

Use Apache Geronimo to build a cluster, Part 1: Exploring manageability

Skill Level: Intermediate

[Matthew Jording \(kingrabbit@gmail.com\)](mailto:kingrabbit@gmail.com)
Freelance writer
Sensis Corporation

30 May 2006

Apache Geronimo, a Java™ Enterprise Edition (Java EE) server, is big news these days. In this five-part tutorial series, you'll explore the new Geronimo, Version 1.0 support for clustering and how its architecture assists in creating highly available and extremely scalable enterprise applications. In this installment, Part 1 of the series, investigate what a cluster is and how to build a Web service to manage one.

Section 1. Before you start

This five-part tutorial series is for the Java EE application developer, application server administrator, or service-oriented architecture (SOA) provider who needs production-quality application servers to serve up Web applications, Web services, or other SOA applications. It's also for those who want to understand the esoteric art of zero-downtime customer service and for those who want to learn the solution to unavailable and poorly scalable applications.

About this series

In this five-part series, explore Java EE clustering for reduced downtime and higher performance. Using the latest version, 1.0, of the Apache Geronimo Java EE application server, clustering for free has never been easier. In this series, you'll use the Geronimo server's newly added built-in clustering support to ensure production-quality performance from your applications. You'll also learn how to deploy to multiple machines and how to ensure load balancing.

This installment, the first part of the series, introduces you to clustering, the responsibility of the cluster nodes, and managing the nodes in an application server cluster. Because of their decoupled service-oriented nature, Web services are a natural choice for our cluster manager. The cluster manager Web service monitors and manages each node and controls the failover and load-balancing abilities of the cluster overall. The tutorial introduces the management of a cluster and gives you a hands-on approach to help you better understand the guts of application server clustering.

Part 2 of the series, "Developing the cluster nodes," will teach you how to create the cluster nodes. You'll revisit the cluster manager Web service, ensuring communication between the Manager and the nodes and testing communication among the nodes. Node intercommunication is essential to preserve the state of the applications distributed and running between them. Last, you'll create, deploy, and monitor a Hello Cluster application to test on the cluster.

Part 3, "Load balancing and failover," will demonstrate the essential parts that comprise Java EE clustering: load balancing and failover. *Load balancing* means that the cluster manager Web service knows how much of a load each node in the cluster is experiencing and balances the load accordingly. *Failover* is where a cluster node fails. The cluster manager needs to know when failover occurs to transfer that node's responsibilities to other nodes in the cluster. You'll test the capabilities of Apache Geronimo to handle these two schemes to bring your applications to a production-quality deployment.

In Part 4, "Starting and stopping the cluster," you'll add the ability to start a cluster based on its last saved state to the cluster manager Web service. You'll also get an introduction to working with Geronimo GBeans and to Geronimo's embedded database, Apache Derby. Derby will be the container with which you persist the state of the cluster. You'll test both an intentional and an *unclean* shutdown of a cluster to determine if the state of our cluster can be safely preserved.

Part 5 will bring it all together in "Collecting statistics, deployment, and testing on multiple machines," which will demonstrate how Geronimo performs in a true clustering situation by collecting and analyzing statistics about its performance. You'll revisit the Hello Cluster application to test the functionality of the cluster. Finally, you'll deploy and test the cluster as a whole on Geronimo, and you'll set up the cluster manager Web service and multiple nodes on multiple machines and test them.

About this tutorial

In clean rooms deep inside the world's Fortune 500 companies, application servers run around the clock, providing commerce, data, and support to their customers and employees. The worst thing that can happen to the Amazons, eBays, CNNs, and

other Web commerces is downtime, unavailable services, database failures, and overall poor response. Whether due to a system failure or an unusual number of users suddenly requesting services, understanding availability and scalability problems can be the difference between a company's success and failure.

When analyzing performance issues, having applications and environments that are designed for management and troubleshooting can be invaluable. In this tutorial, you'll explore how the Apache Geronimo developers have planned for manageability and how you can exploit these well-laid plans. Your first step will be learning the basics of Java enterprise clustering. Then you'll develop a Cluster Management Web service to explore clustering under the hood.

