ICEfaces

Asynchronous HTTP Server

Deployment Guide

Version 1.6

Copyright

ICEsoft Technologies, Inc.
Suite 200, 1717 10th Street NW
Calgary, Alberta, Canada
T2M 4S2

Toll Free: 1-877-263-3822 (USA and Canada)
Telephone: 1-403-663-3322
Fax: 1-403-663-3320

For additional information, please visit the ICEfaces website: **http://www.icesoft.com**

Asynchronous HTTP Server 1.6

June 2007

About this Guide

The **ICEfaces Asynchronous HTTP Server (AHS)** is a deployment technology that provides enterprise-grade scalability for asynchronous ICEfaces applications that use server-initiated rendering. This guide provides an overview of the Asynchronous HTTP Server and specific configuration details for deploying the server and configuring ICEfaces applications to use it.

For additional information, please visit the ICEfaces Community website: **http://www.icefaces.org**

Prerequisites

A complete understanding of standard ICEfaces deployment is a prerequisite for using this guide.

ICEfaces Documentation

You can find the following additional ICEfaces documentation at the ICEfaces Community website (**http://documentation.icefaces.org**):

- **ICEfaces Getting Started Guide** — Includes information to help you configure your environment to run sample applications and a tutorial designed to help you get started as quickly as possible using ICEfaces technology.
- **ICEfaces Developer's Guide** — Includes materials targeted for ICEfaces application developers and includes an in-depth discussion of the ICEfaces architecture and key concepts, as well as reference material related to markup, APIs, components, and configuration.
- **ICEfaces Release Notes** — Read the ICEfaces Release Notes to learn about the new features included in the ICEfaces release.
- **Asynchronous HTTP Server Release Notes** — Read the Asynchronous HTTP Server Release Notes to learn about the new features included with this release.

ICEfaces AHS Technical Support

To obtain support for ICEfaces products, visit our Technical Support page at:

http://support.icefaces.org/

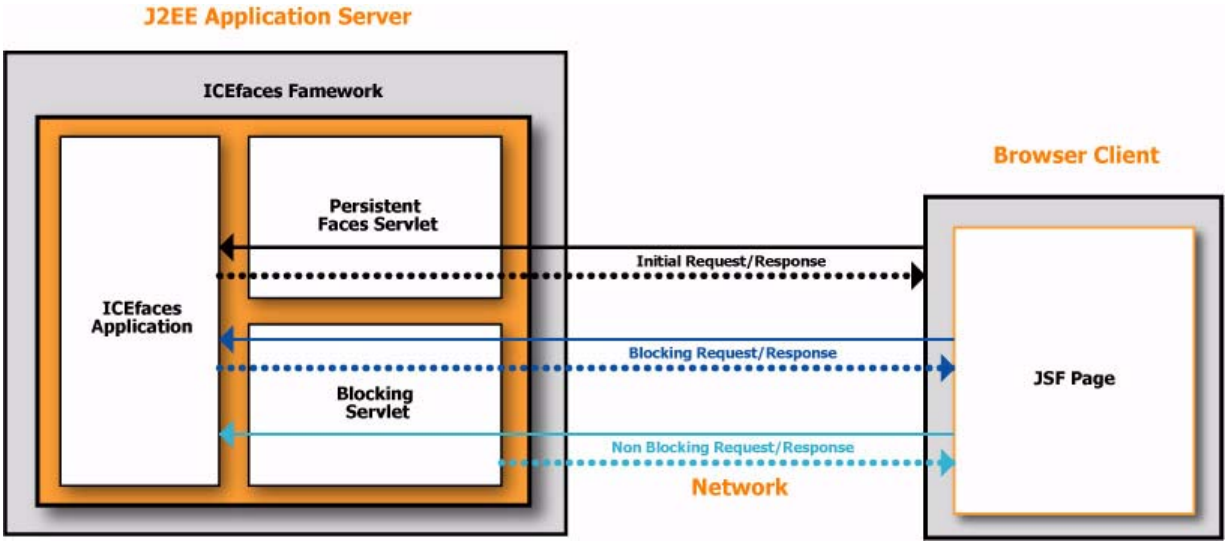
Contents

Chapter 1 Introduction to the Asynchronous HTTP Server

The Asynchronous HTTP Server (AHS) is an HTTP server capable of handling long-lived asynchronous XMLHttpRequests in a scalable fashion. In the standard Servlet model, each outstanding asynchronous XMLHttpRequest occupies its own thread, which means that thread requirements can grow linearly with the number of clients. The long-livedness of these requests will have significant impact on thread-level scalability of the application. If server-initiated rendering occurs frequently, thread consumption will be somewhat mitigated, but if server-initiated rendering occurs infrequently, large numbers of users will cause excessive thread consumption, and the scalability of the application will be compromised. It is strongly recommended that the Asynchronous HTTP Server be used in all ICEfaces application deployments where asynchronous update mode is used.

Before discussing the architecture of an Asynchronous HTTP Server deployment, it is useful to review the communication pattern that ICEfaces uses in Asynchronous mode. Asynchronous updates are primed with a blocking request from the client. When a render occurs in the JSF lifecycle, a response is produced and the blocking request is satisfied. When the response is received at the client, another blocking request is generated. ICEfaces also uses non-blocking requests for user-initiated interactions via the full or partial submit mechanism. Non-blocking requests are passed to the ICEfaces application. In ICEfaces, the Blocking Servlet handles all of this communication. See Figure 1.

Figure 1 Asynchronous Mode Deployment Architecture

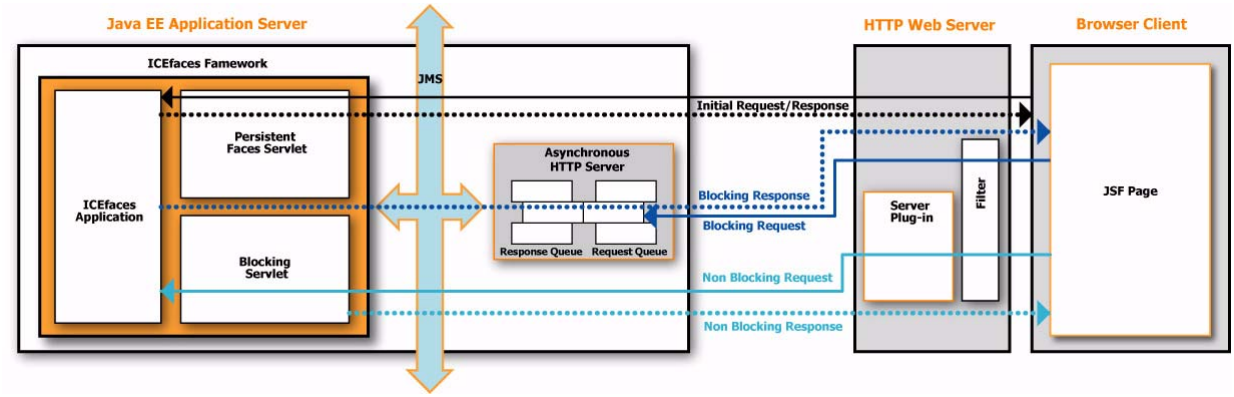


The Asynchronous HTTP Server handles thread-level scalability issues. It is designed to support single as well as clustered application server deployments with one or more ICEfaces applications deployed on it.

