

Ozone Marketplace Configuration Guide

DOD GOSS

Exported on Oct 16, 2018

Table of Contents

1	Introduction	3
1.1	Objectives	3
1.2	Document Scope	3
1.3	Related Documents	3
2	Overview	4
2.1	Purpose	4
2.2	Basic Architecture	4
2.3	Components	5
2.3.1	Apache Tomcat	5
2.3.2	In-memory HSQLDB	5
2.3.3	Store Web Application	5
2.3.4	Pluggable Security	5
2.3.5	Store Security Project	5
3	Installation & Usage	6
3.1	Bundle Overview	6
3.2	Supported Browsers	6
3.3	Runtime Dependencies	6
3.4	Relational Database Management System (RDBMS)	6
3.5	Server Usage	7
3.5.1	Windows start script	7
3.5.2	Linux start script	7
4	Server Configuration	8
4.1	Changing the Server Ports	8
4.2	Adding Users and Assigning Roles	9
4.2.1	Adding Multiple Roles to a User	10
4.3	Creating and Updating User Accounts	10
4.4	Server Settings	10
4.4.1	Session Timeout	10
4.4.2	Timeout Warning	11
4.4.3	Notifications	11
4.4.4	Scheduled Imports	12
4.4.5	Synchronization with OWF	14
5	Security Configuration	16
5.1	Security Setup	16
5.2	Installing the Security Configuration	17
5.2.1	User Certificate Prompt	17
5.3	CAS Settings	17
5.4	Certificates (PKI) Configuration	17
5.4.1	Server Certificate Creation and Installation	18
5.4.2	Installing User PKI Certificates	19
6	Database Configuration	21
6.1	Overview	21
6.2	Supported Databases	21
6.3	General Database Settings	21
6.3.1	Application Configuration File	22
6.3.2	Example Database Settings and Properties	22
6.4	Oracle Setup	25
6.5	MySQL Setup	26
6.5.1	Configuring MySQL JDBC to use SSL	27
6.6	PostgreSQL Setup	30
6.7	MS SQL Server Setup	31

1 Introduction

1.1 Objectives

This guide covers topics relevant to installing, configuring, and administering an OZONE Store. Similar to an online storefront like the Apple App Store or Google Play, the OZONE Store operates as a thin-client registry of applications and services. It enables users to create, browse, download and use a variety of applications or software components that are known as listings. Like commercial software stores, it offers quick and easy access to a variety of listings including—but not limited to—OZONE Apps, App Components, plugins, REST & SOAP services, Web Apps and desktop applications.

1.2 Document Scope

This guide is intended for system administrators of a Store and for Developers who wish to extend a Store beyond the default look. A System Administrator is someone who installs, optimizes and maintains the Store and sets up user authentications and authorizations. A Developer is understood as someone who is comfortable unpacking and packing WAR (.war) files, editing JavaScript (.js) files, Cascading Style Sheets (.css) and editing customized configuration files.

Note: Throughout this guide, there are instances where the swapping of image files is discussed. A .png file can be replaced with .jpgs, .gifs or other standard image files.

1.3 Related Documents

Document	Purpose
Quick Start Guide	Setting up and integrating the Store into OWF.
Configuration Guide	Modifying Default Settings, Security, Database Settings, Upgrading to a newer Store version.
User Guide	Searching, Creating and Editing Listings, Adding Comments, Ratings, Navigating a Store, Scorecards Explanation of Store Elements.
Administrator's Guide	Importing / Exporting Store Data, Adding Affiliated Stores, Approving Listings, Creating Types, States, Categories, and Custom Fields, Scorecard Configuration.
Developer's Guide	Custom Security Modules, Creating and Editing Themes
Release Notes	Major and minor changes for the current release.

