# InterWorx Server Administrator Webserver Guide

by InterWorx LLC

# Contents

# Preface

The webserver is by many considered to be the most critical service of a webhosting service. Most individuals and companies looking for web hosting are looking for a service that allows them to host their websites and files and make them available for download over the HTTP protocol. This guide is intended to educate and enrich you, the NodeWorx System Administrator, on the function and use of the webserver and how it integrates with InterWorx. By default, InterWorx uses Apache 2.x as the webserver for hosting content over the HTTP protocol. InterWorx also supports the LiteSpeed webserver, which gives webhosts more flexibility when it comes to choosing what software to run. In order to get the most out of this guide, the authors recommend that you have experience and knowledge dealing with the following:

- Linux command line and the shell (typically BASH).
- Editing text files.
- RHEL/CentOS-specific ideas
- Understanding of networking and DNS infrastructure.
- Familiarity with what TCP is and the HTTP protocol.
- Knowledge of your preferred webserver software on RHEL/CentOS - this guide is not intended to be a substitute for the Apache documentation.

As always, full detailed knowledge isn't necessary, but this guide is intended to help server admins understand the integration between Apache/LiteSpeed and the Control panel. It is not intended to explain more fundamental skills such as linux or knowledge of how the internet works. Please feel free to use Google if you find you are missing some bits of information necessary to digest the material in this documentation.

# Part I

# Fundamental Webserver Management with InterWorx

# Chapter 1

# A Brief primer on HTTP and Apache

This chapter may be somewhat technical and if you are just looking for information about interface elements in the control panel, you might want to head to chapter 3. Since the beginning of Internet, man has sought to deliver content across the series of tubes that moves data across the world at lightning fast speeds. From the bowels of government funded defense research and university high-speed communication networks, the world wide web and the HTTP protocol was spawned. Furthermore, HTTP protocol and HTML markup language gave birth to what most in the world understand as the internet today. For the most part, the software that runs on servers that performs the function of responding to HTTP requests is the Apache Web Server, an open source, high performance, fully modular, and highly support HTTP and HTTPS webserver software system. In layman's terms, it is the program that listens on port 80, recieves requests for specifc documents or files on the webserver, and sends them back to the web browser that requested them.

## 1.1 A primer of HTTP

This section primarily deals with the basics of TCP and HTTP just so we are on the same page for the rest of the guide. Feel free to skip to the next session if you already have an understanding of the OSI layer and the HTTP protocol. We say lofty things like HTTP protocol and TCP protocol a lot in this guide without really explaining what they mean. If you are a server admin without a strong networking or programming background, these terms might seem somewhat vague. A protocol is essentially a contract. It dictates how two devices or two pieces of software are supposed to talk to each other so they can understand each other. It ensures that there is understanding and when one side says something, the other side can respond appropriately.

**Physical Layer** It is well known that information travels over various mediums - copper wire, fiber optic cable, wireless signal. Each medium has the goal of ultimately communicating one thing: 1's and 0's between devices. 1's and 0's are the basic unit of information for a computer and thus each medium has it's own protocol on what means a 1 and what means a 0. For example on fiber optic, you might have a pulse of light be a 1 and an absence be a 0. You might say that there is a certain frequency to which the pulses come so both devices on each side of the cable knows what's a 1 and what's a 0 and what's just some blank time between a series of two 1's.

**Internet Layer** Well this is all well and good, but many devices share the same communication pathways - heck undersea cables can carry information across the oceans for billions of devices. How do we know what 1's and 0's are from who and what each 1 and 0 means? Well we can logically bundle a group of 1's and 0's together into something called a packet - and each packet can be a chunk of information from one devices headed to another device somewhere else. Every device has a unique address which is dictated by a set of thirty-two 1's and 0's (often referred to as an IP address) and each packet has information about how large it is and other meta information. As such, each packet's 1's and 0's are bundled together and split up so that each n number of bits represents a different piece of information in the packet. You network card has some basic circuitry that is able to detect when a series of ones and 0's is the beginning of a packet and it starts processing the stream and loading data into a buffer for processing by the operating system.

This is known as the Information or IP protocol and it is the fundamental protocol of the internet. Every packet starts with a set of information that allows packets to be routed along the way to their intended recipient.

**Transport Layer** While the internet layer provides nice things like routing and atomicity for chunks of data from a given device, the IP layer really has no ability to:

1. Ensure that one party doesn't overwhelm the other party with too much data if one of the devices is too slow to keep up with the rate of the other device.
2. Ensure that data packets sent to the remote server actually arrive there and are re-sent as necessary
3. Ensure that data packets arrive in order and are reconstructed to be in order on the destination side if necessary
4. Modify the transmission rates of packets if high network latency or data loss is detected - i.e. "congestion control"

This is where the TCP protocol comes in. It allows us to send data between two devices with very good reliability at the cost of speed. It is the fundamental transport layer protocol for most data transmission on the internet where data integrity is important. With TCP we are essentially able to send messages - actual human readable text-messages between two devices and that is how most software programs like webservers and web-browsers talk to each other - on top of the TCP protocol using normal human-readable text.

**Application Layer** Web browsers and web servers talk to each other in what looks like text messages. This is the HTTP protocol. Typically when a web browser connects to a web server, it establishes a TCP connection and sends a text message like:

```
GET /index.html HTTP/1.1
 Host: google.com
```

Completely human-readable, right? No wierd codes or symbols, just text. The first line is the browser saying "get me a document called index.html, by the way I am speaking in HTTP protocol version 1.1". The second line is telling the server "Oh and the domain that the user entered into the URL bar is google.com". This will be important later when understanding how multiple domains can live on the same IP address. Then your server might respond with something like:

```
HTTP/1.1 200 OK
 status: 200 OK
 version: HTTP/1.1
 content-length: 28278
 content-type: text/html;
 date: Sat, 18 Aug 2012 00:26:50 GMT
```

Which is the server's way of saying:

1. I too speak HTTP version 1.1, I am returning status code 200 which means that I was able to locate the document you were requesting.
2. By the way, the status was 200 OK, just in case you forgot
3. And also again, the version of HTTP is 1.1
4. The content length is 28278 bytes, meaning that's how much data you should expect from me.
5. The content type is text/html. Get ready to parse it!
6. And here's today's date.

After this what's called the response header, the server will send the contents of the file I requested to my browser. My browser will hide the header from me since it isn't important and render the HTML according to the rules of how HTML is supposed to work.

## 1.2 How InterWorx ships the Apache Webserver

When you install InterWorx, the installer will remove the version of Apache that may have shipped with your operating system and install the one which is built and maintained in the InterWorx repositories by InterWorx proper. We do this for two reasons:

1. Ability to set certain build parameters to settings that work with InterWorx's docroot structure
2. We don't have to rely on the OS maintainers for web server updates.

### 1.2.1 Multi Processing Module

Apache is highly modular. So modular in fact, you can choose how apache deals with multiple HTTP connections and requests at the same time. The module that controls this behavior is called the Multi Processing Module or MPM. There are a wide variety of MPM's for Apache that target different applications of the web server. Some are operating-system related such as the MPM that is used when you run Apache in a Windows environment (yes, people use Windows to do webhosting occasionally). There are some MPM's which run Apache in a threaded/hybrid process mode where the master apache process spawns children that are multithreaded and can hande multiple connections simulatneously. The MPM that InterWorx ships with and is compatible with is called PREFORK. This MPM uses a model of a master supervisor Apache process that is started and run as root so it can bind to TCP ports 80 and 443. Then it spawns children processes that run as the Apache user and they each service maximum 1 connection at a time. The benefit of this is additional stability. A multi-threaded setup with a crash in a single thread can kill an entire process that is serving multiple connections. A single-threaded setup will localize any crashes to a single process so other users are not effected by the crash. In addition, certain modules are not compatible with the multi-threaded MPM.

### 1.2.2 The NameVirtualHost System

InterWorx supports having multiple domains on the same IP address through Apache's NameVirtualHost system. When a request comes in, the HTTP request always includes the domain being requested from the server, as shown back in section 1.1. The "Host:" header will include the domain entered in the URL bar which allows Apache to select from the domains in it's configuration the correct files to load from the disk and the correct configuration settings to use. Of course, the NameVirtualHost system relies on the browser sending the "Host:" the user entered, which typically requires that DNS be setup and working properly in order for the user to enter a given domain and have it connect to the webserver on the correct IP address. This means that the NameVirtualHost system is reliant on DNS properly being setup for the public internet to visit the domain with ease. You are always able to set your local computer's HOSTS file to override default DNS lookups for certain domains.

# Chapter 2

# Definition of Terms

To avoid confusion, let's get some terminology out of the way.

**Domain** at a basic level is a string of text that refers to an IP address. In InterWorx, the requirement for a domain when creating a SiteWorx account is that the domain have a top-level domain (TLD) (the TLD is not checked for validity so it can be a non-existant TLD), and a second-level domain. "google.com", "internet.site", "very.many.sub.domains.com" are all valid domains that InterWorx will accept as a SiteWorx domain name. "website", "bad,charachters", and "-invalid.com" are not valid because they either do not have a period delimiting the TLD or they contain characters which are illegal in a domain name.[1]

**Website** refers to the domain and content that is served over HTTP/S by the webserver.

**Pointer Domain** is and alias for another domain. The purpose of the domain is to point to the same resource that another domain points to. For example, fb.me and facebook.com go to the same site, so fb.me is a pointer domain to facebook.com.

**Sub Domain** adds another "level" after the domain in order to segregate some part of a domain inside of a second-level or higher-level domain zone. For example, InterWorx wanted to segregate the documentation for the product from the main site that markets and educates about InterWorx, so we added a subdomain for the interworx.com zone to docs.interworx.com.

**SiteWorx Account** essentially refers to the web hosting account. It encompasses multiple domains, sub-domains, pointer domains, etc as well as email and databases for those domains.

**SiteWorx Domain** refers to a domain that is inside of a SiteWorx account. A SiteWorx account must contain at minimum 1 SiteWorx domain - the master domain that you enter when creating the account.

