Multi-Cloud Security Automation Lab Documentation

Palo Alto Networks

Sep 30, 2020

overview

CHAPTER 1

Welcome

Welcome to the Multi-Cloud Security Automation Lab!

In this lab we will be learning how to automate the deployment and configuration of infrastructure supporting a web application within a public cloud provider. A key element of this infrastructure is the Palo Alto Networks NGFW.

Following the deployment, we will automate the configuration of the firewall to support and protect protect the web application.

Lastly, we will ensure that the firewall is able to respond effectively to changes made to the application infrastructure. You will have your choice of deploying your application in Google Cloud Platform (GCP), Amazon Web Services (AWS) or both if time permits.

CHAPTER 2

Objective

The objective of this workshop is to deploy and secure a WordPress content management system in GCP and AWS. This web application will be supported by an Apache web server and a MariaDB database server residing in two separate subnets.

As part of our infrastructure deployment, a VM-Series NGFW will be inserted between the untrusted public subnet, the web subnet, and the database subnet. However, we will need to configure this virtual firewall to support its network environment and the applications it will be protecting.

CHAPTER 3

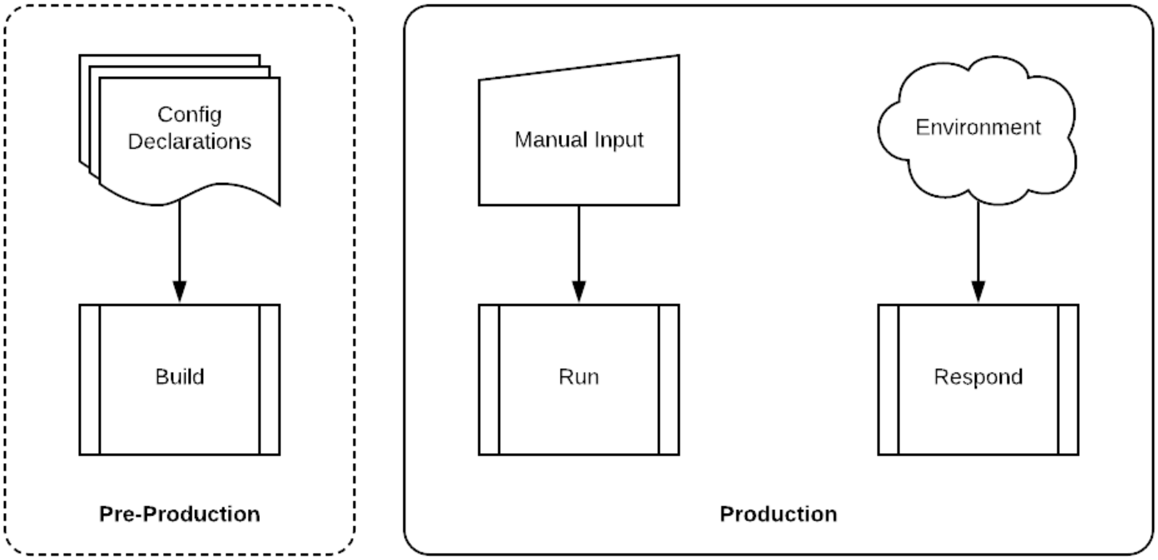
Learning Outcomes

- Understand the various methods for automating the deployment of Palo Alto Networks NGFW instances in cloud environments
- Learn to use industry-leading configuration management automation tools to implement changes to PAN-OS devices
- Learn how the Palo Alto Networks NGFW can automatically respond to changes in the network environment

3.1 Introduction

3.1.1 Automation Overview

This training workshop provides hands-on exposure to the three primary categories of infrastructure automation activities: **Build**, **Run**, and **Respond**.



Build Build automation is the means by which a set of infrastructure elements are declared, instantiated, and orchestrated using automation tools and infrastructure APIs. The result is a set of deployed infrastructure elements that are in production (or production-ready) with a "day one" configuration.

Run Run automation encompasses any API-based configuration management actions that occur once the infrastructure element is in production. These are primarily scheduled changes that are made to support new requirements. The input to these changes is manually defined in a variety of formats such as YAML, JSON, XML, etc.

Respond Response automation includes any automated actions that are triggered by an event. These may be operational events such as changes to the infrastructure or security events such as of a new threat. Response actions are defined in advance but only initiated when a event matching its trigger criteria occurs.

3.1.2 Lab Topology

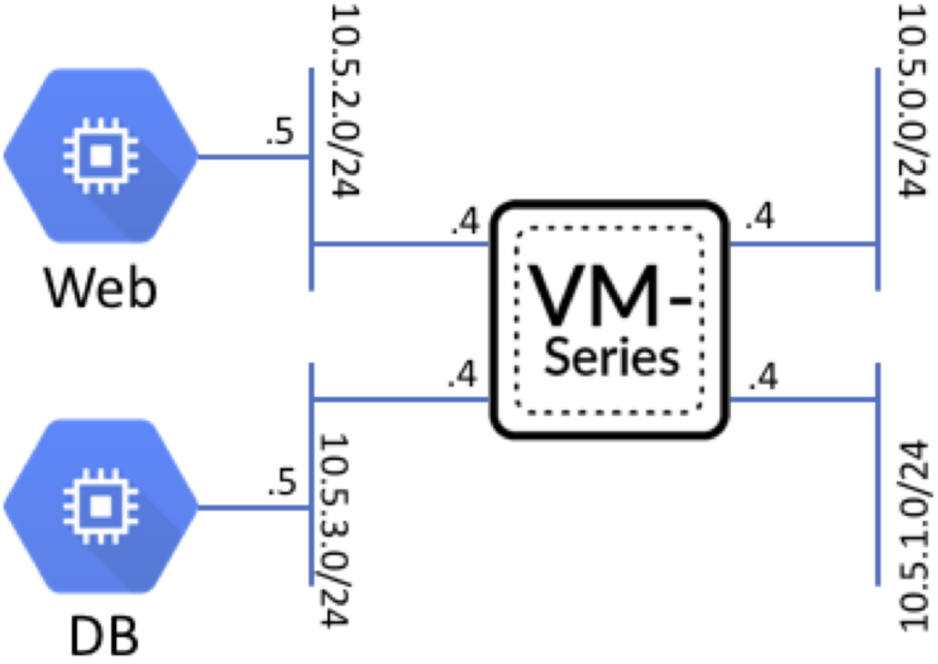
Subnet	Address	Interface
Management	10.5.0.0/24	Management
Untrust	10.5.1.0/24	ethernet1/1
Web	10.5.2.0/24	ethernet1/2
Database	10.5.3.0/24	ethernet1/3

3.1.3 Lab Components

Qwiklabs This lab is launched using Qwiklabs, which is an online learning platform that deploys and provides access to cloud-based lab environments. Qwiklabs will establish a set of temporary set of credentials in the cloud provider in order to deploy and access the cloud infrastructure and services.

Launchpad VM A Debian 9 Linux virtual machine will be deployed in each cloud environment for you to use as your primary workspace for the lab activities, This VM will be provisioned with all the tools and libraries necessary for deploying and managing infrastructure in the cloud provider.

Hashicorp Terraform Each cloud provider offers a mechanism that allow you to define a set of infrastructure element or services and orchestrate their instantiation. However, these tools and templates are specific to each cloud


Web
DB
.5
.5
10.5.2.0/24
10.5.3.0/24
.4
.4
VM-Series
.4
.4
10.5.0.0/24
10.5.1.0/24

provider. We will be using Terraform to perform this function as it provides a common set of capabilities and a template formats acroos all cloud providers.

Red Hat Ansible Whereas Terraform excels at orchestrating deployment activities, Ansible is more effective at automating configuration management tasks. We will be using both Terraform and Ansible to make configuration changes to the VM-Series firewall in order to illustrate their different capabilities.

Google Cloud Platform (GCP) Google Cloud Platform, offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search and YouTube.

Amazon Web Services (AWS) Amazon Web Services is a subsidiary of Amazon that provides on-demand cloud computing platforms to individuals, companies and governments, on a metered pay-as-you-go basis.

3.2 Requirements

The following are requirements of this training workshop:

- A laptop with Internet connectivity (SSH and HTTPS access is required).
- A standards-compliant web browser (Google Chrome is recommended).
- An SSH client (e.g., OpenSSH, PuTTY, SecureCRT, etc).
- An understanding of Linux operating system basics.
- Proficiency with a Linux text editor such as vim or nano.
- A basic understanding of cloud computing concepts.

3.3 Setup

In this activity you will:

- Log into the Qwiklabs portal
- Launch the GCP or AWS Lab
- SSH into the Launchpad VM
- Clone the lab software repository

Warning: Before you start it is recommended that you launch a private instance of your web browser. This will prevent the use of cached Google or Amazon credentials if you log into the GCP or AWS consoles. This will help ensure you do not incur any personal charges within these cloud providers.

3.3.1 Log into the Qwiklabs portal

Navigate to the Qwiklabs URL in your web browser.

```
https://paloaltonetworks.qwiklabs.com
```

Log in with your Qwiklabs credentials (sign up if you are new to Qwiklabs). You must use your corporate email address for the username.

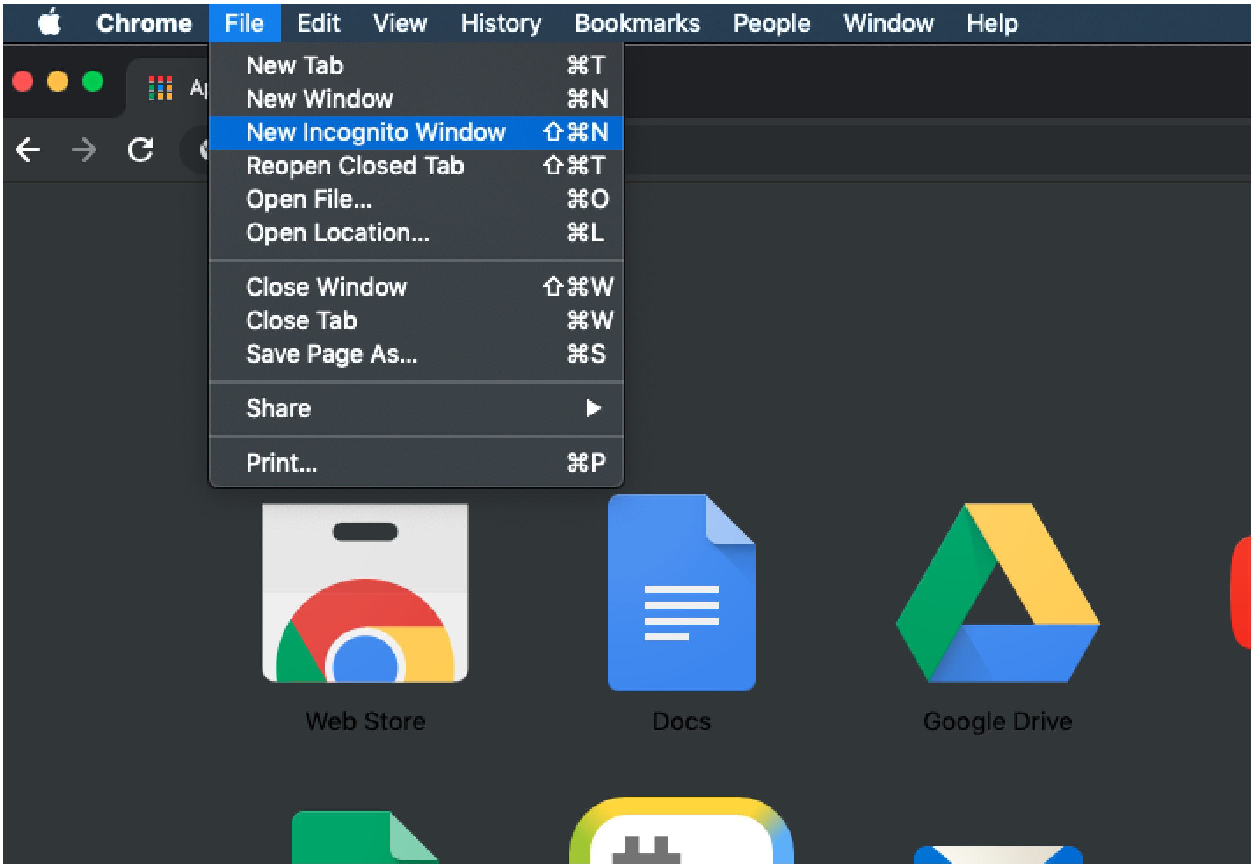
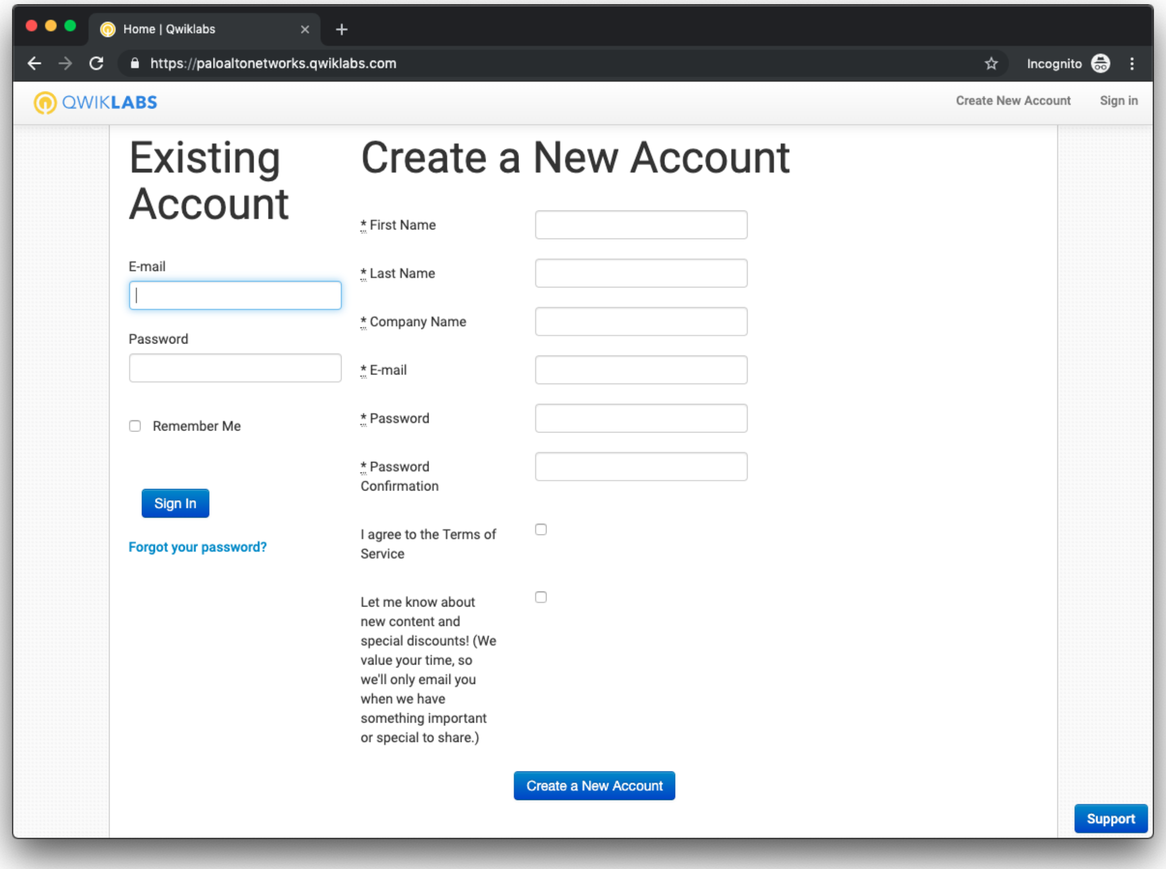


Fig. 1: Chrome Incognito mode



Existing Account

E-mail

Password

□ Remember Me

Create a New Account

* First Name

* Last Name

* Company Name

* E-mail

* Password

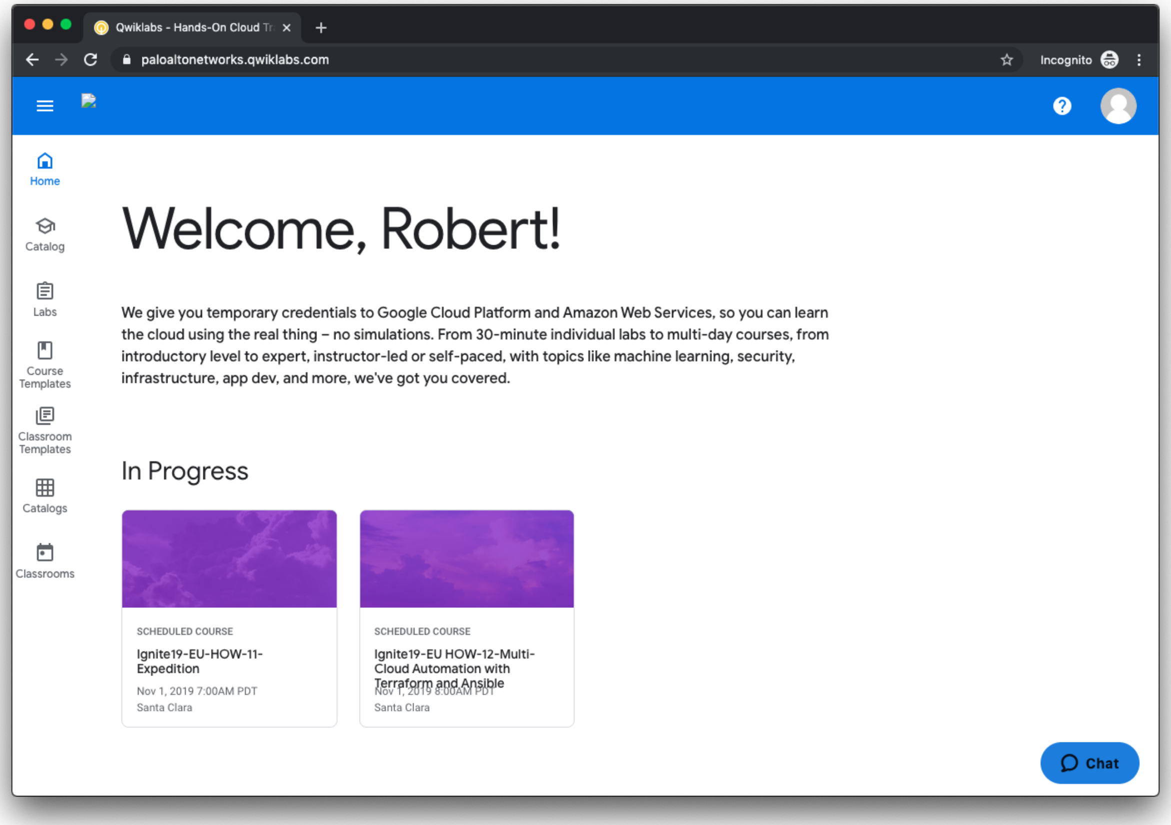
* Password Confirmation

I agree to the Terms of Service □

Let me know about new content and special discounts! (We value your time, so we'll only email you when we have something important or special to share.) □

3.3.2 Launch the lab environment

Confirm that the course containing the phrase *"Multi-Cloud Automation"* is listed under In Progress on the welcome screen. Click on the this course in order to add it to your My Learning inventory.



Welcome, Robert!

We give you temporary credentials to Google Cloud Platform and Amazon Web Services, so you can learn the cloud using the real thing – no simulations. From 30-minute individual labs to multi-day courses, from introductory level to expert, instructor-led or self-paced, with topics like machine learning, security, infrastructure, app dev, and more, we've got you covered.

In Progress

SCHEDULED COURSE
Ignite19-EU-HOW-11-Expedition
Nov 1, 2019 7:00AM PDT
Santa Clara

SCHEDULED COURSE
Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible
Nov 1, 2019 8:00AM PDT
Santa Clara

You will be presented with two lab environments within this course: one for GCP and the other for AWS. You may choose either one depending on your learning objectives or platform familiarity.

Note: If you finish all the activities for one lab environment, you are free to launch the other (time permitting). The lab activities are similar, but there are instructions specific to each cloud provider.

