# Technical Write-up: Securing Our Kafka Cluster with SSL, Kerberos, and LDAP

## **Objective**

To enhance the security of our Kafka cluster by implementing:

* SSL Encryption: Encrypt data in transit between clients and brokers.
* Kerberos Authentication (SASL/GSSAPI): Securely authenticate clients and brokers.
* LDAP Integration: Centralize user management and authentication.

## **Overview**

We will:

1. Enable SSL encryption on Kafka brokers.
2. Set up a Kerberos Key Distribution Center (KDC) for authentication.
3. Integrate LDAP for user management.
4. Update our Docker Compose configuration to include these services.
5. Provide necessary configurations and instructions to the development team for client applications.

## **Serial Instruction List**

### **1. Enable SSL Encryption on Kafka Brokers**

Purpose: Encrypt data transmitted between Kafka clients and brokers to prevent eavesdropping and tampering.

Steps:

1. Generate SSL Certificates and Keystores:
   * Create a Certificate Authority (CA) to sign certificates.

bash

Copy code

openssl req -new -x509 -keyout ca-key -out ca-cert -days 365 -nodes -subj "/CN=Kafka-CA"

* + For each Kafka broker (kafka1, kafka2, kafka3):
    - Generate a keystore.

bash

Copy code

keytool -genkey -noprompt \

-alias <broker> \

-dname "CN=<broker>" \

-keystore <broker>.keystore.jks \

-keyalg RSA \

-storepass changeit \

-keypass changeit

* + - Create a Certificate Signing Request (CSR).

bash

Copy code

keytool -keystore <broker>.keystore.jks -alias <broker> -certreq -file <broker>.csr -storepass changeit

* + - Sign the CSR with the CA.

bash

Copy code

openssl x509 -req -CA ca-cert -CAkey ca-key -in <broker>.csr -out <broker>-cert-signed -days 365 -CAcreateserial

* + - Import the CA certificate into a truststore.

bash

Copy code

keytool -keystore <broker>.truststore.jks -alias CARoot -import -file ca-cert -storepass changeit -noprompt

* + - Import the signed certificate into the broker's keystore.

bash

Copy code

keytool -keystore <broker>.keystore.jks -alias <broker> -import -file <broker>-cert-signed -storepass changeit -noprompt

1. Update Broker Configuration:
   * In docker-compose.yml, for each Kafka broker, add SSL configurations:

yaml

Copy code

environment:

# ... existing configurations ...

KAFKA\_SSL\_KEYSTORE\_LOCATION: /etc/kafka/secrets/<broker>.keystore.jks

KAFKA\_SSL\_KEYSTORE\_PASSWORD: changeit

KAFKA\_SSL\_KEY\_PASSWORD: changeit

KAFKA\_SSL\_TRUSTSTORE\_LOCATION: /etc/kafka/secrets/<broker>.truststore.jks

KAFKA\_SSL\_TRUSTSTORE\_PASSWORD: changeit

KAFKA\_SSL\_ENDPOINT\_IDENTIFICATION\_ALGORITHM: ''

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: 'CONTROLLER:PLAINTEXT,INTERNAL:SSL,EXTERNAL:SSL'

KAFKA\_LISTENERS: 'INTERNAL://<broker>:29092,CONTROLLER://<broker>:29093,EXTERNAL://0.0.0.0:<port>'

KAFKA\_ADVERTISED\_LISTENERS: 'INTERNAL://<broker>:29092,EXTERNAL://<public\_ip>:<port>'

* + - Replace <broker>, <port>, and <public\_ip> accordingly.

1. Mount Keystores and Truststores:
   * In docker-compose.yml, add volume mounts for keystore and truststore files:

yaml

Copy code

volumes:

- ./secrets/<broker>.keystore.jks:/etc/kafka/secrets/<broker>.keystore.jks

- ./secrets/<broker>.truststore.jks:/etc/kafka/secrets/<broker>.truststore.jks

### **2. Set Up Kerberos Authentication (SASL/GSSAPI)**

Purpose: Authenticate clients and brokers securely using Kerberos.

