

INFRASTRUCTURE AS A CODE

A Project Report

Submitted by

SOURADEEP BANERJEE (R110216159)

**in partial fulfilment for the award of the degree
of**

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
At**



**SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM & ENERGY STUDIES**

Bidholi Campus, Energy Acres, Dehradun – 248007.

July – 2019

CONTENTS

CANDIDATE’S DECLARATION	Error! Bookmark not defined.
CERTIFICATE.....	Error! Bookmark not defined.
ACKNOWLEDGEMENT.....	3
ABSTRACT.....	4
INTRODUCTION.....	5
SOFTWARE DEFINED INFRASTRUCTURE (SDI)	5
Layers	5
Benefits of SDI In Cloud Service	6
INTRODUCTION TO DevOps	7
Understanding levels of Scope in Azure	8
AZURE ACTIVE DIRECTORY.....	10
Used by.....	10
Licenses	10
RBAC-Role Based Access Control	11
Work Done	13
STORAGE ACCOUNT IN AZURE	15
Types of Storage Accounts.....	16
Access Tiers.....	17
Replication.....	17
Factors Affecting Cost.....	18
Work Done	18
VIRTUAL NETWORKS IN AZURE	21
Communication between Azure Resources	21
NETWORK SECURITy GROUPS	22
Work Done	22
ARM – AZURE RESOURCE MANAGER TEMPLATES	27
Modes of Deployment	27
Execution of ARM templates	27
Work Done	28
TERRAFORM	33
Execution of Plan.....	33
Configuration.....	34
Work Done	34
PULUMI - The New IaC Tool.....	38
Benefits.....	38
Work Done	39
REFERENCES.....	43



The innovation driven
E-School

CANDIDATE'S DECLARATION

I hereby certify that the project work entitled “**Infrastructure as a Code**” in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in CLOUD COMPUTING AND VIRTUALIZATION TECHNOLOGY and submitted to the Department of Virtualization at School of Computer Science and Engineering , University of Petroleum and Energy Studies, Dehradun, is an authentic record of my work carried out during a period from **6th June,2019 to 19th July,2019** under the supervision of **Mr Manish Kumar, Associate Architect at Applied Information Sciences, Hyderabad.**

The matter represented in this project has not been submitted by me for the award of any other degree or any other University.

(Souradeep Banerjee)
(R110216159)

This is to certify that the statement made by the candidate is correct to the best of my knowledge.

Date: 19 - July - 2019


19/07/19
Mr Manish Kumar
Project Mentor


Dr. Deepshika Bharghava
Head of Department <CCVT>
School of Computer Science and Engineering
University of Petroleum and Energy Studies
Dehradun – 248007 (Uttarakhand)

CERTIFICATE

This is to certify that the project titled “**Infrastructure as a Code**” is the bona fide work carried out by Souradeep Banerjee, a student of B Tech (CSE) of University of Petroleum and Energy Studies, Dehradun(India) during the academic year 2019-20, in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (Computer Science and Engineering) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Place: Hyderabad
Date: 19-July -2019


Signature of the Guide 19/07/19



ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Mr Manish Kumar**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank our respected programme Head of the Department **Dr Deepshika Bharghava**, for her great support in doing our project in Areas in CIT.

We are also grateful to **Dr Manish Prateek**, Director, SoCS and **Dr Kamal Bansal**, Dean, CoES, UPES for giving us the necessary facilities to carry out my project work successfully.

We would like to thank all of my **friends** for their help and constructive criticism during my project work. Finally, I have no words to express my sincere gratitude to my **parents** who have shown us this world and for every support they have given me.

Name: **Souradeep Banerjee**

Roll No: **R110216159**

ABSTRACT

The main purpose of this project is to learn the basics of Infrastructure as a Code, how does it work and what are its advantages over traditional hardware allocation and configuration processes while acting as the cloud service provider. Additionally, we discuss about Software Defined Infrastructure in this respect which mainly depends on the concepts of reusability and repeatability. As a part of this we will focus on ARM templates, Terraform and Pulumi.

INTRODUCTION

Infrastructure as a Code or IaC as the name suggests, helps the IT developer professionals to automatically manage and provision technology stack for an application through software, rather than using a manual process every time to configure a hardware for a specific cloud service provider. The value of IaC can be broken down into cost, speed and risk of error. The main idea behind Infrastructure as a Code is the implementation of Software Defined Architecture. This Software Defined Architecture or SDI leads to the emergence of a whole new role called DevOps.

SOFTWARE DEFINED INFRASTRUCTURE (SDI)

Software-defined infrastructure consists of fully virtualised compute, networking and storage resources that are logically pooled and can be managed as if they were software. This allows policy-based infrastructure provisioning and enables IT automation.

The main idea behind software defined infrastructure is not to be tied with some specific hardware and decrease human intervention. It allows any critical IT function to be fully automated and integrated in case of deploying some service as a cloud service provider.

Layers

The different layers of Software Defined Infrastructure are as follows:

1. **Physical Infrastructure:** At the base level SDI comprises of the different hardware resources such as servers and networking devices, endpoint terminals which are being allocated to a specific need.

2. **Virtualisation Layers:** At this level the physical resources such as storage and networking devices are being virtualised.
3. **Software defined Capabilities:** Different software defined functions such as Software Defined Networking, Software Defined Compute, Software Defined Storage are being applied to the virtualised set of resources.
4. **Management Services:** At the infrastructure management level, SDI may involve the user-interface to define parameters such as SLA performance, availability, scalability and elasticity. IT admins or internal IT users may also request provisioning of resources.

Benefits of SDI In Cloud Service

1. **Simplified IT models:** Now a days, full data center virtualization has enabled us to allow compute, networking and storage resources to be flexibly configured on a per application basis.
2. **Automated configuration and backups:** Functions like application requirements and disaster recovery are being automatically handled by Application aware interface.
3. **Management:** There are dashboards like interface which helps to monitor and provision software-defined infrastructure.
4. SDI is capable of placing loads in both public and private cloud to maximize performance and efficiency.

INTRODUCTION TO DevOps

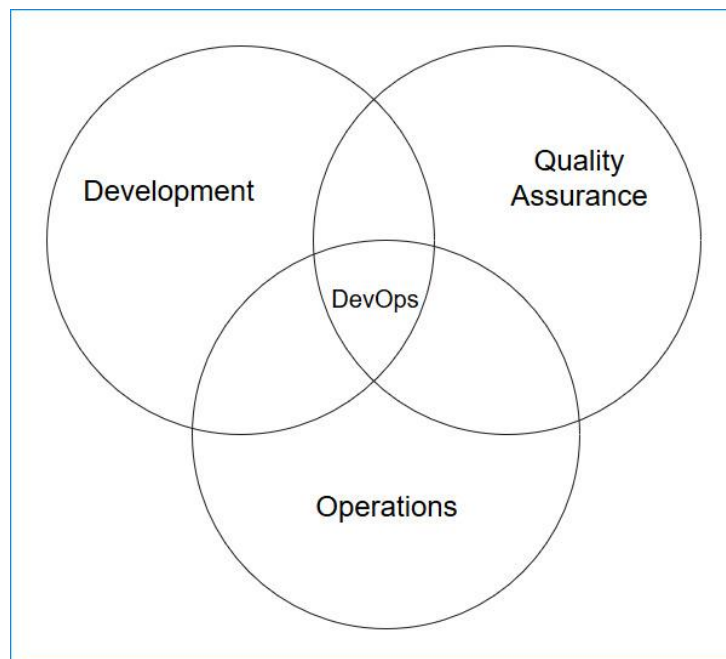


Figure No. 1.1: DevOps

DevOps is a set of software development practices that combines software development and information technology operations to shorten system development life cycle.

The main part of DevOps is software defined architecture. Azure provides this infrastructure as a code service using ARM templates. Azure resource manager or ARM is the deployment and management service for Azure. It provides a consistent management layer that enables us to create, update, and delete resources in our Azure subscription. We can use its access control, auditing, and tagging features to secure and organize our resources after deployment.

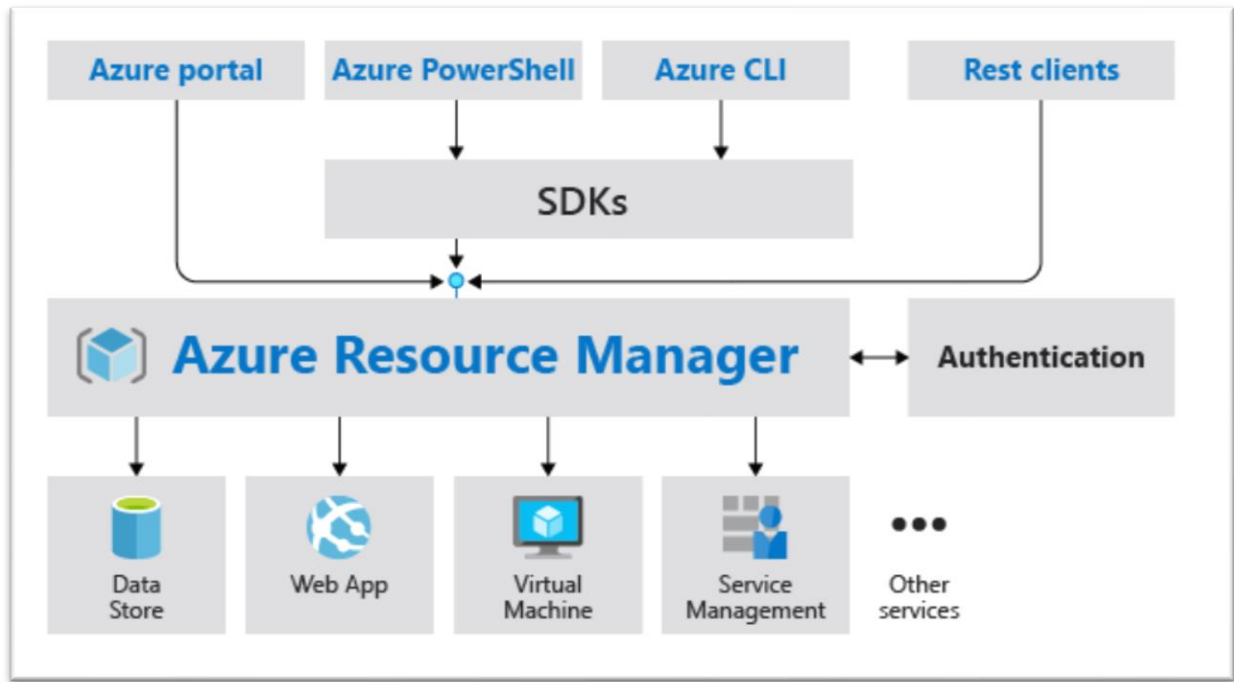


Figure No. 1.2: Azure Resource Manager

Understanding levels of Scope in Azure

Azure environment is a hierarchal structure which consists of different levels of scope. At the base level there is Azure Active Directory which consists of all the subscriptions. There can be 0 to n number of subscriptions in an azure active directory.

