macromedia® white paper

JRun 4 Groundbreaking Clustering Architecture

by Brandon Purcell
Senior Technical Support Engineer

May 2002

Macromedia, Inc.
600 Townsend Street
San Francisco, CA 94103
415-252-2000

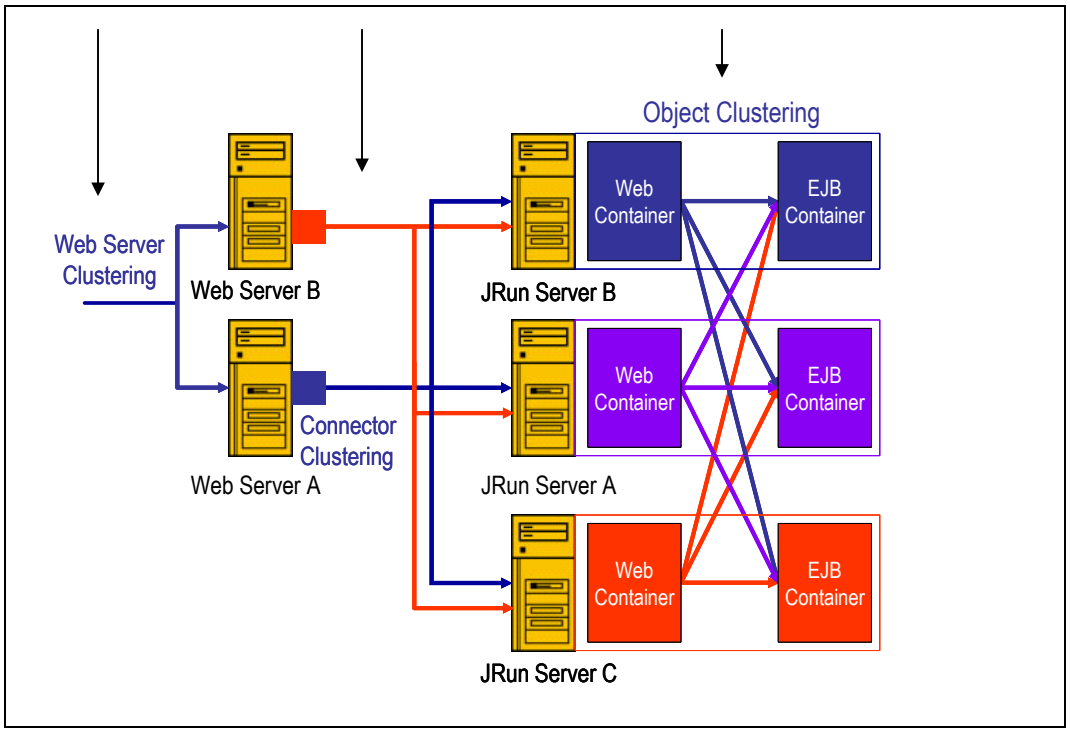
Contents

Businesses today are dependent on system reliability. This is the foundation for e-commerce sites and mission-critical business applications. High-end servers that are designed as stand-alone systems can be very reliable, but any amount of downtime can be deadly to online-only businesses. To solve these problems of reliability and redundancy, Macromedia JRun 4 provides the option of clustering. You can prevent downtime by having multiple servers work together to balance the load and provide failover.

Clustering architecture

JRun 4 includes a new state-of-the-art clustering architecture. Unlike many applications servers, cluster creation and application deployment are very simple with JRun 4. You create clusters using the JRun Management Console (JMC). After creating and adding servers to the cluster, deploying an application is easy. To deploy an application across a cluster, you add an EAR or WAR file into one of the directories for deployment; the application is instantly available across the entire cluster. Then you can connect and configure your web server directly to a cluster of JRun servers using a Swing-based wizard. JRun 4 provides an end-to-end solution for clustering so that you can build multiple levels of redundancy. It also lets you scale your applications at a lower cost by reducing hardware requirements. In the past, if you wanted to add an application server, you were required to add a web server. Now, if your application server is overloaded, you can add another one, and use the clustering functionality to provide load-balancing for the requests from your existing web server.

The following figure shows a typical clustering architecture using JRun 4:



Levels of clustering

In JRun 4, clustering is implemented at three levels to offer the maximum level of reliability, scalability, and performance. As the previous figure shows, JRun 4 provides the following three levels of clustering:

- Web server clustering
- Connector clustering
- Object clustering

The following sections describe these levels of clustering in detail.

