Ozone Metrics Service

Metrics Service Guide

December 21, 2012

Publication/Revision History

Release	Date
Revised – Metrics Service 7	December 21, 2012
Revised – Metrics Service 6	September 28, 2012
Initial Document – Metrics Service 5	June 29, 2012

Contents

Tables

1 Introduction

1.1 Objectives

The purpose of this guide is to explain how to use the OZONE Metrics Service. This predominantly covers, but is not limited to, the configuration of the servers.

1.2 Dependencies

The Metrics bundle is shipped with Tomcat 7.0.21 which requires JDK 1.6 or higher. If running the Metrics Service with a Web server other than Tomcat, please see that Web server's documentation for requirements.

1.3 Components

1.3.1 Metrics Service Web Application

`Metrics.war` – This file contains the components which make up the Metrics Service.

1.3.2 CAS Web Application

`cas.war` – This optional file is responsible for providing the Central Authentication Service (CAS).

2 Integrating the Metrics Service into OWF

An administrator must configure OWF to pass server-to-server data to the Metrics Service. If the Metrics Service is hosted on a different server than the server running OWF, an administrator must incorporate the Metrics Service into OWF. This involves changes to the **OWFConfig.groovy** file:

1) Open **apache-tomcat-7.0.21\lib\OWFConfig.groovy**.

2) Change the following properties from **false** to **true** to enable the Metrics Service. Configure the URL to the server location where the Metrics service is located. Update the following properties:

```
…
  Metrics {
    enabled = false
    url = 'https://servername:port/Metrics/Metrics'

    //Optional additional properties with default values shown
    //keystorePath = System.properties['javax.net.ssl.keyStore']
    //keystorePass = System.properties['javax.net.ssl.keyStorePassword']
    //truststorePath = System.properties['javax.net.ssl.trustStore']
    //timeout = 1800000
  }
…
```

3) If OWF or the Metrics Service is hosted on different servers than their server security certificates, generate certificates by setting **keystorePath**, **keystorePass** and **truststorePath** properties.

4) Start the service.

2.1 Configuring the Metrics Widget in OWF

Administrators can view Metrics data using an OWF widget. To make Metrics data available in OWF:

1) Integrate the Metrics Server into OWF, as described in the previous section.

2) From the Toolbar, click the OWF Administration tools and select Widgets to open the Widget Manager.

3) Use the following information to create the View Metrics Widget:

Table 1: Data for Metrics Widget Definition

Definition	Data Input
URL	https://widget-servername:port/Metrics/admin/Metrics.gsp
Container Icon URL	https://widget-servername:port/Metrics/themes/common/images/icons/16x16_Metrics.png
Launch Menu Icon URL	https://widget-servername:port/Metrics/themes/common/images/icons/64x64_Metrics.png
Width	700
Height	500
Widget Type	Metrics

Note: The Metrics Widget can respond to OWF themes. For information about customizing themes, see the OWF Configuration Guide.

2.2 Viewing the Metrics Widget

The Metrics Service can be viewed in OWF through the Metrics Widget. After following the steps in section 2.1: Configuring the Metrics Widget in OWF and assigning the View Metrics Widget to users, it will appear under the Metrics button in the Toolbar.

Note: OWF users who signed in using CAS will be prompted to sign in to additional security for the Metrics Service.
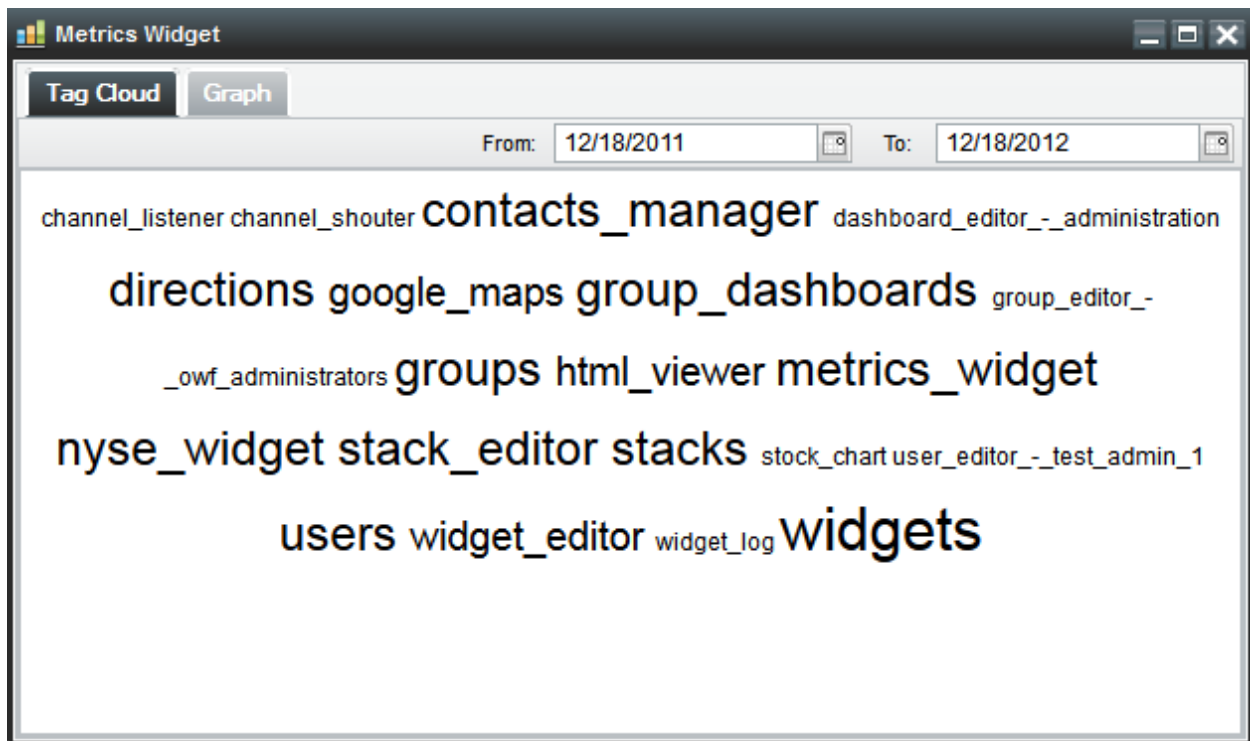
Figure 1: Metrics Widget