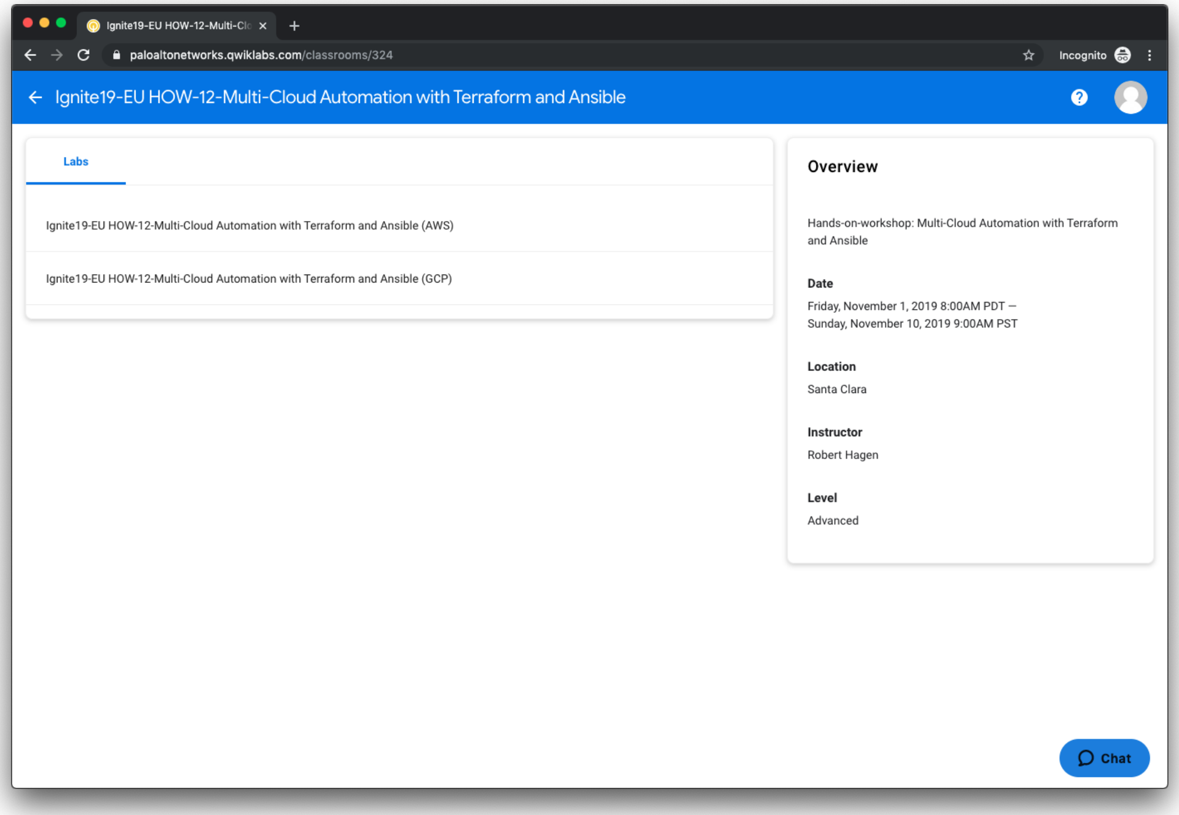
Once you've selected the lab evironment, you will need to click the **Start Lab** button. Qwiklabs will then provision a set of account credentials and instantiate a "launchpad" virtual machine that you will SSH into to perform the rest of the lab activities.

Each lab environment will take a few minutes to provision and deploy the Launchpad VM. Once it is completed, a **Launchpad IP** field will be added to the bottom left panel.

3.3.3 SSH into the Launchpad VM

Once the lab environment has completed the provisioning process and the **Launchpad IP** field is displayed, you may SSH into that IP address using the following credentials.

- **Username:** `student`



Ignite19-EU HOW-12-Multi-Clo
paloaltonetworks.qwiklabs.com/classrooms/324
Incognito
Labs
Chat

Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible

Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (AWS)

Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (GCP)

Overview

Hands-on-workshop: Multi-Cloud Automation with Terraform and Ansible

Date

Friday, November 1, 2019 8:00AM PDT –
Sunday, November 10, 2019 9:00AM PST

Location

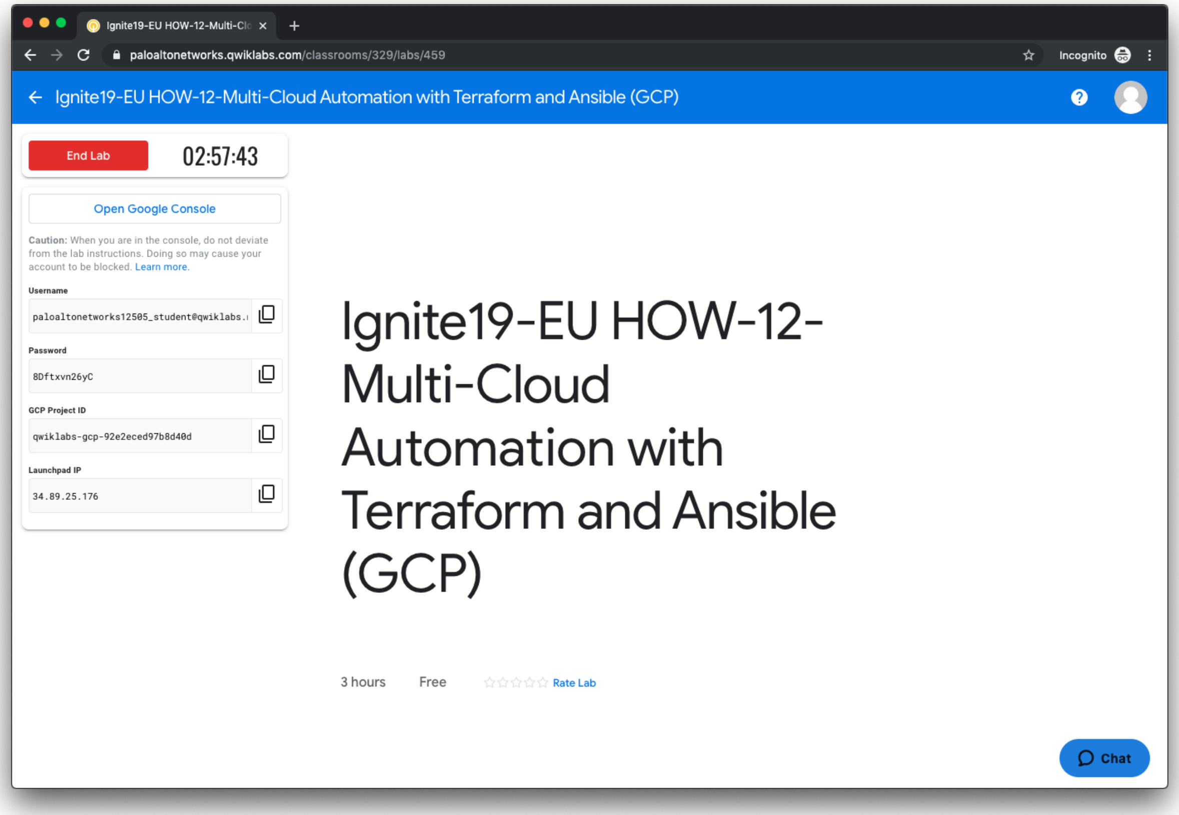
Santa Clara

Instructor

Robert Hagen

Level

Advanced


Ignite19-EU HOW-12-Multi-Clo
paloaltonetworks.qwiklabs.com/classrooms/329/labs/459
Incognito
Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (GCP)
End Lab
02:57:43
Open Google Console
3 hours
Free
Rate Lab
Chat

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username

paloaltonetworks12505_student@qwiklabs.

Password

8Dftxvn26yC

GCP Project ID

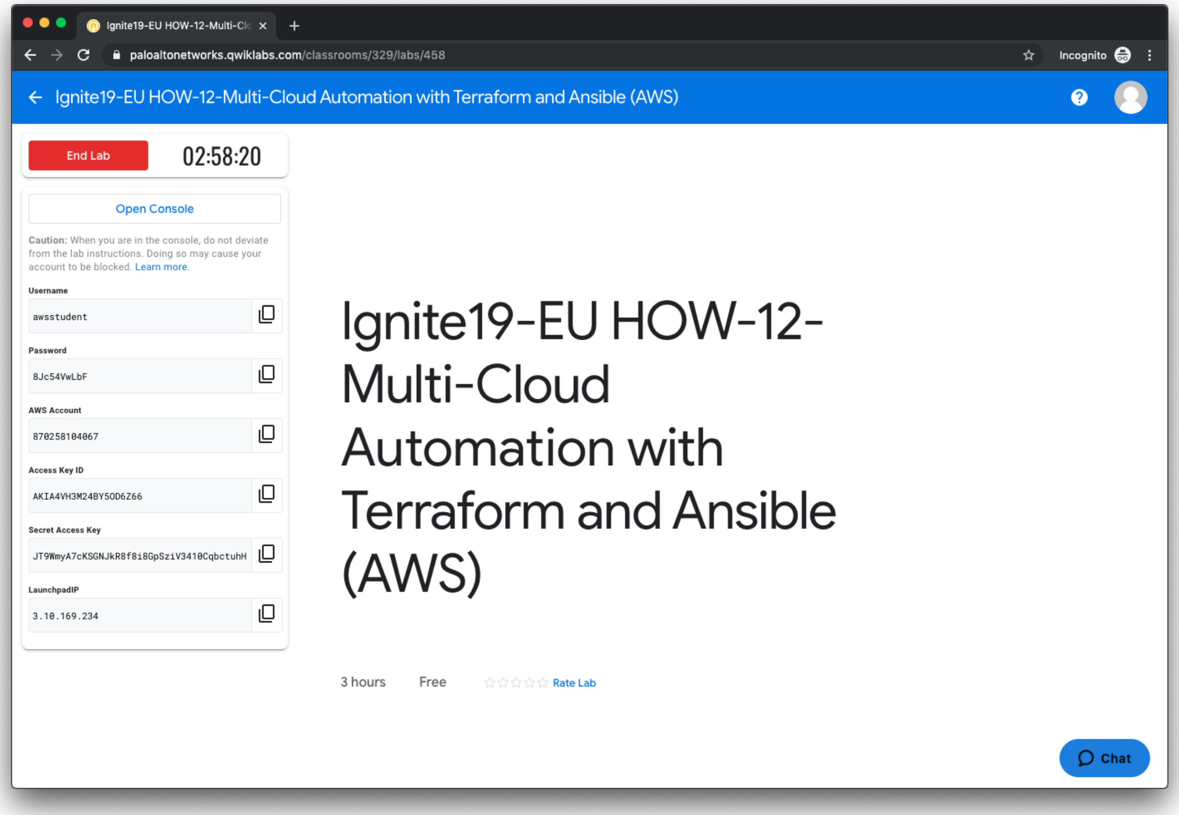
qwiklabs-gcp-92e2eced97b8d40d

Launchpad IP

34.89.25.176

Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (GCP)

Fig. 2: Provisioning the GCP lab environment



Ignite19-EU HOW-12-Multi-Clo
paloaltonetworks.qwiklabs.com/classrooms/329/labs/458
Incognito
Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (AWS)
End Lab
02:58:20
Open Console

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username: awsstudent

Password: 8Jc54VwLbF

AWS Account: 870258104067

Access Key ID: AKIA4VH3M24BY5OD6Z66

Secret Access Key: JT9WmyA7cKSGNJkR8f8i8GpSziV3410CqbctuhH

LaunchpadIP: 3.10.169.234

Ignite19-EU HOW-12-Multi-Cloud Automation with Terraform and Ansible (AWS)

Fig. 3: Provisioning the AWS lab environment

- **Password:** `Ignite2019!`

3.3.4 Clone the lab software repository

Once you have successfully logged into the Launchpad VM you will need to clone the GitHub repository used in this lab. This repository (or *repo*) contains the files needed to deploy the network and compute infrastructure we'll be working with.

```
$ git clone https://github.com/PaloAltoNetworks/multicloud-automation-lab.git
```

You are now ready to deploy the lab infrastructure.

3.4 Lab Deployment (GCP)

> **Warning:** If you are working on the AWS lab, skip this page and proceed to the *Lab Deployment (AWS)*.

In this activity you will:

- Create a service account credential file
- Create an SSH key-pair
- Create the Terraform variables
- Initialize the GCP Terraform provider
- Deploy the lab infrastucture plan
- Confirm firewall bootstrap completion

3.4.1 Create a service account credential file

We will be deploying the lab infrastucture in GCP using Terraform. A predefined Terraform plan is provided that will initialize the GCP provider and call modules responsible for instantiating the network, compute, and storage resources needed.

In order for Terraform to do this it will need to authenticate to GCP. We *could* authenticate to GCP using the username presented in the Qwiklabs panel when the lab was started. However, the Compute Engine default service account is typically used because it is certain to have all the neccesary permissions.

List the email address of the Compute Engine default service account.

```
$ gcloud iam service-accounts list
```

Use the following `gcloud` command to download the credentials for the **Compute Engine default service account** using its associated email address (displayed in the output of the previous command).

```
$ gcloud iam service-accounts keys create ~/gcp_compute_key.json --iam-account <EMAIL_
↪ADDRESS>
```

Verify the JSON credentials file was successfully created.

```
$ cat ~/gcp_compute_key.json
```

3.4.2 Create an SSH key-pair

All Compute Engine instances are required to have an SSH key-pair defined when the instance is created. This is done to ensure secure access to the instance will be available once it is created.

Create an SSH key-pair with an empty passphrase and save them in the `~/.ssh` directory.

```
$ ssh-keygen -t rsa -b 1024 -N '' -f ~/.ssh/lab_ssh_key
```

Note: GCP has the ability to manage all of its own SSH keys and propagate them automatically to projects and instances. However, the VM-Series is only able to make use of a single SSH key. Rather than leverage GCP's SSH key management process, we've created our own SSH key and configured Compute Engine to use our key exclusively.

3.4.3 Create the Terraform variables

Change into the GCP deployment directory.

```
$ cd ~/multicloud-automation-lab/deployment/gcp
```

In this directory you will find the three main files associated with a Terraform plan: `main.tf`, `variables.tf`, and `outputs.tf`. View the contents of these files to see what they contain and how they're structured.

```
$ more main.tf
$ more variables.tf
$ more outputs.tf
```

The file `main.tf` defines the providers that will be used and the resources that will be created (more on that shortly). Since it is poor practice to hard code values into the plan, the file `variables.tf` will be used to declare the variables that will be used in the plan (but not necessarily their values). The `outputs.tf` file will define the values to display that result from applying the plan.

Create a file called `terraform.tfvars` in the current directory that contains the following variables and their values. Fill in the quotes with the GCP project ID, the GCP region, the GCP zone, the path to the JSON credentials file, and the path to your SSH public key file.

```
project                 = "<YOUR_GCP_PROJECT_ID>"
region                  = "<SEE_INSTRUCTOR_PRESENTATION>"
zone                    = "<SEE_INSTRUCTOR_PRESENTATION>"
credentials_file        = "~/gcp_compute_key.json"
public_key_file         = "~/.ssh/lab_ssh_key.pub"
```

3.4.4 Initialize the GCP Terraform provider

Once you've created the `terraform.tfvars` file and populated it with the variables and values you are now ready to initialize the Terraform providers. For this initial deployment we will only be using the GCP Provider. This initialization process will download all the software, modules, and plugins needed for working in a particular environment.

```
$ terraform init
```

3.4.5 Deploy the lab infrastucture plan

We are now ready to deploy our lab infrastructure plan. We should first perform a dry-run of the deployment process and validate the contents of the plan files and module dependencies.

```
$ terraform plan
```

If there are no errors and the plan output looks good, let's go ahead and perform the deployment.

```
$ terraform apply -auto-approve
```

At a high level these are each of the steps this plan will perform:

1. **Run the `bootstrap` module**
    1. Create a GCP storage bucket for the firewall bootstrap package
    2. Apply a policy to the bucket allowing read access to `allUsers`
    3. Create the `/config/init-cfg.txt`, `/config/bootstrap.xml`, `/software`, `/content`, and `/license` objects in the bootstrap bucket
2. **Run the `vpc` module**
    1. Create the VPC
    2. Create the Internet gateway
    3. Create the `management`, `untrust`, `web`, and `database` subnets
    4. Create the security groups for each subnet
    5. Create the default route for the `web` and `database` subnets
3. **Run the `firewall` module**
    1. Create the VM-Series firewall instance
    2. Create the VM-Series firewall interfaces
    3. Create the public IPs for the `management` and `untrust` interfaces
4. **Run the `web` module**
    1. Create the web server instance
    2. Create the web server interface
5. **Run the `database` module**
    1. Create the database server instance
    2. Create the database server interface

The deployment process should finish in a few minutes and you will be presented with the public IP addresses of the VM-Series firewall management and untrust interfaces. However, the VM-Series firewall can take up to *ten minutes* to complete the initial bootstrap process.

It is recommended that you read the *Terraform Background* section ahead while you wait.

3.4.6 Confirm firewall bootstrap completion

SSH into the firewall with the following credentials.

- **Username:** `admin`

- **Password:** `Ignite2019!`

```
$ ssh admin@<FIREWALL_MGMT_IP>
```

Replace `<FIREWALL_MGMT_IP>` with the IP address of the firewall management interface that was provided in the Terraform plan results. This information can be easily recalled using the `terraform output` command within the deployment directory.

> **Warning:** If you are unsuccessful the firewall instance is likely still bootstrapping or performing an autocommit. Hit `Ctrl-C` and try again after waiting a few minutes. The bootstrap process can take up to *ten minutes* to complete before you are able to successfully log in.

Once you have logged into the firewall you can check to ensure the management plane has completed its initialization.

```
admin@lab-fw> show chassis-ready
```

If the response is `yes`, you are ready to proceed with the configuration activities.

> **Note:** While it is a security best practice to use SSH keys to authenticate to VM instances in the cloud, we have defined a static password for the firewall's admin account in this lab (specifically, in the bootstrap package). This is because the firewall API used by Terraform and Ansible cannot utilize SSH keys and must have a username/password or API key for authentication.

3.5 Lab Deployment (AWS)

> **Warning:** If you are working on the GCP lab, skip this page and proceed to *Terraform Background*.

In this activity you will:

- Create AWS environment variables
- Create an SSH key-pair
- Create the Terraform variables
- Initialize the AWS Terraform provider
- Deploy the lab infrastucture plan
- Confirm firewall bootstrap completion

3.5.1 Create AWS environment variables

We will be deploying the lab infrastucture in AWS using Terraform. A predefined Terraform plan is provided that will initialize the AWS provider and call modules responsible for instantiating the network, compute, and storage resources needed.

In order for Terraform to do this it will need to authenticate to AWS using the AWS Access Key and Secret Key values that were presented in the Qwiklabs panel when the lab was started. Rather than write these as Terraform variables, we will use Linux environment variables.

Create the environment variables.

```
$ export AWS_ACCESS_KEY_ID="your-access-key-here"
$ export AWS_SECRET_ACCESS_KEY="your-secret-key-here"
```

3.5.2 Create an SSH key-pair

All AWS EC2 instances are required to have an SSH key-pair defined when the instance is created. This is done to ensure secure access to the instance will be available once it is created.

Create an SSH key-pair with an empty passphrase and save them in the `~/.ssh` directory.

```
$ ssh-keygen -t rsa -b 1024 -N '' -f ~/.ssh/lab_ssh_key
```

3.5.3 Create the Terraform variables

Change into the AWS deployment directory.

```
$ cd ~/multicloud-automation-lab/deployment/aws
```

In this directory you will find the three main files associated with a Terraform plan: `main.tf`, `variables.tf`, and `outputs.tf`. View the contents of these files to see what they contain and how they're structured.

```
$ more main.tf
$ more variables.tf
$ more outputs.tf
```

The file `main.tf` defines the providers that will be used and the resources that will be created (more on that shortly). Since it is poor practice to hard code values into the plan, the file `variables.tf` will be used to declare the variables that will be used in the plan (but not necessarily their values). The `outputs.tf` file will define the values to display that result from applying the plan.

Create a file called `terraform.tfvars` in the current directory that contains the following variables and their values. Fill in the quotes with the AWS region name, the AWS availability zone, and the path to your SSH public key file.

```
aws_region_name     = "<SEE_INSTRUCTOR_PRESENTATION>"
aws_az_name         = "<SEE_INSTRUCTOR_PRESENTATION>"
public_key_file     = "~/.ssh/lab_ssh_key.pub"
```

3.5.4 Initialize the AWS Terraform provider

Once you've created the `terraform.tfvars` file and populated it with the variables and values you are now ready to initialize the Terraform providers. For this initial deployment we will only be using the AWS Provider. This initialization process will download all the software, modules, and plugins needed for working in a particular environment.

```
$ terraform init
```

3.5.5 Deploy the lab infrastucture plan

We are now ready to deploy our lab infrastructure plan. We should first perform a dry-run of the deployment process and validate the contents of the plan files and module dependencies.

```
$ terraform plan
```

If there are no errors and the plan output looks good, let's go ahead and perform the deployment.

```
$ terraform apply -auto-approve
```

At a high level these are each of the steps this plan will perform:

1. **Run the `bootstrap` module**
    1. Create an S3 bucket for the firewall bootstrap package
    2. Assign an IAM policy to the bucket allowing read access from the firewall instance
    3. Create the `/config/init-cfg.txt`, `/config/bootstrap.xml`, `/software`, `/content`, and `/license` objects in the bootstrap bucket
2. **Run the `vpc` module**
    1. Create the VPC
    2. Create the Internet gateway
    3. Create the `management`, `untrust`, `web`, and `database` subnets
    4. Create the security groups for each subnet
    5. Create the default route for the `web` and `database` subnets
3. **Run the `firewall` module**
    1. Create the VM-Series firewall instance
    2. Create the VM-Series firewall interfaces
    3. Create the Elastic IPs for the `management` and `untrust` interfaces
    4. Create an IAM instance profile for accessing the bootstrap bucket
4. **Run the `web` module**
    1. Create the web server instance
    2. Create the web server interface
5. **Run the `database` module**
    1. Create the database server instance
    2. Create the database server interface

The deployment process should finish in a few minutes and you will be presented with the public IP addresses of the VM-Series firewall management and untrust interfaces. However, the VM-Series firewall can take up to *ten minutes* to complete the initial bootstrap process.

