Dennis Zielke

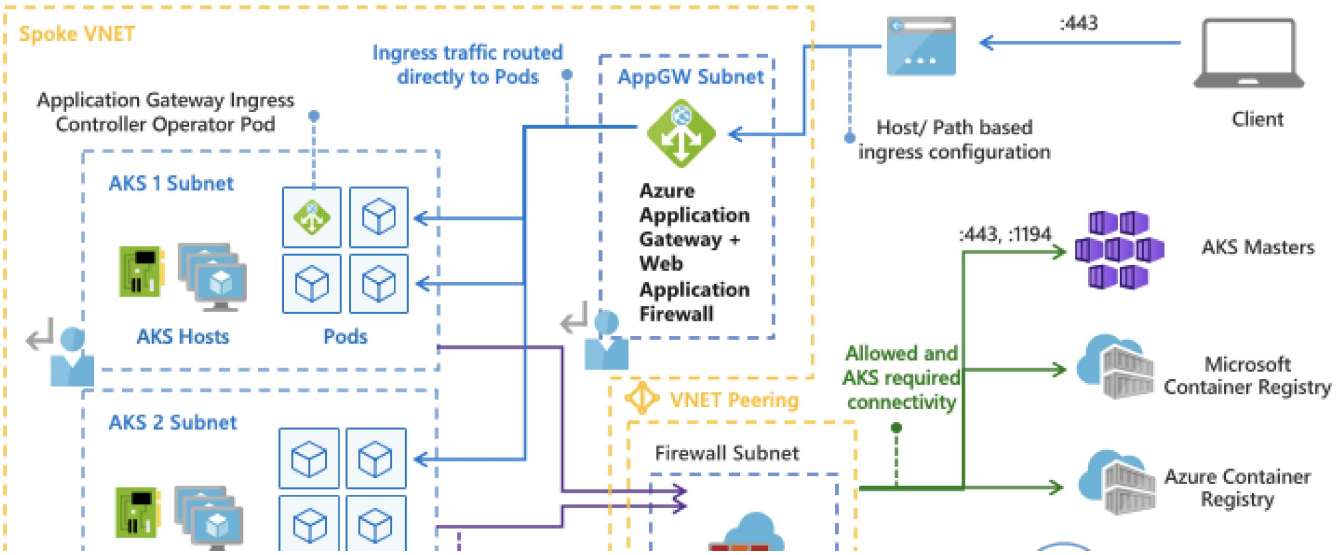
Follow 294 Followers About

Securing ingress with AzureAppGateway and egress traffic with AzureFirewall for Azure Kubernetes Service

Dennis Zielke Sep 28, 2020 · 12 min read

With the general availability of outboundtype routing parameter for AKS and the Application Gateway Ingress Controller we are frequently receiving the question on how to set up an Kubernetes environment that secures both the ingress and the egress traffic with a firewall.

In this post I want to guide you on how to set up an environment in which your AKS cluster is routing all incoming traffic through an Application Gateway and is using an Azure Firewall so ensure that your worker nodes and pods can only connect to services and ip ranges that you have explicitly whitelisted. I will also give you a list of the current known issues around this scenario. Here is how the result will look like:



The setup can be extended to multiple AKS cluster behind the same AppGW and Azure Firewall instance

We want to achieve the following architecture design requirements:

- Hub and Spoke network design in which we can maintain the firewall and the whitelisted egress traffic in the hub network.
- Ability to route traffic from the application gateway into multiple clusters so that we perform advanced deployments like blue/green deployments .
- No public ips on the AKS clusters itself, however this will not implement fully private aks clusters since there are some limitations in this design with respect to private link.
- We want to maintain minimal privileges across the relevant resources VNET, Firewall, AppGateway and AKS and avoid to use secrets while using managed identities where possible