The widget opens to a Tag Cloud tab that lists widget views. Clicking the name of a widget from the Tag Cloud or the grid switches to the widget's Graph tab. This displays how many times that widget was viewed along with a grid listing all widgets and their data. To change the monitoring dates, click the calendar icon(s) above the graph. Use the arrows or click directly on the month and year to select different date durations. Clicking the month and year (identified in the image below) opens a drop-down date selector. After clicking OK, click the highlighted date on the calendar to complete the change.
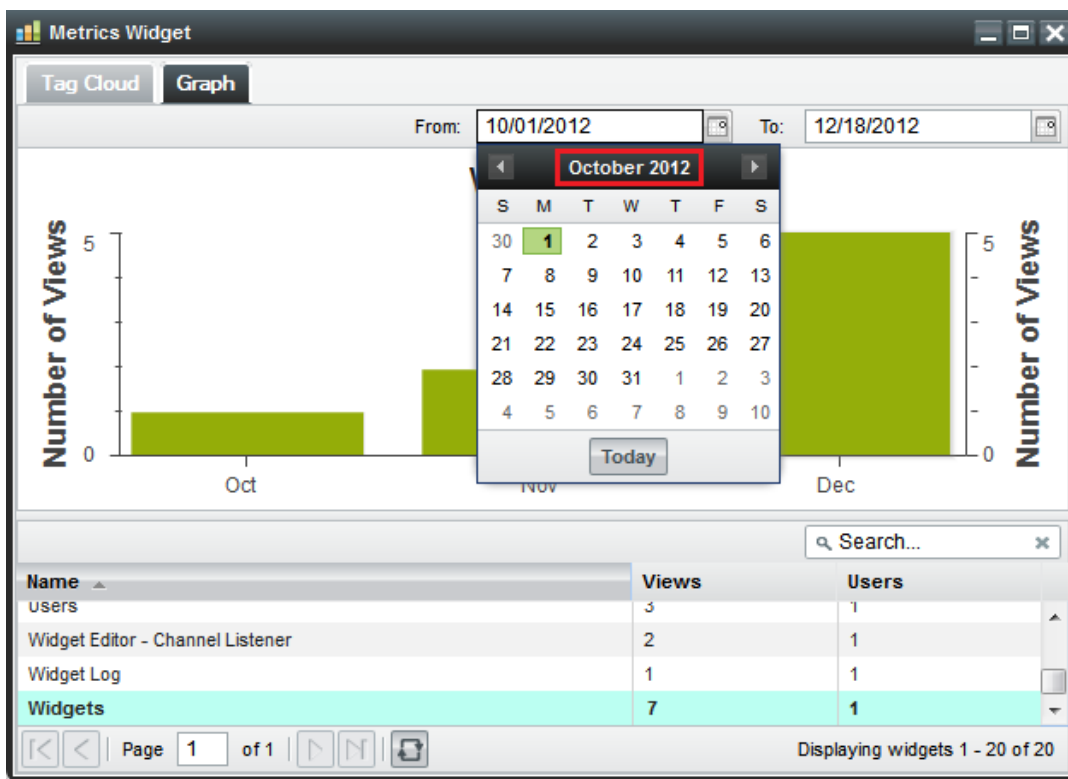
Figure 2: Graph Tab Displaying Date Switcher

2.2.1 Keyboard Navigation

To open the Metrics Switcher on the Toolbar, use the following keyboard combination:

Table 2: Metrics Switcher Keyboard Combination

Icon	Keyboard Combination
	Alt + Shift ⇧ + R

2.3 Sending Metrics to the Metrics Server

Administrators and Developers can send their own Metrics data to the server should it be needed. The JavaScript code shown below must be used to facilitate the data transfer. For full details, be sure to see the Metrics API found in the JavaScript API Reference Document Appendix of the OWF Developer's Guide.

Namespace OWF.Metrics

Defined in: Widget.js.

Namespace Summary	
	OWF.Metrics

Method Summary		
<static>	OWF.Metrics.**logBatchMetrics**(Metrics)	
	Logs a set of Metrics to the server all at once.	
<static>	OWF.Metrics.**logMetrics**(userId, userName, Metricsite, componentName, componentId, componentInstanceId, MetricsTypeId, MetricsData)	
	Basic logging capability - meant to be called by other methods which transform or validate data.	
<static>	OWF.Metrics.**logWidgetRender**(userId, userName, Metricsite, widget)	
	Log view of widget - see calls in dashboards.	

Namespace Detail
OWF.Metrics

Method Detail

<static> *OWF.Metrics.***logBatchMetrics**(Metrics)

Logs a set of Metrics to the server all at once. All Metrics passed into a call to this function will be logged in a single HTTP request, instead of one request per Metrics

> **Parameters:**
> > *{Array}* **Metrics**
> > *{String}* **Metrics[*].userId**
> > *{String}* **Metrics[*].userName**
> > *{Number}* **Metrics[*].MetricsTime**
> > > The time at which is Metrics was collected (in UNIX time)
> > *{String}* **Metrics[*].site**
> > > Identifier, potentially URL, for source of Metrics - typically OWF instance
> > *{String}* **Metrics[*].component**
> > *{String}* **Metrics[*].componentId**
> > *{String}* **Metrics[*].instanceId**
> > *{String}* **Metrics[*].MetricsTypeId**

String describing Metrics - recommend package name construct

{String} **Metrics[*].widgetData**

Any additional data for Metrics - do any necessary validation appropriate to MetricsTypeId before sending through

{String} **Metrics[*].userAgent**

Should be set to the user-agent string of the browser

Since:

OWF 6.0

`<static>` *OWF.Metrics.***logMetrics**`(userId, userName, Metricsite, componentName, componentId, componentInstanceId, MetricsTypeId, MetricsData)`

Basic logging capability - meant to be called by other methods which transform or validate data.

Parameters:

{String} **userId**

{String} **userName**

{String} **Metricsite**

Identifier, potentially URL, for source of Metrics - typically OWF instance

{String} **componentName**

{String} **componentId**

{String} **componentInstanceId**

{String} **MetricsTypeId**

String describing Metrics - recommend package name construct

{String} **MetricsData**

Any additional data for Metrics - do any necessary validation appropriate to MetricsTypeId before sending through

Since:

OWF 3.8.0

`<static>` *OWF.Metrics.***logWidgetRender**`(userId, userName, Metricsite, widget)`

Log view of widget - see calls in dashboards.

Parameters:

{String} **userId**

- see Ozone.Metrics.logMetrics userId

{String} **userName**

- see Ozone.Metrics.logMetrics userName

{String} **Metricsite**

- see Ozone.Metrics.logMetrics Metricsite

{Object} **widget**

Since:

OWF 3.8.0

3   Pluggable Security

The Metrics Service allows an administrator to customize the type of security that is to be implemented for user authentication and authorization. Included with the Metrics Service's **owf-security** directory are **.xml** files that provide examples of optional security configurations. They are intended as examples and should in no way be used in a production environment. Along with the security-related **.xml** files, there is also a **.zip** file which contains the source and configuration files for the pluggable security modules and an Apache ANT build script.

Note: Many security mechanisms can be used for user authentication and authorization. However, for the Metrics Service to record OWF data, at a minimum it must use the security provided by X.509 certificates for server-to-server communication.

3.1   Default Authentication

MetricSecurityContext.xml - This contains the default security implementation for the Metrics Service. It uses a PKI certificate for authentication. If no authentication is provided, it redirects the user to log in using CAS as a fallback.

3.2   X.509 Only Security

MetricSecurityContext_cert_only.xml - This contains the X.509-only security implementation for the Metrics Service. It uses a PKI certificate for authentication. If no authentication is provided, the user is denied access to the system.

3.3   X.509/LDAP

MetricSecurityContext_cert_ldap.xml - This contains an X.509/LDAP security implementation that uses X.509 for authentication and then performs an LDAP-based lookup to determine the user's authorization.

3.4   OWF Security Project

owf-security-project.zip - This bundle contains the source code, configuration files and library files needed to build the security files which are used by the Metrics Service. Additionally, an Apache ANT build script is included for building a **JAR** file. It is used by the aforementioned security **XML** files and supporting resource file **lib/spring-core-3.0.1.RELEASE.jar** which is a file which provides LDAP functionality. The Ozone-LDAP-Security plugin uses the **JAR** file.

4   Installation

4.1   Dependencies

Listed below are the installation dependencies for the Metrics Service:

- Java 1.6 or higher.

- Relational Database Management System. The Metrics Service currently ships with an in-memory HyperSQL (HSQLDB) database for testing and development purposes, but it is expected that a live deployment will use a more robust RDBMS such as Oracle or MySQL.