The Asynchronous HTTP Server is deployed as a separate application in the application server, and it occupies a single port on the network. Individual ICEfaces application deployments use Java Messaging Service (JMS) to communicate with the Asynchronous HTTP Server. Blocking requests are queued at the Asynchronous HTTP Server on a per client basis, and as responses become available, they are communicated to the Asynchronous HTTP Server, again via JMS. A response is matched to its associated blocking request, the request is unblocked, and the response is returned to the client. Because it is possible for a response to be generated prior to a request arriving, the Asynchronous HTTP Server also queues up responses waiting for requests that it can satisfy.

The deployment architecture is fronted with an HTTP web server that filters blocking request and directs them to the Asynchronous HTTP Server for processing. Non-blocking requests are passed directly to the ICEfaces application possibly via an application server-specific plug-in. The web server can also be configured to do user authentication and to use SSL for security. The basic ICEfaces enterprise deployment infrastructure, including Asynchronous HTTP Server and front-end web server, is illustrated in Figure 2.

Figure 2 ICEfaces Enterprise Deployment Architecture



Chapter 2 Deployment Configurations

This chapter describes the deployment configuration that must occur to support enterprise deployments of ICEfaces applications with the Asynchronous HTTP Server. The basic steps include:

1. **Configuring the Asynchronous HTTP Server**
2. **Configuring the ICEfaces Application**
3. **Configuring the Web Server**
4. **Configuring the Application Server**

Configuring the Asynchronous HTTP Server

Deploying the Asynchronous HTTP Server

The Asynchronous HTTP Server is a separate application running in an application server communicating with the ICEfaces applications on the back-end and the clients on the front-end. A default deployment WAR file, async-http-server.war, for the Asynchronous HTTP Server is available in the **/bin** directory. This deployment provides the following defaults:

- port used: 51315
- non-blocking mode (using Java's New I/O)
- persistent connection mode (using persistent HTTP connections)
- compression mode (using HTTP compression)
- execute queue size: 30 (the number of threads used to handle incoming requests)

Should you need to change these defaults, the configuration can be overridden by modifying the web.xml file and repackaging the WAR. The relevant attributes are shown below.

```
<servlet>
    <servlet-name>Asynchronous Servlet</servlet-name>
    ...
    <init-param>
        <param-name>com.icesoft.faces.async.server.port</param-name>
        <param-value>51315</param-value><!-- integer -->
    </init-param>
    <init-param>
        <param-name>com.icesoft.faces.async.server.blocking</param-name>
        <param-value>false</param-value><!-- boolean -->
```

```
    </init-param>
    <init-param>
        <param-name>com.icesoft.faces.async.server.persistent</param-name>
        <param-value>true</param-value><!-- boolean -->
    </init-param>
    <init-param>
        <param-name>com.icesoft.faces.async.server.compression</param-name>
        <param-value>true</param-value><!-- boolean -->
    </init-param>
    <init-param>
        <param-name>
            com.icesoft.faces.async.server.executeQueueSize
        </param-name>
        <param-value>30</param-value><!-- integer -->
    </init-param>
</servlet>
```

Note: It is critical that the port number matches the port number in the Apache HTTP Server's configuration file, which is discussed in **Routing Requests**, *p. 6*,.

Configuring the JMS Provider

Depending on the application server of choice the async-http-server.war needs to be configured to use the proper JMS Provider with the correct JMS settings. By default it uses the jboss.properties file which contains the configuration for a non-clustered JBoss deployment. The Asynchronous HTTP Server comes with a number of pre-configured property files described in the following table:

Table 2-1 ICEfaces EPS Configuration Property Files

Application Server	Configuration File	Usage
JBoss	jboss.properties	Non-clustered deployment
	jboss_ha.properties	Clustered deployment using JBoss' HA (High Availability) JMS
WebLogic	weblogic.properties	(Non-)clustered deployment
WebSphere	websphere.properties	Non-clustered deployment
	websphere_ha.properties	Clustered deployment using WebSphere's messaging via the Service Integration Bus (SIB) on a cluster

To specify the JMS Configuration properties, modify the following context parameter in the web.xml and repackage the async-http-server.war:

```
<context-param>
    <param-name>com.icesoft.util.net.messaging.properties</param-name>
    <!-- properties file of choice -->
    <param-value>weblogic.properties</param-value>
</context-param>
```

If the supplied properties files do not suffice, a custom properties file can be created and added to the classpath. A JMS Provider Configuration properties file can contain the following properties:

```
java.naming.factory.initial
java.naming.factory.url.pkgs
java.naming.provider.url

com.icesoft.util.net.messaging.jms.topicConnectionFactoryNam
com.icesoft.util.net.messaging.jms.topicNamePrefix
```

Refer to the documentation for your application server for these property values. The following example shows the property values for JBoss:

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost:1099

com.icesoft.util.net.messaging.jms.topicConnectionFactoryName=ConnectionFactory
com.icesoft.util.net.messaging.jms.topicNamePrefix=topic/
```

Configuring the ICEfaces Application

Using the Asynchronous HTTP Server with an ICEfaces application requires the following configuration changes:

1. Indicate that the Asynchronous HTTP Server is used by adding the following code to ICEfaces application's **web.xml** file:

```
<context-param>
    <param-name>com.icesoft.faces.async.server</param-name>
    <param-value>true</param-value>
</context-param>
```

2. Just as the Asynchronous HTTP Server (AHS) needs to be configured for your specific application server, an ICEfaces application needs to be configured likewise. Refer to **Configuring the JMS Provider**, *p. 4*, for details on configuring the application to use the JMS Provider.
3. Add the following JAR to the web application build (if not already present):
    - icefaces-ahs.jar

Configuring the Web Server

This section elaborates on the ICEsoft recommended configurations for running ICEfaces applications on application servers with a web server as the front-end. The Asynchronous HTTP Server is spawned inside the application server of choice. The web server is therefore responsible for filtering the incoming HTTP requests and forwarding the blocking requests to the Asynchronous HTTP Server and all other

requests to the application server. A configuration such as this is mandatory to have full support for all mainstream Internet browsers. Refer to Figure 2 on page 2.

> **Note:** In the code examples throughout this section, we use boldface text for variables that should be replaced as required.

Apache HTTP Server

Depending on the deployment, either Apache HTTP Server 2.0.x or 2.2.x can be used as the front-end. When the Asynchronous HTTP Server is to be deployed to a single application server or to a single node in a cluster of application servers, both versions of Apache can be used. However, if the Asynchronous HTTP Server is to be deployed to multiple nodes in a cluster of application servers, Apache 2.2.x is to be used.

Routing Requests

To route all blocking requests to the Asynchronous HTTP Server and all other requests to the application server, add the following code to the end of the Apache configuration file.

> **Note:** The VirtualHost container is not mandatory for Apache 2.0.x; when omitted, its directives, such as ServerAdmin, need to be omitted as well.

