

AKS-Kubernetes-Lab/Demo



Securing your container workloads in Kubernetes

<http://www.paloaltonetworks.com>

© 2018 Palo Alto Networks. Proprietary and Confidential

Table of Contents

About the Azure Kubernetes Service Terraform Template	3
Support Policy.....	4
Instances Used	4
Prerequisites.....	4
Download GitHub files	5
Azure Service Principal Creation	6
Bootstrap storage account creation	8
SSH keys	16
Deploy the Terraform Template.....	18
Review what was deployed.....	21
Task 1 – Look around Azure console.....	21
Task 2 – Review the Kubernetes Cluster.....	26
Task 3 – Connect to the Kubernetes Cluster.....	27
Task 3 – Log into the firewall.....	28
Launch a two tiered WordPress application	32
Task 1 – WordPress Application Deployment YAML file.....	32
Task 2 – Launch the Application	35
Launch a two tiered Guestbook application.....	37
Task 1 – Guestbook Application Deployment YAML file.....	37
Task 2 – Launch the Application	40
Explore the newly deployed applications	41
Securing Inbound Traffic.....	44
Task 1 – Azure Application Gateway IP Address	44
Task 2 – Update the Firewall's Address Objects.....	45
Task 3 – Connect to the Guestbook Frontend.....	47
Securing Outbound Traffic.....	51
Task 1 – Add Outbound Route.....	51
Lab Termination.....	54
Conclusion	56

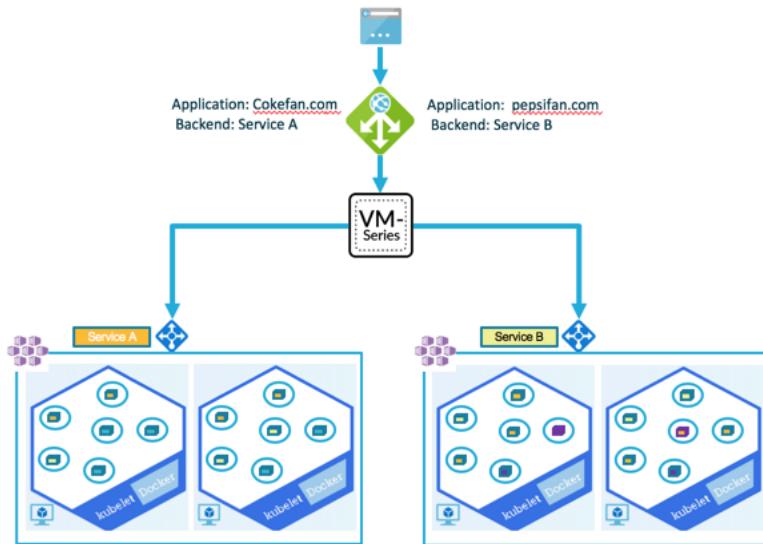
About the Azure Kubernetes Service Terraform Template

Azure Kubernetes Service (AKS) Terraform Templates are files that can deploy, configure, and launch AZURE resources such as Resource Groups, VNets, subnets, security groups, application gateways, route tables, Kubernetes clusters, and more. These templates are used for ease of deployment and are key to any cloud deployment model.

For more information on Templates refer to Google's documentation

<https://docs.microsoft.com/en-us/azure/terraform/>

This document will walk through deploying a Terraform template and a few Kubernetes(k8s) pods that host a few two-tier web applications. This template will create two resource groups. One that has the infrastructure including the bootstrapped VM-Series Firewall and another with the k8s cluster resources. After completing this guide, the following infrastructure will be instantiated:



Support Policy

This template is released under an as-is, best effort, support policy. These scripts should be seen as community supported and Palo Alto Networks will contribute our expertise as and when possible. We do not provide technical support or help in using or troubleshooting the components of the project through our normal support options such as Palo Alto Networks support teams, or ASC (Authorized Support Centers) partners and backline support options. The underlying product used (the VM-Series firewall) by the scripts or templates are still supported, but the support is only for the product functionality and not for help in deploying or using the template or script itself.

Instances Used

When deploying this Terraform template the following machine types are used:

Instance	Machine Type	QTY
PayGo Bundle 1 – VM-Series Firewall	Standard_D3_v2	1
Kubernetes Ubuntu Cluster Nodes	Standard_D3_v2	2
Internal Load Balancer		1
Application Gateway		1

Note: There are Azure costs associated with each machine type launched, please refer to the Microsoft instance pricing page <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/>

Prerequisites

Here are the prerequisites required to successfully launch this template:

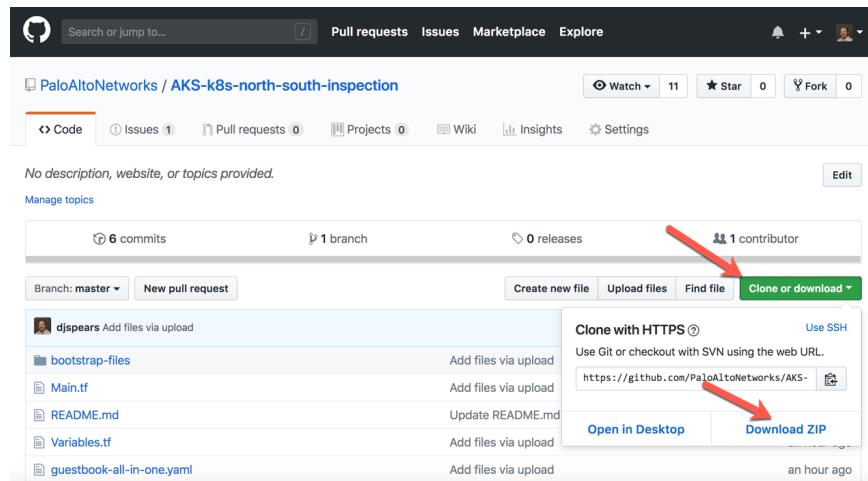
- Terraform application - Instructions on the installation can be found here:
<https://www.terraform.io/intro/getting-started/install.html>
- Azure account- Account creation instructions can be found here: <https://azure.microsoft.com/en-us/resources/videos/sign-up-for-microsoft-azure/>
- Azure command-line tool – Instructions for doing this can be found here:
<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>
- Kubernetes command-line tool – Instructions for doing this can be found here:
<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

Download GitHub files

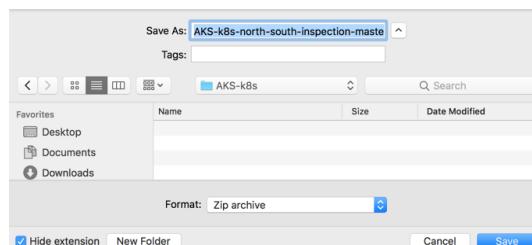
In this activity, you will:

Download a zip copy of the GitHub files used for this lab

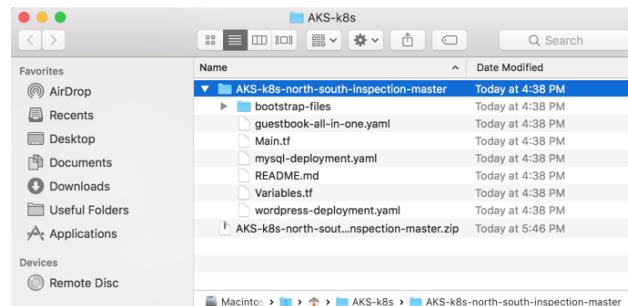
During this lab, the Terraform templates and Kubernetes (k8s) command will be executed from a local computer. This lab requires some customization of the terraform files. To download the files from GitHub, click on the Clone or download drop down and select Download ZIP.



Save the zip file to a new directory. This directory will be used to deploy the Terraform template and will automatically keep the Terraform state files so the deployment can be managed in the future:



Unzip the files:



Azure Service Principal Creation

In this activity, you will:

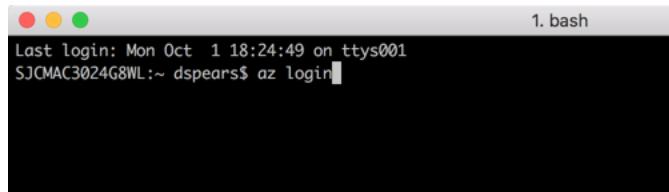
Authenticate to an Azure subscription via the Azure command line tool

Create a Service Principal with the appropriate RBAC to deploy a kubernetes (k8s) cluster

Update the Terraform Variables.tf file with the Service Principal information needed to execute

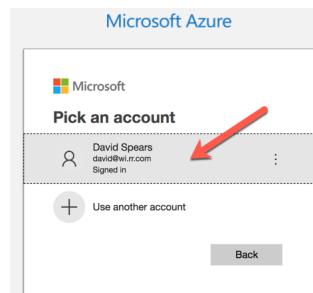
Microsoft has documented the steps to create a service principal that can be used to deploy a k8s cluster. That document can be found here: <https://docs.microsoft.com/en-us/azure/container-service/kubernetes/container-service-kubernetes-service-principal>

This guide assumes that the perquisites have been completed and the Azure command line tool has been installed. Open a terminal window and type the command **az login** to authenticate the command line tool to the appropriate subscription:



A terminal window titled "1. bash" showing the command "az login" being typed. The terminal shows the user's last login details: "Last login: Mon Oct 1 18:24:49 on ttys001" and the command "SJC MAC 3024G8WL:~ dspears\$ az login". The terminal window has a dark background and light-colored text.

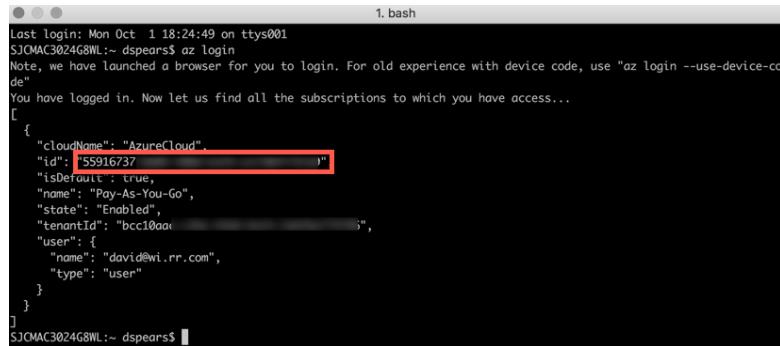
Next a browser window should open that will give the option to select the Azure account associate with the subscription that will get the deployment:



Once the account has been selected, the following message will appear:

You have logged into Microsoft Azure!
You can close this window, or we will redirect you to the [Azure CLI documents](#) in 10 seconds.

Check the terminal window. There should be confirmation that the login process was a success:

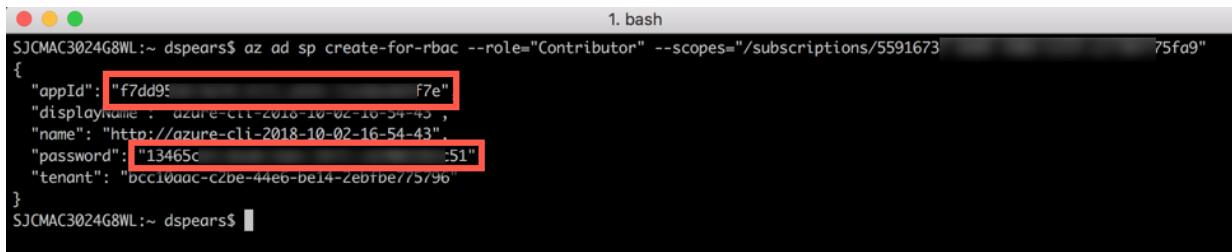


```
Last login: Mon Oct 1 18:24:49 on ttys001
SJCMAC3024G8WL:~ dspears$ az login
Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
You have logged in. Now let us find all the subscriptions to which you have access...
[{"cloudName": "AzureCloud", "id": "55916737", "isDefault": true, "name": "Pay-As-You-Go", "state": "Enabled", "tenantId": "bcc10aac-..."}, {"user": { "name": "david@wi...rr.com", "type": "user"}}]
SJCMAC3024G8WL:~ dspears$
```

Copy the “id” from the output. This is the subscription id for the service principal. To be able to deploy a k8s cluster in Azure the service principal must have the “contributor” role. Use the following command to create the service principal:

```
$ az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/<id>"
```

where “id” is the subscription id copied from the last step:



```
SJCMAC3024G8WL:~ dspears$ az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/55916737"
{
  "appId": "f7dd95...f7e",
  "displayName": "azure-cli-2018-10-02-16-54-43",
  "name": "http://azure-cli-2018-10-02-16-54-43",
  "password": "13465c...51",
  "tenant": "bcc10aac-c2be-44eb-be14-zebfbe775796"
}
SJCMAC3024G8WL:~ dspears$
```

The Terraform deployment files consist of a main, variables, and output files. The Variables.tf file contains information that is easily modified and commonly changed for various situations. The variables in the Variables.tf file are used by the Main.tf file during deployment. Deploying this Terraform template in Azure does require modification of the Variable.tf file to include deployment-specific information.

Copy the “appId” and “password” fields from the service principal creation output. These are needed for the terraform script and need to be added to the Variables.tf file. Open an editor of your choice and update these fields and save the file:

```
1 // PROJECT Variables
2 variable "client_id" {
3     default = "<appId>"           ←
4 }
5 variable "client_secret" {
6     default = "<password>"       ←
7 }
8
9 variable "agent_count" {
10    default = 2
11 }
12
13 variable "ssh_public_key" {
14     default = "<path to public ssh key>" ←
15 }
16
17 variable "dns_prefix" {
18     default = "k8s-AZURE-HOW"
19 }
20
21 variable cluster_name {
22     default = "k8s-Cluster-MGMT"
23 }
24
25 variable resource_group_name {
26     default = "k8s-RG"
27 }
28
29 variable location {
```

```
1 // PROJECT Variables
2 variable "client_id" {
3     default = "f7dd95...F7e"
4 }
5
6 variable "client_secret" {
7     default = "13465c...:51"
8 }
9
10 variable "agent_count" {
11     default = 2
12 }
13
14 variable "ssh_public_key" {
15     default = "/AKS/AKS-PANFW-k8s/djs-gcp-keyb.pub"
16 }
17
18 variable "dns_prefix" {
19     default = "k8s-AZURE-HOW"
20 }
21
22 variable cluster_name {
23     default = "k8s-Cluster-MGMT"
24 }
25
26 variable resource_group_name {
27     default = "k8s-RG"
28 }
29
30 variable location {
```

Bootstrap storage account creation

In this activity, you will:

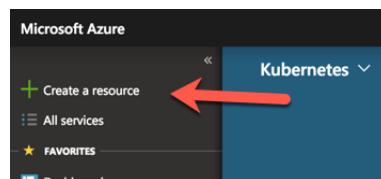
Create an Azure Resource Group and deploy a storage account

Create a file share with the folder structure needed to bootstrap the VM-Series Firewall

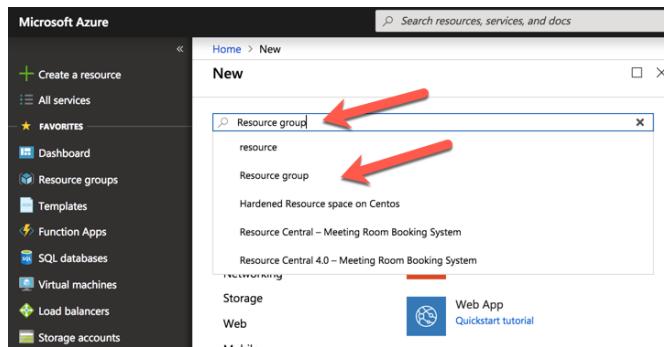
Copy the files to the Azure file share needed for bootstrapping

Update the Terraform Variables.tf file with the Azure storage access key that will allow the VM-Series Firewall to bootstrap

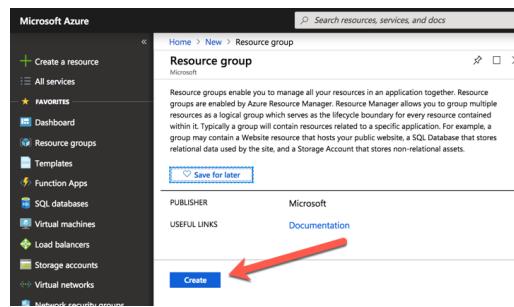
The terraform template is going to bootstrap the initial VM-Series firewall configuration. To accomplish this an Azure storage account will be created with the appropriate files. To start, open the Azure Portal and create a new resource group. Click on the “+ Create a resource” link:



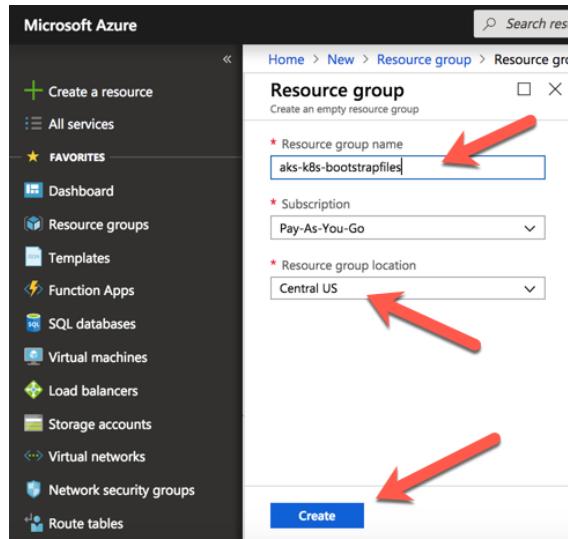
Next enter “Resource group” in the search and select Resource group:



Next select “Create” to create:



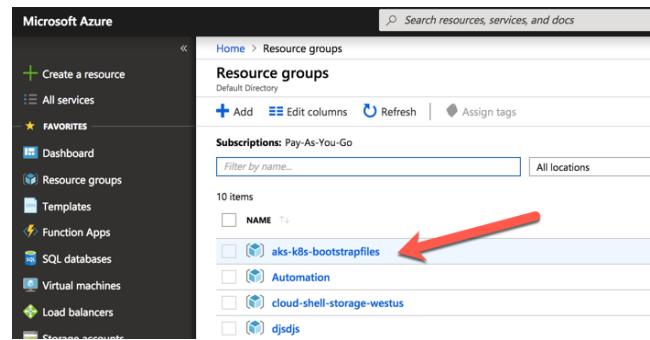
In the next window, create a resource group name and select the Resource group location. It is recommended for this lab to use the same location that the terraform script deploys in. The default setting is Central US. Click “Create” to create the Resource group.



