Goal

Update Firewall Rules to support AKS Cluster for inbound and outbound connections

Note: Run the lab from the same directory where you found this instructions file

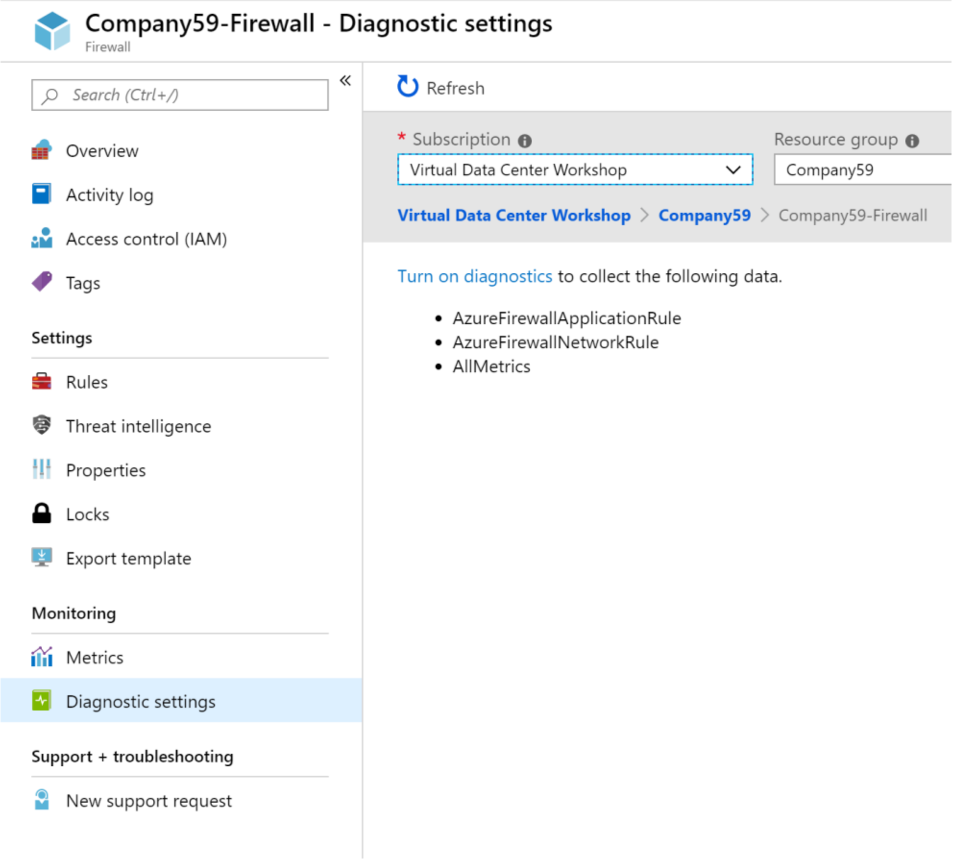
Steps

Create Log Analytics for Azure Firewall

1. Lets set up diagnostics and import the dashboard file as described here: https://docs.microsoft.com/en-us/azure/firewall/tutorial-diagnostics. The goal here is to be able to analyze what Azure FW is seeing from both an ingress and egress perspective throughout the lab.

```
#Setup AzFW LogAnalytics
#https://docs.microsoft.com/bs-latn-ba/azure/firewall/log-analytics-samples
```

Go to Azure Firewall blade and click Diagnostics setttings. Click Turn on Diagnostics:



* Subscription: Virtual Data Center Workshop

Resource group: Company59

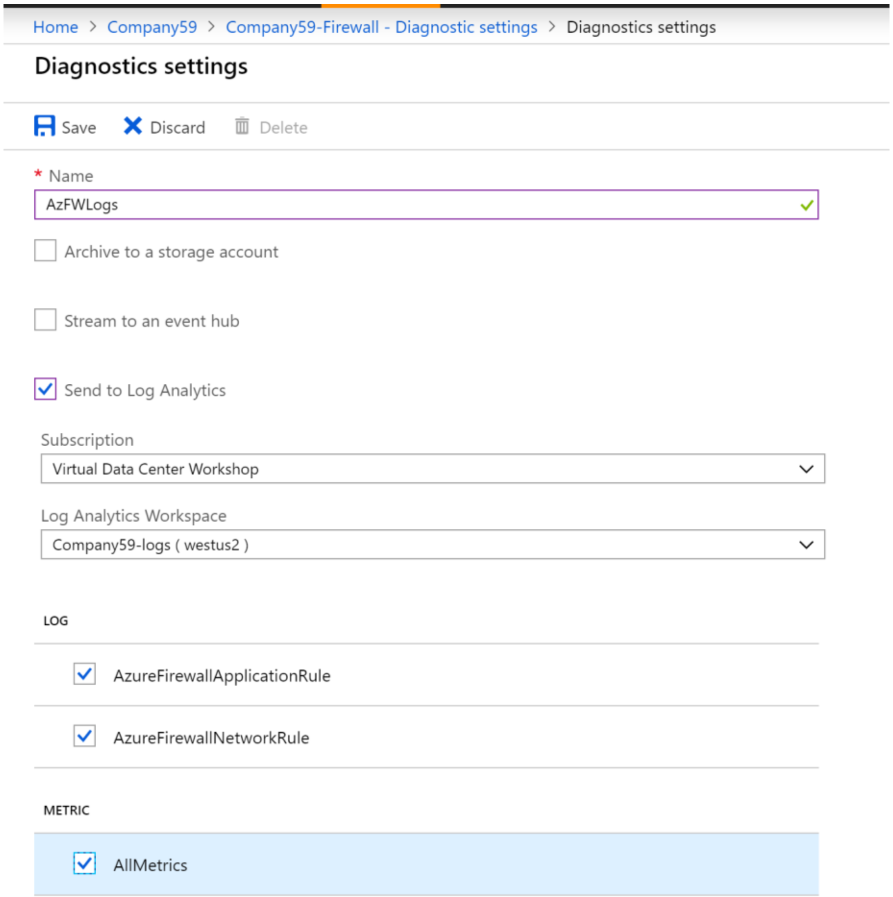
Virtual Data Center Workshop > Company59 > Company59-Firewall

Turn on diagnostics to collect the following data.

- AzureFirewallApplicationRule
- AzureFirewallNetworkRule
- AllMetrics

2. Fill out a name for the Diagnostic settings, including click Send to Log Analytics.

Make sure your subscription and Log Analytics Workspace is selected.
Finally, click on AzureFirewallApplicationRule, AzureFirewallNetworkRule and All Metrics.



Home > Company59 > Company59-Firewall - Diagnostic settings > Diagnostics settings

Diagnostics settings

* Name

AzFWLogs

□ Archive to a storage account

□ Stream to an event hub

☑ Send to Log Analytics

Subscription

Virtual Data Center Workshop

Log Analytics Workspace

Company59-logs (westus2)

