

# Table of Contents

<b>Part I Document Overview</b>	<b>2</b>
<b>Part II Linux Installation</b>	<b>3</b>
<b>Part III IBM's Java Installation</b>	<b>4</b>
1 Installation .....	4
2 Configuration .....	4
<b>Part IV Tomcat Installation</b>	<b>5</b>
1 Installation .....	5
2 Configuration .....	5
3 Test .....	5
<b>Part V Cocoon Installation</b>	<b>6</b>
1 Installation .....	6
2 Installation of Xvfb .....	6
3 Test .....	6
4 Configuration .....	6
<b>Part VI Apache Installation</b>	<b>8</b>
1 Installation .....	8
2 Configuration .....	8
<b>Part VII Firewall / IPTABLES</b>	<b>10</b>
1 Configuration .....	10
<b>Part VIII Finishing Up</b>	<b>12</b>
<b>Part IX Appendix</b>	<b>13</b>
<b>Index</b>	<b>0</b>

# 1 Document Overview



**Author:** NETIKUS.NET ltd  
**Date:** 22nd February  
**Revision:** 2002  
2

## Cocoon Installation on RedHat Linux

<b>Title</b>	Cocoon Installation on RedHat Linux
<b>Summary</b>	This document describes how to setup a Linux server to use Cocoon in combination with the Apache web server on a Linux installation. It assumes familiarity with Linux on the command line. While this installation was performed on Redhat it should work well on any other distribution with a recent kernel version.
<b>Software</b>	Custom Installation of Linux Redhat ( <a href="http://www.redhat.com/">http://www.redhat.com/</a> ) Apache Web server (source) ( <a href="http://httpd.apache.org/">http://httpd.apache.org/</a> ) Jakarta Tomcat (rpm) ( <a href="http://jakarta.apache.org/tomcat/">http://jakarta.apache.org/tomcat/</a> ) Cocoon ( <a href="http://xml.apache.org/cocoon/">http://xml.apache.org/cocoon/</a> ) IBM Developer Kit for Linux, Java 2 ( <a href="http://www.ibm.com/java/">http://www.ibm.com/java/</a> )
<b>Background</b>	Cocoon is a java-based application that, among other features, converts XML documents to HTML documents on the fly using XML style sheets. Cocoon needs a JSP engine; we will use the latest release of Tomcat, at the time version 4.0.1. Even though Tomcat comes with a HTTP engine, we will use Apache since it is renowned for it's stability, reliability and security.
<b>Skill Level</b>	Intermediate
<b>Skills Required</b>	<ul style="list-style-type: none"><li>- Basic understanding of Unix</li><li>- Basic understanding of TCP/IP</li><li>- Basic familiarity of Unix shell (e.g. csh, bash)</li><li>- Basic usage of the vi editor</li></ul>
<b>Download</b>	<a href="http://www.netikus.net/">http://www.netikus.net/</a> (guides section)

## 2 Linux Installation

Any kind of installation is fine, but support for the following has to be included

- Iptables
- SSH Server (for remote administration)

Since the Tomcat RPM (and Cocoon with the XML files in turn) installs into the **/var** mount point, this mount point should be sized accordingly.

## 3 IBM's Java Installation

Tomcat and Cocoon both need Java, so the latest version has to be downloaded and installed.

### 3.1 Installation

Navigate to <http://www.ibm.com/java>, click on Tools and products, choose IBM Developer Kit for Linux, Java™ 2 Technology Edition, Version 1.3 and download the Developer Kit package as a RPM. It is recommended to download the JDK rather than the JRE, however the JRE might be sufficient. The JDK RPM installs into the /opt directory.

### 3.2 Configuration

To set up the **JAVA\_HOME** environment variable add something like this to the global or user specific profile file. I used **/etc/profile**:

```
JAVA_HOME=/opt/IBMJava2-13
export JAVA_HOME

PATH=$PATH:$JAVA_HOME/bin
export PATH
```

Tomcat has its own initialization file but it's still a good idea to create those environment variables. After you set those up you should be able to simply execute **java** from the command-line and see the usage of the **java** command without error messages.

I personally prefer IBM's virtual machine to Sun's virtual machine since it seems to have fewer bugs.

