Enforcing fine grained Role based Authorization in multi-tenant streaming data platforms

Ashish Singh | Software Engineer, Cloudera
About Me

- Software Engineer @ Cloudera
- Contributed to Kafka, Sentry, Hive and Parquet
- Used to work in HPC
- @singhasdev
About Kafka

• Publish/Subscribe Messaging System
• High throughput (100’s of k messages/sec)
• Low latency (sub-second to low seconds)
• Fault-tolerant (Replicated and Distributed)
• Supports agnostic messaging
• Standardizes format and delivery
• Huge community
Architecture

Producers

Kafka Cluster

Consumers
Authorization
Authorization

Authorization is the function of specifying access rights to resources related to information security and computer security in general and to access control in particular. More formally, "to authorize" is to define an access policy. – Wikipedia
Authorization

Authorization is the function of specifying access rights to resources related to information security and computer security in general and to access control in particular. More formally, "to authorize" is to define an access policy. – Wikipedia
Access Policy

WHO can perform WHAT action on a RESOURCE?
Access Policy

WHO can perform WHAT action on a RESOURCE?
Access Policy

WHO can perform WHAT action on a RESOURCE?

• WHO – Requestor.
Access Policy

WHO can perform WHAT action on a RESOURCE?

- WHO – Requestor.
- RESOURCE – An entity of a system.
Access Policy

WHO can perform WHAT action on a RESOURCE?

- WHO – Requestor.
- RESOURCE – An entity of a system.
- WHAT – Action/s the Requestor can perform on the Resource.
Authorization in Apache Kafka
Access Policy

WHO can perform WHAT action on a RESOURCE?
Access Policy

WHO can or can not perform WHAT action on a RESOURCE from WHERE?
Access Policy

WHO can or can not perform WHAT action on a RESOURCE from WHERE?

• Kafka uses a binary protocol over TCP.
• The protocol defines all APIs as request response message pairs.
• Requests are sent through a Request channel after a session is established.
• Session contains Principal and Host.
  • Principal is of the form <PrincipalType>:<PrincipalName>, e.g., User:foo, Group:analyst, etc.
  • Host has info on where the request has originated from.
    • Provides IP-Filtering.
Access Policy

WHO can or can not perform WHAT action on a RESOURCE from WHERE?

- Permission Types.
  - Allow
  - Deny
Access Policy

WHO can or can not perform WHAT action on a RESOURCE from WHERE?

• Operations supported in Kafka.
  • Read
  • Write
  • Create
  • Delete
  • Alter
  • Describe
  • Cluster Action
Access Policy

WHO can or can not perform WHAT action on a RESOURCE from WHERE?

• Resource Types in Kafka.
  • Cluster
  • Topic
  • Consumer Group
APIs and their reqd. Privileges

• Read on Topic and ConsumerGroup for Fetch, JoinGroup, OffsetFetchRequest, HeartBeat and CommitOffset
• Write on Topic for Produce
• Create on Cluster for auto topic creation and KIP-4
• Delete on Topic for KIP-4
• Alter on Topic for KIP-4
• Describe on Topic and Cluster for GetOffset, GetTopicMetadata, GetConsumerMetadata, ListAllTopics(KIP-4), GetTopicInfo(KIP-4)
• Cluster Action on Cluster for LeaderAndIsr, StopReplica, UpdateMetadata and ControlledShutdown
Access Policy

- Access Policy in Kafka is represented as ACL, Access Control List.
- ACL in Kafka is composed of the following.
  - Set of Principals.
  - Permission type, i.e., Allow or Deny.
  - Set of Hosts.
  - Set of Operations.
Kafka Acls - Example

- Topic, topic1, has below Acl.

```
{
    "principals": ["user:foo", "group:devs"],
    "permissionType": "ALLOW",
    "operations": [ "READ", "WRITE" ],
    "hosts": [ "host1", "host2" ]
}
```
Kafka Acls - Example

• Topic, topic1, has below Acl.

```json
{
  "principals": ["user:foo", "group:devs"],
  "permissionType": "ALLOW",
  "operations": ["READ", "WRITE" ],
  "hosts": ["host1", "host2" ]
}
```

• Allow user “foo” and users in group “devs” to READ or WRITE to topic1 from “host1” and “host2”.

