Designing and Building a Virtual Cyber Security Range

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Emil Baggs

Spring, 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Yonghwi Kwon, Department of Computer Science

ABSTRACT

This paper outlines the research and development for a virtual cyber range platform to teach cyber skills to students in more real world environments. In order to help teach computer networking and security, my research focused on designing automation tools to help instructors and students create large dynamic virtual networks with little invested time. These automation tools use simple configuration files to know what a user wants to include on their lab network, then the tools build it. The primary goal of this project is to be able to create an immense variety of unique networks via building blocks of automation pipelined together. This streamlines the process of creating lab networks that can then be experimented on with no consequences, as the network can easily be rebuilt or replaced.

1. INTRODUCTION

Teaching and practicing certain computer science topics can be very difficult to do at a practical level, especially when those topics involve computer networks that expand beyond the scale of a typical student lab. Particularly in the real world, the way that devices on networks rely on each other in order to function (i.e. a web server depending on a database server) creates critical dependencies that often introduce single points of failure or cyber attack vectors. While small labs and courses can be great at teaching individual concepts, students rarely see the bigger picture of how a concept fits into a full network with all of these other critical dependencies. It isn't trivial, however, to create the labs for students to get hands-on with this "big picture" practice, since the resources it takes to build a network of computers can be significant, particularly with the amount of time it takes to set up.

As so much of computer science revolves around the idea of “never do the same exact thing twice”, the top of industry has by and large shifted away from the process of setting up computer networks by hand, and instead automating the creation of entire networks from start to finish, in what is known as cloud orchestration (Redhat, 2023). While existing enterprise solutions promise to make these automation techniques easier for everyone, these tools are still far from perfect and often don’t just work out of the box.Plus, these tools are not always designed for the breadth of networks that educators would ideally use to teach higher level technical skills, as they are designed for legitimate enterprise applications, where consistency and security is highly desired. In lab networks, however, students need variability and mistakes in the network to test their skills adapting to the environment.

In order to help address the difficulties with creating a highly flexible lab environment that doesn’t require significant cost, either in time or money, my capstone project focused on the creation of a computer server cluster with accompanying automation tools to dynamically spin up virtual lab environments for teaching cyber skills. I worked on this project with some help from members of the University of Virginia’s Computer Network Security (UVA CNS) club, as this project will be used and maintained by other students to help teach and practice cybersecurity skills. We have already been able to utilize the cluster to create several functioning lab networks, but hope to improve it in many ways with the hope that these automation tools can be more widely used. During the process of working on this cluster, I also deepened my understanding of particular aspects of computer networks and many specific in and outs of particular services and software.

2. CLUSTER DESIGN

Our computer cluster is made up of only a few physical machines stored in a server rack: 7 Dell Servers and 1 Cisco Switch, all of which were donated and show some signs of age, but are still plenty powerful for our applications. As we don't have a dedicated SAN that we could easily use, I installed Debian on one of the Dell servers, configured a 3TB RAID 5 array, and used NFS as a homebuilt shared storage solution (Jethva, 2022). We then decided to segment the remaining 6 servers into 2 logically different clusters, one that handles the dynamic environments built by automation tools and one that hosts permanent virtual machines for management (e.g. VPN server, web proxy, etc.). While this wasn't strictly necessary, it helps dedicate our more powerful servers towards the lab networks and organizes which machines are dynamic (i.e. handled by automation) and which ones aren't. Pictures of the physical rack are found in Appendix A.

In order to avoid paying for continual licensing fees, we opted to use Proxmox as the underlying hypervisor that would power the virtual machine cluster. While other hypervisor solutions are more popular, such as VMWare, Proxmox is extremely simple to get running, has a community edition that is free to use, and is starting to become more and more widely used (Joshua, 2021). Following Proxmox's instructions, we built a cluster using 3 of the servers and connected to the NFS shared storage (Proxmox, 2023). This ended up being our "Research" cluster that hosts all our more critical/permanent services, such as our primary firewall for the network and authentication servers. The other 3 servers became the "Range" cluster that has no critical content on it, but has plenty of power to build and run temporary networks for playing around with. As the scope of this report focuses on the Range cluster and the tools that power its automation, I will not be discussing in depth our usage of the Research cluster.

