BEA AquaLogic Commerce Services

Deployment Guide

Version 5.1.1
March 2007

Contents

Overview

The *Getting Started Guide* describes how to quickly setup a sample deployment of AquaLogic Commerce Services with a preconfigured demo store. This deployment guide will expand on that and explain how to setup your actual store and configure it to meet your needs.

Path Placeholders

- <BEA_HOME> refers to your root path for BEA applications (for example, C:\bea)
- <WL_HOME> refers to the path where BEA WebLogic Server 9.2 is installed (for example, C:\bea\weblogic92)
- <ALCS_APP_HOME> refers to the root path for AquaLogic Commerce Services applications (for example, C:\bea\user_projects\applications\alcs\commerceApp)

Setting up AquaLogic Commerce Services

This section explains how to install and setup AquaLogic Commerce Services on BEA WebLogic Server 9.2 on Windows XP.

Caution

- If you are upgrading AquaLogic Commerce Services to a newer version, follow the instructions in the *Upgrading AquaLogic Commerce Services* section of the *Getting Started Guide* to avoid losing all product configurations.
- The date and time of all application servers should be synchronized with the date and time of your database server to prevent confusing creation/last-modified timestamps from being created in the database. This issue arises from the fact that some code uses the database server date/time and other code uses the application server date/time.

Install AquaLogic Commerce Services

Run the AquaLogic Commerce Services automated installer to install AquaLogic Commerce Services in the existing BEA WebLogic Server location (see the Getting Started Guide for details on what the installer does).

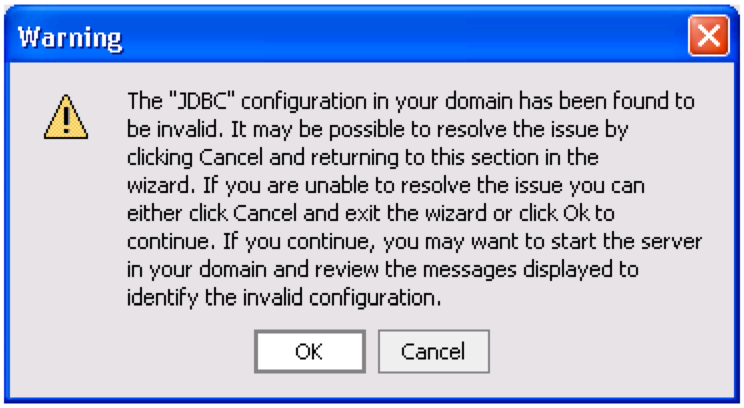
Create a New Domain

1. Click *Start* > *All Programs* > *BEA Products* > *Tools* > *Configuration Wizard*.

For Linux, you can run <WL_HOME>/common/bin/config.sh.

You must have an X server configured to use this GUI configuration tool.

2. Select *Create A new WebLogic domain* in the window that appears. Click *Next*.
3. Select *Base this domain on an existing template*.
4. Click the *Browse* button and select "<WL_HOME>\common\templates\applications\commerce_services.jar" as the template to use. Click *OK* to select this file.
5. Click *Next*.
6. Enter a *User name* and *User password*. For this example, we will use "weblogic" for both the *User name* and *User password*. Write down this information, since you will need it later. Click *Next*.
7. Select the BEA-supplied "Sun SDK 1.5.0" or point the application to the Java Development Kit of your choice. Click *Next*.
8. Select *Yes* if you would like to configure your admin server, managed servers and JDBC settings with the wizard, otherwise select *No*. If you select *No*, the default demo settings will be used and these can be configured manually at a later time. Click *Next*.
9. If you selected *Yes* in step 8, enter the correct configuration for your environment (ensuring that "jdbc/epjndi" is used for the JNDI name field) and continue through the wizard until you get to the *Create WebLogic Domain* page (see the JNDI and JDBC Configuration section for more details on JNDI/JDBC settings).
10. Enter the name and location for the domain and click *Create*. For this example, we will use "alcs" as the name. The locations will be "<BEA_HOME>\user_projects\domains" and "<BEA_HOME>\user_projects\applications".
11. If you selected *No* in step 8, then you will need to click *OK* when you get the following warning.



The "JDBC" configuration in your domain has been found to be invalid. It may be possible to resolve the issue by clicking Cancel and returning to this section in the wizard. If you are unable to resolve the issue you can either click Cancel and exit the wizard or click Ok to continue. If you continue, you may want to start the server in your domain and review the messages displayed to identify the invalid configuration.

This warning appears, because the demo database has no password.

12. The wizard will then create the new domain, creating an "alcs" directory under both the domains and applications directories.
13. When the wizard finishes, click *Done* to close the final window.

Set Up AquaLogic Commerce Services

1. To correctly set up a JDBC connection, the JDBC driver must be on WebLogic Server's CLASSPATH. To do this, copy the JDBC driver file for your database into the <BEA_HOME>\user_projects\domains\alcs\lib directory (see the JNDI and JDBC Configuration section for more information).

2. In WEB-INF\conf\spring\dataaccess\hibernate\hibernate-config.xml (for each of the commerceServices, commerceServicesConnect, and commerceServicesManager directories found in <ALCS_APP_HOME>), find the tag <prop key="hibernate.dialect"> and change the value to the database dialect type for the database that your application is currently connecting to. For example, if you're using an Oracle database, the block should look like the following.

```
<prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect
</prop>
```

   Below are the values for the different databases:

   - org.hibernate.dialect.MySQLDialect
   - org.hibernate.dialect.OracleDialect
   - org.hibernate.dialect.SQLServerDialect
   - org.hibernate.dialect.DerbyDialect

3. In WEB-INF\conf\spring\security\acegi.xml (for each of the commerceServices, commerceServicesConnect, and commerceServicesManager directories found in <ALCS_APP_HOME>), find the “portMapper” bean declaration and under the “portMappings” property change the entry key to 7011 (or whatever your HTTP port is) and the value to 7012 (or whatever your HTTPS port is). The whole block should now look similar to the following.

```
<bean id="portMapper" class="org.acegisecurity.util.PortMapperImpl">
 <property name="portMappings">
  <map>
   <entry key="7011"><value>7012</value></entry>
  </map>
 </property>
</bean>
```

4. You may get compatibility exceptions on *.QName when starting WebLogic Server with some releases of JDK 1.5. To correct this problem, you will need to edit <BEA_HOME>\user_projects\domains\alcs\bin\setDomainEnv.cmd and add the following line.

```
set JAVA_OPTIONS="%JAVA_OPTIONS%
  -Dcom.sun.xml.namespace.QName.useCompatibleSerialVersionUID=1.0"
```

5. If you get some JDK out of memory errors, then you may also need to change the maximum heap size to 512MB (“-Xmx512m”) or higher and the maximum perm space size to 128MB (“-XX:MaxPermSize=128m”) or higher for your platform. These settings can be changed in <BEA_HOME>\user_projects\domains\alcs\bin\setDomainEnv.cmd.

Open the WebLogic Administration Console

The following sections will require the use of the WebLogic Administration Console. Perform the following steps to open and log onto the Administration Console.

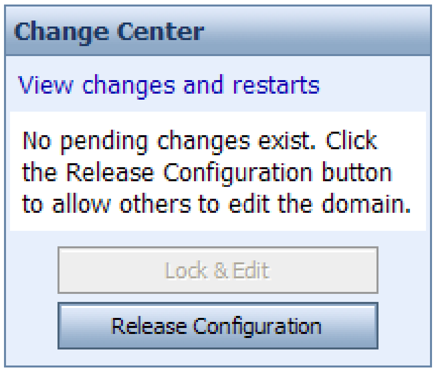
1. Start the WebLogic Server by double-clicking <BEA_HOME>\user_projects\domains\alcs\startWebLogic.cmd
2. Log on to the Administration Console
   a. Open a browser window.
   b. Browse to http://localhost:7011/console (where 7011 is the port that you setup for the admin server).
   c. Type the username and password entered during step 4 of the *Create a New Domain* section. For our example, it will be “weblogic” for both.

Set Up a JNDI Data Source

This section explains how to manually setup a JNDI/JDBC data source in the admin server.

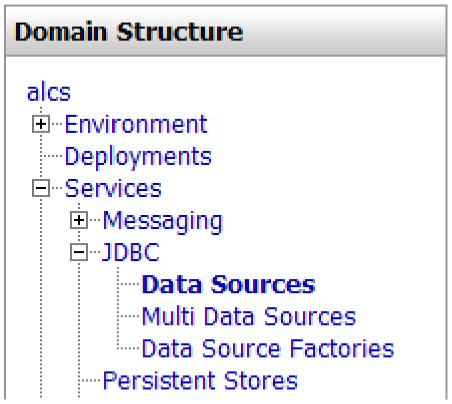
Perform the following steps on the Administration Console to configure your JDBC data source if you have not already set it up in with the configuration wizard.

1. On the left hand side, click the 'Lock and Edit' button under the Change Center. This will allow you to make changes to your domain settings.

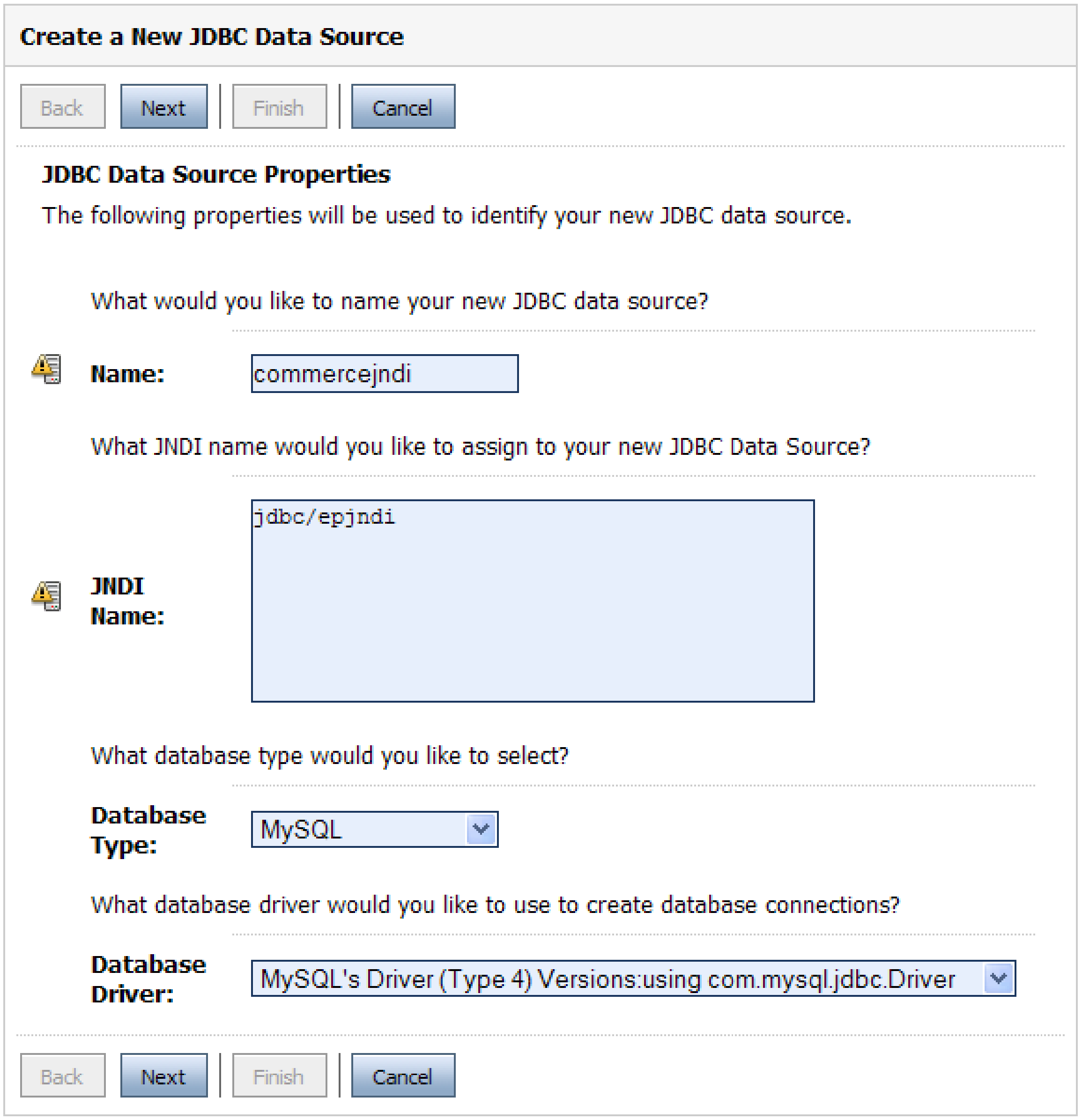


No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

2. Under the alcs domain of the *Domain Structure* Tree, select *Services* > *JDBC* > *Data Sources*.



3. On the right hand pane, click the *New* button under Data Sources.
4. On the next page titled *JDBC Data Source Properties*, enter the following information:



JDBC Data Source Properties

The following properties will be used to identify your new JDBC data source.

What would you like to name your new JDBC data source?

Name: commercejndi

What JNDI name would you like to assign to your new JDBC Data Source?

JNDI Name: jdbc/epjndi

What database type would you like to select?

