Jetty93

Using Jetty 9.3

These pages are examples and do not reflect any normative requirements or assumptions on the part of the IdP software and may be a mix of suggestions from both the project team and deployers. You should take any of this advice with a grain of local salt and consider general security /deployment considerations appropriate to the use of web software in your local environment.

The official information about containers and versions we support is solely maintained on the SystemRequirements page. If you wish to operate without complete responsibility for your Java servlet container, you may consider the Windows package we provide that includes an embedded container.

The following conventions are used this document:

- /opt/shibboleth-idp is used to indicate that an absolute path to the IdP installation directory is required
- `idp.home` refers to the IdP installation directory (as specified during the installation process)
- `JETTY_HOME` refers to the location of the Jetty installation (jetty-dist-$VERSION)
- `JETTY_BASE` refers to the directory containing your deployment-specific Jetty configuration files
- All paths are relative to `JETTY_BASE` unless otherwise noted

We strongly recommend placing all IdP-specific Jetty configuration under `JETTY_BASE` to facilitate Jetty upgrades.

Version Notes

The latest stable version should be used.

Note that Java 8 is required for this version.

A bug was found, reported, and fixed that requires a workaround when changing the TLS keystore type to PKCS12. Our documentation accounts for this bug.

This release includes a number of configuration changes from earlier releases, mostly refactoring and property renaming. Starting from scratch is advisable if upgrading from 9.2.

JETTY_BASE Layout

A typical `JETTY_BASE` directory for the IdP webapp contains the following files, each of which will be described in the following sections.

- start.ini
- start.d/http.ini
- start.d/https.ini
- start.d/ssl.ini
- etc/jetty-ssl-context.xml
- etc/jetty-requestlog.xml (*optional*)
- etc/jetty-rewrite.xml (*optional*)
- lib/ext/jetty9-dta-ssl-1.0.0.jar (*optional*)
- lib/logging/jcl-over-slf4j-1.7.7.jar (*optional*)
- lib/logging/logback-access-1.1.2.jar (*optional*)
- lib/logging/logback-classic-1.1.2.jar (*optional*)
- lib/logging/logback-core-1.1.2.jar (*optional*)
- lib/logging/slf4j-api-1.7.12.jar (*optional*)
- resources/logback.xml (*optional*)
- resources/logback-access.xml (*optional*)
- webapps/idp.xml
- tmp (*optional*)

Required Configuration

Configure Jetty Modules and JVM Settings

File(s): *start.ini*

Create a *JETTY_BASE/start.ini* file with the following contents.

start.ini

```
# Required Jetty modules
--module=server
--module=deploy
--module=annotations
--module=resources
--module=logging
--module=requestlog
--module=servlets
--module=jsp
--module=jstl
--module=ext
--module=plus
--module=rewrite

# Allows setting Java system properties (-Dname=value)
# and JVM flags (-X, -XX) in this file
# NOTE: spawns child Java process
--exec

# Bypass file validation for the SSL module, to work around a bug in Jetty 9.3.X
--skip-file-validation=ssl

# Uncomment if IdP is installed somewhere other than /opt/shibboleth-idp
#-Didp.home=/path/to/shibboleth-idp

# Alternate garbage collector that reduces memory needed for larger metadata files
-XX:+UseG1GC

# Maximum amount of memory that Jetty may use, at least 1.5G is recommended
# for handling larger (> 25M) metadata files but you will need to test on
# your particular metadata configuration
-Xmx1500m

# Maximum amount of memory allowed for the JVM permanent generation (Java 7 only)
-XX:MaxPermSize=128m
```

Configure HTTP Connectors

File(s): */opt/shibboleth-idp/credentials/idp-browser.p12*, *etc/jetty-ssl-context.xml*, *start.d/http.ini, start.d/https.ini, start.d/ssl.ini*

Jetty listens on ports 8080 and 8443 for user-facing web traffic by default. In order to serve requests at the default HTTP/HTTPS ports one of the following is required.