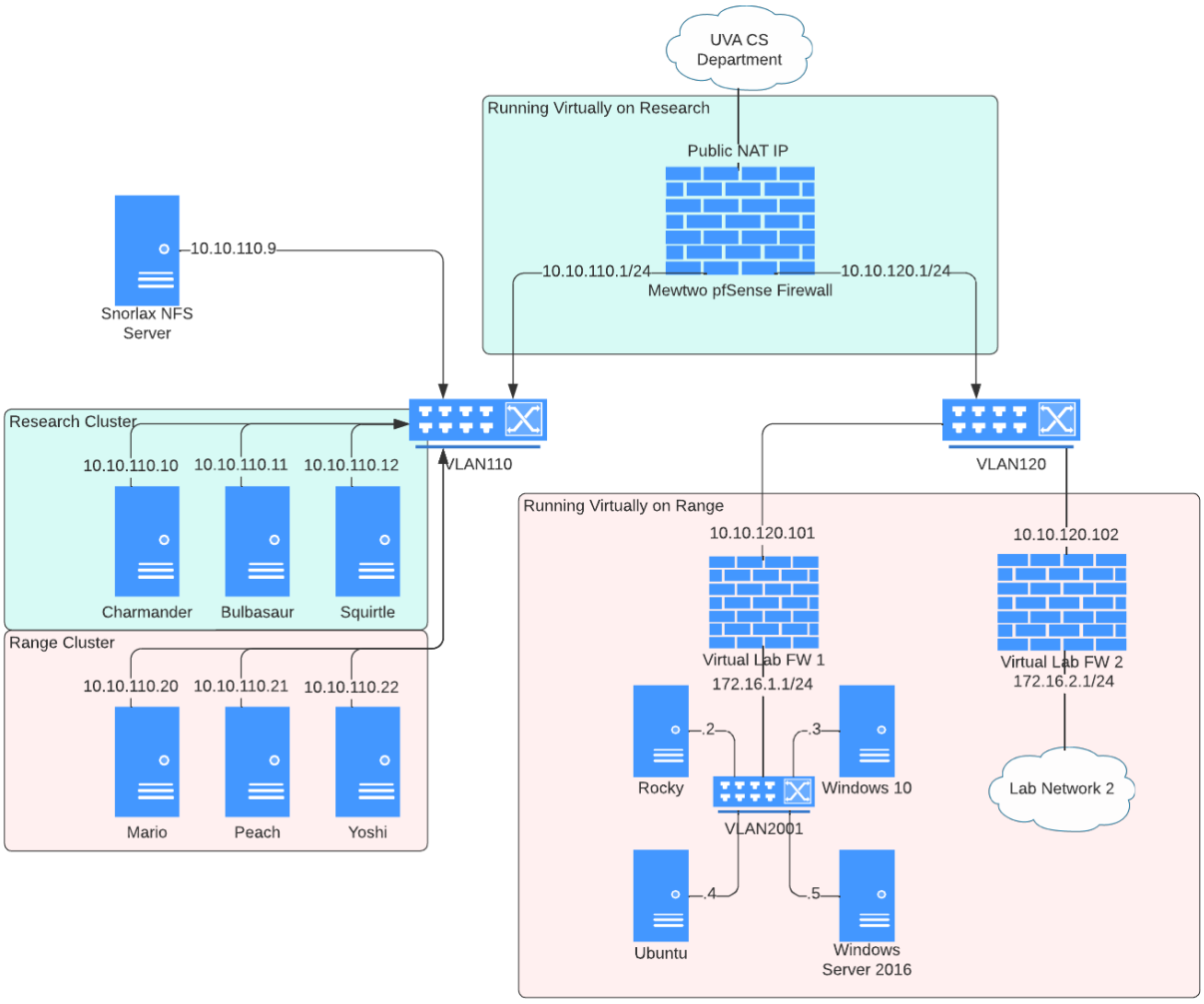


Figure 1. A logical network diagram of the cluster

Proxmox, especially when setup with a simple shared storage solution like NFS, is already extremely powerful. Within the Proxmox web UI, you can create and manage virtual machines without much practice, already giving anyone the tools to create their own virtual networks from a single interface. The process of fully designing and building the individual virtual machines (VMs) in a new network however can take significant amounts of time, even if now everything can be virtually simulated on the same hardware. Proxmox includes many features that speed up this process, such as the ability to easily clone vms or revert them to a

backed up snapshot, but these features still need something providing the logic to know how to assemble a network. And with shared storage enabled, VMs can migrate between physical servers in seconds.