LOG

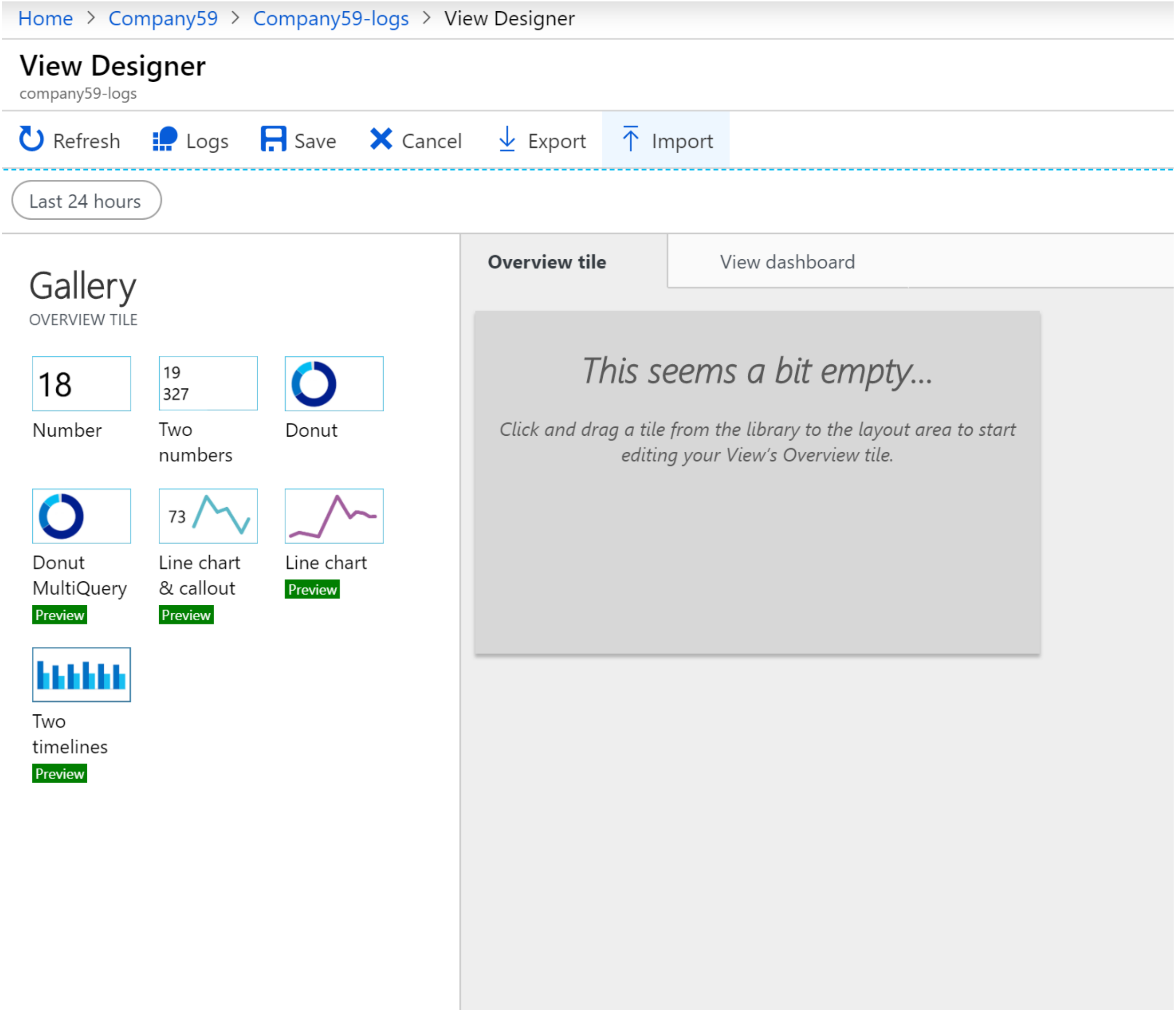
☑ AzureFirewallApplicationRule

☑ AzureFirewallNetworkRule

METRIC

☑ AllMetrics

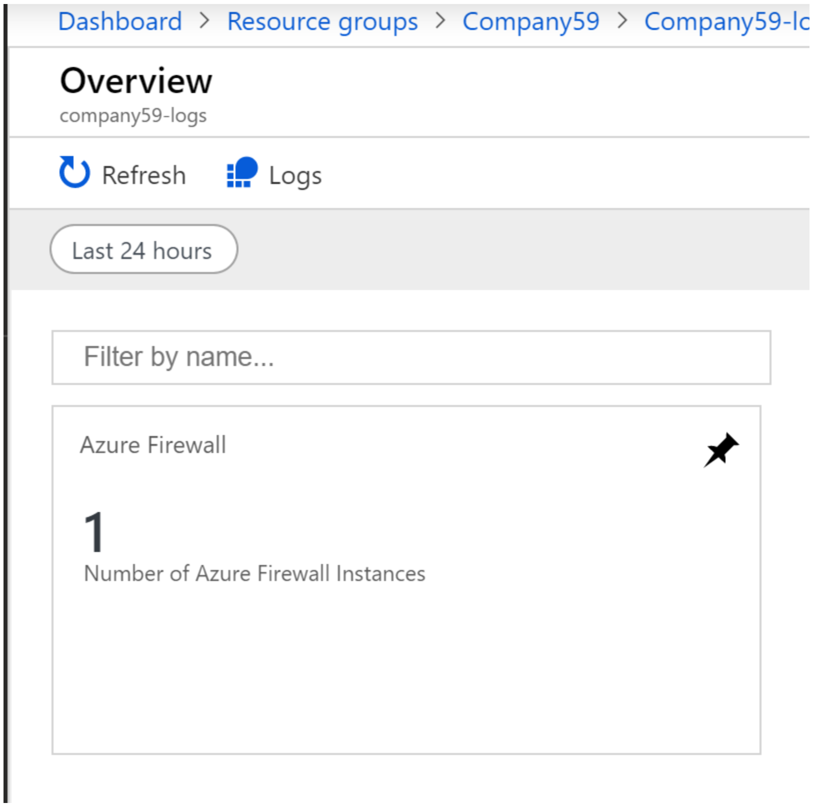
3. Navigatge to LogAnalytics Workspace blade. Click on View Designer.
Click import and select file AzureFirewall.omsview.
Remember to click save and also save as Dashboard.



This seems a bit empty...

Click and drag a tile from the library to the layout area to start editing your View's Overview tile.

4. Click save and then click the pin to save to Dashboard



Dashboard > Resource groups > Company59 > Company59-lc

Overview

company59-logs

Refresh Logs

Last 24 hours

Filter by name...

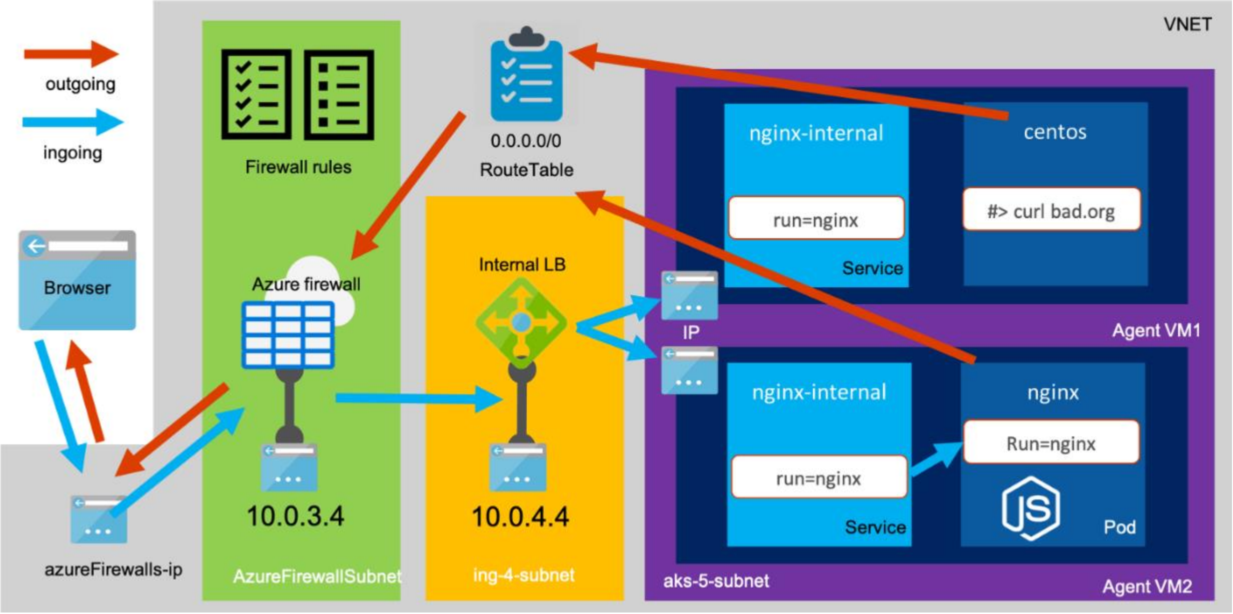
Azure Firewall

1

Number of Azure Firewall Instances

Create Variables and Provision Azure Firewall Rules and Associate to AKS Subnet.

