

# **INFRASTRUCTURE as CODE**

**A Project Report**

*Submitted by:*

**RIYA AGRAHARI (R110216128)**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

at



**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

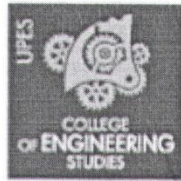
**Dehradun-248007 2019-20**

*under the guidance of*

**Mr. Manish Kumar**

**Associate Architect**

**Applied Information Sciences**



The innovation driven  
E-School

## CANDIDATE'S DECLARATION

I hereby certify that the project work entitled "**Infrastructure as 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 **6<sup>th</sup> June,2019** to **19<sup>th</sup> 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.

(Riya Agrahari)  
(R110216128)

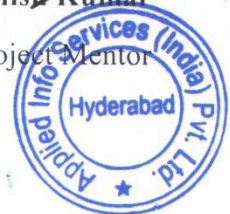
This is to certify that the statement made by the candidate is correct to the best of my knowledge.

Date: 19-Jul - 2019

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)




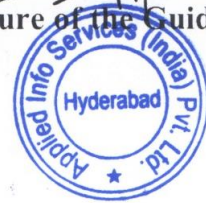
## CERTIFICATE

This is to certify that the project titled “**Infrastructure as Code**” is the bona fide work carried out by Riya Agrahari, 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

  
19/07/19  
Signature of the Guide



## Table of Contents

<b>Sl. No.</b>	<b>Topic</b>	<b>Pg. No.</b>
i	Title Page	1
ii	Declaration of the Student	2
iii	Certificate of the Guide	3
iv	List of Figures	6
v	Abstract	7
vi	Acknowledgement	8
1.	Introduction	9
1.1	Agile Software Development	9
1.2	Software Defined Infrastructure	9
1.3	Infrastructure as Code (IaC) implements Software Defined	9
1.4	Benefits of Infrastructure As Code	10
1.5	DevOps	11
1.6	What is Microsoft Azure?	11
1.6.1	Azure Active Directory (Azure AD)	11
1.6.2	Azure Subscription	12
1.6.3	Azure Resource Group and Azure Resources	13
1.6.4	Azure Resource Manager	14
1.7	Roles in Azure	14
1.7.1	Classic Subscription Administrator Roles	14
1.7.2	Azure Roles-Based Access Control (RBAC)	14
1.7.3	Azure AD Administrator Roles	15
1.8	Storage in Azure	15
1.8.1	Major categories of Storage System in Azure	15
1.8.2	Types of storage accounts in Azure	16

1.8.3	Categories of redundant storage	16
1.9	Infrastructure as Code in Azure	17
1.9.1	ARM Template	17
1.9.2	Terraform	18
1.9.3	Pulumi	18
2.	Problem Definition	18
3.	Objective	19
4.	Project Overview	19
5.	Implementation	26
5.1	Creating and assigning roles to users in Azure Active Directory	26
5.2	Understanding Storage Account through Azure CLI	27
5.3	Understanding Storage Access along with Virtual Network and Network Security Group Rules	30
5.4	Infrastructure As Code For Three-Tier Architecture (WEB-API-DB)	33
5.4.1	Implemented Through ARM Templates	33
5.4.2	Implemented Through Terraform	40
5.4.3	Implemented Through Pulumi	43
6.	Results	46
7.	Future Scope	49
8.	References	49

## **List of Figures**

<b>Sl. No.</b>	<b>Figures</b>	<b>Pg.No.</b>
1.	Single Subscription in an Azure AD	12
2.	Multiple Subscriptions in an Azure AD	13
3.	Hierarchical Structure of Various Components in Azure	13
4.	Three Tier Architecture	20
5.	Detailed Architectural Diagram for Primary VNet	24
6.	Architectural Diagram for Disaster Recovery - Primary and Secondary Vnet	25

## **ABSTRACT**

Configuring and setting up infrastructure manually through hardware involves huge overhead in terms of time, cost, and manpower and also has large possibility of errors. Software solutions need to be built for reducing these overhead, leading to set up of infrastructure within few minutes or seconds. Further, intelligence can be added to software solution which can make it self-healing, self-monitoring and self-automated. Cloud and automation tools like DevOps technologies should be adopted which can make our infrastructure setup more flexible to the changes.

Treating infrastructure provisioning as a software process can allow us to implement software development features like version control, deployment and testing automation and orchestration with continuous integration and continuous delivery.

Microsoft Azure, which uses Azure Resource Manager (ARM) templates as an automation tool for deploying infrastructure through code, HashiCorp Terraform, an open source tool, used for deploying infrastructure to multiple clouds using same configuration files, Pulumi, multi-cloud solution for Infrastructure as a Code which also supports multiple languages like C, C#, JavaScript etc. for coding infrastructure.

Working and analyzing different tools for coding infrastructure, leads to better understanding of which tool to use at a given scenario for optimal result.

## **ACKNOWLEDGEMENT**

I wish to express my deep gratitude to my guide **Mr. Manish Kumar (Associate Architect, AIS)**, for all advice, encouragement and constant support they have given us throughout our project work. This work would not have been possible without their support and valuable suggestions.

I sincerely thank my **Head of the Department, Dr. Deepshikha Bhargava** and **Professor & Director SCS, Dr. Manish Prateek** for providing me with this opportunity to work on internship project and gain vital industrial exposure while doing so.

I am also grateful to UPES and its faculty members for providing the necessary facilities and programs to carry out our project work successfully.

I would like to thank all my **colleagues** for their help and constructive criticism during this project work. Finally, I have no words to express sincere gratitude to my **parents** who have been my constant support and my real strength.

**Name : Riya Agrahari**

**Roll No. R110216128**



# **1. INTRODUCTION**

## **1.1. Agile Software Development**

Agile development process is a method of building Software incrementally by iterating steps so that development work can adapt to dynamic requirements. Conventional model fail to deliver product on time. Agile model comes to rescue here, to fulfill the needs of a changing environment.

Each increment release in agile model is called as Sprint. Scrum is an implementation of agile methodology. Development with Agile practice aims at delivering value product faster and higher adaptability to dynamic changes.

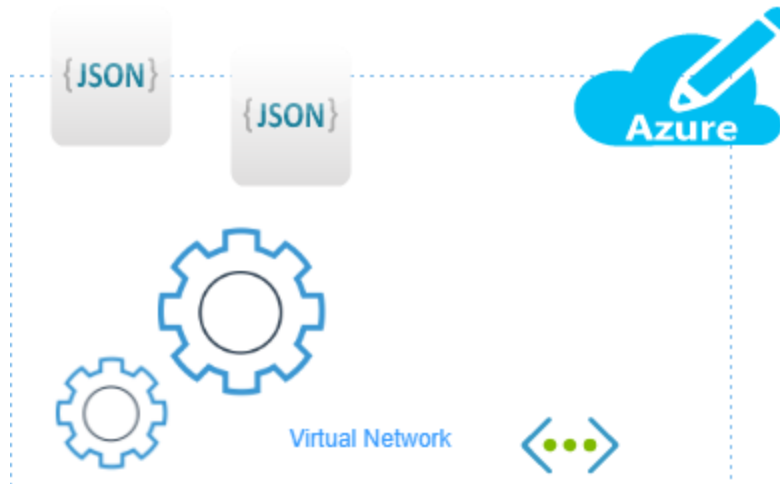
## **1.2. Software Defined Infrastructure**

Virtualization has enabled IT resources to be consumed by the medium of network. Now IT demands rapid and flexible setup of infrastructure where application can be deployed meeting user demands with least management cost for the resources. Fulfilling this demand is done through Software defined infrastructure where operation and management of IT infrastructure is done through software methodology without need of any manpower.

The process is automated, uses modular design features like reusability and repeatability, with intelligence added it to such that it can self-heal, self-scale, self-optimize and be self-aware by itself. Examples of Software defined infrastructure include Software defined Compute, Storage, Network etc.

## **1.3. Infrastructure as Code implements Software Defined Infrastructure**

Provisioning resources and managing it like software through code is enabled by Infrastructure as a Code. Therefore, software defined controls like version control system, automation, orchestration with continuous integration and continuous delivery services can be applied to our infrastructure setup as well.



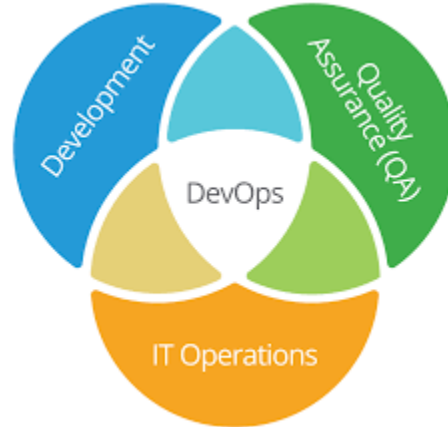
IaC aims at agile development practices allowing it to be flexible to changes and setting up of infrastructure in few minutes. IaC software solution provides software solution in scenario which is specific to cloud service provider.

#### **1.4. Benefits of Infrastructure As Code**

The best way to avoid human errors that might creep in while launching any cloud infrastructure is through automation. Infrastructure as code allows us to automatically launch the required cloud environment efficiently and without errors. Few of its benefits are as follows:

- It allows us to launch a complete and pre-configured infrastructure instantly by running a single script. This saves us a lot of time and is also easy to run.
- It allows for a standardized procedure of setup which helps in maintaining consistency throughout the deployment process. Hence decreasing incompatibility issues.
- Infrastructure as code acts as a documentation of the infrastructure, so it is easier for a team to understand and cooperate with each other with very less chances of conflicts.
- IaC helps in increasing the productivity of developers by allowing them to launch sandbox environments of their own.
- By cutting down on the time spent for manual work of deploying infrastructure, it saves time and cost for the company.

## 1.5. DevOps



DevOps is a software development method which includes combination of development and operations team, focuses on agile practices and mainly encourages collaboration, communication and integration of IT operations and Software Developers so that Software can be delivered at a faster rate with improved quality in the market.

Coding Infrastructure and getting your resources provisioned to the desired state is done through IaC. It requires application of DevOps practices for automation of scripts to ensure error-free re-usability of code. Thus IaC becomes prerequisite for DevOps practices like version control, code audits, automated testing, Continuous Integration and Continuous Testing (CICD).

## 1.6. What is Microsoft Azure?

Microsoft Azure is a cloud computing platform created by Microsoft to build, deploy, test and manage applications and services through data centers managed by Microsoft. It delivers Infrastructure as a Service (IaaS), Software as a Service (SaaS), Platform as a Service (PaaS) and various programming languages and frameworks both Microsoft-specific and third-party are supported by it.

Some of the services provided by it are discussed below:

- **Azure Active Directory (Azure AD)**

Azure active directory is Microsoft's multi-tenant, cloud-based identity and access management service for application running on Microsoft Azure and on premise environment.

It provides Single Sign-On option for cloud applications, read, view or write to other applications if permissions are provided, thus helping in collaboration work.

In Azure AD, two important entities are Users and Group. Users can be Internal and External or Guest Users. User in Azure AD can be added as Work –or Student User or a Microsoft Account User. Groups can be made to perform project works on a single or multiple applications and increase team productivity.

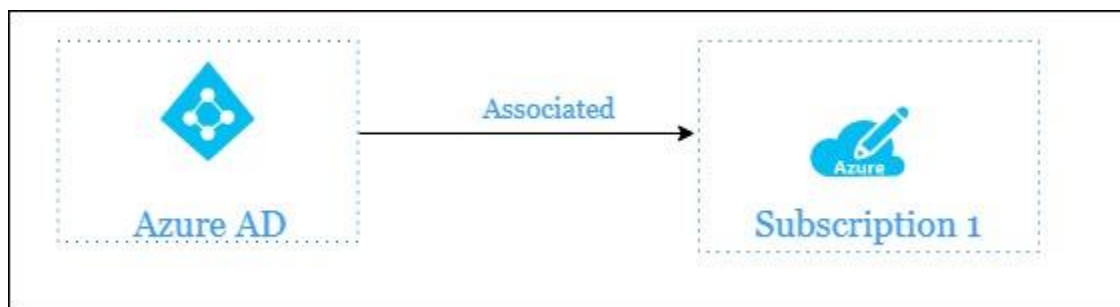
Azure AD authenticates user and returns id token and authorization code. These token in azure are called as JSON Web Token (JWT), signed by Azure Base64 encoded JSON objects.