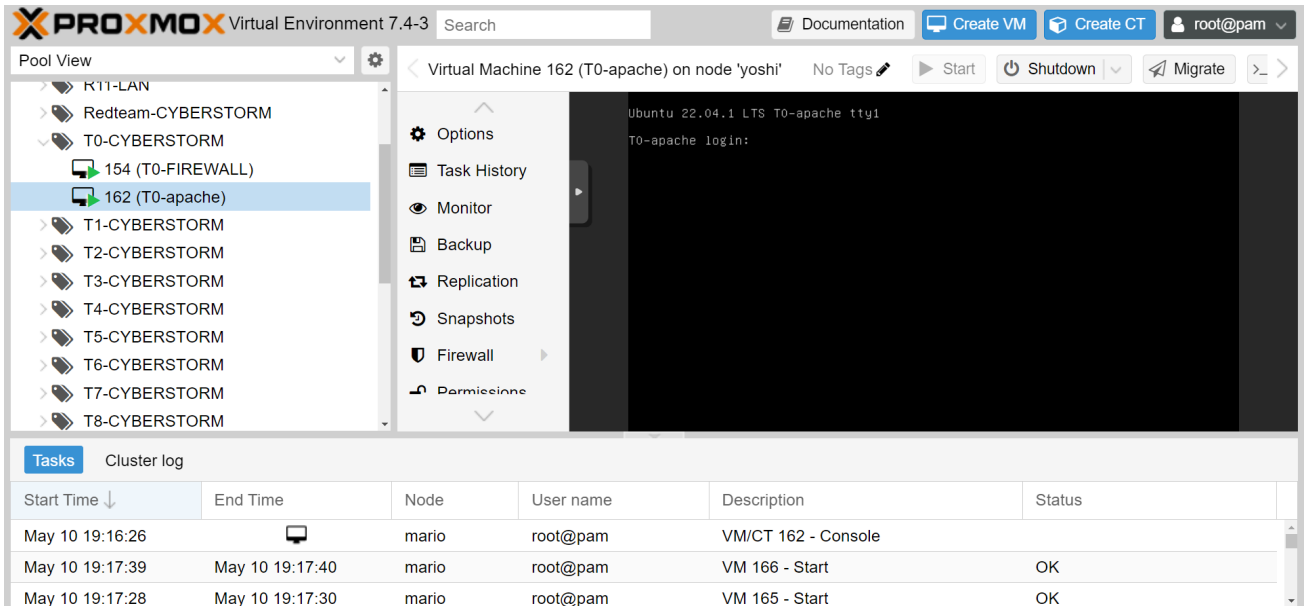


Figure 2. Image of Proxmox Web UI

3. FLOW OF AUTOMATION

Rather than a human operator providing this logic, we use automation tools that plug into Proxmox's API to build and destroy networks based on a minimalistic configuration file that describes what the network should look like in its finished state. For a simple example, we could specify in our configuration file that we want a network with a Windows 10 VM running an IIS web server, a Windows 2016 Server VM running Active Directory, and a Fedora 34 server running MySQL, and the tools should just built out the network based on those specifications. I broke this automation process into three broad steps: first every VM needs to be built and powered on, second each VM needs to get connected to the network, and third automation tools

need to connect to each VM individually to configure specific settings and services to build the network into its finalized state.

In order to solve the first step in the automation process, the team at HashiCorp designed an open source deployment tool called Terraform (Harrington, 2023). Terraform is designed to work with a variety of virtual providers, such as AWS or VMware, but supports custom modules including a Proxmox module maintained by Telmate. The Proxmox provider module enables Terraform to hook into the Proxmox API in order to automatically clone, start, or modify VMs as needed in order to ensure that all of the machines were built to properly fit the specifications of the configuration file. In order to significantly speed up the creation process of all of these VMs, rather than install an Operating System (OS) fresh on a VM everytime we want a new one, the cluster instead has a collection of template VMs that contain clean installations of any OS we need that Terraform can quickly copy and boot up. Terraform also setups up all of the hardware configuration for the VMs it creates, allowing the automation tools to adjust cores, memory, what Virtual LAN (VLAN) they are connected to, etc. That way, once the VM boots up, it has all of the settings set in Proxmox to be ready to connect to the network.

Once a VM boots up, since it was cloned from a template, it is in a completely generic state. While the VM's network adapter has already been attached to the appropriate VLAN in Proxmox, the VM still needs to have a unique IP address on the network in order for it to talk to other machines. On top of that, each VM's unique IP address should be deterministic, as that IP address is how services on the network will depend on each other. Luckily, another open source tool, Cloud Init, is integrated right into Proxmox and allows you to configure basic options for the VM to configure when it boots up, such as a static IP address and default account (RedHat, n.d.). By setting these options in the Cloud Init configuration, Proxmox passes the options into

the VM by storing them in a file on a virtual disk drive. Then, when the VM boots into its OS, the template will run the preinstalled Cloud Init software to read the configuration options and apply them. These configuration options set up the VM with everything needed to remotely connect and manage the system over the network using WinRM for Windows templates and SSH for anything else.

Once every VM is up and running and has remote management configured, the third step of the automation process can run and perform the in depth configuration of every machine, which mostly consists of installing software and managing user accounts, but can realistically do pretty much anything. This step is implemented using another widely used open source tool, Ansible, which connects to machines remotely over the network and uses built in modules to perform various actions on the system based on logic outlined in "playbooks" (Avi, 2022). Based on a configuration file that mirrors the information fed into Terraform (see Appendix B for an example), Ansible knows which machines to connect to and what plays it should run in order to install and configure the desired services across the entire network. Ansible's strength is managing many machines on the network at once, making sure that each VM is configured for whatever individual role it needs while also knowing the complete picture of the network. This allows Ansible to automate configuring individual components in the network but then also create dependencies between these components in a way that mimics real world networks, all based on a single configuration file.