Inside an azure subscription we can have 0 to n number of resource groups and each of these resource groups can have 0 to n number of resources.

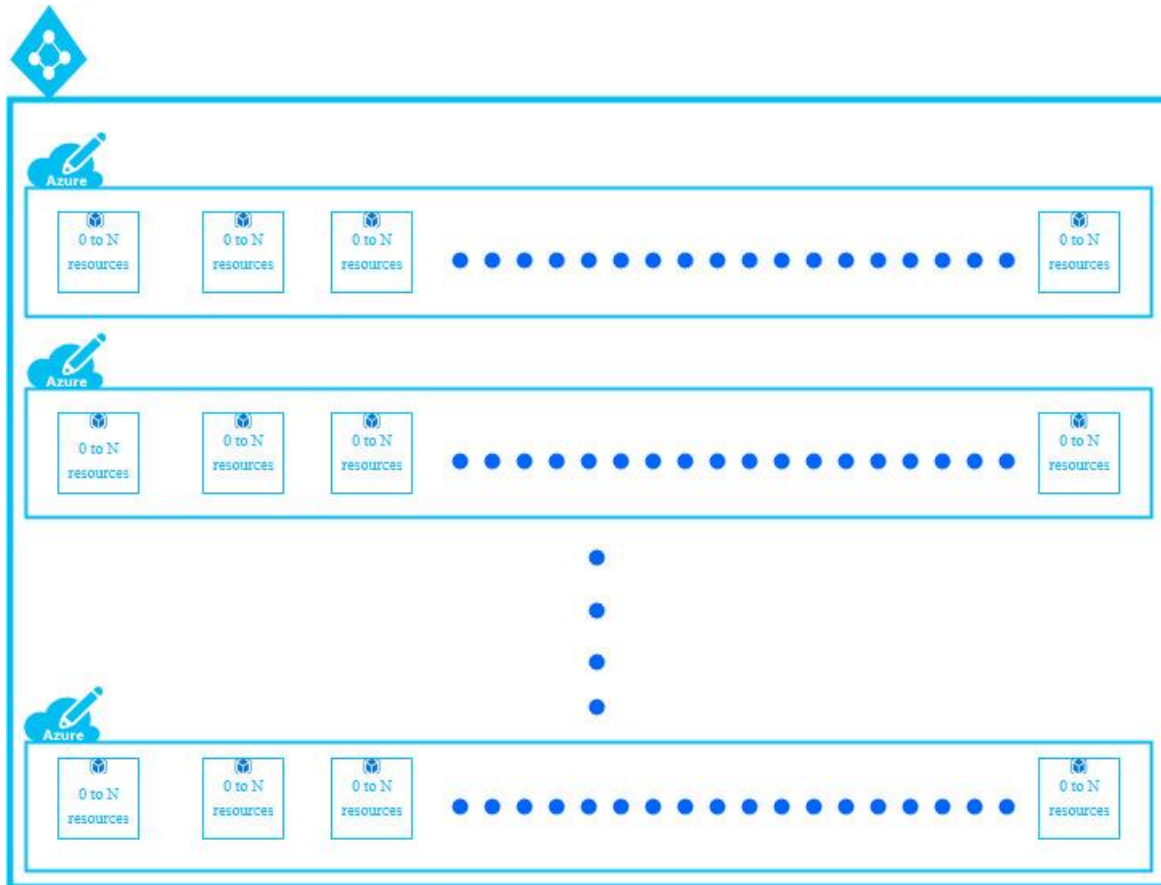


Figure 1.3: Scopes in Azure Environment

Azure uses classes for each type of resources that are being deployed in the resource groups. For example, Microsoft.Networks class is being used for deployment of network related resources like virtual networks. For each resource azure uses a unique resource ID which is a **fully qualified domain name** (FQDN) which contains the subscription id under which the resource is being deployed along with the resource group name, provider and the resource name.

EXAMPLE:

```
/subscriptions/5a4e2629-2885-4a52-b513
ecbe4a3c035f/resourceGroups/MyAzureRsrcGrp/providers/Microsoft.Storage/storageAccounts/afqwdfqewdqecqaeqefqwfd'
```

AZURE ACTIVE DIRECTORY

Azure active directory is Microsoft's cloud-based identity and access management. Various external resources such as Office 365, Azure Portal and other SaaS services use Azure AD so that employees can sign in.

Various internal resources such as applications on our corporate network and intranet along with other cloud applications can also use Azure AD.

Used by

- **IT Admins:** IT admins use Microsoft Azure in order to control access to their applications. We can use Azure AD to enable multi-factor authentication.
- **Application Developers:** Azure AD lets the application developers to add Single Sign On feature in their application and allowing the users to use pre-existing credentials.
- **Microsoft 365, Office 365, Azure, or Dynamics CRM Online subscribers:** By subscribing to one of these services, user is already using Azure AD.

Licenses

Microsoft online business services such as Office 365 or Microsoft Azure, require Azure AD to sign in and use their services. So, with the subscription of Microsoft Office 365 we get Azure AD licenses automatically.

1. **Azure Active Directory Free:** This license provides user and group management on-premises directory synchronization, basic reports, self-service password change for cloud users, and single sign-on across Azure, Office 365, and many popular SaaS apps.
2. **Azure Active Directory Basic:** Basic also provides cloud-centric app access, group-based access management, self-service password reset for

cloud apps, and Azure AD Application Proxy, which lets us to publish on premises web apps in Azure Active Directory.

- 3. Azure Active Directory Premium P1:** P1 lets us to use hybrid access to both on-premises and cloud resources. It also supports advanced administration, such as dynamic groups, self-service group management, Microsoft Identity Manager which allow self-service password reset for on-premises users
- 4. Azure Active Directory Premium P2:** This license, in addition to other features, provides Azure AD Identity Protection to help provide risk-based Conditional Access to our apps and critical company data.
- 5. "Pay as you go" feature licenses:** This license gives additional features like Azure AD Business-to-Customer (B2C). B2C helps to provide identity and access management solutions for customer-facing apps.

RBAC-Role Based Access Control

This hierarchal structure gives us an important tool called RBAC. RBAC helps us to authorize users with different levels of permissions in the present directory. If two roles are being added for the same user then the role with higher privileges is being taken into consideration.

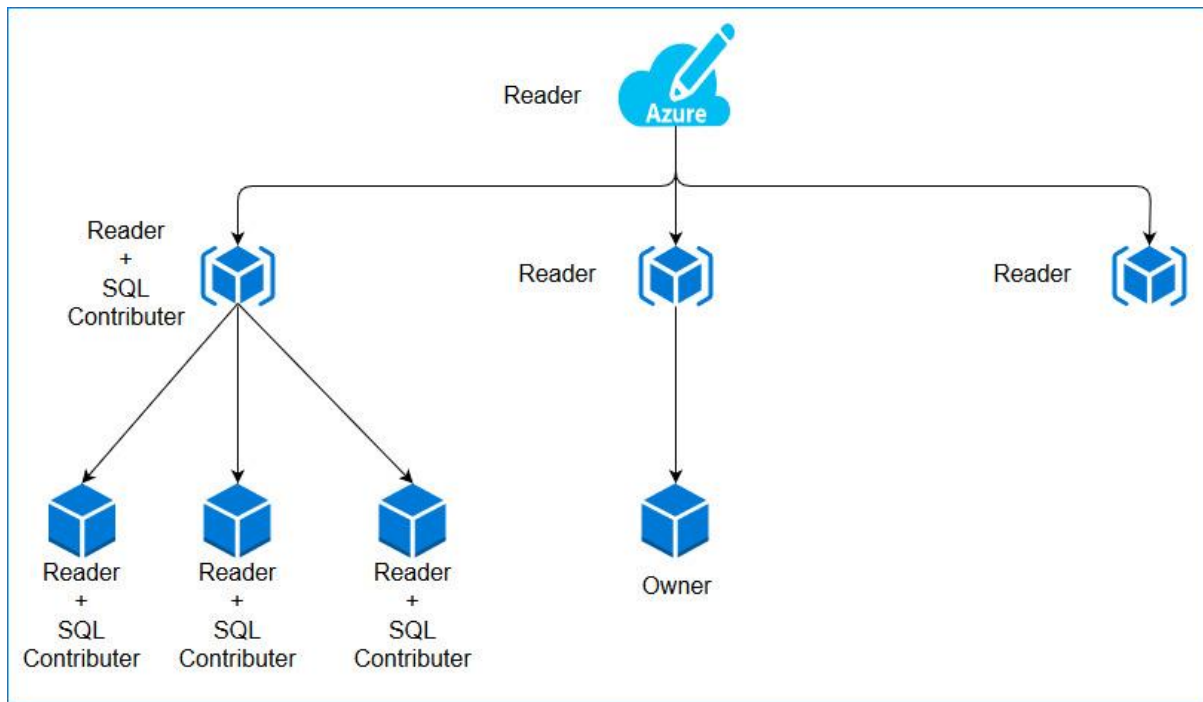


Figure 1.4: Role Assignment in RBAC

There are basically three factors which matter while assigning roles in Azure. These are “who”, “which” and “what”.

- **Who** – To whom are the roles being assigned in the subscription? It is also known as the security principal. A security principal may be a user, group, service principal or managed identity.
- **What** – What role is being assigned to the user. It is also known as the role definition. It contains all the permissions.
- **Which** – To which resources or resource group or subscription is the role being given. It is also known as the scope of the role assigned.

Work Done

Guest users were being created in the Azure portal using Azure Active Directory.

