Esper HQ Web Application Archive

Version 5.3.0

by *EsperTech Inc.* [http://www.espertech.com]

Preface

This document introduces the EsperHQ web application archive (WAR).

The EsperHQ web application is the server-side code that provides the EsperHQ client application to web browser clients and that interfaces with data distribution services.

The *Chapter 1, Overview* chapter is the best place to understand features.

Chapter 1. Overview

Esper HQ is the web application archive containing the web application and HQ Services REST services. The web application is a multi-window rich client application for Esper CEP engine management and real-time continuous displays (aka. eventlets).

The Esper HQ web application archive (WAR) is for use with Java Enterprise Edition -standard web containers. Esper HQ interfaces to one or multiple Esper CEP Management and CEP Push Services REST endpoints that each interface to Esper CEP engine instances.

1.1. Esper HQ Web Application Features

An Esper HQ web application instance performs the following functions:

1. Handle client subscription lifecycle and push data to web client applications based on active client subscriptions (asynchronous publish-subscribe communication style).
2. Consolidate client subscription requests to avoid duplicate data transmission.
3. Handle client disconnect.
4. Handle client requests for CEP management services, such as for creating statements or setting variable values, for example.
5. Present a faceless (HTML-only) launch pad for continuous real-time displays.
6. Make the web application and its components available for download to web browsers.
7. Consolidate data from multiple CEP servers (CEP Push Services endpoints) to support partitioning and consolidation.

Chapter 2. Getting Started

Esper Enterprise Edition comes with Esper HQ prepackaged. Please simply follow the directions in file `RUNNING.txt` or the index web page to start the preconfigured server.

After starting the preconfigured server the web client application is available at *http://localhost:8400/esperhq/esperhq/esperhq.html*.

The Esper HQ web application and web client application consists of the following components:

1. The Esper HQ war file (war stands for web application archive, or WAR) can be found in *installation_root*`/webapps/esperhq`. The distribution provides the WAR file in unpacked form.
2. The Esper HQ configuration file named `esperhq-default.xml` in the `conf` folder of the distribution (the file name is referenced by the web.xml deployment descriptor in the war file).
3. The web client application files reside in the Esper HQ war file in *installation_root*`/webapps/esperhq/esperhq`.
4. A XML XSD schema defining static eventlet configuration. The schema file is provided in the `schema` folder of the distribution and is named `esper_eventlet_`*version*`.xsd`.

Esper HQ minimally requires the following for operation:

1. Java 6 or higher.
2. An Esper HQ configuration file.
3. At least one Esper Data Distribution Services endpoint.
4. A web browser.

The Esper HQ web application requires a J2EE web application container. Esper Enterprise Edition ships with the Jetty web container (see http://www.mortbay.org/jetty/). We do not specifically recommend any particular web application container - you can deploy into any J2EE standard web container.

There are multiple options to deploy and start Esper HQ:

1. Start your favorite J2EE web application container and copy the Esper HQ war into its deployment directory or deploy via the container's web console.

   For use when you already have a web container you would like to use.

2. As an Esper plug-in as part of an Esper configuration XML file or configuration API:

   This requires your application to add the web container plug-in class to the Esper configuration (XML or API) as described next. The Jetty web container and Esper HQ are then started automatically as part of Esper engine initialization.

2.1. Web Application Container Plug-in

Esper Enterprise Edition provides the Jetty web application container and a plug-in class to start the container and deploy the Esper HQ war file.

The plug-in class is `com.espertech.esper.server.webapp.WebAppPlugin` and resides in the jar file `esper-server-`*version*`.jar`.

Under this option the Jetty web container and Esper HQ is initialized when your application first obtains an `EPServiceProvider` instance for a given URI or when your application calls the `initialize` method on an `EPServiceProvider`. The Jetty web container and Esper HQ is destroyed when your application calls the `destroy` method on an `EPServiceProvider` instance or when it calls the `initialize` method on `EPServiceProvider` instance that had the adapter in its configuration.

Configuration is described in detail in the Server documentation.

2.2. Esper HQ File System Access for Project Files

EsperHQ presents a directory structure and files in the `Project Files` folder in the GUI. The files are located under the installation root in the folder `data/hqsvc`.

Chapter 3. Securing the Web Application

Configuring SSL communication requires following a multistep process that includes generating an SSL certificate. Enterprise Edition utilizes the Jetty container for the client application and REST web services. Please follow the steps described in the Jetty documentation at http://www.eclipse.org/jetty/documentation/current/configuring-ssl.html to enable SSL communication.

3.1. Authentication

The EsperHQ client application and eventlet faceless execution can be protected as described below. By default a new installation of the distribution does not authenticate or check authorizations.