**DocumentRoot or Doc Root** refers to directory on the filesystem from where files are served from for a given domain.

**Master Domain** is the SiteWorx domain that is defined when a SiteWorx account is first created. This becomes the domain that is used to refer to that SiteWorx account within the panel and on command-line. It cannot be removed without deleting the SiteWorx account - the Master Domain and the Account go hand-in-hand as it is not possible to have a domain-less SiteWorx account.

**Secondary Domain** is a SiteWorx domain that is not the primary domain. It gets it's own set of email accounts, it's own segregated doc root on the file system, its own DNS zone, its own server logs, and its own web stats. The secondary domain does still live under the same linux user as the master domain, though. You can see later why this might be useful.

**VirtualHost or vhost** refers to the configuration file (and more specifically, the chunk of text between the opening <VirtualHost> and closing </VirtualHost> tags) for the webserver that allows the site to be available on a given IP address.[2]

[1] Wikipedia's article helps explain more: http://en.wikipedia.org/wiki/Domain_name

[2] You can learn more about Virtual Hosts at http://httpd.apache.org/docs/2.2/vhosts/

**ServerAlias** is a directive that goes inside a vhost that allows multiple domains to link to the same virtual host. This is how sub domains and pointer domains are made to link to the same website.[3]

[3] Server aliases can be read about here: http://httpd.apache.org/docs/2.2/mod/core.html#serveralias

# Chapter 3

# The NodeWorx Webserver Management Interface

The Webserver Management page is available at NodeWorx ▷ System Services ▷ Web Server. Keep in mind that this guide is primarily written for the Apache web server, and certain features that do not translate to LiteSpeed will not be available. This page allows you control some of the more important facets of the InterWorx webserver configuration, such as its state, and settings which dictate performance and features.

**Service Control** Much like other service control pages inside NodeWorx, InterWorx provides the ability to start/stop/restart the service which will control the server daemon via the /etc/init.d/httpd service control script as seen in figure 3.1. In addition, InterWorx allows you to set whether the server starts on boot, and whether to restart the server if it goes down. In most cases, you will probably want these both set to yes unless your InterWorx machine is not acting as a webserver.

**Webserver Management Page RRD Graph** Additionally, you can also view the Webserver RRD graph, which shows the number of processes currently running for the webserver, and the maximum allowed. This can help diagnose issues where connections are being refused or taking too long. An example graph is shown in figure 3.2.

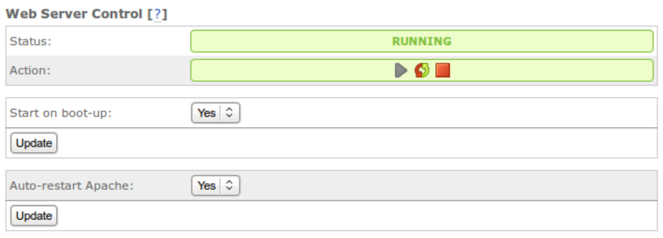
## 3.1 Web Server Information Box and Logs

The webserver information box provides the server's version number, Apache config file checks, links to the Server logs, and links to the Server Status and Server Info pages. This is shown in figure 3.3.

### 3.1.1 Apache httpd.conf Syntax checker and Editor

The Apache configuration checker uses Apache's built in syntax check to do a full parse and verification of all the entire configuration of the webserver. This includes all the configuration files loaded from /etc/httpd/conf.d/*.conf. If there is an issue, the interface will warn you here. Keep in mind syntax errors will prevent the server from starting up,

Figure 3.1: This is the interface which allows you to control the webserver's status.



Web Server Control [?]

| | |
|---|---|
| Status: | RUNNING |
| Action: | |
| Start on boot-up: | Yes |
| Update | |
| Auto-restart Apache: | Yes |
| Update | |

Figure 3.2: The Web Server RRD Graph, showing number of processes and a line showing the maximum number of processes allowed. Max clients corresponds to the maximum number of processes permitted by the current settings.

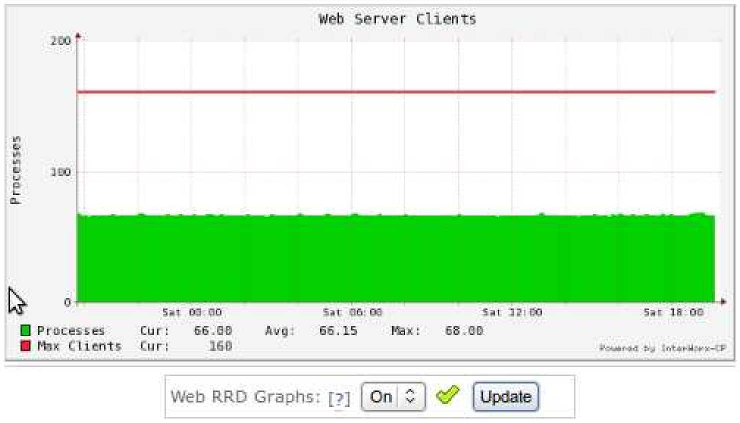
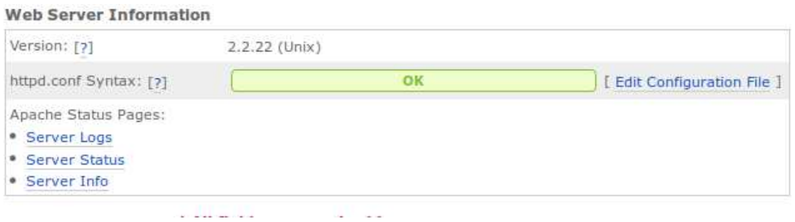


Figure 3.3: The Web Server Information Box provides basic stats and gives you access to your server's configuration file.



**Web Server Information**

| | | |
|---|---|---|
| Version: [?] | 2.2.22 (Unix) | |
| httpd.conf Syntax: [?] | OK | [ Edit Configuration File ] |

Apache Status Pages:
- Server Logs
- Server Status
- Server Info

which means your server won't come back up if it goes down (this occurs regularly during statistics generation and log rotation). The editor allows you to edit the Apache server's main configuration file, /etc/httpd/conf/httpd.conf, and modify settings directly in the file as opposed to using the interface. The advantage is that you can modify settings that aren't specifically available for modification through the interface. An example is configuring the mod_status /server-status page and allowing it to be viewed by the local server[1].

## 3.2 Apache Logs

The Server Logs link takes you to the log viewer portion of the control panel which is shown in figure 3.4. Server logs are the critical for troubleshooting webserver issues, and we allow you to view the 4 global system-wide logs here. These logs are located on the system in /var/log/httpd/.

**access_log** provides information about general traffic to your webserver. Keep in mind that traffic that goes to SiteWorx domains will not be logged here as each SiteWorx domain has its own set of logs in /home/<linux user>/var/<domain>/logs. For the most part, you will see traffic from users that access your server directly by IP and the mod_watch flush operation that InterWorx performs every 5 minutes.

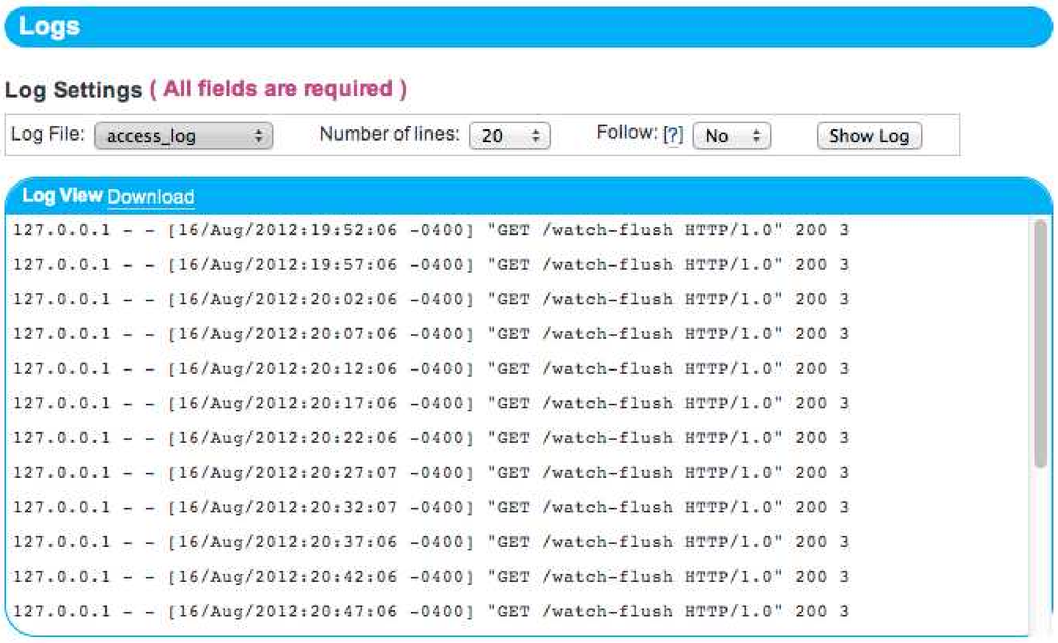
**error_log** is where server-wide errors and warnings go. Again, SiteWorx domains have their own error logs for when things go wrong with their PHP scripts or some other CGI program. Yet if you are having issues like the server crashing or the server having issues starting up, this log will likely contain the culprit. Essentially, any errors not directly related to a SiteWorx domain end up here.

**ssl_request_log** is where globally all access to SSL (port 443) is recorded.

**suexec.log** is the log that contains logged information from the suexec module. This module is responsible for changing a processes userid and groupid to the linux user of the SiteWorx account prior to executing any CGI program.

[1] http://httpd.apache.org/docs/2.2/mod/mod_status.html

Figure 3.4: The webserver log viewer, access_log is shown.



Log File: access_log  Number of lines: 20  Follow: [?] No

```
127.0.0.1 - - [16/Aug/2012:19:52:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:19:57:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:02:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:07:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:12:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:17:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:22:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:27:07 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:32:07 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:37:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:42:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
127.0.0.1 - - [16/Aug/2012:20:47:06 -0400] "GET /watch-flush HTTP/1.0" 200 3
```

Normally if say, the userid of a perl script is UID 0 (root), then suexec will refuse to execute the script, and that would be recorded here.

The log viewer lets you control how many lines of the log you want to view and additionally - you can elect to follow the log which means as new lines are added to the log by the server, you will see them updated in the interface in real time.