Web server clustering

Web server clustering is a technique in which two or more web servers that support one or more domains are grouped together as a cluster of servers. The cluster collectively accommodates increases in load and provides system redundancy. Clustering for scalability works by distributing load among each server in the cluster (**load-balancing**) using an unintelligent-but-regular distribution sequence (round-robin DNS and routers), or a predefined threshold or algorithm. You specify and can adjust the distribution sequence or algorithm for each server in the cluster. With JRun 4, you cluster web servers using the ClusterCATS software- based solution, a hardware-based solution, such as a packet router (Cisco Local Director, F1 Labs Big IP), or a combination of the two. ClusterCATS integrates directly with Cisco LocalDirector by providing intelligent load information from the JRun server.

Connector clustering

A native web server connection module, or **connector**, is compiled for the specific web server, hardware architecture, and operating system. For example, JRun uses Apache modules to create the connectors for the Apache web server for each possible hardware architecture and operating system supported by JRun. The JRun web server connector is different from the Sun J2EE Java Connector Architecture (JCA) connectors, which provide integration with back-end systems. JRun ships with several different connectors that plug into the major web servers on the market (Apache, Iplanet, and IIS). The main difference between the connectors that shipped with JRun 3.x and JRun 4 is the clustering functionality. In JRun 4, you can connect multiple instances of JRun to one web server and load balance incoming requests. Out of the box, Macromedia provides three different algorithms for load-balancing (round-robin, weighted round-robin, and weighted random). As the web server receives requests, the connector directs the requests to JRun based on the algorithms. If a JRun server fails, the connector uses automatic failover to transfer requests to a running server. In-memory session replication ensures that state information is preserved when a failure occurs.

Object clustering

Groundbreaking Jini-based object clustering enables load-balancing and automatic failover of requests sent by clients (for example, JavaServer Pages (JSPs), Servlets, or regular Java clients) to server objects, such as Enterprise JavaBeans, JMS queues and topics, JNDI trees, or other services. The object's state (for example, the state of a stateful EJB) is automatically replicated in an optimized manner, provides the highest level of reliability while maintaining performance.
At server startup, clusterable services use Jini to register with each server. The use of Jini allows the clustering portion of JRun to be lightweight, have a small footprint, yet remain robust. Jini initiates the clustering process using dynamic discovery and join, and deals with events, such as service failure and new service joins. All features in JRun are "services" that plug into the JRun kernel. JRun services are "clusterable" and are monitored through a Jini lookup service. Each JRun server contains a service called the ClusterManager. This service encapsulates the Jini lookup service; so when clustering is enabled, one lookup service is running within each JRun server in a cluster. The JRun servers run in peer fashion, so there is no single ClusterManager cluster-wide, but rather a set of ClusterManagers for a single cluster (one per server); thus, there is no single administrative point of failure. As subsequent services are started (such as deployer services, transaction services, JNDI services, EJB container services, custom services created by users or ISV's) any one of those services might be "clusterable." A clusterable service is a JRun service and a Jini service. As part of the clusterable service's lifecycle, when the clusterable service starts, it dynamically discovers and joins all lookup services that are nearby (**multicast**) and, optionally, statically configures (**unicast**) services using the Jini discovery protocol.

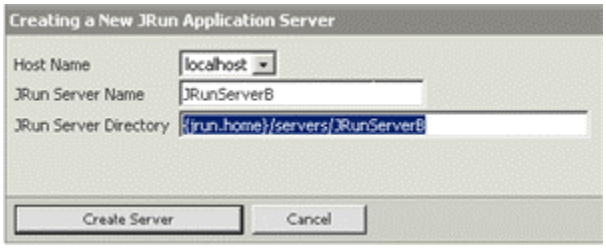
Building and deploying an application to a cluster

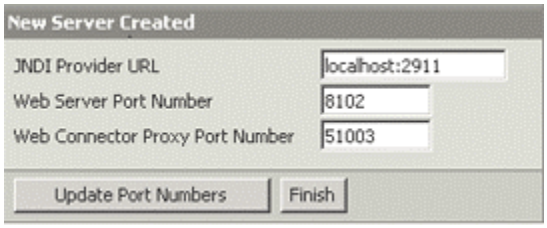
To build and deploy an application and use clustering, perform the following steps using the JMC:

1. Create any number of JRun instances..
2. Start all servers in the cluster.
3. Create a cluster and add the servers to the cluster.
4. Restart all servers in the cluster.
5. Run the Web Server Configuration tool to connect your web server to the cluster.
6. Deploy your application across the cluster by adding it to a deployable directory.