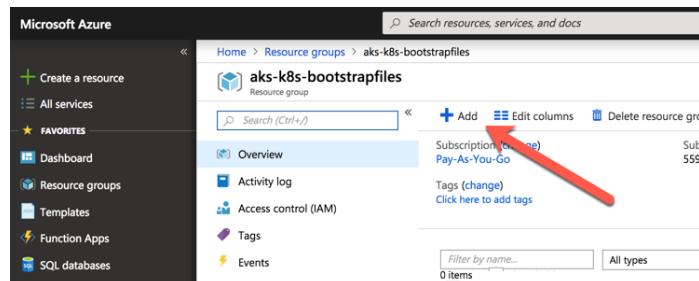
Navigate to the new Resource group. If a favorite is not available, click the “All Services” option on the left Nav and type “resource” in the All services search window. Click on Resource groups to open all the resources.



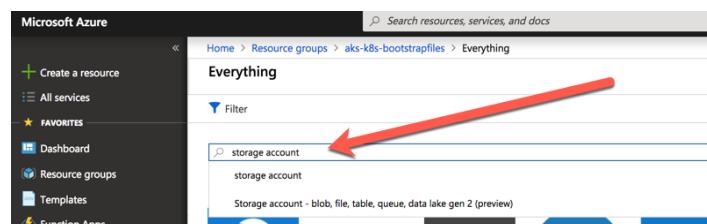
Now click the newly created Resource group:



Once in the resource group the next step is to create a storage account. Click on the plus sign to add a resource in the resource group:



Type storage account in the search field:



Select the Storage account published by Microsoft:

Next click “Create”:

Make sure the Resource group is correct. Enter a Storage account name and select the same location as the rest of the deployment. Finally click “Review and create”

Once the validation is complete, select Create:

After the deployment is complete, click on the go to resource button:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'Microsoft.StorageAccount-20181002173523'. The deployment status is shown as 'Your deployment is complete' with a green checkmark icon. A red arrow points to the 'Go to resource' button. Below the status, deployment details are listed: Deployment name: Microsoft.StorageAccount-20181002173523, Subscription: Pay-As-You-Go, Resource group: aks-k8s-bootstrapfiles.

Once the storage account is open. Click on the Files section. This is where the folders and files to bootstrap the firewall will be placed.

The screenshot shows the 'Files' section of the 'bootstrapfiles' storage account. It displays various service options: Blobs, Tables, Queues, and File shares. A red arrow points to the 'File shares that use the standard SMB 3.0 protocol' section under 'File'.

Next click the plus sign to create a new File Share:

The screenshot shows the 'File share' creation screen for the 'bootstrapfiles' storage account. A red arrow points to the '+ File share' button. The table below shows no existing file shares.

NAME	MODIFIED	QUOTA
You don't have any file shares yet. Click '+ File share' to get started.		

When the dialogue window opens, enter the file share information and click create. Note: The Name will be used to update the Variables.tf file in a few steps:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles'. In the center, there is a 'File share' configuration dialog. It has three red arrows pointing to the 'Name' field (containing 'bootstrap'), the 'Quota' field (containing '5'), and the 'Create' button at the bottom right.

Click on the newly created file share:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles'. The 'File share' section now lists a single entry: 'bootstrap'. A red arrow points to the 'bootstrap' entry.

Click on the “Add directory” to create a directory:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrap'. In the top navigation bar, there is a 'File share' section with several buttons: 'Connect', 'Upload', 'Add directory', 'Refresh', 'Delete share', 'Quota', and 'View snaps'. A red arrow points to the 'Add directory' button.

Enter config and click ok:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrap'. A 'New directory' dialog is open. It has a 'Name' field containing 'config' and two buttons at the bottom: 'OK' and 'Cancel'. Two red arrows point to the 'config' name and the 'OK' button.

Repeat this step to create a content, license, and software directory. It is important that all 4 directories are present:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles - Files'. The 'File share' section displays four directories: 'config', 'content', 'license', and 'software'. Each directory is listed with its type (Directory) and size (indicated by three dots). The 'config' directory is highlighted with a yellow arrow.

Click on the config folder:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles - Files'. The 'File share' section displays four directories: 'config', 'content', 'license', and 'software'. The 'config' directory is highlighted with a red arrow. The 'config' directory is also highlighted with a yellow arrow in the main list.

Click “Upload”. When the upload blade opens, select the folder browse and navigate to the files previously downloaded from GitHub. Select the bootstrap.xml and init-cfg.txt. Then click “Upload”:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles - Files'. The 'File share' section displays four directories: 'config', 'content', 'license', and 'software'. The 'config' directory is highlighted with a yellow arrow. A red arrow points to the 'Upload' button in the top right corner of the upload blade. Another red arrow points to the 'Select a file' input field.

Once the files have been uploaded, they should be visible in the directory:

The screenshot shows the Microsoft Azure Storage Account Overview page for 'bootstrapfiles - Files'. The 'File share' section displays four directories: 'config', 'content', 'license', and 'software'. The 'config' directory is highlighted with a yellow arrow. In the top right corner, there is an 'Upload files' blade. Below it, the 'Current uploads' list shows two files: 'init-cfg.txt' and 'bootstrap.xml'. Both files are listed with their sizes (128 B / 128 B and 40 KB / 40 KB) and a green checkmark icon.

It is also possible to add content updates to the content directory that will get loaded into the firewall during the bootstrapping process. The follow figure shows some content files uploaded to the content directory:

The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar lists various Azure services like All services, Dashboard, Resource groups, etc. The main pane shows a file share named 'bootstrap'. Inside 'bootstrap', there is a folder named 'content'. A red arrow points to the 'content' folder. Another red arrow points to a file named 'panup-all-antivirus-2719-3218'.

The next step is to identify the Access Key and update the Terraform Variables.tf file. Navigate to the Storage account and click Access keys:

The screenshot shows the Azure Storage account settings page for 'bootstrapfiles'. The left sidebar shows the storage account's overview and various settings like CORS, Configuration, and Encryption. A red arrow points to the 'Access keys' section. Another red arrow points to the 'key1' key details, which include the key value 'zAa3YjX9fV' and its corresponding connection string.

Next click the copy button to copy the access key for the storage account:

The screenshot shows the 'Access keys' page for the 'bootstrapfiles' storage account. It displays two access keys: 'key1' with value 'zAa3YjX9fV' and 'key2' with value 'IZaXOIBSfcz7qOX+bgSOP0hR9'. Below each key is its corresponding connection string. A large red arrow points from the 'key1' key value towards the bottom right corner of the screen, indicating where to copy the value.

Open the Variables.tf file in an editor and update the custom data variable. The access key, storage account name, and share name need to be added:

```
3 variable "fwOffer" {
4   default = "vmseries1"
5 }
6
7 variable "fwPublisher" {
8   default = "paloaltonetworks"
9 }
10
11 variable "adminUsername" {
12   default = "paloalto"
13 }
14 variable "adminPassword" {
15   default = "Pal0Alt@123"
16 }
17 variable "gvmSize" {
18   default = "Standard_A1"
19 }
20 variable "customdata" {
21   default = "storage-account=<insert storage account name>,access-key=<insert file key>,file-share=<insert share name>,share-directory=None"
22   |
23 }
24
25 }
```

This is a screen shot of the file with the updated information:

```
93 variable "fwOffer" {
94   default = "vmseries1"
95 }
96
97 variable "fwPublisher" {
98   default = "paloaltonetworks"
99 }
100
101 variable "adminUsername" {
102   default = "paloalto"
103 }
104 variable "adminPassword" {
105   default = "Pal0Alt@123"
106 }
107 variable "gvmSize" {
108   default = "Standard_A1"
109 }
110 variable "customdata" {
111   default = "storage-account=bootstrapfiles,access-key=zAa3YJc5
112   |
113 }
114 }
```

SSH keys

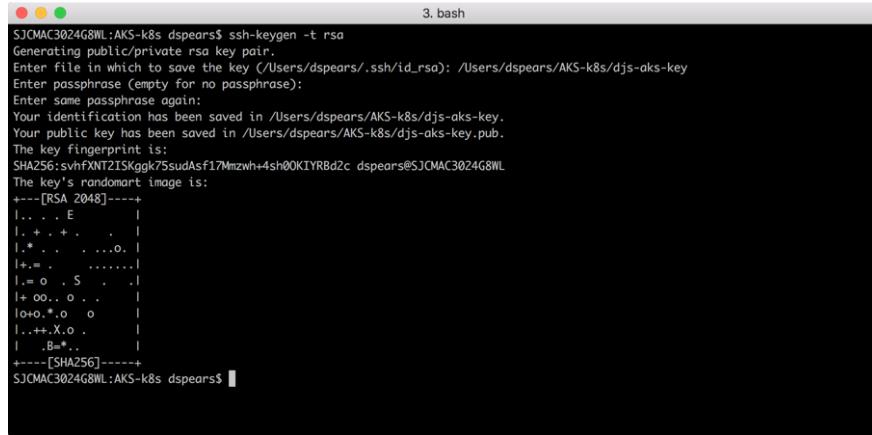
In this activity, you will:

Generate SSH Keys – if needed

Update the Terraform Variables.tf with the path to the SSH keys

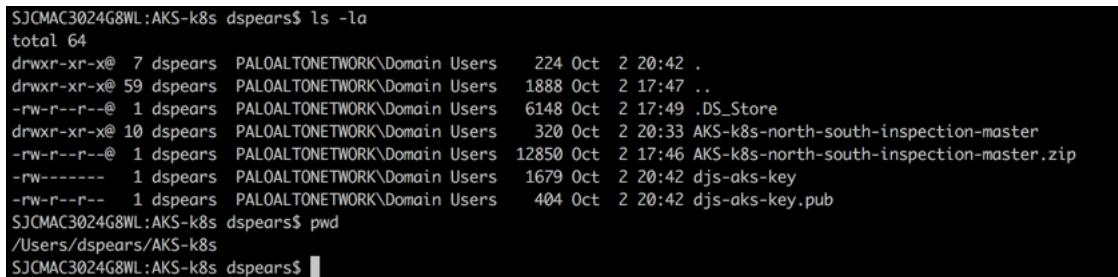
The Terraform Variables.tf file has an option for supplying ssh keys that can be used to log into the Kubernetes nodes after deployment.

If you do not already have an SSH key, the follow example shows how to create an SSH key on a Mac using the **ssh-keygen -t rsa** command:



```
SJCMAC3024G8WL:AKS-k8s dspears$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/dspears/.ssh/id_rsa): /Users/dspears/AKS-k8s/djs-aks-key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/dspears/AKS-k8s/djs-aks-key.
Your public key has been saved in /Users/dspears/AKS-k8s/djs-aks-key.pub.
The key fingerprint is:
SHA256:svfxNT21SKggk75sdAsf17Mzwh+4sh00KIYR8d2c dspears@SJCMAC3024G8WL
The key's randomart image is:
+---[RSA 2048]----+
|... . E |
|. + . . |
|.* . . .o |
|+. . .. |
|.= o . S . |
|+ oo . o . |
|o+o. . o |
|.+X.o . |
| .B=.. |
+---[SHA256]----+
SJCMAC3024G8WL:AKS-k8s dspears$
```

In the previous example the keys were generated and stored in the same directory as the other lab files. The public and private keys can be seen using the **ls -la** command.



```
SJCMAC3024G8WL:AKS-k8s dspears$ ls -la
total 64
drwxr-xr-x@ 7 dspears PALOALTONETWORK\Domain Users 224 Oct 2 20:42 .
drwxr-xr-x@ 59 dspears PALOALTONETWORK\Domain Users 1888 Oct 2 17:47 ..
-rw-r--r--@ 1 dspears PALOALTONETWORK\Domain Users 6148 Oct 2 17:49 .DS_Store
drwxr-xr-x@ 10 dspears PALOALTONETWORK\Domain Users 320 Oct 2 20:33 AKS-k8s-north-south-inspection-master
-rw-r--r--@ 1 dspears PALOALTONETWORK\Domain Users 12850 Oct 2 17:46 AKS-k8s-north-south-inspection-master.zip
-rw----- 1 dspears PALOALTONETWORK\Domain Users 1679 Oct 2 20:42 djs-aks-key
-rw-r--r-- 1 dspears PALOALTONETWORK\Domain Users 404 Oct 2 20:42 djs-aks-key.pub
SJCMAC3024G8WL:AKS-k8s dspears$ pwd
/Users/dspears/AKS-k8s
SJCMAC3024G8WL:AKS-k8s dspears$
```

Next edit the Terraform Variables.tf file to include the path to the public SSH key. The following diagram shows the field that needs to be updated and the field after it has been updated:



```
// PROJECT Variables
variable "client_id" {
  default = "<appId>"
}
variable "client_secret" {
  default = "<password>"
}
variable "agent_count" {
  default = 2
}
variable "ssh_public_key" {
  default = "<path to public ssh key>" ←
}
variable "dns_prefix" {
  default = "k8s-AZURE-HOW"
}
variable cluster_name {
  default = "k8s-Cluster-MGMT"
```

```
// PROJECT Variables
variable "client_id" {
  default = "<appId>"
}
variable "client_secret" {
  default = "<password>"
}
variable "agent_count" {
  default = 2
}
variable "ssh_public_key" {
  default = "/Users/dspears/AKS-k8s/djs-aks-key.pub" ←
}
variable "dns_prefix" {
  default = "k8s-AZURE-HOW"
}
variable cluster_name {
```

Deploy the Terraform Template

In this activity, you will:

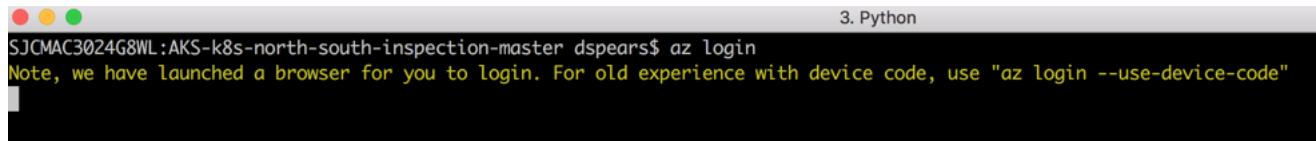
Authenticate to Azure via the Azure command line tool

Initialize Terraform and download the appropriate plugins

Apply the Terraform template

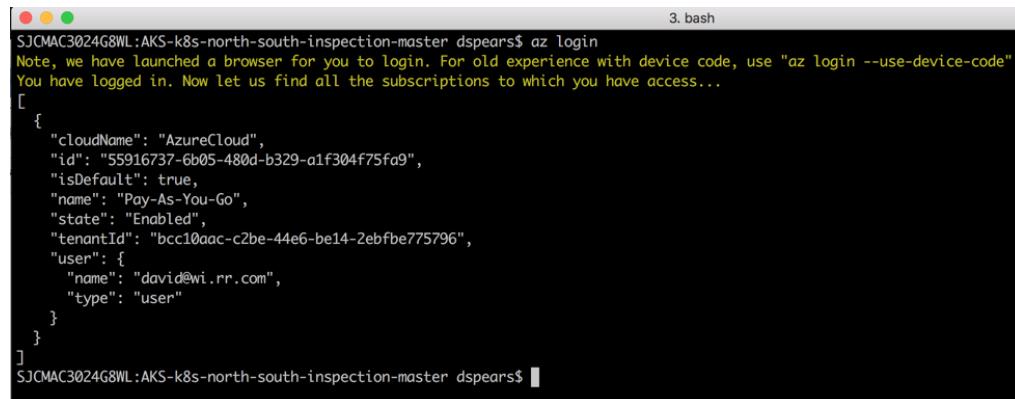
Open a terminal shell and navigate to the directory containing the Terraform template files.