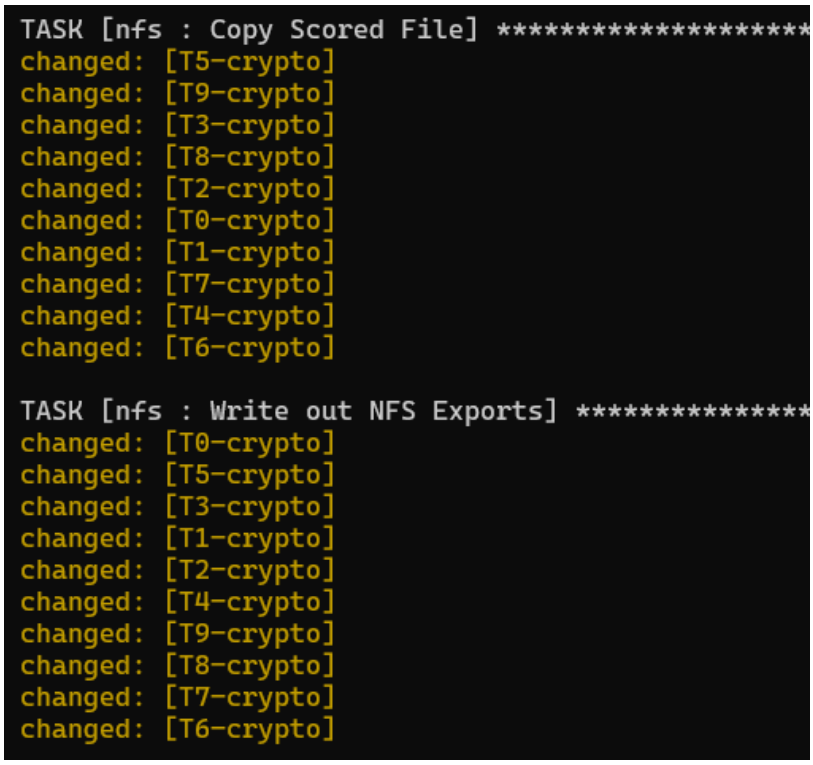


Figure 3. A screenshot of Ansible running the third stage of automation

This specific design flow for the cluster gives our tools an extremely modular design that makes it easy to adjust in the long term while also ensuring that these same strategies will work for larger scale networks. In order to use the tools to build a new network, it is just a matter of picking and choosing which modules to combine and as long as two modules aren't incompatible with each other (i.e. trying to install Windows software on Ubuntu), the modules should all combine to form a single big picture. This modular design also encourages collaboration on the project, as individual contributors can focus on simple smaller modules to fix issues or expand capabilities.

4. CUSTOM TOOLS

Especially when using services that have dependencies on another module, like a web server depending on a database server, I had to ensure that Ansible performs the right order of

operations for the installation to work. Modules that other modules depend on, such as database or authentication servers, are prioritized in the setup processes by our Ansible tools. After those initial services are functional, later modules can run and establish dependencies, such as uploading initial data in a database for an ecommerce site to use.

A large portion of this project was focused on building out our library of VM templates and Ansible roles once we had the first stage of the automation process working. In order to ensure that the second stage of the automation process works, every VM template needs to be built and tested to ensure that it properly runs Cloud Init after it boots, then sets up the network configuration and a user account for remote management to work. In some cases, we had to write scripts ourselves to create a usable template. In order to increase the variability of networks our tools can deploy, we created templates for all of the following Operating Systems:

- pfSense*
- Ubuntu Server Bionic, Focal, and Jammy
- Ubuntu Desktop Jammy
- Debian Stretch, Buster and Bullseye
- Kali
- OpenSuse 15.4
- Centos 7
- Rocky 8
- Fedora 34
- Alpine 3.13, 3.14, 3.15, 3.16
- FreeBSD 12

- Windows Server 2012*
- Windows Server 2012 R2*
- Windows Server 2012 R2 Core*
- Windows Server 2016*
- Windows Server 2016 Core*
- Windows Server 2019*
- Windows Server 2022*
- Windows 7*
- Windows 10*

** Indicates template uses custom Cloud Init scripts*

Aside from variation in the Operating Systems, we also need a large collection of potential Ansible roles that we can apply to VMs to configure them in some way. Many of these roles are designed to set up a particular service, such as installing and configuring a MySQL server, but a role can also do things like create user accounts or even deploy malicious backdoors into the lab environment (Avi, 2022). When we design the network, we can assign these roles out to particular VMs to pick and choose how we want everything to behave. To make our tools as diverse as possible, we wrote many custom Ansible roles and tested them for compatibility with different OS templates. While not every role is compatible with every OS, this is a list of services Ansible can currently build on the network via custom roles:

- Bind9
- Docker Services

    - Elasticsearch
    - Graylog
    - Postgres
    - DroneCI + Gitea Pipeline
- FTP
- Kubernetes Controller
- Kubernetes Worker
- MySQL
- NFS
- NFS Client
- Zencart Ecommerce Site
- Windows Domain Controller
- Windows Domain Join (joins VM to a Windows Domain)
- Windows Hmail
- Windows Internet Information Services
- Windows Server Message Block
- Windows Remote Desktop Protocol

While we hope to continue to grow the lists to choose from when building a lab network, already these lists have demonstrated a considerable amount of potential for creating diverse and complex networks.

5. MAJOR CHALLENGES

Throughout the process of getting the cluster and tools to the state they are in today, plenty of challenges have made it more difficult to implement the overall design of the project. So far, however, the two most significant challenges to this project have been creating templates for Windows based virtual machines and working around issues in the Terraform Proxmox provider.

5.1 Windows Cloud Init

During the second stage of the automation process where each VM actually connects itself to the network utilizing Cloud Init, each VM template needs to be already built to check its configuration on boot and set itself up. While many Unix based systems make it trivial to install and set up Cloud Init to run on boot in order to create a template, we found that the solutions out there for Windows didn't work out of the box as well as we needed them to (Geco-iT, 2022). So in order for Windows VMs to automatically connect to the network, a Windows specialist in UVA CNS helped me write a Powershell script that executes on boot and performs the initial setup steps, notably connecting the VM to the network.

