COSC349 Assignment 2

Bradley Windybank - 4353100

August 2019

1 About

1.1 Application Explanation

This application is a collaborative task list application. It uses two cloud hosted virtual machines that communicate with each other over the internet, along with a cloud database API for storage of tasks. One VM hosts a web app that the user can interact with to view, save, and delete tasks. Another VM takes requests and retrieves, deletes or saves tasks depending on the contents of the request.

1.2 VM interaction

The React web app VM communicates to the Express server VM through RESTful API requests. It receives information back in JSON format. The server VM sends requests over the internet to the MongoDB API hosted in the cloud and receives information back which is in turn passed to the web app to update the user. I chose this database service as it is compatible with my existing mongoDB code within my server application, and is able to be created through the Azure portal, streamlining development. The separation of the database out into a cloud service also simplifies development as there

no longer needs to be any code in the provisioning script for the creation of a mongoDB database and the second VM is now solely for the server API code.

2 VM Deployment

2.1 Semi-Automated Deployment Steps

Prerequisites

- Azure CLI is needed. To install on macOS, install xcode extensions if not already installed. This is needed for the installation of homebrew.
  - `xcode-select --install`
- Install homebrew if not already installed so we can use it to install the Azure CLI.
  - `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
- Now we install the Azure CLI using homebrew.
  - `brew update && brew install azure-cli`
- An Azure account also needs to be created.

Process

This process is for macOS and should be very similar on Linux. The steps are also available in the read-me in the GitHub repo.

- After installing prerequisites, open up a terminal window.

- Change directory to the folder you want this project to be enclosed within.
- Then clone the repo at:
- `https://github.com/bradwindy/azure-multivm-webapp`
- Then enter `cd azure-multivm-webapp`
- The project is viewable and editable from this directory.
- **IMPORTANT: Change the admin password entry in parameters.json**
- Next enter `az login` and follow the steps to login to the Azure CLI.
- Next, enter `az account show --out json` , and save the value in the ID field somewhere. This is your individual subscription ID and will be needed for setup of the virtual machines.
- Then, enter `./deploy.sh` and follow the steps that appear in the terminal window. The subscription ID is the ID value we obtained earlier. A good resource group and deployment name could be something like `multivmwebapp`. The location should be `eastasia`, look into the template file to change all the locations if you want to be in a different location. This process will take a while to complete. Around 6 - 7 minutes.
- The deployment process is not fully automated at this stage. So next go to the URL `https://portal.azure.com/` and log in.
- Here open **Resource Groups** from the sidebar and then select the resource group just created before during the deploy process.
- Click the **Add** button up the top of the new view within the browser window to add a new resource to this group.
- Search `Azure Cosmos DB` in the serch bar, and click the create button from its resource page.
- In the form that appears, add any account name to the account name field, and select **Azure Cosmos DB for MongoDB API** as the API option.

- Choose **East Asia** as the location as done before (or whichever location the values in template.json were changed to).
- Then click **Review + Create** and then **Create** as they appear. Deployment of the database may take some time.
- Next, click **Go to Resource** and then select the **Connection String** option on the inner sidebar.
- Copy the primary connection string to your keyboard, and save the value somewhere for safekeeping on your computer for the time being.
- **IMPORTANT:** Replace the `==` in the connection string with `%3D%3D` or you will get errors due to the way mongo parses URLs.
- Next, open **Resource Groups** from the sidebar again, and again choose the correct group.
- Click the **vm-1-ip** entry in the list that appears to open its resource page and select the **Configuration** option on the inner sidebar.
- Change the IP assignment from Dynamic to **Static**, then click save. Once the change has saved, copy the IP address and save it in the same place as the connection string that was saved before.
- **Do the same above** for *vm-2-ip*, making sure to save the IP again.
- Open up a terminal window again, and ssh into the API server VM using the command `ssh vmadmin@<VM-2 IP YOU COPIED>`
- Log-in using the password you chose above.
- Enter `sudo su` and then

  `cd /var/lib/waagent/custom-script/download/0/vm-2`
- Then enter `nano app2.js` to edit the server file.
- Uncomment the mongo connection function by removing the `/* */` parts from either end of the statement.
- Remove the current connection string.