System requirements

You need to install the following tools to build the cluster manager Web service:

- [Java Platform, Standard Edition \(Java SE\) 5.0 Java Development Kit Version 5](#)
- [Apache Geronimo](#)
- [Apache Axis](#)
- [Maven 2.0](#)

Section 2. Clustering revealed

The Java Platform, Enterprise Edition (Java EE, formerly J2EE) has been a leader in enterprise applications over the past five years -- and with good reason. By calling out solutions for improving an application's availability and scalability directly in the specification and implementation, companies like IBM® and Sun Microsystems have been able to offer unprecedented uptime to their corporate customers. Java EE attacks availability and scalability problems by allowing multiple application servers to cooperate on customer requests. This cooperation is a standard solution to balancing workload and is the basis for clustering technology.

Availability and scalability

A main solution to the problem of lack of availability and scalability is through clustering architecture. Clustering achieves scalability and high availability through a

managed configuration of multiple services rather than a single point of failure. Scalability is guaranteed via duplication of process, which relieves dependence on any one resource. Related tasks are run concurrently on two or more systems without collision. To make a Java EE service highly available, nodes are made redundant. If for any reason one of the nodes fails, then another takes over the failed node's responsibilities.

Over the years, commercial Java EE application servers like IBM WebSphere® Application Server and BEA WebLogic have made strides in application server clustering and have set the standard that others, including many open source solutions, have had to live up to. The first open source Java EE-certified application server was JBoss. JBoss was also the first open source Java EE application server to provide a relatively painless clustering environment. These three application servers dominate the lion's share of production (read: clustering-enabled) enterprise Java environments, but this might soon change with the availability of Apache Geronimo 1.0, the newest Java EE-certified application server.

There are two distinct types of Java EE clustering methods: *shared disk clustering* and *shared nothing clustering*. In a shared disk cluster, both the persistence layer and the application file system are shared between application server nodes. In a shared nothing cluster, each node maintains its own disk space and persistence layer. The shared nothing cluster requires application updates to be synchronously deployed to all nodes in the cluster.

For the purposes of this tutorial, you'll be concerned only with the shared nothing cluster. This helps introduce cluster management challenges that will be covered in more detail.

Clustered tiers

The Web tier, business (Enterprise JavaBean [EJB]) tier, directory (Java Naming and Directory Interface [JNDI]) tier, and messaging (Java Messaging Service [JMS]) tier in the Java EE specification can be independently implemented within Geronimo. Currently, Geronimo uses Tomcat or Jetty for the Web tier, OpenEJB for the EJB tier, and Apache Directory for the JNDI tier.

The Web tier

The Web tier is the most widely documented clustered tier. There are well-established ways of clustering both Tomcat and Jetty. Clustering on the Web tier has two implementations: the HTTP load balancer and the HTTP session. The HTTP load-balancing solution is generally an intermediary HTTP server, such as Apache Web Server. By distributing incoming requests to each available node and queuing requests per node, the bulk of load balancing is accounted for.

The EJB tier

Clustering in the EJB tier (like the Web tier) has two implementations:

1. Clients can't invoke EJBs directly so, similar to Axis Web services, clients must invoke EJBs through stubs. Both the local stub and remote EJB have the same interface, and the job of the stub is to take local client requests and send them to the remote EJBs via the network. Stubs know how to accomplish this through Remote Method Invocation/Internet Inter-Orb Protocol (RMI/IIOP).
2. The server within message-driven beans, stateless session beans, stateful session beans, and entity beans. EJBs can run on separate machines, and can be accessed via clients using RMI/IIOP. Methods of remote EJBs can be called just like any Java object. This is because RMI-IIOP hides whether the calls are local or remote, a term known as *local/remote transparency*.

