

SHRINE 4.1.0 Chapter 8 - SHRINE's Configuration

SHRINE gets its configuration from a configuration file named **shrine.conf** in `/opt/shrine/tomcat/lib`. If you are installing a downstream (non-hub) node, copy the `shrine-setup/qep-and-adapter-shrine.conf` file from `shrine-setup.zip` to tomcat's lib directory.

```
cp shrine-setup/qep-and-adapter-shrine.conf /opt/shrine/tomcat/lib/shrine.conf
```



Hub Admins Only

If you are creating a hub use `hub-and-qep-shrine.conf` - and skip to [SHRINE 4.1.0 Chapter 8.2 - Configuring a Hub](#).

In this guide, we will refer to this file often and will go more in detail on configuring this file in the later chapters). Here is the example `shrine.conf` file from `shrine-setup.zip`. You will need to customize it for your own node on your network. In the example below, the first four lines in the shrine section define values for parameters that are used throughout the configuration file.

shrine.conf

```
shrine {

    shrineHubBaseUrl = "https://shrine-hub.faraway.com:6443" //The shrine hub's URL as observed from this tomcat
server
    i2b2BaseUrl = "http://i2b2.example.com:9090" //The local i2b2's URL as observed from this tomcat server
    i2b2Domain = "yourDomain" //a value for your domain that clearly identifies your institution.
    i2b2ShrineProjectName = "SHRINE"

    nodeKey = "testNode" //node key to get information from the hub about this node.

    //shrineDatabaseType = "mysql" // "mysql" by default. It can be "sqlserver" "mysql" or "oracle"

    webclient {
        siteAdminEmail = "shrine-admin@example.com"
    }

    hiveCredentials {
        username = "demo"
        crcProjectId = "Demo"
    }//HiveCredentials

    steward {
        emailDataSteward {
            //provide the email address of the shrine node system admin, to handle bounces and invalid addresses
            from = "shrine-admin@example.com"
            //provide the email address of the shrine node system admin, to handle bounces and invalid addresses
            to = "shrine-steward@example.com"
            //provide the externally-reachable URL for the data steward
            externalStewardBaseUrl = "https://shrine.example.com:6443/shrine-api/shrine-steward"
        }
    }//steward
}//shrine
```

(Optional) To require users to be a member of a specific PM cell project, add the following to `shrine.conf`:

```
shrine.authenticate.pmProjectName = "<enter PM cell project name here>"
```

All passwords are stored in `password.conf` file in `/opt/shrine/tomcat/lib` instead of `shrine.conf`.

password.conf

```
shrine.hiveCredentials.password = "changeit"
```



nodeKey parameter

Contact your hub administrator for your nodeKey value. The nodeKey value is used to identify your node from the Hub. This value must be a unique alphanumeric identifier in your network.



i2b2Domain parameter

Choose a custom value for your domain parameter that clearly identifies your institution. Do not leave it at its default value.

Configure For AWS SQS

Create an AWS Account

Create an AWS account if you do not already have one. See <https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>

Create an AWS user for Your SHRINE Node

Create a user for your SHRINE node's tomcat in the AWS console. Give it a name that will be distinct both at your institution and this shrine network.

Do not give it console access.

Do not make it a member of a group, copy permissions, or attach permissions. You'll configure permissions in a moment for tomcat to use AWS SQS.

Create an Access Key and Configure password.conf

Create an access key and secret for the tomcat user - not the admin user - as described in https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html.

Tomcat will need the its AWS SQS credentials to send and receive messages. Add your access ID and secret to tomcat's password.conf:

password.conf

```
shrine.aws.accessKeyId = "NODEAWSKEYID" //the node's AWS access key id - usually all capitals and numbers
shrine.aws.secretAccessKey = "nodeAwsSecretKey" //the node's AWS secret key - very long, mixed case letters and numbers
```

Share the User ID with the Hub Admin

In the IAM > User > Summary section, find the ARN for the tomcat user, which identifies your specific AWS account *and* IAM user identity. It will look something like this: arn:aws:iam::9876543210:user/yourHospital-Shrine.

Send this to your hub admins so that they can add your node to the network. It is not secret, so sending in the clear is fine.

Run shrineDownstream setMomUserPolicy

The hub admins will reply with two ARNs: one for the hub's inbound AWS SQS queue, and one for your node's inbound AWS SQS queue.

Download and unzip the shrineDownstream tool from <https://repo.open.catalyst.harvard.edu/nexus/content/groups/public/net/shrine/downstream-setup-tool/4.1.0/downstream-setup-tool-4.1.0-dist.zip>

Create an access key and secret for your admin user - not the tomcat user - as described in https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html.

The shrineDownstream tool will need AWS credentials to replace the tomcat user's permissions to allow it to send and receive messages using those two queues. Add your admin access ID and secret to the shrineDownstream tool's password.conf:

password.conf

```
shrine.aws.accessKeyId = "NODEADMINAWSKEYID" //the node's AWS access key id - usually all capitals and numbers
shrine.aws.secretAccessKey = "nodeAdminAwsSecretKey" //the node's AWS secret key - very long, mixed case letters and numbers
```

Run the command to set the policies:

```
./shrineDownstream setMomUserPolicy yourHospital-Shrine hubQueueArn="arn:aws:sqs:us-east-1:1234567890:network-hub" nodeQueueArn="arn:aws:sqs:us-east-1:1234567890:best-hospital"
```

That will set the policies for your tomcat user to something like:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MayReceiveShrine",
      "Effect": "Allow",
      "Action": [
        "SQS:ChangeMessageVisibility",
        "SQS:DeleteMessage",
        "SQS:ReceiveMessage",
        "SQS:GetQueueUrl"
      ],
      "Resource": "arn:aws:sqs:us-east-1:1234567890:network-hub"
    }
  ]
}
```

and

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MaySendShrine",
      "Effect": "Allow",
      "Action": [
        "sqs:GetQueueUrl",
        "sqs:SendMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-1:1234567890:best-hospital"
    }
  ]
}
```

Configure For Kafka

Receive a Kafka User Name and Password

The hub admin will create an account on the Kafka server for your node, and send you the user name and password via a secure channel.

Add the user name to your shrine.conf:

shrine.conf

```
shrine {
  ...
  kafka {
    sasl.jaas.username = "yourKafkaUserName"
  }
  ...
} //shrine
```

Add the password to your password.conf:

password.conf

```
shrine.kafka.sasl.jaas.password = "yourKafkaUserPassword"
```

Create a Kafka client certificate truststore

In order to secure traffic through the internet with TLS/SSL, Kafka requires clients to authenticate servers via public key infrastructure (PKI). Each client needs a client truststore, in PKCS12 format, containing a list of individual server certificates signed by a Certificate Authority (CA), or alternatively the CA's cert itself. Ask the hub admin for the certificate(s), and import them each with Java *keytool*:

```
keytool -keystore kafka_client_truststore.pkcs12 -alias <name of cert> -import -file <certificate-file>
```

This will create the truststore file if it does not exist. You will be prompted for a password, despite the truststore containing no secret material since certificates are public.

Add the truststore's location and password to the same two sections as your Kafka user credentials:

shrine.conf

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

password.conf

```
shrine.kafka.ssl.truststore.password = "yourClientTruststorePassword"
```