

Hello,

My name is Jacob Small and I am a recent graduate from the Rice University Cybersecurity Bootcamp. I am a former musician, educator, and business owner who saw the push to online virtual learning as a wonderful opportunity to advance my career and gain a new skill set.

What I am sharing with you today is an automated cloud deployment of both a vulnerable web application and a security information and event management system (or SIEM for short). The purpose of this project is to become familiar with the Microsoft Azure Cloud Environment and to build a sandbox of sorts for penetration testers to explore the various tactics, techniques, and procedures (TTPs) used by malicious actors. The inclusion of the SIEM allows Security Operations Analysts to track the malicious actor's movements and experiment with alarms and thresholds for further detection.

Why the Cloud?

Benefits:

- Ground-up Security
- Easy Configuration
- Quick Turnaround
- High Availability/Fault Tolerance
- Easy Implementation
- Affordability

Downsides:

- Complex Architecture
- Extensive Management
 - *Ex: VM Sprawl*
- Different Threats

So, why are companies so interested in the cloud?

- The cloud presents an opportunity to build a secure system from the ground-up, as opposed to trying to implement new security measures on old systems.
- It's easily configurable and allows network administrators to create all their needed items within the cloud service provider's website portal.
- Compromised and insecure machines can be discarded and replaced quickly, and at no additional cost to the organization.
- Without their own physical data centers, engineers can deploy their machines in multiple places allowing for networks to be more robust against power outages, DoS attacks and other threats.
- Security controls can be implemented quickly and the security profiles can be copied to any existing or new projects.
- And finally it is, of course, affordable.** Machines can be created on a need to use basis which eliminates the need to manage on-premises equipment.

But to be fair, it also has its downsides:

- It's typically easier to implement basic security measures with on-premises

machines since they're not exposed to public networks unless they need to be.

- Flexibility is one of the main attractions to the cloud, but without adequate change management procedures, a virtual network can quickly become too big, too complex, or overlooked.

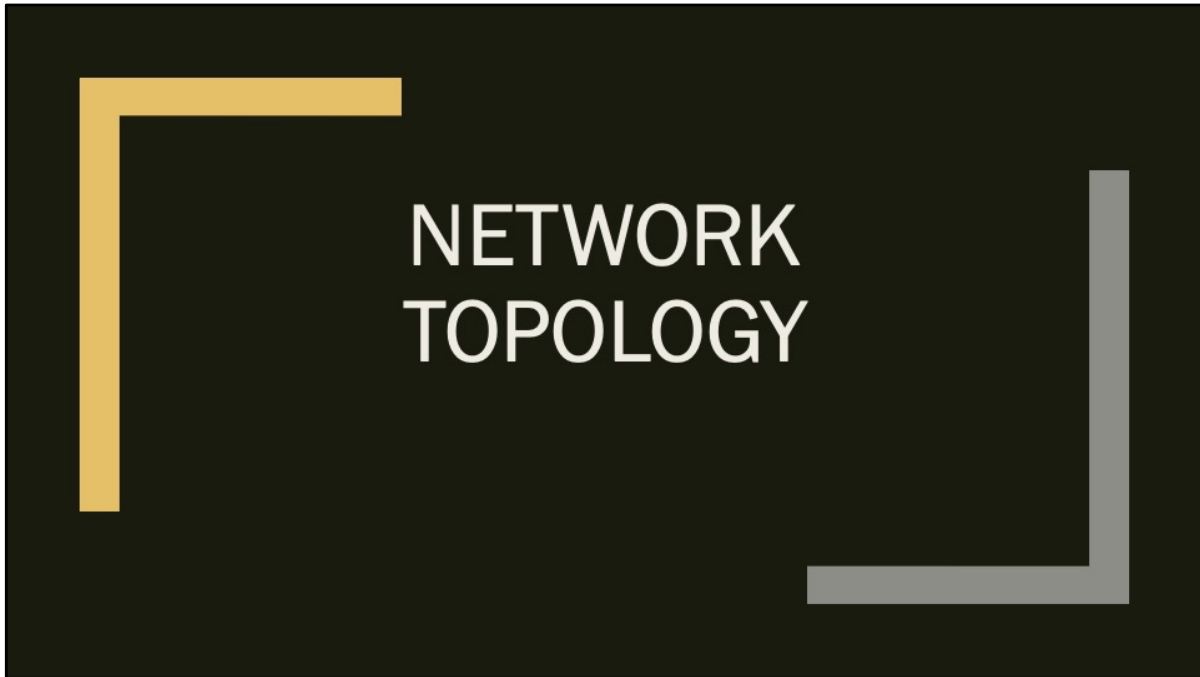
- Threats in the cloud are uniquely different from those found in on-premises equipment and security professionals must be trained to look for these cloud-based exploits.