1. Change the ports to 80/443 in the jetty property files and use the setuid extension to support listening on the privileged ports as a non-root user.
2. Use a port forwarding approach (load balancer, iptables rules, etc).

Copy the following files from `JETTY_HOME/demo-base/start.d` to `JETTY_BASE/start.d`:

1. http.ini
2. https.ini
3. ssl.ini

If you elect to change the default listening ports, modify the `jetty.http.port` property in *http.ini* and `jetty.ssl.port` in *https.ini* accordingly.

Modify *ssl.ini* so that it contains the following properties. Note that these are named differently from the equivalent properties used in 9.2.

```
jetty.sslContext.keyStorePath=/opt/shibboleth-idp/credentials/idp-browser.p12
jetty.sslContext.keyStoreType=PKCS12
jetty.sslContext.keyStorePassword=thepasswordgoeshere
```

If you have deployed the IdP to an alternate location, change the keystore path accordingly. The *idp-browser.p12* file is a PKCS12 file containing the X.509 certificate and private key used to secure the HTTPS channel that users access during authentication and other browser-based message exchanges involving the IdP. This is generally the one you get from a browser-compatible CA.

Create the following file using the example below (it's based on the version that comes with Jetty):

1. JETTY_BASE/etc/jetty-ssl-context.xml

jetty-ssl-context.xml

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure_9_0.dtd">
<Configure id="Server" class="org.eclipse.jetty.server.Server">
  <!-- ============================================================= -->
  <!-- TLS context factory without client auth                       -->
  <!-- ============================================================= -->
  <New id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory">
    <Set name="KeyStorePath"><Property name="jetty.sslContext.keyStorePath" /></Set>
    <Set name="KeyStoreType"><Property name="jetty.sslContext.keyStoreType" /></Set>
    <Set name="KeyStorePassword"><Property name="jetty.sslContext.keyStorePassword" /></Set>
    <!-- This is a tweak to work around a bug in Jetty when using the PKCS12 keystore type. -->
    <Set name="TrustStoreType"><Property name="jetty.sslContext.keyStoreType" /></Set>
    <Set name="EndpointIdentificationAlgorithm"></Set>
    <Set name="NeedClientAuth"><Property name="jetty.sslContext.needClientAuth" default="false"/></Set>
    <Set name="WantClientAuth"><Property name="jetty.sslContext.wantClientAuth" default="false"/></Set>
    <Set name="useCipherSuitesOrder"><Property name="jetty.sslContext.useCipherSuitesOrder" default="true"/><
/Set>
    <Set name="renegotiationAllowed">false</Set>
    <Set name="excludeProtocols">
      <Array type="String">
                <Item>SSL</Item>
        <Item>SSLv2</Item>
        <Item>SSLv3</Item>
      </Array>
    </Set>
    <Set name="IncludeCipherSuites">
      <Array type="String">
        <Item>TLS_ECDHE.*</Item>
        <Item>TLS_RSA.*</Item>
      </Array>
    </Set>
    <Set name="ExcludeCipherSuites">
      <Array type="String">
        <Item>.*NULL.*</Item>
        <Item>.*RC4.*</Item>
        <Item>.*MD5.*</Item>
        <Item>.*DES.*</Item>
        <Item>.*DSS.*</Item>
      </Array>
    </Set>
 </New>
</Configure>
```

Configure IdP Context Descriptor

File(s): *webapps/idp.xml*

In order to deploy the IdP, Jetty must be informed of the location of the IdP war file. This file is called a context descriptor and the recommended content is provided below. Since the following example relies upon the idp.home System Property being set, it must either be defined in start.ini, or included in the command line string used to start Jetty.

Note this file controls the context path to which the application is deployed, which is **/idp** in the following configuration block.

idp.xml