4.2   Metrics Service Bundle Description

The distribution of the Metrics Service consists of a `.zip` file containing the necessary components to set up and run the Metrics Service in a development environment. The bundle contains the following:

- Tomcat-7.0.21 (Simple Java Web Container)

- Sample PKI Certificates for SSL (user and server)

- Metrics Service Web application (`Metrics.war`)

- Central Authentication Service application (`cas.war`)

- Externalized Security Configurations:
  - `MetricSecurityContext.xml`
  - `MetricSecurityContext_cert_ldap.xml`
  - `MetricSecurityContext_cert_only.xml`

- Tomcat start scripts (`start.sh` or `start.bat`)

- Four configurable externalized properties files:
  - `MetricsConfig.groovy`
  - `CASSpringOverrideConfig.xml`
  - `OzoneConfig.properties`
  - `Metrics-override-log4j.xml`

The following example shows how an administrator might copy, unzip, and start the Metrics Service from the bundle deployment on ***nix** operating systems, assuming the bundle is name **Metrics-bundle-7-GA.zip**:

```
cp Metrics-bundle-7-GA.zip/opt/.
cd /opt
```

```
unzip Metrics-bundle-7-GA.zip
cd apache-tomcat-7.0.21
./start.sh
```

The following example shows how an administrator might copy, unzip, and start the Metrics Service from the bundle on **Windows** operating systems, assuming the bundle is named **Metrics-bundle-7-GA.zip**:

1) Create a new directory from where the Metrics Service will be run. This can be done via the *Windows* UI or the command prompt.

2) Copy **Metrics-bundle-7-GA.zip** to the new directory created in step 1.

3) Unzip the **Metrics-bundle-7-GA.zip** file.

4) From a command-line, run **start.bat** from within the **apache-tomcat-7.0.21** directory.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 1.6+ enabled system.

4.3 Default Installation

Running the Metrics Service bundle via the included Tomcat Web server with the default values requires minimal installation. With standard configuration, the Metrics Service makes use of the default authentication module, which provides X.509 authentication/authorization, with CAS as a fallback if the framework cannot authenticate the user via certificates installed in their browser.

The application uses a **keystore** and a **truststore** which are local to the installation. There is no need to install any certificates into the server's Java installation. The default certificates contained in the Metrics Service bundle only function for **localhost** communications. When accessed from a remote machine with a name that differs from **localhost** and while using the included certificates, the Metrics Service will not function correctly.

4.4   Installing User PKI Certificates

By default, the security infrastructure of the Metrics Service bundle is configured to use client certificates with CAS fallback. In order to identify themselves via certificates, clients need to install a PKI certificate into their Web browser. The client certificates that are included with the Metrics Service bundle will be recognized immediately and can be used in the default security configuration. The certificates are located in the **\apache-tomcat-7.0.21\certs** directory of the Metrics Bundle.

The default client certificates can be used by importing the included **testUser1.p12** or **testAdmin1.p12** certificate into the user's browser. In Internet Explorer, client certificates can be added by selecting Tools → Internet Options → Content → Certificates → Personal, and then clicking the Import button. The certificate **testUser1** grants rights to use the application, while **testAdmin1** is a certificate for a user granted both user rights and administrator rights. The private key password for both certificates is **password**.

In Firefox this menu is accessed by selecting Tools → Options → Advanced → Encryption → View Certificates →Your Certificates → Import.

4.5   Custom Installation

The Metrics Service can be customized to run in a variety of environments. The following sections detail how to change default database settings and set up security.

4.5.1 Database Setup

While the full extent of administering the following databases is outside the scope of this guide, this section provides information on how to work with databases for the Metrics Service.

MetricsConfig.groovy is a configuration file that allows an administrator to modify database connectivity information. It is located in the **\apache-tomcat-7.0.21\lib** directory. Once changes are made, restart the system to apply them. Developers comfortable with the Groovy language and the Grails application framework should be comfortable writing additional code for the file.

Listed below are the variable database properties that need to be modified to customize the Metrics Service database. A detailed explanation of each field follows:

```
dataSource {
    pooled = true
    dbCreate = "none"
    username = "sa"
    password = ""
    driverClassName = "org.hsqldb.jdbcDriver"
    url = "jdbc:hsqldb:file:MetricsDb;shutdown=true"
    pooled = true
    properties {
        minEvictableIdleTimeMillis = 180000
        timeBetweenEvictionRunsMillis = 180000
        numTestsPerEvictionRun = 3
        testOnBorrow = true
        testWhileIdle = true
        testOnReturn = true
        validationQuery = "SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS"
    }
}
```

Table 3: Metrics Service Database Properties

Property	Purpose	Example
dbCreate	The way the database is created or updated upon server start	"none"
username	The username for the database connection *NOTE: The appropriate database script should be used to create the database, so this property should always be set to "none". Details on which script to run can be found in the individual database instructions below.*	"admin"
password	The password for the database connection	"password"
driverClassName	JDBC driver	"org.hsqldb.jdbcDriver"

URL	JDBC Connection String	"jdbc:hsqldb:file:MetricsDb; shutdown=true"
pooled	Enable database connection pooling when true	true
minEvictableIdleTimeMillis	Minimum amount of time in milliseconds an object can be idle	"18000"
timeBetweenEvictionRunsMillis	Time in milliseconds to sleep between runs of the idle object evictor thread	"18000"
numTestsPerEvictionRun	Number of objects to be examined on each run of idle evictor thread	"3"
testOnBorrow	When true objects are validated before borrowed from the pool	true
testWhileIdle	When true, objects are validated by the idle object evictor thread	true
testOnReturn	When true, objects are validated before returned to the pool	true
validationQuery	Validation query, used to test connections before use	"SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS"

4.5.1.1 Using Oracle

1) Create an Oracle database user for the Metrics Service. It is recommended that there be a dedicated user for the Metrics Service to avoid database object name collisions. The Metrics Service team recommends using UTF8 encoding.

2) Due to licensing issues, the Metrics Service does not provide a JDBC driver for Oracle. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the `\apache-tomcat-7.0.21\lib` directory.

3) Open the **\apache-tomcat-7.0.21\lib\MetricsConfig.groovy** file and modify the **environments → production → dataSource** section using the values that are appropriate for the Metrics Service environment. For example:

```
dataSource {
    pooled = true
    dbCreate = "none"
    username = "Metrics_user"
    password = "Metrics_password"
    dialect = "org.hibernate.dialect.Oracle10gDialect"
    driverClassName = "oracle.jdbc.driver.OracleDriver"
    url = "jdbc:oracle:thin:@myhost.somewhere.org:1521:DEVDB1"
    properties {
        minEvictableIdleTimeMillis = 180000
        timeBetweenEvictionRunsMillis = 180000
        numTestsPerEvictionRun = 3
        testOnBorrow = true
        testWhileIdle = true
        testOnReturn = true
        validationQuery = "SELECT 1 FROM DUAL"
    }
}
```

In the example above, an Oracle database user named **metrics_user** with a password of **metrics_password** is used for a database named **DEVDB1**.

There are several different types of Oracle drivers (thin, OCI, kprb) and connection options (service, SID, TNSName) available. Please consult the Oracle DBA and Oracle's JDBC documentation to create the connection most appropriate for the installed environment.

1) To create the schema, run the **\dbscripts\OraclePrefsInitialCreate.sql** script prior to starting the Metrics Service.

2) Ensure that the transaction is committed.

4.5.1.2 Using MySQL

1) Create a schema within MySQL for use with the Metrics Service. It is recommended that there be a dedicated schema for the Metrics Service to avoid database object name collisions. The OWF team recommends using UTF-8 encoding.

2) Create a MySQL user with full access to the schema created above.

3) Due to licensing issues, the Metrics Service does not provide a JDBC driver for MySQL. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\apache-tomcat-7.0.21\lib** directory.