Which Cloud Service?

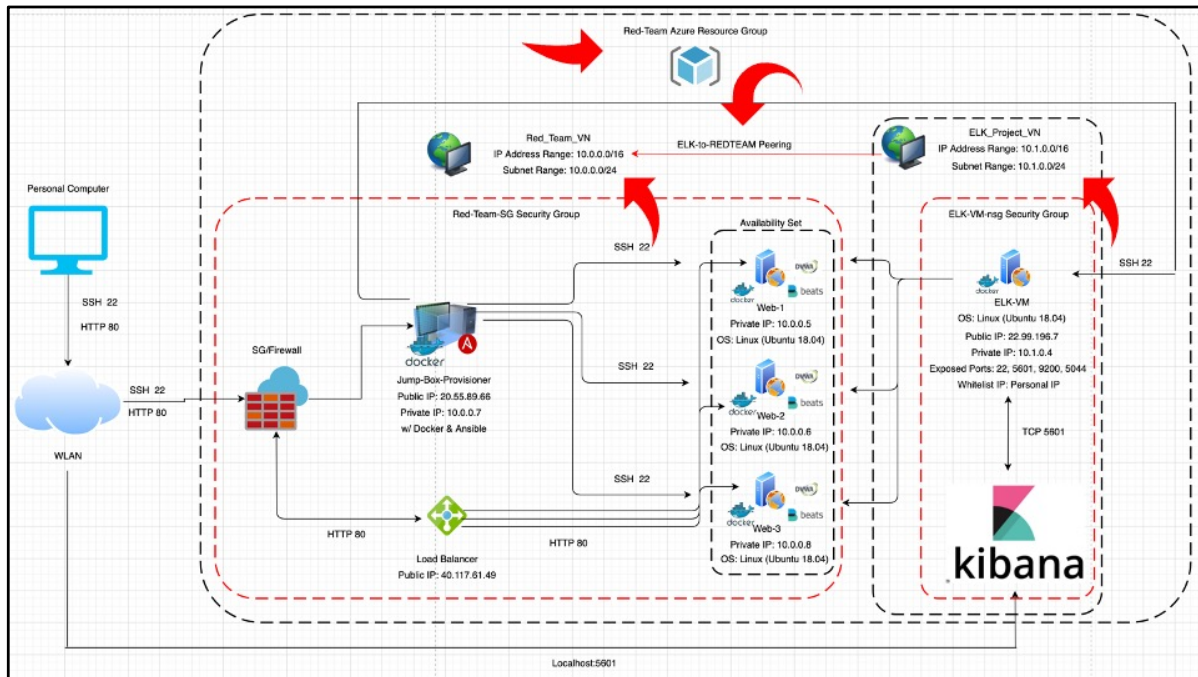
- PaaS – Platform as a Service
 - *A cloud-based environment in which users can build and deliver applications*
- SaaS – Software as a Service
 - *Software and Applications provided through the Internet (Ex: Google Docs)*
- DaaS – Data as a Service
 - *A service that provides a company's data product to the user on demand*
- CaaS – Communications as a Service
 - *An outsourced communication solution (Ex: Voice over IP)*
- IaaS – Infrastructure as a Service
 - *Cloud-based access to storage, networking, servers, and computers*

Here are a few examples of the various cloud services in used today.

The following project can be characterized more specifically as Infrastructure as a Service (IaaS). This is when a service provider offers pay-as-you-go access to storage, networking, servers and other computing resources in the cloud. Its up to your network engineers to build these systems out.