```
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
  <Set name="war"><SystemProperty name="idp.home"/>/war/idp.war</Set>
  <Set name="contextPath">/idp</Set>
  <Set name="extractWAR">false</Set>
  <Set name="copyWebDir">false</Set>
  <Set name="copyWebInf">true</Set>
</Configure>
```

Recommended Configuration

Jetty Logging

File(s): *etc/jetty-requestlog.xml, resources/logback.xml, resources/logback-access.xml*

The recommended approach is to use logback for all Jetty logging. The logback and slf4j libraries are needed to support this configuration and must be copied into `JETTY_BASE/lib/logging`.

1. From the slf4j distribution, copy in *slf4j-api-version.jar*
2. From the logback distribution, copy in *logback-classic-version.jar*, *logback-core-version.jar*, and *logback-access-version.jar*
3. Configure Jetty to use logback for request logging by creating *JETTY_BASE/etc/jetty-requestlog.xml* with the following content:

   jetty-requestlog.xml

   ```
   <?xml version="1.0"?>
   <!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure_9_0.dtd">
   <!-- =============================================================== -->
   <!-- Configure the Jetty Request Log                                 -->
   <!-- =============================================================== -->
   <Configure id="Server" class="org.eclipse.jetty.server.Server">
     <Set name="RequestLog">
       <New id="RequestLog" class="ch.qos.logback.access.jetty.RequestLogImpl">
         <Set name="fileName"><Property name="jetty.base" default="." />/resources/logback-access.xml</Set>
       </New>
     </Set>
     <Ref refid="RequestLog">
       <Call name="start" />
     </Ref>
   </Configure>
   ```

4. Configure logging policy for Jetty internals logging and request logging. Sample logback configuration files are provided for convenience.

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true">
    <appender name="jetty" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <File>${jetty.base}/logs/jetty.log</File>

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <FileNamePattern>${jetty.base}/logs/jetty-%d{yyyy-MM-dd}.log.gz</FileNamePattern>
        </rollingPolicy>

        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <charset>UTF-8</charset>
            <Pattern>%date{HH:mm:ss.SSS} - %level [%logger:%line] - %msg%n</Pattern>
        </encoder>
    </appender>

    <root level="INFO">
        <appender-ref ref="jetty" />
    </root>
    <logger name="org.springframework" level="OFF" />
    <logger name="ch.qos.logback" level="WARN" />
</configuration>
```

logback-access.xml

```
<configuration>
  <statusListener class="ch.qos.logback.core.status.OnConsoleStatusListener" />
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${jetty.base}/logs/access.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>${jetty.base}/logs/access-%d{yyyy-MM-dd}.log.gz</fileNamePattern>
    </rollingPolicy>
    <encoder>
      <pattern>combined</pattern>
    </encoder>
  </appender>

  <appender-ref ref="FILE" />
</configuration>
```

Temporary Files

Jetty will use `/tmp` as a staging area for unpacking the warfile, and if you have cron jobs sweeping that for old files, your IdP can be disrupted. You will probably want to create `JETTY_BASE/tmp`, and add the following configuration directive to *JETTY_BASE/start.ini*:

```
-Djava.io.tmpdir=tmp
```

Disable Directory Indexing

Jetty has vulnerabilities related to directory indexing (sigh) so we suggest disabling that feature at this point. There are a few different ways this can be done (see https://webtide.com/indexing-listing-vulnerability-in-jetty/), but one method that's fairly self-contained within the IdP footprint is to modify web.xml (i.e. copy the original version from idp.home/dist/webapp/WEB-INF/web.xml to idp.home/edit-webapp/WEB-INF/web.xml) and then rebuild the war file.

web.xml addition

```
<servlet>
  <servlet-name>default</servlet-name>
  <servlet-class>org.eclipse.jetty.servlet.DefaultServlet</servlet-class>
  <init-param>
    <param-name>dirAllowed</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>0</load-on-startup>
</servlet>
```

You can place it above the existing `<servlet>` elements in the file.

Optional Configuration

Supporting SOAP Endpoints

File(s): */opt/shibboleth-idp/credentials/idp-backchannel.p12*, *etc/jetty-backchannel.xml, modules/backchannel.mod, start.d/backchannel.ini*

The use of the back-channel is discussed in the SecurityAndNetworking topic, and you should review that to understand whether or not you need to support this feature.