Database Type: MySQL

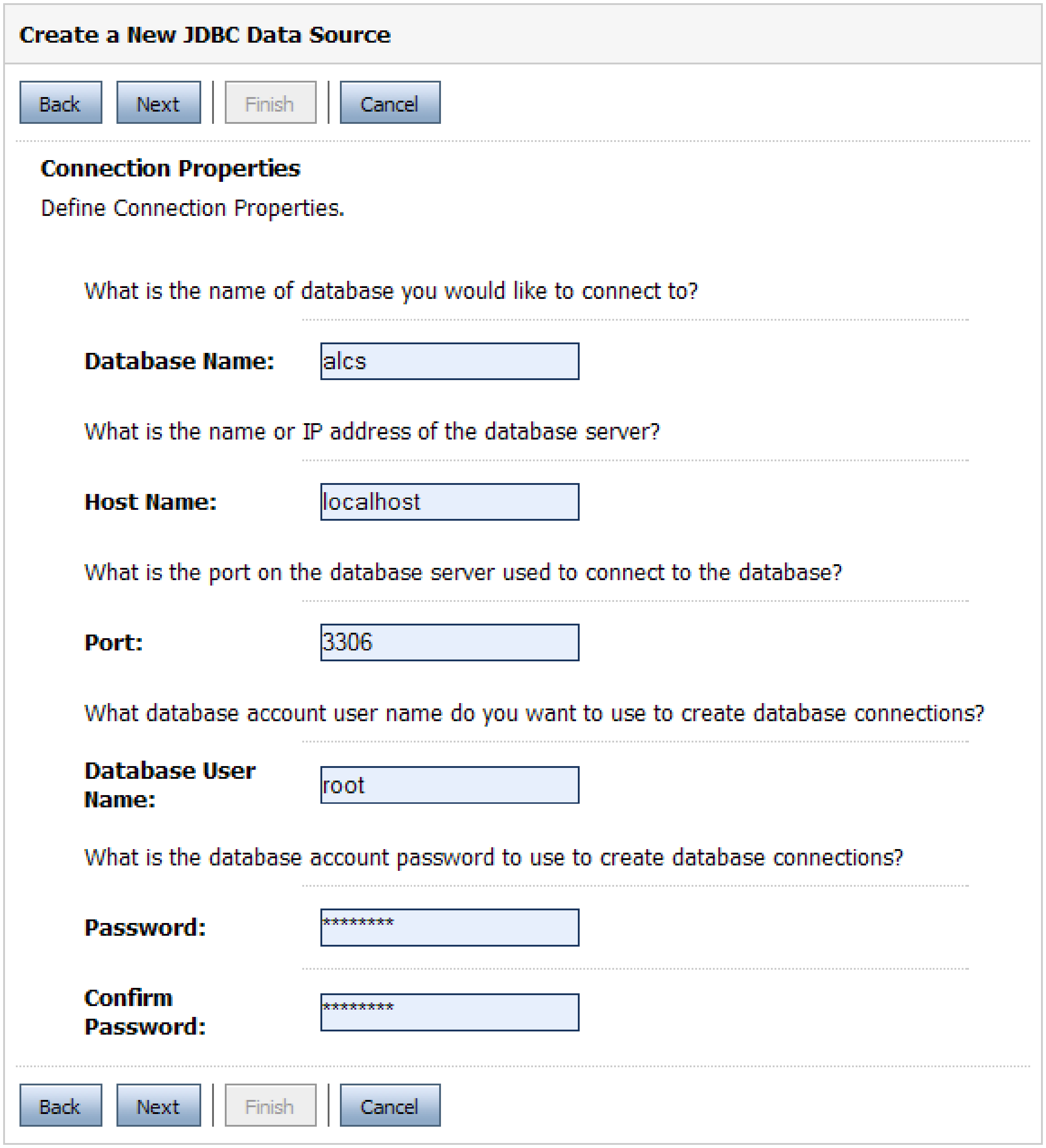
What database driver would you like to use to create database connections?

Database Driver: MySQL's Driver (Type 4) Versions:using com.mysql.jdbc.Driver

- *Name*: commercejndi (any descriptive name may be used here)
- *JNDI Name*: jdbc/epjndi
- *Database Type*: Select your database type
- *Database Driver*: Select the name of the database driver that was copied to the domain lib directory in the Set Up AquaLogic Commerce Services section (See the JNDI and JDBC Configuration section for the correct Driver name).

Click *Next*.

5. Click *Next* on the *Transaction Options* screen.
6. Enter your database properties on the *Connection Properties* screen as seen below and then click *Next*.


Create a New JDBC Data Source

Connection Properties

Define Connection Properties.

What is the name of database you would like to connect to?

Database Name: alcs

What is the name or IP address of the database server?

Host Name: localhost

What is the port on the database server used to connect to the database?

Port: 3306

What database account user name do you want to use to create database connections?

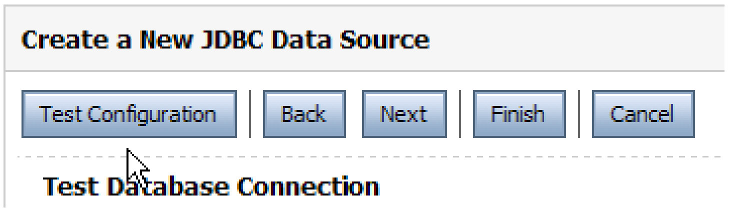
Database User Name: root

What is the database account password to use to create database connections?

Password: ********

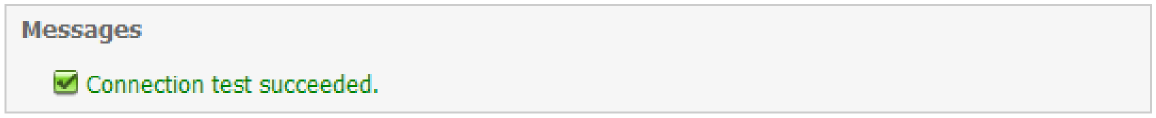
Confirm Password: ********

7. In the *Test Database Connection* screen, click on the *Test Configuration* button.


Create a New JDBC Data Source

Test Database Connection

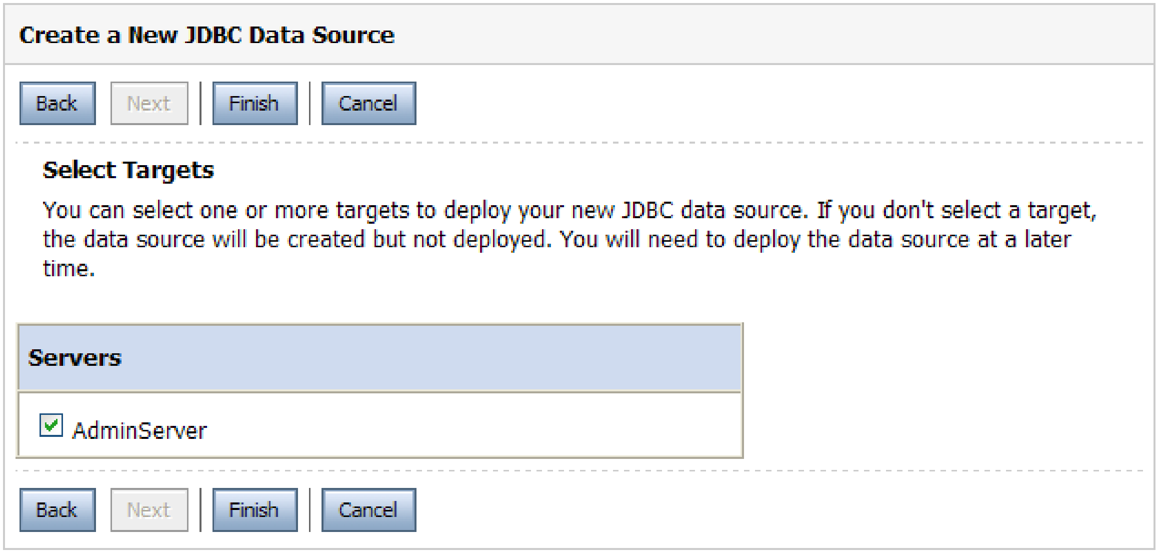
If your connection information is correct, you should see a success message. Otherwise, fix any errors reported.


Messages

✓ Connection test succeeded.

Click *Next*.

8. In the *Select Targets* screen, click on the checkbox next to the name of your admin server (the default demo setting is “CommerceServicesDemoServer”). Click *Next*.

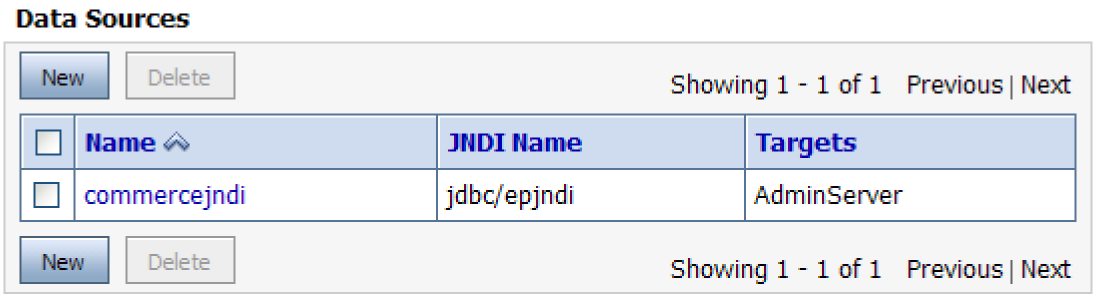


Select Targets

You can select one or more targets to deploy your new JDBC data source. If you don't select a target, the data source will be created but not deployed. You will need to deploy the data source at a later time.

Servers
✓ AdminServer

9. Click *Finish* to save the configuration. The new data source should be visible in the *Data Sources* window.



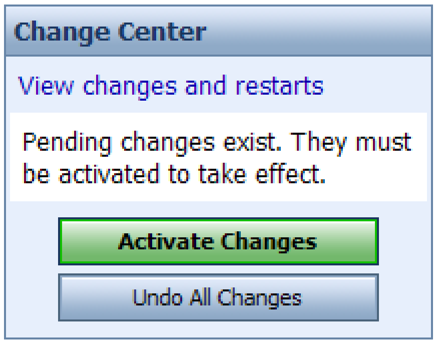
Data Sources

Showing 1 - 1 of 1 Previous | Next

☐	Name	JNDI Name	Targets
☐	commercejndi	jdbc/epjndi	AdminServer

Showing 1 - 1 of 1 Previous | Next

10. On the left hand side, click the *Activate Changes* button under the *Change Center* to commit these changes. You have now completed setting up your JDBC connection.



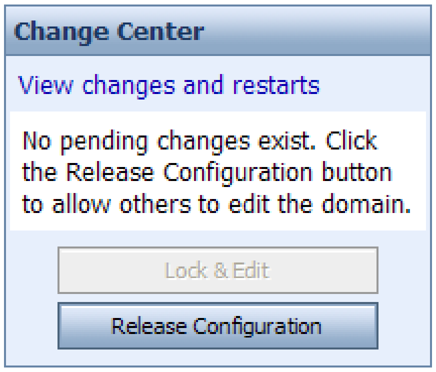
Pending changes exist. They must be activated to take effect.

You may need to restart your WebLogic Server before you can successfully commit changes when setting up a JDBC Connection

Configure SSL (Optional)

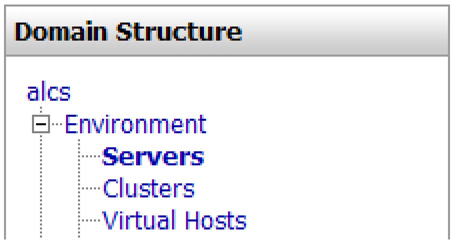
This section explains how to manually setup SSL for the admin server if you have not already set it up with the configuration wizard.

1. On the left hand side, click the *Lock and Edit* button under the *Change Center*.

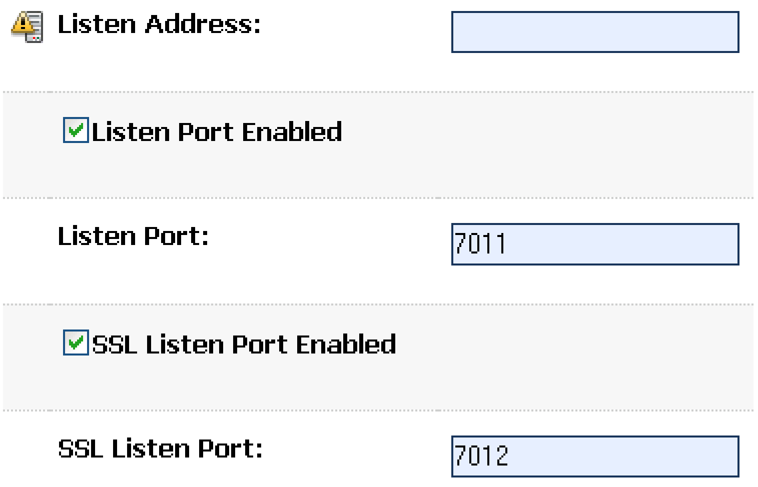


No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

2. Under your domain in the *Domain Structure* tree, expand *Environment* > *Servers*.



3. Click on your admin server name *AdminServer(admin)* in the *Servers* section (the default demo admin server name is “CommerceServicesDemoServer”).
4. In the *General Configurations* window check the *SSL Listen Port Enabled* box and ensure that the ports correspond to the ones setup in the acegi.xml file as per the Set Up AquaLogic Commerce Services section.


Listen Address:

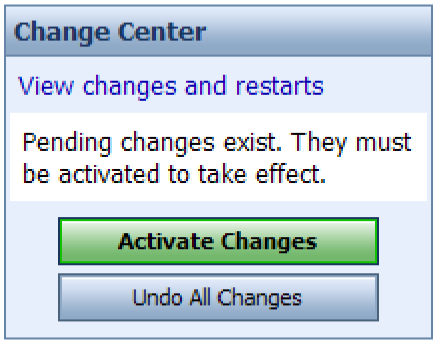
✓ Listen Port Enabled

Listen Port: 7011

✓ SSL Listen Port Enabled

SSL Listen Port: 7012

5. Click *Save*.
6. On the left hand side, click the *Activate Changes* button under the *Change Center* to commit these changes. You have finished configuring SSL.


Change Center

View changes and restarts

Pending changes exist. They must be activated to take effect.