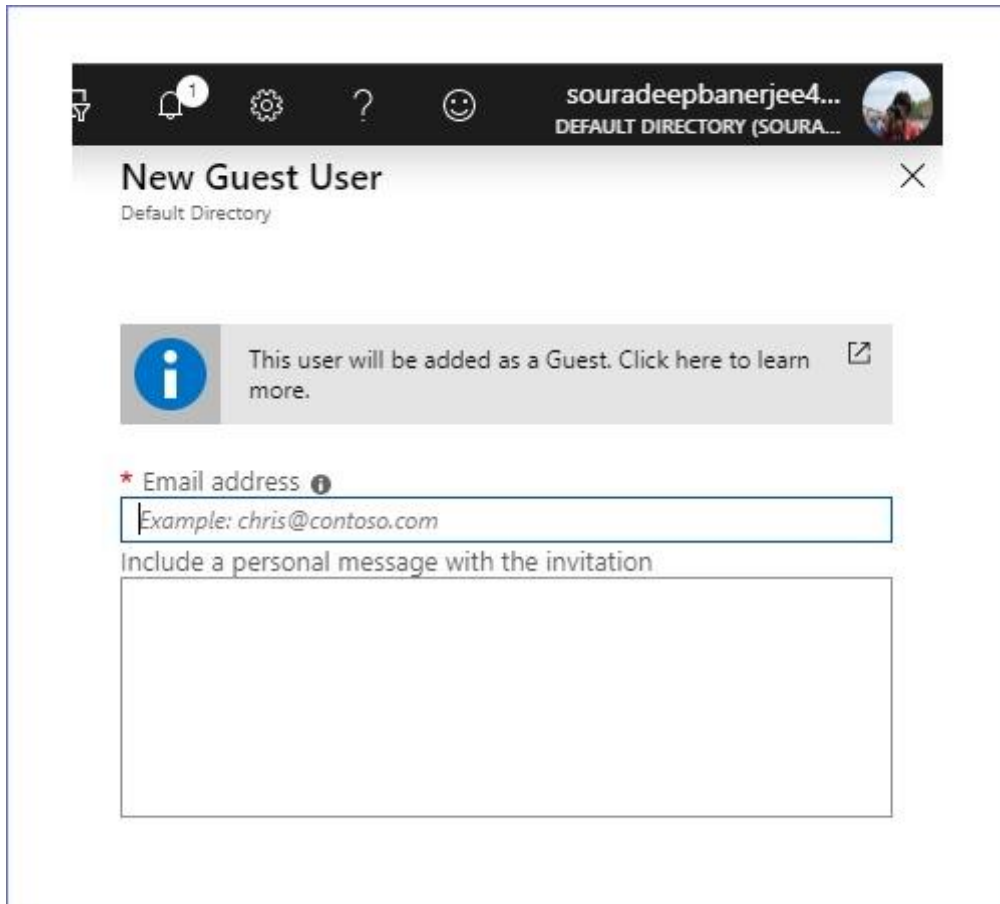
The image shows a screenshot of the 'New Guest User' form in the Azure portal. At the top, there is a dark navigation bar with icons for home, notifications, settings, help, and a user profile. The user profile shows 'souradeepbanerjee4...' and 'DEFAULT DIRECTORY (SOURA...'. Below the navigation bar, the title 'New Guest User' is displayed with a close button (X) on the right. Under the title, it says 'Default Directory'. A grey information box contains an 'i' icon and the text: 'This user will be added as a Guest. Click here to learn more.' with an external link icon. Below this, there is a red asterisk and the label '* Email address' with an information icon. A text input field contains the placeholder text 'Example: chris@contoso.com'. Below the input field, there is a checkbox labeled 'Include a personal message with the invitation' and a large empty text area for the message.

Figure No 1.5: Adding Guest User

After adding a guest user, roles were being assigned to the user through the “Directory role” option.

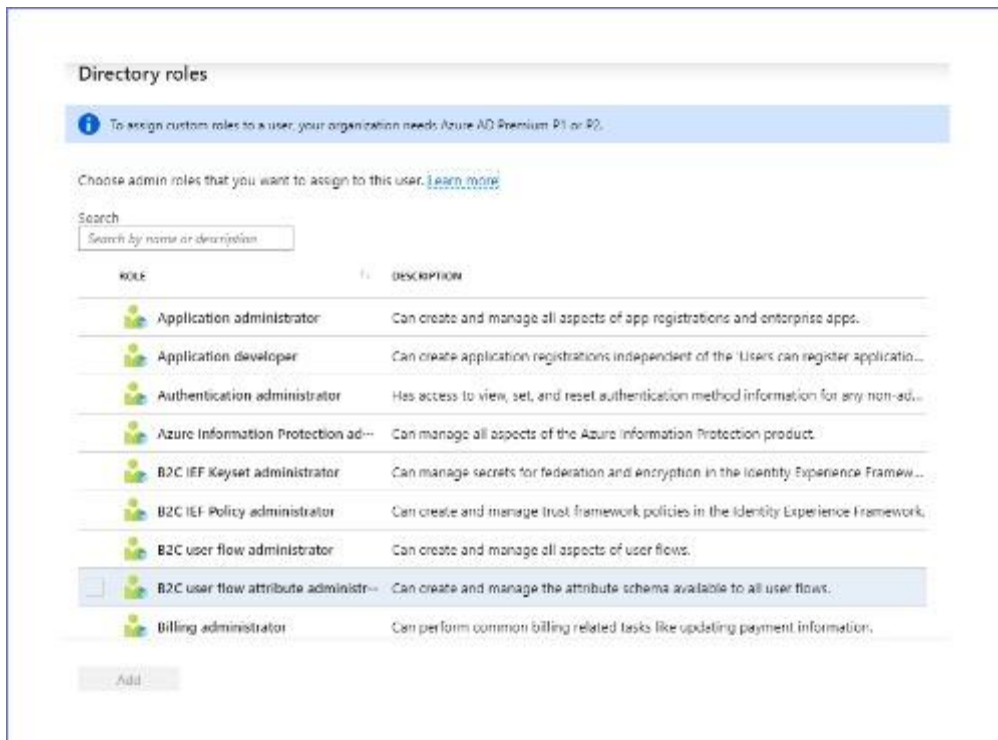


Figure No 1.6: Directory Role Assignment

We also created groups and added members into that group using the Azure Portal. Using groups, we can assign the same role to all the members of that group.

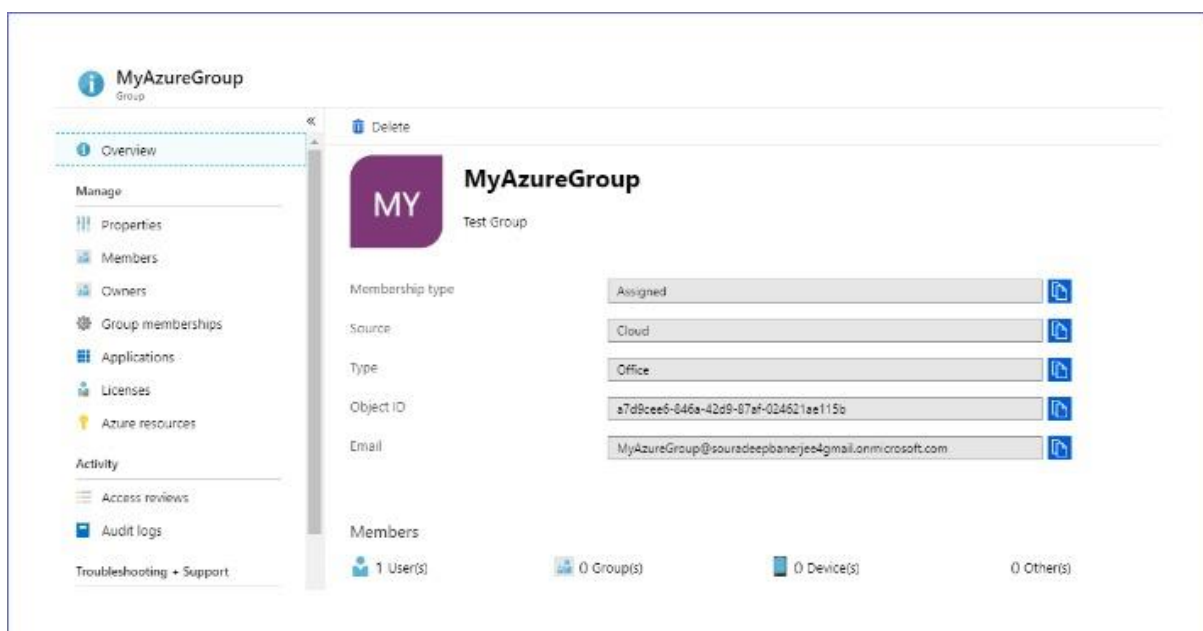


Figure No 1.7: Groups in Azure

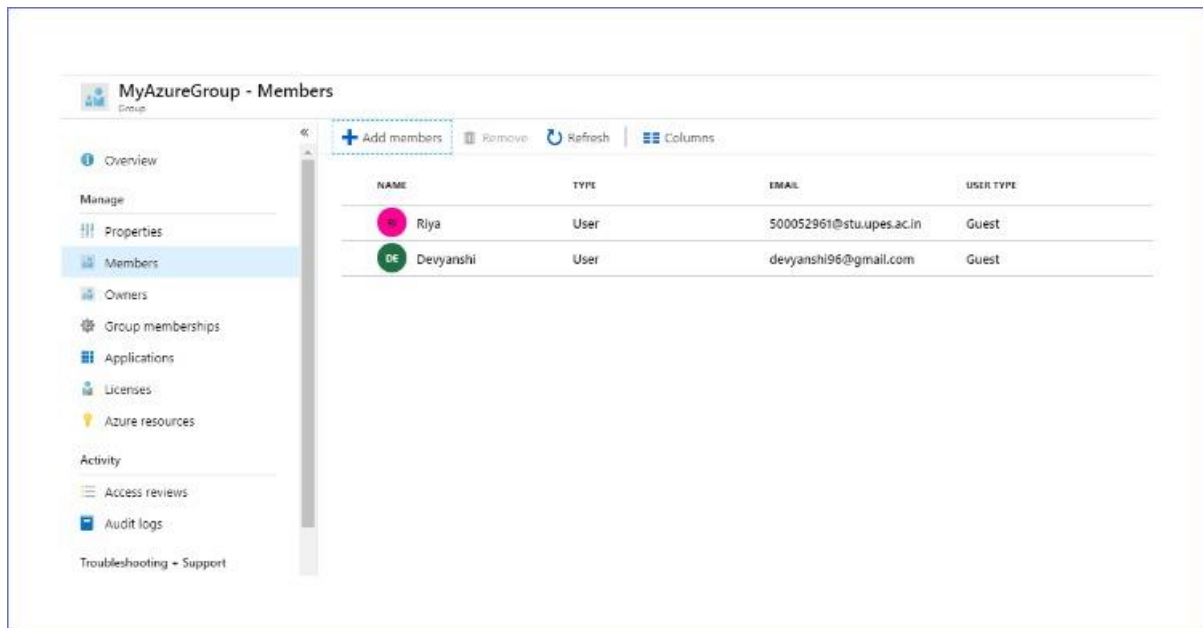


Figure No 1.8: Members in a Group

STORAGE ACCOUNT IN AZURE

A storage account in azure contains all the Azure storage objects like blobs, files, queues, tables and disks. The storage account name provides a unique namespace for Azure Storage data. It is accessible anywhere from the world through HTTP or HTTPS. It restricts Input and Output operations.