2 Overview

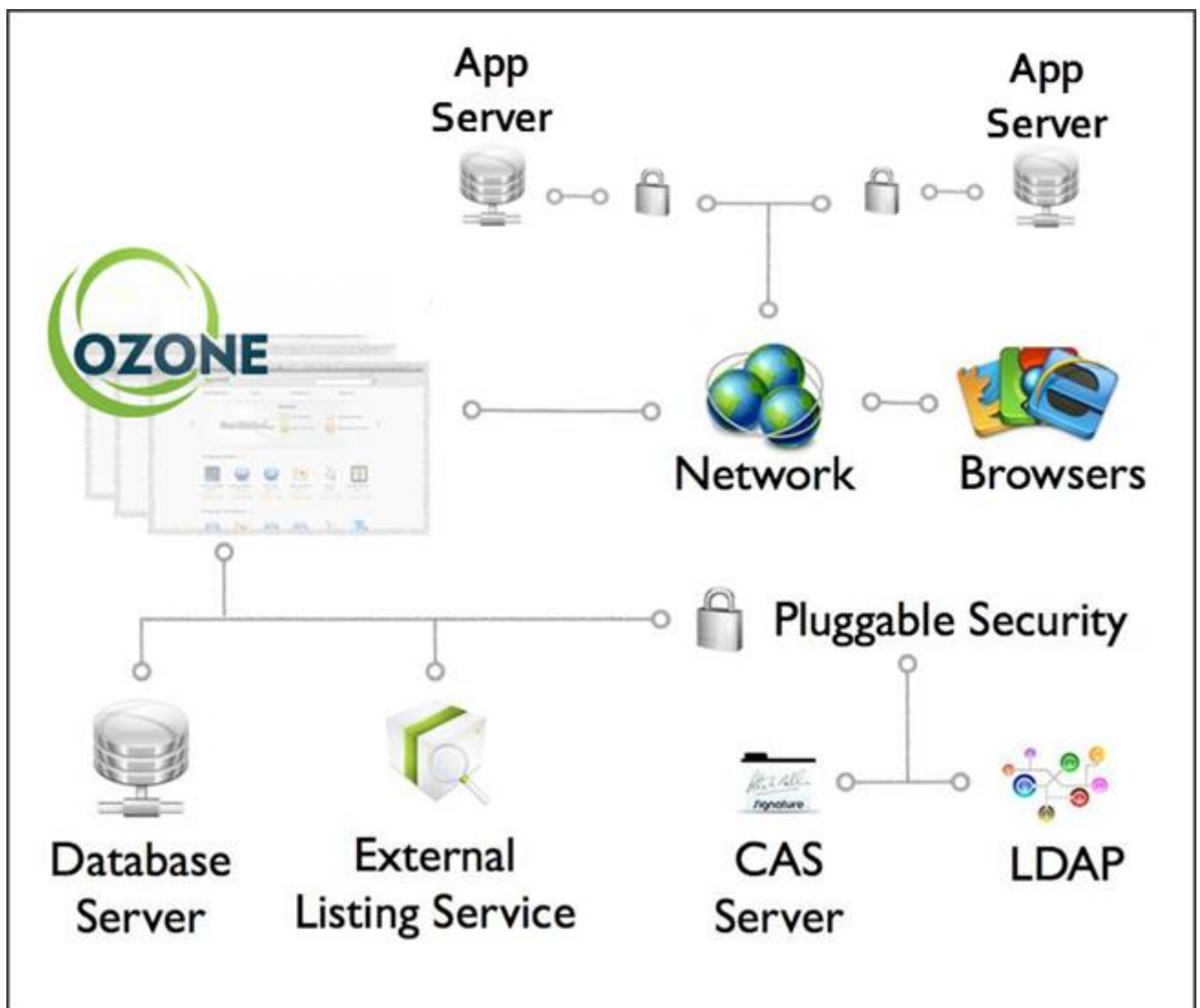
2.1 Purpose

The Store is a thin-client registry of applications and services akin to a commercial storefront. Like the Apple Store or Android Market, users can create and interact with web components or listings such as OZONE Apps, App Components, plugins, REST & SOAP services, Web Apps, and more.

2.2 Basic Architecture

The Store consists of a number of components that were designed to be independently deployed or located on the same server. The simplest deployment scenario places them all on the same physical machine. These components are shown in detail below:

Figure 1 Basic System Architecture Diagram



2.3 Components

2.3.1 Apache Tomcat

Tomcat is a Web application server configured to host a Store.

2.3.2 In-memory HSQLDB

The Store ships with an in-memory HyperSQL database for testing and development purposes. A disk-based RDBMS such as Oracle or MySQL should be used in production.

2.3.3 Store Web Application

marketplace.war – is the Store Web application archive

2.3.4 Pluggable Security

The Store does not provide a security module suitable for a production environment as delivered. Instead, a flexible, pluggable security framework, based on Spring Security, has been included to allow each organization to design and develop its own custom security solution. Find detailed instructions in section Security Setup.

2.3.5 Store Security Project

ozone-security-project.zip – This bundle (located in /ozone-security) contains the source code, configuration files and library files needed to build the security files which are used by the Store. Additionally, the gradle build script is included for building and creating the sample security project in the target directory, called ozone-security.jar. This file contains all six security samples which are used by the security .xml files. For more details see the README.TXT, found in the .zip file.

3 Installation & Usage

3.1 Bundle Overview

The distribution bundle of the Store consists of a .zip file containing the necessary components to set up and run the Store in a development or testing environment. The latest bundle version can be downloaded from <https://github.com/ozoneplatform/omp-marketplace/releases>.

The default configuration provided in the distribution bundle is **NOT** production-ready.

Additional configuration is necessary prior to production deployment.

The bundle contains the following:

- Apache Tomcat v8.5 (Java Web container)
- The Store Web application (marketplace.war)
- Start-up scripts (start.sh, start.bat) and default configuration to run the application under Tomcat in SSL mode.
- Sample PKI Certificates for SSL (both user certificate and server certificates)
- Example application and security configuration files

3.2 Supported Browsers

The Store is fully supported and tested with the following browsers:

Browser	Version(s)
Microsoft Internet Explorer	11
Microsoft Edge	38
Google Chrome	43
Mozilla Firefox	57

3.3 Runtime Dependencies

The Store requires the Java Runtime Environment (JRE) v1.7 or higher.

3.4 Relational Database Management System (RDBMS)

The Store bundle includes an in-memory HyperSQL database for testing and development purposes. It is not recommended for production use.

A production deployment should use a more robust RDBMS, such as PostgreSQL, MS SQL, MySQL, or Oracle.

3.5 Server Usage

Batch files and shell scripts to start the Tomcat server and the Store web application are included in the bundle under the tomcat folder.

The first time running the bundle, the database must be initialized using the `init` flag.

3.5.1 Windows start script

Usage:

```
start.bat [/dev] [/init] [/db database]
  /init      Pre-populate the database with the initial data (only use on
the first launch!)
  /db        Use the selected database configuration
database     h2          - Embedded H2 file-based database (default)
              pg          - PostgreSQL
              mysql       - MySQL
              oracle      - Oracle RDBMS
              mssql       - Microsoft SQL Server
  /dev       Start server in DEVELOPMENT mode
```

3.5.2 Linux start script

Usage:

```
./start.sh [dev] [init] [db [--mysql | --h2 | --pg | --oracle] ]
  init      Pre-populate the database with the initial data (only use on the
first launch!)
  db        Takes an additional parameter for type of database to be used: [-
--mysql | --h2 | --pg | --oracle]
  dev       Start server in DEVELOPMENT mode
```

4 Server Configuration

Customize the Store to run in a variety of environments. The following sections detail how to change default ports and security setups.

For information about customizing database installations, see section: [Database Configuration](#).

For additional information about setting up custom security, see section: [Security Configuration](#).

For administrators or developers who wish to create custom security implementations for production environments, the sample security package at \ozone-security\ozone-security-project.zip can be used as a starting point. The .zip file contains source code complete with build scripts.

4.1 Changing the Server Ports

The initial Store configuration is set up so that Tomcat can be run from a local installation. Throughout this document, “servername:port” is used to reference the server name and port number. For example purposes, this document uses “localhost:8080” or “localhost:8443.” The example below shows how to set up a Store so that it can be used on ports 5050/5443 through the default security module.

To enable ports other than 8080/8443 while using Spring Security options, the desired port numbers need to be explicitly named in the following files:

- \tomcat\lib\ozone\marketplace\OzoneConfig.properties
Ozone Marketplace application configuration file
- \tomcat\conf\server.xml
Tomcat web server configuration file

In the event that a Store is running on a server where a port number is already in use, it must run from a different port number and use a different shutdown port. One server cannot run two applications from the same port.