Authorize.net payment processing

When trying to connect to the Authorize.net testing server with WebLogic, WebLogic gives an error similar to the following:

```
<Warning> <Security> <BEA-090504> <Certificate chain received from
test.authorize.net
- 64.94.118.75 failed hostname verification check. Certificate
contained \*.authorize.net
but check expected test.authorize.net>
```

The workaround for this is to turn off "hostname verification" in WebLogic. This is ONLY an issue with test transactions and does not occur when connecting to the Authorize.net production server.

WebLogic Authentication Plug-in

The WebLogic authentication plug-in allows the Storefront to authenticate against WebLogic Server, and also allows Single Sign-On with WebLogic Server and WebLogic Portal applications running within the same server instance.

AquaLogic Commerce Services comes pre-configured with the WebLogic authentication plug-in already setup, so you should never have to manually set it up. However, this section provides the steps for manually setting up the authentication plug-in for informational purposes. It should help you to better understand the integration.

Copy the plug-in jar file

The first step is to copy the WebLogic authentication plug-in jar file `com.elasticpath.plugins.wls_authenticator-5.1.jar` to the AquaLogic Commerce Services Storefront `WEB-INF\lib` folder. For example into `<BEA_HOME>\user_projects\applications\alcs\commerceApp\commerceServices\WEB-INF\lib`.

Copy the JAAS login configuration file

Copy the file `jaaslogin.config` into your domain root folder. For example, `<BEA_HOME>\user_projects\domains\alcs`

Set the location of the login configuration file

1. For the application to be able to find the JAAS login configuration file we need to add a startup option.
2. Navigate to the location of your domain. For our example, `<BEA_HOME>\user_projects\domains\alcs`
3. Open the “bin” directory. For our example, this will be `<BEA_HOME>\user_projects\domains\alcs\bin`.
4. Open the “setDomainEnv.cmd” file found in the bin directory in a text editor.
5. Add the following option to the end of the “set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES%” line.

```
Djava.security.auth.login.config==jaaslogin.config
```

> There does need to be two equal signs in the line above, since the second equal sign indicates that this login file should override any others.

6. Save the file and exit your text editor.

Change the user id generation mode

To allow customers to log into WebLogic Server after they change their email address, the user id mode needs to be changed with the following steps.

1. Open the WEB-INF\commerce-config.xml under your Storefront web application root folder and change the "userId.mode" value to "2".
   In this mode, the system will generate a unique user id for each customer. This user id will not be changed when customers log into the Storefront to
   change their email address.
2. Repeat last step for your Commerce Manager web application.

Change user maintenance mode

By default, the Commerce Manager does not support the maintenance of a password when using WebLogic authentication. To change this, the user maintenance mode needs to be changed as follows.

1. Open the WEB-INF\commerce-config.xml under your Commerce Manager web application root folder and change the "user.maintain.mode" to "2".
   In this mode the create/delete customer and edit password functions will be disabled.

Change the customer mapping

When using WebLogic authentication, the password should not be stored in a local database. You will need to change the customer Hibernate mapping file settings with the following steps.

1. Open WEB-INF\conf\spring\dataaccess\hibernate\mapping\Customer.hbm.xml under your Storefront web application root and remove or comment out following lines:

```
<many-to-one column="AUTHENTICATION_UID"
 unique="true"
 lazy="false"
 cascade="all"
 name="customerAuthentication"
 class="com.elasticpath.domain.customer.CustomerAuthentication"
 fetch="select"/>
```

2. Repeate last step for your Commerce Manager web application.

Single sign-on cookie

To ensure Single Sign-on works without problems with your WebLogic Server and WebLogic Portal applications, ensure that the `weblogic.xml` file for those applications contains the following settings:

```
<session-descriptor>
  <cookie-name>JSESSIONID</cookie-name>
  <cookie-path>/</cookie-path>
 </session-descriptor>
```

WebLogic Portal Unified User Profile Plug-in

WebLogic Portal supports the concept of a Unified User Profile (UUP) which allows a Portal user to be associated with profile data from external systems. The UUP plug-in provides an EAR file and user profile property set that allows you to manage AquaLogic Commerce Services user properties from the WebLogic Portal Administration Console or within WebLogic Portal code. The details on how to manage user properties from an external system with the WebLogic Portal is described in the WebLogic Portal User Management Guide at http://edocs.bea.com/wlp/docs92/users/index.html

This section describes how to extend your domain to add WebLogic Portal to it and how to install the WebLogic Portal Unified User Profile plug-in for AquaLogic Commerce Services.

Assumptions

This document assumes you already know how to set up a WebLogic Portal project. Refer to the Getting Started with WebLogic Portal documentation at http://edocs.bea.com/wlp/docs92/tutorials/index.html for a tutorial on setting up your Portal environment and creating a Portal.

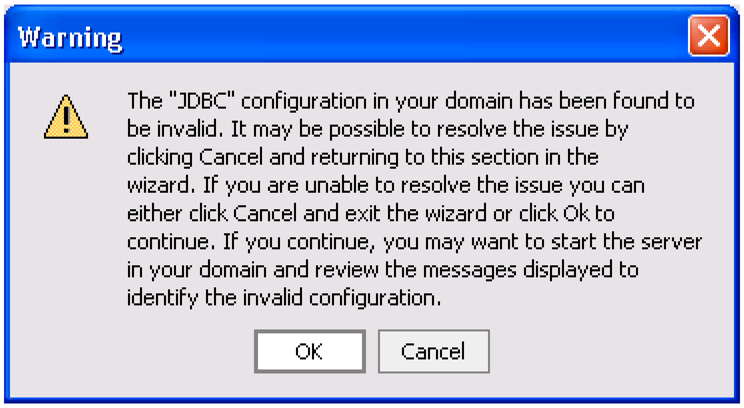
Extending Your Domain to Add WebLogic Portal

The following steps will add WebLogic Workshop and Portal to your AquaLogic Commerce Services domain.

1. Click Start > All Programs > BEA Products > Tools > Configuration Wizard.

   For Linux, you can run <WL_HOME>/common/bin/config.sh.
   You must have an X server configured to use the GUI configuration tool.

2. Select *Extend an existing WebLogic domain* in the window that appears. Click *Next*.
3. Select the domain created for your AquaLogic Commerce Services applications (in our example, this is <BEA_HOME>\user_projects\domains\alcs). Click *Next*.
4. Select *Extend my domain automatically to support the following added BEA products*.
5. Check *Workshop for WebLogic Platform* and *WebLogic Portal*. Click *Next*.
6. Select *No* to keep the default JDBC and JMS settings. Click *Next*.
7. Click *Extend*.
8. If you get the following warning, then you will need to click *OK*.



The "JDBC" configuration in your domain has been found to be invalid. It may be possible to resolve the issue by clicking Cancel and returning to this section in the wizard. If you are unable to resolve the issue you can either click Cancel and exit the wizard or click Ok to continue. If you continue, you may want to start the server in your domain and review the messages displayed to identify the invalid configuration.

This warning appears, because the demo database has no password.

9. Click *Done*.

Since extending a domain to add Portal does not update the setDomainEnv.cmd and startWebLogic.cmd files, you will need to make the following changes to those files all Portal functionality to work correctly.

1. Make the following modifications to setDomainEnv.cmd in the bin directory of your AquaLogic Commerce Services domain.

   a) Replace "set POINTBASE_FLAG=" with the following lines:

```
set debugFlag=true
set testConsoleFlag=true
set iterativeDevFlag=true
set logErrorsToConsoleFlag=true

set POINTBASE_FLAG=true
set POINTBASE_PORT=9093
set POINTBASE_DBNAME=weblogic_eval
```

   b) Replace "set EXTRA_JAVA_PROPERTIES=" with the following line (on one line):

```
set EXTRA_JAVA_PROPERTIES=
-Dweblogic.wsee.bind.suppressDeployErrorMessage=true
-Dweblogic.http.descriptor.merge=true
```

   c) Add the following lines after “set SERVER_CLASS=weblogic.Server”:

```
set WLP_HOME=%WL_HOME%\portal
set P13N_HOME=%WL_HOME%\p13n
set
WLP_POST_CLASSPATH=%WL_HOME%\server\lib\xquery.jar;%WL_HOME%\server\
lib\binxml.jar
```

2. Make the following modifications to startWebLogic.cmd in the bin directory of your AquaLogic Commerce Services domain.
   a) Add following line after “set ALREADY_STOPPED=true”:

```
call "%WLP_HOME%\thirdparty\autonomy-wlp92\autonomy.cmd" stop
```

   b) Add the following lines after “set JAVA_OPTIONS=%SAVE_JAVA_OPTIONS%”:

```
if "%WLP_SEARCH_OPTION%"=="" (
     @REM If not set externally, this portal domain will start the
full Autonomy engine stack.
     set WLP_SEARCH_OPTION=full
)
if "%WLP_SEARCH_OPTION%"=="full" (
     start "Start Portal SearchEngine" /B  cmd /c call
"%WLP_HOME%\thirdparty\autonomy-wlp92\autonomy.cmd" start
)
if "%WLP_SEARCH_OPTION%"=="minimal" (
     start "Start Portal SearchEngine" /B  cmd /c call
"%WLP_HOME%\thirdparty\autonomy-wlp92\autonomy.cmd" start
)
```

Installing the plug-in

Perform the following steps in a new or existing Workshop for a WebLogic Platform Portal Ear Project:

1. If a Datasync project has not already been created in association with the EAR, create one (right click, New > Datasync Project) and associate it with the EAR.
2. Copy `CommerceServices.usr` to the userprofiles folder of your Datasync project
3. Enter Workshop's Project Explorer view (Window > Show View > Project Explorer) and expand Enterprise Applications.
4. Expand the node relating to your Portal EAR
5. Expand the [WebLogic] node
6. Right-click WebLogic J2EE Libraries and choose Add
7. In the Add WebLogic J2EE Library reference window, click Browse
8. In the Select WebLogic J2EE Library window, click Add
9. In the Add WebLogic J2EE Library window, click Browse
10. Browse to the plug-in's `alcs-uup-lib-5.1.ear` and click Open.
11. Click OK in the remaining open windows.
12. Deploy and run your Portal EAR as usual.

You should now be able to access the WebLogic Portal UUP in the Portal Administration Console.

Setting up the Database

This section explains how to create an AquaLogic Commerce Services database schema using one of the following supported databases:

- Microsoft SQL Server 2005
- MySQL 5.0.x
- Oracle 10g

Microsoft SQL Server 2005

This section contains the instructions for creating a Microsoft SQL Server 2005 AquaLogic Commerce Services database.

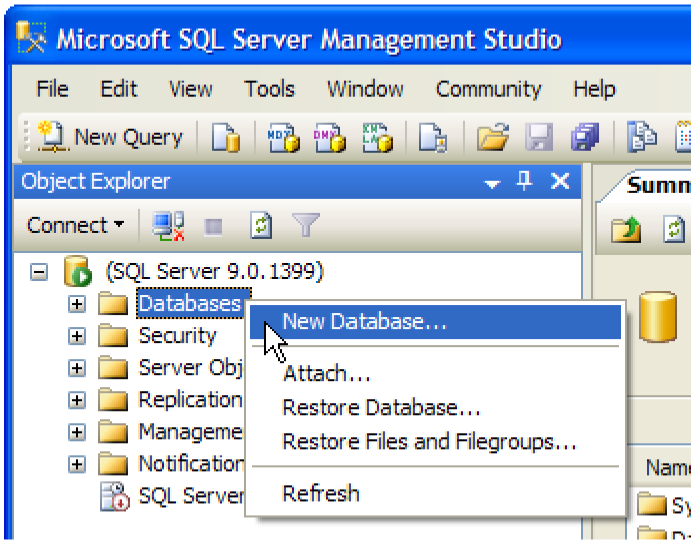
The following instructions assume that Microsoft SQL Server 2005 is running, and that the SQL Server Management Studio is installed.

Connect to Microsoft SQL Server

1. Launch the SQL Server Management Studio by clicking *Start* > *Microsoft SQL Server 2005* > *SQL Server Management Studio*
2. Enter the login information for your SQL Server and click *Connect*.

Create the ALCS Database

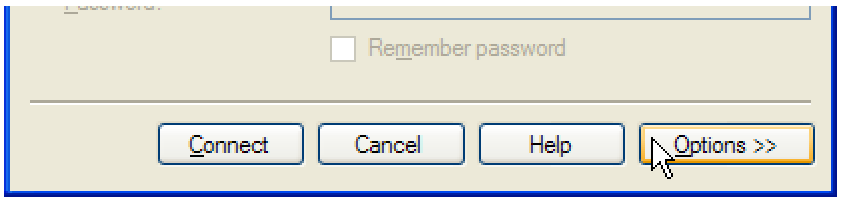
1. Right-click the *Databases* node in the Object Explorer, and click *New Database*...



