

A Fully Automated, Centralized Inspection Architecture with AWS TGW, GWLB, and Layered ALBs *Featuring Palo Alto Networks NGFWs*

Dan Edeen

Palo Alto Networks PCNSE

AWS Certified Advanced Networking – Specialty

March 2023

I. Introduction

As companies migrate their applications and data centers to cloud architectures, the models used to secure networks and computing resources must be updated. With distributed API-centric applications, port-based firewalls at the network perimeter are no longer sufficient. One approach to securing cloud-based environments is creating a centralized inspection architecture, deploying security appliances in a dedicated zone (a VPC in AWS), and routing all traffic through that zone for inspection. Additional security zone services may include traffic logging, DPI, and decryption. The centralized security architecture has proven to be a resilient and scalable approach to securing all traffic into and out of a cloud-based enterprise.

In AWS we call this approach a middlebox architecture. For outbound traffic, a Gateway Load Balancer (GWLB) routes all internally generated, extra-VPC traffic to a redundant pair of Next Generation Firewalls (NGFWs) for inspection, prior to being forwarded to its destination. Externally generated, inbound traffic also passes through the firewalls before entering the enterprise. This is controlled by two layers of Application Load Balancers (ALBs) with the firewalls deployed between them. A Transit Gateway (TGW) serves as the conduit between all of the VPCs in the organization.

Palo Alto Networks, a champion of the centralized security/middlebox architecture, has developed VM-based versions of their firewalls, called PA-VMs, for use in cloud and hypervisor environments. In AWS, the PA-VMs integrate directly with a GWLB and VPC Endpoint Services for outbound and East-West traffic security enforcement. GWLB integration and overlay routing allow inspection of Geneve encapsulated traffic received from the interior, and NAT on the PA-VMs integrates with the dual ALBs to inspect inbound service traffic.

AWS and Palo Alto Networks have published a reference architecture and deployment guide for implementing a centralized security zone with PA-VM firewalls. In this study, I have extended the model to include complete **turn-key IaC automation, protocol-based traffic forwarding, firewall bootstrapping from S3, IPSec VPN encryption, Splunk SIEM logging, Panorama virtual servers, and RSA key pair management**. I have also automated **configuration management** for the PAN firewalls, Panorama servers, Splunk SIEMs, and web servers in this architecture. The full IaC repository for this project, written in a combination of **Terraform, shell script, and AWS CLI**, is available from the author on GitHub.

II. Project Goals

My objectives for implementing this centralized security architecture included the following:

- I. Create a complete, fully-redundant middlebox architecture in AWS. The design includes a service VPC hosting web servers (HTTP and HTTPS), multiple consumer VPCs with end user Linux systems, and a management VPC for administration and logging. The design must be resilient, with all traffic to and from the Internet, plus all inter-VPC traffic, routed through redundant Palo Alto Networks NGFWs. Inspection, application of security policy, and optional SSH and SSL decryption of all enterprise traffic must be supported, with the exception of intra-VPC traffic, which is not typically routed outside of its VPC.
- II. Develop a complete IaC automation suite for creating this architecture. The suite will build all of the AWS infrastructure (VPCs, subnets, route tables, ALBs, GWLB, VPC Endpoint Services, ENIs, public IP addresses, IGWs, NACLs, and Security Groups). It will also deploy the firewalls, management servers, logging servers, web servers, and end user computers, instantiated on EC2s using vendor specified hardware profiles for each platform.
- III. Automate the launch configuration of the PA-VMs, Panorama servers, Splunk SIEMs, and web servers. The intent here is to make the entire architecture operational through automation. Additional development tools are also included in the IaC repository.
- IV. Document the architecture and important features such as the dual ALB design, integration of the PA-VMs with the GWLB, and IPSec tunnel establishment to an external customer gateway. This paper, plus additional engineering notes in the IAC repo, provide this documentation.

The complete architecture that I created and automated is shown in Figure 1.

III. Design Resilience and Redundancy

One of the design requirements for this project is a fully resilient architecture, guaranteeing continued operation in case of failures within the system. This is accomplished through a combination of natively resilient AWS services, design choices, and deployment of redundant resources. Some of the high availability aspects of this architecture are described below.

- All compute resources (EC2s) are duplicated and distributed across AWS Availability Zones (AZs), providing continued operation in case of AZ failures. This approach increases the configuration complexity and run-time cost of the solution by doubling the instance count, AWS service points, subnets, and route tables. The additional computing resources also increase the number of ENIs and publicly routable IP addresses required.
- Two Palo Alto PA-VM firewalls are deployed in separate AZs, ensuring continued operation of security and traffic forwarding if a single firewall becomes unavailable. In addition, dual Panorama management servers and Splunk SIEMs are created in the management VPC, across AZs, configured to operate independently.
- The GLWB and dual ALBs perform health checks on the firewalls and web servers, directing traffic to avoid failures in the architecture. The TGW is a resilient AWS service, and the TGW attachments are deployed in two AZs within each VPC. The ELBs, IGWs and S3 are also resilient AWS services.
- The IPSec endpoint configurations for Site-to-Site VPNs are included in both PA-VM firewalls, and will operate in parallel or in simplex mode, ensuring remote connectivity if a firewall or its associated infrastructure fails.

A detailed overview of the architecture, as implemented in this project, is shown in Figure 1. Additional details are provided throughout this paper.

AWS Centralized Inspection Architecture with PA-VMs, dan edeen 2023

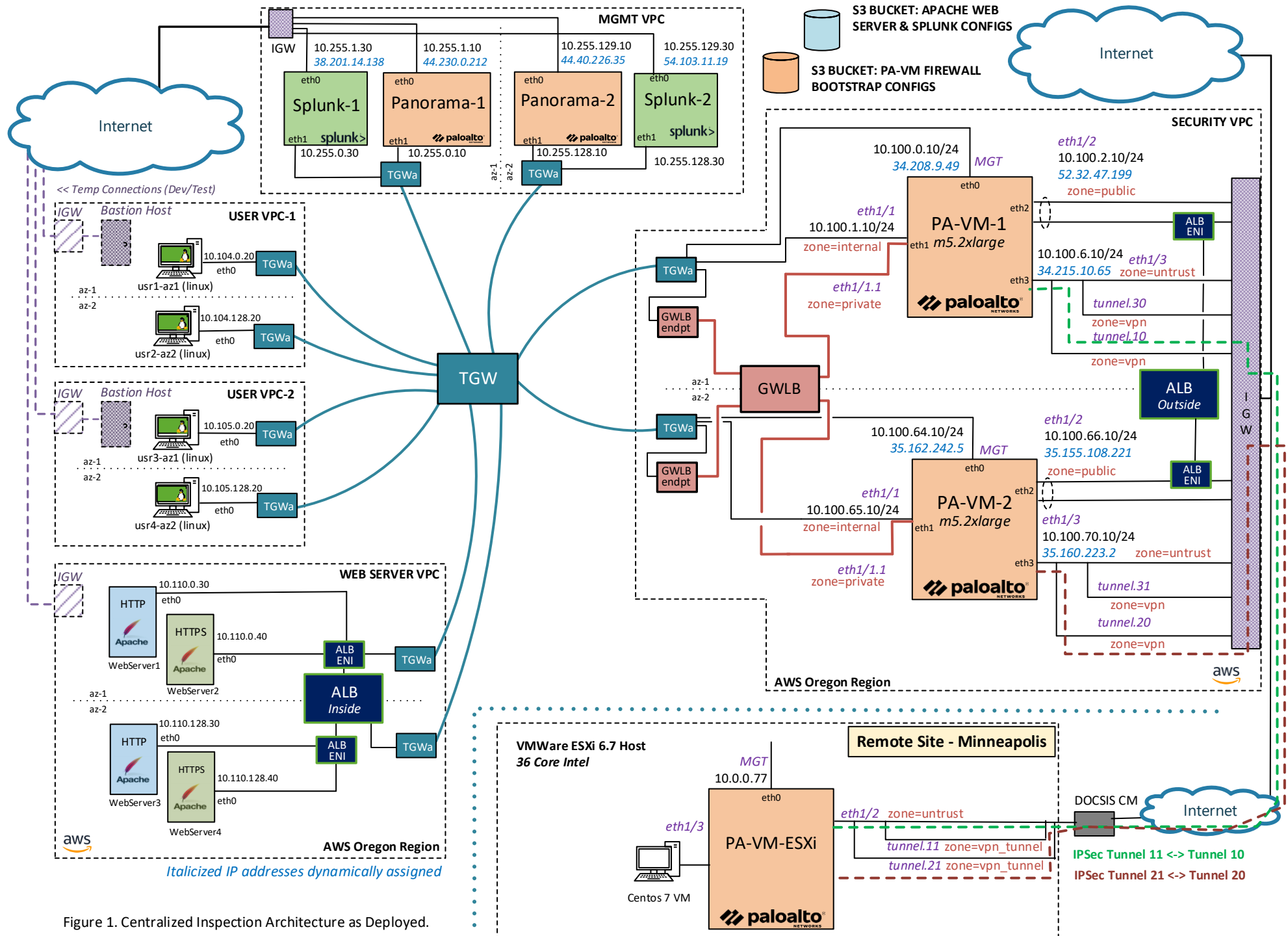


Figure 1. Centralized Inspection Architecture as Deployed.

IV. Architecture

The design for this project is a resilient centralized security architecture, within AWS, with Palo Alto Networks firewalls and Splunk SIEM logging. This section provides additional details of the architecture created.

A. AWS Core Elements

The AWS core consists of 5 VPCs, interconnected with a Transit Gateway for inter-VPC traffic and connectivity to the Internet.

Security VPC: This VPC is the heart of the architecture, providing traffic inspection and policing, plus most of the other security features. All traffic to and from the Internet, as well as all inter-VPC traffic, passes through the security VPC for inspection by the PA-VM firewalls, which are distributed across two AZs. The security VPC is segmented into 14 subnets, each with a separate route table, allowing precise control over traffic through the entire architecture.

Management VPC: This VPC hosts two Panorama management systems, plus two Splunk Enterprise SIEMs collecting syslog data from the PA-VMs. Traffic to and from the management VPC is routed through the TGW for internal connectivity to the other VPCs, and through an Internet Gateway (IGW) for external access to these four systems. Both the Panorama and Splunk servers are distributed across AZs for service resilience.