```
<VirtualHost _default_>
   ServerAdmin webmaster@host.example.com
   DocumentRoot /var/www/html/host.example.com
   ServerName host.example.com
   ErrorLog logs/host.example.com-error_log
   TransferLog logs/host.example.com-access_log

   <IfModule mod_proxy.c>
     ProxyRequests Off

     <Proxy *>
        Order deny,allow
        Allow from all
     </Proxy>

     # The following two directives will route all blocking requests to the
     # Asynchronous HTTP Server (identified by host:port).
     ProxyPass  /application-name/block/receive-updated-views
                http://host:port/application-name/block/receive-updated-views

     ProxyPassReverse /application-name/block/receive-updated-views
                      http://host:port/application-name/block/receive-updated-views

     # The following two directives will route all other requests to the
     # application server (identified by host:port).
     ProxyPass     /application-name
```

```
                    http://host:port/application-name
    ProxyPassReverse /application-name
                     http://host:port/application-name
  </IfModule>
</VirtualHost>
```

> **Note:** The previous ProxyPass and ProxyPassReverse directives must appear on a single line in your configuration file.

If multiple ICEfaces applications are deployed behind the Asynchronous HTTP Server, multiple ProxyPass and ProxyPassReverse directives should be included, two sets for each application. If an Apache HTTP Server plug-in is used, which is discussed in **Apache HTTP Server**, *p. 6*,, the last ProxyPass and ProxyPassReverse set can be omitted.

> **Note:** It is critical that the port number, mentioned in the first ProxyPass and ProxyPassReverse set, matches the port number in the Asynchronous HTTP Server's configuration file, which is discussed in **Deploying the Asynchronous HTTP Server**, *p. 3*,. Evidently the port number, mentioned in the second ProxyPass and ProxyPassReverse set, should match the port number of the application server.

To have proxy support in the Apache HTTP Server, the mod_proxy and mod_proxy_http modules are required. In the Apache configuration file where all modules are being loaded, add the following if not already added:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

For more information on how to load modules into Apache HTTP Server, refer to Apache Module mod_so which can be found at:

http://httpd.apache.org/docs/2.0/mod/mod_so.html (Apache 2.0.x)

http://httpd.apache.org/docs/2.2/mod/mod_so.html (Apache 2.2.x)

Security Considerations

Authentication

To enforce authentication of the user when accessing an ICEfaces application, add the following code to the Apache configuration file before the added IfModule container:

```
<LocationMatch ^/application-name>
    AuthName "ICEfaces Member-Only Access"
    AuthType Basic
    AuthUserFile /var/www/secrets/.members
    require valid-user
</LocationMatch>
```

The <LocationMatch regex> container determines that every location (Request-URI), which begins with (^) the literal string "/**application-name**" where the **application-name** is the name of the ICEfaces

application, is part of the ICEfaces Member-Only Access realm, and therefore, requires basic authentication for every user.

Secure Sockets Layer (SSL)

1. To enforce the usage of SSL when accessing an ICEfaces application, add the following code to the Apache configuration file before the added LocationMatch container:

   ```
   RewriteEngine On
   RewriteCond %{SERVER_PORT} ^80$
   RewriteCond %{REQUEST_URI} ^/application-name
   RewriteRule ^/(.*) https://%{HTTP_HOST}/$1 [R=301,L]
   ```

   The RewriteCond directives define the following conditions:

   - if the port address of the server (%{SERVER_PORT}) begins with (^) and ends with ($), thus exactly matches the literal string "80"; and
   - if the Request-URI (%{REQUEST_URI}) begins with (^) the literal string "/**application-name**", where the **application-name** is the name of the ICEfaces application.

   If both of the previous conditions are met, the RewriteRule directive defines how the Request-URI is rewritten to use HTTPS as follows:

   - $1 corresponds to the string found in the set of parentheses (that is, everything after the root (/)); and
   - rewrite the Request-URI to https://**%{HTTP_HOST}**/$1, where **%{HTTP_HOST}** is the server's host name.

   After rewriting the Request-URI, force an external redirect (301 Moved Permanently) to the client (R=301).

   Finally, tell the rewrite engine to end rule processing immediately (L), so that no other rules are applied to the last substituted Request-URI.

2. In order to have rewrite engine support, the mod_rewrite module is required. In the Apache configuration file where all modules are being loaded, add the following if not already added:

   ```
   LoadModule rewrite_module modules/mod_rewrite.so
   ```

Configuring the Application Server

When deploying applications using the Asynchronous HTTP Server, a J2EE compliant application server is required with support for JMS. The Asynchronous HTTP Server and the ICEfaces applications communicate over multiple JMS topics. It is necessary to configure the following JMS topics:

- icefaces.contextEventTopic
- icefaces.responseTopic

Application server-specific configurations are provided below.

JBoss 4.0

Configuring JMS for JBoss 4.0

When JBoss is started, three different configurations can be specified: default, all, or minimal. The minimal configuration is not sufficient for Asynchronous HTTP Server deployments, as it does not include JMS. The *default* configuration can be used in single node deployments, but the *all* configuration is required for clustered deployments. Table 2-1 below contains the location of the directory containing the jbossmq-destinations-service.xml file for each valid JBoss configuration.

Table 2-2 JBoss Configuration Files

Configuration	Directory
default	[jboss-install-dir]/server/default/deploy/jms/
all	[jboss-install-dir]/server/all/deploy-hasingleton/jms/

Note: All information on JBoss 4.0 in this section is based on the compressed archive (zip or gz) and not the JBoss installer application.

To configure the JMS topics, add the following code to the selected jbossmq-destinations-service.xml file:

```
<mbean code="org.jboss.mq.server.jmx.Topic"
       name="jboss.mq.destination:service=Topic,name=icefaces.contextEventTopic">
    <depends optional-attribute-name="DestinationManager">
        jboss.mq:service=DestinationManager
    </depends>
    <depends optional-attribute-name="SecurityManager">
        jboss.mq:service=SecurityManager
    </depends>
    <attribute name="SecurityConf">
        <security>
            <role name="guest" read="true" write="true" create="true"/>
        </security>
    </attribute>
</mbean>
```

```
<mbean code="org.jboss.mq.server.jmx.Topic"
       name="jboss.mq.destination:service=Topic,name=icefaces.responseTopic">
    <depends optional-attribute-name="DestinationManager">
        jboss.mq:service=DestinationManager
    </depends>
    <depends optional-attribute-name="SecurityManager">
        jboss.mq:service=SecurityManager
    </depends>
    <attribute name="SecurityConf">
        <security>
            <role name="guest" read="true" write="true" create="true"/>
        </security>
    </attribute>
</mbean>
```

Web Server Plug-Ins

Plug-ins are modules that can be added to a web server installation and can be configured to enable interaction between the web server of choice and the application server of choice. Typically, plug-ins can be used as a load-balancer for the web server by proxying the requests to the back-end application servers, or can be used to proxy requests for dynamic content to the back-end server(s).

Apache HTTP Server 2.0.x

JBoss uses the Apache Tomcat Servlet container, which is configured with the Apache HTTP Server via the Tomcat plug-in known as **mod_jk**.

1. First, copy the **mod_jk.so** module supplied by the Apache Software Foundation into the Apache's **/module** directory. You can obtain this module from the following web site: **http://tomcat.apache.org/connectors-doc/howto/apache.html**
2. Create a new file called **mod_jk.conf** in the Apache HTTP Server's /conf directory and add the following code to it:

   ```
   LoadModule jk_module modules/mod_jk.so

