0. Objetivo

Se desea conseguir un entorno capaz de permitir/denegar el acceso a **recursos** a clientes externos (*Consumers* y *Producers*, principalmente). Estos recursos son los **topics, brokers** y **nodos** del **zookeeper**.

Para ello, es necesario:

1. **Autenticar** a los clientes y a los brokers (SASL).
2. Definir **mecanismos de autorización** sobre los principales autenticados.

1. Autenticación SASL

Kafka utiliza ***JAAS*** (*Java Authentication and Authorization Service*) como mecanismo para autenticar (*y autorizar*) a brokers y clientes.
Actualmente existen tres modos para autenticar brokers y clientes en Kafka:

- ✓ **Kerberos**
  Utiliza el mecanismo GSSAPI de SASL para autenticación. Soportado por Ambari, el cual automatiza el proceso de generación de principales y sus llaves.

- ✓ **PLAIN**
  El mecanismo PLAIN se considera el más simple y menos seguro (recomendado utilizar junto a TLS en entornos de producción expuestos). Se basa en un simple mecanismo de usuario-contraseña.

- ✓ **SCRAM**
  El mecanismo SCRAM se introduce en Kafka a partir de la versión 0.11. Basado en usuario-contraseña al igual que PLAIN, pero incluyendo ciertos mecanismos de seguridad para evitar ataques *Man In the Middle*.
  Todas las contraseñas se almacenan en el *zookeeper*, por lo que es necesario que éste se encuentre bien protegido. El funcionamiento es el siguiente:

  1) **Producer** desea acceso a **Broker.**
  2) **Broker** quiere comprobar que **Producer** es quien es. Le pregunta su contraseña, pero no quiere que se le envíe por la red: le pide a **Producer** que realice una serie de operaciones sobre la misma y le envíe el **hash** resultante.
  3) **Broker** recibe el hash y comprueba que **Producer** es quien dice ser. Ahora ya autenticado, Producer puede comenzar a escribir en los Topics en los que esté autorizado.

Independientemente del mecanismo escogido, los requisitos para implementar autenticación en un entorno Kafka son los mismos. Los principales afectados son el **broker**, los **clientes** (producers/consumers) y el **zookeeper**.

A continuación se detallan los requisitos para autenticar a los actores principales de un cluster Kafka:

1.1 - Brokers

- ***JAAS***

  El fichero JAAS del Broker debe incluir dos apartados:

  1. *KafkaServer*: Sección que indica a los brokers el principal de Kerberos que deben utilizar, así como la ruta hasta su keytab. Permite autenticar al broker en el cluster, y es utilizada también para la intercomunicación entre brokers.

  2. *Client*: Permite la conexión al zookeeper por parte de los brokers. *¡No confundir con cliente externo(producer/consumer)!*

- ***Server.properties***

  El fichero de configuración del broker se debe modificar para especificar que se va a hacer uso de autenticación SASL. En este caso, el ejemplo muestra los cambios de configuración para un entorno ***Kerberos***.

```
listeners = SASL_PLAINTEXT://host:Puerto
security.inter.broker.protocol = SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol = GSSAPI
sasl.enabled.mechanisms = GSSAPI
sasl.kerberos.service.name = kafka
authorizer.class.name = kafka.security.auth.SimpleAclAuthorizer
principal.to.local.class = kafka.security.auth.KerberosPrincipalToLocal
```

  El protocolo utilizado es *SASL_**PLAINTEXT***: La autenticación se realiza mediante SASL y el envío de datos no está cifrado.

  En el caso en el que se requiera cifrar las comunicaciones en el cluster Kafka, se podría hacer uso del protocolo *SASL*_**SSL**: El modo de autenticación es el mismo, pero el envío se realiza por TLS.

- ***JVM***

  La máquina virtual de Java debe conocer la ubicación del fichero JAAS para poder autenticar al **broker**. En el caso de una instalación mediante Ambari, este paso no sería necesario ya que automatiza la generación del entorno del JVM.

