In this lab you will get an overview of network security groups (NSG) and learn how to do the following:

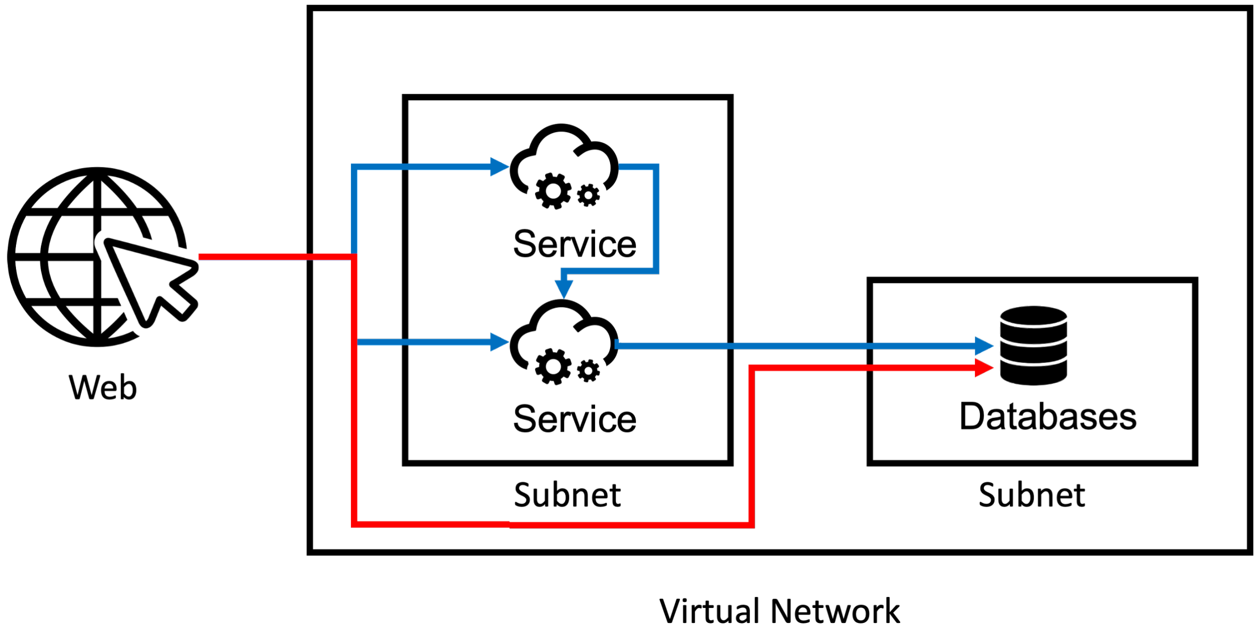
Log in to the Azure CLI

Create an NSG

Create a rules within an NSG

Step 2 - Getting Set Up

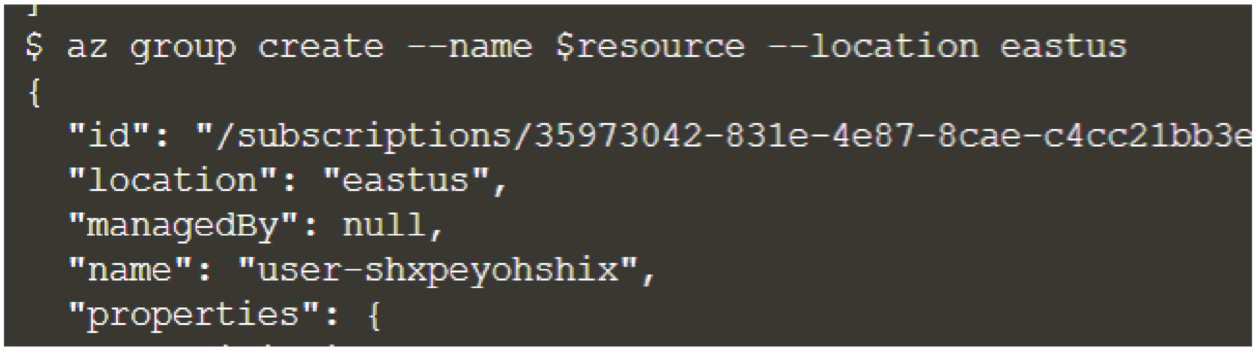
In any connected application (which is pretty much all applications nowadays), we do not want resources to be accessed by just anyone or anything. We need to compartmentalize resources and block certain types of traffic from entering a resource. For example, if we have some web services that are accessible through the internet, we should not allow the backend databases on a separate virtual network to be accessed by the internet as well. We want to prevent these kind of breaches.



In this lab we will create a network security group on a virtual network and configure policies so it blocks certain types of traffic.

First we need a resource group that will be the target of our NSG.

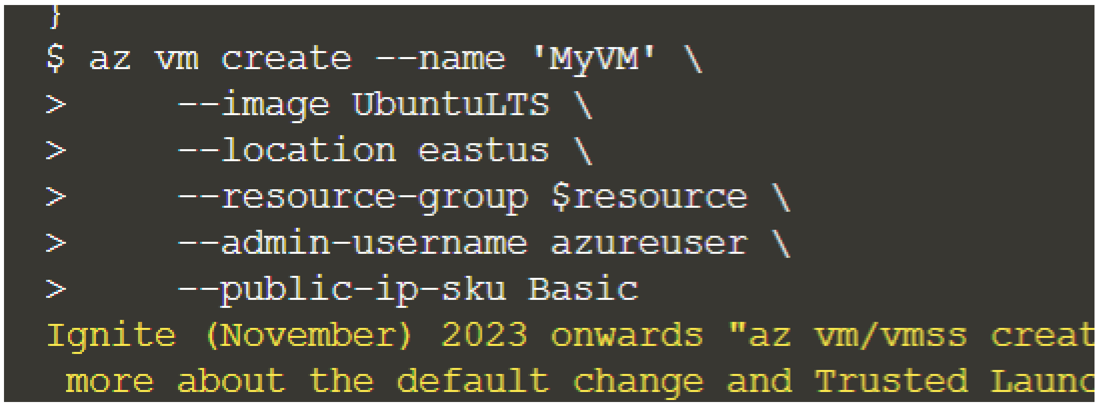
```
az group create --name $resource --location eastus
```



```
}
$ az group create --name $resource --location eastus
{
  "id": "/subscriptions/35973042-831e-4e87-8cae-c4cc21bb3e
  "location": "eastus",
  "managedBy": null,
  "name": "user-shxpeyohshix",
  "properties": {
```

Let's also create a virtual machine in the resource group:

```
az vm create --name 'MyVM' \
   --image UbuntuLTS \
   --location eastus \
   --resource-group $resource \
   --admin-username azureuser \
   --public-ip-sku Basic
```

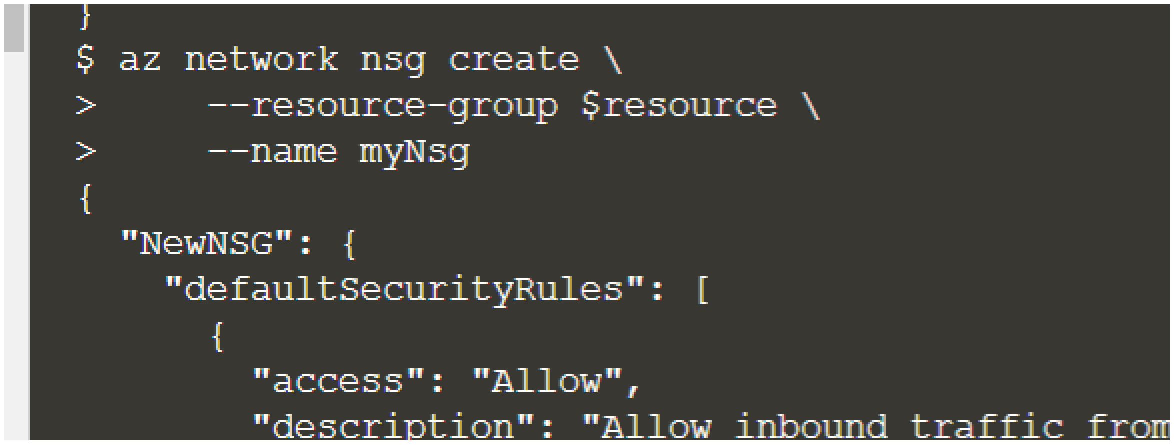


```
}
$ az vm create --name 'MyVM' \
>     --image UbuntuLTS \
>     --location eastus \
>     --resource-group $resource \
>     --admin-username azureuser \
>     --public-ip-sku Basic
Ignite (November) 2023 onwards "az vm/vmss creat
 more about the default change and Trusted Launc
```

Step 3 - Creating an NSG and Rule
Next let's create a network security group on the second resource group $resourceB:

```
az network nsg create \
   --resource-group $resource \
   --name myNsg
```

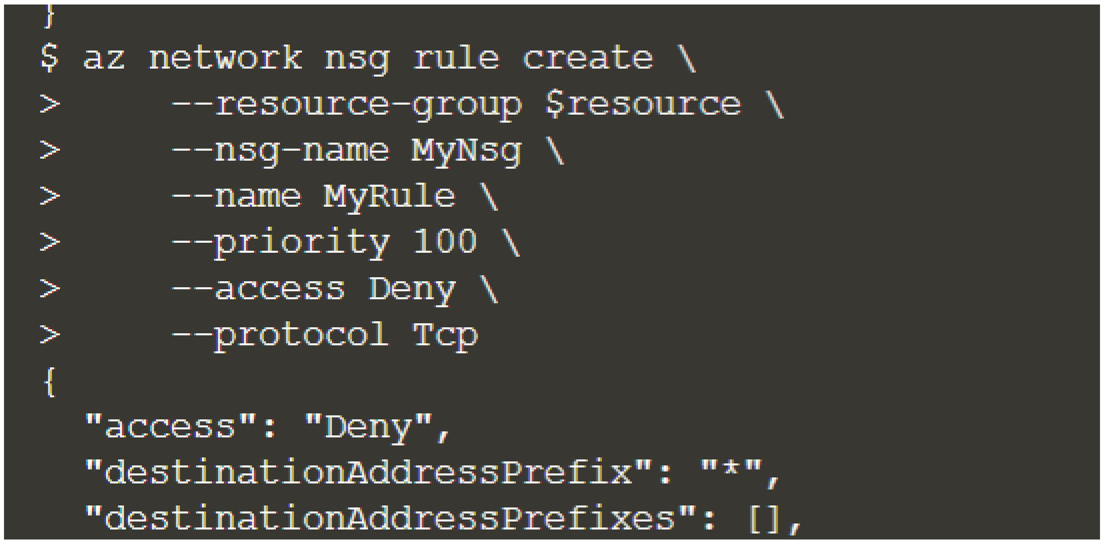
If we wanted to deny all TCP traffic with the highest priority (meaning no other rules will trump it), we could create a rule like this. The highest priority is 100, and 4,096 is the lowest. No two rules can have the same priority. This allows us to create broad policies and specific exceptions that have higher priority. In other words, rules are sortable on criticality to help the system choose one rule over another (e.g., "give nobody access from that department except Joe").



