Networking Config for Readit Application

Vnets : These are the networks in which resources are deployed. Deploying a resource outside this Vnet is not possible as Azure will create a resource inside a Vnet by default.

Resources inside the Vnet can freely communicate with each other with their private IP's. We can use Vnet peering to enable communication among devices within other Vnet.

Vnets are restricted to one Region and a single subscription. We can have 50 Vnets per subscription.

Subnets : Creating a Vnet will create a default subnet with it. When resources are deployed, they are usually deployed in a default subnet unless we specify them to be in another one we will create.

The NIC of a VM is usually attached to the subnet and not the actual VM. We can have 3000 subnet per Vnet.

NSG's : They act as a gatekeeper and are used to control the traffic in and out of a subnet. They are usually attached to the NIC of VM's and are created by default with a VM. We should ensure that RDP/SSH port are not open to anyone with NSG.

They work by analysing:

- source IP
- source port
- destination IP

- destination port
- protocol

These work as a security rule where connections are allowed or denied based on a number. The lower the assigned number, higher the priority.

<u>We can create multiple subnets inside vnets</u> :

1 : When creating a Vnet, we can create multiple subnets maybe 1 for the front-end and 1 for the back-end.

2 : We should ensure that their address spaces do not overlap.

3 : We can create a front-end VM and a backend VM in their respective subnets and try connecting them with **rdp/ssh**.

<u>Using remote desktop protocol, try entering the private ip details of the other vm in the rdp client which is in the different subnet.</u> <u>It should work.</u>

<u>Note</u> :

After creating a VM, for additional security, ensure to deny ssh/rdp access and instead add an inbound nsg rule to :

<u>add a service tag access from cloudshell and equivalent service rather than ssh/rdp.</u>

- service tags are usually used to allow a group of specific IP’s. eg.
    - If you want to block internet traffic but allow Azure services, you can create a deny rule using the Internet Service Tag.

<u>Consider a scenario where we want to move our VM to a different subnet within the Vnet.</u>

We can get the list of devices/VM’s in the Vnet by going in the Connected Devices under Vnet Settings

1 : We can create a new subnet inside the Vnet.

2 : Attach the VM’s NIC to the new subnet by :
VM’s networking settings => Network Interface => IP Configurations => Select the newly created subnet.

<u>The private IP address may change after changing the subnet. As we know, NIC’s are attached to the subnets and not the actual VM’s</u>

3 : Inorder to connect to a Linux VM from cloudshell which is essentially giving **ssh** access to cloudshell, we can add a NSG rule with source => service tag and source service tag => AzureCloud.
This is because the IP configuration of the VM changed after changing its subnet.

4 : Now the VM will work as usual.

<u>We can even create a new and separate standalone NSG from the portal and can attach it to the VM or a subnet.</u>

Creating this new NSG and attaching it to the subnet will prevent us the ssh access to the VM as by default nsg does not allow the port 22 an access because it is standalone one and not created with the VM.
Inorder for the connection to work, we add the an inbound nsg rule with security tag as above to the subnet (new subnet the VM is added to) to allow ssh connection to the machine from cloudshell.

<u>We usually can attach Nsg's of a VM to this subnet or can create a new one.</u>

<u>Moving a VM to a different Vnet :</u>

We usually move a VM to a different Vnet by deleting the VM entirely except its **disk** which we will attach to the new VM we will create in a new Vnet.

1 : After deleting the VM, we can search for disks in the portal and select the disk previously attached to the VM.

2 : Select **Create VM** from the disk overview page and proceed as usual.

<u>Securing an access to VM :</u>

1 : JIT

- We can use just in time VM Access which will open an port for access on demand and then close it.
- It remains closed rest of the time.
- It can be configured from VM's portal page.

2 : VPN :

- Creates a secure tunnel to the Vnet
- Can be configured in a way no one can connect to it.
- Requires License

3 : Jump Box

- It essentially is a VM that is placed in a Vnet which we will use to route the traffic to a specific VM via this VM (Jump box).
- We will only allow access to this Vnet. Access to other VM's will be given via this. Even access to other VM's in other peered Vnet will be given via this Jump Box

4 : Bastion :

It enables us to login in to VM’s via its private IP and is a fully managed service. We can access a VM without exposing it to the internet and without needing any VPN.

