June 10th 2018

Kafka Security

A brief overview of its history, current state and how it can be customized

Sönke Liebau – Co-Founder and Partner @ OpenCore

Who am I?

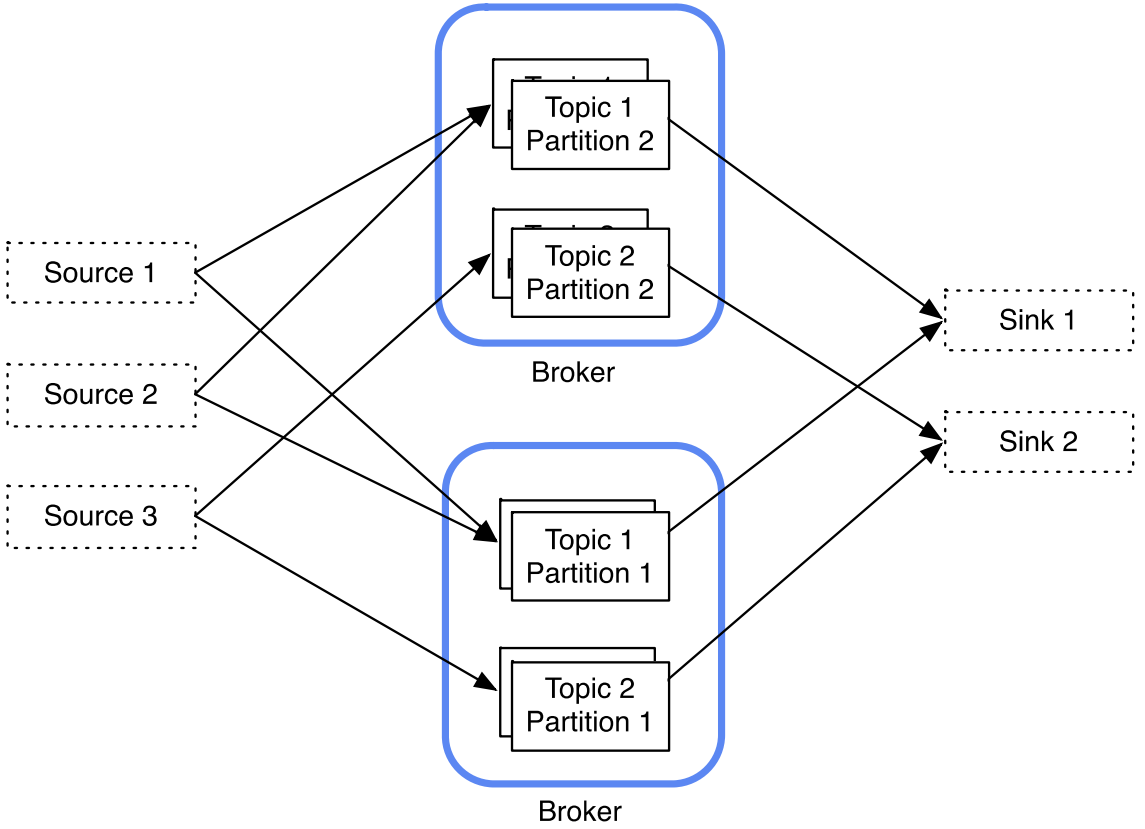
- Partner & Co-Founder at OpenCore
  - Small consulting company with a Big Data & Open Source focus
- Specialized in Kafka & Elasticsearch

- Contact
  - soenke.liebau@opencore.com
  - @soenkeliebau

What is Kafka?

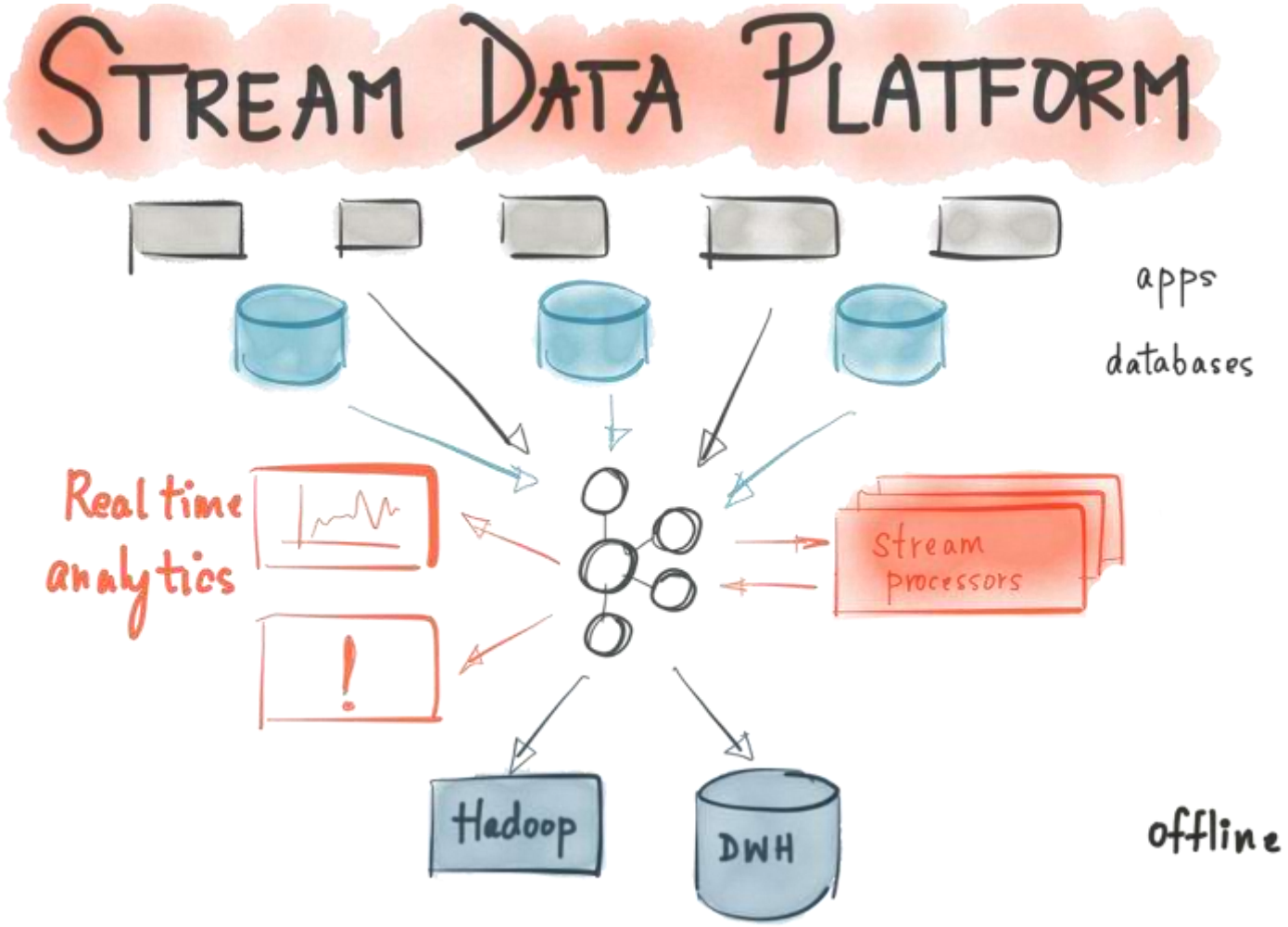
- Kafka is a distributed, topic-oriented, partitioned, **replicated** commit log

