

Documento de configuración de Kafka



Única2
Raras pero no invisibles

Minsait

Febrero 2026

Histórico de versiones:

Versión	Realizado por:	Fecha
1.0	Minsait	05/02/2026
1.1	Revisión Minsait	05/03/2026
1.2	Revisión Minsait: se añade anexo I	16/03/2026

Contenido

1.	Objetivo.....	2
2.	Configuración del listener de acceso externo a Kafka	2
2.1.	Activar Acceso Externo Seguro (TLS) en un Clúster Kafka en Kubernetes	2
2.1.1.	Conceptos Clave	2
2.2.1.	Pasos de Ejecución.....	3
2.3.1.	Rollback (si algo falla)	11
2.4.1.	Checklist.....	12
3.	Configuración de usuarios en Kafka	13
3.1.	Acceso Seguro de Clientes Externos a Kafka: Creación de Usuarios, Permisos y Configuración TLS.....	13
3.1.1.	Conceptos Clave	13
3.2.1.	Alcance	15
3.3.1.	Procedimiento de Creación de usuarios	16
3.4.1.	Configuración de clientes	17
4.	Validación de conexión y prueba de suscripción a topics Kafka	20
4.1.	Validación de conexión mediante CLI (Kafka Console Tools)	20
4.2.	Prueba de suscripción a un topic Kafka	20
	Anexo I	23

1. Objetivo

El presente documento tiene como finalidad describir de forma detallada el procedimiento para habilitar el acceso externo a un clúster Kafka integrado en la plataforma Onesait Healthcare, mediante la configuración de un listener TLS adicional, sin modificar la comunicación interna del clúster.

El documento incluye la descripción de los pasos necesarios para permitir conexiones externas a Kafka a través de la creación de usuarios dedicados, la definición de permisos específicos mediante listas de control de acceso y la configuración de mecanismos de autenticación y cifrado seguros basados en SCRAM y TLS. Estas configuraciones permiten proporcionar un acceso controlado, identificable y protegido a los eventos publicados en determinados topics, manteniendo el aislamiento respecto a los accesos internos del clúster.

Este documento servirá como referencia técnica para los equipos responsables de la configuración y operación de la mensajería Kafka, asegurando una implantación homogénea.

2. Configuración del listener de acceso externo a Kafka

Este manual debe seguirse en el caso de que se quiera permitir el acceso directo a Kafka por parte de diferentes orígenes o clientes externos a los propios módulos de Onesait Healthcare.

De esta forma, se permitirá que sistemas externos puedan suscribirse a los eventos que se producen en determinados topics de forma controlada y usando un acceso independiente al de la propia plataforma de Onesait Healthcare.

2.1. Activar Acceso Externo Seguro (TLS) en un Clúster Kafka en Kubernetes

Procedimiento operativo para habilitar un listener externo seguro (TLS) en un clúster Kafka ya instalado. Para su ejecución, se requiere acceso para operar con kubectl y disponer de permisos para editar el ConfigMap y el StatefulSet, así como para crear los recursos Service y Secret.

2.1.1. Conceptos Clave

¿Qué es un Keystore?

Un keystore es un archivo que contiene la clave privada y el certificado del servidor (en este caso, será Kafka).

Es la identidad del broker. Kafka lo usa para:

- Demostrar que es quien dice ser.

- Establecer conexiones TLS cifradas.
- Generalmente es un archivo .jks o .p12.

¿Qué es un Truststore?

Un truststore es un archivo que contiene los certificados de confianza, normalmente los certificados raíz (CA) que firmaron los certificados del broker. Es lo que permite a los clientes validar que se están conectando a un servidor legítimo.

Kafka lo usa para:

- Verificar certificados presentados por otros.
- Validar conexiones TLS entrantes o salientes.
- También suele ser .jks o .p12.

En resumen: Keystore = identidad de Kafka (privada). Truststore = autoridades de confianza (público).

¿Qué son los Listeners en Kafka?

Un listener es un "puerto" que Kafka abre para que se conecten clientes. Kafka puede tener varios listeners simultáneamente, cada uno escuchando en un puerto diferente y con diferentes configuraciones de seguridad.

Si Kafka ya está instalado, en el clúster actual hay varios:

- Puertos internos: Solo accesibles desde dentro del cluster.
- Con este procedimiento, vamos a crear: Puerto 9095 con cifrado TLS, accesible desde fuera.

IMPORTANTE: No debemos tocar los listeners internos, estos ya funcionan. Solo vamos a agregar uno nuevo en el puerto 9095.

¿Por qué necesitamos un listener externo con TLS?

Inicialmente solo puedes acceder a Kafka desde dentro del cluster. Para acceder desde fuera, necesitas:

1. Un puerto expuesto al exterior (puerto 9095).
2. Cifrado TLS para que la conexión sea segura.

2.2.1. Pasos de Ejecución

Paso 1: Generar o Preparar Certificados TLS

Antes de ejecutar los comandos de este paso, es importante entender que aquí se establece la base de la seguridad TLS. En este apartado se generan o preparan los certificados

necesarios para que Kafka pueda identificarse como un servidor seguro y cifrar las conexiones externas.

Si ya cuentas con los certificados, y tienes `keystore.jks` y `truststore.jks` generados (ej: emitidos por tu PKI interna), **salta a Paso 2**. Solo asegúrate de tener las contraseñas.

¿No tienes certificados?

En caso de no tener los certificados, podemos seguir otros pasos, pero antes de ello:

Recomendación de seguridad:

Generalmente, los certificados TLS deben ser:

- Emitidos por tu autoridad certificadora corporativa (PKI).
- Validados por un equipo de seguridad.
- Almacenados en un sistema seguro (HSM, vault, etc.).

¿Cuándo puedes usar certificados autofirmados (self-signed)?

Usa certificados generados manualmente SOLO si:

- Entornos de desarrollo, QA o laboratorio.
- Acceso interno únicamente (detrás de VPN o firewall).
- Kafka NO expuesto a internet.
- Solo equipos internos consumen el listener.
- No hay requisitos corporativos de seguridad TLS estrictos.