The Azure cli tool token obtained earlier has most likely expired. Use the “az login” login command to get a new token:



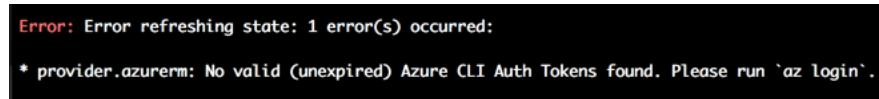
```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ az login
Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
```

After getting redirected to the Microsoft Azure Login and completing the login process successfully, the following prompt will be displayed:



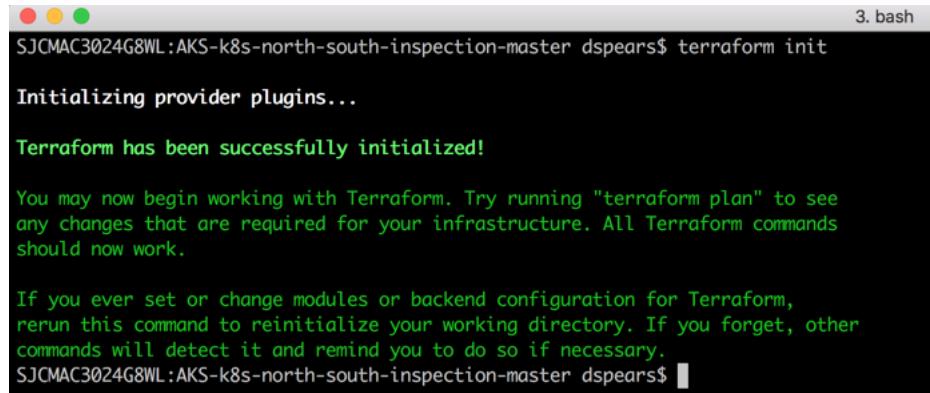
```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ az login
Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
You have logged in. Now let us find all the subscriptions to which you have access...
[{"cloudName": "AzureCloud", "id": "55916737-6b05-480d-b329-a1f304f75fa9", "isDefault": true, "name": "Pay-As-You-Go", "state": "Enabled", "tenantId": "bcc10aac-c2be-44e6-be14-2ebfbe775796", "user": {"name": "david@willr.com", "type": "user"}}]
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$
```

As a note, the following error message is displayed when the azure cli tool token has expired:



```
Error: Error refreshing state: 1 error(s) occurred:
* provider.azurerm: No valid (unexpired) Azure CLI Auth Tokens found. Please run `az login`.
```

Ensure you are in the directory with the Main.tf and Variables.tf files and execute the “**terraform init**” command which will initialize terraform and ensure all the provider plugins are download and up to date:

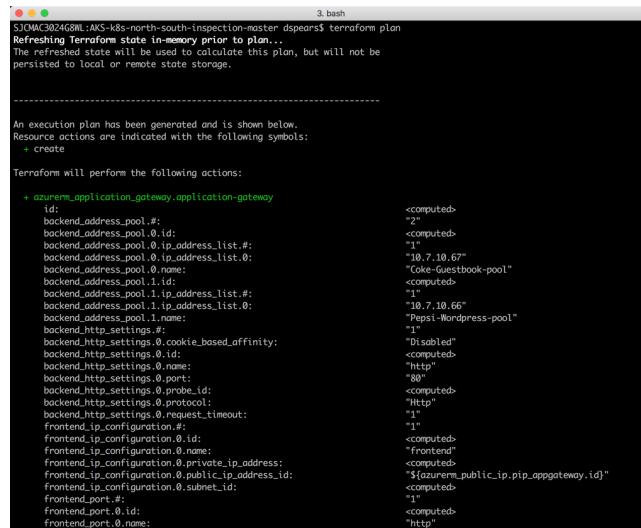


```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ terraform init
Initializing provider plugins...
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$
```

Once the terraform init has completed run the **terraform plan** command. This will show what changes will be implemented with the terraform script. This will also identify if there are any errors detected with the terraform files:



```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

-----
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

+ azurerm_application_gateway.application-gateway
  + id:  "2"
  + backend_address_pool.#: <computed> "2"
  + backend_address_pool.0.id: <computed> "1"
  + backend_address_pool.0.ip_address_list.#: <computed> "1"
  + backend_address_pool.0.ip_address_list.0: <computed> "10.7.10.67"
    + name: "Coke-Guestbook-pool"
  + backend_address_pool.1.id: <computed> "1"
  + backend_address_pool.1.ip_address_list.#: <computed> "1"
  + backend_address_pool.1.ip_address_list.0: <computed> "10.7.10.66"
    + name: "Pepsi-Wordpress-pool"
  + backend_http_setting.#: <computed> "1"
  + backend_http_setting.0.cookie_based_affinity: <computed> "Disabled"
  + backend_http_setting.0.id: <computed> "1"
  + backend_http_setting.0.name: "http"
  + backend_http_setting.0.port: "80"
    + <computed> "Http"
  + backend_http_setting.0.probe_id: <computed> "Http"
  + backend_http_setting.0.request_timeout: "1"
    + <computed> "1s"
  + frontend_ip_configuration.#: <computed> "1"
  + frontend_ip_configuration.0.id: <computed> "1"
  + frontend_ip_configuration.0.name: "frontend"
  + frontend_ip_configuration.0.private_ip_address: <computed> "$azurerm_public_ip.pip_wwwgateway.id"
  + frontend_ip_configuration.0.public_ip_address_id: <computed> "1"
  + frontend_port.#: <computed> "1"
  + frontend_port.0.id: <computed> "1"
  + frontend_port.0.name: "http"
```

Now run the **terraform apply** command to deploy the template. At the action prompt enter **yes**.

```
3. terraform-provider
plan.out[1]: plan.out[2]: plan.out[3]: plan.out[4]:
azurerm_application_gateway.application-gateway: Still creating... (14m10s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m20s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m30s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m40s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m50s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m0s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m10s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m20s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m30s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m40s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m50s elapsed)
azurerm_application_gateway.application-gateway: Creation complete after 16m14s (ID: /subscriptions/55916737-6b05-480d-b329-. .soft.Network/applicationGateways/ag-k8s)

Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

Outputs:

client_key = LS0tLS1CRlJdJTIbSU0EgIJJRKfURSLRVkLS0tL0QnSU1KS0fQjkFB50BZ0WBMd5S3FTY01hNmzdytB0KzHUmxd0Wk1R1QwRk4N3JjdGcz
dmJuZU8xdzR0THK5Cjg3YXRsvYB1ld0V4RTNychnxSW1R2fDcMwNb3hYTmN4SE1tMkZLRDjy0p1252c5u5wAHB5UM0Y1NtWmHd1oKwmpYbWgxr2Z0TTNDTKE0b
3REUnlkNGJ1dWJCcnpoM1B4bE91Ct1UkdtVnBtTyExZn1CRFZ2ZERWZTdsVj1TcwpgBGRPRWruUDBXOTUxYWgNQx0VNDTnB1WjY1U1K02zW0EvRkLksDHm2Z
NheDLWa2NSMyzK0WnHt0JDRsCnVGSUp0SG5LWRtSSJFYj1BczFm0fFnrhEhEM1hckN1nfLbMmxQWFvXUExz1RsNDJwR2hSc09J1djdpxUkoKa2jxL3JR09
FREFEaTFqtk0v1uXrKNHn1ZsbDhS14vJtLOhdMdFvemxChm3asTtCvBQMcryVHzZUaPxm1JdzvTbYbhuZFFpVUJkc01rXwRyeDc3R1pu
Z1pkdFNGdfBnaoVZn1YgUj0cpkFUZEdJVC1PCm1VtWdEMWeVjIvR3dygB3Rje0SV14RtCeVT5b1NjeFFfbzASUJbd00GJvbBns0pTeHhsdGFwC103VDzQ03UK0
mlUoXNFMXNByXRxRzEstvSLJDSHfxdu9TtmE1RxhZMDR0R3JMTjZajjZBxRke1JkUjxhYhzY1BoejdUML1CQpmalNoMnZENmfPQ1dsUmvLe1VuWmfbGoSNkRGt0
Judi1a0WMrB15b1JyerM2TfLIR24TfFd0WxzDfhemzBCKRSGL1zkh1VmDzDErUlgSRC905GN1SDWx15F1V1pTz5hd31Myzd0emlG13TzdqMUL
wQ0F3RUEKQVF1L0fFrQjZ5Qnp0Nbx5s0vzb2UrxYxFQbUcrMDVl0WNNsSEvUmpVzNsXpINTQwSeXbWlzsG1qK0pJbVFRQa3Nlu5M1nb8UdBa0lRjQ8FH7z2ZrmRE
St9tRjFsL2VNa0UyvazkvcEnL3B1WcnZcbVeTERU0uXeHBPK0VxWfd2CmpTlfhdUxNWTkyVsN90W10cNdruBn00AxalJhbix3bWtXVmRiRjFZTEsmK2x3RxVor
1NiU3ISY21wmskmw0QtdUmf4MfIm1xBr-RWSRU2Jsn0pVwxNta1J3Nm1dQTLhtxJUWZkmdM2R0nRj1Qkhsa01rdFHU13amovwpksURm2F2VzWeySm1x03
Y3WpWzjB1VmhiaTQ3t1z2M1U1dUjW11FTYTBFOQzozN1EeJ1cBVK2cw1pmcWczC191Szvsd20ydl5JYjFhWRF3MnVclJwUUTy3cU1R1R0aJxsbfUxahpTeGN
PTVWmcXdpdXbmNHvYz113DVSm0KY01QdEdFu1g4tjVieK5vRnhPbWjJidUvRcGxla09HSkSPDwxDhHWRtFeD1hRELjQTVtUzRa0duUc41cU51YwpqSEYrKhI
TUZYBjDj-TTDRVjBvYF2NG1Z2Zn21X0EJeczPzVRdAFFTbnf1SHexd1H1DJDpDnPRFhBSEgCmrWvOyckNjZ2h3V1ka081mjNj5LhZ2hSY2B3Nkdol0g4d
mWMTjhtC255cmV1YjFHvnFerWtwSnJvYNgRtQKem5jREFhd2pLUlduawWzYykS0xrtGpbLeVpMu1pVjE3Y0xWUvpYUNKQUrMF2zd1VedhhMU8wTdTerGtnWv
p4TW9peEp0uEUsanF3RF13dHRxSwNzTERtVzNpam5GRFZ0WShCnF2Vx3a2s2ekhnUUtDQVFQS9KSETzZFLPcmY2bFzorWRFMeDyQXN00VM0MxE2R1psakZTRUR
Vd1dhIT3Ixv3k4ak1ne0zdn5UUTfZkZBvtGrXBtNVJhcokL3F2UDb1m3UyMxVTRU0p1lRCTDFkVehXZEFhrUqan1tckvGn01rcld60kpNdi.tadktBRF11Mgf0
bytVtm8MgpsU2NrdrJnq0Ev53pYwG51tmWvURoQ2VtS9VbE9pewNSNE0vTM5Y2z3T0VtK2Yrs3SRsr1pHRXV1kM283TnnCnWkd10Rit0TTBsR0k251J0dHU1N
G0xazAs002UjJvU2V1d1cxMwRk2DdEqv3pZcE54Wc2MX1Lz2dSNHLEWgukBewcZkrQ1R1WE1tLopqbktKCTzShdUTngyM1Dnj12d09jmG9hMzBLMchSeEn1dz
ZDMG1URU1elUrQzhKawp2RFB1c2cytzFvHbVUUtDQVFQfTJQVF1bEcYtJQrFuRepQNgxzaUFfb1hvSzrnX1zejNwUGv1d3pWS1RJcm50S1d1jwS9CaGpuRnd
wMzlxRmRqRzVFnjRyaHvlyL0x4YWRfN01tbnZTVEdKMDJ0Rddnnj1f5z1rclw1WXo1a1YkmVYTeSvMGdCU1IVxSHEREMB0Usw5Th1bXjqdk12b1NaoXzV2xy11a
Tn1UYWLONG54apGzN1i1VubWt1qWQpBdk4RExTz3BSWUfqnDdbL1J4amv6RDNZenppUTpsVzYUU2MTNjdzxqTThz0Vh6V1LSS0dqMkttWU1tT2BuCLJ3UnzUS
```

It will take a few minutes to complete. If all goes well, Terraform will output; “**Apply Complete!**” and provide some additional output information about the resources deployed:

```
3. bash
azurerm_application_gateway.application-gateway: Still creating... (14m10s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m20s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m30s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m40s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (14m50s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m0s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m10s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m20s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m30s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m40s elapsed)
azurerm_application_gateway.application-gateway: Still creating... (15m50s elapsed)
azurerm_application_gateway.application-gateway: Creation complete after 16m14s (ID: /subscriptions/55916737-6b05-480d-b329-. .soft.Network/applicationGateways/ag-k8s)

Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

Outputs:

client_key = LS0tLS1CRlJdJTIbSU0EgIJJRKfURSLRVkLS0tL0QnSU1KS0fQjkFB50BZ0WBMd5S3FTY01hNmzdytB0KzHUmxd0Wk1R1QwRk4N3JjdGcz
dmJuZU8xdzR0THK5Cjg3YXRsvYB1ld0V4RTNychnxSW1R2fDcMwNb3hYTmN4SE1tMkZLRDjy0p1252c5u5wAHB5UM0Y1NtWmHd1oKwmpYbWgxr2Z0TTNDTKE0b
3REUnlkNGJ1dWJCcnpoM1B4bE91Ct1UkdtVnBtTyExZn1CRFZ2ZERWZTdsVj1TcwpgBGRPRWruUDBXOTUxYWgNQx0VNDTnB1WjY1U1K02zW0EvRkLksDHm2Z
NheDLWa2NSMyzK0WnHt0JDRsCnVGSUp0SG5LWRtSSJFYj1BczFm0fFnrhEhEM1hckN1nfLbMmxQWFvXUExz1RsNDJwR2hSc09J1djdpxUkoKa2jxL3JR09
FREFEaTFqtk0v1uXrKNHn1ZsbDhS14vJtLOhdMdFvemxChm3asTtCvBQMcryVHzZUaPxm1JdzvTbYbhuZFFpVUJkc01rXwRyeDc3R1pu
Z1pkdFNGdfBnaoVZn1YgUj0cpkFUZEdJVC1PCm1VtWdEMWeVjIvR3dygB3Rje0SV14RtCeVT5b1NjeFFfbzASUJbd00GJvbBns0pTeHhsdGFwC103VDzQ03UK0
mlUoXNFMXNByXRxRzEstvSLJDSHfxdu9TtmE1RxhZMDR0R3JMTjZajjZBxRke1JkUjxhYhzY1BoejdUML1CQpmalNoMnZENmfPQ1dsUmvLe1VuWmfbGoSNkRGt0
Judi1a0WMrB15b1JyerM2TfLIR24TfFd0WxzDfhemzBCKRSGL1zkh1VmDzDErUlgSRC905GN1SDWx15F1V1pTz5hd31Myzd0emlG13TzdqMUL
wQ0F3RUEKQVF1L0fFrQjZ5Qnp0Nbx5s0vzb2UrxYxFQbUcrMDVl0WNNsSEvUmpVzNsXpINTQwSeXbWlzsG1qK0pJbVFRQa3Nlu5M1nb8UdBa0lRjQ8FH7z2ZrmRE
St9tRjFsL2VNa0UyvazkvcEnL3B1WcnZcbVeTERU0uXeHBPK0VxWfd2CmpTlfhdUxNWTkyVsN90W10cNdruBn00AxalJhbix3bWtXVmRiRjFZTEsmK2x3RxVor
1NiU3ISY21wmskmw0QtdUmf4MfIm1xBr-RWSRU2Jsn0pVwxNta1J3Nm1dQTLhtxJUWZkmdM2R0nRj1Qkhsa01rdFHU13amovwpksURm2F2VzWeySm1x03
Y3WpWzjB1VmhiaTQ3t1z2M1U1dUjW11FTYTBFOQzozN1EeJ1cBVK2cw1pmcWczC191Szvsd20ydl5JYjFhWRF3MnVclJwUUTy3cU1R1R0aJxsbfUxahpTeGN
PTVWmcXdpdXbmNHvYz113DVSm0KY01QdEdFu1g4tjVieK5vRnhPbWjJidUvRcGxla09HSkSPDwxDhHWRtFeD1hRELjQTVtUzRa0duUc41cU51YwpqSEYrKhI
TUZYBjDj-TTDRVjBvYF2NG1Z2Zn21X0EJeczPzVRdAFFTbnf1SHexd1H1DJDpDnPRFhBSEgCmrWvOyckNjZ2h3V1ka081mjNj5LhZ2hSY2B3Nkdol0g4d
mWMTjhtC255cmV1YjFHvnFerWtwSnJvYNgRtQKem5jREFhd2pLUlduawWzYykS0xrtGpbLeVpMu1pVjE3Y0xWUvpYUNKQUrMF2zd1VedhhMU8wTdTerGtnWv
p4TW9peEp0uEUsanF3RF13dHRxSwNzTERtVzNpam5GRFZ0WShCnF2Vx3a2s2ekhnUUtDQVFQS9KSETzZFLPcmY2bFzorWRFMeDyQXN00VM0MxE2R1psakZTRUR
Vd1dhIT3Ixv3k4ak1ne0zdn5UUTfZkZBvtGrXBtNVJhcokL3F2UDb1m3UyMxVTRU0p1lRCTDFkVehXZEFhrUqan1tckvGn01rcld60kpNdi.tadktBRF11Mgf0
bytVtm8MgpsU2NrdrJnq0Ev53pYwG51tmWvURoQ2VtS9VbE9pewNSNE0vTM5Y2z3T0VtK2Yrs3SRsr1pHRXV1kM283TnnCnWkd10Rit0TTBsR0k251J0dHU1N
G0xazAs002UjJvU2V1d1cxMwRk2DdEqv3pZcE54Wc2MX1Lz2dSNHLEWgukBewcZkrQ1R1WE1tLopqbktKCTzShdUTngyM1Dnj12d09jmG9hMzBLMchSeEn1dz
ZDMG1URU1elUrQzhKawp2RFB1c2cytzFvHbVUUtDQVFQfTJQVF1bEcYtJQrFuRepQNgxzaUFfb1hvSzrnX1zejNwUGv1d3pWS1RJcm50S1d1jwS9CaGpuRnd
wMzlxRmRqRzVFnjRyaHvlyL0x4YWRfN01tbnZTVEdKMDJ0Rddnnj1f5z1rclw1WXo1a1YkmVYTeSvMGdCU1IVxSHEREMB0Usw5Th1bXjqdk12b1NaoXzV2xy11a
Tn1UYWLONG54apGzN1i1VubWt1qWQpBdk4RExTz3BSWUfqnDdbL1J4amv6RDNZenppUTpsVzYUU2MTNjdzxqTThz0Vh6V1LSS0dqMkttWU1tT2BuCLJ3UnzUS
```

Review what was deployed

In this activity, you will:

Review the resources that have been launched

Inspect k8s cluster

Log into the VM-Series firewall

Confirm bootstrap success

Task 1 – Look around Azure console

Navigate to Resource Groups. Notice that there are two resource groups that were deployed. The first one, k8s-RG, has the infrastructure that was defined in the Terraform template. The second has the k8s nodes and associated resources.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation sidebar is visible with various service icons and 'FAVORITES' section. A red arrow points from the 'Resource groups' icon in this sidebar to the main content area. The main content area is titled 'Resource groups' and shows a table of 12 items. The table includes columns for NAME, SUBSCRIPTION, and LOCATION. Two specific resource groups are highlighted with red arrows: 'k8s' and 'k8s-RG'. The 'k8s' group is located in Central US and is associated with the 'Pay-As-You-Go' subscription. The 'k8s-RG' group is also located in Central US and is associated with the 'Pay-As-You-Go' subscription. Other listed groups include 'aks-k8s-bootstrapfiles', 'Automation', 'cloud-shell-storage-westus', 'djdsj', 'djdsj2', 'djdsjfw', 'djdsjmgmt', 'k8sboot', and 'MC_k8s-RG_k8s-Cluster-MGMT_centralus'.

