## Jesus Galeno - HW-13 - ELK Stack Final Project

Using containers in cloud deployments can have many benefits, inluding mitigating some serious security risks, though it's important to differentiate when it would be appropiate to use a container versus a virtual machine. Both virtual machines and containers may offer various conveniences over the other; however, one major advantage to using containers over a virtual machine would be the isolated environment provided by a containter that can be easily managed. By isolating an environment, a system administrator could have a better overview of the machines being managed. Another major advantage is how lightweight a container is. In a cloud environment where most operations are done over the internet, the smaller the machine and resources, the faster and more efficient the network and operations can be. Nonetheless, there are times when more resources are needed and thus the situation might call for using a virtual machine instead.

For example, in our ELK Stack project we were instructed to deploy a 'Provisioner Jump Box' virtual machine that would be responsible for running an Ansible Docker container (that manages the Web servers and ELK machine). After getting our main Jump Box machine running, I installed Docker in this machine, then pulled and deployed the Ansible image from Docker. Rather than creating a completely new machine to run Ansible, I used Docker within the Jump Box to save resources and also limit the amount of machines that needed to be connected. The Jump Box machine being the only machine I could directly SSH into from my Local Machine at home allowed the Web Server and ELK machines to be even more isolated and deeper within the network. An attacker attempting to penetrate the Web or ELK machines would have a difficult time because they were all only accessible through SSH of the Ansible container. Because the only purpose of the Docker container is to run Ansible, this environment made it convenient to create playbooks and configure any new machines added to the network. Lastly, the Web machines on the network were configured with Ansible to pull, deploy and host the DVWA web page.

The manner in which the ELK and Web machines were protected was achieved by using SSH RSA Public Keys. After establishing a direct SSH connection from my Local Machine to the Jump Box, I attached to the Docker container within that machine and used it's SSH RSA keys to limit administrator traffic to the host machines. Rather than using a connection directly from my Local Machine, this method allowed me completely limit the connections to the Web and ELK servers. For the Web machines, I created a incoming security rule on the network that blocks incoming traffic from all ports except for ports 22, which remained only authenticated by Ansible container, and 80, which was allowed incoming connection over HTTP to access the DVWA page. As for the ELK machine, this machine was created on a completely different virutal network; however, I was able to bridge the connections of the networks by creating a network peering. This peering allowed the newly created ELK-VNET to directly connect to the already existing RED-VN, thus allowing quick and efficient traffic.

The solutions implement provided a very efficient, secure way of connecting and managing the entire resource group. As previously discussed, limiting access to machines can mitigate potential security risks and reduce unwanted traffic. Nevertheless, this deployment can still have its drawbacks. For example, in the event of having to fix a quick problem with any of the machines (Web servers or ELK), the sysadmin would have to first SSH into the Jump Box and then attach to the Docker container. Reducing the amount of machines that can connect directly is safe, but might not be very convenient in the event that an administrator would want to directly connect to any of the host machines. In addition, because the container is lightweight, it might be difficult to manage a larger network in which many machines would have to be configured. When running the Ansible playbooks, I experienced some hang ups and slow deployment which can most likely be traced back to a low-resource machine. A virtual machine with more virtual memory or higher processing power would presumably experience less of these issues.

To best configure a safe, efficient virtual network with cloud containers, we must consider the follwing: how many machines are being managed and thus how much resources are needed. If the situation calls for a small cloud network like what was used in our ELK stack project, using containers would be an appropiate solution for this case. First, it would be needed to deploy a provisioner virtual machine like our Jump Box that allows the sysadmin to control the network. Within this machine we may use a Docker image like Ansible to then configure and manage any additional server/machines on the network. Within the servers themselves, these machines can run their own instances of Docker like our Web machines did. Our Web machines main purpose was to host the DVWA page and because they were configured to run within a backend pool, if one machine went down, the Load Balancer would automatically switch traffic to the next machine. The Ansible container was a bit more instrumental in the management of the network because it also had the ability to run a playbook to the ELK machine. By configuring the Ansible hosts file to allow access to the ELK server, it becomes extremely easy to configure that machine as needed. In a situation with multiple virtual networks like what we had to work with, using network peerings and modifying the playbook files prevents the wrong machine from being configured. It's imperative to make sure that the target host is the desired one to be configured. Overall, the use of containers facilitated many aspects of the system adminitration and made sure that security risks were mitigated while doing so.