The end result will look like the below diagram. The lab will require some steps to configure the vnet, subnets, routetable, firewall rules and azure kubernetes services which are described below and can be adapted to any kubernetes installation on azure:



We will create variables to use throughout the remaining lab. Remember to update with your specific information replacing the XX with your ID:

1. Update information and copy and paste into your terminal:

```
$RG="CompanyXX" # here enter the resources group name of your AKS cluster

$LOC="westus"   # here enter the datacenter location

$NAME="CXX-Spoke-Cluster" # here enter the name of your kubernetes resource

$VNET_NAME="CXX-Spoke-VNet" # here enter the name of your vnet

# DO NOT CHANGE FWSUBNET_NAME - This is currently a requirement for Azure Firewall.

$FWSUBNET_NAME="AzureFirewallSubnet"

$FWNAME="CompanyXX-Firewall"

$FWPUBLICIP_NAME="CompanyXX-Firewall-pip"

$AKSSUBNET_NAME="Cluster"

$FWROUTE_TABLE_NAME="CXX-VNet-rt-fw"
```

1. Create variable to grab existing Azure firewall public and private IP address:

```
$FWPUBLIC_IP=(az network public-ip show -g $RG -n $FWPUBLICIP_NAME --query "ipAddress" -o tsv)

$FWPRIVATE_IP=(az network firewall show -g $RG -n $FWNAME --query
"ipConfigurations[0].privateIpAddress" -o tsv)
```

2. Validate Azure Firewall IP Address Values - This is more for awareness so you can help connect the networking dots

```
echo $FWPUBLIC_IP

echo $FWPRIVATE_IP
```

3. Create rules on Azure Firewall to allow outbound connections to support AKS control plane:

```
# There is a private preview feature that allows you to whitelist what IPs are allowed to access the
Master API server. Secondly, this summer Azure will preview private-link to provide a single customer
owned private IP address to identify Master API server which will further secure control plane traffic.

# Add the Azure Firewall extension to Azure CLI in case you do not already have it.
```

```
az extension add --name azure-firewall
```

4. Create the Outbound Network Rule from Worker Nodes to Control Plane

```
az network firewall network-rule create -g $RG -f $FWNAME --collection-name 'aksfwnr' --name "allow network"  --protocols 'TCP' --source-addresses '*' --destination-addresses '*' --destination-ports 22 443 --action allow --priority 100
```

5. Add Application FW Rules for Egress Traffic that supports AKS control plane

```
az network firewall application-rule create  --firewall-name $FWNAME --collection-name "aksbasics" --name "allow network" --protocols http=80 https=443 --source-addresses "*" --resource-group $RG --action "Allow" --target-fqdns "*eastus.azmk8s.io" "*auth.docker.io" "*cloudflare.docker.io" "*registry-1.docker.io" "k8s.gcr.io" "storage.googleapis.com" "*cloudflare.docker.com"  --priority 100
```

```
az network firewall application-rule create  --firewall-name $FWNAME --collection-name "aksextended" --name "allow network" --protocols http=80 https=443 --source-addresses "*" --resource-group $RG --action "Allow" --target-fqdns "login.microsoftonline.com"  "*.management.azure.com" "mcr.microsoft.com" "download.opensuse.org" "*.azureedge.net" "*.ubuntu.com" "*azurecr.io" "*blob.core.windows.net"  --priority 101
```

6. Associate your AKS subnet with a route table to route 0.0.0.0/32 to Azure Firewall.

```
az network vnet subnet update -g $RG --vnet-name $VNET_NAME --name $AKSSUBNET_NAME --route-table $FWROUTE_TABLE_NAME
```

Now all outgoing traffic will be filtered and you can check that by launching a pod and see if you can curl the outside internet.

Deploy AKS Ingress Controller with Azure Load-Balancer

1. Enter the following AzCLI to get status on your cluster and current nodes deployed.   Check Provisioning Status of AKS Cluster - ProvisioningState should say 'Succeeded'

```
az aks list -o table
```

2. Get AKS Credentials so kubectl works

```
az aks get-credentials -g $RG -n $NAME --admin
```

3. Setup alias to kubectl becuase I am lazy

```
alias k="kubectl"
```

4. Get Nodes

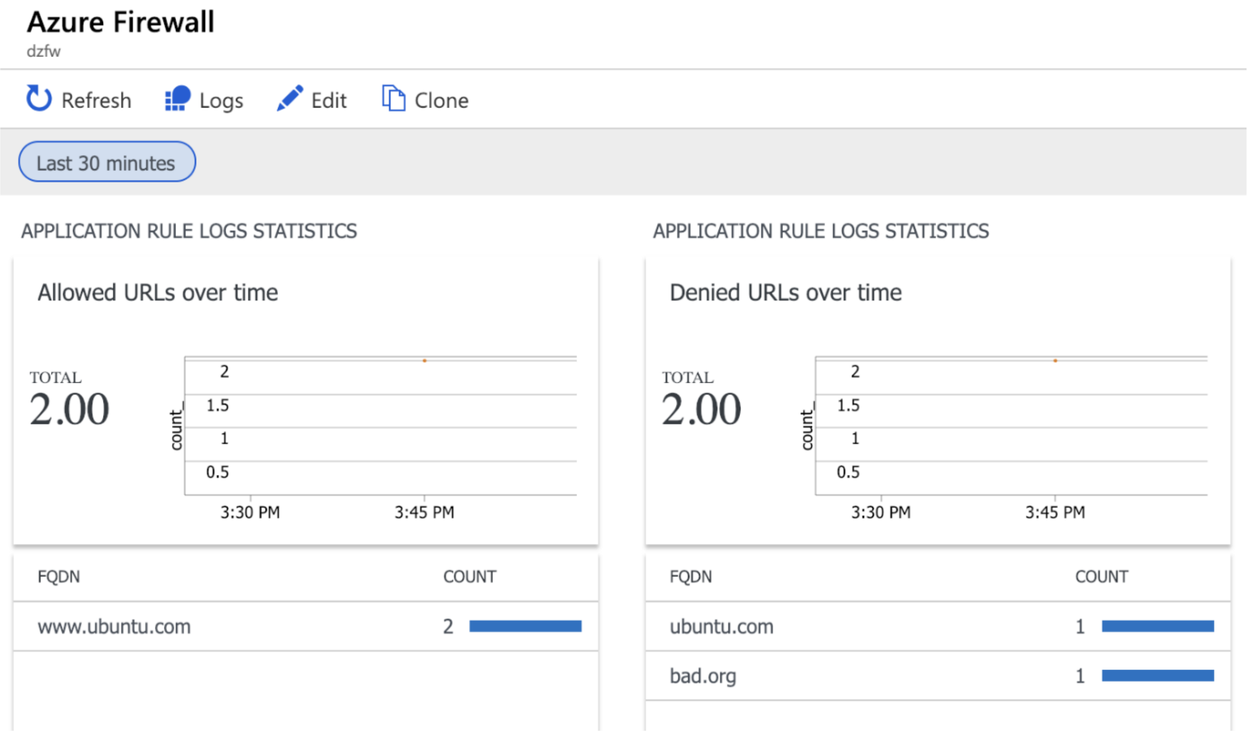
```
k get nodes -o wide
```

5. Let’s check if the Azure Firewall is properly working for egress traffic from a POD. Check Egress Traffic through Firewall

```
kubectl apply -f TestFW.yml

kubectl get po -o wide

kubectl exec -it centos -- /bin/bash

# This curl command should work

curl www.ubuntu.com

# This curl command should fail showing that not even Superman can get through Azure Firewall

curl www.espn.com

exit
```

View Azure Firewall LogAnalytics Dashboard.

1. Navigate to Dashboard and click on your saved Dashboard AzureFirewall. View what urls are being accessed and what IPs are being used.



Azure Firewall
dzfw

APPLICATION RULE LOGS STATISTICS

Allowed URLs over time

FQDN	COUNT
www.ubuntu.com	2

APPLICATION RULE LOGS STATISTICS

Denied URLs over time

FQDN	COUNT
ubuntu.com	1
bad.org	1

Deploy a new Service that is securely accessible via the Internet

A little about this workload. The first service is a web front-end written in .NET Core that sets up a SignalR Hub for receiving real-time communication from the back-end worker node, which in turn relays that to the web client.

The second service is a tensorflow model that classifies pictures of fruit that are located in Azure Files. Each time an image has been classified it communicates back to the SignalR Hub setup on the web front-end which then relays that to the client.