First lets define some variables which you are welcome to customize to fit your naming guidelines.

```
SUBSCRIPTION_ID=$(az account show --query id -o tsv) #subscriptionid

LOCATION="westeurope" # here enter the datacenter location

VNET_GROUP="securevnets" # here enter the network resource group
name

HUB_VNET_NAME="hubnet" # here enter the name of your hub net

KUBE_VNET_NAME="k8snet" # here enter the name of your k8s vnet

FW_NAME="dzk8sfw1" # name of your azure firewall resource

APPGW_NAME="dzk8sappgw1"

APPGW_GROUP="secureappgw" # here enter the appgw resource group name
```

```
KUBE_AGENT2_SUBNET_NAME="aks-2-subnet" # name of your AKS subnet

KUBE_GROUP="securek8s" # here enter the resources group name

KUBE_NAME="secureaks1" # here enter the name of your aks resource

KUBE_VERSION="$(az aks get-versions -l $LOCATION --query
'orchestrators[?default == `true`].orchestratorVersion' -o tsv)" #
here enter the kubernetes version of your AKS

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

As before we will create our resource group, vnet and subnets

```
az group create -n $KUBE_GROUP -l $LOCATION

az group create -n $VNET_GROUP -l $LOCATION

az group create -n $APPGW_GROUP -l $LOCATION

echo "setting up vnet"

az network vnet create -g $VNET_GROUP -n $HUB_VNET_NAME --address-
prefixes 10.0.0.0/22

az network vnet create -g $VNET_GROUP -n $KUBE_VNET_NAME --address-
prefixes 10.0.4.0/22

az network vnet subnet create -g $VNET_GROUP --vnet-name
$HUB_VNET_NAME -n AzureFirewallSubnet --address-prefix 10.0.0.0/24

az network vnet subnet create -g $VNET_GROUP --vnet-name
$HUB_VNET_NAME -n bastionsubnet --address-prefix 10.0.1.0/24

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

az network vnet subnet create -g $VNET_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

az network vnet subnet create -g $VNET_GROUP --vnet-name
$KUBE_VNET_NAME -n $KUBE_AGENT2_SUBNET_NAME --address-prefix
10.0.6.0/24 --service-endpoints Microsoft.Sql
Microsoft.AzureCosmosDB Microsoft.KeyVault Microsoft.Storage
```