The JNDI tier

JNDI Clustering has been achieved through three different methodologies: The independent JNDI tree, The centralized JNDI tree, and the shared global JNDI tree. Each method has been used in various commercial and open source application servers to various degrees of success, the important note being that each method carries with it challenges for both administrators and developers.

A tree apiece

Most commercial Java EE application servers provide a JNDI instance. When clustered as nodes, these JNDI instances can be updated by deployed applications simultaneously. This technique, called the independent JNDI tree, ensures a degree of redundancy and replication that is desired in a clustered application. The independent JNDI tree works well for the distribution of registries for EJBs and for services commonly bound to JNDI repositories and data sources. The independent JNDI tree, however, becomes cumbersome when registering objects normally confined to the application servers' internal class loader resolution. Some vendors have gone so far as to offset this limitation with an additional Lightweight Directory Access Protocol (LDAP) repository to handle simple object registry replication.

The central alternative

For a time, many vendors attempted to work with a centralized JNDI tree for all the clustered Java EE application server nodes. All the other deployed applications would simply point to the centralized server for their registration and updates. This solves the problem of developer deployment hassles, but it introduces administrative difficulties that have all but killed the practice. Having a single JNDI repository

introduces a single point of failure that now has to be networked and configured to be highly available using several of the same clustering techniques.

Central shared JNDI

The happy middle ground that most mature Java EE installations use currently is to have each Java EE server default to its own JNDI instance: the tree apiece technique. You can then have each of these directories coordinate and synchronize their directory contents with each of the other nodes in the cluster. With the central shared JNDI tree technique, the developer deployment issues are put to rest because each Java EE server is treated as an independent JNDI tree from the application point of view. In addition, the headaches of high availability and single point of failure go away thanks to built-in synchronization.

JMS

JMS is supported in most Java EE servers, but might not be fully supported. Load balancing and failover is implemented only for JMS broker, and few products have the failover functions for messages in JMS destinations. (ActiveMQ has some strong clustering built into it. Its developers have even gone as far as to support a failover:// protocol to communicate over.) It's worth noting that Geronimo's clustering support on the messaging layer is far stronger than specified in the Java EE specification.

Section 3. Why Apache Geronimo?

With all the strong application server offerings available today, why would Apache contribute yet another offering to compete with tried and true technologies? As you read this tutorial, you'll see that Geronimo is not just another Java EE server. Take a moment now to go over some of the key advantages that Geronimo offers.

The case for Geronimo

Three major factors make Geronimo rise above other Java EE implementations when it comes to performance and scalability:

1. Geronimo is built on a lightweight framework.
2. Geronimo provides easy coupling and decoupling for Java EE components.

3. Geronimo is using some high-performance components that are centered on tackling the problem of clustering.

Inversion of Control and Dependency Injection

You've probably heard of Inversion of Control (IoC) and Dependency Injection (DI). Many developers use these terms interchangeably, but there is a subtle difference. IoC is a broad concept, while DI is an implementation of an IoC.

Literally, IoC means inverting the control within a framework by giving control to the objects that would typically be controlled. This is a normal occurrence in many architectures. For example, in an event-driven system, the object receiving the event is controlled by the object issuing the event. When the event is received, however, the receiver might act by controlling the system in some way.

Dependency Injection is a term coined by Martin Fowler (see [Resources](#) for a link). It's a pattern where rather than having objects go find their dependencies, their dependencies are injected at run time via some external source, such as a setter method. DI is inverting the control of dependencies by removing control from the referencing objects and giving it to the application framework.

Before DI, it was almost unavoidable to have classes with concrete dependencies hardcoded at some level. You could hide classes behind interfaces, but eventually some object had to know the true type of the object and create it. This would make substituting one class for another on the dependency list difficult, typically requiring code changes, which could be a significant undertaking in a deployed system.

With DI (and assuming all dependencies are abstracted behind an interface), the object never knows the true types of its dependencies. The object is just a Plain Old Java Object (POJO) and has getter and setter methods for all its dependencies (following the JavaBeans naming convention for methods), and the DI framework injects, via the constructor or setter methods, all of the object's dependencies at run time.