- Kafka is also a **publish-subscribe** messaging system



OpenCore

Where does Kafka fit in your architecture?



OpenCore

Source: https://www.confluent.io/blog/announcing-kafka-connect-building-large-scale-low-latency-data-pipelines

Security - The Past

Security – What happened since then

Security – What happened since then

Kafka version	Authentication method	Jira	KIP
0.9.0.0	SSL	KAFKA-1690	KIP 12
0.9.0.0	SASL GSSAPI (Kerberos)	KAFKA-1686	KIP 12
0.10.0.0	SASL Plain	KAFKA-3149	KIP 43
0.10.2.0	SASL SCRAM	KAFKA-3751	KIP 84
1.1.0	Delegation Tokens	KAFKA-1696	KIP 48

Terms & Acronyms

SASL - Simple Authentication and Security Layer

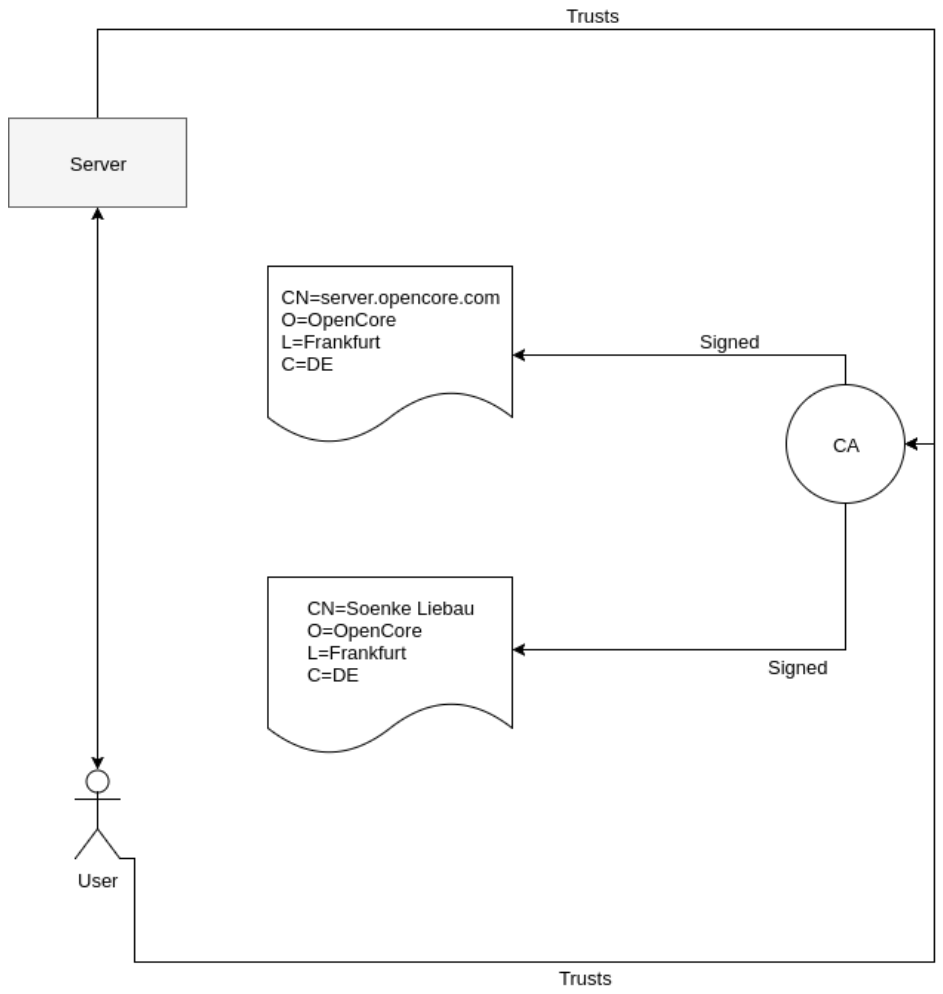
JAAS - Java Authentication and Authorization Service

GSSAPI - Generic Security Services Application Program Interface

SCRAM – Salted Challenge Response Authentication Method

Authentication (1) – SSL

- Certificates are signed by a trusted authority
- Client checks the servers certificate against CAs they trust and verifies information from certificate
- Client presents a certificate to the server which checks it agains known CAs
- Information from Client Cert is made available for authentication



Authentication (2) – SASL_Plain

```
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret"
   user_kafkabroker1="kafkabroker1-secret";
};
```

- Username & Password Authentication
- Users by default are stored directly in the JAAS file
  - Custom implementations possible to retrieve credentials externally
- Unless combined with TLS this will transmit the password in cleartext!

