**COMP-10177 – Network Capstone Project**

# Simple Automated Infrastructure Deployment through Web Interface

***Project Documentation***

**Project Author:** Trever Fulton

**Student Number:** 000829091

**Date:** December 10th, 2023

I, Trever Fulton - 000829091, hereby state that this project is solely my work and no other student helped or did my work for me.

## Executive Summary

This project was originally set out to implement a simple web interface which allows a client to seamlessly deploy an interconnected network infrastructure. The form data from the web interface would be processed on the server, which chooses and deploys the proper configuration files required to build the infrastructure over a network.

In practice, I was successfully able to implement the basic requirements set when planning this project. While there were some more technical difficulties that will be disclosed later in the document, the core elements are still in place and function as intended. Once you submit the form data from the interface, the information along with any related files get stored on a build server. A python script is then run to take the information and use it to deploy the proper ansible playbooks, which then build a network of virtual machines on a target server.

Some technical difficulties that I encountered include (but weren’t limited to): lackluster hardware, improper hypervisor choice, script permission errors, program logical oversights, package version and configuration conflicts, and other generic issues. The hardware and hypervisor were changed to alternative choices that suited the scope of this project better, and all other issues were eventually resolved through various troubleshooting strategies. It is also worth noting that during the development of this project, the “OpenVPN” server option was not included due to time constraints and lack of a properly supported install script on alpine Linux, the Postgres database option was integrated, but then changed to MariaDB so that the [demo project](https://github.com/profjamesmohawk/trs) could be supported, and use of JavaScript was added to add responsiveness and template duplication to the webpage .

## *Network Diagram (Proposed)*

A diagram of a software development process

Description automatically generated

## *Network Diagram (Final)*

A diagram of a software company

Description automatically generated

## *System Description*

**Servers:**

* Hostname: “Buildserver”
* OS: Debian 12.1. 0
* Roles: Webstack, Ansible Control Node, Management/Master scripts & services,
* Hostname: “VmBox”
* OS: Debain 12.1.0
* Roles: Target Device, Virtual Environment (VirtualBox), Receives and Builds Virtual Infrastructure

**Technologies Used:**

A purple square with a white letter b

Description automatically generated

**Technology:**  BootStrap

**Desc:**  BootStrap is an open-source CSS framework that prioritizes responsive webpages. It also offers various free CSS, HTML, and JavaScript templates for anyone to use and customize. Additionally, BootStrap studio is a GUI tool to help design webpages.

A green hexagon with a white letter n

Description automatically generated

**Technology:**  Nginx

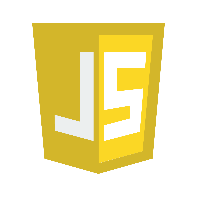
**Desc:**  Nginx is an open-source web server that can also act as a load balancer, mail proxy and/or HTTP cache. It is lighter weight than Apache and focuses on being easily scalable which makes it an excellent choice for production infrastructure.

A blue oval with black letters

Description automatically generated

**Technology:** PHP

**Desc:**  PHP is a web development focused scripting language that can be used to embed scripts into html. It is used in the back end to receive form information about the infrastructure configuration and export it to the server.



**Technology:** JavaScript

**Desc:** JavaScript is a programming language that is used to create dynamic content on webpages. In this project it helps with some webpage animations and dynamically rendering multiple “server option” templates based on a interactable number input.

A black circle with a white letter in it

Description automatically generated

**Technology:** Ansible

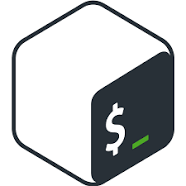
**Desc:** Ansible is an open-source software suite that implements Infrastructure as Code. It allows you to provision, configure, and manage machines, and is considered “agentless” meaning that no additional software is required on every node. In this project Ansible configures a target server to build desired VM’s, and then takes control of those VM’s to customize as per the form data as well as manage them in the future.

A blue and yellow snake logo

Description automatically generated

**Technology:** Python

**Desc:** Python is a general purpose and high-level programming language designed with readability in mind. This project uses a python script as the main script that processes the form data, generates an ansible inventory, and chooses which ansible playbooks to run in order to correctly and effectively build the desired infrastructure.



**Technology:** Bash

**Desc:** Bash is a shell/command line interpreter for Unix-based systems. Using the bash programming language scripts can be created to automate tasks and execute commands sequentially. This project uses bash to run a script that checks every 5 seconds to see if the form data has been submitted and calls the main python handling script to run if so.



**Technology:** MariaDB

**Desc:** MariaDB is a commercially supported fork of the MySQL database system. It is free and open source under the GNU license. This project installs MariaDB on the VM’s that are configured to be database servers.