This setup is different than your typical web to database communication as it flows the other way from back-end to front-end. Keep those communication paths in mind when we get to updating variables for communication between servics as well as service segmentation.

1. First lets make sure we have proper DNS setup. Get the Public IP address of your Azure Firewall

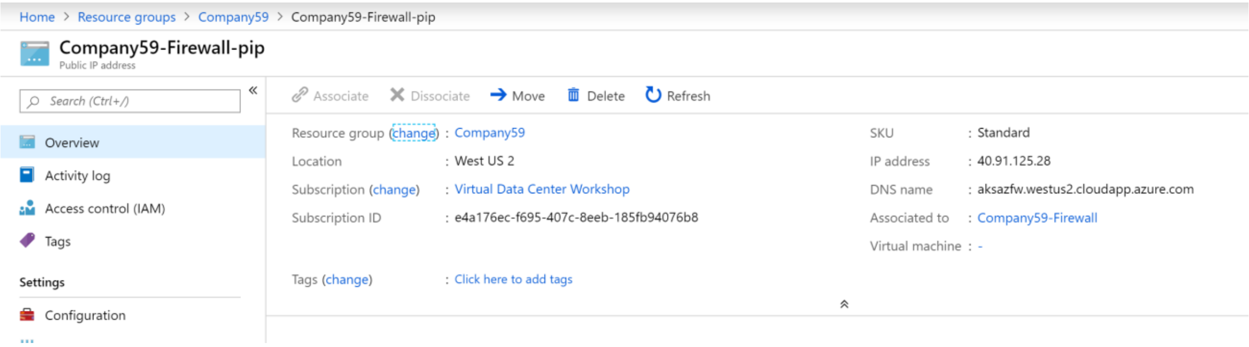
```
$IP=(az network public-ip show -g $RG -n $FWPUBLICIP_NAME --query "ipAddress" -o tsv)
```

```
Echo $IP
```

2. Name to associate with public IP address of your new service you are about to

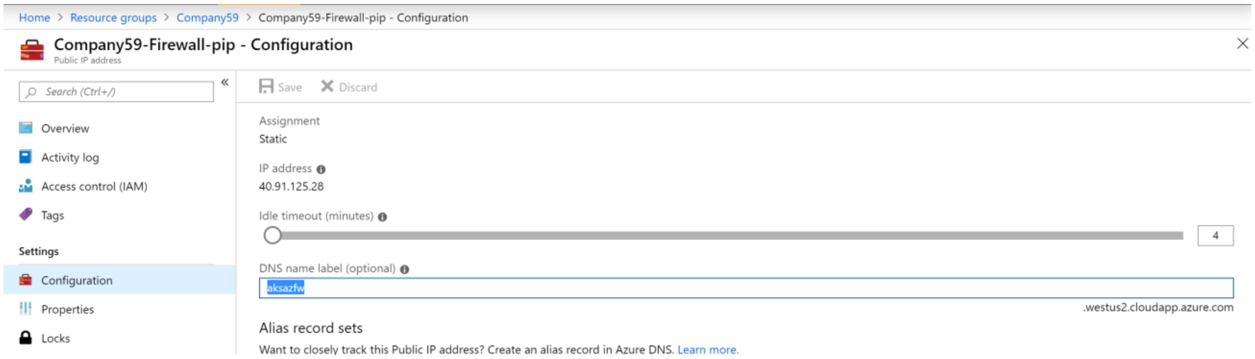
```
$DNSNAME="aksazfw"
```

3. Navigate to the Public IP of your Azure Firewall



Resource group (change) : Company59
Location : West US 2
Subscription (change) : Virtual Data Center Workshop
Subscription ID : e4a176ec-f695-407c-8eeb-185fb94076b8
Tags (change) : Click here to add tags
SKU : Standard
IP address : 40.91.125.28
DNS name : aksazfw.westus2.cloudapp.azure.com
Associated to : Company59-Firewall
Virtual machine : -

