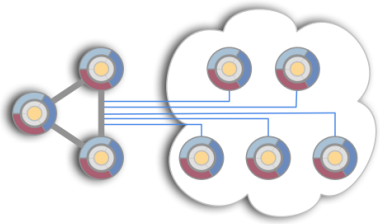
# Resin 3.0 → 4.0 Migration



**Resin 4.0 includes numerous and extensive performance enhancements over its predecessor 3.0, among them:**

- **Improved cloud computing support**
- **Monitoring capabilities at every level of Resin's architecture**
- **Enhanced performance through native optimizations**
- **Simplified configuration**
- **Rewrite of Resin's internal caching**
- **Many more!**

**In order to take advantage of the unprecedented support Resin offers for scaling Java Web Applications, we provide this migration brochure as an outline of how to begin your migration.**

caucho

# Contents



**Preliminaries:**

1. The required JDK version is now Java 1.6. As of this writing, Java 1.5 and earlier have reached the end of service life. Make sure you use the JDK -- the JRE is not sufficient.
2. The build process for compiling the Resin JNI and installing Resin have changed. Before Resin 4.0, you would install Resin by unpacking the distribution directly in the filesystem. Resin 4.0's configure script now creates makefiles that install all the necessary files in standard locations for Linux/Unix. There is also a .deb package for Ubuntu available. Windows installation is unchanged.
3. The main configuration file has been renamed from resin.conf to resin.xml.

As a general recommendation for migration, we suggest starting with a clean installation of Resin 4.0, including using the sample configuration. By beginning with the sample configuration and modifying it to match your deployment setup, you'll be much more likely to configure the server correctly. This approach is usually much easier than starting with your existing configuration and modifying to make it compatible with the new version.

**Starting Resin**

One of the major changes from Resin 3.0 to Resin 4.0 is the introduction of the Resin Watchdog which changes the way in which Resin is started and managed during runtime. The httpd.sh and httpd.exe executables no longer exist. The Watchdog documentation linked above has in-depth details, but essentially Resin now runs with two Java processes, the main Resin process and the Watchdog. This architecture introduces many advantages for management and reliability,

but changes the way that Resin is started and how the Java process is managed.

Example Resin 4.0 command line:

```
java - jar $RESIN_HOME/lib/resin.jar -conf /etc/resin/resin.xml -root-directory=/var/www -server app-a start
```

**JVM arguments**

JVM arguments for the main Resin process are now set within the resin.xml file instead of being passed via a script. Any JVM arguments on the command line actually go to the Watchdog process. JVM arguments can be set per server in a cluster, but in general you can set them in a <server-default> tag to apply to all servers.

For example, if you wanted to set the maximum heap size of all the Resin processes to 1GB:

```
<server-default>
 ...
<jvm-arg>-Xmx1g</jvm-arg>
```

**Linux/Unix**

It is possible to run Resin on the console if you like for debugging or other purposes using the "console" command (instead of the "start" command as above). Note however that if you intend to run Resin as a long running process, you should not run Resin in console mode in the background. This will discard all logging sent to stdout (see the logging section below).

If you are using Linux, a sample init.d script for Resin is included with the distribution. The make install command installs it to /etc/init.d/resin. Edit the file to set the name of the server, the location of the resin.xml file, the log directory, and the root directory. Check with your distribution's documentation to see how to add the script to one or more of your runtime levels. The script may be useful for Unix as well.

Unix allows only root to bind to ports below 1024. If you use Resin as your webserver (recommended) and bind to port 80, you'll need to start Resin as root. In Resin 4.0, the Resin process can drop privileges as soon as it's bound to all its ports. You can configure the user that Resin uses in the <server> or <server-default> sections:

```
<server-default>
  ...
    <resin:if test="${resin.userName == 'root'}">
      <user-name>www-data</user-name>
      <group-name>www-data</group-name>
    </resin:if>
```

**Windows**

On Windows, Resin is now started using the resin.exe program instead of httpd.exe.

Logging

For Resin 4.0, the preferred logging method is to use java.util.logging for all output. Standard output and error are still supported, but should be used for special cases only. The <log> tag of Resin 3.x has been replaced by the <log-handler> tag in Resin 4.0. The <log-handler> can be thought of as a control on output logging, directing log messages from certain sources and of certain levels to a file (or other output stream). Conversely, the <logger> tag controls the source of log messages, controlling the logging level of individual packages. <logger> tags use the same syntax in

Resin 4.0 as in 3.x, but <log> tags should be converted to <log-handler> tags. Conversion is fairly straight forward:

```
<log name="com.caucho" path="stdout:" timestamp="[%H:%M:%S.%s] "/>
```

becomes:

```
<log-handler name="com.caucho" level="all" path="stdout:" timestamp="[%H:%M:%S.%s] "/>
```

This log-handler will send log messages to standard output if you run Resin on the console, but if you run Resin as a daemon process, the output will be in the directory specified by the --log-directory argument passed to the Watchdog on the command line. For example, if you set the log directory to ***/var/www/log*** and ran a server with the id "app-a", the output would be in ***/var/www/log/jvm-app-a.log***. For the "default" server (i.e. the server with an empty string as its id), the filename will be jvm-default.log.

