Dennis Zielke

Follow 298 Followers About

Setting up azure firewall for analysing outgoing traffic in AKS

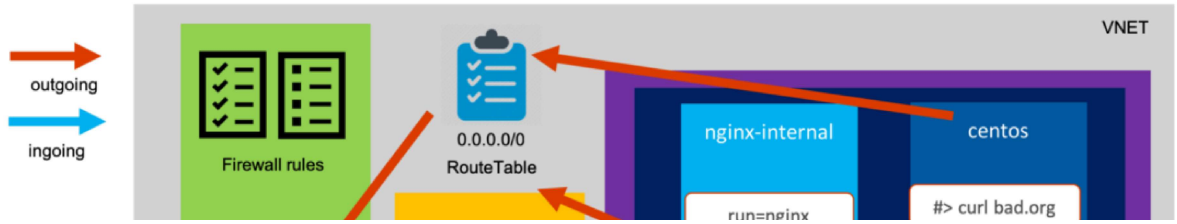
Dennis Zielke Jan 19, 2019 · 9 min read

Every now and then we get the question on how to lock down ingoing to and outgoing traffic from a kubernetes cluster in azure. One option that can be set up relatively easy but is not documented in detail is using the Azure Firewall (https://azure.microsoft.com/en-us/services/azure-firewall/).

> *My personal recommendation on this scenario is to use the firewall to diagnose and watch the network dependencies of your applications — which is why I am also documenting the services that are **currently** needed for AKS to run. If you turn on the rules to **block** outgoing traffic, you **risk** that your cluster **breaks** if the engineering team brings in additional required network dependencies. Be careful on production environments with this.*

The official documentation (once fully supported) will be made available: https://docs.microsoft.com/en-us/azure/aks/limit-egress-traffic

The end result will look like this and requires 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:



First we will set up the vnet — I prefer using azure cli in shell.azure.com over powershell but you can easy achieve the same using terraform or arm. If you have preferences on the naming conventions please adjust the variables below. In most companies the vnet is provided by the networking team, so we should assume that the network configuration will not be done by the team which is maintaining the AKS cluster.

Lets define some variables first:

```
SUBSCRIPTION_ID=$(az account show --query id -o tsv) # here enter
your subscription id

KUBE_GROUP="kubes_fw_knet" # here enter the resources group name of
your AKS cluster

KUBE_NAME="dzkubekube" # here enter the name of your kubernetes
resource

LOCATION="westeurope" # here enter the datacenter location

KUBE_VNET_NAME="knets" # here enter the name of your vnet

KUBE_FW_SUBNET_NAME="AzureFirewallSubnet" # this you cannot change

APPGW_SUBNET_NAME="gw-1-subnet"

KUBE_ING_SUBNET_NAME="ing-4-subnet" # here enter the name of your
ingress subnet

KUBE_AGENT_SUBNET_NAME="aks-5-subnet" # here enter the name of your
AKS subnet

FW_NAME="dzkubenetfw" # here enter the name of your azure firewall
resource

APPGW_NAME="dzkubeappgw"
```

```
KUBE_CNI_PLUGIN="azure" # alternative "kubenet"
```

1. Select subscription, create the resource group and the vnet

```
az account set --subscription $SUBSCRIPTION_ID

az group create -n $KUBE_GROUP -l $LOCATION

az network vnet create -g $KUBE_GROUP -n $KUBE_VNET_NAME
```

2. Create your service principal and assign permissions on vnet for your service principal — usually “virtual machine contributor” is enough

```
SERVICE_PRINCIPAL_ID=$(az ad sp create-for-rbac --skip-assignment --
name $KUBE_NAME-sp -o json | jq -r '.appId')

SERVICE_PRINCIPAL_SECRET=$(az ad app credential reset --id
$SERVICE_PRINCIPAL_ID -o json | jq '.password' -r)

sleep 5 # wait for replication

az role assignment create --role "Contributor" --assignee
$SERVICE_PRINCIPAL_ID -g $KUBE_GROUP
```