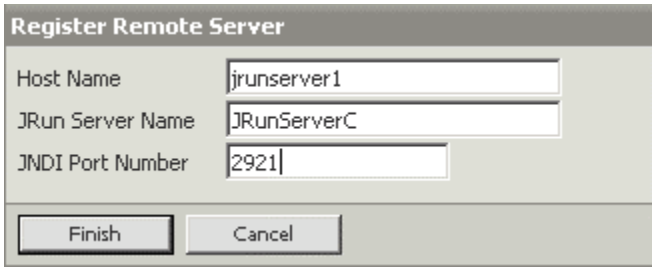
- To build and deploy an application to a cluster:

1. To create JRun instances, open the JMC, click Create New Server, and follow the instructions in the wizard:



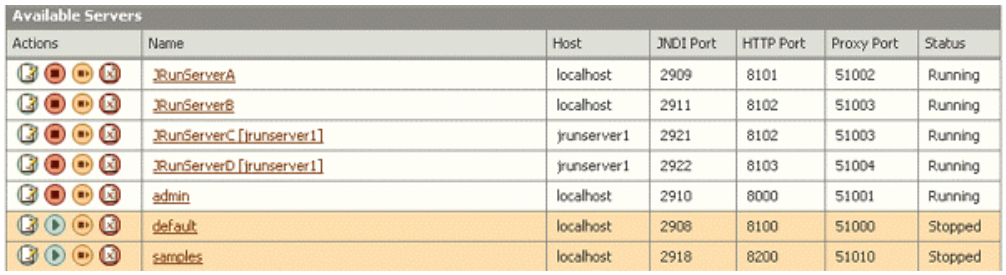


To manage remote servers from the JMC, start a server on the remote computer and register the server in the JMC.



To register the remote server, you only need the remote servers Host Name, JRun Server Name, and JNDI Port.

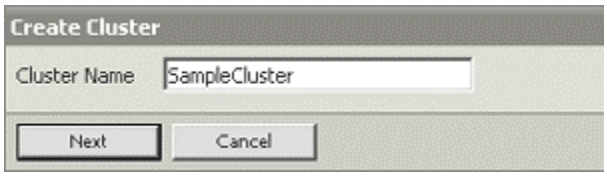
After you register a remote server, you can add/remove, start/stop/restart, and administer the setting for all servers from one JMC.

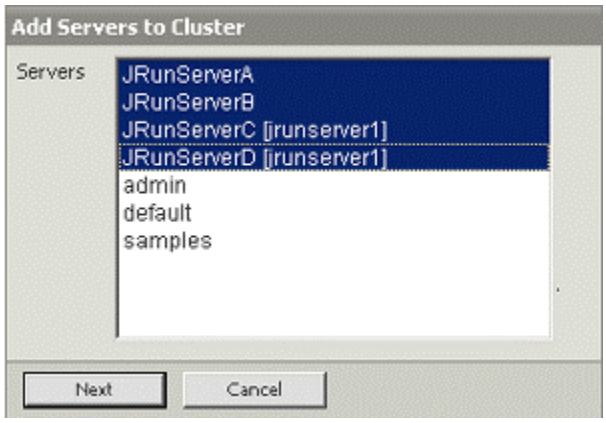


2 Before creating the cluster, start all servers that you created.

3 To create the cluster, in the JMC, click Create New Cluster and add the appropriate servers to the cluster.

Enter a name for the cluster, and choose the servers that will be a part of the cluster.





4 In the JMC, restart all servers in the cluster

5 To connect your web server to the cluster, run the Web Server Configuration tool.

You can also use the command-line tool. For more information, see the "JRun 4 Administrator's Guide

a All servers in the cluster must be running so the installer can modify the applicationropriate xml configuration files.

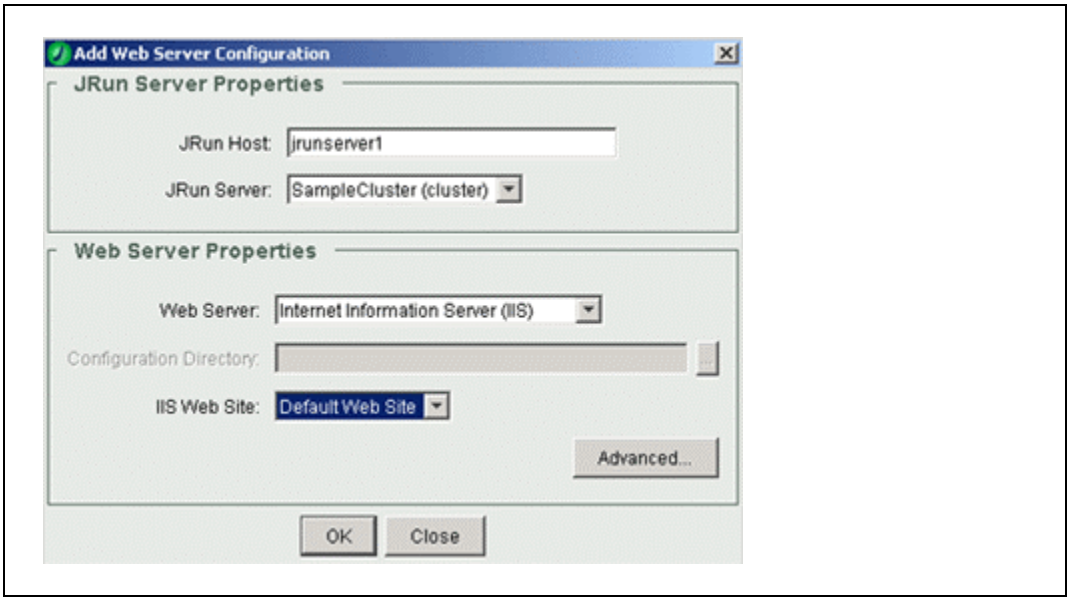
b If you have JRun servers on multiple computers, do not choose the default JRun host or localhost; instead use the host name of one of the servers in the cluster.