```
az network vnet peering create -g $VNET_GROUP -n Spoke1ToHub --vnet-
name $KUBE_VNET_NAME --remote-vnet $HUB_VNET_NAME --allow-vnet-
access
```

As next step we will create our azure firewall instance and a log analytics space to host our azure firewall logs and metrics

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

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

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

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

az network firewall ip-config create --firewall-name $FW_NAME --name
$FW_NAME --public-ip-address $FW_NAME-ip --resource-group
$VNET_GROUP --vnet-name $HUB_VNET_NAME

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

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

Next we need to set up the user defined routes to force all outgoing traffic out of our AKS subnet to the private ip of our azure firewall instance.

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

az network route-table create -g $VNET_GROUP --name $KUBE_NAME-rt

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

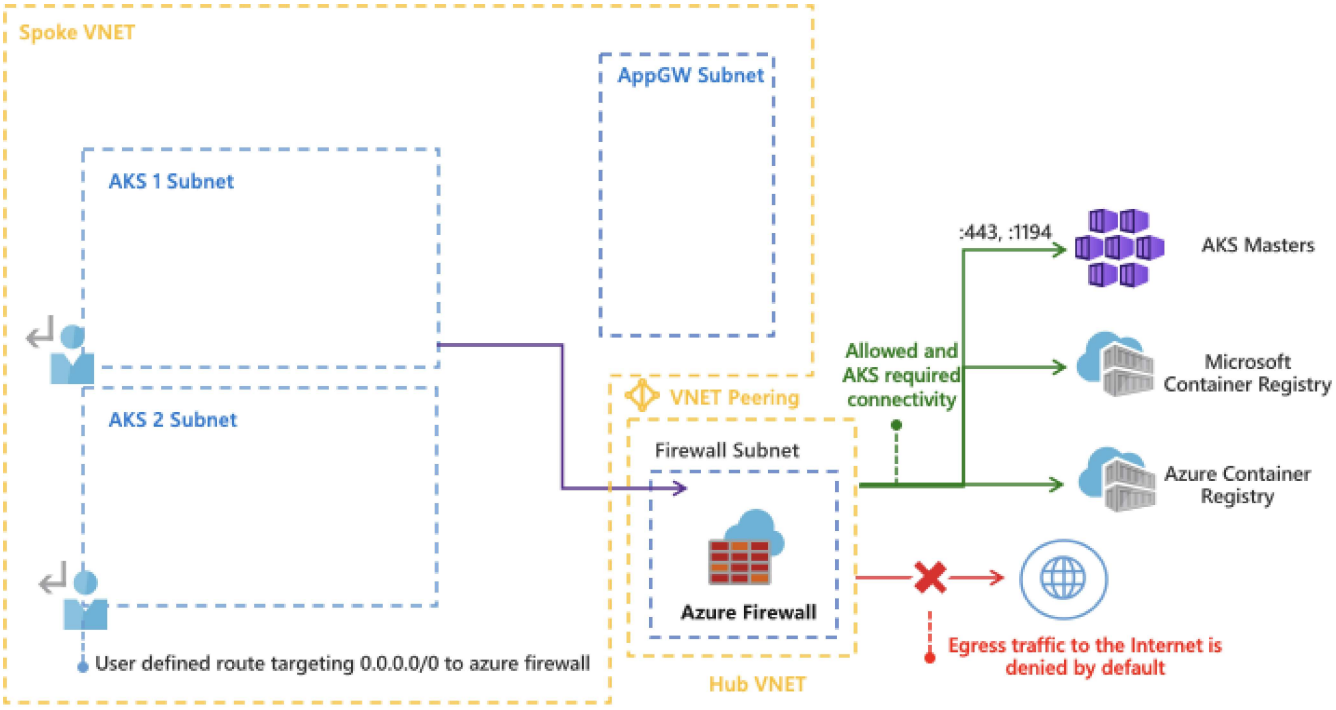
```
az network route-table route list --resource-group $VNET_GROUP --
route-table-name $KUBE_NAME-rt
```

Since we are deploying an AKS instance with a public ip we need to allow a couple of required network dependencies for Azure in the Region, Kubernetes, AKS Masters, Microsoft Container Registry, DNS and Time Server. The official list is maintained here.

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

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

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

az network firewall network-rule create --firewall-name $FW_NAME --
resource-group $VNET_GROUP --collection-name "hcp" --destination-
addresses "AzureCloud.$LOCATION" --destination-ports "1194" --name
"allow master tags" --protocols "UDP" --source-addresses "*" --
action "Allow" --description "allow aks link access to masters" --
priority 120

az network firewall application-rule create --firewall-name $FW_NAME
--resource-group $VNET_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 --resource-group $VNET_GROUP --collection-name "osupdates"
--name "allow network" --protocols http=80 https=443 --source-
addresses "*"  --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
```

containers. This is something you might not want to do for your production environment.

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



Our vnet configuration forces all traffic from the aks subnets to the azure firewall instance

Now it is time to deploy our AKS cluster, which requires to do some preparation and decision with respect to the cluster identities and the integration of pods in the azure network. You can read up on the different cni plugin here, but in most of the cases it comes down to the question if you can afford a routeable ip for every container or not.

```
NAME/subnets/$KUBE_AGENT_SUBNET_NAME"

KUBE_VNET_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$VNET_G
ROUP/providers/Microsoft.Network/virtualNetworks/$KUBE_VNET_NAME"
```

For using azure cni the setup is fairly simple: we provision the the cluster and later need to assign Virtual Machine Contributor permissions of the system assigned managed controller identity to the virtual network resource so that the cluster can manage ip and load balancer assignments.

```
if [ "$KUBE_CNI_PLUGIN" == "azure" ]; then

az aks create --resource-group $KUBE_GROUP --name $KUBE_NAME --node-
resource-group $KUBE_GROUP"_"$KUBE_NAME"_nodes_"$LOCATION --node-
count 2 --network-plugin $KUBE_CNI_PLUGIN --vnet-subnet-id
$KUBE_AGENT_SUBNET_ID --docker-bridge-address 172.17.0.1/16 --dns-
service-ip 10.2.0.10 --service-cidr 10.2.0.0/24 --kubernetes-version
$KUBE_VERSION --no-ssh-key --enable-managed-identity --outbound-type
userDefinedRouting

CONTROLLER_ID=$(az aks show -g $KUBE_GROUP -n $KUBE_NAME --query
identity.principalId -o tsv)

az role assignment create --role "Virtual Machine Contributor" --
assignee $CONTROLLER_ID --scope $KUBE_VNET_ID

fi

az aks get-credentials -g $KUBE_GROUP -n $KUBE_NAME
```