¿Cuándo NO debes usar certificados autofirmados?

Si alguno aplica, **debes usar certificados de tu PKI corporativa o con autorización para ello**:

- Kafka será accesible desde internet.
- Será consumido por proveedores, partners o sistemas externos.
- Existen requisitos corporativos de validación TLS automática.
- Se requiere validación de certificado por múltiples clientes.

Generar certificados autofirmados (solo para development/testing)

Si tu organización no tiene PKI y necesitas un certificado:

Importante: Ejecuta esto en un entorno seguro. No guardes la contraseña en el historial.

Keystore del Broker

```
keytool -genkeypair -alias kafka-broker
```

```
-keyalg RSA -keystore keystore.jks
```

```
-storepass <STORE_PASSWORD_GENERADA_MANUALMENTE> -keypass  
<PASSWORD_GENERADA_MANUALMENTE>
```

```
-dname "CN=<hostname-externo>,OU=Kafka,O=Empresa,L=Ciudad,S=Provincia,C=ES"
```

Qué hace:

Crea dos archivos `jks` (Java KeyStore) con certificados autofirmados.

Notas de seguridad:

- Reemplaza <STORE_PASSWORD_GENERADA_MANUALMENTE> y <PASSWORD_GENERADA_MANUALMENTE> con una contraseña que generes aparte (no la escribas aquí).
- El `CN=` debe coincidir exactamente con tu dirección externa (el hostname/FQDN externo que se usará como después como <hostname-externo> en la configuración de advertised.listeners).

Truststore

```
keytool -import -file ca-cert.pem  
-keystore truststore.jks  
-storepass <STORE_PASSWORD_GENERADA_MANUALMENTE>  
-alias CARoot
```

Qué hace:

Crea el archivo `truststore.jks` importando la CA (autoridad certificadora) que firmó tu certificado. Los clientes externos usarán esto para validar que Kafka es legítimo.

Nota: Debes tener el archivo `ca-cert.pem` disponible antes de ejecutar este comando.

Resumen Paso 1:

Has generado dos archivos (keystore.jks y truststore.jks) con certificados autofirmados y los protegiste localmente.

⚠ ADVERTENCIA sobre Certificados:

- **Autofirmados (self-signed):** se usan cuando

- Entornos de **desarrollo, QA o laboratorios técnicos.**
- Sistemas ubicados en una **red interna**, protegida por **VPN**, segmentación o **firewall corporativo.**
- Casos donde no hay exposición al exterior y el riesgo es bajo.

- **PKI Corporativos:** se usan cuando

- Si el servicio será accesible **desde Internet.**
- Si existen **clientes externos** que deben conectarse a Kafka.
- Si tu organización exige cumplir con **normativas de seguridad** o auditorías.
- En entornos **productivos** donde se requieren garantías formales de identidad y trazabilidad.

- **En caso de dudas:** consultar con el equipo de seguridad o el responsable de infraestructura para validar el tipo de certificado adecuado antes de continuar.

Paso 2: Crear Secret TLS en Kubernetes

Los certificados suministrados por PKI corporativo, o los que generaste localmente, necesitan estar disponibles en Kubernetes. Los Secrets son el lugar seguro para guardar datos sensibles (como certificados y contraseñas). Kubernetes los encripta automáticamente en etcd.

IMPORTANTE ANTES DE CONTINUAR: Si acabas de generar los certificados localmente, protégelos mientras están en tu máquina:

Protect

```
chmod 600 keystore.jks truststore.jks
```

Qué hace:

Esto limita el acceso: solo tú podrás leerlos.

Ahora crea el Secret en Kubernetes con los archivos protegidos:

Secret TLS Completo

```
kubectl create secret generic kafka-external-tls  
--from-file=keystore.jks  
--from-file=truststore.jks  
--from-literal=keystorePassword=<password>  
--from-literal=truststorePassword=<password>  
-n <namespace>
```

Reemplaza `<TU_CONTRASEÑA>` con la contraseña que usaste al generar los certificados (la misma que en `<STORE_PASSWORD_GENERADA_MANUALMENTE>`).

Qué hace:

Almacena los certificados en un Secret de Kubernetes (encriptado en etcd). Una vez aquí, los permisos locales ya no son tomados: Kubernetes gestiona el acceso.

Paso 3: Configurar Kafka para usar el nuevo listener

Kafka necesita saber que tiene un nuevo listener (puerto 9095) disponible y cómo debe configurarlo. Esto se hace modificando el archivo de configuración `server.properties` que está guardado en el ConfigMap de Kubernetes. Sin esta configuración, Kafka no sabría que debe escuchar en el puerto 9095.

Backup del ConfigMap

Haz backup de la configuración del ConfigMap actual:

```
kubectl get configmap ohien-kafka-config -n <namespace> -o yaml > kafka-config-backup.yaml
```

Editar ConfigMap

Abrimos con el editor el configmap:

```
kubectl edit configmap ohien-kafka-config -n <namespace>
```

esto abrirá en nuestro editor, un YAML al cual debe respetarse la indentación.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ohien-kafka-config
  namespace: <namespace>
data:
  server.properties: |
    # Aquí están todas las propiedades de Kafka

listeners=CLIENT://0.0.0.0:<PUERTO_INTERNO>,INTERNAL://0.0.0.0:<PUERTO_INTERNO>,CONTROLLER://0.0.0.0:<PUERTO_INTERNO>
  advertised.listeners=CLIENT://kafka-0.kafka-headless:<PUERTO_INTERNO>,INTERNAL://kafka-0.kafka-headless:<PUERTO_INTERNO>

  listener.security.protocol.map=CLIENT:<INTERNAL_PROTOCOL>,INTERNAL:<INTERNAL_PROTOCOL>,CONTROLLER:<INTERNAL_PROTOCOL>
  # ... más líneas ...
```

Añadir en data, server.properties

Al final de la línea listeners sin modificar los previos agregando un separador de coma (,) y el nombre de nuestro nuevo listener.

