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 approx for downloads of packages/dependencies during provisioning.
- 270MB approx download for box file.
- 5MB repo clone (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

The use of Git/GitHub in this project gives several advantages. One is that it allows for easy version control. The commit history can be looked back on and changes can be reverted or branched if needed. It also allows for easy modification by others and collaboration with others. Many people can work on the same repo with ease and the codebase can also be forked into new projects.

4.2 Setup for Devlopment and Testing

The setup for development and testing is as follows (for macOS):

Initial Setup

- Install mongoDB by following instructions here: `https://treehouse.github.io/installation-guides/mac/mongo-mac.html`
- Install VSCode (or editor of choice): `https://code.visualstudio.com/`

- Install Node LTS version: `https://nodejs.org/en/download/`
- Open up a terminal window.
- Change directory to the folder you want this project to be enclosed within.
- Then clone the repository `git clone <URL>` at: `https://github.com/bradwindy/vagrant-multi-VM.git`
- This folder can now be opened in your editor of choice.

Server

- First the server must be run. In a terminal window at the root directory of the project, enter `cd vm-2` which takes you to the server directory. Next enter `npm run start` to start the server.
- If any changes are made to the server code, you will need to stop the server process using Ctrl+C and restart it again using `npm run start` before any changes are visible.

Web App

- To run a version of the web app for development. First you must change all URLs in the code that mention the IP address and port 192.168.55.11:3000 to instead be localhost:3000
- Then open another terminal window at the root directory of the project, enter `cd vm-1` which takes you to the web app directory. Next enter `npm run start` to start the web app. Press 'y' when prompted about ports.
- The app is now viewable at: `http://localhost:3001` Changes made to any of the code in the web app directory will be automatically loaded on save.

PDF printing program

- In a terminal window at the root directory of the project, enter `cd vm-3` which takes you to the PDF program directory.
- Then enter `npm run start` to run the PDF printing program.
- If any changes are made to the list of tasks, this program can just be run again to overwrite the PDF file.

All these instructions are also available in the README file in the repo.

4.3 Possible Modifications

Once development setup is complete, two modifications that could be made to further the project are as follows:

- There could be a feature where the user is able to edit tasks. I imagine the user would have a edit button next to the 'x' button for each task, and once clicked, the contents of the task would populate the a popup with a textbox. This popup would also have a 'Save Edit' and a 'Cancel' button. Once the user changes the contents and clicks save, the new contents would be posted to the API on the server using Axios. The server would recieve this post request and there would be a new route created using Express that would update the DB with the new contents of the task but keeping the same ID.
- A second extension could be the idea of incorporating due dates into the tasks so that the user could know when they need to be completed by. This would especially be helpful when the system is used in a collaborative sense, as it communicates more information with each task. Several changes would need to be made to the code to achieve this. The DB structure would need to be updated to incorporate this new value. All API routes would need to be updated to make sure they are saving this information. A new field would also need to be added into the form on the web app to take in the due date using a

date picker. An npm library could be used for this. The state within the web app and all display functions will also need to be updated to properly record and display this new value.

- A mobile app that communicates with the server VM over the network could also be a possible extension. This would require development of a mobile app that is similar in interface to the web app. This app would then need to be able to communicate with the server over the network to be able to send POST, GET, and DELETE requests. The server has its port 3000 forwarded to the port 3002 on the host machine and so a mobile app would be able to access this port on the host machine over the network to communicate with the server.

4.4 Re-Running VM's

To reload these changes into the VMs once they have been tested using the development setup, enter `vagrant reload --provision` in the root directory of the project.

5 Development Process

5.1 Issue Driven Development

One of the development methods I implemented when developing this project was the concept of issue driven development. This is the concept of using issues to reference all elements of the development of your project, from features and enhancements, to bugs and issues, to documentation, testing and more. Most or all of the commits should reference an issue, this way, it is clear to see what work has been done regarding each feature and what each commit is involved with. I have included issues for both features and documentation in my project and these can be seen on GitHub in both my closed and open issues sections.

5.2 Development Timeline

All progress mentioned here is also referenced in my commit history of my project. The project started off by using the code I had written in the lab for my VM weather site. At the start I also added a readme file and a license file. I then removed the contents of the original vagrant file and replaced it with code to set up two virtual machines, one empty, and the other a basic Node server. I tested the server and was able to use curl to get information back from the server VM using port forwarding. I then removed the rest of the old files and continued to update my readme file.

I then initialised the React web app in it's respective folder and rewrote the server to use Express.js. I then wrote the route for posting notes to the server and handled their saving into the DB, along with the route to read notes too. These were then tested using the Postman app, which is a program used to test APIs. Next I used Axios within the React app to request notes from the server and then wrote the code to display them. Then I installed Bootstrap, a front end component library. This allowed me to use their components to simplify frontend development.

I then added the ability to add new notes within the React app, this includes all necessary routes, along with updating the readme. I then updated the UI and added delete one and delete all routes along with corresponding buttons on the web app.

I then moved to writing the code for printing the PDF from the list of tasks, starting with test data then adding code to get the tasks from the server instead. I then finalised some parts of the code, got the db to be initialised on start, and then fixed some bugs.