Steps:

1. Deploy Kerberos KDC using Docker:
   * Add a KDC service in docker-compose.yml:

yaml

Copy code

kdc:

image: galexrt/krb5-kdc:latest

container\_name: kerberos-kdc

environment:

KRB5\_REALM: EXAMPLE.COM

KRB5\_KDC\_ADMIN\_PASSWORD: "admin\_password"

KRB5\_KDC\_DATABASE\_PASSWORD: "db\_password"

ports:

- "88:88"

- "749:749"

volumes:

- ./krb5kdc:/var/kerberos/krb5kdc

networks:

- kafka-net

1. Create Kerberos Principals and Keytabs:
   * For each broker:
     + Add a principal:

bash

Copy code

docker exec kerberos-kdc kadmin.local -w admin\_password -q "addprinc -randkey kafka/<broker>@EXAMPLE.COM"

* + - Generate a keytab:

bash

Copy code

docker exec kerberos-kdc kadmin.local -w admin\_password -q "ktadd -k /tmp/<broker>.keytab kafka/<broker>@EXAMPLE.COM"

* + - Copy the keytab to the host:

bash

Copy code

docker cp kerberos-kdc:/tmp/<broker>.keytab ./secrets/<broker>.keytab

1. Update Broker Configuration for SASL/Kerberos:
   * In docker-compose.yml, add SASL configurations:

yaml

Copy code

environment:

# ... existing configurations ...

KAFKA\_SASL\_ENABLED\_MECHANISMS: GSSAPI

KAFKA\_SASL\_MECHANISM\_INTER\_BROKER\_PROTOCOL: GSSAPI

KAFKA\_SASL\_KERBEROS\_SERVICE\_NAME: kafka

KAFKA\_SECURITY\_INTER\_BROKER\_PROTOCOL: SASL\_SSL

KAFKA\_OPTS: "-Djava.security.auth.login.config=/etc/kafka/secrets/kafka\_server\_jaas.conf -Djava.security.krb5.conf=/etc/krb5.conf"

1. Create JAAS Configuration Files:
   * For each broker, create kafka\_server\_jaas.conf:

properties

Copy code

KafkaServer {

com.sun.security.auth.module.Krb5LoginModule required

useKeyTab=true

storeKey=true

keyTab="/etc/kafka/secrets/<broker>.keytab"

principal="kafka/<broker>@EXAMPLE.COM";

};

* + Mount the JAAS config in docker-compose.yml:

yaml

Copy code

volumes:

- ./secrets/kafka\_server\_jaas.conf:/etc/kafka/secrets/kafka\_server\_jaas.conf

- ./secrets/<broker>.keytab:/etc/kafka/secrets/<broker>.keytab

1. Create Kerberos Configuration File (krb5.conf):
   * Define the Kerberos realm and KDC settings:

ini

Copy code

[libdefaults]

default\_realm = EXAMPLE.COM

[realms]

EXAMPLE.COM = {

kdc = kerberos-kdc

admin\_server = kerberos-kdc

}

[domain\_realm]

.example.com = EXAMPLE.COM

example.com = EXAMPLE.COM

* + Mount krb5.conf in docker-compose.yml:

yaml

Copy code

volumes:

- ./secrets/krb5.conf:/etc/krb5.conf

### **3. Integrate LDAP for User Management**

Purpose: Centralize user management and authentication using LDAP.

Steps:

1. Set Up LDAP Server:
   * Add an LDAP service in docker-compose.yml:

yaml

Copy code

ldap:

image: osixia/openldap:1.5.0

container\_name: openldap

environment:

LDAP\_ORGANISATION: "Example Inc."

LDAP\_DOMAIN: "example.com"

LDAP\_ADMIN\_PASSWORD: "admin\_password"

ports:

- "389:389"

networks:

- kafka-net

1. Configure Kafka to Use LDAP:
   * Since Kafka doesn't directly support LDAP for authentication, we'll use LDAP for user management in conjunction with Kerberos.
   * Users can be stored in LDAP and synchronized with Kerberos.

### **4. Update Docker Compose Configuration**

Purpose: Ensure all services are properly configured and connected.

Steps:

1. Include All Services in docker-compose.yml:
   * Kafka Brokers: kafka1, kafka2, kafka3 with updated configurations.
   * Kerberos KDC: kdc.
   * LDAP Server: ldap.
   * Control Center: control-center.
2. Configure Networks and Dependencies:
   * Define a network (kafka-net) for services to communicate.
   * Set depends\_on in services to ensure correct startup order.

### **5. Create Setup Script to Automate the Process**

Purpose: Automate the setup of SSL certificates, Kerberos principals, and starting services.

Steps:

1. Write setup\_kafka\_security.sh Script:
   * Automate SSL certificate generation.
   * Configure Kerberos principals and keytabs.
   * Generate JAAS configuration files.
   * Start the Docker Compose stack.
2. Make the Script Executable and Run It:

bash

Copy code

chmod +x setup\_kafka\_security.sh

./setup\_kafka\_security.sh

### **6. Provide Client Configurations to the Development Team**

Purpose: Ensure the development team can securely connect to the Kafka cluster.

