Apache Kafka Security Overview

Sven Erik Knop, Senior Solutions Architect

Who am I?

- Sven Erik Knop
- Senior Solutions Architect
- Architect, Developer, Presenter
- @sven_erik_knop

Agenda

Why security?

Implementation

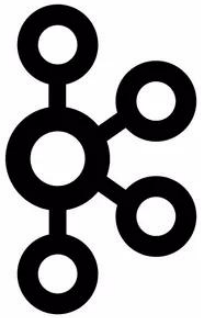
- Encryption
- Authentication
- Authorization

RBAC

- Why RBAC?
- Implementation
- Impact

Reminder: What is Apache Kafka?

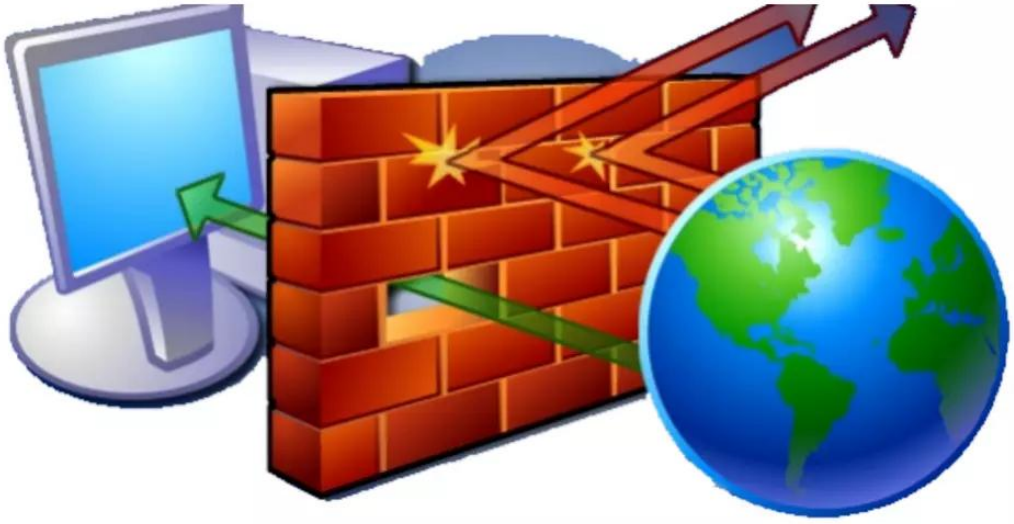
- Distributed, fault-tolerant and immutable log
- Use cases
  - Publish-subscribe queue
  - ETL (Extract, Transform, Load)
  - Event streaming platform
- Rich infrastructure through connectors and language APIs



- Open Source, with enterprise support and enhancements through companies like Confluent

Apache Kafka before security (<0.9)

- Kafka is infrastructure hidden behind the firewall
- Servers within the network are implicitly trusted
- No governance required (or possible)



