Deployment Guide

Configuring Security Certificates for Jetty

4/30/2025

Contents

- 1 Configuring Security Certificates for Jetty
  - 1.1 Loading Certificate for SSL

Configuring Security Certificates for Jetty

Loading Certificate for SSL

The Jetty web server supplied with the Co-browse solution includes a pre-configured, self-signed certificate. This allows you to use HTTPS out of the box in a lab or demo environment, with the restrictions described in Basic Instrumentation. To make the self-signed certificate trusted for Co-browse proxy service (server):

```
keytool -import -trustcacerts -keystore <JAVA_HOME>[/jre]/lib/security/cacerts \
   -storepass changeit -noprompt -alias <your_cert_alias> -file <full_path>/cert_file.crt
```

For a production environment, you should use a certificate issued by a third-party Certificate Authority. The procedures on this page provide examples of ways to load SSL certificates and configure Jetty. These examples may vary depending on your environment.

Load an SSL Certificate and Private Key into a JSSE keystore

> **Important**
> In a development environment, you can use self-signed certificates, but in a production environment you should use a certificate issued by a third-party trusted Certificate Authority, such as VeriSign.

Prerequisites

- An SSL certificate, either generated by you or issued by a third-party Certificate Authority. For more information on generating a certificate, see Configuring SSL/TLS in Jetty

Start of procedure

1. Depending on your certificate format, do **one** of the following:
   - If your certificate is in PEM form, you can load it to a JSSE keystore with the keytool using the following command:
     ```
     keytool -keystore <keystore> -importcert -alias <alias> -file <certificate_file>
      -trustcacerts
     ```
     Where:

     `<keystore>` is the name of your JSSE keystore.

     `<alias>` is the unique alias for your certificate in the JSSE keystore.

`<certificate_file>` is the name of your certificate file. For example, `jetty.crt`.

- If your certificate and key are in separate files, you must combine them into a PKCS12 file before loading it to a keystore.

  1. Use the following command in openssl to combine the files:
     `openssl pkcs12 -inkey <private_key> -in <certificate> -export -out <pkcs12_file>`

     Where:

     `<private_key>` is the name of your private key file. For example, `jetty.key`.

     `<certificate>` is the name of your certificate file. For example, `jetty.crt`.

     `<pkcs12_file>` is the name of the PKCS12 file that will be created. For example, `jetty.pkcs12`.

  2. Load the PKCS12 file into a JSSE keystore using keytool with the following command:
     `keytool -importkeystore -srckeystore <pkcs12_file> -srcstoretype <store_type> -destkeystore <keystore>`

     Where:

     `<pkcs12_file>` is the name of your PKCS12 file. For example, `jetty.pkcs12`.

     `<store_type>` is the file type you are importing into the keystore. In this case, the type is PKCS12.

     `<keystore>` is the name of your JSSE keystore.

> **Important**
> You will need to set two passwords during this process: keystore and truststore. Make note of these passwords because you will need to add them to your Jetty SSL configuration file.

End of procedure

Next Steps

- Configure Jetty

Configure Jetty

Prerequisites

- You have completed Load an SSL Certificate and Private Key into a JSSE keystore

Start of procedure

1. Open the Jetty SSL configuration file in a text editor: `<jetty_installation>/etc/jetty-ssl-context.xml`.

2. Find the <Configure id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory"> element and update the passwords:

```
<Configure id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory">
  <Set name="KeyStorePath"><Property name="jetty.base" default="." />/<Property
name="jetty.keystore" default="etc/keystore"/></Set>
  <Set name="KeyStorePassword"><Property name="jetty.keystore.password"
default="OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4"/></Set>
  <Set name="KeyManagerPassword"><Property name="jetty.keymanager.password"
default="OBF:1u2u1wml1z7s1z7a1wnl1u2g"/></Set>
  <Set name="TrustStorePath"><Property name="jetty.base" default="." />/<Property
name="jetty.truststore" default="etc/keystore"/></Set>
  <Set name="TrustStorePassword"><Property name="jetty.truststore.password"
default="OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4"/></Set>
  <Set name="EndpointIdentificationAlgorithm"></Set>
  <Set name="NeedClientAuth"><Property name="jetty.ssl.needClientAuth"
default="false"/></Set>
  <Set name="WantClientAuth"><Property name="jetty.ssl.wantClientAuth"
default="false"/></Set>
  <Set name="ExcludeCipherSuites">
    <Array type="String">
    <!-- List of vulnerable cipher suites. Left it as default-->
    </Array>
  </Set>
  <New id="sslHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
    <Arg><Ref refid="httpConfig"/></Arg>
    <Call name="addCustomizer">
    <Arg><New class="org.eclipse.jetty.server.SecureRequestCustomizer"/></Arg>
    </Call>
  </New>
</Configure>
```

Note: You can run Jetty's password utility to obfuscate your passwords. See http://www.eclipse.org/jetty/documentation/current/configuring-security-secure-passwords.html.

3. Save your changes.

End of procedure

Choosing a Directory for the Keystore

The keystore file in the example above is given relative to the Jetty home directory. For production, you should keep your keystore in a private directory with restricted access. Even though the keystore has password, the password may be configured into the runtime environment and is vulnerable to theft.

You can now start Jetty the normal way (make sure that jcert.jar, jnet.jar and jsse.jar are on your classpath) and SSL can be used with a URL, such as `https://<your_IP>:8743/`