This process of injecting dependencies into beans at run time is commonly referred to as *wiring* the application. Wiring in a DI framework is typically accomplished via a configuration file, so when the application is run the DI framework reads in this configuration file, wires up your entire application, and you're off and running. This allows the application behavior to change from run to run without touching a line of code. Unit testing becomes a snap. No database? Fine! Just write a dummy class that implements the interface, wire it in (via the config file) as your object's dependency instead of the real database IO class, and you're done!

The Spring Framework and GBeans

Geronimo employs the Spring Framework to control DI. The Geronimo components, called Geronimo Beans (or GBeans) are wired up to each other in the `var/config/config.xml` file. A snippet of this file is shown in [Listing 1](#).

Listing 1. Config.xml

```
<configuration name="geronimo/j2ee-deployer/1.0/car">
  <gbean name="WebBuilder">
    <attribute
      name="defaultNamespace">http://geronimo.apache.org/xml/ns/
      j2ee/web/tomcat-1.0</attribute>
    </gbean>
    <gbean name="EJBBuilder">
      <attribute
        name="listener">geronimo.server:J2EEApplication=null,
        J2EEModule=geronimo/tomcat/1.0/car,
        J2EEServer=geronimo, j2eeType=GBean, name=TomcatWebContainer</attribute>
      </gbean>
    </configuration>
```

For those familiar with Spring configuration files, you'll notice that the layout for this file is a little different from your typical config file. That's because it's not a Spring config file, it's an XBean config file. More on that later. For now, just notice that the file defines the GBeans that Geronimo contains and injects dependencies via the `<attribute>` tag. The dependencies can either be values or references to other GBeans (as in the `listener` attribute of the `EJBBuilder`).

If all of this GBean and DI stuff is new, and you're not interested in learning yet another technology, don't be alarmed. The GBean/DI layer of Geronimo is transparent to most users. In typical deploys of Java EE modules, Geronimo dynamically generates GBean wrappers and plugs them in to the framework automagically without you having to know anything about the Geronimo internals.

Geronimo developers have built it from the ground up with clustering in mind. Not only is Geronimo based on the concept of lightweight frameworks and IoC, but the components that implement the various Java EE tiers surrounding the Geronimo kernel have been chosen for their decoupled and clustering strengths. Geronimo has been steadily acquiring stellar open source projects into its subproject stack. Geronimo contains several projects that provide some component of clustering (see [Resources](#) for links):

- `ActiveCluster` is a generic clustering API used by `ActiveMQ` and `WADI Application Distribution Infrastructure (WADI)`. It simplifies operations that are typical for clustering applications.
- `ActiveIO` is built on top of `ActiveCluster` and is used by `ActiveMQ` for clustering across a network of brokers.
- `ActiveSpace` is used by `ActiveMQ` to support Staged Event-Driven

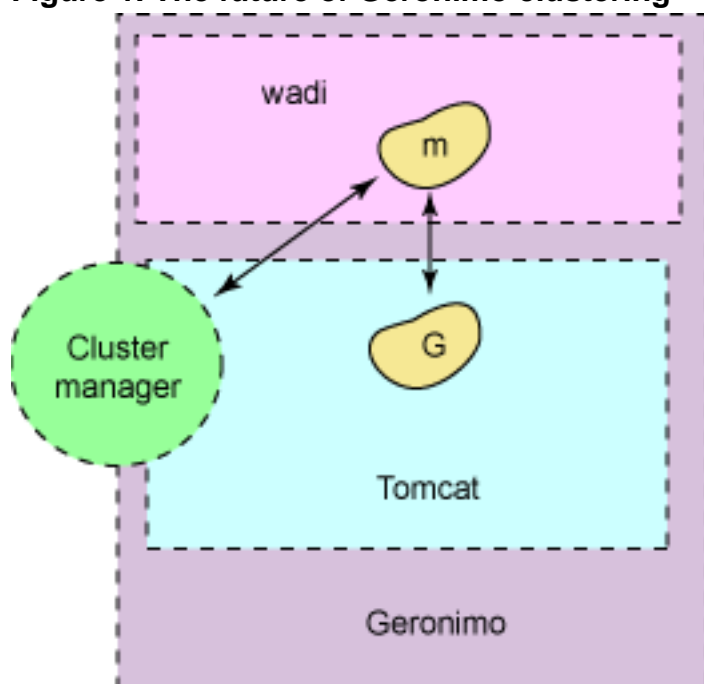
Architecture (SEDA)-style load-balancing data across consumers.

