SHRINE 4.1.0 Chapter 8.2.2 - Configuring a Hub for Kafka

Preliminaries

Set Up a Kafka Cluster

Download Kafka: https://kafka.apache.org/downloads

Unpack to some location, like /opt/kafka, which will be referred to as <kafkaInstallationDir>.

Use Kafka Raft ("KRaft") for Kafka's controller layer. As of Kafka 3.3.1 (Oct 2022) the native KRaft is recommended over the Zookeeper controller software. KRaft improves upon performance and consolidates the server configuration files and process to 1 each (per node). NOTE: These instructions where tested with Kafka version 3.3.1.

Configure server.properties

In KRaft mode, the relevant configuration file is located at <kafkaInstallationDir>/config/kraft/server.properties.

Use a minimum server cluster size of 3 in SHRINE production environments, in which each server node functions as both a broker and controller. This is enabled by setting the *process.roles* to *broker,controller*, as noted in the Server Basics section:

server.properties

```
# The role of this server. Setting this puts us in KRaft mode
process.roles=broker,controller
```

---

node.id is one of the few parameters that **must** be unique to each server node. 1, 2, 3, etc.

---

controller.quorum.voters is a comma-separated list of all server nodes' id, hostname or IP, and port, and can be set consistently across server nodes if using FQDNS:

server.properties

```
controller.quorum.voters=1@kafka-1.yourDomain.com:9093,2@kafka-2.yourDomain.com:9093,3@kafka-3.yourDomain.com:
9093
```

Assuming the server nodes all reside in the same private IP space, to ensure traffic remains within that space you may wish to use local DNS to map hostnames to localhost and private IP addresses. Alternatively, use localhost and private IPs in this parameter.

---

server.properties provides examples of possible configurations of listeners, protocols, and ports. We recommend one for the broker function and one for the controller function as follows:

server.properties

```
listeners=BROKER://0.0.0.0:9092,CONTROLLER://0.0.0.0:9093
inter.broker.listener.name=BROKER
advertised.listeners=BROKER://<thisServersFQDN>:9092
listener.security.protocol.map=BROKER:SASL_SSL,CONTROLLER:SASL_PLAINTEXT
```

where *advertised.listeners* is the second parameter (after *node.id*) that is unique to each server node.

SASL_SSL on the broker listener is required to enforce client/server user authentication and authorization since that traffic is traversing the public internet. SASL_SSL may also be enabled for the controller listener with properly configured keystores and truststores; however if the server nodes communicate exclusively in private network space (as described above), then SASL_PLAINTEXT may be considered sufficient.

---

log.dirs specifies not the location of server logs (that is in <kafkaInstallationDir/logs>, but actual topic data, so provide a reliable location outside the default /tmp; for example /var/opt/kafka.

Set the SASL mechanism parameters to PLAIN:

server.properties

```
sasl.enabled.mechanisms=PLAIN
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.mechanism.controller.protocol=PLAIN
```

Set the authorizer (there are multiple to choose from; StandardAuthorizer is recommended for KRaft mode):

server.properties

```
authorizer.class.name=org.apache.kafka.metadata.authorizer.StandardAuthorizer
```

Come up with an admin username, e.g. “kafka-admin”, to be used for admin tasks from the SHRINE hub instance and for inter-broker communication. We’ll create the user later. Set it as a super user:

server.properties

```
super.users=User:kafka-admin
```

Configure Kafka Users

Kafka supports SASL authentication with a few possible user-management mechanisms, but version 3.3.1 in KRaft mode only makes available the PLAIN mechanism (which is *not* to be confused with the insecure PLAINTEXT protocol). The more full-featured SCRAM mechanism will be available for KRaft in a future release. User authentication via PLAIN mechanism consults a static user list in <kafkaInstallationDir>/config/kraft/kafka_server_jaas.conf file present on each server node.

Define the admin user/password to be used for admin tasks and for inter-broker communication, as well as one user/password for the SHRINE hub and one for each SHRINE node in the network.

kafka_server_jaas.conf

```
KafkaServer {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    serviceName="kafka"
    username="kafka-admin”
    password="<yourKafkaAdminPassword>"
    user_kafka-admin=”<yourKafkaAdminPassword>"
    user_<yourShrineHubUser>=”<yourShrineHubUserPassword>”
    user_<shrineNode1User>=”<shrineNode1UserPassword>”
    user_<shrineNode2User>=”<shrineNode2UserPassword>”;
};
```

The *username* and *password* lines define the user to be used *by this broker* for inter-broker communication. All lines beginning with *user_* define the users that can be authenticated by clients (including other brokers in the inter-broker communication context). When sharing SHRINE user credentials with SHRINE site admins, be sure to use a secure transfer mechanism.

Provide the Kafka application the path to this file using the KAFKA_OPTS environment variable:

```
export KAFKA_OPTS="-Djava.security.auth.login.config=<kafkaInstallationDir>/config/kraft/kafka_server_jaas.conf"
```

Changes to this file (user additions or removals) requires a Kafka process restart on each node. This is one drawback to the PLAIN mechanism which will be alleviated when SCRAM becomes available for KRaft.