Unfortunately, the Windows Operating System isn't designed to work when cloned from a template the way that Terraform does when initially creating all of the VMs, unlike the Unix based systems we tried. While most of the functionality within Windows worked, each VM cloned from the same template had the same Security ID set, which doesn't update when the hostname or IP address of the machine changes. When trying to configure Windows Domain Services, these conflicting Security IDs broke the ability for Windows machines to join the domain setup by the domain controller (Stanisic, 2017). As domain services are critical to pretty

much any enterprise network, in order to effectively mimic real world networks, our tools needed to address this issue.

To add to the challenge, the only way within Windows to update the Security ID for a machine is to run Sysprep.exe, which effectively wipes most of the machine and causes it to reboot, wiping any of the network settings or users we set up to connect remotely (Stanisic, 2017). After analyzing the Sysprep utility in depth, however, we found Sysprep could be configured to invoke a command after it boots into the reset environment. Using that option, we were able to call the script that sets up remote access for WinRM after the reset instead of before. So, when Terraform clones a Window VM template into a new dynamic network, that VM has to boot up, run Sysprep to reset and reboot, then once Sysprep is done, it calls a Powershell script that reads the Cloud Init configuration passed by Proxmox, and configures its network devices accordingly.

Overall, this method works well but still leaves plenty to be desired, as the scripts we use are fairly simple and only account for our limited use case. These scripts also don't follow any of the structure of any other widely used Cloud Init port, meaning it likely won't gain any traction beyond our own internal use case. One popular port, Cloudbase Init appears to have the most traction as a more full featured Cloud Init tool for Windows (Tencent Cloud, 2023), but we faced issues using it in our early testing. In the future, we'll ideally debug those issues and switch to Cloudbase Init as it is more widely used. Right now, however, the current process works perfectly for most versions of Windows and isn't a priority to update.

5.2 Issues with the Terraform Proxmox Provider

While Terraform is an extremely widely used tool for automation engineers everywhere to dynamically create and maintain large networks, the vast majority of people use it with cloud providers like Azure or with locally hosted VMware clusters (Fee, 2023). While Telmate has created a module for Terraform to support the Proxmox API so that Terraform can work in the exact same way as with any other provider, working on this project showed that Terraform plus Proxmox compatibility still has a long way to go. Unlike other provider modules, Terraform's Proxmox provider hasn't been widely developed enough yet for it to fully compare to using Terraform with VMware. Still, Proxmox has only recently started gaining more popularity, so it is possible that these modules will improve more over time. Even during our time developing our initial custom Terraform modules, a major update was pushed that significantly improved our ability to work with Terraform, although we still see plenty of issues with it.

Prior to the update, the most infuriating aspect of Terraform's Proxmox module is that it had no error reporting. Even something as simple as misnaming a template in our configuration could take hours to debug, slowing down development of the Terraform modules we needed for the first stage of the automation process. Luckily, an update to the module introduced better error codes, but we still have noticed significant ongoing issues.

Perhaps the most glaring issue is that Terraform frequently fails when deploying a network, giving various generic error codes, but never consistently. Just running the same Terraform code multiple times, it might fail two or three times for different reasons and then finally work when you run it again. This means that we can utilize our Terraform tools, but it takes multiple attempts, each of which can last several minutes. Even then, there are uncommon cases where Terraform corrupts the state of a VM when deploying, meaning we have to manually delete the VM or a file in order for Terraform to work again. On top of that, even when it does

work, the speed at which Terraform deploys networks seems significantly slower than it should be. At times, Terraform will spend over 10 minutes creating a single VM, while doing it through the GUI only takes about 30 seconds.