c Select the cluster name from the JRun Server drop-down list.

d Choose the applicationropriate web server.

e Choose the site or configuration directory.

f Click OK.
The installer makes all configuration changes and restarts the web server.



6 Deploy your application across the cluster**.**

There are two types of deployments that you can use in a cluster: cluster and local deployments:

- **Cluster deployments**. A cluster deployment lets you place an EAR or WAR file into the cluster deploy directory {servername}/SERVER-INF/cluster of any one servers in the cluster. This deploys the application across the entire cluster. When using cluster deployments, you must turn off local hot deploy for each server in the cluster. To disable hot deploy, in the JMC, click the + next to the ClusterName; a list of all servers in the cluster appears. Click one of the servers in the cluster. In the right frame, select Settings > Deployment Settings , clear the Use Hot Deploy check box, and click Submit. You must follow this procedure for each server in the cluster.
- **Local deployment**. Cluster deployment is the preferred method of deploying an application to a cluster. Local deployment can be used if there are issues with cluster deployment or in a large cluster where multiple buddy groups are formed for session replication. Using local deploy an application is placed in the {servername}/ directory of every server in the cluster. For example if you had a cluster with three servers (server1, server2, server3) with an application myapp.war you would need to copy the myapp.war file into the following locations:
  {jrun.rootdir}/servers/server1/myapp.war
  {jrun.rootdir}/servers/server2/myapp.war
  {jrun.rootdir}/servers/server3/myapp.war

Implementing session failover

JRun 4.0 provides in-memory replication of session data to provide failover functionality for JSPs and servlets. In a clustered environment you may want session information to be immediately available to other servers in the cluster. This section describes the steps in configuring session replication. Session replication should only be used in cases where it is absolutely necessary for sessions to be available across the cluster. The overhead from session replication between instances of JRun can cause an overall decrease in performance.

Session replication is configured manually by modifying the jrun-web.xml or through the JMC. The jrun-web.xml is an XML based JRun configuration file that exists within a web applications WEB-INF directory. In most cases you will create this file and place it in your application to take advantage of JRun specific features. For an example of the jrun-web.xml file see {jrun4-root}/servers/default/default-ear/default-war/WEB-INF/jrun-web.xml

For web applications JRun replicates session data to one or more servers in the cluster (the buddy server). If the JRun server shuts down or the JVM unexpectedly fails, the web server connector automatically routes requests to a buddy server. To use session replication for a web application, update the session-config element in jrun-web.xml file for all web applications that use in-memory session management. You can replicate to all servers in the cluster, or create groups of servers to replicate to. If you have a small number of servers (two to four), you can choose to replicate to all servers. For more than four servers, build groups of servers to replicate. The web server connector detects which servers in each group to failover to.