A blue and white cube with white text

Description automatically generated

**Technology:** VirtualBox

**Desc:** VirtualBox is an open-source hypervisor/virtualization software acquired and now developed by Oracle. It allows users to create virtual machines – running various operating systems and devices within a single host device. This project uses VirtualBox to run the virtual machines that the client wants to configure from a web interface.



**Technology:** Vagrant

**Desc:** Vagrant is a tool that is used to automate the provisioning of virtual machines in a hypervisor environment (such as VirtualBox). Users create a “vagrantfile” which pulls a specified OS from the web/local storage, allocate the proper hardware resources, configure the network options, and run various commands on startup. This project Uses vagrant to provision and start the basic virtual machines, which will then come online and be managed by ansible for further specialized configurations.



**Technology:** Debian

**Desc:** Debian is a open-source distribution of Linux focused on reliability. It serves as the upstream foundation for other various distributions such as ubuntu, kali Linux, Linux Mint, and more. It will be used in this project as the main operating system for both “Buildserver” and “VmBox”.

A blue and white logo

Description automatically generated

**Technology:** Alpine

**Desc:** Alpine is a Linux distribution that focuses on being extremely lightweight and security oriented. It offers high resource efficiency and a very small footprint, making it an excellent choice for this project as it will need to support a high volume of virtual machines on limited hardware resources. Alpine will be the main OS that is used on the VM’s, which will then be configured to be a webserver, database server, or custom server.

## *Technical Appendices*

**Machine Name:** Buildserver

**Operating System:** Debian 12.1.0

**IP Address:** 192.168.0.140 (DHCP)

**Network Adapters:**

* **lo:** (Loopback)
* **enp0s3:** (Bridged, 192.168.0.140/24)

**Virtual/Physical:** Virtual

* **CPU:** 2 cores
* **Memory:** 8GB
* **Drive Space:** 50GB (Dynamic)

**Roles:** Hosts UI on web stack, ansible control node, main processing/handling scripts.

**Packages installed:** nano, nginx, php, php-fpm, ansible, python3, openssh-server.

**VirtualBox Guest Additions:** Yes

**Logins:**

* root:adminpass
* trever:userpass

**Notable File Tree:**

A diagram of a computer

Description automatically generated

**Machine Name:** VmBox

**Operating System:** Debian 12.1.0

**IP Address:** 192.168.0.253 (Static)

**Network Adapters:**

* **lo:** (Loopback)
* **eno1:** (Wired, unused)
* **enp5s0:** (Wired, unused)
* **enp6s0f0:** (Wired, unused)
* **enp6s0f1:** (Wired, unused)
* **wlp1s0:** (Wireless, 192.168.0.253/24)

\*NOTE: These are all physical adapters, server has 3 NIC’s installed\*

**Virtual/Physical:** Physical

* **CPU:** 6 cores (Ryzen 5600g)
* **Memory:** 32GB (DDR4 3200Mhz)
* **Drive Space:** 500GB (M.2 NVMe SSD)

**Roles:** Target Device, Virtual Environment (VirtualBox), Receives and Builds Virtual Infrastructure

**Packages Installed:** nano, vagrant, virtualbox, virtualbox-ext-pack, python3, openssh-server.

**VirtualBox Guest Additions:** Yes

**Logins:**

* root:adminpass
* user:userpass

**Notable File Tree (generated):**

A screen shot of a computer

Description automatically generated

## *Configurations*

**\*\*\*All Project Files have been hosted on:** [**https://github.com/treverfulton/capstone**](https://github.com/treverfulton/capstone)**\*\*\***

These files are all hosted on the “Buildserver” Virtual machine.