1. Shutdown the Tomcat Web server via the appropriate stop command found in \tomcat\bin.
2. Open \tomcat\lib\ozone\marketplace\OzoneConfig.properties and change the port to the new port mapping (This file is referenced from other configuration files.).
3. Save the file.
4. Change the port numbers in the Tomcat Web Server configuration file \tomcat\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="conf/keystore.jks"
```



```

keystorePass="changeit"
clientAuth="want" sslProtocol="TLS" />
...

```

- a. Ports 5050 and 5443 are just examples and can be changed to whatever is needed. Ensure that the port value used in the Web server configuration file match the port value used in the `OzoneConfig.properties` file. In the following example, the port numbers in `OzoneConfig.properties` were updated:

```

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

```

- b. If a Store is running on a server where a port number was already in use, the Store SHUTDOWN port must also be changed. To do this, change the port number in the Tomcat Web server configuration file `\tomcat\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">

```

5. Save `server.xml` and start the Tomcat Web Server with the appropriate start command, found in `\tomcat`.

4.2 Adding Users and Assigning Roles

The addition of Users and assignment of Roles in the Store varies and should be customized based on individual enterprise authentication and authorization infrastructure. The Store supports three Roles:

- **ROLE_USER** – Standard Store user, with no administrator privileges.
- **ROLE_ADMIN** – Store administrator role.
- **ROLE_EXTERN_ADMIN** – Used by external application to allow greater administration privileges for listings that were generated externally.

The following example outlines the procedures for adding Users and Roles to the default sample security module that ships with the Store Bundle.

The sample security module is included as an example and should NOT be used in a production environment.

1. Edit `/tomcat/lib/ozone/marketplace/users.properties`:

```

...
testUser1=password,ROLE_USER,Test User 1,Test 1
Org,testUser1@nowhere.com
testAdmin1=password,ROLE_ADMIN,Test Admin1,Test Admin
Org,testAdmin1@nowhere.com
testAdmin2=password,ROLE_EXTERN_ADMIN,External Admin, Test Admin
Org, testAdmin2@nowhere.com
...

```

2. Add user/users to the file in accordance with format as shown:
username=password, role, display name, organization, email
3. Save the file and restart the server.

Any user added to the `users.properties` file will be added to the database upon the user's initial login.

4.2.1 Adding Multiple Roles to a User

The Store allows for individual Users to have multiple Roles within the `users.properties` file.

1. Edit `/tomcat/lib/users.properties`. For example:

```
...
testUser1=password,ROLE_USER:ROLE_EXTERN_ADMIN,Test User 1,Test 1
Org,testUser1@nowhere.com
...
```

2. Add user(s) to the file in accordance with format as shown:
username=password, role1[:roleN:roleN+1], display name,
organization, email

4.3 Creating and Updating User Accounts

Upon first sign-in (and regardless of the security model in use), the Store automatically creates a User's profile. The Marketplace contains a Profile object which contains the following settings.

- display name
- email address
- last date/time sign in (which is updated after every sign-in)

Administrators can update a User's display name and email address from the Profile section of the Administration Interface (see the Administrators Guide for details).

4.4 Server Settings

Located in the `/tomcat/lib/config` directory you will find 5 configuration files that allow for support of the following databases: H2 (embedded database), PostgreSQL, MySQL, MS SQL Server, and Oracle.

These configuration files can be placed anywhere on the classpath (which is a Web server-dependent setting) and must be on the same server where `marketplace.war` has been deployed.

The following sub-sections include specific instructions for modifying configurable settings that are housed within the `ozone-marketplace_*.yaml` files.

To change default settings, go to `/tomcat/lib/config/ozone-marketplace_*.yaml` and replace the existing values with the desired values. Once finished, restart the Store server.

4.4.1 Session Timeout

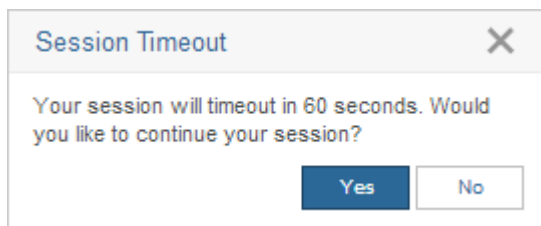
By default, Marketplace will allow for 120-minutes of inactivity before the session performs a timeout operation. However, an administrator can change the length of time a user may remain inactive prior to the session actually "timing out."

1. Set the “`httpsession.timeout`” value to the number of inactive minutes allowed before a session timeout occurs.
2. Save the changes to the configuration file in `/tomcat/lib/config`.
3. Restart the Store server.

4.4.2 Timeout Warning

The Store automatically warns users of a pending session timeout by launching a 120-second (image taken at the 60-second mark) countdown in warning window, as shown below:

Figure 2 Timeout Warning Dialog



Two-minutes prior to a Store timeout, a Timeout Warning window will launch, notifying the user of the pending timeout. A user can restart the session timer by clicking the Yes button.

4.4.3 Notifications

The OWF Notifications system can connect an OWF and Store to an XMPP server through which it can receive information about app components. This feature was originally designed to be used by administrators to monitor the changelogs' of Store listings, but may be used to allow any app components to communicate with OWF users. App components that connect to this XMPP server can use it to publish notification messages for consumption by OWF users. Messages published to the chat room must follow a specific format that designates both the message content and the OWF usernames of the intended recipient users. The OWF server will then receive the messages from the chat room and distribute them to individual users.

To configure the Store to use notifications, follow these steps:

1. Create a room on an XMPP server that serves as a repository for alerts to/from OWF and the Store, respectively.
2. Create user accounts on the XMPP server for the Store and OWF service notifications.
3. Use the XMPP server information to populate the XMPP Settings in the Notifications Configuration files in OWF and the Store. Find OWF instructions in the OWF Configuration Guide. The Store fields are defined below:

Use the user accounts and their passwords from step two as the Username and Password fields in the Notifications fields in your selected YAML configuration file (i.e. `ozone-marketplace_*.yaml`)

- a. XMPP Settings
 - i. username – The domain name for the XMPP server that administers notifications.
 - ii. password – The domain password for the XMPP server that administers notifications.
 - iii. host – The hostname of the XMPP server.
 - iv. room – The XMPP chat room that receives notifications.
 - v. port – The TCP port the XMPP server uses.

- b. Enabled – Turn on or turn off notifications.
4. Create a Public URL for the Store that will serve as the publicly accessible URL for the Store instance.

To access the Store Public URL setting in the user interface: Click the drop-down user menu, click Configuration Pages, click Application Configuration and then click Additional Configurations.

5. Notifications should be enabled and accessible to users.

4.4.4 Scheduled Imports


Scheduled Import allows you to routinely query another Store looking for new or updated listings. If the query finds new or updated listings, your Store will add them or update existing listings.