- **Azure Subscription**

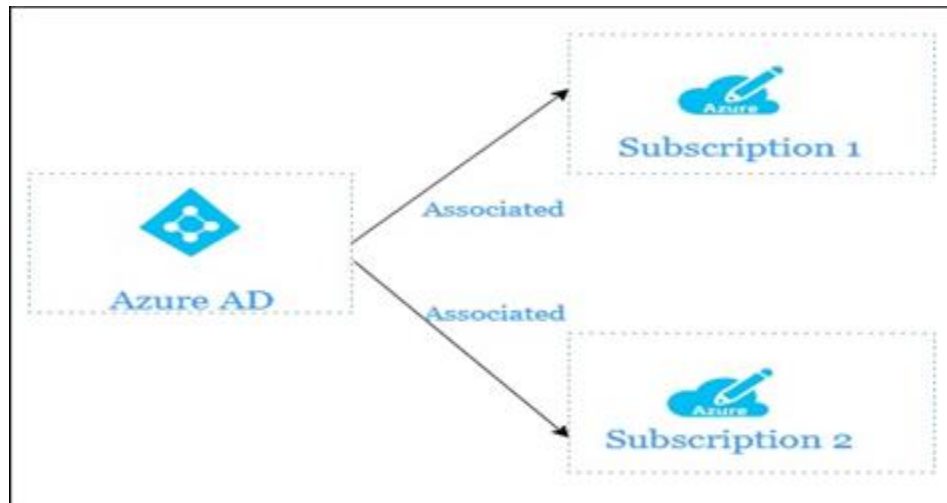
Azure Subscription is a logical billing entity that provides agreement with Microsoft to deploy and use Azure resources, for which cost can be charged per user-license or based on resource usage. Examples for Azure Subscription can be Free Trial, Pay As You Go, Developer Support, Standard Support, and Azure in Open etc.

Depending on the type of subscription being used by the customer, azure provides access to resources and halts the access if request goes against the subscription plan (for e.g. in case of free trial plan)

There can be one or more subscription associated with an Azure Active Directory.



**Fig 1: Single Subscription in an Azure AD**



**Fig 2: Multiple Subscriptions in an Azure AD**

- **Azure Resource Group and Azure Resources**



**Fig 3: Hierarchical Structure of Various Components in Azure**

The word 'resource' here in Azure refers to all entities that are managed by Microsoft Azure, such as web apps, virtual machines, VNets, databases etc.

Resource Groups can be understood as a logical container having a group of related resources. These resources are grouped together so that they can be managed as a single unit. Resources are allocated in a Resource Group based on the needs of an organization, for instance a Resource Group may contain resources with similar life cycle, so that they can be created and/or deleted together.

- **Azure Resource Manager**

Azure Resource Manager hosts RESTFUL API which client uses to manage resources. Each resource in Azure has its own Resource Provider. When a client makes a request for specific resource, Azure Resource Manager links resource to respective resource provider.

## **1.7. Roles in Azure**

In Microsoft Azure, roles are the set of rules that manage the access to resources and resource groups. In the initial days of Azure, there were only three administrator roles which are now known as Classic subscription administrator roles. Later on, further roles i.e. RBAC authorization system and Azure AD administrator roles were added.

- **Classic Subscription Administrator Roles**

Azure's Classic subscription administrator roles include following roles:

- Account Administrator: Billing owner of subscription, no access to Azure portal.
- Service Administrator: Full access to Azure portal. Equal access as Owner role user
- Co-Administrator: Equal access as user with Owner role at the subscription scope.

These administrators have complete access to the subscriptions. They manage the Azure resources through various means like Azure Portal and APIs (Azure Resource Manager APIs and classic deployment model APIs).

- **Azure Role-Based Access Control (RBAC)**

RBAC authorization system provides over 70 built-in roles for very precise access management of Azure resources. Azure RBAC has following four fundamental roles:

- Owner: Full access to all resources, right to delegate access to others.

- Contributor: Create and manage all resources, cannot grant access to others.
- Reader: View resources that exist.
- User Access Administrator: Manage resource access of users.

RBAC works by creation of Role Assignments which has three major components: Security Principal, Role Definition, and Scope.

- Security Principal is secure identity that an application or service uses to have access to particular azure resources.
- Role definition is a set of permissions like read, write, delete
- Scope is the limit of resources on which a particular application or service can apply its power defined by role.
- Deny Assignments can also be now implemented by RBAC but in a limited way.
- **Azure AD administrator roles**

Azure AD administrator roles manage the resources in Azure Active Directory. These administrators have powers to create, edit users, assign administrative roles, manage domain and user license, reset password etc.

These are of three types:

- Global Administrator: Signs up for the Azure AD tenant.
- User Administrator: Creates and manages all aspects of users and groups.
- Billing Administrator: Makes purchases manage subscriptions and support tickets.

## 1.8. Storage in Azure

Azure Storage, a durable, scalable, managed, secure and highly available service provided by Microsoft that supports scalable object data, cloud file system, reliable messaging and NoSQL store for storing unstructured data.

**There are four major categories of storage system in Azure**

- **Azure Blobs** : Azure Blobs provide scalable object storage for binary and text files
- **Azure Files** : Azure files stores file shares for on premise or cloud deployment

- Azure Queues : Stores is a messaging store where reliable communication
- Azure Tables : Azure Tables offer storage in form of optimized tables and are now a part of Azure Cosmos DB

### **Types of storage accounts in Azure**

- General Purpose V2
- General Purpose V1
- Block Blob Storage
- File Storage
- Blob Storage.

Storage accounts are encrypted by SSE (Storage Secure Encryption). Redundancy in Azure Storage is maintained by replicating multiple copies of our data locally, at different zones or at different regions.

### **Categories of redundant storage:**

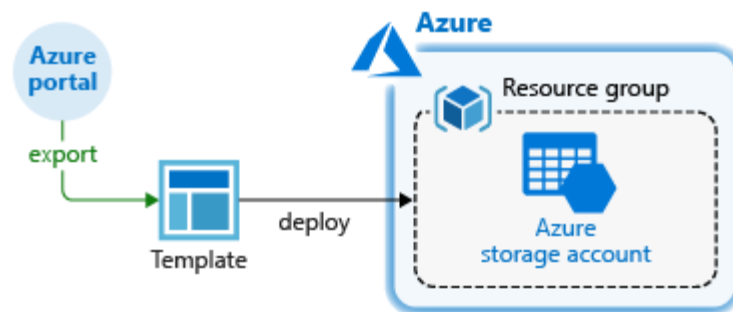
- Locally Redundant Storage (LRS) - Data is replicated within a single storage unit, low cost solution
- Zone Redundant Storage (ZRS) - High-Availability Solution for replication across Availability Zones in a single region.
- Geo Redundant Storage (GRS) - Disaster Recovery Solution for Asynchronous replication of data across multiple geographical regions.
- Read-Access Geo Redundant Storage (RA-GRS) - GRS solution with read access to replicated data.



## 1.9. Infrastructure as Code in Azure

- **ARM Template**

ARM Templates allows for deploying various Azure resources by using a single declarative template built in JSON.



The templates can be used to redeploy resources, with different parameters if required, by just changing the parameters in the JSON script. These templates help in achieving automation for setting up the Infrastructure within a matter of seconds.

ARM Template is the way of declaring resources we want in the form of JSON object with attributes like names, properties, type etc. that can be managed like other version control files.

Main advantage of ARM is that we can deploy several resources in a single unit of code and the deployments are idempotent.

There are two modes in which ARM templates execute:

**COMPLETE:** Objects that do not appear in the template and are already in the Resource Group are deleted. Better for newly set-up infrastructure because previously present component will not conflict with the new ones created.

**INCREMENTAL:** Objects mentioned in the template are added to the existing Resource Group as additional components retaining the older components. We don't lose already build infrastructure.

- **Terraform**

Terraform is an open source tool which can be used to provision and manage cloud infrastructure.



It is an Infrastructure as a Code tool which uses a simple configuration file written in HashiCorp Configuration Language to deploy resources. Terraform has a multi-cloud support, which means it can be used for automating various clouds like Microsoft Azure, GCP, AWS etc, and deploy resources in them using the HashiCorp Configuration Language.

- **Pulumi**



Pulumi, like Terraform, is another open source cloud development tool that can be used to develop and configure resources on multiple cloud platforms. But unlike Terraform that uses single language (HashiCorp Configuration Language), Pulumi has support for various general purpose languages like Python 3, Javascript, Typescript, GoLang etc. Pulumi uses a cloud object model with its evaluation runtime environment which are language-neutral and cloud-neutral, making it possible to support various languages and cloud environments.

## **2. Problem Definition**

In every organization, applications are deployed by provisioning of some resources. These resources in turn need to be handled and managed. Performing this manually through Hardware

configuration may involve risk of errors and huge investment of cost and time. It includes, routine work performed monotonously again and again without the use of any intelligent monitoring tool.

Also today, software development in many organizations works on agile methodology where software is continuously delivered adapting to changing requirements.

Solutions are needed which can overcome these overhead and accept Agile methodology of development in setting up Infrastructure. Solving this problem is motive of this project.

### **3. Objective**

Infrastructure as Code mainly focuses on establishing Software Defined Infrastructure and performing automated provisioning of resources where our application can be deployed.

This aims at minimizing time required for setting up any infrastructure and also removes the overhead cost of Hardware procurement. Infrastructure as code supports agile development practices and for developers, it becomes an important business requirement.

The main objective of the project is to develop and Implement automated provisioning of resources using Azure Resource Manager (ARM) templates, Terraform and Pulumi with modular approach along with features like reusability, repeatability, abstraction, inheritance, Disaster Recovery and High Availability Solution for a given scenario.

### **4. Project Overview**

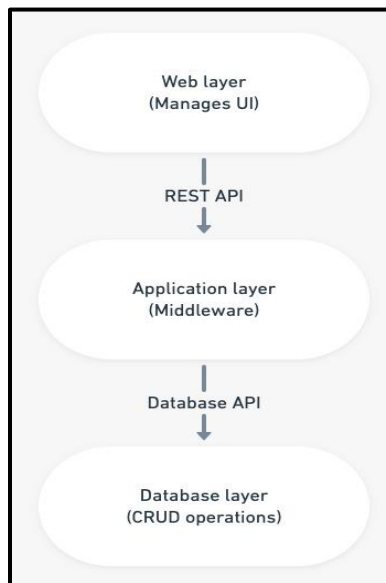
Developed Infrastructure as Code in Azure for Automated Provisioning of resources like Virtual Networks, Sub-Networks, Virtual Machines, Storage, Network Security Rules and other deployable infrastructure resources.

Methods of implementation include: Azure Resource Manager (ARM) templates, Terraform HCL scripts and Pulumi javascript code.

- ARM templates work with JSON data. ARM templates are developed for coding Infrastructure specific to Azure resources.
- Terraform can be used for creating multi-cloud Infrastructure solution through code. It has its own HashiCorp Configuration Language (HCL).
- In Pulumi, deployment of Azure code in JavaScript and as Pulumi has its own API shield so we will call Pulumi API for setting up infrastructure through code.

**Case Study for The Three Tier Architecture:** Web-API-DB is implemented along with consideration of network security rules - inbound and outbound traffic within the subnetworks in a Virtual network.

**Web-API-Database** is a 3 tier architecture which decouples to presentation, business or application and database layers individually. The business layer exposes its API to applications as well as for the remote presentation layer. Remote presentation layer uses REST API. There is an internal database API for communications between application layer and database layer. REST API's input is processed by the application layer to perform CRUD operations on the database.



**Fig 4: Three Tier Architecture**

This template creates a Virtual Network with 3 subnets (Web Server, API and Database). Further, Network Security Group is created for each subnet.

First subnet will be associated with a Web Server for hosting web pages, second subnet with Application Server containing API logic, request for API calls will be made to this VM from Web server and response will be generated from the API VM to Web server.

API VM will interact with Database VM for filtering the data according to the request and response from Database VM will be given to API VM.