Web Server VPC: This VPC hosts two pairs of Apache web servers running linux and distributed across AZs. One pair serves HTTP exclusively and the other serves only HTTPS. I have written a script that creates temporary backend access to the web servers by adding an IGW, ENI, public IP address, and temporary route table. This is required for the automated initial configuration the web servers. This access is also useful for debugging the web server configurations, as the only route to them in production mode is through the security VPC and firewalls, and only HTTP and HTTPS traffic are allowed along this path.

User VPCs: Each of these VPCs contains two end user linux hosts, distributed across AZs. These hosts generate outbound traffic to the Internet, as well as East-West traffic to the web server VPC. I have written a script that provides temporary external access to these hosts through an SSH-enabled bastion host.

S3 Storage: This solution uses two S3 buckets for configuration management of the servers. One contains the bootstrap configurations for the firewalls, and the other contains configuration files and scripts for the web servers and Splunk servers. The buckets are created and content is staged by the IAC automation.

IAM and Key Pairs: Terraform scripts create the RSA key pairs for EC2 instances and the IAM role needed for the EC2s to access the S3 buckets. The role is attached to the firewalls, web servers, and Splunk instances so they can access their initial configuration files.

B. AWS Traffic Routing/Forwarding

This design leverages multiple AWS services to interconnect resources resiliently.

Traffic Flow:

- **Inbound traffic** from the Internet to the web servers enters through the IGW attached to the security VPC, passes through the external ALB, one of the PA-VM firewalls, the TGW to the web server VPC, and through the Internal ALB before reaching the web servers. Traffic flow is routed by protocol (HTTP/80 or HTTPS/443) to one of the applicable web servers. Return traffic follows the same path. Asymmetric return traffic is avoided by the NAT configuration on the firewalls.
- **Outbound traffic** from the end user systems to the Internet travels through the TGW, one of the GWLB endpoints, the GWLB, one of the PA-VM firewalls, and the IGW attached to the security VPC. Return traffic follows the same path. Asymmetric return traffic is avoided by correlating sessions through the firewalls with their source VPC endpoint IDs.
- **East-West traffic** from the end user systems to the internal web servers follows the same path as outbound traffic, but instead of de-encapsulating (Geneve) the traffic, the firewall routes it back through the same port as its ingress port, and sends it back to the GLWB and TGW to the web server VPC. Return traffic follows the same path.
- **Firewall log traffic** (syslog) egresses the MGT interfaces on the PA-VM firewalls, and is routed through the TGW to the Splunk servers in the management VPC.
- **External VPN traffic** connected via IPSec enters through the IGW inside an encrypted tunnel to one of the PA-VM firewalls, where the traffic is de-encapsulated, inspected, and routed to its target within the environment. Return traffic follows the same path and is encapsulated in IPSec prior to egressing the firewall.
- **Development/test traffic** to the end user and web server VPCs enters those VPCs directly via temporarily attached IGWs. These connections are created and removed by scripts in the repo.

Transit Gateway: This architecture is partitioned into 5 VPCs, each containing resources that provide specific services. Traffic between the VPCs is transported by a Transit Gateway with an attachment placed in a dedicated subnet within each AZ of each VPC. These attachments create an ENI in these subnets that receives an IP address from the CIDR block for the subnet. Routing between VPCs via the TGW is directed by a pair of TGW route tables, one for the security VPC and one for the other VPCs (spoke VPCs). **Appliance mode** is required on the TGW Attachments in the security VPC to ensure that traffic for each session returns via the same TGW Attachment, avoiding asymmetric sessions.

Gateway Load Balancer: Within the security VPC, outbound traffic is distributed between the PA-VM firewalls by an AWS Gateway Load Balancer (GWLB), deployed with two GWLB Endpoints in a dedicated subnet in each AZ that hosts a PA-VM. The GWLB performs health checks on the firewalls, providing path resilience in case a firewall becomes unavailable. The GWLB Endpoints are associated with VPC Endpoints, and traffic is encapsulated in the Geneve protocol. This mechanism is described in a later section of this paper. Only outbound traffic (and its return) traverses the GWLB in this architecture. Inbound traffic (initiated outside of the environment) does not pass through the GWLB, but instead is transported through the ALBs, as described next.

Application Load Balancers: The AWS Application Load Balancer (ALB) operates at Layer 7, distributing inbound service requests across a group of targets. Typically, an ALB is used for load balancing web service requests across multiple servers, performing health checks on the targets, and distributing traffic in a round-robin approach. In this architecture I am using two layers of ALBs, one at the perimeter of the network and one in the web server VPC. Inbound requests are routed through the PA-VMs to the internal ALB. I have also configured the ALBs to distribute inbound traffic by protocol, routing HTTP requests to one pair of redundant web servers and HTTPS requests to the other pair. This requires two target groups in each ALB, one each for HTTP and HTTPS. Additional architectural details are provided later in this document.

Internet Connectivity: Internet access is enabled in the security VPC and the management VPC. In both cases, this access is provided by IGWs, and the management servers and firewalls are bound to additional ENIs with publicly routable IP addresses. Using IaC, I also create temporary Internet access to the User VPCs through SSH-enabled bastion hosts, and to the back end of the web servers via a second, temporarily attached ENI with a public IP address.

IKE/IPSec Gateways: To access this architecture from external on-premise networks, I have configured IPSec tunnel endpoints on both PA-VMs. Traffic through the tunnels is encapsulated in IPSec and routed through the Internet to a third PA-VM I am running as a VM on an ESXi host. This provides connectivity between the AWS architecture implemented here and my remote data center. The IPSec configuration is included in the PA-VM XML files loaded via bootstrap from S3, and the PA-VM configuration for the ESXi host is included in the repo for reference.

C. Gateway Load Balancer & PA-VM Architecture

The GWLB configuration and integration with the PA-VMs is key to this architecture so I describe it here in detail. Palo Alto Networks firewalls include features that process Geneve encapsulation and perform overlay routing, both required for this architecture. The GWLB provides path resiliency in case one of the firewalls becomes unavailable, monitoring the health of both firewalls continuously. GWLB endpoints and VPC services endpoints are deployed within both AZs. The GWLB targets PA-VMs by IP address instead of by Instance ID, which avoids the need for interface swapping on the PA-VM/EC2s found with instance ID targeting. Cross-zone load balancing ensures that traffic can be sent to either firewall regardless of the AZ it originates in. The architecture for the GWLB and firewall integration is shown in Figure 2.

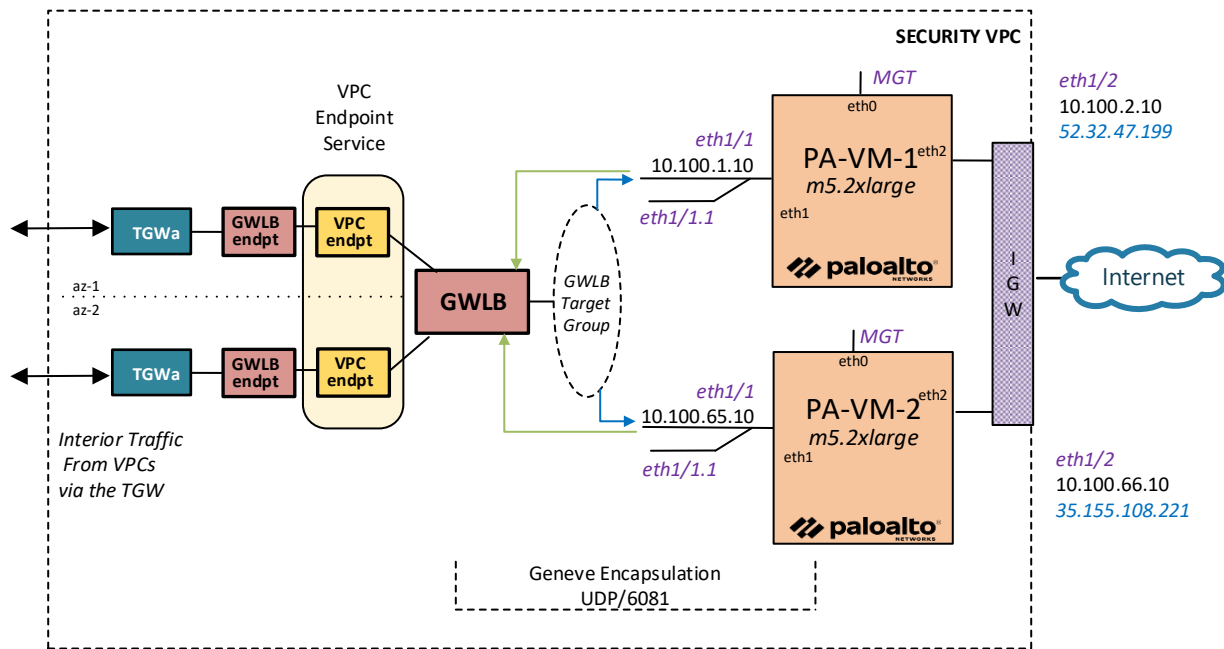


Figure 2. Gateway Load Balancer, PA-VMs, and Endpoints.