It is recommended that you skip ahead and read the *Terraform Background* section while you wait.

3.5.6 Confirm firewall bootstrap completion

SSH into the firewall with the following credentials.

- **Username:** `admin`
- **Password:** `Ignite2019!`

```
$ ssh admin@<FIREWALL_MGMT_IP>
```

Replace `<FIREWALL_MGMT_IP>` with the IP address of the firewall management interface that was provided in the Terraform plan results. This information can be easily recalled using the `terraform output` command within the deployment directory.

Warning: If you are unsuccessful the firewall instance is likely still bootstrapping or performing an autocommit. Hit `Ctrl-C` and try again after waiting a few minutes. The bootstrap process can take up to *ten minutes* to complete before you are able to successfully log in.

Once you have logged into the firewall you can check to ensure the management plane has completed its initialization.

```
admin@lab-fw> show chassis-ready
```

If the response is `yes`, you are ready to proceed with the configuration activities.

Note: While it is a security best practice to use SSH keys to authenticate to VM instances in the cloud, we have defined a static password for the firewall's admin account in this lab (specifically, in the bootstrap package). This is because the PAN-OS XML API cannot utilize SSH keys and requires a username/password or API key for authentication.

3.6 Terraform Background

3.6.1 Terraform At a Glance

- Company: HashiCorp
- Integration First Available: January 2018
- Configuration: HCL (HashiCorp Configuration Language)
- PAN-OS Terraform Provider
- GitHub Repo
- Implementation Language: golang

3.6.2 Configuration Overview

Many Files, One Configuration

Terraform allows you to split your configuration into as many files as you wish. Any Terraform file in the current working directory will be loaded and concatenated with the others when you tell Terraform to apply your desired configuration.

Local State

Terraform saves the things it has done to a local file, referred to as a "state file". Because state is saved locally, that means that sometimes the local state will differ from what's actually configured on the firewall.

This is actually not a big deal, as many of Terraform's commands do a Read operation to check the actual state against what's saved locally. Any changes that are found are then saved to the local state automatically.

Example Terraform Configuration

Here's an example of a Terraform configuration file. We will discuss the parts of this config below.

```
variable "hostname" {
    default = "127.0.0.1"
}

variable "username" {
    default = "admin"
}

variable "password" {
    default = "admin"
}

provider "panos" {
    hostname = "${var.hostname}"
    username = "${var.username}"
    password = "${var.password}"
}

resource "panos_management_profile" "ssh" {
    name = "allow ssh"
    ssh = true
}

resource "panos_ethernet_interface" "eth1" {
    name = "ethernet1/1"
    vsys = "vsys1"
    mode = "layer3"
    enable_dhcp = true
    management_profile = "${panos_management_profile.ssh.name}"
}

resource "panos_zone" "zone1" {
    name = "L3-in"
    mode = "layer3"
    interfaces = ["ethernet1/1"]
    depends_on = ["panos_ethernet_interface.eth1"]
}
```

3.6.3 Terminology

Plan

A Terraform **plan** is the sum of all Terraform configuration files in a given directory. These files are generally written in *HCL*.

Provider

A **provider** can loosely thought of to be a product (such as the Palo Alto Networks firewall) or a service (such as AWS, Azure, or GCP). The provider understands the underlying API to the product or service, making individual parts of those things available as *resources*.

Most providers require some kind of configuration in order to use. For the `panos` provider, this is the authentication credentials of the firewall or Panorama that you want to configure.

Providers are configured in a provider configuration block (e.g. - `provider "panos" {...}`, and a plan can make use of any number of providers, all working together.

Resource

A **resource** is an individual component that a provider supports create/read/update/delete operations for.

For the Palo Alto Networks firewall, this would be something like an ethernet interface, service object, or an interface management profile.

Data Source

A **data source** is like a resource, but read-only.

For example, the `panos` provider has a data source that gives you access to the results of `show system info`.

Attribute

An **attribute** is a single parameter that exists in either a resource or a data source. Individual attributes are specific to the resource itself, as to what type it is, if it's required or optional, has a default value, or if changing it would require the whole resource to be recreated or not.

Attributes can have a few different types:

- *String*: `"foo"`, `"bar"`
- *Number*: `7`, `"42"` (quoting numbers is fine in HCL)
- *List*: `["item1", "item2"]`
- *Boolean**: `true`, `false`
- *Map*: `{"key": "value"}` (some maps may have more complex values)

Variables

Terraform plans can have *variables* to allow for more flexibility. These variables come in two flavors: user variables and attribute variables. Whenever you want to use variables (or any other Terraform interpolation), you'll be enclosing it in curly braces with a leading dollar sign: `"${...}"`

User variables are variables that are defined in the Terraform plan file with the `variable` keyword. These can be any of the types of values that attributes can be (default is string), and can also be configured to have default values. When using a user variable in your plan files, they are referenced with `var` as a prefix: `"${var.hostname}"`. Terraform looks for local variable values in the file `terraform.tfvars`.

Attribute variables are variables that reference other resources or data sources within the same plan. Specifying a resource attribute using an attribute variable creates an implicit dependency, covered below.

Dependencies

There are two ways to tell Terraform that resource “A” needs to be created before resource “B”: the universal *depends_on* resource parameter or an attribute variable. The first way, using *depends_on*, is performed by adding the universal parameter “depends_on” within the dependent resource. The second way, using attribute variables, is performed by referencing a resource or data source attribute as a variable: `"${panos_management_profile.ssh.name}"`

Modules

Terraform can group resources together in reusable pieces called *modules*. Modules can have their own variables to allow for customization, and outputs so that the resources they create can be accessed. Both versions of this lab use modules to group together elements for the base networking components, the firewall, and the created instances.

For example, the AWS firewall configuration is located in `deployment/aws/modules/firewall`. Calling this module creates the firewall instance, the network interfaces, and various other resources.

It can be used in another Terraform plan like this:

```
module "firewall" {
  source = "./modules/firewall"

  name = "vm-series"

  ssh_key_name = "${aws_key_pair.ssh_key.key_name}"
  vpc_id       = "${module.vpc.vpc_id}"

  fw_mgmt_subnet_id = "${module.vpc.mgmt_subnet_id}"
  fw_mgmt_ip        = "10.5.0.4"
  fw_mgmt_sg_id     = "${aws_security_group.firewall_mgmt_sg.id}"

  fw_eth1_subnet_id = "${module.vpc.public_subnet_id}"
  fw_eth2_subnet_id = "${module.vpc.web_subnet_id}"
  fw_eth3_subnet_id = "${module.vpc.db_subnet_id}"

  fw_dataplane_sg_id = "${aws_security_group.public_sg.id}"

  fw_version          = "9.0"
  fw_product_code     = "806j2of0qy5osgjjixq9gqc6g"
  fw_bootstrap_bucket = "${module.bootstrap_bucket.bootstrap_bucket_name}"

  tags {
    Environment = "Multicloud-AWS"
  }
}
```

This calls the firewall module, and passes in values for the variables it requires.

3.6.4 Common Commands

The Terraform binary has many different CLI arguments that it supports. We’ll discuss only a few of them here:

```
$ terraform init
```

`terraform init` initializes the current directory based off of the local plan files, downloading any missing provider binaries or modules.

```
$ terraform plan
```

`terraform plan` refreshes provider/resource states and reports what changes need to take place.

```
$ terraform apply
```

`terraform apply` refreshes provider/resource states and makes any needed changes to the resources.

```
$ terraform destroy
```

`terraform destroy` refreshes provider/resource states and removes all resources that Terraform created.

3.7 Terraform Configuration

In this activity you will:

- Initialize the Provider
- Configure Network Interfaces
- Configure Virtual Router
- Configure Security Zones

For this portion of the lab, you will be using the Palo Alto Networks Terraform for PAN-OS provider.

First, change to the Terraform configuration directory.

```
$ cd ~/multicloud-automation-lab/configuration/terraform
```

3.7.1 Provider Initialization

Your first task is to set up the communications between the provider and your lab firewall. There's several ways this can be done. The IP address, username, and password (or API key) can be set as variables in Terraform, and can be typed in manually each time the Terraform plan is run, or specified on the command line using the `-var` command line option to `terraform plan` and `terraform apply`. You can also reference a JSON file in the provider configuration which can contain the configuration.

Another way you can accomplish this is by using environment variables. Use the following commands to add the appropriate environment variables:

```
$ export PANOS_HOSTNAME="<YOUR FIREWALL MGMT IP GOES HERE>"
$ export PANOS_USERNAME="admin"
$ export PANOS_PASSWORD="Ignite2019!"
```

Note: Replace the text `<YOUR FIREWALL MGMT IP GOES HERE>` with your firewall's management IP address.

Now, you should see the variables exported in your shell, which you can verify using the `env | grep PANOS` command:

```
PANOS_HOSTNAME=3.216.53.203
PANOS_USERNAME=admin
PANOS_PASSWORD=Ignite2019!
```

With these values defined, we can now initialize the Terraform panos provider with the following command.

```
$ terraform init
```

The provider is now ready to communicate with our firewall.

3.7.2 Network Interfaces

Your firewall has been bootstrapped with an initial password and nothing else. We're going to be performing the initial networking configuration with Terraform.

You've been provided with the following Terraform plan in `main.tf`:

```
provider "panos" {}

resource "panos_ethernet_interface" "untrust" {
    name                      = "ethernet1/1"
    vsys                      = "vsys1"
    mode                      = "layer3"
    enable_dhcp               = true
    create_dhcp_default_route = true
}

resource "panos_ethernet_interface" "web" {
    name        = "ethernet1/2"
    vsys        = "vsys1"
    mode        = "layer3"
    enable_dhcp = true
}

resource "panos_ethernet_interface" "db" {
    name        = "ethernet1/3"
    vsys        = "vsys1"
    mode        = "layer3"
    enable_dhcp = true
}
```

This configuration creates your network interfaces. The PAN-OS provider doesn't need any additional configuration specified because it is pulling that information from the environment variables we set earlier.

Now, you can run `terraform apply`, and the interfaces will be created on the firewall.

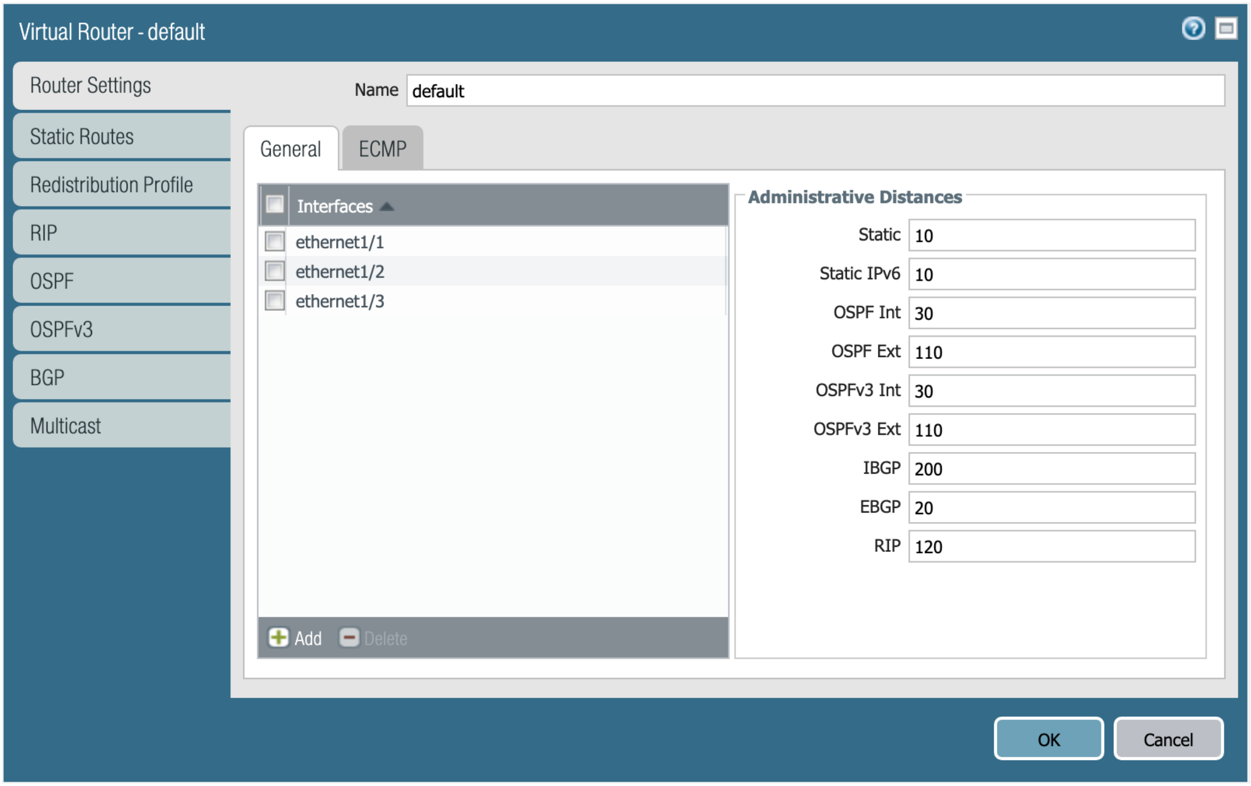
3.7.3 Virtual Router

Now, you'll have to assign those interfaces to the default virtual router. You will need the panos_virtual_router resource.

The example code from that page looks like this:

```
resource "panos_virtual_router" "example" {
    name = "my virtual router"
    static_dist = 15
    interfaces = ["ethernet1/1", "ethernet1/2"]
}
```

Your version will be similar, but it should have the following definition:



Name: default

Interfaces
ethernet1/1
ethernet1/2
ethernet1/3

Administrative Distances

Static	10
Static IPv6	10
OSPF Int	30
OSPF Ext	110
OSPFv3 Int	30
OSPFv3 Ext	110
IBGP	200
EBGP	20
RIP	120

Fig. 4: Virtual router **default**.

Specifying the static distance isn't required.

Define the virtual router resource in `main.tf`, and run `terraform apply`.

> **Warning:** AWS and GCP have slight differences in the way that routing has to be configured. **If you chose GCP as your cloud, you have an additional step!**
>
> If you chose AWS, please continue to *Security Zones* section and skip the following.

GCP requires static routes for each subnet to be defined on the virtual router. You will need the panos_static_route_ipv4 resource.

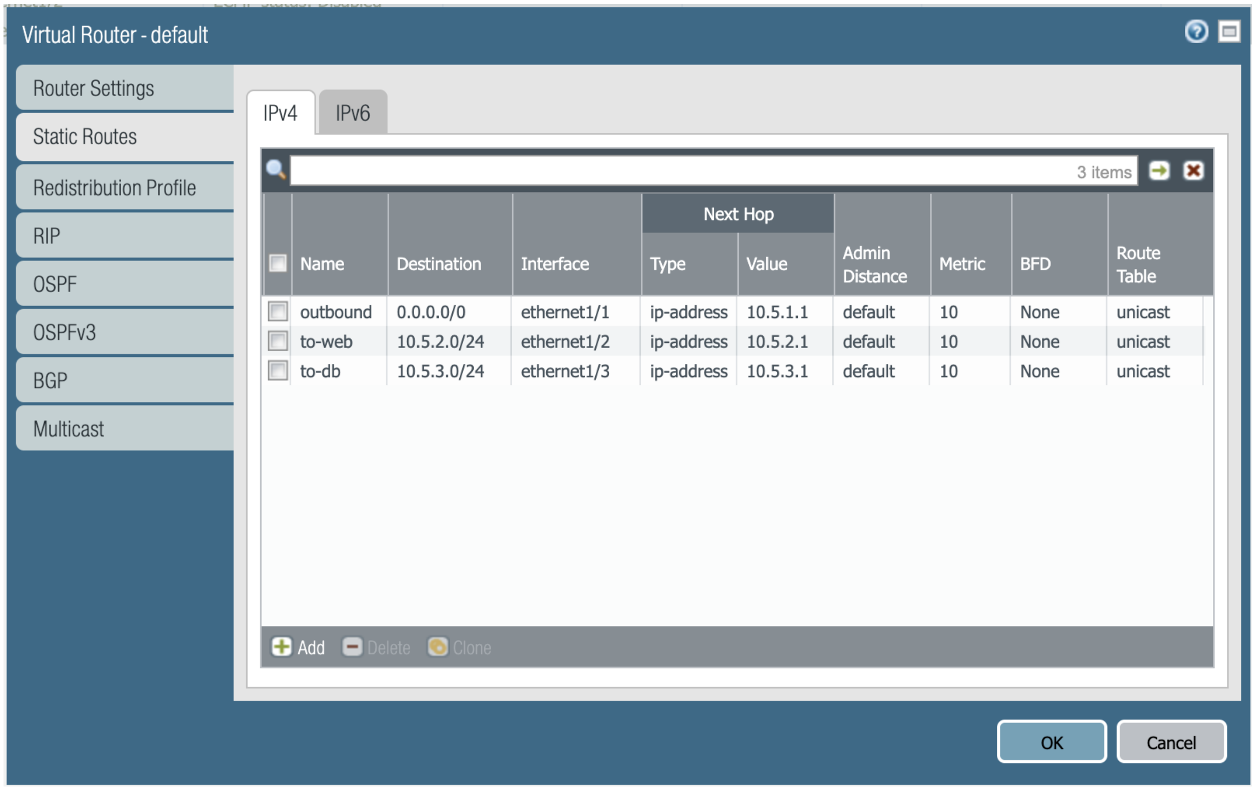
The example code from that page looks like this:

```
resource "panos_static_route_ipv4" "example" {
    name = "localnet"
    virtual_router = "${panos_virtual_router.vr1.name}"
    destination = "10.1.7.0/32"
    next_hop = "10.1.7.4"
}

resource "panos_virtual_router" "vr1" {
    name = "my virtual router"
}
```

This code adds a static route named *localnet*, that routes traffic destined to the network *10.1.7.0/32* to the next hop of *10.1.7.4*.

You will need to create three resources for the static routes depicted below:



□	Name	Destination	Interface	Next Hop		Admin Distance	Metric	BFD	Route Table
				Type	Value				
□	outbound	0.0.0.0/0	ethernet1/1	ip-address	10.5.1.1	default	10	None	unicast
□	to-web	10.5.2.0/24	ethernet1/2	ip-address	10.5.2.1	default	10	None	unicast
□	to-db	10.5.3.0/24	ethernet1/3	ip-address	10.5.3.1	default	10	None	unicast

Fig. 5: Static routes needed in GCP.

Define those resources in `main.tf`, and run `terraform apply`.

3.7.4 Security Zones

Next is creating the security zones for the firewall. You will need the panos_zone resource.