3. Create subnets for the firewall, ingress and AKS

```
az network vnet subnet create -g $KUBE_GROUP --vnet-name
$KUBE_VNET_NAME -n $KUBE_FW_SUBNET_NAME --address-prefix 10.0.3.0/24

az network vnet subnet create -g $KUBE_GROUP --vnet-name
$KUBE_VNET_NAME -n $APPGW_SUBNET_NAME --address-prefix 10.0.2.0/24

az network vnet subnet create -g $KUBE_GROUP --vnet-name
$KUBE_VNET_NAME -n $KUBE_ING_SUBNET_NAME --address-prefix
10.0.4.0/24

az network vnet subnet create -g $KUBE_GROUP --vnet-name
$KUBE_VNET_NAME -n $KUBE_AGENT_SUBNET_NAME --address-prefix
10.0.5.0/24 --service-endpoints Microsoft.Sql
Microsoft.AzureCosmosDB Microsoft.KeyVault Microsoft.Storage
```

network traffic I would recommend to import the diagnostic dashboard via this file as described here: https://docs.microsoft.com/en-us/azure/firewall/tutorial-diagnostics.

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

az network public-ip create -g $KUBE_GROUP -n $FW_NAME --sku
Standard

az network firewall create --name $FW_NAME --resource-group
$KUBE_GROUP --location $LOCATION

az network firewall ip-config create --firewall-name $FW_NAME --name
$FW_NAME --public-ip-address $FW_NAME --resource-group $KUBE_GROUP -
-vnet-name $KUBE_VNET_NAME

FW_PRIVATE_IP=$(az network firewall show -g $KUBE_GROUP -n $FW_NAME
--query "ipConfigurations[0].privateIpAddress" -o tsv)

az monitor log-analytics workspace create --resource-group
$KUBE_GROUP --workspace-name $FW_NAME --location $LOCATION
```

To make sure that all outgoing traffic gets routed through the firewall we need to configure a custom routetable, point it at the internal ip of our firewall and connect it to our AKS cluster subnet.

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

az network public-ip create -g $KUBE_GROUP -n $FW_NAME-ip --sku
Standard

FW_PUBLIC_IP=$(az network public-ip show -g $KUBE_GROUP -n $FW_NAME-
ip --query ipAddress)

az network firewall create --name $FW_NAME --resource-group
$KUBE_GROUP --location $LOCATION

az network firewall ip-config create --firewall-name $FW_NAME --name
$FW_NAME --public-ip-address $FW_NAME-ip --resource-group
$KUBE_GROUP --vnet-name $KUBE_VNET_NAME

FW_PRIVATE_IP=$(az network firewall show -g $KUBE_GROUP -n $FW_NAME
--query "ipConfigurations[0].privateIpAddress" -o tsv)
```

```
KUBE_AGENT_SUBNET_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups
/$KUBE_GROUP/providers/Microsoft.Network/virtualNetworks/$KUBE_VNET_
NAME/subnets/$KUBE_AGENT_SUBNET_NAME"

az network route-table create -g $KUBE_GROUP --name $FW_NAME-rt

az network route-table route create --resource-group $KUBE_GROUP --
name $FW_NAME --route-table-name $FW_NAME-rt --address-prefix
0.0.0.0/0 --next-hop-type VirtualAppliance --next-hop-ip-address
$FW_PRIVATE_IP

az network vnet subnet update --route-table $FW_NAME-rt --ids
$KUBE_AGENT_SUBNET_ID

az network route-table route list --resource-group $KUBE_GROUP --
route-table-name $FW_NAME-rt
```