© Cloudera, Inc. All rights reserved.
Kafka Acls - Example

• Topic, topic1, has below Acl.

```
{
    "principals": ["group:admin"],
    "permissionType": "ALLOW",
    "operations": [ "ALL" ],
    "hosts": [ "*" ]
}
```

• Allow users in group “admin” to perform “ALL” actions on topic1 from any Host.
Kafka’s Default Authorizer

SimpleAclAuthorizer
SimpleAclAuthorizer

- Out of the box implementation of the Kafka Authorizer.
- Self contained and no dependencies with any other vendor or providers.
- It uses zookeeper as the storage layer for acls. ACLs are stored in JSON format described under /kafka-acls/resource-type/<resource-name>.
- Uses Caching to avoid going to ZK for each request.
- Deny takes precedence over Allow in competing ACLs.
- When no ACL is attached to a resource, use config allow.everyone.if.no.acl.found.
- When any ACL is attached to a resource only users that are in the allowed list have access. All users with no explicit allow ACLs are denied access by default.
- READ or WRITE permission => DESCRIBE Operation.
SimpleAclAuthorizer – Not so Simple

• Only supports User principal
  • PR out for Group Principal for some time.
• No way to use user group mapping from external services, like, LDAP, AD, etc.
• Very Kafka specific implementation.
• Not scalable.
  • Has zNode size limitations, default and recommended in only 1MB.
• Concurrency issues.
• Not production ready.
Sentry

- Provides unified role based authorization for various components.
  - Hive
  - Impala
  - HDFS
  - Sqoop
  - Kafka (oh yea!)
RBAC

• Role Based Authorization Control, RBAC, is a powerful mechanism to manage authorization for a large set of users and data objects in a typical enterprise.

• Assigning privileges to a user.
  • Privilege -> User
  • Privilege -> Group -> User
  • Privilege -> Role -> Group -> User

• User group mapping is configurable and can come from Shell or external systems, like, LDAP, AD, etc.
RBAC

Data Engineers

Analysts

CFO

Customer Data

Enriched Data

Sales Predictions
RBAC

- Data Engineers can READ from Customer Data.
- Data Engineers can WRITE to Enriched Data.
- Analysts can READ from Enriched Data.
- Analysts can Write to Sales Prediction.
- CFO can READ from Sales Prediction.
RBAC

• Roles:
  • ReadCustomerData => READ from Customer Data
  • WriteEnrichedData => WRITE to EnrichedData
  • ReadEnrichedData => READ from Enriched Data
  • WriteSalesPredictions => WRITE to Sales Predictions
  • ReadSalesPredictions => READ from Sales Predictions
  • AllRead => READ from Customer, Enriched and Sales Predictions

• Roles to Groups:
  • DataEngineers => ReadCustomerData, ReadEnrichedData, WriteEnrichedData
  • Analysts => ReadEnrichedData, ReadSalesPredictions, WriteSalesPredictions
  • CFO => ReadSalesPredictions
RBAC

• New Roles:
  • ReadCustomerRetention => Read from Customer Retention
  • WriteCustomerRetention => Write to Customer Retention

• Add new roles to groups:
  • Analysts => + ReadCustomerRetention, WriteCustomerRetention
  • CFO => + ReadCustomerRetention
RBAC

- Joe just joined as Data Engineers
  - Just add Joe to DataEngineers group
  - Joe gets all DataEngineers privileges
RBAC

- New team added to maintain data lineage.
  - Just add roles to it, no need redefine privileges.
  - Auditors => ReadAll

Data Engineers

Analysts

CFO

Auditors

Customer Data

Enriched Data

Sales Predictions
SentryKafkaAuthorizer

- Brings role based authorization control to Kafka.
- Use user group mappings from external systems, like, LDAP, AD, etc.
- Scalable architecture.
- Unified authorization control across various data infrastructure components.
SentryKafkaAuthorizer

1. Client authenticates with Broker.
SentryKafkaAuthorizer

1. Client authenticates with Broker.
2. Client sends request to a Broker.
SentryKafkaAuthorizer

1. Client authenticates with Broker.
2. Client sends request to a Broker.
3. Broker authorizes the request with Sentry.
SentryKafkaAuthorizer

1. Client authenticates with Broker.
2. Client sends request to a Broker.
3. Broker authorizes the request with Sentry.
4. Sentry responds with Allow or Deny for the req.
SentryKafkaAuthorizer

1. Client authenticates with Broker.
2. Client sends request to a Broker.
5. Broker sends Not authorized error code if the request is not authorized. Otherwise sends appropriate response for the request.

3. Broker authorizes the request with Sentry.
4. Sentry responds with Allow or Deny for the req.
Demo
Thank you

Ashish Singh
asingh@cloudera.com
@singhasdev