If you do need this support, these connections generally require special security properties that are not appropriate for user-facing/browser use. Therefore an additional endpoint must be configured.

1. Copy the jetty9-dta-ssl-1.0.0.jar (asc) plugin to `JETTY_BASE/lib/ext`
2. Create *JETTY_BASE/modules/backchannel.mod*:

```
[name]
backchannel

[depend]
server

[xml]
etc/jetty-backchannel.xml
```

3. Create *JETTY_BASE/start.d/backchannel.ini:*

```
--module=backchannel

jetty.backchannel.port=8443
jetty.backchannel.sslContext.keyStorePath=/opt/shibboleth-idp/credentials/idp-backchannel.p12
jetty.backchannel.sslContext.keyStoreType=PKCS12
jetty.backchannel.sslContext.keyStorePassword=passwordgoeshere
```

4. Create *JETTY_BASE/etc/jetty-backchannel.xml*:

jetty-backchannel.xml

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure_9_0.dtd">

<Configure id="Server" class="org.eclipse.jetty.server.Server">

 <!-- ============================================================= -->
  <!-- TLS context factory with optional client auth                 -->
  <!-- and no container trust (delegate to application)              -->
  <!-- for backchannel (SOAP) communication to IdP                   -->
  <!-- ============================================================= -->
  <New id="shibContextFactory" class="net.shibboleth.utilities.jetty9.
DelegateToApplicationSslContextFactory">
    <Set name="KeyStorePath"><Property name="jetty.backchannel.sslContext.keyStorePath" /></Set>
```

```
  <Set name="KeyStoreType"><Property name="jetty.backchannel.sslContext.keyStoreType" /></Set>
  <Set name="KeyStorePassword"><Property name="jetty.backchannel.sslContext.keyStorePassword" /></Set>
  <!-- This is a tweak to work around a bug in Jetty when using the PKCS12 keystore type. -->
  <Set name="TrustStoreType"><Property name="jetty.backchannel.sslContext.keyStoreType" /></Set>
  <Set name="EndpointIdentificationAlgorithm"></Set>
  <Set name="renegotiationAllowed">false</Set>
  <Set name="useCipherSuitesOrder">true</Set>
  <Set name="excludeProtocols">
    <Array type="String">
      <Item>SSL</Item>
      <Item>SSLv2</Item>
      <Item>SSLv3</Item>
    </Array>
  </Set>
  <Set name="IncludeCipherSuites">
    <Array type="String">
      <Item>TLS_ECDHE.*</Item>
      <Item>TLS_RSA.*</Item>
    </Array>
  </Set>
  <Set name="ExcludeCipherSuites">
    <Array type="String">
      <Item>.*NULL.*</Item>
      <Item>.*RC4.*</Item>
      <Item>.*MD5.*</Item>
      <Item>.*DES.*</Item>
      <Item>.*DSS.*</Item>
    </Array>
  </Set>
</New>

<New id="shibHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
  <Arg><Ref refid="httpConfig"/></Arg>
  <Call name="addCustomizer">
    <Arg>
      <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
        <Arg type="boolean"><Property name="jetty.ssl.sniHostCheck" default="true"/></Arg>
      </New>
    </Arg>
  </Call>
</New>

