# APACHE HTTPD – VIRTUAL HOSTS, CACHING, SECURING, CONNECTING TO EAP

Joel Tosi
02/03/2011

This technical whitepaper covers Apache HTTPD from installation to configuring it to be used with JBoss EAP. Topics include what is the Apache HTTPD; installing HTTPD from JBoss EWS; setting up virtual hosts; configuring caching; securing your environment; logging and debugging; setting up SSL; and various ways to connect Apache HTTPD to JBoss EAP. The paper wraps up with some recommendations on best practices.

## Contents

## 1. APACHE HTTPD

Apache HTTPD – also called Apache Web Server – is the most popular, web server on the Internet. It is a project of the Apache Software Foundation.

## 2. APACHE HTTPD INCLUSION IN JBOSS EWS

Although JBoss does not require a web server running in front of it, for best performance and as best practice, a web server is recommended. The web server can serve a multitude of functions including serving up static content and caching content which will result in reduced load to your application server. Additionally a web server serves to further separate your application layer from the web, resulting in increased security. JBoss EWS includes a pre-built version of Apache HTTPD along with the Apache Tomcat connector (mod_jk). A default configuration file is delivered as well. This configuration will need to be modified to meet your application needs, though the major points will be covered in the remainder of this document.

## 3. LIMITATIONS OF APACHE HTTPD

Apache HTTPD is the leading web server in use today. However, it is limited to serving up static content and cannot host your Java applications. There are also limitations as to how many requests each web server can handle. This depends on your configuration as well as resources (memory, CPU) available to the web server. Scaling Apache HTTPD is outside the scope of this technical whitepaper.

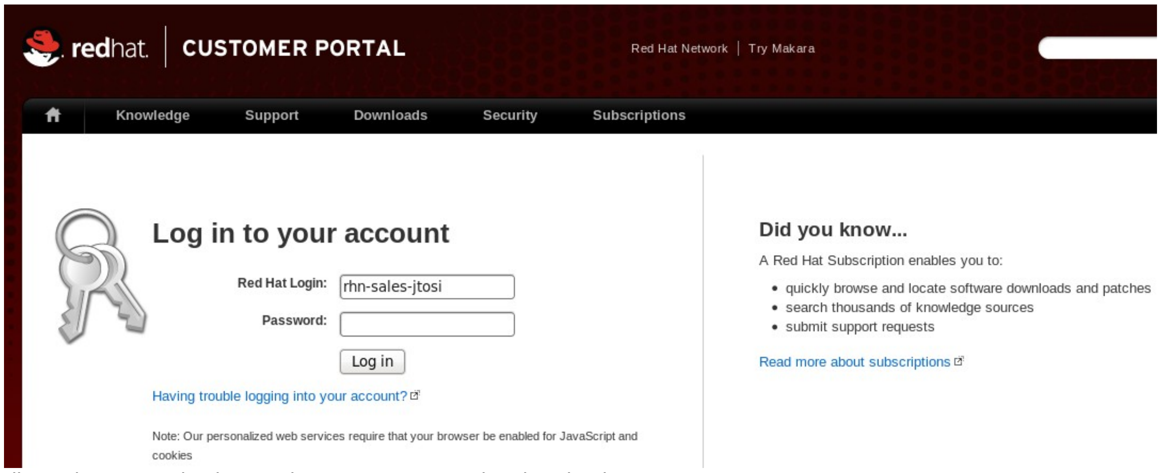
## 4. INSTALLING AND CONFIGURING APACHE HTTPD – COMMON PRACTICES

The steps listed below are not complete or wholistic. It is only intended to show common practices for front-ending JBoss with Apache .

### 4.1 Installation (as rpm from JBoss)

The recommended approach for installing Apache HTTPD in a manner most supported by RedHat / Jboss is using the rpm. To download the rpm manually, execute the following steps:

1. Login to access.redhat.com with your customer account information



Log in to your account

Red Hat Login: rhn-sales-jtosi

Password:

Having trouble logging into your account?

Note: Our personalized web services require that your browser be enabled for JavaScript and cookies

Did you know...

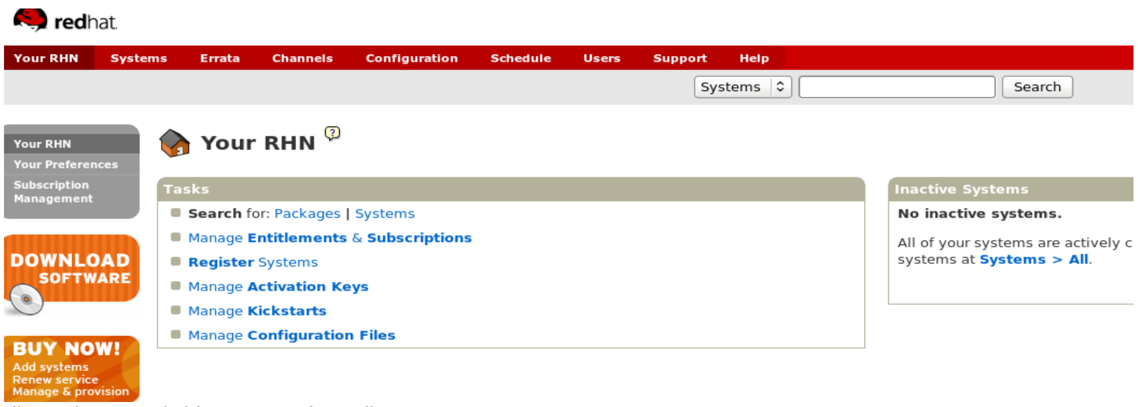
A Red Hat Subscription enables you to:

- quickly browse and locate software downloads and patches
- search thousands of knowledge sources
- submit support requests

Read more about subscriptions

*Illustration 1: Logging into RedHat Customer portal to download HTTPD*

2. Switch the browser url from access.redhat.com to rhn.redhat.com. This will take you to the redhat network where rpms can be found.



Your RHN

Tasks

- Search for: Packages | Systems
- Manage Entitlements & Subscriptions
- Register Systems
- Manage Activation Keys
- Manage Kickstarts
- Manage Configuration Files

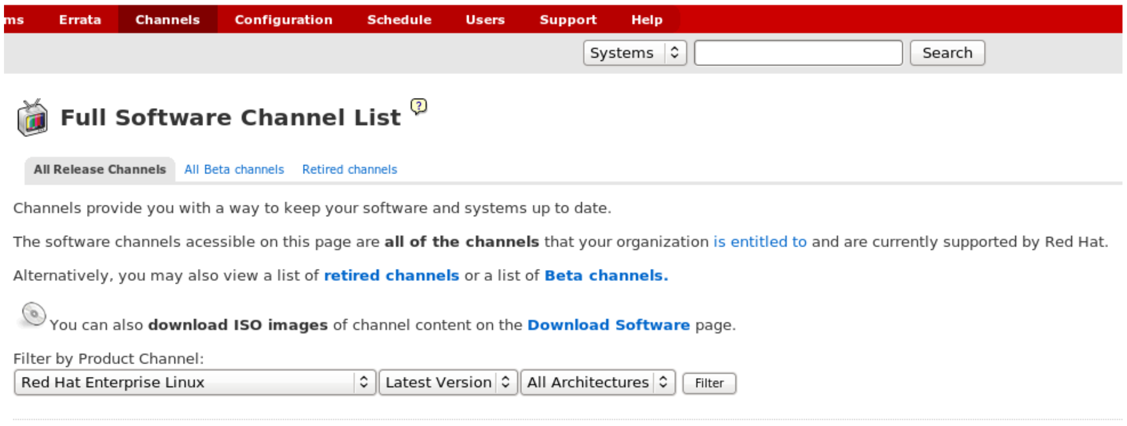
Inactive Systems

No inactive systems.

All of your systems are actively c
systems at Systems > All.

*Illustration 2: Switching over to rhn.redhat.com*

3. Click on the ‘channels’ tab



Full Software Channel List

Channels provide you with a way to keep your software and systems up to date.

The software channels acessible on this page are **all of the channels** that your organization is entitled to and are currently supported by Red Hat.

Alternatively, you may also view a list of **retired channels** or a list of **Beta channels.**

You can also **download ISO images** of channel content on the **Download Software** page.

Filter by Product Channel: Red Hat Enterprise Linux | Latest Version | All Architectures | Filter

*Illustration 3: Channels tab on rhn*

4. Select ‘Package’ from the dropdown and enter the rpm name. Choose httpd22:

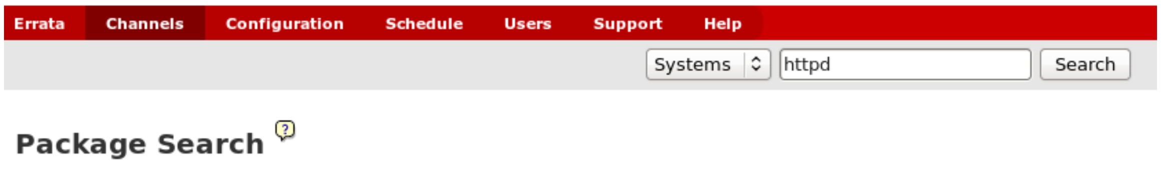
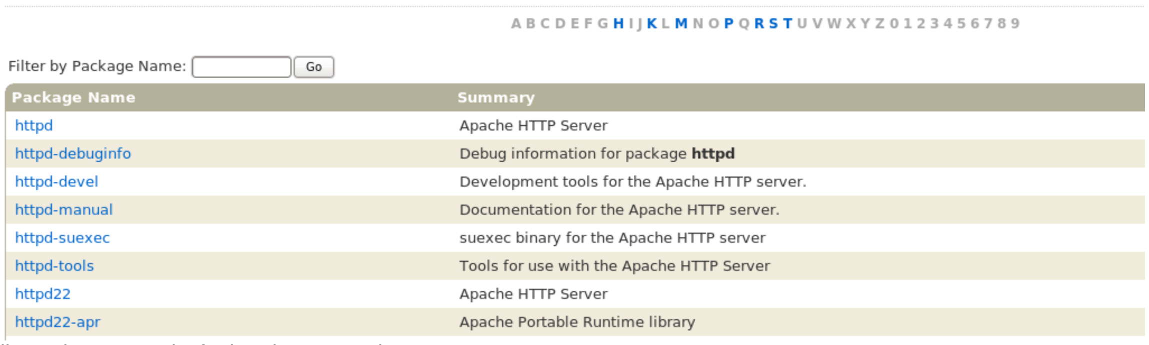


*Illustration 4: Select Package from dropdown and enter rpm name*



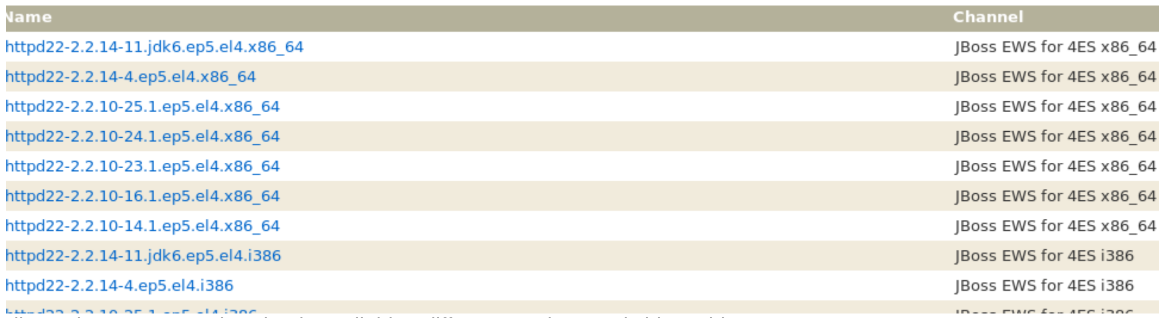
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

Filter by Package Name: Go

| Package Name | Summary |
|---|---|
| httpd | Apache HTTP Server |
| httpd-debuginfo | Debug information for package **httpd** |
| httpd-devel | Development tools for the Apache HTTP server. |
| httpd-manual | Documentation for the Apache HTTP server. |
| httpd-suexec | suexec binary for the Apache HTTP server |
| httpd-tools | Tools for use with the Apache HTTP Server |
| httpd22 | Apache HTTP Server |
| httpd22-apr | Apache Portable Runtime library |

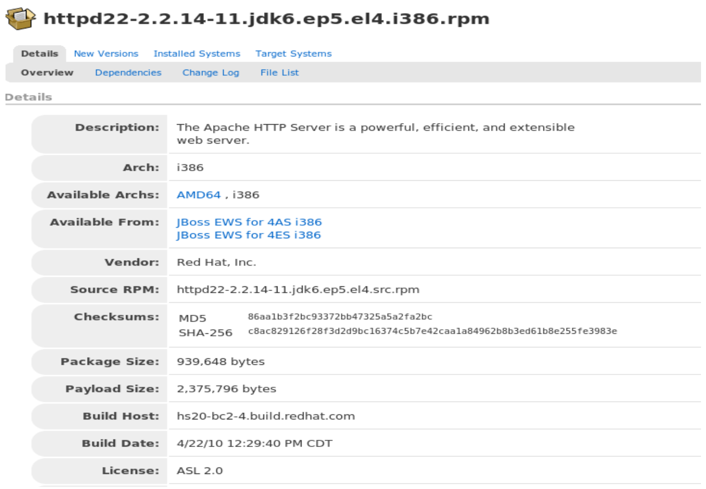
*Illustration 5: Results for httpd rpm search*

5. Download the appropriate version for your environment



| Name | Channel |
|---|---|
| httpd22-2.2.14-11.jdk6.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.14-4.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.10-25.1.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.10-24.1.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.10-23.1.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.10-16.1.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.10-14.1.ep5.el4.x86_64 | JBoss EWS for 4ES x86_64 |
| httpd22-2.2.14-11.jdk6.ep5.el4.i386 | JBoss EWS for 4ES i386 |
| httpd22-2.2.14-4.ep5.el4.i386 | JBoss EWS for 4ES i386 |

*Illustration 6: HTTPD downloads available - different versions and chip architectures*



**httpd22-2.2.14-11.jdk6.ep5.el4.i386.rpm**

Details | New Versions | Installed Systems | Target Systems

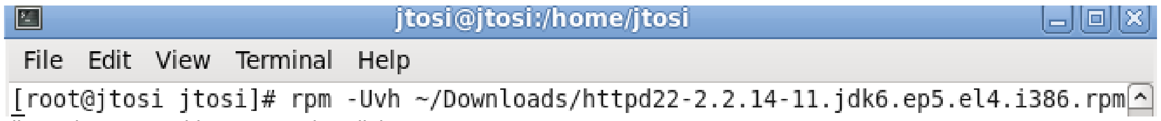
Overview | Dependencies | Change Log | File List

Details

| | |
|---|---|
| **Description:** | The Apache HTTP Server is a powerful, efficient, and extensible web server. |
| **Arch:** | i386 |
| **Available Archs:** | AMD64 , i386 |
| **Available From:** | JBoss EWS for 4AS i386<br>JBoss EWS for 4ES i386 |
| **Vendor:** | Red Hat, Inc. |
| **Source RPM:** | httpd22-2.2.14-11.jdk6.ep5.el4.src.rpm |
| **Checksums:** | MD5 86aa1b3f2bc93372bb47325a5a2fa2bc<br>SHA-256 c8ac829126f28f3d2d9bc16374c5b7e42caa1a84962b8b3ed61b8e255fe3983e |
| **Package Size:** | 939,648 bytes |
| **Payload Size:** | 2,375,796 bytes |
| **Build Host:** | hs20-bc2-4.build.redhat.com |
| **Build Date:** | 4/22/10 12:29:40 PM CDT |
| **License:** | ASL 2.0 |

*Illustration 7: Downloading appropriate version for my environment*

6. Install the rpm as root.



```
[root@jtosi jtosi]# rpm -Uvh ~/Downloads/httpd22-2.2.14-11.jdk6.ep5.el4.i386.rpm
```

*Illustration 8: Run this as root to install the rpm*

## 4.2 Installation (as zip from JBoss EWS)

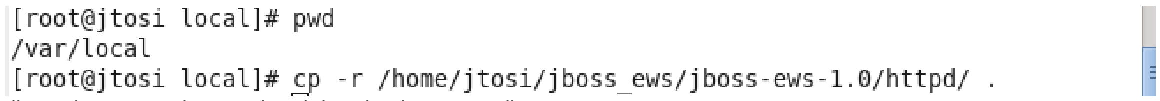
**NOTE** – If you already have an instance of Apache HTTPD installed and just want to configure it, please skip this section.

