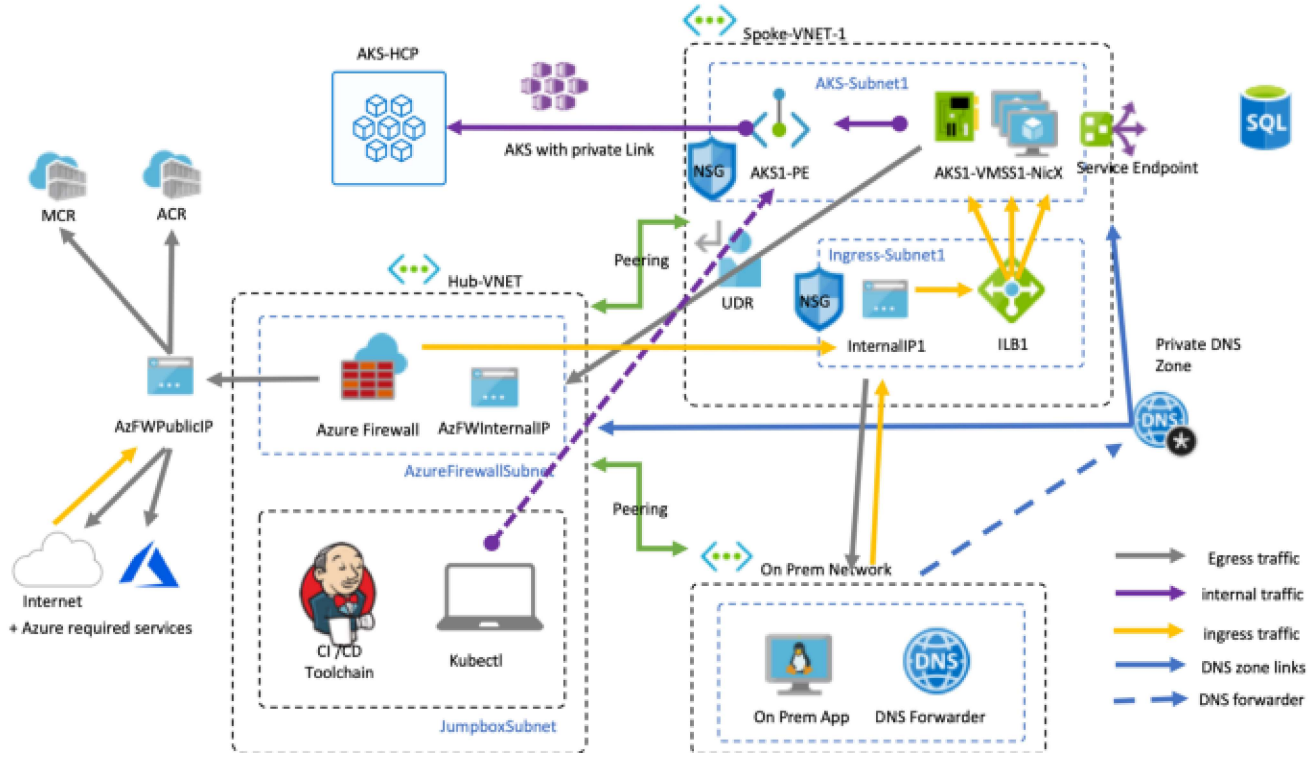
Dennis Zielke

Follow 293 Followers About

Fully private AKS clusters — without any public ips — finally!

Dennis Zielke Feb 5, 2020 · 9 min read

One year after my last post on leveraging an azure firewall I want to revisit the scenario since we just launched a couple of new features that finally allow you to build an AKS cluster that does not use or expose any public ips. The end result should look like this and this is a guide on how you build this in your azure environment:



A fully private AKS cluster that does not need to expose or connect to public IPs.

Update December 31st: ***This scenario is now fully compatible with Bring your own Managed Identity and Uptime-SLA yet.***

As you might know in AKS we maintain the connection between the master and the worker nodes through a pod called "**tunnelfront**" or "**akslink**" (depending on your cluster version) which is running on your worker nodes. The connection between them is created from akslink to the master — meaning from the inside to the outside. Up until recently it was not possible to create this connection without a public IP on the masters. This is now different with private link. Using private link we are projecting the private IP address of the AKS master, which is running in an azure managed virtual network into an ip address that is part of your AKS subnet.