NAME	SUBSCRIPTION	LOCATION
aks-k8s-bootstrapfiles	Pay-As-You-Go	Central US
Automation	Pay-As-You-Go	West Central US
cloud-shell-storage-westus	Pay-As-You-Go	West US
djdsj	Pay-As-You-Go	West Central US
djdsj2	Pay-As-You-Go	West Central US
djdsjfw	Pay-As-You-Go	West Central US
djdsjmgmt	Pay-As-You-Go	West Central US
k8s	Pay-As-You-Go	Central US
k8s-RG	Pay-As-You-Go	Central US
k8sboot	Pay-As-You-Go	Central US
MC_k8s-RG_k8s-Cluster-MGMT_centralus	Pay-As-You-Go	Central US
MC_k8s_k8s-PANFWF_centralus	Pay-As-You-Go	Central US

Open the two resource groups to view what has been deployed:

Resource Group	Type	Name	Location
k8s-RG	Network security group	akc-k8s-nsg	Central US
	Virtual network	akc-k8s-vnet	Central US
	Route table	appgateway-subnet	Central US
	Network interface	FWeth0	Central US
	Network interface	FWeth1	Central US
	Network interface	FWeth2	Central US
	Public IP address	fvPublicIP	Central US
	Kubernetes service	k8s-Cluster-MGMT	Central US
	Storage account	k8sfwstorage3ce1	Central US
	Route table	k8s-subnet	Central US
Virtual machine	k8s-vm-fw	Central US	

Resource Group	Type	Name	Location
MC_k8s-RG_k8s-Cluster-MGMT_centralus	Network security group	aks-agentpool-56371607-nsg	Central US
	Virtual machine	aks-default-56371607-0	Central US
	Disk	aks-default-56371607-0_OsDisk_1_a2ef9081c3304a55ad69...	Central US
	Virtual machine	aks-default-56371607-1	Central US
	Disk	aks-default-56371607-1_OsDisk_1_2e4d11fabd8f47bbb141...	Central US
	Network interface	aks-default-56371607-nic-0	Central US
	Network interface	aks-default-56371607-nic-1	Central US
	Availability set	default-availabilitySet-56371607	Central US

There should be 1 firewall, 1 k8s service master, and two k8s nodes displayed.

Click on the firewall to open a detailed view of the deployed firewall:

The screenshot shows the Microsoft Azure Resource Groups page for the 'k8s-RG' resource group. The left sidebar lists various services like All services, Favorites, Resource groups, Templates, Function Apps, SQL databases, Virtual machines, Load balancers, Storage accounts, Virtual networks, Network security groups, Route tables, Azure Active Directory, Subscriptions, Security Center, and Kubernetes services. The main pane displays a table of resources with columns for Name, Type, and Location. One row, 'k8s-vm-fw', is highlighted with a red arrow.

Name	Type	Location
appgateway-subnet	Route table	Central US
FVeth0	Network interface	Central US
FVeth1	Network interface	Central US
FVeth2	Network interface	Central US
fwPublicIP	Public IP address	Central US
k8s-Cluster-MGMT	Kubernetes service	Central US
k8sFwstorage3ce1	Storage account	Central US
k8s-subnet	Route table	Central US
k8s-vm-fw	Virtual machine	Central US
pip-appgateway	Public IP address	Central US
WebPublicIP	Public IP address	Central US

Explore the options on the firewall. One interesting area to review is the Networking section. The IP address and security information for each interface can be identified:

The screenshot shows the Microsoft Azure Virtual Machine Networking settings page for the 'k8s-vm-fw' virtual machine. The left sidebar has a 'Networking' tab highlighted with a red arrow. The main pane shows network interfaces FWeth0, FWeth1, and FWeth2. It includes sections for Network Interface (FWeth0), Effective security rules, Topology, Application security groups, and Inbound port rules. A red arrow also points to the FWeth0 interface details.

Navigate to akc-k8s-vnet virtual network in the k8s-RG resource group to see the different networks that have been created as part of the lab.

The screenshot shows the Microsoft Azure Resource Groups page for the 'k8s-RG' resource group. The left sidebar lists various services. The main pane displays a table of resources with columns for Name, Type, and Location. One row, 'akc-k8s-vnet', is highlighted with a red arrow.

Name	Type	Location
ag-k8s	Application gateway	
akc-k8s-nsg	Network security group	
akc-k8s-vnet	Virtual network	

Click Subnets on the left Nav. You should see 5 subnets:

- mgmt-subnet, trust and untrust are used by the firewall
- appgateway-subnet is used by the application gateway
- akc-k8s-subnet is where the k8s nodes and load balancing services are deployed

Microsoft Azure

Search resources, services, and docs

Home > Resource groups > k8s-RG > akc-k8s-vnet - Subnets

Subnets

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

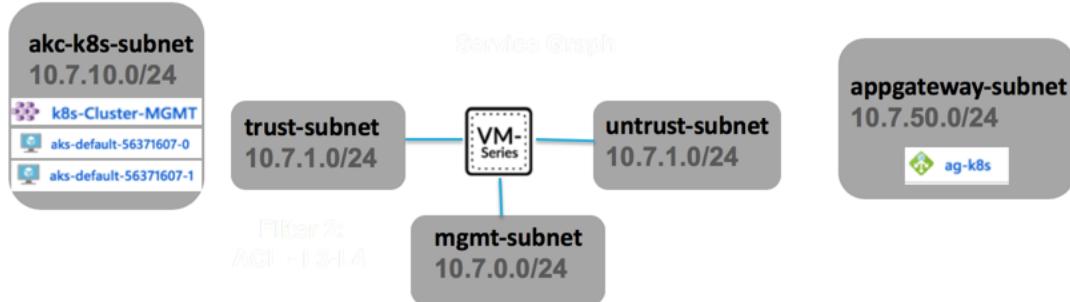
Address space

Connected devices

Subnets

NAME	ADDRESS RANGE	AVAILABLE ADDRESSES	SECURITY GROUP
mgmt-subnet	10.7.0.0/24	250	akc-k8s-nsg
appgateway-subnet	10.7.50.0/24	250	akc-k8s-nsg
trust-subnet	10.7.2.0/24	250	akc-k8s-nsg
untrust-subnet	10.7.1.0/24	250	akc-k8s-nsg
akc-k8s-subnet	10.7.10.0/24	189	akc-k8s-nsg

The following diagram describes the network topology of what has been deployed:



Next Navigate to the k8s-RG resource group and open the application gateway:

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (Dashboard, Resource groups, Templates, Function Apps, SQL databases, Virtual machines, Load balancers, Storage accounts, Virtual networks, Network security groups, Route tables, Azure Active Directory), and 'Settings' (Quickstart, Resource costs, Deployments, Policies, Properties, Locks). The main content area shows the 'k8s-RG' resource group details, including Overview, Activity log, Access control (IAM), Tags, Events, and Settings. Under Settings, there are links for Quickstart, Resource costs, Deployments, Policies, Properties, and Locks. The 'Resource groups' link is highlighted. Below this, a table lists 14 items: ag-k8s (Application gateway, Central US), akc-k8s-nsg (Network security group, Central US), akc-k8s-vnet (Virtual network, Central US), and appgateway-subnet (Route table, Central US). A red arrow points to the 'ag-k8s' row.

Click on the Frontend IP configurations options on the left Nav and notice that there is a single front-end IP address. The application gateways only support a single frontend address. This address will be needed later in the lab.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (Dashboard, Resource groups, Templates, Function Apps, SQL databases, Virtual machines, Load balancers, Storage accounts, Virtual networks, Network security groups, Route tables, Azure Active Directory), and 'Settings' (Configuration, Web application firewall, Backend pools, HTTP settings, Frontend IP configurations). The 'Frontend IP configurations' link is highlighted. The main content area shows the 'ag-k8s - Frontend IP configurations' page for the application gateway. It lists two entries: 'Public' (Configured, frontend, 40.122.109.8 (pip-appgateway), Coke-Guestbook, more) and 'Private' (Not configured, -). A red arrow points to the 'Frontend IP configurations' link in the left sidebar.

Next, go to Listeners on the left Nav. Notice that there are two listeners. This lab will leverage the Applications Gateway's ability to do host header redirection to send traffic to the correct internal load balancer address based on the http request.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (Dashboard, Resource groups, Templates, Function Apps, SQL databases, Virtual machines, Load balancers, Storage accounts, Virtual networks, Network security groups, Route tables, Azure Active Directory), and 'Settings' (Configuration, Web application firewall, Backend pools, HTTP settings, Frontend IP configurations, Listeners). The 'Listeners' link is highlighted. The main content area shows the 'ag-k8s - Listeners' page for the application gateway. It displays a table of listeners:

NAME	PROTOCOL	PORT	ASSOCIATED RULE	HOST NAME
Coke-Guestbook	HTTP	80	Coke-Rule	cokefan.com
Pepsi-WordPress	HTTP	80	Pepsi-Rule	pepsifan.com

Below the table, there is an 'SSL Policy' section with a note about configuring a central SSL policy. It includes radio buttons for 'Default', 'Predefined', and 'Custom', with 'Custom' selected. A red arrow points to the 'Listeners' link in the left sidebar.

Feel free to navigate through other parts of the Azure Console. This will come in handy in activities later on.

Task 2 – Review the Kubernetes Cluster

Kubernetes is a portable, extensible, open-source orchestrator that is used to manage containerized workloads. Kubernetes has a large and rapidly growing ecosystem. The portability of Kubernetes allows for workloads to be migrated between various clouds (public or private). Further documentation is available at: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Navigate to the k8s-RG resource group and click on the k8s-Cluster-MGMT resource. Click on Properties in the k8s-Cluster-MGMT blade. This will show the k8s version, number of nodes deployed, and the infrastructure resource group that was created to deploy k8s resources. This is where the k8s nodes get deployed.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation sidebar with various service icons like All services, Dashboard, Resource groups, etc. The main area shows the 'k8s-Cluster-MGMT - Properties' page for a Kubernetes service. In the left navigation bar, 'Properties' is highlighted with a blue background. Below it, other options like Overview, Activity log, Access control (IAM), Tags, Upgrade, Scale, Locks, Automation script, Monitoring, Insights (preview), Metrics (preview), Logs, Support + troubleshooting, and New support request are listed. The main content area displays details such as KUBERNETES VERSION (1.9.9), DNS PREFIX (k8s-AZURE-HOW), API SERVER ADDRESS (k8s-azure-how-80470d00.hcp.centralus.azmk8s.io), NODE SIZE (Standard D3 v2 (4 vcpus, 14 GB memory)), NODE COUNT (2), TOTAL CORES (8), TOTAL MEMORY (28), and WORKSPACE RESOURCE ID. At the bottom right of the main content area, there's a field labeled 'INFRASTRUCTURE RESOURCE GROUP' with the value 'MC_k8s-RG_k8s-Cluster-MGMT_centralus'. A red arrow points to this field.

Clicking on the Scale link in the left Navigation displays the current number of nodes. From here the number of nodes deployed in the cluster can be increased or decreased.

The screenshot shows the 'k8s-Cluster-MGMT - Scale' page. The left navigation bar has 'Scale' selected, indicated by a blue background. Other options like Properties, Locks, Automation script, Monitoring, Insights (preview), Metrics (preview), Logs, Support + troubleshooting, and New support request are also listed. The main content area contains a message about scaling the cluster, stating 'You can scale the number of nodes in your cluster to increase the total amount of cores and memory available for you. Having at least 3 nodes is recommended for a more resilient cluster.' It also provides links to 'Learn more about scaling your AKS cluster' and 'Total cluster capacity' (Cores: 8 vCPUs, Memory: 28 GB). A large button at the bottom right allows for scaling, with a preview showing 'x Standard D3 v2 (4 vcpus, 14 GB memory)'.

Task 3 – Connect to the Kubernetes Cluster

Navigate back to the terminal window used to deploy the Terraform script. In order to run Kubectl commands, the Kubernetes config from the Terraform state need to be captured and stored in a file that kubectl can read. Execute the following commands in the same directory that the terraform files are in:

```
$ echo "$(terraform output kube_config)" > ./azurek8s
$ export KUBECONFIG=./azurek8s
```

```
3. Shell
$ echo "$(terraform output kube_config)" > ./azurek8s
$ export KUBECONFIG=./azurek8s
```

Let's explore some pods and services that have deployed. Run this command in the cloud shell:

```
$ kubectl get pods
```

```
3. Shell
$ kubectl get pods
No resources found.
```

Since we have not deployed any resources this is normal. Now let us see what system pods have been deployed. Run this command in the shell:

```
$ kubectl get pods --all-namespaces -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	azure-cni-networkmonitor-6pv5t	1/1	Running	0	10h	10.7.10.35	aks-default-56371607-0
kube-system	azure-cni-networkmonitor-tv4jw	1/1	Running	0	10h	10.7.10.4	aks-default-56371607-1
kube-system	heapster-97b7d74b5-6n5qq	2/2	Running	0	10h	10.7.10.15	aks-default-56371607-1
kube-system	kube-dns-v20-7d874cb9b6-gqhg	3/3	Running	0	10h	10.7.10.16	aks-default-56371607-1
kube-system	kube-dns-v20-7d874cb9b6-t5s5w	3/3	Running	0	10h	10.7.10.45	aks-default-56371607-0
kube-system	kube-proxy-rbvt	1/1	Running	0	10h	10.7.10.4	aks-default-56371607-1
kube-system	kube-proxy-rp7vb	1/1	Running	0	10h	10.7.10.35	aks-default-56371607-0
kube-system	kube-svc-redirect-q2sc5	2/2	Running	0	10h	10.7.10.4	aks-default-56371607-1
kube-system	kube-svc-redirect-wzgxr	2/2	Running	0	10h	10.7.10.36	aks-default-56371607-0
kube-system	kubernetes-dashboard-7bb7584f55-28k5l	1/1	Running	1	10h	10.7.10.36	aks-default-56371607-0
kube-system	tunnelfront-7595c7758b-5nkfm	1/1	Running	0	10h	10.7.10.10	aks-default-56371607-1

Note: If the output does not show all the pods in a running state, wait and rerun the **kubectl get pods --all-namespaces -o wide** command until they do. An example of this is state is in the following screen-print:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	heapster-v1.4.3-6644cc4b46-dwn8s	0/2	Pending	0	11m	<none>	gke-cluster-1-default-pool-2df01519-n3w8
kube-system	heapster-v1.4.3-7875d9f9ff-mlv7q	2/2	Terminating	0	12m	10.8.0.4	gke-cluster-1-default-pool-2df01519-6sgf
kube-system	kube-dns-778977457c-2xwtfq	3/3	Running	0	13m	10.8.0.3	gke-cluster-1-default-pool-2df01519-6sgf
kube-system	kube-dns-778977457c-144zrf	0/3	ContainerCreating	0	12m	<none>	gke-cluster-1-default-pool-2df01519-n3w8
kube-system	kube-dns-autoscaler-7db47cb9b7-j8n7n	1/1	Running	0	13m	10.8.1.3	gke-cluster-1-default-pool-2df01519-6sgf
kube-system	kube-proxy-gke-cluster-1-default-pool-2df01519-n3w8	1/1	Running	0	12m	10.5.2.2	gke-cluster-1-default-pool-2df01519-6sgf
kube-system	kubernetes-dashboard-6bb875b5bc-n7wj8	1/1	Running	0	12m	10.5.2.3	gke-cluster-1-default-pool-2df01519-n3w8
davejspears	[redacted]	1/1	Running	0	13m	10.8.0.2	gke-cluster-1-default-pool-2df01519-6sgf

Now let us see what services have been deployed as part of the system:

Run the following in the shell:

\$ kubectl get svc

```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ kubectl get svc
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP  10.21.0.1   <none>        443/TCP   10h
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$
```

As you can see no services besides the system cluster have been deployed.

Task 3 – Log into the firewall

The VM-Series firewall deployed as part of the lab has been bootstrapped. Bootstrapping is a feature of the VM-Series firewall that allows you to load a pre-defined configuration into the firewall during boot-up. This ensures that the firewall is configured and ready at initial boot-up, thereby removing the need for manual configuration. The bootstrapping feature also enables automated deployment of the VM-Series.

Navigate to the k8s-RG resource group and click on the VM-Series firewall Virtual machine:

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation sidebar with options like 'Create a resource', 'All services', 'Resource groups', 'Virtual machines', etc. The main area shows the 'k8s-RG' resource group details. Under 'Overview', it lists 'Subscription (change)', 'Pay-As-You-Go', 'Deployments', and 'Tags'. Below that is a table of resources with columns for 'NAME', 'TYPE', and 'LOCATION'. A red arrow points to the 'k8s-vm-fw' entry, which is listed as a 'Virtual machine'.

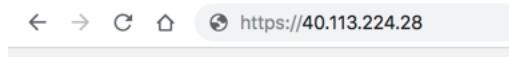
NAME	TYPE	LOCATION
FWeth0	Network interface	
FWeth1	Network interface	
FWeth2	Network interface	
fwPublicIP	Public IP address	
k8s-Cluster-MGMT	Kubernetes service	
k8sfwstorage3cel	Storage account	
k8s-subnet	Route table	
k8s-vm-fw	Virtual machine	

Click on Networking in the left Nav. Copy the Public IP of FWeth0 which is the mgmt interface of the VM-Series firewall:

The screenshot shows the Azure portal interface for managing a VM Series firewall named 'k8s-vm-fw'. The left sidebar has a 'Networking' item under 'Virtual machines'. The main page shows the 'Networking' section for the VM. It lists three network interfaces: FWeth0 (selected), FWeth1, and FWeth2. Below this, it shows the 'Network Interface: FWeth0' details, including its public IP (40.113.224.28) and private IP (10.7.0.4). An 'INBOUND PORT RULES' table is also present.

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
1001	Allow-Outside-From-IP	Any	Any	Any	Any	Allow
65000	AllowNettInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInB...	Any	Any	AzureLoadBal...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Open another browser tab and navigate to the firewall management interface:



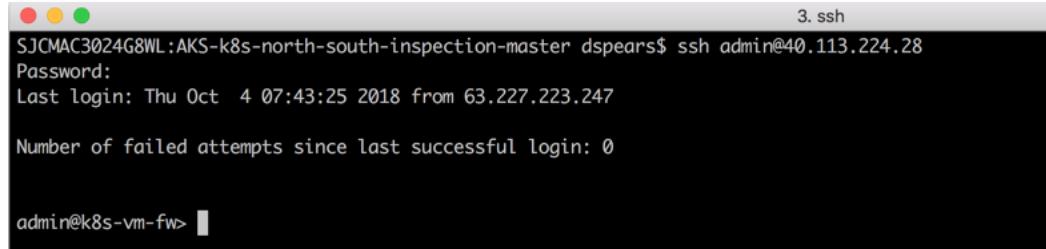
If you get a security exception, please ignore for this lab and proceed to the firewall login page. The VM-Series firewall by default uses a self-signed certificate which causes the exception. Depending on how quickly you do this, you might see the following message. It is normal and part of the bootup process:



If you wish to SSH into the FW, the following syntax can be used:

\$ ssh admin@<ip address of firewall>

The password is: Pal0Alt0@123 -yes, those are zeros.