```
listeners=...,EXTERNAL_TLS://0.0.0.0:9095
```

Advertised externo:

Al final de la línea advertised.listeners sin modificar los previos agregando un separador de coma (,) y el nombre de nuestro nuevo listener.

```
advertised.listeners=...,EXTERNAL_TLS://<hostname-externo>:9095
```

Protocolo:

```
listener.security.protocol.map=...,EXTERNAL_TLS:SSL
```

Configuración TLS:

Al final del bloque server.properties, asegúrate de dejar una línea en blanco antes de añadir estas 4 líneas nuevas:

```
listener.name.external_tls.ssl.keystore.location=/kafka/ssl/keystore.jks
listener.name.external_tls.ssl.keystore.password=${keystorePassword}
listener.name.external_tls.ssl.truststore.location=/kafka/ssl/truststore.jks
listener.name.external_tls.ssl.truststore.password=${truststorePassword}
```

Resumen de Paso 3:

- Modifica 3 líneas existentes (añade al final de cada una).
- Añade 4 líneas nuevas con salto de línea antes.
- Total: 7 cambios en el ConfigMap.

Paso 4: Montar los certificados en el StatefulSet

¿Por qué? Los certificados están guardados en el Secret de Kubernetes, pero los contenedores de Kafka no pueden acceder a ellos automáticamente. Necesitamos "montar" el Secret como archivos dentro del contenedor en la ruta /kafka/ssl. De esta forma, Kafka puede leerlos cuando se inicie.

Backup StatefulSet

Haz backup del StatefulSet actual:

```
kubectl get statefulset ohien-kafka -n <namespace> -o yaml > kafka-statefulset-backup.yaml
```

Editar StatefulSet

Al ejecutar el comando veremos el YAML en el editor.

```
kubectl edit statefulset ohien-kafka -n <namespace>
```

Se abrirá en el editor una estructura YAML como esta, respetar el formato:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: ohien-kafka
  ...
  labels:
    app.kubernetes.io/instance: iengine
    app.kubernetes.io/managed-by: Helm
spec:
  template:
    spec:
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: kafka-data
          # AQUÍ AÑADES UN VOLUMEN NUEVO
      containers:
        - name: kafka
          volumeMounts:
            - name: data
              mountPath: /var/lib/kafka/data
          # AQUÍ AÑADES UN MOUNT NUEVO
```

Volumen TLS:

Cambio 1: En `spec.template.spec.volumes:`, al final de la lista, AÑADE:

```
- name: kafka-external-tls
  secret:
    secretName: kafka-external-tls
```

Mount en contenedor:

Cambio 2: En `spec.template.spec.containers[0].volumeMounts:`, al final de la lista, AÑADE:

```
- name: kafka-external-tls
  mountPath: /kafka/ssl
  readOnly: true
```

Resumen de Paso 4:

- Añade 1 volumen nuevo en `spec.template.spec.volumes:`.
- Añade 1 mount nuevo en `spec.template.spec.containers[0].volumeMounts:`.
- Total: 2 cambios en el StatefulSet.

Paso 5: Crear Service para exponer el puerto externamente

Kafka está dentro del cluster de Kubernetes, así que desde fuera no se puede acceder. Un Service LoadBalancer actúa como una "puerta" que expone el puerto 9095 del clúster hacia el exterior (red corporativa, internet, etc.). Sin esto, el cliente externo no podría conectarse.

Crea un archivo `kafka-external-service.yaml` con este contenido YAML completo:

kafka-external-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kafka-external
  namespace: <namespace>
spec:
  type: LoadBalancer
  ports:
    - name: external-tls
      port: 9095
      targetPort: 9095
      protocol: TCP
  selector:
    app.kubernetes.io/name: ohien-kafka
    app.kubernetes.io/instance: <mykafka>
```

Nota: El valor para el selector, debe coincidir con los valores existentes para el statefulset con el que ohien-kafka con el que trabajamos anteriormente.

labels: [app.kubernetes.io/instance](#): mykafka

Aplicación:

```
kubectl apply -f kafka-external-service.yaml -n <namespace>
```

Qué hace:

Crea un LoadBalancer que expone el puerto 9095 hacia el exterior del clúster.

Nota: El **EXTERNAL-IP**, puede quedar <pending> hasta que el proveedor asigne la IP.

Paso 6: Reiniciar los pods de Kafka

Debido a que hemos modificado la configuración (Paso 3), añadido certificados (Paso 4) y expuesto el Service (Paso 5). Los pods de Kafka actuales aún están corriendo con la configuración antigua. Necesitamos reiniciarlos para que carguen todos estos cambios.

```
kubectl rollout restart statefulset/ohien-kafka -n <namespace>
kubectl rollout status statefulset/ohien-kafka -n <namespace>
```

Qué hace:

Reinicia los pods uno por uno para que carguen la nueva configuración. Espera a que todos estén listos.

Paso 7: Verificar que todo funciona

Antes de intentar conectar desde fuera, queremos asegurarnos de que:

1. Kafka está escuchando en el nuevo listener EXTERNAL_TLS (puerto 9095).

2. El Service está correctamente expuesto y tiene una dirección accesible desde fuera.

Verificación 1: Comprobar que el listener EXTERNAL_TLS está configurado en Kafka:

```
kubectl exec -it <kafka-pod> -n <namespace> --  
cat /kafka/metadata/server.properties | grep EXTERNAL_TLS
```

Qué hace:

Confirma que Kafka está escuchando en el nuevo listener (puerto 9095 con TLS).

Salida esperada:

dentro del archivo del pod deberíamos ver la configuración:

```
listeners=...,EXTERNAL_TLS://0.0.0.0:9095  
advertised.listeners=...,EXTERNAL_TLS://kafka.miempresa.com:9095  
listener.security.protocol.map=...,EXTERNAL_TLS:SSL...  
listener.name.external_tls.ssl.keystore.location=/kafka/ssl/keystore.jks  
listener.name.external_tls.ssl.keystore.password=...  
listener.name.external_tls.ssl.truststore.location=/kafka/ssl/truststore.jks  
listener.name.external_tls.ssl.truststore.password=...
```

✅ **Éxito:** Ves 3 líneas con `EXTERNAL_TLS` (listeners, advertised.listeners, protocol.map) + 4 líneas de configuración TLS.

❌ **Error:** Si no ves estas líneas, el ConfigMap no fue modificado correctamente. Vuelve a Paso 3.

Verificación 2: Comprobar que el Service está expuesto externamente

```
kubectl get svc kafka-external -n <namespace>
```

Que hace:

Confirma que el LoadBalancer ha asignado una dirección IP/nombre para acceso externo.

Salida esperada:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)	AGE
kafka-external	LoadBalancer	10.100.123.45	203.0.113.50	9095:31234/TCP	2m

Busca la columna `EXTERNAL-IP`:

- ✅ Si ves una IP o nombre DNS: El Service está correctamente expuesto. Esa es la dirección que usarán los clientes externos.
- ❌ Si ves ``: El LoadBalancer aún no ha asignado una IP. Espera 1-2 minutos y vuelve a ejecutar el comando.
- ❌ Si no ves la línea: El Service no fue creado. Vuelve a Paso 5.

Prueba TLS: Archivo client-tls.properties:

```
security.protocol=SSL  
ssl.truststore.location=truststore.jks  
ssl.truststore.password=<password>
```

2.3.1. Rollback (si algo falla)

Si algo no funciona y necesitas volver al estado anterior, sigue estos pasos en orden:

Paso 1: Restaurar la configuración de Kafka original**

```
kubectl apply -f kafka-config-backup.yaml -n <namespace>
```

Qué hace:

Revierte el ConfigMap a la versión sin EXTERNAL_TLS (antes de Paso 3).

Nota: Solo funciona si hiciste el backup en Paso 3. Si no lo hiciste, edita manualmente el ConfigMap y elimina las 7 líneas que añadiste.

Paso 2: Eliminar el Service externo

```
kubectl delete svc kafka-external -n <namespace>
```

Qué hace:

Elimina el LoadBalancer que exponía Kafka hacia el exterior.

Verifica:

```
kubectl get svc -n <namespace>
```

Si no ves `kafka-external` en la lista, fue eliminado correctamente.

Paso 3: Eliminar el StatefulSet backup (opcional pero recomendado)

```
kubectl delete secret kafka-external-tls -n <namespace>
```

Qué hace:

Elimina el Secret con los certificados TLS.

Verifica:

```
kubectl get secrets -n <namespace>
```

Si no ves `kafka-external-tls`, fue eliminado correctamente.

Paso 4: Reiniciar Kafka para cargar la configuración original

```
kubectl rollout restart statefulset/ohien-kafka -n <namespace>  
kubectl rollout status statefulset/ohien-kafka -n <namespace>
```

Qué hace:

Reinicia los pods para que carguen la configuración original del ConfigMap.

Espera a que todos los pods estén `Running` y `Ready`.

Verificación final:

```
kubectl exec -it ohien-kafka-0 -n <namespace> -- \  
cat /kafka/metadata/server.properties | grep -c EXTERNAL_TLS
```

Resultado esperado: `0` (cero líneas con EXTERNAL_TLS = rollback exitoso).

⚠ **Notas importantes:**

- Los backups (`kafka-config-backup.yaml`, `kafka-statefulset-backup.yaml`) se guardan localmente. No se eliminan automáticamente.

- Si Kafka sigue sin funcionar después de rollback, **verifica que los pods estén en estado `Ready`**:

```
kubectl get pods -n <namespace> | grep ohien-kafka
```

2.4.1. Checklist

- Certificados generados (`keystore.jks`, `truststore.jks`).
- Secret creado (`kafka-external-tls`).
- ConfigMap actualizado con las 7 líneas de configuración.
- StatefulSet actualizado (volumes + volumeMounts).
- Service externo creado.
- Pods reiniciados y ready.
- Verificación: `grep EXTERNAL_TLS` muestra las líneas.
- Verificación: Service tiene EXTERNAL-IP asignada.

3. Configuración de usuarios en Kafka

Este manual debe seguirse en el caso de que se quiera permitir el acceso directo a Kafka con usuarios con permisos para diferentes clientes externos a los propios módulos de Onesait Healthcare.

De esta forma, se permitirá que sistemas externos puedan conectarse de forma adecuada mediante usuarios y poder suscribirse a los eventos que se producen en determinados topics de forma controlada y usando un acceso independiente al de la propia plataforma de Onesait Healthcare. El acceso se realizará mediante usuarios SCRAM+ACLs y que el objetivo es limitar acciones por topic/consumer group.

3.1. Acceso Seguro de Clientes Externos a Kafka: Creación de Usuarios, Permisos y Configuración TLS

3.1.1. Conceptos Clave

Usuario SCRAM

Kafka permite autenticar clientes mediante un mecanismo basado en usuario y contraseña llamado SCRAM.

Este usuario proporciona:

- Identificación clara del cliente externo.
- Trazabilidad de accesos.
- Capacidad para definir permisos específicos.

El usuario SCRAM es completamente independiente de los módulos internos de la plataforma.

Política de uso de SCRAM-SHA-256 y SCRAM-SHA-512

Kafka permite utilizar distintos mecanismos SCRAM para el almacenamiento y validación de credenciales.

La plataforma OHIEN adopta el siguiente criterio:

1. SCRAM-SHA-512: mecanismo preferente

Se utilizará **SCRAM-SHA-512** para:

- Nuevos desarrollos.
- Librerías Kafka modernas (Java, Go, .NET, Python 3.x).
- Sistemas con soporte actualizado para SCRAM.
- Integraciones cuyo roadmap tecnológico lo permita.

Motivación:

- Función hash reforzada.

- Alineamiento con políticas corporativas de seguridad más estrictas.
- Mayor resiliencia para almacenamiento de contraseñas.

2. SCRAM-SHA-256: mecanismo alternativo para compatibilidad

Se utilizará **SCRAM-SHA-256** únicamente cuando:

- El cliente externo use librerías antiguas o no actualizables.
- Existan herramientas o conectores legacy con soporte limitado.
- Frameworks, drivers o SDKs no implementen SHA-512 correctamente.
- Se requiera asegurar compatibilidad con entornos híbridos o transitorios.

Motivación:

- Máxima compatibilidad.
- Garantiza interoperabilidad con una base más amplia de clientes Kafka.
- Evita errores de autenticación en sistemas que no soportan SHA-512.

ACLs (Access Control Lists)

Las ACLs son reglas que indican qué acciones tiene permitido realizar un usuario dentro de Kafka.

Una ACL puede autorizar:

- Lectura de un topic.
- Escritura en un topic.
- Participación en un consumer group.
- Acciones de descripción o consulta del estado del clúster.

Un usuario sin ACLs no podrá acceder a ninguna operación dentro de Kafka.

¿Qué es un TLS (Transport Layer Security)?

TLS es el mecanismo que permite cifrar la comunicación entre el cliente y Kafka. Sin TLS, las credenciales y los datos viajarían sin protección.

TLS garantiza:

1. Cifrado del tráfico.
2. Validación de que el servidor Kafka es auténtico.
3. Prevención de interceptación o manipulación.

3.2.1. Alcance

3.2.1.1. Usuarios utilizados por módulos internos del clúster (SASL_PLAINTEXT)

Aplicable cuando:

- El acceso a Kafka proviene de servicios que se ejecutan dentro del mismo clúster Kubernetes.
- La comunicación se realiza a través de la red interna del clúster.
- El entorno ya garantiza aislamiento, control de tráfico y seguridad perimetral.

Características:

- No se requiere cifrado TLS.
- El mecanismo de autenticación utilizado es SASL_PLAINTEXT + SCRAM.
- Las credenciales se almacenan en Secrets internos del clúster.
- Los listeners internos permanecen sin modificaciones.
- Este escenario se orienta a módulos de la plataforma o servicios de integración internos.

3.2.1.2. Usuarios destinados a clientes o sistemas externos (SASL_SSL/TLS)

Aplicable cuando:

- Se necesita permitir acceso a Kafka desde fuera del clúster Kubernetes.
- El origen puede ser una aplicación corporativa, un proveedor externo o un sistema remoto seguro.
- La conexión requiere protección criptográfica y validación del servidor mediante certificados TLS.

Características:

- Se utiliza SASL_SSL + SCRAM como mecanismo de autenticación.
- El acceso debe realizarse siempre a través de un listener TLS expuesto específicamente para este propósito (listener definido en el apartado 2).
- El cliente requiere un truststore válido para establecer la conexión.
- Se deben otorgar únicamente los permisos (ACLs) necesarios para el sistema externo.
- Las contraseñas y certificados deben gestionarse mediante repositorios seguros o Secrets cifrados.

3.2.1.3. Consideraciones Generales

Independientemente del tipo de usuario, se recomienda:

- Cada usuario debe ser único y asociarse a un sistema concreto.
- Las ACLs deben limitarse estrictamente al ámbito necesario (principio de mínimo privilegio).
- Las credenciales nunca deben almacenarse en repositorios de código sin cifrado.
- Se recomienda registrar en el inventario qué módulo o cliente externo corresponde a cada usuario creado.

3.2.1.4. Exclusiones

Este procedimiento no cubre:

- Automatización masiva de creación de usuarios.
- Configuraciones multi-cluster o replicación cross-datacenter.
- Gestión avanzada de certificados (PKI, rotación automática, etc.).

3.3.1. Procedimiento de Creación de usuarios

3.3.1.1. Prerrequisitos

Antes de comenzar, es necesario tener previsto diferentes elementos:

- Acceso a los brokers Kafka (`bootstrap.servers`).
- Herramientas de cliente Kafka (`kafka-configs.sh`, `kafka-acls.sh`, `kafka-console-producer.sh`, `kafka-console-consumer.sh`) o una imagen de contenedor con estas utilidades.
- Permisos administrativos para gestionar usuarios y ACLs.

3.3.1.2. Creación de Usuario SCRAM

Es importante comprender que la creación de usuario es igual para ambos casos. El usuario no está “marcado” como interno o externo.

- Se requiere el servidor <broker:9092>.
- Usuario existente con permisos previos necesarios para administración de usuarios .
- Nombre de usuario a crear <USERNAME>.
- Definir el formato SCRAM a utilizar.

Crear usuario en Kafka

Create user/password

```
./bin/kafka-configs.sh --bootstrap-server <broker:9092> \  
--alter \  
--add-config 'SCRAM-SHA-512=[password=<PASSWORD>]' \  

```

```
--entity-type users \  
--entity-name <USERNAME>
```

(Usar SCRAM-SHA-256 solo si el cliente externo no soporta SHA-512.).

Verificar que el usuario fue creado

verify user

```
./bin/kafka-configs.sh --bootstrap-server <broker:9092> \  
--describe --entity-type users --entity-name <USERNAME>
```

Asignar permisos (ACLs)

a) Ver un topic:

ver un topic

```
./bin/kafka-acls.sh --bootstrap-server <broker:9092> \  
--add --allow-principal User:<USERNAME> \  
--operation Describe \  
--topic '<TOPIC>'
```

b) Permiso escritura:

write topic

```
./bin/kafka-acls.sh --bootstrap-server <broker:9092> \  
--add --allow-principal User:<USERNAME> \  
--operation write \  
--topic '<TOPIC>'
```

c) Permiso lectura:

read topic

```
./bin/kafka-acls.sh --bootstrap-server <broker:9092> \  
--add --allow-principal User:<USERNAME> \  
--operation Read \  
--topic '<TOPIC>'
```

d) Permitir consumo en un grupo:

add meber group