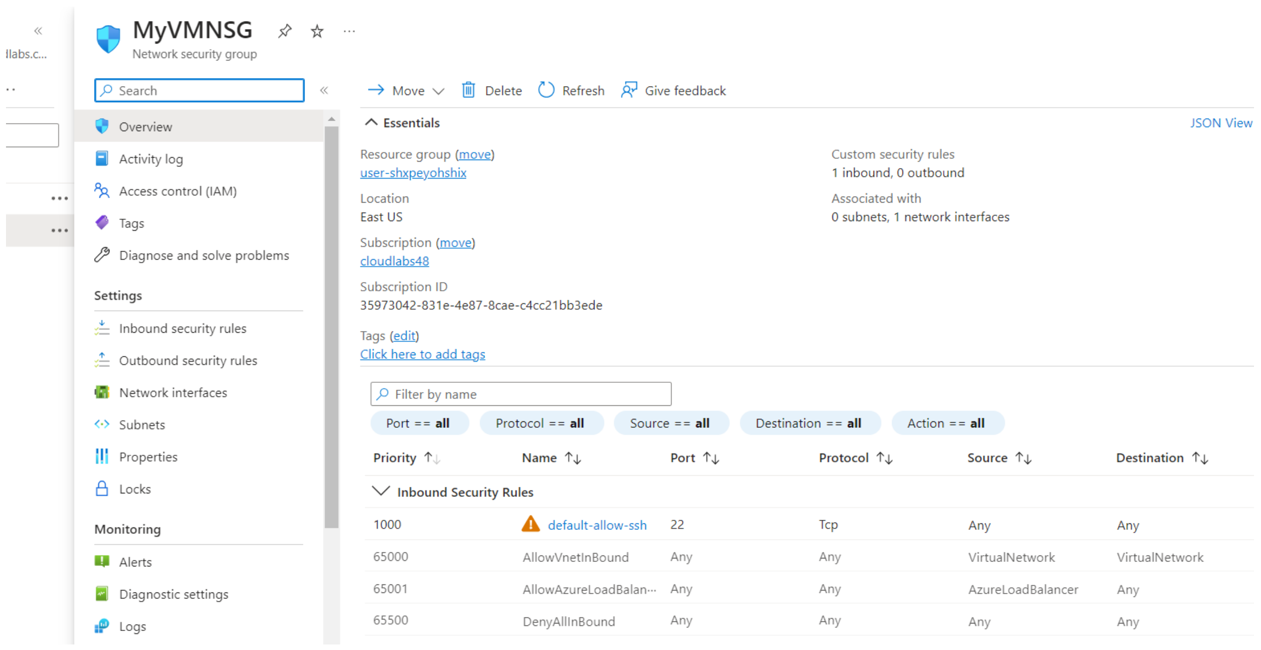
```
}
$ az network nsg create \
>     --resource-group $resource \
>     --name myNsg
{
  "NewNSG": {
    "defaultSecurityRules": [
      {
        "access": "Allow",
        "description": "Allow inbound traffic from
```

Note also that these rules can Deny or Allow access:

```
az network nsg rule create \
    --resource-group $resource \
    --nsg-name MyNsg \
    --name MyRule \
    --priority 100 \
    --access Deny \
    --protocol Tcp
```



```
}
$ az network nsg rule create \
>     --resource-group $resource \
>     --nsg-name MyNsg \
>     --name MyRule \
>     --priority 100 \
>     --access Deny \
>     --protocol Tcp
{
  "access": "Deny",
  "destinationAddressPrefix": "*",
  "destinationAddressPrefixes": [],
```



Resource group (move)
user-shxpeyohshix

Location
East US

Subscription (move)
cloudlabs48

Subscription ID
35973042-831e-4e87-8cae-c4cc21bb3ede

Tags (edit)
Click here to add tags

Custom security rules
1 inbound, 0 outbound

Associated with
0 subnets, 1 network interfaces

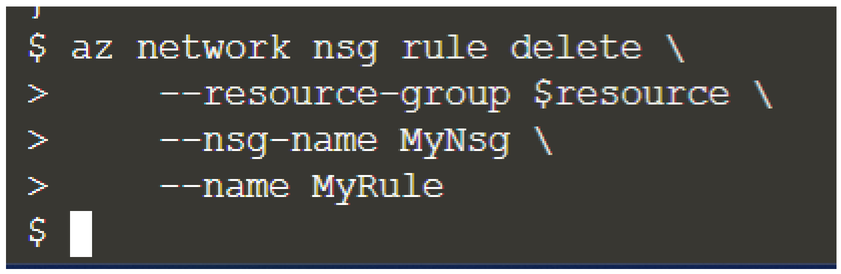
Filter by name

Port == all | Protocol == all | Source == all | Destination == all | Action == all

Priority ↑↓	Name ↑↓	Port ↑↓	Protocol ↑↓	Source ↑↓	Destination ↑↓
Inbound Security Rules					
1000	⚠ default-allow-ssh	22	Tcp	Any	Any
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork
65001	AllowAzureLoadBalan···	Any	Any	AzureLoadBalancer	Any
65500	DenyAllInBound	Any	Any	Any	Any

To remove the rule, use the delete command:

```
az network nsg rule delete \
    --resource-group $resource \
    --nsg-name MyNsg \
    --name MyRule
```



```
$ az network nsg rule delete \
>     --resource-group $resource \
>     --nsg-name MyNsg \
>     --name MyRule
$ 
```

Step 4 - Creating Specific Rules

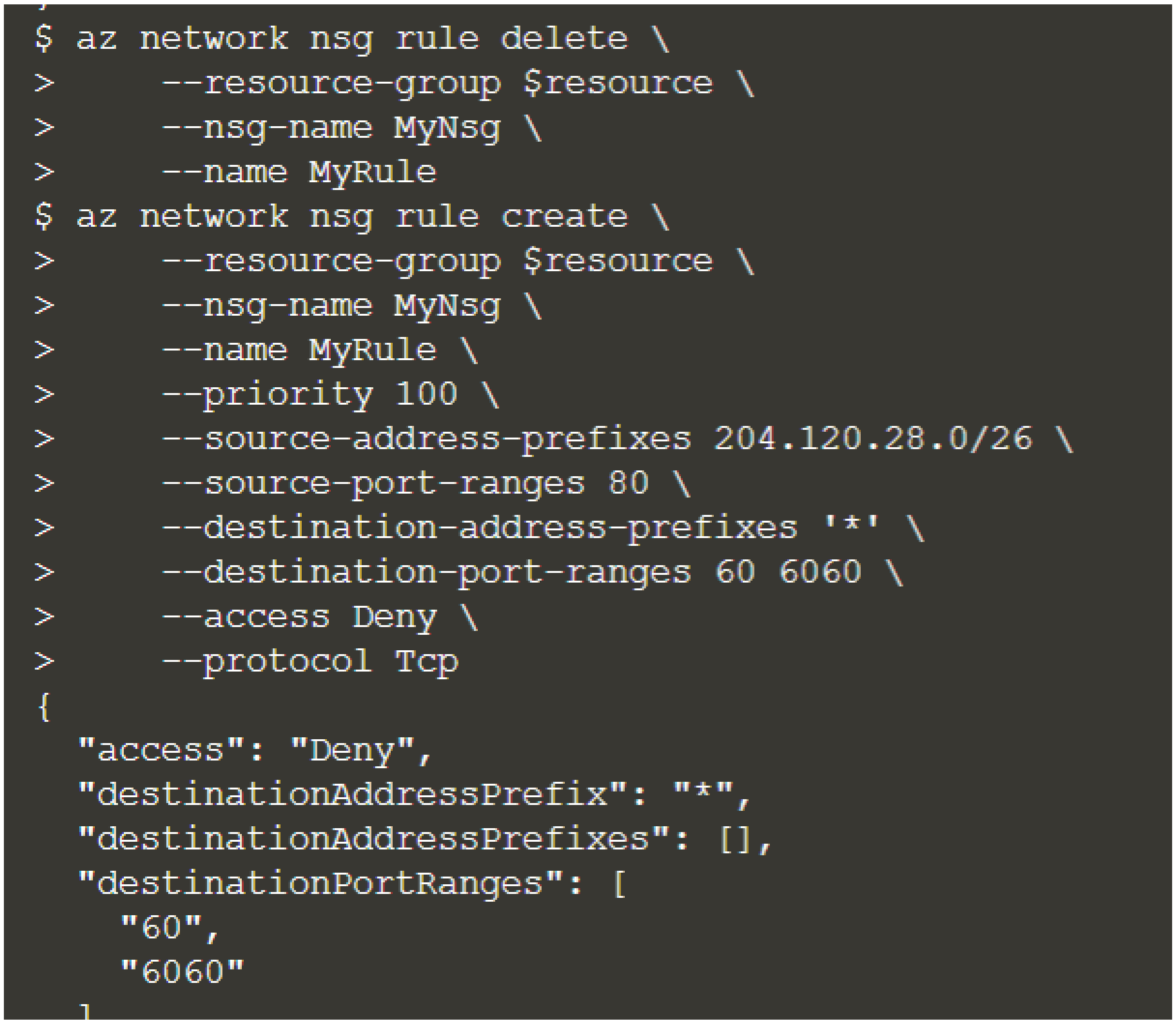
There are many options to configure and filter traffic in the NSG call. Here are all the arguments that can be provided to the rule create command:

```
az network nsg rule create --name
                           --nsg-name
                           --priority
                           --resource-group
                           [--access {Allow, Deny}]
                           [--description]
                           [--destination-address-prefixes]
                           [--destination-asgs]
                           [--destination-port-ranges]
```

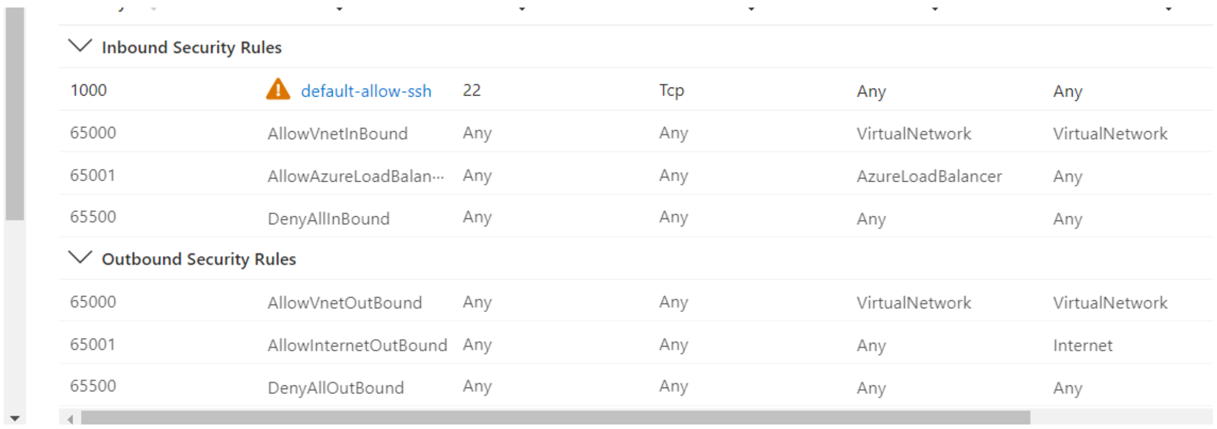
```
                              [--direction {Inbound, Outbound}]
                              [--protocol {*, Ah, Esp, Icmp, Tcp, Udp}]
                              [--source-address-prefixes]
                              [--source-asgs]
                              [--source-port-ranges]
                              [--subscription]
```

Here we deny TCP traffic of a specific range of IP addresses with the highest priority of 100:

```
az network nsg rule create \
    --resource-group $resource \
    --nsg-name MyNsg \
    --name MyRule \
    --priority 100 \
    --source-address-prefixes 204.120.28.0/26 \
    --source-port-ranges 80 \
    --destination-address-prefixes '*' \
    --destination-port-ranges 60 6060 \
    --access Deny \
    --protocol Tcp
```



```
$ az network nsg rule delete \
>     --resource-group $resource \
>     --nsg-name MyNsg \
>     --name MyRule
$ az network nsg rule create \
>     --resource-group $resource \
>     --nsg-name MyNsg \
>     --name MyRule \
>     --priority 100 \
>     --source-address-prefixes 204.120.28.0/26 \
>     --source-port-ranges 80 \
>     --destination-address-prefixes '*' \
>     --destination-port-ranges 60 6060 \
>     --access Deny \
>     --protocol Tcp
{
  "access": "Deny",
  "destinationAddressPrefix": "*",
  "destinationAddressPrefixes": [],
  "destinationPortRanges": [
    "60",
    "6060"
```