Inbound rules and Outbound Rules will be specified in each Network Security Group and different Network Security Groups are made as there can be different traffic for each network

NSG with inbound and outbound rules are mentioned below:

### **Web Server NSG**

#### **Inbound Rules:**

Source Address	Source Port	Destination Address	Destination Port	Name	Priority	Action
Any	Any	<subnet-1>	80, 443	HTTP-HTTPS-request	100	Allow
(Admin)	Any	<subnet-1>	22,3389	SSHAndRDP	110	Allow
<subnet-2>	8083,445	<subnet-1>	8081,446	API-Web-resp	120	Allow

Assumption: Our REST API calls are set to port 8081 for receiving GET and POST requests,

**Outbound Rules:**

Source Address	Source Port	Destination Address	Destination Port	Name	Priority	Action
<subnet-1>	8081,446	<subnet-2>	8083,445	Web-API-request	100	Allow
<subnet-1>	Any	<subnet-3>	Any	DenyDB	110	Deny
Any	Any	Any	Any	DenyAllOutBound	65500	Deny

**API NSG****Inbound Rules:**

Source Address	Source Port	Destination Address	Destination Port	Name	Priority	Action
<subnet-1>	8081, 446	<subnet-2>	8083, 445	HTTP-request-api	100	Allow
(Admin)	Any	<subnet-2>	22,3389	SSHAndRDPtoAPI	110	Allow
<subnet-3>	Any	<subnet-2>	21	FTPfromDB-API	120	Allow

**Outbound Rules:**

Source Address	Source Port	Destination Address	Destination Port	Name	Priority	Action
<subnet-2>	118	<subnet-3>	3306	Api-MySQLreq	100	Allow
<subnet-2>	8083, 445	<subnet-1>	8081, 446	API-web-response	110	Allow
<subnet-2>	Any	<subnet-3>	21	MYSQL-request-api	120	Allow

## **Database NSG**

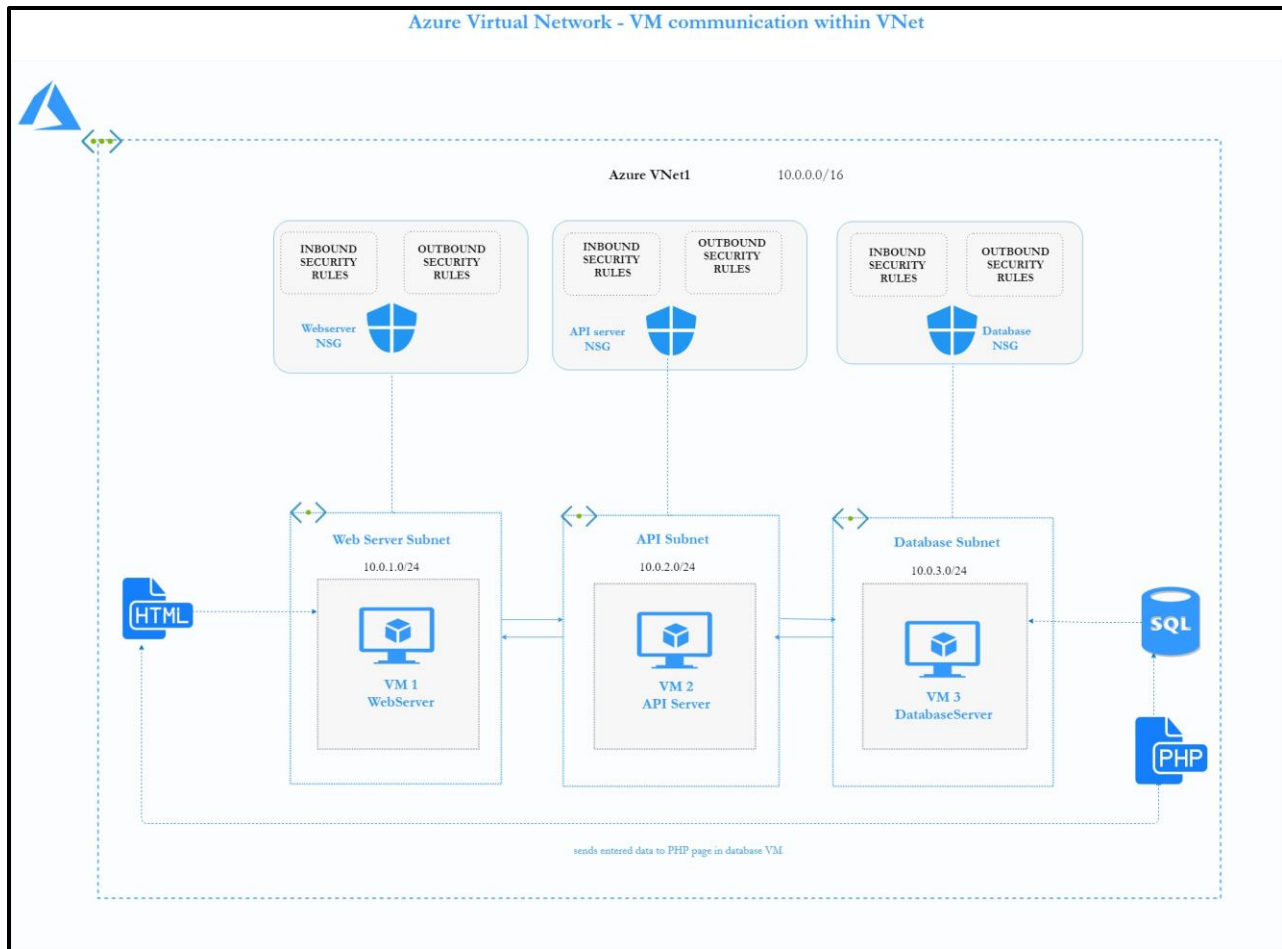
### **Inbound Rules:**

<b>Source Address</b>	<b>Source Port</b>	<b>Destination Address</b>	<b>Destination Port</b>	<b>Name</b>	<b>Priority</b>	<b>Action</b>
<subnet-2>	118	<subnet-3>	3306	API-DB-request	100	Allow
<subnet-1>	Any	<subnet-3>	Any	Web-DB-request	110	Deny
(Admin)	Any	<subnet-3>	22	SSH-RDP-admin	120	Allow
<subnet-2>	Any	<subnet-3>	21	FTP-DBInbound	130	Allow

### **Outbound Rules:**

<b>Source Address</b>	<b>Source Port</b>	<b>Destination Address</b>	<b>Destination Port</b>	<b>Name</b>	<b>Priority</b>	<b>Action</b>
<subnet-3>	3306	<subnet-2>	118	MYSQL-response	100	Allow
<subnet-3>	Any	<subnet-2>	21	FTP-DBOutbound	110	Allow
<subnet-3>	Any	<subnet-1>	Any	DenyConnectionTo Web	120	Deny

Below is the Architectural Diagram for the primary VNets and subnets which are created and network security rules are applied to subnets using NSG.



**Fig 5: Detailed Architectural Diagram for Primary VNet.**

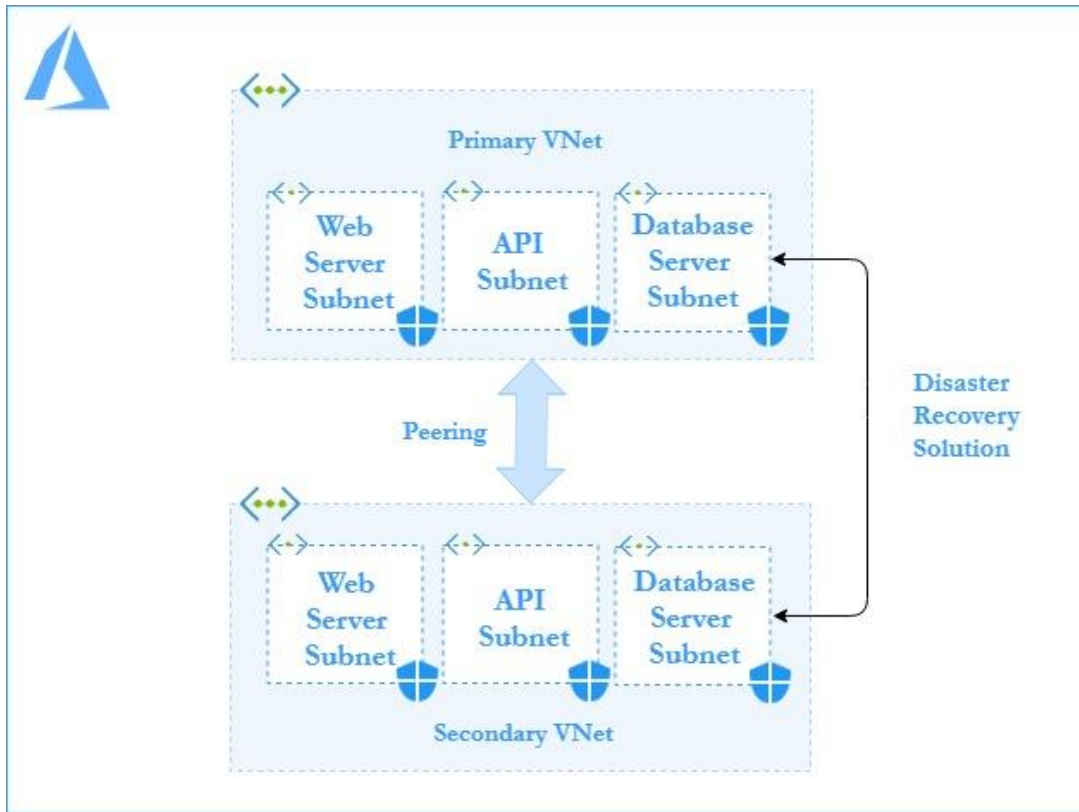
### Disaster Recovery Solution

Replica of the primary Vnet with its subnets and NSGs are created at some secondary location which comes up as recovery solution in case of a disaster. Communication between these Vnets is established using global peering.

In case of disaster, rules of NSG are appended opening custom ports on both Vnet, allowing database synchronization from secondary to primary region.

Below is the architectural diagram for disaster recovery solution developed where a replica (secondary vnet) is created and peering is established between them.





**Fig 6: Architectural Diagram for Disaster Recovery - Primary and Secondary Vnet.**

## 5. Implementation

Basic understanding of Azure Resources was made by working upon Deployment of resources using Azure portal. Below mentioned are hands-on exercises as a part of Group Assignment.

### ❖ Creating users and groups in Azure Active Directory and assigning roles to users.

**Step 1:** Created users and guest users (Guest users are our colleagues).

**Step 2:** Invitation link were sent to users on the account details shared while creating users. Using the invitation link guest users accessed over the default directory. Each guest users were assigned different tenant-id. We accessed groups and were able to see other members added in the group.

**Step 3:** Directory Roles were assigned to users in the group such as Application Administrator, Billing administrator etc. Roles could be re-assigned and the users added could change their password to access invited directory.

**Step 4:** We used IAM where each one of us deployed an app in our own directory and added Role assignment for using the App. We assigned Role to each other in our respective application deployed on azure.