4. Click configuration and update with your dns



Assignment
Static

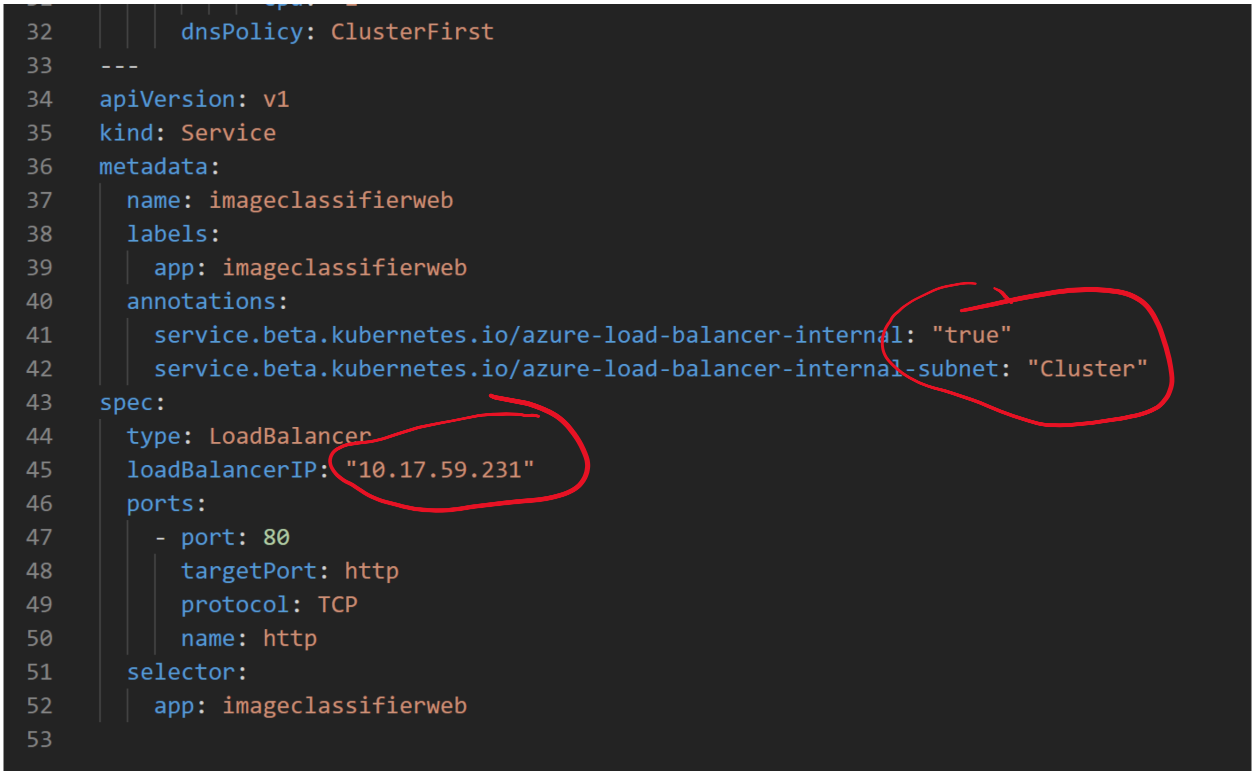
IP address
40.91.125.28

Idle timeout (minutes)
4

DNS name label (optional)
aksazfw
.westus2.cloudapp.azure.com

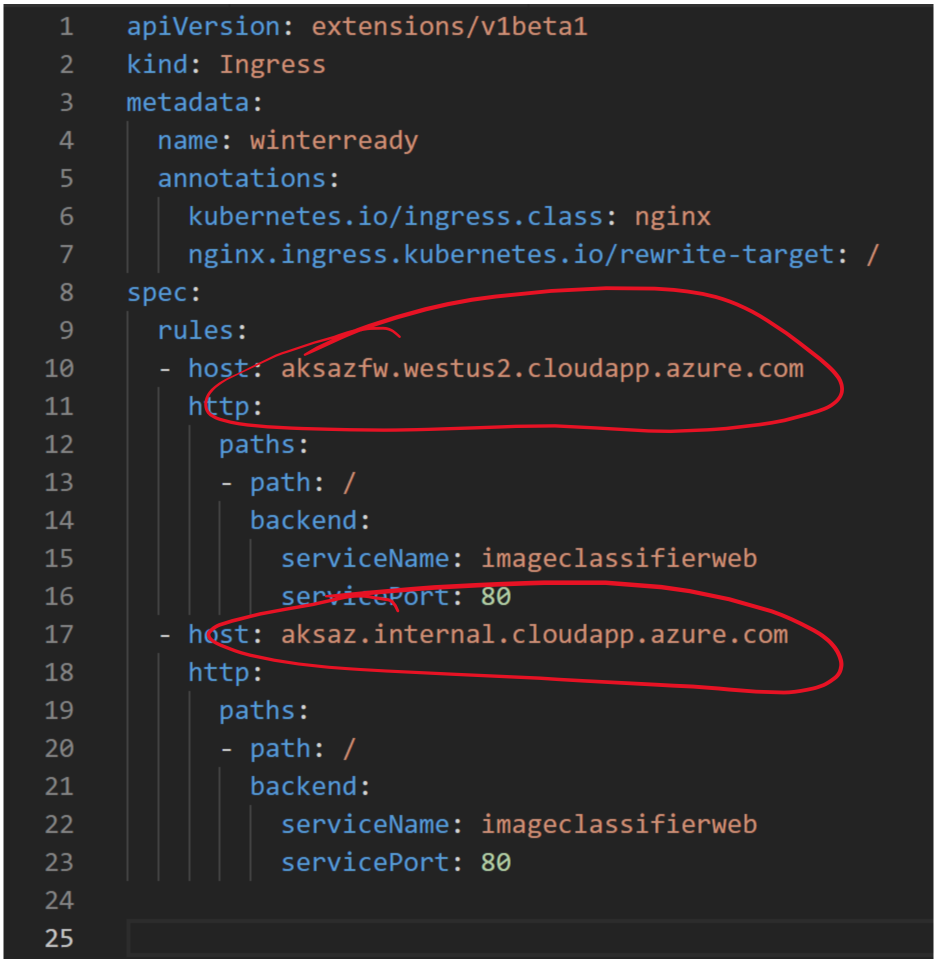
Alias record sets
Want to closely track this Public IP address? Create an alias record in Azure DNS. Learn more.

5. Next before deploying the Pods we need to update winter-ready-web2.yaml We need to edit line 45 and enter an available private IP address from Cluster subnet. #Make sure subnet name and address space is updated in yaml file.



```
      dnsPolicy: ClusterFirst
---
apiVersion: v1
kind: Service
metadata:
  name: imageclassifierweb
  labels:
    app: imageclassifierweb
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    service.beta.kubernetes.io/azure-load-balancer-internal-subnet: "Cluster"
spec:
  type: LoadBalancer
  loadBalancerIP: "10.17.59.231"
  ports:
    - port: 80
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: imageclassifierweb
```

6. Open up winterready-ingress-web2.yaml We need to edit the host entry (line 10 and 17) with the DNS name you created above.



```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: winterready
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: aksazfw.westus2.cloudapp.azure.com
    http:
      paths:
      - path: /
        backend:
          serviceName: imageclassifierweb
          servicePort: 80
  - host: aksaz.internal.cloudapp.azure.com
    http:
      paths:
      - path: /
        backend:
          serviceName: imageclassifierweb
          servicePort: 80

```

7. Now lets deploy the service. We are going to add Web Front-End first.

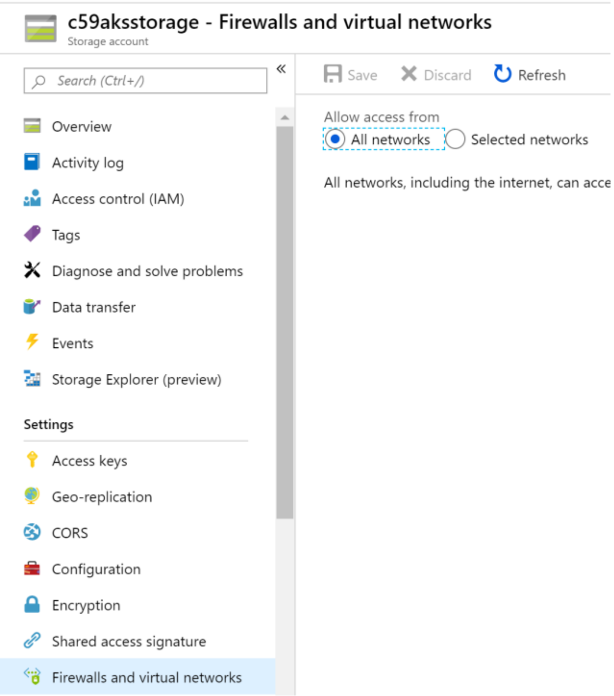
k apply -f winterready-web2.yaml

8. Next we are going to deploy the Ingress Controller which creates an NGINX service for L7 proxy as well as created an Azure internal Load-balancer.

k apply -f winterready-ingress-web.yaml

9. We need to upload the images in the fruit directory into an Azure Storage Account that is setup with a new Fileshare called Fruit where we store the images.

   a. Click on your storage account and then navigate to firewall and virtual networks. Click all networks and save.



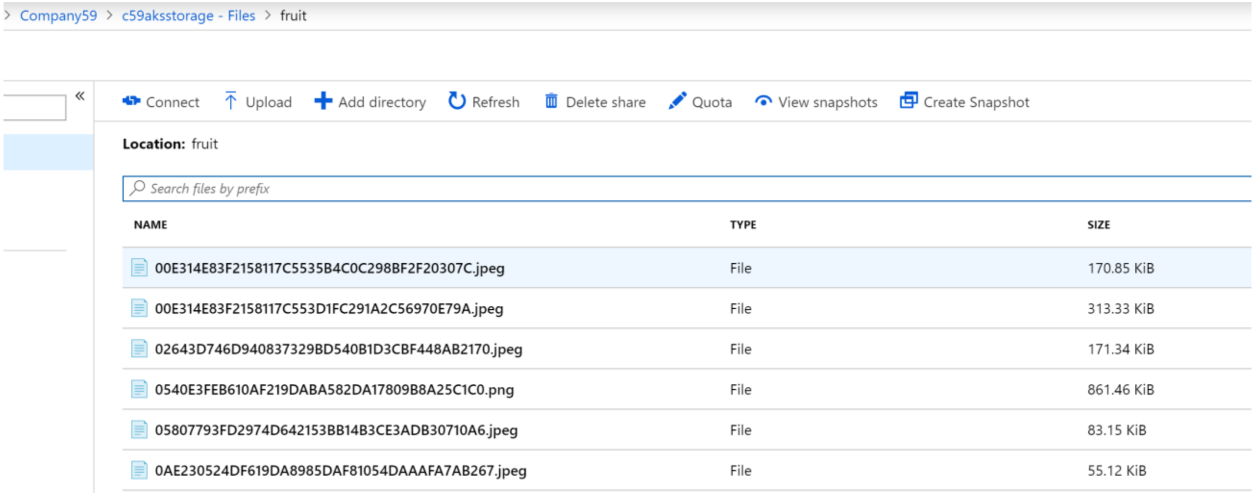
Allow access from

● All networks ○ Selected networks

All networks, including the internet, can acce

a. Click Overview and then click on Files. Click +File share and enter the name fruit with 1GB for Quota.

b. Click the new fileshare Fruit and click upload. Select all fruit images and upload.