Inbound Security Rules					
1000	default-allow-ssh	22	Tcp	Any	Any
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork
65001	AllowAzureLoadBalan···	Any	Any	AzureLoadBalancer	Any
65500	DenyAllInBound	Any	Any	Any	Any
Outbound Security Rules					
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork
65001	AllowInternetOutBound	Any	Any	Any	Internet
65500	DenyAllOutBound	Any	Any	Any	Any

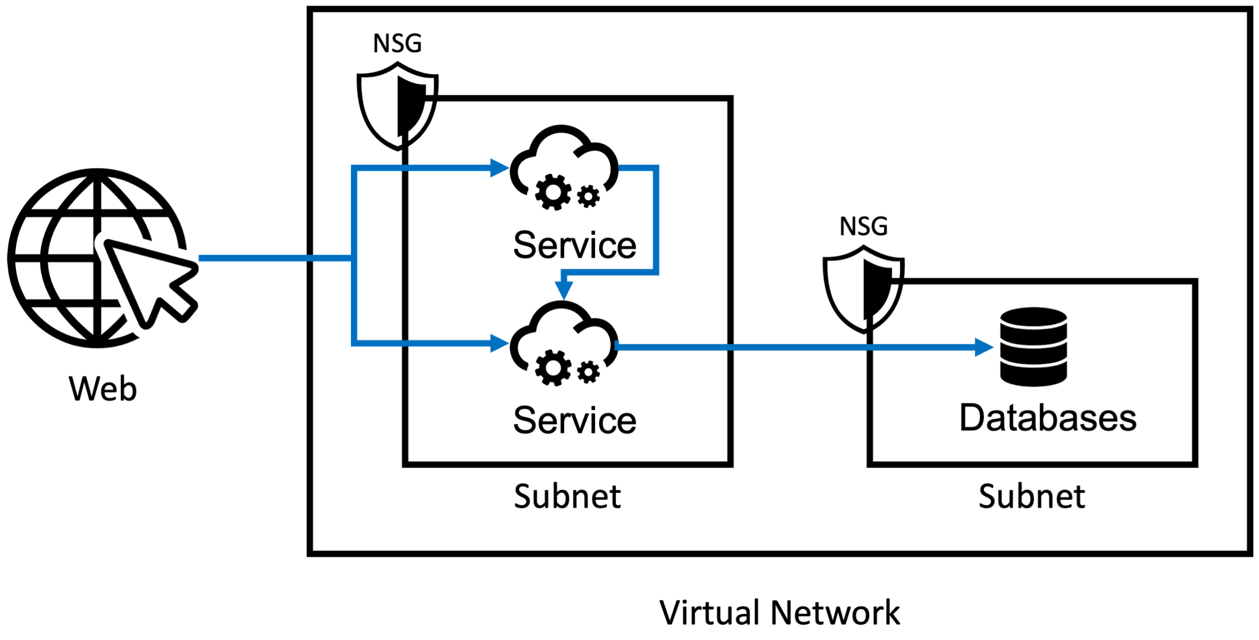
In this lab you will get an overview of application security groups (ASG) and learn how to do the following:

Log in to the Azure CLI
Create an ASG

Step 2 - Getting Set Up

In this lab we are going to create an *application security group* (ASG), which provides a convenient abstraction to not maintain IP addresses in a network security group.

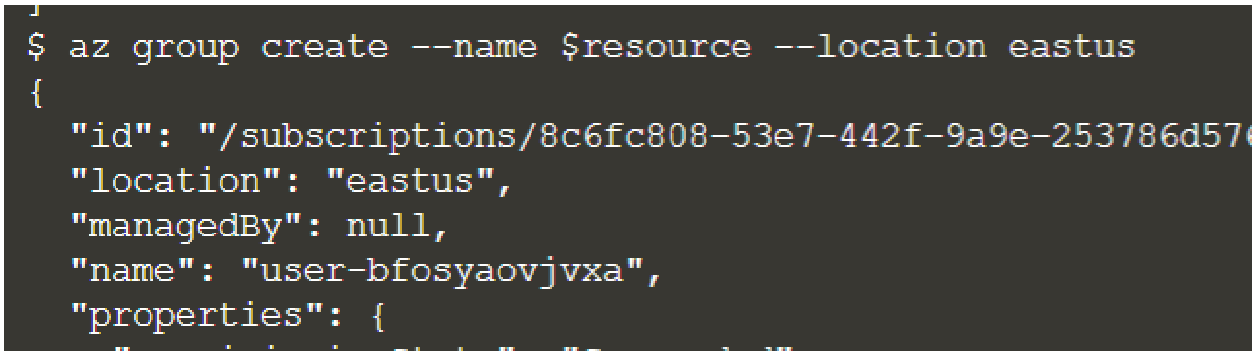
Let's say we want to enforce this network structure:



Maintaining these rules in an NSG can be tedious. That's why we are going to create an ASG to streamline the connecting of networks and enforcing the desired rules.

We are going to create an NSG on a virtual network, configure the ASGs around resources, and create two VMs to test the traffic filter.

First we need a resource group.



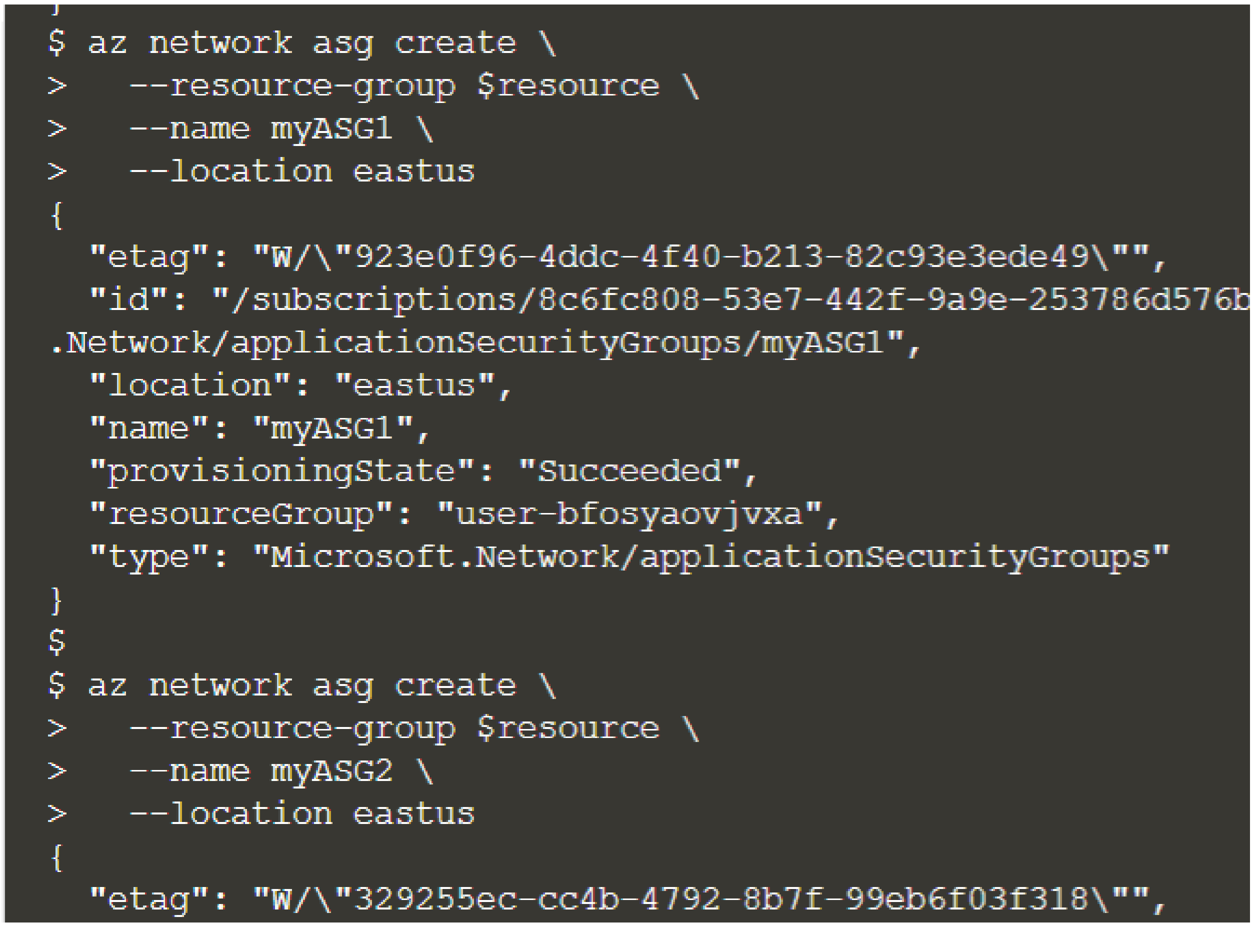
```
$ az group create --name $resource --location eastus
{
  "id": "/subscriptions/8c6fc808-53e7-442f-9a9e-253786d57
  "location": "eastus",
  "managedBy": null,
  "name": "user-bfosyaovjvxa",
  "properties": {
```

Step 3 - Creating the ASGs
Next let's create two ASGs using az network asg create. This will enable you to group resources with similar traffic filter rules:

```
az network asg create \
  --resource-group $resource \
  --name myASG1 \
  --location eastus

az network asg create \
  --resource-group $resource \
  --name myASG2 \
  --location eastus
```

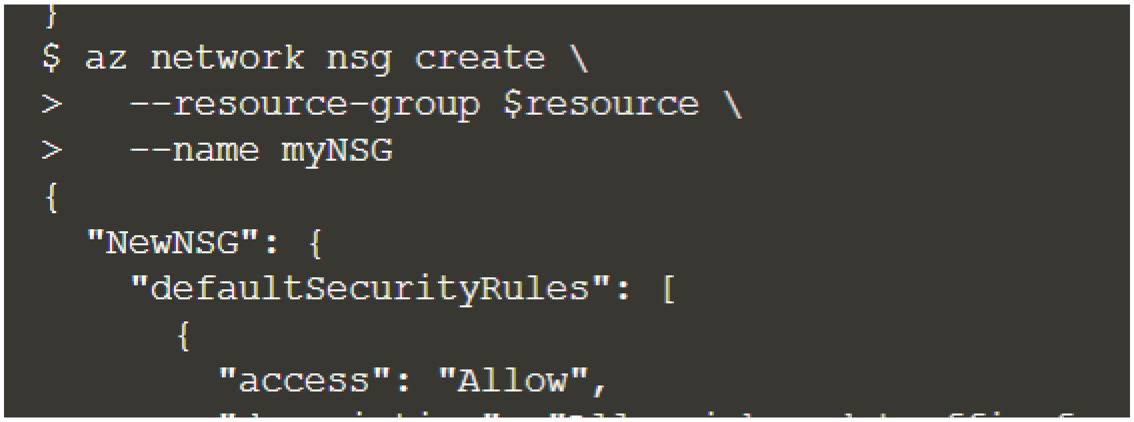
Next we need to create an NSG:



```
$ az network asg create \
>   --resource-group $resource \
>   --name myASG1 \
>   --location eastus
{
  "etag": "W/\"923e0f96-4ddc-4f40-b213-82c93e3ede49\"",
  "id": "/subscriptions/8c6fc808-53e7-442f-9a9e-253786d576b
.Network/applicationSecurityGroups/myASG1",
  "location": "eastus",
  "name": "myASG1",
  "provisioningState": "Succeeded",
  "resourceGroup": "user-bfosyaovjvxa",
  "type": "Microsoft.Network/applicationSecurityGroups"
}
$
$ az network asg create \
>   --resource-group $resource \
>   --name myASG2 \
>   --location eastus
{
  "etag": "W/\"329255ec-cc4b-4792-8b7f-99eb6f03f318\"",
```

```
az network nsg create \
  --resource-group $resource \
  --name myNSG
```