Dashboard > riyaWeb > riyaWebApp > riyaWeb - Access control (IAM)

**riyaWeb - Access control (IAM)**  
App Service

Search (Ctrl+J)

+ Add Edit columns Refresh Remove

Check access **Role assignments** Deny assignments Classic administrators Roles

Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments. [Learn more](#)

Name  Type  Role

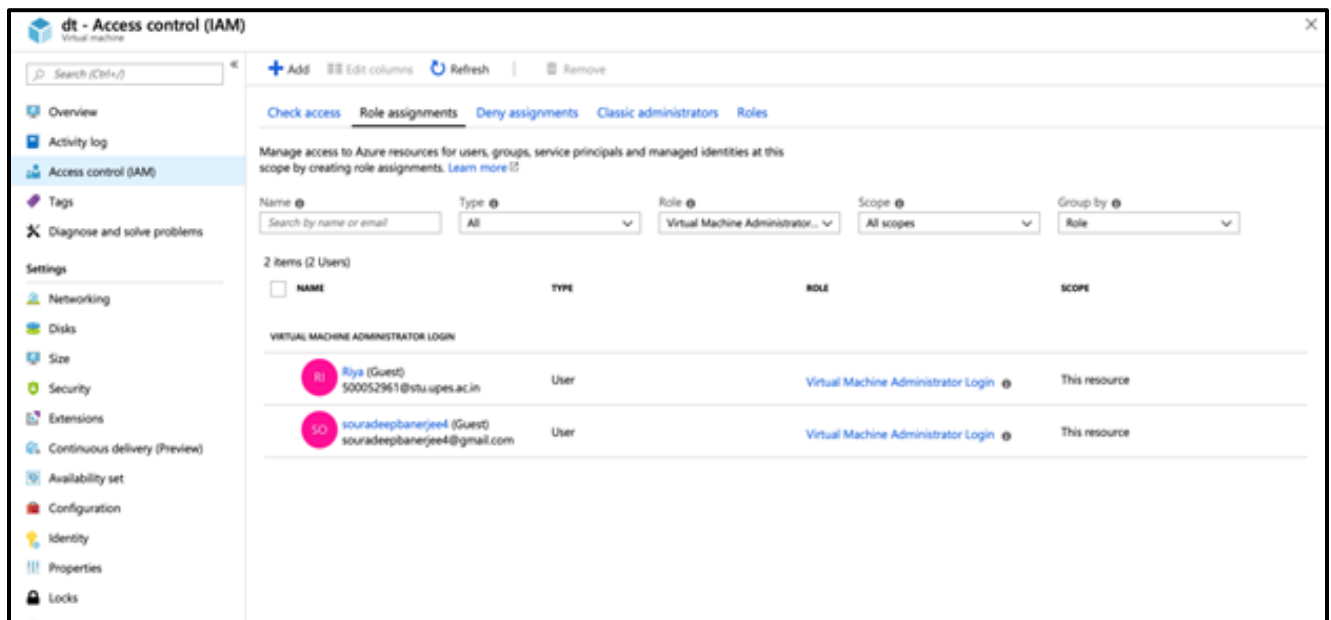
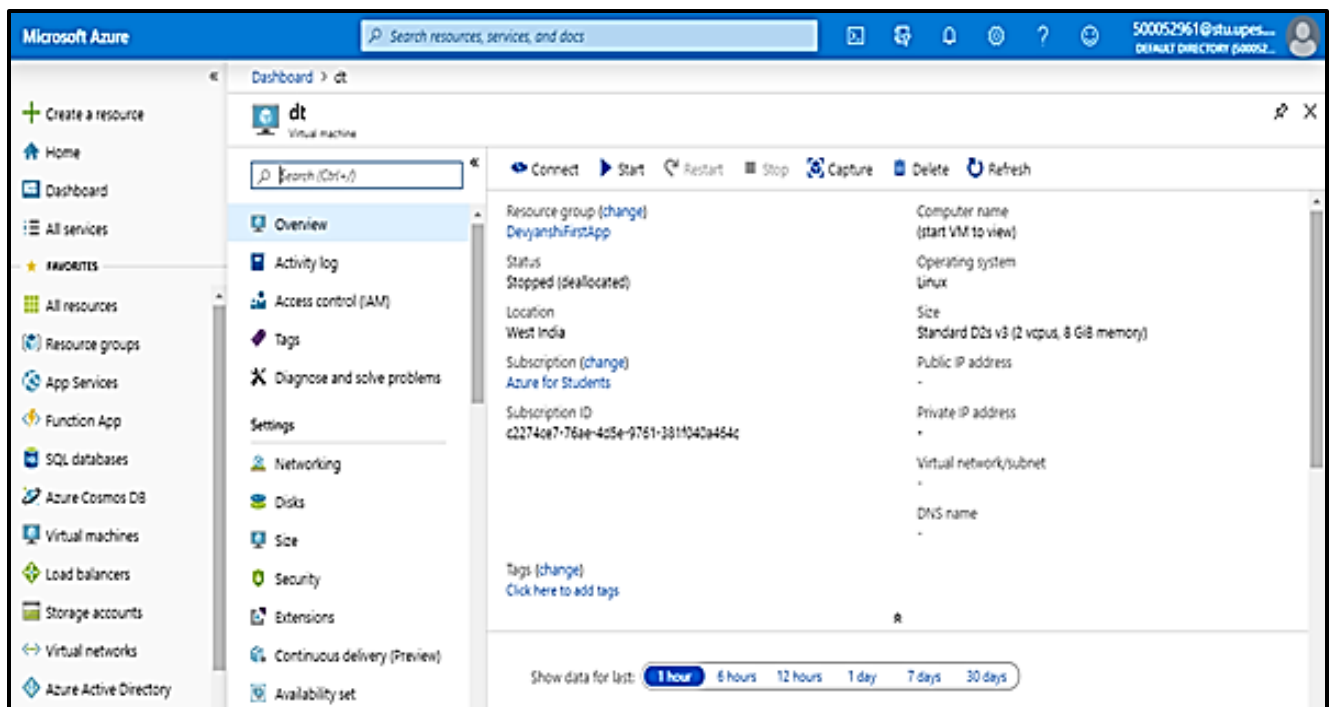
Scope  Group by

2 items (2 Users)

	NAME	TYPE	ROLE	SCOPE
CONTRIBUTOR				
50	500052376 (Guest) 500052376@stu.upes...	User	Contributor ⓘ	This resource
50	souradeepbanerjee4 (G) souradeepbanerjee4...	User	Contributor ⓘ	This resource

**Step 5:** Switching the directory over to the other invited ones, we could view, access or update the application as per role assigned.

For example:

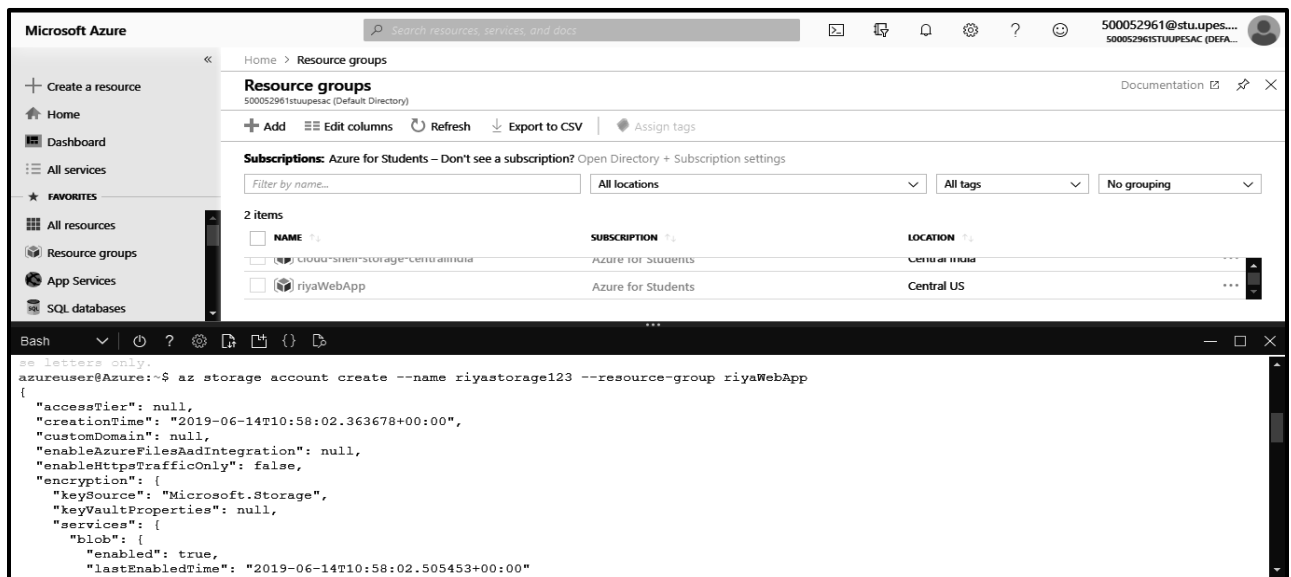
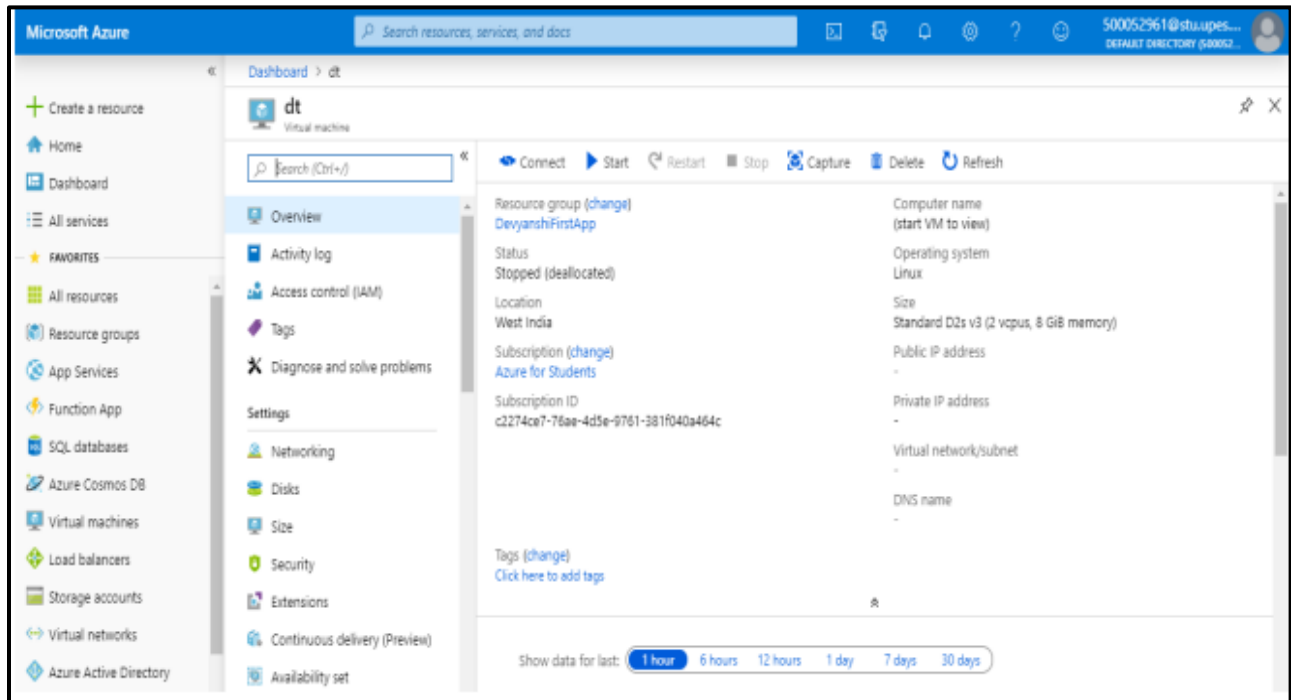


Switching directory I could access Devyanshi's app named as DT where I was assigned role as Virtual machine Administrator Login.