Authentication (3) – SASL_SCRAM

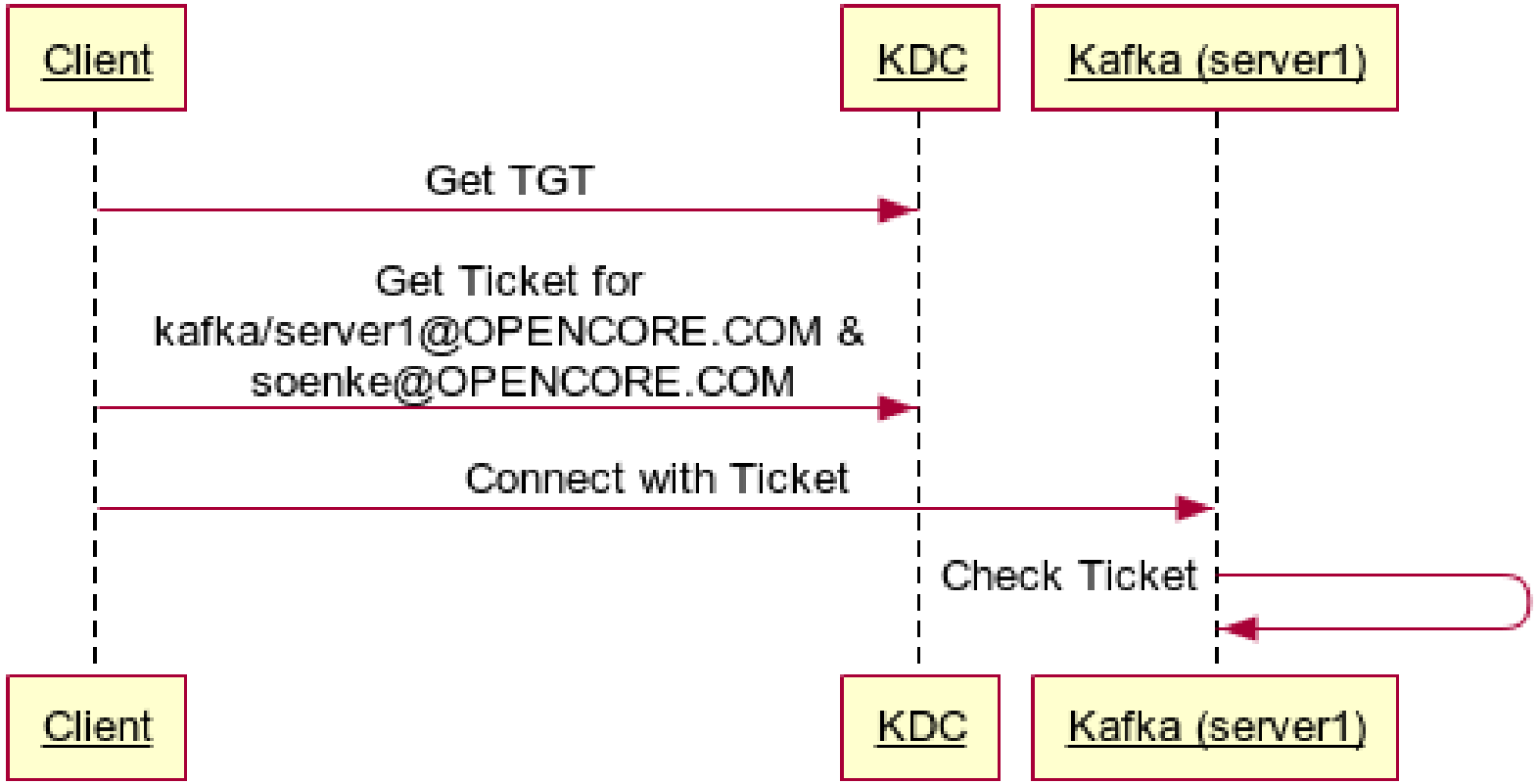
```
KafkaServer {
   org.apache.kafka.common.security.scram.ScramLoginModule required
   username="admin"
   password="admin-secret";
};
```

- No passwords need to be transmitted
- Allows binding to TLS for added security

Authentication (4) – SASL_GSSAPI

- Generic Security Services Application Program Interface
  - Kerberos is the main implementation in use today
- Useful for integration with Microsoft AD or similar directory services
- Authentication is based on Tickets and Principals
  - User Principals Name (UPN) – soenke@OPENCORE.COM
  - Service Principal Name (SPN) – kafka/server1@OPENCORE.COM
- Inital authentication via password or keytab to retrieve a ticket granting ticket
  - TGT is then used to retrieve service tickets
  - TGT expires