## 4 Tomcat Installation

### 4.1 Installation

The following RPMs need to be downloaded and installed (in the order specified below) from <http://jakarta.apache.org/builds/jakarta-tomcat-4.0.1/rpms/> :

- `rpm -i tomcat4-4.0.1-1.noarch.rpm`
- `rpm -i tomcat4-webapps-4.0.1-1.noarch.rpm`

which will create a `/var/tomcat4` directory, a `/etc/init.d/tomcat4` control script and a configuration file `/etc/tomcat4/conf/tomcat4.conf`.

Unfortunately Tomcat does not set the permissions correctly in the `/var/tomcat4` directory. Please change the permissions so that a `ls -al` looks like this:

```
drwxr-xr-x 10 root    root    4096 Feb  4 16:31 .
drwxr-xr-x 17 root    root    4096 Feb  5 16:00 ..
drwxr-xr-x  2 root    tomcat4 4096 Feb  4 16:31 bin
drwxr-xr-x  4 root    tomcat4 4096 Feb  4 16:31 common
drwxr-xr-x  2 root    tomcat4 4096 Feb  5 17:36 conf
drwxr-xr-x  2 root    tomcat4 4096 Feb  4 16:31 lib
drwxr-xr-x  2 tomcat4 tomcat4 4096 Feb  7 11:40 logs
drwxr-xr-x  4 root    tomcat4 4096 Feb  4 16:31 server
drwxr-xr-x  8 tomcat4 tomcat4 4096 Feb  5 17:19 webapps
drwxr-xr-x  4 tomcat4 tomcat4 4096 Feb  5 17:37 work
```

If the permissions are set differently the automatic extraction of the `cocoon.war` file will most likely fail.

### 4.2 Configuration

All the default configuration options for Tomcat seem to be okay for this RPM, just make sure that the `JAVA_HOME` entry in the file `/etc/tomcat4/conf/tomcat4.conf` corresponds to your Java installation.

Please note that the default port that Tomcat listens on is not 8080 (as specified in all documentation files) but **8180**. There seem to be a few problems with the RPM file, so it might be better to avoid RPMs on future installations. This option is configured in the `/var/tomcat4/conf/server.xml` file.

### 4.3 Test

Start Tomcat and connect with your browser to <http://localhost:8180> and you should see a Tomcat test page.

## 5 Cocoon Installation

### 5.1 Installation

Download **cocoon-2.0.1-bin.tar.gz** from <http://apache.get-software.com/cocoon/BINARIES/> and unpack the file. The only important file contained in the archive is the file **cocoon.war**. Copy this file to the Tomcat web applications directory **/var/tomcat4/webapps**.

Now restart Tomcat and the archive file **cocoon.war** should be unpacked automatically, creating a directory in the **webapps** folder. Don't bother going on until the folder **cocoon** has been automatically created in the **/var/tomcat4/webapps** folder.

Tomcat takes several seconds to completely shutdown **after** the shutdown has been reported as **OK**. I usually check with **ps -ef** to see if all Java processes have disappeared before I restart it. On a Pentium II this might take ~20 seconds.

### 5.2 Installation of Xvfb

Cocoon (because of the Java awt library I read) requires X-Windows to be present. Fortunately there is a way to simulate X-Windows by running **Xvfb**. Install the **XFree86-Xvfb-4.1.0-3** RPM from the Redhat 7.2 CD and add the following lines to the Tomcat startup file **/etc/init.d/tomcat4**:

```
pgrep Xvfb >/dev/null
if [ $? -ne 0 ]; then
    echo "Starting X emulator Xvfb ..."
    Xvfb :0 -screen 0 800x600x8 &
fi
export DISPLAY=0:0
```

so that we don't have to do this manually. For the sake of testing you can simply execute the bold lines above to avoid another Tomcat restart.

### 5.3 Test

If the directory has been created automatically and the **Xvfb** process is active, connect to the <http://localhost:8180/cocoon/> and hope that the Cocoon welcome page will appear.

### 5.4 Configuration

If you come this far then you will find the rest relatively easy. Cocoon uses a file called **sitemap.xmap** as its main configuration file. It is there where we tell Cocoon what files to display, map, transform etc. To keep it simple we will simply use an **XML** document with an associated **XSL** file to produce a **HTML** file. Create a test folder in the **.../cocoon** folder called **test** and add the following lines at the bottom of the file **before**

```
</map:pipeline>
</map:pipelines>
```

```
</map:sitemap>
```

