Hands On Lab

Compiling and Installing Nagios and Plugins

Table of Contents

Introduction 2
Goals 2
Packages, Resources and Prerequisites 2
Document Conventions 3
General Process 4
Apache Server Setup 6
Nagios Monitoring Scenarios 7
Obtaining Nagios Core and Nagios Plugins 7
Preliminary Work: User and Group Setup 8
Nagios Core 9
  Compilation 9
  Core Package Installs 10
  Core Package Post installation 11
Plugins 12
  Plugin Compilation 12
Service Setup 13
Web Services Verification 14
Appendix A – Configuration File Example 15

Introduction

Nagios is an IT infrastructure monitoring suite (consisting of both a client and server application) that allows the monitoring and alerting of various infrastructure and service related items in the event of complete failure or performance degradation. We are going to explore the community edition that is offered at http://www.nagios.org.

Goals

This lab will introduce you to the concepts Nagios, the community edition. We will download, compile and install the Nagios core and Nagios plugins directly from the site and configure our server to monitor the most common infrastructure components and view our setup in a web browser remotely. NOTE that this first lab will cover the download, compile and installation of the server only. A subsequent lab will cover the NRPE plugin for remote monitoring of other hosts.

Packages, Resources and Prerequisites

The packages involved in our lab are:

- nagios-core
- nagios-plugins
- wget
- build-essential
- apache2
- php5-gd
- libgd2-xpm-dev
- libapache2-mod-php5

The resources you will be accessing during the course of this lab are:

- Ubuntu 14.04 Server: Nagios Server and Apache Host
  - You will use this to test your server and view the local Nagios website
- Ubuntu 14.04 Server: Remote Monitoring Host
  - This is the server that will be used extensively in the NRPE lab but will not be addressed in this document

Prerequisites to this lab:

- A LinuxAcademy.com Lab+ Subscription

- Internet Access and SSH Client
    - You will need to connect to the public IP of the server in order to configure Nagios, the only method of connectivity is over SSH
        - SSH client can be Windows (i.e. Putty) or from another Linux system shell
- Login Information (Provided When The Server Starts Up)

Document Conventions

Just a couple of housekeeping items to go over so you can get the most out of this Hands On Lab without worrying about how to interpret the contents of the document.

When we are demonstrating a command that you are going to type in your shell while connected to a server, you will see a box with a dark blue background and some text, like this:

```
linuxacademy@ip-10-0-0-0:~$ sudo apt-get install package
[sudo] password for linuxacademy: <PASSWORD PROVIDED>
```

That breaks down as follows:

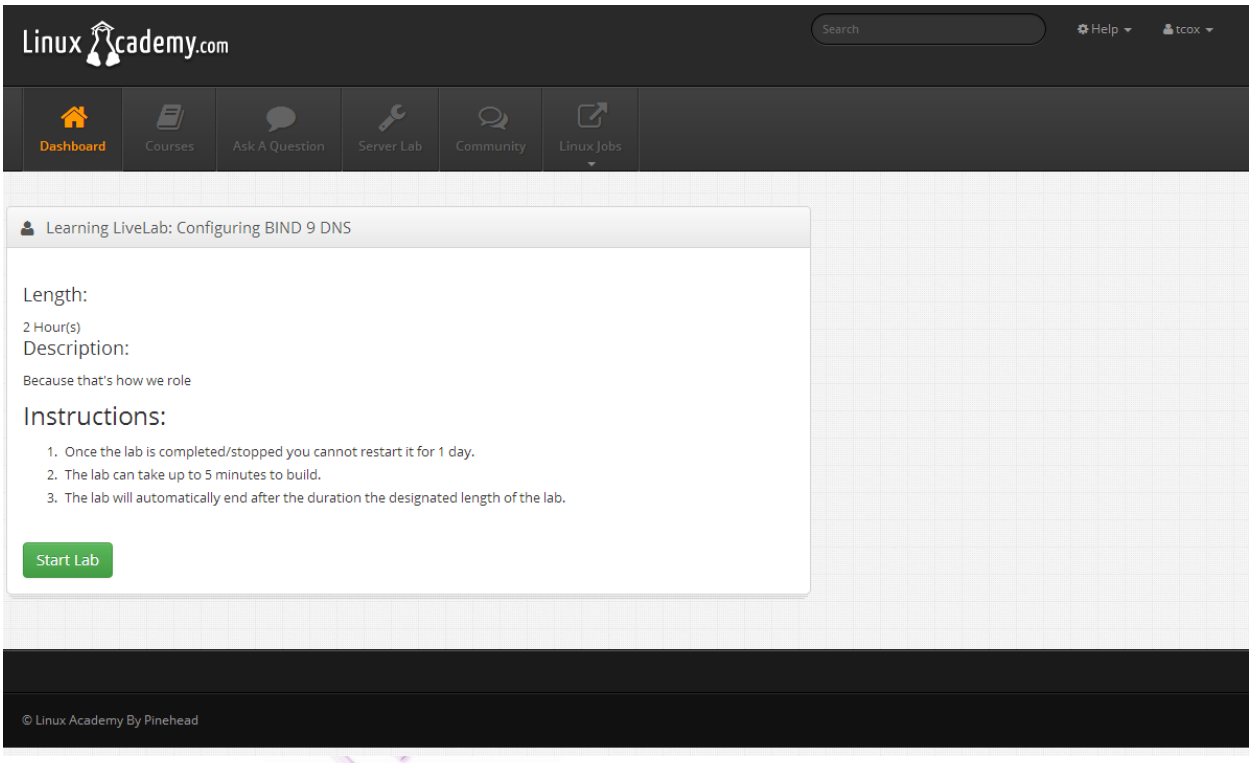
- The white text that looks something like "linuxacademy@ip-10-0-0-0:~$: "should be interpreted as the console prompt your cursor will be sitting at when you are logged into the server. You are not typing that into the console, it is just there. Note that the portion that says "ip-10-0-0-0" will be different for you based on the IP address of your system.
- The bold yellow text in any command example is the actual command you will type in your shell.
- Any lines subsequent to the bold yellow command that you type in is to be considered as the response you can expect from your command entry.

If you do not see the command prompt as the first line in the example, then all the white text is an example of a text, script or configuration file and is intended to be typed in its entirety (in the event you are instructed to create a previously non-existent file) or to compare against the contents of the file on your system.

One final convention, if you see a "~" at the end of a line in a command or text example, that will indicate that the line overflowed the end of line. Meaning you should just keep typing without hitting enter to start a new line until the natural end of the command or entry.

General Process

When you are ready to begin the Hands On Lab, log into your Linux Academy Lab+ subscription and navigate to the "Live Labs" section on the Dashboard. Once you choose the "Nagios Installation and Configuration" Lab from the list, you will see a similar screen to the one below:



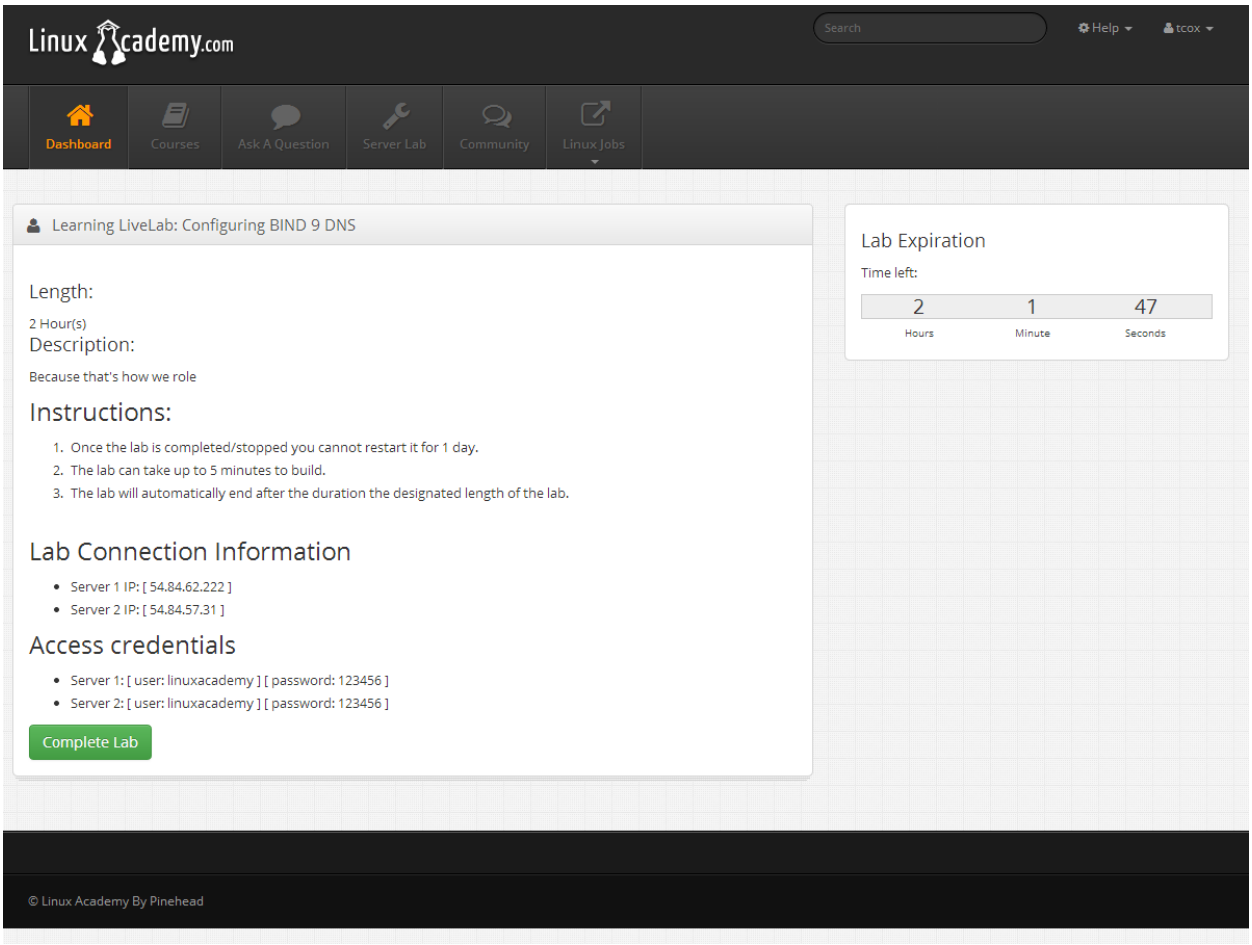
[Linux Academy Live Labs+ - Start Screen]

A few things to note before you start this process:

- When you launch the lab from this screen, it may take up to FIVE MINUTES for your servers to be deployed and be available for your use.
- Do not leave your desktop and come back, once the servers are launched, you will only have TWO HOURS to complete this lab from start to finish. After that point, the servers time out and are deleted permanently. Any and all work that you have done will then be lost.
- You can only use this lab ONCE PER DAY. If you try to use it more than that after completing the lab or the servers timing out, the screen will tell you when it will be available to you again.

- Other than those descriptions, you may retry any of the Labs+ labs as many times as you wish as long as you are a subscriber.

Once you have clicked on the 'Start Lab' button that you see above, a process will launch on our servers that will deploy the two servers we will use in our lab for testing. After a few minutes of processing (and you will see a status message that says "Creating Lab… Please Wait"), you should see a screen that looks similar to this one:



[Linux Academy Live Labs+ - Start Screen]

You will see all the information you need to access your servers from another system. Specifically, you need:

- The two server public IP addresses
- Access credentials for both

One thing to note is that, in addition to the two IPs that you see above, each server will have another IP assigned to it in the 10.0.0.x subnet. This is a private IP address and will not route outside of your private server pool.

In our setup, Server 1 will function as an Apache server that we will use to test our Nagios configuration. Server 2 will function as our remote NRPE host, which will be covered in a subsequent lab called "Nagios NRPE Monitoring".

Apache Server Setup

We are going to configure a VERY basic Apache web server, create a basic HTML file that we can use to verify that it is running and install a text based browser that we can use for our testing later.

Start by installing the Apache 2.x web server (and subsequently starting it) as well as the other dependencies we will need by performing the following commands:

```
linuxacademy@ip-10-0-0-162:~$ sudo apt-get install apache2 wget
build-essential php5-gd libgd2-xpm-dev libapache2-mod-php5
[sudo] password for linuxacademy: <PASSWORD PROVIDED>
```

This will install a number of packages and modules in addition to the Apache server. It will only take a moment or two to complete and once it is done, Ubuntu will automatically launch the Apache server on the default port 80. You can verify that it is listening by executing a local telnet connection on port 80 and you should see the following:

```
linuxacademy@ip-10-0-0-162:~$ telnet localhost 80
Trying 127.0.0.1…
Connected to localhost.
Escape character is `^]`.
```

Pressing the escape character of ']' will return an HTML message to you and close the connection, but that verifies that our server is answering on port 80. We are now going to create the most basic of HTML pages so that we can test our web server. Change to the /var/www/ directory and create the following file called "test.txt" in the /var/www directory (you can copy and paste):

```
<HTML>

<BODY>
   www.linuxacademy.lab
</BODY>

</HTML>
```

We will can browse to this server using its external IP once this set up is complete in order to test that it is available. This will also allow us to set up the browser component for Nagios once we have complete our lab.

Our local Apache server is completed for now. We will be coming back to reconfigure Apache to allow Nagios core to run properly and with authentication in subsequent sections of this Lab.

Nagios Monitoring Scenarios

Although this lab focuses on a single server Nagios monitoring setup, there are several common ways that Nagios is deployed in an organization:

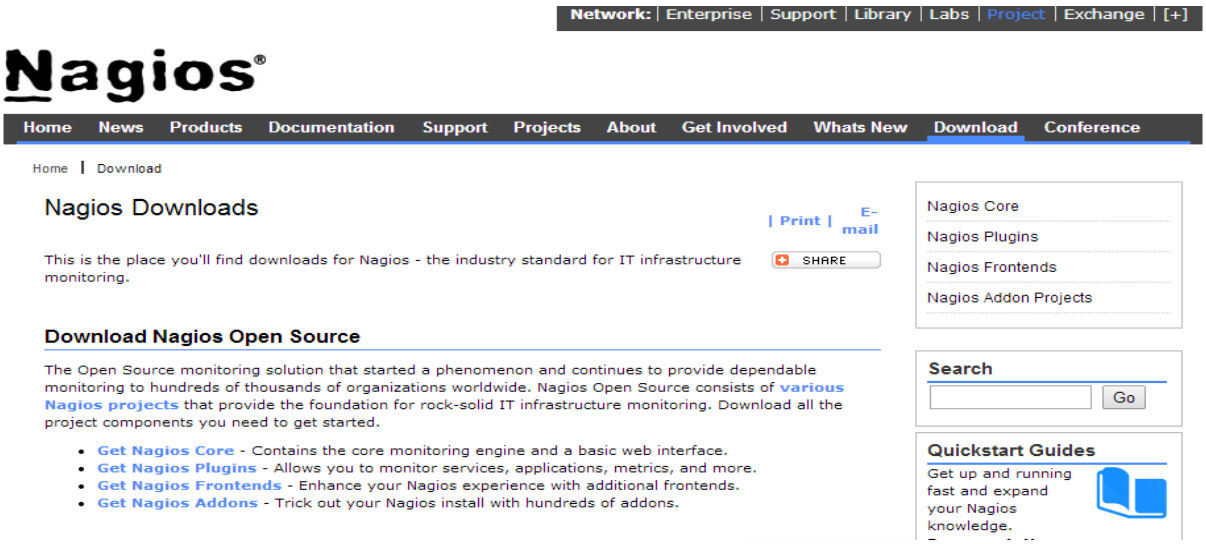
- Single Server and Monitor
  - This is the server we will be setting up today. We are using a single node as both the Nagios server and the node we are monitoring and will configure Apache to authenticate and report on the status of common performance and service statistics.
- Single Monitoring Server and Multiple Nodes
  - Most commonly, Nagios is deployed in order to gather performance and availability statistics for multiple nodes in an organization. These nodes send their statistics back to a single Nagios NRPE server and they can be viewed there (covered in our second lab in this series).

Obtaining Nagios Core and Nagios Plugins

Nagios is offered in two flavors as many Linux server applications are: Nagios XI and Nagios Community Edition.

Nagios XI is offered, with paid support, on the website http://www.nagios.org and does come with a fully compile native binary installation for a full deployment, including web server integration, customized graphs and reporting and database storage.

Nagios Community Edition is the more traditional implementation of Nagios (and until the last two years, was the only one that was available) and the one we are going to use here. It consists of two parts – Nagios Core and Nagios Plugins. We are going to be using the very latest copies of each and they are available on the same website, specifically http://www.nagios.org/download. Visiting that, you will see:



For the purposes of this lab, we are going to download two components:

- Nagios Core – this is the core monitoring engine and the basic web/authentication interface
- Nagios Plugins – these are the most common service, application and metric monitors that are used with Nagios

You can download the latest version 4 core and plugins to your local system and transfer them to your remote lab server using SCP, for example:

```
user@localhost:~$ scp nagios-4.0.7.tar.gz
linuxacademy@64.55.33.22:/home/linuxacademy
```

If you are using Windows locally or just prefer using a GUI, you can also use Filezilla to transfer your files, just note the IP address to connect to is the same as the one you used to login, the account is the same one you are using on your terminal and the port to connect to through Filezilla is the SSH port 22 (secure file transfer).

Once we have these files on our remote Lab Server, we can begin with our installation process for Nagios Core.

Preliminary Work: User and Group Setup

By default, Nagios makes a couple of assumptions in the user and group that it will use for its services. Let's take a moment and get them set up:

```
linuxacademy@ip-10-0-0-162:~$ sudo useradd nagios
linuxacademy@ip-10-0-0-162:~$ sudo groupadd nagcmd
linuxacademy@ip-10-0-0-162:~$ sudo usermod -a -G nagcmd nagios
```

All we did here was add the user ‘nagios’, add a new group called ‘nagcmd’ and finally added that new user to the new group. These are needed for the Nagios Core to work properly and to allow proper authentication and ownership in our web site integration later.

Nagios Core

Now that we have our package and have set up the users we need for Nagios Core, let’s expand and start compiling Nagios on our system. Let’s start by creating a place to compile and then expanding our Nagios Core package:

```
linuxacademy@ip-10-0-0-162:~$ mkdir tmp && cd tmp
linuxacademy@ip-10-0-0-162:~$ tar zxvf ../nagios-4.0.7.tar.gz &&
cd nagios-4.0.7
```

Let’s prepare our installation by passing in the appropriate compile time configuration parameters from our earlier work:

```
linuxacademy@ip-10-0-0-162:~$ ./configure -with-nagios-group=nagios
--with-command-group=nagcmd -with-mail=/usr/bin/sendmail
```

You will notice that we also added an extra parameter for sending mail. In this lab we will not be setting up sendmail for notifications, but we do need to let the application which mail service it will be using to send notification alerts when those are configured.

This process will take some time and will scroll through a large amount of text that will indicate how Nagios Core is being configured to compile. As long as you do not receive any errors or messages about missing dependencies, we can move on to compilation. If you do receive any error or missing dependency claim, please be sure you revisit our earlier package installation in the Apache section and make sure all of the packages listed were successfully installed.

Compilation

Provided that our configuration steps were successful earlier, we can now compile our Nagios Core package and prepare it for installation. Although we could issues each command subsequent to each other in one command, let’s go through each one and talk a little about what they do. We will start with the initial compilation command, like so:

```
linuxacademy@ip-10-0-0-162:~$ make all
... a whole bunch of scrolling text...
...Enjoy ← Should be the last word you see provided no errors
```

This command "makes" or compiles all of the Nagios Core that we have downloaded, all the binaries, libraries and dependencies and prepares all the subsequent packages for installation. Once this is done, we can begin our installation process.

Core Package Installs

We have completed compiling all our packages and dependencies, so now we need to begin installing them. We have themes that are available to us, if you want to install them, you may do so by looking at the theme names from our compile output, but here, we are going to install just the items that are important to get our Nagios Core up and running before we start on our plugins. First up, installing the Nagios Core system:

```
linuxacademy@ip-10-0-0-162:~$ sudo make install
```

Here we are doing a 'sudo' in our installation command to be sure we have permissions to install the binaries and libraries that are needed. If you get errors during this part, you forgot to use 'sudo' in your installation command.

Next, we are going to install the configuration to allow Nagios to run as a service (although not required, we are going to use this server as a true server in subsequent configurations, so we will go ahead and do that now):

```
linuxacademy@ip-10-0-0-162:~$ sudo make install-init
/usr/bin/install -c -m 755 -d -o root -g root /etc/init.d
/usr/bin/install -c -m 755 -o root -g root daemon-init
/etc/init.d/nagios

*** Init Script installed ***
```

This mainly saves us from having to create this service manually. If you miss this step or do not perform this 'install' before you delete the directory after set up, you can perform the set up manually using the commands output above or however you normally set up your system daemons for startup.

Up next is the installation of our sample configuration files so we are not starting completely from scratch once we start our Nagios service:

```
linuxacademy@ip-10-0-0-162:~$ sudo make install-config
```

Let's also be sure we have the Nagios core tools available to us on the command line and the binaries are installed where they need to be:

```
linuxacademy@ip-10-0-0-162:~$ sudo make install-commandmode
```

Finally, we are going to install the web configuration. However, since we are using Ubuntu, we need to add a little something here. By default, Nagios Core uses the "Red Hat" Apache configuration standard for copying its configuration or Vhost file to. Let's make sure there is a directory to copy the file to:

```
linuxacademy@ip-10-0-0-162:~$ sudo mkdir -p /etc/httpd/conf.d
```

Now we can issue our installation command:

```
linuxacademy@ip-10-0-0-162:~$ sudo make install-webconf
```

Then we have to move our configuration file to an appropriate location for Ubuntu:

```
linuxacademy@ip-10-0-0-162:~$ sudo mv /etc/httpd/conf.d/*.conf
/etc/apache2/sites-available
```

Lastly, we need to activate the new Vhost and then restart Apache to pick up the new configuration:

```
linuxacademy@ip-10-0-0-162:~$ cd /etc/apache2/sites-available
linuxacademy@ip-10-0-0-162:~$ sudo a2ensite nagios.conf
Enabling site Nagios…

linuxacademy@ip-10-0-0-162:~$ sudo service apache2 reload
* Reloading web server apache2                [OK]
*
```

Our Nagios Core is now installed, we have just a couple of post installation cleanup items and then we can start on our plugins.

Core Package Post installation

Now that our configuration, compilation and installation is complete on the Nagios Core, we have a few post installation items to complete.

First, be sure you are back in your Nagios installation directory (/home/linuxacademy/tmp/nagios-4.0.7 if you are following this guide exactly) and let's copy our eventhandlers directory to the proper place for our system to use it:

```
linuxacademy@ip-10-0-0-162:~$ sudo cp -R contrib/eventhandlers
/usr/local/nagios/libexec && sudo chown -R nagios:nagios
/usr/local/nagios/libexec/eventhandlers
```

Now we need to start up Nagios and test our configuration. We do that as follows:

```
linuxacademy@ip-10-0-0-162:~$ sudo /usr/local/nagios/bin/nagios -v
/usr/local/nagios/etc/nagios.cfg
```

```
...<--- checking text here
Total Warnings: 0
Total Errors:   0
Things look okay - No serious problems were detected during the
pre-flight check
```

If you get any errors at this point, please check your configuration file against the edits we made in the previous section. Once you pass the verification test, we can then start up our Nagios server by executing the following command:

```
linuxacademy@ip-10-0-0-162:~$ sudo /etc/init.d/nagios start
```

We are now doing very basic monitoring of our current server using Nagios! We have to compile and install our plugins so that we get more granular monitoring in place, but that's a lot easier.

Plugins

Our core Nagios server is now compiled, installed, tested and running. At this point we have very basic monitoring in the core engine, but we want more granular monitoring. So we need to compiled and install our plugins (from our earlier download).

Plugin Compilation

So our plugin compilation is a bit simpler than the core compilation and install as the initial configuration is already in place, we are just adding more granular monitoring ability to our server.

At this point, let's go ahead and stop Apache:

```
linuxacademy@ip-10-0-0-162:~$ sudo service apache2 stop
```

And add a user that we can use for our web server client (login):

```
linuxacademy@ip-10-0-0-162:~$ sudo htpasswd -c
/usr/local/nagios/etc/htpasswd.users nagiosadmin
New password:
Re-type new password:
Adding password for user nagiosadmin
```

Next, expand our plugins file:

```
linuxacademy@ip-10-0-0-162:~$ tar zxvf ../nagios-plugins-2.0.tar.gz
```

Move to our plugins directory:

```
linuxacademy@ip-10-0-0-162:~$ cd nagios-plugins-2.0
```

Configure our compilation:

```
linuxacademy@ip-10-0-0-162:~$ ./configure -with-nagios-user=nagios
-with-nagios-group=nagios
```

If you are missing any prerequisites, please revisit our initial package installation section above in order to satisfy them. Once that is complete, a simple compile:

```
linuxacademy@ip-10-0-0-162:~$ make
```

And finally, we will complete this with an installation. Make sure you use elevated privileges in this last part or this install will fail with permissions errors:

```
linuxacademy@ip-10-0-0-162:~$ sudo make install
```

Service Setup

Now that we have everything setup and installed, we want Nagios to start up after a system restart. So we need to set up the appropriate scripts in the right places in order to do that.

Let's create the proper link from our start up directory (/etc/init.d) where the Nagios Daemon lives to the proper 'rc' directory and startup file:

```
linuxacademy@ip-10-0-0-162:~$ sudo ln -s /etc/init.d/nagios
/etc/rcS.d/S99nagios
```

This adds the startup script S99nagios to the runlevels for reboot. At this point, you could login to the web server after starting it up, but none of the CGI files will run (in fact they will download). We just need to add the modules that support that to our Apache configuration:

```
linuxacademy@ip-10-0-0-162:~$ cd /etc/apache2/mods-available
linuxacademy@ip-10-0-0-162:~$ sudo a2enmod cgi
Enabling module cgi.
To activate the new configuration, you will need to run:
   Service apache2 restart
linuxacademy@ip-10-0-0-162:~$ sudo a2enmod cgid
Enabling module cgid.
To activate the new configuration, you will need to run:
   Service apache2 restart
```

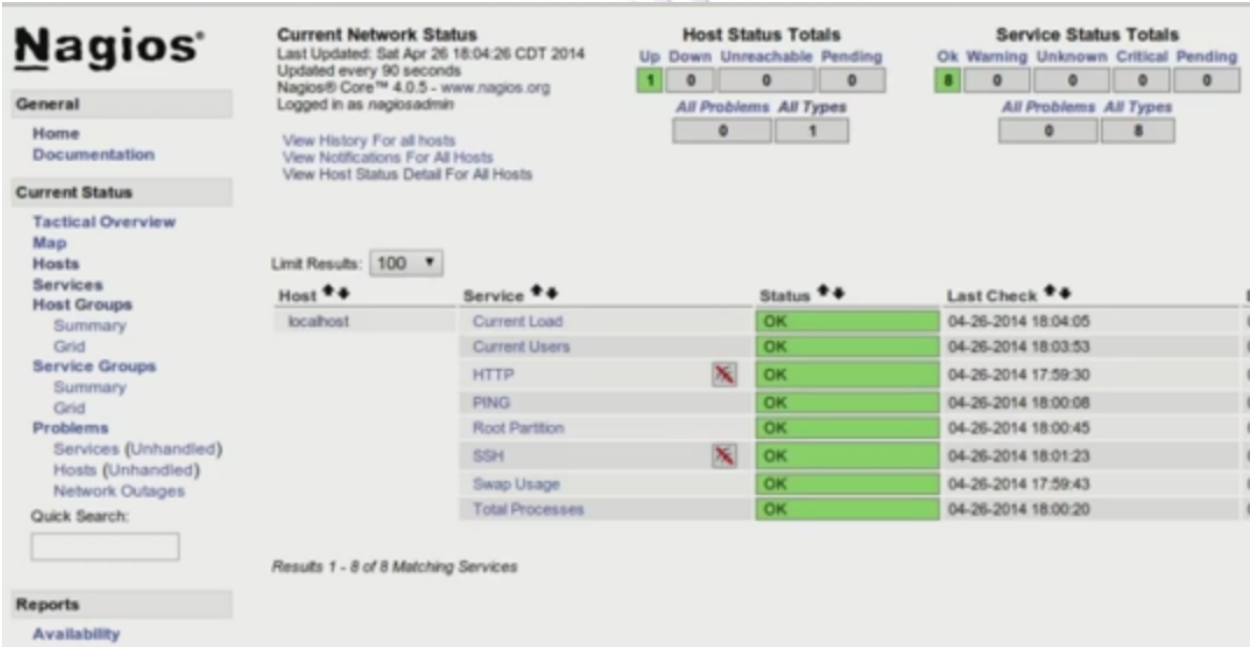
Now that we have done that, we can start Apache:

```
linuxacademy@ip-10-0-0-162:~$ sudo service apache2 start
```

We now have our plugins running, the nagios core system set to install and all supporting web modules loaded in order to see our web interface.

Web Services Verification

We can now verify our instance is running by logging into our web site. In a browser, navigate to http://EXTERNAL.IP.HERE/nagios and log in with the user name and password we created earlier (Username: nagiosadmin and Password: YourPasswordHere). Once logged in, you should see a page for your server, on the left, click on the "Services" link and you will see something like this:



Congratulations! You have completely set up your first Nagios server. Right now, you are only monitoring yourself (the current server), but next we will learn how to set up NRPE and monitor multiple servers in our enterprise and see them all in one browser interface!

Appendix A – Configuration File Example

```
##############################################################################
##############################################################################

# Define an optional hostgroup for Linux machines

define hostgroup{
        hostgroup_name  linux-servers ; The name of the hostgroup
        alias           Linux Servers ; Long name of the group
        members         localhost     ; Comma separated list of hosts that belong to this
group
        }



##############################################################################
##############################################################################
#
# SERVICE DEFINITIONS
#
##############################################################################
##############################################################################


# Define a service to "ping" the local machine

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
```

```
        service_description             PING
         check_command                          check_ping!100.0,20%!500.0,60%
        }


# Define a service to check the disk space of the root partition
# on the local machine.  Warning if < 20% free, critical if
# < 10% free space on partition.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             Root Partition
         check_command                          check_local_disk!20%!10%!/
        }



# Define a service to check the number of currently logged in
# users on the local machine.  Warning if > 20 users, critical
# if > 50 users.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             Current Users
         check_command                          check_local_users!20!50
        }


# Define a service to check the number of currently running procs
# on the local machine.  Warning if > 250 processes, critical if
# > 400 users.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             Total Processes
         check_command                          check_local_procs!250!400!RSZDT
        }
```

```
# Define a service to check the load on the local machine.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             Current Load
         check_command                        check_local_load!5.0,4.0,3.0!10.0,6.0,4.0
        }



# Define a service to check the swap usage the local machine.
# Critical if less than 10% of swap is free, warning if less than 20% is free

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             Swap Usage
         check_command                        check_local_swap!20!10
        }



# Define a service to check SSH on the local machine.
# Disable notifications for this service by default, as not all users may have SSH enabled.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
        service_description             SSH
         check_command                        check_ssh
         notifications_enabled                0
        }



# Define a service to check HTTP on the local machine.
# Disable notifications for this service by default, as not all users may have HTTP enabled.

define service{
        use                             local-service         ; Name of service template to
use
        host_name                       localhost
```

```
service_description             HTTP
 check_command                        check_http
 notifications_enabled                0
}
```