  Para definir la localización del fichero se aprovecha la variable de entorno `KAFKA_OPTS`, reconocida por Kafka en el arranque.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=…/config/kafka_servert_jaas.conf"
```

¡! *Si se realiza un arranque manual, y para evitar necesitar ejecutar este comando cada vez, se recomienda definir el valor del parámetro en el script Kafka-run-class.sh, utilizado en el arranque de los brokers.*

Todos los brokers de un cluster Kafka SASL deben:

1. Compartir el **mismo nombre de principal Kafka autorizado**.
2. Disponer de su propio **fichero JAAS**.
3. Disponer del **keytab** del principal en la ruta especificada por el JAAS.
4. El **usuario** que ejecute los servidores debe tener **permisos de lectura** para los ficheros *JAAS* y *keytab*.
5. El **server.properties** debe indicar el uso de **SASL** para las comunicaciones, así como el **nombre del servicio** referenciado (Kafka).
6. Ejecutarse en un **JVM** que conozca la **ubicación del JAAS** servidor.

1.2 - Clientes externos (Producers/Consumers)

- ***JAAS***

El fichero JAAS del Cliente debe incluir:

1. *KafkaClient*: Permite la conexión de los clientes con el broker SASL. Autentica al Producer/Consumer, especificando el principal a utilizar y su keytab.

- ***Consumer/Producer.properties***

Los siguientes parámetros son necesarios para la correcta implementación del protocolo SASL en clientes (ejemplo basado en ***Kerberos***):

```
Security_protocol = SASL_PLAINTEXT
Sasl.mechanism = GSSAPI
```

- ***JVM***

Al igual que con el broker, es necesario definir la ubicación del fichero ***JAAS*** a la JVM.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=…/config/kafka_client_jaas.conf”
```

Todos los clientes de un Kafka SASL deben:

1. Autenticarse mediante un **principal autorizado**.
2. Tener su propio fichero **JAAS**.
3. Tener el fichero **keytab** del principal en la ruta especificada por el JAAS.
4. Ejecutarse en un **JVM** que conozca la **ubicación del JAAS** cliente.

1.3 - Zookeeper

- ***JAAS***

  El fichero JAAS del **servidor Zookeeper** debe incluir:

  1. *Server*: Permite autenticar a zookeeper en el cluster, indicando el principal y el keytab del servicio.

  El fichero JAAS del **cliente Zookeeper** debe incluir:

  1. *Client*: Sección simple que únicamente indica que se requiere el uso del protocolo SASL para autenticación. En el caso en el que el cliente se conecte al zookeeper sin un principal autenticado, éste mostrará un mensaje de warning indicando que se intentará el acceso al zNode de manera anónima; Serán entonces los permisos del zNode los que permitan/denieguen el acceso a los recursos del Zookeeper.

- ***Zoo.cfg***

  Los siguientes parámetros son necesarios para la correcta implementación del protocolo SASL en el **servidor Zookeeper**:

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
(requireClientAuthScheme=sasl)*
jaasLoginRenew=3600000
```

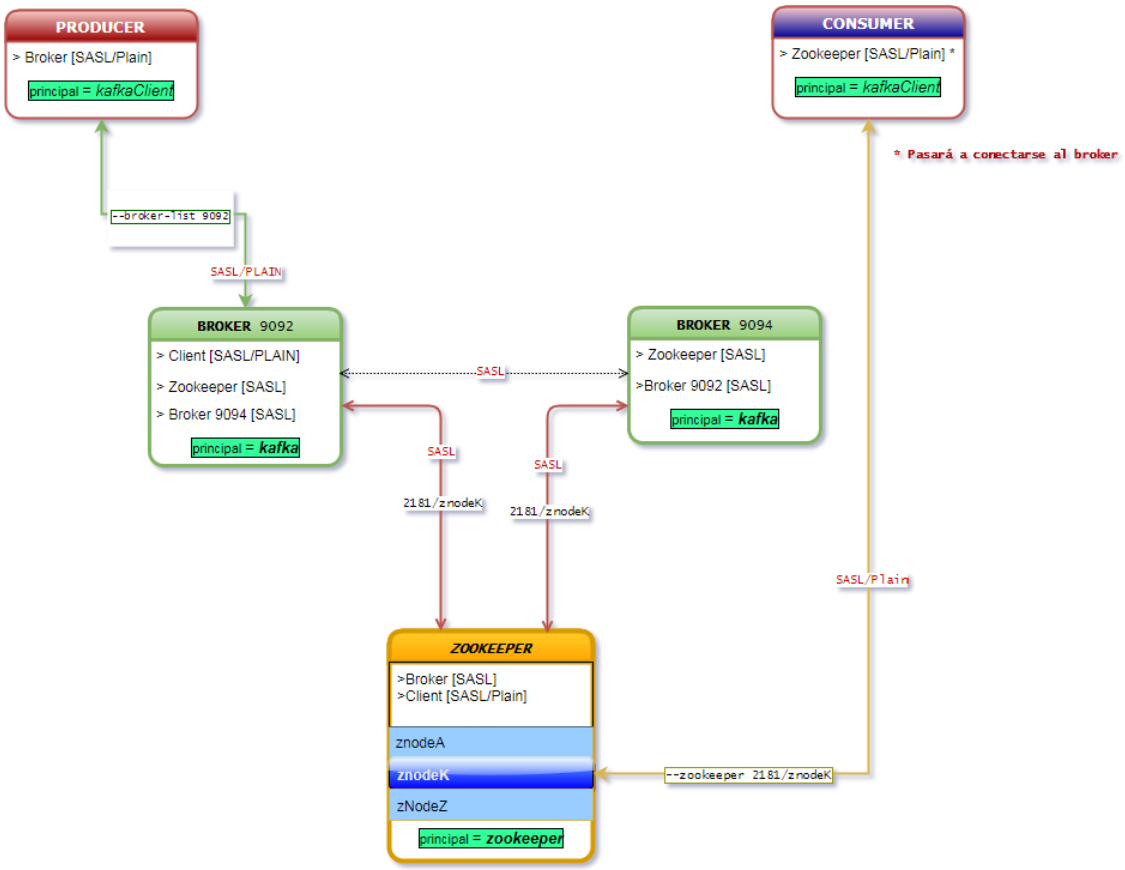
***Opcional.** obliga al cliente a autenticarse por SASL independientemente del zNode a acceder.Los no autenticados sólo podrán hacer ping y abrir y cerrar sesión.*

- ***JVM***

  Exportar la ubicación del fichero ***JAAS*** a la JVM.

```
export JVM_FLAGS="-Djava.security.auth.login.config=…/config/zookeeper_jaas.conf"
```

Una vez realizadas las configuraciones por cada principal, el mapa de comunicaciones en un cluster de Kafka sería el siguiente (con un Consumer basado en Zookeeper):



- **Brokers**
  La conexión al Zookeeper y la comunicación interna con otros Brokers se realiza mediante el protocolo SASL.
  La comunicación con otros clientes puede ser SASL o permitir también la conexión sin autenticación. Los puertos utilizados por el broker para utilizar autenticación / no autenticación con clientes deben ser diferentes:

  ```
  listeners = SASL_PLAINTEXT://localhost:9092, PLAINTEXT://localhost:9093
  ```

  Permite la conexión **sin autenticar** de clientes en el puerto **9093**, pero requiere autenticación **SASL** en el puerto **9092**.

- **Zookeeper**
  La conexión al Zookeeper y la comunicación interna con otros Brokers se realiza mediante el protocolo SASL.
  La comunicación con otros clientes (Consumers, principalmente) puede ser SASL o permitir también la conexión sin autenticación. En el caso en el que el Zookeeper permitiese la conexión no SASL (no se ha definido valor en el parámetro `requireClientAuthScheme`), los permisos del usuario estarán definidos por los *acls* del nodo zookeeper al que se conecta.

En el caso de ejemplo, dando por hecho que no se ha requerido el uso de un esquema de autenticación en la configuración del zookeeper y el ***zNodeK*** no tiene definidos permisos para usuarios "*no-SASL*":

1- El consumer se conectaría correctamente al zookeeper.
2- Al intentar cualquier **operación** (en este caso, ***Read***, ya que es un consumer), esta sería **denegada** por el servidor zookeeper, ya que no contempla el acceso a ese nodo a usuarios sin autenticar.

- **Producers**
  Los producers se conectan al broker tanto por SASL como por PLAIN, dependiendo del protocolo establecido en el puerto del broker. Un producer **sin autenticar nunca podrá producir** conectándose a un **puerto** que establece **SASL** como protocolo de autenticación.3

- **Consumers**
  Los consumers se podrán conectar al **zookeeper*** tanto por SASL como PLAIN, si no se ha definido ningún valor en el parámetro `requireClientAuthScheme`. Como se ha explicado ya en la sección del zookeeper, las operaciones permitidas por cliente estarán definidas en los acls de los zNodes.

*se conectarán al broker en versiones futuras. Los consumer Java API ya se conectan directamente al broker(misma mecánica que los brokers).

2. Autorización – Kafka ACLs

Kafka monta el **KACLI**, *Kafka Authorization Command Line Interface*. Aquí se establece el primer nivel de autorización en el cluster Kafka, donde se definen los permisos de **lectura/escritura/descripción** a nivel de **Topic**.

En un futuro, se espera que esta interfaz implemente los permisos propios del Zookeeper, de manera que sean los ACLs de Kafka los que permitan, por ejemplo, el **borrado** o **creación** de topics mediante propagación de permisos sobre los *zNode*s.

Los ACL de Kafka para un Topic tienen el siguiente formato:

```
Principal P is [Allowed/Denied] Operation O {From Host H} On Topic T
```

Para aplicar las políticas de acceso/permisos se utiliza la clase `kafka-acls`. Algunos ejemplos:

- `bin/kafka-acls.sh --authorizer kafka.security.auth.SimpleAclAuthorizer --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal User:lector --operation Read --topic KerbTopic`

  El usuario `lector` tiene permisos de lectura para el topic `KerbTopic`

- `bin/kafka-acls.sh --authorizer kafka.security.auth.SimpleAclAuthorizer --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal User:escritor --operation Write --topic KerbTopic`

  El usuario `escritor` tiene permisos de escritura para el topic `KerbTopic`

- `bin/kafka-acls.sh --authorizer kafka.security.auth.SimpleAclAuthorizer --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal User:presi --operation Write –operation Read –operation Describe --topic KerbTopic`

  El usuario `presi` tiene permisos de escritura,lectura y descripción para el topic `KerbTopic`

- `bin/kafka-acls.sh --authorizer kafka.security.auth.SimpleAclAuthorizer --authorizer-properties zookeeper.connect=localhost:2181 --remove --allow-principal User:lector --operation Read --topic KerbTopic`

  El usuario `lector` ya **no** tiene permisos de lectura para el topic `KerbTopic.`

- `bin/kafka-acls.sh --authorizer kafka.security.auth.SimpleAclAuthorizer --authorizer-properties zookeeper.connect=localhost:2181 --remove --allow-principal User:presi --operation Write --topic KerbTopic`

  El usuario `presi` ya **no** tiene permisos de escritura para el topic `KerbTopic`. Continúa pudiendo leer y obteniendo detalles del topic (**Read**, **Describe** siguen vigentes).

3. Autorización – Zookeeper ACLs

El último de los niveles de autorización es el del zookeeper, que controla el acceso y operaciones a los zNodes mediante sus propias ACLs. En este nivel se definen los permisos de operaciones que requieran de interacción directa con el zookeeper, como el **borrado** y **creado** de **topics**.

Para consultar y modificar los *acls* se utiliza el cliente del zookeeper, **`zkCli`**.

```
[zk: localhost:2181(CONNECTED) 10] create /znodeA "ejemplo"
Created /znodeA
[zk: localhost:2181(CONNECTED) 11] getAcl /znodeA
'world,'anyone
: cdrwa
[zk: localhost:2181(CONNECTED) 12] setAcl /znodeA world:anyone:r,sasl:kafka:cdrwa,sasl:kafkaClientConsumer:rw
cZxid = 0x1e0002550e
ctime = Fri Sep 22 10:09:59 CEST 2017
mZxid = 0x1e0002550e
mtime = Fri Sep 22 10:09:59 CEST 2017
pZxid = 0x1e0002550e
cversion = 0
dataVersion = 0
aclVersion = 1
ephemeralOwner = 0x0
dataLength = 9
numChildren = 0
[zk: localhost:2181(CONNECTED) 13] getAcl /znodeA
'world,'anyone
: r
'sasl,'kafka
: cdrwa
'sasl,'kafkaClientConsumer
: rw
```

Ejemplo:

1. Creamos un nodo nuevo, znodeA.
2. Comprobamos los permisos por defecto.
3. Modificamos los permisos. En el caso de ejemplo, se establecen tres políticas:
   - **Usuario sin autenticar**: permisos de lectura.
   - SASL – **Kafka** : todos los permisos(lectura, escritura, creado, borrado, admin).
   - SASL – **KafkaClientConsumer** : permisos de lectura/escritura.