   JkWorkersFile conf/workers.properties
   JkLogFile /var/log/httpd/mod_jk.log
   JkLogLevel info
   JkMount /* myworker1
   ```

3. Create a new file called **workers.properties** in the same directory and add the following to it:

   ```
   worker.list = myworker1

   worker.myworker1.port = 8009
   worker.myworker1.host = localhost
   worker.myworker1.type = ajp13
   worker.myworker1.lbfactor = 1
   ```

   The **ajp13** refers to a worker inside JBoss that listens to port 8009. This example shows both the Apache HTTP Server and JBoss running on the same machine (hence, the localhost is the hostname of myworker1), but it is recommended to run the Apache HTTP Server and JBoss instances on separate machines.

4. The **mod_jk.conf** needs to be loaded. To achieve this, add the following to Apache's configuration file right after the LoadModule directives:

```
Include conf/mod_jk.conf
```

5. Finally, the ProxyPass and ProxyPassReverse directives that route all requests other than blocking requests to the application server, need to be removed from the Apache configuration file.

For more information on how to install and configure Apache Tomcat's plug-in, refer to the *Server Configuration Reference for Apache Tomcat 5.0.x* and *Apache Tomcat Configuration Reference for Apache Tomcat 5.5.x* which can be found at:

http://tomcat.apache.org/tomcat-5.0-doc/config/ajp.html

Apache HTTP Server 2.2.x

With the new Apache 2.2.x load-balance and failover capabilities the **mod_jk** became obsolete. However, for non-clustered deployments, it is still more desirable to make use of the AJP protocol. To do this without the **mod_jk** module, the mod_proxy_ajp module is required. In the Apache configuration file where all modules are being loaded, add the following if not already added:

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
```

To utilize the AJP protocol, the ProxyPass and ProxyPassReverse directives, which route all requests other than blocking requests to the application server, need to be rewritten to the following:

```
ProxyPass        /application-name
                 ajp://host:port/application-name
ProxyPassReverse /application-name
                 ajp://host:port/application-name
```

The default port for the AJP protocol is 8009.

> **Note:** The previous ProxyPass and ProxyPass Reverse directives must appear on a single line in your configuration file.

Clustering

This section explains how to set up a clustered deployment of JBoss Application Servers. It includes steps for configuring the cluster, JMS and web server. The following discussion relates to a clean installation of JBoss, and represents a simplified process for configuring JBoss for clustered deployments of ICEfaces applications. Additional deployment-specific issues may exist. Refer to your JBoss documentation for additional information.

This example cluster consists of two servers (nodes) as shown in the table below.

IP Address	Server Name	Description
192.168.1.100	Node1	Server
192.168.1.101	Node2	Server

Configuring the Cluster

The JBoss cluster requires no special configuration.

Configuring JMS for the Cluster

The file **[jboss-install-dir]/server/all/deploy-hasingleton/jms/jbossmq-destinations-service.xml** on each node of the cluster must be modified to include these ICEfaces topics:

- icefaces.contextEventTopic
- icefaces.responseTopic

See **Configuring JMS for JBoss 4.0**, *p. 9*, for specifics on adding these two topics.

Web Server Plug-Ins

On each node, the name of the node needs to be specified in the file `[jboss install dir]/server/all/deploy/jbossweb-tomcat5x.sar/server.xml`. In this file, locate the Engine container and add a jvmRoute attribute like the following:

```
<Engine name="jboss.web" defaultHost="localhost" jvmRoute="node1">
    ...
</Engine>
```

Apache HTTP Server 2.0.x

1. The contents of the file **mod_jk.conf**, created in step 2 of the previous **Apache HTTP Server 2.0.x** section, should be changed to the following:

```
LoadModule jk_module modules/mod_jk.so

JkWorkersFile conf/workers.properties
JkLogFile /var/log/httpd/mod_jk.log
JkLogLevel info
JkMount /* loadbalancer
```

2. The contents of the file **worker.properties**, created in step 3 of the previous **Apache HTTP Server 2.0.x** section, should be changed to the following:

```
worker.list = loadbalancer

worker.node1.port = 8009
worker.node1.host = 192.168.1.100
worker.node1.type = ajp13
worker.node1.lbfactor = 1

worker.node2.port = 8009
worker.node2.host = 192.168.1.101
worker.node2.type = ajp13
worker.node2.lbfactor = 1

worker.loadbalancer.type = lb
worker.loadbalancer.balance_workers = node1, node2
worker.loadbalancer.sticky_session = 1
```

Note: With Apache HTTP Server 2.0.x as the front-end, a clustered deployment of the Asynchronous HTTP Server is not supported. However, when the Asynchronous HTTP Server is deployed to one node in the cluster, this version of Apache can be used. All ICEfaces applications can still be deployed to the cluster on all nodes.

Apache HTTP Server 2.2.x

Apache 2.2.x comes with its own support for load balancing and failover. To utilize the load balancing features, the following steps need to be taken.

1. The ProxyPass and ProxyPassReverse directives for all requests, including the blocking requests, need to be rewritten to the following:

```
# The following two directives will route all blocking requests to the
# Asynchronous HTTP Server.
ProxyPass        /application-name/block/receive-updated-views
                 balancer://application-name-async-http-server-cluster
                 lbmethod=bytraffic nofailover=Off
ProxyPassReverse /application-name/block/receive-updated-views
                 balancer://application-name-async-http-server-cluster

# The following two directives will route all other requests to the
# application server.
ProxyPass        /application-name
                 balancer://application-name-application-server-cluster
                 stickysession=JSESSIONID lbmethod=bytraffic nofailover=On
ProxyPassReverse /application-name
                 balancer://application-name-application-server-cluster
```

Note: The previous ProxyPass and ProxyPass Reverse directives must appear on a single line in your configuration file.