***The following occurs on the “Buildserver” VM:***

1. The webpage is located at /var/www/html/index.html.
2. The JavaScript files used to reproduce form templates are render-custom.js and render-form.js in the /var/www/html/assets/js/ directory.
3. The webpage uses /var/www/html/submit.php to save the form information and files uploaded to /var/www/uploads.
4. /var/www/uploads owner has been changed to www-data (so PHP can write to it)
5. The service /etc/system/systemd/infra\_handler.service has been created to run the /etc/ansible/handler/start\_handler.sh bash script. This service has been started and enabled.
6. The start\_handler.sh bash script checks every 5 seconds to see if /var/www/uploads has files (json files for formdata and files uploaded) and runs /etc/ansible/handler/handler.py if so.
7. Handler.py output is redirected to console.log and errors are redirected to error.log in the /etc/ansible/handler/logs/ directory.
8. The handler.py script reads in infra\_variables.json and infra\_files.json and stores them as arrays.
9. It then uses the information to generate the /etc/ansible/handler/generated\_hosts.ini file.
10. Once the inventory file is created, the initialize-vagrant.yml playbook is run, which ensures that the current target environment exists, and copies over the “Buildservers” public key. **Playbooks can be viewed in /etc/ansible/playbooks/ for further information.**
11. The script Then iterates through the array made 2 steps ago and calls the build-vm.yml playbook to create the VM’s.
12. The script then goes through an if structure, checking to see what type of server is being created (web, db, custom) and runs the associating playbooks (webserver.yml, build-mariadb.yml, custom-install.yml).
13. The webserver.yml, build-mariadb.yml, and custom-install.yml are vastly different and take unique steps to build their own style of server.
14. If a server had files that need to be uploaded, these playbooks copy them over and unzip them into the correct directories.
15. Once They finish running, success output is displayed to the console (console.log in this case) and the script exits.
16. The start\_handler.sh bash script from step 6 continues since the handler.py script has exited, and /var/www/uploads gets cleared of all files.
17. start\_handler.sh then loops over, searching for files created in that directory to repeat the whole process.

***The following steps occur on the “VmBox” server:***

1. When Infrastructure is deployed by the build-vm.yml playbook to the “VmBox” server, the directory /etc/vagrant is created.
2. In /etc/vagrant, subdirectories will be created and named by the hostnames of the VM to be built. For example, you could see “web01, db01, custom-server”.
3. In these respective “web01, db01, custom-server” directories a vagrantfile and setup.sh bash script will be generated.
4. The vagrantfile handles basic provisioning, copies over the “Buildserver” public key to a the tmp folder, and runs setup.sh on the machine.
5. The setup.sh file handles user account creation, takes the public key in the tmp file and adds it to a newly created authorized keys file under /root/.ssh, and then adds some packages that are required by all VM’s
6. As per ansibles agentless requirements, Python must be installed on target machines, and ssh to root must be accessible through key. Unzip is for extracting user uploaded files in their target directories.
7. After this, the VM’s should be picked up by ansible and configured accordingly.

***This is an example of a typical vagrantfile generated:***

Vagrant.configure("2") do |config|

config.vm.box = "generic/alpine318"

config.vm.network "public\_network", type: "static", ip: "192.168.0.201", bridge: "wlp1s0"

config.vm.hostname = "web01"

config.vm.provider "virtualbox" do |vb|

vb.memory = "512"

vb.cpus = 1

end

config.vm.provision "file", source: "/etc/vagrant/setup/id\_rsa.pub", destination: "/tmp/ansible.pub"

config.vm.provision "shell", path: "/etc/vagrant/web01/setup.sh"

end

***This is an example of a typical setup.sh file generated:***

#!/bin/bash

echo "root:adminpass" | chpasswd

adduser -D -h /home/myuser -s /bin/bash myuser

echo "myuser:userpass" | chpasswd

mkdir /root/.ssh/

cp /tmp/ansible.pub /root/.ssh/authorized\_keys

apk update

apk add python3

apk add unzip

## *Conclusion*

**What goal(s) were you not able to achieve?**

While many of the basic requirements were achieved and expanded on, there were some goals that were initially envisioned but unfortunately could not be implemented. They are as follows:

* **Creation of OpenVPN server**

This was originally added as a modification to the proposal to add more depth to the project. The idea was that - like the other server types - ansible would run the OpenVPN install script and have the server be able to generate OpenVPN profiles based on user input. After re-evaluating, it was determined that implementing OpenVPN to a feasible functional state would take an immense amount of time for little benefit to the project, and ultimately was removed.

* **PXEBoot + Kickstart VmBox**

This feature was the idea of taking a blank bare metal server, using PXEBoot to install a bare os over the network, have kickstart make some basic changes so that it could be taken over by ansible, and then have ansible configure the Vagrant + Virtualbox Environment. It was never supposed to be apart of the main project goals, but rather a “nice-to-have” feature that could be implemented if everything else went smoothly and ahead of time (which it did not). Though it would have been a cool feature to implement, I did not find myself having enough time since I was focused on getting the main functions and processes of this project to work before the deadline.

* **Use Of Proxmox**

My original proposal suggested using the Proxmox virtual environment to help provision and host my Virtual Machines. After installing it on my host as a Virtual Machine, enabling nested virtualization, and experimenting in the environment, I found it to be a very extensive solution nearing the *too complex* side of things relative to the scope of this project. I tried bringing up a Alpine virtual machine, but it kept on throwing different kernel panic errors. A Debian image worked just fine too, telling me that proxmox was working properly. After swapping image versions, adjusting virtual hardware settings and properties, and various other troubleshooting methods found online, and spending approximately 10 hours on the issue, I determined that it wasn’t going to work, and I needed to move on.

