Systems Administration

Fall 2018 – Introduction to Full-Stack Development

Instructor: Matthew Fritter

matt@frit.me

Stepping Back: Docker Swarms

- Last week, we took a look at Docker Swarms, and how we can use Docker Swarms to distributed a web service
  - The Swarm has a manager-worker relationship between nodes, with Manager nodes being capable of adding additional nodes, starting services, and managing the swarm
  - Worker nodes have no abilities to change the operation of the swarm, apart from the ability to leave the swarm
  - Each node can run multiple services, services are spread across nodes
  - Built in load distribution allows requests to be automatically redirected to different nodes

Announcements

- Next week is the last class of the semester – no lab, although if students require help I can be there
- Next class we'll talk about the final exam, jobs in Full Stack development, and I'll hand out a take-home quiz
    - This take-home quiz is due the day of the final exam, and will serve as review for the final exam. It will cover all topics that we covered in class, and is worth the same amount (5%) as the previous quizzes
- I'll begin posting lab marks this weekend. You will have your lab marks prior to the final exam

Systems Administration

What is Systems Administration?

- Systems Administration is a field that focuses on maintenance, configuration, and up-time of servers. Systems administrators (often known as sysadmins) are responsible for a number of tasks:
    - Ensuring that software is up-to-date and patches are applied regularly
    - Maintaining system security and recovering from security breeches
    - In the case of servers that host resources (such as a website, or an intranet file server), ensuring that up-time is maximized and downtime is minimized
    - Provide technical support for the non-technically oriented (i.e. managing accounts, providing access, writing documentation)

Why is it Important?

- Without systems administration, servers will slowly decay as updates fall behind, security vulnerabilities appear, storage space fills up, etc
- As full-stack developers, we’ll often be working on projects alone, or in a small team. We’ll also probably have the best grasp of what is actually going on in the backend
  - Ergo, it’s our responsibility to provide support and maintenance for our projects
  - This will often be included as part of a project contract, i.e. paid for development + some set time period of maintenance and support after development is complete
- Systems Administration is also a standalone field, and many companies that have in-house networks and servers will hire professional Systems Administrators to provide 24/7 maintenance

Automation is the Sysadmin’s Friend

- For the full-stack developer with a lot of projects on the go, or the professional system administrator with hundreds of servers to manage, manually performing regular maintenance isn’t an option
- Luckily, many regular maintenance tasks can be automated with relative ease
  - These automation scripts can even be pre-baked into docker images and bash build scripts, making maintenance automation on new systems relatively painless
- There are a vast number of tools out there for automation, many pre-baked into our operating systems – think about why the Windows startup folder has existed for all these years
  - We’ll be focusing on a tool common in Linux environments - Cron

Cron

- Cron is a scheduling utility that is commonly available on Linux and other Unix-based systems (such as Mac and BSD)
- The origins of the cron utility go way back to the days of computer mainframes and the first multi-user systems – originally developed for Unix in the late 1970s
- Cron uses a file known as the Crontab (short for Cron Table) to execute jobs on timers
    - Timers can be set for minutes, hours, days of the month, months, and days of the week
    - i.e. you can have a job set to run every Tuesday, or every minute, or at 12:00 AM every December 25th

The Crontab format

- The Crontab is simply a plain text file. Every user has their own individual crontab, in addition to a system crontab that is accessible only to administrators
- Each job follows this format:

```
* * * * * <job>
```

- The initial 5 values are the timing values. They correspond to minute, hour, day of the month, month, and day of the week. An asterisk character (*) means run on every interval – the above format with 5 stars would run every minute of every hour of every day of every month.
- Here's an example of a crontab entry that would run at 12:30 PM every Monday:

```
30 12 * * 1 /var/www/myScript.sh > /var/www/log.txt
```

Editing the Crontab

- When editing the Crontab, the user is important. Cron jobs are run using the permissions of the owners of the Crontab. For many administration tasks, we'll want to use the root Crontab
  - Note that this could be dangerous. It is not recommended that you use the root Crontab for tasks that involve pulling resources from possibly untrustworthy resources (such as a Github repo or a website)
- To edit the root Crontab, use the following command:

```
sudo crontab -e
```

  - This will open the Crontab in Vim. Press 'i' to enter insert mode, type in your Cron job, then hit 'escape', followed by ':w' to write the file and ':q' to exit

A Typical Crontab Entry

- Consider the following Crontab entry. What does it do?

```
0 12 * * 6 yum update -y >> /var/www/log.txt
```

A Typical Crontab Entry

- Consider the following Crontab entry. What does it do?

```
0 12 * * 6 yum update -y >> /var/www/log.txt
```

- This Crontab entry updates currently installed packages every Saturday at noon, and appends the output of the update command to a log file.
- Some important notes here:
  - Note that we use the -y flag for the update. This automatically approves the update. Cron jobs that require user input (such as update without the -y flag, which requires the user approve the updates) will fail
  - It's always a good idea to have your Cron jobs append their output to a log file. This makes debugging far easier if something isn't working properly.

