Michael Simmons

Project 1

Automated ELK Stack Development

**Objective**:

This activity’s goal is creation of a working web server serving the D\*mn Vulnerable Web Application (DVWA) and an Elasticsearch, Logstash, Kibana (ELK) instance connected to the web server serving Kibana.

Academic course work prompted this effort so the underlying goal is students learning these aspects of Cybersecurity and infrastructure deployment, not building something commercially viable. This reports demonstrate the mastering of those concepts.

**Second Level Objectives**:

Use free version of Azure to build it. The web servers should have high availability and capacity. Thus there are three with traffic metered by a load balancer. This provides availability. If one web server goes down users can still access the application. The load balancer also evens out the traffic which improves capacity. Other objectives were firewall/network security rules to limit access to the servers and Jump-Box. And one practical objective was to build a system that fits within the constraints of the free version of Azure.

Pros/Cons: This approach is fine for an academic endeavor since it provides experience to build an Azure infrastructure that works and learn the concepts. It’s not nearly sufficient for a real commercial deployment since it still has some single points of failure and would be under-powered. I also wonder, but can’t comment directly whether there is a similar AWS free version that could have also been used. Perhaps gaining AWS experience is more valuable. Or perhaps there isn’t a suitable free version and that’s why Azure was chosen.

**Scope**:

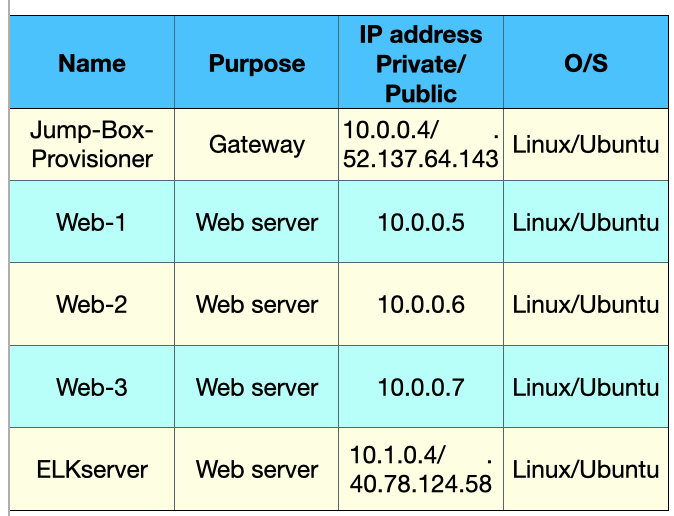
This README provides a high level view. Associated documents stored in this directory supply more details. Those documents are Proj1-Azure, Proj1-docker-ansible, Proj1-DVWA and Proj1-Kibana. All are word docs.

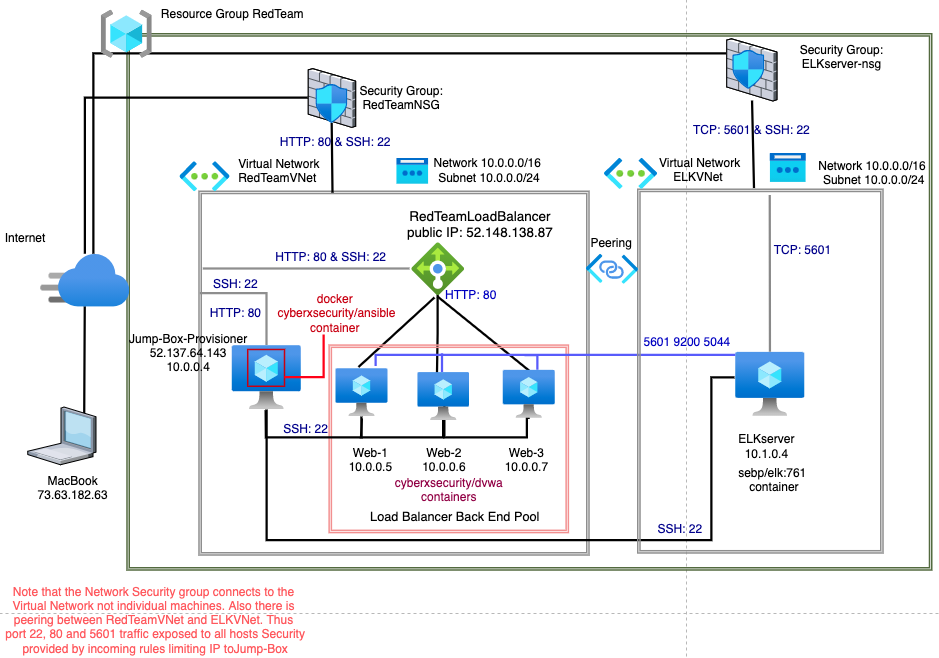
This and supporting documents cover:

* Architecture.
* Blueprint on instantiating the architecture including:
  + All steps to create the instance in Azure.
  + Using docker and ansible to build the web and the ELK servers.
* Following this blueprint enables anyone to create the entire deployment.
* Results on using Kibana to produce various types of analyses
* Exploration of DVMA. The activities did not focus much here, and I wanted to explore it more.
* This document is organized by first presenting an overview with details on the topics above in subsequent sections. Mostly those details are in the associated documents.
* Portions instead of the whole system may be installed as well. For example if you have a working ELK server and want to install Filebeat follow those steps and use the files documented in that document.
* Azure components and associated rules are all documented.

All work done on this project was in the Azure Cloud instance plus using my Mac’s terminal to access the Azure Cloud Jump-box.

**Design**:

Designing the system was not part of this academic exercise. This effort builds a pre-defined architecture. It was built following the Activities sheets and resulting in the diagram below. Details as to each VM’s Name, function, IP, etc appears in the Proj1-Azure document.



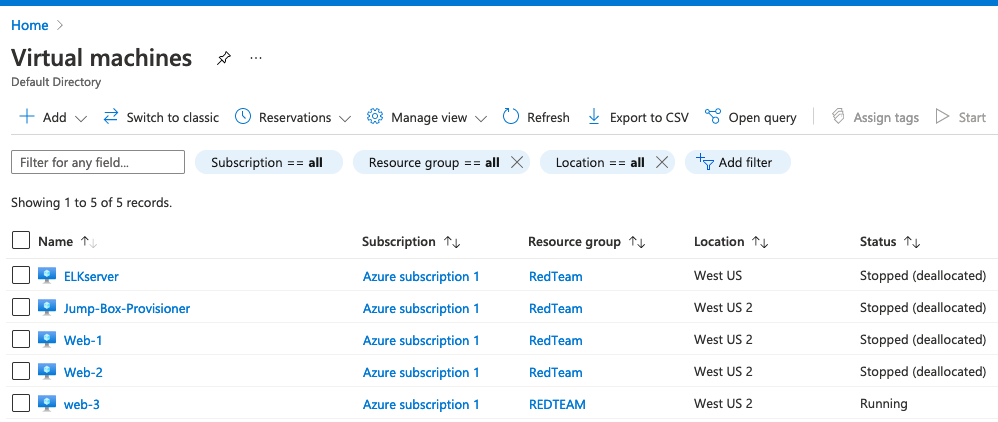
**Features/benefits of the design:**

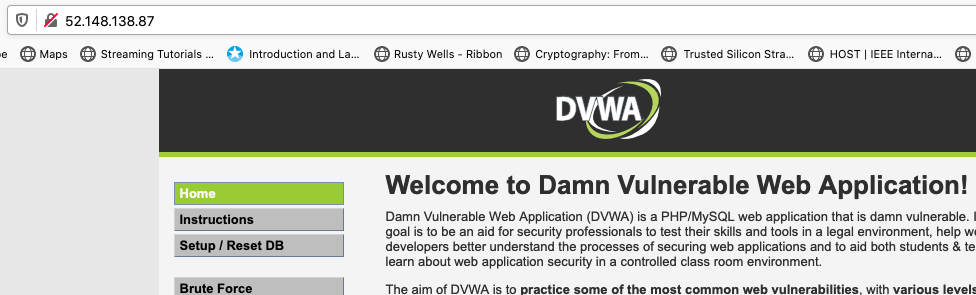
Jump-Box. Here access to the various hosts is only through the Jump-Box and not directly which adds an extra layer of protection.