```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ ssh admin@40.113.224.28
Password:
Last login: Thu Oct  4 07:43:25 2018 from 63.227.223.247
Number of failed attempts since last successful login: 0

admin@k8s-vm-fw> 
```

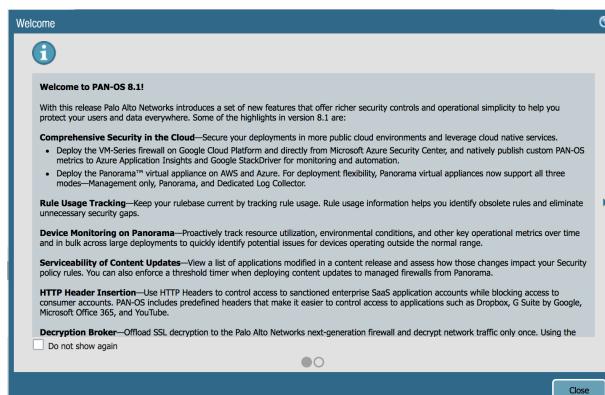
When presented with the login screen you should be able to login to the firewall using (Hint: It's a good idea to jot this password down or save it to a notepad as you will regularly need it):

username: admin

password: Pal0Alt0@123 -yes, those are zeros.



Once logged in you will see a welcome screen, dismiss the welcome dialog box by clicking Close.



Click the Policies tab and you will notice a predefined security policy which was imported using the bootstrapping feature. There are also some predefined NAT policies:

Name	Tags	Type	Zone	Source				Destination	
				Address	User	HIP Profile	Zone	Address	
To Guestbook	none	interzone	untrust	App-Gateway...	any	any	web	guestbook-lb...	
To Guestbook	none	interzone	untrust	App-Gateway...	any	any	web	guestbook-lb...	
To Wordpress	none	interzone	untrust	App-Gateway...	any	any	web	wordpress-lb...	
To Wordpress	none	interzone	untrust	App-Gateway...	any	any	web	wordpress-lb...	
Nodes-Outbound	none	interzone	web	Node-1	any	any	untrust	any	
Nodes-Outbound	none	interzone	web	Node-2	any	any	untrust	any	
App-Gateway-Health...	none	interzone	untrust	App-Gateway...	any	any	web	guestbook-lb...	
App-Gateway-Health...	none	interzone	untrust	App-Gateway...	any	any	web	wordpress-lb...	

Click on the Dashboard tab, check to verify that the firewall has a serial number. The image defined in the terraform template is a Pay as you Go bundle2. This was used because a license will be required to view the logs later in the lab. If you added content files in the bootstrap folder, you should also see that these have been uploaded.

Device Name	k8s-vm-fw
MGT IP Address	10.7.0.4 (DHCP)
MGT Netmask	255.255.255.0
MGT Default Gateway	10.7.0.1
MGT IPv6 Address	unknown
MGT IPv6 Link Local Address	fe80::20d:3aff:fe97:4eca/64
MGT IPv6 Default Gateway	
MGT MAC Address	00:0d:3a:97:4e:ca
Model	PA-VM
Serial #	6530A68D28EF96D
CPU ID	AZRMP-998E0BEFFAACF748:vm300bnd2:centralus
UUID	998E0BEF-FAAC-F748-AEBB-DD56BABC412B
VM License	VM-300
VM Mode	Microsoft Azure
Software Version	8.1.0
GlobalProtect Agent	0.0.0
Application Version	8072-5053 (10/02/18)
Threat Version	8072-5053 (10/02/18)
Antivirus Version	2755-3264 (10/04/18)
WildFire Version	284899-287494 (10/04/18)

Launch a two tiered WordPress application

In this activity, you will:

Optionally: Explore the application's manifest file

Launch a two-tier WordPress application within your cluster

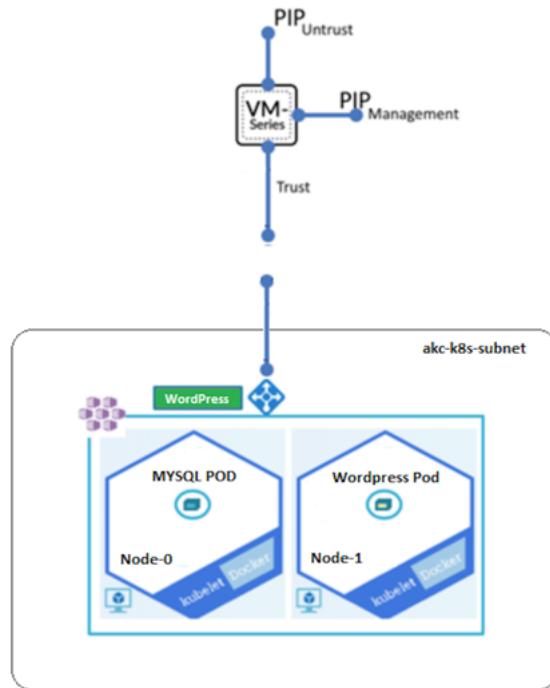
In this activity we will start using Kubernetes specific terms such as Pods, Services, etc. Here is a good primer: <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>

Task 1 – WordPress Application Deployment YAML file

WordPress is a piece of software which has become one of the most widely used content management systems. It is open source, licensed under the GPL, and written in PHP.

WordPress allows users to create and edit websites through a central administrative dashboard, which includes a text editor for modifying content, menus and various design elements. WordPress provides plugins which provide additional functionality through WordPress Plugin Directory. Plugins can be installed through either upload or by one-click installation through the WordPress Plugin Library.

This lab will deploy the following simple WordPress application on the cluster nodes created during the Terraform template deployment:



As you can see this is a two-tiered application with Pods that are dedicated to front-end WordPress services and backend MYSQL DB services.

If interested, the following section dives a bit deeper into the templates being used to create this application. There are two application manifests for this deployment. The first is for the MYSQL DB and the second is for the WordPress frontend. Optionally, open the links below it in a browser of your choice to view the files.

<https://github.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/blob/master/mysql-deployment.yaml> and
<https://github.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/blob/master/wordpress-deployment.yaml>

The manifest file declares various aspects of the application. For instance, it tells the orchestrator what type of resources you intend to deploy. In this case we will first deploy a MYSQL DB server and then a WordPress Frontend.

MYSQL Service:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
      app: wordpress
      tier: mysql
      clusterIP: None
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - RReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
        - image: djspears/panw:wordpress-mysql
          name: mysql
          ports:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
          containerPort: 3306
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
          volumes:
            - name: mysql-persistent-storage
              persistentVolumeClaim:
                claimName: mysql-pv-claim
```

Some things to notice are the listening port, 3306, the container image, and the credentials that will be used during the deployment.

Wordpress-Frontend :

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
        - image: djspears/paw:wordpress
          name: wordpress
          ports:
            - containerPort: 80
              name: wordpress
          volumeMounts:
            - name: wordpress-persistent-storage
              mountPath: /var/www/html
          volumes:
            - name: wordpress-persistent-storage
              persistentVolumeClaim:
                claimName: wp-pv-claim
```

Highlighted in this file are the area that specifies the load balancer service and also the container image. Even though we have two tiers in our application, only one (the frontend service) is exposed to the outside world via a load balancer. The annotation listed above tells AKS and Kubernetes that the load balancer would be of type: Internal.

Task 2 – Launch the Application

As mentioned previously, the application deployment will be done in two steps. The first step will be to deploy the MYSQL DB server. One of the parameters that needs to be passed to the DB server is a root password. To do this securely, the kubectl secrets command will be used. Kubectl secrets are objects intended to hold sensitive information, such as passwords, OAuth tokens, and ssh keys. Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a docker image. To create a secret, execute the following commands in the terminal window:

```
$ kubectl create secret generic mysql-pass --from-literal=password=YOUR_PASSWORD
```

And the following command will verify that the secrets have been stored

```
$ kubectl get secrets
```

```
1. bash
$ kubectl create secret generic mysql-pass --from-literal=password=YOUR_PASSWORD
secret/mysql-pass created
$ kubectl get secrets
NAME          TYPE        DATA   AGE
default-token-hz4q2  kubernetes.io/service-account-token  3      1h
mysql-pass     Opaque      1      3s
$
```

Now the MYSQL pod can be deployed. To do this, execute the following command:

```
$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/master/mysql-deployment.yaml
```

```
1. bash
$ kubectl apply -f https://raw.githubusercontent.com/djspears/aks-k8s/master/mysql-deployment.yaml
service/wordpress-mysql created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/wordpress-mysql created
$
```

You should see the services and deployments being created. Next, validate the new pods in your cluster have been created. In your terminal execute:

```
$ kubectl get pods -o wide
```

You may see the status as Pending or ContainerCreating. This is usually a normal situation:

```
1. bash
$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-mysql-795bf5f54c-jqbdk   0/1     Pending   0          15s     <none>    <none>
$
```

```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-mysql-795bf5f54c-jqbdk   0/1     ContainerCreating   0          1m       <none>    aks-default-56371607-0
$
```

By executing the **kubectl get pods -o wide** again, you start seeing that the Ready and Status of pods change as they start up. Verify that the pod gets to a running status.

```
1. bash
$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-mysql-795bf5f54c-jqbdk   1/1     Running   0          2m       10.7.10.48   aks-default-56371607-0
$
```

With the MYSQL DB Running, create the WordPress frontend by executing the following command:

```
$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/master/wordpress-deployment.yaml
```

```
1. bash
$ kubectl apply -f https://raw.githubusercontent.com/djspears/aks-k8s/master/wordpress-deployment.yaml
service/wordpress created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created
```

Next, validate the new pods in your cluster have been created. In your terminal execute:

```
$ kubectl get pods -o wide
```

Again, you may see the status as Pending or ContainerCreating. This is usually a normal situation:

```
1. bash
$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-8574f9c6f9-hm4cw   0/1     Pending   0          5s       <none>    <none>
wordpress-mysql-795bf5f54c-jqbdk   1/1     Running   0          10m      10.7.10.48   aks-default-56371607-0
$
```

```
1. bash
$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-8574f9c6f9-hm4cw   0/1     ContainerCreating   0          1m       <none>    aks-default-56371607-1
wordpress-mysql-795bf5f54c-jqbdk   1/1     Running   0          11m      10.7.10.48   aks-default-56371607-0
$
```

Again, verify that the pod gets to a running status.

```
1. bash
SJCMAC3024G8WL:AKS-k8s-north-south-inspection master dspears$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE
wordpress-8574f9c6f9-hm4cw   1/1     Running   0          5m      10.7.10.31   aks-default-56371607-1
wordpress-mysql-795bf5f54c-jqbgk  1/1     Running   0          15m     10.7.10.48   aks-default-56371607-0
SJCMAC3024G8WL:AKS-k8s-north-south-inspection master dspears$
```

Launch a two tiered Guestbook application

In this activity, you will:

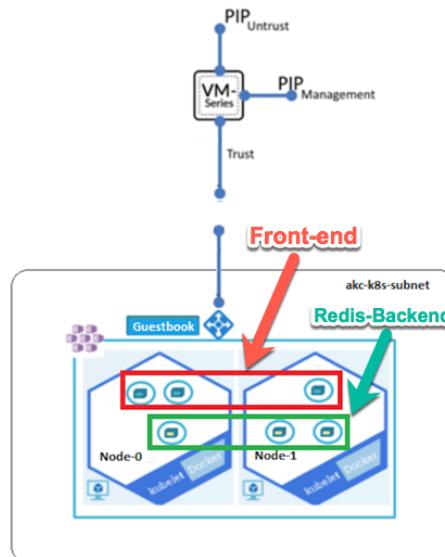
Optionally: Explore the application's manifest file

Launch a two-tier WordPress application within your cluster

Task 1 – Guestbook Application Deployment YAML file

Guestbooks have been used by businesses for many years as a way to connect with customers and obtain contact information for future events and promotions. Today, businesses such as popular retail stores, 5-star hotels and even small family-owned B & B's are turning to iPad guestbook apps to help them gather information and enhance the customer's "in-biz" experience. Acquiring email addresses and a social media following is a crucial part of any marketing plan. With much of the population using computers on a daily basis, an email marketing plan is of the utmost importance. Using a guest book app in your store makes collecting email addresses a snap and offers enticing features with which the traditional paper and pen guestbook just can't compete. The guestbook application we will build and secure today could be used for Hotel website visits, shopping sites or any other business that wants to keep track of their customer and provide them with promotions or advertisements.

This lab will deploy the following simple Guestbook application on the cluster nodes created during the Terraform template deployment:



As you can see this is a two-tiered application with Pods that are dedicated to front-end web services and backend DB services.

If interested, the following section dives a bit deeper into the templates being used to create this application. This is a link to the application manifest. Optionally, click the link below and open it in a browser of your choice.

<https://github.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/blob/master/guestbook-all-in-one.yaml>

The manifest file in this case we will deploy a 2-tier simple redis application with a fronted and backend tier. The backend tier will consist of a redis-master and redis-slave for db redundancy. Front-end Service:

```
90    apiVersion: v1
91    kind: Service
92    metadata:
93      name: frontend
94      annotations:
95        service.beta.kubernetes.io/azure-load-balancer-internal: "true"
96      labels:
97        app: guestbook
98        tier: frontend
99    spec:
100      # if your cluster supports it, uncomment the following to automatically create
101      # an external load-balanced IP for the frontend service.
102      type: LoadBalancer
103      ports:
104        - port: 80
105      selector:
106        app: guestbook
107        tier: frontend
108      ---
109    apiVersion: extensions/v1beta1
110    kind: Deployment
111    metadata:
112      name: frontend
113    spec:
114      replicas: 3
115      template:
116        metadata:
117          labels:
118            app: guestbook
119            tier: frontend
120        spec:
121          containers:
122            - name: php-redis
123              image: gcr.io/google-samples/gb-frontend:v4
124              resources:
125                requests:
126                  cpu: 100m
127                  memory: 100Mi
128              env:
129                - name: GET_HOSTS_FROM
130                  value: dns
131                  # If your cluster config does not include a dns service, then to
132                  # instead access environment variables to find service host
133                  # info, comment out the 'value: dns' line above, and uncomment the
134                  # line below:
135                  # value: env
136          ports:
137            - containerPort: 80
```

This tells the orchestrator that the service will be exposed via an internal load balancer

Redis-backend-master :

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: redis-master
5    labels:
6      app: redis
7      tier: backend
8      role: master
9  spec:
10   type: LoadBalancer
11  ports:
12    - port: 6379
13      targetPort: 6379
14  selector:
15    app: redis
16    tier: backend
17    role: master
18  ---
19  apiVersion: extensions/v1beta1
20  kind: Deployment
21  metadata:
22    name: redis-master
23  spec:
24    replicas: 1
25    template:
26      metadata:
27        labels:
28          app: redis
29          role: master
30          tier: backend
31    spec:
32      containers:
33        - name: master
34          image: gcr.io/google_containers/redis:e2e # or just image: redis
35        resources:
36          requests:
37            cpu: 100m
38            memory: 100Mi
39        ports:
40          - containerPort: 6379
```

Redis-backend-slave:

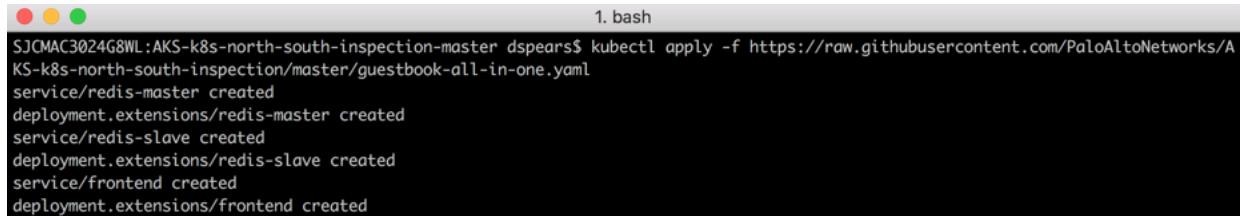
```
42  apiVersion: v1
43  kind: Service
44  metadata:
45    name: redis-slave
46    labels:
47      app: redis
48      tier: backend
49      role: slave
50  spec:
51    type: LoadBalancer
52  ports:
53    - port: 6379
54  selector:
55    app: redis
56    tier: backend
57    role: slave
58  ---
59  apiVersion: extensions/v1beta1
60  kind: Deployment
61  metadata:
62    name: redis-slave
63  spec:
64    replicas: 2
65    template:
66      metadata:
67        labels:
68          app: redis
69          role: slave
70          tier: backend
71    spec:
72      containers:
73        - name: slave
74          image: gcr.io/google_samples/gb-reddisslave:v1
75        resources:
76          requests:
77            cpu: 100m
78            memory: 100Mi
79        env:
80          - name: GET_HOSTS_FROM
81            value: dns
82          # If your cluster config does not include a dns service, then to
83          # instead access an environment variable to find the master
84          # service's host, comment out the 'value: dns' line above, and
85          # uncomment the line below:
86          # value: env
87        ports:
88          - containerPort: 6379
```

Even though there are two tiers in the application, only one (the frontend service) is exposed to the outside world via a load balancer. The annotation listed above tells GCP and Kubernetes that the load balancer would be of type: Internal.