We also removed the need for your worker nodes to have a public ip for egress attached to their standard load balancers. However to make sure that your worker nodes are able to sync time, pull images and authenticate to azure active directory they are still in need of an internet egress path. In our case we using the azure firewall to ensure that egress path.

To get started lets first set up some variables for the following scripts which will help you to customise your deployment:

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

DEPLOYMENT_NAME="dzpraks1" # short unique name, min 5 max 8 letters

LOCATION="westeurope" # here enter the datacenter location

KUBE_GROUP="${DEPLOYMENT_NAME}_kube" # here enter the resources
group name of your aks cluster

KUBE_NAME="${DEPLOYMENT_NAME}" # here enter the name of your
kubernetes resource

VNET_GROUP="${DEPLOYMENT_NAME}_networks" # here the name of the
resource group for the vnet and hub resources

KUBE_VNET_NAME="spoke1-kubevnet" # here enter the name of your vnet

KUBE_ING_SUBNET_NAME="ing-1-subnet" # here enter the name of your
ingress subnet
```

```
HUB_VNET_NAME="hub1-firewalvnet"

HUB_FW_SUBNET_NAME="AzureFirewallSubnet" # this you cannot change

HUB_JUMP_SUBNET_NAME="jumpbox-subnet"

KUBE_VERSION="$(az aks get-versions -l $LOCATION --query
'orchestrators[?default == `true`].orchestratorVersion' -o tsv)"
```

Now lets create the virtual networks, subnets and peerings between them:

```
az account set --subscription $SUBSCRIPTION_ID

az group create -n $KUBE_GROUP -l $LOCATION

az group create -n $VNET_GROUP -l $LOCATION

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 $HUB_FW_SUBNET_NAME --address-prefix 10.0.0.0/24

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

az network vnet subnet create -g $VNET_GROUP --vnet-name
$KUBE_VNET_NAME -n $KUBE_ING_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

az network vnet peering create -g $VNET_GROUP -n HubToSpoke1 --vnet-
name $HUB_VNET_NAME --remote-vnet $KUBE_VNET_NAME --allow-vnet-
access

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

Let's configure the azure firewall public ip, the azure firewall itself and a log analytics workspace for firewall logs. To make sure that you can see outbound 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 $VNET_GROUP -n $DEPLOYMENT_NAME --sku
Standard

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

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

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

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

Next we are creating a user defined route which will force all traffic from the AKS subnet to the internal ip of the azure firewall (which is an ip address that we know to be 10.0.0.4, stored in $FW_PRIVATE_IP).

```
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 $DEPLOYMENT_NAME

az network route-table route create --resource-group $VNET_GROUP --
name $DEPLOYMENT_NAME --route-table-name $DEPLOYMENT_NAME --address-
prefix 0.0.0.0/0 --next-hop-type VirtualAppliance --next-hop-ip-
address $FW_PRIVATE_IP --subscription $SUBSCRIPTION_ID

az network vnet subnet update --route-table $DEPLOYMENT_NAME --ids
$KUBE_AGENT_SUBNET_ID
```

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 kube-system 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
$DEPLOYMENT_NAME --collection-name "time" --destination-addresses
"*"  --destination-ports 123 --name "allow network" --protocols
"UDP" --resource-group $VNET_GROUP --source-addresses "*" --action
"Allow" --description "aks node time sync rule" --priority 101