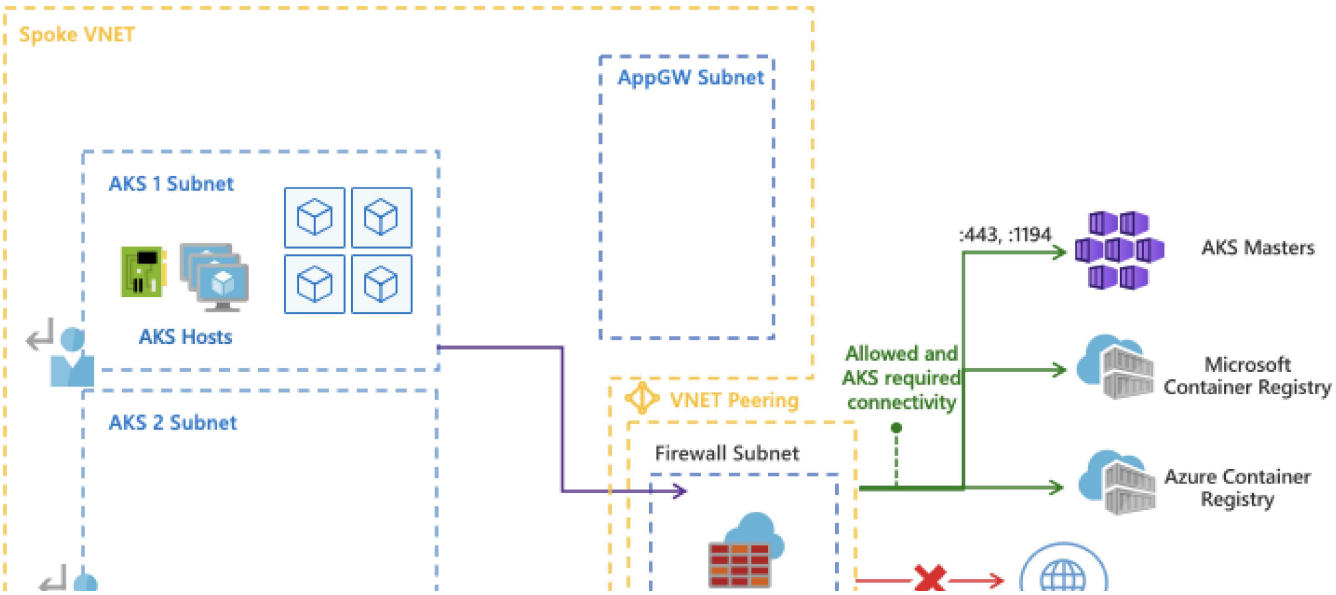
If you want to use kubenet (or you simply do not have ips), then you need to revert back to using a service principal because you cannot use the outboundtype in combination with a managed identity yet. At some point the feature of using a user assigned managed identity for the controller will help, but it is not available yet.

```
if [ "$KUBE_CNI_PLUGIN" == "kubenet" ]; then

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

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

```
az role assignment create --role "Virtual Machine Contributor" --
assignee $SERVICE_PRINCIPAL_ID --scope $KUBE_VNET_ID

az role assignment create --role "Contributor" --assignee
$SERVICE_PRINCIPAL_ID --scope $ROUTETABLE_ID

az aks create --resource-group $KUBE_GROUP --name $KUBE_NAME --node-
resource-group $KUBE_GROUP"_"$KUBE_NAME"_nodes_"$LOCATION --node-
count 2 --network-plugin $KUBE_CNI_PLUGIN --vnet-subnet-id
$KUBE_AGENT_SUBNET_ID --docker-bridge-address 172.17.0.1/16 --dns-
service-ip 10.2.0.10 --service-cidr 10.2.0.0/24 --kubernetes-version
$KUBE_VERSION --no-ssh-key --client-secret $SERVICE_PRINCIPAL_SECRET
--service-principal $SERVICE_PRINCIPAL_ID --outbound-type
userDefinedRouting

fi

az aks get-credentials -g $KUBE_GROUP -n $KUBE_NAME
```

Now that your cluster has been created we want to activate azure monitor and container insights in our environment.