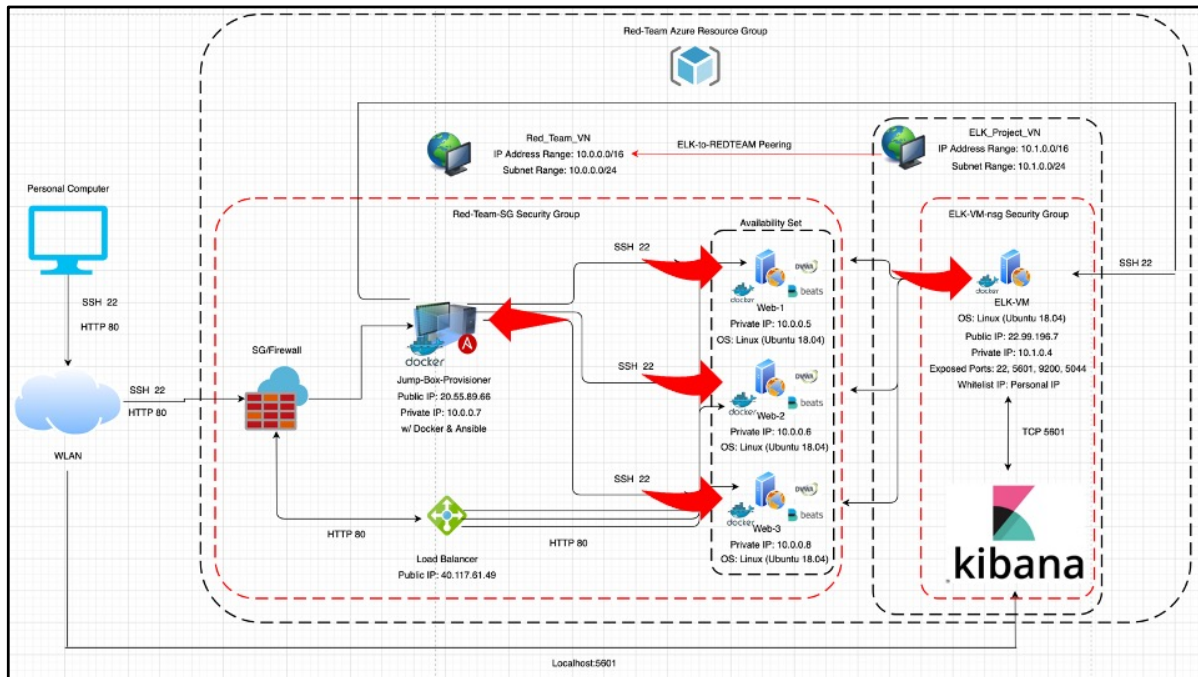


Now for the Network Topology

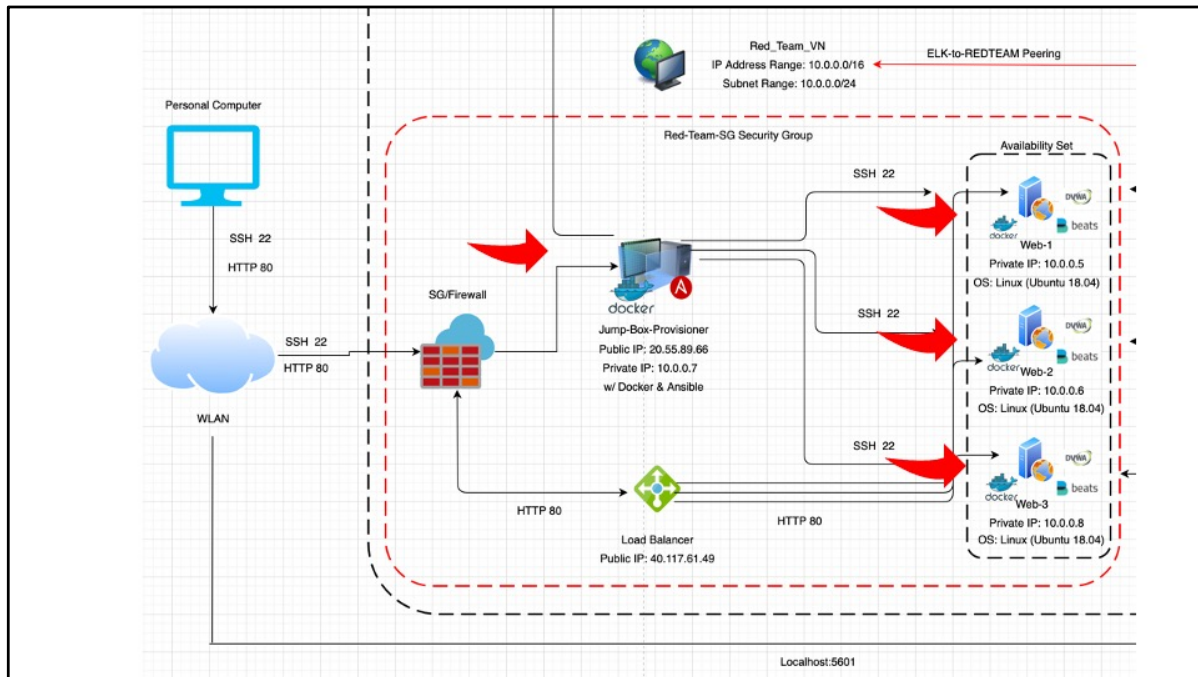


You can see from this diagram that every machine deployed utilizes the same Azure Resource Group. At its core, resource groups allow engineers to organize related resources into groups for a particular setup or project.

