Linux Web Server Management

Table of Contents

Overview

We'll learn the configuration of the web server at vegeta.slc.net (10.22.64.130). This virtual server runs with the following specification:

Specification	
Operating System	Linux Ubuntu 14.04.3 LTS x86_64
RAM	2 GB
Hard Disk	128 GB

This server is used to serve both PHP web app and NodeJs web app. Other than serving web pages, this server also acts as database server and cache server. In order to do so, there're several application stack that is used. There are nginx, apache, pm2 (NodeJS), mysql, and redis.

When serving a HTTP request, the request will go through a pipeline. The pipeline of incoming request is as follow:

1. All incoming request is handled by nginx which listens on port 80
2. And then nginx will read the server configuration at **/etc/nginx/sites-enabled/***
3. The request will be forwarded to the configured upstream according to the request hostname, for example writing.slc.net to NodeJs.
   If there's no suitable server found for the request, then it will default to apache which listens to port 8000, which the default configuration can be found at **/etc/nginx/sites-enabled/default**
4. If it's forwarded to apache, then the request will once again go through apache server configuration which is located at **/etc/apache/sites-enabled/***
   The configurations in this directory will determine where this request will be forwarded to.
5. Finally the response will be returned from the corresponding server to the requesting client.

Install and Configuration

We'll be using the most straight forward and trivial possible way to install and configure the application.

Apache

Apache may be installed when installing Ubuntu, but if it's not, then to manually install apache run the following command:

```
sudo apt-get update

sudo apt-get install apache2
```

Nginx

To install nginx, run the following command:

```
sudo apt-get update

sudo apt-get install nginx
```

PM2 (NodeJS)

PM2 is the process manager for NodeJs. To install PM2, you'll need to install several applications first:

1. Install nodejs

   ```
   sudo apt-get install nodejs
   ```

2. Install npm

   ```
   sudo apt-get install npm
   ```

3. Installs the pm2 package using npm

   ```
   npm install pm2 –g
   ```

4. Configure the auto complete for pm2

   ```
   pm2 completion install
   ```

5. Configure the startup script so that pm2 will auto start on boot

   ```
   pm2 startup
   ```

MySQL

The easiest way to install and configure MySQL database server is running the command below:

```
sudo apt-get install mysql-server
```

And follow the installation wizard till the end, and it should be good to go.

Redis

Unlike the previous installation, redis doesn't have an installation package, so you need to compile the source instead. Follow these steps to compile the redis source:

```
sudo apt-get install make build-essential tcl8.5

wget http://download.redis.io/redis-stable.tar.gz

tar xvzf redis-stable.tar.gz

cd redis-stable

make

sudo make install
```

Redis should be installed now. You may want to move the configuration file to the directory **/etc/redis**. Edit the **redis.conf** find and change the line **daemonize no** to **daemonize yes**. This will make redis to run as a daemon.

Now we only need to configure one more thing that is set the redis startup script. Create a new file named **redis-server.conf** in **/etc/init.d/** with the following content:

```
#!upstart

description "redis server"

# start when the system starts

start on local-filesystems

stop on runlevel [!2345]

# redis 'daemonize = yes' causes a fork

expect fork

# restart the process if it fails
```

```
# but no more than 10 times in a 5 second period

respawn

respawn limit 10 5

# do not connect stdin/stdout/stderr

console log

# create run directory (if necessary)

pre-start script

    mkdir -p /var/run/redis

    chown redis:redis /var/run/redis

end script

# run as redis user and group

setuid redis

setgid redis

# working directory

chdir /var/run/redis

# actual command to run

exec /usr/local/bin/redis-server /etc/redis/redis.conf
```

Now the configuration is finished. You may try to reboot and run **redis-cli** to try it out.

Controlling Web Server Service Daemon

There may be a time where you may want to manage the process of web server, such as restarting the process in the case of memory leaks.

Managing Apache

To manage apache service, you can do it by using the **service** command or by using **apache2ctl** command.

1. Start service

```
sudo service apache2 start
```

2. Stop service

```
sudo service apache2 stop
```

3. Restart service

```
sudo service apache2 restart
```

4. Reload configuration

```
sudo service apache2 reload
```

5. Test configuration

```
apache2ctl -t
```

Managing Nginx

The same goes to managing nginx service, you can do it by using the **service** command or using **nginx** command.

1. Start service

```
sudo service nginx start
```

2. Stop service

```
sudo service nginx stop
```

3. Restart service

```
sudo service nginx restart
```

4. Reload configuration

```
sudo service nginx reload
```

5. Test configuration

```
nginx –t
```

Managing PM2 (NodeJs)

The difference between service and application is that service is the actual pm2 service daemon, and application is the NodeJs web application. Below is the command to control the pm2 service daemon

1. Start service

```
/etc/init.d/pm2-init.sh start
```

2. Stop service

```
/etc/init.d/pm2-init.sh stop
```

3. Restart service

```
/etc/init.d/pm2-init.sh restart
```

Serving PHP Web Application

There are two ways to serve a PHP web app in this server. The first one is through virtual host, and the other is through subdirectory.

Serving via Subdirectory

To serve a PHP web app as a subdirectory of this server, you can just add the folder to the document root which is located at /home (configured at **/etc/apache2/sites-enabled/000-default.conf**).

For example if you add a folder named app there, then you can access the app through **vegeta.slc.net/app** (assuming the hostname of server is vegeta.slc.net).

