COSC349 Assignment 1

Bradley Windybank - 4353100

August 2019

1 About

1.1 Application Explanation

This application is a collaborative task list application. It uses three virtual machines that communicate with each other over a local network to make everything work. One VM hosts a web app that the user can interact with to view, save, and delete tasks. Another VM takes requests from the other two VMs and retrieves, deletes or saves tasks depending on the contents of the request. The third VM is a Node.js JavaScript application that prints the tasks to a PDF. This VM can be interacted with through the use of a bash script.

1.2 VM interaction

The three VMs interact over a private network. Each machine has its own static IP address. The React web app and PDF program communicate to the Express server through RESTful API requests. They recieve information back in JSON format. The reason for seperating each of my services into seperate VMs allows for several advantages over hosting them all on one VM. One benefit is that each VM can be restarted independently rather than

having to restart one VM. This improves reload times when a service needs to be reloaded in a case of error or code modification or the like. VM separation also allows for better separation of the commands that provision the virtual machines. This means that each set up process does not interfere with that of the others and it is also easier to make changes to each provisioning script without worrying about affecting the other services. VM separation also allows for easy expansion of services accessing the server. This means another VM could be easily created to host a desktop GUI program that could be used to bulk edit tasks, and a mobile app for task management could communicate over the network to talk with the server with ease. With seperating each service onto a seperate VM, resources such as memory and storage space can be more specifically allocated to each service, so as to reduce unnecessary resource consumption. Though this may only be a benefit with a large scale system with more users or more complex services.

2 VM Provisioning

2.1 Initial Setup

- After installing prerequisites, (Vagrant, VirtualBox) open up a terminal window.
- Change directory to the folder you want this project to be enclosed within.
- Then clone the repository.
- Next run `cd vagrant-multi-VM`
- The project is viewable and editable from this directory.
- The command `vagrant up` in terminal will run the project.
- You can now view the web app from http://localhost:3001 and can enter and delete notes.
- More information regarding setup is available from the GitHub readme file.

2.2 Download and Build Time

- 400MB for downloads of packages/dependencies during provisioning.
- 270MB download for box file.
- Repo only 5MB (Zipped).
- `vagrant up` from scratch or `vagrant up --provision` takes 4.5 to 5 minutes to complete (this includes downloads and provisioning). Tested on University Library WiFi (50Mbps Download Speed).
- `vagrant up` any time after (without provisioning) only takes 1 minute.

2.3 Setup Automation

Setup automation is done using inline shell scripts within the vagrantfile. Therefore, all a user needs to do to set up the project, as shown in the steps above, is install Vagrant and Virtualbox, then run `vagrant up` to start the automatic provisioning. The steps taken in each provisioning script are as follows.

Server VM

- The package sources are updated with new entries for node and mongodb.
- Node and Mongo are installed.
- The `forever` package that keeps node scripts running permanently is installed using NPM.
- The working directory is then changed to that of the source files for the server VM.
- `npm install` is run to install needed node packages.
- The server node script is now run using `forever`

Web App VM

- The package sources are updated with new entries for node.
- Node is installed.
- The working directory is then changed to that of the source files for the web app VM.
- `npm install` is run to install needed node packages.
- The web app is then run using `nohup` which runs the script in the background.

PDF VM

- The package sources are updated with new entries for node.
- Node is installed.
- The working directory is then changed to that of the source files for the web app VM.
- `npm install` is run to install needed node packages.
- The PDF node script is now run.
- The output file is then moved to the root directory.

3 Use of the Application

The application is primarily used through a web interface. The user can type a task into the text box, then save it using the button below the box. The user can then delete all tasks using the red button marked 'Delete All'. Or they can delete tasks one by one using the 'x' icons on the right of each task.

If the user wants to print out all tasks to a formatted PDF file, they need to open up a terminal window in the root directory of the project, and

then enter `chmod +x task-pdf-script.sh`, then the script can be run by entering `./task-pdf-script.sh` after which the user can then find the pdf file with the tasks listed within. This command needs to be then re-entered every time the user wants to update the file with the current contents of the list of tasks.

Below should be a link containing a recording of the mentioned interactions with the system.

(INCLUDE RECORDING OF ALL FUNCTIONS)

4 Further Expansion

4.1 Use of Git/GitHub

For Easy Modification by Others

4.2 Setup for Devlopment and Testing

4.3 Possible Modifications

editing tasks

4.4 Re-Running VM's

5 Development Process

5.1 Issue Driven Development

5.2 Development Timeline

(Commit History)