To enable the login challenge, and authorization checking of roles, set the provider-type in the authentication settings to `local`, as outlined in *Section 4.1.2, “Authentication”*. You may add individual users and credentials directly to the EsperHQ configuration file.

To integrate EsperHQ with common security authentication and authorization architectures including for example LDAP, Siteminder, OpenID and JA-SIG we provide instructions to configure Spring security below.

As a further alternative, consider configuring your own servlet filter or servlet context listener in the EsperHQ war file to secure EsperHQ URLs.

In addition, if your application requires fine-grained control over individual operations that EsperHQ servlet performs on request by client applications, EsperHQ provides a callback that you may register. We explain service filters below. A service filter can perform authorization checks and may also modify operation request values.

EsperHQ enforces authentication and authorization in the client application and as part of all servlet operations of the web application server.

3.2. Roles

The `EHQ_ADMIN` role is a superset of all roles and provides access to all functions.

Assign the `EHQ_EXEC` role to users that may only execute an eventlet via faceless launch.

Assign the `EHQ_RO` role to users that may only perform read-only operations in the client application.

The below table summarizes authorizations:

Table 3.1. Roles

Function	EHQ_ADMIN	EHQ_RO	EHQ_EXEC
All create, update and delete operations, e.g. create statements, transition statements, set variable values, create eventlets and others.	Yes.	No.	No.
All read-only operations, e.g. view statements, view variable values.	Yes.	Yes.	No.
Launch eventlets via client application or faceless.	Yes.	No.	Yes.

3.3. Securing via Spring Security (ACEGI)

This section provides the instructions to configure the web application with Spring (ACEGI) security. Via Spring ACEGI security it is easy to secure and integrate EsperHQ into many additional security architectures.

When using Spring security, the step-by-step instructions to enable security are as follows. File paths provided below are relative to the installation root directory.

1. Open `bin/setclasspath.sh` (Linux) or `bin/setclasspath.bat` (Windows) and remove the comments in `Spring/ACEGI Security` section, such that Spring and Spring Security jar files are part of the classpath.
2. Change the `webapps/esperhqapp/WEB-INF/web.xml` file to include a reference to the Spring security context and the Spring web context loader, by adding the following lines before `</web-app>`:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:/spring-security.xml
   </param-value>
</context-param>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

3. Add the `spring-security.xml` Spring security configuration to the `webapps/esperhqapp/WEB-INF/classes` folder, create the folder if it does not exist. The distribution provides a sample file in `lib-security`.
4. Configure EsperHQ to set the authentication provider type to Spring by setting `provider-type="spring"`.

After restarting the server, review server console or log output to determine that Spring security initialized successfully and no exceptions were logged.

Chapter 4. Configuration

Your Esper HQ XML configuration file should adhere to the XML XSD schema file provided in the `conf` folder of the distribution by name `esperhq-configuration-5-0.xsd`.

4.1. Client and Web Application Settings

This section discusses EsperHQ rich client configuration and web application archive (WAR) settings in the `web.xml`file.

4.1.1. Web Socket URL

By default, the web socket URL is `ws://myhost:myport/hqpush/eventbus` wherein host and port are automatically derived from the request of the client and without requiring a configuration. The client console trace outputs the web socket url as well.

To override the computed web socket url, please configure the url as shown below:

```
<esperhq-configuration>
  <client websocketurl="ws://myhost:myport/hqpush/eventbus"/>
</esperhq-configuration>
```

4.1.2. Authentication

This configuration item controls authentication and authorization. Please consult *Chapter 3, Securing the Web Application* for an overview.

The synopsis is as follows:

```
<authentication provider-type="[none|local|spring]"
  [remember-me="false|true"]>
  [<password-encoder hash="none|md5"/>]
  [<users>
 [<user name="[name]" password="[credentials]" authorities="[list_of_roles]" />]
  </users>]
</authentication>
```

When *provider-type* is set to `none` then the client application and server web application do not authenticate or check authorization.

When *provider-type* is set to `local` then the client application and server web application authenticate and check role authorities for each client. You must provide a list of users and a password encoding. When set to `spring` you must follow the additional steps as outlined in

Section 3.3, “Securing via Spring Security (ACEGI)” and the Spring security configuration (via LDAP, OpenID etc.) provides users and roles.

When *provider-type* is set to `spring` then the client application and server web application authenticate and check role authorities for each client using Spring security. Please follow the additional steps as outlined in *Section 3.3, “Securing via Spring Security (ACEGI)”* and provide a Spring security configuration that may reference LDAP, OpenID or other authentication providers.

The *remember-me* attribute is optional and enabled by default. When enabled the server application checks if a security context has already been established for the client or session and returns the existing security context. If the security context exists, a login challenge is not presented for the client.