Modify the `/tomcat/lib/config/ozone-marketplace_*.yml` configuration file to create Scheduled Imports by establishing connections with other Stores and scheduling routine query times:

```
marketplace:
  scheduledImports:
    - name: "Test Import"
      enabled: true
      url: "https://localhost:8443/marketplace/public/exportAll"
      keyStore:
        file: null          # null value defaults to the
        'javax.net.ssl.keyStore' system property
        password: null      # null value defaults to the
        'javax.net.ssl.keyStorePassword' system property
      trustStore:
        file: null          # null value defaults to the
        'javax.net.ssl.trustStore' system property
      partial: true
      frequency:
        unit: "hours"       # one of: "minutes", "hours", "days"
        count: 1
```

4.4.4.1 To create or update a Scheduled Import

1. Open the current `/tomcat/lib/config/ozone-marketplace_*.yml` configuration file.
2. Add or modify the `scheduledImport` configuration shown in this section.
3. Customize the code for your needs. Descriptions of each parameter follow:

Parameter	Description	Example
name	A unique title for the import  <i>Note: If you create scheduled imports from more than one Store, each Store should have its own name and section in the configuration file.</i>	My_Store_Import

Parameter	Description	Example
enabled	A flag that decides if the import occurs; configure this to true to start an import or false to stop an import without deleting its configuration.	true or false
url		https://localhost:8443/marketplace/public/exportAll
keyStore.file	A file path pointing to a Java Keystore file which contains the public/private keypair that the server will use to identify itself to the remote server. May be set to null to uses the value of system property <code>javax.net.ssl.keyStore</code>	null
keyStore.password	The password used to unlock the private key within the file (not used to unlock the file itself). May be set to null to uses the value of system property <code>javax.net.ssl.keyStorePassword</code>	null
trustStore.file	A path pointing to a Java Keystore file which contains a trusted certificate that is part of the remote server's trust chain May be set to null to uses the value of system property <code>javax.net.ssl.trustStore</code>  Note: The trustStore should: <ul style="list-style-type: none"> contain the certificate of the remote server itself or the certificate of the Certificate Authority that signed the remote server's certificate or <ul style="list-style-type: none"> be part of the remote server's trust chain.  You cannot enter a password for the trustStore. The trusted certificate in the	null

Parameter	Description	Example
	trust store must NOT be password protected.	
<code>partial</code>	<p>A parameter that determines if the Store will add everything from a remote Store or if it will only add listings that were added or changed after a specific date.</p> <p>To create a partial import, Marketplace adds a timestamp from the last successful import to the URL. The remote Store only returns listings that were created or changed after the timestamp.</p> <p>Configure this to true to use partial imports or false to stop them without deleting the configuration.</p> <p>Note: Partial imports can improve performance by reducing the amount of redundant information transferred over multiple imports.</p>	<code>true or false</code>
<code>frequency.count</code>	Interval (in frequency.units) between scheduled imports.	<code>1, 2, 5, etc.</code>
<code>frequency.unit</code>	Unit of measurement for how often a scheduled import occurs.	<code>minutes, hours, or days</code>

4. *(Optional)* To schedule imports from more than one Store, copy another example of the `scheduledImport` configuration shown in this section. Customize it per instructions in the previous step.
5. Restart the server. The import will begin after one full cycle of the frequency duration completes. A log will appear in the Data Exchange section of the user interface.

4.4.5 Synchronization with OWF

The synchronization feature allows the Store to automatically send app components and their subsequent updates to OWF.

Synchronization is also necessary to push OWF applications to the Store. To use this feature, developers must do the following:

- Configure `/tomcat/lib/config/ozone-framework_*.yaml` to accept synchronization messages from the Store.
- Configure `/tomcat/lib/ozone/framework/security.xml` and `/tomcat/lib/ozone/marketplace/security.xml` (see below).
- Synchronize the OWF server with the Store through the Store's Administration Configuration pages.

To implement the synchronization feature with the OWF sample security plugin, configure the following OWF and Store files (found in the `/tomcat/lib/` directory):

In OWF:

- `/tomcat/lib/ozone/framework/security.xml` must include the following in the top-level `<beans>` element:

```
<sec:http pattern="/marketplace/sync/**" security="none" />
```

In the Store:

- `/tomcat/lib/ozone/marketplace/security.xml` must include the following in the top-level `<beans>` element:

```
<sec:http pattern="/public/descriptor/**" security="none" />
```

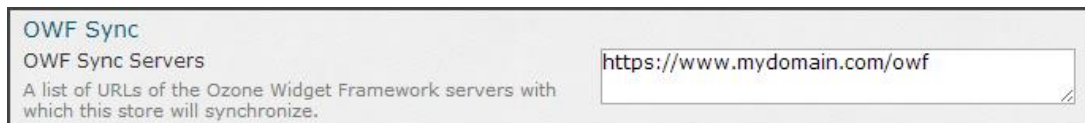
The Store will only synchronize with OWF servers linked to that Store. The Store will not synchronize with unspecified OWF systems or offline OWF systems. Synchronize OWF servers with the Store through the Store Administration Configuration pages. For more information, see the OZONE Store Administrator's Guide.

To use this feature:

1. From the drop-down User Menu, click Configuration Pages.
2. Links to reference data will appear, click Application Configuration.
3. In the left-panel, click Additional Configurations. To synchronize with an OWF instance, add its URL in the OWF Sync Servers section:

To connect to multiple OWF servers, use commas to separate their URLs.

Figure 3 OWF Sync Field



OWF Sync
OWF Sync Servers
A list of URLs of the Ozone Widget Framework servers with which this store will synchronize.

4. Listings that meet the OWF criteria described in the Configuring Listings to Appear in OWF section of the Store Administrator's Guide will appear in the Store in OWF (if listing approval is enabled in OWF) or directly in the administrator's App Component Manager in OWF if auto-approved is enabled.

5 Security Configuration

5.1 Security Setup

These example security options act only as samples and should **NOT** be used in a production environment.

The Store provides a modular security approach (security plugin) based on Spring Security. All provided security options supply an .xml file that can be customized to perform authentication and authorization.

The following files (which house common configurations information and bean definitions that are used by multiple plugins) can be used as the basis for updating a production environment.

Security files contained in `ozone-marketplace\tomcat\lib\ozone\marketplace\`:

- `application.xml`
The main entry point for external bean configuration.
- `security.xml`
The default sample security configuration, loaded by `application.xml`.
Uses Spring's session-based form login feature.
- `security_basic-ldap.xml` Sample configuration using HTTP Basic authentication and LDAP.
- `security_basic-spring-login.xml` Sample configuration using HTTP Basic authentication and Spring login.
- `security_cert-ldap.xml` Sample configuration using an X509 certificate and LDAP.
- `security_cert-only.xml` Sample configuration using an X509 certificate only.
- `session-control.xml`
Configuration for the (cached) session control features.
- `users.properties`
An example file-based registry of users, roles, and passwords.
Should **NOT** be used in production -- development and testing purposes **ONLY**.
- `security\authentication-providers.xml`
Common configuration file. Defines the `cachingPreauthProvider` and `cachingUsernamePasswordAuthProvider` beans.
- `security\ldap.xml`
Common configuration file. Defines the beans and settings related to LDAP authentication.
- `security\method-security.xml`
Common configuration file. Enables Spring's method-based security annotations.
- `security\user-cache.xml` Common configuration file. Enables the EhCache-based user details cache for Spring Security.
- `security\user-service.xml` Common configuration file. Enables a Spring Security authentication details service that uses the `users.properties` file.