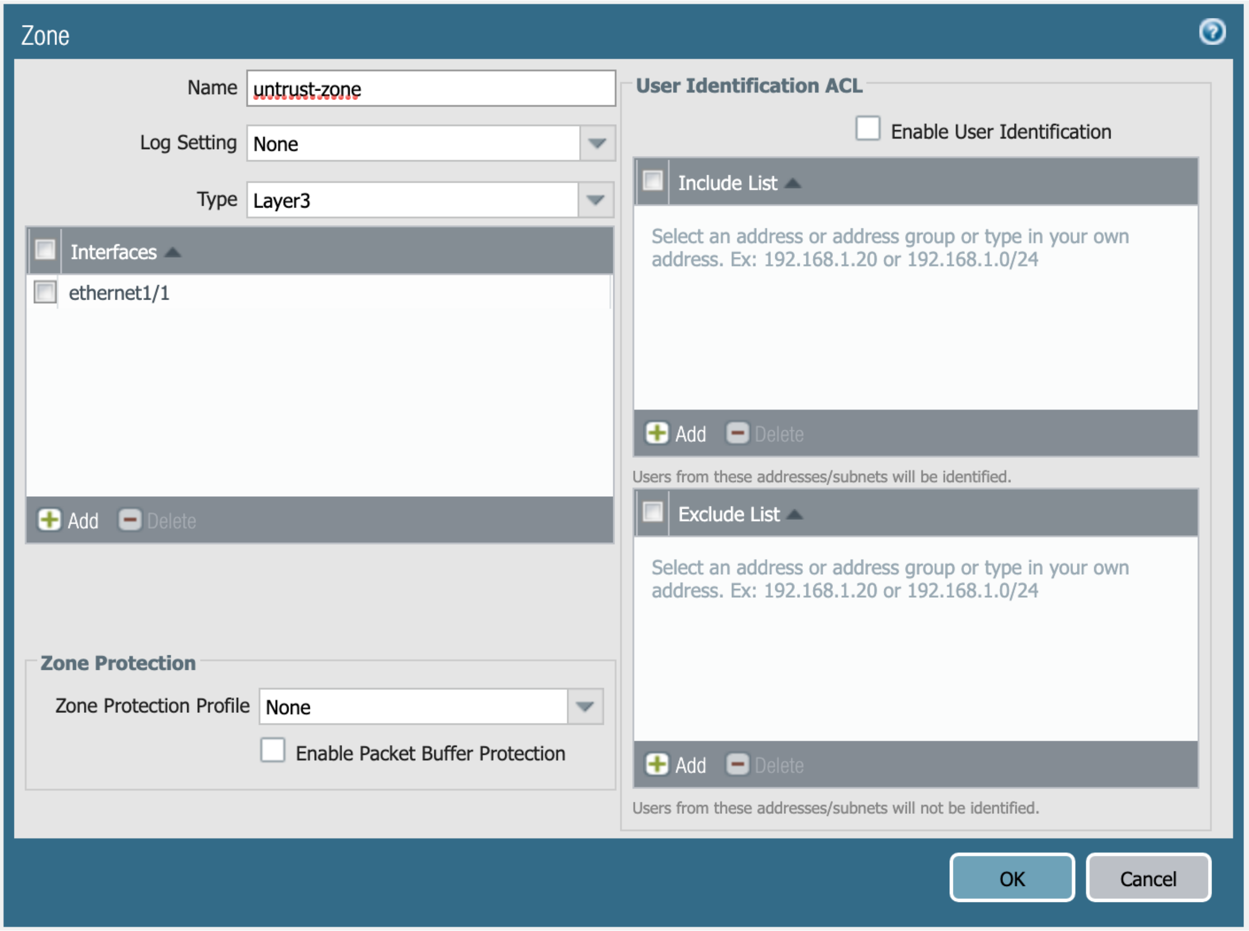
The example code from that page looks like this:

```
resource "panos_zone" "example" {
    name = "myZone"
    mode = "layer3"
    interfaces = ["${panos_ethernet_interface.e1.name}", "${panos_ethernet_interface.
↪e5.name}"]
    enable_user_id = true
    exclude_acls = ["192.168.0.0/16"]
}

resource "panos_ethernet_interface" "e1" {
    name = "ethernet1/1"
    mode = "layer3"
}

resource "panos_ethernet_interface" "e5" {
    name = "ethernet1/5"
    mode = "layer3"
}
```

You need to create three security zones (similar to e1 or e5 in this example), but they need to have the following definition:



Zone

Name: untrust-zone

Log Setting: None

Type: Layer3

□ Interfaces

□ ethernet1/1

Zone Protection

Zone Protection Profile: None

□ Enable Packet Buffer Protection

User Identification ACL

□ Enable User Identification

□ Include List

Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24

Users from these addresses/subnets will be identified.

□ Exclude List

Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24

Users from these addresses/subnets will not be identified.

Fig. 6: Definition of **untrust-zone**.

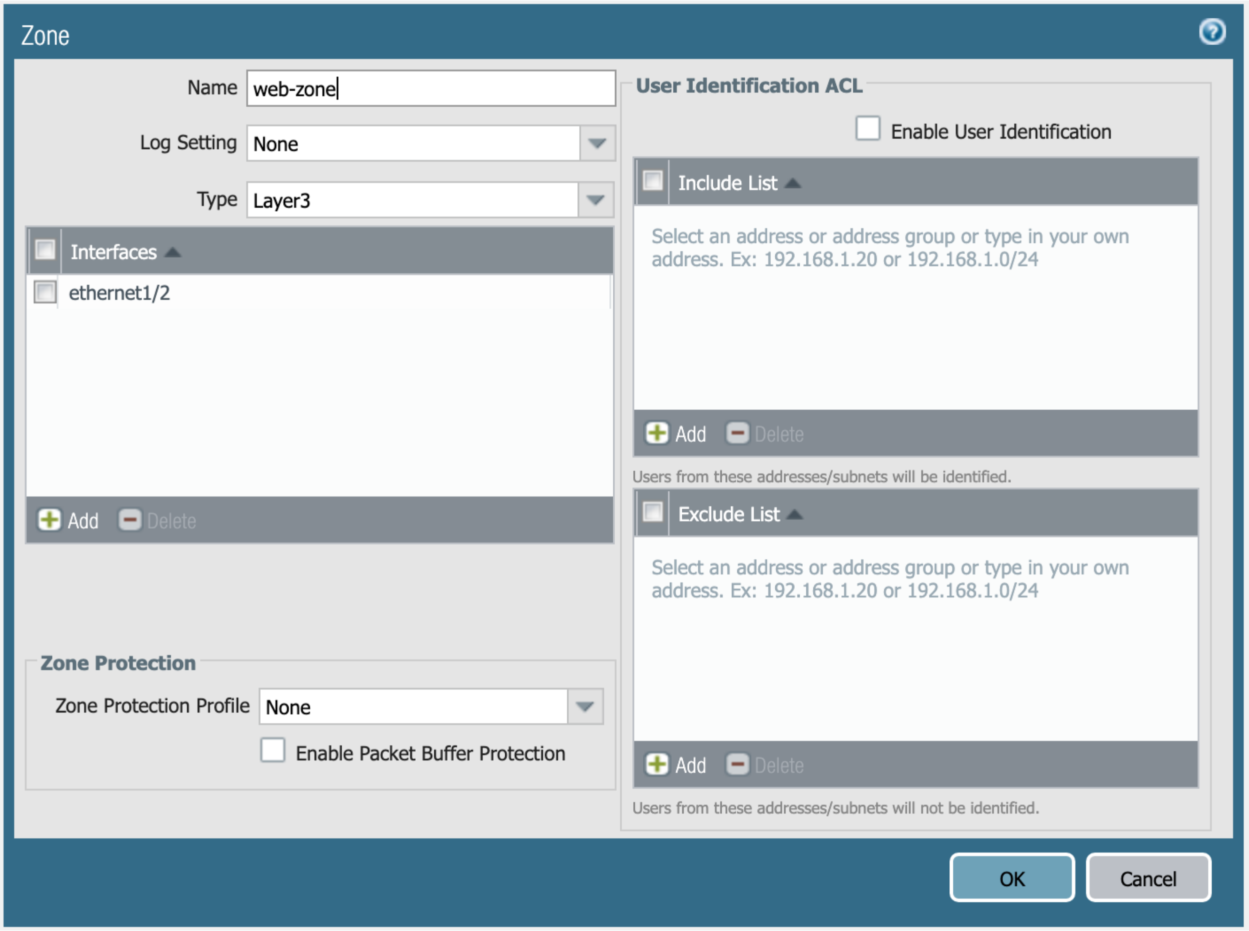
Define those resources in `main.tf`, and run `terraform apply`.

You're done with the Terraform portion of the lab!

3.8 Ansible Background

3.8.1 Ansible At a Glance

- Company: RedHat
- Integration First Available: January 2015
- Configuration: YAML (Yet Another Markup Language)
- Documentation
- GitHub Repo
- Implementation Language: python



Zone

Name: web-zone

Log Setting: None

Type: Layer3

□ Interfaces

□ ethernet1/2

Zone Protection

Zone Protection Profile: None

□ Enable Packet Buffer Protection

User Identification ACL

□ Enable User Identification

□ Include List

Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24

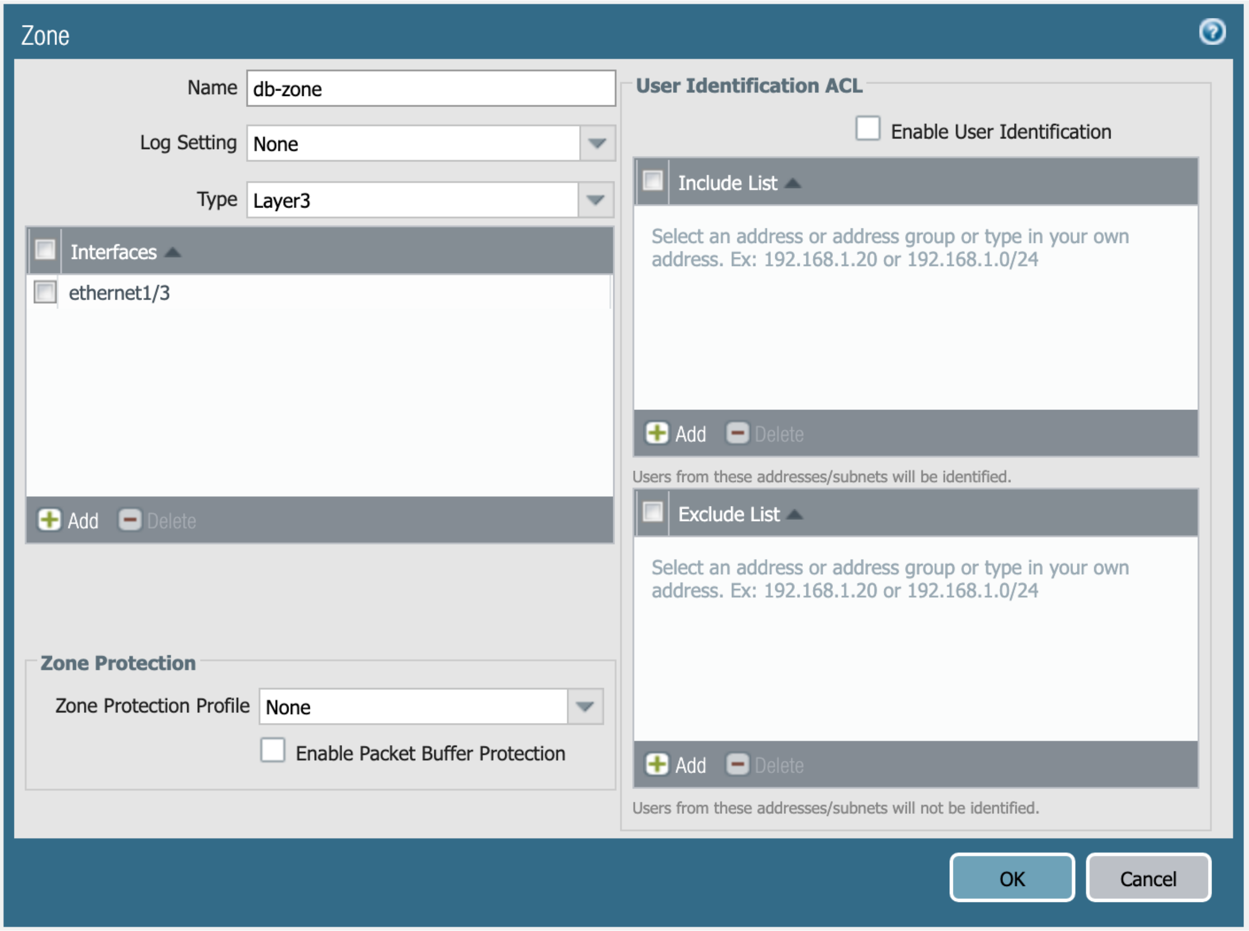
Users from these addresses/subnets will be identified.

□ Exclude List

Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24

Users from these addresses/subnets will not be identified.

Fig. 7: Definition of **web-zone**.



Zone

Name: db-zone

Log Setting: None

Type: Layer3

Interfaces: ethernet1/3

Zone Protection Profile: None

□ Enable Packet Buffer Protection

User Identification ACL

□ Enable User Identification

Users from these addresses/subnets will be identified.

Users from these addresses/subnets will not be identified.

Include List
Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24
Add
Delete
Exclude List
Select an address or address group or type in your own address. Ex: 192.168.1.20 or 192.168.1.0/24
Add
Delete
Zone Protection
Add
Delete
OK
Cancel

Fig. 8: Definition of **db-zone**.

3.8.2 Configuration Overview

Playbooks

Though Ansible allows you to execute ad hoc commands against your desired inventory, the better way to use Ansible is with Ansible playbooks. Ansible playbooks are a list of configuration operations, or plays, to be performed. Ansible playbooks are written in YAML, which you can find out more about here. Playbooks are run from top to bottom, which means that if one configuration depends on another being present, you simply put the dependency higher in the playbook. You can even tell Ansible to run another playbook from within the first playbook by importing it in.

No Local State

Unlike Terraform, Ansible does not keep a local state of what is configured.

Modules Are Use Case Focused

Also unlike the Terraform provider, Ansible modules tend to be more use case focused as opposed to trying to be a single, atomic component controller. The panos_interface module is probably the best example of this to date, as it not only creates interfaces, but can also create zones, place the interface into that zone, then finally put the interface into a virtual router. That same workflow in Terraform would require three separate resources using dependencies.

3.8.3 Example Ansible Configuration

Here's an example of an Ansible playbook. We will discuss the various parts of this below.

```
- name: My Ansible Playbook
  hosts: my-fw
  connection: local
  gather_facts: false

  roles:
    - role: PaloAltoNetworks.paloaltonetworks

  tasks:
    - name: Grab auth creds
      include_vars: 'fw_creds.yml'
      no_log: 'yes'

    - name: Add interface management profile
      panos_management_profile:
        provider: '{{ provider }}'
        name: 'allow ssh'
        ssh: true
        commit: false

    - name: Configure eth1/1 and put it in zone L3-in
      panos_interface:
        provider: '{{ provider }}'
        if_name: 'ethernet1/1'
        zone_name: 'L3-in'
        commit: false
```

3.8.4 Terminology

Hosts

Ansible executes actions against an inventory. If you're going to run Ansible in production, you'll probably want to use the inventory file to organize your firewalls and Panoramas into groups to make management easier. For the purposes of our lab, however, we just want to run the playbooks against a single host. So instead of putting the host in a hosts file, we're going to use variables instead.

If you desire, you can read more about Ansible inventory here.

Connection

Typically Ansible will ssh to a remote machine and perform commands as the specified user account. However, we don't want this for the Palo Alto Networks Ansible modules, as the modules connect to our API. Thus this should be set to "local" as we want Ansible to initiate the connection locally.

Gather Facts

Ansible facts are just information about remote nodes. In our case, we aren't going to use facts for anything, so we're disabling them to ensure that our Ansible invocations are run in a timely manner (this is would probably not be disabled in production).

If you want to read more about facts, you can find that info here.

Roles

Let's discuss the **PaloAltoNetworks.paloaltonetworks** role that our playbook is using. Ansible comes with various Palo Alto Networks packages when you `pip install ansible`, but updating these packages takes a lot of time and effort. In an effort to get new features to customers sooner, we've made newer features available as an Ansible galaxy role. Including this role in our playbook means that Ansible will use the role's code (the newest released code) for the Ansible plays instead of the older code that's merged upstream with Ansible.

Tasks

Each playbook contains a list of tasks to perform. These are executed in order, one at a time against the inventory. Each task will have a "name", and this name is what shows up on the CLI when executing the Ansible playbook. Besides the name, you will specify the module to execute, and then an indented list of the values you want to pass in to that module.

Knowing what you know about tasks, let's take a look at that "include_vars" task. At this point, knowing what the format of tasks is, you can now identify "include_vars" as a module invocation (documentation for "include_vars" is here.

So what's that `no_log` part? This is simply to keep the authentication credentials safe without compromising the verbosity of our Ansible output. You can read more about that here in the Ansible FAQs.

3.8.5 Dependencies

As mentioned previously, if you're using Ansible playbooks, then when you have dependencies, simply place those further up in the playbook.

3.9 Ansible Configuration

In this activity you will:

- Define Module Communications
- Define Address Objects
- Define Service Objects
- Define Security Rules
- Define NAT Rules
- Commit the Configuration
- Run the Playbook

For this portion of the lab, you're going to be using the Palo Alto Networks Ansible modules.

First, let's change to the Ansible configuration directory.

```
$ cd ~/multicloud-automation-lab/configuration/ansible
```

3.9.1 Module Communications

Just like with Terraform, your first task is setting up the communication with the firewall. The IP address, username, and password (or API key) can be set as variables or specified on the command line. However, since we've already got them as environment variables, we can just read them in.

The `vars.yml` file contains the following:

```
provider:
  ip_address: "{{ lookup('env', 'PANOS_HOSTNAME') }}"
  username: "{{ lookup('env', 'PANOS_USERNAME') }}"
  password: "{{ lookup('env', 'PANOS_PASSWORD') }}"
```

This code simply reads the content of the environment variables we set in the Terraform portion into the dictionary `provider`. This is then referenced by our playbook file, `playbook.yml`.

Similar to the Terraform portion of the lab, our firewall doesn't have any objects or rules configured. We're going to implement that with an Ansible playbook.

Note: You wouldn't actually change tools in the middle of configuration like we're doing here. We just want you to get exposure to both tools and see that you can accomplish the same tasks with either one.

3.9.2 Address Objects

Open the `playbook.yml` file in your text editor. It will contain the following:

```
---
- hosts: localhost
  connection: local
  gather_facts: false

vars_files:
```

(continues on next page)

(continued from previous page)

```
  - vars.yml

roles:
  - PaloAltoNetworks.paloaltonetworks

tasks:
  - name: Create web server object
    panos_address_object:
      provider: "{{ provider }}"
      name: "web-srv"
      value: "10.5.2.5"
      commit: False
      state: present

  - name: Create DB server object
    panos_address_object:
      provider: "{{ provider }}"
      name: "db-srv"
      value: "10.5.3.5"
      commit: False
      state: present
```

This playbook creates the following address objects by using the panos_address_object module. Also notice the fact that `commit` is set to **False**, so that we don't have to wait on a commit each time a module runs.

3.9.3 Service Objects

Next, create some service objects. We want to allow SSH on some non-standard ports so we can easily communicate with web and DB servers behind our firewall. You'll need to refer to the panos_service_object module documentation.

The example code for that module looks like this:

```
- name: Create service object 'ssh-tcp-22'
  panos_service_object:
    provider: '{{ provider }}'
    name: 'ssh-tcp-22'
    destination_port: '22'
    description: 'SSH on tcp/22'
```

Use the `panos_service_object` module to create two objects with the following definitions:

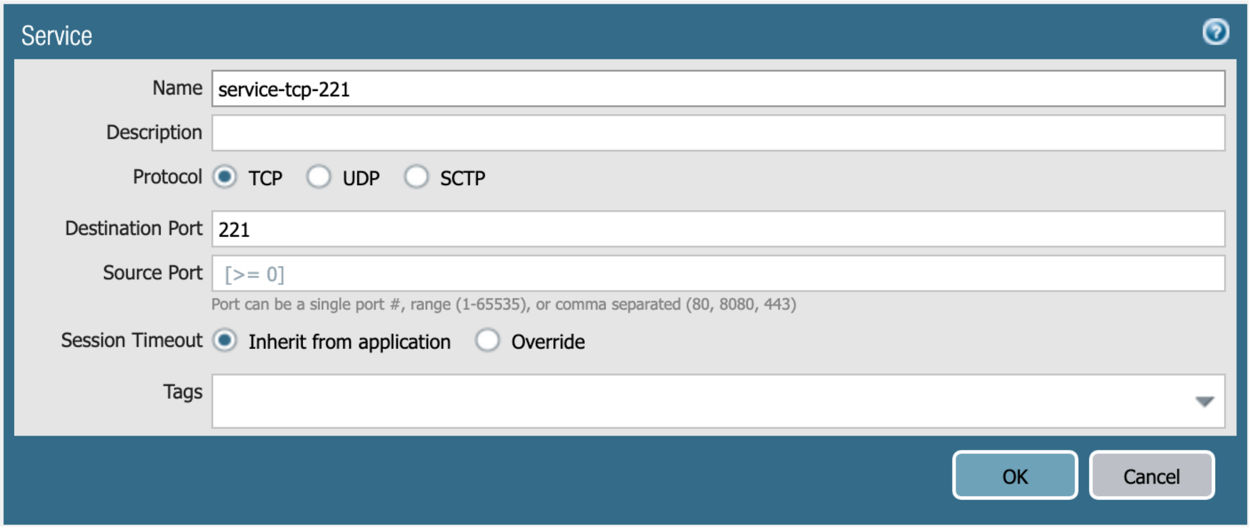
3.9.4 Security Rules

Now we need to create security rules to allow traffic. You'll need to refer to the panos_security_rule module documentation.