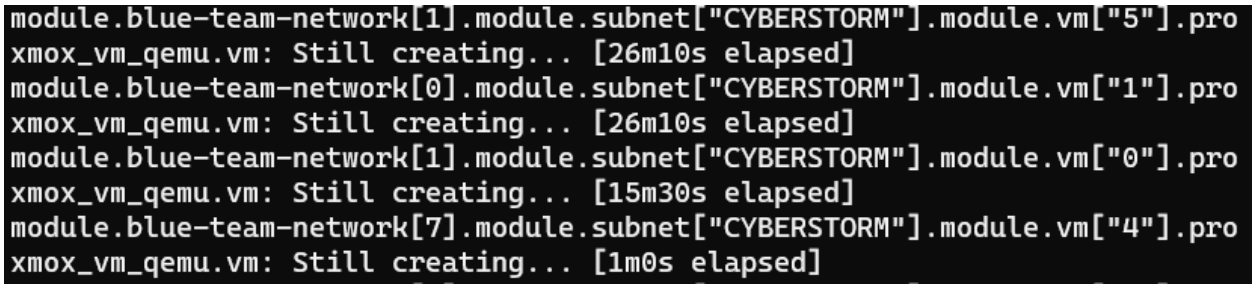


Figure 4. Screenshot of Terraform still creating the same 2 VMs after 26 minutes

While Terraform still is quicker for large networks, especially as it deploys VMs in parallel, this inefficiency appears to be in the Proxmox Terraform module, as using the API directly doesn't yield these issues. For comparison, it takes anywhere from 20-40 minutes for Terraform to deploy a network of 40 VMs (especially when it needs to run more than once), while a simple bash script (Appendix C) is able to interface directly with Proxmox and build the same size network in less than a minute. While this script was just a proof of concept to identify whether Terraform was the bottleneck or not, this speedup is very significant. As such, a goal in the future is to remove Terraform from the picture and write our own tools for the first stage of network automation. Despite Terraform's issues, it does a great job of checking the state of an existing network and applying necessary changes to it, making it great for enterprise applications where the network is constantly evolving (Hashicorp, n.d.). But, as our goal is to create and destroy dynamic lab environments, this feature isn't nearly as critical. If we ever need to modify the underlying VMs, we can just as easily destroy and rebuild the entire network from scratch

using the automation tools. This, coupled with the other issues Terraform has with Proxmox, means writing a replacement tool for Terraform makes a lot of sense going forward in order to optimize the tools for our purposes..

6. EXAMPLES OF USAGE

Throughout development of the cluster and the tools to automate it, many opportunities came up to test how effectively these tools could be leveraged to build networks for students to practice with. During Fall 2022, during the very early stages of work on the cluster, the UVA CNS club hosted a Red vs. Blue style cyber defense competition hosted on the cluster and built with the help of the automation tools. As a lot of the automation was not working yet, a large portion of the network still had to be built by hand, including all of the Windows machines. But with 13 teams and 6 VMs per team, a lot of tedious heavy lifting was offloaded by the automation tools.

This first real use case for the automation tools also showcased a number of areas to prioritize improving the cluster. While the first was certainly prioritizing automating Windows deployments, we also needed to improve parts of the Terraform modules to support building more complex networks. On top of that, while the Ansible automation roles had worked to set up services for the competition, there was very little variety in the networks they could build, so we needed to expand our library of services that could deploy with Ansible.

Once we had support for Windows templates and more Ansible roles, the tool could be used to create a much larger variety of networks. In order to test the tool and help train their cyber security skills, the UVA Cyber Defense team practiced in lab environments built specifically to see how well they could adapt in various environments. For one practice, the team

ran a mock competition where they defended a network with over 30 VMs running from other students attacking their network. In another practice, the team practiced setting up and deploying centralized monitoring services across a large network to quickly get full visibility into a lab environment they'd never seen before. Prior to these automation tools, the team would primarily practice in static environments they built themselves, but couldn't easily practice responding to new environments.

Towards the end of the Spring 2023 semester, UVA CNS hosted another Red vs. Blue competition for students interested in getting into defensive security. As with the first competition run, everything was hosted and built using the cluster, but this time automation tools did all of the heavy lifting. Within 4 hours, I used the automation tools to build the entire competition network from start to finish. Each team had a network of 7 VMs, so I designed the network for one team, then used the tools to deploy that same network for all 10 teams. Figure 5 shows the automated scorebot that checks the status of every service throughout the competition with everything currently functional. Aside from Terraform working inconsistently, the network deployment went smoothly and these tools made it possible for one person to create a network in a small fraction of the time it would take a team of people by hand.

	Team0	Team1	Team2	Team3	Team4	Team5	Team6	Team7	Team8	Team9
Current Score	1,050	1,050	1,050	1,050	1,050	1,050	1,050	1,050	1,050	1,050
Current Place	1	1	1	1	1	1	1	1	1	1
Up/Down Ratio	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼	7 ▲ / 0 ▼
FTP-remote	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
IIS-iis	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
LDAP-ad	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
NFS-crypto	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
SSH-exec	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
SSH-remote	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
ZenCart-apache	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Figure 5. The scoreboard for the competition showing 10 functioning networks

7. FUTURE GOALS

While the cluster and automation tools have already made it possible to quickly deploy new practice networks for students at a scale and rate that otherwise wasn't possible, there are still many ways this project can be expanded and improved. For starters, many of the VM templates the tool uses work, but could be improved to become more standardized (i.e. all have the same software installed by default). Packer is another open source tool maintained by Hashicorp that is designed for creating automation VM templates (Hashicorp, n.d.). Eventually we hope to look more into using Packer to quickly create lots of standardized VM templates, but this hasn't been prioritized. .

As mentioned before, eventually the goal is to move off of Terraform and write our own custom tool for creating and destroying VMs in Proxmox. In addition to hopefully improving speed and reliability, we can also use this tool to expand the capabilities of the networks it deploys. For example, one current limitation of the tool is that a deployed network firewall can only have two network interfaces, one for the WAN and one for the LAN. By improving the tools, though, firewalls should be able to support several internal network interfaces, further expanding the variety of networks possible to create.

In addition to expanding the capabilities of the automation tools, we also hope to develop a more accessible interface for designing lab networks. Right now there is very little intuitive instruction on how to actually utilize the tools, especially if everything doesn't work perfectly. While we are continuing to expand documentation, our goal is also to design a front end web application to make it easy for those without detailed knowledge of the tool to still create practice lab networks to play in. While this front end is essentially a whole new project in and of

itself, this idea would transform the accessibility of the project that already exists into a tool that more students can widely use to perform their own research.