## 3.3 Status Pages

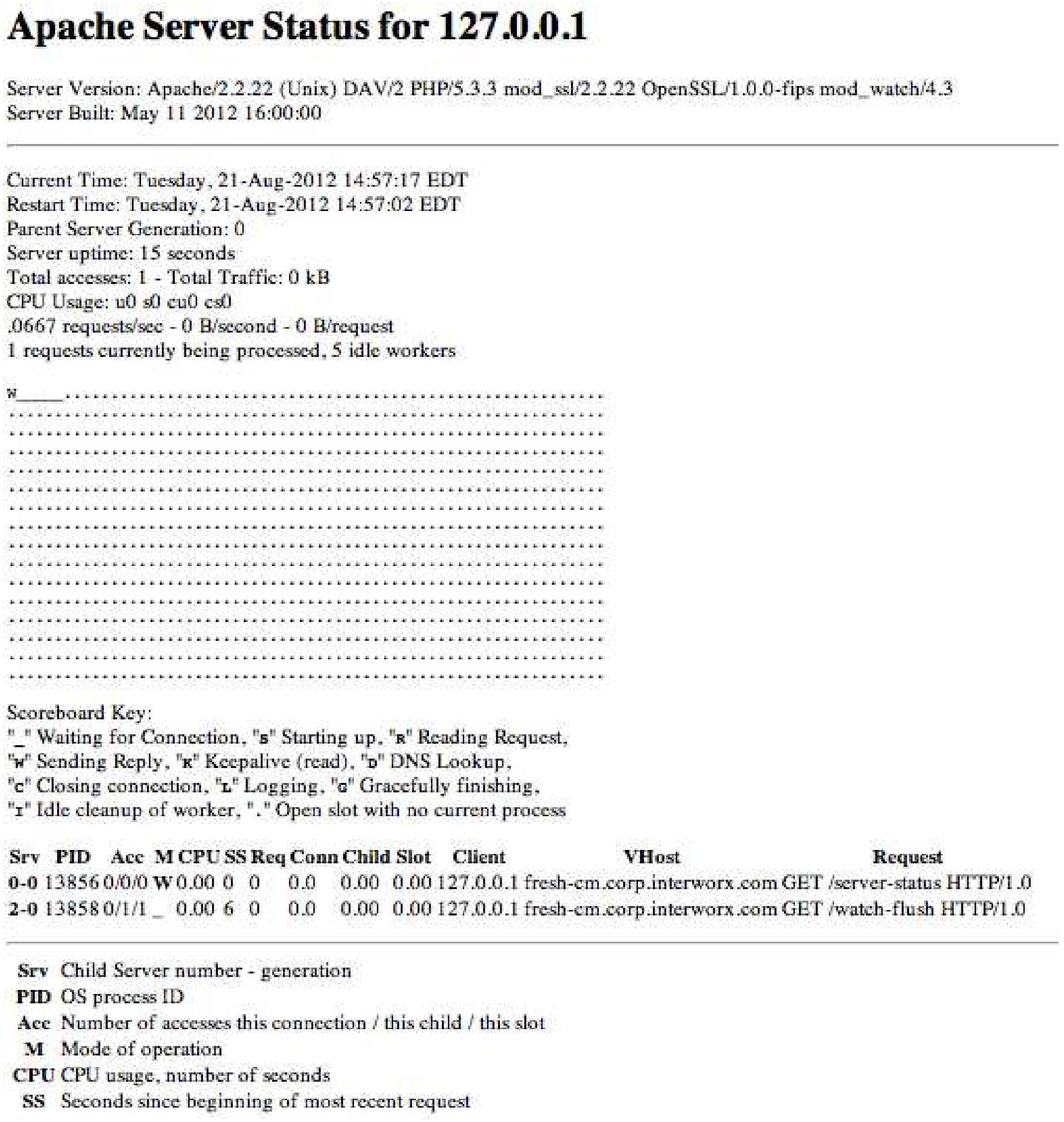
The modules, status_module and info_module, provide comprehensive data about the configuration and performance of the server. They do this by providing a dynamically generated web page on http://<your ip>/server-status and http://<your ip>/server-info, respectively. By default, these pages are not visible from outside the server due to security concerns. They could lead an attacker to gain compromising information about your server, leading to a break-in. In order for you to be able to see these pages, even from within the safety of the control panel, you must modify your server's Apache configuration.

### 3.3.1 /server-status

In order to enable the server status page, edit your system's httpd.conf (which is possible on the Webserver Management page in NodeWorx), and locate the portion of your configuration that looks like:

```
#
# Allow server status reports generated by mod_status,
# with the URL of http://servername/server-status
# Change the ".example.com" to match your domain to enable.
#
#<Location /server-status>
#    SetHandler server-status
#    Order deny,allow
#    Deny from all
```

Figure 3.5: The Apache Server-Status Page



# Apache Server Status for 127.0.0.1

Server Version: Apache/2.2.22 (Unix) DAV/2 PHP/5.3.3 mod_ssl/2.2.22 OpenSSL/1.0.0-fips mod_watch/4.3
Server Built: May 11 2012 16:00:00

---

Current Time: Tuesday, 21-Aug-2012 14:57:17 EDT
Restart Time: Tuesday, 21-Aug-2012 14:57:02 EDT
Parent Server Generation: 0
Server uptime: 15 seconds
Total accesses: 1 - Total Traffic: 0 kB
CPU Usage: u0 s0 cu0 cs0
.0667 requests/sec - 0 B/second - 0 B/request
1 requests currently being processed, 5 idle workers

```
W_____..........................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

| Srv | PID | Acc | M | CPU | SS | Req | Conn | Child | Slot | Client | VHost | Request |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0-0** | 13856 | 0/0/0 | **W** | 0.00 | 0 | 0 | 0.0 | 0.00 | 0.00 | 127.0.0.1 | fresh-cm.corp.interworx.com | GET /server-status HTTP/1.0 |
| **2-0** | 13858 | 0/1/1 | _ | 0.00 | 6 | 0 | 0.0 | 0.00 | 0.00 | 127.0.0.1 | fresh-cm.corp.interworx.com | GET /watch-flush HTTP/1.0 |

---

| | |
|---|---|
| **Srv** | Child Server number - generation |
| **PID** | OS process ID |
| **Acc** | Number of accesses this connection / this child / this slot |
| **M** | Mode of operation |
| **CPU** | CPU usage, number of seconds |
| **SS** | Seconds since beginning of most recent request |

```
#    Allow from .example.com
#</Location>
```

and uncomment the <Location> directive block. You also want to allow the localhost server permission to view the server-status page. The reasoning is that this will still prevent any arbitrary user to view the server-status page while still allowing the page to be visible via the control panel. The resulting configuration block will now look like this:

```
#
# Allow server status reports generated by mod_status,
# with the URL of http://servername/server-status
# Change the ".example.com" to match your domain to enable.
#
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>
```

After you save and restart the server, you should see a similar page as shown in figure 3.5 when you click the “Server Status” link on the NodeWorx Webserver Management page. If you wish to see additional information on a per-process basis, you can enable extended status by adding a few lines to your server’s configuration:

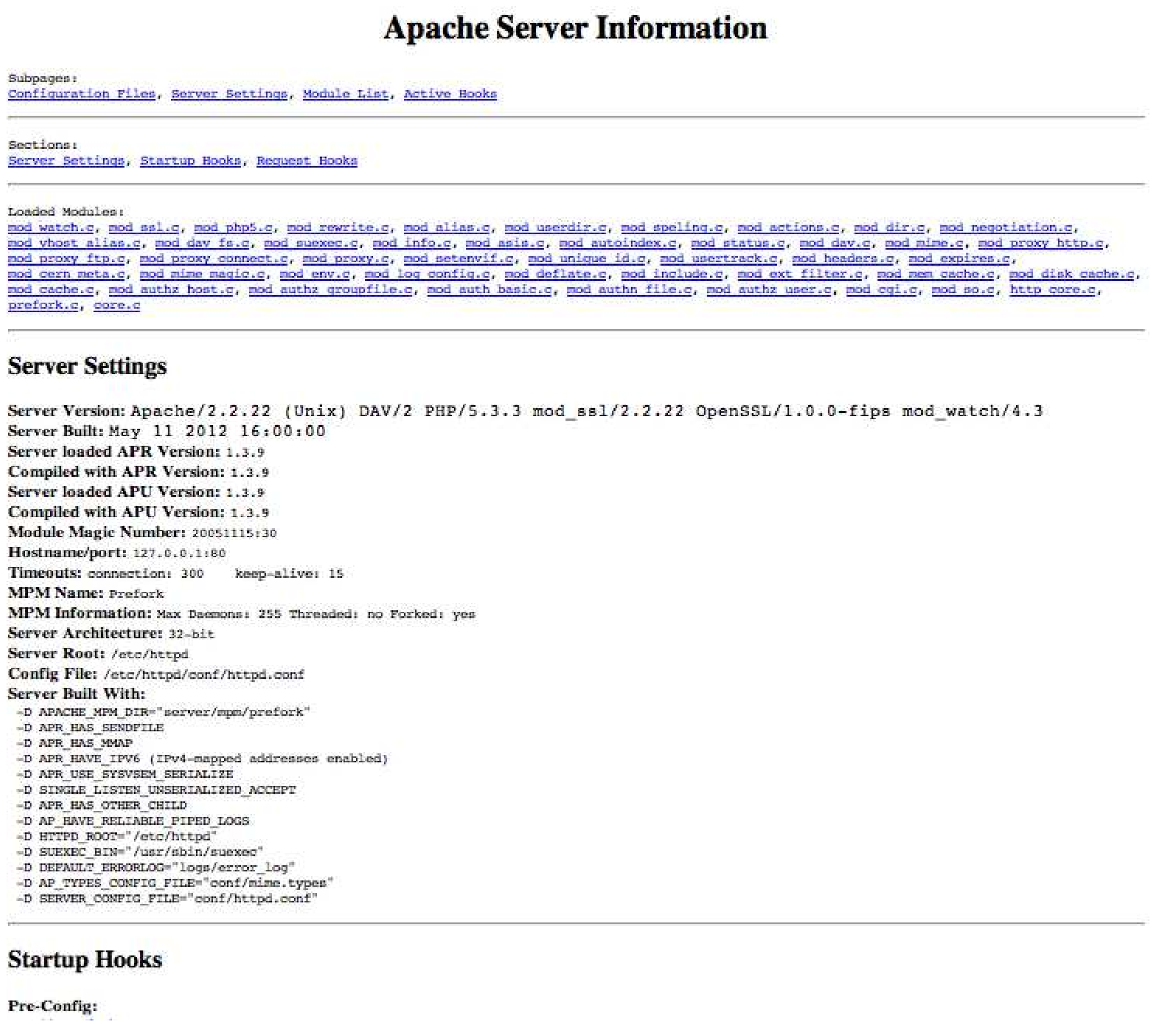
```
#
# Allow server status reports generated by mod_status,
# with the URL of http://servername/server-status
# Change the ".example.com" to match your domain to enable.
#
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>