```
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
```



All outgoing traffic from our AKS cluster has to go through our azure firewall, but no ingress yet

For now there is no means of routing incoming traffic from the internet to our AKS cluster. Due to asymmetric routing issues we cannot simply expose a Kubernetes service with a public LoadBalancer IP and therefore we need to create our Application Gateway instance to route incoming traffic to the pods. Since we want to use not only the Application Gateway as an Ingress Controller but also as a Layer 7 firewall we need to configure the WAF_v2 sku and ensure that it is configured in the right subnet.

```
APPGW_SUBNET_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$VNE
T_GROUP/providers/Microsoft.Network/virtualNetworks/$KUBE_VNET_NAME/
subnets/$APPGW_SUBNET_NAME"

az network public-ip create --resource-group $APPGW_GROUP --name
$APPGW_NAME-pip --allocation-method Static --sku Standard

APPGW_PUBLIC_IP=$(az network public-ip show -g $APPGW_GROUP -n
$APPGW_NAME-pip --query ipAddress -o tsv)

az network application-gateway create --name $APPGW_NAME --resource-
group $APPGW_GROUP --location $LOCATION --http2 Enabled --min-
capacity 0 --max-capacity 10 --sku WAF_v2  --vnet-name
$KUBE_VNET_NAME --subnet $APPGW_SUBNET_ID --http-settings-cookie-
based-affinity Disabled --frontend-port 80 --http-settings-port 80 -
-http-settings-protocol Http --public-ip-address $APPGW_NAME-pip

APPGW_NAME=$(az network application-gateway list --resource-
group=$APPGW_GROUP -o json | jq -r ".[0].name")

APPGW_RESOURCE_ID=$(az network application-gateway list --resource-
group=$APPGW_GROUP -o json | jq -r ".[0].id")

APPGW_SUBNET_ID=$(az network application-gateway list --resource-
group=$APPGW_GROUP -o json | jq -r ".
[0].gatewayIpConfigurations[0].subnet.id")
```

You might have seen that there is a managed version of the Application Gateway Ingress Controller in the form of an AKS addon. Unfortunately this addon does not work in combination with a firewall yet, which is why we need to install the open source version of the aad pod identity, appgateway ingress controller and do the required plumming ourselves to make this work.

the kubelet. If you are using kubenet, then this step is not necessary.

```
if [ "$KUBE_CNI_PLUGIN" == "azure" ]; then

KUBELET_ID=$(az aks show -g $KUBE_GROUP -n $KUBE_NAME --query
identityProfile.kubeletidentity.clientId -o tsv)

CONTROLLER_ID=$(az aks show -g $KUBE_GROUP -n $KUBE_NAME --query
identity.principalId -o tsv)

NODE_GROUP=$(az aks show --resource-group $KUBE_GROUP --name
$KUBE_NAME --query nodeResourceGroup -o tsv)

az role assignment create --role "Managed Identity Operator" --
assignee $KUBELET_ID --scope
/subscriptions/$SUBSCRIPTION_ID/resourcegroups/$NODE_GROUP

az role assignment create --role "Managed Identity Operator" --
assignee $CONTROLLER_ID --scope
/subscriptions/$SUBSCRIPTION_ID/resourcegroups/$NODE_GROUP

az role assignment create --role "Virtual Machine Contributor" --
assignee $KUBELET_ID --scope
/subscriptions/$SUBSCRIPTION_ID/resourcegroups/$NODE_GROUP

fi

helm repo add aad-pod-identity
https://raw.githubusercontent.com/Azure/aad-pod-
identity/master/charts

helm repo update

helm upgrade aad-pod-identity --install --namespace kube-system aad-
pod-identity/aad-pod-identity
```

After the aad pod identity has been bootstapped (which might take a couple of minutes) you can configure and install the application gateway ingress controller. In order for the controller to authenticate to the Application Gateway Instance we created earlier if will use a dedicated user managed identity which we will create and grant permissions to change the routes according to our ingress definitions.