4) Open the **\apache-tomcat-7.0.21\lib\MetricsConfig.groovy** file and modify the **environments** → **production** → **dataSource** section using the values that are appropriate for the Metrics Service environment. For example:

```
dataSource {
    pooled = true
    dbCreate = "none"
    driverClassName = "com.mysql.jdbc.Driver"
    url="jdbc:mysql://myhost.somewhere.org/Metrics"
    username = "Metrics_user"
    password = "Metrics_password"
    dialect = "org.hibernate.dialect.MySQL5InnoDBDialect"
    properties {
        minEvictableIdleTimeMillis = 180000
        timeBetweenEvictionRunsMillis = 180000
        numTestsPerEvictionRun = 3
        testOnBorrow = true
        testWhileIdle = true
        testOnReturn = true
        validationQuery = "SELECT 1"
    }
}
```

In the example above, a MySQL database-user named **Metrics_user** with a password of **Metrics_password** is used, for a database named **Metrics**. The dialect **org.hibernate.dialect.MySQL5InnoDBDialect** will use the **InnoDB** engine which is recommended for interactive **webapps** and explicitly used as the engine on the database scripts.

5) Create the schema by running the **\dbscript\MySqlPrefsInitialCreate.sql** script prior to starting the Metrics Service.

Note: When creating database objects, modify the **.sql** *script (mentioned above) with the appropriate schema name. For example:*

```
use Metrics;
```

4.5.1.3 *Using PostgreSQL*

1) Create either a new login role or a new schema in order to avoid database object name collisions between the Metrics Service and other database applications.

2) Edit the user so that it can create database objects.

3) Create a new database. Use UTF-8 as encoding (default).

4) Due to licensing issues, the Metrics Service does not provide a JDBC driver for PostgreSQL. Obtain the appropriate JDBC driver and place it into the Web

server's classpath. For example, if running Tomcat, the driver can be placed in the **\apache-tomcat-7.0.21\lib** directory.

5) Open the **\apache-tomcat-7.0.21\lib\MetricsConfig.groovy** file and modify the **environments → production → dataSource** section using the values that are appropriate for the Metrics Service environment. For example:

```
dataSource {
    pooled = true
    dbCreate = "none"
    username = "Metrics_user"
    password = "Metrics"
    driverClassName = "org.postgresql.Driver"
    url = "jdbc:postgresql://localhost:5432/Metrics"
    dialect="org.hibernate.dialect.PostgreSQLDialect"
    properties {
        minEvictableIdleTimeMillis = 180000
        timeBetweenEvictionRunsMillis = 180000
        numTestsPerEvictionRun = 3
        testOnBorrow = true
        testWhileIdle = true
        testOnReturn = true
        validationQuery = "SELECT 1"
    }
}
```

In the example above, a PostgreSQL database user named **Metrics_user** with a password of **Metrics** is used, for a database named **Metrics**.

6) Create the schema by running the **PostgreSQLPrefsInitialCreate.sql** script before starting the Metrics Service.

4.5.1.4 Using SQL Server

1) Create a new SQL Server database for use with the Metrics Service.

2) Create a SQL Server user with full access to the Metrics Service database created above.

3) Due to licensing issues, the Metrics Service does not provide a JDBC driver for SQL Server. Obtain the appropriate JDBC driver and place it on the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\apache-tomcat-7.0.21\lib** directory.

4) Open the **\apache-tomcat-7.0.21\lib\MetricsConfig.groovy** file and modify the **environments → production → dataSource** section using the values that are appropriate for the environment. For example:

```
dataSource {
    pooled = true
    dbCreate = "none"
    username = "sa"
    password = "Metrics"
    driverClassName = "net.sourceforge.jtds.jdbc.Driver"
    url = "jdbc:jtds:sqlserver://localhost:1443/Metrics"
    dialect="ozone.owf.hibernate.OWFSQLServerDialect"
    properties {
        minEvictableIdleTimeMillis = 180000
        timeBetweenEvictionRunsMillis = 180000
        numTestsPerEvictionRun = 3
        testOnBorrow = true
        testWhileIdle = true
        testOnReturn = true
        validationQuery = "SELECT 1"
    }
}
```

In the example above, the SQL Server database user named `sa` with password of `Metrics` is used, to access a database named `Metrics`.

5) Create the schema by running the `SQLServerPrefsInitialCreate.sql` script prior to starting the Metrics Service.

4.5.2 Security Setup

The Metrics Service provides a modular security approach that is based on Spring Security. All of the provided options supply both a Spring Security configuration file and Java classes that are written to Spring's security interfaces in order to perform authentication and authorization.

4.5.2.1 Installing The Security Module

The OWF-security files, provided in the distribution bundle, offer multiple examples of security options. These are intended as examples and should in no way be used in a production environment. The default security implementation provides an X.509 certificate authentication with CAS fallback. When using the default security module in a testing environment, the user must present a valid X.509 certificate, or a valid CAS login, in order to gain access to the Metrics Service. For each available security option, there is a specific `.xml` file which must be installed.

4.5.2.2 Operating the Metrics Service From Different Ports

Initial Metrics Service configuration is set up so that Tomcat can be run from a local installation.

Throughout this document, **`servername:port`** implies a **`localhost:8080`** or **`localhost:8443`** location. The example below shows how to set up the Metrics Service so that it can be used on **`5050/5443`** through the default security module.

To enable ports other than **`8080/8443`** while using Spring Security, the desired ports need to be explicitly edited in the Web server configuration file: **`\apache-tomcat-7.0.21\conf\server.xml`**.

Note: In the event that the Metrics Service is running on a server where a port number is already in use, the Metrics Service must run from a different port number. Two applications cannot bind to the same port.

1) For example, in Tomcat, change the port numbers in **`\apache-tomcat-7.0.21\conf\server.xml`**:

```
…
<Connector port="5050" protocol="HTTP/1.1"
              connectionTimeout="20000"
              redirectPort="5443" />
…
<Connector port="5443" protocol="HTTP/1.1" SSLEnabled="true"
              maxThreads="150" scheme="https" secure="true"
              keystoreFile="certs/keystore.jks" keystorePass="changeit"
              clientAuth="false" sslProtocol="TLS" />
…
```

a. Ports 5050 and 5443 are just examples and can be changed to whatever is needed.
If the Metrics Service was running on a server where a port number was already in use, the SHUTDOWN port must also be changed. To do this, change the port number in the Tomcat Web server configuration file **`\apache-tomcat-7.0.21\conf\server.xml`** to another port. In the following example the default shutdown port was changed from 8005 to 8006:

```
<Server port="8006" shutdown="SHUTDOWN">
```

b. Ensure that the port value used in the Web server configuration file match the port value used in **`\apache-tomcat-7.0.21\lib OzoneConfig.properties`**, which is shown below, displaying the default port and host information:

```
ozone.host = localhost
ozone.port = 5443
ozone.unsecurePort = 5050
```

2) Save both files.

3) Restart the Metrics Service server.

4.5.3   Custom Security Logout

The Metrics Service sample security plugins can perform single sign out if the user logged in using CAS authentication. PKI authentication is handled by the browser and requires that the user close the browser to completely sign out. To sign out from LDAP or a custom authentication, the system administrator must implement their own single sign out or instruct the user to close the browser after logout. Use the following lines in the Metrics Security Context file to invoke CAS's single sign out process.

