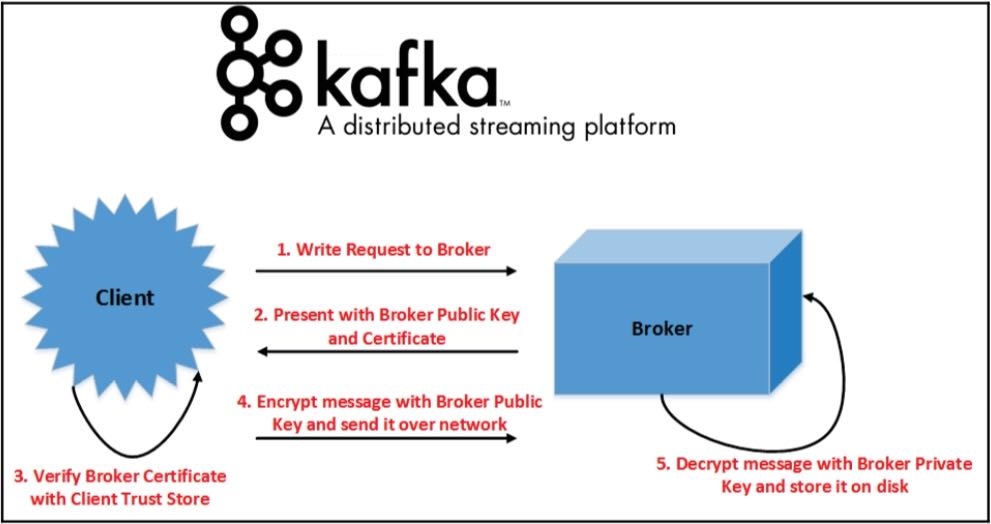
**Steps to enable SSL in Incubator Kafka**



The preceding diagram depicts how communication between broker and client is encrypted. This is valid for both producer and consumer communications. Every broker or client maintains their keys and certificates. They also maintain truststores containing certificates for authentication. Whenever certificates are presented for authentication, they are verified against certificates stored in truststores of respective components.

The incubator chart be updated to support SSL

configurationOverrides: {

"listeners": "PLAINTEXT://:9092,SSL://0.0.0.0:9091",

"ssl.client.auth": "required",

"ssl.truststore.location": "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt",

"ssl.truststore.type": "PEM",

"ssl.keystore.location": "/opt/kafka/data/kafka-server.pfx",

"ssl.keystore.type": "PKCS12",

"ssl.keystore.password": "my-super-secure-secret"

}

**Creating SSL Keys and Certificates using Certmanager, letsencrypt**

cert-manager is a service that automatically creates and manages TLS certs in Kubernetes and it is as cool as its sounds.

**Here is an overview of the steps I took to get cert-manger up and running in my Kubernetes cluster.**

1. launch cert-manager from the [official helm chart](https://github.com/kubernetes/charts/tree/master/stable/cert-manager)
2. create a letsencrypt CA [clusterIssuer](https://cert-manager.readthedocs.io/en/latest/reference/clusterissuers.html" \t "_blank) k8s resource
3. launch incubator kafka in the kubernetes cluster to be access at a TLS endpoint.
4. create a [certificate](https://cert-manager.readthedocs.io/en/latest/reference/certificates.html) object that describes how to create a TLS cert for the test app

**Details to set up cert-manager**

Here are the more detailed steps:

1. Deploy the [cert-manager helm chart](https://github.com/kubernetes/charts/tree/master/stable/cert-manager). Create the [values.yaml](https://github.com/JessicaGreben/example-nodejs/blob/master/deploy/cert-manager-setup/cert-manager-values.yaml" \t "_blank) file then run: helm-install --name my-release -f cert-manager-values.yaml cert-manager. cert-manager can be configured to automatically provision TLS certificates. This is how I set up cert-manager and therefore, I added a two settings to the values.yaml
2. Create the [letsencrypt CA cluster issuer](https://github.com/JessicaGreben/example-nodejs/blob/master/deploy/cert-manager-setup/letsencrypt-clusterissuer-staging.yaml" \t "_blank). Here I used the letsencrypt staging ACME server just for testing, once this worked, I will switch over to letsencrypt production server. I created the following file by running: kubectl create -f letsencrypt-clusterissuer-staging.yaml

apiVersion: certmanager.k8s.io/v1alpha1  
kind: ClusterIssuer  
metadata:  
 name: letsencrypt-staging  
spec:  
 acme:  
 # The ACME server URL  
 server: <https://acme-staging.api.letsencrypt.org/directory>  
 # Email address used for ACME registration  
 email: [myemail@gmail.com](mailto:myemail@gmail.com)  
 # Name of a secret used to store the ACME account private key  
 privateKeySecretRef:  
 name: letsencrypt-staging  
 # Enable the HTTP-01 challenge provider  
 http01: {}

**Creating SSL Keys and Certificates using Truststore, Keystore**

Configure all brokers in the Kafka cluster to accept secure connections from clients. Any configuration changes made to the broker will require a [rolling restart](https://docs.confluent.io/current/kafka/post-deployment.html#rolling-restart).

Enable security for Kafka brokers as described in the section below. Additionally, if you are using Confluent Control Center or Auto Data Balancer, configure your brokers for:

* [Confluent Metrics Reporter](https://docs.confluent.io/current/kafka/authentication_ssl.html#authentication-ssl-metrics-reporter)

1. Configure the truststore, keystore, and password in the server.properties file of every broker. Since this stores passwords directly in the broker configuration file, it is important to restrict access to these files via file system permissions.
2. ssl.truststore.location**=**/var/private/ssl/kafka.server.truststore.jks
3. ssl.truststore.password**=**test1234
4. ssl.keystore.location**=**/var/private/ssl/kafka.server.keystore.jks
5. ssl.keystore.password**=**test1234
6. ssl.key.password**=**test1234

Copy

Note: ssl.truststore.password is technically optional but highly recommended. If a password is not set, access to the truststore is still available, but integrity checking is disabled.

1. If you want to enable SSL for inter-broker communication, add the following to the broker properties file (it defaults to PLAINTEXT):
2. security.inter.broker.protocol**=**SSL

Copy

1. Configure the ports for the Kafka brokers to listen for client and inter-broker SSL connections. We need to configure listeners, and optionally advertised.listeners if the value is different from listeners.
2. listeners**=**SSL://kafka1:9093
3. advertised.listeners**=**SSL://0.0.0.0:9093

Copy

1. Configure both SSL ports and PLAINTEXT ports if:
   * SSL is not enabled for inter-broker communication
   * Some clients connecting to the cluster do not use SSL
2. listeners**=**PLAINTEXT://kafka1:9092,SSL://kafka1:9093
3. advertised.listeners**=**PLAINTEXT://0.0.0.0:9092,SSL://0.0.0.0:9093

Copy

*Note* that advertised.host.name and advertised.port configure a single PLAINTEXT port and are *incompatible* with secure protocols. Please use advertised.listeners instead.

1. To enable the broker to authenticate clients (2-way authentication), you need to configure all the brokers for client authentication. We recommend configuring this to required. (We discourage requested because misconfigured clients will still connect successfully and it provides a false sense of security)
2. ssl.client.auth**=**required

Copy

**Note**

If any of the SASL authentication mechanisms are enabled for a given listener, then SSL client authentication is disabled even if ssl.client.auth=required is configured, and the broker will authenticate clients only via SASL on that listener.