In most of the pluggable security samples, user authentication is configured through an external text file like `users.properties` or LDAP.

5.2 Installing the Security Configuration

Changing the authentication or authorization method is accomplished in just a few easy steps:

- Stop the Tomcat Web Server.
- Update the `resource` property of the `<import>` tag in `\tomcat\lib\ozone\marketplace\application.xml` to the desired security .XML configuration file.
- Restart the application server by running the `\tomcat\start.bat` script.

Some of the pluggable security modules provided use an externalized properties file for the user store, located at `tomcat\lib\ozone\marketplace\users.properties`.

5.2.1 User Certificate Prompt

For the security examples that use certificate-based authentication, the bundled Tomcat instance should be configured to prompt for the user's certificate. To enable the certificate prompt, edit the `\tomcat\conf\server.xml` file and add or change the `clientAuth` property to `"true"`.

5.2.1.1 Example

```
<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" />
```

5.3 CAS Settings

OzoneConfig.properties proxies some of the CAS properties that can be customized by specific organizations. To change where the CAS server, CAS login and CAS logout point, update the corresponding values in OzoneConfig.properties. For additional CAS instructions, see CAS documentation.

5.4 Certificates (PKI) Configuration

The default certificates contained in the Store bundle only function for localhost communications. The Store will not function correctly if accessed from a remote machine while using the bundle-included certificates. See section 3.5.2.2.3: Custom Security Logout for more details.

To access a Store from any location other than localhost, a new server certificate needs to be generated. Otherwise, attempts to login from remote locations will fail.

5.4.1 Server Certificate Creation and Installation

Valid server certificates are needed for configuring the server to allow https authentication. See the steps below for generating and installing a self-signed server certificate.

The default certificates contained in the Store bundle only function for localhost communications. The Store will not function correctly if accessed from a remote machine while using the bundle-included certificates.

To access a Store from any location other than localhost, a new server certificate needs to be generated. Otherwise, attempts to login from remote locations will fail.

5.4.1.1 Generating a New Self-Signed Server Certificate

The Java keytool utility can be used to generate the public/private key and signed certificate. Open a command prompt, navigate to the /apache-tomcat/certs directory and execute the following command:

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

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`

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`

When using an IP address as the Common Name (CN), add an entry 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.

5.4.1.2 Configuring for Different Truststore/Keystore

For server-to-server calls (Store-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
```

3. Modify the JVM Parameters to start the web-application server in order to use the truststore referenced in Step 2. If a Tomcat server is being used, the parameters can be found in setenv.sh or setenv.bat (located in \apache-tomcat\bin), inside of the unpacked marketplace-bundle-7.17.2.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.

Parameter	Note / Value
- Djavax.net .ssl.trustStore=certs/mytruststore.jks	replace 'certs/keystore.jks' with the path and filename to the truststore
- Djavax.net .ssl.trustStorePassword=changeit	replace 'changeit' with the truststore's password (if applicable).

4. Finally, modify the Web server configuration to use the new keystore in SSL. Shown below is the relevant section from the Tomcat configuration script found in /tomcat/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="want"
  sslProtocol="TLS" />
```

5.4.2 Installing User PKI Certificates

Several of the example security configurations included in the Store bundle are configured to use client certificates. In order for a user to identify themselves using certificates, they must install a PKI certificate into their Web browser. The client certificate, included with the Store Bundle, is recognized immediately and can be used with the example security configuration. The certificate is located in the \tomcat\certs directory of the Store Bundle.

The included certificates should **ONLY** be used for testing. They must **NOT** be used in a production environment.

The default client certificate is used by importing the included testAdmin1.p12 or testUser1.p12 certificate into the user's browser.

In Internet Explorer, client certificates are added by selecting:

Tools → Internet Options → Content → Certificates → Personal → Import

In Firefox, this menu is accessed by clicking:

Tools → Options → Advanced → Encryption → View Certificates → Your Certificates → Import

Depending on the browser, importing certificates may cause warning messages to appear before accessing the Store. Web browsers will allow exceptions to be added to permit usage of these certificates the first time they are accessed.

6 Database Configuration

6.1 Overview

Database customization is one of the most common configuration file modifications to the Store. To use a custom database (such as Oracle or MySQL), changes are required to the `dataSource` section of the application configuration `.yaml` file.

While the full extent of administering a database is outside the scope of this guide, the following sections provide information on how to configure the Store to work with various databases. See the sections on individual files to determine the configuration changes that can be made.

The Marketplace bundle contains example application configuration `.yaml` files in the `/tomcat/lib/config/` folder. The `dataSource` section of each example configuration is pre-configured for a specific database, as listed below:

- `ozone-marketplace_h2.yaml` – H2 Database (embedded, in-memory or file-based) sample configuration
- `ozone-marketplace_mysql.yaml` – MySQL sample configuration
- `ozone-marketplace_oracle.yaml` – Oracle sample configuration
- `ozone-marketplace_postgresql.yaml` – PostgreSQL sample configuration
- `ozone-marketplace_sqlserver.yaml` – Microsoft SQL Server sample configuration

Using current database drivers is recommended. For further information, please reference external database documentation.

6.2 Supported Databases

The Store has been tested with the following databases and versions:

Database	Version(s)
PostgreSQL	10.1
Microsoft SQL Server	2012 SP1
MySQL	5.7.20
Oracle	12c

6.3 General Database Settings

Database customization is one of the most common configuration file modifications to the Store. To use a custom database (such as Oracle or MySQL), changes are required to the `dataSource` section of the application configuration `.yaml` file.

While the full extent of administering a database is outside the scope of this guide, the following sections provide information on how to configure the Store to work with various databases. See the sections on individual files to determine the configuration changes that can be made.

6.3.1 Application Configuration File

Each of the configuration files contains four separate `dataSource` blocks: one top-level `dataSource` block, and three contained within a top-level `environments` block.

The top-level `dataSource` block can be used for general settings to be shared by the environment-specific blocks.