Security context file:

```
<sec:custom-filter ref="casSingleSignOutFilter" after="LOGOUT_FILTER"/>
```

MetricsCasBeans.xml file:

```
<bean id="casSingleSignOutFilter" class="org.jasig.cas.client.session.SingleSignOutFilter"/>
<bean id="casSingleSignOutHttpSessionListener"
class="org.jasig.cas.client.session.SingleSignOutHttpSessionListener" />
```

4.5.4   Server Certificate Creation and Installation

Valid server certificates are needed for configuring the server to allow HTTPS authentication. Also, the OWF server acts as a client on the Metrics Service. Thus, it must be identified as a user in the authentication files. See the steps below for generating and installing self-signed server certificates.

4.5.5   Generating a New Self-Signed Server Certificate

To generate a new self-signed certificate, the Java **keytool** utility must be used. This keytool can be used to generate the public/private keys and signed certificates. Using a command window, navigate to the certs directory and execute the following command:

```
keytool -genkey -alias servername -keyalg rsa -keystore servername.jks
```

Note: Some systems do not default to the Java keytool. The keytool can be explicitly called by running the command directly from the JRE/bin directory.

The keytool **genkey** command will prompt a series of questions. The questions are listed below with example entries matching a server with the name www.exampleserver.org and a keystore password of *changeit.*

Enter keystore password: *changeit*

What is your first and last name? [Unknown]: *www.exampleserver.org*

Note: Make sure to enter the FULLY QUALIFIED server name. This needs to match the hostname of the machine exactly or the certificate will not work correctly.

What is the name of your organizational unit? [Unknown]: *sample organization unit*

What is the name of your organization? [Unknown]: *sample organization*

What is the name of your City or Locality? [Unknown]: *sample city*

What is the name of your State or Province? [Unknown]: *sample state*

What is the two-letter country code for this unit? [Unknown]: *US*

Is CN= www.exampleserver.org, OU= sample organization unit, O=sample organization, L= sample city, ST= sample state, C=US correct? [no]: *yes*

Note: When using an IP address as the Common Name (CN), an entry must be added to the Subject Alternative Name entry in the certificate. The better alternative to using an IP address is to add a name/IP pair to the hosts file and register the name as the CN.

The signed certificate must then be imported into a file to add to the JVM truststore (cacerts):

```
keytool -export -file servername.crt -keystore servername.jks -alias servername
```

4.5.6 Configuring For a Different Truststore/Keystore

For server-to-server calls (Metrics-to-CAS communications, for example) the newly created self-signed certificate should be imported into the truststore.

1) Export the certificate from the keystore into a file:

```
keytool -export -file servername.crt -keystore servername.jks -alias servername
```

2) Import the file into the truststore:

```
keytool -import -alias servername –keystore mytruststore.jks -file servername.crt
```

Modify the JVM Parameters that are used to start the web-application server in order to utilize the truststore referenced in step 2, shown above. If a Tomcat server is being used, the parameters can be found in the **setenv.bat** and **setenv.sh** scripts found within the **\apache-tomcat-7.0.21\bin** folder inside of the unpacked **Metrics-bundle-7-GA.zip**.

If an application server other than Tomcat is being used, the parameters will need to be added to the JVM parameters which are loaded when the application server is started.

Table 4: Custom JVM Parameters

Parameter	Note
`-Djava.awt.headless=true`	Only needed for Unix/Linux deployments
`D-javax.net.ssl.trustStore="%CATALINA_HOME%\certs\keystore.jks"`	Replace 'certs/keystore.jks' with the path and filename to the truststore
`-Djavax.net.ssl.keyStorePassword= changeit -server -Xmx1024m -Xms512m -XX:PermSize=128m -XX:MaxPermSize=256m`	Replace 'changeit' with the truststore's password (if applicable) and modify memory as needed.

3) Finally, the server configuration must be modified to utilize the new **keystore\truststore** in SSL. Below is the relevant section from the Tomcat configuration script found in **\apache-tomcat-7.0.21\conf\server.xml**:

```
<Connector port="8443"
protocol="HTTP/1.1"
SSLEnabled="true"
maxThreads="150"
scheme="https"
secure="true"
keystoreFile="certs/keystore.jks"
keystorePass="changeit"
clientAuth="false"
sslProtocol="TLS" />
```

4.5.7 Configuring OWF Certificate to Authenticate against Metrics

When using certificate authentication, the Metrics Service must identify the OWF server as an authorized user. To do this:

1) Generate a new self-sign certificate as described in section 4.5.5: Generating a New Self-Signed Server Certificate.

2) Add the server name to the authentication scheme including the **user.properties** file or any other authentication files such as LDAP.
   Note: The server name must have the "user" role associate with it.

5   Configuration

5.1   Default Configuration

The Metrics Service bundle is configured to run by default on localhost with a predefined set of users. In addition to **users.properties**, the Metrics Service provides two override files which are used to modify the default configuration.

To utilize an override file, place the individual file somewhere on the classpath of the server running the Metrics Service. When using the default Tomcat bundle, externalized configuration files should be placed in the folder **\apache-tomcat-7.0.21\lib**. By default, **MetricConfig.groovy** is located in **\apache-tomcat-7.0.21\lib**. The other override file, **CASSpringOverrideConfig.xml**, is located in the Metrics bundle at **\etc\override**. If using an application server other than Tomcat, copy the override files into the directory that will include them in the classpath for that specific application server.

The two optional override files are:

- **CASSpringOverrideConfig.xml**

- **MetricConfig.groovy**

Each of the override files is detailed in sections that follow.

5.2   Adding Users/Roles/Groups

At this time, the Metrics Service only stores users/roles/groups information in its database. Administrators may want to configure this information for future use. The addition of users, groups, and roles depends on the choice of security implementation. The following example outlines the procedures for adding users, groups, and roles to the sample OWF X.509-only, CAS-only, and X.509-with-CAS security modules:

Note: The sample security modules are included as examples and should NOT be used in a production environment.

```
…
testUser1=password,ROLE_USER,Test User 1,[group1;I am a sample Group 1 from
users.properties;test@gmail.com;active]
testUser2=password,ROLE_USER,Test User 2
testUser3=password,ROLE_USER,Test User 3
testAdmin1=password,ROLE_ADMIN,Test Admin 1,[group1;I am a sample Group 1 from
users.properties;test@email.com;active],[group2;I am a sample Group 2 from
users.properties;test2@email.com;active],[group3;I am a sample Group 3 from
users.properties;test3@email.com;inactive]
…
```

*Note: To have actual spaces between names and numbers, escape spaces using the "\"
character (do not include the quotation marks). Moreover, when using CAS or a
custom setup which employs anything other than X.509 authentication, the user
names MUST be entered in all lower case. This is a technical issue with Spring Security
and will be remedied in a future release.*

Edit **\apache-tomcat-7.0.21\lib\users.properties**:

1) Add a user(s) to the file in accordance with the following rules:

   a. Data Format:
      Username=password, role, display name,[group name, group description,
      group contact email, active/inactive status]

   b. All of the information for a single user, including group information,
      should be on a single line

   c. Multiple groups may be delimited by commas

   d. Group information is optional, and may be left out for any single given
      user

2) Save the file and restart the Metrics Service server.

Any user added to **users.properties** will be granted access to the Metrics Service
upon restart. Any user deleted from users.properties will be denied access to the Metrics
Service upon restart.

*Note: If a custom Web server is being used along with the provided example security,
the **users.properties** file can be copied to any directory that is on the classpath of
the webserver being utilized. For example, if using Jetty, the file can be copied to the
\<jetty root>\resources directory.*

To add users to any security module utilizing X.509 authentication, a PKI User
certificate that can be recognized by the Metrics Service must be generated.