The example code for that module looks like this:

```
- name: add SSH inbound
  panos_security_rule:
    provider: '{{ provider }}'
    rule_name: 'SSH permit'
    description: 'SSH rule test'
    source_zone: ['public']
```

(continues on next page)



Service

Name: service-tcp-221

Description:

Protocol: ● TCP ○ UDP ○ SCTP

Destination Port: 221

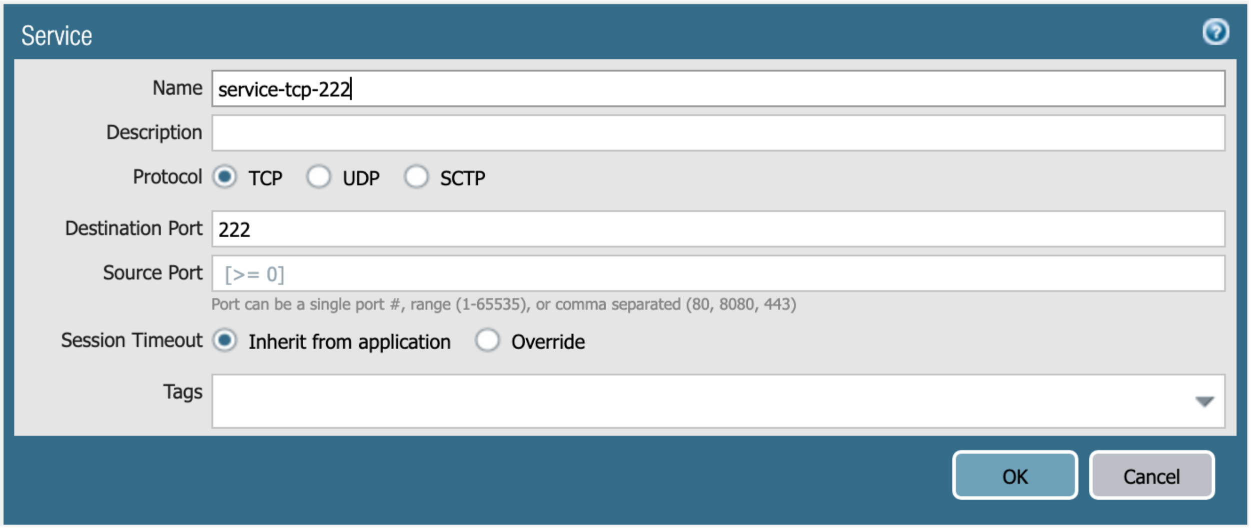
Source Port: [>= 0]

Port can be a single port #, range (1-65535), or comma separated (80, 8080, 443)

Session Timeout: ● Inherit from application ○ Override

Tags:

Fig. 9: **service-tcp-221** service object.



Service

Name: service-tcp-222

Description:

Protocol: ● TCP ○ UDP ○ SCTP

Destination Port: 222

Source Port: [>= 0]

Port can be a single port #, range (1-65535), or comma separated (80, 8080, 443)

Session Timeout: ● Inherit from application ○ Override

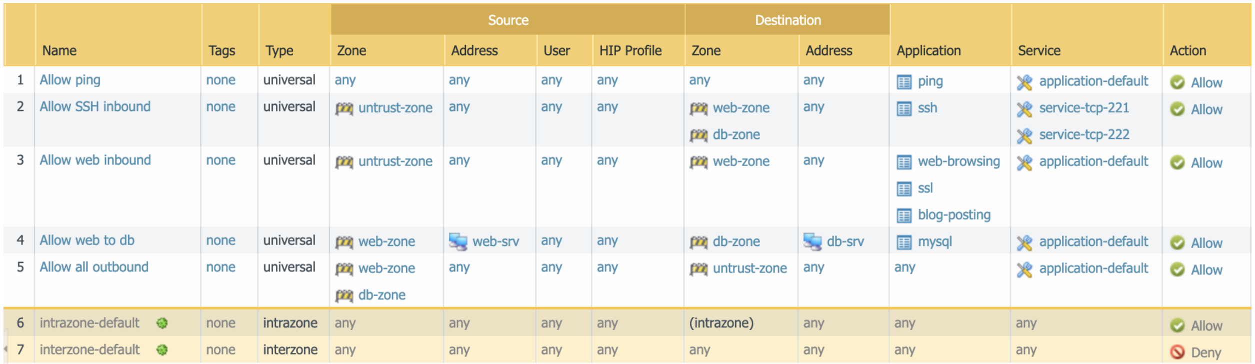
Tags:

Fig. 10: **service-tcp-222** service object.

(continued from previous page)

```
    source_ip: ['any']
    destination_zone: ['private']
    destination_ip: ['1.1.1.1']
    application: ['ssh']
    action: 'allow'
```

Use the `panos_security_rule` module to create the following security rules:



				Source				Destination				
	Name	Tags	Type	Zone	Address	User	HIP Profile	Zone	Address	Application	Service	Action
1	Allow ping	none	universal	any	any	any	any	any	any	ping	application-default	Allow
2	Allow SSH inbound	none	universal	untrust-zone	any	any	any	web-zone db-zone	any	ssh	service-tcp-221 service-tcp-222	Allow
3	Allow web inbound	none	universal	untrust-zone	any	any	any	web-zone	any	web-browsing ssl blog-posting	application-default	Allow
4	Allow web to db	none	universal	web-zone	web-srv	any	any	db-zone	db-srv	mysql	application-default	Allow
5	Allow all outbound	none	universal	web-zone db-zone	any	any	any	untrust-zone	any	any	application-default	Allow
6	intrazone-default	none	intrazone	any	any	any	any	(intrazone)	any	any	any	Allow
7	interzone-default	none	interzone	any	any	any	any	any	any	any	any	Deny

Fig. 11: Security rules to be created.

3.9.5 NAT Rules

Now we need to create the required NAT rules. You'll need to refer to the panos_nat_rule module documentation.

The example code for that module looks like this:

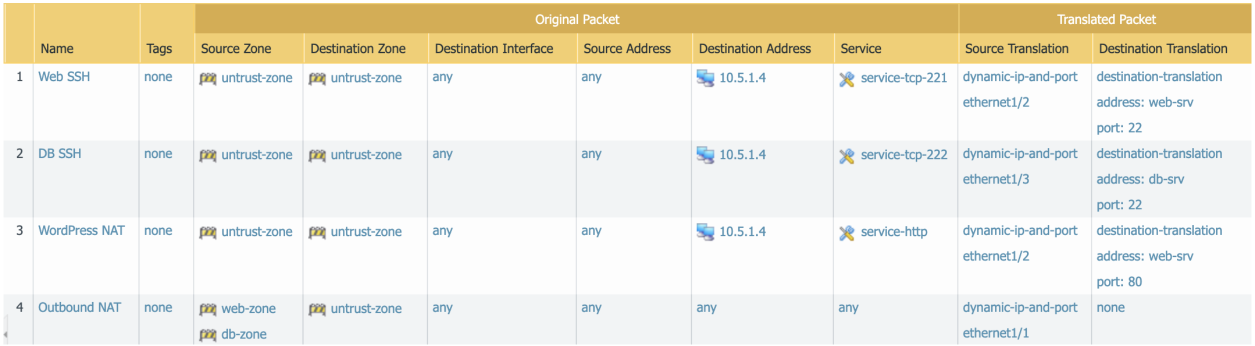
```
- name: Create NAT SSH rule for 10.0.1.101
  panos_nat_rule:
    provider: '{{ provider }}'
    rule_name: "Web SSH"
    source_zone: ["external"]
    destination_zone: "external"
    source_ip: ["any"]
    destination_ip: ["10.0.0.100"]
    service: "service-tcp-221"
    snat_type: "dynamic-ip-and-port"
    snat_interface: "ethernet1/2"
    dnat_address: "10.0.1.101"
    dnat_port: "22"
```

Use the `panos_nat_rule` module to create the following NAT rules:

Note: Pay attention to the module arguments for `panos_nat_rule`. **destination_zone** and **service** are strings here, not lists. This is because you can't write a NAT rule on PAN-OS with multiple destination zones or services.

3.9.6 Commit the Configuration

If you have been writing your playbook with `commit` set to **False** each time, you have an uncommitted candidate configuration. There's a panos_commit module to perform a commit.



			Original Packet						Translated Packet	
	Name	Tags	Source Zone	Destination Zone	Destination Interface	Source Address	Destination Address	Service	Source Translation	Destination Translation
1	Web SSH	none	untrust-zone	untrust-zone	any	any	10.5.1.4	service-tcp-221	dynamic-ip-and-port ethernet1/2	destination-translation address: web-srv port: 22
2	DB SSH	none	untrust-zone	untrust-zone	any	any	10.5.1.4	service-tcp-222	dynamic-ip-and-port ethernet1/3	destination-translation address: db-srv port: 22
3	WordPress NAT	none	untrust-zone	untrust-zone	any	any	10.5.1.4	service-http	dynamic-ip-and-port ethernet1/2	destination-translation address: web-srv port: 80
4	Outbound NAT	none	web-zone db-zone	untrust-zone	any	any	any	any	dynamic-ip-and-port ethernet1/1	none

Fig. 12: NAT rules to be created.

The example code for the module should do what you need:

```
- name: commit candidate config on firewall
  panos_commit:
    provider: '{{ provider }}'
```

3.9.7 Run the Playbook

Save and exit your `playbook.yml` file. Then run your playbook with the following command:

```
$ ansible-playbook -i inventory playbook.yml
```

Log in to the web UI of the firewall, and verify that the configuration matches what you want. If you get errors, indentation is most likely the problem. Ansible is very particular about lines being indented with spaces and not with tabs.

You're now done with the Ansible portion of the lab!

3.10 Validation Testing

In this activity you will:

- Access the Apache web server
- Access the WordPress application
- Post a blog article
- Verify firewall rule matches

The previous two activities had you deploy and configure the infrastructure supporting our WordPress application. Now it's time to see if everything works as planned. If so, you should be able to access the application, post a blog article, and verify that the appropriate firewall rules are being hit. If not, you will need to troubleshoot your configs and make the necessary corrections.

3.10.1 Access the Apache web server

The web server is using the firewall's untrust interface address in a destination NAT rule. Run the following commands to determine the IP address of this interface.

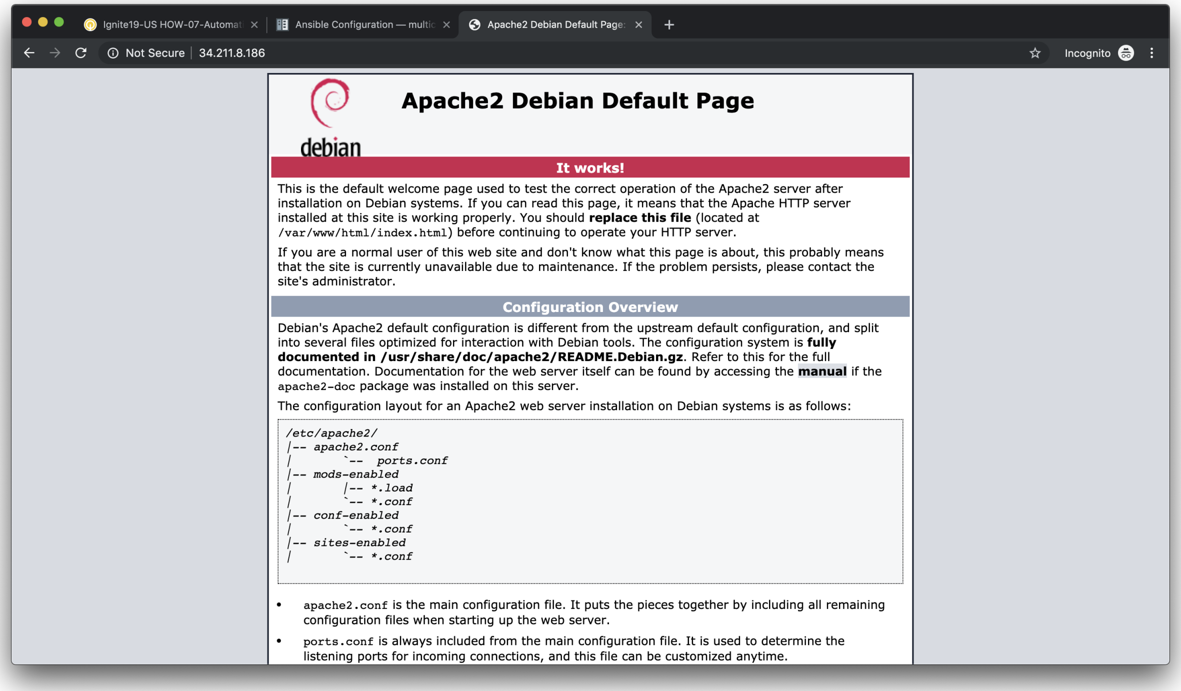
For GCP:

```
$ cd ~/multicloud-automation-lab/deployment/gcp
$ terraform output
```

For AWS:

```
$ cd ~/multicloud-automation-lab/deployment/aws
$ terraform output
```

Open a new tab in your web browser and go to `http://<web-server-ip-address>`. You should see the Apache default home page.


Ignite19-US HOW-07-Automat
Ansible Configuration — multic
Apache2 Debian Default Page
Not Secure
34.211.8.186
Incognito
debian

Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.

3.10.2 Access the WordPress application

Append `/wordpress` to the end of the web server URL and the WordPress installation page should be displayed.

Fill in values of your choosing for the **Site Name**, **Username**, and **Your Email**. These are only for testing and do not need to be real values.

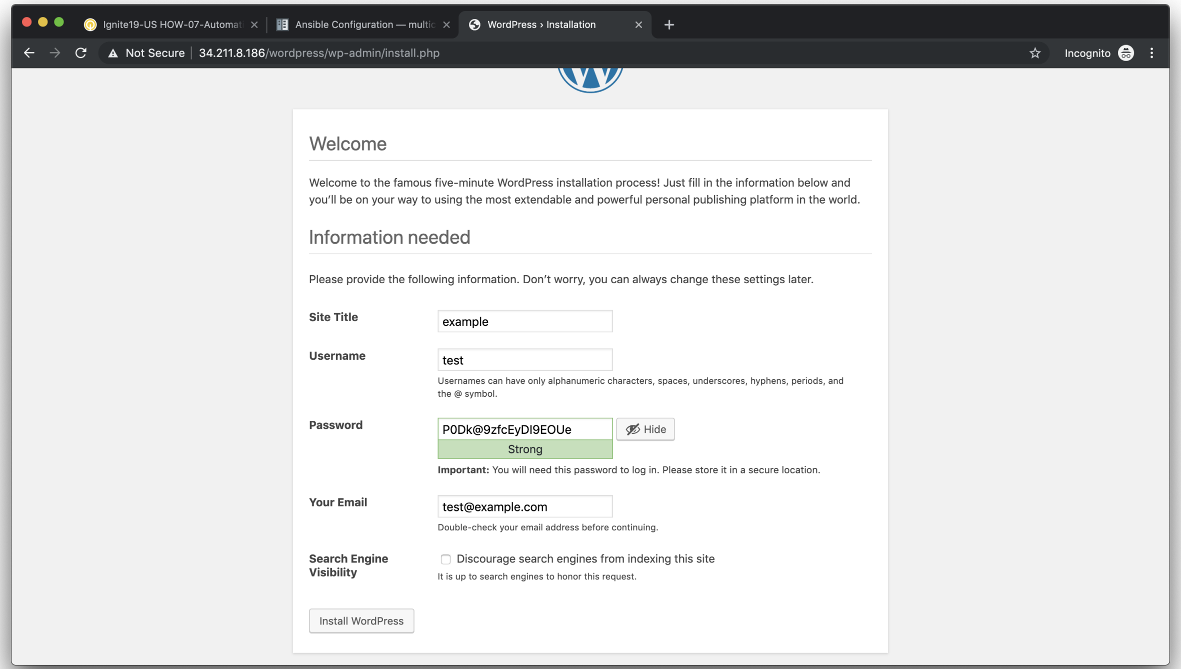
Note: Make sure you copy the password that is provided to your clipboard. Otherwise you may not be able to log in once WordPress is installed.

Click **Install WordPress** when you are done.

On the following page, click on **Log In** to log into the WordPress administrator dashboard.

Log into WordPress using the username and password you created.

You will then be presented with the WordPress administrator dashboard.


Ignite19-US HOW-07-Automat
Ansible Configuration — multic
WordPress › Installation
Not Secure
34.211.8.186/wordpress/wp-admin/install.php
Incognito

Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title: example

Username: test

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password: P0Dk@9zfcEyDI9EOUe Hide

Strong

Important: You will need this password to log in. Please store it in a secure location.

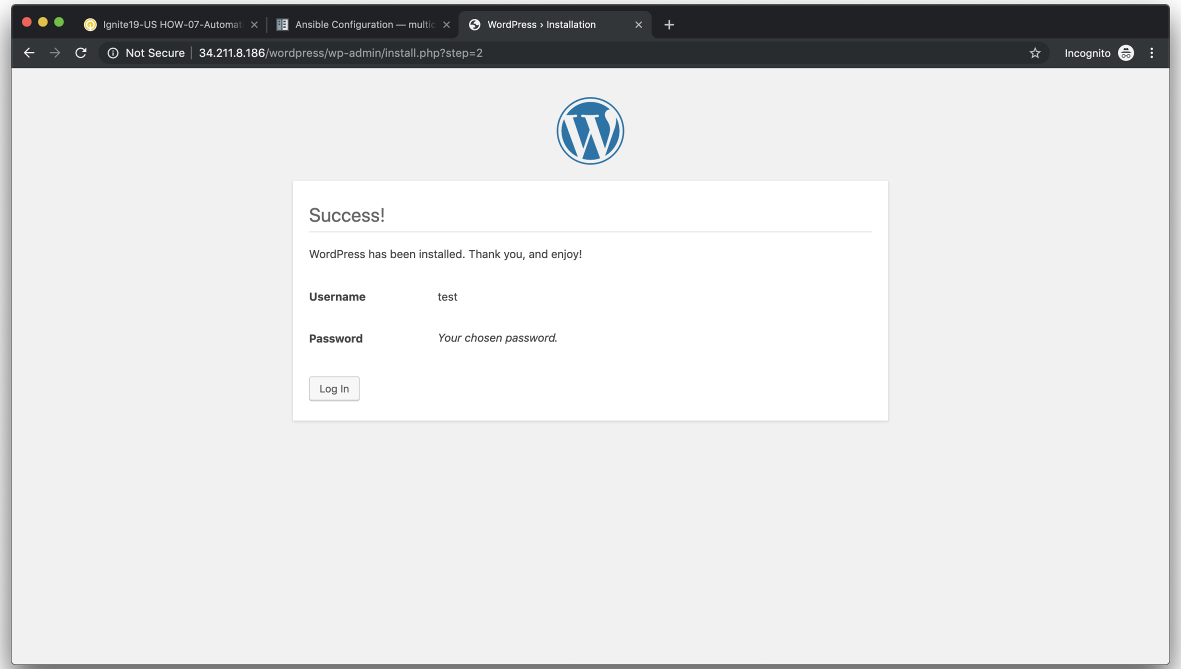
Your Email: test@example.com

Double-check your email address before continuing.

Search Engine Visibility: □ Discourage search engines from indexing this site

It is up to search engines to honor this request.

Install WordPress


Ignite19-US HOW-07-Automat
Ansible Configuration — multic
WordPress › Installation
Not Secure
34.211.8.186/wordpress/wp-admin/install.php?step=2
Incognito

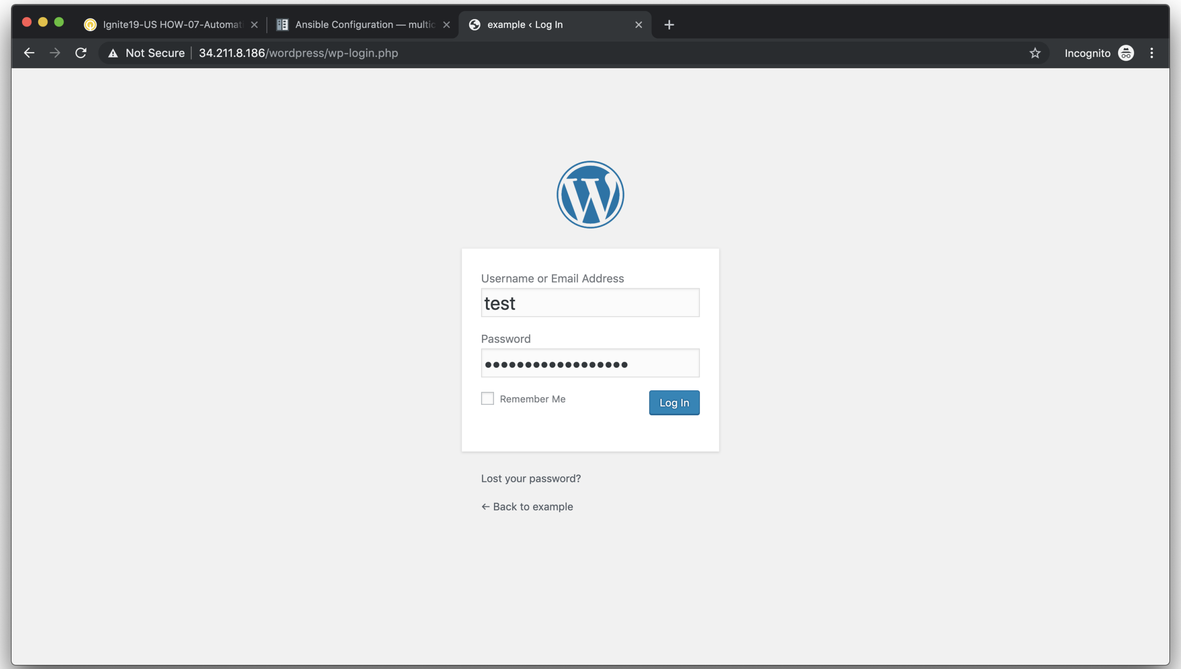
Success!