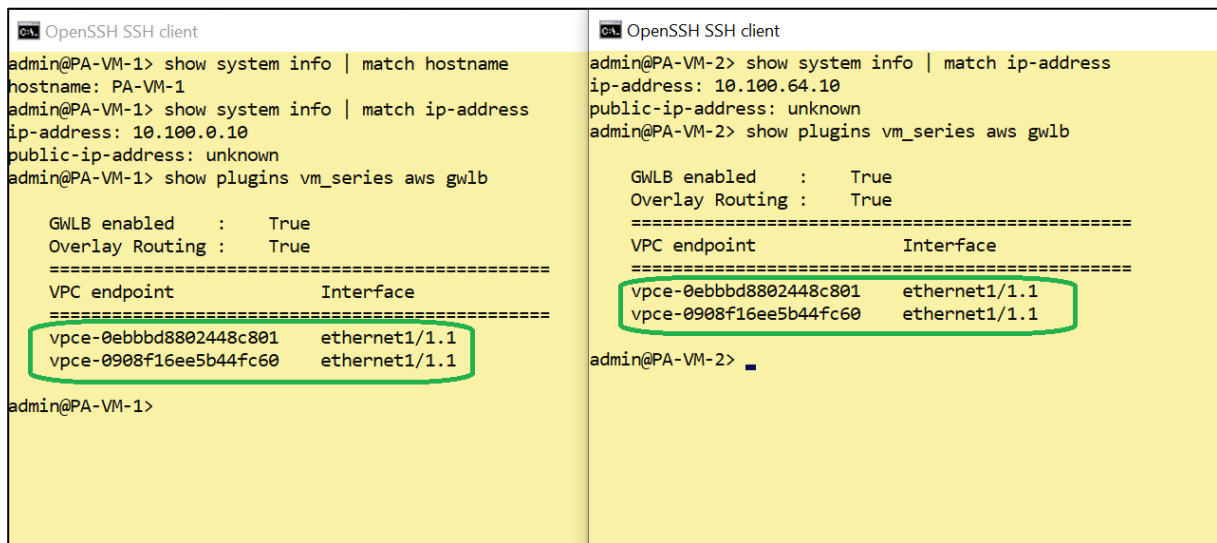
Traffic between the GWLB and the PA-VM firewalls is encapsulated in the Geneve protocol (UDP/port 6081), preserving the original packet headers. In this deployment I am creating two VPC endpoints, one in each AZ of the security VPC. The VPC service endpoint IDs are mapped to the 1/1.1 sub-interfaces on both firewalls (full mesh) via a CLI configuration, after the service endpoint has been created (and ID is known). The VPC endpoint ID is embedded in the Geneve header of each packet, allowing the firewalls to track the source endpoint for session management and return traffic routing.

Outbound Traffic: Traffic generated in the interior User VPCs will reach the security VPC via the TGW and its attachments. Overlay routing is configured on the firewalls (via CLI), allowing the firewall to inspect the inner headers the packets, perform routing lookup, and determine the egress interface (plus evaluate security policy). For outbound traffic to the Internet, the firewall performs NAT on the packet and sends it to the IGW. The IGW again perform NAT and sends the packet outbound. Return traffic follows the same path and process, is “reverse” NAT’ed at both the IGW and the PA-VM, encapsulated in Geneve, and sent to the GWLB.

East-West Traffic: Traffic generated in the interior User VPCs destined for the Web Server VPC follows the same path as outbound traffic to one of the firewalls, but the egress interface will be the same (eth 1/1) as the packet's ingress. The packet will be processed for security policy and routed back to the GWLB.

Health Checks: The GWLB performs periodic health checks on both PA-VM firewalls, and stops sending traffic to a firewall that fails the health check criteria. These packets are sent to the IP address of the private-zone eth1/1 interfaces (10.100.1.10, 10.100.65.1) using SSH on port 443, to the path '/php/login.php'.

The configuration of the GWLB/Firewall interworking must follow this sequence: First build the AWS infrastructure and instantiate the PA-VMs. After the VPC Endpoints have been created, retrieve their IDs from AWS. Finally map each VPC endpoint ID to the sub-interface eth1/1.1 on both firewalls. I have automated this procedure. Figures 3 and 4 show the firewall configurations and Geneve headers captured on the PA-VM. To capture the Geneve headers on the PA-VMs you must configure '*debug dataplane packet-diag set capture encapsulation yes*' on the firewalls.



```
admin@PA-VM-1> show system info | match hostname
hostname: PA-VM-1
admin@PA-VM-1> show system info | match ip-address
ip-address: 10.100.0.10
public-ip-address: unknown
admin@PA-VM-1> show plugins vm_series aws gwlb

  GWLB enabled   :    True
  Overlay Routing :    True
  =====
  VPC endpoint   Interface
  =====
  vpce-0ebbbd8802448c801  ethernet1/1.1
  vpce-0908f16ee5b44fc60  ethernet1/1.1
admin@PA-VM-1>
```

```
admin@PA-VM-2> show system info | match ip-address
ip-address: 10.100.64.10
public-ip-address: unknown
admin@PA-VM-2> show plugins vm_series aws gwlb

  GWLB enabled   :    True
  Overlay Routing :    True
  =====
  VPC endpoint   Interface
  =====
  vpce-0ebbbd8802448c801  ethernet1/1.1
  vpce-0908f16ee5b44fc60  ethernet1/1.1
admin@PA-VM-2>
```

Figure 3. VPCe Mapping to Firewall Sub-Interfaces. Note That Overlay Routing is Also Enabled.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.104.0.20	205.166.94.4	TCP	142	47934 → 22 [SYN] Seq=0
2	0.011942	205.166.94.4	10.100.2.10	TCP	74	22 → 13778 [SYN, ACK] S
3	0.013947	10.104.0.20	205.166.94.4	TCP	134	47934 → 22 [ACK] Seq=1
4	0.013976	10.104.0.20	205.166.94.4	SSHv2	155	Client: Protocol (SSH-2
5	0.065949	205.166.94.4	10.100.2.10	SSH	87	Server: Protocol (SSH-2
6	0.065978	10.104.0.20	205.166.94.4	TCP	134	47934 → 22 [ACK] Seq=22
7	0.066008	10.104.0.20	205.166.94.4	TCP	1582	47934 → 22 [ACK] Seq=22

> Frame 1: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits)

> Ethernet II, Src: 02:4e:ae:1a:01:47 (02:4e:ae:1a:01:47), Dst: MS-NLB-PhysServer-32_10:6b:59:87:e9 (02:30:6b:59:87:e9)

> Internet Protocol Version 4, Src: 10.100.5.215, Dst: 10.100.1.10

> User Datagram Protocol, Src Port: 60920, Dst Port: 6081

> Generic Network Virtualization Encapsulation, VNI: 0x000000

Version: 0

Length: 32 bytes

> Flags: 0x00

Protocol Type: IPv4 (0x0800)

Virtual Network Identifier (VNI): 0x000000 (0)

> Options: (32 bytes)

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x01 (Non-critical)

Class: Amazon.com, Inc. (0x0108)

Type: 0x01 (Non-critical)

Length: 12 bytes

Unknown Option Data: ebbbd8802448c801

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x02 (Non-critical)

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x03 (Non-critical)

> Internet Protocol Version 4, Src: 10.104.0.20, Dst: 205.166.94.4

> Transmission Control Protocol, Src Port: 47934, Dst Port: 22, Seq: 0, Len: 0

Class (geneve.option.class), 2 bytes

> Options: (32 bytes)

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x01 (Non-critical)

Class: Amazon.com, Inc. (0x0108)

Type: 0x01 (Non-critical)

Length: 12 bytes

Unknown Option Data: 908f16ee5b44fc60

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x02 (Non-critical)

Class: Amazon.com, Inc. (0x0108)

Type: 0x02 (Non-critical)

Length: 12 bytes

Unknown Option Data: 0000000000000000

> Unknown, Class: Amazon.com, Inc. (0x0108) Type: 0x03 (Non-critical)

Class: Amazon.com, Inc. (0x0108)

Type: 0x03 (Non-critical)

Length: 8 bytes

Unknown Option Data: fb0e10de

> Internet Protocol Version 4, Src: 10.104.0.20, Dst: 205.166.94.4

> Transmission Control Protocol, Src Port: 55252, Dst Port: 22, Seq: 22, Ack: 22, Len: 0

Figure 4. Decoded Packets Captured Between GWLB and PA-VM.

As shown in Figure 4, the VPC Endpoint ID that each packet is transmitted from is encoded in the Geneve encapsulation header. Here I have captured, exported, and decoded packets sent from each GWLB/VPCe endpoint to the PA-VM-1 firewall. The actual traffic captured was SSH, originating from the EC2 usr1-az1 (10.104.0.20) located in the first user VPC. The destination was a server on the Internet. The VPC endpoint IDs match those shown in Figure 3.

D. Two Layer Application Load Balancer Architecture

The Application Load Balancer (ALB) operates at layer 7, and provides scalable resilient service across multiple web servers. In this architecture, dual ALBs are used in a novel approach, designated an ALB Sandwich by Palo Alto Networks. Two ALBs are deployed, one external facing with a route to an IGW, and another one collocated with the web servers. The PA-VM firewalls are located between the two ALBs.

This design provides resilience for the firewalls in addition to the web servers and the internal AWS infrastructure between them. Here, I am using two target groups for each ALB, one for HTTP servers and one for HTTPS servers. The outside ALB targets the firewall public interfaces (eth1/2) with health checks, using HTTP/80 or HTTPS/443, depending on target URL. Within the firewalls, I have configured NAT to translate the destination address of the packets to the DNS name of the internal ALB, and the source address to the private interface address of the firewall (eth1/1). The health checks and web server traffic pass through the firewalls and through the TGW, and on to the internal ALB. From there the packets are directed to one of the web servers according to the load balancing algorithm employed and service port requested. The paths through this configuration are shown in Figure 5.

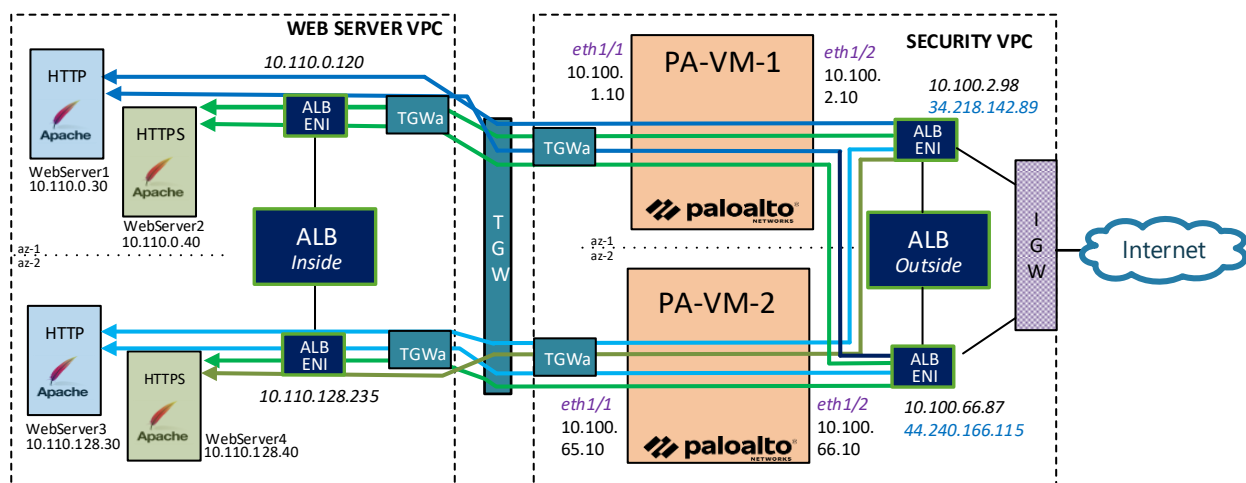


Figure 5. Routes Through the ALBs and Firewalls to the Web Servers.