```
./bin/kafka-acls.sh --bootstrap-server <broker:9092> \  
--add --allow-principal User:<USERNAME_INTERNAL> \  
--operation Read \  
--group '<GROUP>'
```

3.4.1. Configuración de clientes

Hasta este punto, todo el procedimiento ha sido **idéntico** para cualquier tipo de usuario:

- La creación del usuario SCRAM.
- El establecimiento de su contraseña.
- La asignación de ACLs.

Como se ha indicado antes, Kafka no clasifica usuarios como *internos* o *externos*. Lo que cambia **no es el usuario**, sino **el contexto desde donde se conecta el cliente**.

3.4.1.1. Configuración para Clientes Internos

Aunque una aplicación esté dentro del clúster y Kafka sea accesible desde la red interna, Si esta activado correctamente, Kafka no permite accesos anónimos; siempre exige tres cosas obligatorias para establecer una conexión:

- El protocolo de seguridad que debe usar el cliente.
- El mecanismo de autenticación (SASL).
- El usuario y la contraseña.

Kafka no “adivina” estos parámetros. El cliente debe decir explícitamente:

- Cómo se autentica.
- Cómo debe hablar con Kafka.
- Y qué credenciales utiliza.

Para eso se crea el archivo `client-internal.properties`.

`client-internal.properties`

```
security.protocol=SASL_PLAINTEXT  
sasl.mechanism=SCRAM-SHA-512
```

NOTA: Usar SCRAM-SHA-256 solo si el cliente externo no soporta SHA-512.

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \  
  username="<USERNAME>" \  
  password="<PASSWORD>";
```

- Su configuración se realiza para uso de listener interno Kafka.
- **Sin TLS.**
- No requiere truststore.
- Menos configuración en microservicios internos.

3.4.1.2. Configuración para acceso externo (TLS + SASL_SSL)

Esta sección describe el proceso completo para configurar un usuario destinado a sistemas que se conectarán **desde fuera del clúster Kubernetes**, por ejemplo:

- Aplicaciones corporativas externas.
- Sistemas de terceros.
- Proveedores.
- Integraciones que no ejecutan sus servicios dentro de Kubernetes.

A diferencia de los módulos internos, estos clientes **deben** conectarse usando:

- **Autenticación SCRAM** (SHA-512 preferente, pero puede ser SHA-256 si usa librerías antiguas).

- **Cifrado TLS.**
- **Truststore local.**
- **El listener externo configurado para acceso seguro.**

De esta manera se garantiza un acceso protegido, controlado. Para ello se requiere contar con:

Usuario previamente creado

Archivo truststore.jks previamente generado y su password

client-external.properties

```
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-512

ssl.truststore.location=/path/to/truststore.jks
ssl.truststore.password=<TRUSTSTORE_PASSWORD>

sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \
  username="<USERNAME>" \
  password="<PASSWORD>";
```

Los clientes externos deben conectarse usando:

- Listener TLS externo.
- Certificado válido del bróker.
- Truststore local.
- **SASL_SSL + TLS.**

3.4.1.3. Ejemplo práctico: Uso del client-external.properties desde línea de comandos

Una vez creado el usuario, asignadas las ACLs y configurado el archivo client-external.properties, es necesario validar que la conexión funciona correctamente.

Este es el comando real para probar la conexión con Kafka usando el listener externo TLS.

Ejemplo con Kafka-console-producer

```
./bin/kafka-topics.sh \
  --bootstrap-server kafka-ext.<dominio>:9095 \
  --command-config client-external.properties \
  --list
```

Cuando el comando se ejecute correctamente, podrás ver la lista de topics directamente en la consola, esto permitirá comprobar tres cosas:

- Autenticación SCRAM correcta.
 - Si la contraseña o usuario fallan → SaslAuthenticationException.
- TLS bien configurado.
 - Si el truststore es incorrecto → SSLHandshakeException.
- ACLs correctas para el topic.

- Si el usuario no tiene permisos → TopicAuthorizationException.

4. Validación de conexión y prueba de suscripción a topics Kafka

Una vez creados los usuarios, asignadas las ACLs y configurado el fichero `client-external.properties`, es necesario validar la conectividad con Kafka y confirmar que el cliente puede suscribirse correctamente a los topics configurados (los topics para los que se han definido ACLs en el apartado 3). Este proceso consta de dos pasos: primero comprobar la conexión mediante las utilidades de línea de comandos, y posteriormente probar el consumo real de mensajes.

4.1. Validación de conexión mediante CLI (Kafka Console Tools)

Para confirmar que la configuración del cliente es correcta, puede utilizarse el comando `kafka-topics.sh` con el listener externo TLS. El siguiente ejemplo permite listar los topics disponibles usando las credenciales y parámetros definidos en `client-external.properties`:

```
./bin/kafka-topics.sh \  
  --bootstrap-server kafka-ext.<dominio>:9095 \  
  --command-config client-external.properties \  
  --list
```

Si el comando se ejecuta correctamente, se mostrará directamente en la consola la lista de topics accesibles por el usuario. Esto valida tres aspectos fundamentales:

- **Autenticación SCRAM correcta:**
 - Si usuario/contraseña fallan → SaslAuthenticationException.
- **Configuración TLS válida:**
 - Si el truststore no es válido → SSLHandshakeException.
- **ACLs correctas para los topics:**
 - Si el usuario carece de permisos → TopicAuthorizationException.

4.2. Prueba de suscripción a un topic Kafka

Una vez validada la conexión básica, es necesario comprobar que el cliente puede suscribirse y recibir mensajes del topic. Para ello Kafka proporciona librerías oficiales para múltiples lenguajes de programación, disponibles en:

<https://docs.confluent.io/platform/current/clients/index.html>

Además, es posible realizar pruebas desde línea de comandos utilizando `kafka-console-consumer`, disponible en las distribuciones oficiales del cliente Kafka (Linux, macOS, Windows), descargables desde:

<https://kafka.apache.org/downloads>

El Kafka desplegado es compatible con clientes **a partir de la versión 3.5**, aunque se recomienda utilizar la versión más reciente.