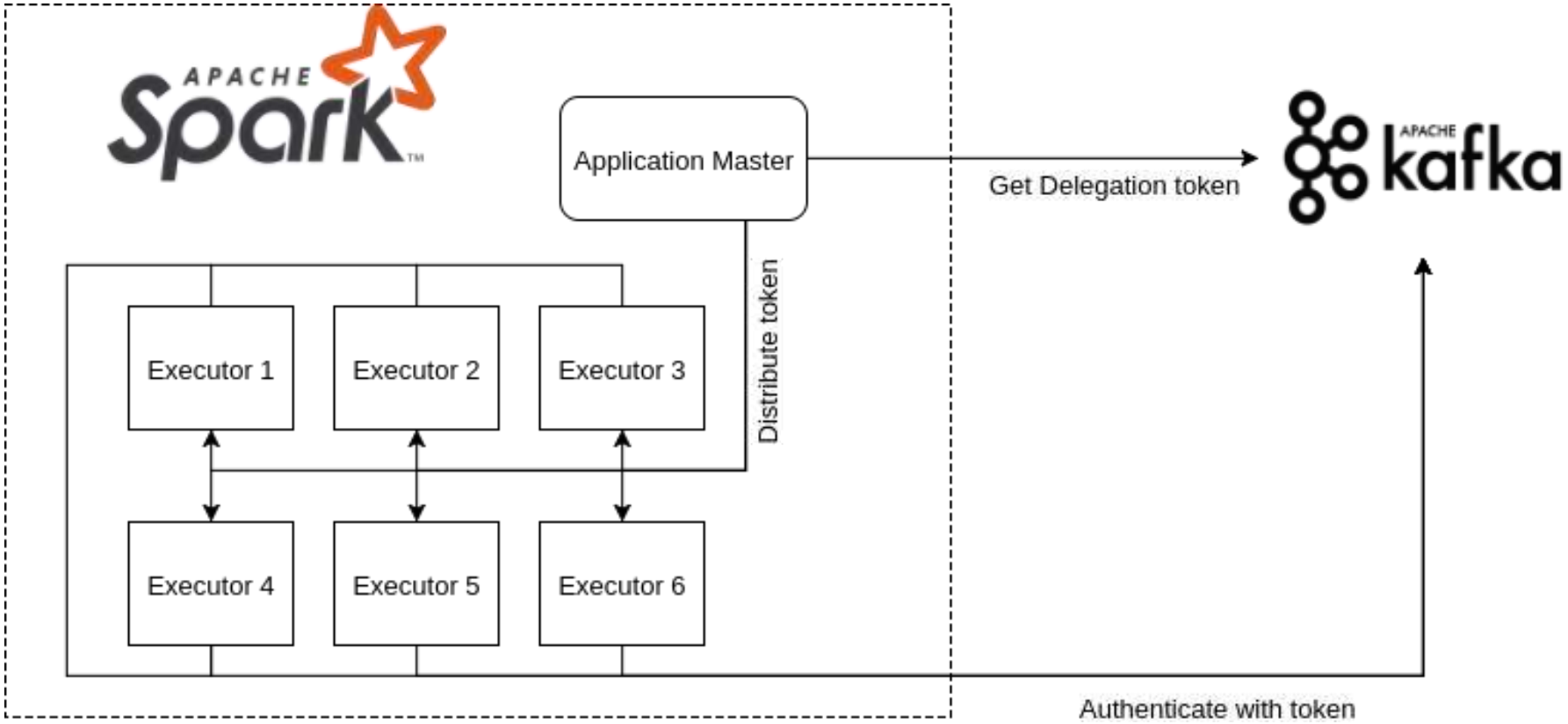
Authentication (5) – SASL_GSSAPI



Authentication (6) – Delegation tokens

- Invention from the Hadoop world
- New in Kafka 1.1.0
- Allow the user to obtain a token from Kafka
  - Use this token to authenticate the user
  - Only valid for Kafka
  - Only valid for a limited amount of time
  - Secondary form of authentication, cannot be used to obtain a new token after it has expired
- Often used in distributed long running jobs (i.e. Spark Streaming)

Authentication (7) – Delegation Tokens



Broker Configuration

- Brokers can support multiple methods of authentication at the same time by defining *listeners*
- Listeners
  - SASL_PLAIN
  - PLAINTEXT
  - SSL_SASL_PLAIN
  - …
- A distinction can be made between internal and external traffic
  - NAT situations used to be complicated (not possible)
  - Since version 0.10.2.0 a clean separation can be defined

Authorization

- To enable access control beyond a simple access/no access scenario authorization is needed
- Kafka introduced ACLs for this in 0.9.0.0 and added the SimpleAclAuthorizer class
  - Based on ACLs that are stored in Zookeeper
  - ACLs are based on resources
  - Superusers can be defined that are allowed anything
- Entire authentication mechanism is pluggable

ACLs

- ACLs grant rights per resource
  - Topic , Consumer Group, Cluster
  - No wildcards beyond „*“ – Yet!
- Available actions:
  - Create, Read, Write, Delete, Describe, ClusterAction
- IP addresses can be used to limit users
  - No wildcards beyond „*“ or ranges
- Allow or Deny
  - Deny takes precedence
- Default no ACL matches

ACL CLI examples (1)

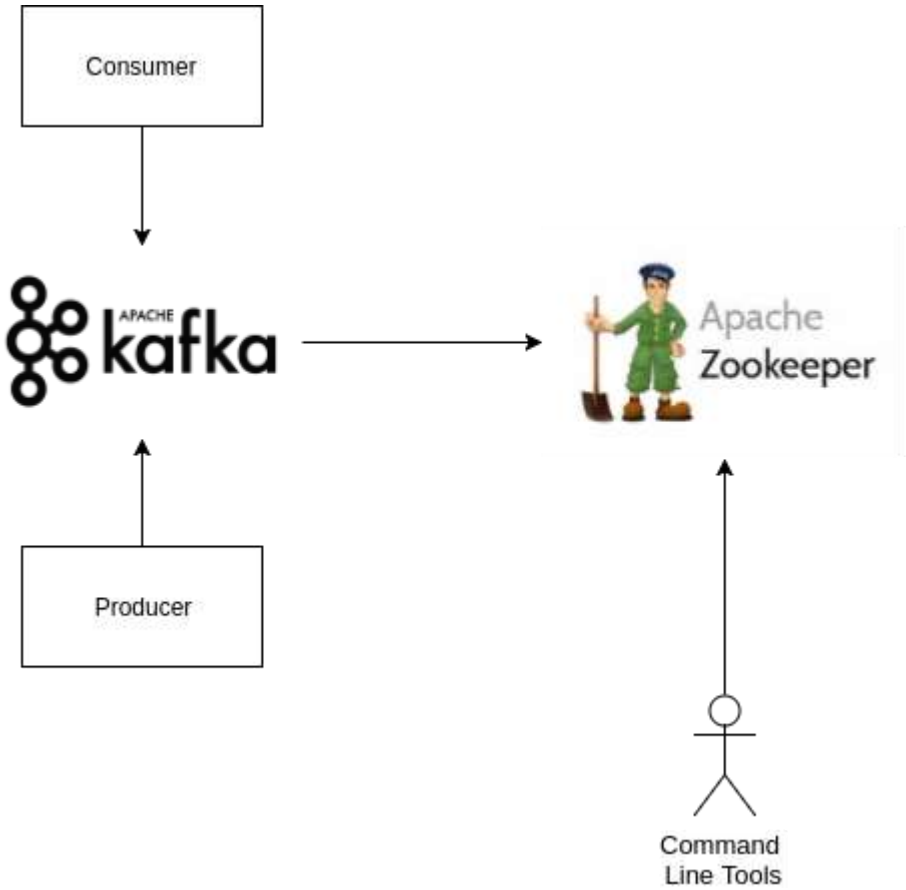
```
→  bin ./kafka-acls.sh \
  --authorizer-properties zookeeper.connect=localhost:2181 \
  --add \
  --allow-principal User:Bob \
  --allow-principal User:Alice \
  --allow-host 198.51.100.0 \
  --allow-host 198.51.100.1 \
  --operation Read \
  --operation Write \
  --topic Test-topic

Adding ACLs for resource `Topic:Test-topic`:
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.1

Current ACLs for resource `Topic:Test-topic`:
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.1

→  bin
```