Cross-zone load balancing is enabled, allowing transmission from the AZ1 ALB ENI to the firewall in AZ2, and conversely from the AZ2 ALB ENI to the AZ1 firewall. The firewalls perform source and destination NAT on the inbound packets, allowing them to traverse the TGW and Inside ALB, terminating at the web servers. **This is a key aspect of the architecture.**

The target group configurations for both ALBs and the GWLB are shown in the Appendix.

This configuration is resilient in the event of a firewall failure or a web server failure, and can scale as needed. I have captured some examples of health check packets passing through both PA-VMs, shown in figure 6.

PA-VM-1_within_AWS

DASHBOARDACCMONITORPOLICIESOBJECTSNETWORKDEVICE

Commit

le eq Inbound_Http_SecPol) or (rule eq Inbound_Https_SecPol)

	SESSION ID	INGRESS I/F	EGRESS I/F	FROM ZONE	TO ZONE	SOURCE	DESTINATION	NAT APPLIED	NAT SOURCE IP	NAT DEST IP	TO PORT	NAT PORT
	13975	ethernet1/2	ethernet1/1	public	internal	10.100.66.87	10.100.2.10	yes	10.100.1.10	10.110.0.120	443	443
	13990	ethernet1/2	ethernet1/1	public	internal	10.100.2.98	10.100.2.10	yes	10.100.1.10	10.110.0.120	80	80
	13976	ethernet1/2	ethernet1/1	public	internal	10.100.66.87	10.100.2.10	yes	10.100.1.10	10.110.0.120	80	80
	13965	ethernet1/2	ethernet1/1	public	internal	10.100.66.87	10.100.2.10	yes	10.100.1.10	10.110.128.235	443	443
	13966	ethernet1/2	ethernet1/1	public	internal	10.100.66.87	10.100.2.10	yes	10.100.1.10	10.110.128.235	80	80
	13979	ethernet1/2	ethernet1/1	public	internal	10.100.2.98	10.100.2.10	yes	10.100.1.10	10.110.0.120	80	80

×+

ure | https://35.82.153.53/?#monitor::vsys1::monitor/logs/traffic

⌂☆⚙☰

PA-VM-2_within_AWS

DASHBOARDACCMONITORPOLICIESOBJECTSNETWORKDEVICE

Commit

Q (rule eq Inbound_Http_SecPol) or (rule eq Inbound_Https_SecPol)

	SESSION ID	INGRESS I/F	EGRESS I/F	FROM ZONE	SOURCE	DESTINATION	TO ZONE	NAT APPLIED	NAT SOURCE IP	NAT DEST IP	TO PORT	NAT DEST PORT
	3637	ethernet1/2	ethernet1/1	public	10.100.66.87	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	80	80
	3617	ethernet1/2	ethernet1/1	public	10.100.2.98	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	443	443
	3618	ethernet1/2	ethernet1/1	public	10.100.2.98	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	80	80
	3626	ethernet1/2	ethernet1/1	public	10.100.66.87	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	80	80
	3606	ethernet1/2	ethernet1/1	public	10.100.2.98	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	443	443
	3607	ethernet1/2	ethernet1/1	public	10.100.2.98	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	80	80
	3626	ethernet1/2	ethernet1/1	public	10.100.66.87	10.100.66.10	internal	yes	10.100.65.10	10.110.0.120	80	80

Figure 6. Health Check Packets From the Outside ALB Traversing Both PA-VM Firewalls.

The NAT processing on both source and destination addresses is shown in Figure 6. The NAT policy in each firewall uses an address object referencing the DNS name of the internal ALB. This needs to be configured after the ALB has initialized, so cannot be preconfigured in the firewall bootstrap configuration. I am using a shell script to configure NAT after retrieving the ALB DNS name via AWS CLI.

E. IPSec Encryption For Site-to-Site VPN Access

One of the requirements for this project was supporting IPSec tunnels to provide Site-to-Site VPN access to remote sites. I implemented this using the IPSec features of the PA-VM firewalls, but this could also have been accomplished with VPN Attachments to the TGW. The customer gateway terminating the IPSec tunnels was also a PA-VM, running on a VMWare ESXi 6.7 host at my external site.

I also created an IPSec tunnel between the two PA-VMs in the security VPC. This was done as an academic exercise, as the traffic between the two firewalls is local to the VPC so inherently secure. However, this approach could be used to tunnel IPSec traffic between VPCs within AWS, or between AWS and another cloud provider hosting a PA-VM or other IPSec termination device. The tunnel configurations are shown in Figure 7.

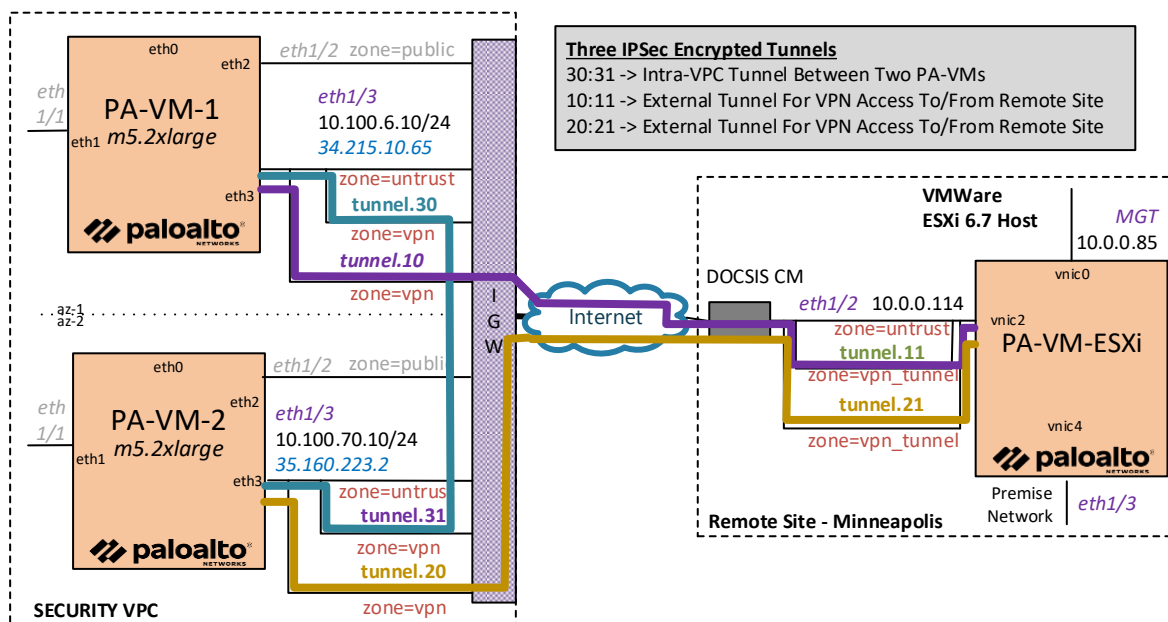


Figure 7. Three IPSec Tunnels Between Internal and External Firewalls.

Implementing and debugging the PA-VM IPSec tunnel configurations was challenging, so I have included my full configs in the Appendix. The XML config for all 3 firewalls (including the external ESXi-hosted PA-VM) are included in the IaC repository.

PA-VM-1

<https://44.224.185.157/?#network::vsys1::network/ipsec-tunnels>

PA-VM-1 within AWS

DASHBOARDACCMONITORPOLICIESOBJECTSNETWORKDEVICE

Commit

PA-VM-1 Firewall

InterfacesZonesVirtual RoutersIPSec TunnelsGRE TunnelsDHCPDNS ProxyGlobalProtect

NAME	STATUS	TYPE	IKE Gateway/Satellite				Tunnel Interface				
			INTERFACE	LOCAL IP	PEER ADDRESS	STATUS	INTERFACE	VIRTU... ROUT...	VIRTU... SYSTE...	SECURI... ZONE	STATUS
tunnel_30	Tunnel Info	Auto Key	ethernet1/3	10.100.6.10/24	10.100.70.10	IKE Info	tunnel.30	vr_vpn (Show Routes)	vsys1	vpn_tunnel	
tunnel_10	Tunnel	Auto Key	ethernet1/3	10.100.6.10/24	dynamic	IKE Info	tunnel.10	vr_vpn (Show)	vsys1	vpn_tunnel	

PA-VM

DASHBOARDACCMONITORPOLICIESOBJECTSNETWORKDEVICE

Commit

PA-VM-2 Firewall

InterfacesZonesVirtual RoutersIPSec TunnelsGRE TunnelsDHCPDNS ProxyGlobalProtect

NAME	STATUS	TYPE	IKE Gateway/Satellite				Tunnel Interface				
			INTERF...	LOCAL IP	PEER ADDRESS	STATUS	INTERF...	VIRTUAL ROUTER	VIRTUAL SYSTEM	SECURI...	STATUS
tunnel_31	Tunnel Info	Auto Key	etherne...	10.100.70.10/24	10.100...	IKE Info	tunnel.31	vr_vpn (Show Routes)	vsys1	vpn_tun...	
tunnel_20		Auto Key	etherne...	10.100.70.10/24	dynamic	IKE	tunnel.20	vr_vpn (Show)	vsys1	vpn_tun...	

PA-VM_ESXi_6_7

DASHBOARDACCMONITORPOLICIESOBJECTSNETWORKDEVICE

Commit

ESXi-Based Firewall

InterfacesZonesVLANsVirtual WiresVirtual RoutersIPSec TunnelsGRE TunnelsDHCP

NAME	STATUS	TYPE	IKE Gateway/Satellite				Tunnel Interface				
			INTERFACE	LOCAL IP	PEER ADDRESS	STATUS	INTERFA...	VIRTUAL ROUTER	VIRTUAL SYSTEM	SECURITY ZONE	STATUS
tunnel_11	Tunnel Info	Auto Key	ethernet1/2		34.215.10.65	IKE Info	tunnel.11	vr_vpn (Show Routes)	vsys1	vpn_tunnel	
tunnel_21	Tunnel	Auto Key	ethernet1/2		35.160.223.2	IKE	tunnel.21	vr_vpn (Show)	vsys1	vpn_tunnel	

Figure 8. Three IPsec Tunnels Established, Viewed From Each Firewall WebUI.