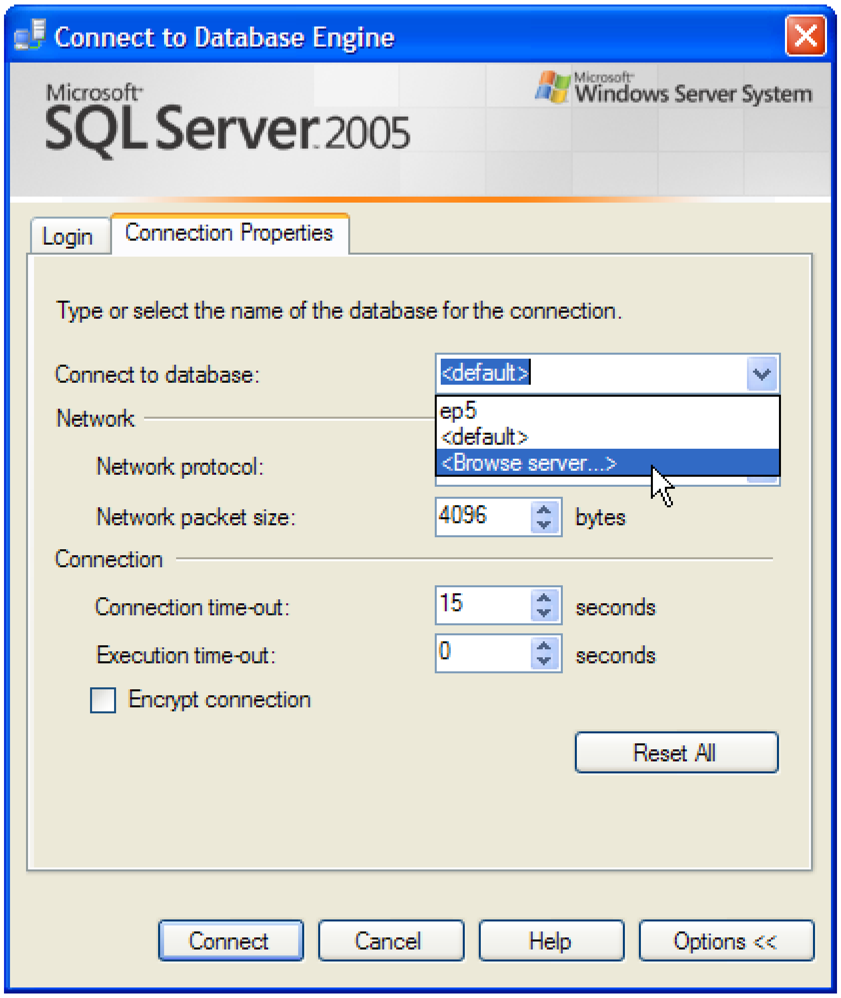
2. Enter “alcs” in the *Database name* field, and click *OK* to create the database. It may take a few seconds.

Populate the Database

1. On the *File* menu point to *Open*, and then click on *File...*
2. Navigate to the <WL_HOME>\commerce\database-scripts\mssql folder
3. Select *schema.sql* and click *Open*
4. In the *Connect to Database Engine* dialog click on the *Options* >> button that is in the bottom right corner.



5. Click on the dropdown next to *Connect to database* and select <*Browse server...*>.


Connect to Database Engine
Microsoft SQL Server 2005
Microsoft Windows Server System
Login
Connection Properties

Type or select the name of the database for the connection.

Connect to database: <default>

ep5
<default>
<Browse server...>

Network

Network protocol:

Network packet size: 4096 bytes

Connection

Connection time-out: 15 seconds

Execution time-out: 0 seconds

☐ Encrypt connection

Reset All
Connect
Cancel
Help
Options <<

6. In the *Browse for databases* dialog click *Yes*
7. Navigate to the database you created in the previous section, and click *OK*

8. Click *Connect*
9. On the *Query* menu click on *Execute*. This may take a few seconds
10. Repeat steps 1 - 9 for the rest of the *.sql files in the following order:
    - base-insert.sql
    - snapitup-insert.sql (optional, deploys the AquaLogic Commerce Services Demo Store "Snap It Up" on your database)
11. When you are finished the above steps you may exit the SQL Server Management Studio
12. If you chose to run the snapitup-insert.sql script you will also need to import the SnapItUp product catalog data

MySQL 4

This section contains the instructions for creating a MySQL AquaLogic Commerce Services database. These instructions assume that you have MySQL 4 installed in c:\mysql.

If you have not configured your MySQL server to start automatically, follow steps 1 to start your server.

1. Start your database server as follows:

   a. Open a DOS command window by clicking: *Start* > *Run*
   b. Type: “cmd” and then press ENTER
   c. Change folder to: c:\mysql\bin
   d. Type: “mysqld-nt” and press ENTER to start the database.
2. Repeat steps (a) to (c) above to open another DOS command window, and type the following to enter mysql console:

```
mysql -u root
```

   If you configured the root user to use a password, type

```
mysql -u root -p
```

   Type the password for the root user when prompted.

3. Create a database called ALCS.

```
mysql> create database ALCS character set utf8;
```

   If you choose to use a database name other than ALCS, please make sure this value is configured correctly when setting up your data source. (See 2 - JNDI and JDBC Configuration)

4. Create a user account.

If the MySQL server and the web application server run on different hosts, create a user account and grant privileges to the host which runs the web application server with the following command:

```
mysql> grant all privileges on ALCS.* to username@'ipaddress';
```

If your database is on a different machine than the web application server, you may want to specify a password by adding identified by 'password' to your grant command.

Commit the permission changes:

```
mysql> flush privileges;
```

5. Change database to ALCS:

```
mysql> use ALCS;
```

6. Copy the sql files from <WL_HOME>\commerce\database-scripts\mysql to c:\mysql\bin.
7. Create a blank ALCS database:

```
mysql> source schema.sql;
mysql> source base-insert.sql;
```

8. Optionally, deploy the AquaLogic Commerce Services Demo Store "Snap It Up" to your database

```
mysql> source snapitup-insert.sql;
```

9. If you chose to run the snapitup-insert.sql script, you will also need to import the SnapItUp product catalog data (see the Running SnapItUp Import Jobs section).

Oracle 10g

This section contains the instructions for creating an Oracle 10g AquaLogic Commerce Services database.

⚠ You will need to set the database to use case-insensitive comparisons and also ensure that the indexes are created in case-insensitive mode to allow searches to be case-insensitive and to avoid full table scans on certain tables.

- When installing Oracle 10g on a Windows system as a domain user as opposed to a local administrator you must set the SQLNET.AUTHENTICATION_SERVICES configuration value to NONE in the <ORACLE_BASE>\NETWORK\ADMIN\sqlnet.ora file (where <ORACLE_BASE> refers to the Oracle home directory on your system, for example C:\Oracle\product\10.2.0\db_1).
- In order to install Oracle on a Windows system with a dynamically assigned IP address you must install the Microsoft Loopback Adapter: http://support.microsoft.com/kb/839013

The following instructions assume that Oracle is already installed and running.

Creating the alcs Database

1. Launch the Oracle Database Configuration Assistant via the start menu by clicking *Start* » *All Programs* » *Oracle - OraDb10g_home1* » *Configuration and Migration Tools* » *Database Configuration Assistant*.
2. Select *Create a Database*, and click *Next*
3. Select *General Purpose*, and click *Next*
4. Enter “alcs” as the *Global Database Name* and the *SID*
5. Click *Next*
6. Enter a password and click *Next*
7. Select *File System* and click *Next*
8. Select *Use Database File Locations from Template* and click *Next*
9. Click *Next*
10. Ensure that *Sample Schemas* is unchecked, and click *Next*
11. On the *Character Sets* tab select *Choose from the list of character sets* and select *UTF8 - Unicode 3.0 UTF-8 Universal character set, CESU-8 Compliant,* and click *Finish*
12. Click *OK* on the confirmation dialog, and the database creation process will start. It may take a few minutes to complete.

Creating and Populating the alcs Schema

1. Click *Start* » *Run*
2. Enter “cmd” into the *Open* field, and click *OK* to launch the Windows command line
3. Navigate to the <WL_HOME>\commerce\database-scripts\oracle folder
4. Type “sqlplus” into the command prompt, and press Enter
5. Type “@schema.sql” into the SQL>* prompt, and press Enter
6. Repeat step 5 for the rest of the *.sql files in the following order:
    - base-insert.sql
    - snapitup-insert.sql (optional, deploys the AquaLogic Commerce Services Demo Store "Snap It Up" on your database)
7. When you are finished type quit to exit SQL*Plus
8. If you chose to run the snapitup-insert.sql script you will also need to import the SnapItUp product catalog data (see the Running SnapItUp Import Jobs section)

Running SnapItUp Import Jobs

This section is for users who wish to import the SnapItUp product catalog data from the CSV files provided with AquaLogic Commerce Services.

1. Log into the Commerce Manager
2. Click on *Admin*
3. Click on *Import*
4. Choose *Select existing import job to run*
5. Choose the first import job from the picklist called "01-SnapItUp"
6. Click *Next*
7. Choose *Execute Import Now*
8. Click *Proceed* and wait for the import job to finish
9. Repeat steps 3 through 8 for each existing import job in the picklist, being sure to import them in the order in which they are numbered (01, 02, 03, etc).

JNDI and JDBC Configuration

This section discusses how to configure your JNDI and JDBC settings to connect your application server to your database.

JNDI Configuration

The default JNDI name used for the Elastic Path data source is “jdbc/epjndi”. This should not be changed. If you must use a different JNDI name, you must change this in all locations where it is listed in this document.

In addition, the JNDI name must be changed in the following places.

- <ALCS_APP_HOME>\commerceServices\WEB-INF\web.xml
- <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\web.xml
- <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\web.xml
- <ALCS_APP_HOME>\commerceServices\WEB-INF\conf\spring\dataaccess\hibernate\hibernate-config.xml
- <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\dataaccess\hibernate\hibernate-config.xml
- <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\conf\spring\dataaccess\hibernate\hibernate-config.xml

JDBC Configuration

The configuration information for the different databases is listed in this section.

MySQL

Database Driver JAR file: mysql-connector-java-3.1.11-bin.jar
Database Driver: com.mysql.jdbc.Driver
Database Connection URL:
jdbc:mysql://*HOSTNAME:PORT/DBNAME*?autoReconnect=true&useUnicode=true&characterE
ncoding=UTF8
(Default Port is 3306)
Database Driver Download: http://dev.mysql.com/downloads/connector/j/3.1.html

Oracle

Database Driver JAR file: ojdbc14.jar
Database Driver: oracle.jdbc.driver.OracleDriver
Database Connection URL: jdbc:oracle:thin:@_HOSTNAME:PORT:SID
(Default port is 1521)
Database Driver Download:
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc9201.html

Microsoft SQL Server

Database Driver JAR file: msutil.jar, msbase.jar, mssqlserver.jar (all 3 are required)
Database Driver: com.microsoft.jdbc.sqlserver.SQLServerDriver
Database Connection URL:
jdbc:microsoft:sqlserver://*HOSTNAME:PORT*;DatabaseName=*DBNAME*;selectMode=cursor
(Default port is 1433)
Database Driver Download:
http://www.microsoft.com/downloads/details.aspx?FamilyID=07287b11-0502-461a-b138-
2aa54bfdc03a&DisplayLang=en

Configuring AquaLogic Commerce Services

This section explains how to configure the various AquaLogic Commerce Services applications.

Each sub-section explains the common configuration parameters. There are many other configuration parameters not described here which are further documented in the comments of the various configuration files listed. These other parameters can use their default values and should only be modified if needed.

Storefront

Configure the following files for your AquaLogic Commerce Services Storefront application:

- acegi.xml
- commerce-config.xml
- ep-treecache.xml

- hibernate-config.xml
- quartz.xml
- treecache.xml
- velocity.xml
- util-config.xml

acegi.xml

AquaLogic Commerce Services uses the Acegi security framework to perform user authentication and authorization.

The acegi.xml file for configuring acegi is located in the following application-specific paths. Since the acegi.xml file contains different settings for each application, you will need to configure each of them separately.

- Storefront – < >\commerceServices\WEB-INF\conf\spring\security\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\security\
- Connect – <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\conf\spring\security\

Configuring the HTTPS port redirect

Specify your HTTP port number in the <entry> tag as described in the example below.

In acegi.xml, specify the HTTP port number in the key attribute of the entry tag and the HTTPS port number in the <value> tag as shown in the example below. This allows the application server switch to the HTTPS port for pages that require it. Not all URLs in an application are secure. For each application, we have specified a default list of URLs which are secure. You can modify these lists if desired as explained in the Acegi - security framework section of the AquaLogic Commerce Services Developer Guide.

In the example below, we have specified the standard port numbers of 80 for HTTP and 443 for HTTPS.

```
<!-- port # are specified in default.xml -->
    <bean id="portMapper"
class="org.acegisecurity.util.PortMapperImpl">
   <property name="portMappings">
     <map>
              <entry key="80"><value>443</value></entry>
         </map>
     </property>
    </bean>
```

guided-navigation.xml

The AquaLogic Commerce Services Storefront uses guided-navigation.xml to generate the guided navigation menu to display to users when they are browsing a category or executing a search. Guided Navigation appears in the User Interface as a set of links on the left side of the pages. Users can click links in the guided navigation area to filter the list of products to find the product that they are looking for.

The guided-navigation.xml file is located in the following location:
<ALCS_APP_HOME>\commerceServices\WEB-INF\

There are 3 types of filters defined in elements in the configuration file:

- attribute – defines the simple value filters for an attribute.
- attributeRange – defines the range values for an attribute.
- price – defines the price ranges.

Attribute Simple Value Filter

This defines the simple value filter for an attribute. It starts with the "attribute" tag as shown in the following example.

```
<attribute key="A00551" localized="false">
  <simple id="01" value="CCD" />
  <simple id="02" value="CMOS" />
  <simple id="03" value="Super HAD CCD" />
  <simple id="04" value="Live MOS" />
  <simple id="05" value="3CCD" />
  <simple id="06" value="LBCAST" />
</attribute>
```

- key="A00551" is the reference to the attribute key. Each attribute can only be defined once in the XML file.
- localized="false" means that this attribute is not localized.
- <simple id="01" value="CCD" /> defines each available value. The "id" should be unique for this attribute since it is used to reference the filter. The "value" should be unique too.

If the attribute is localized, it will have the following differences in the XML.

- The "localized" property should be "true".
- For each simple value, it should have the "language" property defined.
- The "id" should be unique for this attribute since it is used to reference this filter, but the "value" can be the same for different languages.