Similarly, the Watchdog's log will go to /var/www/log/watchdog-manager.log.

**Converting clustered setups**

Clustering in Resin 4.0 is drastically different from that of Resin 3.x. Though conversion is required, the resultant configuration file is more understandable and the capabilities are greater. In Resin 4.0, a cluster is a collection of servers that serve a set of virtual hosts. For each virtual host, there are a number of web-apps. The configuration of Resin 4.0 follows this model, with the resin.xml file having roughly the following outline.

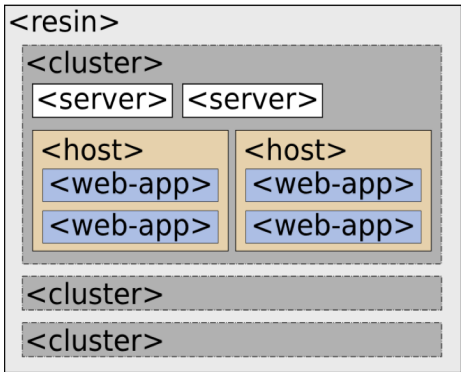
```
<resin>
  <cluster id="app">
    <server>...</server>
    <server>...</server>

    <host id="">
      <web-app>
        ...
      </web-app>
    </host>
  </cluster>

  <cluster id="web">
    ...
  </cluster>
</resin>
```

Notice that <server> is now a child of <cluster> instead of the other way around (clusters contain a set of servers) and that the <srun> tag has disappeared. <host> tags are now side-by-side <server> tags as well, instead of the <host> being a child of <server>. Think of servers as representing a Resin instance and all the configuration that goes along with that (e.g. JVM arguments, ports, ip addresses, etc.). Hosts are collections of web-apps. A cluster then organizes servers and hosts together to show which instances serve which content.

To convert an existing clustered setup to the Resin 4.0 syntax, use the following steps:




1. Replace <server> with <cluster>
2. Delete the old <cluster> tag
3. Rename all <srun> tags to <server>

**Example: Single Server**
In Resin 4.0, all servers are part of a cluster. Note that even if you're using a single server configuration, you are creating a cluster with a single server. Take as an example, the minimal configurations of Resin 3.0 compared to Resin 4.0:

**Resin 3.0 minimal configuration**

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="http://caucho.com/ns/resin/core">

  <server>
    <http server-id="" host="*" port="8080"/>

    <resin:import path="${resin.home}/conf/app-default.xml"/>

    <host id="" root-directory=".">
      <web-app id="/" document-directory="docs"/>
    </host>
  </server>
</resin>
```

**Resin 4.0 minimal configuration**

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="">
    <server id="">
      <http address="*" port="8080"/>
    </server>

    <resin:import path="${__DIR__}/app-default.xml"/>

    <host id="" root-directory=".">
      <web-app id="/" root-directory="docs"/>
    </host>
  </cluster>
</resin>
```

The differences between the two configurations are:

1. The resin namespace is defined differently in Resin 4.0 (it uses a CanDI-style URN)
2. In the Resin 3.0 configuration there is no <cluster> tag, but in Resin 4.0 this configuration is considered as a cluster of one server.
3. The <http> tag is now also within the <server> tag in the Resin 4.0 configuration and the server-id attribute of the <http> tag no longer exists. The host attribute of <http> has been replaced by the address attribute to avoid confusion with virtual hosts.
4. ${__DIR__} refers to the directory containing the configuration file. This replaces the style in the Resin 3.0 configuration and allows relocating the resin.xml easily (e.g. to /etc/resin/resin.xml)

## Example: Two tires with the Resin Load Balancer

In Resin 3.0, it was common to use two configuration files for set ups with a Resin load balancer and a backend app-tier. In Resin 4.0, a single configuration file is used for all servers, including the Resin load balancer. Resin 4.0 also uses its rewrite engine to configure load balancing rather than the explicit servlet configuration in Resin 3.0. This approach allows for more flexible integration of the load balancer with rewrite rules. The following examples show these differences:

**Resin 3.0 minimal configuration**

```
<resin xmlns="http://caucho.com/ns/resin">
<server>
  <cluster id="app-tier">
    <srun id="app-a" host="192.168.0.10" port="6800"/>
    <srun id="app-b" host="192.168.0.11" port="6800"/>
  </cluster>

  ...
</server>
</resin>
```