Serving via Virtual Host

1. Place the codes in a directory, preferably **/home/[app]**
2. Create a new file in **/etc/apache2/sites-available**. The filename should be the hostname of the app appended by **.conf**
3. Edit the new file to configure virtual host. The hostname should be the same with the one configured at DNS server. Directory and document root should point to the public folder of the web app. Below is an example of virtual host configuration:

```
<VirtualHost *:8000>

        Options -Indexes

        ServerName [hostname]

        DocumentRoot /home/[app public]

        <Directory /home/[app public]>

                Allow from all

                Deny from 10.22.64.21

                Order allow,deny

                AllowOverride all

                Options FollowSymLinks MultiViews

        </Directory>

</VirtualHost>
```

4. After saving, change directory to **/etc/apache2/sites-available**.
5. When the current directory is in sites-available, create a soft link to the newly created configuration file here, for example if the configuration file name is **app.slc.conf**, then run this command:

```
root@server:/etc/apache2/sites-available# ln -s ../sites-enabled/app.slc.conf
app.slc.conf
```

6. Restart the apache2 service by running **sudo service apache2 restart**.
7. Make sure that the default apache user (www-data) have the required read or write access to the application code. If not, then just add the permission using **chmod**.
8. The site should be up and running now

Disabling the Web Application

The reason why we put the configuration inside sites-available instead of directly inside **sites-enabled**, is because we can easily enable and disable the web app. If we want to disable, we can just remove the soft link of the configuration file located inside **/etc/apache2/sites-enabled**.

Denying Access to Secret Files

The secret files are the files that may contain credentials that we don't want it to be exposed. Because of the configuration of apache in 10.22.64.130, all files in the /home directory can be accessed through directly open 10.22.64.130 at the browse.

For example, if you separate the database credentials to a separate file name .**env** (just like Laravel does), then you can access the **.env** file through 10.2.64.130/app/.env. This will output the content of the file in plain text.

Please note that this assumes that all file that client can directly access is at a folder named public, and the file that client should not access e.g. source code and **.env** is located at the root of the application.

To disable access to this file while still exposing the /home directory, you should:

1. Create a **.htaccess** file at the root of the application
2. Write this as the content of the file

```
Deny from all
```

3. Now open up the configuration file located in **/etc/apache2/sites-available/[app].conf**, and add another directory block for the directory where the secret files is located (e.g. **/home/app**):

```
<Directory /home/app>

       AllowOverride all

</Directory>
```

4. After saving, restart the apache service by running **sudo service apache2 restart**

5. Request to the secret files should now be denied by the web server

Serving NodeJs Web Application

We use PM2 as the process manager for NodeJs. PM2 makes sure that the node application will be restarted in case of a crash. You can think of PM2 as a tool to daemonize our NodeJs application.

It's pretty direct to start serving a NodeJS app. You just need to add the application to pm2 and it will automatically manage all the other concerns such as restarting and load balancing. In order to do so, you'll need to follow these steps, assuming the app is called **nodeapp**, the hostname is called **nodeapp.slc.net**, and this app listens to port 5050:

1. Put all the codes in **/home/nodeapp/.**
2. Add the application to pm2

```
pm2 start /home/nodeapp/index.js
```

3. Create a new file called **nodeapp.slc.net.conf** inside folder **/etc/nginx/sites-available**. This new file will define the server for our NodeJs app.
   Note that in order to enable web socket, we must sent the required header to our **nodeapp server**,
   Thus the Upgrade and Connection header is configured below.

```
upstream nodeapp {

    server 127.0.0.1:5050;

}

server {

        listen 80;

        server_name nodeapp.slc.net;

        location / {

                proxy_http_version 1.1;

                proxy_set_header Host $http_host;

                proxy_set_header X-Forwarded-For $remote_addr;

                proxy_pass http://nodeapp;

                proxy_set_header Upgrade $http_upgrade;
```

```
                proxy_set_header Connection "upgrade";

        }

}
```

4. Create a soft link to this new file inside the sites-enabled directory (just like how we did for apache).
5. Restart (or reload) the nginx service by running **sudo service nginx restart**.
6. The application should be up now.

PM2 General Commands

Below is the command that is useful for general purpose controlling the NodeJs web application.

1. List application

```
pm2 l
```

2. Start application

```
pm2 start [file]
```

3. Restart application

```
pm2 restart [name]
```

4. Stop application

```
pm2 stop [name]
```

For more detailed command of PM2, you should visit the official documentation here.

Disabling the NodeJs Application

To disable a NodeJS app, you should do several things as stated below:

1. Get the pm2 app name by running **pm2 list**
2. Stop the app from serving request

```
pm2 stop [app name]
```

3. Remove the soft link located at sites-enabled directory **/etc/nginx/sites-enabled**
4. Restart (or reload) the nginx service by running **sudo service nginx restart**

If you want to really remove the app instead of stopping the app from serving request, then you should run **pm2 delete [app name]** instead.

Load Balancing

If there's a need to load balance the request, it can be easily achieved by using nginx for PHP based web app and using PM2 for NodeJS based web app.

PHP Web Application

Below is the steps to load balance PHP web application using nginx:

1. Open the site configuration for the app which needs to be load balanced, for example **/etc/nginx/sites-available/app.conf**
2. Locate the upstream block, and add more server to it. For example

```
upstream app {

    server 127.0.0.1:8080;

    server 192.168.1.2:8080;

}
```

3. Determine the load balance strategy to use. The strategy used defaults to round robin. Other strategies are least-connected (**least_conn** directive) and ip-hash (**ip_hash** directive). To use a strategy, use the related directive inside the upstream block.
4. Restart (or reload) nginx service using **sudo service nginx restart**.

NodeJs Web Application

You can load balance NodeJs app to other server using nginx just like how we did above. But PM2 supports load balancing to available CPU in a server. Below are the steps to load balance NodeJs web application using PM2:

1. Enable load balancer

```
pm2 start app.js -i 0
```

2. Reload all apps

```
pm2 reload all
```

3. Add or reduce process number

```
pm2 scale [app name] [instance number]
```