Shaun Liew 2553068
Josh Casey 3023930

COSC349 Assignment 1: effecting the portable building and deployment of software applications using virtualisation.

By Josh Casey & Shaun Liew

The purpose of three Virtual Machines and how they interact:

This application simulates a basic design and interpretation of an e-commerce store. The representation consists of: a website interface that manages client interaction such as the ability to place orders on the displayed items; an administrative website which displays all the orders that have been placed as well as the ability to affect the shipping status of those orders; and a database server which stores all the information about the products as well as the orders placed.

We created this interpretation with three Virtual Machines (VMs) that were built through Vagrant on ubuntu/xenial64 virtual boxes. Two of the Virtual Machines are web servers (client and administrative web pages), with the third VM existing as a virtual database server. The database server VM directly interacts with both webservers separately. The database server provides product data to the client webserver and collects the orders placed from same webserver. The database server provides information on the orders to the admin webserver, allowing the display of all current orders on the webserver.

The communication between each webserver and the database server is achievable using private networks hosted on different manually assigned IP addresses (i.e. client server IP: “192.168.2.11”, database server IP: “192.168.2.12”, admin server IP: “192.168.2.13”). The database VM provision contains a MYSQL server which we used to store and communicate data between the VMs. For the webservers, PHP installed on the VMs allowed us to build the website interfaces and communicate with the database server using SQL query.

The expected approximate download volumes for first and subsequent builds:

The initial download of a ubuntu/xenial64 Vagrant box is about 270 megabytes which is stored locally, meaning that any other VMs of these sort that are requested there after will reuse the same box file that has already been cached. The cloning of the required git repository should be around 1 megabyte to download. Subsequent downloading and building of the application after these initial requirements are heavily minimized due to the caching of the VMs.

The automatic build and start of application once initiated:

Shaun Liew 2553068
Josh Casey 3023930

Manual interaction is not required for our applications build process - our application builds and starts by itself once initiated. We have included test input values within our code to demonstrate the automatic build process without required user interaction for providing input.

Describe how a user should use your program:

1) Ensure the latest versions of Vagrant and VirtualBox are installed.
2) Make a new (or existing) directory that you want to use our program in and move into that directory.
3) Git clone our repository: git clone https://github.com/josh-casey/349-assign1.git
4) Move into the git cloned repository.
5) Begin the process of building the VMs and applications by using the command: vagrant up
6) Once initiated, connect to
    1) localhost:8080 for the client webserver
    2) localhost:8081 for the admin webserver
7) On the client server, the user will see a range of products listed as well as their respective prices. The user can select as many of each time they want to order, showing the total price after each selection. Once finished selecting the items, the user can then submit an order by inputting their details on the short form listed below that takes information of clients' name, email, and address. Once submitted, the client webpage will refresh back to its original state, allowing for the user to place however many subsequent orders they desire.
8) Once the user has finished placing orders, they can connect to the admin webserver on localhost:8081. On this page the user will see a table containing information of what the user had submitted for each order that was placed. The only user interaction available on the admin page is to update the shipping status of the order. There is a toggle status button next to each order which when clicked, will immediately refresh the page, and update the shipping status of the order.

Possible (at least two) modifications/extensions a developer can make to the application's code:

The way in which we have built our application facilitates for developers to be able to make modifications to our code easily. For example, our inclusion of JavaScript was coded in a sense of modularity, allowing for a simple and concise approach for a developer to add/remove functions as well as debug code. Additionally, our MySQL code for our database server comprises of tables for items, allowing a developer to manipulate and add/remove products with no complications.

Our implementation of an online store is extremely basic in its form, allowing for a developer to contribute a substantial amount of different possible extensions and modifications to our application.
Some potential modifications:

1) *A log-in page for the admin webserver:*
   To represent and simulate an e-commerce software more accurately, the implementation of a required login to provide security to the details that are displayed when entering the admin webserver. A developer can clone our git repository and make changes to the admin server code which can include a JavaScript function that allows this requirement of having to log-in to access sensitive information such as the details of customer orders.

2) *Limits on product stock:*
   This is another potential extension that could further improve our depiction of a mock e-commerce platform as including the stock amount for each product limits the amount a client can order. This can be executed by allowing the admin server to specify the amount of stock available for each given product. This information can then be processed through the database which then updates the stock limit on the client webserver. This modification allows for a limit on the amounts of each product you can purchase as our current basic implementation allows for orders of any amount of each product. Again, this modification can be achieved by adding a function in the admin webserver code that allows for a submission of stock amount for a given product. The SQL database code can be adjusted to accept stock values and provide them as a constraint to the client webserver. The client webserver code can be adjusted to accept stock as a variable and possibly enforce an error message when a user attempts to order over the stock limit set by the admin server for a given product.

These changes made to webserver scripts should take effect immediately and just requires a refresh of the web browser. For modifications made to the Vagrantfile or provisioning shell scripts, for these changes to take effect a reload command should be called on vagrant. This can be executed by running the command **vagrant reload --provision**, where the --provision flag forces the provisioners to re-run.

References:

https://altitude.otago.ac.nz/cosc349/vagrant-multivm
- For initial creation of Vagrantfile for multiple VMs.