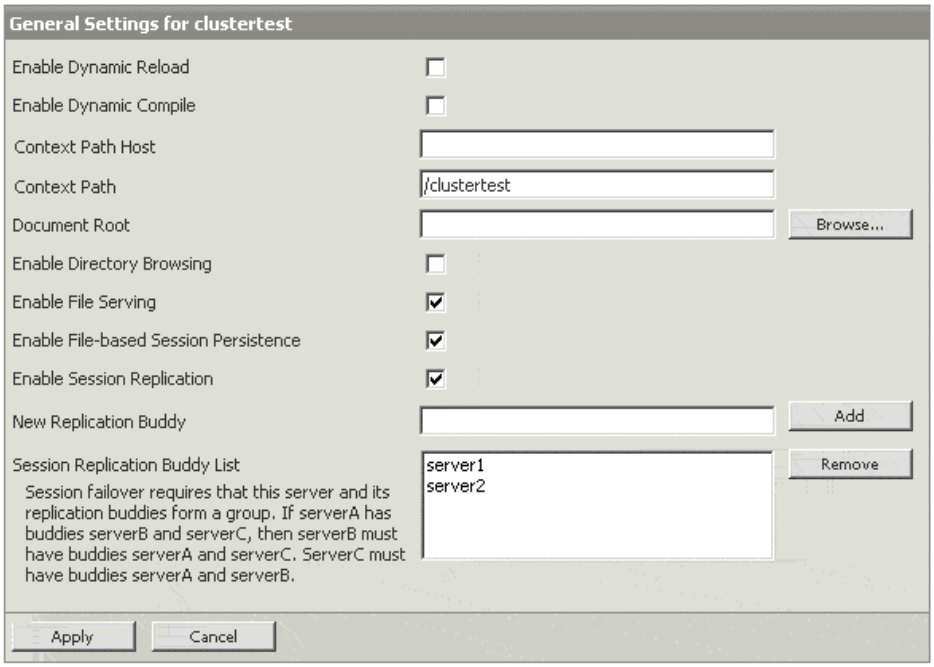
You have two options when modifying the jrun-web.xml file: you can use the wildcard asterisk (*), or specify each server in the cluster. Both options replicate all session information across the entire cluster.

```
jrun-web.xml
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>*</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>JRun1</buddy-name>
  <buddy-name>JRun2</buddy-name>
  <buddy-name>JRun3</buddy-name>
  <buddy-name>JRunX</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

In a cluster containing more than four servers multiple buddy groups should be constructed. This will reduce the amount of overhead caused by replicating between buddy servers. In the following example there are six servers in a cluster (server1,server2, server3, server4, server5, and server6). Session replication is broken down into two buddy groups (server1, server2, server3) and (server4, server5, server6). The servers in each group replicate between each other. This is accomplished by deploying the application locally to each server in the cluster, followed by adding replication buddies through the JMC for each server.

The JRun Management Console (JMC) provides access to modify the list of buddy servers for an application. To modify the buddy names through the JMC click on the + next to the cluster name then click on one of the servers in the cluster. In the right frame click on J2EE components, this will show a list of all applications that are deployed. Click on the application name or war file. The JMC will present the details of the application, similar to the following illustration:



Within the JMC modify the buddy name list, remember to modify the buddy list for each server in a replication group. For example if this were server1 you would need to add server2 and server3 to the buddy group. Then modify the buddy list for all other servers according to the following table:

Server	*Buddy Server*
server1	server2 server3
server2	server1 server3
server3	server1 server2
server4	server5 server6
server5	server4 server6
server6	server5 server6

Application considerations:

Session data must be serializable

In order to support in-memory replication for HTTP session states, all servlet and JSP session data must be serializable. If the servlet or JSP uses a combination of serializable and non-serializable objects, JRun Server does not replicate the session state of the non-serializable objects.

Use setAttribute() to change session state

Session replication occurs only at the end of a request when `setAttribute` or `removeAttribute` is called on the current session. You must be aware of this change when you make a change to an existing attribute; for example:

```
Collection c = (Collection)session.getAttribute("my.collection");
c.add("new collection value");
```

This would not be replicated. You must also call the folllowing method:

```
session.setAttribute("my.collection", c);
```

Or ensure that `setAttribute` is called on a different object. It does not have to be on the Object that you changed.

Consider serialization overhead for session objects

Serializing session data introduces some overhead for replicating the session state. The overhead increases as the size of serialized objects grows. If you plan to create very large objects in the session, first test the performance of your application to ensure that performance is acceptable.

Object clustering: EJB and JMS

Object clustering:

Jini-based object clustering enables load-balancing and automatic failover for server objects, such as Enterprise JavaBeans (EJBs), JNDI trees, or other services. The object's state (for example, the state of a stateful EJB) is automatically replicated in an optimized manner which provides the highest level of reliability while maintaining performance.

The JRun clustering architecture is controlled by the cluster manager, which runs on each JRun server for which you enabled clustering. At server startup time, clusterable services use the Jini lookup service to register with each cluster manager. in the cluster. This enables each cluster manager to discover all clusterable services in the cluster and also enables each clusterable service to discover its peers in the cluster.

JRun provides the following clusterable services:

Service	Description
JNDI Context Manager	When a client requests a clusterable object through a JNDI lookup, JRun returns a context manager that recognizes all servers in the cluster.
RMI Broker	The RMI transport implementation for all remote access in JRun.
Session Replication	When using session replication for web application, the session replication service uses clustering to synchronize state with a buddy server in the cluster.

Object clustering with EJBs:

When a client looks up an EJB through JNDI, JRun returns a context manager that knows about all servers in the cluster. This means that these resources are consistently available through the failover that is automatically in effect for clustered JRun servers. It also means that JRun automatically balances load across the cluster using different load-balancing algorithms, as follows:

- Stateless session beans: Sticky round-robin

- Stateful session bean home: Sticky round-robin
- Stateful session bean EJBObject: Buddy algorithm
- Entity bean homes: Sticky round-robin
- Entity bean EJBObject: Clustering is not supported for entity bean EJBObjects

If a JRun server goes offline during a call to an EJB method, the client stub automatically reroutes the request to another server in the cluster. In addition, object clustering provides replication for stateful session beans by automatically passivating state to the buddy server in the cluster.

JMS in a clustered environment:

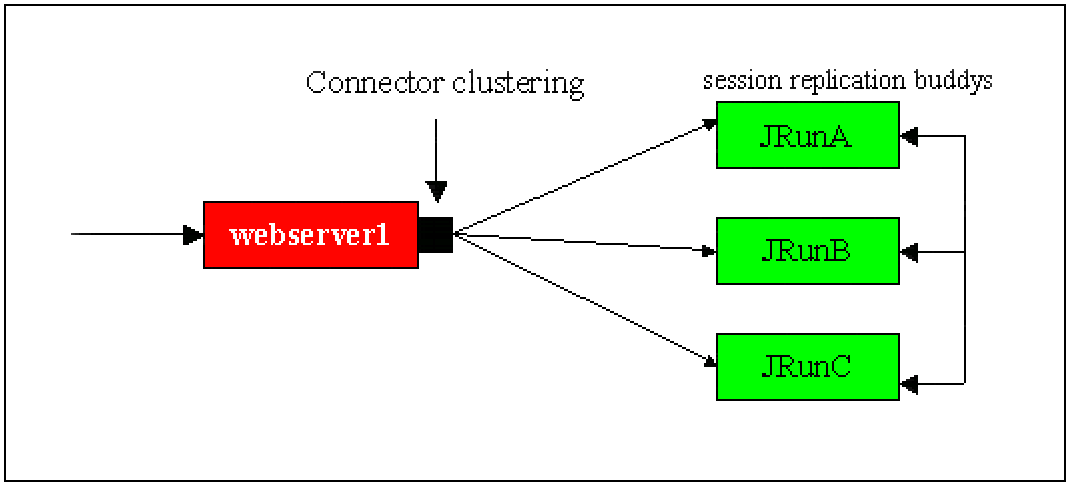
JRun provides load-balancing for JMS queues and topics for JMS in a cluster. Message-driven beans are clustered but share the same JMS backend. Macromedia recommends a third-party JMS solution for fully clustered failover capable JMS solution. For details on supported third-party solutions, see the JRun 4 release notes.

Sample configurations

This section includes three sample configurations of web servers and JRun servers, and describes how to configure the clusters.

Configuration 1: One web server -> three instances of JRun (with session failover)

This configuration provides load balancing and failover on the application server using the connector clustering. It provides session failover in the case of a JRun instance failure, but does not provide failover if the web server fails.



This configuration is the least complicated clustered configuration that you can have with JRun. It provides load balancing and failover at the application server level and allows you to scale by adding application servers. If you are looking for failover at the web server refer to configuration 3. You can use this configuration with or without session-based failover. To configure session failover modify, the jrun-web.xml file in your web application.

To configure one web server and multiple instances of JRun servers:

1 In the JMC, create two to four JRun servers.

2 In the JMC, create a cluster.

3 Run the Web Server Configuration tool to connect the web server to the cluster.

4 Add or modify the jrun-web.xml file in your webapplication.

5 Deploy the application across the cluster by adding the WAR or EAR file in {jrun-root}/servers/JrunA/SERVER-INF/cluster/ .

Note: Remember to disable hot deploy for each individual server in the cluster.

You have two options when modifying the jrun-web.xml: you can use the wildcard asterisk (*),or specify each server in the cluster. Both options replicate all session information across the entire cluster.

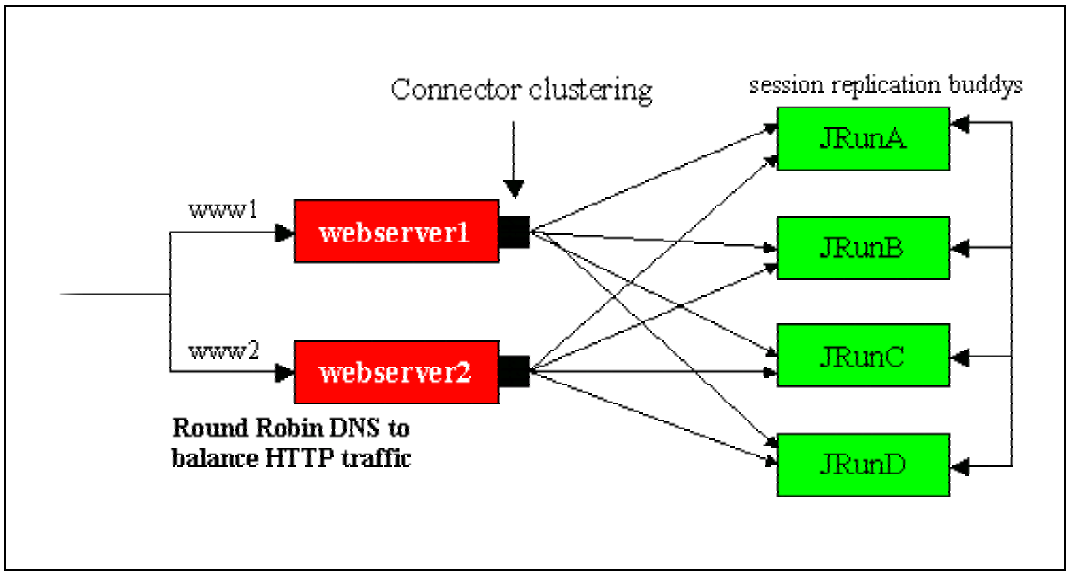
jrun-web.xml

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>*</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>JRunA</buddy-name>
  <buddy-name> JRunB </buddy-name>
  <buddy-name> JRunC </buddy-name>
 </replication-config>
 </session-config>
     </jrun-web-app>
```