WordPress has been installed. Thank you, and enjoy!

Username: test

Password: *Your chosen password.*

Log In



Ignite19-US HOW-07-Automat
Ansible Configuration — multic
example ‹ Log In
Not Secure | 34.211.8.186/wordpress/wp-login.php
Incognito

Username or Email Address

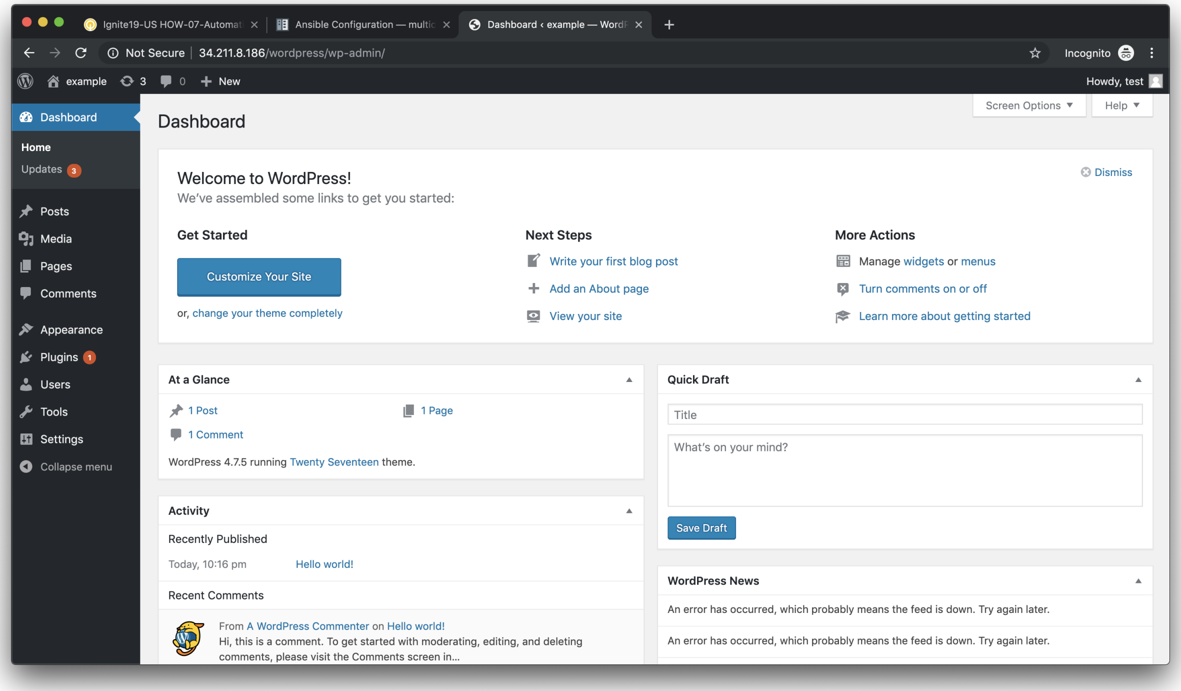
test

Password

●●●●●●●●●●●●●●●●●●

□ Remember Me

Log In
Lost your password?
← Back to example



Dashboard

Welcome to WordPress!

We've assembled some links to get you started:

Get Started

or, change your theme completely

Next Steps

- Write your first blog post
- Add an About page
- View your site

More Actions

- Manage widgets or menus
- Turn comments on or off
- Learn more about getting started

At a Glance

1 Post 1 Page

1 Comment

WordPress 4.7.5 running Twenty Seventeen theme.

Activity

Recently Published

Today, 10:16 pm Hello world!

Recent Comments

From A WordPress Commenter on Hello world!

Hi, this is a comment. To get started with moderating, editing, and deleting comments, please visit the Comments screen in…

Quick Draft

Title

What's on your mind?

WordPress News

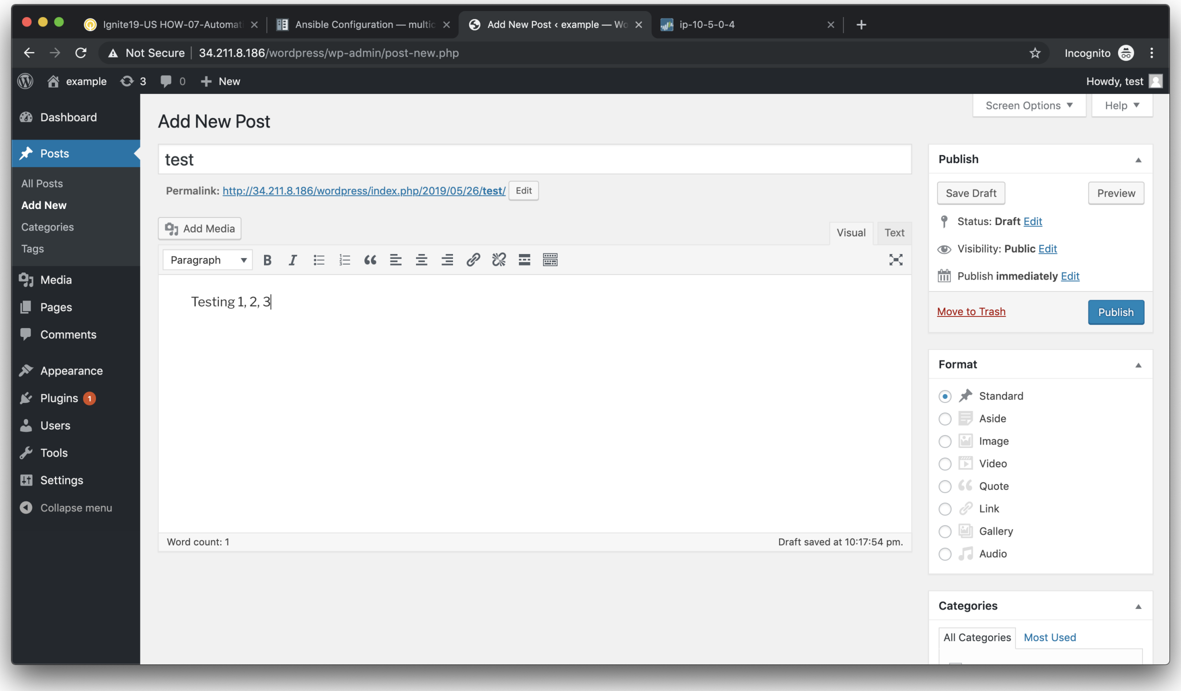
An error has occurred, which probably means the feed is down. Try again later.

An error has occurred, which probably means the feed is down. Try again later.

3.10.3 Post a blog article

Now that you've successfully logged into the WordPress administrator dashboard, let's post an update to the blog.

Click on **Write your first blog post** under the **Next Steps** section. You will be presented with the **Add New Post** editor.



test

Permalink: http://34.211.8.186/wordpress/index.php/2019/05/26/test/ Edit

Testing 1, 2, 3

Word count: 1

Draft saved at 10:17:54 pm.

Status: Draft Edit

Visibility: Public Edit

Publish immediately Edit

● Standard

Enter a title for your post and some sample content. Then click on **Publish** to post the update.

You can then click on **Preview** to see the published blog update.

3.10.4 Verify firewall rule matches

Now that we've confirmed the WordPress application is working properly, let's see what is happening with our firewall rules.

Log into the firewall administrator web interface at `https://<firewall-management-ip>` and navigate to **Policies > Security**.

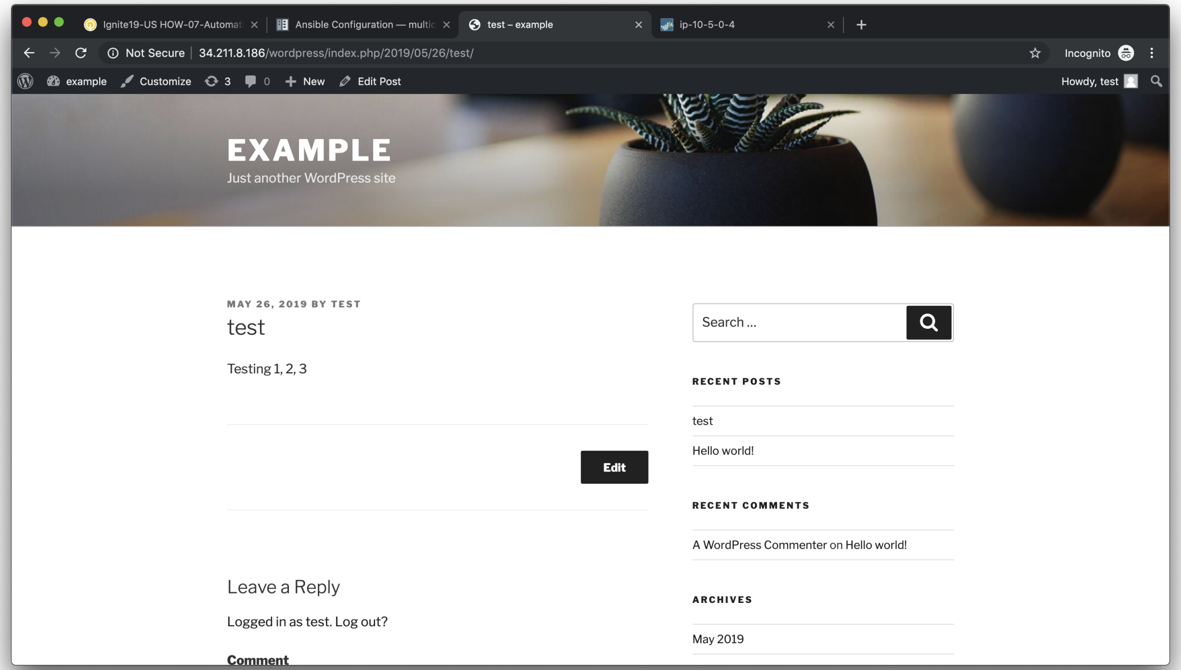
If you scroll to the right you will see details on the security rules that are being hit.

Scroll back to the left, find the security rule entitled *Allow web inbound*. Then click on the drop-down menu icon to the right of the rule name and select **Log Viewer*.

You will see all of the logs associated with inbound web traffic. Notice the applications identified are *web-browsing* and *blog-posting*.

Note: You may find source IPs other than your own as the web server is open to the public and will likely be discovered by web crawlers and other discovery tools aimed at public cloud providers.

Navigate back to **Policies > Security** and click on the **Log Viewer** for the *Allow web to db* rule.


Ignite19-US HOW-07-Automat
Ansible Configuration — multic
test – example
ip-10-5-0-4
Not Secure
34.211.8.186/wordpress/index.php/2019/05/26/test/
Incognito
example
Customize
3
0
New
Edit Post
Howdy, test

EXAMPLE

Just another WordPress site

MAY 26, 2019 BY TEST

test

Testing 1, 2, 3

Edit

Leave a Reply

Logged in as test. Log out?

Comment

Search …

RECENT POSTS

test

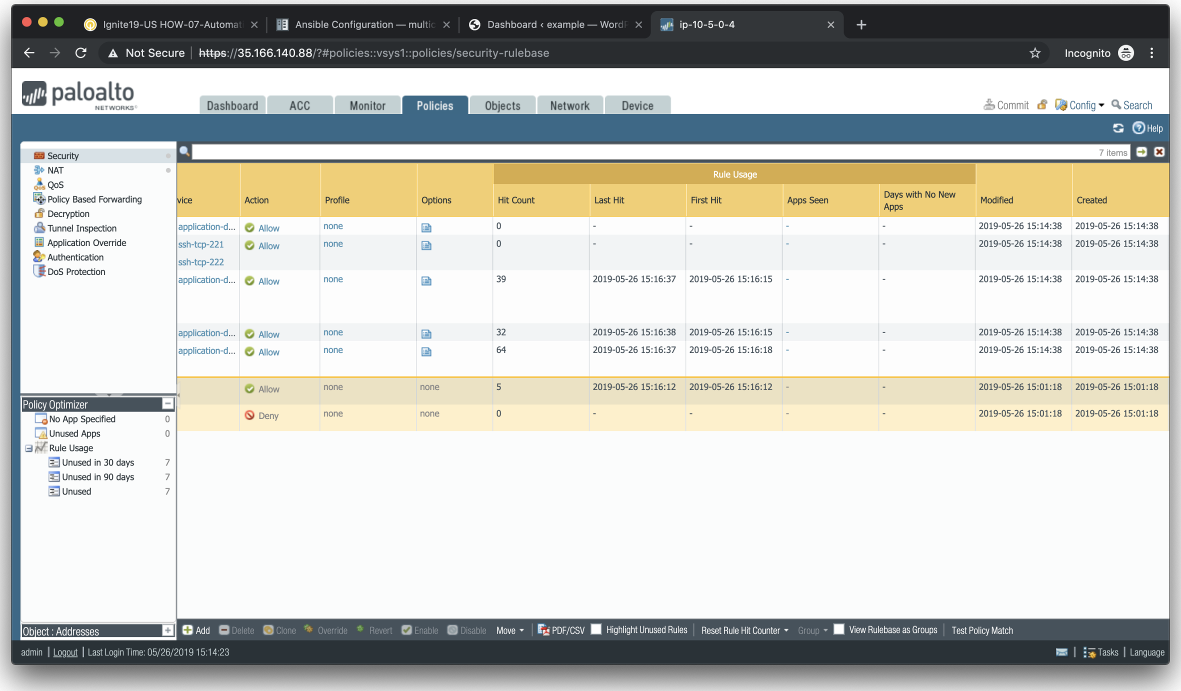
Hello world!

RECENT COMMENTS

A WordPress Commenter on Hello world!

ARCHIVES

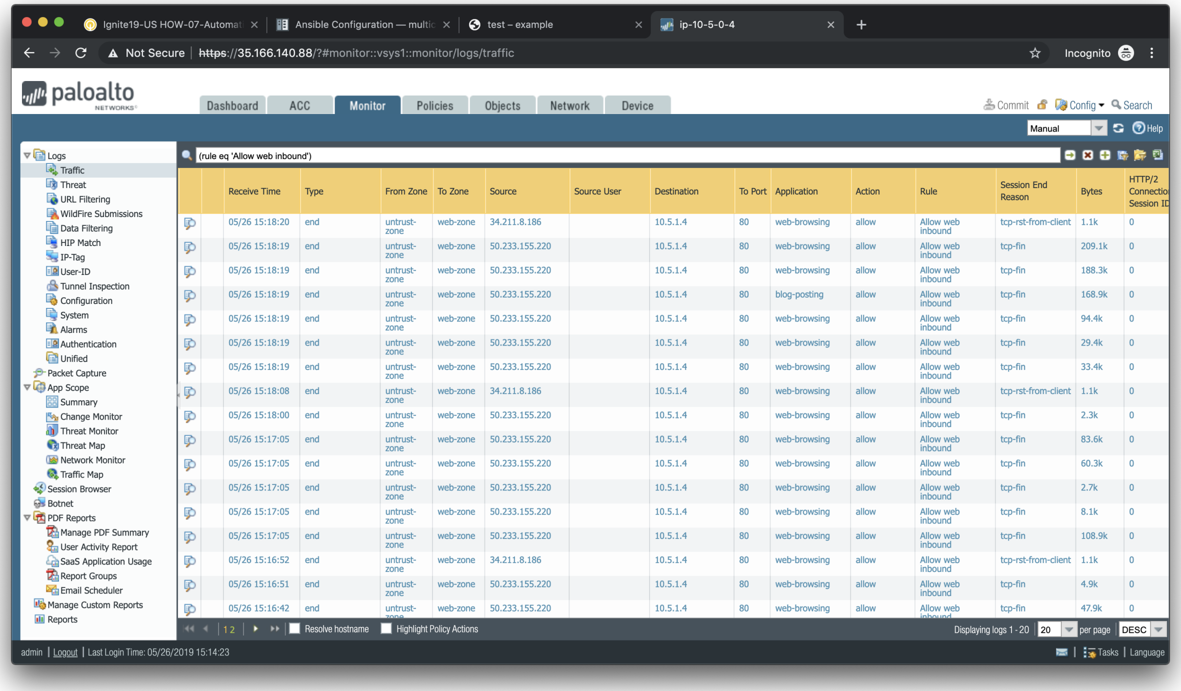
May 2019



Policy Optimizer

No App Specified	0
Unused Apps	0
Rule Usage	
Unused in 30 days	7
Unused in 90 days	7
Unused	7

				Rule Usage						
vice	Action	Profile	Options	Hit Count	Last Hit	First Hit	Apps Seen	Days with No New Apps	Modified	Created
application-d...	Allow	none	▤	0	-	-	-	-	2019-05-26 15:14:38	2019-05-26 15:14:38
ssh-tcp-221 ssh-tcp-222	Allow	none	▤	0	-	-	-	-	2019-05-26 15:14:38	2019-05-26 15:14:38
application-d...	Allow	none	▤	39	2019-05-26 15:16:37	2019-05-26 15:16:15	-	-	2019-05-26 15:14:38	2019-05-26 15:14:38
application-d...	Allow	none	▤	32	2019-05-26 15:16:38	2019-05-26 15:16:15	-	-	2019-05-26 15:14:38	2019-05-26 15:14:38
application-d...	Allow	none	▤	64	2019-05-26 15:16:37	2019-05-26 15:16:18	-	-	2019-05-26 15:14:38	2019-05-26 15:14:38
	Allow	none	none	5	2019-05-26 15:16:12	2019-05-26 15:16:12	-	-	2019-05-26 15:01:18	2019-05-26 15:01:18
	Deny	none	none	0	-	-	-	-	2019-05-26 15:01:18	2019-05-26 15:01:18



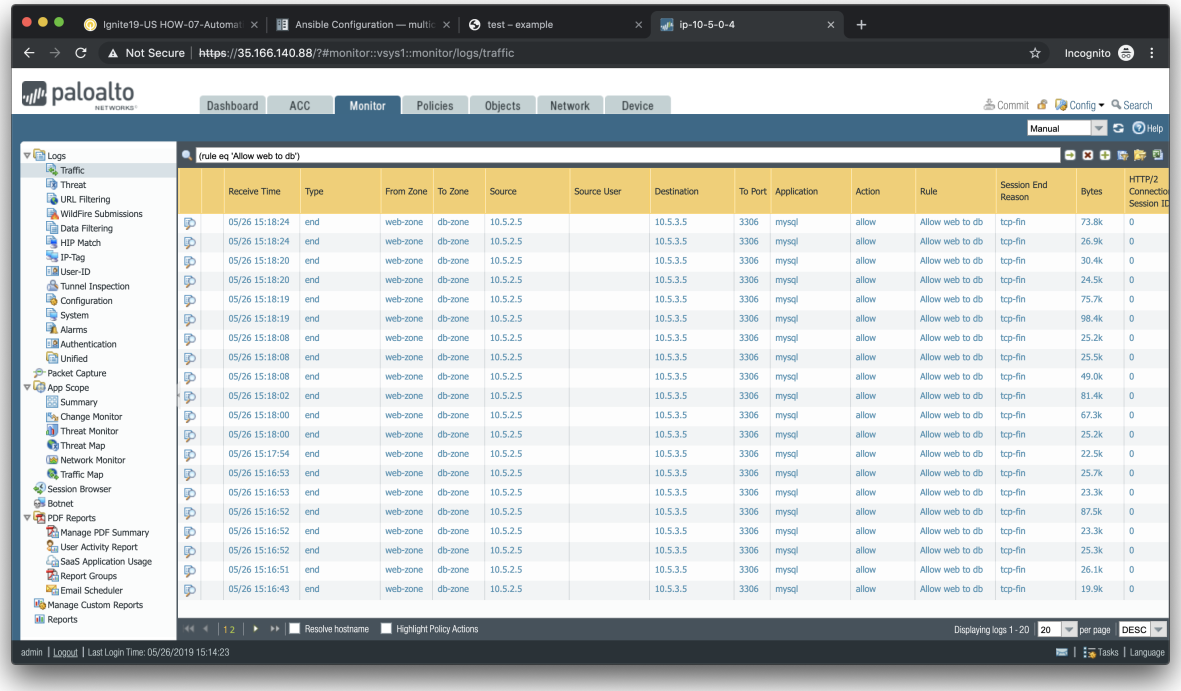
https://35.166.140.88/?#monitor::vsys1::monitor/logs/traffic