New challenges

Multi-tenancy

Topics with sensitive or confidential data

Regulatory requirements

Security Features added in Kafka 0.9

Encryption

Authentication

Authorization

Connection to ZooKeeper

Encryption

Encryption in flight

TLS = Transport Layer Encryption

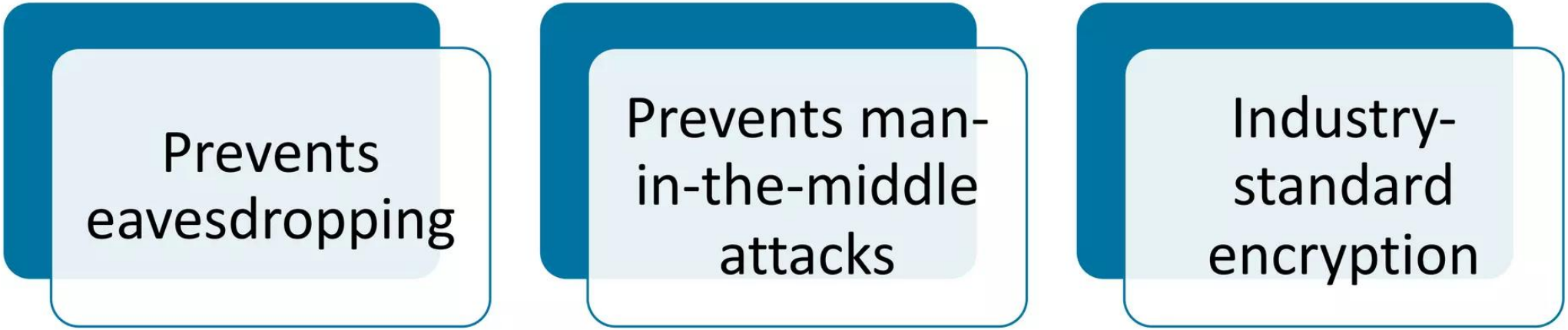
Encryption at rest

OS or file system feature

End-to-End encryption

Needs to be built into the producers and consumers

Encryption via TLS (Transport Layer Security)



Broker certificate signed by either

- External, well-known Certificate Authority (CA) – GlobalSign, Let's Encrypt …
- Internal CA, with clients using a truststore to verify its identity

security.protocol=SSL (instead of the default PLAINTEXT)
ssl.endpoint.identification.algorithm=https

TLS Mutual Authentication

If enabled, provides trust into client connections

Principal is used for authorization

Certificates signed by trusted authority (CA)

Clients additionally use a keystore to provide their own identity

SASL (Simple Authentication and Security Layer)

Framework for authentication and data security

Decouples authentication mechanisms from the application protocols

Common interface for different implementations

SASL Implementations

Implementation	Class	Comments
PLAIN	org.apache.kafka.common.security.plain.PlainLoginModule	Username/Password
SCRAM	org.apache.kafka.common.security.scram.ScramLoginModule	SCRAM_SHA_(256|512)
GSSAPI	com.sun.security.auth.module.Krb5LoginModule	Kerberos
OAUTHBEARER	org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule	Token based used for RBAC

security.protocol = SASL_PLAINTEXT | SASL_SSL
sasl.mechanism = PLAIN | SCRAM_SHA_256 | SCRAM_SHA_512 | GSSAPI | OAUTHBEARER

Configured in the brokers and the clients through a JAAS file

JAAS

- Java Authentication and Authorization Service
- For services (Kafka, ZooKeeper) grouped by *section names*
  - KafkaServer
  - KafkaClient
  - Server (for ZooKeeper Server)
  - Client (for ZooKeeper Client, e.g. Kafka)
- Alternatively, specify as a property

```
KafkaServer {
   org.apache.kafka.common.security.scram.ScramLoginModule required
   username="kafka"
   password="kafka";
};
Client {
   org.apache.zookeeper.server.auth.DigestLoginModule required
   username="admin"
   password=”admin-secret";
};
```

```
sasl.mechanism=SCRAM-SHA-256
security.protocol=SASL_PLAINTEXT
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \
  username="producer" \
  password="producer-secret";
```

sasl.mechanism=PLAIN

- Username/Password
- Broker
  - By default, holds a list of username/password in the SASL config file
  - Requires rolling restart when users are added or deleted
  - Can be overwritten, for example
    - `listener.name.sasl_plaintext.plain.sasl.server.callback.handler.class=io.confluent.security.auth.provider.ldap.LdapAuthenticateCallbackHandler`
- Client

```
sasl.mechanism=PLAIN
security.protocol=SASL_PLAINTEXT
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required \
    username="alice" \
    password="alice-secret" ;
```

SCRAM

- **Salted Challenge Response Authentication Mechanism**
- Users and passwords are stored in ZooKeeper
- Dynamic configuration through *kafka-configs*
- Different encryption levels (SHA-256, SHA-512)
- CAVEAT: Need to add any super user as an entity as well

```
kafka-configs --zookeeper zookeeper:2182 --alter
--add-config 'SCRAM-SHA-256=[password=xxxxx],SCRAM-SHA-512=[password=xxxxx]'
--entity-type users
--entity-name producer
```

```
sasl.mechanism=SCRAM-SHA-256
security.protocol=SASL_PLAINTEXT
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required \
  username="producer" \
  password=”xxxxx";
```