Steps:

1. Generate Client Credentials:
   * Kerberos Principal and Keytab:
     + Create a principal for the client:

bash

Copy code

docker exec kerberos-kdc kadmin.local -w admin\_password -q "addprinc -randkey client@EXAMPLE.COM"

* + - Generate a keytab:

bash

Copy code

docker exec kerberos-kdc kadmin.local -w admin\_password -q "ktadd -k /tmp/client.keytab client@EXAMPLE.COM"

* + - Copy the keytab:

bash

Copy code

docker cp kerberos-kdc:/tmp/client.keytab ./secrets/client.keytab

* + Client Truststore and Keystore:
    - Import the CA certificate into the client truststore.
    - Generate a client certificate if mutual SSL authentication is required.

1. Provide Configuration Files:
   * client.keytab: For Kerberos authentication.
   * client.truststore.jks and client.keystore.jks: For SSL encryption.
   * client\_jaas.conf:

properties

Copy code

KafkaClient {

com.sun.security.auth.module.Krb5LoginModule required

useKeyTab=true

storeKey=true

keyTab="/path/to/client.keytab"

principal="client@EXAMPLE.COM";

};

* + krb5.conf: Kerberos configuration file.

1. Provide Client Properties for Kafka:

properties

Copy code

security.protocol=SASL\_SSL

sasl.mechanism=GSSAPI

sasl.kerberos.service.name=kafka

ssl.truststore.location=/path/to/client.truststore.jks

ssl.truststore.password=changeit

# If mutual SSL is used:

ssl.keystore.location=/path/to/client.keystore.jks

ssl.keystore.password=changeit

ssl.key.password=changeit

1. Assist the Development Team in Configuring Their Applications:
   * Provide sample code for producers and consumers.
   * Guide on setting environment variables and system properties.
   * Ensure they have network access to the Kafka brokers and Kerberos KDC.

### **7. Support the Development Team in Writing Secure Producer and Consumer Applications**

Purpose: Enable developers to create applications that interact securely with the Kafka cluster.

Steps:

1. Update Maven Project (pom.xml):
   * Include Kafka clients and logging dependencies.
2. Provide Sample Java Code:
   * Producer and Consumer Examples with SSL and Kerberos configurations.
3. Guide on Including Configuration Files:
   * jaas.conf and krb5.conf: How to include and reference them in the project.
4. Ensure Environment Setup:
   * Network Connectivity: Developers' machines can reach the Kafka brokers and KDC.
   * Time Synchronization: Clocks are synchronized for Kerberos.
5. Assist in Troubleshooting:
   * Enable debugging for SSL and Kerberos.
   * Check for common issues like incorrect principals or misconfigured paths.

## **Summary of Actions**

* Infrastructure Enhancement:
  + Implemented SSL encryption to secure data in transit.
  + Set up Kerberos authentication for secure client and broker communication.
  + Integrated LDAP for centralized user management.
  + Updated Docker Compose configurations to include all necessary services.
* Development Support:
  + Generated and provided necessary client-side credentials and configuration files.
  + Supplied sample code and guidance for developers to write secure applications.
  + Ensured developers understand how to integrate security configurations.

## **Next Steps**

1. Testing and Validation:
   * Internal Testing: Verify that the Kafka cluster is secure and functioning.
   * Client Testing: Ensure that client applications can connect and interact with the cluster.
2. Documentation and Training:
   * Documentation: Compile detailed documentation of all configurations and processes.
   * Team Training: Hold sessions to walk the development team through the new security measures.
3. Monitoring and Maintenance:
   * Certificate Management: Set up reminders for certificate renewals.
   * Security Audits: Regularly review security configurations.
   * Performance Monitoring: Ensure security measures do not adversely affect performance.

## **Conclusion**

By following these steps, we have significantly improved the security of our Kafka cluster. We now have:

* Encrypted Communication: All data between clients and brokers is encrypted using SSL.
* Secure Authentication: Clients and brokers authenticate using Kerberos.
* Centralized User Management: LDAP integration allows for efficient user management.

Our development team is equipped with the necessary tools and knowledge to develop secure applications that interact with the Kafka cluster.

## **Appendix**

### **Key Configuration Files**

* docker-compose.yml: Defines all services and configurations.
* setup\_kafka\_security.sh: Automates the setup process.
* kafka\_server\_jaas.conf: JAAS configuration for brokers.
* client\_jaas.conf: JAAS configuration for clients.
* krb5.conf: Kerberos realm and KDC settings.

### **Sample Commands**

* Starting the Docker Compose Stack:

bash

Copy code

docker-compose up -d

* Creating Kerberos Principals:

bash

Copy code

docker exec kerberos-kdc kadmin.local -w admin\_password -q "addprinc -randkey principal@EXAMPLE.COM"

* Generating SSL Certificates:

bash

Copy code

openssl req -new -x509 -keyout ca-key -out ca-cert -days 365 -nodes

## **Questions and Support**

If there are any questions or if assistance is needed during implementation or development, please reach out to the infrastructure team.