CROZ JavaCro'22 Spring

How to secure Kafka cluster using OAUTH

Javacro 2022

Miroslav Čerkez

CROZ d.o.o.

16.05.2022.

Contents

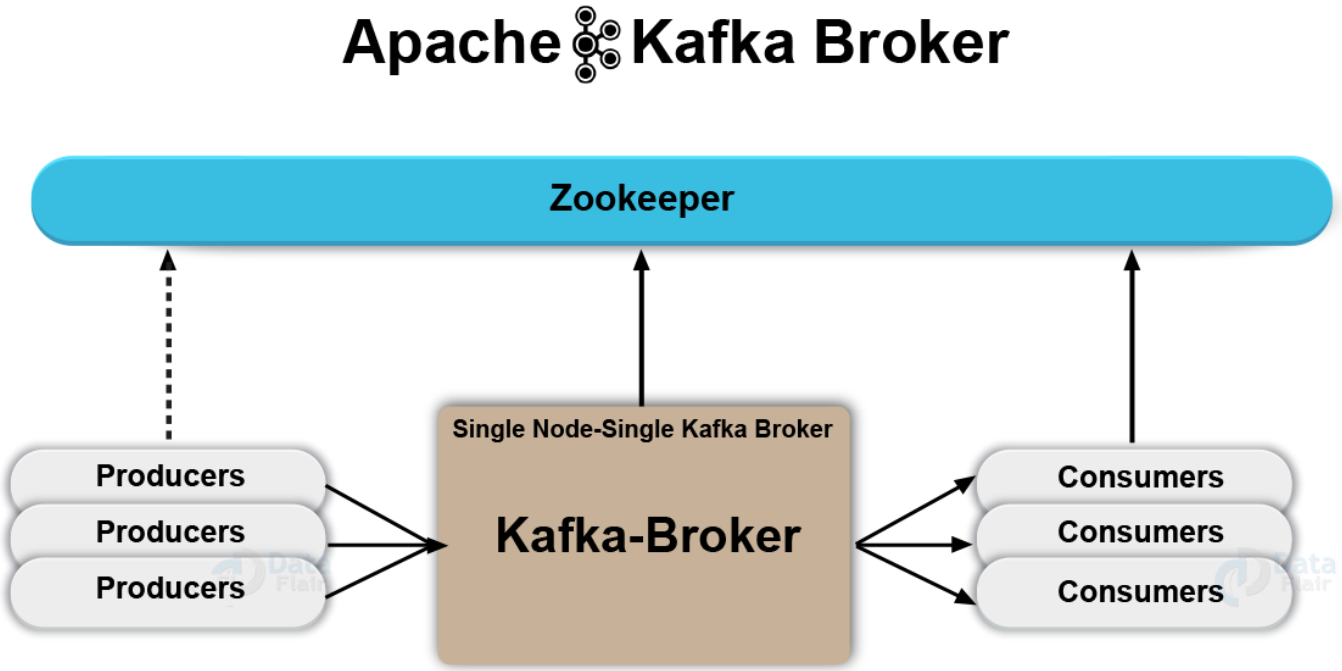
About me

- Software, tools, architecture and integration consultant at CROZ
- 12 years developing IT systems
- Focused on big mission critical systems (banking, transaction processing, ...)

- linkedin - mcerkez88

Kafka ecosystem components

- Infrastructure
  - Zookeeper
  - Kafka broker
- Applications
  - Consumers
  - Producer



Kafka ecosystem installation

- Bare metal
- VMs
- Cloud
  - As part of existing streaming platforms
  - As part of existing cloud offerings

- Kubernetes
  - Strimzi operator available from operator hub
  - https://strimzi.io

  - Provides declarative installation model
  - Battle tested
  - Easy to install

How do we secure access to Kafka Broker resources?