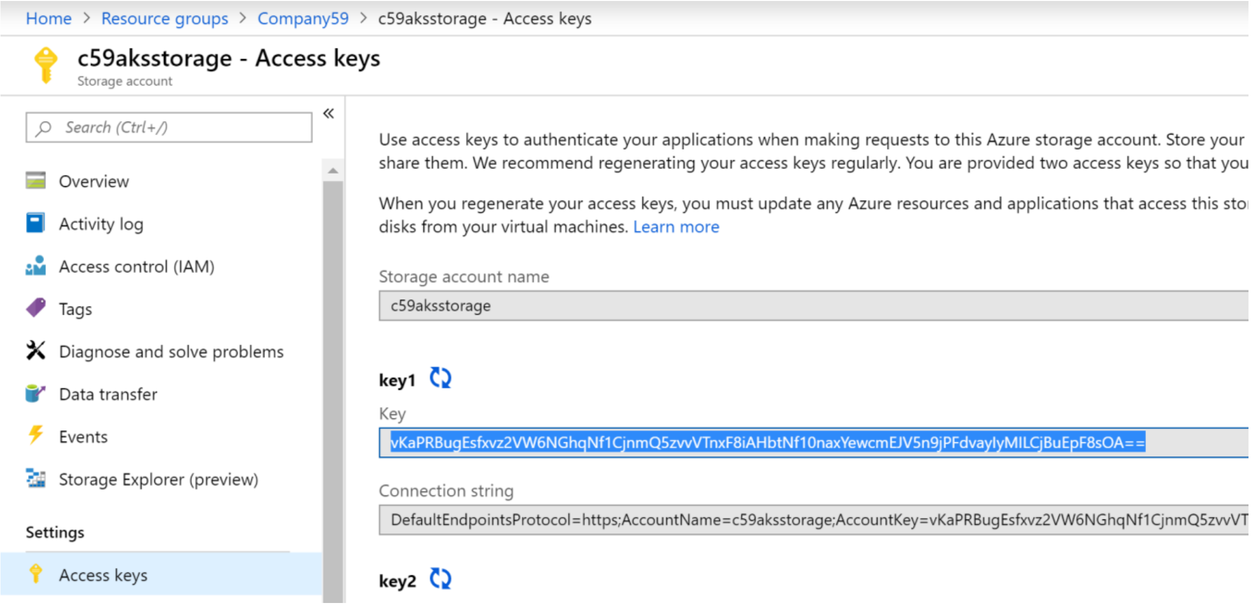


Location: fruit

NAME	TYPE	SIZE
00E314E83F2158117C5535B4C0C298BF2F20307C.jpeg	File	170.85 KiB
00E314E83F2158117C553D1FC291A2C56970E79A.jpeg	File	313.33 KiB
02643D746D940837329BD540B1D3CBF448AB2170.jpeg	File	171.34 KiB
0540E3FEB610AF219DABA582DA17809B8A25C1C0.png	File	861.46 KiB
05807793FD2974D642153BB14B3CE3ADB30710A6.jpeg	File	83.15 KiB
0AE230524DF619DA8985DAF81054DAAAFA7AB267.jpeg	File	55.12 KiB

10. We need to secure access to the file share. We will add in the storage account and keys to AKS.

# Update <STORAGE_ACCOUNT_NAME_GOES_HERE> and <STORAGE_ACCOUNT_KEY_GOES_HERE> below with the correct credentials from the Azure Storage account created above. Make sure you update the AzCLI with your storage acct name and key. Storage acct key can be found here:


Home > Resource groups > Company59 > c59aksstorage - Access keys
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Data transfer
Events
Storage Explorer (preview)
Settings
Access keys

c59aksstorage - Access keys

Storage account

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you

When you regenerate your access keys, you must update any Azure resources and applications that access this sto disks from your virtual machines. Learn more

Storage account name

c59aksstorage

key1

Key

vKaPRBugEsfxvz2VW6NGhqNf1CjnmQ5zvvVTnxF8iAHbtNf10naxYewcmEJV5n9jPFdvaylyMILCjBuEpF8sOA==

Connection string

DefaultEndpointsProtocol=https;AccountName=c59aksstorage;AccountKey=vKaPRBugEsfxvz2VW6NGhqNf1CjnmQ5zvvVT

key2

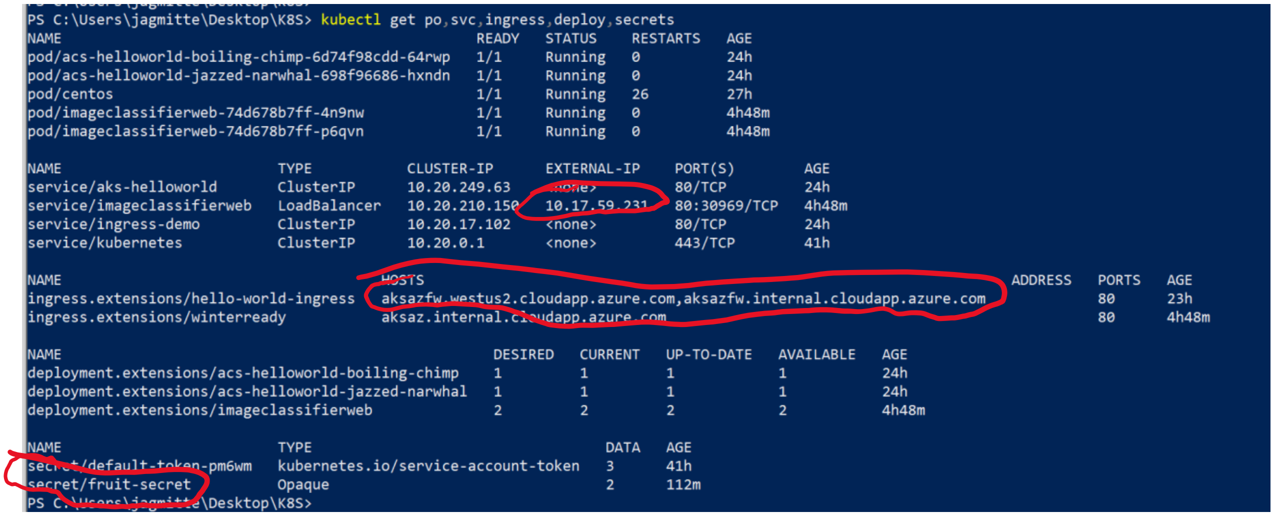
kubectl create secret generic fruit-secret --from-literal=azurestorageaccountname=c59aksstorage --from-literal=azurestorageaccountkey=vKaPRBugEsfxvz2VW6NGhqNf1CjnmQ5zvvVTnxF8iAHbtNf10naxYewcmEJV5n9jPFdvayIyMILCjBuEpF8sOA==

11. Time to deploy the Worker Back-End:

kubectl apply -f .\winterready-worker.yaml

12. Lets take a look at the services we created. Pay close attention to the service with the "external-ip" which defines the private IP on Azure LB. You can also see the hosts FQDNs and Secret key for accessing Azure Files.

kubectl get po,svc,ingress,deploy,secrets



```
PS C:\Users\jagmitte\Desktop\K8S> kubectl get po,svc,ingress,deploy,secrets
NAME                                            READY   STATUS    RESTARTS   AGE
pod/acs-helloworld-boiling-chimp-6d74f98cdd-64rwp   1/1     Running   0          24h
pod/acs-helloworld-jazzed-narwhal-698f96686-hxndn   1/1     Running   0          24h
pod/centos                                      1/1     Running   26         27h
pod/imageclassifierweb-74d678b7ff-4n9nw         1/1     Running   0          4h48m
pod/imageclassifierweb-74d678b7ff-p6qvn         1/1     Running   0          4h48m

NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/aks-helloworld       ClusterIP      10.20.249.63    <none>        80/TCP         24h
service/imageclassifierweb   LoadBalancer   10.20.210.150   10.17.59.231  80:30969/TCP   4h48m
service/ingress-demo         ClusterIP      10.20.17.102    <none>        80/TCP         24h
service/kubernetes           ClusterIP      10.20.0.1       <none>        443/TCP        41h

NAME                                   HOSTS                                                            ADDRESS   PORTS   AGE
ingress.extensions/hello-world-ingress aksazfw.westus2.cloudapp.azure.com,aksazfw.internal.cloudapp.azure.com          80      23h
ingress.extensions/winterready         aksaz.internal.cloudapp.azure.com                                               80      4h48m

NAME                                            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/acs-helloworld-boiling-chimp   1         1         1            1           24h
deployment.extensions/acs-helloworld-jazzed-narwhal  1         1         1            1           24h
deployment.extensions/imageclassifierweb        2         2         2            2           4h48m

NAME                         TYPE                                  DATA   AGE
secret/default-token-pm6wm   kubernetes.io/service-account-token   3      41h
secret/fruit-secret          Opaque                                2      112m
PS C:\Users\jagmitte\Desktop\K8S>
```