The next step is to create the necessary exception rules for the AKS-required network dependencies that will ensure the worker nodes can set their system time, pull ubuntu updates and retrieve the AKS core container images from the Microsoft Container Registry. The following will allow dns, time and the service tags for the azure container registry.

```
az network firewall network-rule create --firewall-name $FW_NAME --
collection-name "time" --destination-addresses "*"  --destination-
ports 123 --name "allow network" --protocols "UDP" --resource-group
$KUBE_GROUP --source-addresses "*" --action "Allow" --description
"aks node time sync rule" --priority 101

az network firewall network-rule create --firewall-name $FW_NAME --
collection-name "dns" --destination-addresses "*"  --destination-
ports 53 --name "allow network" --protocols "Any" --resource-group
$KUBE_GROUP --source-addresses "*" --action "Allow" --description
"aks node dns rule" --priority 102

az network firewall network-rule create --firewall-name $FW_NAME --
collection-name "servicetags" --destination-addresses
"AzureContainerRegistry" "MicrosoftContainerRegistry"
"AzureActiveDirectory" "AzureMonitor" --destination-ports "*" --name
"allow service tags" --protocols "Any" --resource-group $KUBE_GROUP
--source-addresses "*" --action "Allow" --description "allow service
tags" --priority 110

az network firewall network-rule create --firewall-name $FW_NAME --
collection-name "hcp" --destination-addresses "AzureCloud.$LOCATION"
--destination-ports "1194" --name "allow master tags" --protocols
"UDP" --resource-group $KUBE_GROUP --source-addresses "*" --action
```

Since not all of the dependencies can be allowed through a network rule because they neither have a service tag or a fixed ip range. So these allow rules are defined using dns, while using the AKS fqdsible. The latest list of these entries can be found here.

```
az network firewall application-rule create --firewall-name $FW_NAME
--resource-group $KUBE_GROUP --collection-name 'aksfwar' -n 'fqdn' -
-source-addresses '*' --protocols 'http=80' 'https=443' --fqdn-tags
"AzureKubernetesService" --action allow --priority 101

az network firewall application-rule create  --firewall-name
$FW_NAME --collection-name "osupdates" --name "allow network" --
protocols http=80 https=443 --source-addresses "*" --resource-group
$KUBE_GROUP --action "Allow" --target-fqdns "download.opensuse.org"
"security.ubuntu.com" "packages.microsoft.com"
"azure.archive.ubuntu.com" "changelogs.ubuntu.com" "snapcraft.io"
"api.snapcraft.io" "motd.ubuntu.com"  --priority 102
```

In case you want to allow images from docker hub (which we will need later to test the filter rules), you might want to add the following rule:

```
az network firewall application-rule create  --firewall-name
$FW_NAME --collection-name "dockerhub" --name "allow network" --
protocols http=80 https=443 --source-addresses "*" --resource-group
$KUBE_GROUP --action "Allow" --target-fqdns "*auth.docker.io"
"*cloudflare.docker.io" "*cloudflare.docker.com" "*registry-
1.docker.io" --priority 200
```

As you might know there are two different options on how networking can be set up in AKS called “Basic networking” (using kubenet cni) and “Advanced Networking” (using azure cni). I am not going into detail how they differ — you can look it up here: https://docs.microsoft.com/en-us/azure/aks/concepts-network . For the usage of azure firewall in this scenario it does not matter (for new clusters)

```
KUBE_AGENT_SUBNET_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups
/$KUBE_GROUP/providers/Microsoft.Network/virtualNetworks/$KUBE_VNET_
NAME/subnets/$KUBE_AGENT_SUBNET_NAME"

az aks create --resource-group $KUBE_GROUP --name $KUBE_NAME --node-
count 2 --network-plugin $KUBE_CNI_PLUGIN --vnet-subnet-id
```

```
--kubernetes-version $KUBE_VERSION --no-ssh-key --outbound-type
userDefinedRouting
```

