Jetty Web Server Cheatsheet

A quick reference guide for Jetty, covering configuration, deployment, and essential features. Ideal for developers and system administrators.

Jetty Basics and Configuration

Core Concepts

Jetty: A lightweight, embeddable web server and servlet container.

Handlers: Components that process requests.

Connectors: Components that accept incoming connections.

Contexts: Represent web applications deployed in Jetty.

Thread Pools: Manage threads for handling requests efficiently.

Configuration Files

`jetty.xml`	Main configuration file for Jetty. Defines server, connectors, handlers, and other settings. Located typically in `${jetty.base}/jetty.xml`.
`jetty-deploy.xml`	Configuration file for hot deployment of web applications. Enables automatic deployment of WAR files in the `${jetty.base}/webapps` directory.
`web.xml`	Servlet configuration file for web applications. Defines servlets, filters, and other web application components. Placed inside `WEB-INF` directory in war file.

Starting Jetty

From the command line:

```
java -jar start.jar
```

With specific configuration:

```
java -jar start.jar etc/jetty.xml
etc/jetty-deploy.xml
```

Using an IDE (e.g., Eclipse, IntelliJ) by embedding Jetty.

Deployment and Handlers

Deploying Web Applications

WAR Files	Drop the `.war` file into the `${jetty.base}/webapps` directory. If `jetty-deploy.xml` is configured, the application will be deployed automatically.
Context XML	Create a context XML file (e.g., `mywebapp.xml`) in the `${jetty.base}/webapps` directory to configure the web application. Useful for customization.

Example Context XML	`<Configure class="org.eclipse.jetty.webapp.WebAppContext">`
	`  <Set name="contextPath">/mywebapp</Set>`
	`  <Set name="war">/path/to/mywebapp.war</Set>`
	`</Configure>`

Common Handlers

DefaultHandler: Serves static content and handles default requests.

ContextHandler: Maps requests to specific contexts (web applications).

ResourceHandler: Serves static resources from a specified directory.

RequestLogHandler: Logs incoming requests to a file or other output.

Handler Collections

Use `HandlerCollection` to chain multiple handlers together. This allows you to combine different functionalities.

```
<New id="Handlers"
class="org.eclipse.jetty.server.handler.
HandlerCollection">
  <Set name="handlers">
    <Array
type="org.eclipse.jetty.server.Handler">
      <Item><Ref ref="DefaultHandler"/>
</Item>
      <Item><Ref ref="Contexts"/></Item>
      <Item><Ref ref="RequestLog"/>
</Item>
    </Array>
  </Set>
</New>
```

Connectors and Security

Connectors Configuration

`ServerConnector`	Standard connector for HTTP/1.1. Configured in `jetty.xml`.
	`<New id="http" class="org.eclipse.jetty.server.ServerConnector">`
	`<Arg name="server"><Ref ref="Server"/></Arg>`
	`<Set name="port">8080</Set>`
	`</New>`

`SslConnectionFactory`	Used in conjunction with `ServerConnector` to enable HTTPS.
	`<New id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory">`
	`<Set name="keyStorePath">/path/to/keystore.jks</Set>`
	`<Set name="keyStorePassword">password</Set>`
	`</New>`

HTTPS Connector	`<New id="https" class="org.eclipse.jetty.server.ServerConnector">`
	`<Arg name="server"><Ref ref="Server"/></Arg>`
	`<Arg name="factories">`
	`<Array type="org.eclipse.jetty.server.ConnectionFactory">`
	`<Item><New class="org.eclipse.jetty.server.HttpConnectionFactory"><Arg name="config"><Ref ref="httpConfig"/></Arg></New></Item>`
	`<Item><New class="org.eclipse.jetty.server.SslConnectionFactory">`
	`<Arg name="sslContextFactory"><Ref ref="sslContextFactory"/></Arg>`
	`<Arg name="nextProtocol">http/1.1</Arg>`
	`</New></Item>`
	`</Array>`
	`</Arg>`
	`<Set name="port">8443</Set>`
	`</New>`

Security

Authentication: Jetty supports various authentication methods including Basic, Digest, and Form-based authentication.

Authorization: Control access to resources based on user roles.

SecurityHandler: Enforces security constraints defined in `web.xml` or context XML files.

SSL Configuration

Generate a Keystore:

```
keytool -genkeypair -alias jetty -keyalg RSA -keystore keystore.jks -validity 365
```

Configure `SslContextFactory` in `jetty.xml` with the path to the keystore and password.

Advanced Features

WebSockets

Jetty provides excellent support for WebSockets, enabling real-time bidirectional communication.

Implement WebSocket endpoints using `@WebSocket` annotation or by implementing `WebSocketListener` interface.

Example WebSocket Endpoint:

```
@WebSocket
public class MyWebSocket {
  @OnWebSocketConnect
  public void onConnect(Session session)
{
    // Handle connection
  }

  @OnWebSocketMessage
  public void onMessage(Session session,
String message) {
    // Handle message
  }
}
```

JNDI

JNDI Resources	Jetty supports JNDI (Java Naming and Directory Interface) for managing resources like data sources. Configure JNDI resources in the context XML file.

Example JNDI Configuration	`<New id="myDataSource" class="org.apache.commons.dbcp2.BasicDataSource">`
	`  <Set name="driverClassName">com.mysql.cj.jdbc.Driver</Set>`
	`  <Set name="url">jdbc:mysql://localhost:3306/mydb</Set>`
	`  <Set name="username">user</Set>`
	`  <Set name="password">password</Set>`
	`</New>`
	
	`<New class="org.eclipse.jetty.plus.jndi.Resource">`
	`  <Arg><Ref ref="Server"/></Arg>`
	`  <Arg>jdbc/mydb</Arg>`
	`  <Arg><Ref ref="myDataSource"/>`
	`</Arg>`
	`</New>`

Logging

Jetty uses `Slf4j` as a logging facade. Configure the underlying logging implementation (e.g., Logback, Log4j) to control logging behavior.

Configure `RequestLogHandler` to log HTTP requests. Customize the log format and output destination.