<!-- =========================================================== -->
<!-- IdP SOAP protocol connector                                 -->
<!-- =========================================================== -->
<Call id="shibConnector" name="addConnector">
  <Arg>
    <New class="org.eclipse.jetty.server.ServerConnector">
      <Arg name="server"><Ref refid="Server" /></Arg>
      <Arg name="acceptors" type="int"><Property name="jetty.ssl.acceptors" default="-1"/></Arg>
      <Arg name="selectors" type="int"><Property name="jetty.ssl.selectors" default="-1"/></Arg>
      <Arg name="factories">
        <Array type="org.eclipse.jetty.server.ConnectionFactory">
          <Item>
            <New class="org.eclipse.jetty.server.SslConnectionFactory">
              <Arg name="next">http/1.1</Arg>
              <Arg name="sslContextFactory"><Ref refid="shibContextFactory"/></Arg>
            </New>
          </Item>
          <Item>
            <New class="org.eclipse.jetty.server.HttpConnectionFactory">
              <Arg name="config"><Ref refid="shibHttpConfig"/></Arg>
            </New>
          </Item>
        </Array>
      </Arg>
      <Set name="host"><Property name="jetty.backchannel.host" default="0.0.0.0" /></Set>
      <Set name="port"><Property name="jetty.backchannel.port" default="8443"/></Set>
      <Set name="idleTimeout"><Property name="jetty.ssl.timeout" default="30000"/></Set>
      <Set name="soLingerTime"><Property name="jetty.ssl.soLingerTime" default="-1"/></Set>
      <Set name="acceptorPriorityDelta"><Property name="jetty.ssl.acceptorPriorityDelta" default="0"/><
```

```
/Set>
        <Set name="selectorPriorityDelta"><Property name="jetty.ssl.selectorPriorityDelta" default="0"/><
/Set>
        <Set name="acceptQueueSize"><Property name="jetty.ssl.acceptQueueSize" default="0"/></Set>
      </New>
    </Arg>
  </Call>

</Configure>
```

Offloading TLS

There may be situations where you wish to "offload" TLS to a load balancer or http proxy, a setup looking something like:

---(https)---> Apache/LB ---(http)---> Jetty/Shibboleth IdP

Attempting a user login service request with the default Jetty configuration may result in an error similar to "SAML message intended destination endpoint https://hostname... did not match the recipient endpoint http://hostname...".

Jetty can be configured to consume the 'x-forwarded-proto' HTTP header to override the connection protocol originating at the load balancer, instead respecting the protocol being used between the client and the load balancer, communicated in the x-forwarded-proto header. The Proxy / Load Balancer Configuration section of the Jetty documentation provides instruction on the required configuration.

The following example achieves that using Apache httpd's `mod_proxy` and `mod_headers`. The last line allows passing of `REMOTE_USER` through to the IdP, useful for external authentication in a browser or ECP.

```
RequestHeader set X-Forwarded-Proto "https" env=HTTPS
ProxyPass /idp http://localhost:8080/idp connectiontimeout=5 timeout=15
RequestHeader set REMOTE-USER %{REMOTE_USER}s
```

Supporting X-Forwarded-For

If you are running the Jetty engine behind a proxy or load balancer Jetty 9.3 has built-in support for forwarding the client address and other details via headers.

> As with any proxied deployment, you MUST take care to lock down the path between the proxy and the Jetty server, and the proxy MUST have support for sanitizing and preventing any client attempt to smuggle and hijack those headers. Failure to do so will result in a variety of security compromises. There are many other considerations to proxying far beyond the scope of this document.

1. Copy the file JETTY_BASE/etc/jetty.xml to JETTY_HOME/etc/jetty.xml
2. Edit the file in JETTY_HOME/etc/jetty.xml, locate the:

<New id="httpConfig" class="org.eclipse.jetty.server.HttpConfiguration">

section and add:

```
<Set name="outputBufferSize">32768</Set>
<Set name="requestHeaderSize">8192</Set>
<Set name="responseHeaderSize">8192</Set>

<Call name="addCustomizer">
   <Arg><New class="org.eclipse.jetty.server.ForwardedRequestCustomizer" /></Arg>
</Call>
```

If you are using a custom header, you can change the addCustomizer section to specify the custom header. An example is below:

```
<Call name="addCustomizer">
  <Arg>
      <New class="org.eclipse.jetty.server.ForwardedRequestCustomizer" >
         <Set name="forwardedForHeader">X-MyCustom-Header</Set>
      </New>
   </Arg>
</Call>
```