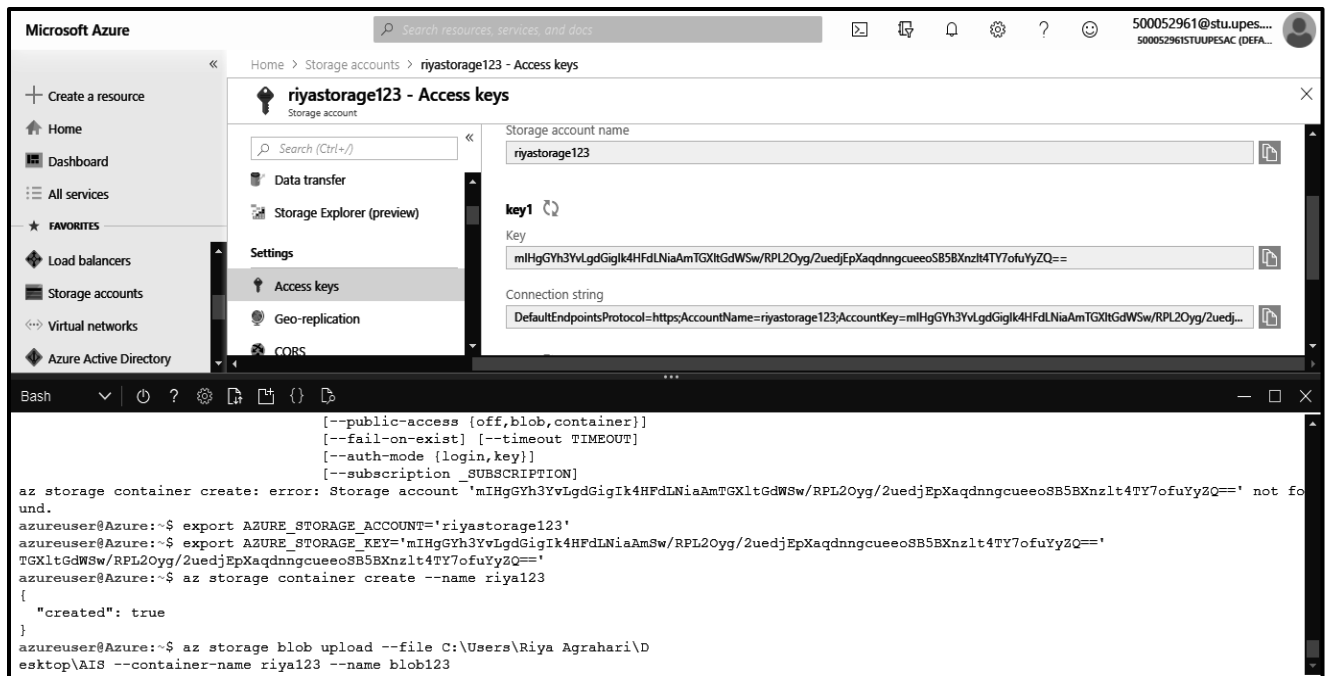
### ❖ Understanding storage account through azure CLI

**Step 1:** Firstly, we used Azure CLI to create storage account.

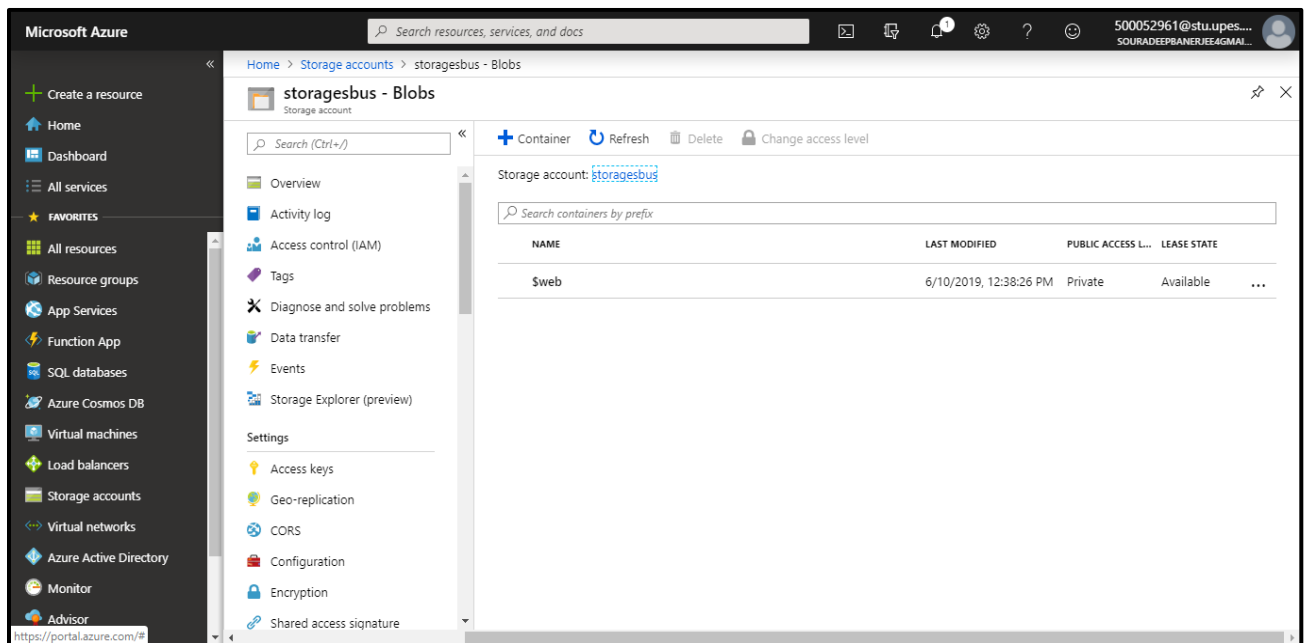
`-$ az storage account create -n <name> -g <resource-group>`



**Step 2:** Then, we setup the storage account and key for creating container on required storage account.



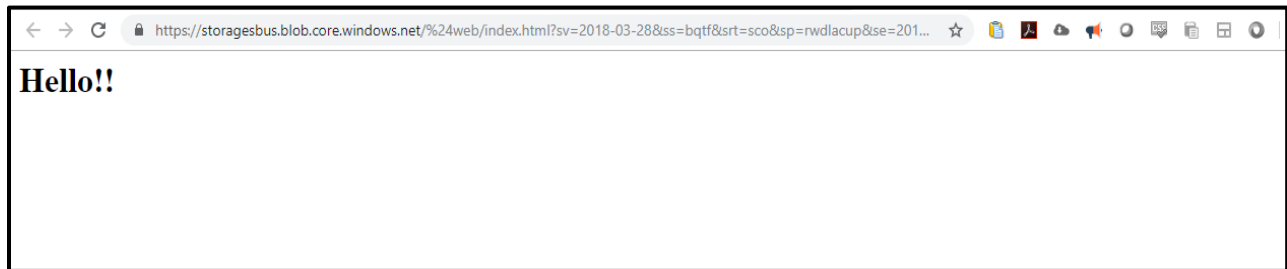
**Step 3:** After that, we created a blob. Inside this blob, files that were required to be given access were uploaded.



**Step 4:** Now, Roles are assigned for Reader and Data Access using the Access Management to each other.

**Step 5:** Now switch to other's directory and access the created storage account.

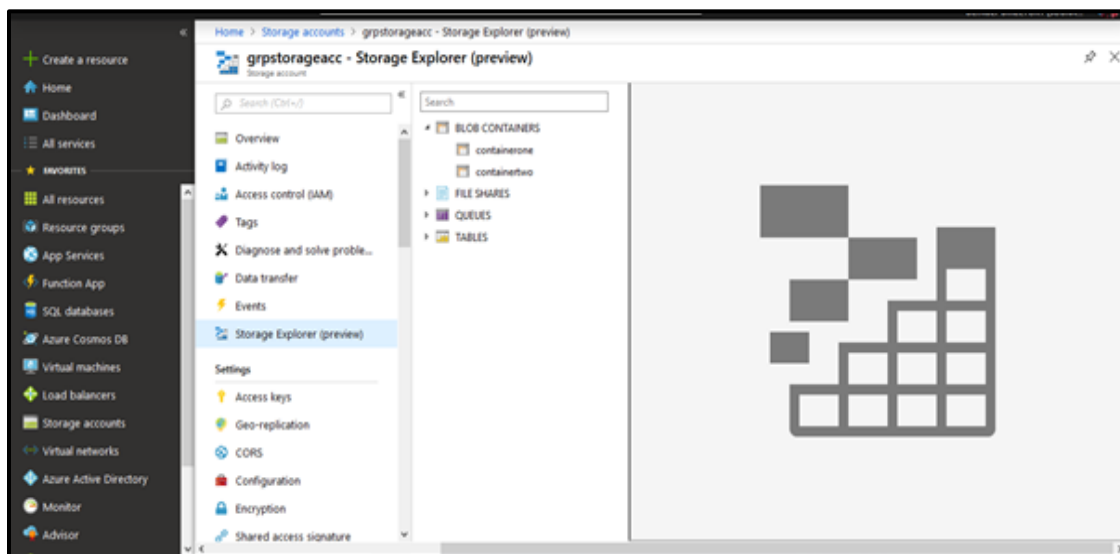
**Step 6:** It can be observed that where only Reader Role was assigned, the file cannot be downloaded to the container, while changing the role and setting to Reader and Data access, the file can be downloaded into the container.



## ❖ Understanding Storage Access along with Virtual Network and Network Security Group Rules

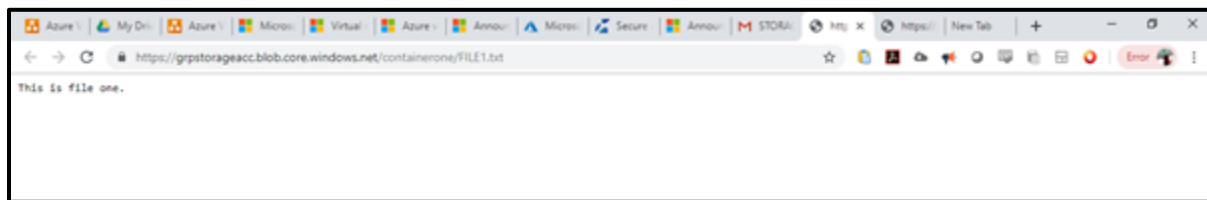
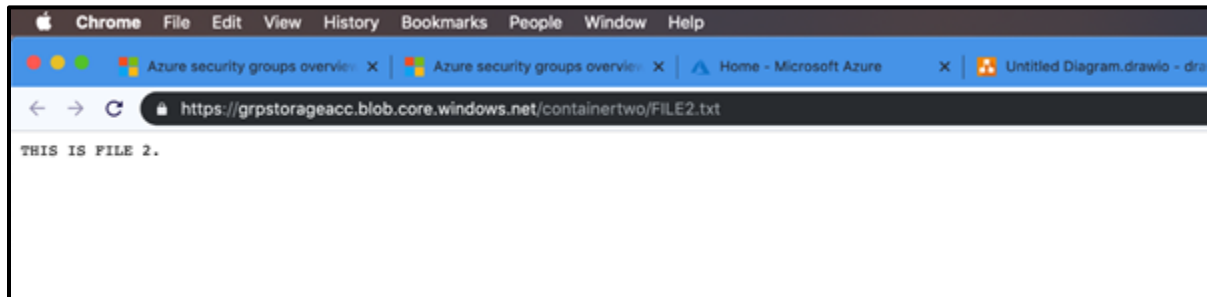
### Part 1

- We created a Storage account in the Resource Group 'MyAzureRsrcGrp' on the Azure portal and assigned no Virtual Network. We named the storage account as 'grpstorageacc'.
- Once the storage account was created, I went on to create two publicly accessible containers 'containerone' and 'containertwo'.



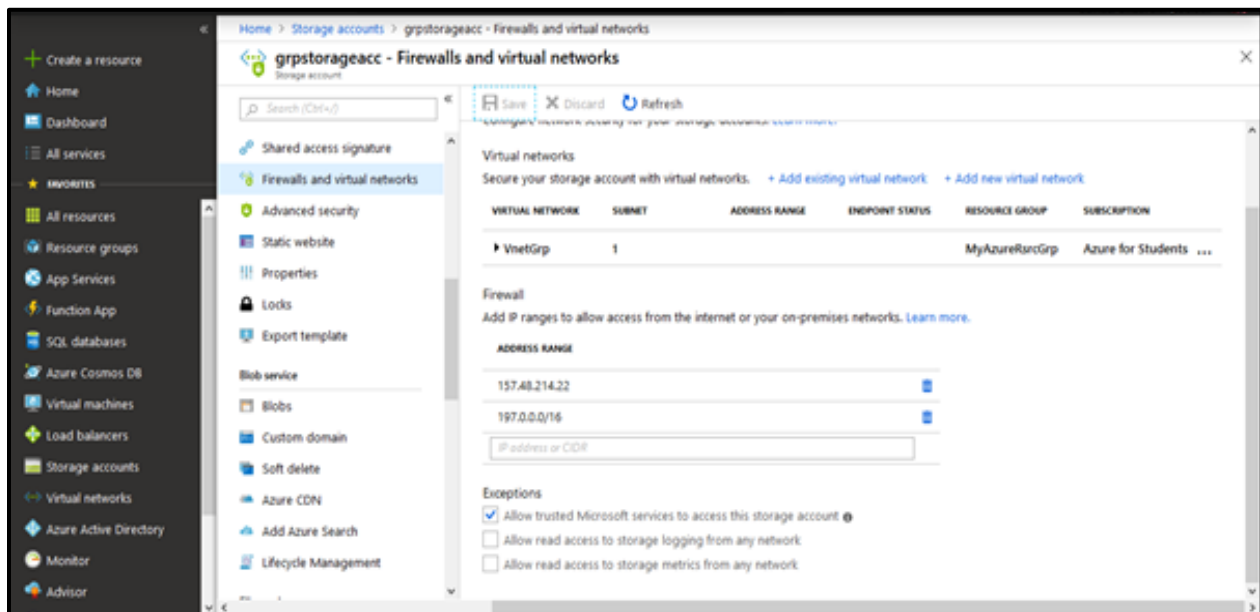
- Then, two text files (FILE1.txt and FILE2.txt) were uploaded in both the created containers

- After that, I passed on the links of the files to my teammates and they tried accessing those files on their respective devices. Both of them were able to access the files on their systems.