Kerberos

- Network authentication protocol created by the MIT
- sasl.enabled.mechanisms=GSSAPI
  - Generic Security Service Application Program Interface
- External Key Distribution Center (KDC)
- Service (Broker, ZooKeeper ...) registers itself
  - sasl.kerberos.service.name must match principal
  - Default service names are 'kafka', 'zookeeper'

Kerberos client

- Either: create a ticket with *kinit* (login)
- Or: provide the keytab location and the principal name
- List existing tickets with *klist*

```
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
    useKeyTab=true \
    storeKey=true \
    keyTab="/etc/security/keytabs/kafka_client.keytab" \
    principal="kafkaclient1@EXAMPLE.COM";
```

Authorization via ACLS (Access Control Lists)

- Permission to access or modify resources like
  - Topics
  - Clusters
  - Consumer Groups
- All, Literal and Prefix
- Allow and deny access
- Stored in ZooKeeper

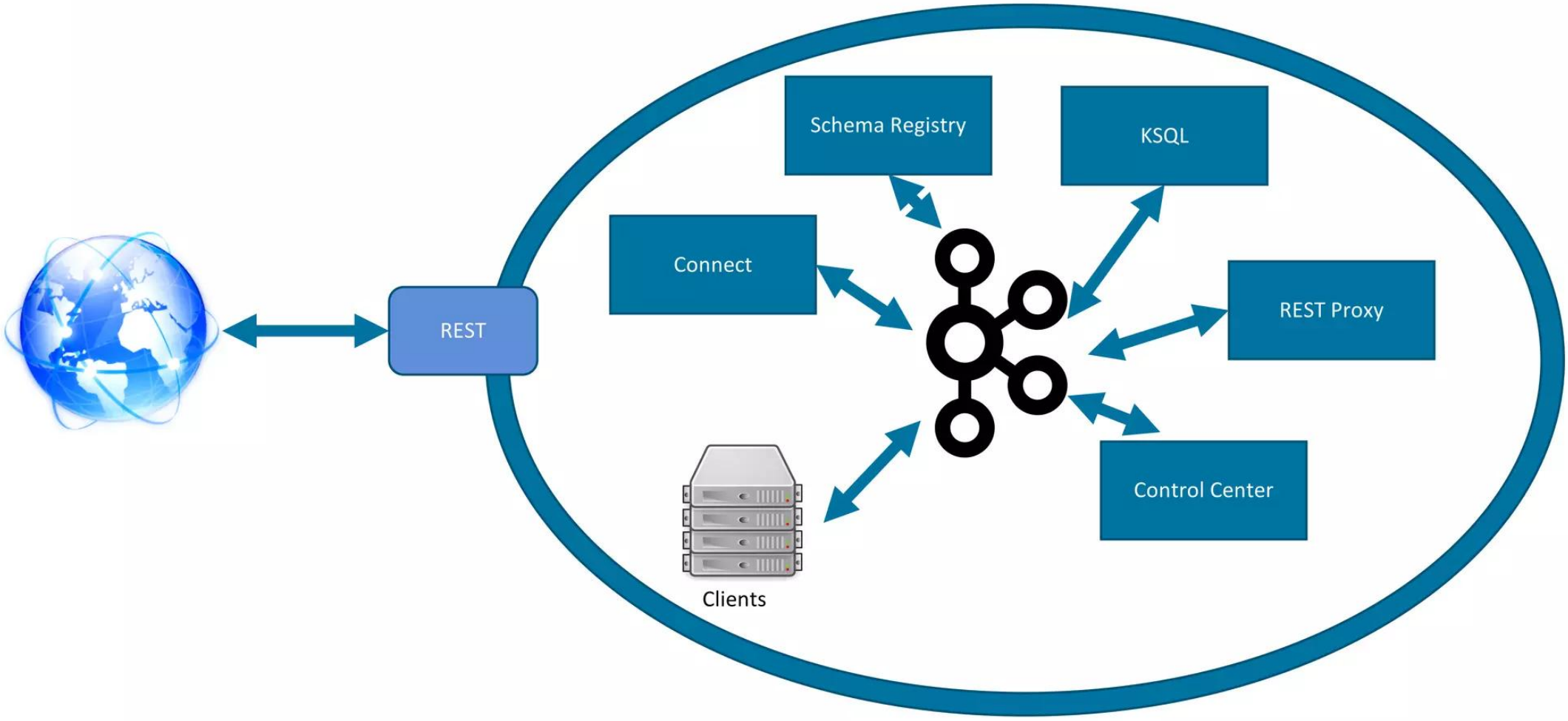
ACL for users

ACLs are stored in ZooKeeper	• kafka-acls • Can check in ZooKeeper directly
ACLs defined for	• Topic, Cluster, Group, TransactionId, DelegationToken • (Principal, PermissionType, Operation, Host)
Principal for users is User:<user-name>	• Confluent extension to use Group:<group-name> set in LDAP
Can use resource name '*' for catch all	
To specify a subset of resources with the same prefix, use	• --resource-pattern-type prefixed (default is literal)