JBoss EWS comes with a version of Apache HTTPD web server.  To install it, execute the following steps.

If you haven't done so already, download JBoss EWS from the customer support portal - https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=webserver&downloadType=distributions&productChanged=yes&version=5.0.2%20GA

Unzip this download.  For the sake of consistency we will refer to the location where you unzipped this file to as $EWS_HOME.

Copy recursively the httpd directory to the location on your filesystem where you want Apache HTTPD installed at.  In this example we install to /var/local/httpd:



```
[root@jtosi local]# pwd
/var/local
[root@jtosi local]# cp -r /home/jtosi/jboss_ews/jboss-ews-1.0/httpd/ .
```

*Illustration 9: Copying ews httpd download to a new directory*

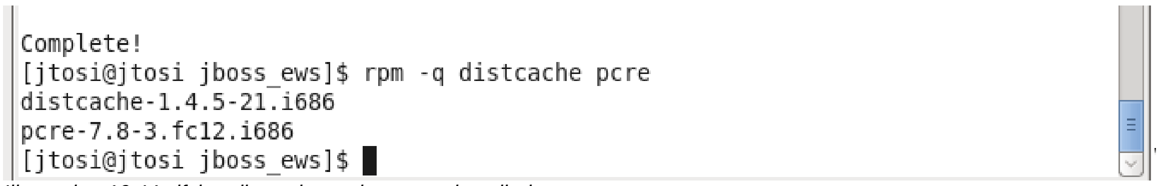
The version of Apache that ships with EWS comes ready with SSL.  Because of this dependency, you will need to make sure that *distcache* and *pcre* are installed on your system (*nix)

```
rpm -q distcache pcre
```

If you get a version number back, then you are fine.  If either is not installed, then you will need to install them, for example

```
yum install distcache
```

Once installation is complete, run the following again to make sure you see both installed:



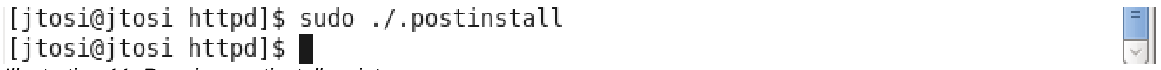
```
Complete!
[jtosi@jtosi jboss_ews]$ rpm -q distcache pcre
distcache-1.4.5-21.i686
pcre-7.8-3.fc12.i686
[jtosi@jtosi jboss_ews]$
```

*Illustration 10: Verifying distcache and pcre are installed*

If you see the above (or similar / higher version numbers) then continue on.

Go to the $EWS_HOME/httpd directory (in these examples it is /var/local/httpd)

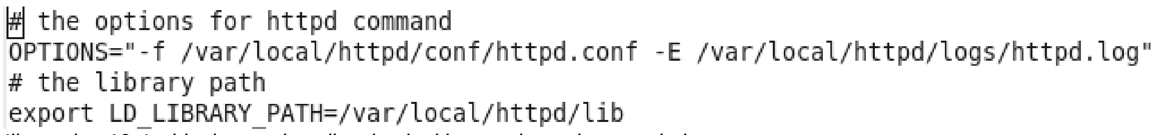
Execute the postinstall script: (note you will need to have root access for this)



```
[jtosi@jtosi httpd]$ sudo ./.postinstall
[jtosi@jtosi httpd]$
```

*Illustration 11: Running postinstall script*

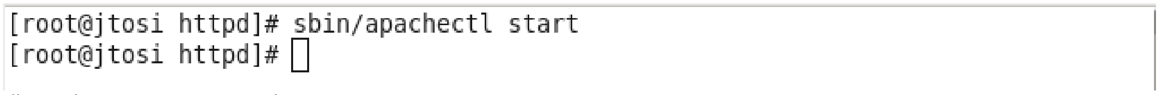
If you do not see any errors, then the script has executed successfully.  This script creates an SSL certificate to use when you first start up apache. **NOTE –** you will need to apply an enterprise level SSL certificate for proper use.  The script also sets up some environment variables to be used as well as updates the configuration file with installation location information.  The configuration file will be covered in upcoming sections:



```
# the options for httpd command
OPTIONS="-f /var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log"
# the library path
export LD_LIBRARY_PATH=/var/local/httpd/lib
```

*Illustration 12: Inside the postinstall script, looking at what values are being set*

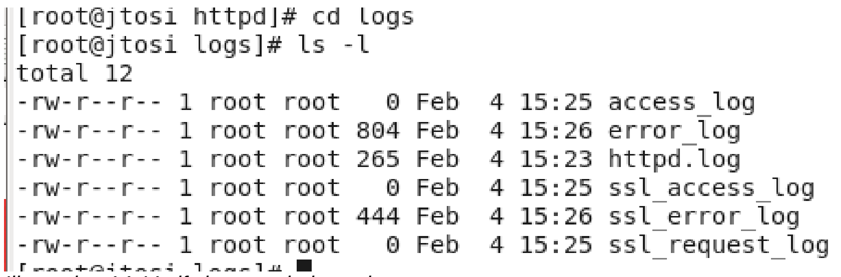
At this point in time, we should verify that we can start up Apache HTTPD and that it starts without error.



```
[root@jtosi httpd]# sbin/apachectl start
[root@jtosi httpd]#
```

*Illustration 13: Start up Apache HTTPD*

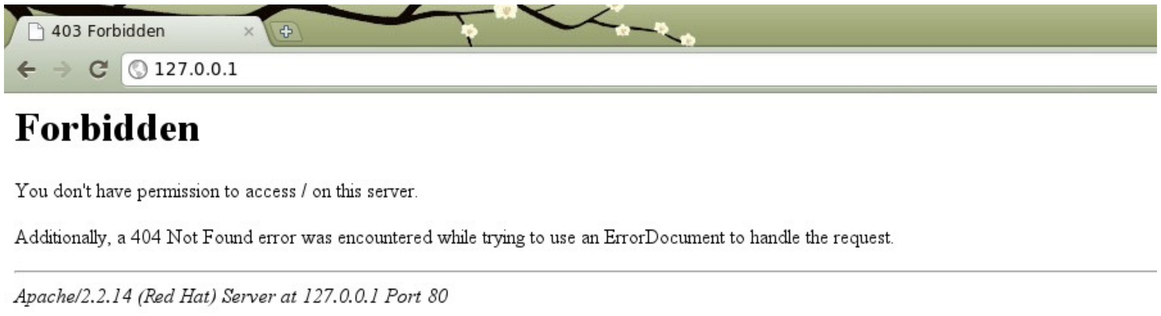
Go to the logs directory and make sure the following log files are present:



```
[root@jtosi httpd]# cd logs
[root@jtosi logs]# ls -l
total 12
-rw-r--r-- 1 root root   0 Feb  4 15:25 access_log
-rw-r--r-- 1 root root 804 Feb  4 15:26 error_log
-rw-r--r-- 1 root root 265 Feb  4 15:23 httpd.log
-rw-r--r-- 1 root root   0 Feb  4 15:25 ssl_access_log
-rw-r--r-- 1 root root 444 Feb  4 15:26 ssl_error_log
-rw-r--r-- 1 root root   0 Feb  4 15:25 ssl_request_log
```

*Illustration 14: Verify logs are being written to*

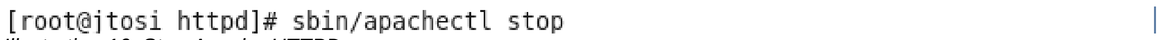
Finally verify that you can access the web server through a browser by hitting http://localhost. **NOTE** – you will get a 403 error because of the default HTTPD configuration shipped with EWS.



```
Forbidden

You don't have permission to access / on this server.

Additionally, a 404 Not Found error was encountered while trying to use an ErrorDocument to handle the request.

Apache/2.2.14 (Red Hat) Server at 127.0.0.1 Port 80
```

*Illustration 15: First time accessing HTTPD through a browser*

If you have confirmed that your Apache HTTPD instance is up, stop the server now so we can finish configuring it. Do this by executing the following command:



```
[root@jtosi httpd]# sbin/apachectl stop
```

*Illustration 16: Stop Apache HTTPD*

### 4.3 Setting up Virtual Hosts

Virtual Hosts allow your web server to respond to request from multiple domains. Using Virtual Hosts, you can have a smaller web server farm fronting a large and diverse application layer. Virtual Hosts can be setup in one of 2 different ways (or blended):

- Name based – Allows you to respond to multiple domains with one IP address. With this approach, you define the domain that a configuration will apply to
- IP based – Only one domain per IP address

To setup Virtual Hosts, we will modify the httpd.conf file located at $EWS_HOME/httpd/conf

Name based Example:

```
##Listen on all IPS, NameVirtualHost is required for name based
##virtual hosting

## NOTE IPv6 addresses must be in brackets:

## NameVirtualHost [2001:db8::a00:20ff:fea7:ccea]:80
NameVirtualHost *:80


##If you only have one domain, you don’t need a virtual host
```

```
##As soon as you add a second, the original domain must be added as
##a virtual host

##First defined virtual host will be the default
<VirtualHost *:80>
 ServerName www.jboss.org
 ServerAlias jboss.org *.jboss.org
  DocumentRoot /www/jboss_org
</VirtualHost>

<VirtualHost *:80>
  ServerName www.jboss.com
  DocumentRoot /www/jboss_com
</VirtualHost>
```

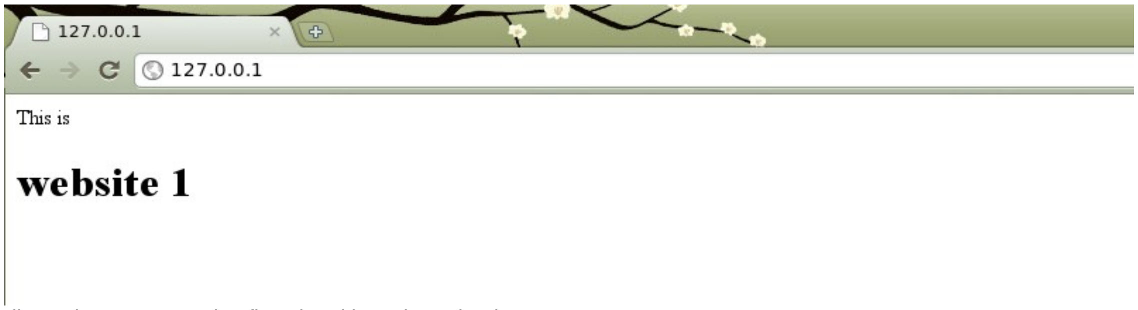
2 IP based Example:

```
##If you only have one domain, you don't need a virtual host

##As soon as you add a second, the original domain must be added as
##a virtual host

##First defined virtual host will be the default
<VirtualHost 127.0.0.1:80>
 ServerName www.jboss.org
 ServerAlias jboss.org *.jboss.org
 DocumentRoot /www/jboss_org
</VirtualHost>

<VirtualHost 127.0.0.2:80>
 ServerName www.jboss.com
 DocumentRoot /www/jboss_com
</VirtualHost>
```

In the example below, we modified $EWS_HOME/httpd/conf/httpd.conf with the following changes, resulting in two different domains being served up from one server.

```
<VirtualHost 127.0.0.1>

  DocumentRoot /home/jtosi/website1

  ServerName www.site1.com

  ErrorLog logs/site1-error.log

  CustomLog logs/site1-access.log common
  <Directory "/home/jtosi/website1">
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>

<VirtualHost 192.168.1.102>
  DocumentRoot /home/jtosi/website2
  ServerName www.site2.come
  ErrorLog logs/site2-error.log
  CustomLog logs/site2-access.log common

  <Directory "/home/jtosi/website2">
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

Hitting the first IP in a browser:

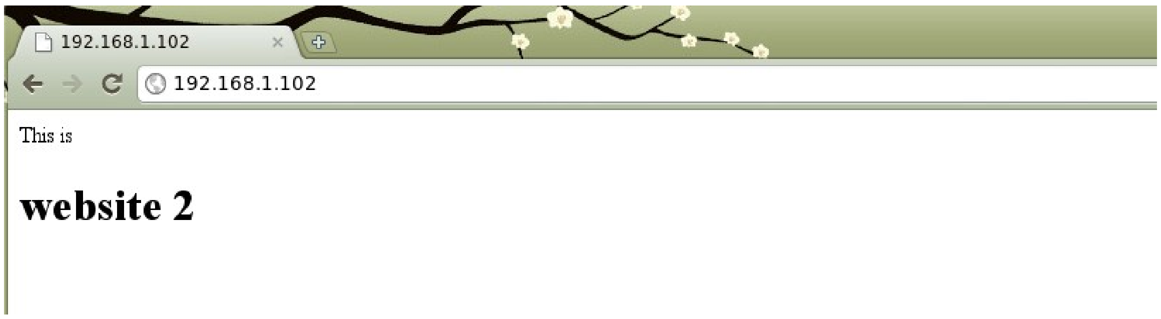


This is

website 1

*Illustration 17: Accessing first virtual host through a browser*

Hitting the second IP:


192.168.1.102

This is

**website 2**

*Illustration 18: Accessing second virtual host through a browser*

The rest of this paper will continue expanding on one VirtualHost.

### 4.4 Configuring Caching

Apache HTTPD can perform caching in two different ways – either by caching to disk the header and body of a response (mod_cache_disk) or by opening a file(s) at httpd start and saving that file handle in memory (mod_file_cache). Mod_cache disk allows you to cache different request types, including ones with query strings whereas mod_file_cache will not allow this since it just is caching filehandles for quicker access. Most importantly, if you are not very careful with mod_file_cache, you can create a very large number of open file handles, resulting in an unresponsive system. Therefore most organizations choose to leverage mod_cache_disk.

In the example below, we do the following configuration with mod_cache_disk:

1) Wrap our configuration in an IfModule clause. We do this because we don't want the directives to attempt to be ran if the module isn't available. This way our configuration will be syntactically correct even if the module isn't available.

2) Define where on disk this cache should reside. **NOTE –** this directory must be writable by the user that apache is running under, otherwise caching will not work.

3) Enable caching of type disk and configure what should be cached. This can be a complete file path (relative to domain root), wildcarded for extension types, or a regular expression. This directive can also be repeated.

4) Put bounds on the size of the cache in terms of the number of directories on disc as well as how large the directory names can be and finally the total size of the cache. If we want deep caches, we want a high value of CacheDirLevels and a low value for length. **NOTE** – the product of CacheDirLevels and CacheDirLength cannot exceed 20.

5) Put upper and lower bounds on the size of the items to cache. Small items (like images that are just 1K) will not give you that much performance gain by caching and will also fill up your cache (think depth of tree)

with small items.  On the flip side, large files that are cached won't boost performance much because the majority of the latency in performance would be do to transmission, not file I/O.  Because of these reasons, reasonable upper and lower bounds should be set.

6) A default time for cached items to expire is set.  This is applied if an expiry header is not created for the items that explicitly sets its time to live.

7) Logging is configured so we can get visibility into our cache hits and misses.

8) Finally, we explicitly disable caching for some volatile content.

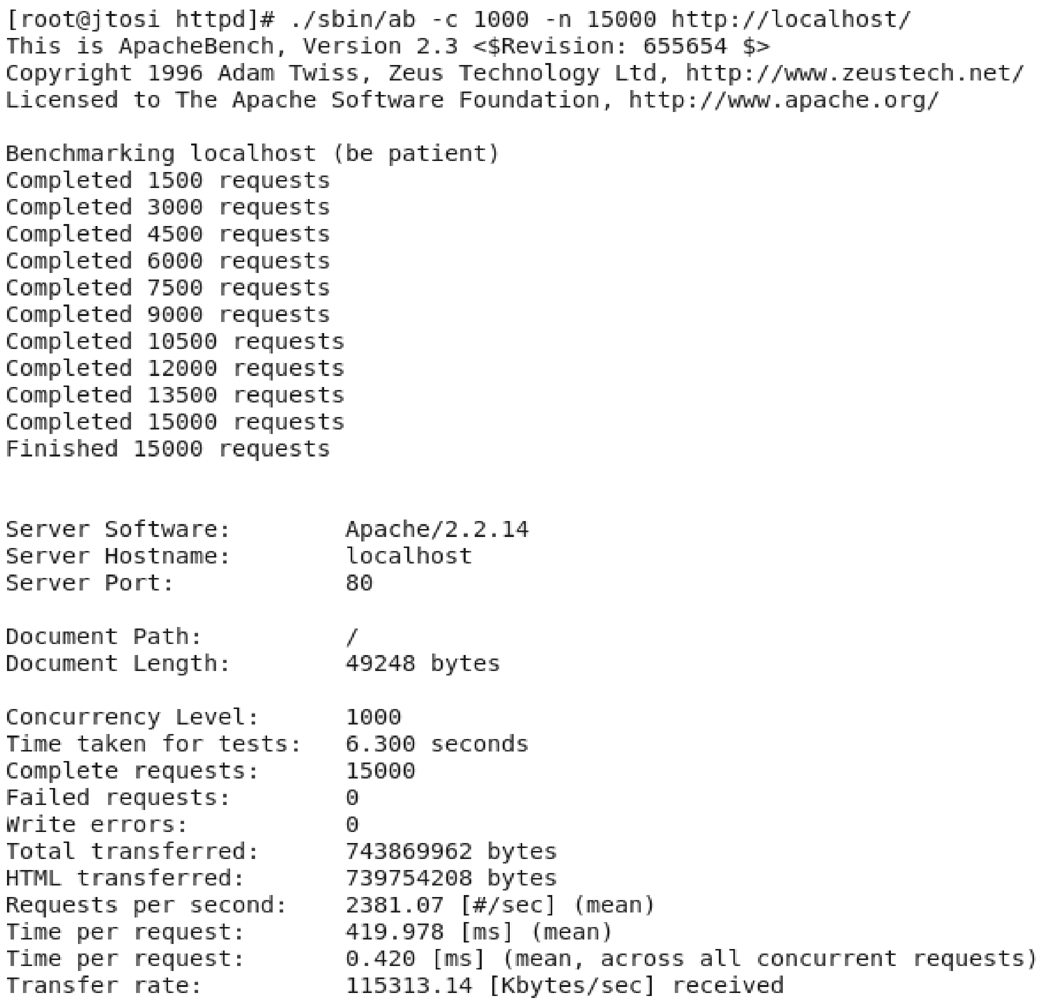
```
##Best Practice – check module defined / available before using
##directives