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

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

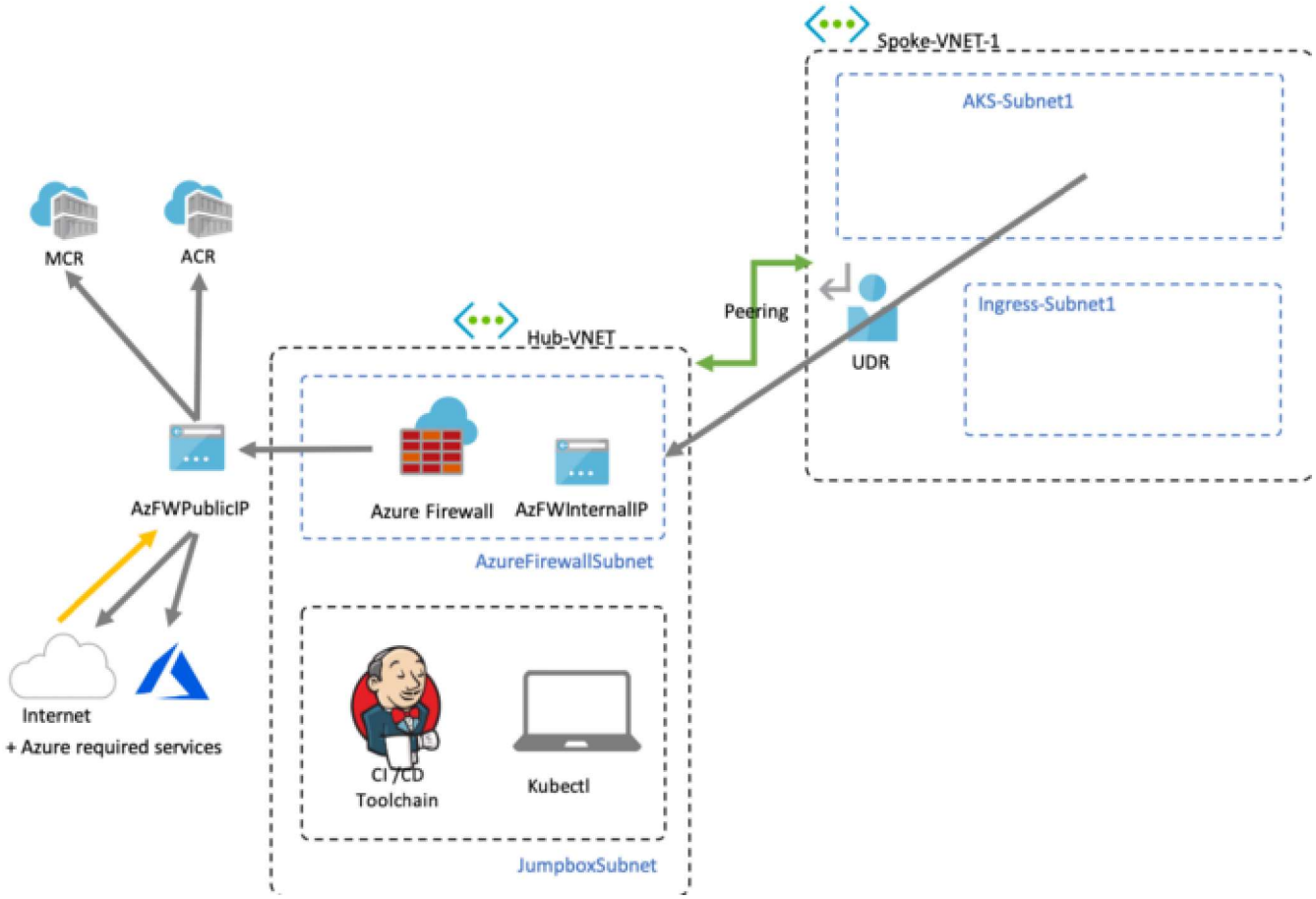
Not all of the dependencies can be allowed through a network rule because they neither have a service tag or a fixed ip range. So these allow rules are defined using dns, while using the AKS fqdn tags where possible (be aware that the FQDN Tag will contain the complete for all dependencies for all possible addons including all of github). The latest list of these entries can be found here — in case you want to define your own whitelist.

```
az network firewall application-rule create --firewall-name
$DEPLOYMENT_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
$DEPLOYMENT_NAME --collection-name "osupdates" --name "allow
```

```
 packages.microsoft.com"  "azure.archive.ubuntu.com"
"changelogs.ubuntu.com" "snapcraft.io" "api.snapcraft.io"
"motd.ubuntu.com"  --priority 102
```

Now we have created the virtual networks and subnets, the peering between them and a route from the AKS subnet to the internet which is going through our azure firewall that only allows the required AKS dependencies.

The current state looks like this without Kubernetes:



Your empty network design before you deploy the AKS cluster

I would also recommend to create a log analytics space and configure the diagnostic logs for the azure firewall to validate the traffic flows. See here and download the dashboard here.

balancer. In our case we want to prevent this by setting the *outboundType* to *userDefinedRouting* in our deployment and configure private cluster with a dedicated service principal. In the following we will be using kubenet as cni plugin. You can of course achieve the same using azure cni, but it will consume more ip addresses, which are typically a scare resource in most enterprise environments.