Create Server Keystores and Truststores

In order to secure traffic through the internet with TLS/SSL, Kafka requires client/server authentication via public key infrastructure (PKI). Each Kafka **server** needs a keystore for identifying itself, while each **client *and* server** needs a truststore to authenticate servers. As the Kafka administrator you may choose to have all server certificates signed by a true Certificate Authority (CA), or to manage a private CA within your organization and use it for signing.

In either case, each Kafka server node's **keystore** must store a unique private key (always kept secure) and certificate (which gets signed). Each server node's **truststore** must store a list of all signed server certificates, or alternatively a CA's own cert, in order to let server nodes behave as logical clients during inter-broker communication. The servers' truststore contents can be identical to that of clients. One benefit of managing a private CA is enabling all client and server truststores to identically contain only the CA's cert, in effect telling all systems in the network to trust the CA *and every cert signed by it*. Additionally this enables server nodes to join or leave the cluster without truststores needing cert addition or revocation. See here for comprehensive documentation on PKI architecture and keystore management in Kafka:

https://kafka.apache.org/33/documentation.html#security_ssl

https://docs.confluent.io/platform/current/security/security_tutorial.html

Create keystores and truststores in PKCS12 format. When creating, you will be prompted for passwords. Add the file locations and passwords to the end of server.properties:

server.properties

```
ssl.key.password=<thisServersKeystorePassword>
ssl.keystore.location=<path/to/this/servers/kafka_server_keystore.pkcs12>
ssl.keystore.password=<thisServersKeystorePassword>
ssl.truststore.location=<path/to/shared/kafka_server_truststore.pkcs12>
ssl.truststore.password=<sharedServerTruststorePassword>
```

Format the Kafka storage directories

Now that server.properties is complete, format the Kafka storage directories.

On **one** server node, generate a cluster UUID:

```
<kafkaInstallationDir>/bin/kafka-storage.sh random-uuid
```

On **all** server nodes, format the storage directories using that uuid:

```
<kafkaInstallationDir/bin/kafka-storage.sh format --cluster-id <uuid> --config <kafkaInstallationDir>/config
/kraft/server.properties
```

The storage directory is provided to Kafka in server.properties as *log.dirs.* Kafka will create the directory if it does not exist.

Run Kafka

On all server nodes, run:

```
<kafkaInstallationDir>/bin/kafka-server-start.sh -daemon <kafkaInstallationDir>/config/kraft/server.properties
```

Configure the Hub's shrine.conf

In tomcat's shrine.conf in the hub, turn off the in-tomcat messaging by setting shrine.hub.messagequeue.blockingWebApi.enabled to it's default false value by removing it from the hub block:

shrine.conf

```
shrine {
...
  hub {
    create = true
  }//hub
...
}//shrine
```

Add your hub's Kafka user (set in the Kafka servers' *kafka_server_jaas.conf)* and your client truststore location to shrine.conf:

shrine.conf

```
shrine {
...
  kafka {
    sasl.jaas.username = "yourShrineHubUser"
    ssl.truststore.location = "path/to/your/kafka_client_truststore.pkcs12"
  }
...
}//shrine
```

And your hub's Kafka user password as well as the client truststore password to password.conf:

password.conf

```
shrine.kafka.sasl.jaas.password = "yourShrineHubUserPassword"
shrine.kafka.ssl.truststore.password = "clientTruststorePassword"
```

Configure shrineLifecycle tool

The shrine-network-lifecycle-tool (aka "shrineLifecycle") needs the admin Kafka credentials to create, modify, and delete Kafka topics, as well as your client truststore. Add them to shrine-network-lifecycle-tool's conf files:

override.conf

```
shrine.kafka.sasl.jaas.username = "kafka-admin"
shrine.kafka.ssl.truststore.location = "path/to/your/kafka_client_truststore.pkcs12"
```

password.conf

```
shrine.kafka.sasl.jaas.password = "yourKafkaAdminPassword"
shrine.kafka.ssl.truststore.password = "clientTruststorePassword"
```

Note: it's safe to mix the "curly-bracket" and "dot" syntax styles in the same file.

In network.conf, add a Kafka section with the specifics to share with downstream nodes:

```
shrine {
  network {
    network {
      name = "Network Name"
      hubQueueName = "hub"
      adminEmail = "yourEmail@yourhospital.edu"
      momId = "yourShrineHubKafkaUser"
      kafka = {
            networkPrefix = "NetworkName"
        bootstrapServers = "kafka-1.yourDomain.com:9092,kafka-2.yourDomain.com:9092,kafka-3.yourDomain.com:9092"
        securityProtocol = "SASL_SSL"
        securityMechanism = "PLAIN"
      }
    }
    nodes = [
      {
        name = "Network Name hub"
        key = "hub-qep"
        userDomainName = "network-hub"
        queueName = "hubQep"
        sendQueries = "false"
        adminEmail = "yourEmail@yourhospital.edu"
        momId = "yourShrineHubKafkaUser"
      }
    ]
  }
}
```

Choose a network prefix. This will be prepended to queue names to enable the scenario of multiple networks on the same Kafka cluster.

Note that the network's momId is the same as hub's node momId.

The *bootstrapServers* parameter is a list of **some** or **all** server nodes running in the cluster. Once connection is made to one of them, the cluster's *controller.quorum.voters* setting takes precedence. At least 2 *bootstrapServers* are recommended, but the real number of server nodes can grow or shrink without clients needing to know.