- (The part that starts with `mongodb://`)
- Replace it with the modified connection string from earlier.
- `Ctrl-o` then `enter` then `Ctrl-x` saves the changes.
- Then run `forever start app2.js` and then `exit` and `exit` to leave su and then ssh.
- Next, ssh into the web app VM using the command:
- `ssh vmadmin@<VM-1 IP YOU COPIED>`
- Enter `sudo su` in terminal and then enter (all one line):

  ```
  cd /var/lib/waagent/custom-script/download/0/
  vm-1/src/components
  ```

- Enter `nano Nav.jsx` into terminal, and replace the IP `13.70.6.93` with the IP of the **VM-2 VM** that was saved earlier. There are two entries of it in this file.
- `Ctrl-o` then `enter` then `Ctrl-x` saves the changes again.
- **Do the same for NoteList.jsx**, it has two addresses to be changed as well.
- Once done, run `npm run build` to build the files to be served. Then serve the web page using `forever start server.js` and then `exit` and `exit` to leave su and then ssh.
- **All done.** The web app is now available by entering the IP of VM-1 in your browser.
- Check out my example available at `http://13.70.0.241/`

2.2 Setup Automation Workflow

Deployment Script

Setup automation is done using a shell script and some template files generated on the Azure Portal. This process will be explained more down below. How this script works, is that it takes some information regarding setup parameters from user in both the command line and from a pre-written file, it then runs an Azure CLI deployment command with this information and the template.json file included. This results in a resource group being created on the Azure platform containing two virtual machines with necessary networking components, such as a virtual network, and appropriate security groups. Each VM then has a provisioning script run within to set up the necessary files and dependencies that are needed for its operation. This process is as follows.

Provisioning Scripts

Server VM

- The repo is first cloned into the VM. The vm-2 folder is extracted and the rest of the files from the repo are deleted.
- Node is installed.
- `npm install` is run to install the necessary dependencies.
- Then the command line tool `forever` is installed globally so that the server file can be run correctly later.
- The rest of the provisioning steps are run by the user.

Web App VM

- The repo is first cloned into the VM. The vm-1 folder is extracted and the rest of the files from the repo are deleted.

- Node is installed.
- `npm install` is run to install the necessary dependencies.
- Then the command line tool `forever` is installed globally so that the web app can be served correctly later.
- The rest of the provisioning steps are run by the user.

2.3 How Deployment was Automated

The deploy.sh script and the template.json file were both created using tools available on the Azure Cloud Portal. I created a resource group containing all the necessary virtual machines along with associated networking settings. The resource group is then able to exported to a template from the portal. This results in a zip containing these files being downloaded. The template and deploy script were then modified slightly to accept a password parameter and provisioning scripts. Once this was done, I set up the virtual machines from the terminal using these files and documented the process above to fit my exact project.

I wasn't able to fully automate the setup as I have limited time, and in this time, I wasn't able to find a way to correctly pass in the IP's of the VM's and the Mongo connection string. This is the reason why this process has to be done by the user. It results in simpler project deployment instructions, compared to it being fully done through the portal, but not as simple as I would have liked. I would prefer the user to just run one script and be able to enter all necessary information directly into that script so the portal never has to be accessed during the setup process.

This has been is my reason for choosing Azure over AWS as a cloud provider. AWS probably has similar tools for this exact process, but the documentation for these was a lot more complicated to navigate and so I wasn't able to find out easily how to complete this process on AWS instead of Azure.

3 Use of the Application

3.1 How to Reach the Web App

My deployed web app is available at: `http://13.70.0.241/`

3.2 How to Use the Web App

The application is 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.

3.3 Evidence of Use

Here is a link to a YouTube video that shows me using my application. You can see it is through the link given above and I am able to add a remove tasks as expected.

`https://youtu.be/ccWDpbCgu4c`

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

- The server code must be modified with an actual mongo DB connection string. You can use one hosted in the cloud or on your local device.
- Once the string is replaced, 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 13.70.6.93:3000 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 build && npm run start` to start the web app.
- The app is now viewable at: `http://localhost`

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

5 Development Timeline

Development of the system started with researching into how automatic deployment could work for this project. I looked at the documentation regarding automation for both Azure and AWS. I found that Azure had clearer documentation for automated template deployment, and so I went down that route. I then created the template files using the tools available on the Azure Cloud Portal. Next I added some of the files I would need for this assignment from work I had done for the previous assignment. The system I am implementing here is same in function as my previous assignment, but interacts with the virtual machines in a slightly different way, and is deployed to the cloud.

I then modified the existing server/web app networking code and deployment networking code to work with the cloud environment. I also, as a learning experience more than anything, adding loading wheels to the buttons in the UI when requests to the server are happening. As with the cloud environment, requests now take a while, and I want to inform that user that something is indeed happening in the background.

The provisioning scripts were then created and incorporated into the deployment code. They were then tested in the deployment process and updated until they were doing what I wanted them to do. The server and web

app code along with the network security groups were then modified again to work together and successfully send requests across the network.