2. Additionally the balance members for each balancer need to be specified right after the ProxyPass and ProxyPassReverse directives as follows:

```
<Proxy balancer://application-name-application-server-cluster>
    BalancerMember ajp://192.168.1.100:8009/application-name route=node1
    BalancerMember ajp://192.168.1.101:8009/application-name route=node2
</Proxy>

<Proxy balancer://application-name-async-http-server-cluster>
    BalancerMember http://192.168.1.100:51315/application-name
                                              /block/receive-updated-views
    BalancerMember http://192.168.1.101:51315/application-name
                                              /block/receive-updated-views
</Proxy>
```

3. In order to have load balancing support, the mod_proxy_balancer module is required. In the Apache configuration file where all modules are being loaded, add the following if not already added:

```
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

WebLogic Server 8.1 Service Pack 4

Configuring JMS for WebLogic Server 8.1

To configure the JMS topics, go through the following steps:

1. Using an Internet browser, go to WebLogic's console (for example http://localhost:7001/console) and login.
2. On the left panel, navigate to **Services** > **JMS** > **Connection Factories**.
3. On the right panel, select **Configure a new JMS Connection Factory**.
4. Enter a **Name** (for example, *ConnectionFactory*) and a **JNDI name** (for example, *ConnectionFactory*), and then click **Create**.
5. Select a server as the target and click **Apply**.
6. On the left panel, navigate to **Services** > **JMS** > **Servers** > **WSStoreForwardInternalJMSServermyserver** > **Destinations** (where myserver is the name of your server).
7. On the right panel, select **Configure a new JMS Topic**.
8. Enter *icefaces.contextEventTopic* for both the **Name** and **JNDI name**, and then click **Create**.
9. Repeat steps 6 to 8 for *icefaces.responseTopic*.

For more information on how to configure JMS on WebLogic, refer to **JMS: Configuring** which can be found at:

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jms_config.html

Web Server Plug-Ins

Plug-ins are modules that can be added to a web server installation and can be configured to enable interaction between the web server of choice and the application server of choice. Typically, plug-ins can be used as a load-balancer for the web server by proxying the requests to the back-end application servers, or can be used to proxy requests for dynamic content to the back-end server(s).

Apache HTTP Server 2.0.x

The following is a simple solution for installing and configuring WebLogic's plug-in for the Apache HTTP Server.

1. First, copy the **mod_wl_20.so** module supplied by BEA into the Apache's /**module** directory. You can obtain the **mod_wl_20.so** module from WebLogic's installation from this directory: **[bea home dir]/weblogic81/server/lib/[os]/**
2. Create a new file called **mod_wl_20.conf** in the Apache's /**conf** directory and add the following code to it:

```
LoadModule weblogic_module modules/mod_wl_20.so

<IfModule mod_weblogic.c>
    WebLogicHost host
```

```
    WebLogicPort port
</IfModule>

<LocationMatch ^/application-name(?!/block/receive-updated-views)>
    SetHandler weblogic-handler
</LocationMatch>
```

The LocationMatch container is responsible for forwarding all the requests to the ICEfaces application deployed to WebLogic with the exception of the blocking requests. To explain the regular expression used in the LocationMatch container, the following is a breakdown:

- ^/**application-name**
  if the location begins with (^) the literal string "/**application-name**", where **application-name** is the name of the ICEfaces application, and
- (?!/block/receive-updated-views)
  is not followed by (?!) the literal string "/block/receive-updated-views".

This regular expression ensures that the following possible locations get handled by the plug-in:

- /**application-name** (initial request)
- /**application-name**/ (initial request)
- /**application-name**/xmlhttp/icefaces-d2d.js (part of initial request)
- /**application-name**/block/receive-send-updates?<query> (synchronous request)
- /**application-name**/block/send-updates?<query> (UI request)

But it prevents that the following possible location gets handled by the plug-in:

- /**application-name**/block/receive-updated-views?<query> (asynchronous request)

3. The **mod_wl_20.conf** file needs to be loaded. To achieve this, add the following to Apache's configuration file right after the LoadModule directives:

   ```
   Include conf/mod_wl_20.conf
   ```

4. Finally, the ProxyPass and ProxyPassReverse directives, which route all requests other than the blocking requests to the application server, need to be removed from the Apache configuration file.

For more information on how to install and configure WebLogic's plug-ins, refer to **BEA WebLogic Server - Using Web Server Plug-Ins with WebLogic Server**, which can be found at:

http://e-docs.bea.com/wls/docs81/pdf/plugins.pdf

Apache HTTP Server 2.2.x

BEA plans to have a plug-in for Apache HTTP Server 2.2.x available with the WebLogic Server 8.1 SP 7 and 9.5 releases.

Clustering

This section explains how to set up a clustered deployment of WebLogic Servers. It includes steps for configuring the cluster, JMS and the web server. The following discussion relates to a clean installation of WebLogic Server, and represents a simplified process for configuring WebLogic for clustered

deployments of ICEfaces applications. Additional deployment-specific issues may exist. Refer to your WebLogic documentation for additional information.

This example cluster consists of two Managed Servers and an Administration Server, which itself is not part of the cluster, as shown in the table below.

IP Address	Server Name	Description
192.168.1.100	Admin	Administration Server
192.168.1.101	Managed1	Managed Server
192.168.1.102	Managed2	Managed Server

Configuring the Cluster

Ensure that you have WebLogic Server installed on each machine. Refer to your WebLogic documentation for additional information.

The following three procedures describe the steps to create a cluster formed by **Managed1_Server** and **Managed2_Server** and administered by **Admin_Server**.

Configuring the Individual Servers

1. On machine Admin go to `[bea-home-dir]/weblogic81/common/bin` and start the BEA WebLogic Configuration Wizard using:

   ```
   ./config.sh -mode=console
   ```

   Use **config.cmd** on a Windows platform.
2. Select **Create a new WebLogic configuration**.
3. Select **Basic WebLogic Server Domain**.
4. Do not run in express mode; select **No**.
5. Modify the **Name** of the Administration Server to *Admin_Server* and select **Next**.
6. Do not configure the Managed Servers, Clusters and Machines; select **No**.
7. Do not configure JDBC; select **No**.
8. Do not configure JMS; select **No**.
9. Do not configure Advanced Security; select **No**.
10. Modify the user as desired and select **Next**.
11. Select **Production Mode**.
12. Select **JRockit SDK version 1.4.2**.
13. Leave the **Target Location** at its default.
14. Modify the Name of the domain to *ICEfaces_Cluster_Domain* and select **Next**.
15. Repeat steps 1 to 14 for **Managed1_Server** on *Managed1* and **Managed2_Server** on *Managed2*.

Configuring the Cluster

1. On machine Admin go to
   `[bea-home-dir]/user_projects/domains/ICEfaces_Cluster_Domain`
   and start the server using:

   ```
   ./startWebLogic.sh
   ```

   Use **startWebLogic.cmd** on a Windows platform.
2. Using an Internet browser, go to `http://192.168.1.100:7001/console` and login as the user specified in step 10 of **Configuring the Individual Servers**.
3. On the left panel, navigate to **Clusters**.
4. On the right panel, select **Configure a new Cluster**.
5. Enter *ICEfaces_Cluster* for **Name** and then click **Create**.
6. On the left panel, navigate to **Machines**.
7. On the right panel, select **Configure a new Machine**.
8. Enter *Managed1* for **Name** and then click **Create**.
9. Repeat steps 6 to 8 for *Managed2*.
10. On the left panel, navigate to **Servers**.
11. On the right panel, select **Configure a new Server**.
12. Enter *Managed1_Server* for **Name**, select *Managed1* for **Machine** and *ICEfaces_Cluster* for **Cluster**; enter *192.168.1.101* in the **Listen Address** field, and then click **Create**.
13. Repeat steps 10 to 12 for *Managed2_Server* running on **Managed2** (192.168.1.102).

Starting the Cluster

1. On machine Managed1 go to
   `[bea-home-dir]/user_projects/domains/ICEfaces_Cluster_Domain`
   and start the server using the following command:

   ```
   ./startManagedWebLogic.sh Managed1_Server http://192.168.1.100:7001
   ```

2. Use **startManagedWebLogic.cmd** on a Windows platform.
3. Repeat step 1 for Managed2_Server.
4. Using an Internet browser, go to `http://192.168.1.100:7001/console` and login.
5. On the left panel, navigate to **Servers**.
6. On the right panel, verify that all servers (Admin_Server, Managed1_Server and Managed2_Server) have their state set as *RUNNING*.

Configuring JMS for the Cluster

The following steps describe how to create a ConnectionFactory that is available throughout the cluster and an ICEfaces_JMS_Server deployed on Managed1 servicing the JMS topics, icefaces.contextEventTopic, icefaces.renderTopic, and icefaces.responseTopic, throughout the cluster.

1. Using an Internet browser, go to `http://192.168.1.100:7001/console` and login.
2. On the left panel, navigate to **Services** > **JMS** > **Connection Factories**.
3. On the right panel, select **Configure a new JMS Connection Factory**.
4. Enter *ConnectionFactory* for **Name** and **JNDI Name,** and then click **Create**.
5. Select *All servers in the cluster* for **Targets** and then click **Apply**.
6. On the left panel, navigate to **Services** > **JMS** > **Servers**.
7. On the right panel, select **Configure a new JMS Server**.
8. Enter *ICEfaces_JMS_Server* for **Name** and then click **Create**.
9. Select *Managed1_Server* for **Target** and then click **Apply**.
10. On the left panel, navigate to **Services** > **JMS** > **Servers** > **ICEfaces_JMS_Server** > **Destinations**.
11. On the right panel, select **Configure a new JMS Topic**.
12. Enter *icefaces.contextEventTopic* for **Name** and **JNDI Name**, select *False* for **Enable Store**, and then click **Create**.
13. Repeat steps 10 to 12 for *icefaces.renderTopic* and *icefaces.responseTopic*.

Web Server Plug-Ins

Apache HTTP Server 2.0.x

The following is a simple solution for installing and configuring WebLogic's plug-in for Apache 2.0.x and a cluster of WebLogic Servers. Refer to the previous Apache HTTP Server 2.0.x section for the initial installation and configuration of WebLogic's plug-in for Apache 2.0.x.

1. The contents of the file **mod_wl_20.conf**, created in step 1 of the previous **Apache HTTP Server 2.0.x** section, should be changed to the following:

```
LoadModule weblogic_module modules/mod_wl_20.so

<IfModule mod_weblogic.c>
    WebLogicCluster 192.168.1.101:7001,192.168.1.102:7001
</IfModule>

<LocationMatch ^/application-name(?!/block/receive-updated-views)>
    SetHandler weblogic-handler
</LocationMatch>
```

Apache HTTP Server 2.2.x

BEA is planning to have a plug-in for Apache HTTP Server 2.2.x available with the releases of WebLogic Server 8.1 SP 7 and 9.5.

WebSphere Application Server 6.0.2

This section assumes that you have a working installation of WAS 6.0.2, IBM HTTP Server (IHS), and the Web Server Plug-In for WAS where you can successfully deploy web applications. For complete details on installing and configuring WebSphere Application Server (WAS), see IBM's official documentation for WAS 6.0.x at:

http://www.ibm.com/software/webservers/appserv/was/library/library60.html

There are several different JMS (Java Message Service) options available to run with WAS. The most common solution and the simplest one to configure is the Default messaging provider based on the System Integration Bus (SIB). To configure the JMS topics, follow these steps:

Login to the Admin Console

1. From your web browser, login to the WAS console (for example, http://localhost:9060/ibm/console/). By default there is no security so you should be able to login using any name.

Create the Service Integration Bus

1. In the tree on the left panel, navigate to **Service Integration** > **Buses**.
2. Click **New**.
3. Enter the information for the Bus. You will need to provide a name (e.g,. ICEfacesBus). To make the initial configuration easier, you may also want to turn off security by de-selecting the **Secure** check box. You can enable security later once you have everything running.
4. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
5. In the next panel, click **Save**. When it is done saving, click **OK**.

Add the JMS Topic Connection Factory

1. In the tree on the left panel, navigate to **Resources** > **JMS Providers** > **Default messaging**.
2. In the Default messaging provider **Configuration** panel, under **Connection Factories**, click the **JMS topic connection factory** link.
3. Click **New**.
4. Supply the following information for the connection factory:

   Name=ConnectionFactory

   JNDI name=jms/ConnectionFactory
5. Select the Service Integration Bus you created previously by choosing it from the drop-down list under **Bus** name.
6. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
7. In the next panel, click **Save**. When it is done saving, click **OK**.

Add the JMS Topics

1. In the tree on the left panel, navigate to **Resources** > **JMS Providers** > **Default messaging**.
2. In the Default messaging provider **Configuration** panel, under **Connection Factories**, click the **JMS topic** link.
3. Click **New**.
4. Supply the following information for the topic:

   Name=icefaces.contextEventTopic

   JNDI name=jms/icefaces.contextEventTopic

   Topic name=icefaces.contextEventTopic
5. Select the Service Integration Bus you created previously by choosing it from the drop-down list under **Bus** name.
6. To add one more topic, repeat steps 3 and 4 using the value "icefaces.responseTopic" for the **Name** in step 4.
7. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
8. In the next panel, click **Save**. When it is done saving, click **OK**.

JMS should now be properly configured to support ICEfaces applications running with the Asynchronous Server.

Web Server and Web Server Plug-Ins

IBM provides an option to install and use a custom version of the Apache web server, known as IBM HTTP Server (IHS). It also provides a custom Apache plug-in to support efficient communication between IHS and WAS. Although WAS can run with a standalone Apache configuration as well, the IBM HTTP Server provides tighter administrative integration with WAS. Official documentation for IHS 6.0.x can be found at:

http://www.ibm.com/software/webservers/httpservers/library/

To get ICEfaces with Asynchronous HTTP Server working with IHS and WAS requires that we include additional configuration information to IHS.

You can modify the IHS configuration in one of two ways. You can use the WAS administration application or you can modify it manually. The default location of the configuration file varies by platform so you'll need to consult the documentation to determine where it is. On Linux, the default installation location of the web server's configuration file is `/opt/IBMIHS/conf/httpd.conf`. Whether you do it manually or use the administration tool, you'll need to modify the configuration file in the following way:

1. Ensure that the rewrite and proxy modules are loaded. If they are commented out, uncomment them.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule rewrite_module modules/mod_rewrite.so
```

2. Ensure that the WebSphere plug-in module is loaded and that the plug-in configuration file is loaded from the correct location. This information will depend on what platform you are running on as well as the topology of your servers. Typically this will have been done already when you installed the plug-in for IHS.

```
LoadModule was_ap20_module /opt/IBM/WebSphere/Plugins/bin/mod_was_ap20_http.so
WebSpherePluginConfig <path-to-plugin-config>/plugin-cfg.xml
```

3. Add the following lines to engage the rewrite engine and add a rewrite rule that redirects all blocking requests to the Asynchronous HTTP Server. The [P] flag at the end tells Apache to forward the re-written request to the Proxy directive. Replace the host:port entry with the host and port of the Asynchronous HTTP Server (e.g., myhost.com:51315)

```
RewriteEngine on
RewriteRule ^/(.*)/block/receive-updated-views(.*)
                    http://host:port/block/receive-updated-views$2 [P]
```

> **Note:** The RewriteRule directive must appear on a single line in your configuration file.