Service Endpoint :

When we connect to a resource like an App Service or a database from a VM in a VNet, the traffic by default traverses the internet because these services expose their public IPs. This is not secure, as the traffic is routed through the public network.

Service Endpoints help address this issue by ensuring that the traffic remains within the Azure backbone network rather than traversing the internet. However, the traffic still leaves the VNet but does so securely over Azure’s internal network.

With Service Endpoints, the VM connects from its private IP to the public endpoint of the service, but Azure ensures that the traffic remains within its trusted network.

Key Use Cases:

- Connecting PaaS services like Azure SQL Database, App Service, Storage Accounts, etc., securely within an Azure environment.

- Ensuring that Application Gateway (AG) or other services communicate without exposing traffic to the public internet.
- Restricting access to Azure PaaS services so that only specific VNets and subnets can connect.

By default, Azure PaaS services are publicly accessible, but Service Endpoints allow secure, direct connectivity from a VNet while keeping traffic within Azure's infrastructure.

Key Points :

- If you enable a service endpoint for a storage account, all services within that storage account (like Blob, Table, Queue, and File) can be accessed from the VM
- Service endpoints do not restrict access to just one VM. Any resource in the entire subnet can access the storage account.
- You must configure the Storage Account Firewall to allow traffic only from the selected virtual network and subnet.

We setup this Endpoint in 2 steps :

We first add a Service endpoint to the Subnet the VM is in and then add the subnet as a Resource to the Storage Account's firewall settings. The ACL's in the SA (Storage Account) wont be needed any more. The connection is secure and follows an optimised route.

1 : We set it on the subnet level by going in to Service Endpoint tab and then selecting the service we want to connect to from our Vnet/Subnet i.e Storage Account.

2 : Specify the subnet that we want to connect to the storage with.

Decide Where the Service Endpoint is Needed :

Purpose of Service Endpoint:
Service Endpoints secure connections to Azure PaaS services (e.g., App Service, Storage, SQL Database).

Determine the Protected Resource:

If you want the Application Gateway to securely connect to the App Service, the Service Endpoint should be configured on the subnet where the Application Gateway is deployed.
If the App Service needs to securely access another Azure service (e.g., Azure SQL Database), configure a Service Endpoint for the respective service that need to access a resource on the App Service's subnet (via VNet Integration).

Private Endpoint :

We usually create PE to extend the PAAS Service to the Vnet and allocate an IP from the Subnet itself to the PAAS Service.

This secures the connection as the connection happens between the Private IP’s of the Services.

We essentially enable the PE on the Storage Account by :

- 1 : searching the Private Link in the portal
- 2 : adding the Service we want the PE for under its resource tab
- 3 : Configuring the Subnet in which the VM resides in.

<u>Private Link :</u>

This is an enhanced solution compared to Service Endpoint. It essentially creates a direct route from the Vnet to the external Managed Service/resource like App Service.

The traffic does not leave the Vnet in this case which ensures best security. Without Private Endpoint, the traffic would leave the Vnet and traverse the Internet.

The communication happens between the VM’s Private IP and Managed Service’s Public IP.

<u>Scenario :</u>

<u>I : Connecting to the PAAS Service from on-prem network :</u>

1 : Create a VPN Gateway.

2 : Connect Your On-Premises Network to the VNet

3 : Enable a Private Endpoint on the PaaS Service

This assigns a private IP to the PaaS resource from your VNet's subnet.

4 : Configure DNS

Set up Azure Private DNS Zone to resolve the PaaS service’s private IP from your on-premises network.

<u>II : If we are to connect 2 Vnets without Peering them, we can use Private Link :</u>

- Create an Load Balancer in the Vnet which has a VM in its subnet we wish to connect to from another Vnet. Create a Private Link on that LB
- In the other Vnet, create a PE and connect to the LB’s Private Link with this Private Endpoint of the other Vnet

To set up Private Link :

1 : Search for Private Link and then Create one with the RG and Link name.

2 : Under Resource tab, select the DB or Service we wish to connect.

3 : Under Configuration tab, select the subnet that we want to connect with and even configure the DNS appropriately.