The following is a localized attribute example.

```
<attribute key="A00556" localized="true">
  <simple id="01" value="TFT active matrix" language="en" />
  <simple id="02" value="Matrice active TFT" language="fr" />
  <simple id="03" value="LCD passive matrix" language="en" />
  <simple id="04" value="LCD à matrice passive" language="fr" />
  <simple id="05" value="None" language="en" />
  <simple id="06" value="Aucun(e)" language="fr" />
  <simple id="07" value="LCD" language="en" />
  <simple id="08" value="LCD" language="fr" />
</attribute>
```

Attribute Range Filter

This defines the range value for an attribute. Only attributes with type "Integer", "Decimal" or "Short-Text" can have ranges defined. It starts with the "attributeRange" tag. The range filter can have subranges defined.

```
<attributeRange key="A00140" localized="false">
  <range lower="" upper="1.9" id="_1.9">
    <display language="en">
      <value>1.9 in. &amp; Under</value>
      <seo>1.9in-and-under</seo>
    </display>
    <display language="fr">
      <value>1.9 in. et dessous</value>
      <seo>1.9in-et-dessous</seo>
    </display>
  </range>
  <range lower="2" upper="2.9" id="2_2.9">
    <display language="en">
      <value>2 to 2.9 in.</value>
      <seo>2-to-2.9in</seo>
    </display>
    <display language="fr">
      <value>2 et 2.9 in.</value>
      <seo>2-et-2.9in</seo>
    </display>
  </range>
  <range lower="3" upper="" id="3_">
    <display language="en">
```

```
      <value>3in. &amp; Up</value>
      <seo>3in-and-up</seo>
    </display>
    <display language="fr">
      <value>3in. et Plus</value>
      <seo>3in-et-plus</seo>
    </display>
  </range>
</attributeRange>
```

The attribute range filter also has the "key" property and "localized" property. Each "range" tag represents one range, which may have the "lower" value and "upper" value set. If the "lower" value is not set, then there is no minimum allowed value for the attribute filter. If the "upper" value is not set, then the there is no maximum allowed value for the attribute filter.

The "display" tag contains the information to display the filter on the guided navigation links.

Price Filter

This defines the range value for the price. It starts with the "Price" tag. The range filter can have subranges defined.

```
<price currency="USD" localized="false">
  <range lower="" upper="100" id="_100">
    <display language="en">
      <value>Under $100</value>
      <seo>under-$100</seo>
    </display>
    <display language="fr">
      <value>au-dessous de $100</value>
      <seo>au-dessousd-de-$100</seo>
    </display>
  </range>
  <range lower="100" upper="300" id="100_300">
    <display language="en">
      <value>$100 to $300</value>
      <seo>$100-to-$300</seo>
    </display>
    <display language="fr">
      <value>$100 et $300</value>
      <seo>$100-et-$300</seo>
    </display>
    <range lower="100" upper="200" id="100_200">
```

```
      <display language="en">
        <value>$100 to $200</value>
       <seo>$100-to-$200</seo>
      </display>
      <display language="fr">
        <value>$100 et $200</value>
       <seo>$100-et-$200</seo>
      </display>
    </range>
    <range lower="200" upper="300" id="200_300">
      <display language="en">
        <value>$200 to $300</value>
       <seo>$200-to-$300</seo>
      </display>
      <display language="fr">
        <value>$200 et $300</value>
       <seo>$200-et-$300</seo>
      </display>
    </range>
  </range>
  <range lower="300" upper="" id="300_">
    <display language="en">
      <value>More than $300</value>
      <seo>more-than-$300</seo>
    </display>
    <display language="fr">
      <value>au-dessous de $300</value>
      <seo>au-dessousd-de-$300</seo>
    </display>
  </range>
</price>
```

The price filter has the "currency" and "localized" property. Each "range" tag represents one price range, which may have the "lower" and "upper" values set. If the "lower" value is not set, then there is no minimum allowed value for the price filter. If the "upper" value is not set, then there is no maximum allowed value for the price filter.

The "display" tag contains the information to display the filter on the guided navigation links.

commerce-config.xml

AquaLogic Commerce Services's configuration settings are stored in the commerce-config.xml files located in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\
- Connect – <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\

The following are the names and descriptions of the key elements in the configuration file.

- commerceConfig - the main element containing all the configuration settings for an AquaLogic Commerce Services implementation.
    - web - the element containing all web-related settings.
        - web.sf.context.url - the context part of the URL for the Storefront application. For example, "alcssf". Set this to an empty string if the application is in the root context.
        - catalog.topSeller.count - the number of top selling products to display on the category pages
        - catalog.view.pagination - the number of items to display per page on the catalog browsing and search pages
    - customer - the element containing customer settings
        - userid.mode - determines whether the customer's email address will be used as userid or a unique userid will be generated based on the email address
    - asset - the element containing the paths to all assets
        - catalog.asset.path - the root path where catalog assets are stored
        - image.asset.subfolder - the subdirectory under the root path where images are stored
        - file.asset.subfolder - the subdirectory under the root path where non-image files are stored
        - digitalgoods.asset.subfolder - the subdirectory under the root path where digital goods (such as electronic books) are stored
    - mail - the element containing all mail-related settings.
        - mail.host - domain or IP address of your SMTP server. This is used to send email confirmations.
        - mail.host.port - the port of the SMTP server.
        - email.from - the "From" email address for all emails generated by AquaLogic Commerce Services, e.g. customerservice@elasticpath.com.
        - email.from.name - the "From" personal name for all emails generated by AquaLogic Commerce Services, e.g. AquaLogic Commerce Services.

    - email.store.url - URL for images etc. in emails generated by the Storefront
    - email.cm.url - URL for images etc. in emails generated by the Commerce Manager
  - locale - the element containing all locale-related settings.
    - locale.default - system default locale. Must be valid ISO Language Code and country code defined by ISO-639 and ISO-3166, such as en_CA etc.
    - locale.all - all locales that the AquaLogic Commerce Services application supports. E.g. "en_CA, fr_CA, en_US"
    - locale.date.format - the format of the dates stored in the properties files
    - content.encoding - the character encoding for the web pages and emails
    - datafile.encoding - the character encoding for import data files, export data files, and report files
    - country.default - default country for the Storefront (only required if the default locale set does not include the country code)
  - currency - the element containing all currency-related settings.
    - currency.default - system default currency. Must be valid ISO Currency Code - 3-letter codes as defined by ISO-4217, such as CAD etc.
    - currency.all - all currencies that the AquaLogic Commerce Services application supports. E.g. "CAD, USD"
  - units - the element containing the settings for units of measurement
  - units.length - the unit of length to display. For example, "CM" for centimeters.
  - units.weight - the units of weight to display. For example, "KG" for kilograms.
  - payment - the element containing information about configured payment gateways. Configure your available payment methods here.
  - seo - the element containing search engine optimization (SEO) settings, which generates search engine friendly URLs for products and categories
    - seo.enable - enables/disables SEO URL generation
  - attributeFilter - the element containing settings for guided navigation, which allows customers to narrow down their browsing/search results to find a product that they are looking for
    - attributeFilter.enable - enables/disables guided navigation
  - dynamicimagesizing - the element for setting image display options
    - dynamicimagesizing.jpegquality - the quality of the displayed jpeg images
    - dynamicimagesizing.noimage - the name of the image to display if the image to be displayed is not found
  - search - the element containing catalog search settings

    - maxReturnNumber - the maximum number of products to return in a search result
    - minimumSimilarity - used in searches that do not require an exact match (fuzzy searches). This is the fraction of similarity required between the search term and a matching term
    - prefixLength - used in searches that do not require an exact match (fuzzy searches). This is the number of characters at the beginning of the term that must match exactly between the search term and matching term
    - minimumResultsThreshold - the number of results under which alternate queries (such as spelling suggestions) will be suggested to the user
    - maximumSuggestionsPerWord - the maximum number of suggestions generated for each word in the query
    - accuracy - the fraction of similarity required between suggested words and original words in the query
  - onepage - the element containing settings for One Page checkout
    - onepage.enable - the boolean true/false value specifying if One Page checkout is enabled or not
  - powerreviews - the element containing settings for PowerReviews product reviews
    - powerreviews.enable - the boolean true/false value specifying if PowerReviews is enabled or not
    - powerreviews.merchantid - the merchant account id provided by PowerReviews
  - security - the element containing encryption information
    - encryption.key - the encryption key for encrypting customer credit card numbers stored in the TORDERPAYMENT table. You must configure this before order checkout will work.
  - productcache - the element containing product cache settings
    - productcache.preload - the boolean true/false value specifying if all products will be pre-loaded into the cache upon startup or not
  - database - the element containing database settings (**applies to AquaLogic Commerce Services 5.1.1 only**)
    - database.product.name - the name of the database being used ("postgresql", "db2", "sqlserver", "mysql", or "oracle")

Customer User Id Generation

AquaLogic Commerce Services by default uses a customer's email address as the user id, but you can change the user id generation mode in commerce-config.xml as shown below.

```
<property name="userid.mode" value="1" />
```

The possible values for the user id generation mode are:

1 - Use user's email as the user id, this is the default value. If you use JAAS, you can't use this mode.

2 - Generates a unique permanent user id by appending a random four digit suffix to the email address and uses this as the user id. The user id is created when the customer first creates an account. Later, when the customer changes the email address, the user ID will not be changed. For example, if a customer email address is a@a.com, the user id would be a@a.comXXXX, where XXXX is a random generated string.
3 - Independent email and user id. This mode is not yet supported.

Customer Maintenance Mode

This setting indicates whether to add, delete and maintain user passwords through the Commerce Manager. By default the customer password is maintained in the AquaLogic Commerce Services database. If customers are authenticated through JAAS, (e.g. authenticated in WebLogic Server) the customer maintenance mode must be set to 2. This setting will only affect the Commerce Manager, not the Storefront.

```
<property name="user.maintain.mode" value="1" />
```

The possible values are:

1 - Maintain user passwords in the local DB, this is the default value.

2 - Maintain user passwords through JAAS, the Commerce Manager will disable the create/delete and change Password functions in this mode.

Payment Configuration

The payment gateway used by the Storefront is configured in the <payment> block within the commerce-config.xml file as follows.

- The checkout transaction behavior can be set to one of the following two values:
    - authorization - When an order is placed, the payment will be pre-authorized only. The payment will be captured later when the shipment for the goods is released. This mode is typically used whenever a store sells shippable goods.
    - sale - When an order is placed, the payment for the order will be captured immediately. This mode is typically used when only digital goods can be purchased in the store.
- The <gateway> block specifies the fully-qualified class name of the payment processor to be used, which must implement PaymentGateway.
    - The names of supported card types are also specified in this block. Note that the supported card types are for display only, since the payment processor determines the card type from the card number.
    - The ValidateCvv2 element determines whether the card security code should be requested from the user and sent to the payment processor.

- Only ONE "credit card processing" payment gateway is supported at a time. Changing to a different payment gateway requires re-configuring the payment gateway settings in the commerce-config.xml file.
- PayPal Express Checkout can also be configured alongside the chosen "credit card processing" payment gateway in the <payment> block.
  Each gateway has its own gateway-specific properties.

CyberSource Credit Card Sample Configuration

```
<gateway name="CyberSource"
class="com.elasticpath.domain.payment.impl.CyberSourcePaymentGatewayI
mpl">
     <property name="merchantID" value="YOUR_MERCHANT_ID"/>
     <property name="keysDirectory"
value="RELATIVE/PATH/TO/CERT/DIRECTORY (from WEB-INF)" />
     <property name="targetAPIVersion" value="1.19"/>
     <property name="sendToProduction" value="false"/>
     <property name="logMaximumSize" value="10"/>
     <property name="enableLog" value="false"/>
     <property name="logDirectory" value="WEB-INF/log"/>
</gateway>
```

Paypal Payflow Pro (formerly Verisign) Sample Configuration

```
<gateway name="PayflowPro"
class="com.elasticpath.domain.payment.impl.PayflowProPaymentGatewayIm
pl">
    <property name="user" value="YOUR_USERNAME"/>
    <property name="password" value="YOUR_PASSWORD"/>
    <property name="vendor" value="YOUR_VENDOR"/>
    <property name="partner" value="Verisign"/>
    <property name="certificateLocation"
value="RELATIVE/PATH/TO/CERT/DIRECTORY (from WEB-INF)"/>
    <property name="hostAddress" value="test-payflow.verisign.com"/>
    <property name="hostPort" value="443"/> <!--optional - defaults
to 443-->
    <property name="proxyAddress" value=""/> <!--optional-->
    <property name="proxyPort" value=""/> <!--optional-->
    <property name="proxyLogon" value=""/> <!--optional-->
    <property name="proxyPassword" value=""/> <!--optional-->
</gateway>
```

Authorize.Net Sample Configuration

```
<gateway name="AuthorizeNet"
class="com.elasticpath.domain.payment.impl.AuthorizeNetPaymentGateway
Impl">
```

```
    <property name="authorizeNetURL"
value="https://certification.authorize.net/gateway/transact.dll"/>
    <property name="testMode" value="true"/>
    <property name="loginID" value="XXXX"/>
    <property name="transKey" value="XXXX"/>
    <property name="version" value="3.1"/>
    <property name="delimChar" value="|"/> <!--optional-->
    <property name="encapChar" value=""/> <!--optional-->
</gateway>
```

Fuzzy Search in Lucene

The search mechanism in AquaLogic Commerce Services takes advantage of the Fuzzy Search feature in Lucene. Fuzzy Search provides a way to search for terms similar to a specified term (for example with misspelled words).

The Fuzzy Search can be configured with two settings in the commerce-config.xml

1. minimumSimilarity - Value between 0 and 1 to set the required similarity between the query term and the matching terms. For example, for a minimumSimilarity of 0.5 a term of the same length as the query term is considered similar to the query term if the edit distance between both terms is less than length(term)*0.5 where edit distance is a measure of similarity between two strings and distance is measured as the number of character deletions, insertions, or substitutions required to transform one string to the other string.

2. prefixLength - length of common non-fuzzy prefix

Spelling Suggestions

Spelling suggestions can be generated for Storefront keyword queries. The Lucene Spellchecker is used to find words similar to terms in the original keyword query. This is configured in the following elements.