5.3 Custom Configuration

Metrics Service externalized configuration files are **MetricConfig.groovy** and
CASSpringOverrideConfig.xml. When the Metrics Service is deployed to a non-
localhost environment, the externalized configuration files must be deployed and
modified. Those modification are explained in the individual sections about each file.

Use of a production quality database like Oracle or MySQL, instead of the default
HSQLDB, will require a change to the **MetricsConfig.groovy** file, detailed in the
following section.

5.3.1 MetricsConfig.groovy File

MetricConfig.groovy is a Metrics Service configuration file that allows an administrator to modify the database connectivity information see section 4.5.1 Database Setup. Once changes are made, restart the system to apply the changes. Developers comfortable with the Groovy language and the Grails application framework should be comfortable writing additional code for this file.

Listed below is an example of the file in its entirety:

```
environments {
    production {
        dataSource {
            dbCreate = "none"
            username = "sa"
            password = ""
            driverClassName = "org.hsqldb.jdbcDriver"
            url = "jdbc:hsqldb:file:MetricsDb;shutdown=true"
            pooled = true
            properties {
                minEvictableIdleTimeMillis = 180000
                timeBetweenEvictionRunsMillis = 180000
                numTestsPerEvictionRun = 3
                testOnBorrow = true
                testWhileIdle = true
                testOnReturn = true
                validationQuery = "SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS"
            }
        }
        //enable uiperformance plugin which bundles and compresses javascript
        uiperformance.enabled = true
    }
}

beans {


            //This block is equivalent to using an
org.springframework.beans.factory.config.PropertyOverrideConfigurer
            //See Chapter 14 of the Grails documentation for more information:
http://grails.org/doc/1.1/


}

//databasemigration settings
grails.plugin.databasemigration.updateOnStart = false


println('MetricsConfig.groovy completed successfully.')
```

5.3.2   metric-override-log4j.xml file

General logging can be enabled by editing the **Metric-override-log4j.xml** file which can be found in the **apache-tomcat-7.0.21\lib** directory:

```
<logger name="AuditOWFWebRequestsLogger" additivity="false">
  <level value="error" />
  <appender-ref ref="ozone-async" />
</logger>


<!-- For security logging, set this log level to "info". -->
<logger name="ozone.securitysample.authentication.audit.SecurityAuditLogger" additivity="false">
  <level value="info" />
  <appender-ref ref="ozone-async-audit" />
</logger>


<!—Add this to enable general Metrics Debug logging -->
<logger name="grails.app" additivity="false">
  <level value="trace" />
  <appender-ref ref="ozone-async" />
</logger>
```

Note: The **Metric-override-log4j.xml** *file shown above does not ship with the code shown at the bottom of the sample. However, it can be pasted into the file at an administrator's discretion in order to enable the logging of general server debug messages.*

To confirm that the log files are being written, examine the **apache-tomcat-7.0.21\logs** directory. Developers familiar with Log4j configurations should be comfortable with this file.

Note: Useful configurations and common requests are called out in comments in the file. For example, audit logging describing each user's Web calls can be enabled by setting **AuditOWFWebRequesterLogger** *and* **ozone.filter** *to logging level* **info***.*

Different third party libraries within the Metrics Service have also been called out so that administrators can easily modify logging levels.

5.3.2.1 Audit Logging

The Metrics Service includes an option to audit all user entries and exits from the system. The Metrics Service bundle ships with this feature enabled by default. The Audit Log tracks the following types of changes:

- Both successful and unsuccessful login attempts

- User Logout Events:

  o   A user logging out on purpose

  o   A session times out

Note: References to the CAS and Metrics Service must match the settings of the current installation.

5.3.2.2 Configuring Audit Log Levels

Metrics Service logging levels can be set by editing the **/apache-tomcat-7.0.21/lib/metric-override-log4j.xml** file, which ships with the Metrics Service bundle. To change the audit log level, open the file and search for the following section:

```
<!-- For security Audit logging, set this log level to "info". -->
 <logger name="ozone.securitysample.authentication.audit.SecurityAuditLogger" additivity="false">
    <level value="info" />
    <appender-ref ref="ozone-async-audit" />
</logger>
```

The log statement shown above, **ozone.securitysample.authentication.audit.SecurityAuditLogger**, captures login and logout events. But, in debug mode, the logger will record authentication credentials, such as SubjectDN, IssuerDN and validity dates for X.509 Certificate logins, as well as CAS credentials for CAS login. When deploying a custom security plugin, use the logger shown above to capture all login events for the system. This logger supports "info", "debug" and "off" levels, as described in the section below.

When distributed, the default log level is set to "info." Audit logging supports the following three log levels:

1) **Info -** The minimal amount of information concerning a database change is logged and consists of the following fields within the log statement:

   a) **Log Level -** This will set to "INFO" or "DEBUG" while logging is turned on.

   b) **Log Date/Time -** The date and time that an event occurred. The time pattern can be changed by editing the **layout** tag of the **ozone-audit-log appender**.

   c) **Remote IP -** The IP address of the remote client that triggered the log event.

   d) **Session ID -** The HTTP request session ID of the log event.

   e) **User -** The username of the authenticated user that caused the log event.

   f) **Event Type -** USER LOGIN or USER LOGOUT.

   g) **Event Message -** A description of the event.

2) **Debug -** This level provides all of the same information as the INFO level, but provides more detail in the event message.

3) **Off -** No login events will be logged.

When the audit log levels are modified, it is not necessary to restart the Metrics Service, as the server has a log-change listener which periodically (every 3 minutes, by default) checks for log file changes and reloads the changes.

5.3.2.3 Login Events

When using the sample pluggable security modules included in the Metrics Service bundle, successful login authentication is captured in **ListenerBeans.xml**.

A login failure will occur and be recorded in the log if a user has a valid PKI certificate but the associated username is not registered as a valid user within the Metrics Service. A failed login produces the following log statement at the info level:

```
INFO [02/15/2011 15:24:04 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]: LOGIN FAILURE - ACCESS
DENIED with FAILURE MSG [Login for 'testAdmin1' attempted with authenticated credentials [CERTIFICATE
LOGIN]; However, the Provider was not found. Access is DENIED.]
```

A failed login produces the following log statement at the debug level:

```
DEBUG [02/15/2011 15:27:18 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]:LOGIN FAILURE - ACCESS
DENIED with FAILURE MSG [Login for 'testAdmin1' attempted with authenticated credentials [CERTIFICATE
LOGIN >> Signature Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5]; Subject:
[EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US];
Validity: [From: Thu Feb 04 13:58:52 EST 2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer:
[EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; ];
However, the Provider was not found. Access is DENIED. Login Exception Message: [No AuthenticationProvider
found for org.springframework.security.web.authentication.preauth.PreAuthenticatedAuthenticationToken]]
```

A successful PKI Certificate login produces the following log statement at the info level:

```
INFO [02/15/2011 15:39:13 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]: LOGIN SUCCESS - ACCESS
GRANTED USER [testAdmin1], with DISPLAY NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with
ORGANIZATION [Test Admin Organization], with EMAIL [testAdmin1@nowhere.com]with CREDENTIALS [CERTIFICATE
LOGIN]
```

A successful PKI Certificate login statement produces the following log statement at the debug level:

```
DEBUG [02/15/2011 15:42:10 -0500] IP: 127.0.0.1 User: testAdmin1 [USER LOGIN]:LOGIN SUCCESS - ACCESS
GRANTED USER [testAdmin1], with DISPLAY NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with
ORGANIZATION [Test Admin Organization], with EMAIL [testAdmin1@nowhere.com] with CREDENTIALS [CERTIFICATE
LOGIN >> Signature Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5]; Subject:
[EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US];
Validity: [From: Thu Feb 04 13:58:52 EST 2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer:
[EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone, L=Columbia, ST=Maryland, C=US]; ]
```

5.3.2.4 Logout Events

Logout events are logged by the **ozone.securitysample.authentication.audit.SecurityAuditLogger**. This logger supports the same two levels of logging: info and debug. The latter provides more detailed information about each logout event.