The environment-specific blocks (`development`, `test`, and `production`) are applied based on the current environment setting, which is set at launch using the system environment variable `grails.env`.

The environment-specific settings take precedence over any settings in the top-level `dataSource` block, and will over-write any settings from the top-level `dataSource` block.

This guide focuses primarily on configuration of the production environment `dataSource` settings.

The development and test environment `dataSource` settings are not used when running in the production environment, and configuring them is outside the scope of this guide.

6.3.1.1 Environment Variables

The configuration `.yml` file supports resolving environment variables.

While the configuration `.yml` file may be configured with values directly, in some situations it may be preferable (for security or deployment flexibility) to defer setting certain values.

For example, the `dataSource` block may be configured to resolve the database settings from system environment variables, as in the following example:

```
dataSource:
  driverClassName: oracle.jdbc.driver.OracleDriver
  dbCreate: none
  url: ${APP_DB_URL}
  username: ${APP_DB_USER}
  password: ${APP_DB_PASSWORD}
```

Configured this way, the database URL, username, and password may be set as environment variables prior to launching the Store, without requiring additional changes to the `.yml` file:

```
marketplace/tomcat/> export
APP_DB_URL="jdbc:oracle:thin:@localhost:1521/ORCLCDB"
marketplace/tomcat/> export APP_DB_USER=omp
marketplace/tomcat/> export APP_DB_PASSWORD=mypassword
marketplace/tomcat/> ./start.sh init db --oracle
```

6.3.2 Example Database Settings and Properties

Listed below are the variable database elements that need to be modified as new databases are implemented. A detailed explanation of each field follows.

```
dataSource:
  driverClassName: org.h2.Driver
```

```

    dbCreate: create-drop
    url:
jdbc:h2:./ompProdDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
    username: sa
    password: ""
    pooled: true
    jmxExport: true
    properties:
        jmxEnabled: true
        initialSize: 5
        maxActive: 50
        minIdle: 5
        maxIdle: 25
        maxWait: 10000
        maxAge: 600000
        timeBetweenEvictionRunsMillis: 5000
        minEvictableIdleTimeMillis: 60000
        validationQuery: SELECT 1
        validationQueryTimeout: 3
        validationInterval: 15000
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: false
    jdbcInterceptors: ConnectionState
    defaultTransactionIsolation: 2 # TRANSACTION_READ_COMMITTED

```

Property	Purpose	Example
dbcreate	How database is created/updated.	none
username	Username for database connections. <i>Note: This field is database specific.</i>	omp-user
password	Password for database connections. <i>Note: This field is database specific.</i>	password
driverClassName	The JDBC driver. <i>Note: This field is database specific.</i>	org.postgresql.Driver
url	The connection string. <i>Note: This field is database specific.</i>	jdbc:postgresql://localhost:5432/omp
pooled	Enable database connection pooling.	true
minEvictableIdleTimeMillis	Minimum amount of time in milliseconds an object can be idle in pool before eligible for eviction	180000
timeBetweenEvictionRunsMillis	Time in milliseconds to sleep between runs of the idle object evictor thread	180000
testOnBorrow	Validate objects before they are borrowed from the pool	true
testWhileIdle	Validate objects in the idle object evictor thread	false
testOnReturn	Validate objects before they are returned to the pool.	true

Property	Purpose	Example
validationQuery	Validation query to run on target database. <i>Note: This field is database specific.</i>	SELECT 1
validationInterval	Frequency of validations on connections.	3000
maxActive	Maximum number of active connections that can be allocated from this pool at the same time.	100
initialSize	Initial number of connections that are created when the pool is started.	10
maxIdle	Maximum number of connections that should be kept in the pool at all times.	50
minIdle	Minimum number of established connections that should be kept in the pool at all times.	10
removeAbandoned	Ends connections if they exceed the removeAbandonedTime. If set to true, a connection is considered abandoned and eligible for removal if it has been in use longer than the removeAbandonedTimeout. <i>Note: Setting this to true can recover database connections from applications that fail to close a connection.</i>	false
jdbcInterceptors	A semicolon separated list of classnames extending the org.apache.tomcat.jdbc.pool.JdbcInterceptor class.	null

For additional details regarding the data source configuration, see the documentation for your application container or connection pool implementation, such as:

- https://tomcat.apache.org/tomcat-8.5-doc/jdbc-pool.html#Common_Attributes
- or
- <http://commons.apache.org/proper/commons-dbcp/configuration.html>

6.4 Oracle Setup

1. Create an Oracle Database user for the Store. It is recommended that there be a dedicated user for the Store, in order to avoid database object name collisions.
2. The bundle does not provide a JDBC driver for Oracle. Download the appropriate JDBC driver and place it into the web server's classpath.
When using Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
3. Open `/tomcat/lib/config/ozone-marketplace_oracle.yml` and navigate to the `dataSource` section. Make any required changes, such as username, password, and URL.
Refer to the General Database Settings section or the Oracle documentation for more details about specific settings.

```
environments:
  production:
    dataSource:
      dbCreate: none # for production environments, ensure
      "dbCreate" is set to "none"
      url: jdbc:oracle:thin:@localhost:1521/ORCLCDB
      driverClassName: oracle.jdbc.driver.OracleDriver
      username: omp
      password: mypassword
      pooled: true
      jmxExport: true
      properties:
        jmxEnabled: true
        initialSize: 5
        maxActive: 50
        minIdle: 5
        maxIdle: 25
        maxWait: 10000
        maxAge: 600000
        timeBetweenEvictionRunsMillis: 5000
        minEvictableIdleTimeMillis: 60000
        validationQuery: SELECT 1 FROM DUAL
        validationQueryTimeout: 3
        validationInterval: 15000
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: false
        jdbcInterceptors: ConnectionState
        defaultTransactionIsolation: 2 #
      TRANSACTION_READ_COMMITTED
```

- 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.
4. **IMPORTANT:** Verify that the `dbCreate` setting in `/tomcat/lib/config/ozone-marketplace_oracle.yml` is set to `none`.
 5. Save the changes to `/tomcat/lib/config/ozone-marketplace_oracle.yml`.
 6. Run the database scripts to initialize or upgrade the database.
 - If you are upgrading an existing Store from v7.17.0:
 - Run `/dbscripts/oracle/omp-7.17.0-oracle-reset.sql` to remove the old Liquibase changelog tables.