As you can see in Figure 8, all three IPsec Tunnels have been established between the two PA-VMs in the security VPC and the third in an external data center.

F. Firewalls, Management Servers, Log Collectors, and Web Servers

The primary purpose of this architecture is providing firewall-based security, both for the services and the traffic passing through it. This functionality is provided by the PAN NGFWs deployed in the security VPC. They are deployed redundantly, distributed across AZs, with duplication of traffic facilities to and from each firewall.

Firewall Interfaces: Each PA-VM has four Ethernet interfaces configured and bound to ENIs. The first is dedicated to the firewall MGT interface for the control plane. The second is placed on the internal side of the firewall to connect to internal VPCs and resources, and is configured with a sub-interface that is associated with the GWLB VPC endpoints. The third interface is placed on the external side of the firewall and used for traffic to/from the Internet. The fourth interface is also external facing, and configured with tunnel sub-interfaces for IPSec VPN connections.

Firewall Configuration: The firewall configurations are loaded via bootstrap from S3 when the instances are launched the first time. This is achieved by passing a command to the firewall via user data in the Terraform script that creates the instances. Specifically, the command is `'vmseries-bootstrap-aws-s3bucket=pavm-s3-ds/PA-VM-1'`, where the bucket name and path are the location of each firewall's XML configuration file. The association of sub-interface to the VPC endpoints, as well as the NAT configuration for the ALB, are configured by shell scripts after the rest of the environment has been provisioned.

Panorama Servers: Two Panorama servers are created in this architecture, located in the management VPC, distributed across AZs, with external Internet access via an IGW. The servers also have second internally facing interfaces to connect to the firewalls in the security VPC.

Splunk Servers: Two Splunk Enterprise servers are placed in the management VPC, distributed across AZs, and configured by a shell script after they launch. The script configures the administrator credentials, customizes the WebUI, and configures the servers to receive firewall syslog on UDP port 5514, tagging it as PAN Firewall traffic.

Images Used: The PA-VMs and the Panorama servers are launched with AWS Marketplace AMIs, configured for BYOL licensing. The Splunk servers are also launched with a Marketplace AMI that is preconfigured with a 30-day trial license.

Web Servers: The four web servers are launched with AWS linux, and configured with an Apache LAMP stack. Two are configured for HTTP, and two for HTTPS. Custom HTML files are loaded and certificate configuration is performed for the HTTPS servers via initialization scripts.

Additional configuration details are provided in the next section and the Appendix.

IV. Project Automation

A. Automation Overview

One of my project goals was to fully automate building and configuring the environment, servers, and firewalls. This was accomplished and the project is available in GitHub. Building the environment is somewhat complex, as it includes 23 Terraform scripts and 12 AWS CLI/shell scripts, with sequencing constraints and timing dependencies to build the project. Terraform manages most of the dependencies, but in some cases I wrote shell scripts that execute AWS CLI commands to achieve the desired results.

Creating the architecture is logically divided into building the environment and configuring the servers and services.

B. Building the Environment

The sequence for creating the environment is described here.

1. Create the AWS infrastructure (5 VPCs, 29 subnets, 24 route tables, NACLs, AWS security groups, 2 IGWs attached to the security and management VPCs). These entities are all created by Terraform scripts.
2. Create the S3 buckets used for configuration management and load the contents from the repository. Create the IAM role needed for the EC2s to access the buckets and attach that role to the instances as they are built. This is done with Terraform.
3. Create the AWS transport facilities (TGW, 10 TGW attachments, GWLB, GWLB endpoints, VPC service endpoints, and two layers of ALB). These are created with Terraform scripts.
4. Instantiate the (14) EC2s used by this architecture: (2) PA-VMs, (2) Panorama servers, (2) Splunk servers, (2) HTTP servers, (2) HTTPS servers, and (4) end user linux systems. These hosts are created with Terraform scripts.
5. Create the route table to subnet associations required for the environment. Terraform cannot modify route table to subnet associations without running out-of-band imports and creating dummy internal resource states. To work around this limitation, I created a shell script to make the route table associations required for the architecture. My script uses AWS CLI to retrieve the existing route table/subnet associations, then changes the associations to those that are required (by creating new associations), based on a list of input variables.
6. Interim back door access to the internal VPCs can be enabled by running scripts in the repo. This is useful to test routing and transport through the environment.

C. Configuring the Servers and Services

While Terraform works well to build the infrastructure, it is not a suitable tool for configuration management of the servers and firewalls. To completely automate this project, I relied on other methods for configuration management.

PA-VM Firewall Configuration

- The PA-VM firewall configurations are loaded into freshly created instances via the bootstrap process, by loading XML files from an S3 bucket. This is done by passing the bootstrap command with S3 bucket name via the cloud-init/user data mechanism. The XML files are fully configured for this architecture except for the next two items.
- After the PA-VMs have loaded and committed their bootstrap configurations, and the VPC Endpoint service for the GWLB has initialized, the firewall needs to be configured to allow GWLB inspection and overlay routing, and the firewall private zone sub-interface needs to be mapped to the VPCe IDs that were created in AWS. These commands must be executed with CLI on the firewalls, and are performed by a shell script.
- The final firewall configuration step requires adding a NAT rule for web traffic and ALB health checks through the firewalls. This NAT rule references an address object previously created in the firewall configuration, but must be bound to the DNS name for the internal ALB that was created. I accomplish this with a script that retrieves the ALB DNS name using AWS CLI, and then configures the firewall NAT policy via CLI.

Firewall Logging Configuration

- The PA-VM firewalls are configured with syslog server and log forwarding profiles for the Splunk servers in their bootstrap configuration files.
- The two Splunk Enterprise servers are configured by shell scripts to customize the admin credentials, the WebUI, and set up receiver profiles for UDP on port 5514 from the PA-VMs. The config scripts are copied to the Splunk servers from S3 during the launch process, and run during the first launch of their linux OS. There are some timing dependencies for this process so that it works without disrupting the cloud-init process, and these are handled by the scripts.

Web Server Configuration

- The web servers deployed in the interior run AWS linux and an Apache LAMP stack. There are two types of web server deployed: one that serves only HTTP on port 80 and the other that serves only HTTPS on port 443. The configurations are different for each server type, and include protocol-specific *index.html* files. The SSL certificate configuration is performed on the HTTPS servers. I also created a utility that runs on each instance and collects instance meta-data, storing it locally to be displayed on the index page of each web server. This helps determine which web server each service request is being provided from. Examples of this are included in the Appendix.

The web servers require temporary Internet access to complete their automated initial configurations. This is provided by a shell script that opens a back door to each web server,

allowing the configuration to proceed. The script attaches an IGW to the VPC, creates an ENI with a public IP address, and builds a temporary route table. These facilities are removed once the server configuration is complete. The HTTPS servers are configured with dummy SSL certificates to make them operational, and as you can see in the Appendix, generates a certificate warning in browsers on external systems accessing them.

Panorama Configuration

- The Panorama systems have a very limited scope in this project, primarily due to a lack of functionality in an unlicensed system. I was **not** able to obtain a Panorama demo license from PAN for this project. However, I did create the automation and deploy two Panorama servers in the management VPC.

Development Back Doors

- I use scripts to provide back door access to the web server VPC to configure the servers, and create temporary bastion hosts in the end user VPCs to access the linux systems and generate outbound traffic. Since these are closed VPCs in this architecture, this access is intended to be temporary.

RSA Key Pair Management

- AWS key pairs, consisting of a public and private key, are required when EC2s are instantiated. AWS stores the public key internally. The private key, stored externally, is used to connect to the instance from other systems. In this architecture, I create two distinct key pairs, one for the bastion hosts, and one for the rest of the EC2s (end user computers, web servers, PA-VM firewalls, Panoramas, and the SIEMs). I developed a Terraform script to generate and save the *keypair.pem* files, and the rest of my IaC scripts use this key pair to instantiate EC2s and access them via SSH. The key files are saved in the Terraform execution directory and can be copied elsewhere for SSH access to the instances.

External ESXi-Based PA-VM Firewall Configuration

- I use a PA-VM system running on an ESXi 6.7 host in my lab as the Customer Gateway to connect to the IPSec tunnels on the PA-VMs in the security VPC. The XML configuration for the remote PA-VM is included in the repo, but will need modification to run in other networks (IP addressing, VMWare Port Groups, etc.). However, the IPSec configuration may be useful for reference.

D. GitHub IaC Repository

The entire project has been automated and is stored in a repository on GitHub. Please contact the author if you are interested in access.

V. Project Summary

This section summarizes the project and my observations, and are solely the opinion of the author.

A. Observations

- The architecture functioned exactly as expected, albeit within a project of limited scale. The two-layer ALB design with PA-VMs between is a clever approach to provide resilience/redundancy for both the firewalls and the web servers. The NAT configuration on the firewalls is the key to implementing this element of the architecture.
- The architecture can be (and was) completely automated as a turn-key IaC project. This requires Terraform or CloudFormation to create the AWS infrastructure, and a configuration management approach for the servers. In this study I used shell scripts with SSH and AWS CLI, but this could also be accomplished with a configuration management tool/framework.
- This is a complex architecture to implement from scratch, and automation is very useful to build it correctly. I believe this would be a difficult architecture to troubleshoot in production without implementing other automated support tools. FWaaS products would mitigate some of these challenges. These solutions are available from multiple vendors, including Palo Alto Networks, and are built with cloud-native AWS integration.
- The PA-VM launch and bootstrap initialization required 12 minutes to complete, even with the limited firewall configurations used here. Due to this lengthy initialization, a cold-standby approach to replace a failed PA-VM might not be suitable for a production environment. PA-VMs have intrinsic HA capability, but HA failover has limitations when deployed in AWS, as documented by Palo Alto Networks. Given these considerations, a backup strategy for the PA-VMs needs additional investigation.
- My use of Panorama was limited in this project, primarily due to a lack of functionality with an unlicensed system. However, and seemingly unrelated to licensing, the launch and initialization of a virtual Panorama instance was **very** slow, approaching 20 minutes. This was using PAN's recommended EC2 instance profile and a default server configuration. A better approach may be using an external hardware-based Panorama system to manage the AWS-hosted firewalls.
- IPSec Tunnel configuration on the PA-VMs was difficult to create and troubleshoot, requiring log parsing and running debug commands. Once configured correctly it functioned as expected, but the PA-VM product would benefit from a more useful WebUI/IPSec interface with access to underlying counters and other information.
- The mapping of sub-interfaces to GWLB/VPC Endpoints must be done in the PA-VM CLI, after the endpoints have initialized. I used a script to retrieve the VPCe IDs with AWS CLI and configure them on the PA-VMs. This worked here but may not be a good operational practice. I suggest that an API-driven approach between AWS and the PA-VMs would be a better architecture.