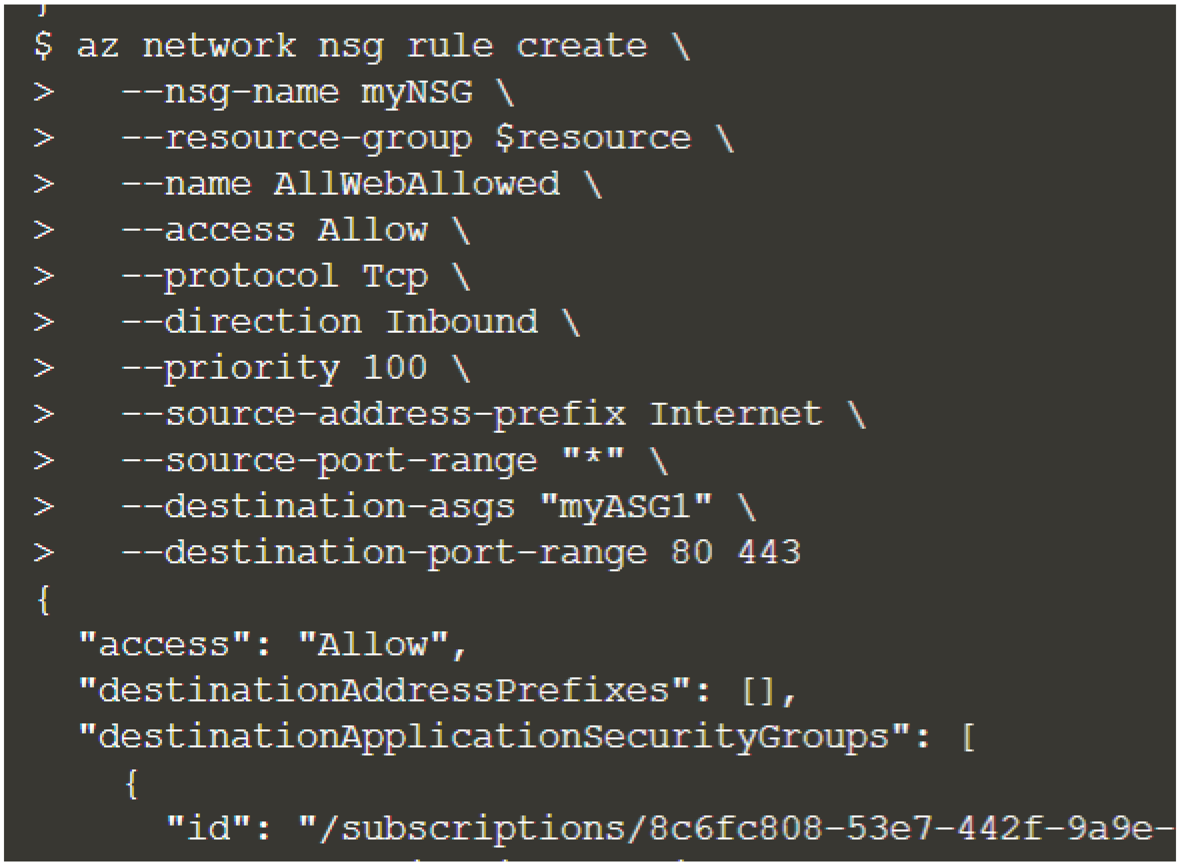
Now we must create two new rules in our NSG with the command az network nsg rule create. We will allow inbound internet traffic from myASG1 with ports 80 through 443. This will allow web traffic:



```
$ az network nsg create \
>   --resource-group $resource \
>   --name myNSG
{
  "NewNSG": {
    "defaultSecurityRules": [
      {
        "access": "Allow",
```

```
az network nsg rule create \
  --nsg-name myNSG \
  --resource-group $resource \
  --name AllWebAllowed \
  --access Allow \
  --protocol Tcp \
  --direction Inbound \
  --priority 100 \
  --source-address-prefix Internet \
  --source-port-range "*" \
  --destination-asgs "myASG1" \
  --destination-port-range 80 443
```

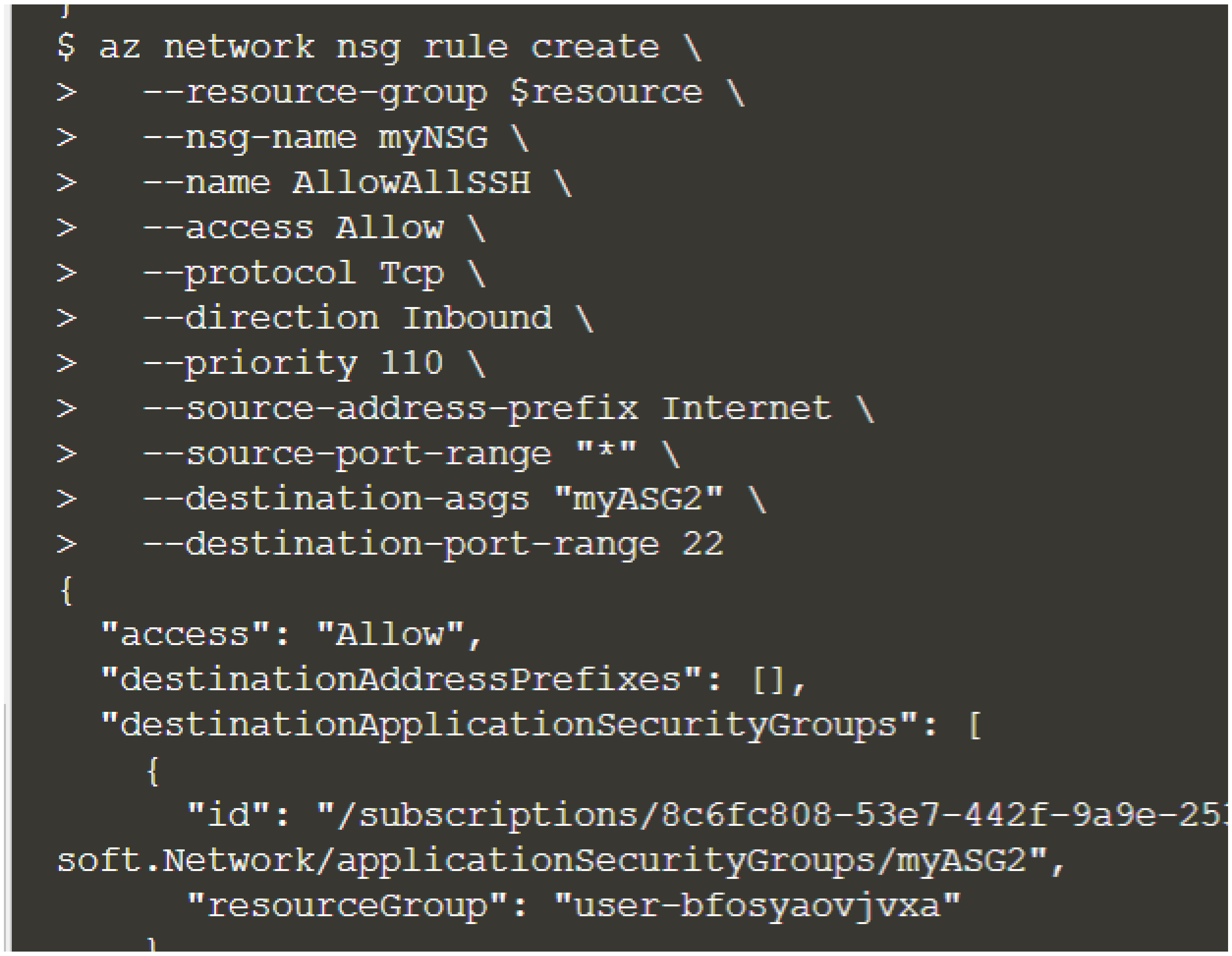
Next we will create another rule to allow SSH traffic to myASG2. We will give it slightly less priority than the previous rule (so they don't clash on shared ranking) and limit to port 22:



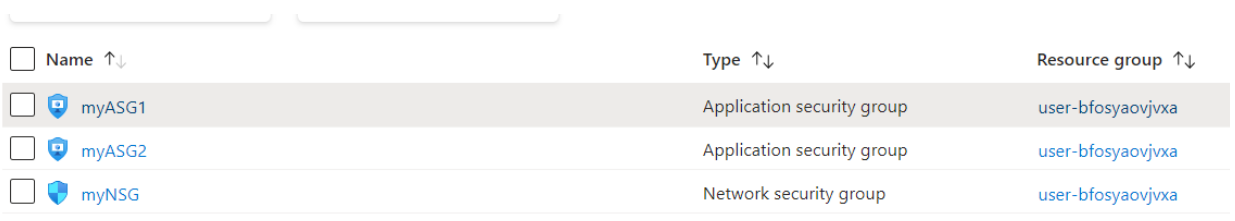
```
$ az network nsg rule create \
>   --nsg-name myNSG \
>   --resource-group $resource \
>   --name AllWebAllowed \
>   --access Allow \
>   --protocol Tcp \
>   --direction Inbound \
>   --priority 100 \
>   --source-address-prefix Internet \
>   --source-port-range "*" \
>   --destination-asgs "myASG1" \
>   --destination-port-range 80 443
{
  "access": "Allow",
  "destinationAddressPrefixes": [],
  "destinationApplicationSecurityGroups": [
    {
      "id": "/subscriptions/8c6fc808-53e7-442f-9a9e-
```

```
az network nsg rule create \
  --resource-group $resource \
  --nsg-name myNSG \
  --name AllowAllSSH \
  --access Allow \
  --protocol Tcp \
  --direction Inbound \
  --priority 110 \
  --source-address-prefix Internet \
  --source-port-range "*" \
  --destination-asgs "myASG2" \
  --destination-port-range 22
```

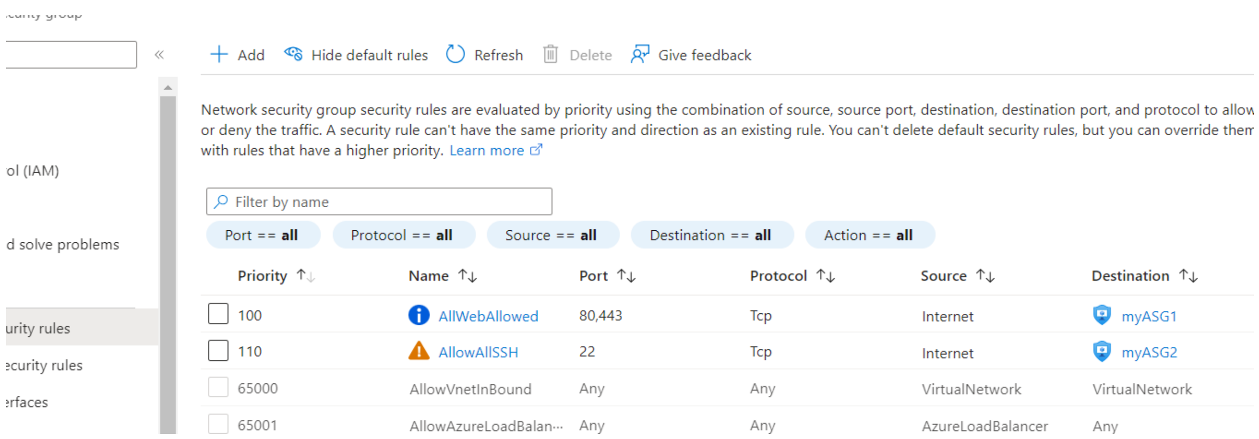
Note that in production, we would typically use a private connection or VPN rather than exposing port 22 to the internet like this.



```
}
$ az network nsg rule create \
>   --resource-group $resource \
>   --nsg-name myNSG \
>   --name AllowAllSSH \
>   --access Allow \
>   --protocol Tcp \
>   --direction Inbound \
>   --priority 110 \
>   --source-address-prefix Internet \
>   --source-port-range "*" \
>   --destination-asgs "myASG2" \
>   --destination-port-range 22
{
  "access": "Allow",
  "destinationAddressPrefixes": [],
  "destinationApplicationSecurityGroups": [
    {
      "id": "/subscriptions/8c6fc808-53e7-442f-9a9e-25
soft.Network/applicationSecurityGroups/myASG2",
      "resourceGroup": "user-bfosyaovjvxa"
    }
```