- WADI is built on top of the ActiveCluster API to provide clustering for HttpSession.

The future of cluster management

The future of cluster management looks to be WADI based. Using Geronimo GBeans in the Tomcat module to communicate node information to WADI, you can access the WADI MBean or whatever facility will be in place. In the current version of Geronimo, some of the foundation is laid, and we can take advantage of it, but the complete architecture is still under construction (see [Figure 1](#)).

Figure 1. The future of Geronimo clustering



Future cluster management within Geronimo is already receiving the attention it needs, and with the skilled Geronimo and WADI developers, it's sure to be in place for Geronimo 2.

Section 4. The cluster manager Web service

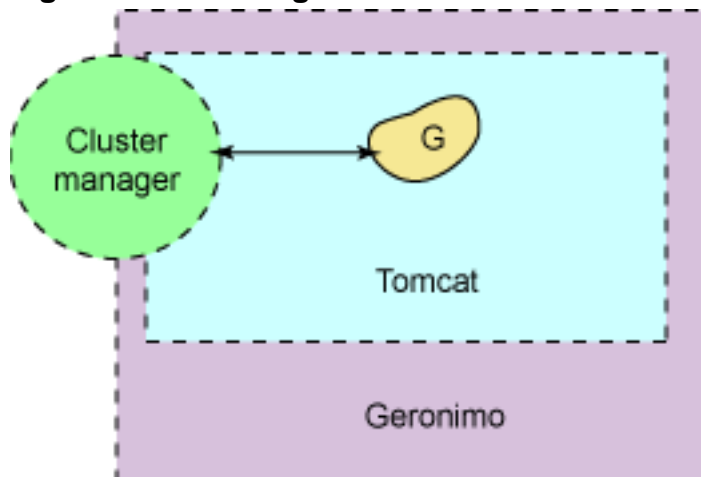
This tutorial defines the interfaces to your manager and focuses on general Web service builds and deploying to Geronimo. This section defines what our cluster

manager Web service needs to hook into Geronimo's various tiers to provide the statistics and analysis you'd like to see. The first area you'll explore is the Tomcat clustering GBeans. These are the beginnings of WADI integration, and potential integration of other management apps. You'll define the Tomcat GBeans and MBeans to gather information about your node.

Monitoring Geronimo

The cluster manager Web service you're going to create will be in charge of monitoring Geronimo and displaying clustering in action. Geronimo's performance as a clustering engine will be analyzed by collecting statistics of the cluster under certain conditions. The cluster manager will monitor each cluster node, displaying its current load and any failures or recoveries. Additionally, the cluster manager will control the startup and shutdown of each node and assign manual load changes.

Figure 2. Clustering the Web tier



To build the scaffolding for the cluster manager to hook into the various facilities of Geronimo, you need to first take a short look at Maven 2.0 and Apache Axis2.

Using Maven 2.0

Maven 2.0 is a powerful utility for Java project building, testing, and deployment. Maven uses Project Object Model files (pom.xml) to define the project dependencies, project deployment packaging, and other project metadata. You'll use some predefined project setups called Maven archetypes. These archetypes define directory layouts for Web applications and other specialized Java projects.

The choice for project build, test, and deploy today is Maven 2.0.