- minimumResultsThreshold - If the number of results does not exceed this setting the search will attempt to suggest possible alternate queries
- maximumSuggestionsPerWord - The maximum amount of suggestions that will be generated for each word in the query
- accuracy - The degree of similarity that suggested words will have to the original keywords

Encryption Key

For security reasons, this key field is not pre-set to any particular value out-of-the-box. You must make up your own site-specific value, and set this field. Otherwise, the application will NOT work with the empty default encryption key. Any string at least 24 characters may be used for the key. This must be entered in all AquaLogic Commerce Services application commerce-config.xml configuration files (Storefront, Commerce Manager, and Connect).

The same key must be used in all places in order for the credit card numbers that have been encrypted in the Storefront to be correctly decrypted and masked in the other applications.

One Page Checkout

One Page checkout is included as part of the AquaLogic Commerce Services install. It is enabled by setting "onepage.enable" to "true". You can always revert back to using the standard checkout process by changing this value to "false".

```
<onepage>
  <property name="onepage.enable" value="false"/>
</onepage>
```

PowerReviews Product Reviews (optional)

You must have setup an account with PowerReviews to use this functionality. Once setup correctly, PowerReviews product reviews can be enabled by setting "powerreviews.enable" to "true". You can disable PowerReviews product reviews by setting this value to "false" and no product review information will display on the Storefront.

AquaLogic Commerce Services trial edition comes setup with a demo PowerReviews merchant account with some sample reviews, but any reviews created with this demo merchant id will not display on your Storefront. Once you have setup an account with PowerReviews, replace the value of "powerreviews.merchantid" with the merchant id assigned to you by PowerReviews.

```
<powerreviews>
  <property name="powerreviews.enable" value="true"/>
  <property name="powerreviews.merchantid" value="7609"/>
</powerreviews>
```

ep-treecache.xml

AquaLogic Commerce Services implements a specialized caching system for product information only. JBoss Cache is used as the underlying implementation of this product cache and several parameters of the JBoss Cache can be configured in ep-treecache.xml. Note that this caching system is used only when the singleCachingProductRetrieveStrategy is configured as the strategy for retrieving products. Product retrieval strategies are configured in ServiceSF.xml.

Tree Cache file locations:

- ep-treecache.xml is located in <ALCS_APP_HOME>\commerceServices\WEB-INF\classes\
- ServiceSF.xml is located in <ALCS_APP_HOME>\WEB-INF\conf\spring\service\

The following parameters can be configured in ep-treecache.xml's EvictionPolicyConfig element

- timeToLiveSeconds - The idle time in seconds that should elapse before a node is removed from the tree cache. A value of 0 denotes no limit.
- maxAgeSeconds - The maximum time in seconds that a node in the tree cache can exist regardless of idle time. A value of 0 denotes no limit.

There are additional configurable parameters in ep-treecache.xml, however, we do not recommend that you modify them unless you are performing advanced JBoss Cache optimization or troubleshooting.

hibernate-config.xml

Hibernate is used in AquaLogic Commerce Services for object to relational database mapping. The hibernate-config.xml file used to configure Hibernate is located in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\conf\spring\dataaccess\hibernate\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\dataaccess\hibernate\
- Connect – <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\conf\spring\dataaccess\hibernate\

Setting Hibernate properties for a production environment

Ensure that the "hibernate.show_sql" property is set to false in production, since having it turned on adversely affects performance. The setting of this property determines whether or not the Hibernate generated SQL is written to system log files.

The "hibernate.dialect" property needs to be set to a value appropriate for the database type that your application is using. Below are the values for each database.

- org.hibernate.dialect.MySQLDialect
- org.hibernate.dialect.OracleDialect
- org.hibernate.dialect.SQLServerDialect

The following is a set of example hibernate settings found in hibernate-config.xml.

```
<property name="hibernateProperties">
   <props>
      <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
      <prop key="hibernate.show_sql">false</prop>
      <prop key="hibernate.jdbc.use_streams_for_binary">true</prop>
```

```
        <prop key="hibernate.jdbc.batch_size">20</prop>
    </props>
</property>
```

quartz.xml

Quartz is used in AquaLogic Commerce Services to execute scheduled jobs (for example, building the Lucene index). A quartz.xml file is used to configure quartz jobs for the different AquaLogic Commerce Services applications. The quartz.xml file is located in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\conf\spring\scheduling\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\scheduling\
- Connect – <ALCS_APP_HOME>\commerceServicesConnect\WEB-INF\conf\spring\scheduling\

Quartz files have three basic types of blocks of XML:

- Factory block – The definition of the Scheduler Factory, which is used to populate a Scheduler class with the list of Triggers that it will be responsible for executing.
- Job block – The definition of the job that is being scheduled. This defines the method that will be called by the Trigger and references the class in which the method lives.
- Trigger block – The definition of the trigger that will be used to trigger the job. This will usually be either a SimpleTriggerBean (which runs a job every x milliseconds) or a CronTriggerBean (which will run a job at a specified time).

There is usually one Factory definition in a quartz.xml file, and then each job has both a Job definition and a Trigger definition.

Extended information about Quartz is available from the Quartz website at http://www.opensymphony.com/quartz/

Quartz Cron Configuration

The time/trigger to execute the scheduled job can be set with the cronExpression property. The cron expression contains six required components and one optional component. A cron expression is written on a single line and each component is separated from the next by space. Only the last, or rightmost, component is optional. The table below describes the cron components in detail.

Components of a Cron Expression:

Position	Meaning	Allowed Special Characters
1	Seconds (0-59)	, - * /
2	Minutes (0-59)	, - * /
3	Hours (0-23)	, - * /
4	Day of month (1-31)	, - * / ? L C
5	Month	(either JAN-DEC or 1-12) , - * /
6	Day of week (either SUN-SAT or 1-7)	, - * / ? L C #
7	Year (optional, 1970-2099), when empty, full range is assumed	, - * /

Each component accepts the typical range of values that you would expect, such as 0-59 for seconds and minutes and 1-31 for day of the month. For the month and day of the week components, you can use numbers, such as 1-7 for day of the week, or text such as SUN-SAT.

Each field also accepts a given set of special symbols, so placing a * in the hours component means every hour, and using an expression such as 6L in the day-of-the-week component means last Friday of the month. The table below describes cron wildcards and special symbols in detail.

Cron Expression Wildcards and Special Symbols:

Special Character	Description
*	Any value. This special character can be used in any field to indicate that the value should not be checked. Therefore, our example cron expression will be fired on any day of the month, any month, and any day of the week between 1970 and 2099.
?	No specific value. This special character is usually used with other specific values to indicate that a value must be present but will not be checked.
-	Range. For example 10-12 in the Hours field means hours 10, 11, and 12.
,	List separator. Allows you to specify a list of values, such as MON, TUE, WED in the

Special Character	Description
	Day of week field.
/	Increments. This character specifies increments of a value. For example 0/1 in the Minute field in our example means every 1-minute increment of the minute field, starting from 0.
L	L is an abbreviation for Last. The meaning is a bit different in Day of month and Day of week. When used in the Day of month field, it means the last day of the month (31st of March, 28th or 29th of February, and so on). When used in Day of week, it has the same value as 7---Saturday. The L special character is most useful when you use it with a specific Day of week value. For example, 6L in the Day of week field means the last Friday of each month.
#	This value is allowed only for the Day of week field and it specifies the nth day in a month. For example 1#2 means the first Monday of each month.
C[PD:*]	The Calendar value is allowed for the Day of month and Day of week fields. The values of days are calculated against a specified calendar. Specifying 20C in the Day of month field fires the trigger on the first day included in the calendar on or after the 20th. Specifying 6C in the Day of week field is interpreted as the first day included in the calendar on or after Friday.

[PD:*]C At the time of writing, support for the C special character and specifying both Day of week and Day of month values has not been not completed.

* info extracted from Pro Spring

Storefront Quartz Jobs

The Storefront has three configured Quartz Jobs:

- IndexBuildJob – builds the Lucene search index
- RulebaseCompileJob – builds the promotion rules
- UpdateProductCacheJob – updates the product cache

Building the Lucene search index

- Storefront index build scheduling should normally have a longer time lag to reduce overhead. The default setting is to start the first build 60 seconds after the application server has started and then build every hour.
- In a development environment, you can disable the index build by setting the following flag to true in the env.config file as follows.

```
        ep.disable.index.build=true
```

Using the env.config value to disable an index build will also disable the index build for the Commerce Manager when it is built, so be careful to remember whether index building is enabled or not in your current environment

Building the promotion rule base

Rules are read from the database periodically as invoked by the Quartz scheduler. The scheduler is configured in the quartz.xml configuration file. This configuration file specifies parameters for how often the rule base is to be "recompiled." For performance reasons, the rule base should not be recompiled more than once every few minutes.

For demonstrations and testing, the recompile frequency can be set to a shorter interval such as three seconds so that rules defined in the Commerce Manager will take effect immediately in the Storefront. However, any caching of the rules objects will prevent updates to the set of rules in use. Therefore, it is necessary to turn off caching in hibernate-config.xml file if rules are to take effect immediately (see the hibernate-config.xml section).

Updating the product cache

Depending on your Product Caching Strategy, you may need to periodically update the Product Cache so that changes in product information take effect in the Storefront. If you are using the default caching strategy, it is essential to enable the quartz update product cache job; it is initially configured to run 30 seconds after startup, and then every hour after that. Please see the Product cache section for more information about product caching.

treecache.xml

The Hibernate persistence framework used in AquaLogic Commerce Services can be configured to use JBoss Cache to cache persistent domain objects. The configuration for this caching behavior is specified in treecache.xml which is found in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\classes\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\classes\

The objects that will be cached are listed in hibernate-config.xml (see the hibernate-config.xml section for details).

Warning
Currently, the use of Hibernate caching is not recommended due to a bug in the JBoss Cache eviction policy. It is disabled by default in hibernate-config.xml.

velocity.xml

Velocity is used for UI html rendering in AquaLogic Commerce Services. The velocity.xml file for configuring Velocity is located in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\conf\spring\views\velocity\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\views\velocity\

Setting Velocity properties for a production environment

Changes made to velocity.xml should be applied to both the Storefront and Commerce Manager deployments.

Ensure that the cacheSeconds property is set to -1. When set to a positive value this property allows you to change and test changes to message source resource files without having to restart your application or servlet container. In production it is recommended that this property is disabled and set to -1.

```
<bean id="globalMessageSource"
class="org.springframework.context.support.ReloadableResourceBundleMe
ssageSource">
   <property name="basenames">
      <list>
         <value>/WEB-INF/templates/velocity/globals</value>
      </list>
   </property>
   <!-- Set cacheSeconds to -1 in Production Environment -->
   <property name="cacheSeconds"><value>-1</value></property>
</bean>
```

The following properties in util-config.xml (Storefront and Commerce Manager) should also be set for optimal performance:

- file.resource.loader.cache - "true"
- velocimacro.library.autoreload - "false"
- velocimacro.messages.on - "false"

See the util-config.xml section for more details on these configuration options.

util-config.xml

This configuration file contains the Spring configuration for utility classes. It is located in the following application-specific paths.

- Storefront – <ALCS_APP_HOME>\commerceServices\WEB-INF\conf\spring\commons\
- Commerce Manager – <ALCS_APP_HOME>\commerceServicesManager\WEB-INF\conf\spring\commons\

Within the velocityProperties bean definition, the following properties can be configured.

- file.resource.loader.cache – Set this value to "true" for production to cache templates in Velocity's loader. Set this value to "false" for development so that changes to templates will take effect immediately.
- velocimacro.library.autoreload – This parameter enables/disables automatic reloading of Velocity macros. Set this value to "false" for production. For development, setting this value to "true" will allow changes to Velocity macros to take effect without restarting the server.
- velocimacro.messages.on – Set this value to "false" for production for improved performance. When developing Velocity templates, this value can be set to "true" to see Velocity logging and debugging information.
- runtime.log.logsystem.class – Set this value to org.apache.velocity.runtime.log.NullLogSystem as shown below for production.

```
<entry
key="runtime.log.logsystem.class"><value>org.apache.velocity.
runtime.log.NullLogSystem</value></entry>
```

Product Cache

AquaLogic Commerce Services implements a special cache for Storefront product information only. The underlying cache mechanism is JBoss Cache and configuration for this instance of JBoss Cache is in ep-treecache.xml (see the ep-treecache.xml section for details). There are several caching strategies that are available to choose from and the nature of each strategy is described below. The strategies are configured in ServiceSF.xml.

Tree Cache file locations:

- ep-treecache.xml is located in <WL_HOME>\samples\commerce\commerceApp\commerceServices\WEB-INF\classes\
- ServiceSF.xml is located in <WL_HOME>\samples\commerce\commerceApp\commerceServices\WEB-INF\conf\spring\service\

Configuring AquaLogic Commerce Services to use a particular caching strategy

Caching strategies are configured in the serviceSF.xml Spring configuration file. There are several places in this file where the caching strategy to be used is specified by referencing the bean id of the strategy. See below for an example.

```
<bean id="productViewService"
```

```
class="com.elasticpath.service.catalogview.impl.ProductViewServiceImp
l">
     <property name="elasticPath">
          <ref bean="elasticPath" />
     </property>
     <property name="productService">
     <ref bean="productService" />
     </property>
     <property name="productRetrieveStrategy">

          <!-- Specifiy the bean id of the caching strategy
               to be used here -->
          <ref bean="noCachingProductRetrieveSrategy" />
     </property>
</bean>
```

To change the caching strategy, **find all instances** where the current strategy is referenced by bean id as shown above and change the id to the id of new strategy you wish to use.

Caching Strategies

The available caching strategies and their bean ids that are to be set in serviceSF.xml are described in the following sections.

NoCachingProductRetrieveStrategy

This strategy always retrieves products from the database. Retrieved products are passed through the rules engine.