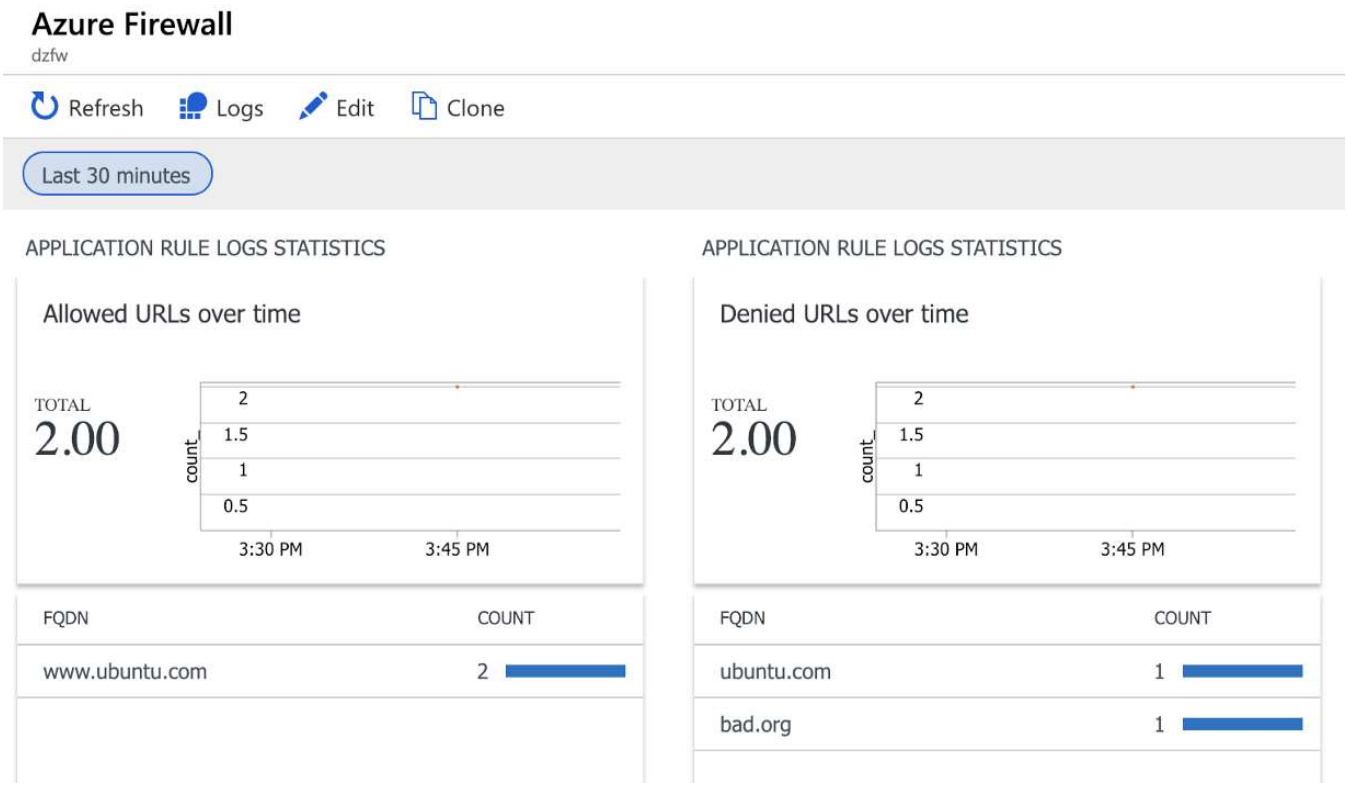
Finally we also want to add logging to our AKS cluster by deploying a log analytics space and connect our AKS monitoring addon to use that space.

```
az monitor log-analytics workspace create --resource-group
$KUBE_GROUP --workspace-name $KUBE_NAME --location $LOCATION

WORKSPACE_ID=$(az monitor log-analytics workspace show --resource-
group $KUBE_GROUP --workspace-name $KUBE_NAME -o json | jq '.id' -r)

az aks enable-addons --resource-group $KUBE_GROUP --name $KUBE_NAME
--addons monitoring --workspace-resource-id $WORKSPACE_ID
```

Now that the firewall works lets set up diagnostics and import the dashboard file as described here: https://docs.microsoft.com/en-us/azure/firewall/tutorial-diagnostics.



