This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Deployment Guide

Loading Certificate for SSL

12/14/2025

Loading Certificate for SSL

Contents

The procedures on this page provide examples of ways to load SSL certificates and configure Jetty. These examples may vary depending on your environment.

Note: You must use the Java Development Kit version 1.6.0_29 or higher to support the JSSE keystore.

Load an SSL Certificate and Private Key into a JSSE keystore

Note: 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 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

- 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> -import -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.

Note: 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

Next Steps

- Configure Jetty

Configure Jetty

Prerequisites

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

Start

1. Open the Jetty SSL configuration file in a text editor: `<jetty_installation>/etc/jetty-ssl.xml`.
2. Add a new "addConnector" section to the file, with the following information:

```
<Call name="addConnector">
     <Arg>
       <New class="org.mortbay.jetty.security.SslSocketConnector">
         <Set name="Port">8443</Set>
         <Set name="maxIdleTime">30000</Set>
         <Set name="keystore">
                    <SystemProperty name="jetty.home" default="." />
                    /etc/keystore
                </Set>
         <Set name="password">OBF:<obfuscated_truststore_password></Set>
         <Set name="keyPassword">OBF:<obfuscated_keystore_password></Set>
         <Set name="truststore">
                    <SystemProperty name="jetty.home" default="." />
                    /etc/keystore
                </Set>
         <Set name="trustPassword">OBF:<obfuscated_truststore_password></Set>
       </New>
         </Arg>
</Call>
```

Note: You can run Jetty's password utility to obfuscate your passwords. See http://docs.codehaus.org/display/JETTY/Securing+Passwords.

3. Save your changes.

End

Choosing a Directory for the Keystore

The keystore file in the example above is given relative to the Jetty home directory. For production, choose a private directory with restricted access to keep your keystore in. Even though it has a password on it, 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 like: https://localhost:8443/

Setting the Port for HTTPS

Remember that the default port for HTTPS is 443 not 80, so change 8443 to 443 if you want to be able to use URLs without explicit port numbers. For a production site it normally makes sense to have an HttpListener on port 80 and a SunJsseListener on port 443. Because these are privileged ports, you might want to use a redirection mechanism to map port 80 to 8080 and 443 to 8443, for example. The most common mistake at this point is to try to access port 8443 with HTTP rather than HTTPS.

Redirecting HTTP requests to HTTPS

To redirect HTTP to HTTPS, the webapp should indicate it needs CONFIDENTIAL or INTEGRAL connections from users. You need to tell the plain HTTP connector that if users try to access that webapp with plain HTTP, they should be redirected to the port of your SSL connector (the "confidential port"):

```
<Call name="addConnector">
  <Arg>
     <New class="org.mortbay.jetty.nio.SelectChannelConnector">
        <Set name="port">8080</Set>
        <Set name="maxIdleTime">30000</Set>
        <Set name="Acceptors">2</Set>
        <Set name="confidentialPort">443</Set>
     </New>
  </Arg>
</Call>
```

Next Steps

- Configure Java

Configure Java

Prerequisites

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

Start

1. Navigate to your installation directory for the Backend Server and open the launcher.xml file with a text editor.
2. Add the following parameters:

```
<parameter name="javax.net.ssl.trustStore" displayName="javax.net.ssl.trustStore"
mandatory="false">
   <description><![CDATA[]]></description>
   <valid-description />
   <effective-description />
   <format type="string" default="<cacerts_filepath>" />
   <validation />
</parameter>
<parameter name="javax.net.ssl.trustStorePassword"
displayName="javax.net.ssl.trustStorePassword" mandatory="false">
   <description><![CDATA[]]></description>
   <valid-description />
   <effective-description />
   <format type="string" default="<truststore_password>" />
   <validation />
</parameter>
<parameter name="javax.net.ssl.keyStore" displayName="javax.net.ssl.keyStore"
mandatory="false">
   <description><![CDATA[]]></description>
   <valid-description />
   <effective-description />
   <format type="string" default="<keystore_filepath>" />
   <validation />
</parameter>
<parameter name="javax.net.ssl.keyStorePassword"
displayName="javax.net.ssl.keyStorePassword" mandatory="false">
   <description><![CDATA[]]></description>
   <valid-description />
   <effective-description />
   <format type="string" default="<keystore_password>" />
   <validation />
</parameter>
<parameter name="javax.net.ssl.keyAlias" displayName="javax.net.ssl.keyAlias"
mandatory="false">
   <description><![CDATA[]]></description>
   <valid-description />
   <effective-description />
   <format type="string" default="<alias>" />
   <validation />
</parameter>
```

3. Save your changes.
4. Repeat steps 1-3 for the Frontend Server.
5. Save your changes.

End