- Run `/dbscripts/oracle/omp-7.17.1.0-oracle-upgrade.sql` to apply the required migrations for v7.17.1.0.
 - If you are initializing a new Store database:
 - Run `/dbscripts/oracle/omp-7.17.0-oracle-create.sql` to create the initial v7.17.0 database.
 - Run `/dbscripts/oracle/omp-7.17.1.0-oracle-upgrade.sql` to apply the required migrations for v7.17.1.0.
7. Start the Store using the start scripts included in `/tomcat/`

```
marketplace/tomcat/> start.bat /init /db oracle
```

or

```
marketplace/tomcat/> ./start.sh init db --oracle
```

The Store team is aware of a known issue with the Oracle Web-based admin Console returning truncated characters when dealing with large data sets. Accordingly, using SQLPlus or SQLDeveloper to run the script mentioned herein is recommended.

6.5 MySQL Setup

1. Create a schema within MySQL for use with the Store. It is recommended that there be a dedicated schema for the Store to avoid database object name collisions.
2. Create a MySQL user with full access to the Store schema created above.
3. The bundle does not provide a JDBC driver for MySQL. Download the appropriate JDBC driver and place it into the Web server's classpath.
In Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
4. Open `/tomcat/lib/config/ozone-marketplace_mysql.yml` and navigate to the `dataSource` section. Make any required changes, such as username, password, and URL.
Refer to the General Database Settings section or the MySQL documentation for more details about specific settings.

```
environments:
  production:
    dataSource:
      dbCreate: none # for production environments, ensure
      "dbCreate" is set to "none"
      url:
      jdbc:mysql://localhost:3306/omp?verifyServerCertificate=false&useSSL=false
      driverClassName: com.mysql.jdbc.Driver
      username: omp
      password: mypassword
      pooled: true
      jmxExport: true
      properties:
```

```
jmxEnabled: true
initialSize: 5
maxActive: 50
minIdle: 5
maxIdle: 25
maxWait: 10000
maxAge: 600000
timeBetweenEvictionRunsMillis: 5000
minEvictableIdleTimeMillis: 60000
validationQuery: SELECT 1
validationQueryTimeout: 3
validationInterval: 15000
testOnBorrow: true
testWhileIdle: true
testOnReturn: false
jdbcInterceptors: ConnectionState
defaultTransactionIsolation: 2 #
TRANSACTION_READ_COMMITTED
```

5. **IMPORTANT:** Verify that the dbCreate setting in /tomcat/lib/config/ozone-marketplace_mysql.yml is set to none.
6. Save the changes to /tomcat/lib/config/ozone-marketplace_mysql.yml.
7. Run the database scripts to initialize or upgrade the database.
 - If you are upgrading an existing Store from v7.17.0:
 - Run /dbscripts/mysql/omp-7.17.0-mysql-reset.sql to remove the old Liquibase changelog tables.
 - Run /dbscripts/mysql/omp-7.17.1.0-mysql-upgrade.sql to apply the required migrations for v7.17.1.0.
 - If you are initializing a new Store database:
 - Run /dbscripts/mysql/omp-7.17.0-mysql-create.sql to create the initial v7.17.0 database.
 - Run /dbscripts/mysql/omp-7.17.1.0-mysql-upgrade.sql to apply the required migrations for v7.17.1.0.
8. Start the Store using the start scripts included in /tomcat/

```
marketplace/tomcat/> start.bat /init /db mysql
```

or

```
marketplace/tomcat/> ./start.sh init db --mysql
```

6.5.1 Configuring MySQL JDBC to use SSL

By default MySQL communicates over an unencrypted connection. However, in most cases, it can be configured to use SSL. This is somewhat implementation specific.

This capability completely depends on how the implementation's MySQL binaries were compiled, packaged, etc.

The following procedure covers how to configure an SSL capable MySQL server to work with a Franchise Store bundle. The starting point for the server implementation used in this example is:

1. Operating System: CentOS 6.4, 64 bit minimal installation (no updates were applied)
2. MySQL Server v5.1.69 (achieved by installing the "mysql-server" package with yum)

This procedure was developed from the MySQL 5.1 documentation, specifically the following sections:

- [Using SSL for Secure Connections at http://dev.mysql.com/doc/refman/5.1/en/ssl-connections.html](http://dev.mysql.com/doc/refman/5.1/en/ssl-connections.html)
- [Connector/J \(JDBC\) Reference at http://dev.mysql.com/doc/refman/5.1/en/connector-j-reference.html](http://dev.mysql.com/doc/refman/5.1/en/connector-j-reference.html)

This procedure relies on self-signed certificates. It is for testing and demonstration purposes only. The following three sections explain the configuration steps.

6.5.1.1 Creating MySQL Server's CA and Server Certificates

The following steps guide a developer through creating the CA and Server certificates for a MySQL server:

1. Create the CA Key and Certificate

From a shell prompt on a MySQL server, type the following commands:

```
> openssl genrsa 2048 > ca-key.pem
> openssl req -new -x509 -nodes -days 365 -key ca-key.pem -out
cacert.pem
```

After the second command, the system prompts the developer to provide basic identity information. For the purpose of this demonstration, it is not important what information the developer provides here. However, it will be necessary to provide the same information in the next step.

2. Create the Server Certificate

From a shell prompt on your MySQL server, type the following commands. After the first command, the developer will again be prompted to provide identity information. The developer must provide the same information that they provided in Step 1. However, when prompted for the Common Name, provide the MySQL server's hostname (e.g. mysql.mydomain.com).

```
> openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-
key.pem -out server-req.pem
> openssl rsa -in server-key.pem -out server-key.pem
> openssl x509 -req -in server-req.pem -days 365 -CA cacert.pem -
CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

3. Consolidate the Output from Steps 1 & 2

Copy the following files (produced in the preceding two steps) to a location where at a minimum the MySQL user has read access.

In this example, the location is `/etc/ssl/certs/`

- o `cacert.pem`
- o `server-key.pem`

```
o server-cert.pem
```

6.5.1.2 Configure MySQL

1. Edit my.cnf

Edit the MySQL configuration file. For this example, the file is located at /etc/my.cnf. Add the lines shown in bold to the [mysqld] section of the file.