Using Cron with Bash

- While we could run individual commands as individual cron jobs, it will often make more sense to use a Bash script and simply schedule our Bash script to run at a set interval
- To run a bash script as a Cron job, simply use the ‘sh’ command and pass the location of your bash script. Again, be careful of what user you’re under – home locations will change between users, so a script in your user home will not be in your root’s home!

```
* * * * * sh /path/to/script.sh >> /path/to/log.txt
```

- Using Bash, you can perform a number of useful maintenance tasks and then use Cron to automate it

Handling Things Remotely

- If we're dealing with a large number of servers, it's a bit of a chore to log in to each server via SSH and read through logs to make sure everything is alright
- Much easier: automatic emails that provide us with useful information regarding the server
- Setting up email systems can be complicated, but for administration email it really doesn't have to be:
  - Install Heirloom mailx. This is a Mail Transfer Agent (MTA) that supports Simple Mail Transfer Protocol (SMTP) for sending out emails

    ```
    yum install mailx
    ```
  - Once you've installed an MTA (like mailx), you can simply use the 'mail' command in bash to send an email

Example: Checking Login Logs

- The following script will capture both failed and successful password logins via SSH on our server, then send it as an email to the specified email account:

```
#!/bin/bash
passlog=$(grep “password” /var/log/secure)
echo $passlog | mail -s “Server Logins” mattfritter@gmail.com
```

- Note that most webmail inboxes will probably throw these emails into the spam or junk folders by default, so you’ll probably have to whitelist them
  - This is because we’re sending mail without a qualified domain name, so the sender looks suspicious

Backing Up to Remote Servers

- If we’re using a script to perform a backup, it’s often a good idea to create a backup on a remote server
  - While local backups can be useful (say, you accidentally nuke a database – a local backup in SQL format will save you), if the server goes down or there is a security breech and you lose control of your server, the local backup won’t help
- There are a number of potential options here:
  - Push to a remote Github/Bitbucket/etc respository – simple, but may be available to the public (not desirable)
  - If the filesize is small, email it to yourself – simple, but won’t work with larger file sizes and is thus not suitable for files that may change in size (like databases)
  - Secure Copy (scp) – More complex, but most secure and can handle very large files without a problem

Using Attachments in Mail

- For smaller files and directories, setting up a bash script to email ourselves a backup is an option. This is simple and doesn’t require interfacing with other web services (Github) or external SSH servers (Secure Copy)
- Consider the following bash script:

```
#!/bin/bash

tar -czvf backup.tar.gz /var/www/myapp

echo Backup | mail -s “Website Backup” -a /var/www/backup.tar.gz
mattfritter@gmail.com
```

- This script would automatically compress our Flask application into a tar archive, and then email it to us
    - This will cause problems on many webmail clients, as our already suspicious address is now sending potentially dangerous attachments

Using Secure Copy

- If you have a second server available, you can use the Secure Copy command ‘scp’ to securely copy files to and from the server using SSH
- Normally, the scp command will prompt the user to enter the password for the specified SSH user. Obviously, if we’re using bash, we want it to be able to execute without the need for user input
  - We’ll use SSH keys instead to allow for passwordless authentication
- First, run ‘ssh-keygen -t rsa’ to generate an SSH key. Then, run ‘ssh-copy-id <username>@<serveraddress>’. You will be prompted to enter the password for the user on the remote server. Once you do, the SSH key will be copied over, allowing us to access it without a password.
- To use scp with an SSH key, use the following syntax:

```
scp -i <SSHpublickey> <localfilename>
<user>@<serveraddress>:/home/<user>/<filename>
```

Things You'll Want to Backup

- Databases have the utmost priority in terms of backup. Databases on web applications will usually contain user data (usernames, passwords, emails, application specific data) that will be next to impossible to regenerate if it is lost.
- Dumping out a mySQL database into a single file is simple:

```
mysqldump --all-databases > backup.sql
```

- Backing up SQLite databases is even easier – they're already contained in a single file, which you can simply compress and then send to a remote location
- After databases, the next thing you'll want to backup is application-specific configuration files:
  - Apache config files
  - Laravel/Flask configuration files
- If you're using SSL to provide an HTTPS domain name, you'll also want to make sure you back up your SSL certifications
- Depending on whether you're using remote source control like Git+Github, backing up your actual backend code may be unnecessary

Cron Time Intervals

- The intervals you use for many of these jobs will be dependent on the application
- If your application sees a large amount of content created every day, you'll probably want to perform backups daily or even hourly
  - However, you must balance your backup schedule with the understanding that backups take space, and space is limited – you'll want to keep a history of backups (so you can revert to a previous backup if necessary)
- If your application sees less traffic, then a weekly or bi-weekly backup schedule may be more reasonable
- You'll probably want to perform package updates on a daily or hourly basis. Fast enough to quickly patch the server if a security patch comes out, but not so fast that you become a burden on the package management mirrors
- Email updates based on log files – entirely dependent on how often you think you'll need to check up on the server.

Recap

- We can use a combination of Bash and Cron to automate many server tasks and make our lives easier as systems administrators:
  - Perform package updates using yum update -y
  - Send automated emails of important log files (SSH logs, Apache logs, Fail2Ban/mod_security logs) so that we can keep tabs on how the server is doing without having to log in
  - Backup important data like databases and configuration files, either via email or using a remote server through Secure Copy or via service like Github
- The beauty of automation – spend some time once working on your scripts, and then sit back and let it handle itself

That’s All Folks!