Name	Type	Resource group
myASG1	Application security group	user-bfosyaovjvxa
myASG2	Application security group	user-bfosyaovjvxa
myNSG	Network security group	user-bfosyaovjvxa



Network security group security rules are evaluated by priority using the combination of source, source port, destination, destination port, and protocol to allow or deny the traffic. A security rule can't have the same priority and direction as an existing rule. You can't delete default security rules, but you can override them with rules that have a higher priority. Learn more

Filter by name

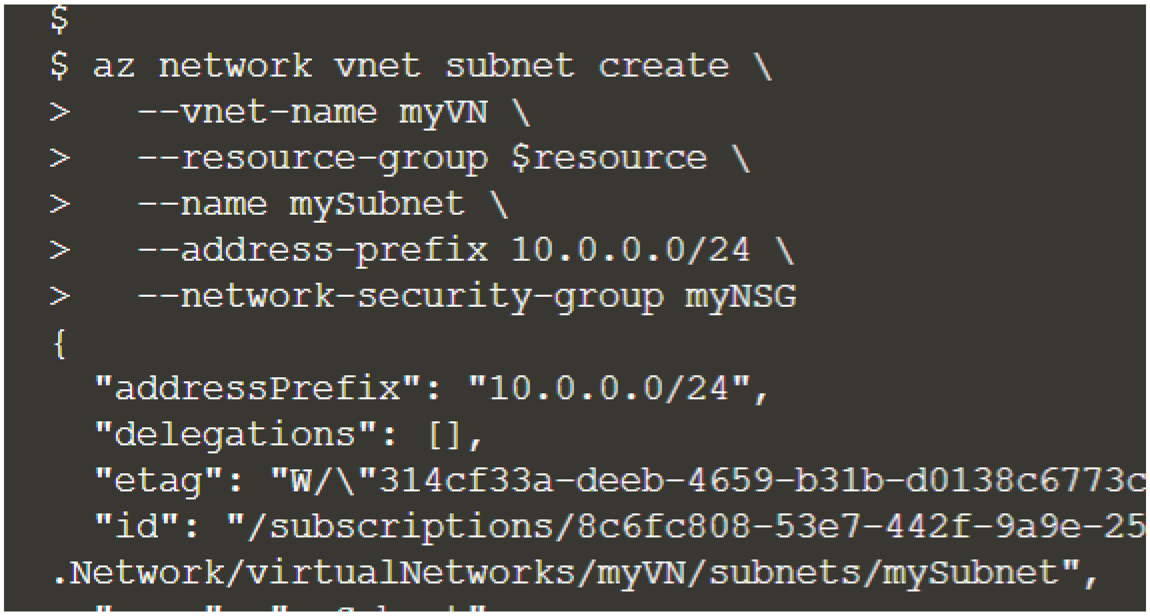
Port == all | Protocol == all | Source == all | Destination == all | Action == all

Priority	Name	Port	Protocol	Source	Destination
100	AllWebAllowed	80,443	Tcp	Internet	myASG1
110	AllowAllSSH	22	Tcp	Internet	myASG2
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork
65001	AllowAzureLoadBalan…	Any	Any	AzureLoadBalancer	Any

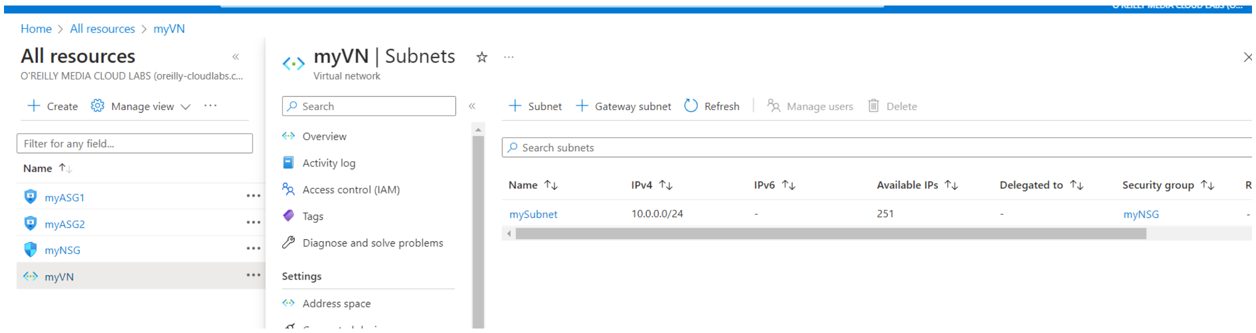
Step 4 - Creating the VMs
Let's create a virtual network and a subnet inside it. Note for the az network vnet subnet create command, we are tying it to the network security group we just created:

```
az network vnet create \
  --name myVN \
  --resource-group $resource \
  --address-prefixes 10.0.0.0/16

az network vnet subnet create \
  --vnet-name myVN \
  --resource-group $resource \
  --name mySubnet \
  --address-prefix 10.0.0.0/24 \
  --network-security-group myNSG
```



```
$
$ az network vnet subnet create \
>   --vnet-name myVN \
>   --resource-group $resource \
>   --name mySubnet \
>   --address-prefix 10.0.0.0/24 \
>   --network-security-group myNSG
{
  "addressPrefix": "10.0.0.0/24",
  "delegations": [],
  "etag": "W/\"314cf33a-deeb-4659-b31b-d0138c6773c
  "id": "/subscriptions/8c6fc808-53e7-442f-9a9e-25
.Network/virtualNetworks/myVN/subnets/mySubnet",
```



Home > All resources > myVN

All resources
O'REILLY MEDIA CLOUD LABS (oreilly-cloudlabs.c...

Name
myASG1
myASG2
myNSG
myVN

myVN | Subnets
Virtual network

Name	IPv4	IPv6	Available IPs	Delegated to	Security group
mySubnet	10.0.0.0/24	-	251	-	myNSG

Let's create a virtual machine to act as our web server. We will put it on the subnet and force the nsg argument to be blank so a default NSG is not applied. We are going to have the admin-password be Password1234. Of course, your security in real life would use a better (more concealed) password as well as SSH and keys:

```
az vm create \
  --resource-group $resource \
  --name myVM1 \
  --image UbuntuLTS \
  --vnet-name myVN \
  --subnet mySubnet \
  --nsg "" \
  --asgs myASG1 \
  --admin-username azureuser \
  --admin-password Password1234
```
Finally, let's create the backend VM that will act as the backend server:

```
az vm create \
  --resource-group $resource \
  --name myVM2 \
  --image UbuntuLTS \
  --vnet-name myVN \
  --subnet mySubnet \
  --nsg "" \
  --asgs myASG2 \
  --admin-username azureuser \
  --admin-password Password1234
```
After that runs, extract the public IP address of the virtual machine and save it to a variable:

```
myIp=$(az vm show -d --resource-group $resource --name myVM2 --query publicIps -o tsv)
```

Step 5 - Testing and Cleanup
Let's test this out. Log in to the second virtual machine acting as the backend:

```
ssh azureuser@$myIp
```
Notice we get in because we allow connecting through port 22. From that machine, we can now bounce to the other machine acting as the frontend. But here we can do it by addressing as myVM1 rather than an IP address:

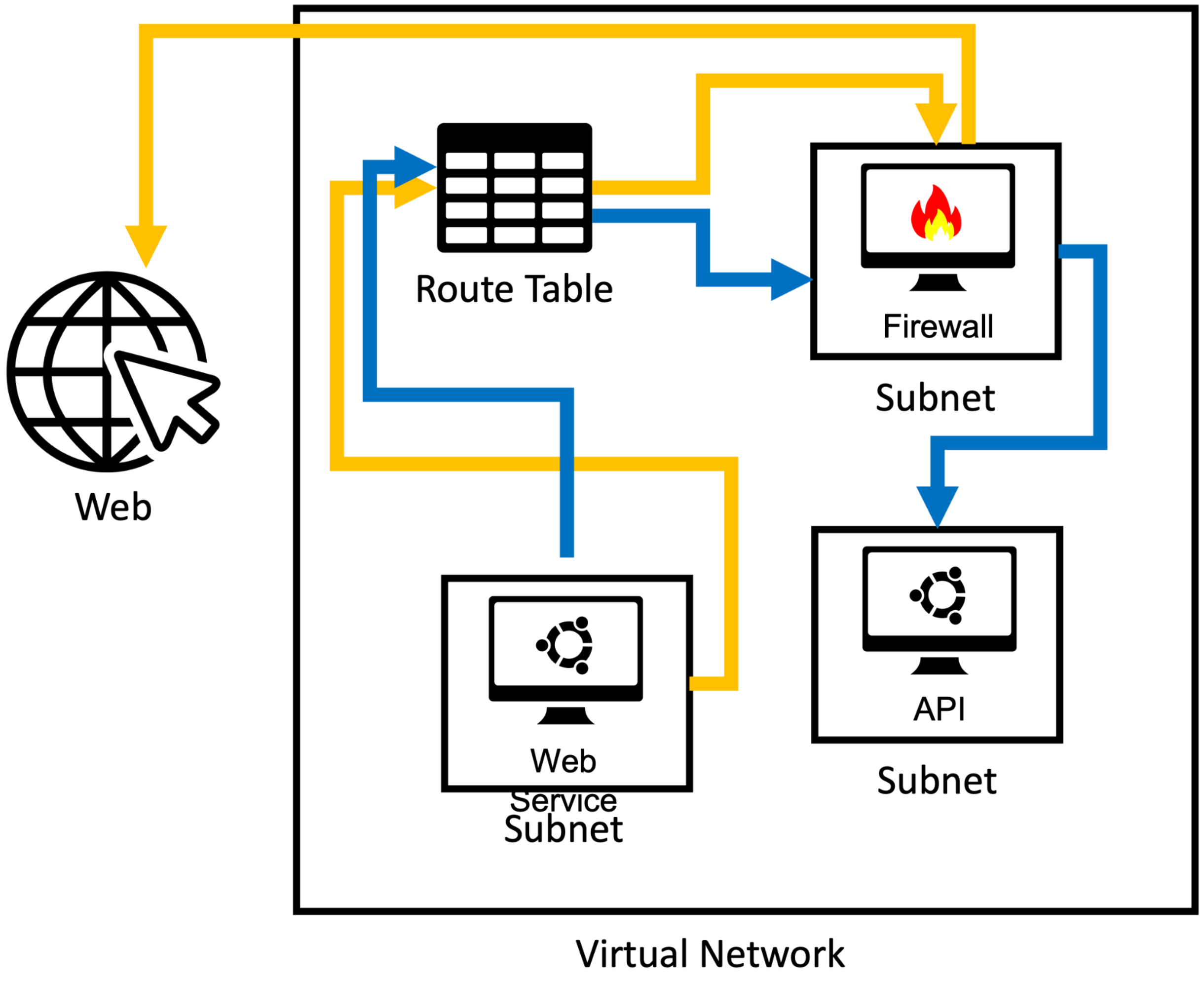
```
ssh azureuser@myVM1
```
We would not have been able to log in to myVM1 through the internet because it was restricted in our rules. But because we were inside the same virtual network with myVM2, we were able to log in to that machine. This effectively shows our ASGs are working! You will also find that if you go back to the frontend myVM1, you can access outbound traffic to the internet because our rule allows that.