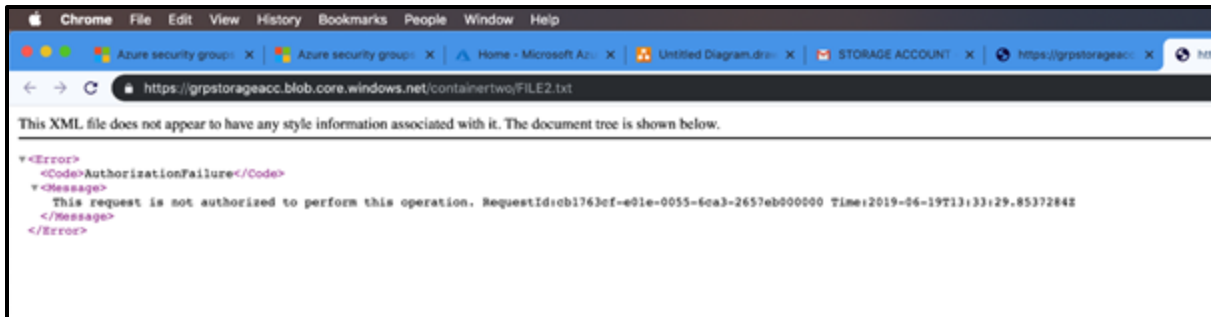


## Part 2

- This time we assign virtual network named (VnetGrp) to the created storage account.

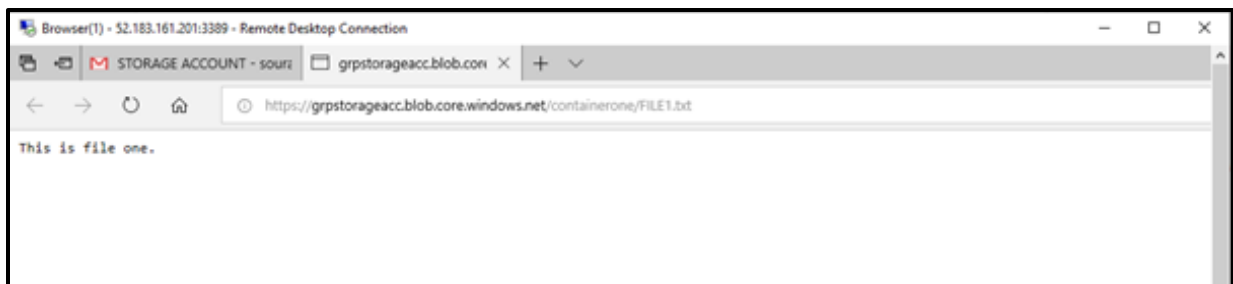


- Once the Virtual Network is assigned to the storage account my teammates tried to access the same link they were provided to access. But this time they were not able to access it and received the following errors.



### Part 3

- We created a VM running Windows 10 and assigned it to the previously created virtual Network VnetGrp. It was placed in the same subnet where the storage account was put. A RDP connection was made to the Virtual Machine, a web browser was then launched on the VM. On trying to access the previously made containers through the browser, we were successfully able to access it.



- We placed the same Virtual Machine in different subnet and went on to access the container. This time we received an error while trying to do so.





## **Infrastructure as Code For Three-Tier Architecture (WEB-API-DB)**

For the Web-API-Database case-study (For detailed discussion: refer Section 4)

### **➤ Resources needed :**

- 2 Virtual Network - Primary and Secondary
- 6 Subnets - 3 in Primary and 3 in Secondary Vnet. Subnets will be as follows:
  - ➔ Web, API, DB subnet
  - ➔ Web-Replica, API-Replica, DB-Replica Subnet
- Additional 4 Subnet - 2 in Primary and 2 in Secondary
  - ➔ Management-JumpBox (Admin), AzureFirewallSubnet
  - ➔ Management-JumpBox-Replica (Admin-Replica), AzureFirewallSubnet (Replica)
- 6 Network Security Group (NSG) - Network Security Groups for each Subnet.
- 2 Peering Connection
  - ➔ From Vnet-Primary to Vnet-Secondary
  - ➔ From Vnet-Secondary to Vnet-Primary

### **➤ Learning and Solution:**

#### **• Implemented Through ARM Templates:**

For the setup of three tier infrastructure (Web-API-Database),

- ➔ Nested ARM template is created and from the Master template, sub-templates are called which are as follows: NSG-Creation Sub-Template, Vnet-creation with subnets Sub-Template and Peering Network Sub-Template.

### **Master.json**

**NSG creation** 'CreateNSG.json' Templates are called from Master.json:

For Primary NSG, we have separate parameters passed as values for the properties.

For Secondary NSG, we have separate parameters passed as values for the properties.

```

"resources": [{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "LinkedTemplate1",
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/riyaagrahari/ARM-Templates/master/NestedTemplate_for_DisasterRecovery/CreateNSG.json",
      "contentVersion": "1.0.0.0"
    }
  },

```

```

    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "name": "LinkedTemplate5",
    "properties": {
      "mode": "Incremental",
      "templateLink": {
        "uri": "https://raw.githubusercontent.com/riyaagrahari/ARM-Templates/master/NestedTemplate_for_DisasterRecovery/CreateNSG.json",
        "contentVersion": "1.0.0.0"
      }
    },

```

**VNet creation Templates** 'VnetTemplate.json' are called from Master.json:

For Primary VNet, we have separate parameters passed as values for the properties.

For Secondary VNet, we have separate parameters passed as values for the properties.

```

{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "LinkedTemplate2",
  "dependsOn": ["LinkedTemplate1"],

  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/riyaagrahari/ARM-Templates/master/NestedTemplate_for_DisasterRecovery/VnetTemplate.json",
      "contentVersion": "1.0.0.0"
    }
  },

```

```

{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "LinkedTemplate3",
  "dependsOn": ["LinkedTemplate1", "LinkedTemplate5"],
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/riyaagrahari/ARM-Templates/master/NestedTemplate_for_DisasterRecovery/VnetTemplate.json",
      "contentVersion": "1.0.0.0"
    }
  },
  "parameters": {

```

**Peering Templates** 'Peering.json' are called from Master.json:

```

{
  "type": "Microsoft.Resources/deployments",
  "apiVersion": "2018-05-01",
  "name": "LinkedTemplate4",
  "dependsOn": ["LinkedTemplate3", "LinkedTemplate2"],
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "https://raw.githubusercontent.com/riyaagrahari/ARM-Templates/master/Vnet_Peering_Template/Template.json",
      "contentVersion": "1.0.0.0"
    }
  },

```

→ Network Security Group - Creation Template is called twice for creation of NSG at primary and secondary location.

Below is some Part of Code for NSG-Creation

```
"resources": [
  {
    "apiVersion": "2019-02-01",
    "type": "Microsoft.Network/networkSecurityGroups",
    "name": "[parameters('NSG1_name')]",
    "location": "[parameters('Location')]",
    "properties": {
      "securityRules": [
        {
          "name": "HTTPOrHTTPS-request",
          "properties": {
            "description": "Rule for HTTP request and HTTPS SSL request ",
            "protocol": "Tcp",
            "sourcePortRange": "*",
            "sourceAddressPrefix": "*",
            "destinationPortRanges": [
              "80",
              "443"
            ],
            "destinationAddressPrefix": "[parameters('subnet1_Address_Prefix')]",
            "access": "Allow",
            "priority": 100,
            "direction": "Inbound"
          }
        },
        {
          "name": "SSHAndRDP-to-WEB",
          "properties": {
            "description": "Rule for SSH and RDP Request",
            "protocol": "Tcp",
            "sourcePortRange": "*",
            "sourceAddressPrefix": "[parameters('Management_JumpBox_Subnet_AddressPrefix')]",
            "destinationPortRanges": [
              "22",
              "3389"
            ],
            "destinationAddressPrefix": "[parameters('subnet1_Address_Prefix')]",
            "access": "Allow",
            "priority": 110,
            "direction": "Inbound"
          }
        }
      ]
    }
  }
]
```

```

    }
  },
  {
    "name": "API-Web-response",
    "properties": {
      "description": "Rule for accepting response from API-Web Server",
      "protocol": "Tcp",
      "sourcePortRanges": [
        "8083",
        "445"
      ],
      "sourceAddressPrefix": "[parameters('subnet2_Address_Prefix')]",
      "destinationPortRanges": [
        "8081",
        "446"
      ],
      "destinationAddressPrefix": "[parameters('subnet1_Address_Prefix')]",
      "access": "Allow",
      "priority": 120,
      "direction": "Inbound"
    }
  },
  {
    "name": "DenyOutBoundToDB",
    "properties": {
      "description": "Rule for denying Out-bound traffic to Database ",
      "protocol": "Tcp",
      "sourcePortRange": "*",
      "sourceAddressPrefix": "[parameters('subnet1_Address_Prefix')]",
      "destinationPortRange": "*",
      "destinationAddressPrefix": "[parameters('subnet3_Address_Prefix')]",
      "access": "Deny",
      "priority": 100,
      "direction": "Outbound"
    }
  },
  {
    "name": "Web-API-request",
    "properties": {
      "description": "Rule for sending request from Web-API Server",

```

```

    "protocol": "Tcp",
    "sourcePortRanges": [
      "8081",
      "446"
    ],
    "sourceAddressPrefix": "[parameters('subnet1_Address_Prefix')]",
    "destinationPortRanges": [
      "8083",
      "445"
    ],
    "destinationAddressPrefix": "[parameters('subnet2_Address_Prefix')]",
    "access": "Allow",
    "priority": 110,
    "direction": "Outbound"
  }
}

```

This is example of one NSG creation (Web-NSG) with Inbound and Outbound Rules. In the similar format 5 more NSGs are made with different security rules (discussed in Section 4).

→ Vnet-Creation template is called twice for creation of two Virtual Networks- Primary and Secondary at different Location.

Below is some Part of Code for Vnet Creation:

```

"resources": [
  {
    "apiVersion": "2019-02-01",
    "name": "[parameters('Virtual_Network1-name')]",
    "type": "Microsoft.Network/virtualNetworks",
    "location": "[parameters('Location')]",
    "properties": {
      "addressSpace": {
        "addressPrefixes": ["[parameters('Vnet1addressPrefix')]"]
      },
      "subnets": [{
        "name": "[parameters('subnet1-name')]",
        "properties": {
          "addressPrefix": "[parameters('subnet1-AddressPrefix')]",
          "networkSecurityGroup": {
            "id": "[resourceId('Microsoft.Network/networkSecurityGroups', parameters('NSG1-name'))]"
          }
        }
      },
      {
        "name": "[parameters('subnet2-name')]",
        "properties": {
          "addressPrefix": "[parameters('subnet2-AddressPrefix')]",
          "networkSecurityGroup": {
            "id": "[resourceId('Microsoft.Network/networkSecurityGroups', parameters('NSG2-name'))]"
          }
        }
      },
      {
        "name": "[parameters('subnet3-name')]",
        "properties": {
          "addressPrefix": "[parameters('subnet3-AddressPrefix')]",
          "networkSecurityGroup": {
            "id": "[resourceId('Microsoft.Network/networkSecurityGroups', parameters('NSG3-name'))]"
          }
        }
      }
    ]
  }
]

```

```

    }
  },
  {
    "name": "[parameters('Management-JumpBox_Subnet_Name')]",
    "properties": {
      "addressPrefix": "[parameters('Management-JumpBox_Subnet_AddressPrefix')]",
    }
  },
  {
    "name": "AzureFirewallSubnet",
    "properties": {
      "addressPrefix": "[parameters('firewallSubnet-AddressPrefix')]"
    }
  }
],
"enableDdosProtection": "[parameters('enableDdosProtection')]"
}

```

→ Then Peering is established by calling Peering Sub-Template where it creates Peering Connection from Primary Virtual Network to Secondary Virtual Network and from Secondary to Primary Virtual Network for synchronization in case of any disaster.