*app-tier.conf, used by the Resin app tier servers*

```
<resin xmlns="http://caucho.com/ns/resin">
<server>
  <http id="web-a" port="80"/>

  <cluster id="app-tier">
    <srun id="app-a" host="192.168.0.10" port="6800"/>
    <srun id="app-b" host="192.168.0.11" port="6800"/>
  </cluster>

  <host id="">
    <web-app id="/">
      <!-- balance all requests to the servers in cluster a -->
      <servlet>
        <servlet-name>balance-a</servlet-name>
        <servlet-class>com.caucho.servlets.LoadBalanceServlet</servlet-class>
        <init cluster="app-tier"/>
      </servlet>

      <servlet-mapping url-pattern="/*" servlet-name="balance-a"/>
    </web-app>
  </host>
</server>
```

*web-tier.conf, used by the Resin load balancer only*

**Resin 4.0 minimal configuration**

```
<resin xmlns="http://caucho.com/ns/resin"
       xmlns:resin="urn:java:com.caucho.resin">

  <cluster id="app-tier">
    <server id="app-a" address="192.168.0.10" port="6800"/>
    <server id="app-b" address="192.168.0.10" port="6800"/>
    ...
  </cluster>

  <cluster id="web-tier" root-directory="web-tier">
    <server-default>
      <http address="*" port="80"/>
    </server-default>

    <server id="web-a" address="127.0.0.1" port="6800"/>

    <host id="">
      <web-app id="/">
        <resin:LoadBalance regexp="" cluster="app-tier"/>
      </web-app>
    </host>
  </cluster>
</resin>
```

*resin.xml*, used by both app-tier servers and the Resin load balancer

The Resin 4.0 configuration is much more compact overall and avoids the need to synchronize the contents of each file separately. Notice that one difference is that the load balancer server is now a fully fledged Resin server within a cluster as well. It has a name and its cluster can be expanded with other servers easily.

### Server configuration

Resin 4.0's <server> tag contains a number of configuration items that were not previously configured in the Resin 3.0 <srun> tag, so the two should not be considered exactly equivalent. Specifically threading, socket, and JVM configuration is now done on a per <server> basis, giving greater control for systems with heterogeneous hardware or software configurations:

- Thread pool (moved from <resin> in Resin 3.0)
- Keepalive (moved from <server> in Resin 3.0)
- <memory-free-min>] (moved from <resin> in Resin 3.0, renamed from <min-free-memory>)

#### *RewriteFilter*

A simple RewriteFilter was available in Resin 3.0, but has been superseded by Resin 4.0's powerful Rewrite mechanism. The RewriteFilter performed much like the <resin:Forward> tag of Resin 4.0, but the newer Rewrite mechanism adds redirection, request modification, and more.

### Hessian

The implementation of Hessian 2.0 shipped with Resin 4.0 (still in draft form) is not compatible with the earlier Hessian 2.0 draft in Resin 3.0 or with Hessian 1.0. Make sure to update all clients and services to have matching versions.

#### *Deprecated and removed features*

- Support for Apache 1.x has been removed
- Resin 3.0 configuration compatibility has been removed
- Resin 3.1 rewrite tags are deprecated

#### *Resin IoC*

The Resin IoC system has evolved from using tags such as <resource> and the resin:type attribute to a more sophisticated and complete system based on JSR-299, called CanDI. The basic steps to convert from a bean declared as a <resource> to a CanDI bean are:

1. Create a tag for your bean, using an appropriate namespace prefix (e.g. <test:MyBean>)
2. Create a URN namespace for your class' package (e.g. xmlns:test="urn:java:com.example.test")
3. If you have assigned a name for your bean using the name attribute, use the javax.inject.Named annotation instead
4. Remove the <init> tag
5. If you have assigned a jndi-name, use the @Resource annotation. (Consider removing JNDI and using type safe annotations or the @Named annotation instead.)

**Resin 3.0 <bean> configuration**

```
<web-app xmlns="http://caucho.com/ns/resin">

  <bean name="test">
    <type>com.example.test.MyBean</type>
    <init>
      <greeting>Hello</greeting>
      <server>${serverId}</server>
      <sub-bean>
        <value>${2 + 2}</value>
      </sub-bean>
    </init>
  </bean>

</web-app>
```

**Resin 4.0 CanDI configuration**

```
<web-app xmlns="http://caucho.com/ns/resin"
            xmlns:inject="urn:java:javax.inject">

  <test:MyBean xmlns:test="urn:java:com.example.test">
    <inject:Named>test</inject:Named>

    <test:greeting>Hello</test:greeting>
    <test:server>${serverId}</test:server>
    <test:sub-bean>
      <test:value>${2 + 2}</test:value>
    </test:sub-bean>
  </test:MyBean>

</web-app>
```

Example: Basic bean configuration