Task 2 – Launch the Application

Back in terminal shell type the following command to deploy the application pods:

```
$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/master/guestbook-all-in-one.yaml
```



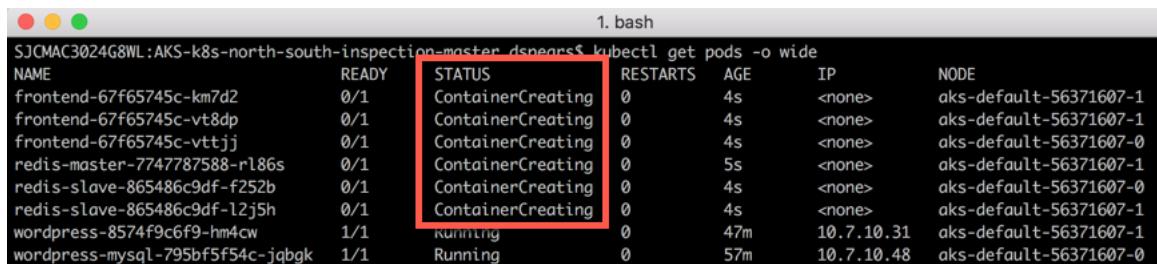
A screenshot of a terminal window titled "1. bash". The window shows the output of a command to apply a Kubernetes YAML file. The output indicates that various components are being created, including a Redis master, a Redis slave, and a Frontend pod. The status of these components is shown as "ContainerCreating" or "Running".

```
SJCMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ kubectl apply -f https://raw.githubusercontent.com/PaloAltoNetworks/AKS-k8s-north-south-inspection/master/guestbook-all-in-one.yaml
service/redis-master created
deployment.extensions/redis-master created
service/redis-slave created
deployment.extensions/redis-slave created
service/frontend created
deployment.extensions/frontend created
```

You should see the services and deployments being created. Next, validate the new pods in your cluster have been created. In your terminal execute:

```
$ kubectl get pods -o wide
```

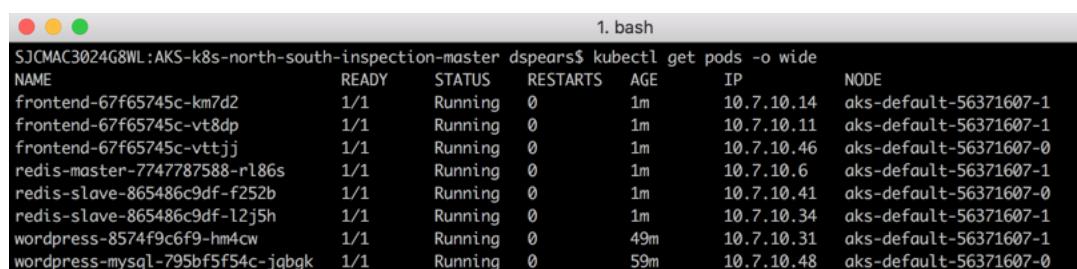
You may see the status as Pending or ContainerCreating. This is usually a normal situation:



A screenshot of a terminal window titled "1. bash". The window shows the output of the "kubectl get pods -o wide" command. It lists several pods, including "frontend", "redis-master", "redis-slave", and "wordpress", all in a "ContainerCreating" state. Other pods like "mysql" are already in a "Running" state.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
frontend-67f65745c-km7d2	0/1	ContainerCreating	0	4s	<none>	aks-default-56371607-1
frontend-67f65745c-vt8dp	0/1	ContainerCreating	0	4s	<none>	aks-default-56371607-1
frontend-67f65745c-vttjj	0/1	ContainerCreating	0	4s	<none>	aks-default-56371607-0
redis-master-7747787588-rl86s	0/1	ContainerCreating	0	5s	<none>	aks-default-56371607-1
redis-slave-865486c9df-f252b	0/1	ContainerCreating	0	4s	<none>	aks-default-56371607-0
redis-slave-865486c9df-l2j5h	0/1	ContainerCreating	0	4s	<none>	aks-default-56371607-1
wordpress-8574f9c6f9-hm4cw	1/1	Running	0	47m	10.7.10.31	aks-default-56371607-1
wordpress-mysql-795bf5f54c-jqbdk	1/1	Running	0	57m	10.7.10.48	aks-default-56371607-0

By executing the **kubectl get pods -o wide** again, you start seeing that the Ready and Status of pods change as they start up. Verify that the pods gets to a running status.



A screenshot of a terminal window titled "1. bash". The window shows the output of the "kubectl get pods -o wide" command again. This time, all the pods listed are in a "Running" state, indicating they have started successfully.

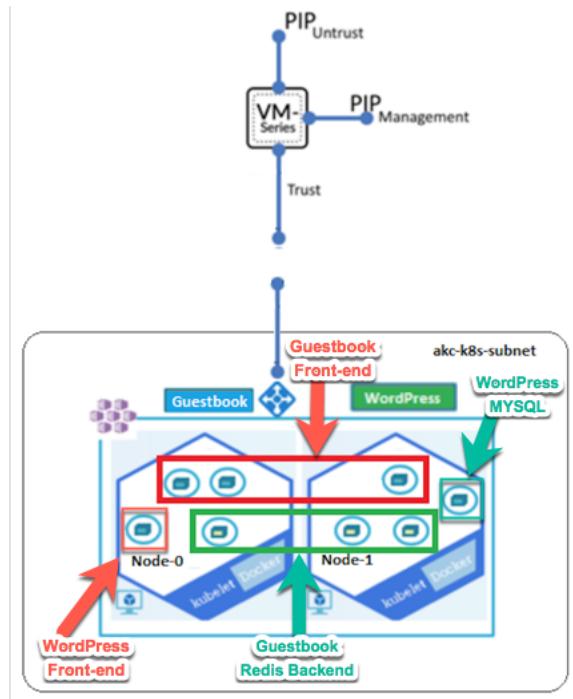
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
frontend-67f65745c-km7d2	1/1	Running	0	1m	10.7.10.14	aks-default-56371607-1
frontend-67f65745c-vt8dp	1/1	Running	0	1m	10.7.10.11	aks-default-56371607-1
frontend-67f65745c-vttjj	1/1	Running	0	1m	10.7.10.46	aks-default-56371607-0
redis-master-7747787588-rl86s	1/1	Running	0	1m	10.7.10.6	aks-default-56371607-1
redis-slave-865486c9df-f252b	1/1	Running	0	1m	10.7.10.41	aks-default-56371607-0
redis-slave-865486c9df-l2j5h	1/1	Running	0	1m	10.7.10.34	aks-default-56371607-1
wordpress-8574f9c6f9-hm4cw	1/1	Running	0	49m	10.7.10.31	aks-default-56371607-1
wordpress-mysql-795bf5f54c-jqbdk	1/1	Running	0	59m	10.7.10.48	aks-default-56371607-0

Explore the newly deployed applications

In this activity, you will:

Explore aspects of the application deployments

The following diagram shows what has been instantiated:



Let's validate this by listing the new pods in your cluster. In your terminal window execute:

```
$ kubectl get pods -o wide
```

You should see the pods for both the WordPress and Guestbook Application:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
Frontend-67f65745c-km7d2	1/1	Running	0	1m	10.7.10.14	aks-default-56371607-1
Frontend-67f65745c-vt8dp	1/1	Running	0	1m	10.7.10.11	aks-default-56371607-1
frontend-67f65745c-vttijj	1/1	Running	0	1m	10.7.10.46	aks-default-56371607-0
redis-master-7747787588-rl86s	1/1	Running	0	1m	10.7.10.6	aks-default-56371607-1
redis-slave-865486c9df-f252b	1/1	Running	0	1m	10.7.10.41	aks-default-56371607-0
redis-slave-865486c9df-1215h	1/1	Running	0	1m	10.7.10.34	aks-default-56371607-1
wordpress-8574f9c6f9-hm4cw	1/1	Running	0	49m	10.7.10.31	aks-default-56371607-1
wordpress-mysql-795bf5f54c-jqbqk	1/1	Running	0	59m	10.7.10.48	aks-default-56371607-0

Next let's look at the load balancing service for the front-end pod. Execute the following command in the shell:

```
$ kubectl get svc
```

You can see there is a load balancer External IP for both the frontend Guestbook application and an External IP address for the WordPress server. Note that the IP address is in the 10.7.10.0/24 subnet. This is one of the subnets that was deployed in the Azure VNET during the Terraform execution.

NAME	TYPE	CLUSTER_IP	EXTERNAL_IP	PORT(S)	AGE
frontend	LoadBalancer	10.21.151.190	10.7.10.67	80:32498/TCP	1h
Kubernetes	ClusterIP	10.21.0.1	<none>	443/TCP	4h
redis-master	ClusterIP	10.21.51.117	<none>	6379/TCP	1h
redis-slave	ClusterIP	10.21.58.87	<none>	6379/TCP	1h
wordpress	LoadBalancer	10.21.165.47	10.7.10.66	80:30231/TCP	2h
wordpress-mysql	ClusterIP	none	<none>	3306/TCP	2h

These load balancer IP addresses can be seen via the Azure Dashboard as well. Navigate to the Resource Groups and click on the “MC_k8s-RG_k8s-Cluster-MGMT_centralus” Resource group. This group was created automatically for the k8s node resources.

The screenshot shows the Microsoft Azure portal interface. On the left, there is a sidebar with various service icons and a 'Resource groups' section under 'FAVORITES'. The main area is titled 'Resource groups' and shows a list of 12 items. A red arrow points to the item 'MC_k8s-RG_k8s-Cluster-MGMT_centralus' in the list.

NAME
aks-k8s-bootstrapfiles
Automation
cloud-shell-storage-westus
djsdjs
djsdjs2
djsdjsfw
djsdjsmgmt
k8s
k8s-RG
k8sboot
MC_k8s-RG_k8s-Cluster-MGMT_centralus
MC_k8s_k8s-PANWFW_centralus

Click on the Kubernetes-internal Load balancer:

The screenshot shows the Microsoft Azure Resource Group Overview page for the resource group 'MC_k8s-RG_k8s-Cluster-MGMT_centralus'. The left sidebar lists various services under 'Kubernetes services'. The main pane displays the 'Overview' tab for the resource group, showing details like 'Subscription (change) Pay-As-You-Go', 'Deployment 1 Succeeded', and 'Tags (change) Click here to add tags'. A table lists 11 items, including network security groups, virtual machines, disk resources, network interfaces, availability sets, and the 'kubernetes-internal' load balancer. A red arrow points to the 'kubernetes-internal' row.

NAME	TYPE	LOCATION
aks-agentpool-56371607-nsg	Network security group	Central US
aks-default-56371607-0	Virtual machine	Central US
aks-default-56371607-0_OsDisk_1_00e924c72fffb49088...	Disk	Central US
aks-default-56371607-1	Virtual machine	Central US
aks-default-56371607-1_OsDisk_1_2be37512773c4b299...	Disk	Central US
aks-default-56371607-nic-0	Network interface	Central US
aks-default-56371607-nic-1	Network interface	Central US
default-availabilitySet-56371607	Availability set	Central US
kubernetes-dynamic-pvc-307474e9-c837-10e8-98d2-0...	Disk	Central US
kubernetes-dynamic-pvc-987100e8-c838-11e8-98d2-0...	Disk	Central US
kubernetes-internal	Load balancer	Central US

Click on the Frontend IP configuration on the left Nav. The application load balancer IP ADDRESS are displayed:

The screenshot shows the Microsoft Azure Load Balancer - Frontend IP configuration page for the 'kubernetes-internal' load balancer. The left sidebar lists various settings for the load balancer. The main pane displays the 'Frontend IP configuration' tab, which lists two frontend IP configurations with their names and IP addresses. Both entries are highlighted with a red box.

NAME	IP ADDRESS	RULES COUNT
a9865f94dc83811e898d20a58ac1f064	10.7.10.66	1
a39b9351bc83f11e898d20a58ac1f064	10.7.10.67	1

Securing Inbound Traffic

In this activity, you will:

Secure traffic that is inbound to your frontend services

Validate that traffic is visible in the Firewall logs

Task 1 – Azure Application Gateway IP Address

This Terraform deployment created an Azure Application gateway in front of the VM-Series firewall. As previous discussed, the Application Gateway is configured to do host header redirection. In order for this to function the frontend IP addresses must be identified and a few hosts entries need to be made on the testing machine. Open the Application Gateway Frontend IP configurations in the Resource groups > k8s-RG > ag-k8s blade:

Type	Status	Name	IP Address	Associated Listeners
Public	Configured	frontend	23.99.213.220 (pip-aggate...)	Coke-Guestbook, 1 more
Private	Not configured	-	-	-

Copy this address as it will be needed to create a DNS entry in the local host file. Go to Application Gateway Listeners on the left Nav to see the DNS entries that the Application Gateway is configured to serve.

Name	Protocol	Port	Associated Rule	Host Name
Coke-Guestbook	HTTP	80	Coke-Rule	cokefan.com
Pepsi-WordPress	HTTP	80	Pepsi-Rule	pepsifan.com

Open the local hosts file and create a pepsifan.com and cokefan.com entry. Each entry will have the IP address of the Application Gateway Frontend IP address:



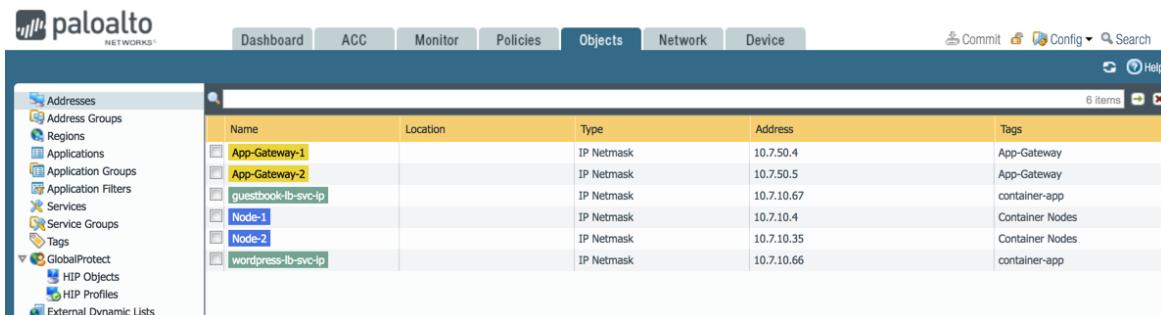
```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1      localhost
255.255.255.255 broadcasthost
::1             localhost
127.0.0.1      vmware-localhost
::1             vmware-localhost
40.122.106.252 djs-wordpress.com
40.122.106.252 djs-questbook.com
23.99.213.220  cokefan.com
23.99.213.220  pepsifan.com

~/etc/hosts" 15L, 399C
```

Task 2 – Update the Firewall’s Address Objects

Open the VM-Series firewall. This design is not using any NATs for the inbound traffic flow. The bootstrapped configuration should have the correct addressing but this task will validate that.