Ejemplo de suscripción con kafka-console-consumer:

```
kafka-console-consumer \  
  --bootstrap-server <<servidor_kafka>> \  
  --consumer.config <<fichero_configuracion_cliente>> \  
  --topic <<nombre_topic>> \  
  --group <<grupo_topic>>
```

Parámetros:

- **<<servidor_kafka>>**

Dirección del listener Kafka externo.

Ejemplo:

```
cluster-kafka-bootstrap-oh-kafka-pre.apps.clupropenshift.ssib.es:443
```

- **<<fichero_configuracion_cliente>>**

Fichero .properties con los parámetros de autenticación y TLS.

Ejemplo:

```
ssl.truststore.location=D:/temp/kafka-truststore-des.jks  
ssl.truststore.password=password  
ssl.protocol=TLS  
security.protocol=SASL_SSL  
sasl.mechanism=SCRAM-SHA-512  
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule  
required username="cliente_1" password="123456789.";
```

Donde:

- **ssl.truststore.location:** Indicará la ubicación del almacén de claves con el certificado de conexión que hemos creado.
- **ssl.truststore.password:** Contraseña del almacén de claves que hemos creado
- **ssl.protocol:** Valor fijo "TLS".
- **security.protocol:** Valor fijo "SASL_SSL".
- **sasl.mechanism:** Valor fijo "SCRAM-SHA-512" según el mecanismo definido.
- **sasl.jaas.config:** Parámetro donde se proporcionará el usuario y contraseña para la autenticación de los clientes creados en el apartado anterior, según el siguiente formato:
 - org.apache.kafka.common.security.scram.ScramLoginModule required username="**<<Usuario>>**" password="**<<Contraseña>>**";

- <<nombre_topic>>
Nombre del topic al que se desea suscribir.
- <<grupo_topic>>
Identificador del grupo de consumidores.
Cada sistema origen debe usar **un grupo único y persistente**, ya que Kafka mantiene el desplazamiento (*offset*) consumido para cada grupo.

Anexo I

Para la obtención del usuario “admin” de Kafka y poder crear el usuario, tenemos que realizar los siguientes pasos:

- Obtener el usuario y contraseña del secret “**ohien-kafka-internal-users**” de las claves:
 - o admin-username (habrá que decodificarlo)
 - o admin-password (habrá que decodificarla)

Una forma para poder obtener estos valores ya codificados sería con los siguientes comandos¹:

- o admin-username:

```
kubectl get secret -n <namespace> ohien-kafka-internal-users -o yaml |yq  
.data.admin-username | base64 -d
```

- o admin-password:

```
kubectl get secret -n <namespace> ohien-kafka-internal-users -o yaml |yq  
.data.admin-password | base64 -d
```

- Crear un fichero de propiedades llamado “*connect.properties*”, que contendrá lo siguiente:

```
security.protocol=SASL_PLAINTEXT  
sasl.mechanism=SCRAM-SHA-512  
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required u  
sername="<usuario de la clave admin-username>" password="<contraseña decodificada del  
paso anterior>";
```

¹ Debe tener instalado el paquete yq y el paquete base64

Anexo II : Subscripción Kafka UNICAS

Datos requeridos

Para llevar a cabo la subscripción la información mínima necesaria es:

Dato	config	Descripción
Bootstrap server	<u>bootstrap.servers</u>	Dirección y puerto del broker principal de Kafka que permite la conexión inicial al clúster.
Mecanismo de autenticación	<u>sasl.mechanism</u>	Tipo de autenticación utilizada para validar las credenciales.
Protocolo de seguridad	<u>security.protocol</u>	Protocolo de comunicación empleado para la conexión.
Usuario/Contraseña		Credenciales necesarias para autenticarte en el clúster.
Identificador de grupo	<u>group.id</u>	Nombre que identifica el grupo de consumidores.

Estos datos se deben solicitar el administrador del clúster. Además de estos datos, es fundamental conocer el Topic del que se quieren consumir los mensajes. En el marco del proyecto ÚNICAS se han definido los siguientes:

- Alta de paciente: En caso de alta de paciente en ÚNICAS se notificará en el topic **"alta-paciente-unicas"**.
- Modificaciones de datos de paciente: Cuando se modifiquen datos core del paciente se notificará a través del topic **"modificacion-core-paciente-unicas"**.

Formato del mensaje

En ambos casos, el formato del mensaje que se recibe en la notificación es una versión de FHIR Patient en JSON con los datos core. Como **key** en el topic se utilizará el **CIPSNS** del paciente, asegurando la lectura en orden por parte de los consumidores.

Los datos core son:

Dato core	Entrada FHIR (FHIRPath)	Cardinalidad

idPacienteUnicas	Patient.id	1..1
cipSns	Patient.identifier.where(system='urn:oid:2.16.724.4.40')	1..1
estadoPaciente	Patient.active	1..1
estadoEnrolamiento	Patient.extension.where(url='https://unicas-fhir.sanidad.gob.es/StructureDefinition/EstadoEnrolamientoPaciente').extension.where(url='estado-enrolamiento')	1..1
motivoBaja	Patient.extension.where(url='https://unicas-fhir.sanidad.gob.es/StructureDefinition/EstadoEnrolamientoPaciente').extension.where(url='motivo-baja')	0..1
momentoUnicas	Patient.extension.where(url='https://unicas-fhir.sanidad.gob.es/StructureDefinition/MomentosUnicas')	1..1
naOrigenAlta	Patient.extension.where(url='https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoAlta')	1..1
naInscripcionPaciente	Patient.extension.where(url='https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoInscripcion')	1..1
dni	Patient.identifier.where(system='urn:oid:1.3.6.1.4.1.19126.3')	0..1
cipaut	Patient.identifier.where(system.startsWith('urn:cite:'))	0..*
etiquetaSeguridad	Patient.meta.security.where(system='https://unicas-fhir.sanidad.gob.es/CodeSystem/UNICASConfidencialidad')	1..1

Estos datos se recibirán siempre, los modificados y los no modificados. El motivo de baja, dni y cipaut se enviarán solo si existen en el momento de la notificación.