Log in to the Azure CLI
Create a firewall
Custom route a frontend
Custom route a backend

Step 2 - Getting Set Up
When you connect resources, such as making one virtual machine ping another, or pinging an external website, Azure will automatically manage the routings for you. For example, if a VM running a web

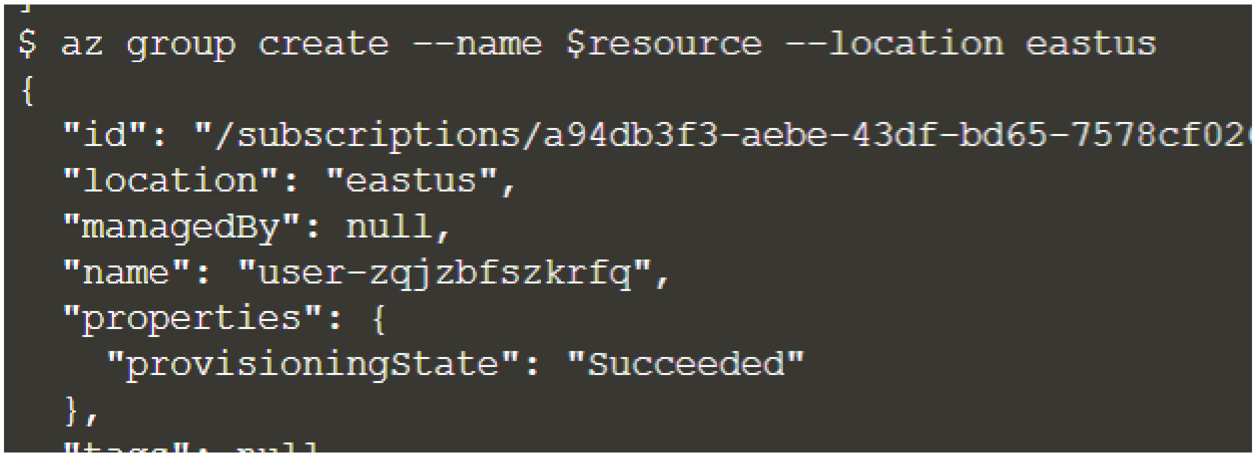
service requests resources from the web, Azure will automatically create and manage that connection for you. If another VM holding an API connects to the web service VM, Azure will create and manage that connection too.

But there may be times when you want to route traffic differently, such as having a specific appliance (e.g., a "firewall") intercept all incoming traffic from one resource to another. We could also control the outbound routing to the internet to go through the firewall too.



We use route tables to redirect traffic and to forward it through the appropriate resources. Azure generates its own route tables by default, but we will learn how to customize our user-defined routes (UDRs) to redirect traffic and associate them with subnets. Specifically, we will run our traffic through a virtual network appliance (firewall).

First we need a resource group.



```
$ az group create --name $resource --location eastus
{
  "id": "/subscriptions/a94db3f3-aebe-43df-bd65-7578cf02
  "location": "eastus",
  "managedBy": null,
  "name": "user-zqjzbfszkrfq",
  "properties": {
    "provisioningState": "Succeeded"
  },
```

Next let's create our virtual network:

```
az network vnet create \
    --resource-group $resource \
    --name myVnet \
    --address-prefix 10.0.0.0/16  \
    --location eastus \
    --subnet-name myFrontendSubnet \
    --subnet-prefix 10.0.1.0/24
```

Then let's create our network security group (NSG):

```
az network nsg create \
    --resource-group $resource \
    --name myNSG \
    --location eastus
```

Let's then allow HTTP and HTTPS traffic:

```
az network nsg rule create \
    --resource-group $resource \
    --nsg-name myNSG \
    --name AllowHTTP \
    --access Allow \
    --protocol Tcp \
    --direction Inbound \
    --priority 100 \
    --source-address-prefix Internet \
    --source-port-range "*" \
    --destination-address-prefix "*" \
    --destination-port-range 80
az network nsg rule create \
    --resource-group $resource \
    --nsg-name myNSG \
    --name allowHTTPS \
    --access Allow \
    --protocol Tcp \
    --direction Inbound \
    --priority 200 \
    --source-address-prefix Internet \
    --source-port-range "*" \
    --destination-address-prefix "*" \
    --destination-port-range 443
```

Step 3 - Creating the Firewall

Let's associate our NSG with our frontend subnet:

```
az network vnet subnet update \
    --vnet-name myVnet \
    --name myFrontendSubnet \
    --resource-group $resource \
    --network-security-group myNSG
```

Now let's create a subnet for the backend:

```
az network vnet subnet create \
    --address-prefix 10.0.2.0/24 \
    --name myBackendSubnet \
    --resource-group $resource \
    --vnet-name myVnet
```

And then let's create the DMZ subnet, or the subnet that separates the trusted network from the untrusted network that is the internet:

```
az network vnet subnet create \
    --address-prefix 10.0.0.0/24 \
    --name myDMZSubnet \
    --resource-group $resource \
    --vnet-name myVnet
```

Next, create the public IP for the firewall virtual machine:

```
az network public-ip create \
    --resource-group $resource \
    --name myPublicFirewallIp
```

And then the network interface that will do the IP forwarding:

```
az network nic create \
    --resource-group $resource \
    --name myFirewallNic \
    --vnet-name myVnet \
    --subnet myDmzSubnet \
    --public-ip-address myPublicFirewallIp \
    --ip-forwarding
```

Finally, we'll create the firewall appliance itself, which will be a VM running Ubuntu. Note that we are wiring it up with the network interface for the firewall we just created:

```
az vm create \
    --resource-group $resource \
    --name myFirewallVM \
    --nics myFirewallNic \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys \
    --public-ip-sku BASIC
```

Step 4 - Routing the Frontend

Get the private IP address from the firewall VM we just created and save it to a variable:

```
firewallVMIp=$(az vm list-ip-addresses \
    --resource-group $resource \
    --name myFirewallVM \
    --query '[].virtualMachine.network.privateIpAddresses[0]' --out tsv)

echo $firewallVMIp
```

Now let's start creating the routings. First, create the routing table for the frontend subnet:

```
az network route-table create \
    --name routeTableFESubnet \
    --resource-group $resource
```

Then we can add a routing from the frontend to the backend via the firewall appliance:

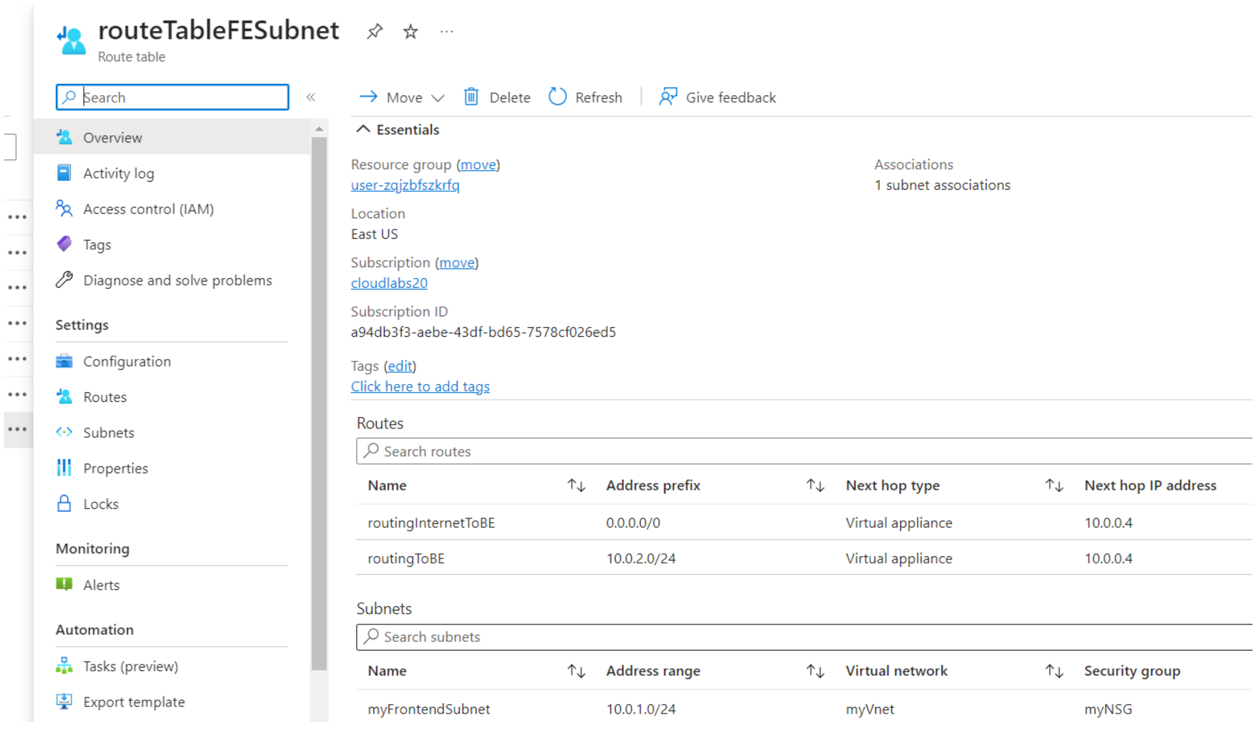
```
az network route-table route create \
    --name routingToBE  \
    --resource-group $resource \
    --route-table-name routeTableFESubnet \
    --address-prefix 10.0.2.0/24 \
    --next-hop-type VirtualAppliance \
    --next-hop-ip-address $firewallVMIp
```

Passing through the firewall VM, route traffic from the frontend subnet to the internet. Note the hop-related arguments that will redirect requests to the firewall VM:

```
az network route-table route create \
    --name routingInternetToBE \
    --resource-group $resource \
    --route-table-name routeTableFESubnet \
    --address-prefix 0.0.0.0/0 \
    --next-hop-type VirtualAppliance \
    --next-hop-ip-address $firewallVMIp
```

After we create and populate the route table, let's associate it with the frontend subnet:

```
az network vnet subnet update \
    --name myFrontendSubnet \
    --vnet-name myVnet \
    --resource-group $resource \
    --route-table routeTableFESubnet
```



routeTableFESubnet
Route table

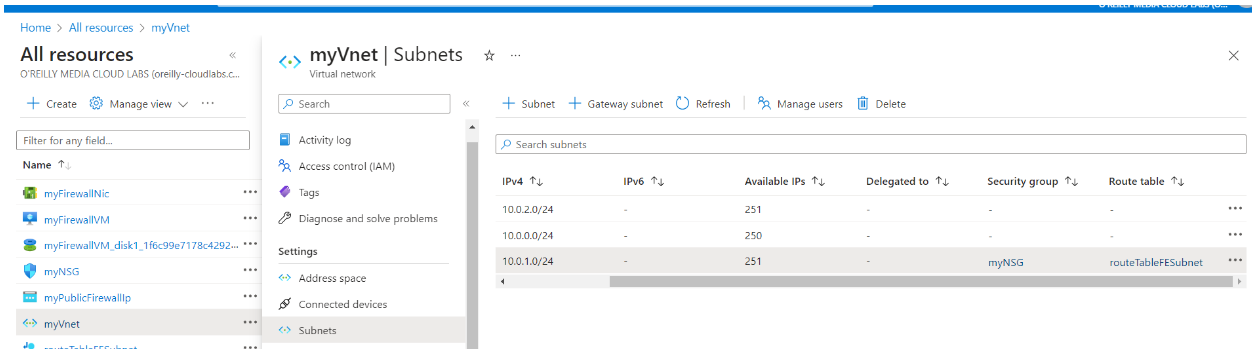
Resource group (move): user-zqjzbfszkrfq
Location: East US
Subscription (move): cloudlabs20
Subscription ID: a94db3f3-aebe-43df-bd65-7578cf026ed5
Tags (edit): Click here to add tags
Associations: 1 subnet associations

Routes

Name	Address prefix	Next hop type	Next hop IP address
routingInternetToBE	0.0.0.0/0	Virtual appliance	10.0.0.4
routingToBE	10.0.2.0/24	Virtual appliance	10.0.0.4

Subnets

Name	Address range	Virtual network	Security group
myFrontendSubnet	10.0.1.0/24	myVnet	myNSG



Home > All resources > myVnet

All resources: myFirewallNic, myFirewallVM, myFirewallVM_disk1_1f6c99e7178c4292..., myNSG, myPublicFirewallIp, myVnet

myVnet | Subnets
Virtual network

IPv4	IPv6	Available IPs	Delegated to	Security group	Route table
10.0.2.0/24	-	251	-	-	-
10.0.0.0/24	-	250	-	-	-
10.0.1.0/24	-	251	-	myNSG	routeTableFESubnet