- By default Kafka has no authentication or authorization enabled and configured
- Anybody with access can perform any action on cluster
  - Great for testing and extermination

Authentication methods

- Different Kafka distributions support different authentication methods
  - 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

- Possible to configure multiple methods on same broker using multiple listeners

Authentication using OAUTH2 token

- Supported by Strimzi
- Externalized and centralized user management
- Most organization already familiar with OAUTH2

- Authentication only solves part of the problem.
- Once client is authenticated to broker it can still access all resources (topics)

Authorization methods

- Authorization is separated from authentication
- Kafka delegates authorization to Authorizer interface implementations (https://kafka.apache.org/28/javadoc/org/apache/kafka/server/authorizer/Authorizer.html)
- Based on ACLs
- Methods supported by Strimzi:
  - Simple (defined using operator)
  - Oauth
  - Open policy agent
  - Custom (usually Ranger)

  - Authorization is always configured for the whole Kafka cluster.

Authorization using Keycloak authorization services

- Create keycloak client with enabled authorization services
- Configure strimzi cluster to use kaycloak client for authorization service
- Permissions are defined using combination of Kafka authorization primitives
  - Operations
  - Resources

Operations and resources in Kafka

- Read
- Write
- Create
- Delete
- Alter
- Describe
- ClusterAction
- DescribeConfigs
- AlterConfigs
- IdempotentWrite
- All

- Topic
- Group
- Cluster
- TransactionalId
- DelegationToken

Demo

javacro / 02_kafka_broker / oauth_auth / kafka_cluster.yaml

Project

- javacro C:\Users\mcerkez\git\javacro
  - .idea
  - 01_keycloak
    - realm
    - keycloak.yaml
  - 02_kafka_broker
    - oauth_auth
      - cleanup.sh
      - kafka_cluster.yaml
      - kafka_cluster_ca.jks
      - kafka_cluster_ca.p12
      - kafka_cluster_oauth_template.yaml
      - keycloak.crt
      - keycloak_trustsore.jks
      - setup_kafka.sh
  - Consumers
    - src
      - main
        - java
          - hr.javacro.consumers
            - ConsumerMain
            - TeamAConsumer
        - resources
      - test
    - target
    - Consumers.iml
    - pom.xml
  - Producers
    - src
      - main
        - java
          - hr.javacro
            - ProducerMain
            - TeamAProducer
        - resources
      - test
    - target
    - pom.xml
    - Producers.iml
  - External Libraries
  - Scratches and Consoles

keycloak.yaml | TeamAProducer.java | kafka_cluster.yaml

```yaml
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      inter.broker.protocol.version: "3.1"
    listeners:
      - name: external
        port: 9094
        tls: true
        type: route
        authentication:
          checkIssuer: true
          jwksEndpointUri: >-
            https://keycloak-javacro.apps-crc.testing/auth/realms/kafka-authz/protocol/openid-connect/certs
          userNameClaim: preferred_username
          checkAccessTokenType: true
          accessTokenIsJwt: true
          enableOauthBearer: true
          validIssuerUri: >-
            https://keycloak-javacro.apps-crc.testing/auth/realms/kafka-authz
          tlsTrustedCertificates:
            - certificate: keycloak.crt
              secretName: ca-keycloak
          type: oauth
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    authorization:
      type: keycloak
      clientId: kafka
      tokenEndpointUri: https://keycloak-javacro.apps-crc.testing/auth/realms/kafka-authz/protocol/openid-connect/token
      tlsTrustedCertificates:
        - secretName: ca-keycloak
          certificate: keycloak.crt
      delegateToKafkaAcls: true
```

javacro 〉 02_kafka_broker

Project

- javacro C:\Users\mcerkez\git\javacro
  - .idea
  - 01_keycloak
    - realm
    - keycloak.yaml
  - 02_kafka_broker
    - oauth_auth
      - cleanup.sh
      - kafka_cluster.yaml
      - kafka_cluster_ca.jks
      - kafka_cluster_ca.p12
      - kafka_cluster_oauth_template.yaml
      - keycloak.crt
      - keycloak_trustsore.jks
      - setup_kafka.sh
  - Consumers
    - src
      - main
        - java
          - hr.javacro.consumers
            - ConsumerMain
            - TeamAConsumer
        - resources
      - test
    - target
    - Consumers.iml
    - pom.xml
  - Producers
    - src
      - main
        - java
          - hr.javacro
            - ProducerMain
            - TeamAProducer
        - resources
      - test
    - target
    - pom.xml
    - Producers.iml
- External Libraries
- Scratches and Consoles

keycloak.yaml

```yaml
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: keycloak
  labels:
    app: sso
  namespace: javacro
spec:
  instances: 1
  externalAccess:
    enabled: True
  podDisruptionBudget:
    enabled: True
  # User needs to provision the external database this is for PoC only
  externalDatabase:
    enabled: false
```