Securing ZooKeeper

- Authentication
  - SASL
    - Digest
    - Kerberos
- Authorization
  - Securing your nodes through Kafka
  - **zookeeper**.set.**acl**=true
- Migrating from a non-secured node
  - zookeeper-security-migration

Kafka Infrastructure



Challenge

- Clients directly connected to the brokers use SASL or mutual auth for authentication
- Tools like KSQL or Connect are connected using a single user with *that* user's permissions

- ASK: Users of these tools need their own authentication and propagate their credentials

The Need for A Unified Approach

- Requirements
  - Services connect to Kafka using TLS Mutual Auth or SASL
  - External access via REST needs to impersonate an internal user
- Want
  - Single service to assign roles
  - Changes need to be traceable (ZooKeeper does not keep a log)

RBAC – Role Based Access Control

(Pre-defined) roles can be assigned to users or groups for a resource

Stored in a Kafka topic accessed via a new Kafka interface, the MDS (Meta Data Service)

Requires Confluent Server instead of Apache Kafka

User authentication backed by LDAP

Authorized users are given an OATHBEARER Token to identify themselves

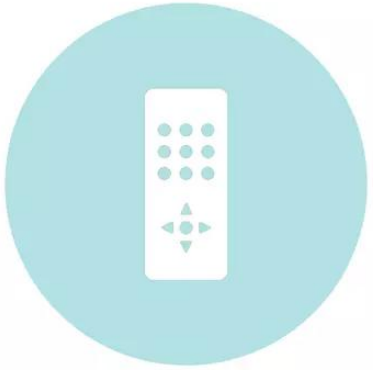
Roles

- Administration roles (SystemAdmin, ClusterAdmin, UserAdmin, …)
- Operators
- ResourceOwners
- Developers