Within the resource group, we deployed two virtual local area networks, or VLANs. Our Red Team Virtual Network, shown on the left, and our ELK Project Virtual Network shown on the right. Both networks are in two different parts of the United States and to allow these two networks to communicate, a network peering is created here.

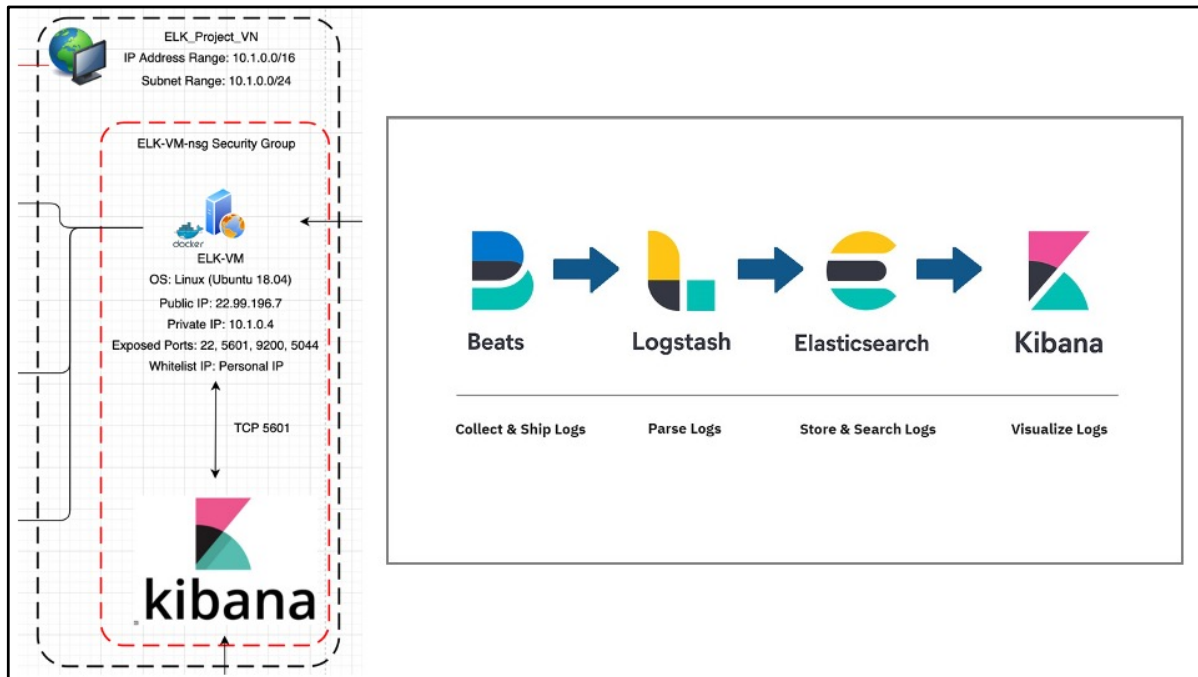


Every Machine deployed on the network is a headless Linux Ubuntu machine. A “headless machine” is a computer that does not employ the usage of the Graphical User Interface. This frees up a substantial amount of CPU and Memory, ultimately allowing us to deploy lighter, faster machines that will save a company on time, resources, and money.



Our Red Team Virtual Network, shown here, hosts a jump-box-provisioner machine and three, load-balanced web servers. The jump-box-provisioner machine is configured to only allow connections from my personal computer via SSH, or secure shell. Any machine within the network is allowed to accept connections from the jump-box-provisioner via SSH. This allows us to move freely within the network without exposing our web servers to the public. Using SSH with a password is inherently weak, therefore we have generated SSH key-pairs which utilizes asymmetric encryption and only allows a user with the private key to access the machine.