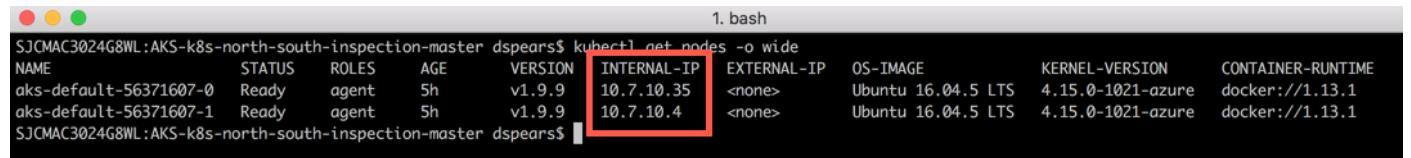
Click the Objects Tab and navigate to “Addresses” on the left. The Addresses used in the policy are defined here:



Name	Location	Type	Address	Tags
App-Gateway-1		IP Netmask	10.7.50.4	App-Gateway
App-Gateway-2		IP Netmask	10.7.50.5	App-Gateway
guestbook-lb-svc-ip		IP Netmask	10.7.10.67	container-app
Node-1		IP Netmask	10.7.10.4	Container Nodes
Node-2		IP Netmask	10.7.10.35	Container Nodes
wordpress-lb-svc-ip		IP Netmask	10.7.10.66	container-app

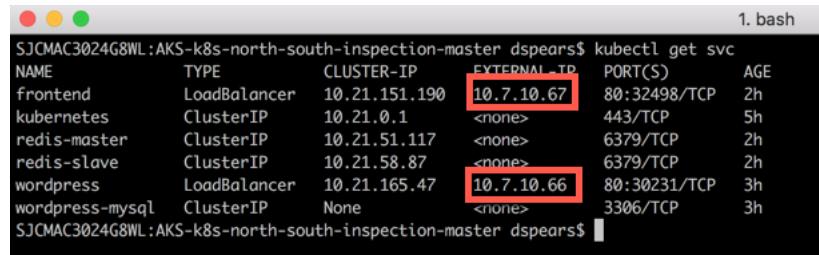
Open the terminal window and check that the Address objects are correct. Execute the following command to verify the nodes:

\$ kubectl get nodes -o wide



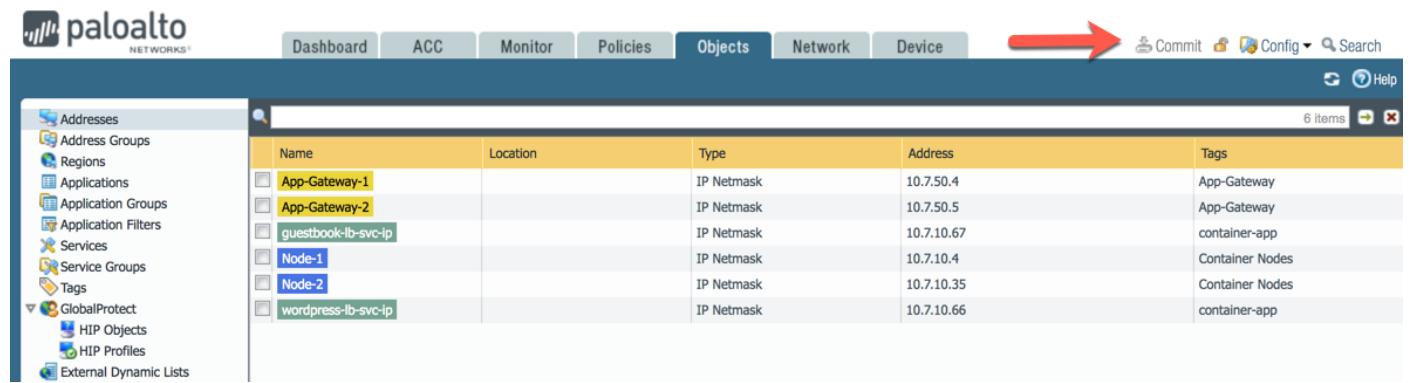
NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
aks-default-56371607-0	Ready	agent	5h	v1.9.9	10.7.10.35	<none>	Ubuntu 16.04.5 LTS	4.15.0-1021-azure	docker://1.13.1
aks-default-56371607-1	Ready	agent	5h	v1.9.9	10.7.10.4	<none>	Ubuntu 16.04.5 LTS	4.15.0-1021-azure	docker://1.13.1

Next enter the “**kubectl get svc**” command to verify the lb-svc-ip's:



```
1. bash
SJC MAC 3024G8WL : AKS-k8s-north-south-inspection-master dspears$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
frontend       LoadBalancer   10.21.151.190  10.7.10.67   80:32498/TCP   2h
kubernetes     ClusterIP    10.21.0.1    <none>        443/TCP       5h
redis-master   ClusterIP    10.21.51.117  <none>        6379/TCP      2h
redis-slave    ClusterIP    10.21.58.87   <none>        6379/TCP      2h
wordpress      LoadBalancer  10.21.165.47  10.7.10.66   80:30231/TCP   3h
wordpress-mysql ClusterIP    None         <none>        3306/TCP      3h
SJC MAC 3024G8WL : AKS-k8s-north-south-inspection-master dspears$
```

If a change is needed, make the changes and click the commit link on the top right



The screenshot shows the Palo Alto Networks UI for managing network objects. The left sidebar contains navigation links for Addresses, Address Groups, Regions, Applications, Application Groups, Application Filters, Services, Service Groups, Tags, GlobalProtect, and External Dynamic Lists. The main area displays a table titled "Addresses" with the following data:

Name	Location	Type	Address	Tags
App-Gateway-1		IP Netmask	10.7.50.4	App-Gateway
App-Gateway-2		IP Netmask	10.7.50.5	App-Gateway
guestbook-lb-svc-ip		IP Netmask	10.7.10.67	container-app
Node-1		IP Netmask	10.7.10.4	Container Nodes
Node-2		IP Netmask	10.7.10.35	Container Nodes
wordpress-lb-svc-ip		IP Netmask	10.7.10.66	container-app

Task 3 – Connect to the Guestbook Frontend

The VM-Series is now protecting your Kubernetes workload. In order to connect to the guestbook's frontend service, you will open a browser and navigate to the <http://cokefan.com> website:

The screenshot shows a web browser window with the following details:

- Title bar: Guestbook, Listeners - Microsoft Azure
- Address bar: Not Secure | cokefan.com
- Page content:
 - Header: Guestbook
 - Form: Messages (input field)
 - Button: Submit

Enter something in the Messages box and click submit. The messages should be echoed below:

The screenshot shows a web browser window with the following details:

- Title bar: Guestbook, Listeners - Microsoft Azure
- Address bar: Not Secure | cokefan.com
- Page content:
 - Message box: hello world
 - Text area: hello world
hello world from AKS
 - Button: Submit

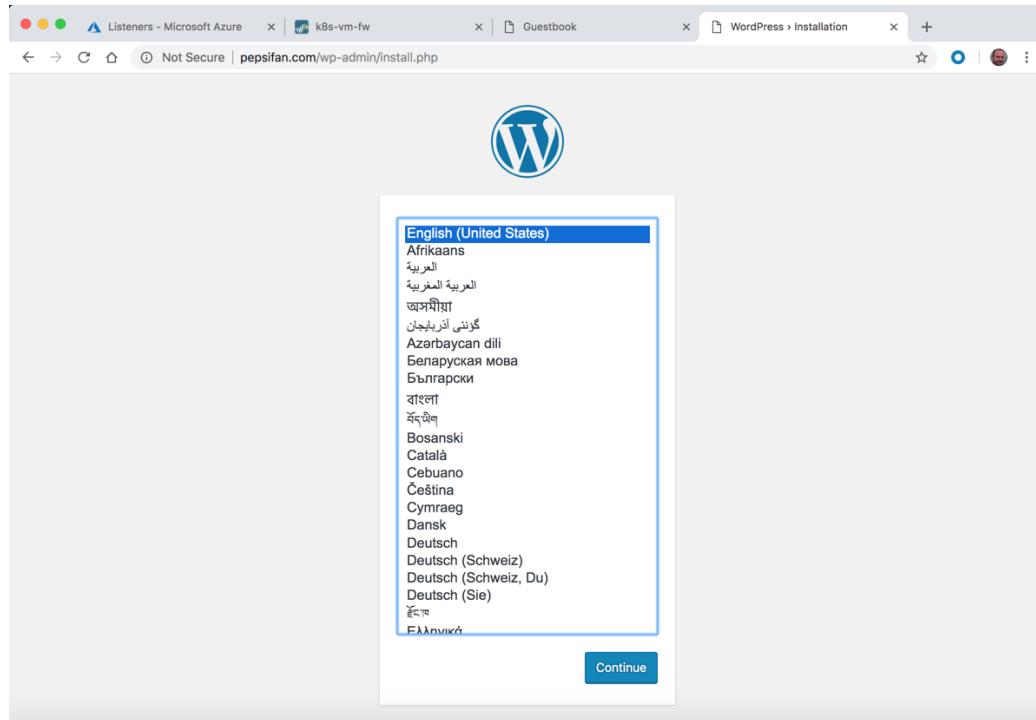
Open the VM-Series firewall monitor tab and validate that traffic is flowing through the firewall:

The screenshot shows the Palo Alto Networks VM-Series firewall monitor tab with the following details:

- Left sidebar: Logs, Traffic, Threat, URL Filtering, WildFire Submissions, Data Filtering, SIP, HTP Match, User-ID, Tunnel Inspection, Configuration, System, Alarms, Authentication, Unified, Packet Capture, App Scope, Change Monitor, Thread Monitor, Threat Map, Network Monitor, Traffic Map, Session Browser, Botnet, PDF Reports, Manage PDF Summary, User Activity Report, SaaS Application Usage, Report Groups, Email Scheduler, Manage Custom Reports, Reports.
- Top navigation: Dashboard, ACC, Monitor, Policies, Objects, Network, Device, Commit, Config, Search, 10 Seconds.
- Table: Logs (Traffic) showing network traffic logs. A red box highlights a row of logs from App-Gateway-1 to guestbook-lb-svc-ip. A red arrow points to the 'Resolve hostname' checkbox at the bottom of the table.
- Bottom: Page navigation (1-10), Resolve hostname checkbox, Highlight Policy Actions, Displaying logs 1-20, 20 per page, DESC, Tasks, Language.

ProTip: Tick the Resolve hostname to make the logs more readable.

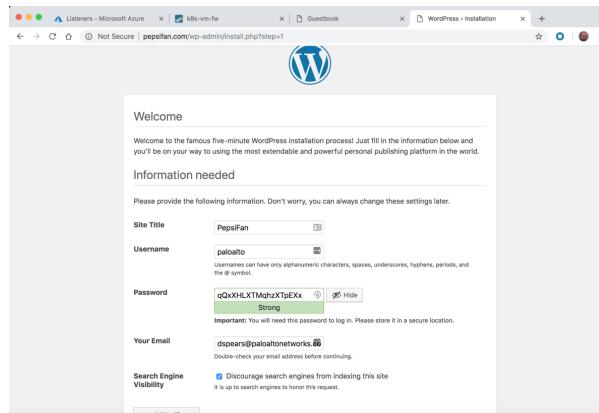
Now check that the <http://pepsifan.com> site works. Open a new tab and open the pepsifan.com site:



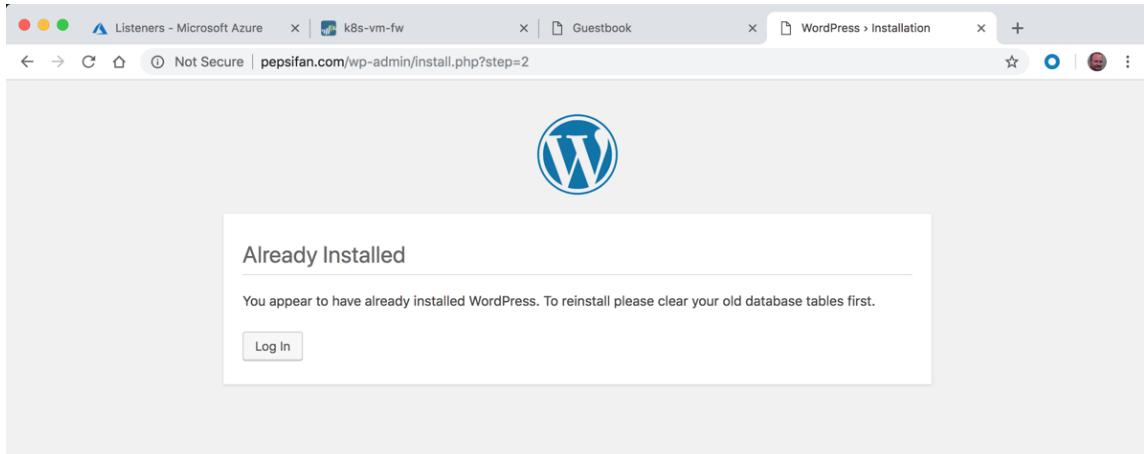
You may see the following error message. This is usually because the WordPress website takes a little time to get up and running. Click refresh a few times to get to the next step:



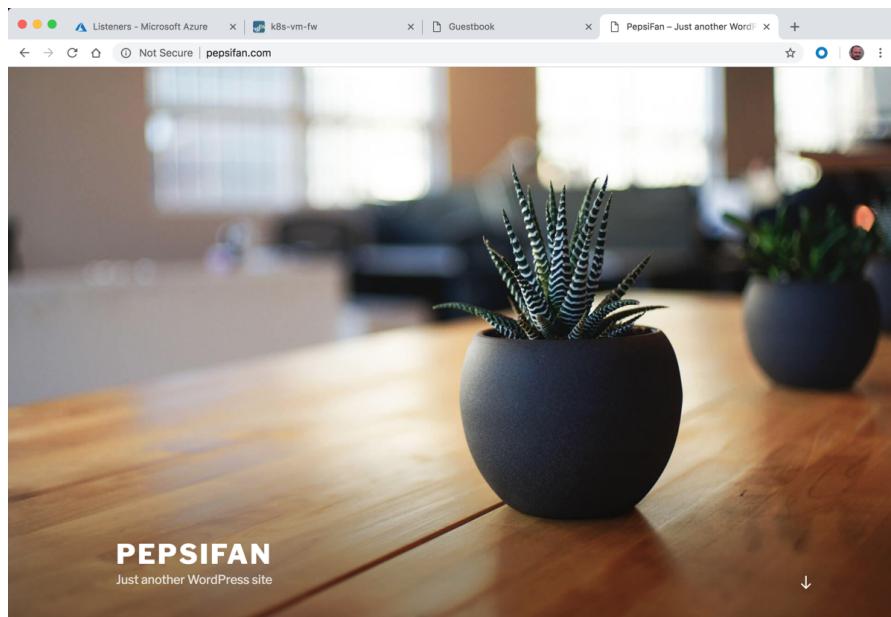
You should see the WordPress install page. Click Continue if you wish to go through the installation process:



After pressing Install WordPress, you might see a 502 error from the Application Gateway. If you press refresh a few times you should see the following:



Go to the root of the <http://pepsifan.com> site and you should now see the default theme:



Verify that the pepsifan.com traffic is running through the firewall:

The screenshot shows the Palo Alto Networks Firewall interface with the 'Logs' tab selected. The left sidebar contains a tree view of monitoring categories: Logs, Threat, URL Filtering, Data Filtering, WildFire Submissions, User-ID, SIP Inspection, Configuration, System, Alarms, Authentication, Unified, Packet Capture, App Scope, Security, Change Monitor, Threat Monitor, Threat Map, Network Monitor, Traffic Map, Session Browser, PDF Reports, Manage PDF Summary, User Activity Report, SaaS Application Usage, Report Groups, Email Scheduler, Manage Custom Reports, and Reports.

The main pane displays a table of log entries. The columns are: Receive Time, Type, From Zone, To Zone, Source, Source User, Destination, To Port, Application, Action, and Rule. The table shows several entries for traffic from 'guestbook' to 'wordpress-lb-svc-ip' port 80, categorized under 'To Guestbook' and 'To WordPress' rules. The logs are timestamped from 10/04/21:45:58 to 10/04/21:47:23.

Receive Time	Type	From Zone	To Zone	Source	Source User	Destination	To Port	Application	Action	Rule
10/04/21:45:58	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:45:53	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress
10/04/21:45:28	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:45:27	drop	untrust	untrust	5.188.206.248		10.7.1.4	3389	not-applicable	deny	default-deny
10/04/21:45:25	drop	untrust	untrust	205.185.113.156		10.7.1.4	8088	not-applicable	deny	default-deny
10/04/21:45:23	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress
10/04/21:45:08	drop	untrust	untrust	205.185.122.121		10.7.1.4	8088	not-applicable	deny	default-deny
10/04/21:44:58	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:44:53	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress
10/04/21:44:28	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:44:23	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress
10/04/21:44:19	drop	untrust	untrust	31.184.237.140		10.7.1.4	3491	not-applicable	deny	default-deny
10/04/21:43:58	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:43:53	drop	untrust	untrust	host190-246-177-94.static.arubado...		10.7.1.4	8088	not-applicable	deny	default-deny
10/04/21:43:53	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress
10/04/21:43:52	drop	untrust	untrust	209.141.42.153		10.7.1.4	8088	not-applicable	deny	default-deny
10/04/21:43:28	end	untrust	web	App-Gateway-1		guestbook-lb-svc-ip	80	web-browsing	allow	To Guestbook
10/04/21:43:26	drop	untrust	untrust	5.188.206.14		10.7.1.4	11504	not-applicable	deny	default-deny
10/04/21:43:25	drop	untrust	untrust	180.247.43.104		10.7.1.4	8000	not-applicable	deny	default-deny
10/04/21:43:23	end	untrust	web	App-Gateway-1		wordpress-lb-svc-ip	80	web-browsing	allow	To WordPress

Displaying logs 1-20 | 20 per page | DESC | Resolve hostname | Highlight Policy Actions | admin | Logout | Last Login Time: 10/04/2018 20:39:04 | Tasks | Language

Securing Outbound Traffic

In this activity, you will:

Secure outbound traffic from the cluster nodes

Validate traffic is in the Firewall logs

Task 1 – Add Outbound Route

To secure any traffic that is originating from within the cluster we need to add a user defined route (UDR) to the routing table on the VNET subnet that the nodes are on. In this deployment that is the 10.7.10.0/24 subnet which is labeled akc-k8s-subnet. Navigate to the k8s-RG resource group and click on the k8s-subnet route tab:

The screenshot shows the Microsoft Azure portal interface. The left sidebar lists various services like Dashboard, Resource groups, and Virtual machines. The main area shows the 'k8s-RG' resource group details. Under the 'Routes' section of the 'k8s-subnet' table, a red arrow points to the row for 'appgateway-subnet'. The table columns include NAME, TYPE, and LOCATION.