(rule eq 'Allow web inbound')

Receive Time	Type	From Zone	To Zone	Source	Source User	Destination	To Port	Application	Action	Rule	Session End Reason	Bytes	HTTP/2 Connectio Session ID
05/26 15:18:20	end	untrust-zone	web-zone	34.211.8.186		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-rst-from-client	1.1k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	209.1k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	188.3k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	blog-posting	allow	Allow web inbound	tcp-fin	168.9k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	94.4k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	29.4k	0
05/26 15:18:19	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	33.4k	0
05/26 15:18:08	end	untrust-zone	web-zone	34.211.8.186		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-rst-from-client	1.1k	0
05/26 15:18:00	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	2.3k	0
05/26 15:17:05	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	83.6k	0
05/26 15:17:05	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	60.3k	0
05/26 15:17:05	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	2.7k	0
05/26 15:17:05	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	8.1k	0
05/26 15:17:05	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	108.9k	0
05/26 15:16:52	end	untrust-zone	web-zone	34.211.8.186		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-rst-from-client	1.1k	0
05/26 15:16:51	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	4.9k	0
05/26 15:16:42	end	untrust-zone	web-zone	50.233.155.220		10.5.1.4	80	web-browsing	allow	Allow web inbound	tcp-fin	47.9k	0

1 2 □ Resolve hostname □ Highlight Policy Actions Displaying logs 1 - 20 | 20 per page | DESC

admin | Logout | Last Login Time: 05/26/2019 15:14:23



https://35.166.140.88/?#monitor::vsys1::monitor/logs/traffic

(rule eq 'Allow web to db')

Receive Time	Type	From Zone	To Zone	Source	Source User	Destination	To Port	Application	Action	Rule	Session End Reason	Bytes	HTTP/2 Connectio Session ID
05/26 15:18:24	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	73.8k	0
05/26 15:18:24	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	26.9k	0
05/26 15:18:20	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	30.4k	0
05/26 15:18:20	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	24.5k	0
05/26 15:18:19	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	75.7k	0
05/26 15:18:19	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	98.4k	0
05/26 15:18:08	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	25.2k	0
05/26 15:18:08	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	25.5k	0
05/26 15:18:08	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	49.0k	0
05/26 15:18:02	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	81.4k	0
05/26 15:18:00	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	67.3k	0
05/26 15:18:00	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	25.2k	0
05/26 15:17:54	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	22.5k	0
05/26 15:16:53	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	25.7k	0
05/26 15:16:53	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	23.3k	0
05/26 15:16:52	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	87.5k	0
05/26 15:16:52	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	23.3k	0
05/26 15:16:52	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	25.3k	0
05/26 15:16:51	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	26.1k	0
05/26 15:16:43	end	web-zone	db-zone	10.5.2.5		10.5.3.5	3306	mysql	allow	Allow web to db	tcp-fin	19.9k	0

1 2 □ Resolve hostname □ Highlight Policy Actions Displaying logs 1 - 20 | 20 per page | DESC

admin | Logout | Last Login Time: 05/26/2019 15:14:23

You will see all of the MySQL (actually MariaDB) database traffic between the WordPress web server and the database backend.

3.11 Cloud Monitoring

In this activity you will:

- Create a VM Information Source (GCP)
- Create a VM Information Source (AWS)
- Verify cloud API connectivity

The automation tasks we've accomplished thus far have focused on deploying the VM-Series firewall and making changes to it externally via the API. We'll now shift our focus to how PAN-OS can leverage third-party APIs to monitor its environment and automatically respond to changes it observes.

3.11.1 Create a VM Information Source (GCP)

Note: If you are working on the AWS deployment you should skip ahead to *Create a VM Information Source (AWS)*.

We will be creating a VM Information Source on the firewall to monitor the GCP Compute Engine environment for meta-data about the running VM instances. Open a web browser and go to `https://<your-firewall-ip>`. You will log in with the following credentials.

- **Username:** `admin`
- **Password:** `Ignite2019!`

Once you have logged into the firewall, go to the **VM Information Sources** under the **Device** tab and click **Add**.

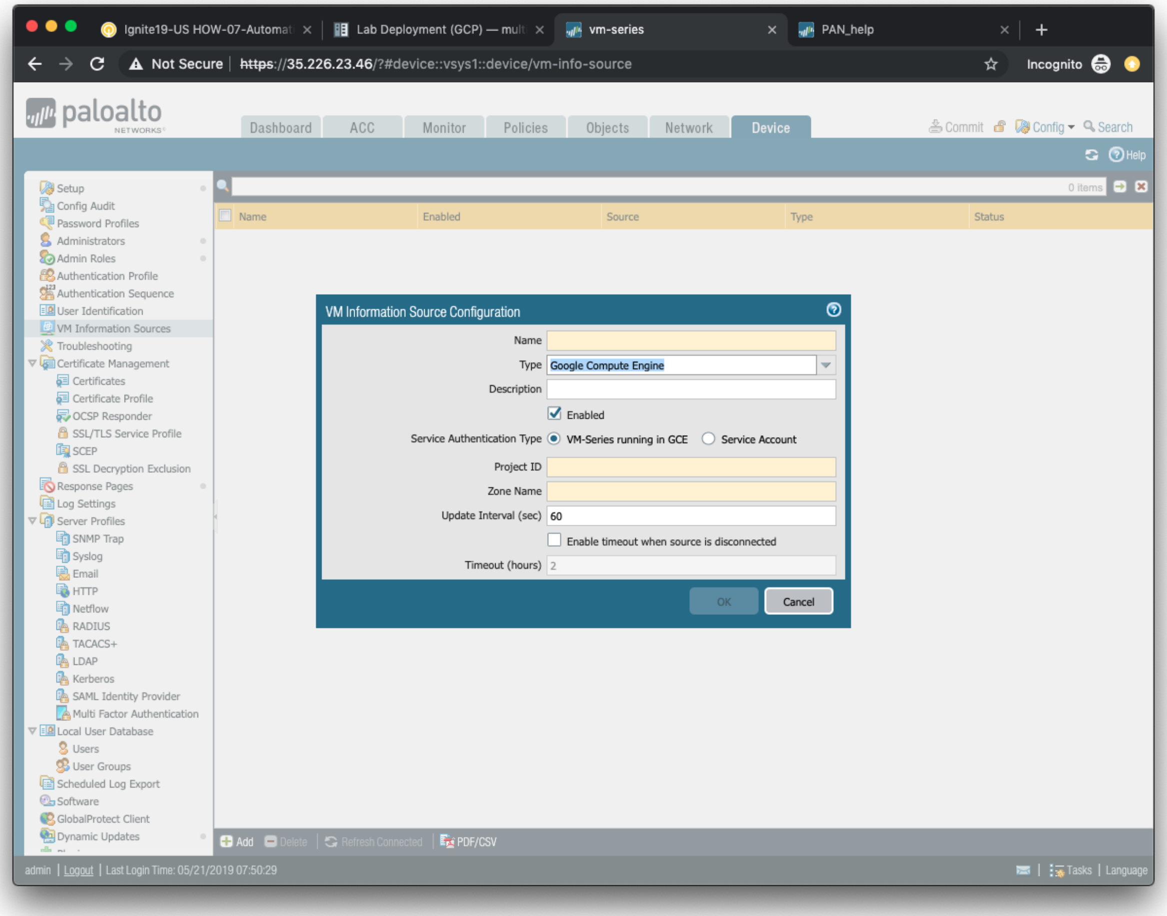
- Provide a name for your monitored source in the **Name** field.
- Ensure that *Google Compute Engine* is selected from the **Type** field selection.
- (optional) Provide a description of the monitored source in the **Description** field.
- Ensure that the **Enabled** button is selected.
- Select *VM-Series running in GCE* from the **Service Authorization Type** selector.
- The **Project ID** field will contain the Access Key ID provided in the Qwiklabs portal.
- The **Zone Name** field will contain the GCP zone in which the lab has been deployed.
- The **Update Interval**, and timeout fields can keep their default values.

Click **OK** to accept the configuration.

3.11.2 Create a VM Information Source (AWS)

Note: If you are working on the GCP deployment you should skip ahead to *Verify cloud API connectivity*.

We will be creating a VM Information Source on the firewall to monitor the AWS EC2 environment for meta-data about the running VM instances. Open a web browser and go to `https://<your-firewall-ip>`. You will log in with the following credentials.



VM Information Source Configuration

Name

Type: Google Compute Engine

Description

☑ Enabled

Service Authentication Type: ● VM-Series running in GCE ○ Service Account

Project ID

Zone Name

Update Interval (sec): 60

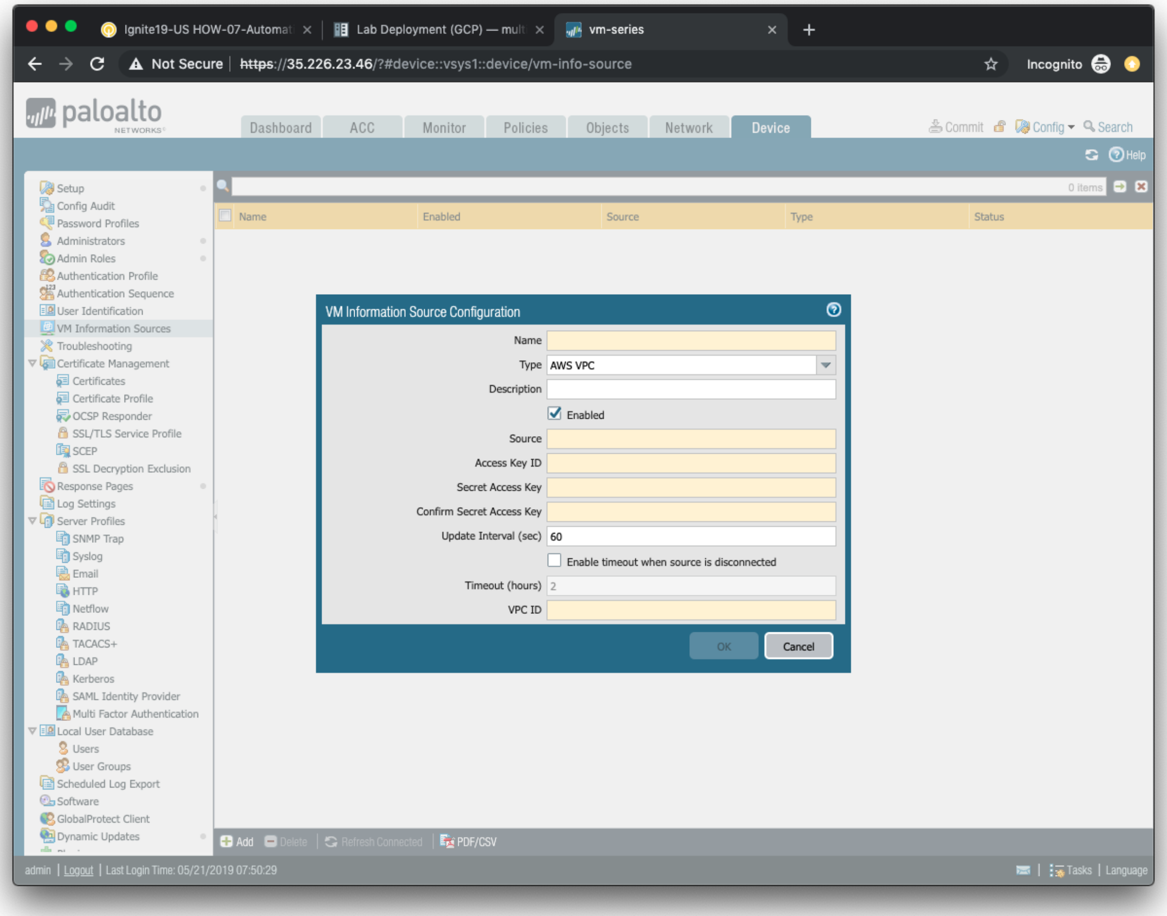
□ Enable timeout when source is disconnected

Timeout (hours): 2

OK Cancel

- **Username:** `admin`
- **Password:** `Ignite2019!`

Once you have logged into the firewall, go to the **VM Information Sources** under the **Device** tab and click **Add**.



https://35.226.23.46/?#device::vsys1::device/vm-info-source

VM Information Source Configuration

Name	
Type	AWS VPC
Description	
	☑ Enabled
Source	
Access Key ID	
Secret Access Key	
Confirm Secret Access Key	
Update Interval (sec)	60
	□ Enable timeout when source is disconnected
Timeout (hours)	2
VPC ID	

admin | Logout | Last Login Time: 05/21/2019 07:50:29

- Provide a name for your monitored source in the **Name** field.
- (optional) Provide a description of the monitored source in the **Description** field.
- Ensure that *AWS VPC* is selected from the **Type** field selection.
- Ensure that the **Enabled** button is selected.
- The **Source** field will contain the URI of the AWS region in which the lab is deployed. The format for this is *ec2.<your_AWS_region>.amazonaws.com*. For example, if the region is *us-west-2* then the URI will be *ec2.us-west-2.amazonaws.com*.
- The **Access Key ID** field will contain the Access Key ID provided in the Qwiklabs portal.
- The **Secret Access Key** field (and confirmation field) will contain the Secret Access Key provided in the Qwiklabs portal.
- The **Update Interval**, and timeout fields can keep their default values.
- The **VPC ID** field will contain the AWS VPC value that was output during the deployment phase. You can

change into the AWS deployment directory and display the Terraform output values with the following commands.

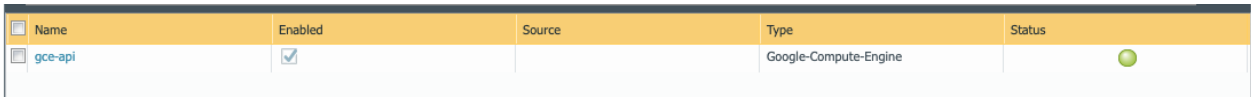
```
$ cd ~/multicloud-automation-lab/deployment/aws
$ terraform output
```

Click **OK** to accept the configuration.

3.11.3 Verify cloud API connectivity

Click **Commit** and commit the candidate configuration.

If the VM Information Source configuration was correct, you should see the status indicator for your source turn green.



☐ Name	Enabled	Source	Type	Status
☐ gce-api	☑		Google-Compute-Engine	●

If the status indicator is green, you can proceed to the next section.

3.12 Dynamic Address Groups

In this activity you will:

- Create a Dynamic Address Group
- Define the attribute match criteria
- Apply the Dynamic Address Group to a rule

Dynamic Address Groups are policy object groups whose members are ephemeral in nature. IP addresses are dynamically mapped to a Dynamic Address Group based on attribute match criteria. These attributes are discovered from instances deployed in cloud environments and learned via cloud provider APIs.

3.12.1 Create a Dynamic Address Group

Navigate to **Objects > Address Groups** in the firewall web interface.

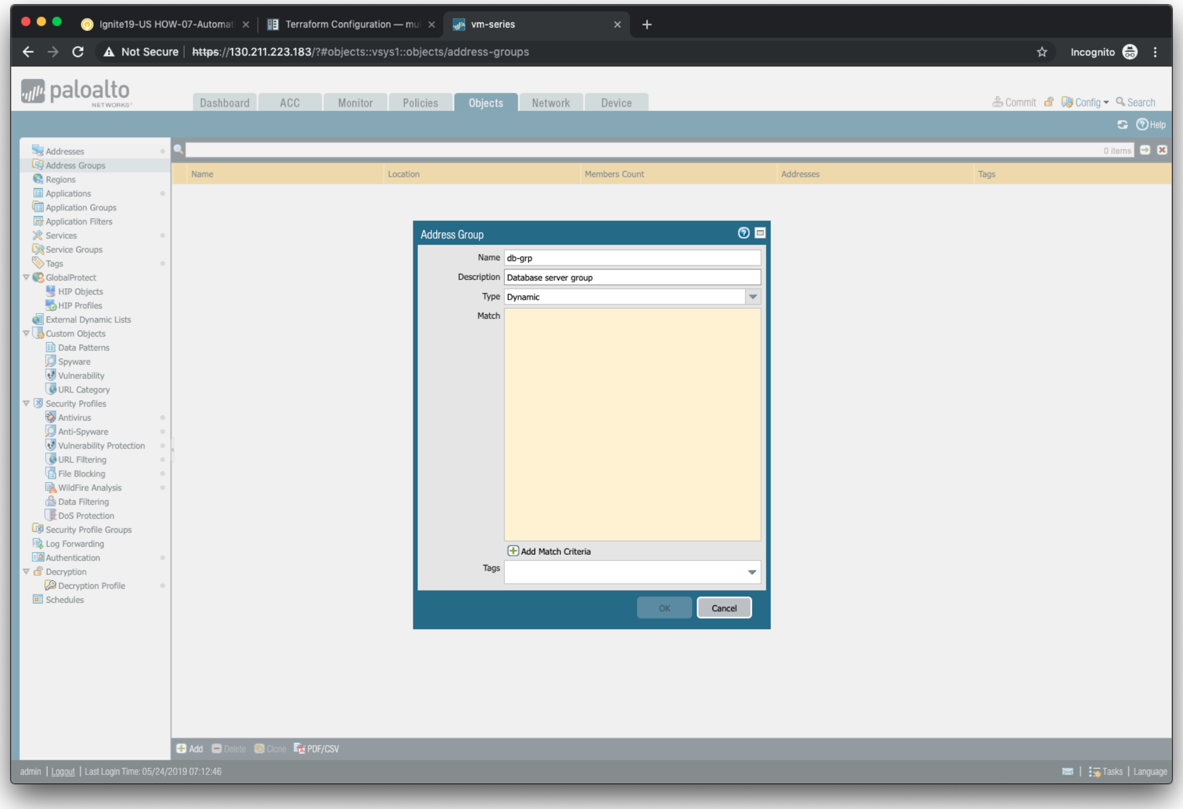
Click **Add** to create a new Dynamic Address Group.

In the **Address Group** window:

- Assign the name `db-grp` to the address groups.
- (optional) Provide a description of the address group.
- Select *Dynamic* from the **Type** drop-down menu.
- Click on **Add Match Criteria** to view the available attributes.

3.12.2 Define the attribute match criteria

The attributes displayed are discovered from the cloud provider API and are refreshed every *60 seconds*. You will select the attributes that will need to be matched in order to associate a VM instance to your Dynamic Address Group.

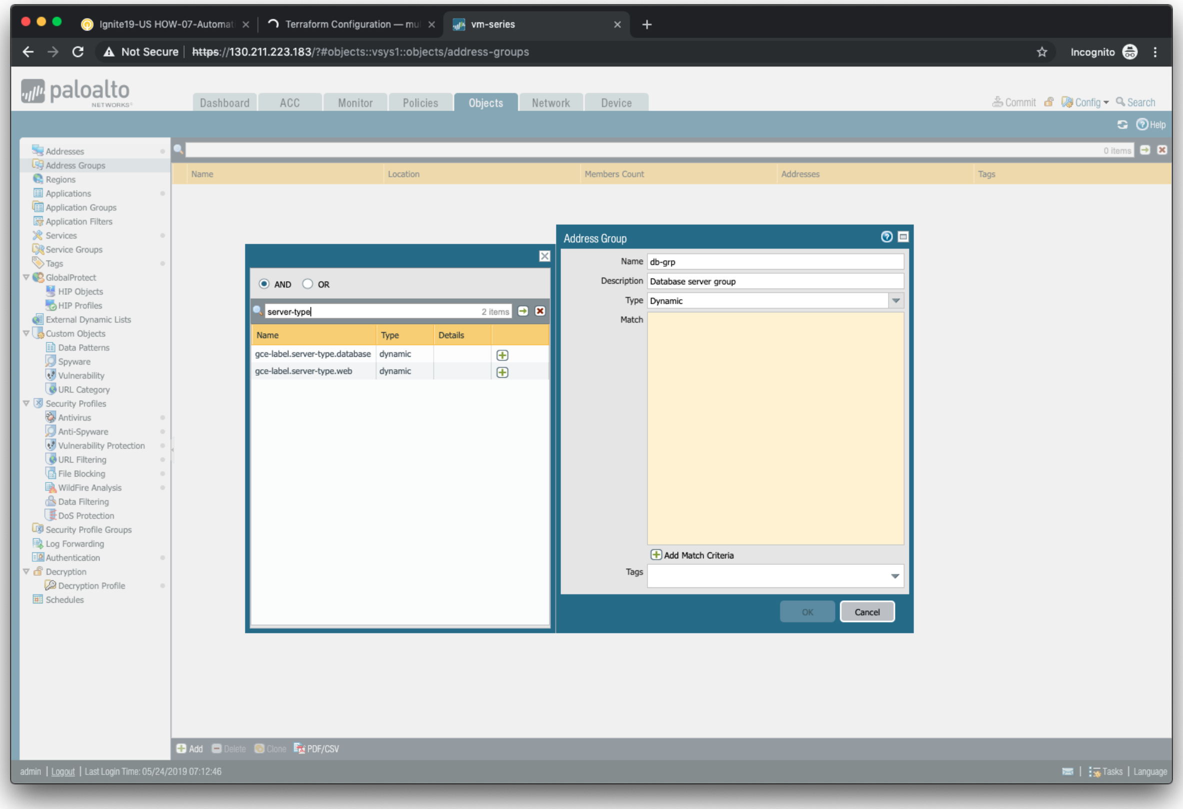


Name db-grp

Description Database server group

Type Dynamic

Most of the attributes displayed are not needed. However, each of the VM instances we've deployed have used a tag entitled `server-type`. Using the search bar at the top of the match criteria pop-up window, search for the term *server-type*. Then add the result that has a value of `database` to the match criteria list.