Kafka

Settings | Credentials | Keys | Roles | Client Scopes | Mappers | Scope | Authorization | Revocation | Sessions | Offline Access | Clustering | Installation | Service Account Roles

Settings | Resources | Authorization Scopes | Policies | Permissions | Evaluate | Export Settings

Name

Create

Name	Actions
Alter	Create Permission
AlterConfigs	Create Permission
ClusterAction	Create Permission
Create	Create Permission
Delete	Create Permission
Describe	Create Permission
DescribeConfigs	Create Permission
IdempotentWrite	Create Permission
Read	Create Permission
Write	Create Permission

Clients › kafka › Authorization › Permissions › team-a-javacro-topic

Team-a-javacro-topic

Name *

team-a-javacro-topic

Description

Apply to Resource Type

OFF

Resources *

Topic:javacro

Apply Policy

Select existing policy...

Create Policy...

Name	Description	Actions
Dev Team A		Remove

Decision Strategy

Unanimous

Save Cancel

```java
public TeamAProducer () {
    Properties props = new Properties();

    // Configure kafka settings
    props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOSTRAP_SERVER);
    props.putIfAbsent(ProducerConfig.CLIENT_ID_CONFIG, "producer-user");
    props.putIfAbsent(ProducerConfig.ACKS_CONFIG, "all");
    props.putIfAbsent(ProducerConfig.BATCH_SIZE_CONFIG, 0);
    props.putIfAbsent(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
    // Use the Apicurio Registry provided Kafka Serializer for Avro
    props.putIfAbsent(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

    //props.putIfAbsent(ProducerConfig.RETRIES_CONFIG, 3);
    props.putIfAbsent(ProducerConfig.LINGER_MS_CONFIG, 2*1000);
    props.putIfAbsent(ProducerConfig.REQUEST_TIMEOUT_MS_CONFIG, 2*1000);
    props.putIfAbsent(ProducerConfig.DELIVERY_TIMEOUT_MS_CONFIG, 5*1000);

    //configure the following three settings for SSL Encryption
    props.putIfAbsent(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
    props.setProperty("sasl.mechanism", "OAUTHBEARER");
    props.setProperty("sasl.jaas.config", "org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;");
    props.setProperty("sasl.login.callback.handler.class", "io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandler");
    props.putIfAbsent(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, "C:\\Users\\mcerkez\\git\\javacro\\02_kafka_broker\\oauth_auth\\kafka_cluster_ca.jks");
    props.setProperty("ssl.keystore.type", "jks");
    props.putIfAbsent(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "X0zuGhejaQYK");



    Properties defaults = new Properties();
    defaults.setProperty(ClientConfig.OAUTH_TOKEN_ENDPOINT_URI, "https://keycloak-javacro.apps-crc.testing/auth/realms/kafka-authz/protocol/openid-connect/token"
    defaults.setProperty(ClientConfig.OAUTH_CLIENT_ID, "team-a-client");
    defaults.setProperty(ClientConfig.OAUTH_CLIENT_SECRET, "1dfk2aOz2jnj3fY6SbhW5EdviANYPd1V");
    defaults.setProperty(ClientConfig.OAUTH_USERNAME_CLAIM, "preferred_username");
    defaults.setProperty(ClientConfig.OAUTH_SSL_TRUSTSTORE_PASSWORD, "password");
    defaults.setProperty(ClientConfig.OAUTH_SSL_TRUSTSTORE_LOCATION, "C:\\Users\\mcerkez\\git\\javacro\\02_kafka_broker\\oauth_auth\\keycloak_trustsore.jks");
    defaults.setProperty(ClientConfig.OAUTH_SSL_TRUSTSTORE_TYPE, "jks");
    ConfigProperties.resolveAndExportToSystemProperties(defaults);

    // Create the Kafka producer
    producer = new KafkaProducer<String, String>(props);
```

Q&A

Resources

- Github link - https://github.com/mcerkez88/javacro2022/tree/master

CROZ

THANK YOU!

sales@croz.net | www.croz.net