```
az identity create -g $NODE_GROUP -n $APPGW_NAME-id
```

```
id  --query clientId -o tsv)"

AGIC_ID_RESOURCE_ID="$(az identity show -g $NODE_GROUP -n
$APPGW_NAME-id  --query id -o tsv)"

NODES_RESOURCE_ID=$(az group show -n $NODE_GROUP -o tsv --query
"id")

KUBE_GROUP_RESOURCE_ID=$(az group show -n $KUBE_GROUP -o tsv --query
"id")

sleep 5 # wait for replication

az role assignment create --role Contributor --assignee
$AGIC_ID_CLIENT_ID --scope $APPGW_RESOURCE_ID

az role assignment create --role Reader --assignee
$AGIC_ID_CLIENT_ID --scope $KUBE_GROUP_RESOURCE_ID

az role assignment create --role Reader --assignee
$AGIC_ID_CLIENT_ID --scope $NODES_RESOURCE_ID

helm repo add application-gateway-kubernetes-ingress
https://appgwingress.blob.core.windows.net/ingress-azure-helm-
package/

helm repo update

helm upgrade ingress-azure application-gateway-kubernetes-
ingress/ingress-azure --namespace kube-system --install --set
appgw.name=$APPGW_NAME --set appgw.resourceGroup=$KUBE_GROUP --set
appgw.subscriptionId=$SUBSCRIPTION_ID --set appgw.usePrivateIP=false
--set appgw.shared=false --set armAuth.type=aadPodIdentity --set
armAuth.identityClientID=$AGIC_ID_CLIENT_ID --set
armAuth.identityResourceID=$AGIC_ID_RESOURCE_ID --set
rbac.enabled=true --set verbosityLevel=3 --set
kubernetes.watchNamespace=default
```

Just in case you have made a decision to use **kubenet** as your cni plugin you have to do the following extra steps. Since the Application Gateway will be routing traffic directly to our pods and by selecting kubenet you made the pod ips non-routeable in azure you have to configure an additional routetable to the Application Gateway subnet to make sure that the Application Gateway can actually route directly to the host where your pods are running. The following will show you how to achieve the correct routing configuration initially. If you change the cluster size you have to adjust the routing configuration accordingly.

```
AKS_NODE_NSG=$(az network nsg list -g ${NODE_GROUP} --query "[].id |
[0]" -o tsv)

az network route-table create -g $APPGW_GROUP --name $APPGW_NAME-rt

APPGW_ROUTE_TABLE_ID=$(az network route-table show -g ${APPGW_GROUP}
-n $APPGW_NAME-rt --query "id" -o tsv)

az network nsg create --name $APPGW_NAME-nsg --resource-group
$APPGW_GROUP --location $LOCATION

az network nsg rule create --name Allow_GWM --nsg-name $APPGW_NAME-
nsg --priority 100 --resource-group $APPGW_GROUP --access Allow --
direction Inbound --destination-port-ranges 65200-65535 --source-
address-prefixes GatewayManager --description "Required for Gateway
Manager."

az network nsg rule create --name agic_allow --nsg-name $APPGW_NAME-
nsg --resource-group $APPGW_GROUP --priority 120 --source-address-
prefixes '*' --source-port-ranges '*' --destination-address-prefixes
"$APPGW_PUBLIC_IP" --destination-port-ranges '80' --access Allow --
direction Inbound --protocol "*" --description "Required allow rule
for Ingress Controller."