B. Caveats and Additional Notes

- The AWS Security Groups, NACLs, and PA-VM Security Policies were configured for development and testing purposes, and should be tightened up significantly for use in a production environment.
- The DNS name for the Inside ALB was dynamically assigned when the ALB is created, and must be populated in the firewall NAT rules. I created a script to make this change via SSH, but the ALB name could be configured with a static name hosted in a Route 53 PHZ, providing a better approach.
- The web servers require Internet connectivity to complete their configuration after they are launched. I solved this with a script that creates a temporary IGW and binds a second ENI to the instances, allowing the configuration to proceed. This must be executed soon after the EC2s are created so the process completes before the cloud-init process times out.
- The Palo Alto Networks PA-VMs used were unlicensed, using a BYOL AMI, for the sole purpose of evaluating their use in this architecture. Panorama also used a BYOL image, had limited functionality as an unlicensed system, and was not used to configure the firewalls. I relied on Splunk for logging and other automated methods for configuration of the firewalls and servers in this architecture.
- The PA-VMs support decryption of SSL/TLS traffic outbound (SSL Forward Proxy) and inbound (SSL Inbound Inspection), plus decryption of SSH traffic (SSH Proxy). SSL decryption requires setting up signed certificates and an enterprise PKI. I've left this as an exercise for the reader . . .
- The servers were not configured with SSL certificates, as a result posting cert warnings on the WebUI for PA-VM, Panorama, and the HTTPS servers. This is not an issue with the Splunk servers, as their WebUI was accessed via HTTP.
- The AWS costs to run this environment can be significant for a proof-of-concept effort. The AWS environment used was that of the author, and expenses incurred borne by him alone. The build time is approximately 45 minutes, including the initialization and configuration of the servers. The Oregon AWS Region was used to build this project, with IPSec connections to my lab in Minneapolis.
- The certificates used to create HTTPS target groups in both ALBs were the author's privately owned CA-signed certificates, and are not available for use with the GitHub IaC repo.

Appendix

This appendix provides additional configuration details and screen captures for selected features.

The IPSec configuration parameters used are listed in Tables A1 through A4.

<u>IKE Gateways For Firewall: PA-VM-ESXi</u>		
Name:	<u>IKEGW_ESXi1_to_PAVM1</u>	<u>IKEGW_ESXi1_to_PAVM2</u>
Version:	IKEv2 preferred	IKEv2 preferred
Interface:	ethernet1/2	ethernet1/2
Local IP:	none	none
Peer IP:	IP/34.215.10.65	IP/ 35.160.223.2
Pre-shared Key:	DSBLUE	DSBLUE
Local ID:	IP/10.0.0.12	IP/10.0.0.12
Peer ID:	IP/34.215.10.65	IP/ 35.160.223.2
NAT (IKEv1/IKEv2):	Enabled/Enabled	Enabled/Enabled
<u>IKE Gateways For Firewall: PA-VM-1</u>		
Name:	<u>IKEGW_to_ESXi1</u>	<u>IKEGW_IntraVPC_PAVM1</u>
Version:	IKEv2 preferred	IKEv2 preferred
Interface:	ethernet1/3	ethernet1/3
Local IP:	10.100.6.10	10.100.6.10
Peer IP:	Dynamic	IP/10.100.70.10
Pre-shared Key:	DSBLUE	DSBLUE
Local ID:	IP/54.191.232.182	IP/10.100.6.10
Peer ID:	IP/10.0.0.12	IP/10.100.70.10
NAT (IKEv1/IKEv2):	Enabled/Enabled	Enabled/Enabled
<u>IKE Gateways For Firewall PA-VM-2</u>		
Name:	<u>IKEGW_to_ESXi1</u>	<u>IKEGW_IntraVPC_PAVM2</u>
Version:	IKEv2 preferred	IKEv2 preferred
Interface:	ethernet1/3	ethernet1/3
Local IP:	10.100.70.10	10.100.70.10
Peer IP:	Dynamic	IP/10.100.6.10
Pre-shared Key:	DSBLUE	DSBLUE
Local ID:	IP/52.88.202.63	IP/10.100.70.10
Peer ID:	IP/10.0.0.13	IP/10.100.6.10
NAT (IKEv1/IKEv2):	Enabled/Enabled	Enabled/Enabled

Table A1. IKE Gateway Configurations.

<u>Ethernet Interface Configs:</u>			
Firewall:	<u>PA-VM-ESXi</u>	<u>PA-VM-1</u>	<u>PA-VM-2</u>
Interface:	ethernet1/2	ethernet1/2	ethernet1/2
V Router:	vr_vpn	vr_vpn	vr_vpn
Zone:	untrust	untrust	untrust
IPv4:	10.0.0.114 (dhcp)	10.100.6.10	10.100.70.10
Int Mgmt Prof:	Allow_Ping	Allow_Ping	Allow_Ping

Table A2. Ethernet Interfaces Used For IPSec Tunnels.

Tunnel Interface Configs:						
Firewall:	PA-VM-ESXi	PA-VM-ESXi	PA-VM-1	PA-VM-1	PA-VM-2	PA-VM-2
Tunnel:	tunnel.11	tunnel.21	tunnel.10	tunnel.30	tunnel.20	tunnel.31
V Router:	vr_vpn	vr_vpn	vr_vpn	vr_vpn	vr_vpn	vr_vpn
Zone:	vpn_tunnel	vpn_tunnel	vpn_tunnel	vpn_tunnel	vpn_tunnel	vpn_tunnel
IPv4:	3.1.1.11/24	4.1.1.11/24	3.1.1.10/24	2.1.1.30/24	4.1.1.10/24	2.1.1.31/24
Int Mgt Prof:	Allow_Ping	Allow_Ping	Allow_Ping	Allow_Ping	Allow_Ping	Allow_Ping

Table A3. IPSec Tunnel Interface Configurations.

<u>IPSec Tunnel Pairs:</u>		
<u>First External Tunnel : Tunnel 11 to Tunnel 10 (PA-VM-ESXi to PA-VM-1)</u>		
	PA-VM-ESXi	PA-VM-1
Name:	tunnel_11	tunnel_10
Tunnel Intf:	tunnel.11	tunnel.10
Type:	Auto Key	Auto Key
Addr Type:	IPv4	IPv4
IKE GW:	IKEGW_ESXi1_to_PAVM1	IKEGW_to_ESXi1
IPSec Crypto:	Suite-V-GCM-128	Suite-V-GCM-128
<u>Second External Tunnel : Tunnel 21 to Tunnel 20 (PA-VM-ESXi to PA-VM-2)</u>		
	PA-VM-ESXi	PA-VM-2
Name:	tunnel_21	tunnel_20
Tunnel Intf:	tunnel.21	tunnel.20
Type:	Auto Key	Auto Key
Addr Type:	IPv4	IPv4
IKE GW:	IKEGW_ESXi1_to_PAVM2	IKEGW_to_ESXi1
IPSec Crypto:	Suite-V-GCM-128	Suite-V-GCM-128
<u>(VPC to VPC) Internal Tunnel : Tunnel 31 to Tunnel 30 (PA-VM-2 to PA-VM-1)</u>		
	PA-VM-2	PA-VM-1
Name:	tunnel_31	tunnel_30
Tunnel Intf:	tunnel.31	tunnel.30
Type:	Auto Key	Auto Key
Addr Type:	IPv4	IPv4
IKE GW:	IKEGW_ESXi1_to_PAVM2	IKEGW_IntraVPC_PAVM1
IPSec Crypto:	Suite-V-GCM-128	Suite-V-GCM-128

Table A4. IPSec Tunnel Configurations For Each Firewall.

To initiate IPSec connections for testing, execute the 'test vpn ike-sa gateway' command from the PA-VM firewall command line. This is illustrated on the external ESXi-hosted PA-VM in Figure A1. The same command is used to initiate the other IPSec connections in this architecture.

```

Select OpenSSH SSH client

admin@PA-VM_ESXi_6_7> test vpn ike-sa gateway IKEGW_ESXI1_to_PAVM2

Start time: Mar.28 10:33:26
Initiate 1 IKE SA.

admin@PA-VM_ESXi_6_7> show vpn ike-sa gateway IKEGW_ESXI1_to_PAVM2

There is no IKEv1 phase-1 SA found.

There is no IKEv1 phase-2 SA found.

IKEv2 SAs
Gateway ID      Peer-Address      Gateway Name
Role SN         Algorithm         Established      Expiration      Xt Child ST
-----
-----
2              35.155.127.130    IKEGW_ESXI1_to_PAVM2
Init 4          PSK/DH19/A128/SHA256 Mar.28 10:33:26 Mar.28 18:33:26 0 1      Established

IKEv2 IPSec Child SAs
Gateway Name
-----
ID      Parent  Role SPI(in)  SPI(out)  Tunnel  MsgID  ST
-----
-----
IKEGW_ESXI1_to_PAVM2
6        4      Init BC92FBD8 9CD460FF 2      tunnel_21 00000001 Mature

Show IKEv2 SA: Total 2 gateways found. 1 ike sa found.

admin@PA-VM_ESXi_6_7>

```

Figure A1. Initiating an IPSec Tunnel Between the External PA-VM and the PA-VM-2 Firewall Within the Security VPC on AWS.

Table A5 lists the IP addressable resources for the entire architecture, by subnet and VPC. Italicized IP addresses are dynamically assigned. Color coding denotes AWS transport entities.