```
ssl-ca=/etc/ssl/certs/cacert.pem
ssl-cert=/etc/ssl/certs/server-cert.pem
ssl-key=/etc/ssl/certs/server-key.pem
```

2. Restart MySQL

```
sudo service mysqld restart
```

3. Create the MySQL Franchise Store Schema

At a minimum, create the schema and assign permissions as you would normally, following the steps in Section: Adding Users and Assigning Roles. This will leave the choice of using an encrypted connection up to the connecting client (in this case the Franchise Store application).

To enforce the use of SSL from the database server side, add REQUIRE SSL to the end of the grant statement where user permissions to the Franchise Store schema are assigned. For example:

```
GRANT ALL ON franchise.* TO 'franchise'@'storehost.mydomain.com'
IDENTIFIED BY 'franchise' REQUIRE SSL;
```

6.5.1.3 Configure the Franchise Store Bundle

1. Modify /tomcat/lib/config/ozone-marketplace_mysql.yml

Add useSSL=true to the dataSource URL configuration. This can also be enforced from the client side with the requireSSL option. For example:

```
url:
  "jdbc:mysql://192.168.56.11/franchise?useSSL=true&requireSSL=true&
  rustCertificateKeyStoreUrl=file://${javax.net.ssl.trustStore}&trust
  CertificateKeyStorePassword=changeit"
```

2. Modify the Application TrustStore

Add the CA certificate, created above (cacert.pem) to the application's trust store. In the case of the franchise bundle, the trust store is a file called keystore.jks found in \$BUNDLE_PATH/apache-tomcat/certs. Do this with the following command (assuming you have the JDK installed and JAVA_HOME/bin on your PATH, if they aren't there, add them first).

```
> keytool -import -alias mysqlCAcert -file cacert.pem -keystore
keystore.jks
```

3. Start the Application

```
> ./start.sh
```

6.6 PostgreSQL Setup

1. Create either a new login role or a new schema in order to avoid database object name collisions between the Store and other database applications.
2. Edit the user so that it can create database objects.
3. Create a new database. Use UTF8 as the encoding (default).
4. The bundle does not provide a JDBC driver for PostgreSQL. Download the appropriate JDBC driver and place it into the web server's classpath.
In Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
5. Open `/tomcat/lib/config/ozone-marketplace_postgresql.yml` and navigate to the `dataSource` section. Make any required changes, such as username, password, and URL.

Refer to the General Database Settings section or the PostgreSQL documentation for more details about specific settings.

```
environments:
  production:
    dataSource:
      dbCreate: none # for production environments, ensure
      "dbCreate" is set to "none"
      url: jdbc:postgresql://localhost:5432/omp
      driverClassName: org.postgresql.Driver
      username: omp
      password: mypassword
      properties:
        jmxEnabled: true
        initialSize: 5
        maxActive: 50
        minIdle: 5
        maxIdle: 25
        maxWait: 10000
        maxAge: 600000
        timeBetweenEvictionRunsMillis: 5000
        minEvictableIdleTimeMillis: 60000
        validationQuery: SELECT 1
        validationQueryTimeout: 3
        validationInterval: 15000
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: false
        jdbcInterceptors: ConnectionState
        defaultTransactionIsolation: 2 #
        TRANSACTION_READ_COMMITTED
```

6. **IMPORTANT:** Verify that the `dbCreate` setting in `/tomcat/lib/config/ozone-marketplace_postgres.yml` is set to `none`.
7. Save the changes to `/tomcat/lib/config/ozone-marketplace_postgres.yml`.
8. Run the database scripts to initialize or upgrade the database.
 - If you are upgrading an existing Store from v7.17.0:
 - Run `/dbscripts/postgresql/omp-7.17.0-postgresql-reset.sql` to remove the old Liquibase changelog tables.
 - Run `/dbscripts/postgresql/omp-7.17.1.0-postgresql-upgrade.sql` to apply the required migrations for v7.17.1.0.
 - If you are initializing a new Store database:

- Run `/dbscripts/postgresql/omp-7.17.0-postgresql-create.sql` to create the initial v7.17.0 database.
- Run `/dbscripts/postgresql/omp-7.17.1.0-postgresql-upgrade.sql` to apply the required migrations for v7.17.1.0.

9. Start the Store using the start scripts included in `/tomcat/`

```
marketplace/tomcat/> start.bat /init /db pg
```

or

```
marketplace/tomcat/> ./start.sh init db --pg
```

6.7 MS SQL Server Setup

1. Create a new SQL Server database for use with the Store.
2. Create a SQL Server user with full access to the Store database created above.
3. Run the following SQL command from the master database to enable row level versioning for the Store database:

```
ALTER DATABASE <name of database>
SET READ_COMMITTED_SNAPSHOT ON;
```

4. The bundle does not provide a JDBC driver for SQL Server. Download the appropriate JDBC driver and place it on the Web server's classpath.
In Tomcat, the driver can be placed in the `/tomcat/lib/` directory.
5. Open `/tomcat/lib/config/ozone-marketplace_sqlserver.yml` and navigate to the `dataSource` section. Make any required changes, such as username, password, and URL.
Refer to the General Database Settings section or the MS SQL Server documentation for more details about specific settings.

```
environments:
  production:
    dataSource:
      dbCreate: none # for production environments, ensure
      "dbCreate" is set to "none"
      url: jdbc:jtds:sqlserver://localhost:1433/omp
      driverClassName: net.sourceforge.jtds.jdbc.Driver
      username: omp
      password: mypassword
      properties:
        jmxEnabled: true
        initialSize: 5
        maxActive: 50
        minIdle: 5
        maxIdle: 25
        maxWait: 10000
        maxAge: 600000
        timeBetweenEvictionRunsMillis: 5000
        minEvictableIdleTimeMillis: 60000
        validationQuery: SELECT 1
```

```
validationQueryTimeout: 3
validationInterval: 15000
testOnBorrow: true
testWhileIdle: true
testOnReturn: false
jdbcInterceptors: ConnectionState
defaultTransactionIsolation: 2 #
TRANSACTION_READ_COMMITTED
```

6. **IMPORTANT:** Verify that the dbCreate setting in /tomcat/lib/config/ozone-marketplace_sqlserver.yml is set to none.
7. Save the changes to /tomcat/lib/config/ozone-marketplace_sqlserver.yml.
8. Run the database scripts to initialize or upgrade the database.
 - If you are upgrading an existing Store from v7.17.0:
 - Run /dbscripts/mssql/omp-7.17.0-mssql-reset.sql to remove the old Liquibase changelog tables.
 - Run /dbscripts/mssql/omp-7.17.1.0-mssql-upgrade.sql to apply the required migrations for v7.17.1.0.
 - If you are initializing a new Store database:
 - Run /dbscripts/mssql/omp-7.17.0-mssql-create.sql to create the initial v7.17.0 database.
 - Run /dbscripts/mssql/omp-7.17.1.0-mssql-upgrade.sql to apply the required migrations for v7.17.1.0.
9. Start the Store using the start scripts included in /tomcat/

```
marketplace/tomcat/> start.bat /init /db mssql
```

or

```
marketplace/tomcat/> ./start.sh init db --sqlserver
```