Below is a typical user-initiated logout event which has been saved as a log entry, with the log level set to info:

```
INFO  [02/03/2011 16:13:35 -0500] IP: 127.0.0.1 SessionID: 8ki2ttimdxc User: testAdmin1 [USER LOGOUT]:
```

Below is a typical user-initiated logout event which has been saved as a log entry, with the log level set to debug:

```
DEBUG [02/03/2011 15:59:53 -0500] IP: 127.0.0.1 SessionID: 1tjefhsxz1x6t User: testUser1 [USER SESSION
TIMEOUT] with ID [2], with EMAIL [testUser1@nowhere.com], with ACCOUNT CREATED DATE [02/03/2011 15:58:50 -
0500], with LAST LOGIN DATE [02/03/2011 15:58:50 -0500]
```

A user can also be forced to logout when their session times out. Below are both info and debug log statements:

```
INFO  [02/07/2011 10:08:21 -0500] IP: 127.0.0.1 SessionID: 1b4nvaqnb0qx8 User: testAdmin1 [USER SESSION
TIMEOUT]
```

```
DEBUG [02/07/2011 10:24:21 -0500] IP: 127.0.0.1 SessionID: d0pq3g4xguv3 User: testAdmin1 [USER SESSION
TIMEOUT] with ID [1], with EMAIL [testAdmin1@nowhere.com], with ACCOUNT CREATED DATE [02/07/2011 10:23:18
-0500], with LAST LOGIN DATE [02/07/2011 10:23:18 -0500]
```

5.3.2.5 *Auditing Login Attempts From Custom Security Modules*

Audit logging of custom security modules can be achieved by adding logging capabilities via security authentication event listeners in the **/owf-security/MetricSecurityContext.xml** file, as in the case of the **ozone.securitysample.authentication.listener.AuthenticationSuccessListener** and **ozone.securitysample.authentication.listener.AuthenticationFailureListener** beans (both of which implement **org.springframework.context.ApplicationListener<org.springframework.security.authentication.event.AbstractAuthenticationEvent>**) located in **/apache-tomcat-7.0.21/lib/ozone-security-beans/ListenerBeans.xml** and shown below:

```
<!-- REQUIRED FOR AUDIT LOGGING OF AUTHENTICATION FAILURES -->
    <bean id="authenticationFailureListener"    class="ozone.
securitysample.authentication.listener.AuthenticationFailureListener"/>

<!-- REQUIRED FOR AUDIT LOGGING OF AUTHENTICATION SUCCESS -->
    <bean id="authenticationSuccessListener"           class="ozone.
securitysample.authentication.listener.AuthenticationSuccessListener"/>
```

Once an **onApplicationEvent** event of type **InteractiveAuthenticationSuccessEvent** is fired in the Spring Security framework, the **authenticationSuccessListener** bean will be used to log the

details of the successful authentication. Moreover, once an **onApplicationEvent** event of type **AbstractAuthenticationFailureEvent** is fired in the Spring Security framework, the **authenticationFailureListener** bean will be used to log the details of the failed authentication.

5.4 Server Settings

All references to the CAS and Metrics must match the settings of the current installation. Based on the settings in **OzoneConfig.properties**, the variables (e.g. ${ozone.host}) are filled in at runtime.

Table 5: MetricsCasBeans.xml Server Settings

Property	Purpose	Example
casProcessingFilterEntryPoint. loginUrl	Must point to the CAS login page.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverLoginLocation} (e.g. https://servername:port/cas/login)
serviceProperties. Service	Must point to the Metrics Service web server.	https://${ozone.host}:${ozone.port}/${ozone.cas.Metrics.jSpringCasSecurityCheckLocation} (e.g. https://servername:port /Metrics/j_spring_cas_security_check)
ticketValidatorFactory. casServiceUrl	Must point to the CAS server.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverName} (e.g. https://servername:port/cas)
ticketValidatorFactory. proxyCallbackUrl	Must point to the Metrics Service web server.	https://${ozone.host}:${ozone.port}/${ozone.cas.Metrics.serverSecureReceptorLocation} (e.g. https://servername:port/prefs/secure/receptor)
OzoneLogoutSuccessHandler/ constructor-arg/index=1	Must point to the CAS logout page.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverLogoutLocation} (e.g. https://servername:port/cas/logout)

5.4.1 CASSpringOverrideConfig.xml File

CASSpringOverrideConfig.xml, located in the **/etc/override directory**, is a Spring framework override file and should be deployed to the same server as **Metrics.war**.

If using **CASSpringOverrideConfig.xml** from a server without connectivity to the outside Internet, copy the **CASSpringOverrideConfig.xml** to the **\apache-tomcat-7.0.21-6.1.11\lib** folder. From there, the file will use classpath references to override the online Springframework URLs that the header points to during start up.

5.4.2 JVM Memory Settings

Adjusting a server's memory settings can increase performance or resolve **PermGen OutOfMemoryError** occurrences. To adjust memory settings:

1) In the Tomcat start script (**apache-tomcat-7.0.21\bin\setenv.sh** or **setenv.bat**,) set the initial **PermGen** size to at least 256 MB. This can be accomplished by adding **-XX:PermSize=256m** to the Java options. If more server memory is available, increasing this **PermGen** size may increase performance.

2) Set the maximum **PermGen** size to at least 384 MB. This can be accomplished by adding **-XX:MaxPermSize=384m** to the Java options. If you have more memory available on your server increasing this **PermGen** size may increase performance.

3) If you have a server JVM, point to it when starting Java to increase performance. To do this, navigate to **serverjvm.dll** or add the server flag **-server** to the deployment command line.

6 Metrics Service Security

The Metrics Service allows an administrator to customize the type of security that is implemented for authentication and authorization. The Metrics Service uses a pluggable Spring Security 3.0.2 solution and ships with sample security plugins that can be used as a basis for building a custom security plugin. Familiarity with Spring Security will help administrators customize the Metrics Service.

6.1 Basic Security Concepts and the Metrics Service

While this guide is not intended as a comprehensive guide to basic security concepts, Web security, or Spring Security, there are a few key concepts that must be understood in order to use the sample Metrics Service security plugins and the Metrics Service security plugin architecture.

First are the twin concepts of authentication and authorization, known colloquially as auth & auth. Authentication essentially means providing proof that the user is exactly who they are presenting themselves to be. Some authentication techniques include a username/password combination, an X.509 certificate, a CAC card and card reader, or various bioMetrics solutions. Authorization, on the other hand, is determining the specific access rights that an individual user should have. Consider the following:

- "Bill is allowed to log into the system – prove that you are Bill," is a matter of authentication.

- "Bill has access to resources," is a question of authorization.

By necessity, authentication occurs before authorization. Once authentication is satisfied, the Metrics Service moves to authorize. It has two authorization concepts at this time. First, the Metrics Service needs to know whether or not a user has administrative access via ROLE_ADMIN or is only a regular user via ROLE_USER.

6.2 Production Deployments

The samples included with the Metrics Service are not production quality samples. They are intended to provide examples on how to easily integrate various security solutions with the Metrics Service, not to provide a comprehensive security solution out of the box or a comprehensive tutorial on Spring Security. It is expected that each organization using the Metrics Service will examine its security guidelines and enterprise-wide authentication/authorization solutions and produce a Metrics Service security plugin that is both secure and meets its standards. That solution can then be shared among Metrics Service deployments within the organization.