Below is Part of Code of Peering Creation Template

```
"resources": [
  {
    "apiVersion": "2019-02-01",
    "type": "Microsoft.Network/virtualNetworks/virtualNetworkPeerings",
    "name": "[variables('VNet1_to_VNet2_Peering')]",
    "properties": {
      "allowVirtualNetworkAccess": "[parameters('Allow_Virtual_Network_Access')]",
      "allowForwardedTraffic": "[parameters('Allow_Forwarded_Traffic')]",
      "allowGatewayTransit": "[parameters('Allow_Gateway_Transit')]",
      "useRemoteGateways": "[parameters('Use_Remote_Gateways')]",
      "remoteVirtualNetwork": {
        "id": "[resourceId('Microsoft.Network/virtualNetworks',parameters('Virtual_Network2_Name'))]"
      }
    }
  },
  {
    "apiVersion": "2019-02-01",
    "type": "Microsoft.Network/virtualNetworks/virtualNetworkPeerings",
    "name": "[variables('VNet2_to_VNet1_Peering')]",
    "properties": {
      "allowVirtualNetworkAccess": "[parameters('Allow_Virtual_Network_Access')]",
      "allowForwardedTraffic": "[parameters('Allow_Forwarded_Traffic')]",
      "allowGatewayTransit": "[parameters('Allow_Gateway_Transit')]",
      "useRemoteGateways": "[parameters('Use_Remote_Gateways')]",
      "remoteVirtualNetwork": {
        "id": "[resourceId('Microsoft.Network/virtualNetworks',parameters('Virtual_Network1_Name'))]"
      }
    }
  }
]
```

[For full code you can visit this link: [https://github.com/riyaagrahari/ARM-Templates/tree/master/NestedTemplate\\_for\\_DisasterRecovery](https://github.com/riyaagrahari/ARM-Templates/tree/master/NestedTemplate_for_DisasterRecovery)]

- **Implemented Through Terraform:**

In Terraform, WEB-API-DB infrastructure with disaster Recovery solution was created in two ways:

Firstly by Calling ARM Template in Terraform, this is as shown below:

```

0 references
resource "azurerm_template_deployment" "test" {
  name                = "template-01"
  resource_group_name = "${azurerm_resource_group.test.name}"
  template_body       = "${file("./Master.json")}"
}

parameters = {
  "Location-of-Virtual_Network1"          = "West US",
  "NSG1-name"                             = "NSG-1-web",
  "NSG2-name"                             = "NSG-1-api",
  "NSG3-name"                             = "NSG-1-db",
  "NSG1-name-replica"                     = "NSG-1-web-replica",
  "NSG2-name-replica"                     = "NSG-1-api-replica",
  "NSG3-name-replica"                     = "NSG-1-db-replica",
  "Virtual_Network1-name"                 = "Vnet-1",
  "Vnet1addressPrefix"                    = "10.0.0.0/16",
  "subnet1-name"                         = "subnet-1",
  "subnet1-AddressPrefix"                 = "10.0.1.0/24",
  "subnet2-name"                         = "subnet-2",
  "subnet2-AddressPrefix"                 = "10.0.2.0/24",
  "subnet3-name"                         = "subnet-3",
  "subnet3-AddressPrefix"                 = "10.0.3.0/24",
  "firewallSubnet-AddressPrefix-Vnet1"    = "10.0.4.0/24",
  "Management-JumpBox_Subnet_Name"        = "admin",
  "Management-JumpBox_Subnet_AddressPrefix" = "10.0.5.0/24",
  "Virtual_Network2-name"                 = "Vnet-2",
  "Vnet2addressPrefix"                    = "20.0.0.0/16",
  "Location-of-Virtual_Network2"          = "East US",
  "subnet1-Name-Vnet2"                   = "subnet-1-replica",
  "subnet1-AddressPrefix-Vnet2"           = "20.0.1.0/24",
  "subnet2-Name-Vnet2"                   = "subnet-2-replica",
  "subnet2-AddressPrefix-Vnet2"           = "20.0.2.0/24",
  "subnet3-Name-Vnet2"                   = "subnet-3-replica",
  "subnet3-AddressPrefix-Vnet2"           = "20.0.3.0/24",
  "firewallSubnet-AddressPrefix-Vnet2"    = "20.0.4.0/24",
  "Management-JumpBox_Subnet_Name-Vnet2"  = "admin-replica",
  "Management-JumpBox_Subnet_AddressPrefix-Vnet2" = "20.0.5.0/24",
  //"enableDdosProtection"                 = false,
  //"virtual-Network-Access"                = true,
  //"forwarded-Traffic"                     = null,
  //"gateway-Transit"                       = null,
  //"remoteGateways"                       = null,
}
deployment_mode = "Incremental"
}

```



Secondly, by creating pure Terraform HCL script for WEB-API-DB Architecture without using ARM template

```
resource "azurerm_resource_group" "deploy" {
  count = "${length(var.location)}"
  name = "${element(var.resource_group_name,count.index)}"
  location = "${element(var.location,count.index)}"
}

1 references
resource "azurerm_virtual_network" "deploy" {
  count = "${length(var.location)}"
  name = "${element(var.vnet_name,count.index)}"
  address_space = ["${element(var.address_space,count.index)}"]
  location = "${element(var.location,count.index)}"
  resource_group_name = "${element(var.resource_group_name,count.index)}"

  subnet {
    name = "${element(var.subnet1_name,count.index)}"
    address_prefix = "${element(var.subnet1-address,count.index)}"
    security_group = "${element(azurerm_network_security_group.deploy1.*.id,count.index)}"
  }
  subnet {
    name = "${element(var.subnet2_name,count.index)}"
    address_prefix = "${element(var.subnet2-address,count.index)}"
    security_group = "${element(azurerm_network_security_group.deploy2.*.id,count.index)}"
  }
  subnet {
    name = "${element(var.subnet3_name,count.index)}"
    address_prefix = "${element(var.subnet3-address,count.index)}"
    security_group = "${element(azurerm_network_security_group.deploy3.*.id,count.index)}"
  }
  subnet {
    name = "${element(var.Management-JumpBox_Subnet_Name,count.index)}"
    address_prefix = "${element(var.Management-JumpBox_Subnet_AddressPrefix,count.index)}"
  }
  subnet {
    //count = "${length(var.location)}"
    name = "AzureFirewallSubnet"
    address_prefix = "${element(var.firewallSubnet-AddressPrefix-Vnet1,count.index)}"
  }
}
```

```

resource "azurerm_network_security_group" "deploy1" {
  count = "${length(var.location)}"
  name = "${element(var.NSG-name1,count.index)}"
  location = "${element(var.location,count.index)}"
  resource_group_name = "${element(var.resource_group_name,count.index)}"
  security_rule {
    name = "HTTPorHTTPS-request"
    protocol = "Tcp"
    source_address_prefix = "*"
    source_port_range = "*"
    destination_address_prefix = "${element(var.subnet1-address,count.index)}"
    destination_port_ranges = ["80","443"]
    priority = 100
    direction = "Inbound"
    access = "Allow"
  }
  security_rule {
    name = "SSHAndRDP-to-WEB"
    protocol = "Tcp"
    source_address_prefix = "${element(var.Management-JumpBox_Subnet_AddressPrefix,count.index)}"
    source_port_range = "*"
    destination_address_prefix = "${element(var.subnet1-address,count.index)}"
    destination_port_ranges = ["22","3389"]
    priority = 110
    direction = "Inbound"
    access = "Allow"
  }
  security_rule {
    name = "API-Web-response"
    protocol = "Tcp"
    source_address_prefix = "${element(var.subnet2-address,count.index)}"
    source_port_range = "8083"
    destination_address_prefix = "${element(var.subnet1-address,count.index)}"
    destination_port_range = "8081"
    priority = 120
    direction = "Inbound"
    access = "Allow"
  }
}

```

[For full code you can visit this link: <https://github.com/riyaagrahari/Terraform-Azure/tree/master/Terraform-DR-using-loop-OPTIMIZED> ]

- Implemented Through Pulumi:

In Pulumi modules are made and used for the creation of various resources and objects of this class are created in main file (index.js).

Functions made in classes are called using respective objects and properties of resources are passed as parameters in the function.

Below is some Part of Code in JavaScript for deployment of resources using Pulumi:

```
"use strict";
const pulumi = require("@pulumi/pulumi");
const azure = require("@pulumi/azure");
const resourceGroup = require("./create-rg.js");
const nsgCreate = require("./create-nsg.js");
const virtualNet = require("./create-vnet.js");
const peering = require("./create-peering.js");

let rg1 = "Resource-Group-Pulumi-1";
let rg2 = "Resource-Group-Pulumi-2-replica";
let location1 = "CentralUS";
let location2 = "WestUS";
let vnet1Name = "Vnet-1Pulumi-Riya";
let vnet1IP = "10.0.0.0/16";
let subnet1Name = "sub-1";
let subnet1IP = "10.0.1.0/24";
let subnet2Name = "sub-2";
let subnet2IP = "10.0.2.0/24";
let subnet3Name = "sub-3";
let subnet3IP = "10.0.3.0/24";
let ManagementJumpboxName = "adminNet";
let ManagementJumpboxIP = "10.0.4.0/24";
let FirewallIP = "10.0.5.0/24";
let vnet2Name = "Vnet-2Pulumi-Riya";
let vnet2IP = "20.0.0.0/16";
let subnet1NameReplica = "sub-1-replica";
let subnet1IPReplica = "20.0.1.0/24";
let subnet2NameReplica = "sub-2-replica";
let subnet2IPReplica = "20.0.2.0/24";
let subnet3NameReplica = "sub-3-replica";
let subnet3IPReplica = "20.0.3.0/24";
let ManagementJumpboxNameReplica = "adminNet-Replica";
let ManagementJumpboxIPReplica = "20.0.4.0/24";
let FirewallIPReplica = "20.0.5.0/24";

let nsgNames = ["WEB", "API", "DB"];
var randomNumber = Math.floor(Math.random() * 10000);
// Create Azure Resource Group object by passing Resource Group name and location in constructor parameter
// Primary Resource Group
let azureResourceGroup = new resourceGroup.ResourceGroup(rg1, location1);
// Secondary Resource Group at different location
let azureResourceGroup2 = new resourceGroup.ResourceGroup(rg2, location2);
```

```
// Create Network Security Group
let nsgobj = new nsgCreate.NetworkSecurityGroup();
// Create Primary Network Security Group
let nsg = [];
```

```
//creation of 3 NSG- web,api,db
for(var i=1;i <= 3;i++)
nsg[i] = nsgobj.NetworkSecurityGroupWeb("Nsg-Azure-Primary"+randomNumber+"-"+nsgNames[i-1],azureResourceGroup.location,
azureResourceGroup.resourceGroupName,
subnet1IP,subnet2IP,subnet3IP,
ManagementJumpboxIP,subnet1IPReplica,subnet2IPReplica,subnet3IPReplica,ManagementJumpboxIPReplica);
// Create Secondary Network Security Group
let nsgReplica = []
//creation of 3 NSG replica- web,api,db
for(var i=1;i <= 3;i++)
nsgReplica[i] = nsgobj.NetworkSecurityGroupWeb("Nsg-Azure-Secondary"+randomNumber+"-"+nsgNames[i-1],azureResourceGroup2.location,
azureResourceGroup2.resourceGroupName,subnet1IP,subnet2IP,
subnet3IP,ManagementJumpboxIP,subnet1IPReplica,subnet2IPReplica,subnet3IPReplica,ManagementJumpboxIPReplica);

const vnetobj = new virtualNet.VirtualNetwork();
const virtualnet = vnetobj.VirtualNetworkCreate(vnet1Name,azureResourceGroup.location,vnet1IP,azureResourceGroup.resourceGroupName,subnet1IP,
... subnet2IP,subnet3IP,ManagementJumpboxIP,
... FirewallIP,subnet1Name,subnet2Name,subnet3Name,ManagementJumpboxName,nsg[1].id,nsg[2].id,nsg[3].id);
const virtualNetworkReplica = vnetobj.VirtualNetworkCreate(vnet2Name,azureResourceGroup2.location,vnet2IP,azureResourceGroup2.resourceGroupName,
... subnet1IPReplica,subnet2IPReplica,subnet3IPReplica,ManagementJumpboxIPReplica,FirewallIPReplica,subnet1NameReplica,subnet2NameReplica,
... subnet3NameReplica,ManagementJumpboxNameReplica,nsgReplica[1].id,nsgReplica[2].id,+nsgReplica[3].id);

const peeringobj = new peering.VirtualNetworkPeering();
const peeringConnection = peeringobj.VnetPeering(virtualnet.id,virtualnet.name,virtualNetworkReplica.id,
... virtualNetworkReplica.name,azureResourceGroup.resourceGroupName,
... azureResourceGroup2.resourceGroupName);
```