The Availability Set grouping you see here is a grouping of virtual machines, where each virtual machine is in a different Availability Zone. This means that if one of the Availability Zones hosting our machine goes down the other machines in the set are able to pick up the workload. This is built in to ensure that there is no single point of failure for our web servers.



Our second virtual network, shown here, hosts a deployment of the ELK container which stands for Elasticsearch, Logstash, and Kibana. This is the security information and management system mentioned earlier that will monitor activity taking place within our network. Access to the Kibana interface has been allowed only from my personal computer through my web browser.

After the installation and deployment of our web application, we will install forwarders, called beats, that will send data from the web servers to our ELK stack deployment for analysis.

Overview of Security Group Rules

- Implicit Deny
 - *Deny all inbound traffic unless it is explicitly permitted*
- Allow SSH connections from local IP address
 - *Used to access the jump-box-provisioner*
- Allow SSH connections from jump-box-provisioner
 - *Used to move around the web servers*
- Allow Virtual Network Inbound
 - *This allows the machines to communicate across the local virtual network*
- Allow HTTP traffic to the Virtual Network
 - *This allows traffic sent to load balancer from the public network*
- Allow HTTP traffic forwarding on Load Balancer
 - *This allows the load balancer to distribute traffic to our web applications*



An overview of our security group rules are as follows.

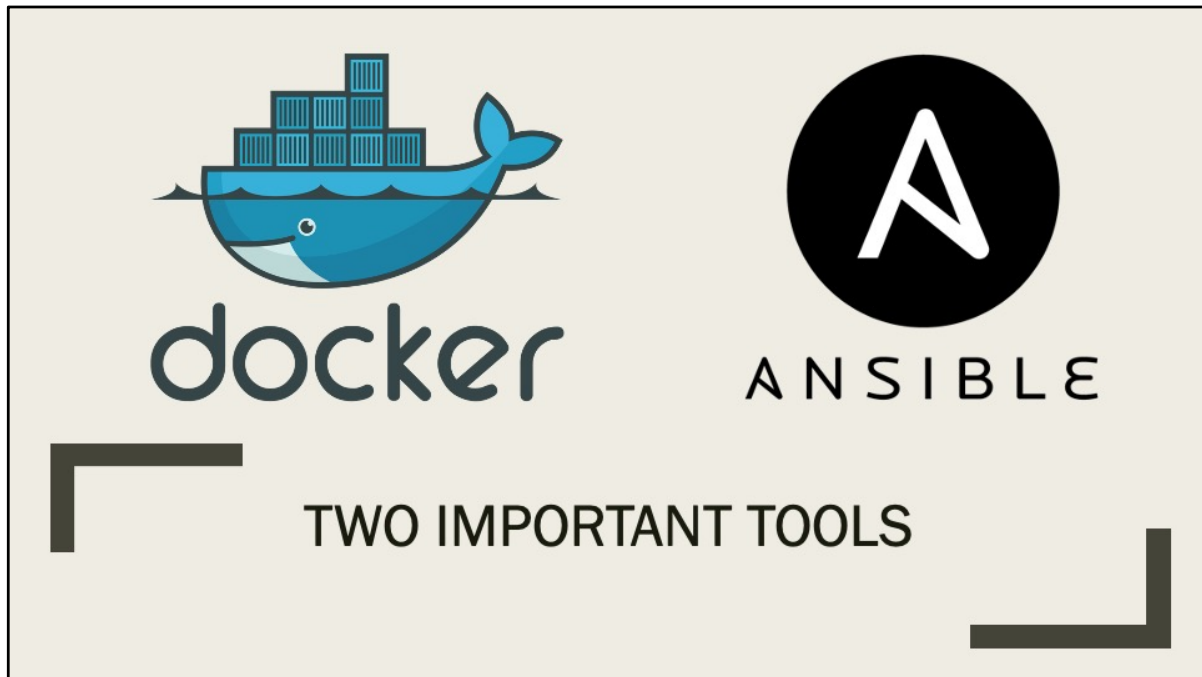
We only allow public network traffic to the Load balancer, which then forwards the traffic to the web servers allowing us to focus on strong configuration for one router as opposed to multiple servers. This is an example of secure architecture and is also called fanning in.