Most of the examples provided contain various obvious security hazards—for example, the X.509-only and CAS + X.509 plugins contain a list of usernames, roles, and user groups on the hard drive in plain text in a properties file. **The CAS+X.509 file contain passwords in plain text. These are undeniable security hazards.** Keep this in mind when using the samples.

6.3   Sample Plugin Summary

The Metrics Service ships with three simple sample security plugins that are described in this section:

6.3.1   Default Authentication: CAS + X.509

`MetricSecurityContext.xml` – This contains the default security implementation for the Metrics Service. It uses a PKI certificate for authentication. If no certificate is provided, it redirects the user to login using CAS as a fallback. CAS stores valid usernames and passwords in a **`users.properties`** file on the server. Once the user has been authenticated, the authorization information is provided in the same properties file, **`users.properties`**.

To use a non-default security configuration for authentication, replace the active security-based **`.xml`** files, (for example, **`\apache-tomcat-7.0.21\lib\MetricSecurityContext.xml`**) with the **`\owf-security\MetricSecurityContext_cert_only.xml`** file.

6.3.1.1   Customizing CAS

`OzoneConfig.properties` proxies some of the CAS properties that can be customized by specific organizations. Update the corresponding values in **`OzoneConfig.properties`** to change where the default CAS server, CAS login and CAS logout point.

6.3.2   X.509 Only Security

`MetricSecurityContext_cert_only.xml` – This contains the X.509-only security implementation for the Metrics Service. It uses a PKI certificate for authentication. If no certificate is provided, the user is denied access to system. The users' roles (admin and user) are stored in **`users.properties`** for authorization.'

6.3.3 X.509 with LDAP

MetricSecurityContext_cert_ldap.xml – This contains an X.509/LDAP security implementation that uses X.509 for authentication and then performs an LDAP-based lookup to determine the user's authorization. Authorization includes the user's role (**ROLE_ADMIN** or **ROLE_USER**).

The **owf-security-project.zip** directory contains the following supporting resource in **/src/main/resources/conf**.

- **apache-ds-server.xml** — A sample.xml file used by Apache Directory Server (ApacheDS, an open-source LDAP v3-compliant embeddable directory server) that sets up the initial directory service partitions with the test data.

- **testUsers.ldif** — An LDAP Data Interchange Format test file that can be imported to set up test entries that match the certificates bundled with the Metrics Service. This test data includes testUser1 and testAdmin1, roles **ROLE_USER** and **ROLE_ADMIN**, and two example groups, group1 and group2. It is designed to work with the sample user PKI certificates that ship with the Metrics Service.

6.4 Installing the Security Module

The **MetricSecurityContext** files in the **/owf-security** directory offer multiple examples of security options. These are intended as examples and should in no way be used in a production environment. As mentioned previously, the default security implementation provides an X.509 certificate authentication with CAS fallback. When using the default security module in a testing environment, the user must present a valid X.509 certificate, or a valid CAS login, in order to gain access to the Metrics Service.

For each available security option, there is a specific **.xml** file which must be installed. Installing a new security module is accomplished in just a few simple steps:

Note: The following instructions act as a summary for installing individual security modules. Depending on the module being used or tested, module-specific instructions may be needed. See **\owf-security\owf-security-project.zip\readme.txt** *for the installation details specific to each module type. Additionally the summary instructions below assume that the default installation is being used with Tomcat as the application server/container.*

1) Stop the application server. An administrator can accomplish this by double-clicking the **\apache-tomcat-7.0.21\bin\shutdown.bat** or **\shutdown.sh** file, depending on the operating system in use.

2) Delete any security-based **.xml (MetricSecurityContext*.xml)** files that might currently be present in the **\apache-tomcat-7.0.21\lib** directory.

3) Copy the appropriate **.xml** file from **\owf-security** to the application server's class path. When running Tomcat, the classpath is the **\apache-tomcat-7.0.21\lib** directory.

4) Remove **CASSpringOverridesConfig.xml** from **\apache-tomcat-7.0.21\lib** if it was deployed there.

5) Remove CAS from the Metrics Service instance if switching to a security plugin that does not use CAS by deleting **cas.war** and the **cas** directory if it exists from **\apache-tomcat-7.0.21\webapps**.

6) Restart the application server by double-clicking either **\apache-tomcat-7.0.21\start.bat** or **\start.sh** file, depending on the operating system in use.

Note: The following instructions describe how to remove CAS from the Metrics Service instance. This should only be done if switching to the X.509 sample, the X.509+LDAP sample, or a custom plugin that does not require CAS.

6.5 X.509-Only Specific Instructions

The **MetricSecurityContext_cert_only.xml** file eliminates CAS as a fallback to authentication. If the user does not present a valid X.509 certificate, they will be denied access to the system. Authorization is provided by the **users.properties** file.

To use this security plugin, replace the active security-based **.xml** file (e.g., **\apache-tomcat-7.0.21\lib\MetricSecurityContext.xml**) with the **MetricSecurityContext_cert_only.xml** file, which can be found in the **\owf-security** directory. Follow the directions above to stop the Metrics Service, remove CAS, and restart.

6.6 X.509/LDAP

The **MetricSecurityContext_cert_ldap.xml** file provides X.509 client authentication with an LDAP-based lookup to determine the user's authorization. The default configuration attempts to connect to a local installation of Apache Directory Server on port 10389 using the default system account. It determines the user's authorization by searching on the full, distinguished name presented in the X.509 certificate.

Sample configuration files are provided to set up an Apache Directory Server with user information that matches the X.509 certificates provided with OWF, including a server configuration **XML** file and an LDAP Data Interchange Format file **(*.ldif)** which loads users to match the distinguished names in the certificates. For more information about LDAP, refer to http://directory.apache.org/.

Included is a sample Apache DS **server XML** file called **\owf-security\owf-security-project\src\main\resources\conf\apache-ds-server.xml**. It adds the partition owf-1 to Apache DS. To do so, it adds the following line of **XML** to the XPATH **spring:beans\defaultDirectoryService\partitions**:

```
<jdbmPartition id="owf-1" suffix="o=Ozone,l=Columbia,st=Maryland,c=US" />
```

It is also necessary to load the sample data into the directory service. The OWF team has provided a sample **LDIF** file, called **\owf-security\owf-security-project\src\main\resources\conf\testUsers.ldif**.

Note: Downloading the Apache Directory Studio may be helpful.

It is also straightforward to modify how the LDAP search is conducted for both user roles and user groups. In order to run the plugin with the default data no adjustment is required. However, to modify the plugin to run off of a different data set, adjust **ozone-security-beans\LdapBeans.xml**. It is recommended that the administrator get the plugin working from the default data set before trying to migrate to a different data set by modifying the LDAP queries.

To use the X.509(cert)/LDAP security implementation, replace the provided **MetricSecurityContext.xml** file with **MetricSecurityContext_cert_ldap.xml**, and follow the directions in section to stop and restart the server and to remove CAS.

Appendix A   Contact Information

A.1   Discussion Group

The OZONE Developers Discussion Group is hosted through Google Groups at http://groups.google.com/group/ozone-developers. This forum is for the distribution of release announcements, Q&A related to OWF and for additional inquiries about widgets and features being developed across the user base. To access the group, request an invitation at http://groups.google.com/group/ozone-developers or contact the Community Support Team at goss-support@owfgoss.org.

A.2   Additional POCs

For information about the OZONE Widget Framework or access to its resources, please email goss-support@owfgoss.org. Additional resources can be found at http://owfgoss.org.