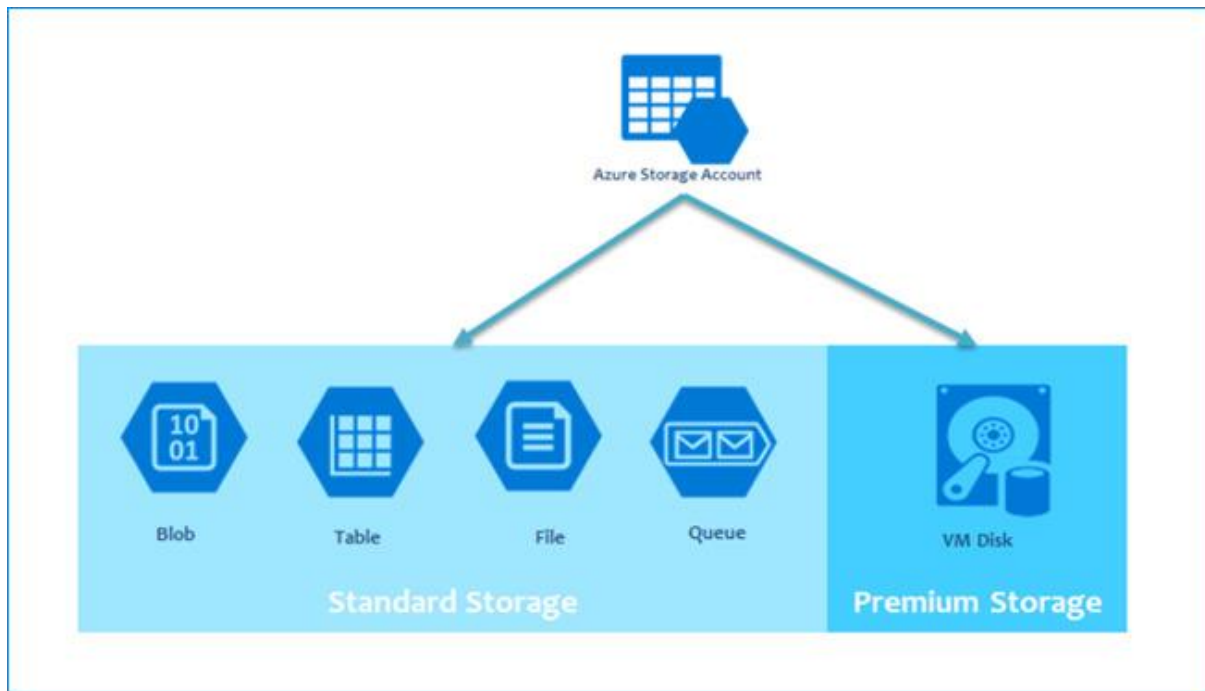


Figure No. 2.1: Storage Account

Types of Storage Accounts

Azure provides a wide range of storage accounts and each of them have their own features and pricing model. The storage account types are as follows:

1. **General-purpose V2 accounts:** These are the basic storage accounts for blobs, files, queues and tables.
2. **General-purpose V1 accounts:** These are legacy account types used for blobs, files, queues, and tables. General-purpose V1 is advised to use if possible.
3. **Block blob storage accounts:** These are blob only storage accounts with premium performance. These storage accounts are being used for scenarios with higher transaction rates, using smaller objects or requiring consistently low storage latency.
4. **FileStorage storage accounts:** These are storage accounts for files-only storage with premium characteristics. These are being recommended for enterprise or high-performance scale applications.

5. **Blob storage accounts:** These are Blob-only storage accounts. It is preferred to use general-purpose V2 accounts if possible.

Access Tiers

Azure provides different kinds of access to blob storage data which vary based on usage patterns. They also differ in pricing. The various access tiers available are:

1. **Hot Access:** This tier is optimized for frequent access of objects in the storage account. Accessing data in hot access tier is most cost effective while storage cost is the highest.
2. **Cool Access:** This tier is optimized for storing large amounts of data that is infrequently accessed and stored for at least 30 days. Storing data in the cool tier is the most cost effective while accessing data in col tier may be more costly than hot tier.
3. **Archive:** This tier is only available for individual block blobs. It is optimized for data that can tolerate hours of retrieval latency and remain in this tier for 180 days at least. It is the most cost-effective option for storing data but accessing data in archive tier is the costliest.

Replication

Replication options for a storage account as provided by azure are as follows:

1. **Locally-redundant storage (LRS):** This is a simple low-cost redundant strategy being followed by azure for the storage accounts. Data is being replicated within a single unit.
2. **Zone-redundant storage (ZRS):** This is a replication strategy which is being followed for high availability and durability. Data is being replicated synchronously across three availability zones.
3. **Geo-redundant storage (GRS):** This is a cross region replication strategy followed by Azure in order to avoid region-wide unavailability.

4. **Read-access geo-redundant (RA-GRS):** This is a cross region replication strategy with read access to the replica.

Factors Affecting Cost

The different factors involved in the cost determination for creating a storage account are as follows:

1. **Region:** The region where the account is based, the geographical location.
2. **Account type:** It refers to the type of storage account being used.
3. **Access tier:** It refers to the tier of access i.e. hot or cool or archived, which one is required.
4. **Storage Capacity:** The storage capacity defines the amount of storage to be allotted for the storage account.
5. **Replication:** It refers to the replication plan being used in-order to follow for the storage account.
6. **Transaction:** It refers to all the CRUD operations being performed on the Azure Storage.
7. **Data egress:** It refers to the amount of data being transferred outside the Azure storage zone. If the data in the storage account is being accessed by an application running in different location then there is a charge.

Work Done

A storage account is created by using the Microsoft azure portal and two containers are being created inside it. Two text files are being kept inside each of them respectively and then the access to the files is being made public.

Access to the files is being checked if they are accessible through url or not.

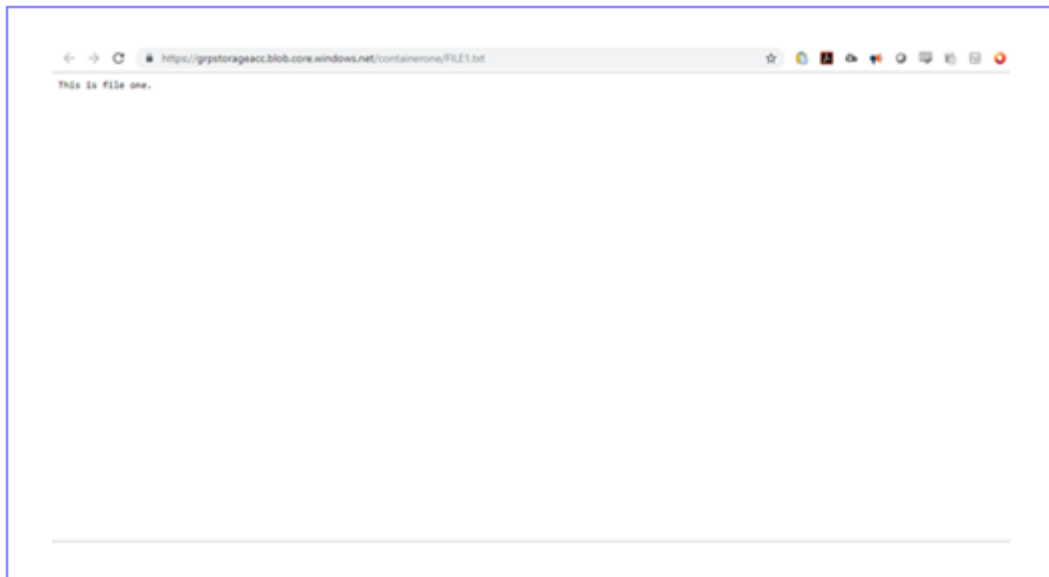


Figure 2.2: Checking File Access

Now after doing so, a vnet is being created and the same storage account is being assigned to it. Now when the files are being accessed using the same url, there is an error.

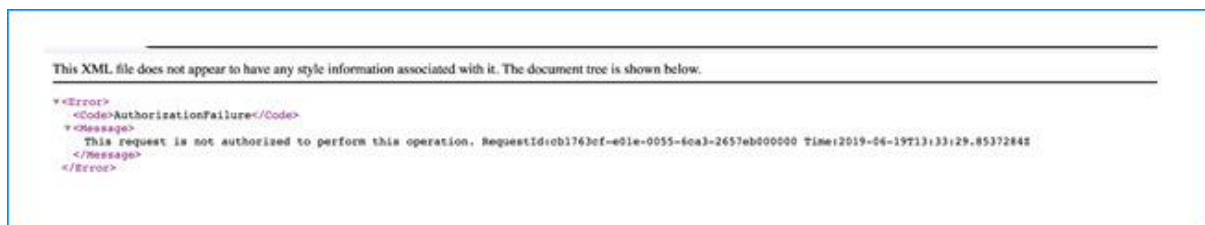


Figure 2.3: Error in accessing the container

Now when a virtual machine is created in the same virtual network and the file is being accessed through the browser in that virtual machine, the file is accessible.

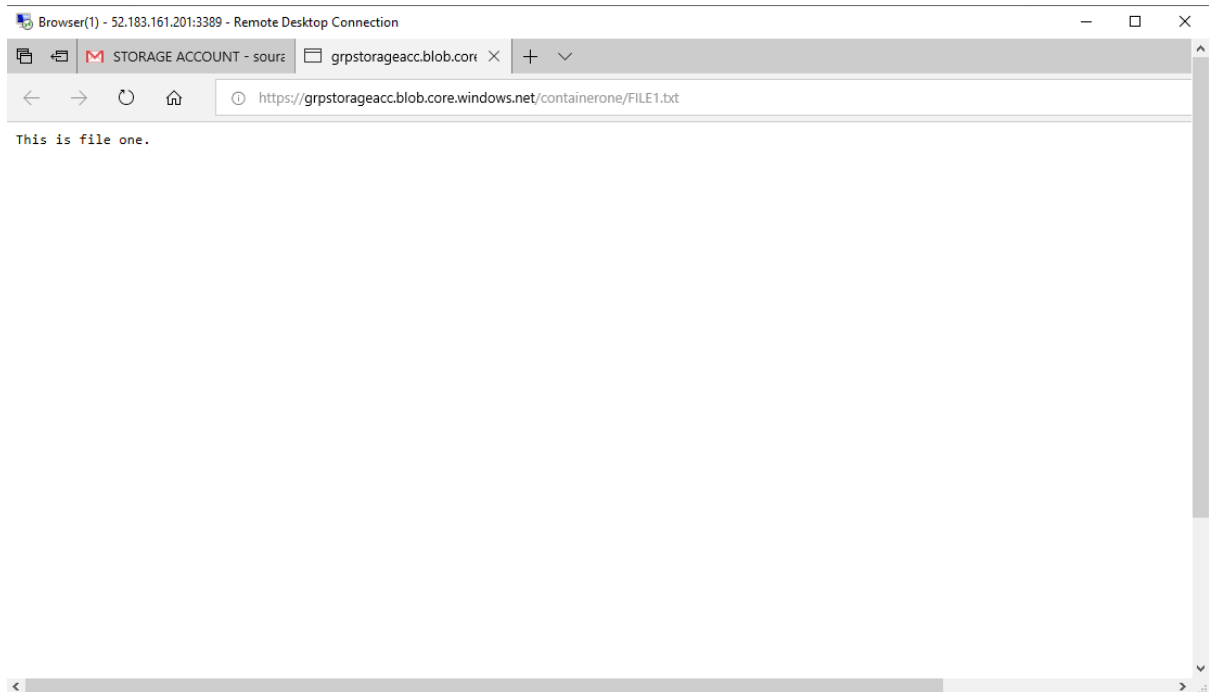


Figure 2.4: Accessing the file using a VM

VIRTUAL NETWORKS IN AZURE

Azure virtual network is the most fundamental building block for our private network in Azure. It helps different Azure resources such as Azure Virtual Machines. There are a few key concepts regarding the creation of a Virtual Network in Azure:

1. **Address Space:** While creating a virtual network we need to specify an address space in CIDR format. The resources inside this particular virtual network will be assigned private IPs within the specified range in address space.
2. **Subnets:** Segmentation of a virtual network is done by creating subnets in the virtual network. Each of the subnets must have different address spaces so as to avoid overlapping of IP addresses and must be a subset of the address space of the virtual network.
3. **Region:** It is the region where the virtual network is supposed to be created.
4. **Subscription:** We can create multiple virtual networks in a specific subscription.

