

# Running Multiple Lasso Servers in Parallel

## Using Apache 2.2 Proxy Balancer to Gain Redundancy and Capacity

Greg Willits • May 12, 2007  
corrections added Nov 20, 2007

Based on configuration efforts originated by Yogi Kulkarni for rezzline.com

[http://www.ldml.org/articles/lasso\\_proxy\\_balancer:lasso](http://www.ldml.org/articles/lasso_proxy_balancer:lasso)

Whether your application needs the capacity of multiple boxes or you simply want multiple machine redundancy for reliability, you can turn a low-cost (or an older) server into an effective load balancer to improve Lasso application performance.

Apache 2.1 introduced `mod_proxy_balancer`<sup>1</sup> which is designed to direct incoming requests to multiple possible target servers. It supports both simple count and more advanced weighted traffic algorithms. The latter one is useful for where servers of various capacities are available, so you can apportion the number of requests that go to your original 1Ghz server and your shiny new multi-core machine.

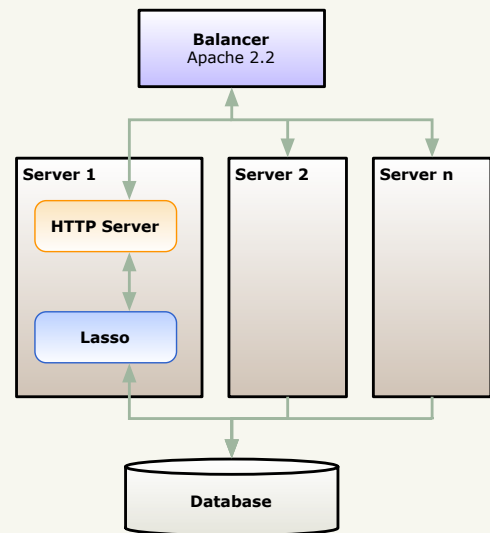
One term to clarify. I am going to use the term *parallel instances* instead of *distributed*. This article isn't about distributed Lasso. We're not breaking a large application into pieces to run on multiple boxes. We are taking one instance of an entire application, and duplicating that application on many identically configured boxes. As indicated, you might be doing this to maximize service availability through redundancy, or you might need the processing capacity of several machines.

I should also clarify that we'll be focusing on the Lasso layer of all this. If redundancy is your goal, you'll want redundant balancers and databases, but this article focuses only on creating redundant Lasso application code. The others are topics for another day.

And now for my usual disclaimer: this technique comes out of the first time I've tried

### Parallel Lasso Overview

In a nutshell, this method uses the Apache 2.2 `mod_proxy_balancer` to distribute requests coming into one machine out to an arbitrary number of servers to handle the processing of the requests..



this. There were several little niggles to figure out along the way to accomplish our goals. There may be simpler methods than what I ended up with, but we tried to keep it as simple as possible. Feedback is good, so if you any, let 'er rip.

<sup>1</sup> [http://httpd.apache.org/docs/2.2/mod/mod\\_proxy\\_balancer.html](http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html)

## Objectives

We have several objectives. First, the obvious is to vector requests for a page to one of an arbitrary number of machines that can fulfill it.

Second, the not so obvious objective comes from taking an opportunity to separate the delivery of static content from dynamic content. If you look at the figure on page 1, the original page request is going to come into the Balancer. That request is then forwarded to ServerX. We need to use ServerX to process Lasso code, but we don't need ServerX to deliver an image or a pure HTML static page. So, why waste time having those things processed by ServerX?

If there's any content which the Balancer can deliver on its own, especially if that content can be cached, then we may as well have Balancer take care of that directly. (Obviously at some point Balancer gets overloaded too, but you'd want more than one Balancer in a fully redundant setup anyway, and at the scale we're talking about for most real-world websites, one or two modern servers delivering images and static HTML can deliver one very serious load of requests).

The third objective is that we want to be able to run multiple applications. For example, we'd want to run production and beta versions of a site. Perhaps there's actually more than one application such as a public site and an internal-use-only company site or admin site.

The configuration details are designed to take care of all these objectives.

## What You'll Need

Regardless of why you want/need to run multiple Lasso boxes, the technique is the same, so we'll leave the reasons *why* behind, and focus on the *how*.

### Machines

To implement this technique for production, ideally you'll need at least three servers. Technically, for experimenting, two servers will be enough to know you've got a working configuration. After that, it's just more of the same thing.

No matter how many servers you'll be using, a preferred setup is to have one dedicated to being the balancer, then one or more are dedicated to serving Lasso.

For the purposes of the article, I am going to assume three machines are being used. I am going to call these Balancer, Server1, and Server2. Servers 1 and 2 are generically going to be called an application server. (That means something particular in the Java world, but for us it just means a server with the main application on it, as opposed to the database server or whatever).

If you have only two boxes to experiment with, you'll be setting one up as the Balancer and the second one as Server1.

### Apache

For the Balancer, you'll need to run Apache 2.2, but for each server that runs Lasso, you can use any HTTP server you prefer.

The way this system works is that the Balancer gets a request, and it forwards that request to one of many application servers maintained in a list. Those application servers are configured similarly to how you run a single Lasso box. So, the HTTP server on the application server box doesn't really matter. You can continue to use the built-in Apache with OS X/Linux, or IIS on a Windows.

### Lasso

I'll be assuming some version of Lasso 8. However, this only because I'll be talking about maintaining production, beta, and test versions of the application on each box as separate Sites. If you take that out of the equation, then, the basic parallel boxes technique will work with any version of Lasso all the way back to 5.0.

### Database

Your application probably uses a database of some type, but I'm going to largely ignore that facet. There's really nothing special about the database server integration. However, to complete the visual picture, just imagine one database server for now, and each Lasso server has a connector host defined to point to that one database server just as if you only had one Lasso

box. So, however you integrate your database to Lasso now, you do the exact same thing with each Lasso box in a parallel setup.

## Dummy Setup Info

In order to have some dummy info to play with we'll assume this is our setup info:

- domain = www.sprockets.com
- Balancer real IP = 100.0.0.255
- Server1 real IP = 100.0.0.1
- Server2 real IP = 100.0.0.2
- Balancer internal IP = 10.10.1.255
- Server1 internal IP = 10.10.1.1
- Server2 internal IP = 10.10.1.2
- http server root = /web/ (you can adjust that for your servers)
- eventually we're going to configure a production instance (Prod) and a beta instance (Beta) of the sprockets application.

## Preparing Server1 and Server2 with Lasso

For the application server boxes, begin by installing Lasso and the HTTP server you prefer in the normal way for a single-box setup. With one exception, go ahead and complete a default setup (with the database too if applicable) and test it so that you know the application is working correctly in your normal setup manner. That exception? Do not use the default Site, create a new Site, and use that Site for your application. Name this Site `app_production`. This will become more meaningful later, and we can tailor it to your prefs later as well. For now, go with the flow.

As we progress through the setup process, we'll alter some of the configuration of this machine.

## Preparing Balancer

This can be a fairly simple machine. We'll be using content caching, so you'll want enough RAM to cover that, but otherwise, there's not going to be a lot of demand on this machine.

## Two HTTP Servers per Application Box

When this setup was originally developed, it was designed to allow each application server to run Lasso and Ruby on Rails. The application being deployed is actually several separate applications: a consumer site (on Rails), a business-to-business site (on Lasso), and an admin site (on Lasso).

Normally, mixing languages is no big deal, but this proved a little tricky in that each box ended up with two HTTP servers. Rails used Mongrel (an HTTP server for Rails apps), while Lasso was setup to use Apache. We had trouble with HTTP greediness where one would grab the requests intended for another. Seems obvious now, but it was confusing in the beginning.

Ultimately, we had to assign applications from Rails and Lasso their own IP address on each machine. So Rails requests went to one IP address, and the Lasso requests went to another IP address.

While we won't necessarily focus on that complex of a setup in this article, it can be done.

One thing you need to do is copy the web application code onto the Balancer machine. Technically we only want some of those files, but from an installation and maintenance standpoint it will be a lot easier just to copy the entire application code. For discussion purposes we're using /web/ as the web root and /sprocketsProd/ as our site code folder. So, we'd expect to see the entire application's source code installed in the folder /web/sprocketsProd/. We're not installing Lasso, or any other service, just the source code.

## Installing Apache 2.2 on the Balancer

The first major thing that is possibly different from your normal setup is that we'll need to use Apache 2.2 on the Balancer. I suppose some Linux systems might have that natively, but it won't be native to the Mac or Windows. This isn't an exhaustive how-to on installation, but I figured some info would be beneficial.

### Apache 2.2 for Mac

There's no one-button solution for doing this, you'll have to get your hands dirty. The best option I found for installing Apache 2.2 on the

Mac is to use MacPorts. I've avoided doing anything through MacPorts or Fink for a long time, but the project that started this whole thing strives to automate the entire system installation and configuration of machines to enable rapid and reliable scaling, so using something like MacPorts helps accomplish that.

I'm not very fluent with all the ins and out of MacPorts, so if you run into trouble, get help wherever you can, but here's a few tidbits to help you get started.

I find the organization of MacPorts site to be terrible, starting with a highly unintuitive location of the documentation you need to get started being the wiki link. You'll need to make sure you have the latest developer tools and X11 installed, then install the MacPorts manager itself. All the gory details can be found here: <http://trac.macosforge.org/projects/macports/wiki/InstallingMacPorts>. From that point read through the QuickStart guide for general how-to info.

To install Apache 2.2, make sure any older version is stopped. Then you'll use a command in Terminal like this:

```
sudo port install apache2
```

After a boatload of stuff goes screaming by in the terminal window, you should have an

installed and ready to run version of Apache 2. If you have properly updated PATH, you should be able to use the following to start it:

```
sudo apachectl start
```

Otherwise you may need to cd or use the full path of `/opt/local/apache/bin/apachectl`. Modifying the PATH is the best option as it also helps to prevent accidentally starting up the built-in version of Apache.

By the way, if you're using OS X Server, don't get misled by the `/opt/apache2/` path! That's the one installed by Apple as part of OS X Server. Why not use that one? It's old, and doesn't include the balancer stuff. You have to install the newer 2.2 version. All MacPorts software is in `/opt/local/`.

## Apache 2.2 for Windows

If I don't know much about ports, then I know nothing about Apache on Windows, but I'm told that getting the binary installer from the Apache site directly is perfectly fine for installing Apache 2.2 on a Windows system:

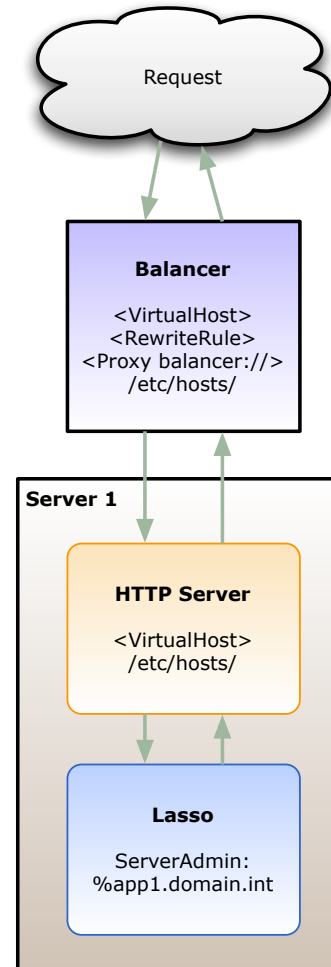
<http://httpd.apache.org/download.cgi>

## Summary of Chain of Events

Before we get bogged down in the setup details, let's take a quick visual overview of the chain of events:

- A request is received at the Balancer machine.
- The request is captured by a typical named virtual host directive in the Balancer machine.
- Inside the Balancer's vhost directive, a rewrite rule is used to differentiate whether the request requires the use of an application server or not. If not (i.e. images, CSS, JavaScript, or a static .html files), then the file is served directly from the Balancer machine.
- If the request requires the use of an app server (i.e. page code that needs processed by Lasso), the rewrite rule will forward the request to the proxy directive which internally uses to the proxy\_balancer module.
- The balancer directive includes a list of available app servers. The list uses internal domain names which are mapped to local IP addresses in /etc/hosts.
- When the request is forwarded to the app server (Server 1), a virtual host definition in that machine's HTTP server captures the request, and location directives are used as usual to trigger the handling by Lasso.
- Lasso's ServerAdmin for the Site uses the internal domain (again defined in /etc/hosts) to process the forwarded request which will bear the internal domain in the URI.
- An external domain name is used to provide access to the app server's SiteAdmin tools.

This article will examine the details of each configuration step.



## Configuring Apache 2.2 On Balancer

I am going to assume that you have some level of prior experience with Apache. You don't have to be a guru, but you should have had some tinkering with the config files and creating virtual hosts. If not, you may want to find a good tutorial or pester one of your Lasso buddies that works with Apache. Also, I'm going to be providing the paths for things based on OS X (you'll have to adjust for Windows or Linux).

### httpd.conf

Locate the httpd.conf file for Apache 2.2 (on the Mac it will be in /opt/local/apache2/conf/). Make sure that the balancer module is set to be loaded. You should see lines beginning like these not commented out:

```
LoadModule proxy_module modules/...
LoadModule proxy_balancer_module modules/...
```

If you have Apache experience, you can configure your vhosts includes however you want, but for those which have little experience, I'm going to suggest you do the following as a beginner tactic: copy (not move) the file /opt/local/apache2/conf/extra/httpd-vhost.conf one level up into the /conf/ folder. We'll use this file to define our balancer config. Then in the httpd.conf file look for these lines near the bottom:

```
# Virtual Hosts
# Include conf/extra/httpd-vhosts.conf
```

and change them to look like this:

```
# Virtual Hosts
  Include conf/httpd-vhosts.conf
```

### httpd-vhosts.conf

To start with we'll assume a single application. Also, remember that this is not the Apache that fulfills Lasso requests, so we are not really considering Lasso when configuring the virtual host. We'll just do the essentials. You can embellish with logging and whatever other tricky things you like to do.

```
01 # ----- www.sprockets.com
02 <VirtualHost 10.10.1.255:80>
03     ServerName      www.sprockets.com
04     DocumentRoot    /web/sprocketsProd
05
06     <Directory "/web/sprocketsProd">
07         Options FollowSymLinks
08         AllowOverride None
09         Order allow,deny
10         Allow from all
11     </Directory>
12
13 # Redirect all non-static requests to cluster
14     RewriteEngine On
15     RewriteCond     %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
16     RewriteRule     ^(.*)$ balancer://sprocketsProd_cluster%{REQUEST_URI} [P,QSA,L]
17 </VirtualHost>
```

Lines 03 and 04 are the usual suspects. Lines 06–11 counteract the default `<Directory>` directive in `httpd.conf` which will deny everything in order to start out with a more secure server. So this allows Apache to serve the contents of our web root folder.

## Determining What to Forward to Server1/Server2

Lines 14–16 are where the magic starts. Using `rewrite`, we redirect requests to the balancer module defined with a name of `sprocketsProd_cluster` (we’ll see that shortly).

One of the objectives was to forward requests to the app servers only when necessary – meaning, only when Lasso was truly needed to handle the request. Therefore anything not needing Lasso should not be handled by `proxy_balancer`. How do we differentiate requests? It so happens that an assumption we can make is that with a dynamic application, page URIs don’t map to real files on disk. This would be true for pages delivered using extensionless URIs, and in my real world app, this was the case with using the PageBlocks framework. This leaves files like images, CSS, and JavaScript with request URIs that *do* map to real disk files which do not need to be processed by Lasso and can served directly by the Balancer box. So, using the inverse of that logic, any request that Apache receives that is *not* be found to be a real file on disk in the web root should be passed on to Lasso.

Line 15 sets the condition for when requests are sent to the balancer module. The `!-f` command says if the URI path and “file” is not a real file on disk, the condition is true and the rule in line 16 to forward the request to the balancer is exercised.

Of course, with some Lasso sites this can be a problem. If you’re using simple Lasso format files where `/www.sprockets.com/products/widget.lasso` is a real file `/web/sprocketsProd/products/widget.lasso` then this method isn’t going to work well for you. You’ll have to do one of two things: eliminate those files from the copy of the application code on the Balancer machine, or change the rewrite conditions to forward requests for files names `.lasso` (or whatever logic makes sense for your code).

I’m going to make the lazy assumption that anyone interested in this balancing technique is probably writing sites that at least use extensionless URIs for pages or oter virtual URIs, or has the capability of tweaking the rewrite conditions.

You can monkey with the rewrite conditions to meet your needs, but the rule on line 16 should remain the same.

On line 16, the entire request, captured by `^(.*)$`, is converted to a balancer URI (the balancer cluster is defined in the next section). The rewrite flags include `P` to force the request through `mod_proxy` (which the proxy directive described in the next section defines how to handle the request). The `QSA` flag preserves any query string, and `L` says this is the last rule to apply. There’s only one rule, but I think explicitly using `L` makes Apache a little more efficient.

## Identifying Machines in the Balance Cluster

So, now we get to the heart of the balancer, and identify the cluster of machines to be used as application servers by the Balancer box. Are you ready?

```
18 <Proxy balancer://sprocketsProd_cluster>
19     BalancerMember http://app1.sprocketsProd.int:80
20     BalancerMember http://app2.sprocketsProd.int:80
21 </Proxy>
```

Yep, that’s all there is to it. A Proxy directive with a list of balance members – one for each box running Lasso. OK, you’re wondering why the weird domain name. This is where Lasso proved to be a bit trickier to deal with thanks to its GUI admin system. The details of why are covered when we get to the Lasso ServerAdmin config section.

Obviously, the .int is not an official TLD. Indeed, we're using .int to indicate the use of an internal URI (i.e. one on our local network). In order to make that work, we need to modify the hosts file. On OS X and Linux, you should find that in /etc/hosts. Add lines that look like this:

```
# Balancer Member declarations in /etc/hosts
10.10.1.1  app1.sprocketsProd.int
10.10.1.2  app2.sprocketsProd.int
```

The app1 and app2 prefixes are just a convention to indicate the machine is an appserver with the IP address ending with 1 and 2 respectively. So if you have addresses 10.10.1.168, then use 168app.sprocketsProd.int. It comes in handy for debugging and logging if the URI can help tell you which machine was involved. Of course, you can use any convention you like, and you need one entry for each box running Lasso.

## Some Extra Lasso-Specific Options

The above config details are the essentials. You may have some embellishments to add, and there's some that should be added primarily for security and reliability. Certain versions of Lasso are prone to problems if requests have URI longer than 245 characters (before the GET form portion). We may as well use the Balancer to reject these requests rather than waste processing by forwarding them on to application boxes. So these three directives will reject URIs with .lasso or with no extension, or with a / as the last character where the total length is 246 or greater.

```
<Location ~ "^.{246,}\.[Ll][Aa][Ss][Ss][Oo]$" >
    Deny from all
</Location>

<LocationMatch "^[\^\.]{246,}$" >
    Deny from all
</LocationMatch>

<LocationMatch "^[\^\.]{246,}/$" >
    Deny from all
</LocationMatch>
```

Additionally, we may as well reject requests for files that are attempts to peer into source code files. In PageBlocks, I use several extensions to identify file types (.dsp, .lgc, .ctag, .ctyp, .cnfg). So, my list is a bit long. Adapt this to suit the extensions you use.

```
<FilesMatch "\.([dD][sS][pP]|[Ll][gG][cC]|[cC][tT][aA][gG]|[cC][tT][yY][pP]|[cC][nN][fF][gG])$" >
    Order allow,deny
    Deny from all
</FilesMatch>

# prevent Apache from logging legitimate requests by Lasso for the above files
LogLevel crit
```

I end up using these directives in all Lasso virtual hosts, so I put them in a separate file named lassoCommon.conf and use an Include statement in the virtual host container just after line 11 in the sample vhost config above.

# Configuring Apache on Server1

I mentioned that for the application boxes, you can use any HTTP server you prefer. In my application, I've opted to stick with the built-in Apache 1.3 on OS X Server so there's less work to do when setting up a new box. However, I also bypass the GUI admin as it's just not as efficient to use, nor as flexible.

So, going on the assumption that you're ready to forsake the comfort of the GUI admin as well, here's how I set up my application box Apache. First, create a new folder named `/etc/httpd/sitesOriginal`. Move the files from `/etc/httpd/sites` to `/etc/httpd/sitesOriginal`. This will disconnect (without destroying) the config files managed by the GUI application.

Copy the file `/etc/httpd/sitesOriginal/virtual_host_global.conf` into the new `/sites/` folder. I prefer to rename this file `_listen.conf`, where the underscore ensures it will be listed first. Strip out all the comments added by Apple, and edit it to include just these two lines:

```
Listen          10.10.1.1:80
NameVirtualHost 10.10.1.1:80
```

I found that it was more effective if virtual hosts were declared using explicit IPs rather than the default `*:80` method. This solved some unique problems we had in our setup, so it may be prudent to use unless you have a specific reason not to.

Next, create a new file for virtual host directives. I'll use a variety of naming schemes depending on the complexity of the entire setup, but for now, let's just call it `vhosts.conf`, and place it in `/etc/httpd/vhosts/`. The contents of that file should start out like this:

```
01 # ----- www.sprockets.com
02 <VirtualHost 10.10.1.1:80>
03     ServerName      sprocketsProd.int
04     ServerAlias     *.sprocketsProd.int
05     DocumentRoot    /web/sprocketsProd
06 </VirtualHost>
```

## Lasso Specific Directives

While the above is a generic minimum, there are some Lasso-specific tweaks that are likely useful. If you're using extensionless URIs, you're probably already using something like these:

```
<LocationMatch "^[^\.]{1,245}$">
    SetHandler lasso8-handler
</LocationMatch>

<LocationMatch "^[^\.]{1,245}/$">
    SetHandler lasso8-handler
</LocationMatch>
```

Additionally, even though we already did this in the Balancer conf file, I prefer to reiterate these protections in each application box, just in case there's some exploit which circumvents the Balancer by going directly to an app server's IP address.

```
<Location ~ "^.{246,}\.[Ll][Aa][Ss][Ss][0o]$">
    Deny from all
</Location>
```

```
<LocationMatch "^[^\.]{246,}$">
    Deny from all
</LocationMatch>

<LocationMatch "^[^\.]{246,}/$">
    Deny from all
</LocationMatch>

<FilesMatch "\.([dD][sS][pP]|[lL][gG][cC]|[cC][tT][aA][gG]|[cC][tT][yY][pP]|[cC][nN][fF][gG])$">
    Order allow,deny
    Deny from all
</FilesMatch>
```

Of course, you may have any number of customized tweaks to your virtual hosts, and those should apply as you normally use them. If these are reusable for multiple virtual hosts, put them in a separate file and use an Include.

## /etc/hosts

Obviously, the .int in our vhost is not an official TLD. Indeed, we're using .int to indicate the use of an internal URI (i.e. one on our local network). In order to make that work, we need to modify the hosts file. On OS X and Linux, you should find that in /etc/hosts. Add lines that look like this:

```
# internal Balancer Member domain names
10.10.1.1 app1.sprocketsProd.int
```

## Lasso ServerAdmin on Server1

There's one aspect to ServerAdmin setup that must be done a certain way (at least that's what I discovered by trial and error). In the Site Detail we need both an external and an internal host definition in Site Criteria, but the external host must be defined first, or the Admin app gets wrapped around its own axle during the administrator authentication.

	Host Name	Host Root
1	%app1.sprocketsProd.com	/%
2	%app1.sprocketsProd.int	/%
New		

We need the external host in order to access the SiteAdmin application, and we need the internal host so the Site will respond to requests by forwarded by the proxy\_balancer.

SiteAdmin needs no special configuration.

## Lasso Client\_IP and Similar Header Detail Tags

Something to be aware of is that all requests coming to the application server will come from the Balancer machine, and the HTTP headers will look that way. So, in your application code, if you are depending on `client_ip`, in our dummy case, every request will be reported as coming from 10.10.1.255. Similarly, code which determines domain names by looking for HOST in the headers will appear to be different on each application box because of the IP-specific subdomain we're using.

So, for such cases, you're going to have to use some custom adaptations. What I found is that with Balancer and Apache 2.2 feeding Apache 1.3 on the app servers, the following headers are added by the proxy to tell us the original request details: X-Forwarded-For, X-Forwarded-Host, and X-Forwarded-Server.

The -For is the original requestor IP address, and the other two both have the original URI of `www.sprocketsProd.com` (instead of the `.int` version).

Use the following code to extract the IP address to replace `client_ip`.

```
var:'fw_forwardedFor' = string;  
if: client_headers >> 'X-Forwarded-For';  
  $fw_forwardedFor = (string_findregexp:  
    client_headers,  
    -find='X-Forwarded-For:\\s?(.*)',  
    -ignorecase)->last;  
  $fw_forwardedFor->trim;  
/if;
```

What I can't say for sure is whether other HTTP servers will report the same X headers or not. You may need to experiment by peeking at the full headers to see what is there.

## Configuring Server2

Everything on Server2 should be identical to Server1 except those few places where IP address is used.

## Configuring Production and Beta Instances

I find it easier to use unique domains for production beta, and even test instances of an application. So, if your main website is `sprockets.com`, I'd register `sprocketsbeta.com` and `sprocketstest.com`, or maybe `sprockbeta.com` and `sprocktest.com`.

Using unique domains makes it much easier to keep subdomains simple. For example if you use subdomains like `admin.sprockets.com` and `store.sprockets.com`, then having `sprockbeta.com` allows all the subdomains to stay the same (`admin.sprockbeta.com`, etc).

This also makes it much easier to set up parallel redundancy for these versions of the application. I found having to use subdomains to distinguish between instances and machines got hairy and unreliable.

Additionally, production and beta instances must be in separate Lasso Sites.

In the Appendix, I give complete sample files based on what I implemented on our production systems that show a complete set of these apps (`www`, `admin`, `store`), and two versions (production and beta).

# Appendix A • Balancer Config Files

## /opt/local/apache/conf/httpd.conf

```
# =====
# Basic settings
# =====
ServerRoot "/opt/local/apache2"

# =====
# Performance settings
# =====
KeepAliveTimeout 10
StartServers 5
MaxRequestsPerChild 1000

# =====
# Modules
# =====
LoadModule authz_host_module      modules/mod_authz_host.so
LoadModule filter_module          modules/mod_filter.so
LoadModule deflate_module         modules/mod_deflate.so
LoadModule log_config_module      modules/mod_log_config.so
LoadModule expires_module         modules/mod_expires.so
LoadModule setenvif_module        modules/mod_setenvif.so
LoadModule dir_module             modules/mod_dir.so
LoadModule proxy_module           modules/mod_proxy.so
LoadModule proxy_http_module      modules/mod_proxy_http.so
LoadModule proxy_balancer_module  modules/mod_proxy_balancer.so
LoadModule mime_module            modules/mod_mime.so
LoadModule status_module          modules/mod_status.so
LoadModule rewrite_module         modules/mod_rewrite.so

# =====
# General settings
# =====
User www
Group www
ServerAdmin admin@sprockets.com

# =====
# Access control
# =====
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

# Prevent access to all subversion directories
<DirectoryMatch "^/.*\\.svn/">
    Order deny,allow
    Deny from all
</DirectoryMatch>
```

```

# Mac-specific filesystem protection
<Files "rsrc">
    Order allow,deny
    Deny from all
    Satisfy all
</Files>
<FilesMatch "^\.+">
    Order allow,deny
    Deny from all
    Satisfy all
</FilesMatch>
<Directory ~".*\.\.namedfork">
    Order allow,deny
    Deny from all
    Satisfy all
</Directory>

# =====
# Logs
# =====
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
ErrorLog logs/error_log
CustomLog logs/access_log combined

# =====
# MIME encoding
# =====
DefaultType text/plain
TypesConfig conf/mime.types
AddEncoding x-compress .Z
AddEncoding x-gzip .gz .tgz
AddType application/x-tar .tgz

# =====
# Virtual hosts
# =====
Listen 80
NameVirtualHost 10.10.1.40:80

# -----
# Lasso apps

Include /opt/local/apache2/conf/vhosts/sprockets_prod.conf
Include /opt/local/apache2/conf/vhosts/sprockets_beta.conf

```

## /opt/local/apache/conf/vhosts/lassoCommon.conf

```
ServerAdmin    admin@example.com
```

```
<IfModule mod_dir.c>  
    DirectoryIndex default.lasso index.lasso  
</IfModule>
```

```
# set log level to crit to stop all the gripes  
# about denied PageBlocks file extensions when they're loaded  
LogLevel crit
```

```
# deny URLs with more than 245 chars  
<Location ~ "^.{246,}\.[Ll][Aa][Ss][Ss][Oo]$" >  
    Deny from all  
</Location>
```

```
# deny URLs with more than 245 chars  
<LocationMatch "^[\^\.]{246,}$" >  
    Deny from all  
</LocationMatch>
```

```
# deny URLs with more than 245 chars  
<LocationMatch "^[\^\.]{246,}/$" >  
    Deny from all  
</LocationMatch>
```

```
# deny all files that end in extensions used for source code by pageblocks  
# must specify something that covers all uppler/lower case variations  
# incllgcldsp|ctag|ctyplcnfg
```

```
<FilesMatch "\.([iI][nN][cC]|[dD][sS][pP]|[lL][gG][cC]|[cC][tT][aA][gG]|[cC][tT][yY][pP]|[cC][nN]  
[fF][gG])$" >  
    Order allow,deny  
    Deny from all  
</FilesMatch>
```

## /opt/local/apache/conf/vhosts/sprockets\_prod.conf

```
# -----
# www.sprockets.com

<VirtualHost 10.10.1.255:80>
    ServerName      www.sprockets.com
    DocumentRoot    /web/sprocketsProd

    <Directory "/web/sprocketsProd">
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

    Include /opt/local/apache2/conf/vhosts/lassoCommon.conf
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsProd_access_log" 2592000' combined
    ErrorLog  '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsProd_error_log" 2592000'

    RewriteEngine On
    RewriteCond  %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule  ^(.*)$ balancer://sprocketsProd_cluster%{REQUEST_URI} [P,QSA,L]
</VirtualHost>

# -----
# admin.sprockets.com

<VirtualHost 10.10.1.255:80>
    ServerName      admin.sprockets.com
    DocumentRoot    /web/sprocketsAdminProd

    <Directory "/web/sprocketsAdminProd">
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

    Include /opt/local/apache2/conf/vhosts/lassoCommon.conf
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminProd_access_log" 2592000'
    combined
    ErrorLog  '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminProd_error_log" 2592000'

    RewriteEngine On
    RewriteCond  %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule  ^(.*)$ balancer://sprocketsAdminProd_cluster%{REQUEST_URI} [P,QSA,L]
</VirtualHost>

# -----
<Proxy balancer://sprocketsProd_cluster>
    BalancerMember http://app1.sprocketsProd.int:80
    BalancerMember http://app2.sprocketsProd.int:80
</Proxy>
<Proxy balancer://sprocketsAdminProd_cluster>
    BalancerMember http://app1.sprocketsAdminProd.int:80
    BalancerMember http://app2.sprocketsAdminProd.int:80
</Proxy>
```

## /opt/local/apache/conf/vhosts/sprockets\_beta.conf

```
# -----
# www.sprocketsbeta.com

<VirtualHost 10.10.1.255:80>
    ServerName      www.sprocketsbeta.com
    DocumentRoot    /web/sprocketsBeta

    <Directory "/web/sprocketsBeta">
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

    Include /opt/local/apache2/conf/vhosts/lassoCommon.conf
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsBeta_access_log" 2592000' combined
    ErrorLog  '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsBeta_error_log" 2592000'

    RewriteEngine On
    RewriteCond  %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule  ^(.*)$ balancer://sprocketsBeta_cluster%{REQUEST_URI} [P,QSA,L]
</VirtualHost>

# -----
# admin.sprocketbeta.com

<VirtualHost 10.10.1.255:80>
    ServerName      admin.sprocketsbeta.com
    DocumentRoot    /web/sprocketsAdminBeta

    <Directory "/web/sprocketsAdminBeta">
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

    Include /opt/local/apache2/conf/vhosts/lassoCommon.conf
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminBeta_access_log" 2592000'
    combined
    ErrorLog  '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminBeta_error_log" 2592000'

    RewriteEngine On
    RewriteCond  %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule  ^(.*)$ balancer://sprocketsAdminBeta_cluster%{REQUEST_URI} [P,QSA,L]
</VirtualHost>

# -----
<Proxy balancer://sprocketsBeta_cluster>
    BalancerMember http://app1.sprocketsBeta.int:80
    BalancerMember http://app2.sprocketsBeta.int:80
</Proxy>
<Proxy balancer://sprocketsAdminBeta_cluster>
    BalancerMember http://app1.sprocketsAdminBeta.int:80
    BalancerMember http://app2.sprocketsAdminBeta.int:80
</Proxy>
```

## /etc/hosts

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##

127.0.0.1      localhost
255.255.255.255 broadcasthost
::1           localhost

#-----

10.10.1.1      app1.sprocketsProd.int
10.10.1.1      app1.sprocketsAdminProd.int

10.10.1.1      app1.sprocketsBeta.int
10.10.1.1      app1.sprocketsAdminBeta.int

#-----

10.10.1.2      app2.sprocketsProd.int
10.10.1.2      app2.sprocketsAdminProd.int

10.10.1.2      app2.sprocketsBeta.int
10.10.1.2      app2.sprocketsAdminBeta.int
```

# Appendix B • App Server 1 Config Files

## httpd.conf

Use your preferred setup for this (even the default file), but if you're using OS X Server, you'll have to abandon using the GUI admin application and edit virtual hosts manually.

## /etc/httpd/sites/\_listen.conf

```
Listen          10.10.1.1:80
NameVirtualHost 10.10.1.1:80
```

## /etc/httpd/lassoCommon.conf

```
<IfModule mod_dir.c>
    DirectoryIndex default.lasso index.lasso default.html index.html
</IfModule>

# deny all source code files for PageBlocks
# must specify something that covers all uppler/lower case variations
# incllasllgcldsplctaglctyplcnfg

<FilesMatch "\.([iI][nN][cC]|[dD][sS][pP]|[lL][gG][cC]|[cC][tT][aA][gG]|[cC][tT][yY][pP]|[cC][nN]
    [fF][gG])$" >
    Order allow,deny
    Deny from all
</FilesMatch>

<LocationMatch "^([^\.]+" >
    SetHandler lasso8-handler
</LocationMatch>

<LocationMatch "^([^\.]+" >
    SetHandler lasso8-handler
</LocationMatch>

# set log level to crit to stop all the gripes
# about PageBlocks file extensions when they're loaded
LogLevel crit
```

## /etc/httpd/sites/sprockets\_prod.conf

```
# -----
# www.sprockets.com

# handle requests for app via internal network
<VirtualHost 10.10.1.1:80>
    ServerName      sprocketsProd.int
    ServerAlias     *sprocketsProd.int
    ServerAdmin     admin@sprockets.com
    DocumentRoot    /web/sprocketsProd

    Include         "/etc/httpd/lassoCommon.conf"

    CustomLog       '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsProd_access_log" 2592000' "%v %h %l
    %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
    ErrorLog        '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsProd_error_log" 2592000'

</VirtualHost>

# handle requests for Lasso Admin from external network
<VirtualHost 10.10.1.1:80>
    ServerName      app1.sprockets.com
    ServerAdmin     admin@sprockets.com
    DocumentRoot    /web/sprocketsProd

    Include         "/etc/httpd/lassoCommon.conf"

    <Directory /web/sprocketsProd>
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>

</VirtualHost>

# -----
# admin.sprockets.com

# handle requests for app via internal network
<VirtualHost 10.10.1.1:80>
    ServerName      sprocketsAdminProd.int
    ServerAlias     *sprocketsAdminProd.int
    ServerAdmin     admin@sprockets.com
    DocumentRoot    /web/sprocketsAdminProd

    Include         "/etc/httpd/lassoCommon.conf"

    CustomLog       '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminProd_access_log" 2592000' "%v
    %h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
    ErrorLog        '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminProd_error_log" 2592000'

</VirtualHost>
```

```
# handle requests for Lasso Admin from external network
<VirtualHost 10.10.1.1:80>
  ServerName      app1.sprocketsAdmin.com
  ServerAdmin     admin@sprocketsAdmin.com
  DocumentRoot    /web/sprocketsAdminProd

  Include         "/etc/httpd/lassoCommon.conf"

  <Directory /web/sprocketsAdminProd>
    Options FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

## /etc/httpd/sites/sprockets\_beta.conf

```
# -----  
# www.sprockets.com  
  
# handle requests for app via internal network  
<VirtualHost 10.10.1.1:80>  
    ServerName      sprocketsBeta.int  
    ServerAlias     *sprocketsBeta.int  
    ServerAdmin     admin@sprockets.com  
    DocumentRoot    /web/sprocketsBeta  
  
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsBeta_access_log" 2592000' "%v %h %l  
    %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""  
    ErrorLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsBeta_error_log" 2592000'  
  
    Include "/etc/httpd/lassoCommon.conf"  
  
</VirtualHost>  
  
# -----  
# admin.sprockets.com  
  
# handle requests for app via internal network  
<VirtualHost 10.10.1.1:80>  
    ServerName      sprocketsAdminBeta.int  
    ServerAlias     *sprocketsAdminBeta.int  
    ServerAdmin     admin@sprockets.com  
    DocumentRoot    /web/sprocketsAdminBeta  
  
    CustomLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminBeta_access_log" 2592000' "%v  
    %h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""  
    ErrorLog '/usr/sbin/rotatelogs "/var/log/httpd/sprocketsAdminBeta_error_log" 2592000'  
  
    Include "/etc/httpd/lassoCommon.conf"  
  
</VirtualHost>
```

## /etc/hosts

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##

127.0.0.1          localhost
255.255.255.255   broadcasthost
::1               localhost

#-----

10.10.1.1          app1.sprocketsProd.int
10.10.1.1          app1.sprocketsAdminProd.int

10.10.1.1          app1.sprocketsBeta.int
10.10.1.1          app1.sprocketsAdminBeta.int
```

## Lasso ServerAdmin

The screenshot displays the Lasso 8.5 ServerAdmin web interface. The top navigation bar includes tabs for Setup, Sites, Utility, and Support. The 'Sites' tab is active, showing sub-tabs for Preferences, File Paths, and Import/Export. The main content area is divided into two panels: 'Site Listing' and 'Site Detail'.

**Site Listing**

ID	Name	Enabled	Status	Links
1	sprockets_prod	Enabled	Running	Stop   Visit
3	sprocketsAdmin_prod	Enabled	Running	Stop   Visit
5	sprockets_beta	Enabled	Running	Stop   Visit
6	sprocketsAdmin_beta	Enabled	Running	Stop   Visit

Below the table are buttons: Add Site..., Refresh, Stop All, and Disable All.

**Site Detail**

**Name:** sprockets\_prod  
**ID:** 3  
**Enabled:** Enabled (dropdown)  
**Status:** Running  
**Uptime:** 4 Days 12 Hours  
**Description:** [Empty text area]  
**Visit URL:** <http://app1.sprockets.com:80/siteadmin.0.LassoApp>

**Site Criteria**

	Host Name	Host Root	
1	%app1.sprockets.com	/%	[-]
2	%app1.sprocketsProd.int	/%	[-]
New			

Buttons at the bottom: Update, Delete, Stop, Restart, Site Preferences..., Allowed File Paths...