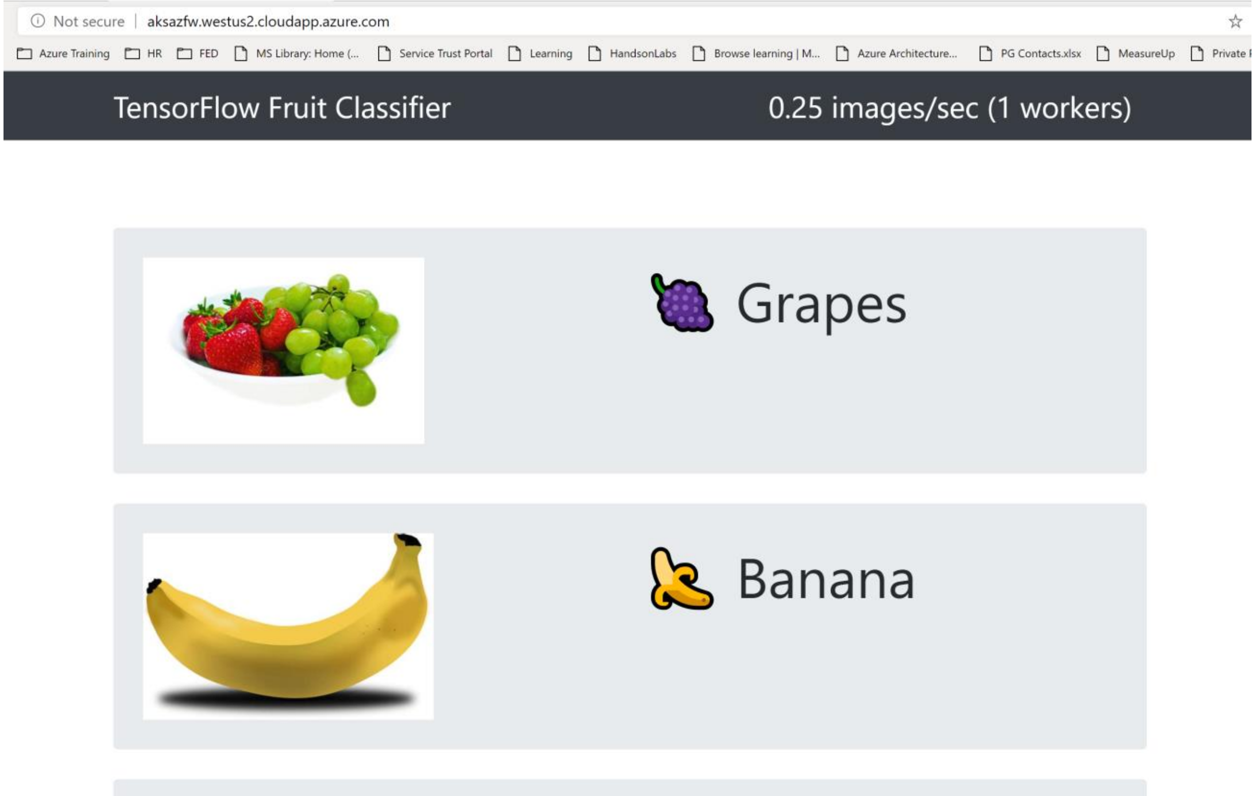
13. Now we can retrieve the internal load balancer ip and register it in the azure firewall as a Dnat rule.

```
$SERVICE_IP=$(kubectl get svc imageclassifierweb --template="{{range
.status.loadBalancer.ingress}}{{.ip}}{{end}}")
```

14. Lets up the Azure FW rule to permit traffic from the Internet with DNAT.

```
az network firewall nat-rule create  --firewall-name $FWNAME --collection-name "inboundlbrules" --
name "allow inbound port 80" --protocols "TCP" --source-addresses "*" --resource-group $RG --action
"Dnat"  --destination-addresses $IP --destination-ports 80 --translated-address $SERVICE_IP --translated-
port "80"  --priority 600
```

15. Now you can access the internal service by going to the DNS assigned to the public IP of your azure firewall on port 80
http://$DNSNAME.westus2.cloudapp.azure.com

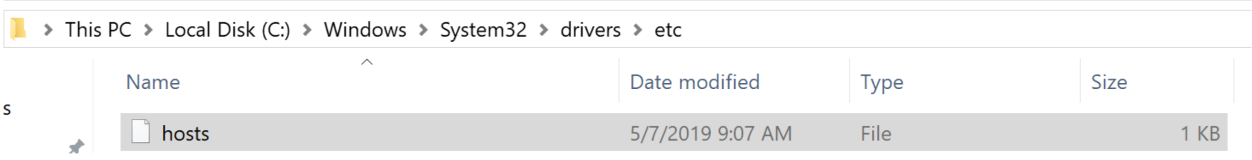


Last item is accessing the service via ExpressRoute Private-Peering.

1. Navigate to VM that is on-premise. Update the host file on the host and update with the private FQDN you created above with the private IP address of Azure Load Balancer.

2. Search for notepad and right click and open as administrator

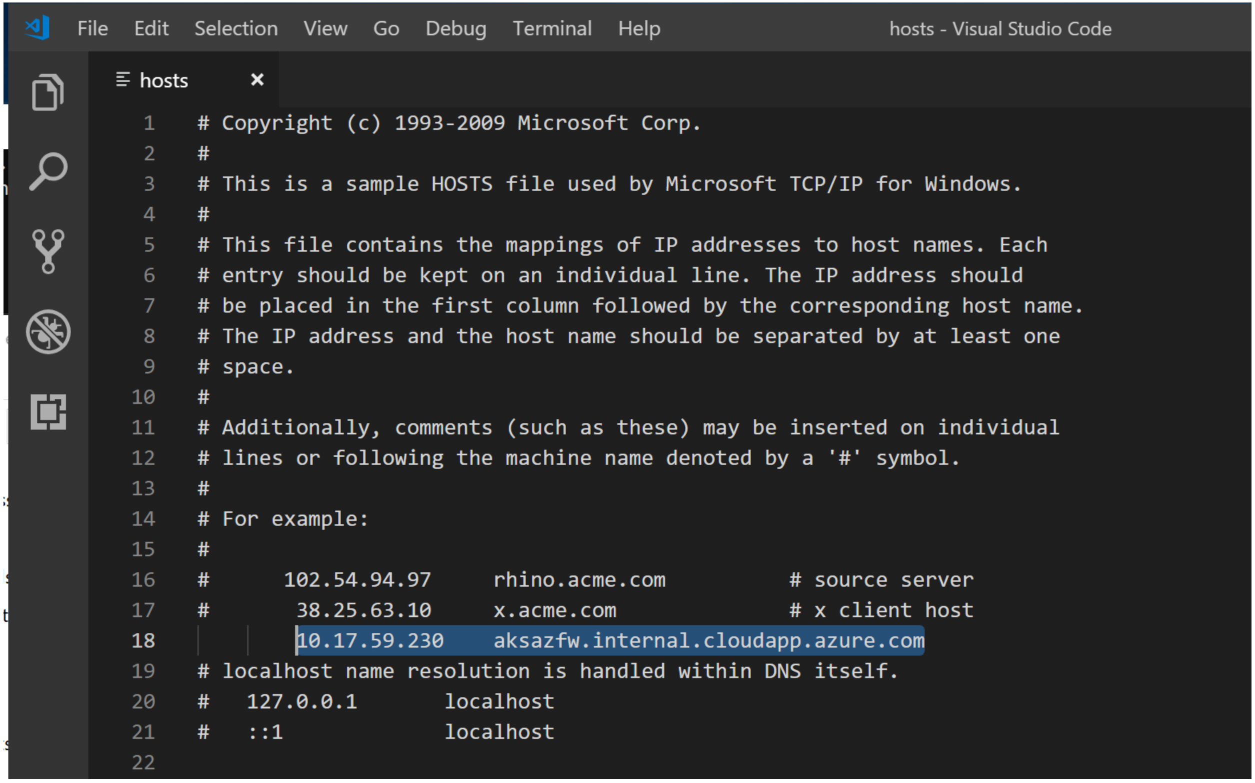
3. Open the host file using the following path:



This PC > Local Disk (C:) > Windows > System32 > drivers > etc

Name	Date modified	Type	Size
hosts	5/7/2019 9:07 AM	File	1 KB

Edit the host file with the private IP address of your Azure LoadBalancer and Internal FQDN you defined earlier in the hello-world-ingress.yml



```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host
        10.17.59.230    aksazfw.internal.cloudapp.azure.com
# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost

```

4. Create a new Azure Firewall rule that permits traffic to the Azure Load-Balancer IP address from any source.

az network firewall network-rule create -g $RG -f $FWNAME --collection-name 'onpremises' --name "allow network" --protocols 'TCP' --source-addresses '*' --destination-addresses $SERVICE_IP --destination-ports 80 --action allow --priority 300

5. Create a route-table that routes the private IP of the Azure Load Balancer with its next-hop being the private IP address of Azure Firewall.