After some more experimenting I managed to get the alpine *container* to work, but realized that a lot of the proxmox-ansible API configurations were container only and didn’t mention Virtual machines. This - along with wanting to stay true to the lightweight environment that comes with alpine - led me to start looking into other Virtual environment tools. I ended up choosing Vagrant + VirtualBox as I was very familiar with virtualbox, and Vagrant is just a provisioning tool that allows you to spin up virtualbox VM’s through “vagrantfiles”. This setup proved to be a lot more streamlined, being able to perform 100% of my required tasks without any bloat or overhead.

Overall, I feel like I went into this design with the idea of using proxmox and building the project around that, rather than designing the project with the correct tools for the job. As you may have seen through the documentation and project demonstration, Vagrant + Virtualbox has shown itself to be a very successful alternative and overall better choice for this project.

* **PostgreSQL**

The original proposal outlined use of the PostgreSQL database. This was a recommendation from a colleague, and chosen because of its enterprise level, and focus on stability and security. I got to a point where the playbooks were implemented, and the database server could smoothly be built and configured. When I went to choose a demonstration application to run ([TRS](https://github.com/profjamesmohawk/trs)) however, the database would fail to upload the provided SQL Files. After some further investigation, it was producing errors related to unsupported syntax and features. This was because the TRS application uses .SQL files that have code proprietary to MySQL/MariaDB. After learning about this, I decided to switch over to a MariaDB server as the main database platform, but still left the old PostgreSQL playbook in the playbooks directory for potential implementation in the future. I also implemented playbooks for creating database users, but they have been left unused for the time being due to project scope and permission concerns.

* **Use of Raspberry Pi**

The original proposal has a Raspberry Pi 4 as the host for Proxmox. This was quickly changed due to the limited hardware resources available on the Raspberry pi. With only 4gb of RAM it would be difficult to scale as running multiple virtual machines quickly get resource intensive. The Pi also runs on arm64 architecture which is not officially supported by Proxmox. PiMox, a community fork, was available, however at a glance it did not offer as much support and reliability as one would hope for. This ultimately led to the purchase of a bare metal server with 32gb of ram, a 64-bit architecture, 500gb of storage, and 6 cores.

**What was easier than you expected?**

* I found working with ansible to be a lot easier than I anticipated. While the playbook and its yml format seems daunting at first, it really becomes easy once you get used to working with it (and have a reference on hand). The integration of passing in environment variables and extra variables worked without issue the first time I tested it, required very little troubleshooting at any point later down the line
* Creating a web interface using Bootstrap Studio. I initially almost removed this technology from the plan as I believed it would be too difficult to make a nice page and wanted to get something working in bare html + CSS first. After not making up my mind and playing around in Bootstrap Studio, I was able to come up with a decent looking interface after not too long. In addition to that, implementing the proper JavaScript to reproduce templates with no prior knowledge of it was quick and painless, requiring minimal troubleshooting for it to work as intended. Overall, this tool really saved the look of my webpage, and encouraged me to add multiple and optional servers as a core functionality.