<IfModule mod_status.c>
    ExtendedStatus on
</IfModule>
```

### 3.3.2 /server-info

The server-info page is shown in figure 3.6 and will give you a detailed description of the server’s configuration, build parameters, and loaded modules. Like the status_module’s /server-status page, your httpd.conf’s configuration must be modified in order to make this page visible anywhere. Again, we recommend you make it visible only to the localhost server and the control panel will allow you to see the info page through the NodeWorx Webserver Management page. Find this portion of your configuration:

```
#
# Allow remote server configuration reports, with the URL of
#  http://servername/server-info (requires that mod_info.c be loaded).
# Change the ".example.com" to match your domain to enable.
#
#<Location /server-info>
#    SetHandler server-info
#    Order deny,allow
#    Deny from all
```

Figure 3.6: The server-info page



## Apache Server Information

Subpages:
Configuration Files, Server Settings, Module List, Active Hooks

Sections:
Server Settings, Startup Hooks, Request Hooks

Loaded Modules:
mod watch.c, mod ssl.c, mod php5.c, mod rewrite.c, mod alias.c, mod userdir.c, mod speling.c, mod actions.c, mod dir.c, mod negotiation.c, mod vhost alias.c, mod dav fs.c, mod suexec.c, mod info.c, mod asis.c, mod autoindex.c, mod status.c, mod dav.c, mod mime.c, mod proxy http.c, mod proxy ftp.c, mod proxy connect.c, mod proxy.c, mod setenvif.c, mod unique id.c, mod usertrack.c, mod headers.c, mod expires.c, mod cern meta.c, mod mime magic.c, mod env.c, mod log config.c, mod deflate.c, mod include.c, mod ext filter.c, mod mem cache.c, mod disk cache.c, mod cache.c, mod authz host.c, mod authz groupfile.c, mod auth basic.c, mod authn file.c, mod authz user.c, mod cgi.c, mod so.c, http core.c, prefork.c, core.c

## Server Settings

**Server Version:** Apache/2.2.22 (Unix) DAV/2 PHP/5.3.3 mod_ssl/2.2.22 OpenSSL/1.0.0-fips mod_watch/4.3
**Server Built:** May 11 2012 16:00:00
**Server loaded APR Version:** 1.3.9
**Compiled with APR Version:** 1.3.9
**Server loaded APU Version:** 1.3.9
**Compiled with APU Version:** 1.3.9
**Module Magic Number:** 20051115:30
**Hostname/port:** 127.0.0.1:80
**Timeouts:** connection: 300    keep-alive: 15
**MPM Name:** Prefork
**MPM Information:** Max Daemons: 255 Threaded: no Forked: yes
**Server Architecture:** 32-bit
**Server Root:** /etc/httpd
**Config File:** /etc/httpd/conf/httpd.conf
**Server Built With:**

```
 -D APACHE_MPM_DIR="server/mpm/prefork"
 -D APR_HAS_SENDFILE
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_SYSVSEM_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D HTTPD_ROOT="/etc/httpd"
 -D SUEXEC_BIN="/usr/sbin/suexec"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D AP_TYPES_CONFIG_FILE="conf/mime.types"
 -D SERVER_CONFIG_FILE="conf/httpd.conf"
```

## Startup Hooks

**Pre-Config:**

```
#       Allow from .example.com
#</Location>
```

and uncomment the <Location> block, replacing Allow from with localhost. The resulting configuration should look like:

```
#
# Allow remote server configuration reports, with the URL of
#  http://servername/server-info (requires that mod_info.c be loaded).
# Change the ".example.com" to match your domain to enable.
#
<Location /server-info>
    SetHandler server-info
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>
```

The /server-info page should now be accessible in the Webserver Management interface in NodeWorx by clicking the “Server Info” link.

# Chapter 4

# Web Server Options

This part of the interface lets you set some of the main settings of the server that essentially dictate your server's performance and ability to handle traffic. We will explain the settings in detail here.

**HTTP Port** This is the TCP port on the server that the server will bind to on startup for HTTP connections. 99% of the time, this should be set to port 80 as the HTTP protocol spec dictates that the server be on port 80 and browsers by default connect to HTTP servers on port 80. The exception here might be if you are going to setup a proxy in front of the webserver on port 80 - such as a varnish cache or nginx. That is outside the scope of this documentation.

**HTTPS Port** The HTTPS port that the webserver will listen on for HTTPS requests. Again, as with HTTP port, you will probably want this set to 443 unless you are putting some sort of proxy infront of Apache.

**Server Limit** This setting limits the maximum number of processes you are allowed to set with the "Max Clients" setting. The thinking here is that if the configuration is re-loaded while the server is still live, you are allowed to change the Max Clients setting, but not the Server Limit setting. This will ensure that while the server is not fully restarted, the Max Clients setting cannot be raised above this limit. If you exceed Max Clients beyond this limit, you'll get the something like this error:

```
   WARNING: MaxClients of 750 exceeds ServerLimit value of 256 servers,lowering MaxClients
to 256.  To increase, please see the ServerLimit directive.
```

**Max Clients** This controls the maximum number of processes that are allowed to be spawned by the Apache server. InterWorx ships Apache with the MPM prefork module, so each process is only capable of handling 1 HTTP request/connection. This means this limit essentially sets the maximum the number of simultaneous connections to the webserver. Keep in mind that raising this value will increase the CPU and Memory demands of the entire Apache process group and thus some consideration should be put into the hardware capabilities of your server prior to increasing this value outright. Also keep in mind that the "Server Limit" value must be increased too if you want to increase Max Clients higher than the value of Server Limit.

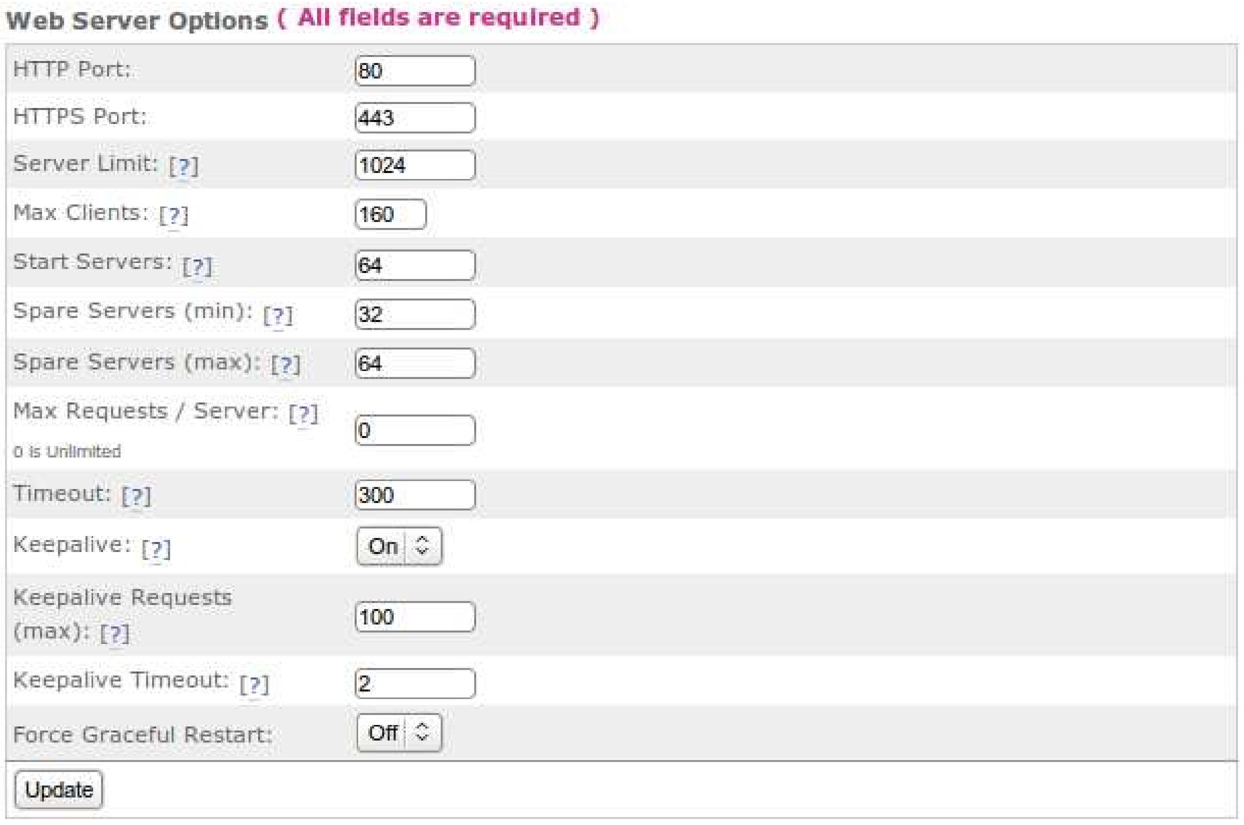
**Start Servers** From the Apache Documentation:

> The StartServers directive sets the number of child server processes created on startup. As the number of processes is dynamically controlled depending on the load, there is usually little reason to adjust this parameter. [1]

If you have very tight system resources on a VPS and super low web traffic, it might make sense to lower Start Servers to reduce the footprint of the Apache process group. Otherwise there's not much point as Apache spawns processes on demand as needed. Apache even will spawn processes and have them sit waiting idle if your traffic load goes up, just to ensure that new connections are not waiting on a process to spawn.

[1] http://httpd.apache.org/http://httpd.apache.org/docs/2.2/mod/mpm_common.html#startservers/2.2/mod/mod_status.html

Figure 4.1: The Web Server Options box provides a graphical interface for the most important Apache configuration settings.



**Web Server Options** ( All fields are required )

| | |
|---|---|
| HTTP Port: | 80 |
| HTTPS Port: | 443 |
| Server Limit: [?] | 1024 |
| Max Clients: [?] | 160 |
| Start Servers: [?] | 64 |
| Spare Servers (min): [?] | 32 |
| Spare Servers (max): [?] | 64 |
| Max Requests / Server: [?] 0 is Unlimited | 0 |
| Timeout: [?] | 300 |
| Keepalive: [?] | On |
| Keepalive Requests (max): [?] | 100 |
| Keepalive Timeout: [?] | 2 |
| Force Graceful Restart: | Off |

**Spare Servers (min)** This is the minimum number of idle apache server processes that apache will have running at once. Idle means the process is spawned but there are no connections or requests coming in for it to service. Apache does this to always have extra processes available to handle new requests so there is little to no delay waiting on a new process to spawn. If the number of idle processes drops below this number, Apache will spawn new ones at the rate of 1 per second until the minimum spare server is reached. It is generally not recommended that this number be increased to high because the group of apache processes will begin consuming a large portion of memory and CPU, possibly deteriorating your machine's performance or causing instability.

**Spare Servers (max)** This controls the maximum number of idle apache server processes. If the number of idle apache processes goes higher than this value, apache will begin killing idle processes until this number is reached. This number should always be higher than Spare Servers (min), or apache will automatically set Spare Servers (max) to the (min) value plus one. Again, setting this value high is a bad idea. You don't want your traffic to drop and then have a bunch of Apache processes sitting around consuming resources that the other busy apache processes and system services could be using.

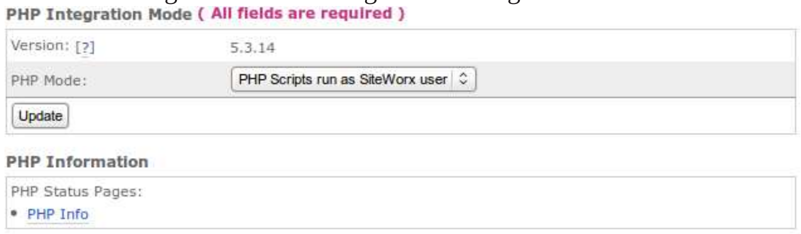
**Max Requests / Server** This setting controls how many requests an individual trial process may handle before being killed. With the prefork MPM (which is what InterWorx Apache uses), this essentially amounts to how many different connections a child process will handle before the master process steps in and kills the child. With KeepAlive turned on, only the first requests counts against this value, and thus this setting is truly unique connections per child process. Setting this to 0 will make the value unlimited, which is probably somewhat desirable. The impetus for manipulating this value is if you perhaps installed a module that might be leaking memory over time in which case this value would effectively limit the lifetime of a child process to a certain number of connections.

**Timeout** Timeout affects the timeout values for a multitude of different parts of the webserver. It's value is in seconds.

- When waiting on the HTTP client to send data, this is the maximum amount of time that the server will wait on the client to send data.
- When sending data to the HTTP client, this is the maximum amount of time that the server will wait on the client to acknowledge (via TCP) that the data has been recieved successfully.
- With mod_cgi, this controls how long to wait for the CGI script to return data
- With mod_ext_filter, this controls how long to wait on a "filtering process" - or program that intercepts the outgoing data and manipulates it. In most cases this is not extremely applicable because most hosts do not run external program filters. An example of a reason this filter module might be used is to append javascript advertising code to the outgoing HTML of a website if you offer free webhosting and want to advertise on the user's sites.
- With mod_proxy, this is the default timeout value if the ProxyTimeout value is not set. Proxying is used to have the webserver connect to another HTTP server and act as a middleman between the HTTP client and the endpoint HTTP server.

**Keepalive** Keepalive enables the persistent connection feature of HTTP/1.1. This essentially allows a client to issue multiple requests utilizing the same HTTP connection. In HTTP/1.0, traditionally the client would have to make a new TCP connection per HTTP request they wanted to issue. This resulted in a lot of wasted overhead and CPU cycles on both the client and server side constantly bringing up new connections. The persistent connection feature allows the server to "feel faster" to the HTTP client as once a connection is established an entire site can be pulled down via a single TCP connection. Most sites these days have 10-20 assets per page load that each need an individual HTTP request to pull down. Keepalive is therefore recommended to provide the best experience to the enduser as it results in faster page loads.

Figure 4.2: PHP Integration Settings and Control



PHP Integration Mode ( All fields are required )

Version: [?] 5.3.14

PHP Mode: PHP Scripts run as SiteWorx user

PHP Information

PHP Status Pages:

- PHP Info

**Keepalive Requests (max)** With keepalive enabled, this limits the maximum number of HTTP requests that can be issued per TCP connection. It is recommended that this be a high-ish value for the best server performance. The default of 100 is typically fine for most hosts. Once an HTTP client on a single TCP connection hits the max keepalive requests, it must establish a new connection before continuing to issue new requests. 0 is unlimited.

**Keepalive Timeout** The amount of time in seconds that a child process with Keepalive will wait for a subsequent request before terminating the connection. By default, it is 5 seconds. This usually is enough to keep a connection alive for a page load or two, but short enough that a child process isn't waiting around longer than necessary for an idle HTTP client that isn't going to issue more requests. On busy servers it might help to lower this slightly as for a page load, browsers will issue HTTP requests as fast as they are able to, and thus typically a 5 seconds is much longer than necessary. On the other hand, 5 seconds does allow for some leniancy for high latency clients.

## 4.1 Force Graceful Restart

This setting does not manipulate any configuration directive in /etc/httpd/conf/httpd.conf per say, but it does control how InterWorx issues restarts to the webserver. By default, InterWorx does a full restart by bringing down the apache process tree, and then starting it back up. In technical terms, InterWorx will issue the SIGTERM signal to the master apache process. The issue with this on servers with a lot of virtualhost configurations, (i.e. thousands of SiteWorx domains), this process might take a bit longer than necessary because it takes a long time for the server to completely process the entirety of it's configuration during startup. As a result, InterWorx offers the abilty to do a Graceful restart, which sends a USR1 signal to the master apache process. Then the following occurs:

1. The master apache will then *advise* it's children to terminate when they are done handling the current HTTP request. This allows any currently active HTTP clients to continue accessing the webserver until they are done.

2. The master apache process re-reads the entire webserver configuration and re-opens all log files. This allows for log rotation to occur if necessary.

3. As children processes die off, the master apache process will replace them with a new generation of children that have the updated configuration settings.

This allows for a seamless restart without down time. The down side is that:

- Certain settings, such as ServerLimit cannot be modified without a full restart. This might be an issue if you need to up this value.

- The old generation children might still be writing to logs, so log rotation has no guarantee of whether it is safe to move logs around. As a result log rotation should wait an unspecified amount of time before trying to move logs around.

- /server-status pages will not be zeroed out or cleared.

## 4.2 PHP Settings

As PHP is the most popular web application programming language on the web, InterWorx mostly focuses on the integration of PHP and Apache as the main methodology for offering dynamic web application support to end users. The PHP Integration settings area of the Webserver Management Interface gives some basic control over PHP integration and is shown in figure 4.2.

**PHP Version** displays the version of PHP installed on the system that SiteWorx accounts will use and have access to.

**PHP Mode** controls how PHP is integrated into the webserver software. The two choices, "PHP Scripts run as apache" corresponds to mod_php and "PHP scripts run as SiteWorx user" corresponds to suPHP. The two differ in that mod_php has the PHP interpreter integrated into the webserver process while suPHP calls the PHP executable externally and runs it as the linux user associated with the SiteWorx account. This topic is covered in more depth in Chapter 8.

**PHP Info Status Page** will show you the current PHP server configuration as visible when the phpinfo() function is called. It will display the current setting from the /etc/php.ini configuration file, as well as all the PHP extentions that are loaded.

# Chapter 5

# Apache Modules

This lists the Apache modules installed on the server and their status. It also allows you to enable, disable and remove Apache modules from the webserver configuration. Here we will cover the purpose of the modules that ship with a default InterWorx install. Most modules are not readily used by the default system configuration, but are provided such that SiteWorx users are able to use the module's directives locally to their site in .htaccess files.