Roles Assignment to User/Group

CLI

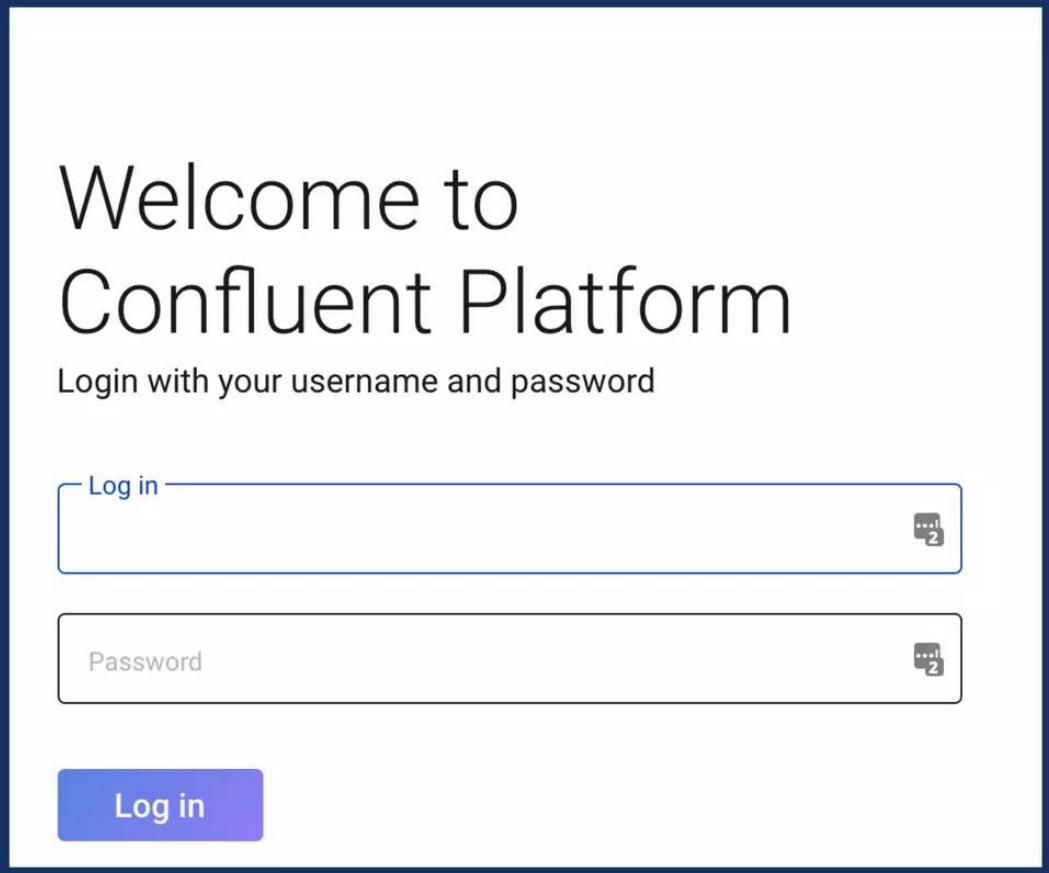
API



CONTROL CENTER

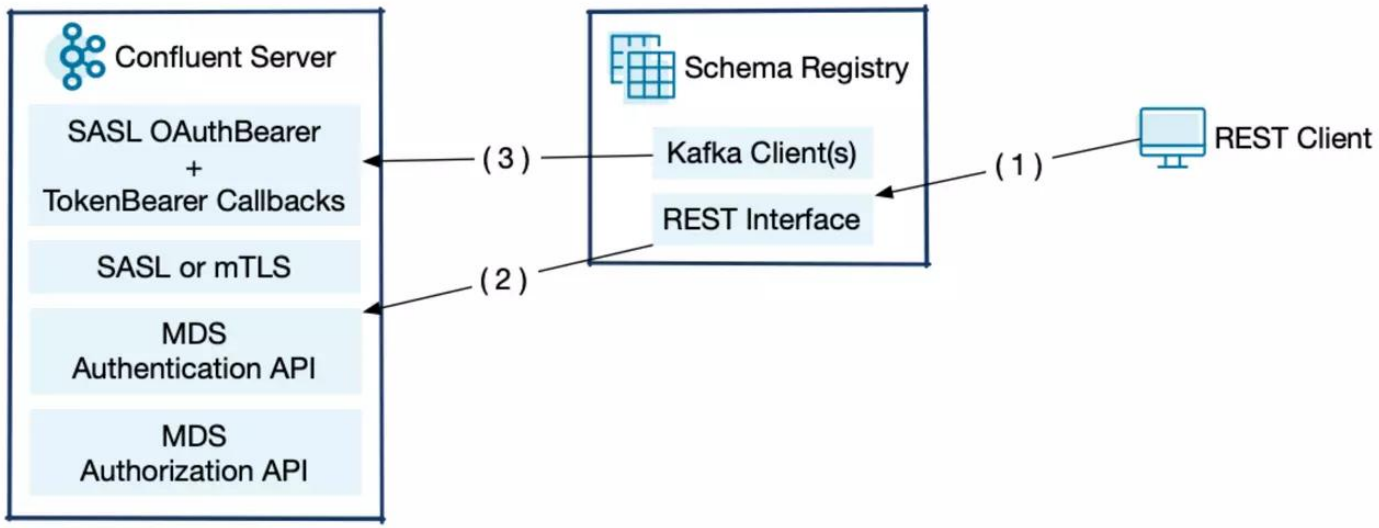
Group membership management via LDAP

Confluent Control Center

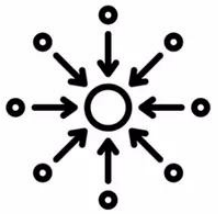


Schema Registry, Connect, KSQL, REST Proxy

- Standard REST username/password security (from LDAP)
- MDS Server provides OAUTHBEARER token
- Credentials are propagated