<IfModule mod_cache_disk>


##Where on disk we are going to store the cache
 CacheRoot /usr/apache/cacheroot


##Enabling caching to disk of all content start at root.  Can also use
##wildcards to narrow down what is cached, i.e. CacheEnable disk /*.css

 CacheEnable disk /


## How many directories to have in cache

 CacheDirLevels 5


##Length of directory names in the cache
 CacheDirLength 3


## Put a cap on how much cache space we want to use in Kbytes

CacheSize 2000000


## Only cache files between 64 and 64K bytes

CacheMinFileSize 64
```

```
CacheMaxFileSize 64000


## If an item doesn’t have an expiry header, expire it from cache after
##1 day

CacheDefaultExpire 86400


##Setup logging so we can see cache hits, misses, and revalidate cache
##items

CustomLog cached-reqeusts.log common env=cache-hit
CustomLog uncached-requests.log common env=cache-miss
CustomLog revalidated-requests.log common env=cache-revalidate
</IfModule>


## We don’t want to cache highly volatile content
CacheDisable http://www.somedomain.com/real-time-stock-market-quotes/
```

To verify our cache, we will use Apache Benchmark (included with EWS at $EWS_HOME/sbin/ab). Apache Benchmark is a relatively robust tool that can handle authentication as well as posting data. However, for this example, we are simply looking at creating a large concurrent load. To do this, we will use ab with 2 flags:

1) -c for the number of concurrent users

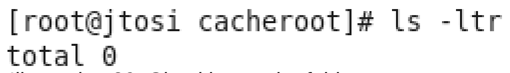
2) -n for the number of requests to run



```
[root@jtosi httpd]# ./sbin/ab -c 1000 -n 15000 http://localhost/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 1500 requests
Completed 3000 requests
Completed 4500 requests
Completed 6000 requests
Completed 7500 requests
Completed 9000 requests
Completed 10500 requests
Completed 12000 requests
Completed 13500 requests
Completed 15000 requests
Finished 15000 requests


Server Software:        Apache/2.2.14
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        49248 bytes

Concurrency Level:      1000
Time taken for tests:   6.300 seconds
Complete requests:      15000
Failed requests:        0
Write errors:           0
Total transferred:      743869962 bytes
HTML transferred:       739754208 bytes
Requests per second:    2381.07 [#/sec] (mean)
Time per request:       419.978 [ms] (mean)
Time per request:       0.420 [ms] (mean, across all concurrent requests)
Transfer rate:          115313.14 [Kbytes/sec] received
```

*Illustration 19: Running Apache Benchmark against site without cache*

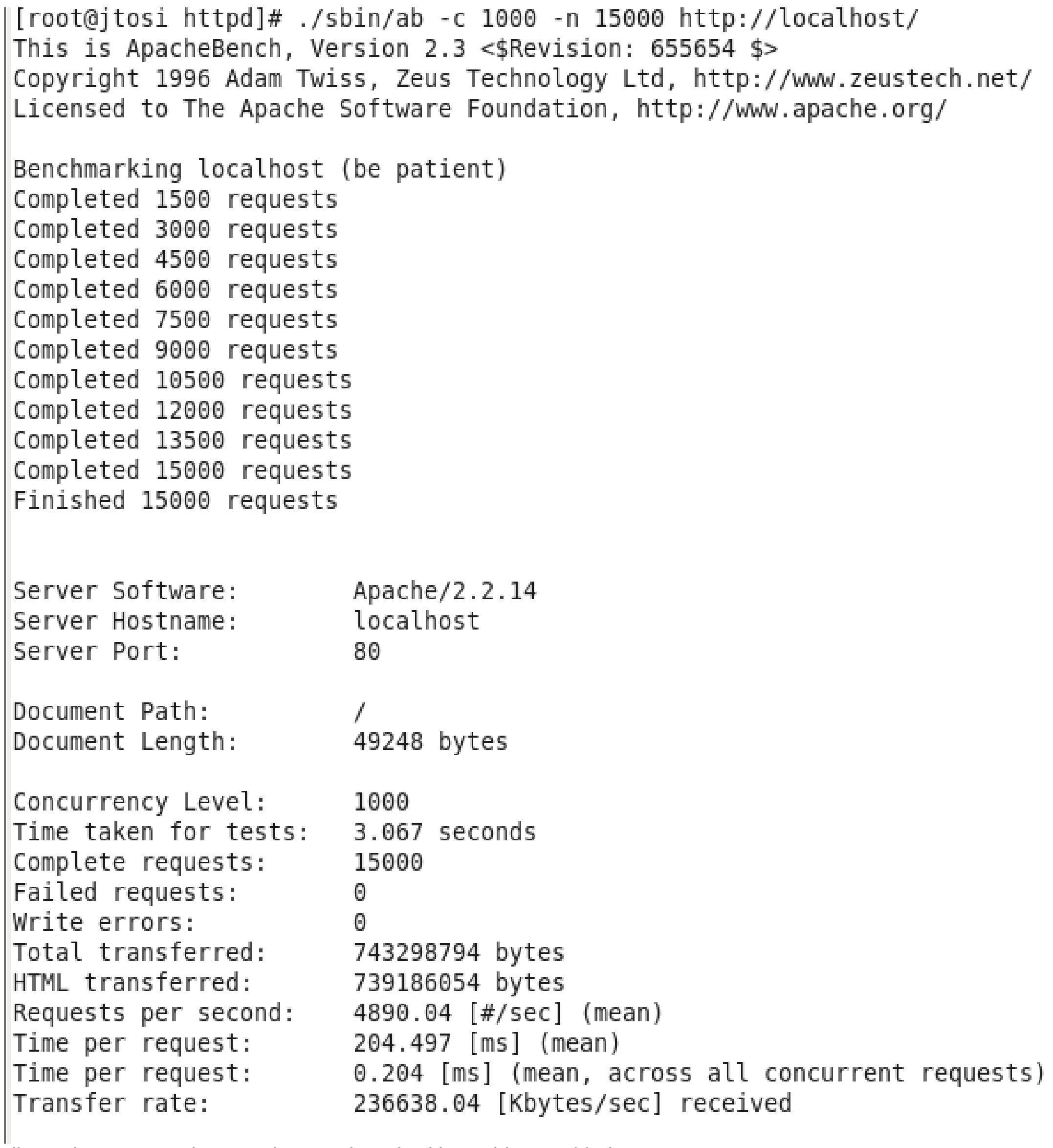
First we run this without caching enabled.  Below is the command with the output (mean times not shown). Note that when you run ab, you have to go against either a complete url to an asset or a complete domain with trailing slash:We can verify that no cache was used by checking our cache folder:



```
[root@jtosi cacheroot]# ls -ltr
total 0
```

*Illustration 20: Checking cache folder*

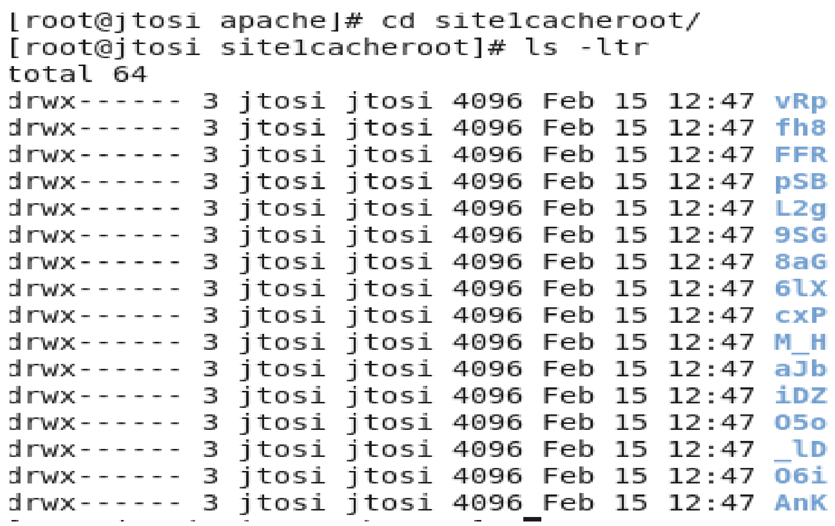
Next, we enable the caching with the directives we showed earlier and rerun the same test:



```
[root@jtosi httpd]# ./sbin/ab -c 1000 -n 15000 http://localhost/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 1500 requests
Completed 3000 requests
Completed 4500 requests
Completed 6000 requests
Completed 7500 requests
Completed 9000 requests
Completed 10500 requests
Completed 12000 requests
Completed 13500 requests
Completed 15000 requests
Finished 15000 requests


Server Software:        Apache/2.2.14
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        49248 bytes

Concurrency Level:      1000
Time taken for tests:   3.067 seconds
Complete requests:      15000
Failed requests:        0
Write errors:           0
Total transferred:      743298794 bytes
HTML transferred:       739186054 bytes
Requests per second:    4890.04 [#/sec] (mean)
Time per request:       204.497 [ms] (mean)
Time per request:       0.204 [ms] (mean, across all concurrent requests)
Transfer rate:          236638.04 [Kbytes/sec] received
```

*Illustration 21: Running Apache Benchmark with Caching enabled*

We verify again our cache directory on disc to make sure items were being cached:



```
[root@jtosi apache]# cd site1cacheroot/
[root@jtosi site1cacheroot]# ls -ltr
total 64
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 vRp
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 fh8
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 FFR
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 pSB
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 L2g
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 9SG
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 8aG
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 6lX
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 cxP
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 M_H
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 aJb
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 iDZ
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 O5o
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 _lD
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 O6i
drwx------ 3 jtosi jtosi 4096 Feb 15 12:47 AnK
```

*Illustration 22: Checking cache folder, verifying that cache is populated*

This confirms that we are caching our content. Also note that in our second load test the same amount of html was transferred, but our performance times were halved while our requests per second was doubled. We are getting significant gains with simple caching.

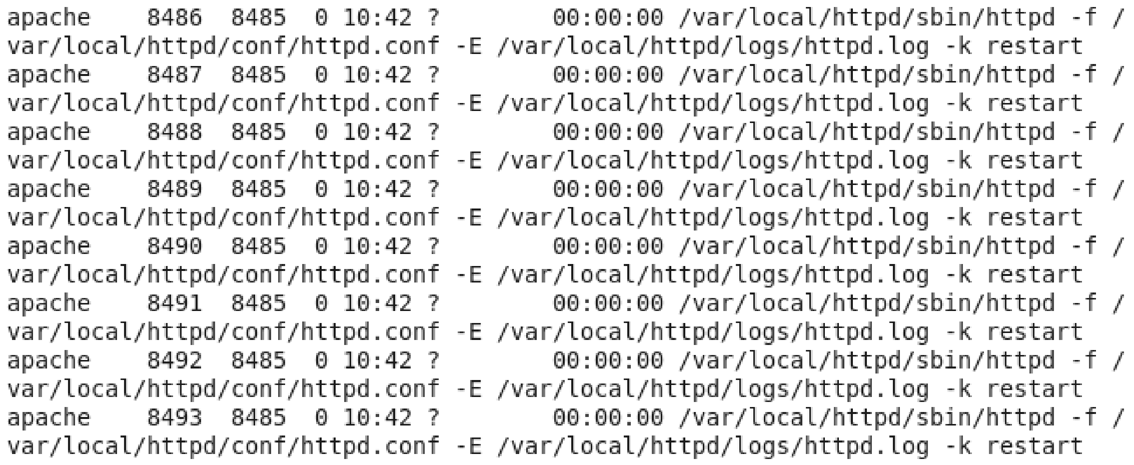
Other items to consider with caching:

- Static content as well as output from dynamic applications can be cache.
- When leveraging virtual includes in your pages or site, be sure to use '*include virtual'* instead of '*include file'* to be able to take advantage of caching (since caching is URL based, not file reference based)
- Highly time-sensitive content should not be cached
- Only GET requests with a response code of 200, 203, 300, 301, or 410 can be cached

## 4.5 Securing your Environment

Apache should be installed and configured only by a root user or with sudo permissions. The thread that apache httpd runs under is defined by the User and Group directives. This user / group combination should not have access to any other system resources. If you do nothing else, be sure to do this.

```
User apache
```



```
apache    8486  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8487  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8488  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8489  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8490  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8491  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8492  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
apache    8493  8485  0 10:42 ?        00:00:00 /var/local/httpd/sbin/httpd -f /
var/local/httpd/conf/httpd.conf -E /var/local/httpd/logs/httpd.log -k restart
```

*Illustration 23: Observing the thread that HTTPD runs in for user and group ownership*

```
Group apache
```

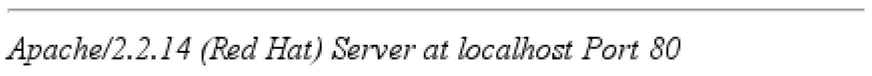
We can verify that the apache user is who the process is running under by looking at the httpd process (ps -ef | grep http)

Other items to do

1) Hide all sensitive information (version, build, etc).  To do this, we use the ServerSignature and ServerTokens directives

```
## Turn off the signature of the server, as seen on default 404 pages

ServerSignature Off

## Limit information in the HTTP response header

ServerTokens Prod
```

Before these values are set:



Apache/2.2.14 (Red Hat) Server at localhost Port 80

*Illustration 24: Exposing Server information*

After the values are set:

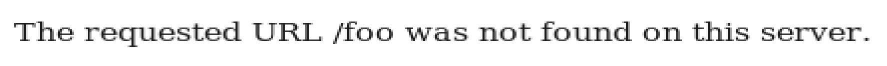

The requested URL /foo was not found on this server.

*Illustration 25: Error page once server information is removed*

**NOTE** – we could also do this test using curl or verify through various plugins in Firefox

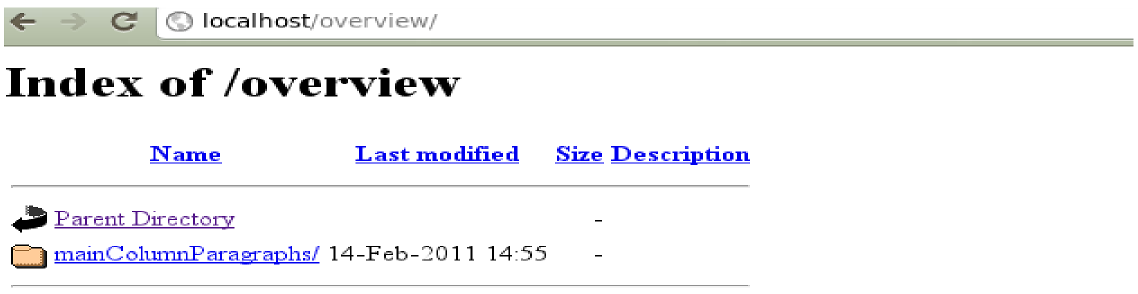
2) To block traversal of files that were not meant to be accessible via the web, we need to have access to all files and directories removed by default.  We then add back in only those as needed. We do this with the Order and Allow directives, applying them both at the default directory level and then in our VirtualHosts:

```
<Directory>
## Default deny all access
 Order Deny, Allow
 Deny from all
 Options None
 AllowOverride None
</Directory>
...
<Directory /website>
## Allow access only to ‘website’ directory and below
 Order Allow, Deny
 Allow from all
</Directory>
```

3) To avoid users of your website / application from being able to retrieve a listing of files in a directory, turn off the Indexes flag using the Option directive as follows:

```
Option -Indexes
```

Before the setting:

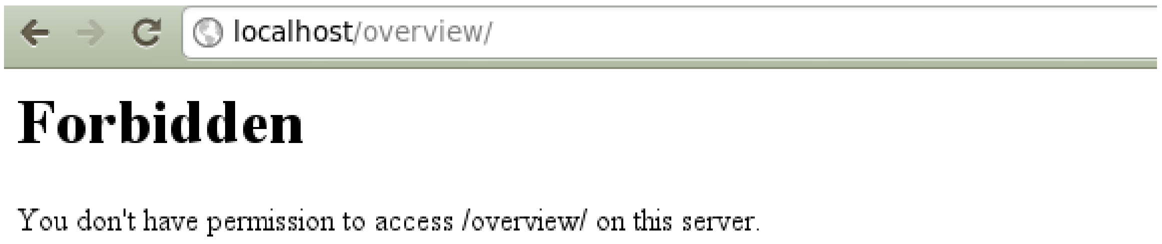

localhost/overview/

Index of /overview

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| mainColumnParagraphs/ | 14-Feb-2011 14:55 | - | |

*Illustration 26: With index listings enabled*

After applying the setting:


localhost/overview/

Forbidden

You don't have permission to access /overview/ on this server.

*Illustration 27: After index listings are removed*

4) Remove all unused modules.  By default, JBoss EWS ships with quite a bit of modules enabled.  Go through these modules and the ones that you aren't using, remove its 'LoadModule' directive.  This will not only limit your exposure to potential security holes / flaws in modules, but also increase your performance by minimizing the footprint of Apache HTTPD.

5) DoS (Denial of Service) attacks are still prevalent in the web.  To reduce the impact any potential DoS / DSoS attack may have on your site, lower the timeout value.  This will free up threads quicker and thereby reduce the impact of 'long pull' attacks like DoS / DdoS.  **NOTE** – this does not remove your risk. You will still want monitoring around your services / requests, solid firewall rules, and possibly consider looking at mod_security – a module for apache that can dynamically apply rules and drop requests that have attack based signatures.

```
##Lower timeout value from default of 5 minutes to 30 seconds

Timeout 30
```

6) If your website / web application allows uploads, be sure to limit the size of these uploads. From a security perspective, the last thing you want is to provide the ability for someone to easily consume a thread / threads as well as bandwidth. Because of these reasons, we need to limit the size of large requests to our site. If you allow file uploads, set this value to the maximum reasonable value for your uploads – for example a maximum size for a resume might be 200Kb. If your site does not allow uploads set this as low as you feel comfortable with.

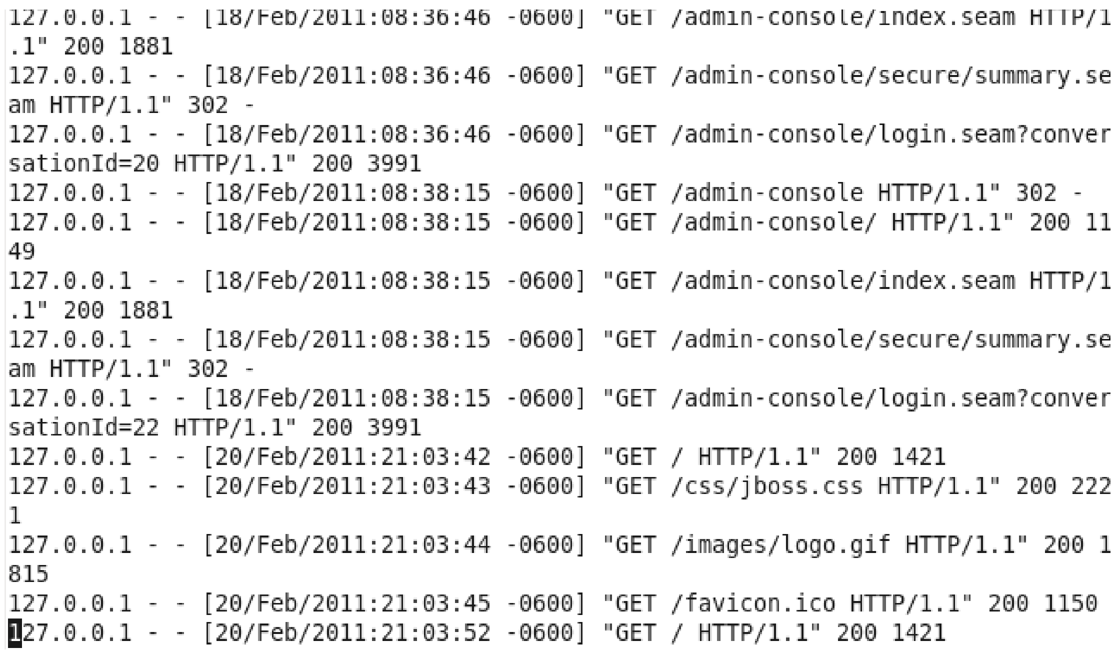
```
##Limit Request Body to ~1Mb

LimitRequestBody 1000000
```

7) Finally, as with all software, be sure to stay up to date with any security patches.

### 4.6 Logging and Debugging

There are three main log files you will interact with when debugging any issues that arise. These files are as follow:

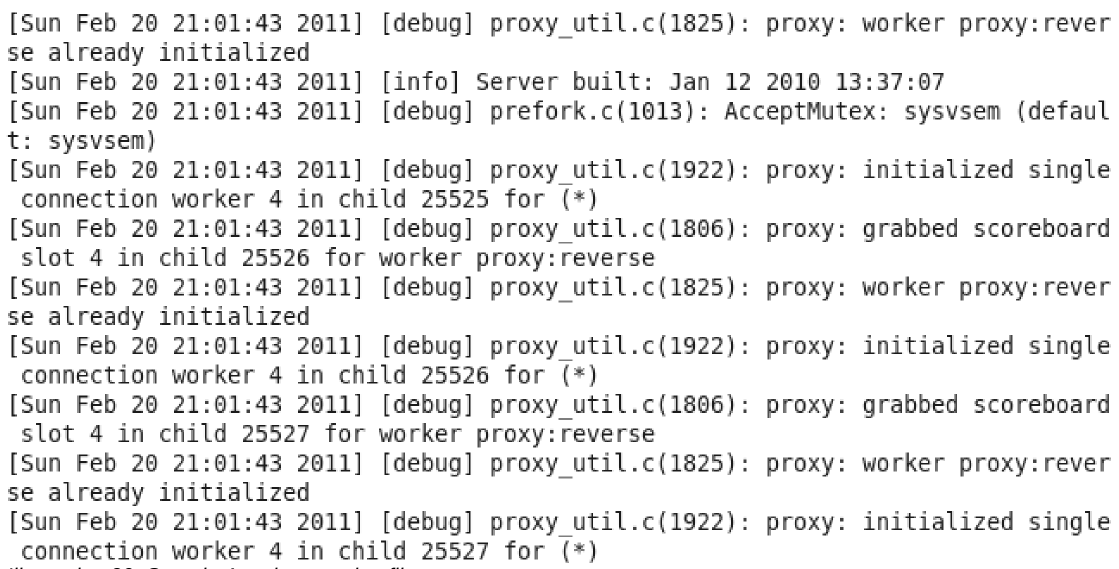
1. Apache Access Log – this log shows all requests coming in to the web server. This is naturally the first log to look at. Here you will verify that the request is actually coming into the web server. This file is typically located at /logs/access.log but each Virtual Host can create its own access log. Because of that, it is best to get the location from the configuration of the domain you are debugging.



```
127.0.0.1 - - [18/Feb/2011:08:36:46 -0600] "GET /admin-console/index.seam HTTP/1
.1" 200 1881
127.0.0.1 - - [18/Feb/2011:08:36:46 -0600] "GET /admin-console/secure/summary.se
am HTTP/1.1" 302 -
127.0.0.1 - - [18/Feb/2011:08:36:46 -0600] "GET /admin-console/login.seam?conver
sationId=20 HTTP/1.1" 200 3991
127.0.0.1 - - [18/Feb/2011:08:38:15 -0600] "GET /admin-console HTTP/1.1" 302 -
127.0.0.1 - - [18/Feb/2011:08:38:15 -0600] "GET /admin-console/ HTTP/1.1" 200 11
49
127.0.0.1 - - [18/Feb/2011:08:38:15 -0600] "GET /admin-console/index.seam HTTP/1
.1" 200 1881
127.0.0.1 - - [18/Feb/2011:08:38:15 -0600] "GET /admin-console/secure/summary.se
am HTTP/1.1" 302 -
127.0.0.1 - - [18/Feb/2011:08:38:15 -0600] "GET /admin-console/login.seam?conver
sationId=22 HTTP/1.1" 200 3991
127.0.0.1 - - [20/Feb/2011:21:03:42 -0600] "GET / HTTP/1.1" 200 1421
127.0.0.1 - - [20/Feb/2011:21:03:43 -0600] "GET /css/jboss.css HTTP/1.1" 200 222
1
127.0.0.1 - - [20/Feb/2011:21:03:44 -0600] "GET /images/logo.gif HTTP/1.1" 200 1
815
127.0.0.1 - - [20/Feb/2011:21:03:45 -0600] "GET /favicon.ico HTTP/1.1" 200 1150
127.0.0.1 - - [20/Feb/2011:21:03:52 -0600] "GET / HTTP/1.1" 200 1421
```

*Illustration 28: Sample Apache access log*

2. Apache Error Log – this log shows all errors that the web server is encountering. Errors typically are for artifacts (images, stylesheets, html pages) that could not be found. You will see other errors in here though if the web server is running low on resources which could be a sign of long-running threads or requests. Finally, you will also see more verbose debugging information here; for example information around ajp connections. This file is typically located at /logs/error.log but each Virtual Host can create its own access log. Because of that, it is best to get the location from the configuration of the domain you are debugging.

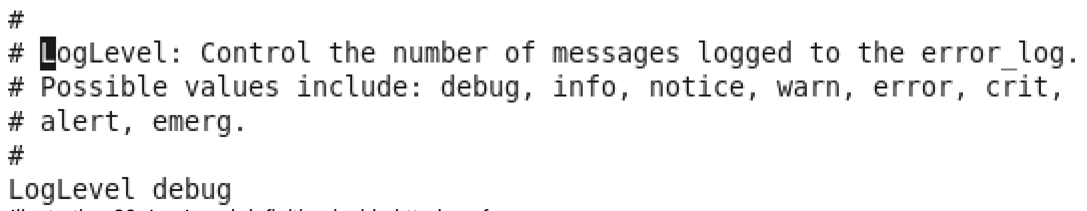


```
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1825): proxy: worker proxy:rever
se already initialized
[Sun Feb 20 21:01:43 2011] [info] Server built: Jan 12 2010 13:37:07
[Sun Feb 20 21:01:43 2011] [debug] prefork.c(1013): AcceptMutex: sysvsem (defaul
t: sysvsem)
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1922): proxy: initialized single
 connection worker 4 in child 25525 for (*)
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1806): proxy: grabbed scoreboard
 slot 4 in child 25526 for worker proxy:reverse
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1825): proxy: worker proxy:rever
se already initialized
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1922): proxy: initialized single
 connection worker 4 in child 25526 for (*)
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1806): proxy: grabbed scoreboard
 slot 4 in child 25527 for worker proxy:reverse
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1825): proxy: worker proxy:rever
se already initialized
[Sun Feb 20 21:01:43 2011] [debug] proxy_util.c(1922): proxy: initialized single
 connection worker 4 in child 25527 for (*)
```

*Illustration 29: Sample Apache error log file*

3. Mod_jk / mod_cluster / mod_proxy.log – This log file will show all of the requests and responses over the connectors to JBoss including keepalive information or cluster information in the case of mod_cluster. Look in this log to verify that requests are being sent to JBoss appropriately as well as to troubleshoot the expected translation and response from JBoss. This file is typically located at /logs/mod_*connector*.log but each Virtual Host can create its own access log. Because of that, it is best to get the location from the configuration of the domain you are debugging.

Inside the httpd.config file, you can set the logging level of Apache using the LogLevel directive. The examples above are with LogLevel set to Debug to show more verbose samples.



```
#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel debug
```

*Illustration 30: LogLevel definition inside httpd.conf*

The format of your log files is configurable. For the sake of this whitepaper, we will stick with the default logging configuration, but more information can be found here - [http://httpd.apache.org/docs/current/logs.html](http://httpd.apache.org/docs/current/logs.html)

Finally, be sure to rotate your log files.  As log file growth is directly dependent upon LogLevel as well as site requests, be sure to tune your scenario accordingly.  In general, you always want to avoid log files getting to large.  Most organizations are fine with a nightly log rotation and that is what the below example will cover (just note that your needs may vary).

The easiest way to set up nightly log rotation is to use the rotatelogs script that comes with Apache (located at *EWS_HOME*/sbin/rotatelogs.  To rotate our access log after 24 hours, we would add the following line:

```
CustomLog "|sbin/rotatelogs /var/local/httpd/logs/access.log 86400" common
```

There are a few items above that are of interest.  Lets look at them individually

`CustomLog  -` We are using the CustomLog directive to set up a custom logging structure.  At the end of this line, we can see the 'common' parameter.  This is telling CustomLog to use the common logging format and apply that to our CustomLog.

`|(Pipe Character) -` We start off the CustomLog by entering the pipe character.  This tells apache to use piped logging.  This way Apache does not keep a filehandle lock on the log file.  If we didn't use piped logging, we could still rotate the files, but we would need to stop apache to release the lock before we could rotate.

`sbin/rotatelogs -` This is the relative path to the rotatelogs executable

`/var/local/httpd/logs/access.log -` This is the path to the log file we want to rotate.  Note that for each log you want to rotate, you will want to add a CustomLog.

`86400-` This is the number of seconds, offset from webserver start, when the log should be rotated.  This could also be defined in filesize, i.e. 5M to rotate after 5Mb of logging.

`common -` Finally, we define that this log should use the common definition for log message formats.

Again, the above information shows the best approach for setting up log rotation.  In the example above, we are rotating the access log file every 24 hours.

### 4.7 Connecting to JBoss

There are three main ways to connecting Apache HTTPD to JBoss EAP.

**1) Using mod_proxy** – mod_proxy acts like just that – a proxy to your application server.  With mod_proxy, you simply tell apache that for certain request patterns, the request will be responded to by the application server.  mod_proxy can act as a regular forward proxy or as a reverse proxy.  Mod_proxy is simple to configure, supports SSL, but lacks a considerable amount of control that you will get with mod_jk and mod_cluster around how you will forward requests as well as information around application server availability.

**2) Using mod_jk –** mod_jk is the jakarta connector from Apache.  Development on it has slowed down recently.  Mod_jk is nicer than mod_proxy as it allows a tighter integration with the application server as well as a small communication channel (leveraging AJP) and more fine-grained control of how requests are forwarded (i.e. mod_proxy forwards based on URLs, so typically a whole directory would be proxied.  With mod_jk, you could say for a given directory with a given extension type, forward those requests.  This works well with virtual includes). However, mod_jk does require a slightly more involved configuration and does not support an encrypted communication channel between web server and application server.

**3) Using mod_cluster –** mod_cluster is the next generation connector from JBoss. Like the others, it serves to be the bridge between web server and application server.  Like mod_proxy, it can work with an encrypted channel and has a simple configuration.  Like mod_jk, mod_cluster also allows fine-grained control of requests and configuration parameters.  **Unlike mod_jk and mod_proxy, mod_cluster allows an application server to grow / shrink dynamically without needs for reconfiguration.**  Additionally, mod_cluster leverages a second communication channel which allows it to have more robust load balancing options as well as being more than just application server aware – it is also application aware meaning that it can detect scenarios where an application server is up but an application is not available and thereby not route traffic to it.

Now that we have provided a background on the 3 ways to connect Apache HTTPD to JBoss EAP, we will go through an exercise configuring these connectors.

In each example, we will do the following:

1) Send requests to *websiteDomain/theboss* to the admin-console of a cluster of JBoss EAP instances

2) We will configure these to communicate using AJP – a performant, binary format used for communication between a web server and an application server

3) We will configure it such that the second instance will get 3 times the traffic as the first

4) Finally, we will configure our connector to use sticky session.  This way once a user has established session on one application server instance, they will continue to go to that one as opposed to being load balanced between the two.  This is ideal for a situation such as this where we are logging into an administration console.

**Setting up mod_proxy**

All of the changes in this example will be made to the httpd.conf file located at *EWS_HOME*/conf.
mod_proxy is dependent upon the following modules, so be sure to have them loaded:

mod_proxy

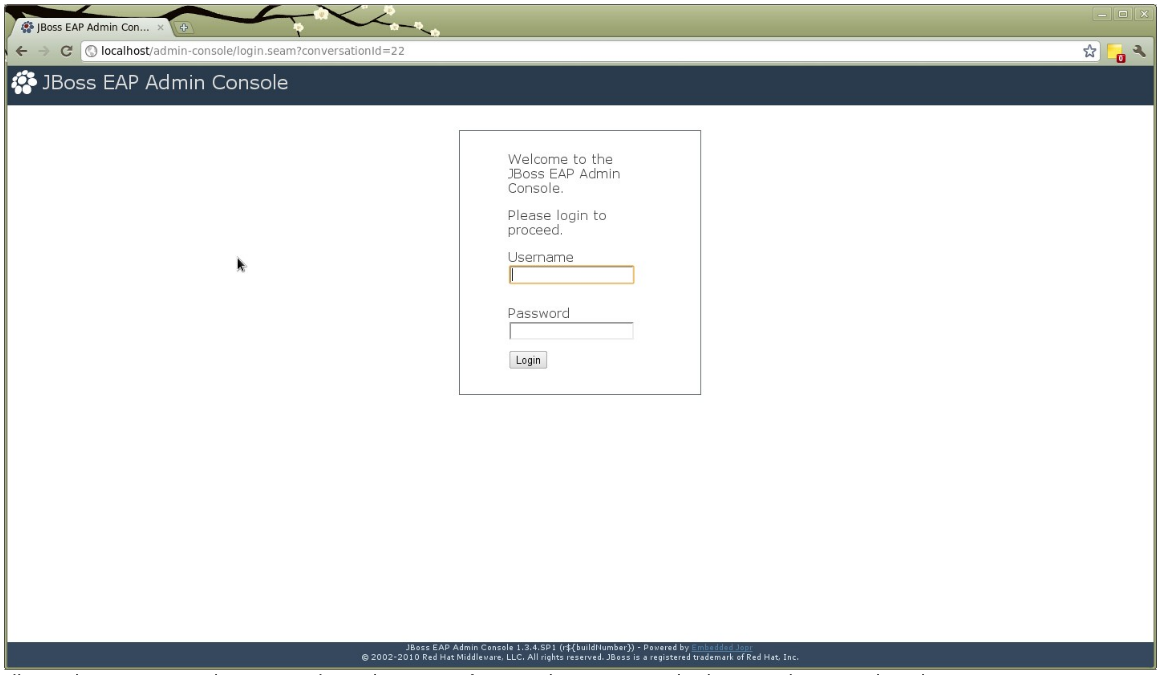
mod_proxy_balancer

mod_proxy_ajp / http / ftp (depending on your connecting protocol)

Because we only want this forwarding applied to one domain, we will put these directives inside the VirtualHost definition.  If we wanted the proxy applied globally, these directives can stand alone in the configuration file.

```
<VirtualHost>

...

##Identify the proxy

<Proxy balancer://jbossCluster>

##Setup the members for this proxy. 2 members, using ajp protocol (could also

##use http.  ‘loadfactor’ tells the balancer that the second instance should get

##3 times the load of the first.

##Use ProxySet to establish sticky session

##ProxyPass defines the URL pattern to match to the balancer

BalancerMember ajp://127.0.0.1:8009 loadfactor=1

BalancerMember ajp://127.0.0.1:8109 loadfactor=3

ProxySet stickysession=JSESSIONID

</Proxy>

ProxyPass / balancer://jbossCluster

ProxyPassReverse / balancer://jbossCluster

...

</VirtualHost>
```

Here is what we see in a browser (note running over port 80)



Welcome to the JBoss EAP Admin Console.

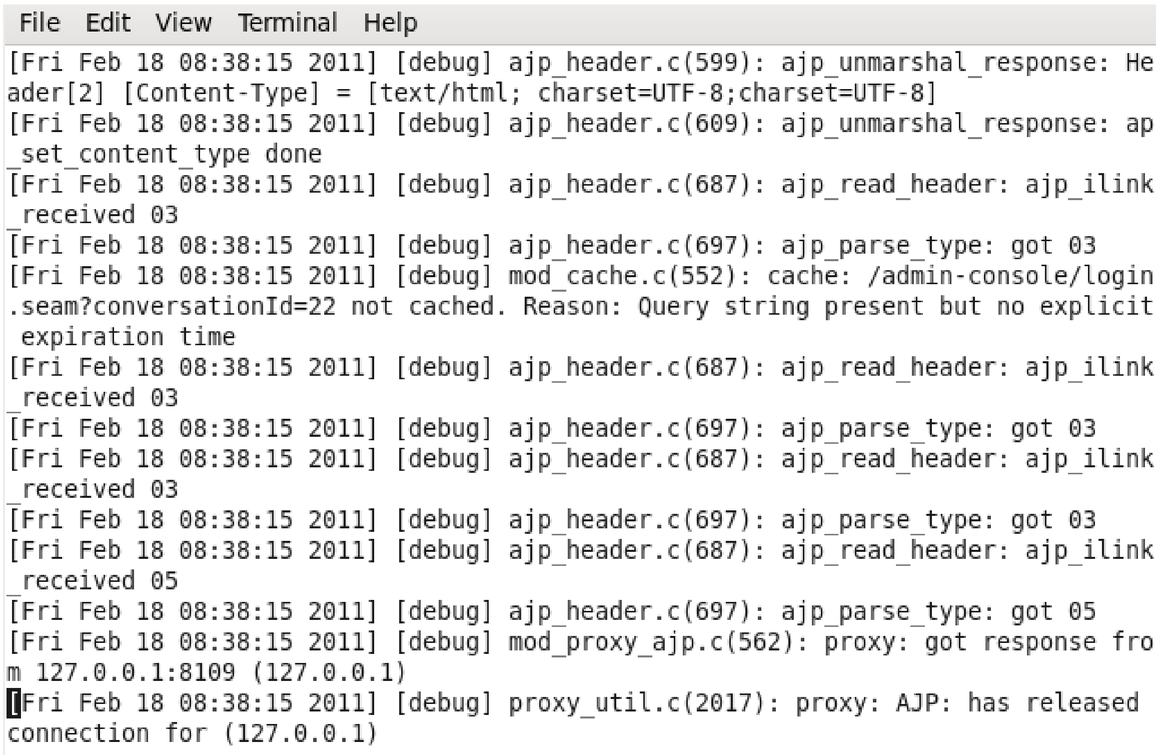
Please login to proceed.

JBoss EAP Admin Console 1.3.4.SP1 (r${buildNumber}) - Powered by Embedded Jopr
© 2002-2010 Red Hat Middleware, LLC. All rights reserved. JBoss is a registered trademark of Red Hat, Inc.

*Illustration 31: Accessing JBoss through HTTPD front-end proxy, note the browser is accessing site over port 80*

We could continue using the application as normal, all over port 80.

If you want to see the AJP communication, set your `LogLevel` to `Debug` and view the error log file:



```
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(599): ajp_unmarshal_response: He
ader[2] [Content-Type] = [text/html; charset=UTF-8;charset=UTF-8]
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(609): ajp_unmarshal_response: ap
_set_content_type done
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(687): ajp_read_header: ajp_ilink
_received 03
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(697): ajp_parse_type: got 03
[Fri Feb 18 08:38:15 2011] [debug] mod_cache.c(552): cache: /admin-console/login
.seam?conversationId=22 not cached. Reason: Query string present but no explicit
 expiration time
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(687): ajp_read_header: ajp_ilink
_received 03
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(697): ajp_parse_type: got 03
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(687): ajp_read_header: ajp_ilink
_received 03
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(697): ajp_parse_type: got 03
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(687): ajp_read_header: ajp_ilink
_received 05
[Fri Feb 18 08:38:15 2011] [debug] ajp_header.c(697): ajp_parse_type: got 05
[Fri Feb 18 08:38:15 2011] [debug] mod_proxy_ajp.c(562): proxy: got response fro
m 127.0.0.1:8109 (127.0.0.1)
[Fri Feb 18 08:38:15 2011] [debug] proxy_util.c(2017): proxy: AJP: has released
connection for (127.0.0.1)
```

*Illustration 32: Log files for information showing AJP connectivity*

**Setting up mod_jk**

We will be doing the same example as we did with mod_proxy, except this time using mod_jk.   mod_jk is only dependent on the mod_jk module to be loaded.  However, the configuration requires more directives and files.  The section below covers the changes needed to send requests to a clustered JBoss EAP instance with a weighted load balancer, leveraging sticky sessions (same as with mod_proxy exercise).

*Modifications to httpd.conf*

EWS ships with the mod_jk module but not explicitly included in the httpd.conf file.  The first thing we will need to do is add the mod_jk module to be loaded.  We do this by adding the following line at the end of the LoadModule block:

```
LoadModule jk_module /var/local/httpd/modules/mod_jk.so
```

Next, we need to define the location where our workers (application server instances that will respond to requests) are defined at.  The common practice for this is to define this information in a file called workers.properties.  We add this directive with the following:

```
JkWorkersFile /var/local/httpd/conf/workers.properties
```

Next we define the location of a shared memory file.  This is used by the load balancer internal to mod_jk for knowing where requests are at and where to direct traffic.

```
JkShmFile /var/local/httpd/mod_jk.shm
```

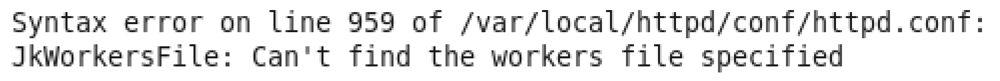
Now we define where our logs for mod_jk should go be written at, what logging level we want (error, debug, info), and the format of the logs.

```
JkLogFile /var/local/httpd/logs/mod_jk.log

JkLogLevel info

JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
```

Finally, similar to what we did with mod_proxy, we need to define which URLs are mapped over.  We do this by defining the location for a URI mapping file (we look at the values of this file next). Since we only want this to affect one domain, we apply the following change inside our VirtualHost.

```
JkMountFile /var/local/httpd/conf/uriworkermap.properties
```

If we were to restart apache now, it would fail as the workers.properties file has not been created yet.  Try it out, you should see something similar in the httpd log



```
Syntax error on line 959 of /var/local/httpd/conf/httpd.conf:
JkWorkersFile: Can't find the workers file specified
```

Before we fix this error, lets address our uriworkermap.properties file and then we will create our workers.properties file.

The `uriworkermap.properties` file does not exist and you will need to create this file at the location you identified in the last step above.  This file is very straightforward.  We simply mount the servlet context that we want to be added as defined by a group (which we identify in workers.properties). Note that the servlet context will match the URL Apache HTTPD forwards over to JBoss.  Here is the content of our uriworkermap.properties file:

```
# Mount the Servlet context to the ajp13 worker
```

```
/admin-console=examplebalancer

/admin-console/*=examplebalancer
```

Finally, we identify our application servers in our workers.properties file.

Just like our uriworkermap.properties file, the workers.properties file does not exist and will need to be created at the location you defined in httpd.conf.  The following exercise walks through creating a load balanced connection to 2 JBoss servers.

The first item to create in our workers.properties file is the listing of workers that will be available.  We do this by using the workers.list directive like the following (notice how it matches our uriworkermap.properties definition):

```
worker.list=examplebalancer
```

With our worker list defined, we now go about identifying the servers that will make up our load balanced group.  For the rest of the configuration, you will notice that the pattern is ‘worker.’ + *workerName* + property where workerName is how we are identifying the worker.  There are quite a few configurable values here.  This example will walk through the most critical values to set.  These include the hostname where the JBoss instance resides, the type of worker this is, what the communication port should be, the load balancer weight, and finally how mod_jk should check to ensure that the identified application server is up and responding (CPing).  Lets look at the configuration:

```
# Define Node1

# modify the host as your host IP or DNS name.

worker.node1.port=8009

worker.node1.host=localhost

worker.node1.type=ajp13

worker.node1.ping_mode=A

worker.node1.lbfactor=1
```

In the example above, we are connecting to a JBoss instance on our local machine.  We will be communicating using the ajp1.3 protocol, and JBoss is handling AJP on port 8009.  We set a load balance factor of 1 (which makes more sense once we add our second node).  Final, we set our ping_mode to A.  This means that we want mod_jk to check all routes to verify that JBoss is up.  This includes:

Check once after the initial connection to JBoss

Before each request is sent to JBoss

At a regular interval (heartbeat)

We set our second node (using the same parameters but updated for our second instance):

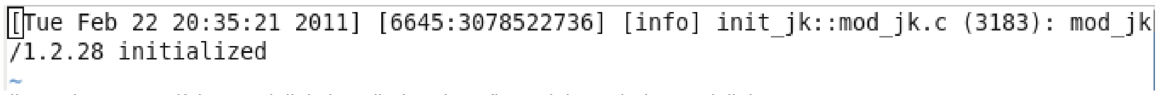
```
# Define Node2

# modify the host as your host IP or DNS name.

worker.node2.port=8109

worker.node2.host=localhost

worker.node2.type=ajp13

worker.node2.ping_mode=A

worker.node2.lbfactor=3
```

With our JBoss servers identified and configured, we wrap up the configuration by defining the loadbalancer aspects for the group and bind it to 'examplebalancer' which is what is defined in our uriworkermap.properties.

In the following example, we define the type of this worker as a loadbalancer(remember, named example balancer), what workers make up this load balancer, and we enable sticky sessions.

```
# Load-balancing behaviour

worker.examplebalancer.type=lb

worker.examplebalancer.balance_workers=node1,node2

worker.examplebalancer.sticky_session=1
```

Now that we have our uriworkermap.properties and worker.properties files defined, we restart Apache HTTPD and test it out.  First thing we check after the restart is our mod_jk.log.  The following shows us that the mappings have all lined up and the backend servers are found and available:

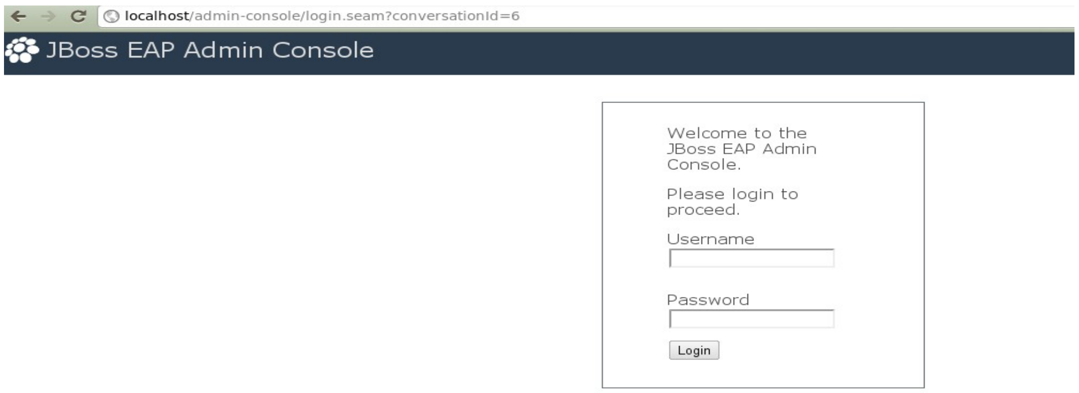