The following is the quick-start guide for using Maven 2.0 for the Web service:

1. Download Maven 2.0 by selecting the proper file for your machine platform (see [Resources](#) for a link).
2. Unpack the archive (for example, `tar zxvf maven-2.0.tar.gz` or `unzip maven-2.0.zip`). A directory called `maven-2.0` will be created.
3. Add the `bin` directory to your `PATH` (for example, export `PATH=/usr/local/maven-2.0/bin:$PATH` or set `PATH="c:\program files\maven-2.0\bin";%PATH%`).
4. Make sure `JAVA_HOME` is set to the location of your JDK.
5. Run `mvn --version` to verify that it's correctly installed.

Now create the Web application project by running the following command:

```
mvn archetype:create -DgroupId=com.ibm.ClusterManager
-DartifactId=GIAWebService
```

This creates a basic directory structure for a Maven project (see [Listing 2](#)).

Listing 2: Maven project directory structure

```
-GIAWebService
--- pom.xml
--- src
----- main
----- java
----- com
----- ibm
----- ClusterManager
----- test
----- java
----- com
----- ibm
----- ClusterManager
```

Apache Axis2

Apache Axis2, the component currently providing Web service support to the Geronimo kernel, makes developing a Web service mystifyingly easy. By creating an ordinary Java interface, you can generate a full WSDL description for your Web service (see [Listing 3](#)).

Listing 3. ClusterManager.java

```
package com.ibm.ClusterManager;
```

```
import java.util.Date;
import java.util.List;
public interface ClusterManager {
    public int getNodeSessions(int nodeId);
    public Date getNodeUptime(int nodeId);
    public boolean startNode(int nodeId);
    public boolean stopNode(int nodeId);
    public List getNodeFailureReport(int nodeId);
    public int getNodeLoadCapacity(int nodeId);
    public int setNodeLoadCapacity(int nodeId);
}
```

In this interface, you define methods to get the number of sessions for the node specified:

```
getNodeSessions(int nodeId)
```

From the directory containing the project folder, run the code shown in [Listing 4](#).

Listing 4. Java to wsdl axis tool

```
java org.apache.axis.wsdl.Java2WSDL
-o GIAWebService/ClusterManager.wsdl
-l "http://localhost:8080/ClusterManager/services/ClusterManager"
-n "urn:ClusterManager"
-p "com.ibm.ClusterManager" \
"urn:com.ibm.ClusterManager" com.ibm.ClusterManager.ClusterManager
```

This produces a valid WSDL named ClusterManager.wsdl derived from the interface in the project directory.

Another convenient application provided by Axis2 is the ability to generate all the stubs, locators, and bindings needed for a Web service. As you may have guessed, this ability is provided by Running WSDL2Java. If it's run with the following parameters, you should have all of the Java code predefined for your Web service:

```
java org.apache.axis.wsdl.WSDL2Java -S true
src/ClusterManager.wsdl
-Nurn:ClusterManager com.ibm.ClusterManager -o src/main/java/
```

This generates source files in the src/main/java/com/ibm/ClusterManager/ folder (see [Listing 5](#)).

Listing 5. Generated source files

```
ClusterManager.java
ClusterManagerService.java
ClusterManagerServiceLocator.java
ClusterManagerSoapBindingImpl.java
ClusterManagerSoapBindingSkeleton.java
ClusterManagerSoapBindingStub.java
deploy.wsdd
```

`undeploy.wsdd`

Next you need to implement the interfaces with Geronimo's built-in clustering services.

Implement the interfaces with Geronimo clustering services

After everything is generated, you can build and deploy the Web service. First you need to edit the project object file, called `pom.xml`, to create a WAR and deploy it to Geronimo (see [Listing 6](#)).