Multiple web-servers behind a load balancer provide redundancy making the web servers highly available since if one or even two go down there is still another one running. The load balancer also adds security by shielding the web servers. It’s not necessary for them to have public IPs so they can’t be reached that way.

The ELK server with Kibana allows monitoring of key security aspects of the system for protection. The webservers can be monitored for a variety of items. Filebeat provides visibility into issues that the logs capture. Metricbeat provides visibility into as its name suggests, metrics. There is visibility into many metrics. CPU and memory usage to name two.

**Demonstration of high availability:**

In Azure only started Web-3. All the other 4 servers are stopped. So that simulates 4 machines going down.

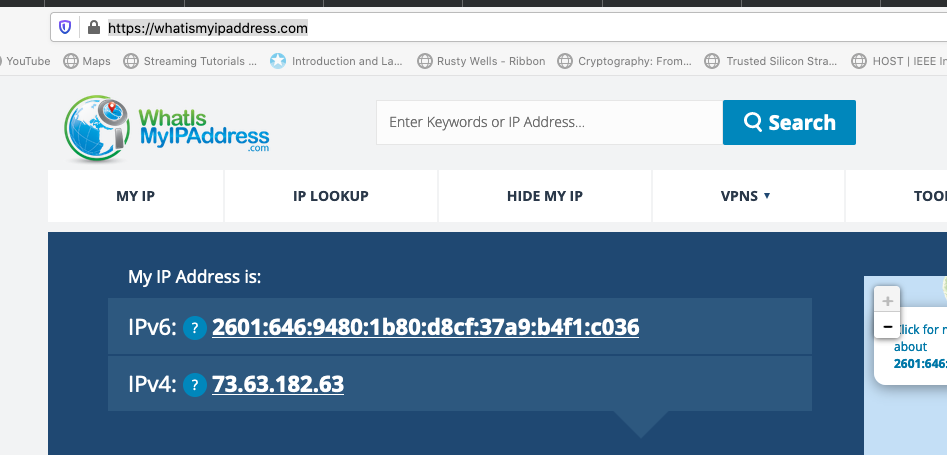
Now access the application:

So even if any machines go down as long as one web server is up the app is still served.

**Access policies**.

The Web servers don’t have public IP’s and are behind a load balancer so they can’t be directly accessed. The Jump-Box and ELKServer are publicly accessible but are restricted by the following rules:

Ports 22 and 80 are allowed onto Virtual Network from only IP 73.63.182.63 for RedTeamNSG rules.

Port 5601 is allowed onto Virtual Network from only IP 73.63.182.63 for ELKServer-nsg rules. Since there is peering ports 22, 80 and 5601 are allowed onto any machine but only from IP 73.63.182.63. Since that IP is my Mac’s public IP it means essentially nothing besides me is allowed to access them. So this is very tight security but also very restrictive in that only I can run the web applications on ELK and Web servers. So fine for this exercise but may not be so useful in the real world.

**First things first**:

These items are used during the setup and should be done early on.

**Determine public IP address of Mac.**

Go to <https://whatismyipaddress.com>

The IPv4 IP address us 73.63.182.63

This will be used in setting up the NSG rules. Note it’s possible this could change since public IP’s don’t last forever. If it changes would have to update the rules the new public IP.

**Azure free account.** Go to [portal.azure.com](http://portal.azure.com) and sign up for the free $200/30 day subscription under an email different from the one used in the previous Azure labs for this course. Once established always make sure to turn off any VMs created when the work is over.

**SSH keys**. Jump-Box gateway uses SSH to access the various machines. This requires generating public keys. Public keys were generated both natively on the Mac to use to connect to the Jump-Box and on the docker container on the Jump-Box to connect to the web machines.

**Building the Azure Infrastructure.**

The entire diagram aside from the Mac is built in Microsoft Azure. A subscription was created for the free 30 days $200 dollar credit. I used less than $20.00 of it. Need to be careful to turn off the VM’s when not using them. The document Proj1-Azure has every step needed to build the entire Azure infrastructure. Docker and ansible were then used to fully enable the application capability.