● AND ○ OR

server-type — 2 items

Name	Type	Details	
gce-label.server-type.database	dynamic		⊞
gce-label.server-type.web	dynamic		⊞

Address Group

Name: db-grp

Description: Database server group

Type: Dynamic

Match:

⊞ Add Match Criteria

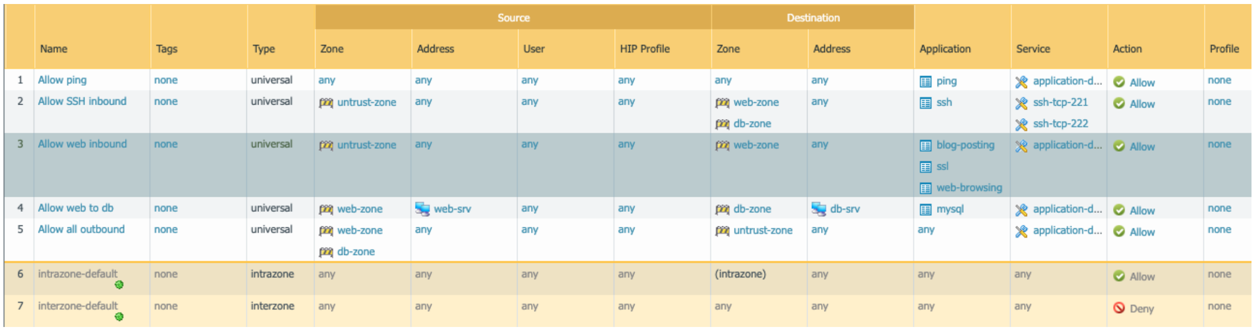
Tags:

OK Cancel

Click **OK** when you are done.

3.12.3 Apply the Dynamic Address Group to a rule

Now that we've defined a VM Information Source and a Dynamic Address Group, let's put them to use. Navigate to **Policies > Security** in the firewall web interface.



				Source				Destination					
	Name	Tags	Type	Zone	Address	User	HIP Profile	Zone	Address	Application	Service	Action	Profile
1	Allow ping	none	universal	any	any	any	any	any	any	ping	application-d...	✓ Allow	none
2	Allow SSH inbound	none	universal	untrust-zone	any	any	any	web-zone db-zone	any	ssh	ssh-tcp-221 ssh-tcp-222	✓ Allow	none
3	Allow web inbound	none	universal	untrust-zone	any	any	any	web-zone	any	blog-posting ssl web-browsing	application-d...	✓ Allow	none
4	Allow web to db	none	universal	web-zone	web-srv	any	any	db-zone	db-srv	mysql	application-d...	✓ Allow	none
5	Allow all outbound	none	universal	web-zone db-zone	any	any	any	untrust-zone	any	any	application-d...	✓ Allow	none
6	intrazone-default	none	intrazone	any	any	any	any	(intrazone)	any	any	any	✓ Allow	none
7	interzone-default	none	interzone	any	any	any	any	any	any	any	any	🚫 Deny	none

Find the rule that allows *mysql* traffic from the `web-srv` address object in the `web-zone` to the `db-srv` address object in the `db-zone`.

Replace the `db-srv` destination with the `db-grp` Dynamic Address Group you've created.

Click **OK** and then commit your changes by clicking **Commit**.

3.13 Compute Scaling

In this activity you will:

- Determine Dynamic Address Group membership
- Scale out the database instances
- Confirm Dynamic Address Group changes

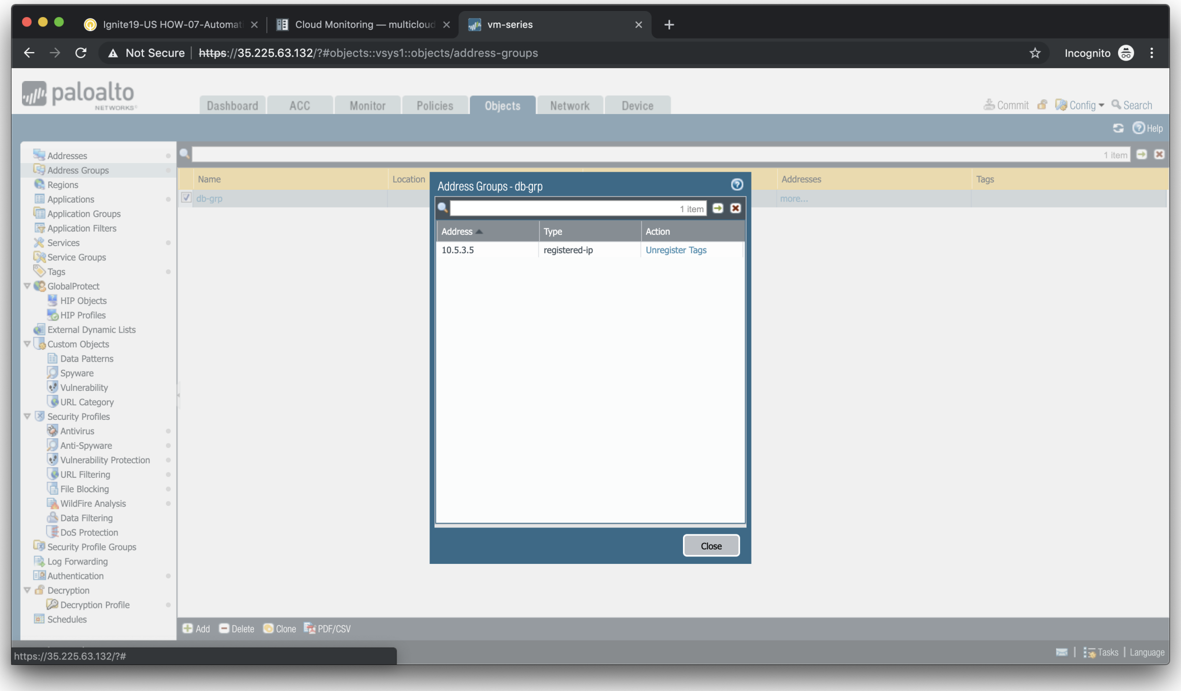
The combination of VM Information Sources and Dynamic Address Groups allows the firewall to respond to changes made to the cloud environment. In this lab scenario you will scale out the number of database instances used to support the web application. This should result in the automatic update of the Dynamic Address Group membership.

3.13.1 Determine Dynamic Address Group membership

First, we should confirm that the one database instance we've already deployed has already been mapped to the Dynamic Address Group based on it's `server-type` attribute.

Navigate to **Objects > Address Groups** in the firewall web interface and select the Dynamic Address Group `db-grp` that you previously created.

Under the *Addresses* column, click on the link entitled `more...`



Address Groups - db-grp

1 item

Address	Type	Action
10.5.3.5	registered-ip	Unregister Tags

Close

You should see the IP address `10.5.3.5`, which is the IP address of the existing database instance.

Click **Close** to close the pop-up window.

3.13.2 Scale out the database instances

To scale out the number of database instances we'll go back to our Terraform deployment.

For GCP:

```
$ cd ~/multicloud-automation-lab/deployment/gcp
```

For AWS:

```
$ cd ~/multicloud-automation-lab/deployment/aws
```

In the `main.tf` file there is a module called `scale` that is commented out. Open `main.tf` in a text editor and uncomment that entire section.

Save the file and exit.

By uncommenting the `scale` module you have just added a new module to the Terraform plan. This will require a re-initialization of the plan.

```
$ terraform init
```

You can now apply the Terraform plan.

```
$ terraform apply
```

This will result in four new database instances being added to the database subnet.

3.13.3 Confirm Dynamic Address Group changes

Now go back to the **Objects > Address Groups** section of the firewall web interface and click `more...` under the **Addresses** column of the `db-grp` entry.

You should now see a total of five IP addresses as members of the `db-grp` Dynamic Address Group. These are now part of the destination match criteria for the databaase security rule.

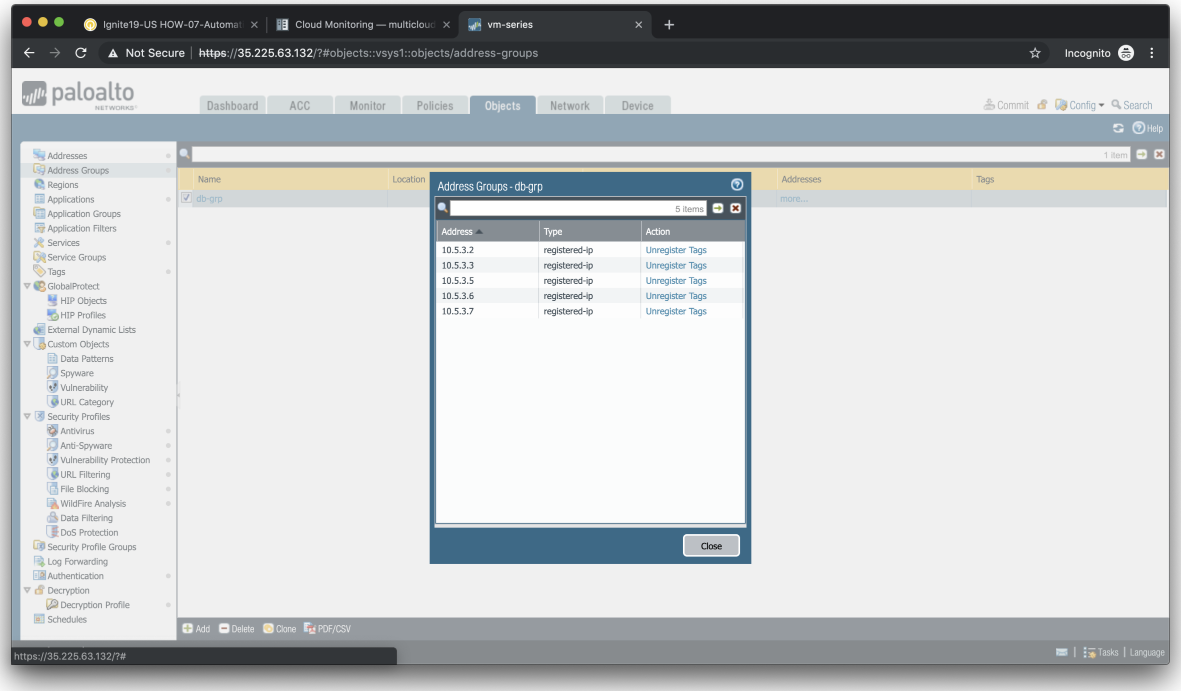
Note: Remember that the VM Information Source is polling the cloud provider API every *60 seconds*. If you do not see a total of five IP addresses in the Dynamic Address Group, close the window and click on `more...` again after a few moments.

3.14 Summary

Congratulations! You have completed the hands-on workshop. If you only completed the lab activities for one of the public cloud providers you are welcome to run back through the activities in the other cloud provider environment. What you'll find is that (with a few cloud-specific exceptions) the methods used to deploy and configure the VM-Series firewall are essentially the same.

3.14.1 What We've Accomplished

We've covered all three categories of network security automation:



Address Groups - db-grp

5 items

Address	Type	Action
10.5.3.2	registered-ip	Unregister Tags
10.5.3.3	registered-ip	Unregister Tags
10.5.3.5	registered-ip	Unregister Tags
10.5.3.6	registered-ip	Unregister Tags
10.5.3.7	registered-ip	Unregister Tags

- **Build:** We used Terraform to orchestrate the deployment of the lab environment. Rather than utilizing cloud-specific deployment tools such as AWS CloudFormation or Google Deployment Manager, we were able to use a common tool for both environments.
- **Run:** We used both Terraform and Ansible for configuring the VM-Series firewall instance. Both tools leverage the PAN-OS XML API and have libraries that allow those tools to communicate with the API.
- **Respond:** We leveraged two PAN-OS features, VM Information Sources and Dynamic Address Groups, to identify changes in the cloud provider environment and automatically adapt to those changes.

3.15 Tool Comparison

At this point, you've now used both Ansible and Terraform to configure a Palo Alto Networks firewall. Though you've used these two tools to deploy the same configuration, they differ in some important ways. Let's discuss some of those differences now.

3.15.1 Strengths

Both tools have a certain reputation associated with them. Terraform is known more for its power in deployment, while Ansible is known more for its flexibility in configuration. Both products can do both jobs just fine.

Regardless of their reputations, the most important part is that Palo Alto Networks has integrations with both, and either way will get the job done. It's just a matter of preference.

3.15.2 Idempotence

Both Terraform and Ansible support idempotent operations. Saying that an operation is idempotent means that applying it multiple times will not change the result. This is important for automation tools because they can be run to change configuration **and** also to verify that the configuration actually matches what you want. You can run `terraform apply` continuously for hours, and if your configuration matches what is defined in the plan, it won't actually change anything.

3.15.3 Commits

As you've probably noticed, a lot of the Ansible modules allow you to commit directly from them. There is also a dedicated Ansible module that just does commits, containing support for both the firewall and Panorama.

So how do you perform commits with Terraform? Currently, there is no support for commits inside the Terraform ecosystem, so they have to be handled externally. Lack of finalizers are a known shortcoming for Terraform and, once it is addressed, support for it can be added to the provider. In the meantime, we've provides some Golang code in the appendix (*Terraform and Commits*) that you can use to fill the gap.

3.15.4 Operational Commands

Ansible currently has a `panos_op` module allows users to run arbitrary operational commands. An operational command could be something that just shows some part of the configuration, but it can also change configuration. Since Ansible doesn't store state, it doesn't care what the invocation of the `panos_op` module results in.

This is a different story in Terraform. The basic flow of Terraform is that there is a read operation that determines if a create, update, or delete needs to take place. But operational commands as a whole don't fit as neatly into this paradigm. What if the operational command is just a read? What if the operational command makes a configuration change, and should only be executed once? This uncertainty is why support for operational commands in Terraform is not currently in place.

3.15.5 Facts / Data Sources

Terraform may not have support for arbitrary operational commands, but it does have a data source that you can use to retrieve specific parts of a `show system info` command from the firewall or Panorama and then use that in your Terraform plan file. This same thing is called "facts" in Ansible. Many of the Ansible modules for PAN-OS support the gathering of facts that may be stored and referenced in an Ansible playbook.

3.16 Cleaning Up

In this activity you will:

- Destroy the lab deployment

3.16.1 Destroy the lab deployment

When deploying infrastructure in the public cloud it is important to tear it down when it is no longer needed. Otherwise you will end up paying for services that are no longer needed. We'll need to go back to the deployment directory and use Terraform to destroy the infrastructure we deployed at the start of the lab.

Change into the `deployment` directory.

For GCP:

```
$ cd ~/multicloud-automation-lab/deployment/gcp
```

For AWS:

```
$ cd ~/multicloud-automation-lab/deployment/aws
```

Tell Terraform to destroy the contents of its plan files.

```
$ terraform destroy
```

Note: The Qwiklabs training environment will actually take care of destroying everything that we've created at the end of this lab, but it is a good habit to be aware of the cloud resources you've deployed and to destroy it when you are done with it.

3.17 Further Reading

3.17.1 Terraform

- Terraform Documentation
- Terraform panos Provider
- Terraform: Up & Running

3.17.2 Ansible

- Ansible Docs
- ansible-pan
- Ansible: Up & Running

3.18 Terraform and Commits

One thing to know when working with Terraform is that it does not have support for committing your configuration. To commit your configuration, you can use the following Golang code.

```
package main

import (
    "encoding/json"
    "flag"
    "log"
    "os"

    "github.com/PaloAltoNetworks/pango"
)

type Credentials struct {
```

(continues on next page)

(continued from previous page)

```
    Hostname string `json:"hostname"`
    Username string `json:"username"`
    Password string `json:"password"`
    ApiKey string `json:"api_key"`
    Protocol string `json:"protocol"`
    Port uint `json:"port"`
    Timeout int `json:"timeout"`
}

func getCredentials(configFile, hostname, username, password, apiKey string)␣
↪(Credentials) {
    var (
        config Credentials
        val string
        ok bool
    )

    // Auth from the config file.
    if configFile != "" {
        fd, err := os.Open(configFile)
        if err != nil {
            log.Fatalf("ERROR: %s", err)
        }
        defer fd.Close()

        dec := json.NewDecoder(fd)
        err = dec.Decode(&config)
        if err != nil {
            log.Fatalf("ERROR: %s", err)
        }
    }

    // Auth from env variables.
    if val, ok = os.LookupEnv("PANOS_HOSTNAME"); ok {
        config.Hostname = val
    }
    if val, ok = os.LookupEnv("PANOS_USERNAME"); ok {
        config.Username = val
    }
    if val, ok = os.LookupEnv("PANOS_PASSWORD"); ok {
        config.Password = val
    }
    if val, ok = os.LookupEnv("PANOS_API_KEY"); ok {
        config.ApiKey = val
    }

    // Auth from CLI args.
    if hostname != "" {
        config.Hostname = hostname
    }
    if username != "" {
        config.Username = username
    }
    if password != "" {
        config.Password = password
    }
    if apiKey != "" {
```

(continues on next page)

(continued from previous page)

```
        config.ApiKey = apiKey
    }

    if config.Hostname == "" {
        log.Fatalf("ERROR: No hostname specified")
    } else if config.Username == "" && config.ApiKey == "" {
        log.Fatalf("ERROR: No username specified")
    } else if config.Password == "" && config.ApiKey == "" {
        log.Fatalf("ERROR: No password specified")
    }

    return config
}

func main() {
    var (
        err error
        configFile, hostname, username, password, apiKey string
        job uint
    )

    log.SetFlags(log.Ldate | log.Ltime | log.Lmicroseconds)

    flag.StringVar(&configFile, "config", "", "JSON config file with panos connection␣
↪info")
    flag.StringVar(&hostname, "host", "", "PAN-OS hostname")
    flag.StringVar(&username, "user", "", "PAN-OS username")
    flag.StringVar(&password, "pass", "", "PAN-OS password")
    flag.StringVar(&apiKey, "key", "", "PAN-OS API key")
    flag.Parse()

    config := getCredentials(configFile, hostname, username, password, apiKey)

    fw := &pango.Firewall{Client: pango.Client{
        Hostname: config.Hostname,
        Username: config.Username,
        Password: config.Password,
        ApiKey: config.ApiKey,
        Protocol: config.Protocol,
        Port: config.Port,
        Timeout: config.Timeout,
        Logging: pango.LogOp | pango.LogAction,
    }}
    if err = fw.Initialize(); err != nil {
        log.Fatalf("Failed: %s", err)
    }

    job, err = fw.Commit(flag.Arg(0), true, true, false, true)
    if err != nil {
        log.Fatalf("Error in commit: %s", err)
    } else if job == 0 {
        log.Printf("No commit needed")
    } else {
        log.Printf("Committed config successfully")
    }
}
```

This code reads the hostname, username, and password from the environment variables we set earlier.

You will need to do the following to compile and run this code:

1. Open a text editor, add the code above to it and save the file as `commit.go`.
2. Install the Go libraries for PAN-OS.

```
$ go get github.com/PaloAltoNetworks/pango
```

3. Compile the source code.

```
$ go build commit.go
```

4. Run the executable (using your existing environment variables).

```
$ ./commit <optional commit comment>
```