In addition to designing a front end to interface with the tool, we also hope to include a "Random Network Generator" utility that builds a random network that is still fully operational based on some basic parameters, like how to generate the network. This would even further streamline the ability for students to practice throwing themselves into unknown environments without needing to invest large amounts of time designing those environments. Especially for students looking to perfect their skills in highly complex environments, a tool like this can present challenges that otherwise would take hours or days to design.

Finally, another major goal with this project going forward is to continue to expand the library of VM templates and services to choose from when building a network. While the tools already have a lot of variety in what they can build, there is still a huge back catalog of new things we want to be able to automatically deploy onto a network. For example, we currently have automation in place to deploy pfSense firewalls for each network, but if we want to utilize another firewall vendor, such as a Palo Alto VM, we need to manually clone a template and set it up, while hopefully this can be automated in the future.

Already the tools in place demonstrate a large potential for creating training environments that better mimic real world networks than other lab solutions. While there are still certainly plenty of ways to expand this project to make it even more useful, the foundational blocks are in place for this project to function. While some students have already been able to get hands-on with the lab environments hosted in the cluster, hopefully the progression of these tools will make it easier for even more students to learn networking and security skills.

ACKNOWLEDGEMENTS

I thank the University of Virginia Department of Computer Science for the donations in hardware and internet connectivity used to run the cluster. I also thank the UVA CNS club for their help in designing and testing many of the individual components within the cluster. I especially thank Chase Hildebrand and Grant Matteo for their individual contributions to this project and their commitment to helping teach computer science.

References

Avi. (2022, November 30). *Ansible for Beginners - Ansible Basics and How it Works.* Geekflare Retrieved 2023, from https://geekflare.com/ansible-basics/

Fee, R. (2023, March 14). *The Top 20 Terraform Providers.* Scalr. Retrieved 2023, from https://www.scalr.com/blog/top-20-terraform-providers

Geco-iT. (2022, January 19). *HOWTO: Scripts to make cloudbase work like cloudinit for your windows based instances*. Proxmox Forums. Retrieved 2023, from https://forum.proxmox.com/threads/howto-scripts-to-make-cloudbase-work-like-cloudinit-for-your-windows-based-instances.103375/

Hashicorp. (n.d.). *Purpose of Terraform State*. Hashicorp. Retrieved 2023, from https://developer.hashicorp.com/terraform/language/state/purpose

Hashicorp. (n.d.). *Why Use Packer?*. Hashicorp. Retrieved 2023, from https://developer.hashicorp.com/packer/docs/intro/why

Harrington, D. (2023, April 6). *What is Terraform: Everything You Need to Know*. Varonis. Retrieved 2023, from https://www.varonis.com/blog/what-is-terraform

Jethva, H. (2022, March 24). *How to Install and Configure NFS Server on Debian 11*. Atlantic.Net Blog. Retrieved 2023, from https://www.atlantic.net/dedicated-server-hosting/how-to-install-and-configure-nfs-server-on-debian-11/

Joshua, S. (2021, September 24). *What is Proxmox, and why is it awesome?*. Medium. Retrieved 2023, from https://medium.com/@stephendjoshua/what-is-proxmox-and-why-is-it-awesome-f16ef5a57ca0

RedHat. (n.d.) *Configuring and managing cloud-init for RHEL 8.* RedHat. Retrieved 2023, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_cloud-init_for_rhel_8/index

RedHat. (2023, February 20). *What is cloud orchestration?*. RedHat. Retrieved 2023, from https://www.redhat.com/en/topics/automation/what-is-cloud-orchestration