Step 5 - Routing the Backend
Next, we wire up the backend table and subnet, just like we did for the frontend. First let's create the route table for the backend:

```
az network route-table create \
    --name routeTableBESubnet \
    --resource-group $resource
```

We will then route traffic from the backend subnet to the frontend subnet through the firewall device. Again note the hop-related arguments that will bounce to the firewall VM:

```
az network route-table route create \
    --name routingToFE \
    --resource-group $resource \
    --route-table-name routeTableBESubnet \
    --address-prefix 10.0.2.0/24 \
    --next-hop-type VirtualAppliance \
    --next-hop-ip-address $firewallVMIp
```

Last routing! Again passing through the firewall, create a route for traffic from the backend subnet to the internet:

```
az network route-table route create \
    --name routeToInternetBE \
    --resource-group $resource \
    --route-table-name routeTableBESubnet \
    --address-prefix 0.0.0.0/0 \
    --next-hop-type VirtualAppliance \
    --next-hop-ip-address $firewallVMIp
```

Lastly, associate the route table to the subnet of the backend:

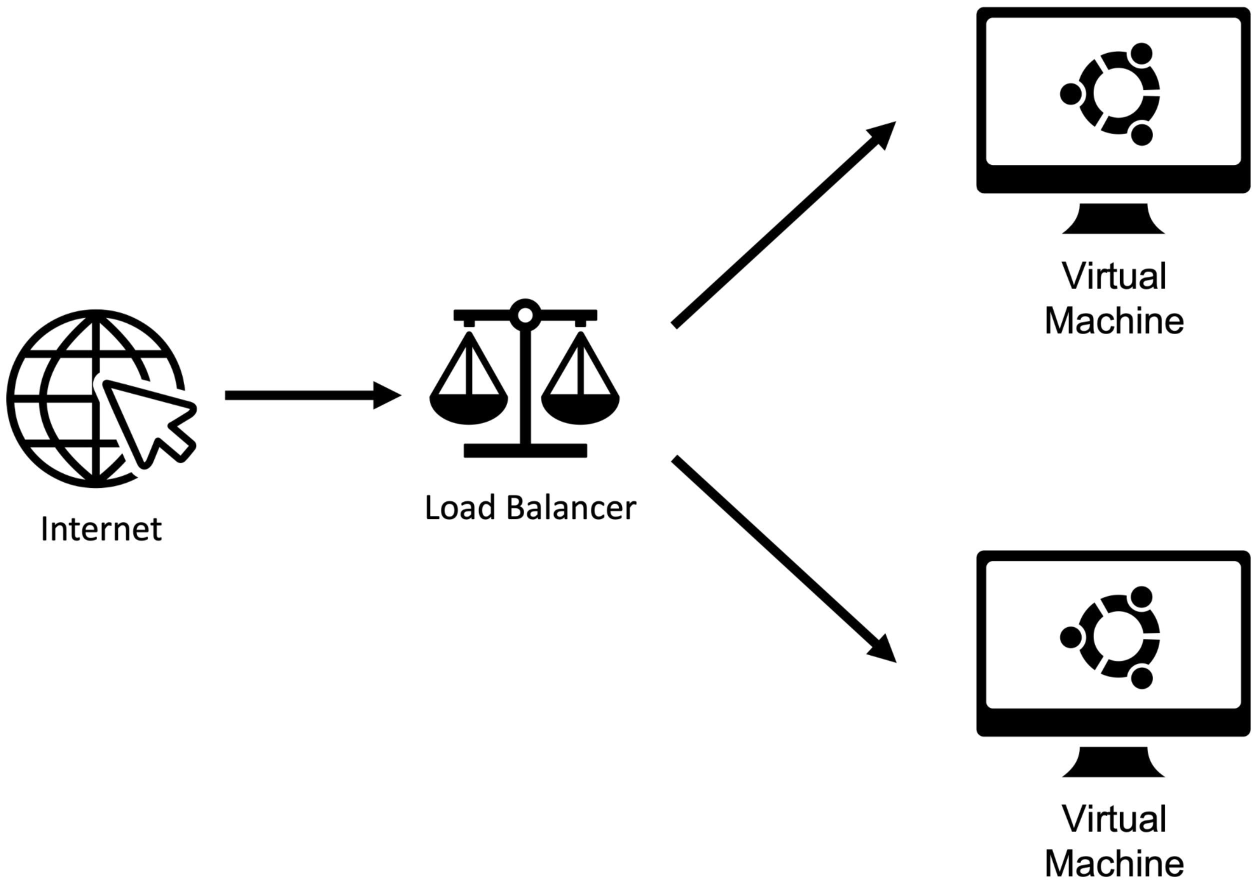
```
az network vnet subnet update \
    --name myBackendSubnet \
    --vnet-name myVnet \
    --resource-group $resource \
    --route-table routeTableBESubnet
```

Now we have configured all routings to put traffic through our virtual network appliance (firewall). This way, we configure how traffic is routed and controlled rather than letting Azure automatically manage the routings. There is nothing wrong with Azure automatically managing routings, unless you have a custom traffic filtering policy you want to impose.

Log in to the Azure CLI
Create a virtual network
Create a bastion service and security group
Configure VMs and load balancer

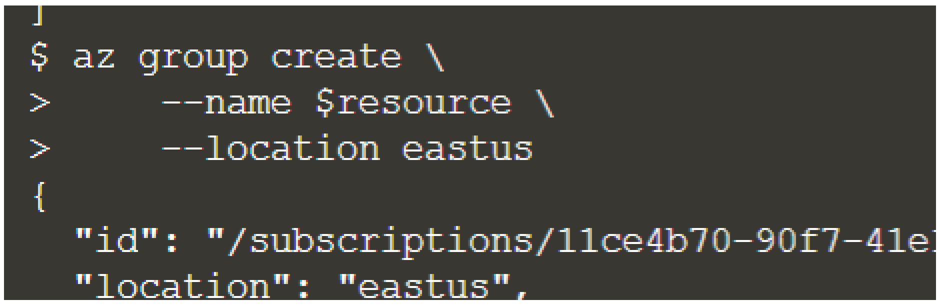
Step 2 - Create Virtual Network
Load balancing means evenly distributing incoming traffic across across a group of backend resources to process those requests. This helps scale traffic loads and process them effectively. In this lab we will build a public load balancer, which provides outbound connections access to a balanced set of virtual machines. Private IP addresses are converted to public IP addresses and effectively handle internet traffic.

We are going to create a load balancer between two virtual machines as depicted here:



Let's first create a resource group called $resource:

```
az group create \
    --name $resource \
    --location eastus
```

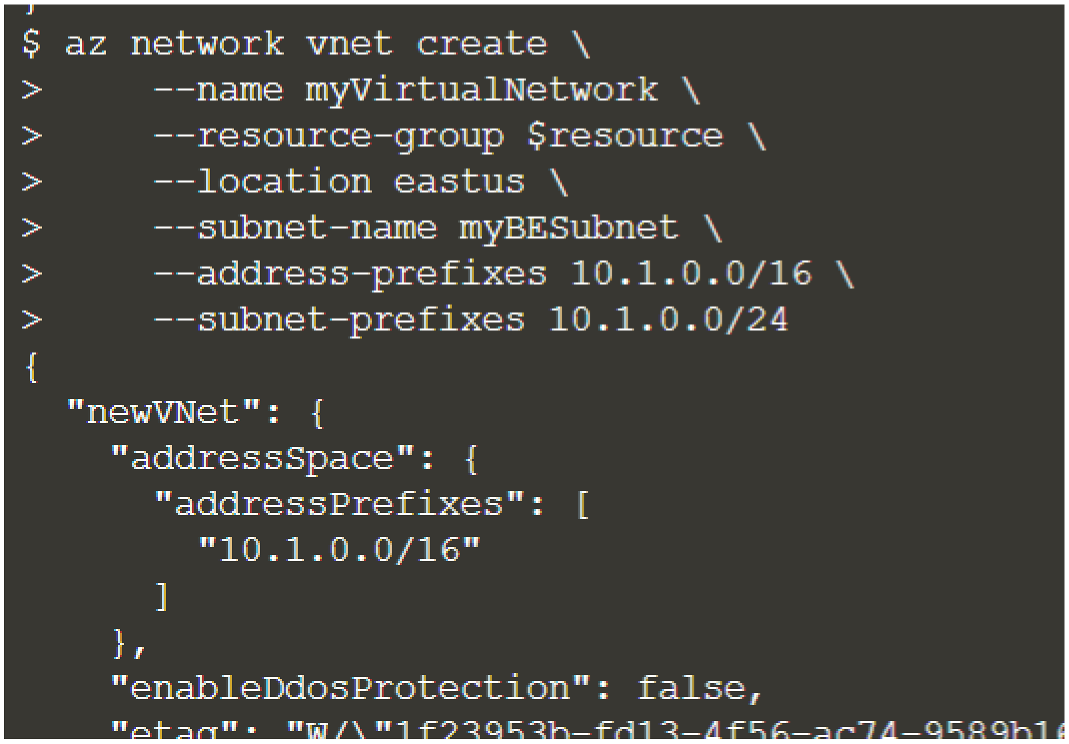


```
]
$ az group create \
>     --name $resource \
>     --location eastus
{
  "id": "/subscriptions/11ce4b70-90f7-41e
  "location": "eastus",
```

Typically you want to use a Standard SKU for load balancers, but we will use a Basic SKU for this exercise. The difference is we will create VMs in an availability set rather than across several availability zones.

Let's first create a virtual network myVirtualNetwork with a backend subnet called myBESubnet:

```
az network vnet create \
    --name myVirtualNetwork \
    --resource-group $resource \
    --location eastus \
    --subnet-name myBESubnet \
    --address-prefixes 10.1.0.0/16 \
    --subnet-prefixes 10.1.0.0/24
```



```
$ az network vnet create \
>     --name myVirtualNetwork \
>     --resource-group $resource \
>     --location eastus \
>     --subnet-name myBESubnet \
>     --address-prefixes 10.1.0.0/16 \
>     --subnet-prefixes 10.1.0.0/24
{
  "newVNet": {
    "addressSpace": {
      "addressPrefixes": [
        "10.1.0.0/16"
      ]
    },
    "enableDdosProtection": false,
    "etag": "W/\"1f23953b-fd13-4f56-ac74-9589b1
```

Step 3 - Create Bastion and Security Group
Things get slightly more complicated here. We are going to create a bastion, which provides Remote Desktop Protocol (RDP) or SSH (Secure Shell) connectivity to your virtual machines over a TLS protocol. This protects your VMs from becoming vulnerable by exposing RDP/SSH ports to the public internet, but providing a means to connect through RDP/SSH. It will keep your VMs private.