APPGW_NSG=$(az network nsg list -g ${APPGW_GROUP} --query "[].id |
[0]" -o tsv)

az network vnet subnet update --route-table $APPGW_ROUTE_TABLE_ID --
network-security-group $APPGW_NSG --ids $APPGW_SUBNET_ID

AKS_ROUTES=$(az network route-table route list --resource-group
$VNET_GROUP --route-table-name $KUBE_NAME-rt)

az network route-table route list --resource-group $VNET_GROUP --
route-table-name $KUBE_NAME-rt --query "[]
[name,addressPrefix,nextHopIpAddress]" -o tsv |

while read -r name addressPrefix nextHopIpAddress; do

echo "checking route $name"

echo "creating new hop $name selecting $addressPrefix configuring
$nextHopIpAddress as next hop"

az network route-table route create --resource-group $APPGW_GROUP --
name $name --route-table-name $APPGW_NAME-rt --address-prefix
$addressPrefix --next-hop-type VirtualAppliance --next-hop-ip-
address $nextHopIpAddress --subscription $SUBSCRIPTION_ID

done
```

```
fi
```

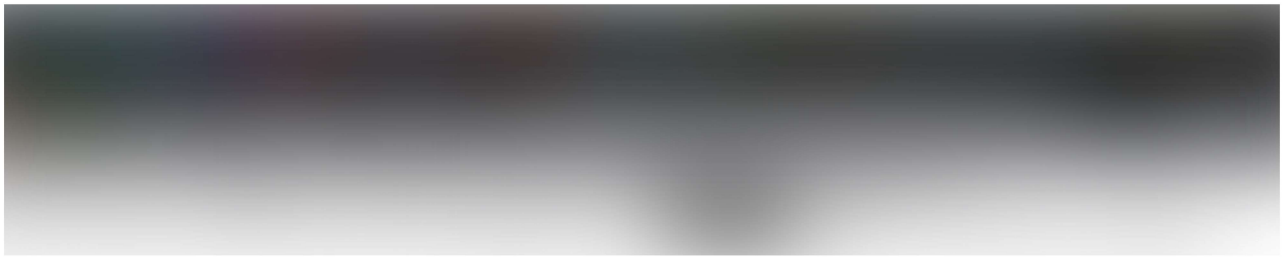
If everything worked out we can now see an ingress-azure pod in our kube-system namespace which will continuously watch out for new ingress objects with the ‘azure/application-gateway’ ingress class annotation and configure the routes in our Application Gateway accordingly.

Now that our Ingress Controller is correctly configures let’s deploy a sample application and an ingress object to try it out.

```
kubectl apply -f
https://raw.githubusercontent.com/denniszielke/container_demos/master/logging/dummy-logger/depl-logger.yaml

kubectl apply -f
https://raw.githubusercontent.com/denniszielke/container_demos/master/logging/dummy-logger/svc-logger.yaml

cat <<EOF | kubectl apply -f -
apiVersion: extensions/v1beta1
kind: Ingress
```

```
    kubernetes.io/ingress.class: azure/application-gateway
spec:
  rules:
  - host: $APPGW_PUBLIC_IP.xip.io
    http:
      paths:
      - backend:
          serviceName: dummy-logger
          servicePort: 80
EOF
```

If we now open a browser on our the dns address that we just configured for our application gateway public ip we can see the response from the dummy-logger.



Once the routing has been applied you can navigate to your ingress url

We can see the source ip address of the call which has reached our dummy logger is 10.0.2.4 which is one of the internal egress ips of the application gateway. However we can also see the forwarded ip address from your actual client ip which the application gateway is populating in the x-forwarded-header of the request, which is a very useful feature.

Same as before we can also launch a dummy container and validate that our egress limitations enforced by our azure firewall are still in place.

if you want you can repeat the same for additional AKS clusters in different subnets of the same VNET and put them behind the same AppGateway and AzureFirewall instance by installing the application gateway ingress controller in shared mode. Clearly the usage of kubenet makes this setup more tricky and considering that you have to update the routetable entries I would not recommend to use kubenet in production yet. The same goes for the fully private shared mode with one private AppGW ingress ip for multiple clusters — please wait until the issues have been fixed.

I hope this helps you in your own environment.

The end ;)

Azure Kubernetes Service Firewall Azure Ingresscontroller Security