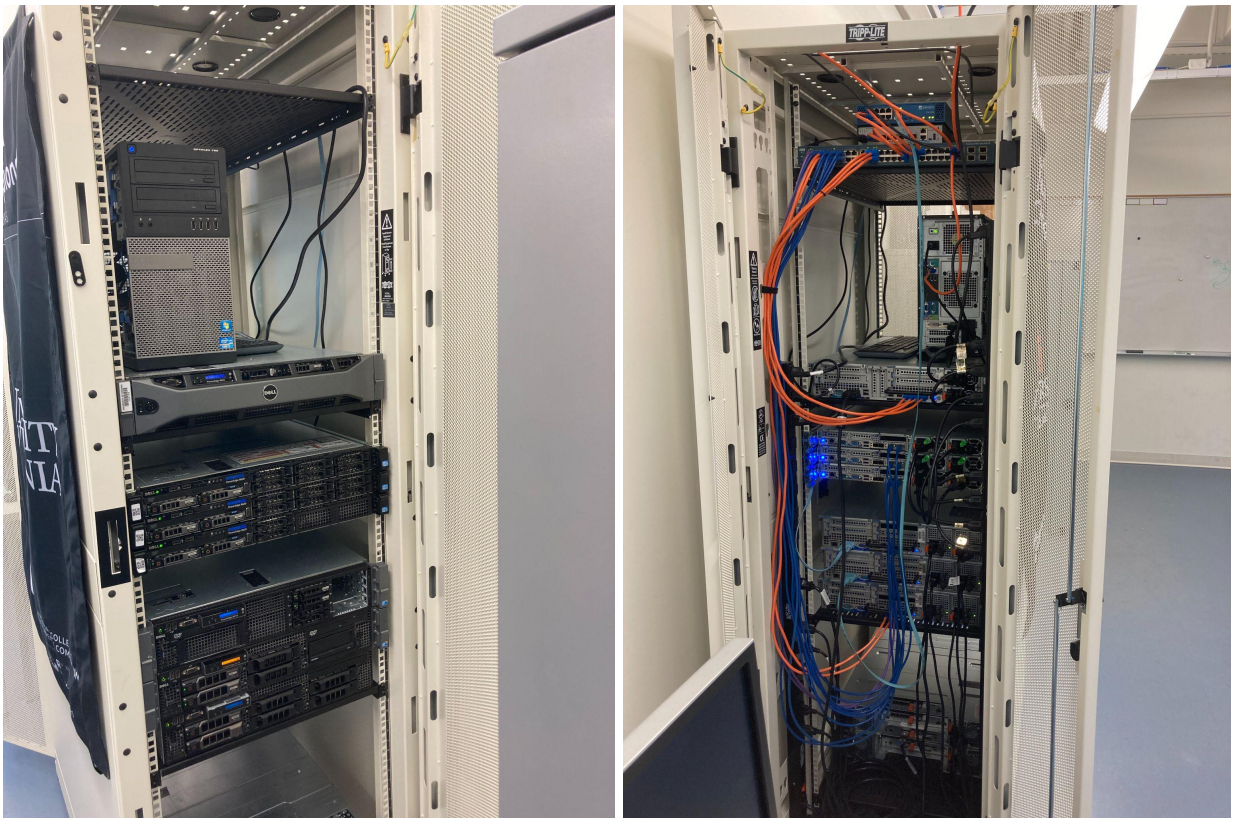
Proxmox. (2023, March 22). *Cluster Manager*. Proxmox. Retrieved 2023, from https://pve.proxmox.com/pve-docs/chapter-pvecm.html

Stanisic, S. (2017, June 29). *Changing SID of cloned VMs*. Mivilisnet. Retrieved 2023, from https://mivilisnet.wordpress.com/2017/06/29/changing-sid-of-cloned-vms/

Tencent Cloud. (2023, May 8). *Cloud-Init & Cloudbase-Init*. Tencent Cloud. Retrieved 2023, from https://www.tencentcloud.com/document/product/213/19670

APPENDIX

A. Front and back images of the physical server rack holding the cluster



B. Sample Ansible inventory.yaml configuration for randomized lab network

---

```
all:
  vars:
    default_users:
    - bzorzoni
    - cmougel
    - bgrelak
    default_admins:
    - bzorzoni
    default_password: Chiapet1
  children:
    scoreboard:
      hosts:
        r11:
```

```
      ansible_host: 10.10.110.90
      teams: 1
      scoreboard_lan: 11
nix:
  hosts:
    LIN-WORK0:
      template: ubuntudesktop-jammy
      ansible_host: 172.16.11.2
    LIN-WORK1:
      template: ubuntudesktop-jammy
      ansible_host: 172.16.11.3
    LIN-WORK2:
      template: ubuntudesktop-jammy
      ansible_host: 172.16.11.4
    LIN-WORK3:
      template: ubuntudesktop-jammy
      ansible_host: 172.16.11.5
    VERMONT:
      template: ubuntuserver-jammy
      ansible_host: 172.16.11.6
    ALASKA:
      template: ubuntuserver-focal
      ansible_host: 172.16.11.7
    IDAHO:
      template: opensuse-15.4
      ansible_host: 172.16.11.8
    TEXAS:
      template: ubuntuserver-focal
      ansible_host: 172.16.11.9
deb:
  hosts:
    LIN-WORK0: null
    LIN-WORK1: null
    LIN-WORK2: null
    LIN-WORK3: null
    VERMONT: null
    ALASKA: null
    TEXAS: null
ssh:
  hosts:
    IDAHO: null
mysql:
  hosts:
    TEXAS: null
zencart:
  vars:
    zencart_db_server: 172.16.11.9
```

```
      hosts:
        ALASKA:
          zencart_db: storage
    bind9:
      hosts:
        VERMONT: null
    nfs:
      vars:
        nfs_export_path: /data/
      hosts:
        VERMONT: null
    nfs-client:
      vars:
        nfs_server: 172.16.11.6
      hosts:
        IDAHO:
          local_mount: /data
          nfs_share: /data/IDAHO
    docker:
      children:
        droneci-gitea:
          hosts:
            TEXAS: null
```

C. create.sh BASH script for deploying several VMs quickly

```
#!/bin/bash
set -eu

file="vms_`date -u +%Y-%m-%dT%H:%M:%S`.txt"
create() {
    #vmid=$(pvesh get /cluster/nextid)
    vmid=$1
    qm clone 1000 $vmid
    qm start $vmid
    echo $vmid >> $file
}

for i in {10000..10020}; do
    create $i &
Done
```