Listing 6. `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ibm.GIAWebService</groupId>
  <artifactId>GIAWebApp</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Webapp Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    ...
    <dependency>
      <groupId>geronimo-spec</groupId>
      <artifactId>geronimo-spec-activation</artifactId>
      <version>1.0.2-rc4</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>GIAWebService</finalName>
  </build>
</project>
```

The POM above calls out ways in which the project is organized that are important to building, testing, and deploying. Specifically the `<package>war</packaging>` section tells Maven that you're going to use the zip algorithm to create a Web ARchive (WAR) resource. Most of the file calls out dependencies to third-party libraries that Maven can find via public repositories. The `<dependency>` tag specifies the artifact, which could be a JAR, WAR, or other archive type, or another POM file to build.

From the project folder where the `pom.xml` file lives, run `mvn compile`.

To have Maven package the cluster manager and place the resulting deployable WAR in the target subdirectory of your project, run `mvn package`.

Geronimo versions

The three releases of Geronimo 1.0 available at the time of this writing are:

1. Source download, which is the source code from which to build Geronimo.
2. Geronimo built with Jetty as the servlet container.
3. Geronimo built with Tomcat as the servlet container, which is the most popular release and what you'll use in the example.

Download and install the Geronimo 1.0 release with Tomcat integrated (see [Resources](#) for a link). For Microsoft® Windows®, use `geronimo-tomcat-j2ee-1.0.zip`; for UNIX® or Mac OS X, use `geronimo-tomcat-j2ee-1.0.tar.gz`. Place the file in a memorable location on your hard drive, uncompress the archive, and navigate to the folder created, which will hereafter be referred to as `<geronimo_home>`. Apache's release notes provide clear instructions on system requirements and how to install and start Geronimo.

Start Geronimo by navigating via a command prompt to `<geronimo_home>` and executing `java -jar bin/server.jar`.

The display will fill with status information, finally letting you know that the server has started. After that's done, you can deploy from the project.

Geronimo provides one of the most intuitive administrative Web consoles. The administrative console makes it painless to create data sources, manage security, and deploy new applications. With the Geronimo server started, open a Web browser and navigate to `http://localhost:8080/console`; the initial username and password are set to `system` and `manager`, respectively (see [Figure 3](#)).

Figure 3. Geronimo console login



Log In to the Geronimo Console

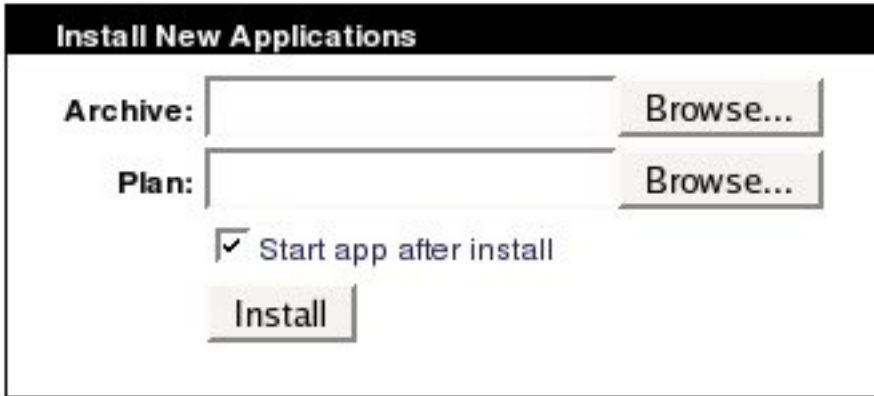
Username:

Password:

Welcome to the Geronimo™ Console

In the left panel there are folders with task buttons cascading beneath them. Under Applications, click the **Deploy New** button to bring up a page allowing you to upload your newly created .war file. Browse for the archive, and click the **Install** button (see [Figure 4](#)).

Figure 4. Geronimo console deployment



Install New Applications

Archive:

Plan:

Start app after install

That's all there is to creating a Web service and deploying it to Geronimo.

Section 5. Summary

You should now have a good understanding of what application server clustering is, how Java EE succeeds with clustering, and why Geronimo is uniquely positioned to be the Java EE application server of choice for clustered environments -- and not simply because it is free. Additionally, you should now be prepared to work with a Geronimo cluster of your own.

By defining the interfaces for the cluster manager Web service, most of the required information for the Tomcat Web tier is defined. You can see why harvesting this information from every node on a cluster can increase the manageability of an application server cluster. Throughout this series, you'll build upon this foundation and hook into the various tiers of the cluster to manage and monitor each portion of your Java EE architecture.