VPC	Subnet	AZ	Subnet Name	Resource	Type	Private IP	Pub IP	Notes
Usr01	10.104.0.0/24	1	usr1-az1-inst	usr1-az1	EC2	10.104.0.20 (eth0)		End user linux
	10.104.1.0/24	1	usr1-az1-TGW	TGW Attachment	ENI	10.104.1.212		TGWA (AZ1)
	10.104.3.0/24	1	usr1-az1-bastion	bastion-host1	EC2	10.104.3.16 (eth0)	dyn IP	Temp - Dev Only
	10.104.128.0/24	2	usr1-az2-inst	usr1-az2	EC2	10.104.128.20 (eth0)		End user linux
	10.104.129.0/24	2	usr1-az2-TGW	TGW Attachment	ENI	10.104.129.168		TGWA (AZ2)
Usr02	10.105.0.0/24	1	usr2-az1-inst	Usr2-az1	EC2	10.105.0.20 (eth0)		End user linux
	10.105.1.0/24	1	usr2-az1-TGW	TGW Attachment	ENI	10.105.1.94		TGWA (AZ1)
	10.105.3.0/24	1	usr2-az1-bastion	bastion-host2	EC2	10.105.3.16 (eth0)	dyn IP	Temp - Dev Only
	10.105.128.0/24	2	usr2-az2-inst	Usr2-az2	EC2	10.105.128.20 (eth0)		End user linux
	10.105.129.0/24	2	usr2-az2-TGW	TGW Attachment	ENI	10.105.129.148		TGWA (AZ2)
Mgmt	10.255.0.0/24	1	mgmt-az1-inst	Panorama-1	EC2	10.255.0.10 (eth0)		Panorama1
	"		"	Splunk-1	EC2	10.255.0.30 (eth0)		Splunk1
	10.255.1.0/24	1	mgmt-az1-pub	Panorama-1	EC2	10.255.1.10 (eth1)	dyn IP	Panorama1 2 nd int
	"		"	Splunk-1	EC2	10.255.1.30 (eth1)	dyn IP	Splunk1 2 nd int
	10.255.2.0/24	1	mgmt-az1-TGW	TGW Attachment	ENI	10.255.2.207		TGWA (AZ1)
	10.255.128.0/24	2	mgmt-az2-inst	Panorama-2	EC2	10.255.128.10 (eth0)		Panorama2
	"		"	Splunk-2	EC2	10.255.128.30 (eth0)		Splunk2
	10.255.129.0/24	2	mgmt-az2-pub	Panorama-2	EC2	10.255.129.10 (eth1)	dyn IP	Panorama2 2 nd int
	"		"	Splunk-2	EC2	10.255.129.30 (eth1)	dyn IP	Splunk2 2 nd int
Web Srv	10.255.130.0/24	2	mgmt-az2-TGW	TGW Attachment	ENI	10.255.130.38		TGWA (AZ2)
	10.110.0.0/24	1	websrv-az1-inst	websrv1-p80-az1	EC2	10.110.0.30 (eth0)		Apache http only
			"	websrv2-p443-az1	EC2	10.110.0.40 (eth0)		Apache https only
			"	ALB-inside-ENI-az1	ENI	10.110.0.120	None	ALB ENI
	10.110.1.0/24	1	websrv-az1-TGW	TGW Attachment	ENI	10.110.1.247		TGWA (AZ1)
	10.110.2.0/24	1	resv
	10.110.128.0/24	2	websrv-az2-inst	websrv3-p80-az2	EC2	10.110.128.30 (eth0)		Apache http only
			"	websrv4-p443-az2	EC2	10.110.128.40 (eth0)		Apache https only
			"	ALB-inside-ENI-az2	ENI	10.110.128.235	None	ALB ENI
Sec01	10.110.129.0/24	2	websrv-az2-TGW	TGW Attachment	ENI	10.110.129.142		TGWA (AZ2)
	10.110.130.0/24	2	resv
	10.100.0.0/24	1	sec-az1-mgt	PA-VM-1	EC2	10.100.0.10 (eth0)	dyn IP	PA-VM-1: Mgt
	10.100.1.0/24	1	sec-az1-int	PA-VM-1	EC2	10.100.1.10 (eth1)		PA-VM-1: Private
	10.100.2.0/24	1	sec-az1-pub	PA-VM-1	EC2	10.100.2.10 (eth2)	dyn IP	PA-VM-1: Public
	"		"	ALB-Outside-ENI	ENI	10.100.2.98	dyn IP	ALB ENI
	10.100.3.0/24	1	sec-az1-TGW_att	TGW Attachment	ENI	10.100.3.104		TGWA (AZ1)
	10.100.4.0/24	1	sec-az1-GWLBep	ENI-gwlb-ep	ENI	10.100.4.136		Health Check Src
	10.100.5.0/24	1	sec-az1-GWLB	ENI-gwlb	ENI	10.100.5.136		GWLB EP ENI
	10.100.6.0/24	1	sec-az1-vpn	PA-VM-1 (VPN)	ENI	10.100.6.15 (eth3)	dyn IP	PA-VM-1: VPN
	10.100.64.0/24	2	sec-az2-mgt	PA-VM-2	EC2	10.100.64.10 (eth0)	dyn IP	PA-VM-2: Mgt
	10.100.65.0/24	2	sec-az2-int	PA-VM-2	"	10.100.65.10 (eth1)		PA-VM-2: Private
	10.100.66.0/24	2	sec-az2-pub	PA-VM-2	"	10.100.66.10 (eth2)	dyn IP	PA-VM-2: Public
	"		"	ALB-Outside-ENI		10.100.66.144	dyn IP	ALB ENI
	10.100.67.0/24	2	sec-az2-TGW_att	TGW Attachment	ENI	10.100.67.241		TGWA (AZ2)
	10.100.68.0/24	2	sec-az2-GWLBep	ENI-gwlb-ep	ENI	10.100.68.180		GWLB EP ENI
	10.100.69.0/24	2	sec-az2-GWLB	ENI-gwlb	ENI	10.100.69.114		Health Check Src
	10.100.70.0/24	2	sec-az1-vpn	PA-VM-2 (VPN)	ENI	10.100.70.15 (eth3)	dyn IP	PA-VM-2: VPN

Table A5. VPCs, Subnets, and Resource Mapping.

The web servers used in this architecture are configured with two different profiles, serving either HTTP (port 80) or HTTPS (port 443) exclusively. Servers providing both are distributed across AZs in the web server VPC. The web servers are configured by the project automation, and display instance meta-data on their home page as seen in Figure A2. These screen captures are from Chrome on an external Win10 system, accessing them through the centralized security architecture.

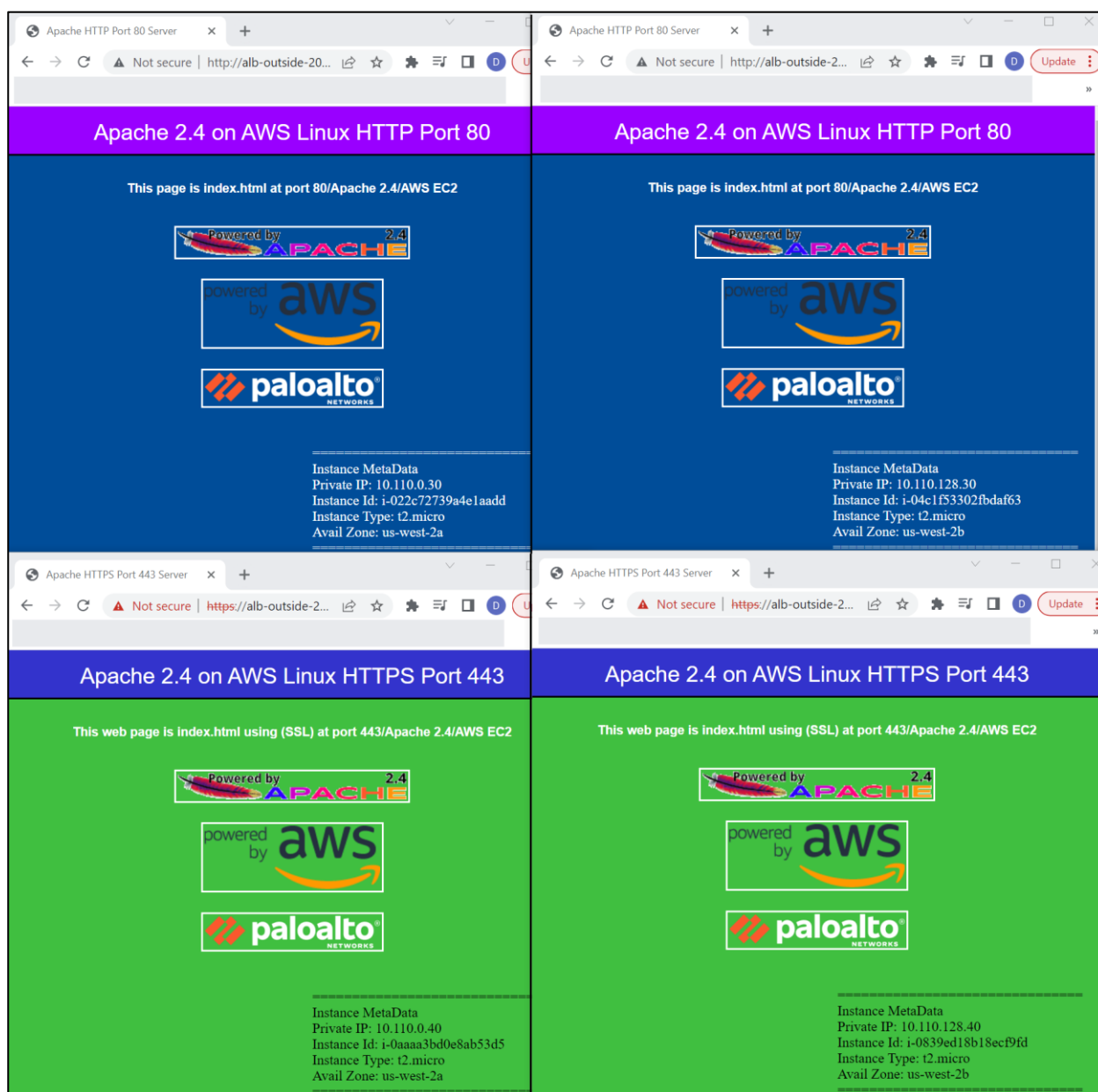


Figure A2. Four Web Servers in Web Server VPC Accessed Via the Internet Through the Security VPC, Dual ALBs, and PA-VM Firewalls.

The WebUI from both EC2-based Panorama systems deployed into the management VPC are shown in Figure A3.

The figure displays two screenshots of the Palo Alto Networks Panorama WebUI, showing the dashboard for two different Panorama systems.