**What was harder than you expected?**

* Implementing a PHP script that would save the form data and files to the server was quite possible the largest hurdle to overcome in this capstone project. It was a combination of not planning/knowing what I was in for, and multiple errors and failure. First and foremost, I needed to learn what PHP was and how to write basic code with it. After learning the basics, I tried to apply it to my project which was okay at first but went extremely downhill. I found out after many hours of troubleshooting that the $\_FILES array iterates completely different to the $\_POST array. I then had to play with the permissions of folders and went through many different approaches to get PHP to store the files somewhere safe and accessible. I eventually opted to use /var/www/uploads and chown it to www-data. When trying to upload files to it I learned that any file over 8mb was failing. This led me to modify the nginx config and PHP configs to increase the max size, but it *still didn’t work.* After more debugging, I learned that I had to modify my php-fpm.ini rather than just php.ini and eventually things started rolling again.
* I also had the pleasure of more countless hours of troubleshooting PHP in the Alpine environment for my webserver VM option. For a reason I still don’t understand, PHP on alpine has multiple different naming schemes and will not work unless you are explicitly correct with how you use them. For example, to install the php package you need to specify a version. I opted for php 8.2 and php-fpm 8.2. Those only exist on the repo as php82 and php82-fpm. However, when configuring them on the server, you might find them referenced as php-fpm8.2, php8.2-fpm, php-fpm, and so on.
* Related to the previous point, I struggled with creating a functional nginx configuration file for the webserver alpine VM’s as I needed to enable php-fpm but could not find the right reference format. Eventually I figured out that I could substitute the php socket reference for 127.0.0.1 and the config ended up working as intended.
* Another issue I had to resolve was getting the python script to run ansible playbook commands. I looked into using the ansible API with python originally, however, was having troubles running the playbooks while inserting extra variables from a data structure. I ended up importing and using the subprocess module as a workaround, which essentially lets you run console commands from python. Since I was successfully able to run the extra variables manually in a test environment, I understood this method better, and was able to successfully implement it with a lot less troubles.
* Determining how I was going to activate my handler.py script proved to be harder than I initially thought. This was due to the identity errors associated with ansible and the allowed ssh keys on “VmBox”. If the script was called via php code it would be run by www-data user who does not have the correct permissions to control ansible and its hosts. I experimented with php calling running python scripts but concluded that the permissions needed to be extremely relaxed, and having a file in the /var/www/html folder that can execute a script with which requires high privilege was not a good idea. I then investigated creating a chron job to check if /var/www/uploads has new files but chron’s shortest time interval is 60 seconds. This led me to my final implementation of a service + startup script. The service is started and enabled, running this startup script in the background after boot. The startup script (/etc/ansible/handler/start\_handler.sh) checks in 5 second intervals to see if anything has been uploaded to /var/www/uploads, runs the handler.py, clears /var/www/uploads, and then goes back to checks every 5 seconds. I was most successful with this service + startup script method as it avoided the permission issued and automatically detected and started the python script without manual intervention.

**What limitations (if any) did you encounter by working in a virtual environment?**

* Working with the “Buildserver” in a virtual environment did not impose any limitations. Since the network adapter was bridged, it appears as a separate node on my network. If it were to be deployed to a bare metal server, I believe that the change would be seamless with no issues.
* In the early stages of the project, I tested Proxmox as a virtual machine. This introduced nested virtualization as proxmox would be the host to various virtual machines. Since the proxmox VM could only be allocated a portion of my host computers resources, each VM withing proxmox would only be allocated a fraction of that. This nested virtualization might have also been the cause of the kernel panics errors that I encountered when trying to get the alpine VM to boot up.

**What personal lessons did you learn in completing this project?**

The journey I have taken alongside this capstone project has allowed me to reflect on myself as a NESA student, a project coordinator, developer, and overall person. As the project unfolded, I was excited to work on it while simultaneously dreading it. After weeks of planning, the first step was the hardest, but I soon found myself sucked into it extremely large scope, attempting to implement and experiment with features as they popped into my head. Halfway through the semester I had not gotten much work done and the project was not getting any smaller in size. Theoretically the ideas were there, and certain parts were \*technically\* tested out, but it by no means was a working demo. I had run into multiple issues highlighted above and decided at this point I needed to re-evaluate my priorities and goals outlined in the proposal.

This is when I learned the lesson of choosing the tools before the job. I had Been so set on creating project using proxmox and bootstrap that I didn’t even stop and think if they were the exact tools I needed for the job. With so many solutions out there, the right tools do exist, and you just need to find them. So, I stopped trying to climb over this wall preventing my progress and found a way around with Vagrant + VirtualBox.

The next lesson I learned about myself was allocating time for mistakes. I initially had planned out the project to be developed straight forward with bonus features here and there. What I failed to account for was the steps that I didn’t know would be needed. My milestones were very broad in some areas due to lack of true understanding, and when it came time to implement a feature there would actually be 20 micro-steps needed for it to work, and I might have found my self stuck on 3 or 4 of them for quite some time. Looking back at the original timeline proposed, I estimate that I had exceeded the “unexpected troubleshooting” milestone by about >500% .

Lastly one on the positive side of things was the lesson of risk vs reward. I knew going into this project that I chose a more difficult option that would require a lot of effort and time to fully complete. Since this project essentially encompassed the whole course, a lot of my peers opted to do a safer and more controllable approach to capstone. I was tempted to follow suit, but a small part of me really wanted to make something that I could be proud of in the coming years. I wanted something that really shows off the skills learned throughout my time in the NESA program so that is exactly what I did.

The process was by no means easy, and there were many times that made me want to appeal or swap to something easier, but I knew that I would be disappointed with myself for accepting defeat. Looking back now, it was one of the best choices I could have made, and made me realize that I am capable of a lot more than I might value myself for.