Stay tuned for Part 2, where you'll build the nodes that the cluster manager will monitor and manage. To follow along with Part 2, it's ideal to have more than one computer on a local network, but if you'd rather not run out and get an additional computer, you can deploy to multiple instances on the same machine. The skills involved in clustering are in high demand, so as you work with clusters, you'll not only see the advantages of clustering, but also increase your own value as a developer and administrator.

Downloads

Description	Name	Size	Download method
Cluster Manager example source file	ClusterManagerService.zip	110KB	HTTP

[Information about download methods](#)

Resources

Learn

- Get information on [Inversion of Control Containers \(IoC\) and the Dependency Injection \(DI\) pattern](#).
- Get all the latest Apache Geronimo news and opinions at "[The Geronimo renegade](#)" column here on developerWorks.
- Read a classic article on the basics of clustering, "[Clustering: a basic 101 tutorial](#)" (developerWorks, April 2002).
- Visit the [Geronimo Wiki](#).
- Get more information on "[Migrating Apache Axis Apps to Axis2 with Geronimo](#)" (developerWorks, March 2006).
- Check out more great pieces on Geronimo:
 - "[Building a better J2EE server, the open source way](#)" (developerWorks, May 2005)
 - "[Geronimo! Part 1: The J2EE 1.4 engine that could](#)" (developerWorks, May 2005)
 - "[Geronimo! Part 2: Tame this J2EE 1.4 bronco](#)" (developerWorks, May 2005)
 - "[Three ways to connect a database to a Geronimo application server](#)" (developerWorks, June 2005)
 - "[Create, deploy, and debug Apache Geronimo applications](#)" (developerWorks, May 2005)
 - "[Apache Geronimo uncovered](#)" (developerWorks, August 2005)
- Visit the [developerWorks Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Check out the developerWorks [Apache Geronimo project area](#) for articles, tutorials, and other resources to help you get started developing with Geronimo today.
- Check out the [IBM Support for Apache Geronimo](#) offering, which lets you develop Geronimo applications backed by world-class IBM support.
- Find helpful resources for beginners and experienced users at the [Get started now with Apache Geronimo](#) section of developerWorks.

- Browse all the [Apache articles](#) and [free Apache tutorials](#) available in the developerWorks Open source zone.
- Browse for books on these and other technical topics at the [Safari bookstore](#).

Get products and technologies

- Get information about [CodeHaus](#), an open source project repository with a strong emphasis on Java technology, focused on quality components that meet real-world needs.
- Check out [ActiveCluster](#), a generic clustering API used by ActiveMQ and WADI that simplifies operations that are typical for clustering applications.
- Get the scoop on [ActiveIO](#), which is built on top of ActiveCluster and is used by ActiveMQ for clustering across a network of brokers.
- Find out more about [ActiveSpace](#), used by ActiveMQ to support SEDA-style load-balancing data across consumers.
- Explore [WADI](#), a geeky recursive acronym that stands for WADI Application Distribution Infrastructure. WADI is built on top of the ActiveCluster API to provide clustering for HttpSessions.
- Download [Maven 2.0](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.
- Download [Apache Geronimo, Version 1.0](#).
- Download your free copy of [IBM WebSphere® Application Server Community Edition V1.0](#) -- a lightweight J2EE application server built on Apache Geronimo open source technology that is designed to help you accelerate your development and deployment efforts.

Discuss

- [Participate in the discussion forum for this content](#).
- Stay up to date on Geronimo developments at the [Apache Geronimo blog](#).
- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the author

Matthew Jording

Matthew Jording has spent the last decade developing Java Enterprise applications for Fortune 500 companies. Currently employed by Sensis

Corporation, Matthew is working on a Geronimo-based service-oriented architecture for the Aerospace Industry. Matthew and co-author Nathan Mittler are writing *Geronimo In Action* for Manning Publications. Matthew is married with two teenage daughters and lives in Syracuse, New York.