NAME	TYPE	LOCATION
ag-k8s	Application gateway	Central US
akc-k8s-nsg	Network security group	Central US
akc-k8s-vnet	Virtual network	Central US
appgateway-subnet	Route table	Central US
FWehd0	Network interface	Central US
FWehd1	Network interface	Central US
FWehd2	Network interface	Central US
FvPublicIP	Public IP address	Central US
k8s-Cluster-MGMT	Kubernetes service	Central US
k8s-fwstorage3ca1	Storage account	Central US
k8s-subnet	Route table	Central US
k8s-vm-fw	Virtual machine	Central US

You can see a route to the app gateway subnet and that this is assigned to the 10.7.10.0/24 subnet:

The screenshot shows the 'k8s-subnet' route table configuration. The 'Routes' section shows a route for 'appgateway-subnet' with an 'ADDRESS PREFIX' of '10.7.50.0/24' and a 'NEXT HOP' of '10.7.2.4'. The 'Subnets' section shows a subnet for 'akc-k8s-subnet' with an 'ADDRESS RANGE' of '10.7.10.0/24', a 'VIRTUAL NETWORK' of 'akc-k8s-vnet', and a 'SECURITY GROUP' of 'akc-k8s-nsg'. Red arrows highlight these fields.

NAME	ADDRESS PREFIX	NEXT HOP
appgateway-subnet	10.7.50.0/24	10.7.2.4

NAME	ADDRESS RANGE	VIRTUAL NETWORK	SECURITY GROUP
akc-k8s-subnet	10.7.10.0/24	akc-k8s-vnet	akc-k8s-nsg

Click on Routes on the left NAV and the click the “+Add” to add a new route:

The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation bar with various service icons like Dashboard, Resource groups, Templates, etc. Below this is a 'FAVORITES' section with links to Dashboard, Resource groups, Templates, Function Apps, SQL databases, Virtual machines, Load balancers, Storage accounts, and Virtual networks. The main content area is titled 'k8s-subnet - Routes'. It shows a table with one row: 'appgateway-subnet' with address prefix '10.7.50.0/24' and next hop '10.7.2.4'. At the top right of this area is a '+ Add' button, which has a red arrow pointing to it. Another red arrow points to the 'Routes' link in the left sidebar under the 'Virtual networks' category.

Create a route with the following Parameters:

The screenshot shows the 'Add route' dialog box. It has a title 'Add route' and a subtitle 'k8s-subnet'. Inside, there are four input fields with validation stars: 'Route name' (value: 'default'), 'Address prefix' (value: '0.0.0.0/0'), 'Next hop type' (value: 'Virtual appliance'), and 'Next hop address' (value: '10.7.2.4'). Below these fields is a note: 'Ensure you have IP forwarding enabled on your virtual appliance. You can enable this by navigating to the respective network interface's IP address settings.' At the bottom of the dialog is a blue 'OK' button, which has a red arrow pointing to it.

Route name: default

Address prefix: 0.0.0.0/0

Next hop type: Virtual Appliance

Next hop address: 10.7.2.4

And then click Create

The new route should appear in the list:

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (with 'Dashboard' selected), 'Resource groups', 'Templates', 'Function Apps', 'SQL databases', 'Virtual machines', and 'Load balancers'. The main content area is titled 'k8s-subnet - Routes' under 'Route table'. It features a search bar, a 'Add' button, and a table with columns 'NAME', 'ADDRESS PREFIX', and 'NEXT HOP'. The table contains two entries: 'appgateway-subnet' with address prefix '10.7.50.0/24' and next hop '10.7.2.4', and 'default' with address prefix '0.0.0.0/0' and next hop '10.7.2.4'.

Navigate back to the firewall monitor tab and you can now see outbound traffic as well from the cluster nodes.

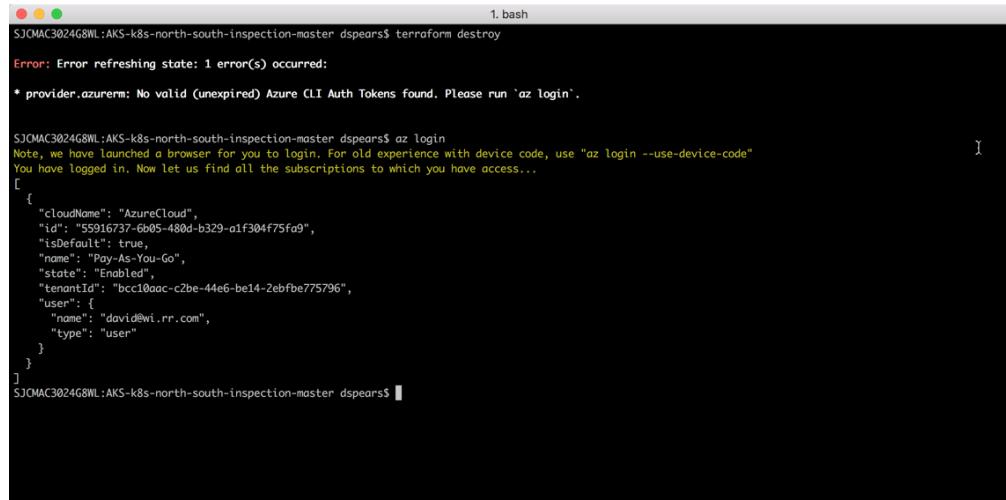
The screenshot shows the Palo Alto Networks Firewall Monitor interface. The left sidebar lists various monitoring categories like Logs, Traffic, Threat, URL Filtering, etc. The main pane displays a table of logs with columns: Receive Time, Type, From Zone, To Zone, Source, Source User, Destination, To Port, Application, Action, and Rule. A specific row in the log table is highlighted with a red box, showing traffic from 'Node-1' (IP 168.61.128.212) to '10.7.10.4' (Port 443) via 'ssl' application. The log table also includes other entries for Node-2 and the App-Gateway.

These source addresses are the instance addresses of the Kubernetes cluster node servers:

Lab Termination

One advantage of Terraform is that it provides the ability to remove the deployment so it is not incurring ongoing cost but could be easily instantiated at a later time for testing and demonstrations. To destroy the lab, go to the terminal prompt and navigate to the directory that was used to deploy the environment and execute:

```
$ terraform destroy
```



The terminal window shows the command `terraform destroy` being run. It displays an error message about invalid Azure CLI Auth Tokens and a note to run `az login`. The terminal then shows the JSON configuration for the provider, specifically the AzureCloud provider settings.

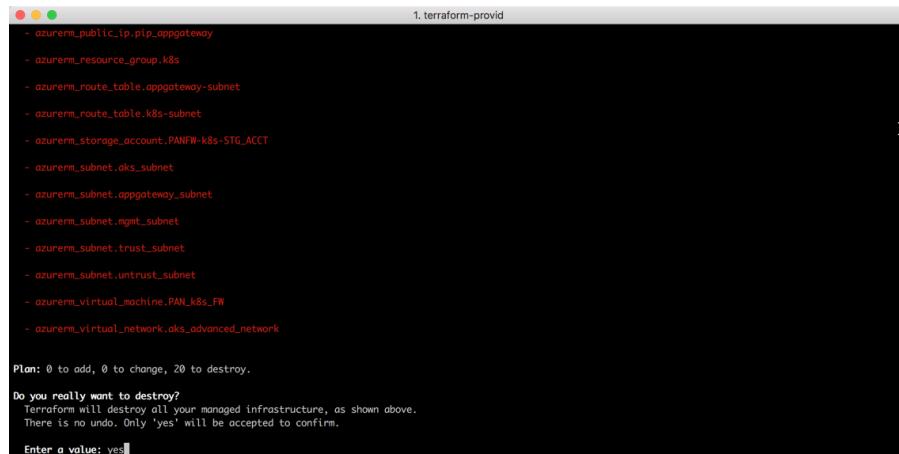
```
1. bash
SJOMAC3024G8NL:AKS-k8s-north-south-inspection-master dspears$ terraform destroy
Error: Error refreshing state: 1 error(s) occurred:

  * provider.azurerm: No valid (unexpired) Azure CLI Auth Tokens found. Please run `az login`.

Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
You have logged in. Now let us find all the subscriptions to which you have access...
[{"cloudName": "AzureCloud",
"id": "55916737-6b05-480d-b329-a1f304f75fa9",
"isDefault": true,
"name": "Pay-As-You-Go",
"state": "Enabled",
"tenantId": "bc10aac-c2be-44e6-be14-2ebfb775796",
"user": {
"name": "david@wi.rr.com",
"type": "user"
}
}]
SJOMAC3024G8NL:AKS-k8s-north-south-inspection-master dspears$
```

If an error message appears regarding the CLI Auth Tokens, run the `az login` command to get a new token.

Terraform will show the list of items that will be removed. Type `yes` at the prompt to start the process:



The terminal window shows the confirmation step for `terraform destroy`. It lists the resources to be destroyed and asks if the user really wants to proceed. The user types `yes` to confirm.

```
1. terraform-provid
- azurerm_public_ip.pip_oppgateway
- azurerm_resource_group.k8s
- azurerm_route_table.oppgateway_subnet
- azurerm_route_table.k8s_subnet
- azurerm_storage_account.PANFW-k8s-STG_ACCT
- azurerm_subnet.aks_subnet
- azurerm_subnet.oppgateway_subnet
- azurerm_subnet.mgmt_subnet
- azurerm_subnet.trust_subnet
- azurerm_subnet.untrust_subnet
- azurerm_virtual_machine.PAN_k8s_FW
- azurerm_virtual_network.aks_advanced_network

Plan: 0 to add, 0 to change, 20 to destroy.

Do you really want to destroy?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes
```

This should result in the complete removal of all the resources:

```
1. bash
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 9m20s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 9m30s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 9m40s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 9m50s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m0s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m10s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m20s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m30s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m40s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m50s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 11m0s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 11m10s elapsed)
azurerm_kubernetes_cluster.k8s: Destruction complete after 11m0s
azurerm_subnet.aks_subnet: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...ks/akc-k8s-vnet/subnets/akc-k8s-subnet)
azurerm_subnet.aks_subnet: Destruction complete after 1s
azurerm_route_table.k8s-subnet: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...crossoft.Network/routeTables/k8s-subnet)
azurerm_network_security_group.aks_advanced_network: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...work/networkSecurityGroups/akc-k8s-nsg)
azurerm_virtual_network.aks_advanced_network: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...t.Network/virtualNetworks/akc-k8s-vnet)
azurerm_route_table.k8s-subnet: Destruction complete after 1s
azurerm_network_security_group.aks_advanced_network: Destruction complete after 1s
azurerm_virtual_network.aks_advanced_network: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...t.Network/virtualNetworks/akc-k8s-vnet, 10s elapsed)
azurerm_virtual_network.aks_advanced_network: Destruction complete after 11s
azurerm_resource_group.k8s: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 10s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 20s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 30s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 40s elapsed)
azurerm_resource_group.k8s: Destruction complete after 48s

Destroy complete! Resources: 20 destroyed.
$JCMAC3024G8NL:AKS-k8s-north-south-inspection-master dspears$
```

This can be validated by executing the **terraform destroy** command one more time:

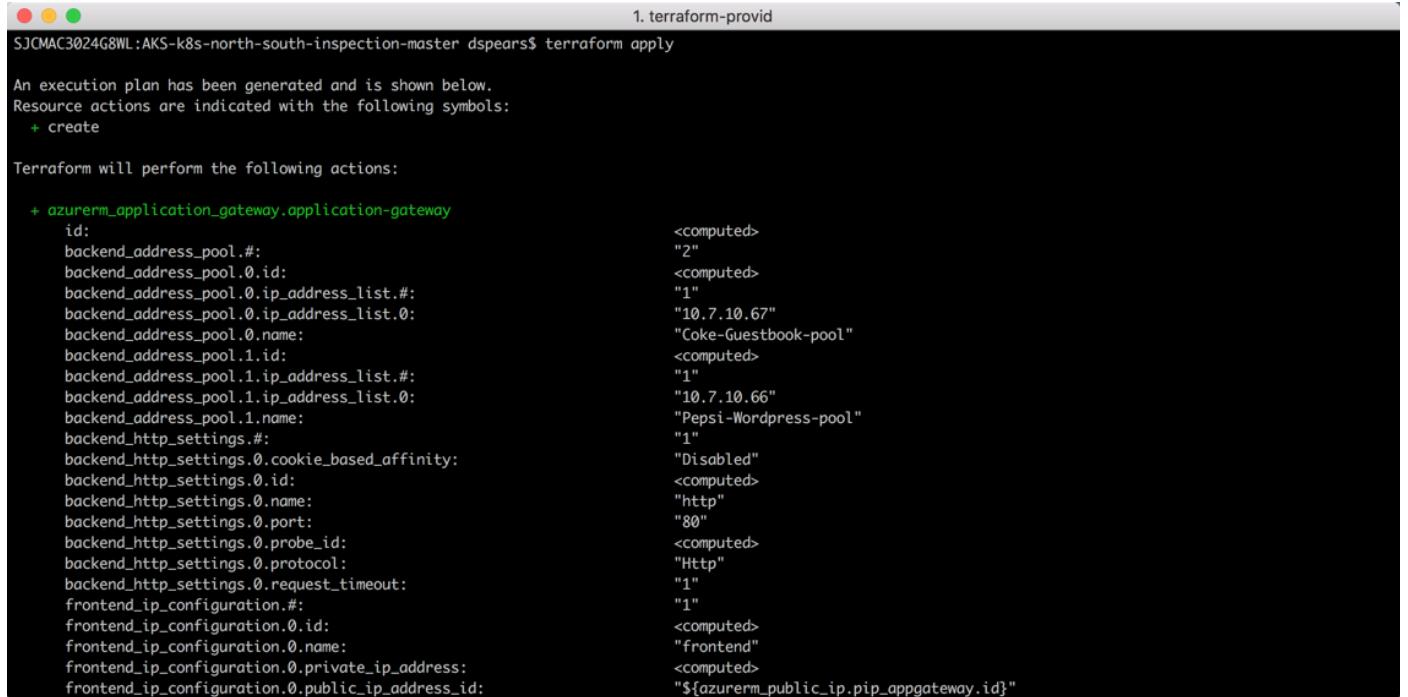
```
1. bash
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 10m50s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 11m0s elapsed)
azurerm_kubernetes_cluster.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...rvice/managedClusters/k8s-Cluster-MGMT, 11m10s elapsed)
azurerm_kubernetes_cluster.k8s: Destruction complete after 11m0s
azurerm_subnet.aks_subnet: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...ks/akc-k8s-vnet/subnets/akc-k8s-subnet)
azurerm_subnet.aks_subnet: Destruction complete after 1s
azurerm_route_table.k8s-subnet: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...crossoft.Network/routeTables/k8s-subnet)
azurerm_network_security_group.aks_advanced_network: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...work/networkSecurityGroups/akc-k8s-nsg)
azurerm_virtual_network.aks_advanced_network: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329...t.Network/virtualNetworks/akc-k8s-vnet)
azurerm_route_table.k8s-subnet: Destruction complete after 1s
azurerm_network_security_group.aks_advanced_network: Destruction complete after 1s
azurerm_virtual_network.aks_advanced_network: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329...t.Network/virtualNetworks/akc-k8s-vnet, 10s elapsed)
azurerm_virtual_network.aks_advanced_network: Destruction complete after 11s
azurerm_resource_group.k8s: Destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 10s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 20s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 30s elapsed)
azurerm_resource_group.k8s: Still destroying... (ID: /subscriptions/55916737-6b05-480d-b329-a1f304f75fa9/resourceGroups/k8s-RG, 40s elapsed)
azurerm_resource_group.k8s: Destruction complete after 48s

Destroy complete! Resources: 20 destroyed.
$JCMAC3024G8NL:AKS-k8s-north-south-inspection-master dspears$ terraform destroy
Do you really want to destroy?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

Destroy complete! Resources: 0 destroyed.
$JCMAC3024G8NL:AKS-k8s-north-south-inspection-master dspears$
```

At any point in the future, it is possible to come back to this directory and simply run the **terraform apply** command and quickly install the environment again:



```
SJOMAC3024G8WL:AKS-k8s-north-south-inspection-master dspears$ terraform apply
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

+ azurerm_application_gateway.application-gateway
  id: <computed>
  backend_address_pool.#: <computed>
  backend_address_pool.0.id: "2"
  backend_address_pool.0.ip_address_list.#: <computed>
  backend_address_pool.0.ip_address_list.0: "1"
  backend_address_pool.0.name: "Coke-Guestbook-pool"
  backend_address_pool.1.id: <computed>
  backend_address_pool.1.ip_address_list.#: "1"
  backend_address_pool.1.ip_address_list.0: "10.7.10.67"
  backend_address_pool.1.name: "Pepsi-Wordpress-pool"
  backend_http_settings.#: "1"
  backend_http_settings.0.cookie_based_affinity: "Disabled"
  backend_http_settings.0.id: <computed>
  backend_http_settings.0.name: "http"
  backend_http_settings.0.port: "80"
  backend_http_settings.0.probe_id: <computed>
  backend_http_settings.0.protocol: "Http"
  backend_http_settings.0.request_timeout: "1"
  frontend_ip_configuration.#: "1"
  frontend_ip_configuration.0.id: <computed>
  frontend_ip_configuration.0.name: "frontend"
  frontend_ip_configuration.0.private_ip_address: <computed>
  frontend_ip_configuration.0.public_ip_address_id: "${azurerm_public_ip.pip_appgateway.id}"
```

End of Activity

Conclusion

Congratulations! You have now successfully integrated the VM-Series firewall to gain visibility into North/South traffic for two container application hosted in a Kubernetes cluster.