Configuration 2: Round-robin DNS -> two web servers -> four instances of JRun

This configuration provides load-balancing and failover on the application server using the connector clustering. It also provides load-balancing on the web server using round-robin DNS . It does not provide failover on the web server.



This configuration allows you to scale by adding multiple web servers and JRun servers, The downside to this configuration is that it does not provide failover at the webserver if you are looking for a solution that provides failover see configuration 3.

To configure two web servers and multiple JRun server with round-robin DNS:

1. In the JMC, create two to four JRun servers.
2. Create a cluster through the JMC
3. Run the Web Server Configuration tool on each web server in the cluster.
4. Add or modify the jrun-web.xml file in your web application.
5. Deploy the application across the cluster by adding the WAR or .EAR file in {jrun-root}/servers/JrunA/SERVER-INF/cluster/

Note: Remember to disable hot deploy for each individual server in the cluster

You have two options when modifying the jrun-web.xml file: you can use the wildcard asterisk (*), or specify each server in the cluster. Both options replicate all session information across the entire cluster.

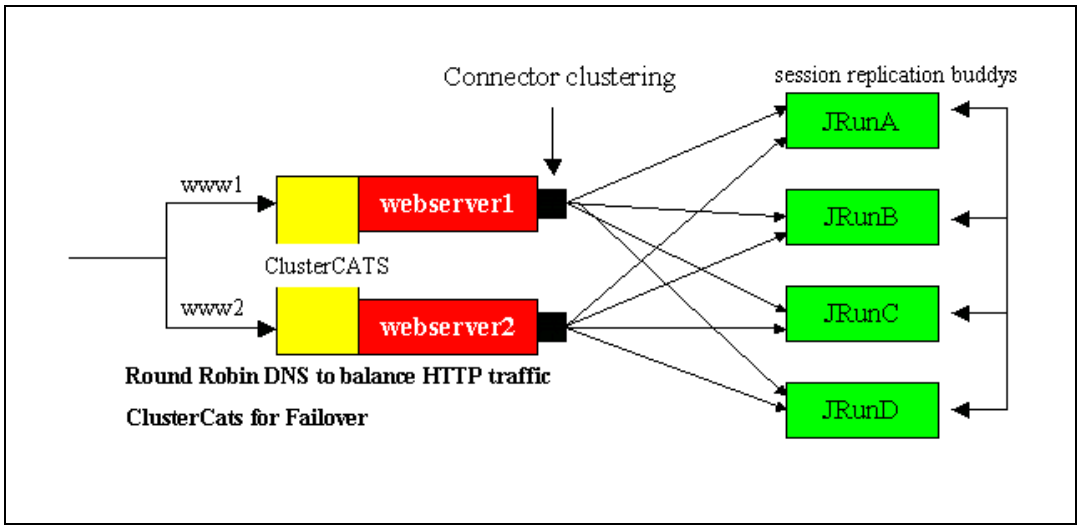
jrun-web.xml

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>*</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>JRunA</buddy-name>
  <buddy-name> JRunB</buddy-name>
  <buddy-name> JRunC</buddy-name>
  <buddy-name> JRunD</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

Configuration 3: ClusterCATS -> two web servers -> four or more instances of JRun

This configuration provides load-balancing and failover on the web server and application server. Load-balancing on the web server uses round-robin DNS. Macromedia ClusterCATS, available with JRun 4.0 as a separate download, provides failover capabilities. Load-balancing and failover on the application server uses connector clustering in JRun.



To configure two web servers and multiple JRun servers with ClusterCATS:

1. In the JMC, create two to four JRun servers .
2. In the JMC, create a cluster .
3. Run the Web Server Configuration tool on each web server in the cluster.
4. Add or modify the jrun-web.xml file in your webapplication
5. Deploy the applicationlication across the cluster by adding the WAR or EAR file in {jrun-root}/servers/JrunA/SERVER-INF/cluster/ or use local deploy.