Let's first create a public IP address for the bastion we are about to create. For security and availability reasons, we are likely to use a Standard SKU in practice instead of Basic. You can learn more about the public IP address configurations, including the SKU, in the docs:

```
az network public-ip create \
    --name publicBastionIp \
    --resource-group $resource \
    --sku Standard
```

After that, create a subnet called AzureBastionSubnet inside our virtual network for the bastion service. Note that this subnet name is required for the bastion service we are about to create:

```
az network vnet subnet create \
   --name AzureBastionSubnet \
   --vnet-name myVirtualNetwork \
   --address-prefixes 10.1.1.0/24 \
   --resource-group $resource
```
Create the bastion service. Note this may take several minutes:

```
az network bastion create \
   --name myBastionHost \
   --resource-group $resource \
   --public-ip-address publicBastionIp \
   --vnet-name myVirtualNetwork \
   --location eastus
```
There are a few more administrative steps we need to take. We next have to create a network security group (NSG), which filters traffic and has rules on inbound and outbound traffic:

```
az network nsg create \
   --name myNetworkSecurityGroup \
   --resource-group $resource
```
Now let's add a rule to filter traffic. This is done with the command az network nsg rule create as shown here:

```
az network nsg rule create \
   --name myHTTPRule \
   --nsg-name myNetworkSecurityGroup \
   --resource-group $resource \
   --protocol '*' \
   --direction inbound \
   --source-address-prefix '*' \
   --source-port-range '*' \
   --destination-address-prefix '*' \
   --destination-port-range 80 \
   --access allow \
   --priority 200
```

Step 4 - Configure VMs

We now need to modify our network interfaces for two virtual machines. We will achieve this using the az network nic command. As stated earlier, we are going to load balance between just two machines for this exercise:

```
az network nic create \
   --name myVMNic1 \
   --resource-group $resource \
   --vnet-name myVirtualNetwork \
   --subnet myBESubnet \
   --network-security-group myNetworkSecurityGroup

az network nic create \
   --name myVMNic2 \
   --resource-group $resource \
   --vnet-name myVirtualNetwork \
   --subnet myBESubnet \
   --network-security-group myNetworkSecurityGroup
```

We are not quite ready to build the two virtual machines yet. Let's create an availability set, which is a redundancy configuration so if one virtual machine goes down, the other can take over. We will create them using the az vm availability-set create command:

```
az vm availability-set create \
    --name myAvailabilitySet \
    --location eastus \
    --resource-group $resource
```

Let's create the two virtual machines. Notice that we configure the network interfaces for each by using the --nics argument. This will attach our VMs to our myBESubnet we created earlier:

```
az vm create \
    --resource-group $resource \
    --name myVirtualMachine1 \
    --nics myVMNic1 \
    --image UbuntuLTS \
    --admin-username azureuser \
    --public-ip-sku Standard \
    --availability-set myAvailabilitySet \
    --custom-data vm_settings.yaml \
    --no-wait

az vm create \
    --resource-group $resource \
    --name myVirtualMachine2 \
    --nics myVMNic2 \
    --image UbuntuLTS \
    --admin-username azureuser \
    --public-ip-sku Standard \
    --custom-data vm_settings.yaml \
    --availability-set myAvailabilitySet
```

Step 5 - Configure Load Balancer

We have reached final approach for our flight landing. After all those other administrative tasks, it's time to get the public load balancer spun up. First, create a public IP address for the load balancer:

```
az network public-ip create \
    --name myPublicIP \
    --resource-group $resource \
    --sku Basic
```

Now let's create the load balancer resource:

```
az network lb create \
    --name myPublicLoadBalancer \
    --resource-group $resource \
    --public-ip-address myPublicIP \
    --sku Basic \
    --frontend-ip-name myFrontEnd \
    --backend-pool-name myBackEndPool
```

Before we use the load balancer, we need to create a rule for it specifying the tcp protocol as well as the ports and time-out policy, which we will set to 20 seconds:

```
az network lb rule create \
    --resource-group $resource \
    --lb-name myPublicLoadBalancer \
    --name myLoadBalancerRule \
    --protocol tcp \
    --frontend-port 80 \
    --backend-port 80 \
    --frontend-ip-name myFrontEnd \
    --backend-pool-name myBackEndPool \
    --idle-timeout 20
```

Next, let's configure the VMs to the backend pool and attach those configurations to the load balancer:

```
az network nic ip-config address-pool add \
    --address-pool myBackendPool \
    --ip-config-name ipconfig1 \
    --nic-name myVMNic1 \
    --resource-group $resource \
    --lb-name myPublicLoadBalancer

az network nic ip-config address-pool add \
    --address-pool myBackendPool \
    --ip-config-name ipconfig1 \
    --nic-name myVMNic2 \
    --resource-group $resource \
    --lb-name myPublicLoadBalancer
```

Let's also apply the CustomScript extension to our VMs. This allows us to easily run scripts on our virtual machines, which in this case will be used to install NGINX as a web service. We will do that with this script and apply it with the az vm extension set command:

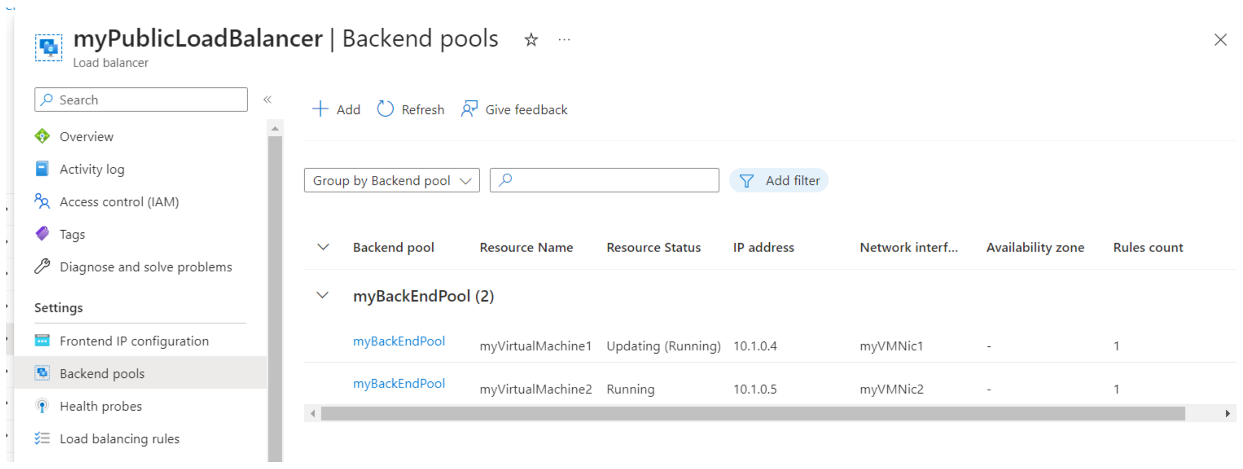
```
az vm extension set \
  --publisher Microsoft.Azure.Extensions \
  --name CustomScript \
  --resource-group $resource \
  --vm-name myVirtualMachine1 \
  --settings '{ "fileUris":
["https://raw.githubusercontent.com/Azure-Samples/compute-automation-configurations/master/automate
_nginx.sh"], "commandToExecute": "./automate_nginx.sh"}'

az vm extension set \
  --publisher Microsoft.Azure.Extensions \
  --name CustomScript \
  --resource-group $resource \
  --vm-name myVirtualMachine2 \
  --settings '{ "fileUris":
["https://raw.githubusercontent.com/Azure-Samples/compute-automation-configurations/master/automate_nginx.sh"], "commandToExecute": "./automate_nginx.sh"}'
```

Print the IP address and copy it to a browser. You should land on a page being greeted by one of the virtual machines:

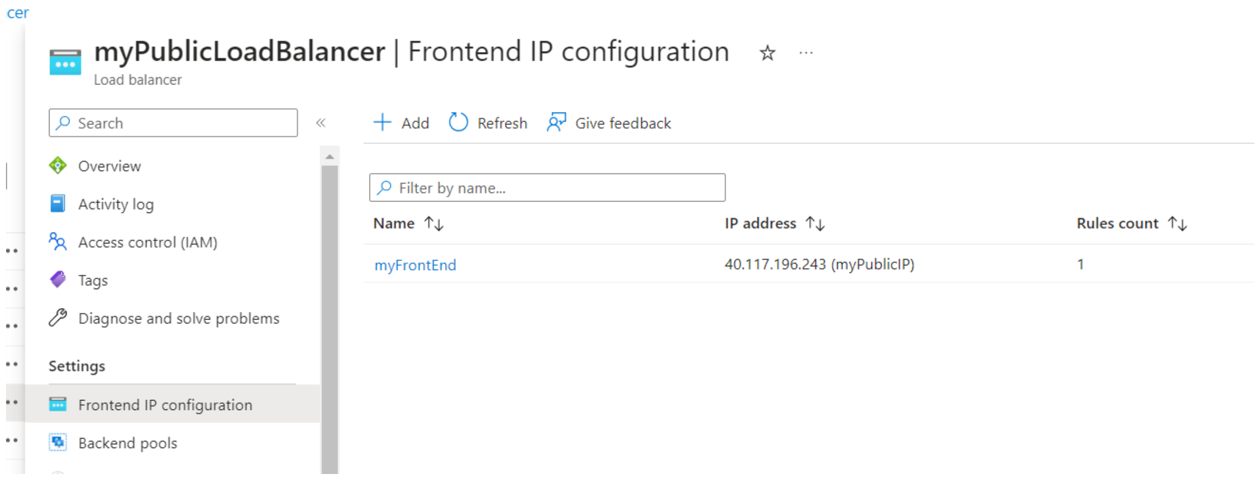
```
az network public-ip show \
    --resource-group $resource \
    --name myPublicIP \
    --query ipAddress \
    --output tsv
```

That was a lot of work, but we reached the end! You will see your virtual machines are at work and serving up the web page greeting. However, the load balancer served as a proxy and served the page from one of the two virtual machines in the availability set. If the load balancer experienced a heavy load of requests, it would distribute those requests as effectively as possible between the allocated virtual machines.



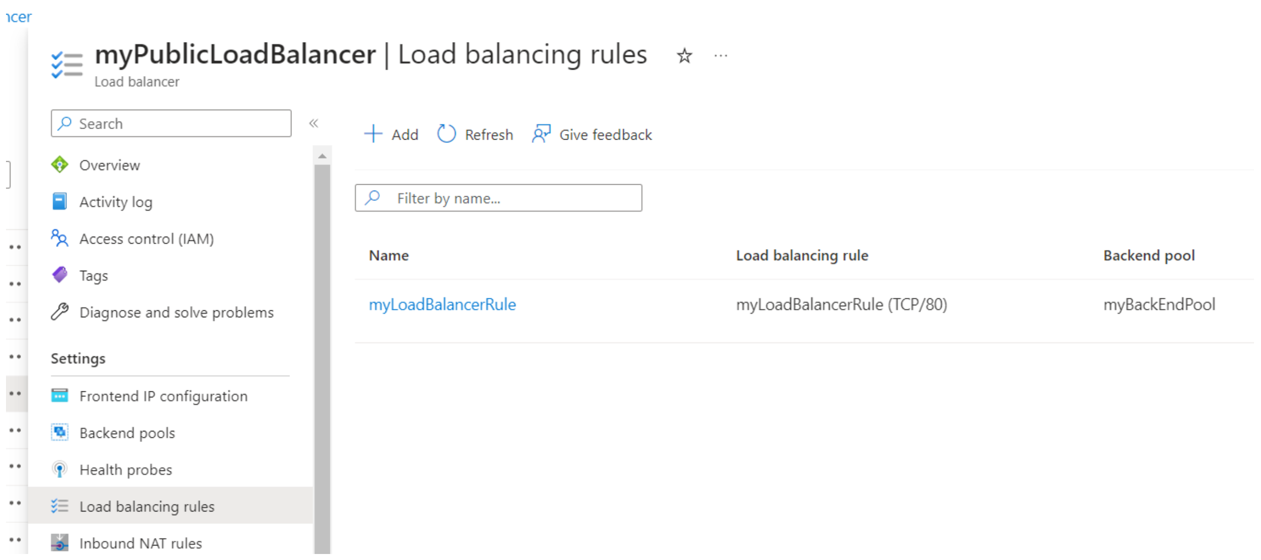
myPublicLoadBalancer | Backend pools
Load balancer

Backend pool	Resource Name	Resource Status	IP address	Network interf...	Availability zone	Rules count
myBackEndPool (2)						
myBackEndPool	myVirtualMachine1	Updating (Running)	10.1.0.4	myVMNic1	-	1
myBackEndPool	myVirtualMachine2	Running	10.1.0.5	myVMNic2	-	1



myPublicLoadBalancer | Frontend IP configuration
Load balancer

Name	IP address	Rules count
myFrontEnd	40.117.196.243 (myPublicIP)	1



Name	Load balancing rule	Backend pool
myLoadBalancerRule	myLoadBalancerRule (TCP/80)	myBackEndPool