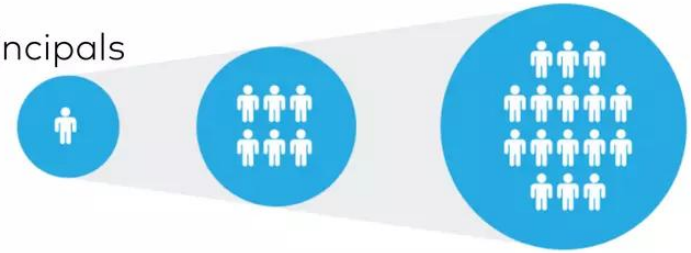


RBAC Key Benefits



Key Benefits Over Traditional ZK ACLs:

- Efficiency with Scale
  - Confluent Platform-wide security
  - Authorization at scale - with predefined Roles and LDAP Groups as Principals
  - Manage multiple clusters from 1 central location
- Increased Granular Policies
  - Connector level access control.
  - Granular access for Control Center users
  - On top of the Topic and SR Subjects
- Improved Multi-Tenancy support
  - Items without access do not show
  - Delegation of Administration Responsibilities



Requirements

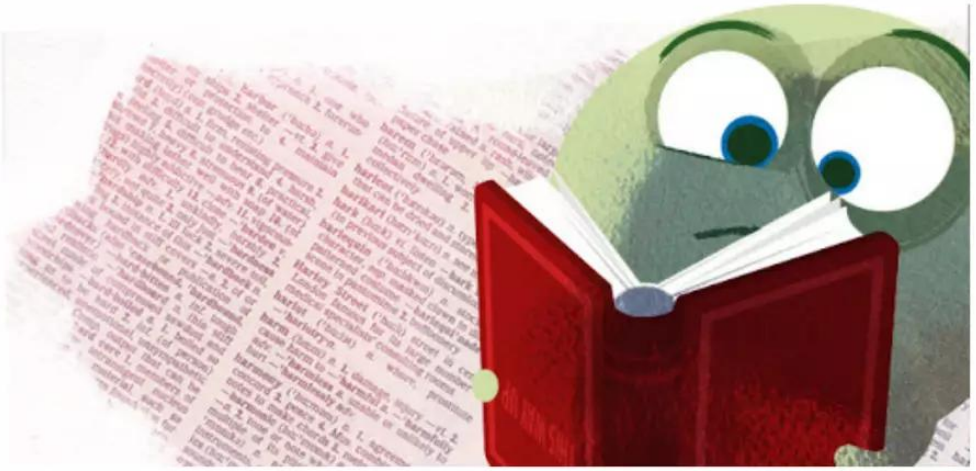
- Confluent Server >= CP 5.4
- LDAP (Usually Microsoft Active Directory) for
  - User **authentication** (usually Username/Password SIMPLE BIND)
  - Group membership for **authorization**
- All services configured with
  - sasl.enabled.mechanism=OAUTHBEARER
  - sasl.login.callback.handler.class=
    io.confluent.kafka.clients.plugins.auth.token.TokenUserLoginCallbackHandler
  - metadata.bootstrap.server.urls=http://mds-server:8090
- Do not configure standard producers/consumer/streams with OAUTHBEARER
  - Unsupported
  - Would require access to MDS server

Summary

- Security is an important feature of a streaming platform
- Encryption and ACLs provide security for clients directly accessing the Kafka brokers
- RBAC adds a unified authentication and authorization interface for services around Kafka

Further reading

- https://docs.confluent.io/current/security/security_tutorial.html
- https://docs.confluent.io/current/security/rbac/index.html



Any questions?


CONFLUENT