Communication between Azure Resources

Azure resources communicate each-others using the following methods:

1. **Through a virtual network:** Resources such as Virtual Machines, Azure service end points, AKS and Azure Virtual Machine scale sets can communicate with other if they are inside the same virtual network.
2. **Through a virtual network service endpoint:** We can extend our virtual network identity and address space to Azure service resources such as Storage accounts and Azure SQL databases, over a direct connection.

3. **Through VNet Peering:** Communication between two separate virtual networks can be made possible with the help of peering. Resources in these networks can also communicate with other.

NETWORK SECURITY GROUPS

Network security groups are used to filter traffic in and out from the resources in a virtual network. They contain security rules which govern the flow of packets of data from one virtual network to other. Evaluation is done based on the 5 tuples of Network Security groups which are:

1. Source – Source address space from which the traffic is generating.
2. Source port – Source port numbers of the originating traffic.
3. Destination – Destination address space for the traffic of data.
4. Destination port – Destination port where the traffic is directed to.
5. Protocol – The protocol followed by the security rule which may be TCP or UDP or Any. We also have the option of advanced configuration for other protocols.

Work Done

We created a scenario in which we implemented the 3-tier web architecture consisting of Web, API and Database layers.

WEB-API-DATABASE architecture is a three-tier architecture which is the basic model for developing a web database application and communication is done using this 3-level application logic.

At the base of the model lies the database tier where data gets stored in a database server and various **CRUD** operations are being performed here. Above the database tier lies the middle tier which contains the application logic and does all the communication between the user and the database. Lastly on the top lies the web tier which is the web client that is used to interact with the user. User puts their request through the web tier, which gets processed by the middle tier and accordingly operations are being performed on the data base tier.

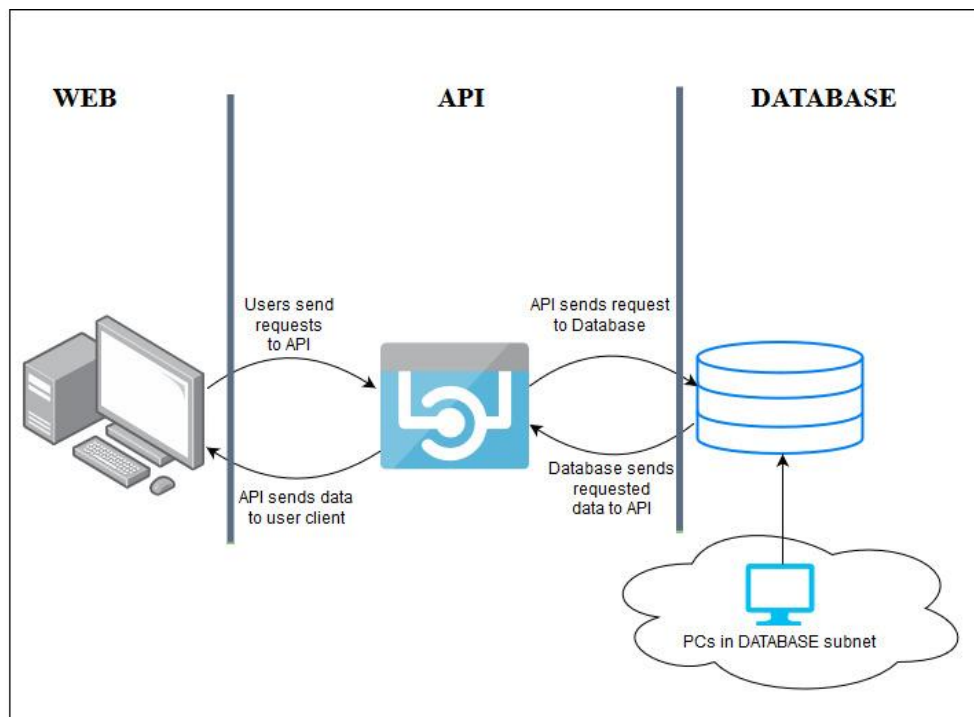


Figure No. 4.1: 3-Tier Architecture

We created three network security groups for the three subnets dedicated to Web, API and Database respectively- WebNSG, ApiNSG and DatabaseNSG.

WebNSG

In this network security group, we need to decide the inbound and outbound rules so that WEB subnet can only communicate with API subnet via port 80 and it should not be able to communicate with the DATABASE subnet directly (Figure No: 4.2).

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION	
100	HTTP	80,443	Any	Any	191.1.0.0/24	✓ Allow	...
65000	AllowVnetInBound	Any	Any	VirtualNetwo...	VirtualNetwo...	✓ Allow	...
65001	AllowAzureLoadBalancer1...	Any	Any	AzureLoadBa...	Any	✓ Allow	...
65500	DenyAllInBound	Any	Any	Any	Any	✗ Deny	...

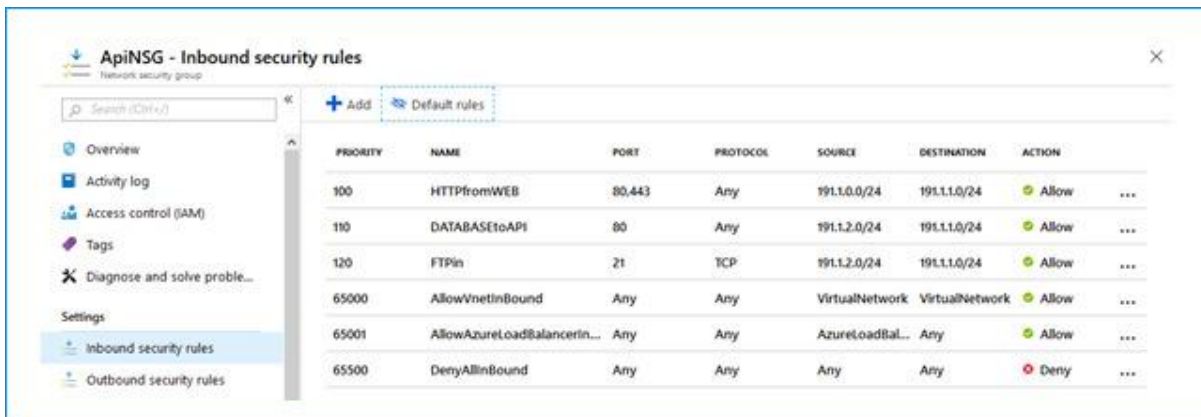
Outbound security rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION	
100	DenyCommToDatabase	Any	Any	Any	191.1.2.0/24	✗ Deny	...
110	WEBtoAPI	80,443	Any	191.1.0.0/24	191.1.1.0/24	✓ Allow	...
65000	AllowVnetOutBound	Any	Any	VirtualNetwo...	VirtualNetwo...	✓ Allow	...
65001	AllowInternetOutBound	Any	Any	Any	Internet	✓ Allow	...
65500	DenyAllOutBound	Any	Any	Any	Any	✗ Deny	...

Figure No 4.2: Security rules for WebNSG

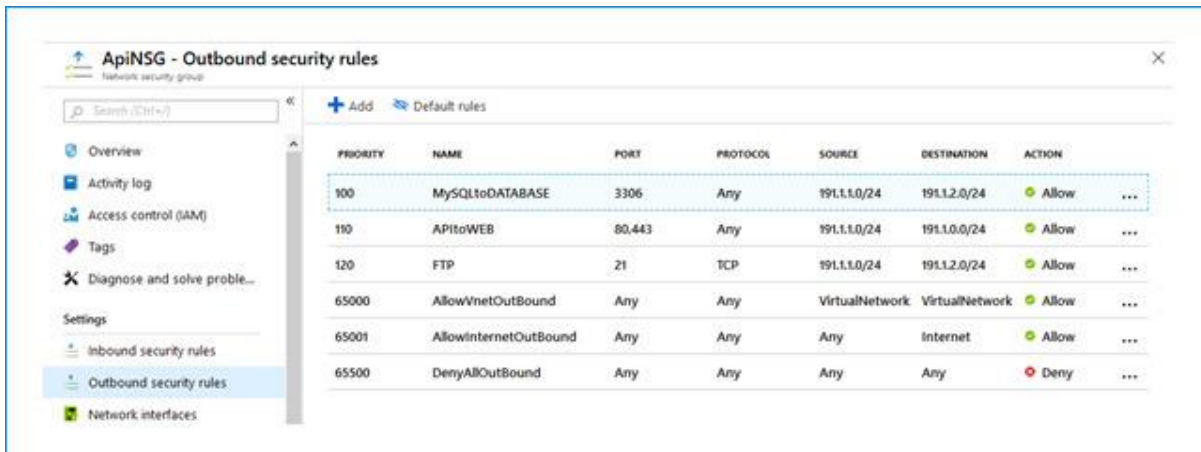
ApiNSG

The API subnet needs to communicate with both the WEB subnet and the DATABASE subnet and acts as a medium for communication between WEB and DATABASE subnet. So, in ApiNSG we need to apply the inbound and outbound rules such that it can do a both way communication with each of the subnets (Fig No - 4.3.1 & 4.3.2).



PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	HTTPfromWEB	80,443	Any	191.1.0.0/24	191.1.1.0/24	Allow
110	DATABASEtoAPI	80	Any	191.1.2.0/24	191.1.1.0/24	Allow
120	FTPin	21	TCP	191.1.2.0/24	191.1.1.0/24	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Figure No. 4.3.1: Inbound Security rules for ApiNSG

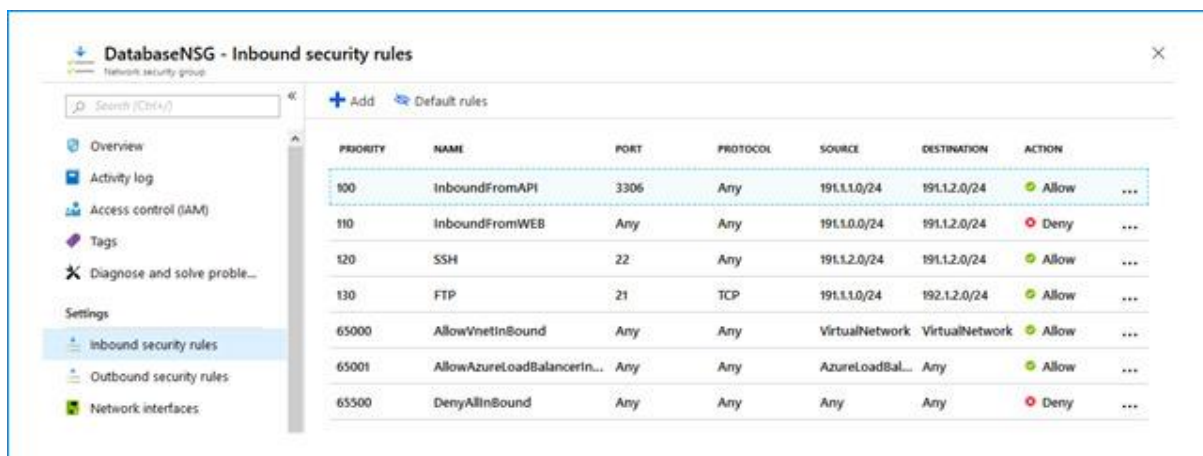


PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	MySQLtoDATABASE	3306	Any	191.1.1.0/24	191.1.2.0/24	Allow
110	APitoWEB	80,443	Any	191.1.1.0/24	191.1.0.0/24	Allow
120	FTP	21	TCP	191.1.1.0/24	191.1.2.0/24	Allow
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Figure No. 4.3.2: Outbound Security rules for ApiNSG

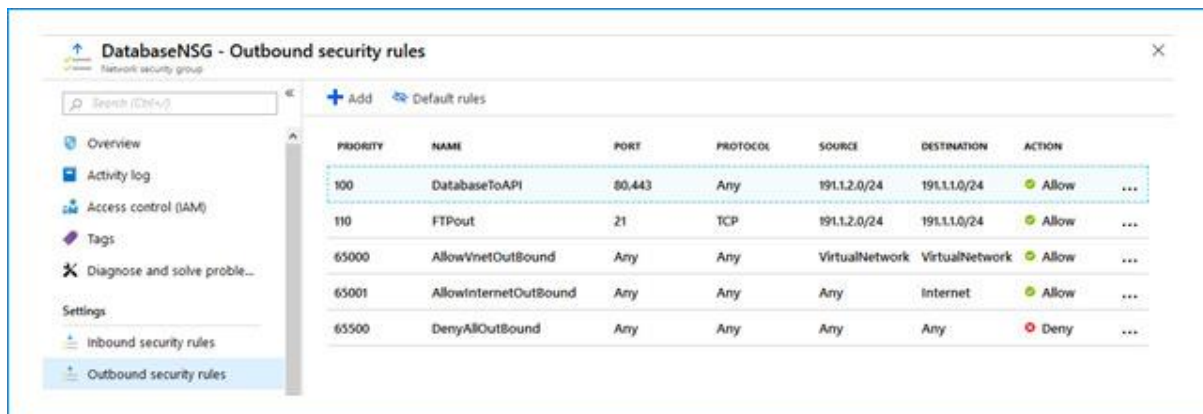
DatabaseNSG

DATABASE subnet is the most important part of the whole architecture. Data comes, gets stored and accessed from this subnet. So, we need to design the DatabaseNSG in such a way that it only communicates with the WEB subnet and denies any direct request from WEB subnet. We also need to take care of the fact that it enables SSH communication only from the IP range belonging to the same subnet (Fig No - 4.4.1 & 4.4.2).



PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	InboundFromAPI	3306	Any	191.1.1.0/24	191.1.2.0/24	Allow
110	InboundFromWEB	Any	Any	191.1.0.0/24	191.1.2.0/24	Deny
120	SSH	22	Any	191.1.2.0/24	191.1.2.0/24	Allow
130	FTP	21	TCP	191.1.1.0/24	192.1.2.0/24	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerIn...	Any	Any	AzureLoadBal...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Figure No. 4.4.1: Inbound Security rules for DatabaseNSG



PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
100	DatabaseToAPI	80,443	Any	191.1.2.0/24	191.1.1.0/24	Allow
110	FTPout	21	TCP	191.1.2.0/24	191.1.1.0/24	Allow
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Figure No 4.4.2: Outbound Security rules for DatabaseNSG

ARM – AZURE RESOURCE MANAGER TEMPLATES

ARM templates are being used in order to deploy infrastructure with the help of code. ARM templates are being designed in JSON language which can be understood by the ARM api. Prior to the release of ARM, the Azure Service Management api used to handle all kinds of deployments which were then termed as classic. This was being replaced by the ARM api.

The main benefits of ARM api is that we can deploy several units of infrastructures as a single unit. In ARM template user defines the resource and type of resource to deploy and the ARM api either creates a new object or modifies the existing one to get the required deployment.

Modes of Deployment

ARM templates either contains the whole resource group or some of the resources in a resource group. ARM templates can be deployed in two types of modes:

- **Incremental:** This mode is used to add resources to an existing resource group to which the template is being deployed. Benefit is that we don't lose any resources pre-existing the resource group but on the other hand we need to manually remove the resources that we don't need.
- **Complete:** This mode deletes any object i.e. resource that is not mentioned in the template from the resource group.

Execution of ARM templates

REST API plays an important role in the execution of ARM templates. There is this one set of REST API called "Resource Management" where we send an ARM template. The REST API does the following:

1. Parse the JSON file.

2. Filling the parameters that are being passed to the JSON template file.
3. Execution of the functions inside the ARM template.
4. Calling the REST API of the concerned resource that needs to be created.

Work Done

We have designed ARM templates for creating Azure Storage Accounts, Virtual Network, virtual network with subnets, peering between virtual networks and the 3-tier WEB-API-DATABASE architecture in which we create two vnets in different locations with 3 subnets each for web, api and database layers. We also create two sets of network security groups each consisting of 3 network security groups for the 2 vnets.

1. Vnet Template

In this template we have designed a deployment in which a vnet is created in a given location with a given name. It has a default subnet called “subnet1” and the address space for the vnet is also being given by the user.



Figure No. 5.1: Template for Vnet Creation

2. Storage Account Template

In this template we have designed a deployment in which we create a storage account of name given by the user in a desired location, with the desired account type which is the replication, access tier and kind of storage account needed.

```
"resources": [  
  {  
    "name": "[parameters('storageAccountName')]",  
    "type": "Microsoft.Storage/storageAccounts",  
    "apiVersion": "2018-07-01",  
    "location": "[parameters('location')]",  
    "properties": {  
      "accessTier": "[parameters('accessTier')]"  
    },  
    "dependsOn": [],  
    "sku": {  
      "name": "[parameters('accountType')]"  
    },  
    "kind": "[parameters('kind')]"  
  }  
],
```

Figure No. 5.2: Template for Storage Account Creation

3. Vnet Peering

Vnet peering is done in order to enable communication between two vnets in Azure. The vnets may be in same subscription or in different subscription. The resources in in one vnet can communicate with the resources of other vnet using private IP address.

In this template we have designed the deployment of vnet peering between two vnets being created earlier.

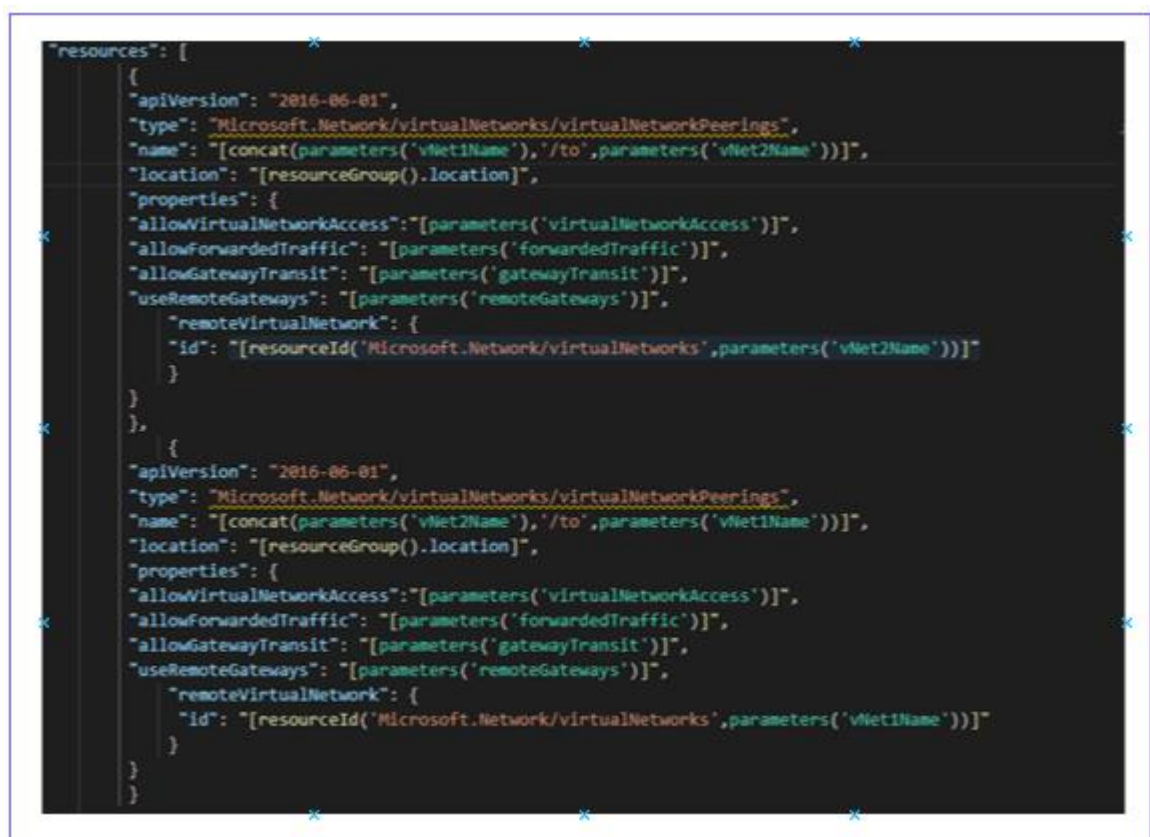


Figure No. 5.3: Template for Vnet Peering

4. Nested Template for deploying 3-tier architecture

This is the implementation of 3 tier WEB-API-DATABASE architecture in which we created a master template to call other sub templates. The master template contains the link to the sub templates and the deployment

names for the resources. We can also add dependencies by using the dependsOn feature to do sequential deployment.