We can see that the firewall is working correctly and blocking ubuntu.com but not www.ubuntu.com

your api server first use the following command to get the IP address of your API server.

Get the IP address of your api server

```
HCP_IP=$(kubectl get endpoints -o=jsonpath='{.items[?
(@.metadata.name == "kubernetes")].subsets[].addresses[].ip}')
```

Be aware that the API Server IP address should stay stable during normal operations but it might change in case of an internal failover. The default rule above will allow traffic from your AKS subnet to the whole azure region on target port 1194 . In case you want to narrow this down further you can do this after cluster deployment by limiting access to port 1194 only on your master ip address by running the following command

```
az network firewall network-rule create --firewall-name $FW_NAME --
collection-name "aksnetwork" --destination-addresses "$HCP_IP"  --
destination-ports 1194 --name "allow network" --protocols "UDP" --
resource-group $KUBE_GROUP --source-addresses "*" --action "Allow" -
-description "aks master rule" --priority 120
```

Currently AKS needs the following outgoing network dependencies (check official docs for updated list):

- `*.hcp.<location>.azmk8s.io` (eg. *.hcp.westeurope.azmk8s.io) — this is the dns that is running your api server(HTTPS:443, UDP: 1194)
- `mcr.microsoft.com` , `*.data.mcr.microsoft.com` — This is required since some AKS images are coming from the Microsoft Container Registry (HTTPS: 443)
- `management.azure.com` — This is where azure management endpoint lives (HTTPS: 443)

- `acs-mirror.azureedge.net` — This is where azure keeps the cni plugin (HTTPS:443)

These cannot be filtered on a network level and need therefore be configured using application rules.

Optionally you might want to allow the following traffic:

- “download.opensuse.org” “security.ubuntu.com” “packages.microsoft.com” “azure.archive.ubuntu.com” “changelogs.ubuntu.com” “snapcraft.io” “api.snapcraft.io” “motd.ubuntu.com” — This is needed for security patches and updates — if the customer wants them to be applied automatically
- “*auth.docker.io” “*cloudflare.docker.io” “*cloudflare.docker.com” “*registry-1.docker.io” — In case you want to download images from DockerHub

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.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: centos
spec:
  containers:
  - name: centoss
    image: centos
    ports:
    - containerPort: 80
    command:
    - sleep
    - "3600"
EOF
```

Now log inside and verify that we cannot curl to an external url that is not allowed while www.ubuntu.com is working fine.

```
➜  ~ kubectl exec -ti centos -- /bin/bash
[root@centos /]# curl www.ubuntu.com
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
```

Azure firewall working as expected, allowing www.ubuntu.com but blocking everything else.

One more thing might interest you: “How to expose a kubernetes service through the azure firewall?” If you expose a service through the normal LoadBalancer with a public ip, it will not be accessible because the traffic that has not been routed through the azure firewall will be dropped on the way out. Therefore you need to create your service with a fixed internal ip, internal LoadBalancer and route the traffic through the azure firewall both for outgoing and incoming traffic. If you only want to use internal load balancers, then you do not need to do this.

```
kubectl run nginx --image=nginx --port=80

cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: nginx-internal
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    service.beta.kubernetes.io/azure-load-balancer-internal-subnet:
"ing-4-subnet"
spec:
  type: LoadBalancer
  loadBalancerIP: 10.0.4.4
  ports:
  - port: 80
  selector:
    run: nginx
EOF
```

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

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

az network firewall nat-rule create  --firewall-name $FW_NAME --
collection-name "inboundlbrules" --name "allow inbound load
balancers" --protocols "TCP" --source-addresses "*" --resource-group
$KUBE_GROUP --action "Dnat"  --destination-addresses $FW_PUBLIC_IP -
-destination-ports 80 --translated-address $SERVICE_IP --translated-
port "80"  --priority 101
```

on port 80

```
open http://$FW_PUBLIC_IP:80
```

Our internal service is no reachable through the azure firewall public ip on port 80

THE END ;-)

Kubernetes Azure Firewall Azure Kubernetes Service Networking