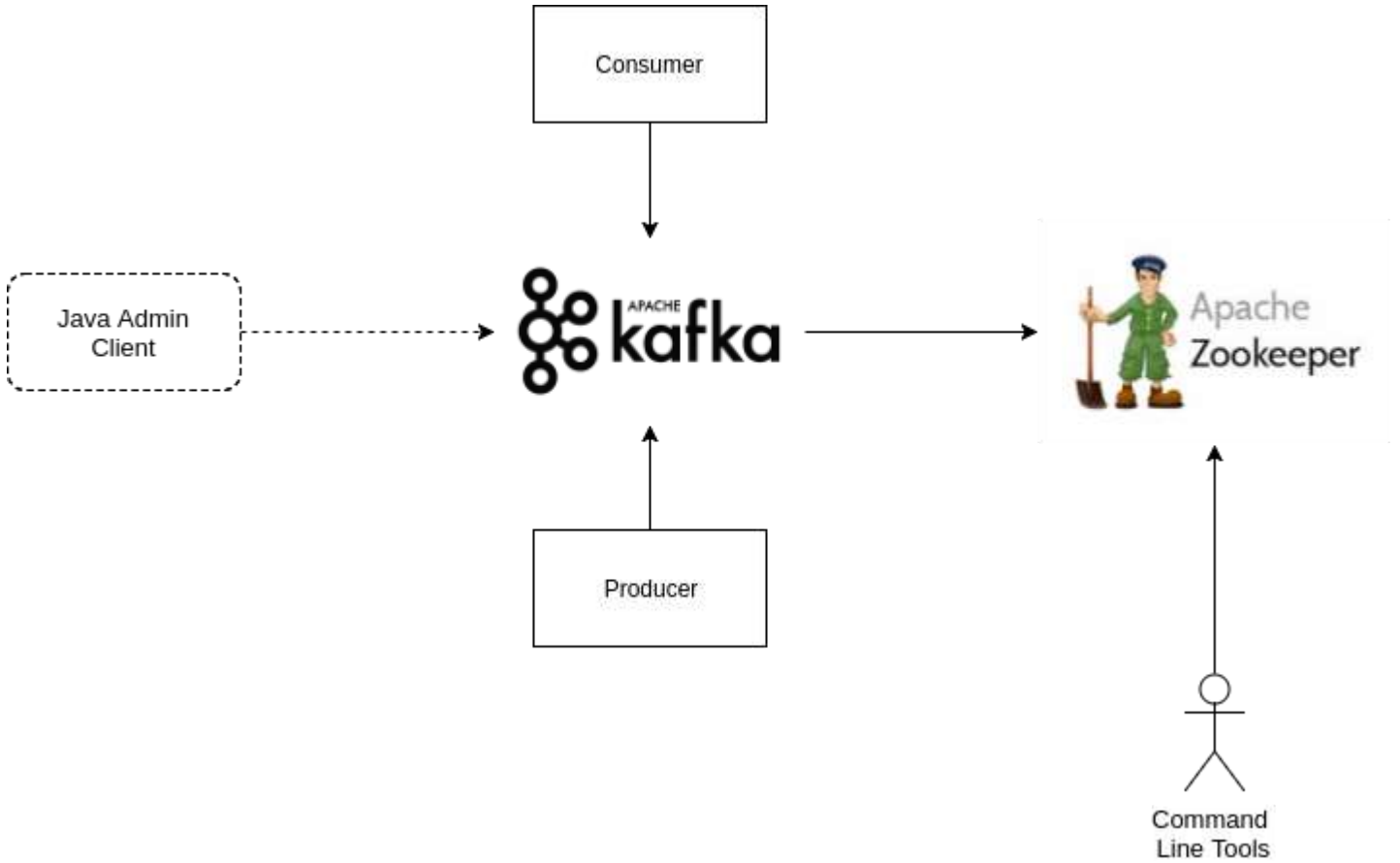
ACL CLI examples (2)

```
→  bin ./kafka-acls.sh \
  --authorizer-properties zookeeper.connect=localhost:2181 \
  --remove \
  --allow-principal User:Alice \
  --operation Read \
  --operation Write \
  --topic Test-topic

Are you sure you want to remove ACLs:
        User:Alice has Allow permission for operations: Write from hosts: *
        User:Alice has Allow permission for operations: Read from hosts: *
 from resource `Topic:Test-topic`? (y/n)
y
Current ACLs for resource `Topic:Test-topic`:
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.1

→  bin
```

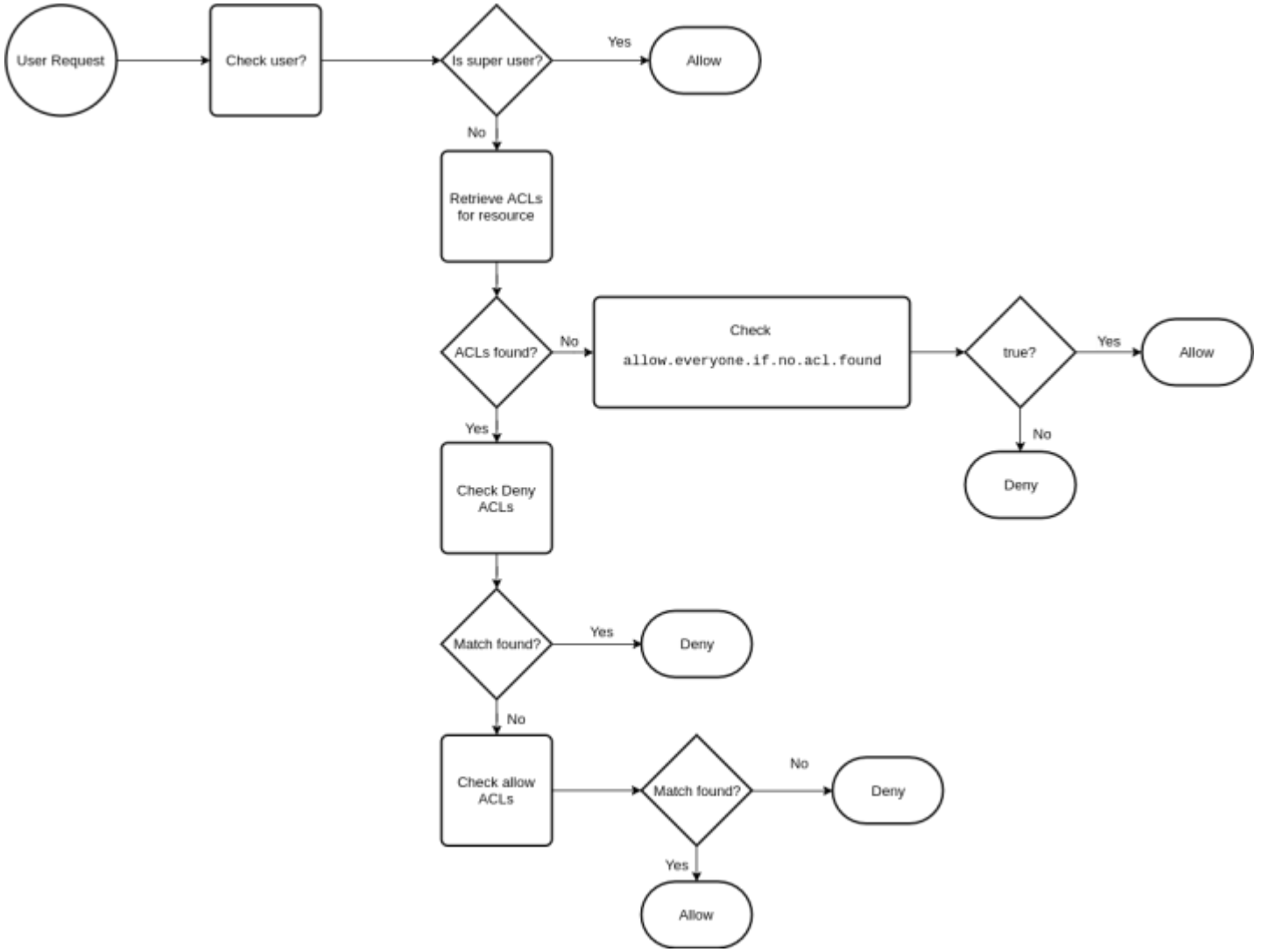
ACL CLI examples (3)

```
→  bin ./kafka-acls.sh \
  --authorizer-properties zookeeper.connect=localhost:2181 \
  --remove \
  --allow-principal User:Alice \
  --allow-host 198.51.100.0 \
  --allow-host 198.51.100.1  --operation Read \
  --operation Write \
  --topic Test-topic

Are you sure you want to remove ACLs:
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Alice has Allow permission for operations: Write from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.1
        User:Alice has Allow permission for operations: Read from hosts: 198.51.100.0
 from resource `Topic:Test-topic`? (y/n)
y
Current ACLs for resource `Topic:Test-topic`:
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.1
        User:Bob has Allow permission for operations: Read from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.0
        User:Bob has Allow permission for operations: Write from hosts: 198.51.100.1
```