If you want to use a managed identity you need to create the user managed identity for the control plane ahead of time and assign it the correct permissions on the network resources.

```
az identity create --name $KUBE_NAME --resource-group $KUBE_GROUP
MSI_RESOURCE_ID=$(az identity show -n $KUBE_NAME -g $KUBE_GROUP -o
json | jq -r ".id")
MSI_CLIENT_ID=$(az identity show -n $KUBE_NAME -g $KUBE_GROUP -o
json | jq -r ".clientId")

az role assignment create --role "Virtual Machine Contributor" --
assignee $MSI_CLIENT_ID -g $VNET_GROUP

az aks create --resource-group $KUBE_GROUP --name $KUBE_NAME --load-
balancer-sku standard --vm-set-type VirtualMachineScaleSets --
enable-private-cluster --network-plugin azure --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 --enable-managed-
identity --assign-identity $MSI_RESOURCE_ID --kubernetes-version
$KUBE_VERSION --outbound-type userDefinedRouting
```

If you want to use a service principal (Managed Identities are recommended) you can use the following:

```
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)

sleep 5 # wait for service principal to propagate

az role assignment create --role "Contributor" --assignee
$SERVICE_PRINCIPAL_ID -g $VNET_GROUP

az aks create --resource-group $KUBE_GROUP --name $KUBE_NAME --load-
balancer-sku standard --vm-set-type VirtualMachineScaleSets --
```

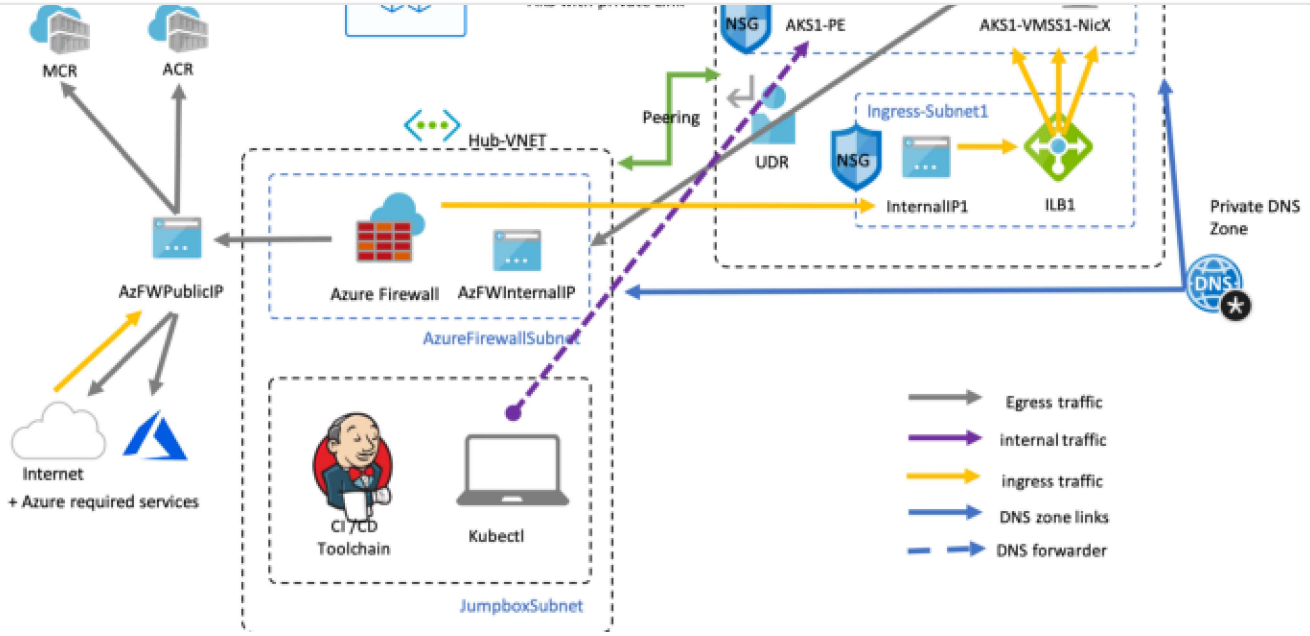
```
$SERVICE_PRINCIPAL_ID --client-secret $SERVICE_PRINCIPAL_SECRET --
kubernetes-version $KUBE_VERSION --outbound-type userDefinedRouting
```

After the deployment went through you need to create a link in your DNS zone to the hub net to make sure that your jumpbox is correctly able to resolve the dns name of your **private link** enabled cluster using kubectl from the jumbox. This will ensure that your jumpbox can resolve the private ip of the api server using azure dns. In case you need to bring your own private DNS resolution implementation I encourage you to check this out.