I don’t follow exactly the activities flow. I present a flow to be used if the entire architecture is built at one time instead of incrementally adding the ELK server later.

The following, in some cases multiple instances, were created:

* Network Security Groups
* VIrtual machines
* Availability set
* Virtual Networks
* Load balancer
* Resouce Group

The free version limits 4 cpus per region (Location where the component is physically located.) This architecture has 5 VMs. Spreading the VMs across regions works around the constraint. Thus the ELKserver is in a different region. Virtual networks are associated with a region so need two. The are connected seamlessly through peering so the net effort is a single network. This also requires a 2nd Network Security Group which also means two sets of rules.

Only the Jump-Box and ELK server have public IPs.

The Resource Group is the high level container and is created first. Next build Virtual networks to connect everything. Connect the two networks through peering. Then create the Network Security Groups which are similar to firewalls in that they have rules limiting access by IP and port. Add rules that limit traffic from only my Mac’s IP and allow only ports 22, 80 and 5601. All that traffic is exposed to the entire peered virtual network. It seems better to be able to limit traffic to individual VMs for better control and security but this flavor of Azure didn’t provide that capability. Next build the load balancer. It provides both availability and spreading the work for better performance. It also allows the web servers to hide behind it so they don’t need public IPs better protecting them. The web servers need to be added later to the load balancer’s backend pool (which also needs to be constructed) to enable this functionality. Finally build the VMs and add them to the back end pool. There is also an availability set created.

See Proj1-Azure for all the detailed steps.

**Docker/ansible playbook**

See Proj1-docker-ansible document for step by step. It also contains the required cfg files and YAML files. A combination of docker and ansible fully provisioned all the servers. On the Jump-Box docker was installed and a cyberxsecurity/ansible bash container was instantiated.

This serves as the access method to all other machines. The Jump-Box container was then used to enable the Web servers through ansible. Jump-Box container ran the ansible playbook that both installed docker and did the web setup. Same for the ELK server.

The ansible cfg file on the Jump-Box container also needed to be modified to account for the web machines IPs so it could access and configure them.

This means both the ELK and web servers were configured automatically through code. One major benefit is all three web servers are configured exactly the same . Whereas doing it manually is prone to errors. Also once a good known method of configuring a web server is created it can be cloned and re-used in the future. Presumably this playbook is based upon a previous one that was demonstrated to be good. So it’s a solid automated way of propagation of known good instances.

To provide a glimpse into the ELK server ansible process:

It uses YAML. Here are the key steps it does automatically through the playbook:

* Install [docker.io](http://docker.io)
* Allocate more memory
* Download & Launch elk containsr sebp/elk:761

The purpose of the ELK server is to monitor the web servers health through log collection/aggregation and metrics visibility. This is done with metribeats and filebeats. With that one can see file/log based events like logins, significant events, etc. One can also see metrics such as CPU usage.

Those machine’s IPs

10.0.0.5

10.0.0.6

10.0.0.7

After SSH to Jump-Box and starting and attaching to the docker container,

Install\_elk.yml was obtained from Activities 13 day 1 in the Resources folder. It was copied to /etc/ansible/roles/files. The /etc/ansible/hosts file was also updated to include the ELK server’s IP for installation and make it a separate group. Then run the playbook in Ansible. If it runs with no errors then you can SSH to the ELK server to check the installation. You can also run curl <http://40.78.124.58:5601/app/kibana#/home> You can also put that URL in a browser to check it’s running.

The Proj1-docker-ansible file provides the details of what’s in the playbook. It also show how to configure and run it.

**Kibana Analysis.**

The purpose of creating the ELK server was to setup an ELK (Elasticsearch, Logstash, and Kibana) stack, enabling Kibana analyses. The Proj1-Kibana file documents all the filebeat and metricbeat analyses. Some of the output was run on the sample data set provide as part of Kibana and some on actual data collected from the web servers.

**DVWA Analysis.**

The intended functionality of the web servers in the Azure architecture was to create an instance of DVWA. DVWA provides some valuable lessons on the web aspects of Cybersecurity so I explore a few of those since the information and practical knowledge gained is useful. Those details appear in the Proj1-DVWA document.