- Implicit Deny - Denies all inbound traffic unless it is explicitly permitted
- Two SSH rules to allow us to move through our network
- An inbound rule that allows our machines to communicate with each other within the local network
- And two HTTP rules to allow our website to be load balanced, live, and usable



AUTOMATED DEPLOYMENT

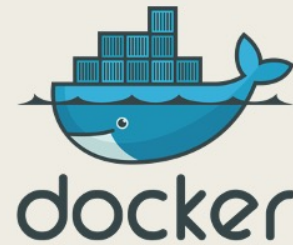
Now we have our Architecture set up, but no software to run.



To increase security, scalability, and allow for continuous Integration and deployment, we can best achieve this through the implementation of containers, provisioners, and the idea of Infrastructure as Code (IaC).

To do this we will be using two important tools: Docker and Ansible.

Docker



- The most common program used to manage containers
- Open source and free
- Hosts an extensive community and container hub for future support and resources
- What is a Container?
 - *A container is a standard unit of software that packages up code and its dependencies into an executable, standalone application.*
 - *Containers can be thought of as lightweight VMs*
 - *Containers are easily distributed, scalable, and secure*

Containers have been compared to lightweight virtual machines, but can be downloaded and distributed more easily than normal virtual machines. Docker is the most common program used to manage containers. It is a free open source software that has a thriving community online through their website and docker hub. Docker hub is a resource anyone can use to store containers or download containers created by other people. For this project, we pulled our web application from docker hub.

Ansible

- Provisioning Software used in automating IT infrastructure deployments
- Open sourced and free
- Infrastructure as Code (IaC) – Deployment configurations are defined in human readable, text files
- YAML “playbooks” provide the configuration instructions for Ansible
 - “YAML Ain’t Markup Language”
 - *YAML is a human-readable data-serialization language*
- Other common provisioners include Puppet and Chef



Ansible is a free, open-source provisioning software that is used in automating infrastructure deployments. Ansible Uses YAML “playbooks” to run “plays” or “actions” on a container or virtual machine. YAML stands for “YAML Ain’t Markup Language” and is also used in other provisioning software like Puppet and Chef. Without this, we would need to access each computer individually to implement updates and new configurations.

Automatically configuring machines through the usage of provisioners and text-files ensures that there is repeatability, consistency, and a significant reduction in the potential for human error. With provisioners, you can potentially configure thousands of identical machines all at once and include measures for proper version control.

All together this is known as Infrastructure as Code.

This is known as Infrastructure as Code which allows for machines and network deployments to be defined in text files like the YAML files mentioned here. Since these deployments are defined in human readable text files, any change to the machines or network can undergo a structured version control for added security . If something goes wrong, remediation to previous network settings would be easily

implemented with a previous version. #####

Summary

- We deployed a cloud environment that:
 - *Followed the rules of least functionality*
 - *Followed the Rules of least privilege*
 - *Is headless to save resources*
 - *Is redundant for high availability*
 - *Is monitored through a SIEM*
 - ELK - Elasticsearch, Logstash, Kibana
 - *Is Repeatable, Scalable, Version Controlled*
 - Docker, Ansible, YAML
 - *Is a vulnerable web application*
 - DVWA - Darn Vulnerable Web Application



In Summary:

Each process followed the rules of least functionality which is the idea that access should be limited to only the information and resources necessary for its legitimate purpose. The usage of containers is a perfect example of this since containers by their very nature are self contained and do not need to interact with other containers.

Each process implemented also followed the rules of least privilege which states that any user, program, or process should have only the bare minimum privileges necessary to perform its function. A perfect example of this are the steps we took to only allow access to the machines through an SSH encrypted key-pair on our jump-box-provisioner machine.


We utilized only the resources necessary through headless servers and created three load-balanced instances of the web application for high availability.

We implemented the concept of Infrastructure as Code (IaC) to allow for repeatable, scalable, and version-controlled deployments for current and future applications.

But ultimately, we built a playground for penetration testers and security analysts to practice executing and monitoring the various Tactics, Techniques, and procedures (TTPs) that companies are likely to face in the real-world as we transition more and more to the cloud each day.

There is always more we could do to further harden these structures, but this is a good start towards exploring the ideas of a Defense in Depth model.

If you would like to see the exact files used for this project, you may find them at my Github located on the next slide.



Thank you for your time!

- Jacob Small
- jacobjohnsmall@protonmail.com
- <https://www.linkedin.com/in/jacobjohnsmall/>
- <https://github.com/Jacobsmall2424>
 - *Files for this deployment can be downloaded here*

Thank you all so much for your time. I will take any questions now and, if you have any comments or advice you can lend, I would be happy to hear those as well.