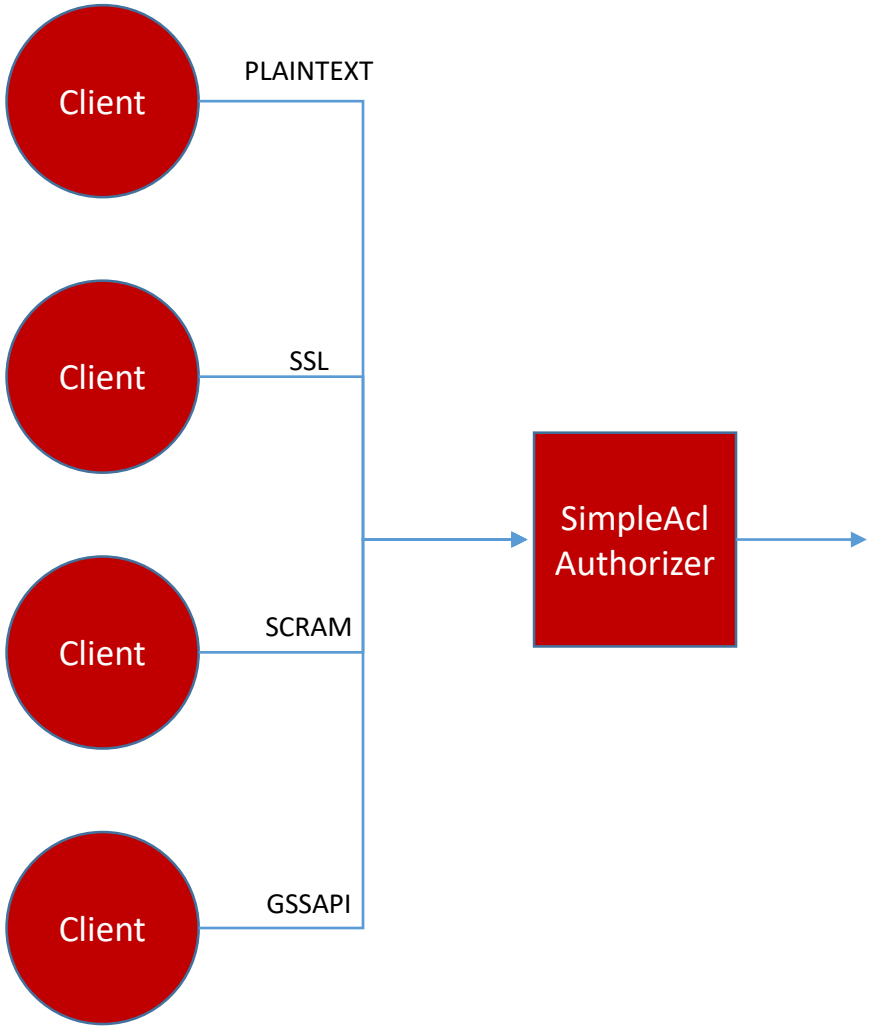




Authorization Sequence

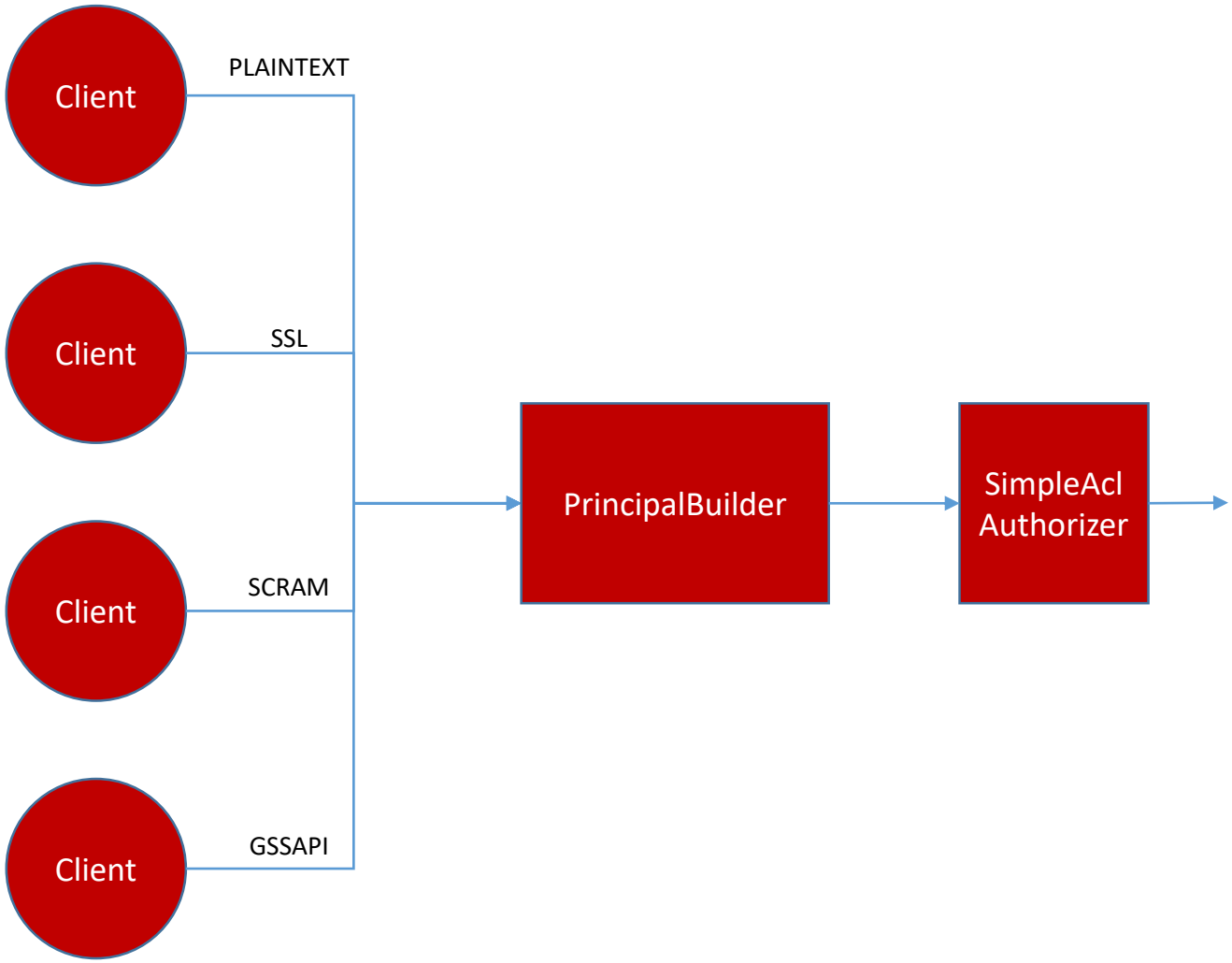


Authorization




```
./kafka-acls
  --authorizer-properties zookeeper.connect=localhost:2181
  --add
  --allow-principal ???
  --operation All