First of all we call the network security group template two times in order to create the two sets of network security groups in two different locations- West US and East US.

Sub template link- <https://github.com/Souradeep2304/Azure-Templates/blob/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/NSGcreation.json>

```
{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "L21",
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/Souradeep2304/Azure-Templates/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/NSGcreation.json",
      "contentVersion": "1.0.0.0"
    },
    "parameters": {
      "location": {"value": "[parameters('location1')]"},
      "networkSecurityGroup1": {"value": "[parameters('firstNetworkSecurityGroupInFirstLocaion')]"},
      "networkSecurityGroup2": {"value": "[parameters('secondNetworkSecurityGroupInFirstLocaion')]"},
      "networkSecurityGroup3": {"value": "[parameters('thirdNetworkSecurityGroupInFirstLocaion')]"},
      "subnetAddressPrefix1": {"value": "[parameters('subnetAddressPrefix1')]"},
      "subnetAddressPrefix2": {"value": "[parameters('subnetAddressPrefix2')]"},
      "subnetAddressPrefix3": {"value": "[parameters('subnetAddressPrefix3')]"},
    }
  }
},
```

Figure No 5.4: Deployment of first set of NSG

```
{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "L2",
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/Souradeep2304/Azure-Templates/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/NSGcreation.json",
      "contentVersion": "1.0.0.0"
    },
    "parameters": {
      "location": {"value": "[parameters('location2')]"},
      "networkSecurityGroup1": {"value": "[parameters('firstNetworkSecurityGroupInSecondLocaion')]"},
      "networkSecurityGroup2": {"value": "[parameters('secondNetworkSecurityGroupInSecondLocaion')]"},
      "networkSecurityGroup3": {"value": "[parameters('thirdNetworkSecurityGroupInSecondLocaion')]"},
      "subnetAddressPrefix1": {"value": "[parameters('subnet1AddressPrefixOfSecondVnet')]"},
      "subnetAddressPrefix2": {"value": "[parameters('subnet2AddressPrefixOfSecondVnet')]"},
      "subnetAddressPrefix3": {"value": "[parameters('subnet3AddressPrefixOfSecondVnet')]"},
    }
  }
},
```

Figure No 5.5: Deployment of second set of NSG

After the deployment of Network security groups in both the locations we go for the deployment of the two vnets in two different locations- West US and East US. Each of these vnets have 3 subnets each, one for web layer one for database layer and one for api layer. The respective network security groups are being attached to these subnets.

Sub Template link- <https://github.com/Souradeep2304/Azure-Templates/blob/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/Vnet.json>

```
{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "L1",
  "dependsOn": ["L2", "L21"],

  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/Souradeep2304/Azure-Templates/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/Vnet.json",
      "contentVersion": "1.0.0.0"
    },
    "parameters": {
      "vnet": {"value": "[parameters('vNet1')]"},
      "location": {"value": "[parameters('location1')]"},
      "networkSecurityGroup1": {"value": "[parameters('firstNetworkSecurityGroupInFirstLocaion')]"},
      "networkSecurityGroup2": {"value": "[parameters('secondNetworkSecurityGroupInFirstLocaion')]"},
      "networkSecurityGroup3": {"value": "[parameters('thirdNetworkSecurityGroupInFirstLocaion')]"},
      "subnetName1": {"value": "[parameters('subnetName1')]"},
      "subnetName2": {"value": "[parameters('subnetName2')]"},
      "subnetName3": {"value": "[parameters('subnetName3')]"},
      "subnetAddressPrefix1": {"value": "[parameters('subnetAddressPrefix1')]"},
      "subnetAddressPrefix2": {"value": "[parameters('subnetAddressPrefix2')]"},
      "subnetAddressPrefix3": {"value": "[parameters('subnetAddressPrefix3')]"},
      "addressPrefix": {"value": "[parameters('addressPrefix1')]"},
      "firewallSubnet": {"value": "[parameters('firewallSubnetOfVnet1')]"},
      "enableDdosProtection": {"value": "[parameters('enableDdosProtection')]" }
    }
  }
},
```

Figure No 5.6: Deployment of first Vnet

```
{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "L3",
  "dependsOn": ["L2", "L21"],
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/Souradeep2304/Azure-Templates/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/Vnet.json",
      "contentVersion": "1.0.0.0"
    },
    "parameters": {
      "vnet": {"value": "[parameters('vNet2')]"},
      "addressPrefix": {"value": "[parameters('addressPrefixOfSecondVnet')]"},
      "location": {"value": "[parameters('location2')]"},
      "subnetName1": {"value": "[parameters('subnet1NameOfSecondVnet')]"},
      "subnetName2": {"value": "[parameters('subnet2NameOfSecondVnet')]"},
      "subnetName3": {"value": "[parameters('subnet3NameOfSecondVnet')]"},
      "networkSecurityGroup1": {"value": "[parameters('firstNetworkSecurityGroupInSecondLocaion')]"},
      "networkSecurityGroup2": {"value": "[parameters('secondNetworkSecurityGroupInSecondLocaion')]"},
      "networkSecurityGroup3": {"value": "[parameters('thirdNetworkSecurityGroupInSecondLocaion')]"},
      "subnetAddressPrefix1": {"value": "[parameters('subnet1AddressPrefixOfSecondVnet')]"},
      "subnetAddressPrefix2": {"value": "[parameters('subnet2AddressPrefixOfSecondVnet')]"},
      "subnetAddressPrefix3": {"value": "[parameters('subnet3AddressPrefixOfSecondVnet')]"},
      "firewallSubnet": {"value": "[parameters('firewallSubnetOfSecondVnet')]"},
      "enableDdosProtection": {"value": "[parameters('enableDdosProtection')]" }
    }
  }
},
```

Figure No 5.7: Deployment of second Vnet

After the vnets are being deployed, we deploy the peering between these two vnets by calling the template file.

Sub Template link- <https://github.com/Souradeep2304/Azure-Templates/blob/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/peering.json>

```
{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "L4",
  "dependsOn": ["L3", "L1"],
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/Souradeep2304/Azure-Templates/master/Nested%20Templates/Creation%20of%20Two%20Vnets%20and%20peering/Subtemplates/peering.json",
      "contentVersion": "1.0.0.0"
    },
    "parameters": {
      "vNet1Name": {"value": "[parameters('vNet1')]"},
      "vNet2Name": {"value": "[parameters('vNet2')]"},
      "virtualNetworkAccess": {"value": "[parameters('virtualNetworkAccess')]"},
      "forwardedTraffic": {"value": "[parameters('forwardedTraffic')]"},
      "gatewayTransit": {"value": "[parameters('gatewayTransit')]"},
      "RemoteGateways": {"value": "[parameters('remoteGateways')]" }
    }
  }
}
```

Figure No 5.8: Deployment of Peering between the Vnets

TERRAFORM

Terraform is a tool, developed by HashiCorp, that is being used to deploy infrastructure safely and easily via code. It makes the use of configuration files in order to run applications on premises. It makes use of the concept of ‘desired state’. It compares the existing state with the desired state and then does the required deployments.

It mainly does the implementation of what is known as **Infrastructure as a Code** using high level configuration syntax. It introduces the idea of re-usability and repeatability.

Execution of Plan

Terraform uses the concept of “Plan” where it shows us the execution plan of the current deployment before being it is actually deployed. This helps us to

notice if the deployment is actually according to our needs before terraform starts to manipulate the infrastructure.

Configuration

Before using terraform, we need to configure the same with a service principal, A service principal is a type of security principal that represents an application. Service principal provides an identity for our application, allowing the application to do the necessary deployments.

```
{
  "appId": "160ed74f-3a03-45bc-bf69-ecfbbf4040cb",
  "displayName": "azure-cli-2019-07-09-08-44-51",
  "name": "http://azure-cli-2019-07-09-08-44-51",
  "password": "10993ad1-7dad-4da6-9330-7844ce97c490",
  "tenant": "eb94a06c-59cb-4aa9-9ed9-cc1ef939066a"
}
```

Figure No 6.1: Service Principal created using Azure CLI

Using the following script, we can assign the value to terraform for completing the configuration.

```
#!/bin/sh
echo "Setting environment variables for Terraform"
export ARM_SUBSCRIPTION_ID=your_subscription_id
export ARM_CLIENT_ID=your_appId
export ARM_CLIENT_SECRET=your_password
export ARM_TENANT_ID=your_tenant_id

# Not needed for public, required for usgovernment, german, china
export ARM_ENVIRONMENT=public
```

Figure No 6.2: Configuring Terraform

Work Done

Using terraform we deployed the 3-tier web-api-database architecture, which we previously did with ARM templates.

There are two providers in case of terraform which are azure and azurerm. In this case we are going to use azurerm. After configuring terraform with azure we code the .tf file for deployment.

For creating a resource group, we will use `azurerm_resource_group` for resource group deployment. We will use an existing resource group named TerraformRG for this scenario. We will import the same into the configuration file using the command `terraform import`. If the resource group is not already created then it gets created in the process.

```
resource "azurerm_resource_group" "tera" {  
  name="TerraformRG"  
  location="East US"  
}
```

Figure No 6.3: Deployment of Resource Group

After that we create all the necessary network security groups using `azurerm_network_security_group`. We give the names of the security groups and also the location of the security groups. We also pass the names of the resource group in which the resource is to be deployed.

```
resource "azurerm_network_security_group" "nsg1" {  
  name          = "W1"  
  location      = "East US"  
  resource_group_name = "${azurerm_resource_group.tera.name}"  
}
```

Figure 6.4: Deployment of Security Groups.