The *password-encoder* element is required for the `local` provider type and indicates the password encoding. Use `md5` for MD5 encoding and `none` for plain-text passwords.

If configuring the `local` provider type then repeat the *user* element for each user and provide a name, password and a comma-separated list of roles.

An example XML to configure authentication is:

```
<esperhq-configuration>
  <client>
    <authentication provider-type="local" remember-me="true">
      <password-encoder hash="md5"/>
      <users>
        <!-- Sample users: rod/koala, dianne/emu, peter/cat -->
                 <user  name="rod"  password="a564de63c2d0da68cf47586ee05984d7"
 authorities="EHQ_ADMIN" />
               <user  name="dianne"  password="65d15fe9156f9c4bbffd98085992a44e"
 authorities="EHQ_RO" />
               <user  name="peter"  password="d077f244def8a70e5ea758bd8352fcd8"
 authorities="EHQ_EXEC" />
      </users>
    </authentication>
  </client>
</esperhq-configuration>
```

4.1.3. Service Filters

Your application can provide a service filter to introduce additional security checks or to modify client application requests.

The synopsis is as follows:

```
<filters>
  <filter class-name="[class_name]"/>
```

```
</filters>
```

The *class_name* is name of the class implementing the `com.espertech.esper.hqsvc.config.ClientRequestFilter` interface. Every client operation can be intercepted by the implementation of this interface, client operation parameters can be interrogated and changed. Please see the API documentation for additional detail.

An example XML to configure authentication is:

```
<esperhq-configuration>
  <client>
    <filters>
      <filter class-name="com.mycompany.MyServiceFilter"/>
    </filters>
  </client>
</esperhq-configuration>
```

4.2. Endpoint Configuration

This section describes configuration of CEP endpoints. We use the term *endpoint* to describe a server process that hosts one or more CEP engine instances.

The default configuration configures a single endpoint. You may configure multiple endpoints for a multi-server setup.

For endpoints that provide only Management REST services, it is not necessary to configure a JMS provider and a Push REST service token.

For endpoints that provide only Push REST services, it is not necessary to configure a Management REST service token.

Please ensure that required Push REST service or Management REST service are active for the configured endpoints.

You may designate one or more endpoints as default endpoints. The web client application uses the default endpoint in the configuration to present the default choice in the endpoint dropdown.

The `context` element configures the JMS provider JNDI context.

The `restservice` element configures the REST URI and token information. The REST URI must be a value specifying the HTTP or HTTPS protocol and must specify the host and port, such as `http://myhost:myport`. The special value `local` is supported and is detailed below.

The below XML configuration configures a provider by name `defaultendpoint`:

```
<esperhq-configuration>
```

```
  <endpoints>
    <endpoint name="defaultendpoint" default="true">
      <context object-name="ConnectionFactory">
        <env name="java.naming.factory.initial"
           value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/>
        <env name="java.naming.provider.url"
           value="tcp://localhost:61616?wireFormat.maxInactivityDuration=0"/>
      </context>
      <restservice hosts="local" cepmgmtsvc-token="cepmgmtsvctoken"
          ceppushsvc-token="ceppushsvctoken"/>
    </endpoint>
  </endpoints>
  ... more settings...
</esperhq-configuration>
```

4.2.1. Configuring the JMS Provider Context

The information under `context` configures the JNDI object name and parameters for the JNDI context lookup to resolve the JMS provider.

The information under `context`, in the default configuration, matches the embedded ActiveMQ broker that is configured by the Enterprise Edition default configuration.

If your environment utilizes external JMS brokers, please update hostname and port numbers accordingly.

For Push Services, wherein CEP engine(s) push data to the same or other web layer servers, it is required to configure the JMS provider context.

4.2.2. Configuring REST Service Addressing

The information under `restservice` describes the URI (protocol, host, port) and tokens for access to CEP Management REST services and Push REST Services.

In the `hosts` attribute you can simply specify `local` to indicate that the REST services are available at the same host and port as the enclosing web application container.

In the `hosts` attribute, for multi-server setups, specify a new value for `hosts` that contains the hostname and port number formatted as *protocol*://*hostname*:*port*. For example `http://cepserverhost:8400`.

The `cepmgmtsvc-token` is optional and by default set to the same token as configured for the web application container under `cepmgmtsvc`, further described in the Server documentation under REST Services security. If no token is provided, authentication of CEP Management service clients does not take place only if the configuration of the web application container also does not list a token.

The `ceppushsvc-token` is optional and by default set to the same token as configured for the web application container under `ceppushsvc`, further described in the Server documentation under

REST Services security. If no token is provided, authentication of CEP Push service clients does not take place only if the configuration of the web application container also does not list a token.