The bean id for this strategy is "noCachingProductRetrieveSrategy"

SingleCachingProductRetrieveStrategy

This strategy retrieves a product from the database, executes rules on it, and then caches this single instance of the product. On all subsequent requests for the product, the single cached instance is returned. Because only a single instance of the product is cached, this strategy cannot be used in combination with promotion rules that compute different prices for different customers.

The bean id for this strategy is "singleCachingProductRetrieveStrategy"

RefreshableCachedProductRetrieveStrategy

This strategy is an extension of the SingleCachingProductRetrieveStrategy that allows the cached copy of a product to be periodically refreshed from the database. Since products in the cache can be updated, you may choose to configure JBoss Cache so that products are never evicted. To prevent eviction, modify ep-treecache.xml so that the timeToLiveSeconds and maxAgeSeconds attributes have a value of 0 as shown below.

```
<!--  Cache wide default -->
<region name="/_default_">
     <!--  This is the maximum number of nodes allowed in this
region.
           Any integer less than or equal to 0 will throw an
exception
           when the policy provider is being configured for use. -->
     <attribute name="maxNodes">50000</attribute>

     <!--  Time to idle (in seconds) before the node is swept away.
           0 denotes no limit. -->
     <attribute name="timeToLiveSeconds">0</attribute>

     <!-- Time an object should exist in TreeCache (in seconds)
          regardless of idle time before the
          node is swept away. 0 denotes no limit. -->
     <attribute name="maxAgeSeconds">0</attribute>
</region>
```

When using this strategy, you will need to enable a job in quartz.xml (Storefront) that will periodically retrieve modified products from the database and update their cache entries. To enable this job, ensure that the updateProductCacheTrigger bean is declared in the schedulerFactory bean as shown below.

```
<bean id="schedulerFactory"

class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
     <property name="triggers">
          <list>

              <!-- The presence of this trigger enables the cache
                     refresh job -->
               <ref bean="updateProductCacheTrigger" />

          </list>
     </property>
</bean>
```

You can configure the start delay and the interval that products are refreshed in the quartz.xml XML block shown below.

```
<bean id="updateProductCacheTrigger"
     class="org.springframework.scheduling.quartz.SimpleTriggerBean">
     <property name="jobDetail" ref="updateProductCacheJob" />

     <!-- Start delay in *milliseconds* -->
     <property name="startDelay" value="30000" />

     <!-- repeat interval in *milliseconds* (3600000 = 1 hour) -->
     <property name="repeatInterval" value="3600000" />
</bean>
```

When using the RefreshableCachedProductRetrieveStrategy, it is necessary to periodically re-execute promotion rules on cached products because the promotion rules that apply to the cached product may have changed. You can configure the amount of time for which a product may be returned from the cache without re-running rules in the serviceSF.xml configuration file as shown in the XML block below.

```
<bean init-method="start"
id="refreshableCachedProductRetrieveSrategy"
     class="com.elasticpath.service.catalogview.

impl.RefreshableCachedProductRetrieveStrategyImpl">
     <property name="epRuleEngine">
          <ref bean="epRuleEngine" />
     </property>
     <property name="productService">
          <ref bean="productService" />
     </property>
     <property name="defaultProductLoadTuner">
          <ref bean="productLoadTunerForProductPage" />
     </property>

     <!-- Set the rule execution interval here (in seconds).
     If a product is requested from the cache and it has
     existed in the cache for longer than this period of
     time then it will have rules re-executed on it. -->
     <property name="ruleExecutionIntervalSeconds">
          <value>30</value>
     </property>
</bean>
```

The bean id for this strategy is "refreshableCachedProductRetrieveSrategy"

Important

If you configure JBoss Cache in ep-treecache.xml to never evict products (by setting time values of 0), then it is necessary to use the RefreshableCachedProductRetrieveStrategy and configure the updateProductCache job in quartz.xml.

Commerce Manager (CM)

Follow the Storefront instructions for configuring acegi.xml in your Commerce Manager application.

Then configure the following files in your Commerce Manager application:

- commerce-config.xml
- hibernate-config.xml
- quartz.xml
- velocity.xml
- util-config.xml

commerce-config.xml

See the Storefront commerce-config.xml section for the details on property configurations that are common to the Storefront, Commerce Manager, and Connect applications.

The following are the names and descriptions of the key Commerce Manager elements the commerce-config.xml configuration file that differ from the Storefront commerce-config.xml.

- commerceConfig - the main element containing all the configuration settings for an AquaLogic Commerce Services implementation.
    - web - the element containing all web-related settings.
        - web.cm.context.url - the context part of the URL for the Commerce Manager application. For example, "alcscm". Set this to an empty string if the application is in the root context.
    - search - the element containing catalog search settings
        - maxReturnNumber - the maximum number of records returned in search results, such as customers and orders.
    - shipmentcarrier - the element containing the shipment carrier settings
        - carrier.all - the list of all shipment carriers, separated by commas

- productrecommendations - the element containing product recommendation settings
  - productrecommendations.numorderhistorymonths - the number of months of previous order history that is used in product recommendation calcuations
  - productrecommendations.maxrecommendations - the maximum number of recommendations that will be computed for each product

Setting either value to -1 prevents recalculation of the product recommendations. However, existing recommendations will NOT be removed as a result of setting the value to -1.

hibernate-config.xml

See the Storefront hibernate-config.xml section for the details on configuring hibernate-config.xml for both the Storefront and Commerce Manager.

quartz.xml

The quartz.xml file is used to configure regularly scheduled jobs. See the Storefront quartz.xml section for a description of the basic structure of a quartz.xml file.

Commerce Manager Quartz Jobs

The Commerce Manager has three configured jobs:

- indexBuildJob – builds the lucene search index
- productRecommendationJob – recomputes product recommendations
- topSellerJob – determines the latest top selling products

Building the Lucene Search Index

- The index build process will pick up all added, modified, and deleted objects since the last build and update the index accordingly.

Since orders cannot be deleted from the system, the index build process won't pick up deleted orders.

- Commerce Manager index build scheduling should normally have a shorter lag time than the Storefront to ensure the index is as up-to-date as possible. The default setting is to start the first build 5 seconds after application server started and then build every 5 seconds.
- Indexes are created under the following subcategories of the application WEB-INF directory:

    - categoryIndex
    - productIndex
    - customerIndex
    - orderIndex

- In a development environment, you can disable the index build by setting the following flag to true in the env.config file as follows.

```
ep.disable.index.build=true
```

Disabling the index build at this level will disable indexes in the Storefront as well.

Computing Product Recommendations

A trigger is declared for computing product recommendation data. A SimpleTriggerBean is typically used to compute recommendations every 30 seconds for demonstration purposes. For production, a CronTriggerBean should be used to compute recommendations with a longer interval such as one day.

Computing the Top Sellers

A trigger is declared for computing the top selling product statistics. Like the product recommendation job, A SimpleTriggerBean is typically used to compute recommendations every 30 seconds for demonstration purposes. For production, a CronTriggerBean should be used to compute recommendations with a longer interval such as one day.

velocity.xml

See the Storefront velocity.xml section for the details on configuring velocity template properties for both the Storefront and Commerce Manager.

util-config.xml

See the Storefront util-config.xml section for the configuration options for both the Storefront and Commerce Manager.

Connect

Follow the instructions in the following sections for configuring your Connect application.

- Storefront acegi.xml – Configuring https security and adding new users / roles / permissions for web services.
- Commerce Manager commerce-config.xml – Configuring max number of results for order and customer search.
- Storefront hibernate-config.xml – Database-related configurations.

- Commerce Manager quartz.xml – Configuring the Lucene index for order and customer searches.

One Page Checkout

One Page checkout is included and turned on by default in AquaLogic Commerce Services. See the commerce-config.xml section for the One Page checkout configuration options.

PowerReviews Product Reviews (optional)

The basic steps required to setup PowerReviews product reviews are as follows:

1. Setup a merchant account with PowerReviews (see the PowerReviews website at http://www.powerreviews.com/ for details)
2. Setup a new login on your FTP server for PowerReviews to send your scheduled ZIP file with review data and the latest JavaScript code
3. Create a script to unzip the contents of the PowerReviews ZIP file from your FTP server to the <ALCS_APP_HOME>\commerceServices\template-resources\power-reviews\ directory on your AquaLogic Commerce Services Storefront server (overwriting the existing directory)
4. Configure PowerReviews settings in AquaLogic Commerce Services (see the Storefront `commerce-config.xml` section for details)

Consult your PowerReviews documentation for more details on how to setup and customize PowerReviews.

Configuring for Optimal Performance

This page lists tips and settings for configuring AquaLogic Commerce Services for optimal performance. The first section outlines general performance tuning considerations that affect both the Storefront and Commerce Manager. The following sections describe specific optimizations for the Storefront and Commerce Manger. The Advanced optimization section covers additional techniques for troubleshooting performance problems.

> This document only covers performance topics from a configuration and deployment perspective.

General

Key Configuration files and settings

Refer to the documentation of the following configuration files to ensure you have configured AquaLogic Commerce Services for a production environment. Remember to check these files in both your Storefront and your Commerce Manager deployments.

- velocity.xml - It is very important that you have configured Velocity for production according to these instructions.
- util-config.xml - It is very important that you have configured Velocity for production according to these instructions.
- ep-treecache.xml - You can tweak cache settings in this file to tune your performance (Storefront only).
- hibernate-config.xml - Check that "show SQL" is off. You can also tweak the commit batch size to tune import manager performance.
- quartz.xml - Check this file to ensure that you are not running batch jobs during peak times or during performance tests.

Performance tip
Check that you have configured Velocity for production by following the instructions in velocity.xml and especially util-config.xml.

Run your JVM in server mode

The JVM server mode will take longer to start but performs better once it is running. Use server mode for production environments and client mode for development only.

Specify the JVM heap size

By default, a JVM may only be able to use 64M or 128M memory.
You should specify the heap size of the application server JVM to make more memory accessible to it.

- -Xms specifies the initial Java heap size
- -Xmx specifies the maximum Java heap size

```
Sample setting:
-Xmx4096m -Xms4096m
```

Linux Memory Limitation

In 32bit Linux, the JVM may only be able to use up to 2GB memory.

Use a different garbage collector

In JVM version 1.5, several types of garbage collectors are provided. On a 2-CPU server, our test results show that the concurrent low pause collector gives a little better performance under stress.

Application Server

- Shorten the session timeout
  If you define a long session timeout, data stored in the session have a higher chance of being pushed up into slower segments of the JVM memory where they are expensive to garbage collect. Shorter timeouts will also increase the amount of memory available.

- Consider disabling session replication
  Session replication makes it possible to retain customer sessions when an application server is down. However, this will slow down application servers to some extent. If it's not critical to retain customer sessions, disabling replication can result in better performance.

Reverse Proxy Server

Reverse proxy servers are often used to cache static content to reduce load on application servers. AquaLogic Commerce Services implements an image renderer which can render an image in any desired size. This simplifies image management but also incurs CPU overhead each time an image is rendered. However there is a little cpu overhead for the application server to use this image renderer again and again on the same image. We recommend enabling caching for the rendered images on a reverse proxy server.

Infrastructure

Ensure that enough network bandwidth is provided between the following servers.

- Internet and the reverse proxy server
- Reverse proxy server and the application servers
- Application servers and the database server

Storefront

Use the AquaLogic Commerce Services product cache

Ensure that the product retrieval strategy in use is the SingleCachingProductRetrieveStrategy. The strategy to use is configured in the serviceSF.xml Spring configuration file. Check that all objects which use a product retrieval strategy are using the SingleCachingProductRetrieveStrategy. See ep-treecache.xml to fine tune this cache, which is based on JBoss Cache.

Advanced optimization

Monitor JVM memory usage in JVM 5.0

You can use the following tools that ship with JVM 1.5 to monitor memory usage.

```
jstat -gc JVM_PROCESS_ID MONITORING_INTERVAL
e.g. jstat -gc 1234 5000
```

How to get a thread trace

A thread trace will help you see thread status and find thread blockers.
You can use the following command in Linux to get a thread trace.

```
kill -3 JVM_PROCESS_PID
```

Load Balancing and Clustering (optional)

This section explains how to setup a cluster of WebLogic servers with Apache HTTP Server as a load balancer and rsync for file synchronization.

General Clustering Architecture

A server cluster is a group of computers that act as one server. The most common use of a cluster is to increase performance and/or availability. When setup as a load balancing cluster, the work goes through one server and then is distributed among all nodes in the cluster.

Cluster Components

The following are the typical components of a load balancing cluster.

- Load Balancer – All traffic to your website goes through this server and it distributes the work to the nodes in the cluster.
- Cluster Nodes – The servers in the cluster that each have an instance of the web application and perform all of the work.
- File Synchronization Server – Synchronizes changes to common files across all servers in the cluster.

AquaLogic Commerce Services has been architected to run on a load balanced cluster. A typical configuration for this would be as follows.

- Load Balancer – An Apache server with a load balancing module configured.
- Cluster Nodes – Multiple application servers setup in a cluster each with the AquaLogic Commerce Services Storefront deployed to it.
- Commerce Manager Server – One application server with the AquaLogic Commerce Services Commerce Manager deployed to it.
- File Synchronization Server – An rsync server setup on the Commerce Manager Server and an rsync client setup on each Cluster Node.

Clustering Your Application Server

This section provides you with the tools and information you need to setup an application server cluster with WebLogic.

WebLogic 9.2 Clustering

This section describes how to setup a WebLogic 9.2 cluster on Linux servers using the BEA WebLogic Scripting Tool (WLST). WLST is a scripting tool installed with WebLogic that allows for command-line configuration of the WebLogic server. The set of scripts provided here along with these instructions will allow you to setup a WebLogic cluster in your environment with ease.