```
[Tue Feb 22 20:35:21 2011] [6645:3078522736] [info] init_jk::mod_jk.c (3183): mod_jk
/1.2.28 initialized
~
```

*Illustration 33: Verifying mod_jk is installed and configured through the mod_jk.log*

Any errors in here would tell us that a backend server isn't available or that we possibly have a type in our configurations.

And finally we verify that we can connect to our JBoss application server through Apache by testing through a browser over port 80:



Welcome to the JBoss EAP Admin Console.

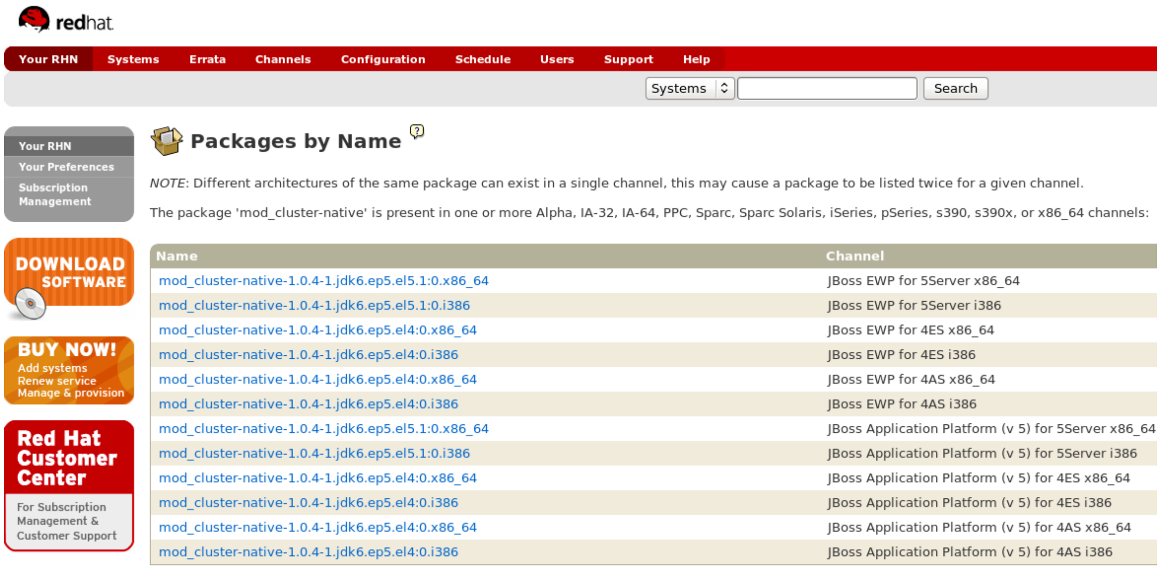
Please login to proceed.

*Illustration 34: Accessing JBoss through Apache HTTPD using mod_jk*

**Setting up mod_cluster**

Mod_cluster requires modules to be added to Apache HTTPD as well as deploying a new application to your JBoss application instance. The following steps walk through getting these additional components, installing, and configuring them.

Similar to how you downloaded EWS, download the mod_cluster modules from access.redhat.com (mod_cluster-native)



*NOTE*: Different architectures of the same package can exist in a single channel, this may cause a package to be listed twice for a given channel.

The package 'mod_cluster-native' is present in one or more Alpha, IA-32, IA-64, PPC, Sparc, Sparc Solaris, iSeries, pSeries, s390, s390x, or x86_64 channels:

| Name | Channel |
|---|---|
| mod_cluster-native-1.0.4-1.jdk6.ep5.el5.1:0.x86_64 | JBoss EWP for 5Server x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el5.1:0.i386 | JBoss EWP for 5Server i386 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.x86_64 | JBoss EWP for 4ES x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.i386 | JBoss EWP for 4ES i386 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.x86_64 | JBoss EWP for 4AS x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.i386 | JBoss EWP for 4AS i386 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el5.1:0.x86_64 | JBoss Application Platform (v 5) for 5Server x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el5.1:0.i386 | JBoss Application Platform (v 5) for 5Server i386 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.x86_64 | JBoss Application Platform (v 5) for 4ES x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.i386 | JBoss Application Platform (v 5) for 4ES i386 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.x86_64 | JBoss Application Platform (v 5) for 4AS x86_64 |
| mod_cluster-native-1.0.4-1.jdk6.ep5.el4:0.i386 | JBoss Application Platform (v 5) for 4AS i386 |

*Illustration 35: Downloading mod_cluster from rhn*

Depending on your installation route for Apache HTTPD (rpm or from EWS zip) – downloading this rpm may update your httpd modules instance for you.  What you need to make sure you have is the following modules inside your httpd modules directory:

mod_proxy.so

mod_proxy_ajp.so

mod_slotmem.so

mod_manager.so

mod_proxy_cluster.so

mod_advertise.so

Once these module are in the modules directory, we need to add the modules to our httpd.conf so that they are loaded when Apache starts up.

```
LoadModule proxy_module /var/local/httpd/modules/mod_proxy.so

LoadModule proxy_ajp_module /var/local/httpd/modules/mod_proxy_ajp.so

LoadModule slotmem_module /var/local/httpd/modules/mod_slotmem.so

LoadModule manager_module /var/local/httpd/modules/mod_manager.so

LoadModule proxy_cluster_module /var/local/httpd/modules/mod_proxy_cluster.so

LoadModule advertise_module /var/local/httpd/modules/mod_advertise.so
```

With the modules added, we now show the minimal configuration needed to get mod_cluster running. Notice how different this is as compared to mod_proxy and mod_jk – we are not defining any workers or contexts.

In the following example, we are setting up a new VirtualHost.  The key here is to create a new socket for communication with the backend servers.  We restrict access to a certain IP range (in this case, just local). Finally, we name our ManagerBalancer and tell it to advertise out, looking for backend connections.

```
Listen 127.0.0.1:6666

<VirtualHost 127.0.0.1:6666>

<Location />

      Order deny,allow

      Deny from all

      Allow from 127.0.0.

   </Location>
```

```
 KeepAliveTimeout 60

 MaxKeepAliveRequests 0


 ManagerBalancerName mycluster

 ServerAdvertise On

</VirtualHost>
```

We do one more step – that is setup the mod_cluster manager so that we can verify mod_cluster is functioning properly.  We do this by adding the following to our httpd.conf:

```
<Location /mod_cluster-manager>

SetHandler mod_cluster-manager

Order deny,allow

Deny from all

Allow from 127.0.0.1

</Location>
```

The above section simply restricts access to the mod_cluster-manager URI to only come from localhost.  It defers handling of the requests to the mod_cluster_manager module.

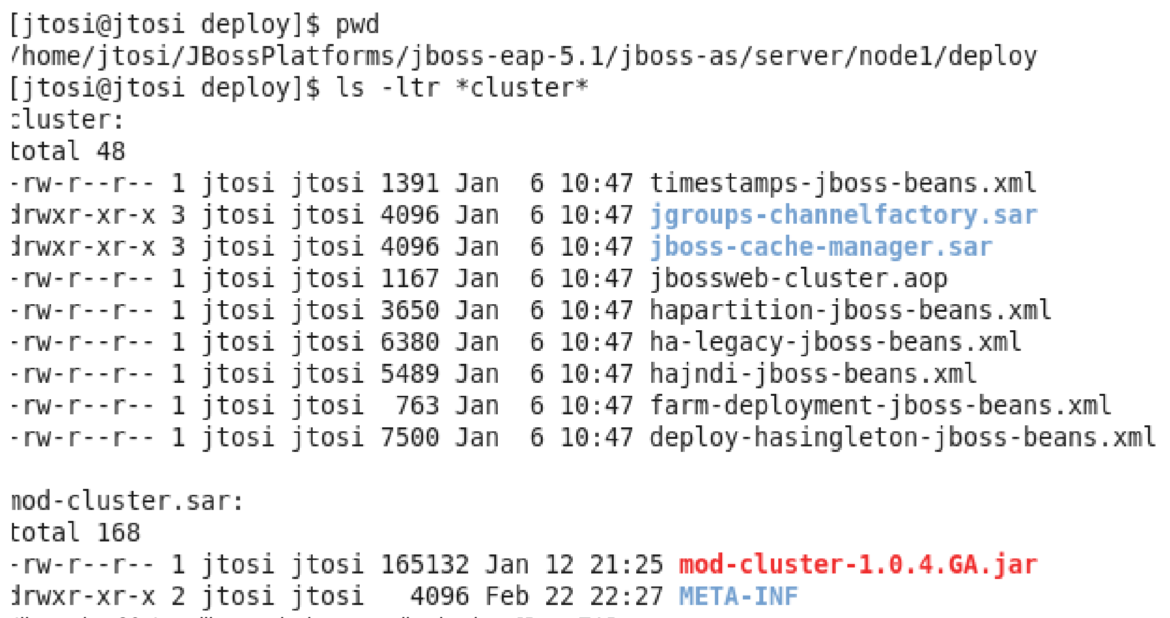
No other httpd configuration is necessary.  Remember – mod_cluster dynamically (via the advertise channel) finds JBoss application servers as well as their context.

Next, we need to make some changes to our JBoss application server instances. These changes are necessary to that the instances know to broadcast to mod_cluster.  The changes are relatively simple and are easily repeatable.  A best practice for dynamically adding instances to a cluster and having it picked up by mod_cluster would be to simply have these changes as part of your core JBoss instance that you create (copy) others from.

**Setting up JBoss for mod_cluster**

In the following steps, we will be making the necessary changes to a JBoss instance to make mod_cluster aware of it and vice versa. Only one server is shown, for subsequent instances, the changes are the same.

First copy the mod_cluster.sar file from the earlier download to the ‘deploy’ directory of your JBoss instance. In the example below, an instance named ‘node1’ was created so we are copying mod_cluster.sar to *Jboss_HOME/*jboss_as/server/node1/deploy.  This application is what will do the communication with Apache HTTPD.

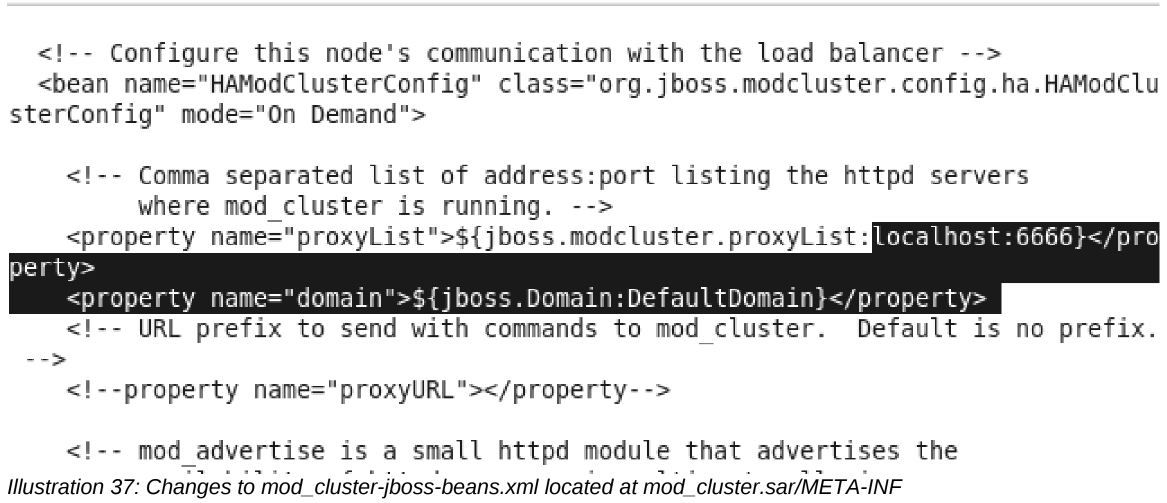


```
[jtosi@jtosi deploy]$ pwd
/home/jtosi/JBossPlatforms/jboss-eap-5.1/jboss-as/server/node1/deploy
[jtosi@jtosi deploy]$ ls -ltr *cluster*
cluster:
total 48
-rw-r--r-- 1 jtosi jtosi 1391 Jan  6 10:47 timestamps-jboss-beans.xml
drwxr-xr-x 3 jtosi jtosi 4096 Jan  6 10:47 jgroups-channelfactory.sar
drwxr-xr-x 3 jtosi jtosi 4096 Jan  6 10:47 jboss-cache-manager.sar
-rw-r--r-- 1 jtosi jtosi 1167 Jan  6 10:47 jbossweb-cluster.aop
-rw-r--r-- 1 jtosi jtosi 3650 Jan  6 10:47 hapartition-jboss-beans.xml
-rw-r--r-- 1 jtosi jtosi 6380 Jan  6 10:47 ha-legacy-jboss-beans.xml
-rw-r--r-- 1 jtosi jtosi 5489 Jan  6 10:47 hajndi-jboss-beans.xml
-rw-r--r-- 1 jtosi jtosi  763 Jan  6 10:47 farm-deployment-jboss-beans.xml
-rw-r--r-- 1 jtosi jtosi 7500 Jan  6 10:47 deploy-hasingleton-jboss-beans.xml

mod-cluster.sar:
total 168
-rw-r--r-- 1 jtosi jtosi 165132 Jan 12 21:25 mod-cluster-1.0.4.GA.jar
drwxr-xr-x 2 jtosi jtosi   4096 Feb 22 22:27 META-INF
```

*Illustration 36: Installing mod_cluster application into JBoss EAP*

Now that we have the application installed, we need to do some configuration.  The first configuration we will do is located at mod_cluster.sar/META-INF/mod_cluster-jboss-beans.xml.  In this file we will be completing the information for the location to where mod_cluster is running (the VirtualHost we set up in httpd.conf a few steps back).  We will also add some domain information.

Edit this file, and add the following to achieve this (changes highlighted):



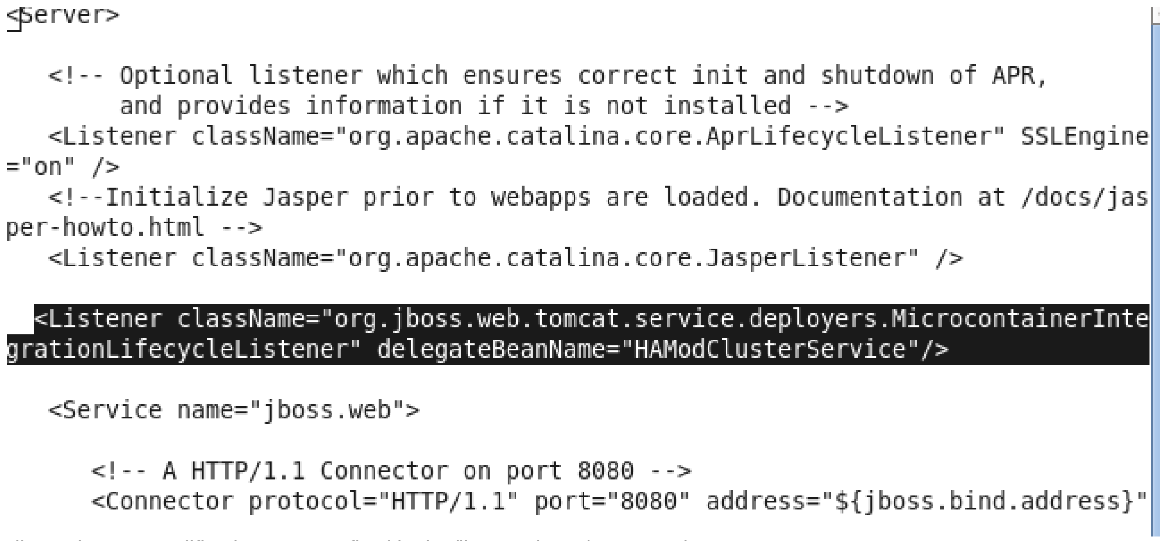
```
  <!-- Configure this node's communication with the load balancer -->
  <bean name="HAModClusterConfig" class="org.jboss.modcluster.config.ha.HAModClu
sterConfig" mode="On Demand">

    <!-- Comma separated list of address:port listing the httpd servers
         where mod_cluster is running. -->
    <property name="proxyList">${jboss.modcluster.proxyList:localhost:6666}</pro
perty>
    <property name="domain">${jboss.Domain:DefaultDomain}</property>
    <!-- URL prefix to send with commands to mod_cluster.  Default is no prefix.
 -->
    <!--property name="proxyURL"></property-->

    <!-- mod_advertise is a small httpd module that advertises the
```

*Illustration 37: Changes to mod_cluster-jboss-beans.xml located at mod_cluster.sar/META-INF*

Save your changes and close this file.

The next file we need to modify is located at node1/deploy/jbossweb.sar/server.xml

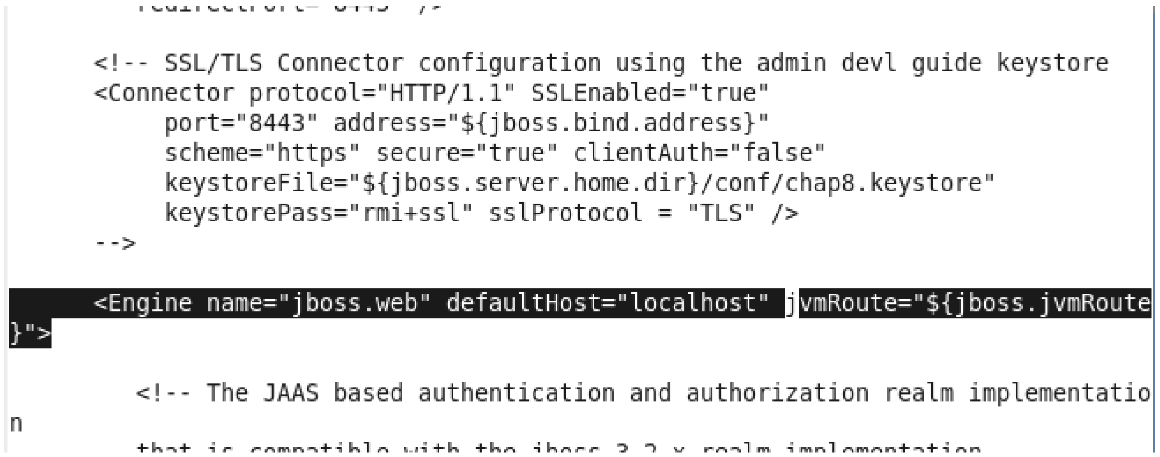
In this file, we need to make 2 changes.  The first change is to add another listener to our web instance. This will make our mod_cluster service initialized when JBoss starts up.  Your change will look like the highlighted area below:



```
<Server>

   <!-- Optional listener which ensures correct init and shutdown of APR,
        and provides information if it is not installed -->
   <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine
="on" />
   <!--Initialize Jasper prior to webapps are loaded. Documentation at /docs/jas
per-howto.html -->
   <Listener className="org.apache.catalina.core.JasperListener" />

  <Listener className="org.jboss.web.tomcat.service.deployers.MicrocontainerInte
grationLifecycleListener" delegateBeanName="HAModClusterService"/>

   <Service name="jboss.web">

      <!-- A HTTP/1.1 Connector on port 8080 -->
      <Connector protocol="HTTP/1.1" port="8080" address="${jboss.bind.address}"
```

*Illustration 38: Modifications to <config>/deploy/jbossweb.sar/server.xml*

And the second change we need to add is a jvmRoute. This is needed for identifying the server to apache and mod_cluster for load balancing as well as for sticky sessions.

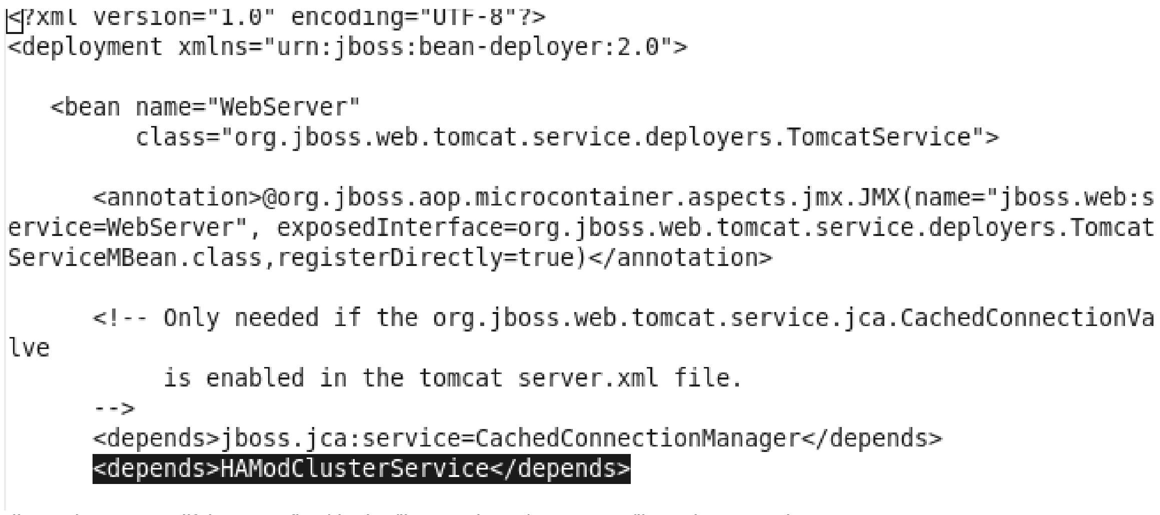


```
         redirectPort="8443" />

      <!-- SSL/TLS Connector configuration using the admin devl guide keystore
      <Connector protocol="HTTP/1.1" SSLEnabled="true"
           port="8443" address="${jboss.bind.address}"
           scheme="https" secure="true" clientAuth="false"
           keystoreFile="${jboss.server.home.dir}/conf/chap8.keystore"
           keystorePass="rmi+ssl" sslProtocol = "TLS" />
      -->

      <Engine name="jboss.web" defaultHost="localhost" jvmRoute="${jboss.jvmRoute
}">

         <!-- The JAAS based authentication and authorization realm implementatio
n
         that is compatible with the jboss 3.2.x realm implementation
```

*Illustration 39: Adding jvmRoute to same server.xml file as in Illustration 38*

Note that we are defining the jvmRoute as jboss.jvmRoute.  This flags it as a dynamic property and we will be passing in during server startup.  Save and close this file.

The final file we need to modify is located at node1/deploy/jbossweb.sar/META-INF/jboss-beans.xml.  In this file, we need to let jbossweb know that our webserver is dependent upon the HAModClusterService bean that is defined in our mod_cluster.sar (defined for us).  Your changes to this file will look like the following:



```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="urn:jboss:bean-deployer:2.0">

   <bean name="WebServer"
         class="org.jboss.web.tomcat.service.deployers.TomcatService">

      <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="jboss.web:s
ervice=WebServer", exposedInterface=org.jboss.web.tomcat.service.deployers.Tomcat
ServiceMBean.class,registerDirectly=true)</annotation>

      <!-- Only needed if the org.jboss.web.tomcat.service.jca.CachedConnectionVa
lve
           is enabled in the tomcat server.xml file.
      -->
      <depends>jboss.jca:service=CachedConnectionManager</depends>
      <depends>HAModClusterService</depends>
```

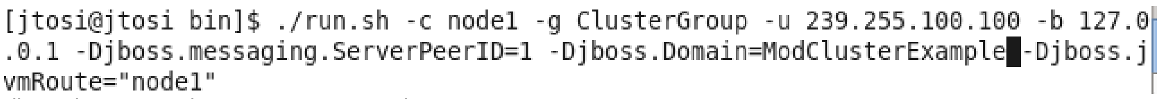
*Illustration 40: Modifying <config>/deploy/jbossweb.sar/META-INF/jboss-beans.xml.*

Save this file and close it.

With that all of our changes are complete.  To reiterate – while there were 3 files we needed to modify and the changes seemed maybe not the most intuitive, the best practice is to make these changes once and then just copy the JBoss instance around as a foundation for other instances.  With that approach, the only time you would need to modify any fields is if the HTTPD server configuration needed to change because the JBoss instance would have a new / different HTTPD front-ending it.

To add a second instance to the JBoss cluster, just copy the 'node1' directory recursively to a new directory, say name 2.  Add more instances in the same manner.

With all of our changes complete, we start up our JBoss instance:

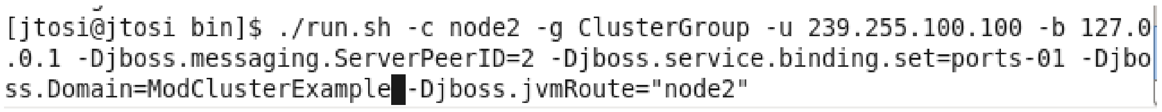


```
[jtosi@jtosi bin]$ ./run.sh -c node1 -g ClusterGroup -u 239.255.100.100 -b 127.0
.0.1 -Djboss.messaging.ServerPeerID=1 -Djboss.Domain=ModClusterExample -Djboss.j
vmRoute="node1"
```

*Illustration 41: Starting up our JBoss EAP instance*

Please refer to the **JBoss Clustering Considerations** whitepaper in the Red Hat Customer portal.  The only items we interacted with in this example are the last 2 parameters – the domain and the jvmRoute.

For sake of seeing a cluster in action, here is the same command, just starting a second JBoss instance (will come in handy when we look at mod_cluster-manager).  DO NOT START UP YOUR SECOND INSTANCE JUST YET!:

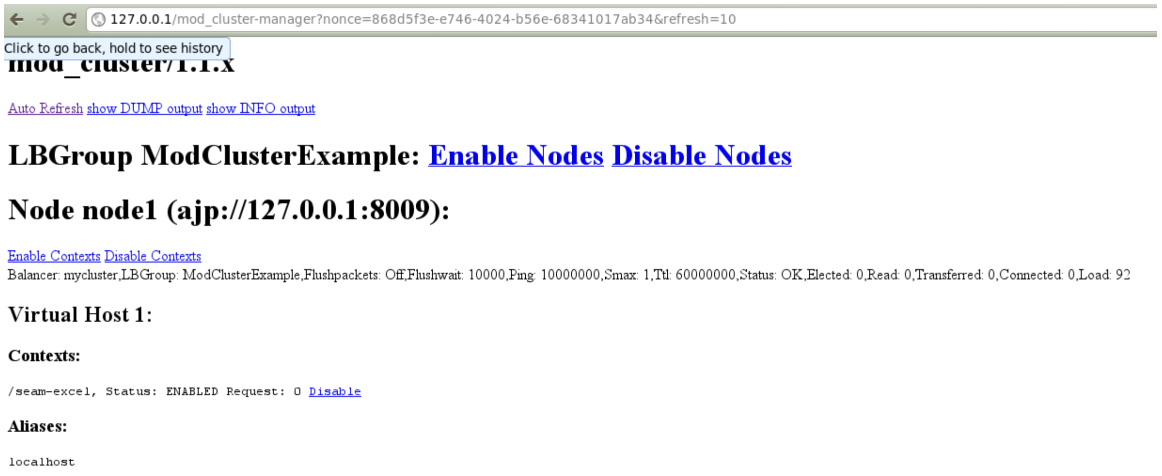


```
[jtosi@jtosi bin]$ ./run.sh -c node2 -g ClusterGroup -u 239.255.100.100 -b 127.0
.0.1 -Djboss.messaging.ServerPeerID=2 -Djboss.service.binding.set=ports-01 -Djbo
ss.Domain=ModClusterExample -Djboss.jvmRoute="node2"
```

*Illustration 42: Starting up our second JBoss EAP instance*

If Apache HTTPD is not up and running, be sure to start it up now.

To verify that mod_cluster in Apache has discovered your JBoss instance (node1), point your browser to *domain/*mod_cluster-manager. You should see the following:



```
127.0.0.1/mod_cluster-manager?nonce=868d5f3e-e746-4024-b56e-68341017ab34&refresh=10
```

mod_cluster/1.1.x

Auto Refresh show DUMP output show INFO output

LBGroup ModClusterExample: Enable Nodes Disable Nodes

Node node1 (ajp://127.0.0.1:8009):

Enable Contexts Disable Contexts
Balancer: mycluster,LBGroup: ModClusterExample,Flushpackets: Off,Flushwait: 10000,Ping: 10000000,Smax: 1,Ttl: 60000000,Status: OK,Elected: 0,Read: 0,Transferred: 0,Connected: 0,Load: 92

Virtual Host 1:

Contexts:

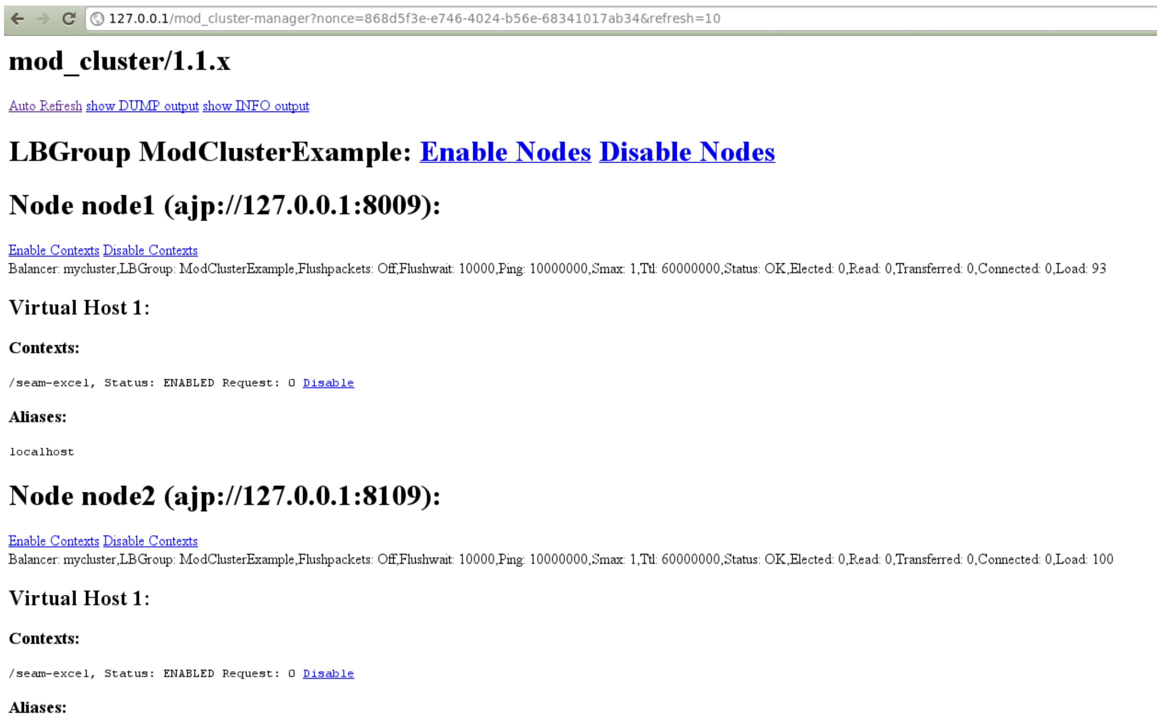
/seam-excel, Status: ENABLED Request: 0 Disable

Aliases:

localhost

*Illustration 43: Verifying mod_cluster using mod_cluster-manager – http://127.0.0.1/mod_cluster-manager. Note only one server is visible as we haven't started our second instance yet.*

Now start up your second instance of JBoss. Once it is up, you will see in mod_cluster-manager that mod_cluster has discovered your new instance:


127.0.0.1/mod_cluster-manager?nonce=868d5f3e-e746-4024-b56e-68341017ab34&refresh=10

mod_cluster/1.1.x

Auto Refresh show DUMP output show INFO output

LBGroup ModClusterExample: Enable Nodes Disable Nodes

Node node1 (ajp://127.0.0.1:8009):

Enable Contexts Disable Contexts
Balancer: mycluster,LBGroup: ModClusterExample,Flushpackets: Off,Flushwait: 10000,Ping: 10000000,Smax: 1,Ttl: 60000000,Status: OK,Elected: 0,Read: 0,Transferred: 0,Connected: 0,Load: 93

Virtual Host 1:

Contexts:

```
/seam-excel, Status: ENABLED Request: 0 Disable
```

Aliases:

```
localhost
```

Node node2 (ajp://127.0.0.1:8109):

Enable Contexts Disable Contexts
Balancer: mycluster,LBGroup: ModClusterExample,Flushpackets: Off,Flushwait: 10000,Ping: 10000000,Smax: 1,Ttl: 60000000,Status: OK,Elected: 0,Read: 0,Transferred: 0,Connected: 0,Load: 100

Virtual Host 1:

Contexts:

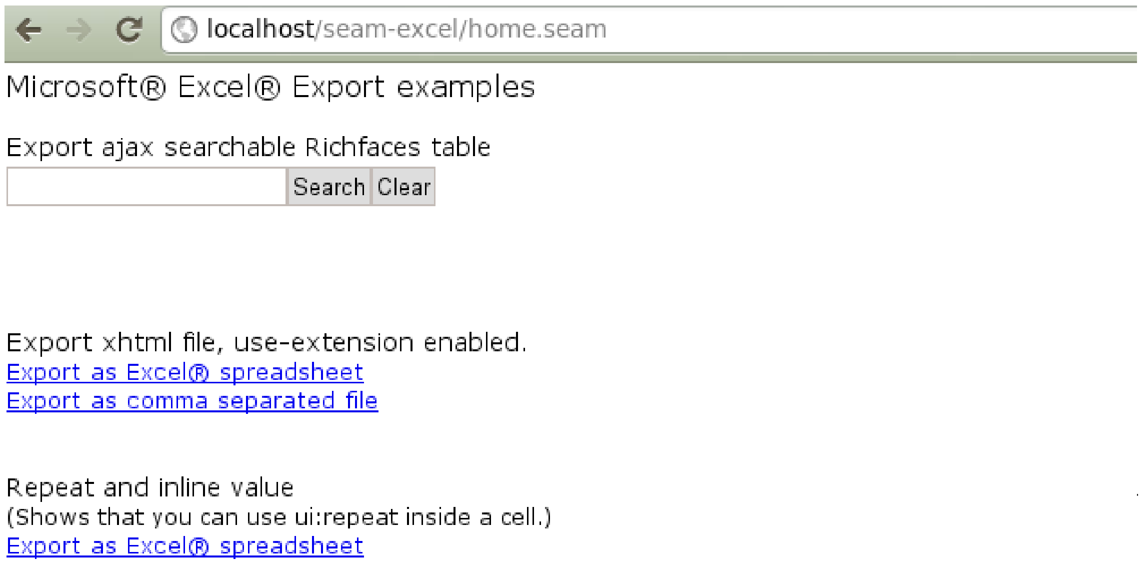
```
/seam-excel, Status: ENABLED Request: 0 Disable
```

Aliases:

*Illustration 44: Mod_cluster-manager after our second JBoss EAP instance was started. Note that no change to apache was needed*

At this point, we have verified that mod_cluster knows about our JBoss instances. What about the applications? We never defined them in httpd.conf...but we can see them in the mod_cluster-manager console.

In your browser, now simply point to that context in your domain – in our example we are using localhost:


localhost/seam-excel/home.seam

Microsoft® Excel® Export examples

Export ajax searchable Richfaces table

Search Clear

Export xhtml file, use-extension enabled.
Export as Excel® spreadsheet
Export as comma separated file

Repeat and inline value
(Shows that you can use ui:repeat inside a cell.)
Export as Excel® spreadsheet

*Illustration 45: Accessing an application through Apache HTTPD setup with mod_cluster. Note that applications were never identified explicitly to Apache HTTPD like we have to do with mod_jk*

From this you can see that our apache web server is now sending these requests down to our application servers.  As we add new applications or even new application servers to our environment, we don't need to make any changes to our webserver – mod_cluster discovers all of this.

To wrap up this section, we went through the 3 main ways of connecting Apache HTTPD to our JBoss instances – using mod_proxy, mod_jk, and mod_cluster.  The recommended approach is using mod_cluster as it is the most robust solution, providing better user experiences and ease of environment management.

## 4.8 Putting it all Together

The exercises covered in this whitepaper are now complete.  Below is a complete reference of the httpd.conf, workers.properties, and uriworkermap.properties files used in this whitepaper. Note that some areas are commented out (depending on connection strategy) and general comments are removed for brevity.

**Httpd.conf:**

```
### Section 1: Global Environment

#

ServerTokens Prod

ServerRoot "/var/local/httpd/"

PidFile run/httpd.pid

Timeout 120

 KeepAlive On

MaxKeepAliveRequests 0

KeepAliveTimeout 60

<IfModule prefork.c>

StartServers       8

MinSpareServers    5

MaxSpareServers   20

ServerLimit      256

MaxClients       256

MaxRequestsPerChild  4000
```

```
</IfModule>


<IfModule worker.c>

StartServers         2

MaxClients         150

MinSpareThreads     25

MaxSpareThreads     75

ThreadsPerChild     25

MaxRequestsPerChild  0

</IfModule>


Listen 80


LoadModule auth_basic_module /var/local/httpd/modules/mod_auth_basic.so

LoadModule auth_digest_module /var/local/httpd/modules/mod_auth_digest.so

LoadModule authn_file_module /var/local/httpd/modules/mod_authn_file.so

LoadModule authn_alias_module /var/local/httpd/modules/mod_authn_alias.so

LoadModule authn_anon_module /var/local/httpd/modules/mod_authn_anon.so

LoadModule authn_dbm_module /var/local/httpd/modules/mod_authn_dbm.so

LoadModule authn_default_module /var/local/httpd/modules/mod_authn_default.so

LoadModule authz_host_module /var/local/httpd/modules/mod_authz_host.so

LoadModule authz_user_module /var/local/httpd/modules/mod_authz_user.so

LoadModule authz_owner_module /var/local/httpd/modules/mod_authz_owner.so

LoadModule authz_groupfile_module
/var/local/httpd/modules/mod_authz_groupfile.so

LoadModule authz_dbm_module /var/local/httpd/modules/mod_authz_dbm.so

LoadModule authz_default_module /var/local/httpd/modules/mod_authz_default.so

LoadModule ldap_module /var/local/httpd/modules/mod_ldap.so

LoadModule authnz_ldap_module /var/local/httpd/modules/mod_authnz_ldap.so
LoadModule include_module /var/local/httpd/modules/mod_include.so

LoadModule log_config_module /var/local/httpd/modules/mod_log_config.so

LoadModule logio_module /var/local/httpd/modules/mod_logio.so

LoadModule env_module /var/local/httpd/modules/mod_env.so

LoadModule ext_filter_module /var/local/httpd/modules/mod_ext_filter.so

LoadModule mime_magic_module /var/local/httpd/modules/mod_mime_magic.so

LoadModule expires_module /var/local/httpd/modules/mod_expires.so

LoadModule deflate_module /var/local/httpd/modules/mod_deflate.so

LoadModule headers_module /var/local/httpd/modules/mod_headers.so

LoadModule usertrack_module /var/local/httpd/modules/mod_usertrack.so

LoadModule setenvif_module /var/local/httpd/modules/mod_setenvif.so

LoadModule mime_module /var/local/httpd/modules/mod_mime.so

LoadModule dav_module /var/local/httpd/modules/mod_dav.so

LoadModule status_module /var/local/httpd/modules/mod_status.so

LoadModule autoindex_module /var/local/httpd/modules/mod_autoindex.so

LoadModule info_module /var/local/httpd/modules/mod_info.so

LoadModule dav_fs_module /var/local/httpd/modules/mod_dav_fs.so

LoadModule vhost_alias_module /var/local/httpd/modules/mod_vhost_alias.so

LoadModule negotiation_module /var/local/httpd/modules/mod_negotiation.so

LoadModule dir_module /var/local/httpd/modules/mod_dir.so

LoadModule actions_module /var/local/httpd/modules/mod_actions.so

LoadModule speling_module /var/local/httpd/modules/mod_speling.so

LoadModule userdir_module /var/local/httpd/modules/mod_userdir.so

LoadModule alias_module /var/local/httpd/modules/mod_alias.so

LoadModule rewrite_module /var/local/httpd/modules/mod_rewrite.so

#LoadModule proxy_module /var/local/httpd/modules/mod_proxy.so

#LoadModule proxy_balancer_module
/var/local/httpd/modules/mod_proxy_balancer.so

#LoadModule proxy_ftp_module /var/local/httpd/modules/mod_proxy_ftp.so
```

```
#LoadModule proxy_http_module /var/local/httpd/modules/mod_proxy_http.so

#LoadModule proxy_connect_module /var/local/httpd/modules/mod_proxy_connect.so

#LoadModule proxy_ajp_module /var/local/httpd/modules/mod_proxy_ajp.so

LoadModule cache_module /var/local/httpd/modules/mod_cache.so

LoadModule suexec_module /var/local/httpd/modules/mod_suexec.so

LoadModule disk_cache_module /var/local/httpd/modules/mod_disk_cache.so

LoadModule file_cache_module /var/local/httpd/modules/mod_file_cache.so

LoadModule mem_cache_module /var/local/httpd/modules/mod_mem_cache.so

LoadModule cgi_module /var/local/httpd/modules/mod_cgi.so


#LoadModule jk_module /var/local/httpd/modules/mod_jk.so


LoadModule proxy_module /var/local/httpd/modules/mod_proxy.so

LoadModule proxy_ajp_module /var/local/httpd/modules/mod_proxy_ajp.so

LoadModule slotmem_module /var/local/httpd/modules/mod_slotmem.so

LoadModule manager_module /var/local/httpd/modules/mod_manager.so

LoadModule proxy_cluster_module /var/local/httpd/modules/mod_proxy_cluster.so

LoadModule advertise_module /var/local/httpd/modules/mod_advertise.so


User apache

Group apache


### Section 2: 'Main' server configuration


ServerAdmin root@localhost

UseCanonicalName Off

DocumentRoot "/var/local/httpd/www/html"

<Directory />
```

```
    Options FollowSymLinks

    AllowOverride None

</Directory>


<Directory "/var/local/httpd/www/html">

   Options Indexes FollowSymLinks

   AllowOverride None

   Order allow,deny

    Allow from all

</Directory>

<IfModule mod_userdir.c>

    UserDir disable

</IfModule>


DirectoryIndex index.html index.html.var

AccessFileName .htaccess

<Files ~ "^\.ht">

    Order allow,deny

    Deny from all

</Files>


TypesConfig /etc/mime.types


DefaultType text/plain


<IfModule mod_mime_magic.c>

    MIMEMagicFile conf/magic

</IfModule>
```

```
HostnameLookups Off

ErrorLog logs/error_log

LogLevel debug


LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined

LogFormat "%h %l %u %t \"%r\" %>s %b" common

LogFormat "%{Referer}i -> %U" referer

LogFormat "%{User-agent}i" agent


CustomLog logs/access_log combined


ServerSignature Off


Alias /icons/ "/var/local/httpd/www/icons/"


<Directory "/var/local/httpd/www/icons">

    Options Indexes MultiViews

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>


<IfModule mod_dav_fs.c>

    DAVLockDB /var/lib/dav/lockdb

</IfModule>


ScriptAlias /cgi-bin/ "/var/local/httpd/www/cgi-bin/"
```

```
<Directory "/var/local/httpd/www/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>

IndexOptions FancyIndexing VersionSort NameWidth=* HTMLTable

AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu

AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl

AddIcon /icons/tex.gif .tex

AddIcon /icons/bomb.gif core


AddIcon /icons/back.gif ..

AddIcon /icons/hand.right.gif README

AddIcon /icons/folder.gif ^^DIRECTORY^^

AddIcon /icons/blank.gif ^^BLANKICON^^


DefaultIcon /icons/unknown.gif


ReadmeName README.html

HeaderName HEADER.html


IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t


AddLanguage ca .ca

AddLanguage cs .cz .cs

AddLanguage da .dk

AddLanguage de .de

AddLanguage el .el

AddLanguage en .en

AddLanguage eo .eo

AddLanguage es .es

AddLanguage et .et

AddLanguage fr .fr

AddLanguage he .he
AddLanguage hr .hr

AddLanguage it .it

AddLanguage ja .ja

AddLanguage ko .ko

AddLanguage ltz .ltz

AddLanguage nl .nl

AddLanguage nn .nn

AddLanguage no .no

AddLanguage pl .po

AddLanguage pt .pt

AddLanguage pt-BR .pt-br

AddLanguage ru .ru

AddLanguage sv .sv

AddLanguage zh-CN .zh-cn

AddLanguage zh-TW .zh-tw


LanguagePriority en ca cs da de el eo es et fr he hr it ja ko ltz nl nn no pl
pt pt-BR ru sv zh-CN zh-TW


ForceLanguagePriority Prefer Fallback


AddDefaultCharset UTF-8


AddType application/x-compress .Z

AddType application/x-gzip .gz .tgz


AddHandler type-map var


AddType text/html .shtml
```

```
AddOutputFilter INCLUDES .shtml


Alias /error/ "/var/local/httpd/www/error/"


<IfModule mod_negotiation.c>
<IfModule mod_include.c>
    <Directory "/var/local/httpd/www/error">
        AllowOverride None
        Options IncludesNoExec
        AddOutputFilter Includes html
        AddHandler type-map var
        Order allow,deny
        Allow from all
        LanguagePriority en es de fr
        ForceLanguagePriority Prefer Fallback
    </Directory>

</IfModule>
</IfModule>


BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0


BrowserMatch "Microsoft Data Access Internet Publishing Provider" redirect-
carefully
BrowserMatch "MS FrontPage" redirect-carefully
BrowserMatch "^WebDrive" redirect-carefully

BrowserMatch "^WebDAVFS/1.[0123]" redirect-carefully

BrowserMatch "^gnome-vfs/1.0" redirect-carefully

BrowserMatch "^XML Spy" redirect-carefully

BrowserMatch "^Dreamweaver-WebDAV-SCM1" redirect-carefully


##MOD_CLUSTER ADDITIONS

<Location /mod_cluster-manager>

SetHandler mod_cluster-manager

Order deny,allow

Deny from all

Allow from 127.0.0.1

</Location>


##MOD_JK ADDITIONS

#JkWorkersFile /var/local/httpd/conf/workers.properties

#JkShmFile /var/local/httpd/mod_jk.shm

#JkLogFile /var/local/httpd/logs/mod_jk.log

#JkLogLevel info

#JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "


### Section 3: Virtual Hosts

<VirtualHost 127.0.0.1>

  DocumentRoot /home/jtosi/website1

  ServerName www.site1.com

  ErrorLog logs/site1-error.log

  CustomLog logs/site1-access.log common

  CustomLog "|/var/local/httpd/sbin/rotatelogs /var/local/httpd/logs/site1-
access.log 86400" common
```

```
<IfModule mod_cache_disk>

 CacheRoot /home/jtosi/apache/site1cacheroot

 CacheEnable disk /

 CacheDirLevels 5

 CacheDirLength 3

CacheSize 2000000

CacheMinFileSize 4
CacheMaxFileSize 256000

CacheDefaultExpire 86400

CustomLog logs/cached-reqeusts.log common env=cache-hit
CustomLog logs/uncached-requests.log common env=cache-miss
CustomLog logs/revalidated-requests.log common env=cache-revalidate
</IfModule>

## We don’t want to cache highly volatile content
#CacheDisable http://www.somedomain.com/real-time-stock-market-quotes/

 <Directory "/home/jtosi/website1">
    Order allow,deny
    Allow from all
  </Directory>

#JkMountFile /var/local/httpd/conf/uriworkermap.properties

#<Proxy balancer://jbossCluster>
#BalancerMember ajp://127.0.0.1:8009 loadfactor=1
#BalancerMember ajp://127.0.0.1:8109 loadfactor=3
#ProxySet stickysession=JSESSIONID
#</Proxy>
#ProxyPass / balancer://jbossCluster/
#ProxyPassReverse / balancer://jbossCluster/
</VirtualHost>

Listen 127.0.0.1:6666
<VirtualHost 127.0.0.1:6666>
<Location />
      Order deny,allow
      Deny from all
      Allow from 127.0.0.
   </Location>

 KeepAliveTimeout 60
 MaxKeepAliveRequests 0

 ManagerBalancerName mycluster
 ServerAdvertise On
</VirtualHost>
```

```
<VirtualHost 192.168.1.102>
  DocumentRoot /home/jtosi/website2
  ServerName www.site2.come
  ErrorLog logs/site2-error.log
  CustomLog logs/site2-access.log common

  CacheRoot /home/jtosi/apache/site2cacheroot
  CacheEnable disk /
  CacheDirLevels 5
  CacheDirLength 3



  <Directory "/home/jtosi/website2">
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

**uriworkermap.properties**

```
# Mount the Servlet context to the ajp13 worker
/jmx-console=examplebalancer
/jmx-console/*=examplebalancer
/web-console=examplebalancer
/web-console/*=examplebalancer
/admin-console=examplebalancer
/admin-console/*=examplebalancer
```

**workers.properties**

```
# Define list of workers that will be used
# for mapping requests
worker.list=examplebalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=localhost
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8109
worker.node2.host=localhost
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=3

# Load-balancing behaviour
worker.examplebalancer.type=lb
worker.examplebalancer.balance_workers=node1,node2
worker.examplebalancer.sticky_session=1
#worker.list=loadbalancer

# Status worker for managing load balancer
```

```
worker.status.type=status
```

## RESOURCES

### Apache Directives Reference

A complete explanation as well as a complete listing on the Apache directives can be found on the apache website at http://httpd.apache.org/

### Mod_jk Reference

More information on mod_jk can be found at the mod_jk website at http://tomcat.apache.org/connectors-doc/index.html Also, please refer to the **Mod_jk with multicast** whitepaper available on the Red Hat Customer portal.

### Mod_cluster Reference

More information on mod_cluster can be found at http://www.jboss.org/mod_cluster. Also, please refer to the **Mod_cluster with multicast** whitepaper available on the Red Hat Customer portal.

## Questions/Comments/Issues

If you have questions or comments about this whitepaper, please enter them in the Red Hat customer portal for this specific whitepaper: https://access.redhat.com/knowledge/techbriefs . If you have a technical issue following this whitepaper please open a support case: https://access.redhat.com/support/cases/new