform/current/kafka/authorization.html]

```
admin.createAcls(Collections.singleton(
            new AclBinding(new ResourcePattern(TOPIC, "temperature2", LITERAL),
                  new AccessControlEntry("User:admin", "*", AclOperation.ALL, AclPermissionType.ALLOW))))
      .values().forEach((aclBinding, voidKafkaFuture) -> log.info(">>>ACL Entry: {}", aclBinding));
admin.deleteAcls(Collections.singleton(new AclBindingFilter(
      new ResourcePatternFilter(TOPIC,"temperature2", ANY),
      new AccessControlEntryFilter("trayan", null,  AclOperation.ANY, AclPermissionType.ANY))))
      .values().forEach((aclBindingFilter, filterResultsKafkaFuture) -> {
         try {
            log.info(">>>>> DELETED ACL Entry: {}", filterResultsKafkaFuture.get());
         } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
         }
      });
admin.describeAcls(
      new AclBindingFilter(new ResourcePatternFilter(TOPIC,"temperature2", ANY),
      new AccessControlEntryFilter(null, null,  AclOperation.ANY, AclPermissionType.ANY)))
      .values().get().forEach((aclBinding) -> log.info("|||  CL Entry: {}", aclBinding));;
```

Configuring ACLs Using CLI - kafka-acls

[https://kafka.apache.org/documentation.html#security_authz_cli,
https://jaceklaskowski.gitbooks.io/apache-kafka/content/kafka-admin-AclCommand.html]

bin\windows\kafka-acls --bootstrap-server localhost:9093 --list --topic temperature2

```
-> Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=temperature2,
    patternType=LITERAL)`:
   (principal=admin, host=localhost, operation=ALL, permissionType=ALLOW)
```

bin\windows\kafka-acls --bootstrap-server localhost:9093 --list --topic temperature2 --resource-pattern-type match

```
-> Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=temperature2,
   patternType=LITERAL)`:
      (principal=User:admin, host=*, operation=DESCRIBE, permissionType=ALLOW)
      (principal=User:admin, host=*, operation=WRITE, permissionType=ALLOW)
      (principal=User:trayan, host=*, operation=ALL, permissionType=DENY)
      (principal=User:admin, host=*, operation=CREATE, permissionType=ALLOW)
```

bin\windows\kafka-acls --bootstrap-server localhost:9093 --add --allow-principal User:trayan --producer --topic temperature2

bin\windows\kafka-acls --bootstrap-server localhost:9093 --add --allow-principal User:trayan --allow-host * --operation Read --topic temperature2

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD