MERN-Stack Deployment Project

Author: Drew Topel
email: drew@AtlasSoftwareProductions.com
Timezone: US-Pacific

Objectives

- The end-product of this project is a set of documentation that allows me to setup a remote production-ready AWS Ubuntu box with the MERN stack with two Node applications that can access and return data from a Mongo database as described below.
  - MERN = Mongo, Express, React, Node
- The two original demo Node applications must be ported to this setup. These two apps will implement a RESTful API listening on ports 3000 and 4000 respectively,
  - Note: the box itself only accepts traffic on HTTP (80) and HTTPS (443).
- The node apps must run Express for the routing framework. I must be able to access routed functions in the test apps like /auth/login and /data/GetRecord/Id.
- Each Node app must Implement different routing, 2 and 3 levels deep.
- Mongo must be installed locally on the box. The apps must be able to login securely to it and access objects of one data model. That is, the GetRecord method must actually function. It will retrieve the named document from the Mongo Database.
  - For this proof of concept, it is ok to only have one data model.
- The delivery of this project includes a one-page React website that demonstrates the above features. That is, it will
  - Expose access to the routed functions with a simple interface where a text input box is used for the input, a button sends the request, and the result shows up in another text box
- There shall be no CORS issues when running the application. How CORS is handled needs to be spelled out in the documentation.
- Documentation must spell out the directories that are to be used for the following functions
  - Webserver Files
    - When installing a website that works with this installation, what directory should I put the index.html file?
    - What subdirectories are assumed to be present for the webserver to function?
  - Installation directories for each MERN component
  - Configuration file locations

        - Admin tools for Mongo, how to view its current status and data?
        - Admin tools for Nginx
        - I intend to use GIT to update the node applications and website. Where does the GIT repository go?
            - Its OK if there are shell scripts that copy files from there to the directories where the files need to live. But the initial load and subsequent updates of the system will be done via the git repo.

Provided

- I will provide a working setup that runs on Windows that does all of the above so the developer need only port the functionality into the Ubuntu/Nginx environment making the necessary changes and adapting to the required technology as they go.
- Caveats:
    - The MongoDB used in the provided demo is remotely hosted. The solution that I require has the Mongo DB installed on the Ubuntu box so it is accessed locally
    - The front end I provide will not be in React, just vanilla jQuery. But it will show how one exposed API function is accessed.

Background

My startup is developing a program using the MERN stack. Development has all been done on windows and now its time to deploy the app to the production servers. After trying to get this deployment done myself, I've found that it is taking too long due to the learning curve and the uncertainty about what could possibly be wrong due to my lack of background with Linux, and the webserver configurations required to make this work. I must get back to the project itself and not spend time on deployment issues.

I need someone who already knows how to do this setup and can write me a set of instructions that address my specific needs so that I can bring up my remote AWS-based box with all the software needed to support the project and have it work.

Acceptance Testing

For this project to be considered complete, I must be able to follow your written instructions and, without additional help, setup a new AWS Ubuntu Box so that it implements the above requirements.

After I follow your instructions to configure, install and then run the box, the two node apps should run automatically when the box is started up. I should be able to connect to the website presented by the box and send requests and get response from the two apps per the above description.

Finally, I will replace one of the node apps with the actual app from my project. Any changes that I need to make to get the app to connect to the outside interface should be spelled out. The interface is exactly as described. It runs a node server that listens on port 3000 and implements a RESTful API that accesses functions via an Express routing mechanism to access and return data to the interface.

Appendix A

Base Software Requirements, Implementation Approach

Details of Ubuntu Box Software Requirements

- Ubuntu version 18.04

Latest versions of:

- Node
- NPP
- Git
- MongoDb
- Nginx Web Server
- Nginx configured with SSL using Let's Encrypt certificates

Approach

What's important to me is the end result. I need to have the solution quickly and the documentation that comes with it must take me from start to finish so that in the end I have a working solution requiring no tweaks or changes on my side.

I don't need you to invent anything, you should be drawing from your own experience and you should know the pitfalls and solutions for doing an installation like this. I know that there is a ton of information on the web about how to do this and I would expect you to use whatever means possible to finish this, but as an expert in this area, your need to 'look it up' should be minimal.

I am a software engineer with a long background in desktop, proprietary hardware integration, and in the last 15 years, web technology using Windows. I intended to get this deployment working myself and I started down this path with Apache and ran into problems when it came time to connect from the outside to the ports on the inside. This is using a reverse proxy with all the information that I could find about how to do it.

I'm here asking for your help because when I ran into issues, I don't have the current background to know if the issue is due to a configuration error, some missing details in the steps I needed to follow, or in my own application. I did not intend to become an expert in Linux and Apache configuration just to get my project deployed. I just need to know enough to deploy and that's it, I've learned too much already, Lol

This project calls for Nginx and not Apache - that is on purpose because Nginx is a more robust solution for what I need in the end, and so switching to that webserver now is the best time to do that.

I don't care how you solve this problem of setup and configuration, just that it gets done right. I'll help you out too - Here is a very promising link:

How To Set Up a Node.js Application for Production on Ubuntu 18.04
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-18-04

An implied part of the delivery is that the instructions will show me where things should be installed in the directory structure. Where I have files now is not important, I want to follow the lead of others who have installed this and probably have good reasons to use the directories that they did (e.g. for permissions, or speed or compatibility with other tools or whatever.)

This is a bit of information about what I've found on the directory structure. I don't need it to be this way, I just need you to choose the structure on purpose and be able to explain why it is what it is.

Where are others putting their stuff on Linux?

In summary:

```
/opt/webserver/     (your node.js application)
    server.js
    package.json
    node_modules/
    ...

/etc/opt/webserver/
    config.json     (configuration file for your web server)

/srv/opt/webserver/ (opt subdirectory suggested, but not required)
    index.html
    images/
    css/
    ...

/var/opt/webserver
    error.log
    request.log
```

For What Its Worth

None of the below is actually part of the project. These are copied from various websites as I hunted for information on this project. So for what it's worth, I left it here.

Testing Access to the Servers

If a server is running you should be able to get to it with curl

$> curl http://localhost:3000

To get curl to do a POST call using these switches
$> curl -d "param1=value1&param2=value2" -X POST http://qa1.ez-e.com/tester/test/select

Manually Starting the Node Apps

The package.json file is being leveraged to run the three servers. Using the putty command line tool execute the following:

```
    $> npm run start &
```

How to Have the Node Apps Start Automatically Using PM2

You can setup the startup js file to automatically go when Apache starts up using pm2 for configuration.

Full documentation for PM2:
http://pm2.keymetrics.io/docs/usage/process-management/

/var/www/html/testproxy1.js /var/www/html/testproxy2.js /var/www/html/testproxy3.js
Then run the node processes using pm2:

sudo pm2 start testproxy1.js sudo pm2 start testproxy2.js sudo pm2 start testproxy3.js

and got the running processes: We can ceck process log by command:

sudo pm2 logs

Next let's install PM2, a process manager for Node.js applications. PM2 makes it possible to daemonize applications so that they will run in the background as a service. Use npm to install the latest version of PM2 on your server:

```
sudo npm install pm2@latest -g
```

The -g option tells npm to install the module globally, so that it's available system-wide. Let's first use the pm2 start command to run your application, hello.js, in the background:

```
pm2 start hello.js
```

This also adds your application to PM2's process list, which is outputted every time you start an application:

```
Output
[PM2] Spawning PM2 daemon with pm2_home=/home/sammy/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/sammy/hello.js in fork_mode (1 instance)
[PM2] Done.
┌──────────┬────┬──────┬──────┬────────┬─────────┬────────┬──────
─────┬──────────┬──────────────┬──────────┐
│ App name │ id │ mode │ pid  │ status │ restart │ uptime │ cpu │ mem      │ user  │
watching │
├──────────┼────┼──────┼──────┼────────┼─────────┼────────┼──────
─────┼──────────┼──────────────┼──────────┤
│ hello    │ 0  │ fork │ 1338 │ online │ 0       │ 0s     │ 0%  │ 23.0 MB  │ sammy │
disabled │
└──────────┴────┴──────┴──────┴────────┴─────────┴────────┴──────
─────┴──────────┴──────────────┴──────────┘
 Use `pm2 show <id|name>` to get more details about an app
```

As you can see, PM2 automatically assigns an App name (based on the filename, without the .jsextension) and a PM2 id. PM2 also maintains other information, such as the PID of the process, its current status, and memory usage.

Applications that are running under PM2 will be restarted automatically if the application crashes or is killed, but we can take an additional step to get the application to launch on system startup using the startup subcommand. This subcommand generates and configures a startup script to launch PM2 and its managed processes on server boots:

```
pm2 startup systemd
```

The last line of the resulting output will include a command to run with superuser privileges in order to set PM2 to start on boot:

```
Output
```

```
[PM2] Init System found: systemd
[PM2] To setup the Startup Script, copy/paste the following command:
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -
u sammy --hp /home/sammy
```

Run the command from the output, with your username in place of sammy:

```
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -
u sammy --hp /home/sammy
```

As an additional step, we can save the PM2 process list and corresponding environments:

```
pm2 save
```

You have now created a systemd unit that runs pm2 for your user on boot. This pm2 instance, in turn, runs hello.js.
Start the service with systemctl:

```
sudo systemctl start pm2-sammy
```

Check the status of the systemd unit:

```
systemctl status pm2-sammy
```

For a detailed overview of systemd, see Systemd Essentials: Working with Services, Units, and the Journal.
In addition to those we have covered, PM2 provides many subcommands that allow you to manage or look up information about your applications.
Stop an application with this command (specify the PM2 App name or id):

```
pm2 stop app_name_or_id
```

Restart an application:

```
pm2 restart app_name_or_id
```

List the applications currently managed by PM2:

```
pm2 list
```

Get information about a specific application using its App name:

```
pm2 info app_name
```

Now you want to delete the app from the PM2 process list. You just have to enter the following commands:

```
pm2 delete web-interface
```

Since PM2 2.4.0, you can also restart/delete/stop/reload applications via regex (this one will only restart http-1 and http-2 but not http-3):

```
pm2 restart /http-[1,2]/
```

Note : Regex are defined by leading and ending '/' and they are tested against applications name only (not the process id).

Logging

When you run the system, you will see a lot of log messages written to the screen (which will be cut down dramatically as we further down the delivery pipeline). When that is the case, then the file logging that the system does will be even more important to troubleshoot issues that come up.

The logfiles on the Linux system are located in the '**/home/ubuntu/EzGap** directory. In there are three subdirectories for the three servers: EzGap, Casino and Game. They are configured to start a new log every hour when there are messages to capture.

During development on the PC, the logs are located in the **\users\temp** folder.

Apache's own logs are in /var/log/apache2. In there is a history of error and access logs. The most recent ones will be

- Access.log
- error.log

Use TLS

If your app deals with or transmits sensitive data, use Transport Layer Security (TLS) to secure the connection and the data. This technology encrypts data before it is sent from the client to the server, thus preventing some common (and easy) hacks. Although Ajax and POST requests might not be visibly obvious and seem "hidden" in browsers, their network traffic is vulnerable to packet sniffing and man-in-the-middle attacks.
You may be familiar with Secure Socket Layer (SSL) encryption. TLS is simply the next progression of SSL. In other words, if you were using SSL before, consider upgrading to TLS. In general, we recommend Nginx to handle TLS. For a good reference to configure TLS on Nginx (and other servers), see Recommended Server Configurations (Mozilla Wiki).
Also, a handy tool to get a free TLS certificate is Let's Encrypt, a free, automated, and open certificate authority (CA) provided by the Internet Security Research Group (ISRG).

Prevent brute-force attacks against authorization

Make sure login endpoints are protected to make private data more secure.

A simple and powerful technique is to block authorization attempts using two metrics:

1. The first is number of consecutive failed attempts by the same user name and IP address.
2. The second is number of failed attempts from an IP address over some long period of time. For example, block an IP address if it makes 100 failed attempts in one day.

rate-limiter-flexible package provides tools to make this technique easy and fast. You can find an example of brute-force protection in the documentation

Delegate anything possible (e.g. gzip, SSL) to a reverse proxy

Node is awfully bad at doing CPU intensive tasks like gzipping, SSL termination, etc. Instead, use a 'real' middleware services like nginx, HAproxy or cloud vendor services

Otherwise: Your poor single thread will keep busy doing networking tasks instead of dealing with your application core and performance will degrade accordingly

THE GIST popup: click here for quick examples, quotes and code examples

Create a 'Maintenance Endpoint'

Expose a set of system-related information, like memory usage and REPL, etc in a secured API. Although it's highly recommended to rely on standard and battle-tests tools, some valuable information and operations are easier done using code

Otherwise: You'll find that you're performing many "diagnostic deploys" – shipping code to production only to extract some information for diagnostic purposes
click here for quick examples, quotes and code examples

Discover Errors and Downtime Using APM Products

Monitoring and performance products (a.k.a APM) proactively gauge codebase and API so they can auto-magically go beyond traditional monitoring and measure the overall user-experience across services and tiers. For example, some APM products can highlight a transaction that loads too slow on the end-users side while suggesting the root cause