so that it looks like this:

```
<!-- serve all .gif graphic files from the vincent dir -->
<map:match pattern="**.gif">
  <map:read src="vincent/{1}.gif" mime-type="image/gif"/>
</map:match>

<!-- serve all .jpg graphic files from the vincent dir -->
<map:match pattern="**.jpg">
  <map:read src="vincent/{1}.jpg" mime-type="image/jpg"/>
</map:match>

<!-- serve all .png graphic files from the vincent dir -->
<map:match pattern="**.png">
  <map:read src="vincent/{1}.png" mime-type="image/png"/>
</map:match>

<!-- directory listing of root (/) -->
<!-- this is only important if you want to browse the root dir -->
<map:match pattern="">
  <map:generate type="directory" src="vincent"/>
  <map:transform src="stylesheets/system/directory2html.xsl"/>
  <map:serialize/>
</map:match>

<!-- show source of xml files rather than converting them -->
<map:match pattern="source/*.xml">
  <map:read src="vincent/{1}.xml" mime-type="text/plain"/>
</map:match>

<!-- convert .xml files with a .xsl file to a html file -->
<map:match pattern="**.xml">
  <map:generate type="file" src="vincent/{1}.xml"/>
  <map:transform type="xslt" src="hogwarts.xsl"/>
  <map:serialize/>
</map:match>

</map:pipeline>
</map:pipelines>

</map:sitemap>
```

Here we tell Cocoon

- how to serve graphics files
- how to show a directory listing
- how to convert xml files with a stylesheet file

You will need to restart Cocoon (Tomcat) and copy the test files into that folder to make this work. Then simply navigate to <http://localhost:8180/cocoon/test/> and you should be able to click on any **.xml** and have it transformed into HTML format (while keeping its extension).

## 6 Apache Installation

### 6.1 Installation

While not necessary it is wise to use Apache to handle HTTP requests rather than using Tomcat, at least if that is possible. We will then connect to port 80 of our server instead of port 8180.

To use Cocoon (or better Tomcat) through Apache we will need to install a module that communicates with the Tomcat server. Tomcat comes with the **webapps** module, which we need to download first.

Navigate to <http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.1/bin/linux/i386/> and download the file **webapp-module-1.0-tc40-linux-glibc2.2.tar.gz**. We will install the module once Apache is setup and working, but since it's already compiled this will be easy.

To download Apache navigate to <http://www.apache.org/dist/httpd/> and retrieve the file **apache\_1.3.23.tar.gz**. Unpack the archive and run the following commands to compile the source:

```
./configure --enable-module=rewrite --enable-module=so
make
make install
```

I then copied (and adjusted the file and directory locations) the **httpd** startup script from another Redhat installation to the **/etc/init.d/** directory.

If you install Apache from an **.rpm** file then the startup script **httpd** will be placed in the **/etc/init.d** directory automatically.

### 6.2 Configuration

To make sure that Apache is installed correctly, execute **/usr/local/apache/bin/apachectl start** and navigate to <http://localhost/> and see if the default page appears. If it works stop Apache again issuing the same command with the **stop** option.

At this point Apache knows nothing about Tomcat or Cocoon, so we will need to tell Apache that we want it to pass certain requests to Tomcat using the webapps module.

To accomplish this we need to extract the file **mod\_webapps.so** from **webapp-module-1.0-tc40-linux-glibc2.2.tar.gz**. This module is already compiled so we only need to copy the file **mod\_webapps.so** to the **/usr/local/apache/modules/** directory. You can create this directory if it doesn't exist yet.

Then add the following two lines to the file **/usr/local/apache/conf/httpd.conf** (after the **Dynamic Shared Object (DSO)** section) to use the newly installed module:

```
LoadModule webapp_module      modules/mod_webapp.so
AddModule mod_webapp.c
```

This will make Apache load the module every time it starts. To actually map a "virtual" subdirectory to Cocoon, add the following two lines anywhere (outside a directive like **<directory>** though) to **httpd.conf**:

```
WebAppConnection conn warp localhost:8008
```

```
WebAppDeploy cocoon conn /xmlconverter
```

The first line, **WebAppConnection**, creates a connection to the Tomcat server, running on the localhost on port 8008 – naming that connection **conn**. The keyword **warp** specifies how to connect, but **warp** seems to be the only supported option at the moment.

The second line, **WebAppDeploy**, maps the requested subdirectory **xmlconverter** to the **/cocoon** subfolder of the **Tomcat webapps** folder, using the previously defined connection.

It seems as if one has to start Tomcat **before** Apache, since the webapp module will otherwise complain that the **cocoon** web application has not yet been deployed. This can be tricky since Tomcat takes quite some time to start and "deploy" its web applications. I managed to solve this problem by loading Tomcat **early** (S40) and Apache **very late** (S98) in the boot process. Additionally I added a 20 seconds timeout to the **/etc/init.d/httpd** startup script.