  --topic '*'
  --group '*'
```

Principals

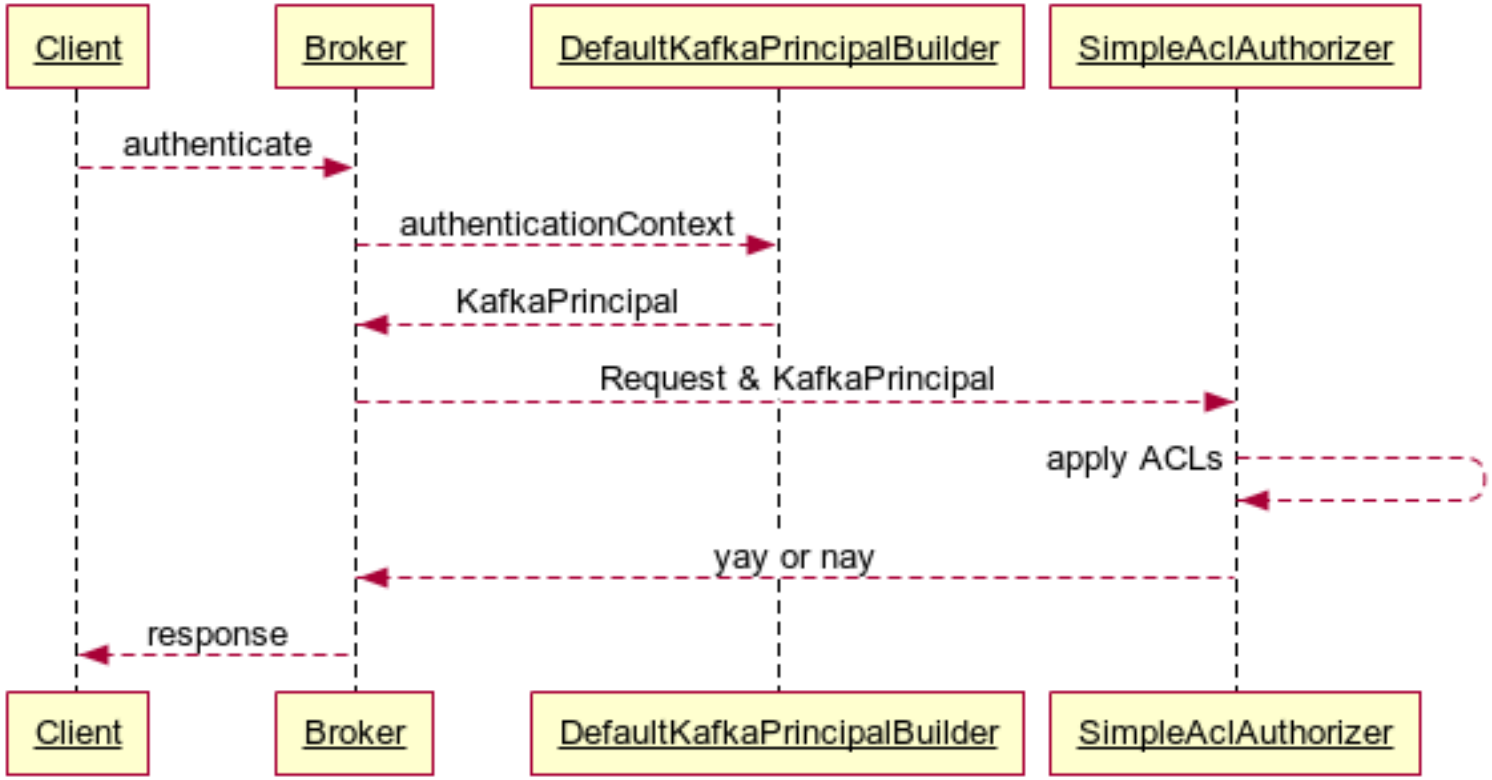



```
./kafka-acls
  --authorizer-properties zookeeper.connect=localhost:2181
  --add
  --allow-principal Principal
  --operation All
  --topic '*'
  --group '*'
```