4. Enable the proxy.

> **Note:** The ProxyPass and ProxyPassReverse settings outlined in the Apache HTTP Server section are not required when running with WAS.

```
<IfModule mod_proxy.c>
    ProxyRequests On
        ProxyVia Off

     <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
</IfModule>
```

Deploying ICEfaces Applications

Once WebSphere is fully configured, you can deploy your ICEfaces application. Ensure you review **Deploying the Asynchronous HTTP Server**, *p. 3*, and **Configuring the ICEfaces Application**, *p. 5*, before deploying your own application. As outlined in those sections, you will need to configure the context-param named *com.icesoft.util.net.messaging.properties* in the web.xml file of both the async-http-server.war and your application war. For running in non-clustered mode in WebSphere, the parameter should be set to *websphere.properties*.

You can deploy the Asynchronous Server (async-http-server.war) and your application separately via the Administration console. However, since we have more than one piece to deploy, it is more convenient to use an .ear file that contains all the necessary pieces of your application (which could also include other things like EJBs).

Clustering

Installing and Configuring the Cluster

You can run ICEfaces applications in a WebSphere Application Server Network Deployment (WAS ND) 6.0.2 cluster. Installing and configuring a WAS ND cluster can be somewhat daunting in that there are a number of different cluster architectures to choose from and a great deal of documentation to wade through. Given that, ICEsoft is providing this high-level summary of what we did to get our test cluster up and running. This summary is in no way intended to replace the official IBM documentation. To keep things simple, we do not include any security configuration. You'll need to adjust to suit your own security policies.

For our example, we're going to assume our cluster consists of four machines running in the following clustered architecture. You can run with fewer machines and with different configurations to suit your available resources.

IP Address	Server Name	Description
192.168.1.100	admin	Administration Server, IBM HTTP Server (IHS), and WebSphere Plug-Ins
192.168.1.101	managed1	Managed Application Server
192.168.1.102	managed2	Managed Application Server
192.168.1.103	database	DB2 Server

The following instructions assume:

- You have a copy of WebSphere Application Server Network Deployment (WAS ND) version 6.0.2. We recommend getting the latest fix pack for this version as well. This includes the IBM HTTP Server (IHS) and Plug-In installers as well.
- You have a copy of a WAS ND supported database. We used IBM DB2 Universal Database Express Edition.
- You have four WAS ND supported platforms as outlined above. We installed and ran our cluster on RedHat Enterprise 3 Linux.
- You have Root or superuser privileges on the machines and local or remote access to a command line shell.

> **Note:** In the steps that follow, node, cell, and host names must be modified as required to match your environment.

Install the Core WAS Files

1. Install the Core WAS ND files on the admin, managed1, and managed2 nodes. You can use the supplied GUI installer or shell script. Refer to IBM's documentation on running the installer of your choice. For example, on our machines we used the script and a custom response file to do the installation:

```
cd /home/icesoft/WAS-ND-V6/WAS/
./install -options /home/icesoft/WAS-ND-V6/WAS/responsefile.nd.ice.txt -silent
```

The default location for the installed files varies by platform. On Linux, the standard location for the installed files is /opt/IBM/WebSphere/AppServer.

When running on a standalone WAS server, you can take advantage of the built in database (Cloudscape) that comes with WAS. For a messaging engine to work in a cluster, you must install a separate database. This is solely a WebSphere clustering requirement as the JMS messaging used by ICEfaces does not require the messages to be persisted. Out of the box, WAS supports a number of common database configurations. To avoid problems with databases that are not officially supported, we chose DB2.

Install the Database

1. Using the documentation provided by the vendor, install and start a database instance on the database node.
2. Configure a name/password that has enough privileges to create a schema and tables. Note the **database name**, **user name**, and **password** as you will need them later.

Create the Profiles

1. On the admin server, create a Deployment Manager profile:

```
cd /opt/IBM/WebSphere/AppServer/bin

./wasprofile.sh -create -profileName deployMgrProfile -profilePath
/opt/IBM/WebSphere/AppServer/profiles/deployMgr -templatePath
/opt/IBM/WebSphere/AppServer/profileTemplates/dmgr -nodeName adminNode -cellName
mgrCell -hostName admin.com
```

2. Start the Deployment Manager:

```
cd /opt/IBM/WebSphere/AppServer/profiles/deployMgr/bin/
./startManager.sh
```

3. On the managed1 server, create a Managed Server profile:

```
cd /opt/IBM/WebSphere/AppServer/bin
./wasprofile.sh -create -profileName managedServer1 -profilePath
/opt/IBM/WebSphere/AppServer/profiles/managedServer1 -templatePath
/opt/IBM/WebSphere/AppServer/profileTemplates/managed -nodeName managedNode1 -
cellName managed1Cell -hostName managed1.com -dmgrHost admin.com
```

4. Federate and start the node:

```
cd /opt/IBM/WebSphere/AppServer/profiles/managedServer1/bin/
./addNode.sh admin.com
```

5. Create the same type of profile on the managed2 server:

```
cd /opt/IBM/WebSphere/AppServer/bin
./wasprofile.sh -create -profileName managedServer2 -profilePath
/opt/IBM/WebSphere/AppServer/profiles/managedServer2 -templatePath
/opt/IBM/WebSphere/AppServer/profileTemplates/managed -nodeName managedNode2 -
cellName managed2Cell -hostName managed2.com -dmgrHost admin.com
```

6. Federate and start the node:

```
cd /opt/IBM/WebSphere/AppServer/profiles/managedServer2/bin/
./addNode.sh admin.com
```

At this point, the Deployment Manager should be running as well as the node agents on the managed servers so you can centrally administrate the individual servers from the Deployment Manager console. Using the console, we create a cluster and add application server instances to the newly created cluster.

Login to the Admin Console

1. From your web browser, login to the WAS console (for example, http://localhost:9060/ibm/console/). By default there is no security so you should be able to log in using any name.

Create the Cluster and Server Instances

1. Create a new cluster and add managed server instances to the managed nodes:
   a. In the tree on the left panel, navigate to **Servers Clusters**.
   b. Click **New**.
   c. Enter the information for the Cluster. You will need to provide a name (e.g., IceCluster) in the **Cluster name** field. Click **Next**.
   d. On the **Create cluster members** page, enter managed1Member into the **Member name** field and select the managed1Node from the **Select node** list, and then click **Apply**. This creates a new application server instance for this cluster.
   e. Enter managed2Member into the **Member name** field and select the managed2Node from the **Select node** list, and then click **Apply**. This creates another server instance in the other node and includes it in this cluster.
   f. Click **Next**.
   g. Click **Finish**.
2. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
3. In the next panel, click **Save**.