## 7 Firewall / IPTABLES

### 7.1 Configuration

IPTABLES should already be installed so we only need to setup an Iptables script to activate our rules. The following rules will be allowed:

- Incoming connections to port 80 (http)
- Incoming connections to port 22 (ssh)
- Incoming connections to port 21/20 (ftp)
- Any outgoing connection
- ICMP will be allowed

Below is the firewall script, saved as **/etc/init.d/firewall**:

```
# Firewall script by ingmar@02/06/2002
#
# * Remote hosts can access: FTP SSH HTTP
# * Local host can connect: to everywhere
# * ICMP is enabled

# Source function library
. /etc/rc.d/init.d/functions

# Path to IPTABLES
IPTABLES=/sbin/iptables

case "$1" in
start)
echo -n "Activating Firewall: "
# Load the ftp module (for passive connections?)
modprobe ip_conntrack_ftp

# Flush Input & Output chain -> empty
$IPTABLES -F INPUT
$IPTABLES -F OUTPUT
$IPTABLES -F FORWARD

# Disallow everything we don't allow later
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

# Accept ICMP packets
$IPTABLES -A INPUT -p ICMP -j ACCEPT

# Accept packets to port 22 & 80 & SAMBA
$IPTABLES -A INPUT -p TCP --dport 22 -j ACCEPT
$IPTABLES -A INPUT -p TCP --dport 80 -j ACCEPT
$IPTABLES -A INPUT -p TCP --dport 137:139 -j ACCEPT
$IPTABLES -A INPUT -p UDP --dport 137:139 -j ACCEPT

# Accept incoming packets that are related
# If this's skipped we won't receive replies from our own packets
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Finally allow all packets from this host to go anywhere
$IPTABLES -A OUTPUT -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
    # This is needed for Apache and SWAT to connect to local ports
    $IPTABLES -A INPUT -i lo -j ACCEPT
    echo_success
    echo
    ;;
stop)
    echo -n "Deactivating Firewall: "

    # Flush Input & Output chain -> empty
    $IPTABLES -F INPUT
    $IPTABLES -F OUTPUT

    # Allow all packets
    $IPTABLES -P INPUT ACCEPT
    $IPTABLES -P OUTPUT ACCEPT

    echo_success
    echo
    ;;
status)
    # Display Filter
    $IPTABLES --list
    ;;
*)
    echo "Usage $0 {start|stop|status}"
    exit 1
esac

exit 0
```

A port scan should now only list ports 22 and 80 and 137-139 as being accessible.

The current version of this script always reports OK when one starts or stops the script. If the script should fail however IPTABLES would display error messages. A future version might „fix“ this problem – it is very unlikely for this script to fail once it has been tested however.

## 8 Finishing Up

Since everything is already in place we just need to create symbolic links in the rc0 rc1 rc3 and rc6 directory for Tomcat, Apache and the firewall script. Here are the values that I have used:

Tomcat: S40, K01

Apache: S98, K06

Firewall: S08 (Disabling not necessary)

## 9 Appendix

This section contains the source of a **.xml** and **.xsl** file which you can use to test the most basic Cocoon functionality. Create the following files and copy them to a subdirectory of Cocoon, I will use **xmltest**:

### test.xml:

```
<?xml version="1.0"?>
  <page>
    <title>XML and XSL become HTML</title>
    <greeting>Hello NETIKUS.NET</greeting>
  </page>
</?xml version="1.0"?>
```

### test.xsl:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="page">
    <html>
      <head>
        <title><xsl:value-of select="title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="title"/></h1>
        <p><xsl:value-of select="greeting"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Then add the following lines (as explained in chapter 4.d) to **sitemap.xmap**,

```
<map:match pattern="xmltest/*.xml">
  <map:generate type="file" src="xmltest/{1}.xml">
  <map:transform type="xslt" src="xmltest/{1}.xsl">
  <map:serialize/>
</map:match>
```

restart Tomcat and Apache, and get yourself a cup of tea or coffee. Please note that we could have also written

```
<map:transform type="xslt" src="xmltest/test.xsl">
```

in line three to map every **.xml** file in the **xmltest** subdirectory to **test.xsl**.

Please send comments to [ingmar.koecher@netikus.net](mailto:ingmar.koecher@netikus.net)