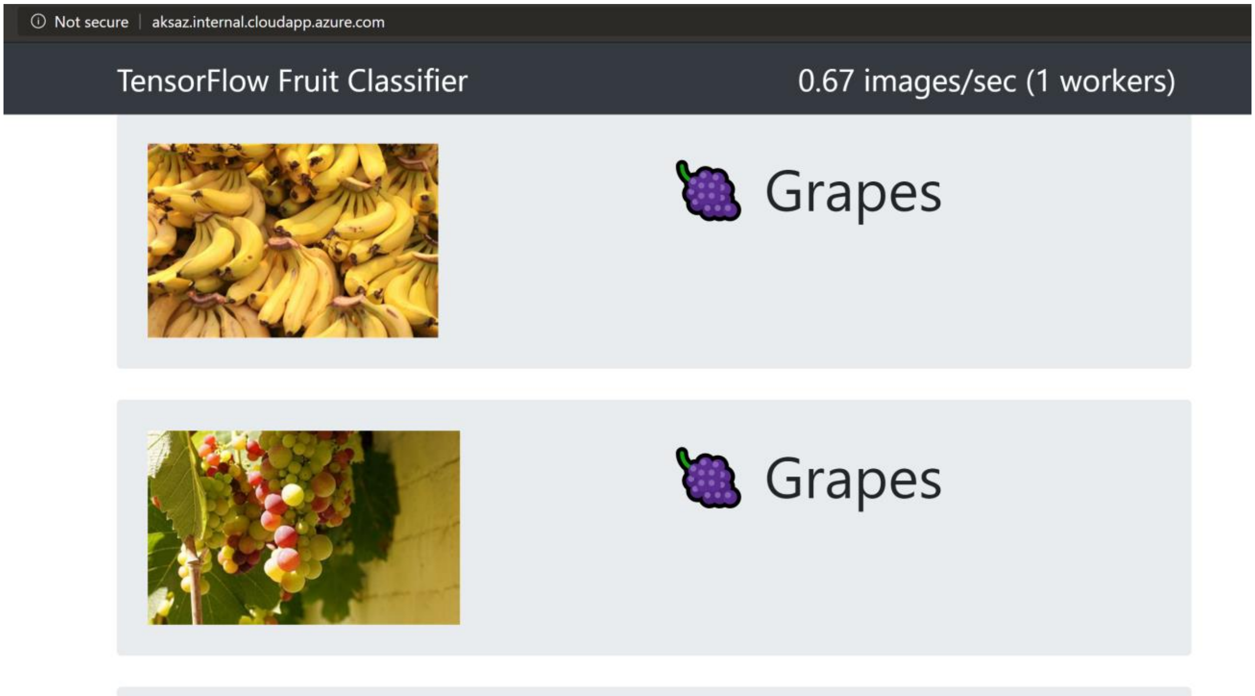
```
az network route-table create --resource-group $RG --name OnPremisesAKS
```

```
az network route-table route create --name AKS --resource-group $RG --route-table-name OnPremisesAKS --address-prefix "$SERVICE_IP/32" --next-hop-type VirtualAppliance --next-hop-ip-address $FWPRIVATE_IP
```

6. Assign the route-table to the Gateway Subnet to service-chain Azure Firewall in path to Azure Load-Balancer.

```
az network vnet subnet update --vnet-name C59-VNet --name GatewaySubnet --resource-group $RG --route-table OnPremisesAKS
```

7. Open a browser on test vm and access the internal FQDN.



ⓘ Not secure | aksaz.internal.cloudapp.azure.com

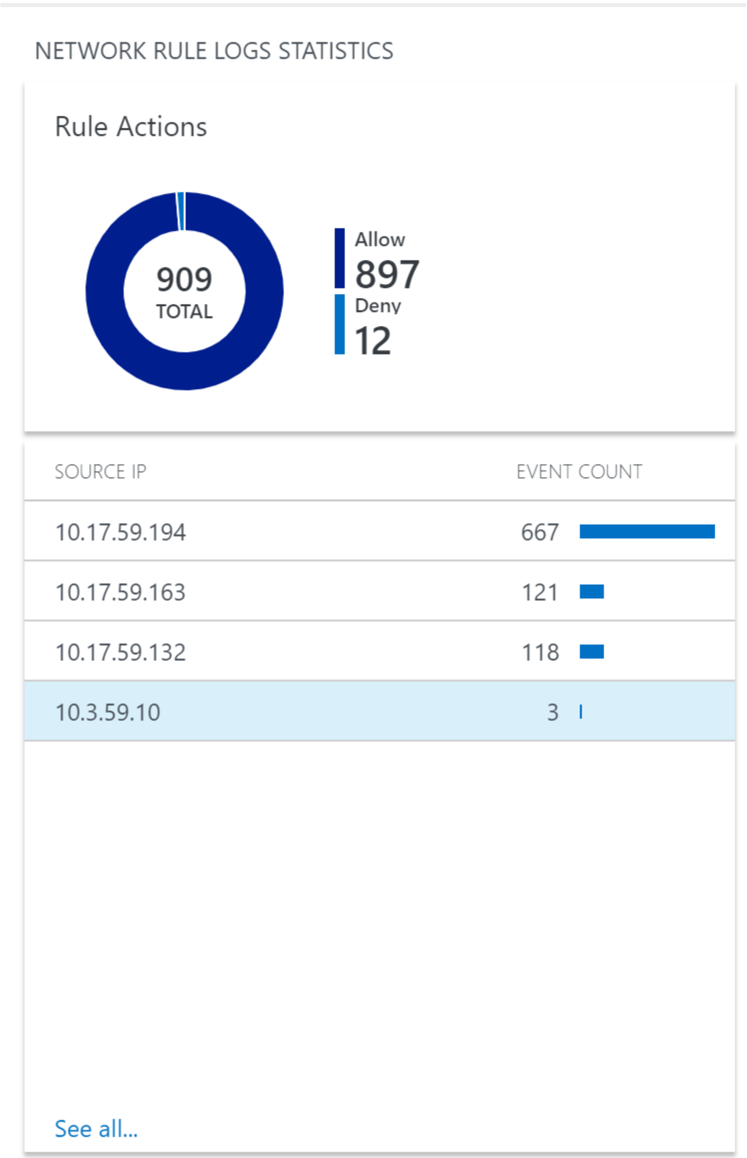
TensorFlow Fruit Classifier

0.67 images/sec (1 workers)

🍇 Grapes

🍇 Grapes

8. Review Logs within Azure Log Analytics to verify. Navigate to Dashboard and click on Azure FW. Scroll over to Network Rule Logs Statistics. Find the private IP or your on-premises VM:



NETWORK RULE LOGS STATISTICS

Rule Actions

909 TOTAL

Allow 897

Deny 12

SOURCE IP	EVENT COUNT
10.17.59.194	667
10.17.59.163	121
10.17.59.132	118
10.3.59.10	3

See all...

Review the status of all the containers in the deployment. Run the following commands:

```
kubectl get po,svc,ingress,deploy,secrets  --all-namespaces
```

```
kubectl get pods -o wide
kubectl describe pod “PODNAME”
kubectl get pods -o wide --all-namespaces
kubectl get svc
kubectl describe svc “SVCNAME”
```

```
kubectl get ingress
kubectl get ingress hello-world-ingress
kubectl describe ingress hello-world-ingress
```