Principal From Authentication

Authentication Method	Principal
PLAINTEXT	ANONYMOUS
SSL (without client authentication)	ANONYMOUS
SASL_PLAIN	Username
SASL_SCRAM	Username
SASL_GSSAPI	Kerberos UPN (sliebau@OPENCORE.COM) Configurable via auth-to-local rules in config.
SSL (with client authentication)	Built from Certificate details: CN=writeuser,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown

Authorization Sequence



Extending Kafka Authorization

- PrincipalBuilder and Authorizer are both configurable
  - principal.builder.class
  - authorizer.class.name
- Two projects have implementations available
  - Ranger
  - Sentry

BYO - Objective

Enable definition of ACLs based on Active Directory groups instead of individual users

BYO - Tasks

- Create a PrincipalBuilder that retrieves groups from AD for a username
- Create a Principal that can store group information
- Create an Authorizer that understands the stored groups and applies ACLs
- Enable the user to create and manage ACLs

Extended Principal To Store Group Information

```java
public class ComplexKafkaPrincipal extends KafkaPrincipal{

  protected List<KafkaPrincipal> additionalPrincipals = new ArrayList<>();
  public ComplexKafkaPrincipal(String principalType, String name) {
      super(principalType, name);
  }

  public ComplexKafkaPrincipal(KafkaPrincipal kafkaPrincipal) {
      this(kafkaPrincipal.getPrincipalType(), kafkaPrincipal.getName());
  }

  public ComplexKafkaPrincipal(String principalType, String name, List<KafkaPrincipal> additionalPrincipals) {
      this(principalType, name);
      this.additionalPrincipals = additionalPrincipals;
  }

  public List<KafkaPrincipal> getGroupMemberships() {
      return additionalPrincipals;
  }
}
```

BYO – Group lookup

```
public class HadoopGroupMappingPrincipalBuilder implements KafkaPrincipalBuilder, Configurable {
  private GroupMappingServiceProvider groupMapper;
  private DefaultKafkaPrincipalBuilder principalBuilder;

  @Override
  public KafkaPrincipal build(AuthenticationContext context) {
    // Create a base principal by using the DefaultPrincipalBuilder
    ComplexKafkaPrincipal basePrincipal = new ComplexKafkaPrincipal(principalBuilder.build(context));

    // Resolve username based on what kind of AuthenticationContext the request has
    // and perform groups lookup
    if (context instanceof SaslAuthenticationContext) {
      basePrincipal.additionalPrincipals = getGroups(basePrincipal.getName());
    } else if (context instanceof SslAuthenticationContext) {
      basePrincipal.additionalPrincipals = getGroups(getUserFromCertificate(basePrincipal.getName()));
    }
    return basePrincipal;
  }

  private List<KafkaPrincipal> getGroups(String userName) {
    List<KafkaPrincipal> groupPrincipals = new ArrayList<>();
    try {
      // Add user principal to list as well to make later matching easier
      groupPrincipals.add(new KafkaPrincipal(KafkaPrincipal.USER_TYPE, userName));

      principalLogger.fine("Resolving groups for user: " + userName);
      List<String> groups = groupMapper.getGroups(userName);
      principalLogger.fine("Got list of groups for user " + userName + ": " + Utils.join(groups, ", "));
      for (String group : groups) {
        groupPrincipals.add(new KafkaPrincipal("Group", group));
      }
    } catch (IOException e) {
      principalLogger.warning("Groups for user " + userName +
          " could not be resolved, proceeding with authorization based on username only.");
    }
    return groupPrincipals;
  }

}
```

- Steal this part from Hadoop
  - GroupMappingServiceProvider
- Checks against local groups
  - Manifest with SSSD, Centrify, …
- Other implementations available

ACL matching

```
private def aclMatch(operations: Operation, resource: Resource, principal: KafkaPrincipal,
                     host: String, permissionType: PermissionType, acls: Set[Acl]): Boolean = {
  // Build a list of all Principals for this ComplexPrincipal
  var allPrincipals = List[KafkaPrincipal]()

  if (principal.isInstanceOf[ComplexKafkaPrincipal]) {
    // For
    allPrincipals = principal.asInstanceOf[ComplexKafkaPrincipal].getGroupMemberships.asInstanceOf[List[KafkaPrincipal]]
  } else {
    // A KafkaPrincipal was passed
    allPrincipals ::= new KafkaPrincipal(principal.getPrincipalType, principal.getName)
  }

  // Match principals against ACLs
  allPrincipals
    .map(p => singleAclMatch(operations, resource, p, host, permissionType, acls))
    .foldLeft(false)(_ || _)
}
```

BYO – Putting It To The Test

```
# kafka-console-producer.sh --topic test --...
[2018-02-20 17:35:54,999] DEBUG Principal = User:sliebau@OPENCORE.COM is Allowed Operation = Describe from
host = 10.0.0.9 on resource = Topic:test (kafka.authorizer.logger)
[2018-02-20 17:35:56,213] DEBUG operation = Write on resource = Topic:test from
host = 10.0.0.9 is Allow based on acl = User:sliebau@OPENCORE.COM has Allow permission for operations: Write from
hosts: * (kafka.authorizer.logger)
[2018-02-20 17:35:56,213] DEBUG Principal = User:sliebau@OPENCORE.COM is Allowed Operation = Write from
host = 10.0.0.9 on resource = Topic:test (kafka.authorizer.logger)

# kafka-console-producer.sh --topic test2 --...
[2018-02-20 17:36:11,388] DEBUG Principal = User:sliebau@OPENCORE.COM is Allowed Operation = Describe from
host = 10.0.0.9 on resource = Topic:test2 (kafka.authorizer.logger)
[2018-02-20 17:36:12,457] DEBUG operation = Write on resource = Topic:test2 from
host = 10.0.0.9 is Allow based on acl = Group:supergroup has Allow permission for operations: Write from
hosts: * (kafka.authorizer.logger)
[2018-02-20 17:36:12,457] DEBUG Principal = User:sliebau@OPENCORE.COM is Allowed Operation = Write from
host = 10.0.0.9 on resource = Topic:test2 (kafka.authorizer.logger)
```

BYO – Code Along

For more details on the custom authorizer presented here as well as the full code used please visit:

https://www.opencore.com/blog/2018/3/2018-group-based-authorization-in-kafka

Thank You!

@soenkeliebau