**actions_module** allows you to set certain actions to occur when a given MIME content type is requested with the Action directive. For example, you could have a CGI script execute everytime a GIF image is requested. Additionally, you can use the Script directive to override the default handling of certain HTTP requests (GET/HEAD/POST) and have a script execute instead.

**alias_module** allows you to map different URLs to different parts of the filesystem, possibly outside the docroot of the account. For example, you can make it so anything requested from http://domain.com/files loads from /mnt/disk2/files. In addition, you can do simple redirects when certain resources are requested.

**asis_module** allows you to send data to the HTTP client sans the normal HTTP headers that are included with a server response. This means in order to conform to the HTTP protocol, the file being served must include HTTP headers itself or the browser will be unable to process the server's response. Useful if you want to server flat files with headers embeded in them already. The server will add a Date: and Server: header to the response.

**auth_basic_module** This module allows you to restrict access to content. It's sole purpose is to act as a front-end configuration for dictating which URL's are restricted and how they are restricted, while actual authentication is handled by another module.

**authn_file_module** provides flat-file authentication to the authn_basic_module. Essentially this is when you have a .htpasswd file with username, colon, and encrypted password. Typically this file is used in conjunction with the htpasswd command to generate users and encrypted passwords on the fly.

**authz_groupfile_module** allows you to specify groups of users in a flat file, and then grant authentication to users based on what group they are in.

**authz_host_module** allows you to allow and deny access based on host. This is heavily used across the webserver configuration to limit certain parts of the filesystem and virtual pages to only specific clients, notably the localhost server. The directives are the popular Allow From and Deny From.

**authz_user_module** allows you to restrict certain URLS to specific users, or require a valid-user to log in.

**autoindex_module** is the module responsible for creating directory listings when a user visits a directory on the server that does not contain an index.html or index.php. This module allows apache to generate a directory listing on the fly, possibly excluding hidden files and also inserting or appending code to make the generated page look nice.

**cache_module** is a module which allows fine-grained control over the sever-side caching of content that is frequently accessed. The cache module integrates with disk_cache_module and mem_cache_module to provide 2 different caching abilities to the website administrator for caching.

**cern_meta_module** is a module that is primarily used by the CERN community to add additional meta headers to an HTTP response that are CERN-specific. This allows Apache to emulate CERN HTTPD file semantics.

**cgi_module** allows for the execution of CGI scripts for files that have the handler cgi-script. CGI scripts essentially recieve web-server data as part of their environment variables and POST data as part of their stdin input. They return data as part of their stdout output back to the webserver which the server sends back to the user.

**cgid_module** is a CGI module designed for a multi-threaded MPM such as worker. Instead of forking a multi-threaded Apache process which is very expensive, a daemon standsby which is what is used to fork and execute external CGI scripts. This is probably not going to be used by your server.

**dav_module** provides WebDAV or Web-based Distributed Authoring and Versioning for Apache. This essentially provides some sort of versioning system in Apache using the HTTP protocol by creating, moving, copying and deleting resources with HTTP requests specific to WebDAV.

**dav_fs_module** is a module responsible for interacting with dav_module and providing filesystem-resources to that module. This is a support module.

**deflate_module** provides compression utility support to Apache such that files can be compressed before being sent over the network to the client.

**dir_module** is the module that controls what the index file of a directory is (typically index.html or index.php). In addition it allows autoindex_module to be used in place where a specific index file is not user-defined.

**disk_cache_module** is a support module for cache_module that provides disk-based caching. This may sound like a bad idea, but if processing needs to occur on files per-request basis, the cache can store a processed version of a file and serve that as opposed to re-doing expensive processing. In addition, the operating system and filesystem cache often used files in memory to speed up access. Therefore, a heavily used disk cache is probably being serviced from system memory more often than the actual secondary storage.

**env_module** is a module which allows you to manipulate the system environment variables the httpd process is running with. In addition it allows you to control what environment variables are passed to CGI scripts which are executed.

**expires_module** works in conjunction with Cache-Control HTTP header to tell the client-side browser when content should be retrieved from the server again as opposed to being loaded from the browser cache. Allows web developers to have control over how long a client-side browser caches images and scripts from the Apache server.

**ext_filter_module** allows you to run a program to filter input and ouput in and out of the webserver. This allows you to add filters just by making a program that accepts arbitrary outgoing HTML in a program's stdin input, applying some text manipulations, and sending the webserver your filtered output via stdout. This could be used to, for example, append advertisements to all HTML coming out of your server for a free webhosting service.

**headers_module** is the module that provides the ability to control and manipulate HTTP request and response headers.

**include_module** according to Apache's documentation:

> This module provides a filter which will process files before they are sent to the client. The processing is controlled by specially formatted SGML comments, referred to as elements. These elements allow conditional text, the inclusion of other files or programs, as well as the setting and printing of environment variables.

This essentially allows for config-level filtering of content without having to explicitly write an external program to apply text transformations.

**info_module** is the module that provides the Apache /server-info page. This module can be used to provide information about your server from an HTTP-accesible web page.

**log_config_module** is the module which allows you to configure the output format of the logs, where they are stored, and what kind of information they contain. This is used heavily by the default InterWorx configuration to control where logs live for each SiteWorx account.

**mem_cache_module** is a support module for mod_cache. This module provides support for in-memory caches for Apache. According to the Apache documentation, mod_disk_cache is preferred as this cache will actually be per-process which is not idea for the prefork MPM.

**mime_magic_module** This module is responsible for determining the content type of a file, much like the file command of the unix operating system. It reads the first bytes of a file and looks at magic numbers to try and determine what kind of file is being opened.

**mime_module** is the module which links a file's filename (with extension) to meta information about the file - such as what it's character set is, what it's encoding is, what language it's in, etc. This module can also add handlers and filters to specific mime types based on filename extension.

**negotiation_module** is a content-negotation module that is used to select a document that best matches the capability of a client browser from several available documents. It has a couple different ways to implement content negotiation. The documentation is much more verbose regarding the use of this module.

**proxy_module** This module implements proxy/gateway features for Apache. It allows the server to connect to another HTTP or FTP server and relay requests on the behalf of the HTTP client, acting as a middle man. Each type of service it can connect to and all proxying features are for the most part added by additional support modules.

**proxy_connect_module** is the support module for proxy_module that provides support for the CONNECT HTTP method, mainly used to tunnel SSL requests through proxy servers.

**proxy_ftp_module** The support module that provides proxy module support for the FTP protocol.

**proxy_http_module** The support module that provides HTTP and HTTPS support for the proxy module.

**rewrite_module** The rewrite module is the bread-and-butter module for doing redirects. It is capable of usic PCRE-compatible regular expressions to parse URL strings and selectively apply rewrites or redirects. This module is also critical for InterWorx configuration purposes, as well as most WordPress installs.

**setenvif_module** is a module which allows the setting of specific environment variables based on certain conditions, such as the detected browser. This can be helpful for web applications and CGI scripts.

**spelling_module** This module attempts to counter-act spelling mistakes made in the HTTP request from the client browser. It is most commonly used to correct capitalization issues so that URL's are case insensitive.

**status_module** This module provides the /server-status status page which lists number of working requests, the number of worker processes, the status of each process, etc.

**suexec_module** With suEXEC support, this module allows apache to execute CGI scripts as a specific user and group (which is necessary in InterWorx).

**unique_id_module** This module allows for a unique ID to be associated with each request, across all requests. This might be useful for debugging, or other purposes where similar requests need to be differentiated.

**userdir_mod** This module allows the SiteWorx sites to be accessed via http://<server IP>/~<linux user>, which is useful in the event that DNS is not setup for a given SiteWorx account.

**usertrack_module** This module allows for you to generate a log of user activity on a site that uses cookies. Essentially, usertrack is a user tracking module while they are on a given site.

**vhost_alias_module** From the Apache documentation:

> This module creates dynamically configured virtual hosts, by allowing the IP address and/or the Host: header of the HTTP request to be used as part of the pathname to determine what files to serve. This allows for easy use of a huge number of virtual hosts with similar configurations.

Some of this modules directives appear in InterWorx's VirtualHost configurations.

# Chapter 6

# Default Sites for each IP address

The Default Sites feature is shown in figure 6.1 and is located in NodeWorx under Server ▷ IP Management ▷ Default Sites. By default, when you visit an IP address with the NameVirtualHost system, the webserver will check the Host: <domain> portion of the HTTP request, check it's in-memory listing of the virtualhosts on that IP address, and load the correct virtual host configuration to continue processing the HTTP request. If it is unable to match the domain sent in with its list of virtual hosts, then it will by default load the first virtualhost on its list and use that. The order is based on the order the /etc/httpd/conf.d/vhost_*.conf files were read, which means alphabetical. That means the domain with the highest alphabetical domain name is going to show up on that IP. We realized that for many this behavior is not desirable as someone's site might show up that shouldn't be visible, so we created a Default Sites system which lets you specify a URL that the user should be redirected to if they either:

- Visit a domain that is pointing at an IP address on your server but there is no domain with that name registered on your server
- Visit an IP address on your server directly in their browser

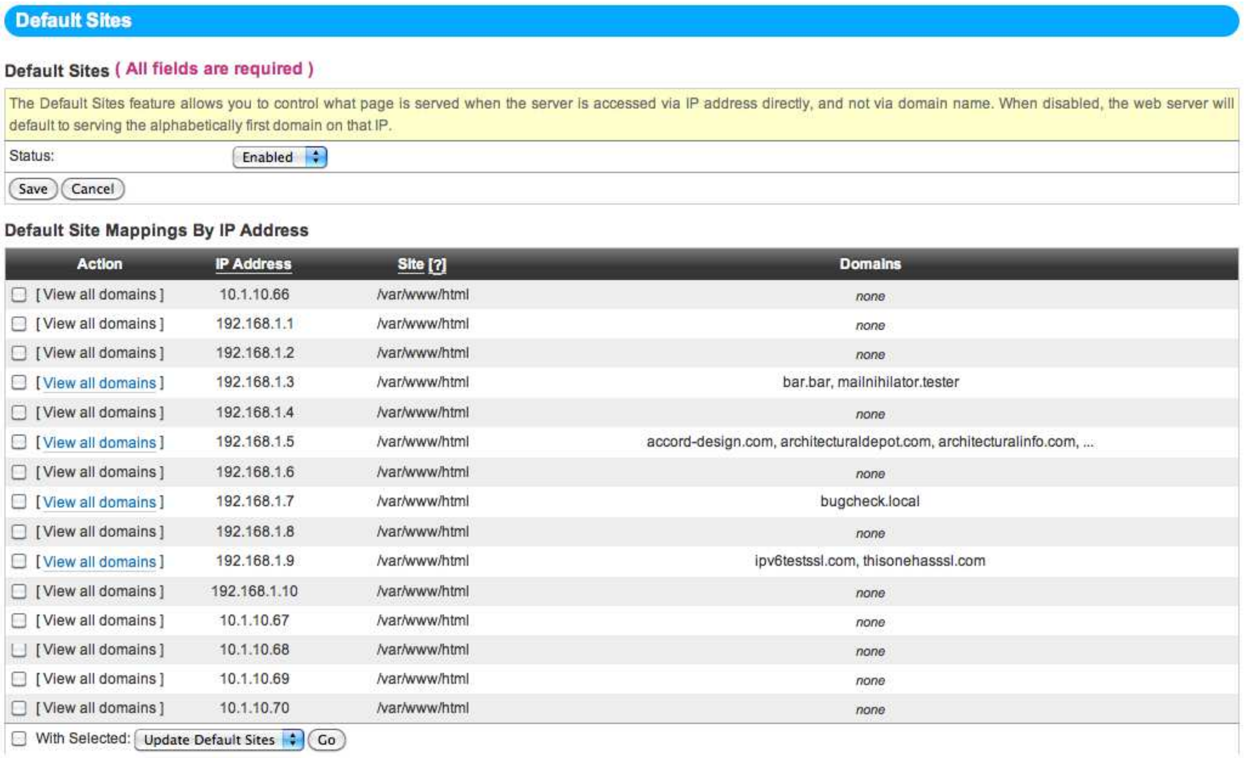
The URL should be something that can be redirected to, thus something resolve-able via DNS. In addition, you can have default site be /var/www/html where you can put some sort of splash page for your hosting service. To use the Default Sites feature:

1. Go to Server ▷ IP Management ▷ Default Sites in NodeWorx.
2. Make sure the Default Sites feature is enabled by setting the "Status" drop-down to "Enabled" if it's not already.
3. Check the boxes next to the IP addresses you wish to have redirect to a different URL if they are accessed directly.
4. Select "Update Default Sites" from the drop-down and click "Go".
5. In the window that pops up, either put the URL in "Site:" that you want the IP(s) to redirect to, or leave blank for /var/www/html.
6. Click save.

By default /var/www/html is an InterWorx splash page, but you can edit or upload files to replace the index file we have in /var/www/html. Keep in mind with suPHP, PHP will not function on the default site load because suPHP does not have a user it can switch to to execute PHP code.

**Default Sites Under the Panel** On the file system, the configuration file that sets default sites is located at /etc/httpd/-conf.d/vhost_000_defaults.conf. The 000 is our attempt to keep this VirtualHost file parsed first alphabetically so it is the one that gets loaded by default when the IP is visited without a matching domain. When the feature is turned off, the file is renamed with .disabled appened to the filename to prevent it from being loaded by the webserver (yes, only files ending in .conf are parsed and loaded by the webserver).

Figure 6.1: An Example Default Sites Configuration Screen



**Default Sites**

**Default Sites** ( All fields are required )

The Default Sites feature allows you to control what page is served when the server is accessed via IP address directly, and not via domain name. When disabled, the web server will default to serving the alphabetically first domain on that IP.

Status: Enabled

Save Cancel

**Default Site Mappings By IP Address**

| Action | IP Address | Site [?] | Domains |
|---|---|---|---|
| ☐ [ View all domains ] | 10.1.10.66 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.1 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.2 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.3 | /var/www/html | bar.bar, mailnihilator.tester |
| ☐ [ View all domains ] | 192.168.1.4 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.5 | /var/www/html | accord-design.com, architecturaldepot.com, architecturalinfo.com, ... |
| ☐ [ View all domains ] | 192.168.1.6 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.7 | /var/www/html | bugcheck.local |
| ☐ [ View all domains ] | 192.168.1.8 | /var/www/html | *none* |
| ☐ [ View all domains ] | 192.168.1.9 | /var/www/html | ipv6testssl.com, thisonehasssl.com |
| ☐ [ View all domains ] | 192.168.1.10 | /var/www/html | *none* |
| ☐ [ View all domains ] | 10.1.10.67 | /var/www/html | *none* |
| ☐ [ View all domains ] | 10.1.10.68 | /var/www/html | *none* |
| ☐ [ View all domains ] | 10.1.10.69 | /var/www/html | *none* |
| ☐ [ View all domains ] | 10.1.10.70 | /var/www/html | *none* |

☐ With Selected: Update Default Sites Go

# Part II

# Advanced Topics

Chapter 7

# How Apache or LiteSpeed integrate with InterWorx

## 7.1 Basics of Webserver Integration

InterWorx distributes the RPM packages for Apache, it does not use the CentOS-provided packages. InterWorx makes some modifications to the build process so the integration is smoother, and also allows us to keep the server's version up to date without having to rely on the operating system or a 3rd party maintainer. It is not recommended that you use packages from another source for Apache. Much like the other services that InterWorx integrates with, the daemon itself is controlled via /etc/init.d scripts (or through the `service` command which essentially uses the /etc/init.d scripts). In order to control the server's behavior, config files in /etc/httpd/ are added/modified/deleted. InterWorx uses the base /etc/httpd/conf/httpd.conf which essentially remains unchanged. In order to update and change the configuration, InterWorx will add or modify Apache *.conf files in /etc/httpd/conf.d and issue a server restart or reload.

## 7.2 SiteWorx Websites and Domains

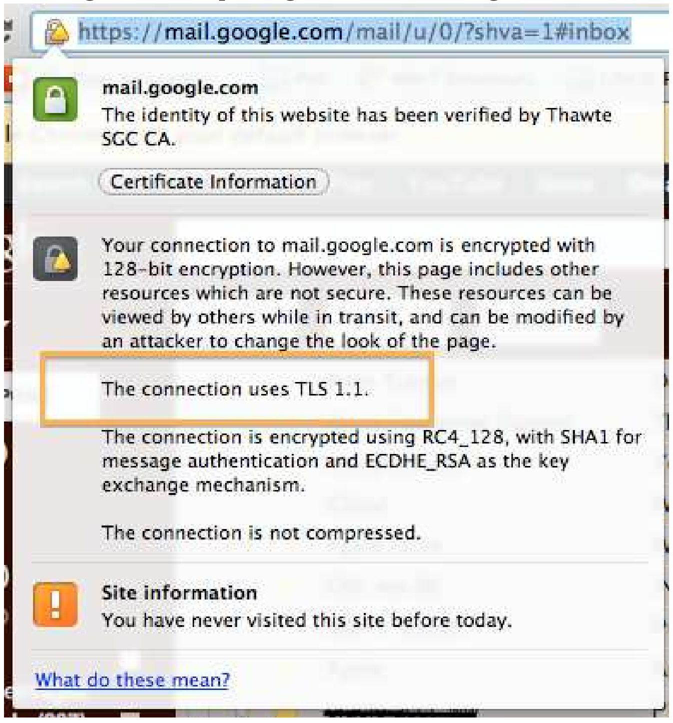
When adding a new website, InterWorx adds a new vhost config file in /etc/httpd/conf.d/ called vhost_[domain name].conf and uses a template to generate a <VirtualHost> configuration on the fly for that website based on the options you select when creating the site, and the IP address the site lives on. When a sub-domain or pointer domain is added to a SiteWorx domain, InterWorx will update the configuration file for that domain and add a ServerAlias. Secondary domains on the other hand get their own vhost config file since they require a different docroot.

This is also how InterWorx can tell you in the IP Management Screen in NodeWorx under System ▹ IP Management ▹ System IP's what sites are on which IP's. InterWorx does not base this page off of an internal database - it scans the current webserver vhost configuration to figure out what domains are mapped to what IPs.

## 7.3 HTTPS (Secure HTTP)

Transport Layer Security (and its predecessor Secure Sockets Layer SSL) allows for a secure channel of communication between the client's browser and the webserver by sending data in an encrypted stream as opposed to plain text. It also allows the client's browser to validate that the domain name being used by the client is pointing at the correct IP and hasn't been hijacked. Now a days, TLS has taken over the role of what the SSL protocol used to do. While they function similarly and use the same concepts to encrypt data, the newer TLS protocols address security concerns found in the older SSL protocols. You can investigate yourself by visiting an HTTPS secured site with Google Chrome and inspecting the encryption certificate in the browser to see what protocol is being used (this is shown in Figure 7.1). For the most part, the web hosting community and certificate authorities still refer to the certificates as SSL certificates. From now on we will refer to the secure channel as "SSL" and the certificates as "SSL Certificates" to avoid confusion and maintain congruence with the rest of the SSL certificate industry.

Figure 7.1: Inspecting SSL Cert on Google Chrome



https://mail.google.com/mail/u/0/?shva=1#inbox

mail.google.com
The identity of this website has been verified by Thawte SGC CA.

Your connection to mail.google.com is encrypted with 128-bit encryption. However, this page includes other resources which are not secure. These resources can be viewed by others while in transit, and can be modified by an attacker to change the look of the page.

The connection uses TLS 1.1.

The connection is encrypted using RC4_128, with SHA1 for message authentication and ECDHE_RSA as the key exchange mechanism.

The connection is not compressed.

Site information
You have never visited this site before today.

### 7.3.1 Limitation of SSL

Unfortunately, the secure handshake occurs below the application layer in the transport layer. What this means is that before any HTTP protocol messages are exchanged, (e.g. the client's browser issuing their `GET /index.html HTTP/1.1` request), the handshake and communication channel has to be established with the server. Since the VirtualHost system relies on the HTTP request to figure out what website on a given IP is being requested, you are limited to one SSL certificate per IP address. Thus, in order for SSL to function correctly on a website, it must be the only website on a given IP address since its SSL certificate has to be the one used to make the secure connection. In order to enforce this limitation, InterWorx will only allow a single SiteWorx account on an IP address that is serving a website with SSL enabled. There are new extensions which will eventually allow for the client to indicate what the domain is that they are trying to visit to the server during the SSL handshake phase, but these extensions are not yet wide spread. When they become widely adopted in both client and server software, InterWorx will adapt the system to allow multiple domains with SSL on the same IP.

### 7.3.2 How SSL is added to the Webserver Configuration

When SSL is added to a SiteWorx domain, an additional <VirtualHost> section is added to the domain's vhost configuration file. This allows the webserver to serve the correct data when someone connects to SSL on TCP port 443. If there are multiple SiteWorx domains in the SiteWorx Account, only one is permitted to have SSL enabled and installed on it. The other domains will be availble HTTP requests on TCP port 80 only.

## 7.4 Webmail and /nodeworx /siteworx redirects

Every SiteWorx domain has the cability of accessing the InterWorx-provided webmail clients at http://<domain>/webmail. Furthermore, specific webmail clients can be reached at http://<domain>/horde, /roundcube, and /squirrelmail. In order to provide an easy URL for clients to remember to access their control panel, http://<domain>/siteworx acts as a redirect to the SiteWorx control panel login for their domain. This is all done with redirects and proxy rules defined in /etc/httpd/conf.d/iworx.conf. The base server config is set to:

- Redirect (i.e. Rewrite the URL) any URL that contains and *only* contains and /siteworx or /siteworx/ after the domain and port portion of the url to https://<domain>:2443/siteworx/?domain=<domain>. This is because the control panel web front-end is not served from the standard port 80/443 web server instance. It instead it is served from a different instance of Apache that is running on port 2080/2443 for HTTP/HTTPS, respectively. This rule will also trigger if the SiteWorx domain has SSL on it and the
- Similarly, /nodeworx and /nodeworx/ will redirect to http://<domain>:2443/nodeworx which is the NodeWorx login page. NodeWorx is not domain-specific, so no ?domain=<domain> needs to be added.

Webmail functions a bit differently from the control panel redirects, though. Instead of redirecting we instead *proxy* from the "main" Apache instance on port 80/443 to the InterWorx Control Panel instance on port 2080. This is because the webmail clients are actually served from the InterWorx Control Panel Apache instance as they are integrated with the Panel's software distribution. The benefit is:

- SiteWorx users can access their webmail clients transparently on http://<domain>/webmail without really having to think about ports/etc.
- SiteWorx accounts with SSL certificates can make use of their certificates to encrypt and authenticate the connection to the front-end Apache server on port 443, which will then proxy the connection to port 2080. Thus the connection remains encrypted over the wire and the user is unaware that they are in fact connecting to a different Apache instance.
- As the InterWorx control panel instance is running on the local server and the proxy connection occurs all internally on the loopback address (127.0.0.1), there is no latency from the proxy connection and the data cannot be snooped from the internet without someone having unauthorized access to the server's root user.

## 7.5 Bandwidth Monitoring

In order to keep users within their bandwidth constraints and also to provide users real-time-ish data on the amount of traffic they are recieving, InterWorx interfaces with both Apache and LiteSpeed to gather bandwidth data per SiteWorx domain (i.e. VirtualHost).

### 7.5.1 Apache

In order to collect bandwidth data from the Apache web server, InterWorx distributes the Apache webserver with a module called mod_watch which silently records bandwidth used by each virtualhost and IP address on the system. In order to make this data externally available to the control panel, a file is created for each Virtualhost in /var/lib/-mod_watch/ with whitespace-delimited data. Each virtualhost and IP address on the system has a file that is updated in real time by the webserver in order to give the control panel access to the most up to date usage stats when bandwidth limits are being checked. The data is collected every 5 minutes by the iworx.pex fively cron job, which also updates the SiteWorx account's real-time bandwidth tracking RRD graphs and the NodeWorx webserver graphs.

### 7.5.2 LiteSpeed

The LiteSpeed team made a plugin for their webserver that allows InterWorx to gather the number of bytes in and out of the server for each specific virtual host. The plugin puts a file on the filesystem at /var/log/httpd/interworx_traffic_log which InterWorx then scrapes for data on bandwidth usage. Unlike mod_watch for Apache, though, InterWorx can't query the number of total requests, the number of documents from the docroot that have been served, the number of current active connections, or the bytes/second rate that data is being sent out because this data is not made available to InterWorx.

## 7.6 Statistics

One of the most important features to users is the ability to see statistics about how their website is performing on the internet, what their average traffic patterns are like, and what content is most often visited on their site. With tools like

Google Analytics rapidly becoming more popular, most serious users trying to improve their search engine rankings will default to Google for that sort of data. Yet with many users also running script and ad blockers in their browsers, it is often desirable to also check the statistics reported by the webserver as they tend to give you a more complete picture of what data is being requested from your server.

### 7.6.1 The 3 Statistics Programs

Integrated into the SiteWorx control panel InterWorx offers statistics pages generated by the 3 different stats programs bundled with the Panel: Analog, Webalizer, and AWStats. All 3 parse the logs generated by your webserver and deposited in the SiteWorx domain's /home/<linux user>/var/<domain>/logs/ directory. Since Litespeed is compatible with Apache's configuration format, it is able to properly generate Apache-compatible logs in that directory. InterWorx has the data generated by the stats software placed in /home/<linux user>/var/<domain>/stats/. Stats are generated once a day, with weekly and monthly stats agragating the daily and weekly data such that users are able to see slices of their sites performance in the past on a weekly or monthly basis. This means that stats are generated by the iworx.pex daily cron job and no updates will be visible to a user until that cron job has run.

### 7.6.2 Dealing with High-Traffic Sites

For sites with extremely heavy traffic and extremely large logs (often gigabytes in size), InterWorx will try to split the file up to reduce the load on the server before running the stats programs. In addition, in order to lose as little data as possible, InterWorx will take a snapshot of the log and move it elsewhere (unless it's too large, then slicing occurs) before running the stats such that all SiteWorx accounts have stats generated from about the exact same time in the log file. If this wasn't done, a very busy site with a large log would delay other SiteWorx accounts from having their stats processed. As a result, the data shown on those sites that were delayed would not be for a period of 24 hours, but instead for a period of longer than 24 hours. Provided that processing time isn't guaranteed to be constant, this could lead to unreliable data.

Chapter 8

# PHP integration

InterWorx comes with 2 methods of PHP integration with the Apache webserver. Litespeed has it's own system for integrating PHP.

## 8.1 mod_php with Apache

The default out-of-the-box PHP integration method for Apache is using the mod_php module. This integrates PHP's interpreter directly into the apache daemon processes such that when a PHP script is requested, Apache is able to execute the PHP code on the fly right there without having to summon PHP in an external process.[1]

### 8.1.1 Advantages

- Better speed - Apache is able to execute the PHP code directly inside it's own process, it does not need to spawn another program which has additional CPU and memory overhead.
- PHP settings can be modified on a per-site basis within .htaccess files using `php_value` directives.

### 8.1.2 Disadvantages

- Security - All user scripts execute as the apache linux user. PHP web applications are often prone to getting attacked by hackers as they are often the most widely-used applications on the internet. Often times, users will not keep their versions of WordPress or Joomla, or even plugins used inside those applications, up to date. This often will allow attackers, who are made aware of a vector of attack from a version changelog, that old versions of the software can be hacked. If a hacker is able to exploit a script in order to upload their own PHP code to the server, they would have:
  - Ability to see all files on the filesytem that the apache user can see, including all files across all sites hosted on your server
  - Ability to open and read files within directories across the entire filesystem that the apache user can see (e.g. they would be able to see MySQL DB passwords for all applications across all sites since the apache user must be able to open and read files)
  - Ability to execute PHP code as the apache user
- It is much more difficult to trace the domain responsible for the intrusion as all logs will report the apache user as responsible instead of the linux user attached to the SiteWorx account.
- Changes to the system-wide php.ini will not take effect until Apache is restarted.

[1]Fun fact: if you `lsof -p [PID]` an apache process with mod_php, you will see all the PHP extentions opened by the apache process, indicative that the interpreter is literally running inside apache.

### 8.1.3 Conclusion

mod_php is fine to use if you have a small number of SiteWorx accounts or you are running 1 or 2 large sites off the InterWorx server and you need performance. For larger high- account servers, it is strongly recommended that the next integration type, suPHP is used instead.

## 8.2 suPHP with Apache

suPHP attacks the security issue by externalizing the execution of PHP scripts from the webserver process and running the script externally in it's own process as the linux user of the SiteWorx account. It does this by having Apache call /usr/sbin/suphp when a PHP script is requested, which is able to switch user id's[2] to the linux user of the SiteWorx account and then execute the PHP script with the PHP interpreter, /usr/bin/php-cgi, which recieves browser data via the Common Gateway Interface (CGI)[3]. The output of the interpreter is returned to Apache to be sent as the response to the client.

### 8.2.1 Advantages

- Per linux user script execution - PHP scripts are executed as the linux user of the SiteWorx account. As such, a hacker gaining access via an exploitable web application will be limited to reading, writing, and executing what the SiteWorx account's linux user can read, write and execute. With permissions set correctly, this would prevent a hacker from seeing other site's content or files.
- System logs (for example, the mail logs), will report the UID of the SiteWorx Account linux user instead of the apache linux user, making it easier to track down the source of spam, or some sort of other intrusion. In addition, the process list will show php-cgi processes with the linux user of the SiteWorx account, making it easier to pinpoint the source of load issues.
- Per-SiteWorx account php.ini's in /home/<linux user>/etc/. If no per-siteworx account ini is found, the default system /etc/php.ini is used instead.
- Users/Groups below UID/GID 500 are not permitted to execute scripts. I.e. suPHP will not allow scripts owned by root to be executed.
- If the the linux user that is executing a PHP script is writing to a file or making a new file in a directory owned by another user/group (even if permissions are set that this would be allowed), suPHP will disallow the script from writing to that file or that directory. This can be further limited by configuring suPHP's settings.
- Changes made to either the system /etc/php.ini or SiteWorx account's php.ini will see their changes reflected in the next PHP script execution. No server restart required.

### 8.2.2 Disadvantages

- Performance Penalty - Due to the need to spawn 2 processes, first suphp and then php-cgi, there is extra overhead added to every PHP script execution. On most servers, this penalty will not incur enough of a load or delay. High-traffic servers may suffer too much and might need to switch to mod_php. The downside in a shared hosting environment is you lose some security.
- The initial switch to suPHP might need permissions tweaked on existing php files or their enclosing directories in order to get them running correctly. Additionally some permissions might need to be tweaked in order to secure your siteworx accounts and full segregate them.
- Users expecting to configure PHP settings via .htaccess files will be unable to do so.

[2]The SUID bit is set and the file is owned by the root user and group. Permissions: 4755 or rwsr-xr-x
[3]http://en.wikipedia.org/wiki/Common_Gateway_Interface

### 8.2.3 Conclusion

For most hosts, this is recommended. In shared hosting environments, it's impossible to keep an eye on all the applications running on your server. Despite there being a performance penalty, most servers do not see enough traffic on a regular basis to justify using the less secure mod_php.

## 8.3 Litespeed PHP Integration

If you are using Litespeed in place of Apache, the Litespeed documentation[4] does a good job of explaining how to setup integration. Basically, litespeed uses a specially patched version of PHP that is able to run in something of a "daemon mode" in which the PHP process is constantly running, and responds to requests from the Litespeed server through a special API where the server sends PHP needed for processing, and the lsphp daemon returning with computation and the output of the PHP script. According to Litespeed, the performance improvement is significant.

[4] http://www.litespeedtech.com/support/wiki/doku.php