Create a Security Alias

1. In the tree on the left panel, navigate to **Security** > **Global security**.
2. Under **Authentication**, click **JAAS Configuration** to expand the tree.
3. Click on **J2C Authentication** data.
4. Click **New**.
5. Enter an alias for the security configuration, and then supply the **user name** and **password** required to access the database.
6. Click **OK**.
7. Click the **Save** link near the top of the page.
8. Ensure that **Synchronize changes with Nodes** is checked, and then click **Save**.

9. When the changes have finished synchronizing, click **OK**.

Create a DataSource

1. In the tree on the left panel, navigate to **Resources** > **JDBC Providers**.
2. Make sure you are creating the resource in the proper scope (cluster). To set the scope of the resource definition to cluster:
   a. Click **Browse Clusters**.
   b. Click the radio button beside the correct cluster then click **OK**.
3. Click **New**.
4. For the **General Properties**, select the items that apply to your environment. For ours, we chose:

   Step 1: Select the database type -> DB2

   Step 2: Select the provider type -> DB2 Universal JDBC Driver Provider

   Step 3: Select the implementation type -> Connection pool data source
5. Click **Next**.
6. With DB2, the default settings on this page should be sufficient. If you are using another database, you may need to adjust one or more of the entries. The Class path and Native library path values can use environment variables. Ensure that the variables are properly set. They can be found by navigating to Environment -> WebSphere Variables.
7. When you have made all the necessary adjustments, click **Apply**.
8. Under **Additional Properties**, click the **Data sources** link.
9. Click **New**.
10. Adjust the **General Properties**:
    a. Under Component-managed authentication alias, choose the security alias you created in the previous section (e.g., deployMgrNode/db2Alias).
    b. Provide the **database name** that you created in the Install the Database section.
    c. Provide the host name (or IP address) of the server that the database is running on.

> **Note:** Make a note of the **JNDI name** as you will need this information later. With DB2, the default is something like jdbc/DB2 Universal Driver Provider.

11. Provide the port number on which the database is configured to accept remote connections.
12. Click **OK**.
13. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
14. In the next panel, click **Save**. When it is done saving, click **OK**.
15. Test the configuration using the **Test Connection** button.

Now we can create the bus and add the cluster to it. We'll be going through some of the same steps as in the non-clustered configuration.

Create the Service Integration Bus

1. In the tree on the left panel, navigate to **Service Integration** > **Buses**.
2. Click **New**.
3. Enter the information for the Bus. You will need to provide a name (e.g., ClusterBus). To make the initial configuration easier, you may also want to turn off security by de-selecting the **Secure** check box. You can enable security later once you have everything running.
4. Click **OK**.
5. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
6. In the next panel, click **Save**. When it is done saving, click **OK**.

Add the Cluster to the Service Integration Bus (SIB)

1. In the tree on the left panel, navigate to **Service Integration** > **Buses**.
2. Click on the bus that you created.
3. Under the topology heading, click **Bus members**.
4. Click **Add**.
5. Select the **Cluster** radio button and choose the cluster you created from the list.
6. Enter the **JNDI name** of the data source you created earlier. For example, the one we created was:

   ```
   jdbc/DB2 Universal JDBC Driver DataSource
   ```

7. Click **Next** and then click **Finish**.
8. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
9. In the next panel, click **Save**. When it is done saving, click **OK**.

Adding the cluster to the SIB automatically creates a bus-wide messaging engine. We need to ensure that the general properties are set correctly for the messaging engine to use the database.

Adjust the Messaging Engine Properties

1. In the tree on the left panel, navigate to **Service Integration** > **Buses**.
2. Click on the bus you created (e.g., ClusterBus).
3. Under **Topology**, click **Messaging engines**.
4. Click the name of the messaging engine. This name is generated and will look like *clusterName.000-busName*.
5. Under **Additional Properties**, click the **Data store** link.
6. Set the **Authentication** alias to the security alias we created earlier.

7. Adjust the Schema value so that it matches your database configuration.
8. Ensure the **Create tables** check box is enabled.
9. Click **OK**.
10. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
11. In the next panel, click **Save**. When it is done saving, click **OK**.

Add the JMS Topic Connection Factory

1. In the tree on the left panel, navigate to **Resources** > **JMS Providers** > **Default messaging**.
2. In the Default messaging provider **Configuration** panel, under **Connection Factories**, click the **JMS topic connection factory** link.
3. Make sure you are creating the resource in the proper scope (cluster). To set the scope of the resource definition to cluster:
   a. Click **Browse Clusters**.
   b. Click the radio button beside the correct cluster then click **OK**.
4. Click **New**.
5. Supply the following information for the connection factory:

   Name=ConnectionFactory

   JNDI name=jms/ConnectionFactory
6. Select the Service Integration Bus (e.g., ClusterBus) you created previously by choosing it from the drop-down list under **Bus** name.
7. Click **OK**.
8. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
9. In the next panel, click **Save**. When it is done saving, click **OK**.

Add the JMS Topics

1. In the tree on the left panel, navigate to **Resources** > **JMS Providers** > **Default messaging**.
2. In the Default messaging provider **Configuration** panel, under **Destinations**, click the JMS topic link.
3. Click **New**.
4. Supply the following information for the topic:

   Name=icefaces.contextEventTopic

   JNDI name=jms/icefaces.contextEventTopic

   Topic name=icefaces.contextEventTopic

5. Select the Service Integration Bus (e.g., ClusterBus) you created previously by choosing it from the drop-down list under **Bus** name.
6. Click **OK**.
7. To add one more topic, repeat steps 3 – 5 using the value *icefaces.responseTopic* in place of *icefaces.contextEventTopic* for all the values in step 4.
8. A **Messages** box at the top of the screen will prompt you to save your changes to the master configuration. Click the **Save** link.
9. In the next panel, click **Save**. When it is done saving, click **OK**.

JMS should now be properly configured to support ICEfaces applications running with the Asynchronous Server.

Web Server and Web Server Plug-Ins

ICEfaces supports running applications in a WebSphere cluster. However, due to the way the WebSphere plug-in works, it is not currently possible to run multiple Asynchronous Servers for load-balancing and fail-over purposes. Instead, you'll have a single active Asynchronous Server designated to handle all the asynchronous request traffic. So the configuration of the web server and the web server plug-ins is the same as it is for non-clustered WebSphere installations (see **Web Server and Web Server Plug-Ins** on page *20*).

Deploying

Once the cluster is fully configured, you can deploy your ICEfaces application. Ensure you review **Deploying the Asynchronous HTTP Server**, *p. 3*, and **Configuring the ICEfaces Application**, *p. 5*, before deploying your own application. As outlined in those sections, you'll need to configure the context-param named *com.icesoft.util.net.messaging.properties* in the web.xml file of both the async-http-server.war and your application war. For running in clustered mode in WebSphere, the parameter should be set to *websphere_ha.properties*.

Using the Administration console, deploying to a cluster is the same process as deploying to a single, standalone node.

Index