Now we need to add required security rules for the security groups that we created in the previous step. For this we are going to use `azurerm_network_security_rule`. We pass the name of the rules and other

important parameters such as source_address_prefix, destination_address_prefixes, priority, access, etc.

```
resource "azurerm_network_security_rule" "rule1" {
  name= "HTTPorHTTPS"
  network_security_group_name = "${azurerm_network_security_group.nsg1.name}"
  direction= "Inbound"

  access= "Allow"
  priority= 100
  source_address_prefix = "*"
  source_port_range = "*"
  destination_address_prefixes= ["13.0.0.0/24"]
  destination_port_ranges = ["80","443"]
  protocol = "Tcp"
  resource_group_name= "${azurerm_resource_group.tera.name}"
}
```

Figure No 6.5: Adding Security rules to NSGs

After adding all the necessary rules, we create the virtual networks in different locations. For creation of virtual networks, we use the module `azurerm_virtual_network`. We pass the name of the virtual network along with the location, resource group and address space of the virtual network along with 3 subnets.


```

resource "azurerm_virtual_network" "vnet1" {
  name                = "V1"
  location            = "East US"
  resource_group_name = "${azurerm_resource_group.tera.name}"
  address_space       = ["13.0.0.0/16"]

  subnet {
    name                = "S1"
    address_prefix      = "13.0.0.0/24"
    security_group      = "${azurerm_network_security_group.nsg1.id}"
  }
  subnet {
    name                = "S2"
    address_prefix      = "13.0.1.0/24"
    security_group      = "${azurerm_network_security_group.nsg2.id}"
  }
  subnet {
    name                = "S3"
    address_prefix      = "13.0.2.0/24"
    security_group      = "${azurerm_network_security_group.nsg3.id}"
  }
}

```

Figure No 6.6: Creation of Virtual Networks

Finally, after the creation of the virtual networks we go for the deployment of virtual network peering between these two virtual networks. For this we use `azurerm_virtual_network_peering`. It takes the remote virtual network id and the origin virtual network name for creation of peering.

```

resource "azurerm_virtual_network_peering" "peer1" {
  name                = "V1toV2"
  resource_group_name = "${azurerm_resource_group.tera.name}"
  virtual_network_name = "${azurerm_virtual_network.vnet1.name}"
  remote_virtual_network_id = "${azurerm_virtual_network.vnet2.id}"
}

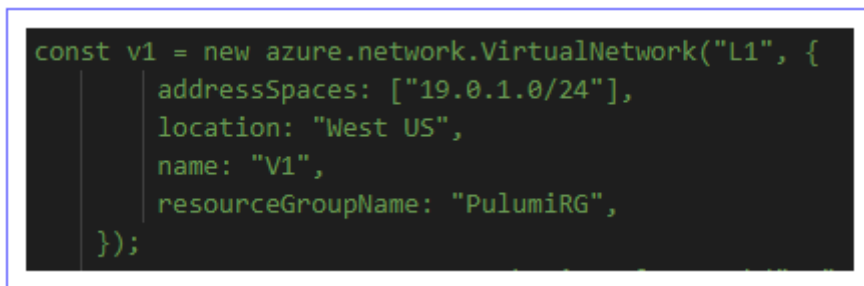
resource "azurerm_virtual_network_peering" "peer2" {
  name                = "V2toV1"
  resource_group_name = "${azurerm_resource_group.tera.name}"
  virtual_network_name = "${azurerm_virtual_network.vnet2.name}"
  remote_virtual_network_id = "${azurerm_virtual_network.vnet1.id}"
}

```

Figure No 6.7: Virtual Network Peering

PULUMI - The New IaC Tool

Pulumi is an open source platform for building and deploying infrastructure in different languages such as JavaScript, Python, Go or typescript and on multiple cloud platforms like Microsoft Azure, Amazon Web Services, Google Cloud Platform and Kubernetes.



```
const v1 = new azure.network.VirtualNetwork("L1", {  
  addressSpaces: ["19.0.1.0/24"],  
  location: "West US",  
  name: "V1",  
  resourceGroupName: "PulumiRG",  
});
```

Figure No 7.1: Example Code Snippet for Pulumi

Benefits

Benefits of using these languages gives us the advantage of:

1. **Familiarity:** We do not need to learn any kind of separate YAML templating language or any other Domain-Specific Language (DSL) to write code in Pulumi.
2. **Abstraction:** Pulumi provides a kind of abstraction layer, like we can create files for deploying different kinds of resources and then use them to create something big. We can hide the implementation details of the smaller resources.
3. **Sharing and Reuse:** We can use existing packages in the community to do the deployments we require or share our code with others so that they can use the same. This enables code re-usability and repeatability.
4. **Full Control:** We can apply the full control of the programming language in Pulumi. Loops, conditions can be used in Pulumi to deploy infrastructure.

Work Done

Using Pulumi we deployed the same 3 tier architecture that we deployed through ARM templates. In this case we created separate modules for separate resources.

First, we created a JavaScript class file V-NET.js which is used to create a vnet with 3 subnets and attach the network security groups to it. It takes the address space for the subnets and the virtual network and the network security group ids as parameters.

```
"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");

class VNET extends pulumi.ComponentResource {
  constructor(vnetname, location, rg, vspace, nsg1, nsg2, nsg3, s1, s2, s3, i) {
    super(vnetname, location, rg, vspace, nsg1, nsg2, nsg3, s1, s2, s3, i);
    const vnet = new azure.network.VirtualNetwork("N"+i, {
      addressSpaces: [vspace],

      location: location,
      name: vnetname,
      resourceGroupName: rg,
      subnets: [
        {
          addressPrefix: s1,
          name: "S1",
          securityGroup: nsg1,
        },
        {
          addressPrefix: s2,
          name: "S2",
          securityGroup: nsg2,
        },
        {
          addressPrefix: s3,
          name: "S3",
          securityGroup: nsg3,
        },
      ],
      tags: {
        environment: "Public",
      },
    }, {
      parent: this
    });
    this.vnetid=vnet.id;
    this.vname=vnet.name;
  }
}

module.exports.VNET = VNET;
```

Figure No 7.2: V-NET.js

Now we create the class file for enabling peering between two different vnets. Id and names of the vnets are being passed as parameters to the class file.

```
"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");

class peering extends pulumi.ComponentResource {

  constructor(pname, v2, v1, rg, i) {
    super(pname, v1, v2, rg, i);
    const peering = new azure.network.VirtualNetworkPeering("P"+i, {
      name: pname,
      remoteVirtualNetworkId: v2,
      resourceGroupName: rg,
      virtualNetworkName: v1,
    },
    {
      parent: this
    });
  }
}

module.exports.peering = peering;
```

Figure No 7.3: peering.js

Peering is being enabled between the two virtual networks so that the resources in those virtual networks can communicate with other using private IP addresses.

In the next part we have to create separate class files for the three types of network security groups for subnets representing web, api and database layers. Each of these class files will take in the name of the network security group, resource group name, location and address space of the subnets as parameters.

```

"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");

class NSG extends pulumi.ComponentResource {
  constructor(location, rg, nsgname, i, s1, s2, s3) {
    super(location, nsgname, rg, i, s1, s2, s3);
    const nsg = new azure.network.NetworkSecurityGroup("L"+i, {
      location: location,
      name: nsgname,
      resourceGroupName: rg,
      securityRules: [{ ...
    },
    {
      parent: this
    }
  });
  this.nsgid=nsg.id;
}
}

module.exports.NSG=NSG;

```

Figure No 7.4; apiNSG.js

```

"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");

class NSG extends pulumi.ComponentResource {
  constructor(location, rg, nsgname, i, s1, s2, s3) {
    super(location, rg, nsgname, i, s1, s2, s3);
    const nsg = new azure.network.NetworkSecurityGroup("L"+i, {
      location: location,
      name: nsgname,
      resourceGroupName: rg,
      securityRules: [{ ...
    },
    {
      parent: this
    }
  });
  this.nsgid=nsg.id;
}
}

module.exports.NSG=NSG;

```

Figure No 7.5: webNSG.js

```

"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");

class NSG extends pulumi.ComponentResource{
  constructor(location,rg,nsgname,i,s1,s2,s3){
    super(location,nsgname,rg,i,s1,s2,s3);
    const nsg = new azure.network.NetworkSecurityGroup("L"+i, {
      location: location,
      name: nsgname,
      resourceGroupName: rg,
      securityRules: [{
        },
        {
          parent: this
        }
      ]
    });
    this.nsgid=nsg.id;
  }
}

module.exports.NSG=NSG;

```

Figure No 7.6: dbNSG.js

Finally, we create the main script file, main.js which will import all the predefined class files and use them to deploy the three-tier architecture as defined.

```

const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");
const vnet = require("./V-NET.js");
const peer=require("./peering.js");
const web=require("./webNSG.js");
const api=require("./apiNSG.js");
const db=require("./dbNSG.js");

const nsg1=new web.NSG("East US","PulumiRG","K1",1,"10.0.0.0/24","10.0.1.0/24","10.0.2.0/24");
const nsg2=new api.NSG("East US","PulumiRG","K2",2,"10.0.0.0/24","10.0.1.0/24","10.0.2.0/24");
const nsg3=new db.NSG("East US","PulumiRG","K3",3,"10.0.0.0/24","10.0.1.0/24","10.0.2.0/24");
const nsg4=new web.NSG("West US","PulumiRG","K4",4,"15.0.0.0/24","15.0.1.0/24","15.0.2.0/24");
const nsg5=new api.NSG("West US","PulumiRG","K5",5,"15.0.0.0/24","15.0.1.0/24","15.0.2.0/24");
const nsg6=new db.NSG("West US","PulumiRG","K6",6,"15.0.0.0/24","15.0.1.0/24","15.0.2.0/24");

const nsg1id=nsg1.nsgid;
const nsg2id=nsg2.nsgid;
const nsg3id=nsg3.nsgid;
const nsg4id=nsg4.nsgid;
const nsg5id=nsg5.nsgid;
const nsg6id=nsg6.nsgid;

let v1=new vnet.VNET("V1","East US","PulumiRG","10.0.0.0/16",nsg1id,nsg2id,nsg3id,"10.0.0.0/24","10.0.1.0/24","10.0.2.0/24",1);
let v2=new vnet.VNET("V2","West US","PulumiRG","15.0.0.0/16",nsg4id,nsg5id,nsg6id,"15.0.0.0/24","15.0.1.0/24","15.0.2.0/24",2);

const v1id=v1.vnetid;
const v2id=v2.vnetid;
const v1name=v1.vname;
const v2name=v2.vname;

const p1= new peer.peering("V1toV2",v2id,v1name,"PulumiRG",1);
const p2= new peer.peering("V2toV1",v1id,v2name,"PulumiRG",2);

```

Figure No. 7.4: main.js

REFERENCES

Following are the references that I used while doing this project:

1. <https://www.bmc.com/blogs/software-defined-infrastructure/>
2. https://en.wikipedia.org/wiki/Software-defined_infrastructure
3. <https://searchmicroservices.techtarget.com/definition/RESTful-API>
4. <https://www.atlassian.com/devops>
5. <https://www.red-gate.com/simple-talk/cloud/infrastructure-as-a-service/azure-resource-manager-arm-templates/>
6. <https://github.com/Azure/azure-quickstart-templates>
7. <https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction>
8. <https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-faq>
9. <https://www.pulumi.com>
10. <https://www.terraform.io/docs/index.html>