Clustering Scripts Setup

In preparation for running the WLST scripts to create a cluster, you will need to download the WLST Clustering Scripts package at http://edocs.bea.com/alcs/docs51/pdf/wlstClusteringScripts.zip (ZIP, 7KB) and unzip it to a directory. After that, configure the environment variables in all of the scripts to match your environment. A description of the variables that will need to be set can be found in the section Environment Variables to Configure in the WLST Scripts. After that, copy the files to the servers that you will be using in the cluster. You will then need to navigate to this directory on a server to execute any of the scripts on that server.

Steps to Create the WebLogic Domain and Setup the Cluster

On the Server hosting the AdminServer:

1	Create a domain with AdminServer and a cluster of two managed servers	WL_HOME/common/bin/wlst.sh createcluster.py DOMAIN_NAME CLUSTER_NAME
2	Start the node manager and AdminServer	WL_HOME/common/bin/wlst.sh startadminserver.py DOMAIN_NAME
3	Setup JDBC Data Source	WL_HOME/common/bin/wlst.sh createjdbc.py t3://AdminServerIP:AdminServerHttpPort CLUSTER_NAME
4	Pack the domain	WL_HOME/common/bin/pack.sh -managed=true -domain=DOMAIN_PATH -template=DOMAIN_TEMPLATE -template_name=DOMAIN_TEMPLATE_NAME

On all servers not hosting the AdminServer:

5	Unpack the domain	Copy DOMAIN_TEMPLATE from the server hosting the AdminServer to the same directory on all other servers that are a part of the cluster and then execute the following command on the servers to create the base domain: WL_HOME/common/bin/unpack.sh -domain=DOMAIN_PATH -template=DOMAIN_TEMPLATE
6	Enroll the node manager with AdminServer and then start it	Execute the following command on the servers to setup the node managers and start them: WL_HOME/common/bin/wlst.sh enrollnodemanager.py t3://AdminServerIP:AdminServerHttpPort DOMAIN_NAME

On the Server hosting the AdminServer:

7	Deploy the AquaLogic Commerce Services code to the cluster	WL_HOME/common/bin/wlst.sh deploy.py t3://AdminServerIP:AdminServerHttpPort DEPLOYMENT_NAME APPLICATION_PATH CLUSTER_NAME

8	Start the cluster	WL_HOME/common/bin/wlst.sh startcluster.py t3://AdminServerIP:AdminServerHttpPort CLUSTER_NAME

Additional Useful Scripts

To create another server in the cluster, execute this command after step 1	WL_HOME/common/bin/wlst.sh createmanagedserver.py DOMAIN_NAME CLUSTER_NAME
To shut down server ServerName	WL_HOME/common/bin/wlst.sh stopserver.py DOMAIN_NAME ServerName
To remove a deployment	WL_HOME/common/bin/wlst.sh undeploy.py t3://AdminServerIP:AdminServerHttpPort DEPLOYMENT_NAME

Descriptions of Constants Used in Setup Steps

BEA_HOME	The BEA Home directory where files common to all BEA products are stored (eg. /opt/bea/)
WL_HOME	The WebLogic Server product installation directory (eg. /opt/bea/weblogic92/)
AdminServerIP	The IP address of the server that the AdminServer is on
AdminServerHttpPort	The port for the AdminServer to listen to http requests on
DOMAIN_NAME	The name of the clustered domain to be created (eg. alcsclusterdomain)
DOMAIN_PATH	BEA_HOME/user_projects/domains/DOMAIN_NAME (the path to the clustered domain)
DOMAIN_TEMPLATE	The filename of the domain template to be created or accessed (eg. BEA_HOME/user_templates/alcsclusterdomain_managed.jar)
DOMAIN_TEMPLATE_NAME	The descriptive name of the domain template to be created (eg. "ALCS Clustered Domain")
CLUSTER_NAME	The name of the cluster to create (eg. wlsCluster)
DEPLOYMENT_NAME	The name for the deployment of AquaLogic Commerce Services application code (eg. alcssf_cluster_deployment)
APPLICATION_PATH	The path to where the AquaLogic Commerce Services application has been setup

	(eg. WL_HOME/samples/commerce/commerceApp/commerceServices)

Environment Variables to Configure in the WLST Scripts

BEA_HOME	The BEA Home directory where files common to all BEA products are stored (eg. /opt/bea/)
WL_HOME	The WebLogic Server product installation directory (eg. /opt/bea/weblogic92/)
JAVA_HOME	The root directory of the Java JDK install that is used to run WebLogic (eg. /opt/j2sdk)
AdminServerIP	The IP address of the server that the AdminServer is on
AdminServerHttpPort	The port for the AdminServer to listen to http requests on
AdminServerHttpsPort	The port for the AdminServer to listen to https requests on
AdminServerPassword	The password used to connect to the AdminServer as default user weblogic
Machine1IP	The IP address of the server hosting the first managed server in the cluster
Machine1Name	The machine name of the server hosting the first managed server in the cluster
Server1HttpPort	The port for the first managed server to listen to http requests on
Server1HttpsPort	The port for the first managed server to listen to https requests on
Server1Name	The name to use for the first managed server in the cluster (eg. alcsServer1)
Machine2IP	The IP address of the server hosting the second managed server in the cluster
Machine2Name	The machine name of the server hosting the second managed server in the cluster
Server2HttpPort	The port for the second managed server to listen to http requests on
Server2HttpsPort	The port for the second managed server to listen to https requests on
Server2Name	The name to use for the second managed server in the cluster (eg. alcsServer2)
JdbcName	The descriptive name of the JDBC data source (eg. ALCS)
JndiName	The JNDI name of the JDBC data source (eg. jdbc/alcsjndi)

Url	The JDBC connection URL (eg. jdbc:oracle:thin:@11.11.1.111:1111:alcs)
JdbcDriverName	The name of the JDBC driver (eg. oracle.jdbc.OracleDriver)
DbUserName	The username for accessing the database
DbUserPassword	The password for accessing the database

Useful Websites

Example setting up a clustered deployment on a single server with WLST	http://dev2dev.bea.com/pub/a/2006/03/wlst-clustered-deployment.html?page=1
Steps for Creating and starting a managed server on a remote machine	http://edocs.bea.com/common/docs92/pack/tasks.html#wp1068348
WLST Documentation	http://edocs.beasys.com/wls/docs92/config_scripting/
WLST Command and Variable Reference	http://e-docs.bea.com/wls/docs92/config_scripting/reference.html
WebLogic Server Mbean Reference	http://e-docs.bea.com/wls/docs92/wlsmbeanref/core/index.html

Load Balancing

Apache HTTP Server may be used along with an installed module to load balance among a cluster of application servers. This section explains how to setup Apache HTTP Server and the sub-sections give the details on how to setup the WebLogic Apache HTTP Server Plug-in module for load balancing. WebLogic provides the WebLogic Apache HTTP Server Plug-in module to be used with Apache for load balancing WebLogic servers.

Setup of the Apache HTTP Server as a Load Balancer

- Download the latest version of Apache that can be used with the module that you will be using from the Apache Downloads Page at http://httpd.apache.org/download.cgi.

If you download the source code and build it based on the instructions, don't forget to enable ssl when you do the configuration.

```
$ {APACHE_HTTP_SERVER_SRC_DIR}/configure --enable-ssl
  --enable-so -enable-mods-shared="proxy \
  proxy_http proxy_ftp proxy_connect headers cache
  disk_cache mem_cache"
  --prefix={INSTALL_FULL_PATH}
$ {APACHE_HTTP_SERVER_SRC_DIR}/make
$ {APACHE_HTTP_SERVER_SRC_DIR}/make install
```

- Setup SSL for the Apache server
    - Create an SSL key and certificate
      Refer to the article at
      http://www.samspublishing.com/articles/article.asp?p=30115&amp;seqNum=4
      for instructions on how to do this.
    - Copy the SSL key to APACHE_HOME/conf/, renaming the file to "server.key".
    - Copy the SSL certificate to APACHE_HOME/conf/, renaming it to "server.crt".
- Apache Http Server Control
  The following commands can be used to start and stop the Apache server. See the documentation on the version of Apache that you downloaded for the full details of all commands.

```
• Start with ssl enabled:
• {APACHE_HOME}/bin/apachectl startssl
•
• Stop:
• {APACHE_HOME}/bin/apachectl stop
```

Configuring Apache with WebLogic Apache Http Server Plug-in

This section explains how to setup Apache HTTP Server with WebLogic's Apache Http Server Plug-in module to load balance a cluster of WebLogic 9.2 servers. See BEA's WebLogic 9.2 Apache HTTP Server Plug-In documentation at http://e-docs.bea.com/wls/docs92/plugins/apache.html for more detailed instructions.

Load Balancing Setup with the WebLogic Apache HTTP Server Plug-in

1. Setup the Apache WebLogic clustering configuration files
    - Unzip the Apache WebLogic cluster configuration file at http://edocs.bea.com/alcs/docs51/pdf/apacheWebLogicClusterConfiguration.zip (ZIP, 2KB) to APACHE_HOME/conf/ on the Apache server.
    - Configure the values of the elements in the unzipped files weblogic.conf and ssl-weblogic.conf to match your environment. The following are descriptions of the elements in the files that will need to be configured:

APACHE_HOME	The BEA Home directory where files common to all BEA products are stored (eg. /opt/httpd.2.0.52/)
APACHE_IP	The IP address of the server hosting Apache
APACHE_MACHINE_NAME	The descriptive machine name of the server hosting Apache

APACHE_HTTP_PORT	The port that the Apache server will be listening to http requests on
APACHE_HTTPS_PORT	The port that the Apache server will be listening to https requests on
SERVER1_IP	The IP address of the server hosting the first managed server in the cluster
SERVER1_HTTP_PORT	The port that the first managed server is listening to http requests on
SERVER2_IP	The IP address of the server hosting the second managed server in the cluster
SERVER2_HTTP_PORT	The port that the second managed server is listening to http requests on
APPLICATION_CONTEXT_NAME	The name of the Elastic Path Storefront context. Apache will only handle requests with this context in it. If the context is empty or you want Apache to handle all requests against APACHE_HTTP_PORT/APACHE_HTTPS_PORT, then this line can be taken out of the files.

2. Setup Apache Http Server Plug-in Module

   - Copy the correct version of the WebLogic Apache Http Server Plug-in for the Operating System of your Apache server from the WebLogic install to APACHE_HOME/modules. See the Apache HTTP Server Plug-In documentation at http://e-docs.bea.com/wls/docs92/plugins/apache.html to determine which is the correct version to copy.
   - Comment out the following line in APACHE_HOME/conf/httpd.conf so it can be replaced with a clustered WebLogic SSL configuration file.

```
# Include conf/ssl.conf
```

   - Add the following lines to APACHE_HOME/conf/httpd.conf under the other LoadModule lines to include the weblogic clustering configurations (replacing mod_wl_20 with the correct name of the server plug-in module that you copied over).

```
LoadModule weblogic_module  modules/mod_wl_20.so
<IfModule mod_ssl.c>
    Include conf/ssl-weblogic.conf
</IfModule>
Include conf/weblogic.conf
```

File Synchronization with rsync

AquaLogic Commerce Services uses rsync to synchronize asset and template files between a Commerce Manager server and Storefront servers.

If your Commerce Manager and Storefront are located on the same server, you do not need to setup an rsync server.

If they are located on different servers, however, then you will need to setup an rsync daemon on the server hosting the Commerce Manager and rsync clients on all Storefront servers.

Download rsync

- Download the rsync software from the rsync download page at http://samba.anu.edu.au/rsync/download.html.
  The rsync server and client are bundled together. If you downloaded the source code, you can run the following commands to build it (a C compiler is required for this).

```
• {RSYNC_SRC_DIR}/configure
• {RSYNC_SRC_DIR}/make
• {RSYNC_SRC_DIR}/make install
```

Set Up the rsync Server

1. Setup the rsync daemon on the Commerce Manager server (replacing SF1_SERVER_IP and SF2_SERVER_IP with the IP addresses of the servers that you will be using as the nodes in the cluster and <ALCS_APP_HOME> with the path to the root of the AquaLogic Commerce Services code base).

```
motd file = /etc/rsyncd.motd
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsync.lock
hosts allow = SF1_SERVER_IP, SF2_SERVER_IP
exclude = .svn .svn/* *.bak

[assets]
   path = <ALCS_APP_HOME>/ assets
   comment = assets

[templates]
   path = <ALCS_APP_HOME>/commerceServicesManager/WEB-
INF/templates
   comment = view templates
```

2. Create the welcome message file /etc/rsyncd.motd.

```
Welcome to rsyncd server.
```

3. Start the rsync daemon.

```
rsync --verbose  --progress --stats --compress --
rsh=/usr/local/bin/ssh \
      --recursive --times --perms --links --delete \
      -daemon
```

Setup the rsync Client on the Storefront Servers

1. Create a new file /sbin/rsync_alcs as follows (replacing CM_SERVER_IP with the IP address of the server that will be hosting the Commerce Manager and ALCS_INSTALL_PATH with the path to the root of the AquaLogic Commerce Services code base).

```
#!/bin/bash

rsync --verbose  --progress --stats --compress --recursive --
times --perms
  --links --delete
  CM_SERVER_IP::assets/*  <ALCS_APP_HOME>/ /assets

rsync --verbose  --progress --stats --compress --recursive --
times --perms
  --links --delete
  CM_SERVER_IP::templates/*  <ALCS_APP_HOME>/commerceServices
/template-resources/templates
```

1. Execute "chmod 744 /sbin/rsync_alcs"
2. Create a new cron job /etc/cron.d/rsync as follows:

```
1.       # run rysnc every 10 minutes
2.       */10 * * * * root /sbin/rsync_alcs
```

⚠ Keep in mind that if a large number of files need to be replicated, it may take a while before they are transferred to each Storefront. If you are planning on importing a lot of products into a live production catalog, you should first upload the media files to the Commerce Manager application and let the replication finish before running the import.

References

Linux rsync Tutorial: http://everythinglinux.org/rsync/