Note: Remember to disable hot deploy for each individual server in the cluster.

6. Configure ClusterCATS.

You have two options when modifying the jrun-web.xml file: you can use the wildcard asterisk (*), or specify each server in the cluster. Both options replicate all session information across the entire cluster.

jrun-web.xml

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>*</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

```
<jrun-web-app>
 <session-config>
 <replication-config>
 <active>true</active>
  <buddy-name>JRunA</buddy-name>
  <buddy-name> JRunB</buddy-name>
  <buddy-name> JRunC</buddy-name>
  <buddy-name> JRunD</buddy-name>
 </replication-config>
 </session-config>
 </jrun-web-app>
```

Securing a cluster

High availability is not the only concern with a large-scale application. Security is also high on the list of priorities for such an application. There are several methods that you can use to secure your JRun cluster, as the following sections describe.

Set up host-based authentication

JRun provides a basic mechanism to secure the communication between the JRun server and external web servers. After you create a connection between a computer running JRun and another computer running a web server, ensure that unauthorized users cannot access the JRun server from elsewhere on the network. To do this, JRun provides host-based authentication for the JRun connector, so only hosts from a defined set of addresses can send requests to the JRun server. For information on setting this up, see the section “Host-based authentication for connectors” in “JRun Administrator’s Guide”.

Secure Sockets Layer

JRun lets you use Secure Sockets Layer (SSL) for communication between the connector and the JRun server. Typically, this is not necessary, because the web server is behind a firewall in most production configurations. However, for maximum security, you can use SSL with the web server connector. For information on setting up this security, see the section “Using SSL with the web server connector” in “JRun Administrator’s Guide”.

Configuring JNDI Access

JRun 4 provides a security.properties file that limits access to the JRun server by limiting access to JNDI. You must specify the subnet mask or the host nameand IP address of each server in the cluster.

You can control JRun JNDI access at the global or server level. You can limit JNDI access to specific hosts and subnet masks, depending on the level of security and flexibility that you require. The JRun installation includes a *jrun_root*/ lib/security.properties file that contains two properties, jrun.subnet.restriction and jrun.trusted.hosts, which control the level of access to JNDI resources for all servers in a JRun installation based on the specified subnet mask and hosts, respectively. You can override this global security.properties file for a specific server by creating a security.properties file in that server's *jrun_root*/servers/*jrun_server*/SERVER-INF directory. JRun never combines property values from global and server-specific security.properties files; when a server-specific file exists, JRun always reads properties from that file rather than the global file.

By default, the jrun.subnet.restriction property is set to the value 255.255.255.0, which limits JNDI access to the 255.255.255.0 subnet. Because no default value is set for the jrun.trusted.hosts property, it is ignored. You can use jrun.trusted.hosts on its own or in combination with jrun.subnet.restriction to allow JNDI access to a comma-separated list of IP addresses (recommended) or host names. If you are not concerned about security in a development environment, you can allow access from all subnets and hosts by specifying an asterisk (*) as the value of jrun.subnet.restriction or jrun.trusted.hosts. Do not use an asterisk in a production environment.

When working with clusters, the security.properties file must specify either the submask or IP address and host name of each server in the cluster.

If there is no security.properties file in either the *jrun_root*/lib directory or the *jrun_root*/servers/*jrun_server*/SERVER-INF directory, the JRun server will not start. When an invalid subnet mask is specified, JRun uses the default value of 255.255.255.0. A Security Alert message is printed to the JRun server event log every time a JNDI access request is rejected.

Any change to the *jrun_root*//lib/security.properties file is not reflected in the servers that use that file until you restart the servers. For the latest information on using of security.properties file, see the JRun 4 release notes.

Resources, services and server settings

JDBC data sources, security stores and JVM server settings need to be configured individually on each server in the cluster. For example if an application uses a JDBC data source, the data source will need to be configured individually for each server in the cluster. Nearly every configuration needed can be made through the JMC. The same rule applies to security stores and JVM specific settings such as heap size, classpath, and JNI library paths.

Also all JRun servers in a cluster are expected to have the same security store.ie: If the user uses the default security store which is jrun-users.xml, then the same jrun-users.xml should be present in all the nodes or the nodes should all point ot the same jrun-users.xml. This can be modified by changing the securityStore attribute of JRunUserManagerService in jrun.xml. In addition, if a user is authenticated to one node, this authentication is automatically propagated to other nodes, so that in the event of a fail over, no additional authentication is necessary.

The introduction of clustering in JRun 4 offers an easy to implement, scalable solution that provides load balancing and failover for a much lower cost than many other application servers on the market.

To download JRun 4 visit the Macromedia JRun download center. For more information on clustering see the JRun Administrator's Guide or the Macromedia JRun Support Center.