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

DNS_ZONE_NAME=$(az network private-dns zone list --resource-group
$NODE_GROUP --query "[0].name" -o tsv)

HUB_VNET_ID=$(az network vnet show -g $VNET_GROUP -n $HUB_VNET_NAME
--query id -o tsv)

az network private-dns link vnet create --name "hubnetdnsconfig" --
registration-enabled false --resource-group $NODE_GROUP --virtual-
network $HUB_VNET_ID --zone-name $DNS_ZONE_NAME
```

This is how your private dns zone should look like after the registration.

The final task is to deploy an Ubuntu Jumpbox into the JumboxSubnet so that you can, configure azure cli and pull kubeconfig for the private cluster. The end result should look like this:



A private link enabled cluster without any public ips on the worker nodes

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
```

If everything worked out you can create a jumpbox in the jumpbox-subnet, install the azure cli, and kubectl, pull the kubeconfig and connect to your now fully private cluster.

```
dennis@asjumpbox:~$ kubectl get pod,svc --all-namespaces
NAMESPACE     NAME                                        READY   STATUS    RESTARTS   AGE
default       pod/dummy-logger                            1/1     Running   0          39s
kube-system   pod/azure-cni-networkmonitor-7ns42          1/1     Running   0          8h
kube-system   pod/azure-cni-networkmonitor-f5p6c          1/1     Running   0          8h
kube-system   pod/azure-cni-networkmonitor-gxzb5          1/1     Running   0          8h
kube-system   pod/azure-ip-masq-agent-4hnrt               1/1     Running   0          8h
kube-system   pod/azure-ip-masq-agent-ls9zk               1/1     Running   0          8h
kube-system   pod/azure-ip-masq-agent-q8qht               1/1     Running   0          8h
kube-system   pod/coredns-698c77c5d7-cgj6f                1/1     Running   0          8h
kube-system   pod/coredns-autoscaler-79b778686c-zc2j2     1/1     Running   0          8h
kube-system   pod/kube-proxy-4kpst                        1/1     Running   0          7h23m
kube-system   pod/kube-proxy-7zbx7                        1/1     Running   0          7h24m
```

```
NAMESPACE     NAME                              TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
default       service/dummy-logger-int-lb       LoadBalancer   10.2.0.240    10.0.5.98     80:30837/TCP     19s
default       service/kubernetes                ClusterIP      10.2.0.1      <none>        443/TCP          8h
default       service/nginx-internal            LoadBalancer   10.2.0.143    10.0.4.25     80:32473/TCP     8h
kube-system   service/kube-dns                  ClusterIP      10.2.0.10     <none>        53/UDP,53/TCP    8h
kube-system   service/kubernetes-dashboard      ClusterIP      10.2.0.110    <none>        80/TCP           8h
kube-system   service/metrics-server            ClusterIP      10.2.0.50     <none>        443/TCP          8h
```

Your very own fully private cluster

Same as before you can now launch a container see if its access attempts to the internet are correctly blocked. In this case we need to allow access to docker hub in our firewall.

```
az network firewall application-rule create  --firewall-name
$DEPLOYMENT_NAME --collection-name "dockerhub" --name "allow
network" --protocols http=80 https=443 --source-addresses "*" --
resource-group $VNET_GROUP --action "Allow" --target-fqdns
"auth.docker.io" "registry-1.docker.io" --priority 110

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
```

```
➜  ~ 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>
<p>The document has moved <a href="https://www.ubuntu.com/">here</a>.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at www.ubuntu.com Port 80</address>
</body></html>
[root@centos /]# curl batman.com
HTTP  request from 10.0.5.4:37668 to batman.com:80. Action: Deny. No rule matched. Proceeding with default action
```

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

A couple more considerations are on how you can make sure that dns resolution works, which is described here. Unfortunately there are a couple of limitations like these and

Still to be solved is how you can ensure routing from on prem network. The simple way is to use a DNAT rule, which unfortunately only works when you target the public ip of your azure firewall as ingress point and not the private ip of the azure firewall. The more complicated setup requires gateway transit and remote gateways as described here.

The end ;)

Kubernetes Security Firewall Azure Azure Kubernetes Service