## Vnet Creation in Pulumi

```
"use strict";

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

class VirtualNetwork {
  constructor(){}

  VirtualNetworkCreate(vNetName, location, Vnetaddr, rgName, subAddr1, subAddr2, subAddr3, subAddr4, subAddr5,
    subname1, subname2, subname3, subname4, nsgID1, nsgID2, nsgID3)
  {
    const testVirtualNetwork = new azure.network.VirtualNetwork(vNetName,
    {
      name: vNetName,
      addressSpaces: [Vnetaddr],
      location: location,
      // name: vNetName,
      resourceGroupName: rgName,
      subnets: [
        {
          addressPrefix: subAddr1,
          name: subname1,
          securityGroup: nsgID1,
        },
        {
          addressPrefix: subAddr2,
          name: subname2,
          securityGroup: nsgID2,
        },
        {
          addressPrefix: subAddr3,
          name: subname3,
          securityGroup: nsgID3,
        },
        {
          addressPrefix: subAddr4,
          name: subname4,
        },
        {
          addressPrefix: subAddr5,
          name: "AzureFirewallSubnet",
        },
      ],
    });
    return testVirtualNetwork;
  }
}

module.exports.VirtualNetwork = VirtualNetwork;
```

Similarly separate classes are created for NSG-creation and Peering. Two objects for Vnet and NSG class are created in index.js - one in primary location and the other in secondary location (objects for resources are created in different resource groups) for disaster recovery solution. Then Peering is created between Primary and Secondary Virtual Networks.

[For full code you can visit this link : <https://github.com/riyaagrahari/IaC-using-Pulumi/tree/master/Create-Vnet-NSG-Subnet-DisasterRecovery> ]

## 6. RESULTS

Installation and Mode of Execution are discussed in the documentation of Github:

Refer following link for detailed procedure:

<https://github.com/riyaagrahari/ARM->

[Templates/tree/master/NestedTemplate\\_for\\_DisasterRecovery](#)

<https://github.com/riyaagrahari/IaC-using-Pulumi/tree/master/Create-Vnet-NSG-Subnet->

[DisasterRecovery](#)

[https://github.com/riyaagrahari/Terraform-Azure/tree/master/Terraform\\_Disaster\\_Recovery](https://github.com/riyaagrahari/Terraform-Azure/tree/master/Terraform_Disaster_Recovery)

### AZURE PORTAL - Resources Created

NAME	TYPE	LOCATION	
NSG-1-api	Network security group	West US	...
NSG-1-api-replica	Network security group	East US	...
NSG-1-db	Network security group	West US	...
NSG-1-db-replica	Network security group	East US	...
NSG-1-web	Network security group	West US	...
NSG-1-web-replica	Network security group	East US	...
Vnet-1	Virtual network	West US	...
Vnet-2	Virtual network	East US	...

## TERRAFORM

```
azurerm_network_security_group.deploy3: Still creating... [10s elapsed]
azurerm_network_security_group.deploy3: Creation complete after 12s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg/providers/Microsoft.Network/networkSecurityGroups/NSG2]
azurerm_virtual_network.deploy: Creating...
azurerm_network_security_group.deploy5: Creation complete after 17s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg-replica/providers/Microsoft.Network/networkSecurityGroups/NSG2-replica]
azurerm_network_security_group.deploy4: Creation complete after 18s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg-replica/providers/Microsoft.Network/networkSecurityGroups/NSG1-replica]
azurerm_network_security_group.deploy6: Creation complete after 18s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg-replica/providers/Microsoft.Network/networkSecurityGroups/NSG3-replica]
azurerm_virtual_network.deploy-replica: Creating...
azurerm_virtual_network.deploy: Still creating... [10s elapsed]
azurerm_virtual_network.deploy: Creation complete after 10s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg/providers/Microsoft.Network/virtualNetworks/first-vnet]
azurerm_virtual_network_peering.Peering2: Creating...
azurerm_virtual_network.deploy-replica: Creation complete after 7s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg-replica/providers/Microsoft.Network/virtualNetworks/first-vnet-replica]
azurerm_virtual_network_peering.Peering2: Creation complete after 2s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg-replica/providers/Microsoft.Network/virtualNetworks/first-vnet-replica/virtualNetworkPeerings/Peering2to1]
azurerm_virtual_network_peering.Peering1: Creating...
azurerm_virtual_network_peering.Peering1: Still creating... [10s elapsed]
azurerm_virtual_network_peering.Peering1: Creation complete after 13s [id=/subscriptions/e65ce201-f8c1-4485-8016-5b9a171dad0c/resourceGroups/first-rg/providers/Microsoft.Network/virtualNetworks/first-vnet/virtualNetworkPeerings/Peering1to2]

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.

C:\Users\Riya Agrahari\Desktop\Terraform-Templates\Terraform_Disaster_Recovery>
```

## PULUMI maintains its project and stack on its own dashboard which is as shown below:

cmd Administrator: Command Prompt

Type	Name	Status	Info
+ pulumi:pulumi:Stack	Pulumi-Azure-DR-devDR	created	2 messages
+   az-pulumi:ResourceGroup	Resource-Group-Pulumi-1	created	
+     azure:core:ResourceGroup	Resource-Group-Pulumi-1	created	
+   az-pulumi:ResourceGroup	Resource-Group-Pulumi-2-replica	created	
+     azure:core:ResourceGroup	Resource-Group-Pulumi-2-replica	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-API	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-DB	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-WEB	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-DB	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-WEB	created	
+   azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-API	created	
+   azure:network:VirtualNetwork	Vnet-1Pulumi-Riya	created	
+   azure:network:VirtualNetwork	Vnet-2Pulumi-Riya	created	
+   azure:network:VirtualNetworkPeering	Peering1	created	
+   azure:network:VirtualNetworkPeering	Peering2	created	

Diagnostics:

```
pulumi:pulumi:Stack (Pulumi-Azure-DR-devDR):
  Resource Group Resource-Group-Pulumi-1:location
  Resource Group Resource-Group-Pulumi-2-replica:location
```

Resources:

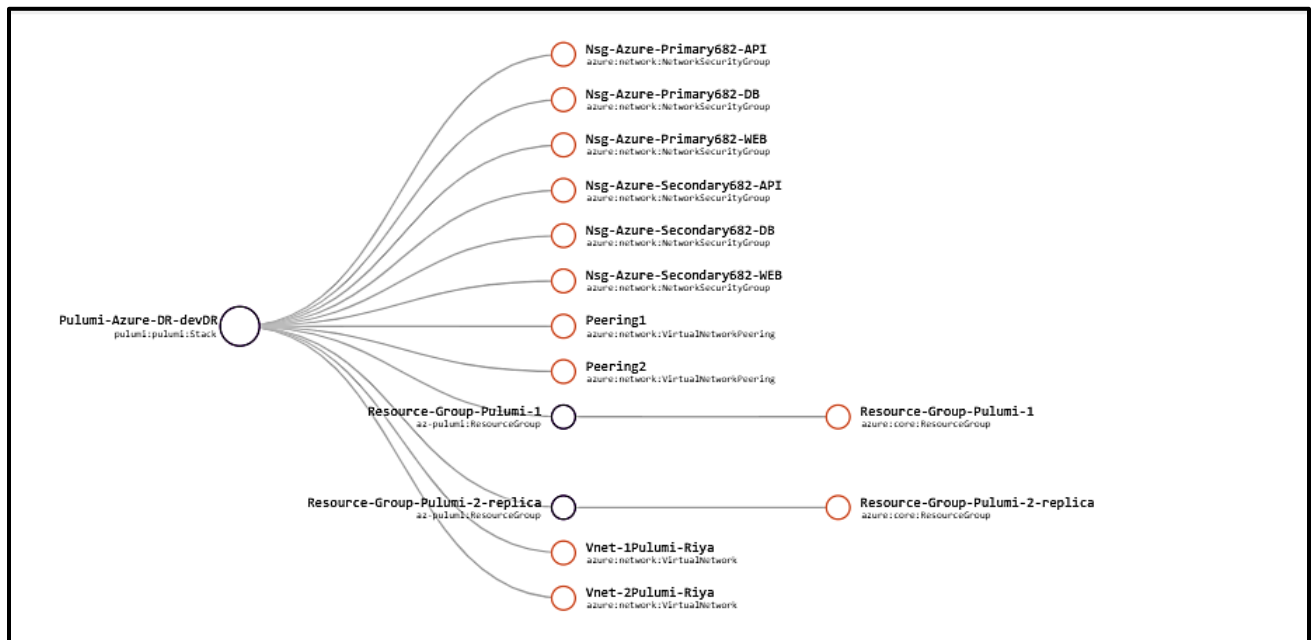
```
+ 15 created
```

Duration: 1m8s

Permalink: <https://app.pulumi.com/riya-agrahari/Pulumi-Azure-DR/devDR/updates/1>

Type	Name ↑	Status
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-API	🔗
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-DB	🔗
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Primary682-WEB	🔗
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-API	🔗
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-DB	🔗
🔗 azure:network:NetworkSecurityGroup	Nsg-Azure-Secondary682-WEB	🔗
🔗 azure:network:VirtualNetworkPeering	Peering1	🔗
🔗 azure:network:VirtualNetworkPeering	Peering2	🔗
🏠 az-pulumi:ResourceGroup	Resource-Group-Pulumi-1	
🏠 azure:core:ResourceGroup	Resource-Group-Pulumi-1	🔗
🏠 az-pulumi:ResourceGroup	Resource-Group-Pulumi-2-replica	
🏠 azure:core:ResourceGroup	Resource-Group-Pulumi-2-replica	🔗
🔗 azure:network:VirtualNetwork	Vnet-1Pulumi-Riya	🔗
🔗 azure:network:VirtualNetwork	Vnet-2Pulumi-Riya	🔗
🏠 pulumi:providers:azure	default	

## RESOURCE-GRAPH





## 7. FUTURE SCOPE

This project in future can be used to deploy real world applications on three-tier architecture with Virtual Machines serving as Web server, Application server and Database Server, developed using Azure resources. Features of this project like disaster recovery can be extended to real world scenarios where the whole architecture could be deployed in the way described in this project.

This project can be extended further to incorporate Load balancing capabilities, deploying both internal load balancing and external traffic load balancing. Load Balancer will help in providing high availability for our services since it balances the incoming traffic from Internet with use of Public Load Balancer and also across the Virtual Machines inside a Virtual Network which serves as Internal Load Balancer. It will also decrease the overall delay due to heavy traffic in provisioning of services. Optimizing the current project working and adding additional services to get maximum throughput with least cost will be our main motive.

## 8. REFERENCES

- <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-authoring-templates>
- <https://docs.microsoft.com/en-us/azure/site-recovery/azure-to-azure-powershell>
- <https://docs.microsoft.com/en-us/azure/role-based-access-control/rbac-and-directory-admin-roles>
- <https://www.pulumi.com/docs/quickstart/azure/>
- <https://www.pulumi.com/docs/reference/languages/>
- <https://docs.microsoft.com/en-us/azure/terraform/>
- <https://www.bmc.com/blogs/software-defined-infrastructure/>