Ejemplo de mensaje:

```
{
  "resourceType": "Patient",
  "id": "AC17636581806854100",
```

```
"meta": {
  "security": [
    {
      "system": "https://unicas-fhir.sanidad.gob.es/CodeSystem/UNICASConfidentiality",
      "code": "REV",
      "display": "Revocación consentimiento."
    }
  ]
},
"extension": [
  {
    "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/EstadoEnrolamientoPaciente",
    "extension": [
      {
        "url": "estado-enrolamiento",
        "valueBoolean": false
      },
      {
        "url": "motivo-baja",
        "valueCodeableConcept": {
          "coding": [
            {
              "system": "https://unicas-fhir.sanidad.gob.es/CodeSystem/MotivosBajaUNICAS",
              "code": "01",
              "display": "Revocamiento consentimiento"
            }
          ]
        }
      }
    ]
  }
]
```

```
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/MomentosUnicas",  
  "valueCodeableConcept": {  
    "coding": [  
      {  
        "system": "https://unicas-  
fhir.sanidad.gob.es/CodeSystem/CodigosMomentosUNICAS",  
        "code": "SLD",  
        "display": "Salida"  
      }  
    ],  
    "text": "Salida"  
  }  
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoAlta",  
  "valueCodeableConcept": {  
    "coding": [  
      {  
        "system": "https://unicas-  
fhir.sanidad.gob.es/CodeSystem/CodigosNodosAutonomicos",  
        "code": "ES-CT",  
        "display": "Catalunya"  
      }  
    ],  
    "text": "Catalunya"  
  }  
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoInscripcion",  
  "valueCodeableConcept": {
```

```
"coding": [  
  {  
    "system": "https://unicas-  
fhir.sanidad.gob.es/CodeSystem/CodigosNodosAutonomicos",  
    "code": "ES-CT",  
    "display": "Catalunya"  
  }  
],  
  "text": "Catalunya"  
}  
],  
"identifier": [  
  {  
    "type": {  
      "coding": [  
        {  
          "system": "http://snomed.info/sct/900000001000122104",  
          "code": "1551000122105",  
          "display": "código de identificación personal en el Sistema Nacional de Salud"  
        }  
      ]  
    },  
    "system": "urn:oid:2.16.724.4.40",  
    "value": "BBBBBBBBBHH000849"  
  },  
  {  
    "type": {  
      "coding": [  
        {  
          "system": "http://snomed.info/sct/900000001000122104?fhir_vs-  
refset/900000251000122107",
```

```
"code": "22851000122109",  
  "display": "documento nacional de identidad"  
}  
]  
},  
"system": "urn:oid:1.3.6.1.4.1.19126.3",  
"value": "11112222W"  
},  
{  
  "type": {  
    "coding": [  
      {  
        "system": "http://snomed.info/sct/900000001000122104",  
        "code": "1551000122105",  
        "display": "código de identificación personal autonómico"  
      }  
    ]  
  },  
  "system": "urn:cite:80724000015",  
  "value": "GAPU1201030008"  
},  
{  
  "type": {  
    "coding": [  
      {  
        "system": "http://snomed.info/sct/900000001000122104",  
        "code": "1551000122105",  
        "display": "código de identificación personal autonómico"  
      }  
    ]  
  },  
}
```

```
"system": "urn:cite:80724000016",  
  "value": "GAPU1201030009"  
}  
],  
"active": true  
}
```

Ejemplo de mensaje sin motivo de baja:

```
{  
  "resourceType": "Patient",  
  "id": "AC1758213130932",  
  "meta": {  
    "security": [  
      {  
        "system": "https://unicas-fhir.sanidad.gob.es/CodeSystem/UNICASConfidentiality",  
        "code": "AS-N",  
        "display": "Asistencia sanitaria sin investigacion"  
      }  
    ]  
  },  
  "extension": [  
    {  
      "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/EstadoEnrolamientoPaciente",  
      "extension": [  
        {  
          "url": "estado-enrolamiento",  
          "valueBoolean": true  
        }  
      ]  
    }  
  ]  
}
```

```
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/MomentosUnicas",  
  "valueCodeableConcept": {  
    "coding": [  
      {  
        "system": "https://unicas-fhir.sanidad.gob.es/CodeSystem/CodigosMomentosUNICAS",  
        "code": "DA",  
        "display": "Diagnostico-Asignación"  
      }  
    ],  
    "text": "Diagnostico-Asignación"  
  }  
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoAlta",  
  "valueCodeableConcept": {  
    "coding": [  
      {  
        "system": "https://unicas-fhir.sanidad.gob.es/CodeSystem/CodigosNodosAutonomicos",  
        "code": "ES-CT",  
        "display": "Catalunya"  
      }  
    ],  
    "text": "Catalunya"  
  }  
},  
{  
  "url": "https://unicas-fhir.sanidad.gob.es/StructureDefinition/NodoInscripcion",  
  "valueCodeableConcept": {
```

```
"coding": [  
  {  
    "system": "https://unicas-  
fhir.sanidad.gob.es/CodeSystem/CodigosNodosAutonomicos",  
    "code": "ES-CT",  
    "display": "Catalunya"  
  }  
],  
  "text": "Catalunya"  
}  
],  
"identifier": [  
  {  
    "type": {  
      "coding": [  
        {  
          "system": "http://snomed.info/sct/900000001000122104",  
          "code": "1551000122105",  
          "display": "código de identificación personal en el Sistema Nacional de Salud"  
        }  
      ]  
    },  
    "system": "urn:oid:2.16.724.4.40",  
    "value": "BBBBBBBBBQR759608"  
  }  
],  
"active": true  
}
```