Vnet Integration :

Similar to the SE/PL above, Vnet Integration provides a similar functionality but in reverse. We configure Vnet Integration from inside the Web App settings to allow the traffic from the Managed Service i.e. App Service to reach the VM in the Vnet.

Vnet Integration is implemented in the following steps :
1 : Go to the Managed Service i.e App Service that we have running.

2 : Go to its Network settings => Vnet Integration => Select the Vnet => Subnet

This works only in case of an empty subnet.

Access Restrictions :

Just as we have Nsg's for VM's and Subnets, we have Access Restrictions for App Service. These are rules we configure to allow/deny access to other resources.

egs :

1 : We can allow the Access to the App Service from the front-end App service or a VM.

2 : Restrict to a specific customer.

Steps to implement Access Restrictions :

1 : Go to the Networking tab of the App Service.

2 : Under Inbound Traffic, select access restrictions.

3 : Check/uncheck Allow Public Access

4 : Add our own IP preferably like we do in the Nsg and then click Add Rule and then save on the top of the page.

Note : After we add a rule, the default rule that is allow all traffic will change to deny after we have the newly created rule set to higher priority.

5 : If we delete the created rule, we have to select **Unmatched Rule Action** to allow again so that the default rule changes to allow all as previously mentioned.

<u>Load Balancer :</u>

LB's are used to route the traffic to multiple VM's so that the traffic is distributed evenly across all the machines. It uses an algorithm to route the traffic to the appropriate machine. It works with VM's and VMSS.

Similar to an NSG, they work by analysing :

- source IP
- source port
- destination IP
- destination port
- protocol

There are 2 types of LB's :
<u>1 : Public (External) :</u>

It exposes a Public Endpoint to the Internet. It helps all the external traffic to connect to the backend services which sit behind the LB. It is commonly used to connect to the external API's that need traversing the internet.

eg: API gateways for third-party integration.

<u>2 : Private (Internal)</u>

It operates entirely within a private network (e.g., Virtual Private Cloud or on-premises network). It does not have a public IP and is accessible only from within the private network. It is helpful for Applications that need to communicate without needing to expose themselves to the internet.

It is assigned a Private IP and works well Internal Firewalls. Protects the application from external threats. It is best for using with internal resources but not recommended for use with external resources.
<u>It cannot handle Http and has no protection.</u>

We have 2 Azure-specific types :

Basic :
It is free but offers no redundancy. Can handle upto 300 instances and is open offering little security

Standard :
It is not free but offers redundancy. Can handle upto 1000 instances and is secure.

There are 4 main configurations :

<u>1 : Front-end IP Configuration :</u>

It exposes a single public IP to the internet so that it can later distribute the traffic to the underlying VM's.

<u>2 : Backend Pools :</u>

Contains a list of VM's the LB will route the traffic to.

<u>3 : Health Probes :</u>

Checks the health of the underlying VM's so that it can decide whether to route the traffic to that specific VM or not. We can choose the intervals in seconds the health probe will check for the unhealthy VM and then decide to declare it as down.
	It runs on the VM's host and no outside traffic is allowed by default.

<u>4 : Load Balancing Rules :</u>

It contains rules which connect front-end IP to the backend pool.

<u>**Application Gateway :**</u>

It essentially is a web traffic LB. It works somewhat similar to a load balancer but works on Layer 7 i.e. working with the actual application data where as the LB works on Layer 4.
	This essentially means that AG has access to Url's and its parameters which we can configure to allow/deny the traffic based on the Url path.

eg. a request with params destined for /movies will reach the backend pool with VM's hosting /movies resources.

AG comes with WAF in its WAF_V2 plan where it runs in 2 modes :

1 : Detection
Detects threats but does not do anything to resolve

2 : Prevention
Prevets threats in the first place

<u>AG usually has it own Subnet or even a Vnet. So its our responsibility to allow the traffic to our other services only from this subnet/vnet housing the AG and not from anywhere else.</u>

eg. in an architecture, we can allow the traffic only from the AG's subnet using Nsg. Similarly, we can allow traffic to the App Service by placing a Private Link/Service Endpoint.

<u>Configuring the AG :</u>

1 : Backend Pools
Describes the VM's, VMSS or App Service we have connected to the AG. It describes what all services would reside behind the AG to which we will route the requests to.