Top Screenshot (Panorama-1): The URL is <https://34.208.202.173/#dashboard::>. The dashboard shows the following information:

- General Information:**
 - Device Name: Panorama
 - Public IP Address: unknown
 - MGT IP Address: 10.255.1.10
 - MGT Netmask: 255.255.255.0
 - MGT Default Gateway: 10.255.1.1
 - MGT IPv6 Address: unknown
 - MGT IPv6 Link Local Address: fe80::3b:85ff:fe33:3003/64
- Logged In Admins:**

Admin	From	Client	Session Start	Idle For
admin	73.94.124.109	Web	03/26 20:27:34	00:00:00s
- Data Logs:** No data available.
- System Logs:** No data available.
- Config Logs:**

Command	Path	Admin	Time
add	config mgt-config devices	admin	03/26 20:37:07
commit		admin	03/26 20:32:39
add	device-group	admin	03/26 20:30:26
edit	deviceconfig system device-telemetry	admin	03/26 20:27:58

Bottom Screenshot (Panorama-2): The URL is <https://44.232.95.20/#dashboard::>. The dashboard shows the following information:

- General Information:**
 - Device Name: Panorama
 - Public IP Address: unknown
 - MGT IP Address: 10.255.129.10
 - MGT Netmask: 255.255.255.0
 - MGT Default Gateway: 10.255.129.1
 - MGT IPv6 Address: unknown
 - MGT IPv6 Link Local Address: fe80::4db:6ff:fe94:6235/64
 - MGT IPv6 Default Gateway: 06:db:06:94:62:35
 - Model: Panorama
 - Serial #: unknown
- Logged In Admins:**

Admin	From	Client	Session Start	Idle For
admin	35.91.152.92	CLI	03/26 20:28:25	00:07:56s
admin	73.94.124.109	Web	03/26 20:33:23	00:00:00s
- Data Logs:** No data available.
- System Logs:**

Description	Time
Connection to Update server: updates.paloaltonetworks.com completed successfully, initiated by 10.255.129.10	03/26 20:36:34
- Config Logs:**

Command	Path	Admin	Time
add	device-group	admin	03/26 20:34:36
edit	deviceconfig system device-telemetry	admin	03/26 20:33:43
commit		admin	03/26 20:29:12
set	config mgt-config users admin	admin	03/26 20:29:07
- Locks:** No locks found.

Figure A3. Panorama-1 and Panorama-2 Systems Accessed Via WebUI.

Two Splunk Enterprise servers are deployed in the management VPC. They are configured to receive syslog messages from both PA-VM firewalls on UDP port 5514 and tag them as *pan:firewall*. The PA-VM firewalls are configured with syslog server profiles for the Splunk systems, and log forwarding profiles. Log captures from both firewalls on both Splunk servers are shown in Figure A4.

The image shows two side-by-side Splunk web interfaces. The left interface is labeled 'Splunk-1' and the right is 'Splunk-2'. Both interfaces display a table of syslog messages. The table has columns for event number, time, IP host, source, and raw data. The messages are from PA-VM firewalls on UDP port 5514.

* _time	IP host	a sou...	a sou...	> _raw
1 2023-03-26T19:42:26.000Z	10.100.0.10	udp:5514	pan:firewa 11	Mar 26 19:42:26 PA-VM-1 1,2019:42:26,10.100.66.87,10.100.1,public,internal,ethernet1,19:42:26,2189,1,14334,80,765
2 2023-03-26T19:42:26.000Z	10.100.0.10	udp:5514	pan:firewa 11	Mar 26 19:42:26 PA-VM-1 1,2019:42:26,10.100.66.87,10.100.1,public,internal,ethernet1,19:42:26,2190,1,3940,443,352
3 2023-03-26T19:42:25.000Z	10.100.64.10	udp:5514	pan:firewa 11	Mar 26 19:42:25 PA-VM-2 1,2019:42:25,10.100.2.98,10.100.1,public,internal,ethernet1,10:42:25,2166,1,15724,80,52

Figure A4. Splunk Servers Showing Syslog Received From Both PA-VMs.

Figures A5 and A6 show the target groups for the load balancers deployed in this architecture. Figure A5 shows the target groups for the two ALBs: ALB-Outside and ALB-Inside. Note the target IP addresses for the outside ALB are the public side private IP addresses of the firewalls. NAT is configured on the firewalls to translate the destination IP address to that of the inside ALB, and the source IP address to the private side private IP address of the firewall. This allows packets through the outside ALB to traverse the firewall, TGW, internal ALB, and reach the web servers.

The screenshot displays two panels from the AWS Management Console, showing the configuration of target groups for two Application Load Balancers (ALBs) in the Oregon region.

Top Panel: ALB-Inside-TG-Group443

IP address	Port	Zone	Health status
10.110.128.40	443	us-west-2b	healthy
10.110.0.40	443	us-west-2a	healthy

Bottom Panel: ALB-Outside-TG-Group443

IP address	Port	Zone	Health status
10.100.66.10	443	us-west-2b	healthy
10.100.2.10	443	us-west-2a	healthy

Right Panel: ALB-Inside-TG-Group80

IP address	Port	Zone	Health status
10.110.0.30	80	us-west-2a	healthy
10.110.128.30	80	us-west-2b	healthy

Bottom Right Panel: ALB-Outside-TG-Group80

IP address	Port	Zone	Health status
10.100.2.10	80	us-west-2a	healthy
10.100.66.10	80	us-west-2b	healthy

Figure A5. ALB Load Balancer Target Groups (Both ALBs).

Figure A6 shows the target group for the Gateway Load Balancer. This is used for outbound traffic and east-west traffic. Notice that the IP addresses are the IP addresses of the internal firewall interfaces (ethernet 1/1). NAT is performed in the firewalls on these packets, and for outbound, they traverse to the public side the firewall, and are again NAT'ed by the IGW on the outside of the security VPC. As mentioned previously, the firewalls use the VPCe ID, mapped to the ethernet 1/1.1 sub-interface for the Geneve inspection and overlay routing features of the firewalls.

EC2 > Target groups > PAVMTargetGroup

PAVMTargetGroup

Details

arn:aws:elasticloadbalancing:us-west-2:500112433998:targetgroup/PAVMTargetGroup/00d59f12e2df1bb721

Target type IP	Protocol : Port GENEVE: 6081	VPC vpc-057a810fcbbad7741	Load balancer PAVM-GWLB
-------------------	---------------------------------	--	--

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
2	2	0	0	0	0

► **Distribution of targets by Availability Zone (AZ)**
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (2)

Filter resources by property or value

	IP address	Port	Zone	Health status	Health status details
<input type="checkbox"/>	10.100.1.10	6081	us-west-2a	healthy	
<input type="checkbox"/>	10.100.65.10	6081	us-west-2b	healthy	

Figure A6. ALB Load Balancer Target Groups (Both ALBs).

Figure A7 shows running the shell script that configures the associations between route tables and subnets required for this architecture. This is used as a work around to a Terraform limitation with their AWS VPC module.

```

[cloudshell-user@ip-10-6-93-206 PAN-GWLB5]$ ./RT-Associations.sh
-----
Subnet<->Input Route Table Associations To Change
-----
#      Subnet      Orig-RT      New-RT
--      -
0.  sec-az1-pub : Sec01-VPC-intra -> Secvpc-public-subnets-RT
1.  sec-az2-pub : Sec01-VPC-intra -> Secvpc-public-subnets-RT
2.  sec-az1-mgt : Sec01-VPC-intra -> Secvpc-mgmt-subnets-RT
3.  sec-az2-mgt : Sec01-VPC-intra -> Secvpc-mgmt-subnets-RT
4.  sec-az1-int : Sec01-VPC-intra -> Secvpc-private-subnets-RT
5.  sec-az2-int : Sec01-VPC-intra -> Secvpc-private-subnets-RT
6.  app1-az1-inst : App01-VPC-intra -> App1-instances-RT
7.  app1-az2-inst : App01-VPC-intra -> App1-instances-RT
8.  app2-az1-inst : App02-VPC-intra -> App2-instances-RT
9.  app2-az2-inst : App02-VPC-intra -> App2-instances-RT
10. mgmt-az1-int : Mgmt-VPC-intra -> Mgmt-private-subnets-RT
11. mgmt-az2-int : Mgmt-VPC-intra -> Mgmt-private-subnets-RT
12. mgmt-az1-pub : Mgmt-VPC-intra -> Mgmt-public-subnets-RT
13. mgmt-az2-pub : Mgmt-VPC-intra -> Mgmt-public-subnets-RT
14. sec-az1-TGW Att : Sec01-VPC-intra -> devonly_secvpc_tgwatt-az1-subnet
15. sec-az2-TGW Att : Sec01-VPC-intra -> devonly_secvpc_tgwatt-az2-subnet
16. webserv-az1-inst : WebSrv-VPC-intra -> WebSrv-subnets-RT
17. webserv-az2-inst : WebSrv-VPC-intra -> WebSrv-subnets-RT
-----
AWSCLI Query Results->False rtbassoc-08fc9e1dce99ddab0 rtb-0e261c56dbe654656 subnet-0000fefe8b876ea91 ASSOCIATIONSTATE associated
AWSCLI Query Results->False rtbassoc-06898164adb46f087 rtb-0e261c56dbe654656 subnet-057254e31e9a2444c ASSOCIATIONSTATE associated
AWSCLI Query Results->False rtbassoc-03bc2b881cc2ba286 rtb-0e261c56dbe654656 subnet-0722890a0bc3ca588 ASSOCIATIONSTATE associated
AWSCLI Query Results->False rtbassoc-04f364e95d861750e rtb-0e261c56dbe654656 subnet-085d6b2ee5e3e96d7 ASSOCIATIONSTATE associated
AWSCLI Query Results->False rtbassoc-03a89d97560824832 rtb-0e261c56dbe654656 subnet-00dc076e726d86353 ASSOCIATIONSTATE associated
AWSCLI Query Results->False rtbassoc-0989bffb9fc078802 rtb-0e261c56dbe654656 subnet-02ca92e99b8f3c983 ASSOCIATIONSTATE associated

```

Figure A7. Creating the Correct Route Table Associations to Subnets via Script.

I hope you enjoyed reading about this AWS middlebox security implementation with Palo Alto Networks firewalls. Please reach out if you have questions or comments.

Dan Edeen, dan@edeem.com