2 : Http
Setting for the incoming request as AG knows about the path and params of the request

3 : Rules
Rule that help connecting backend pool with the AG

4 : Listeners
Receives request on a specific port or a protocol

5 : Front-end IP Configurations
Public IP (PIP) exposed by the AG.

<u>Deploying AG with portal : (Most expensive Service)</u>

1 : Search for AG as usual in the search and then create it in the Specific region along with its resource-group.

2 : Select the SKU as Std V2 which is without the WAF and disable autoscale.

3 : Leave http2 enabled.

4 : Create a new Vnet for the AG. Ensure the address space does not overlap with the other VM Vnets. Place the AG in its own subnet inside the Vnet.

5 : Under Front-ends tab, we should select the IP for the AG to which the request from the Internet or from outside the network will connect to which will later be routed to the backend pools with the help of Backend IP of the AG. Make sure the IP is a static public IP.

6 : Under back-end tabs, we will create backend-pools for our resources i.e. 1 pool for app service and another pool for our catalog VM.

- app service might be visible when creating a BE pool as it is not inside a Vnet
- catalog-vm wont be visible if it is in a different vnet. We can just create an empty pool with catalog-pool name and then set it to *add backend pool without targets* setting.

7 : We created front-ends and backends for the request to get routed to and from. Next we will create routing rules to actually establish a connection between them and define how the req will flow through AG.

8 : We will create separate routing rules for different services ie. for catalog vnet or inventory app. In every rule, there is a configuration to be done under listener (front-end) and Backend targets.

eg:

- Under listener tab, we add listener for catalog application alongwith the front-end IP, protocol and port. We can select any port in this tab but 8080 is preferred.
- Under Backend Targets tab, when adding backend-settings, select the backend-pool we created for catalog previously and add a backend setting to connect to the VM. *This is the actual setting that we will use to*

call our backend Application. Here we need to specify the exact protocol and port our application is running on.

- Similarly for Inventory service, under backend targets tab, when adding backend-settings, select https as the AG will connect app Service using https on port 443. Select yes for Backend-server certificate.

We should configure some more settings in the Application Gateway Backend Settings tab where we will

- *<u>Host name</u>: select a backend setting and enable Host name, which ensures the value on the backend server in the pool matches with the one sent by the client in the host header.*
- *<u>Host name override</u> : Pick host name from backend target* setting to match the App service host name with the host name the client will send the request to.

Try calling the AG with its public IP. It should run the Inventory Service.

After creating the Application Gateway, we should now go ahead with connecting the resources to it.

<u>Connecting the Inventory App (App Service) to the AG :</u>

We cant allow App Service to be directly accessible from the internet ie. the way we access it by clicking the **default**

domain which is a link in App service overview. Instead we have the AG for the exact same purpose ie. to act as a gatekeeper and forward all the requests to the backend pool/ resources.
Our resources will be accessible only from the AG and from nowhere else.

We can now use **Service Endpoint** to establish a connection between AG and the App Service. This will enable us to create some Access Restrictions on the App service as well.

- Go to subnet of the AG, select ServiceEndpoint and then add Microsoft.Web from the dropdown to select the Subnet on which we will add the service endpoint to.
- we are essentially adding the service endpoint to the AG's subnet. This will help us configure the Access restrictions on the App Service in a way so that we can just allow the AG's subnet to access it. This is possible only if we have placed the service endpoint on the AG's subnet.

<u>Similar to how we connected our Inventory App, we will now connect and configure our Catalog VM (Windows VM) with the AG :</u>

We will connect it with the VM's Private IP.

The steps are :

1 : Add Network Peering between the Catalog and the AG Vnet. This is because both are in different Vnets. Initiate the peering preferably from the VM's Vnet.

2 : Add the Catalog VM's Private IP to the Backend pool of the AG as we already added the configuration. Inside the backend pool, select **target type** as IP address and add the IP.

3 : Ensure to deny the Inbound traffic from the internet if any is allowed as we will just allow the traffic from the AG.

4 : Inorder to test whether the connection is established and the Catalog App opens, connect to the AG's Public IP and add the catalog's app's port as well. It should run.