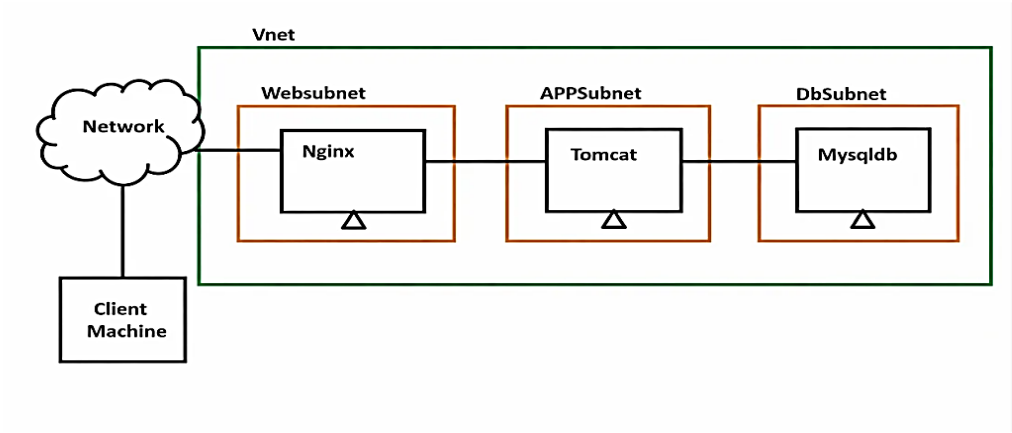


3-Tier Architecture

This project explains how a 3-tier architecture is implemented on Microsoft Azure to create a secure and scalable web application. The architecture is divided into three layers: the Web Server, the Application Server, and the Database Server. Each layer is deployed on a separate Azure Virtual Machine to improve security and make management easier.

The Web Server is the first point of contact for users and handles incoming requests. These requests are then forwarded to the Application Server, which runs Apache Tomcat on port 8080 and processes the application logic. The Database Server uses MySQL on port 3306 to store and manage data. It is placed in a private network so that only the Application Server can access it, keeping the data safe.

Azure services such as Virtual Networks (VNets), subnets, and Network Security Groups (NSGs) are used to control communication between the servers and block unauthorized access.

Requirements:

Azure Free Trail Subscription

Resource Group

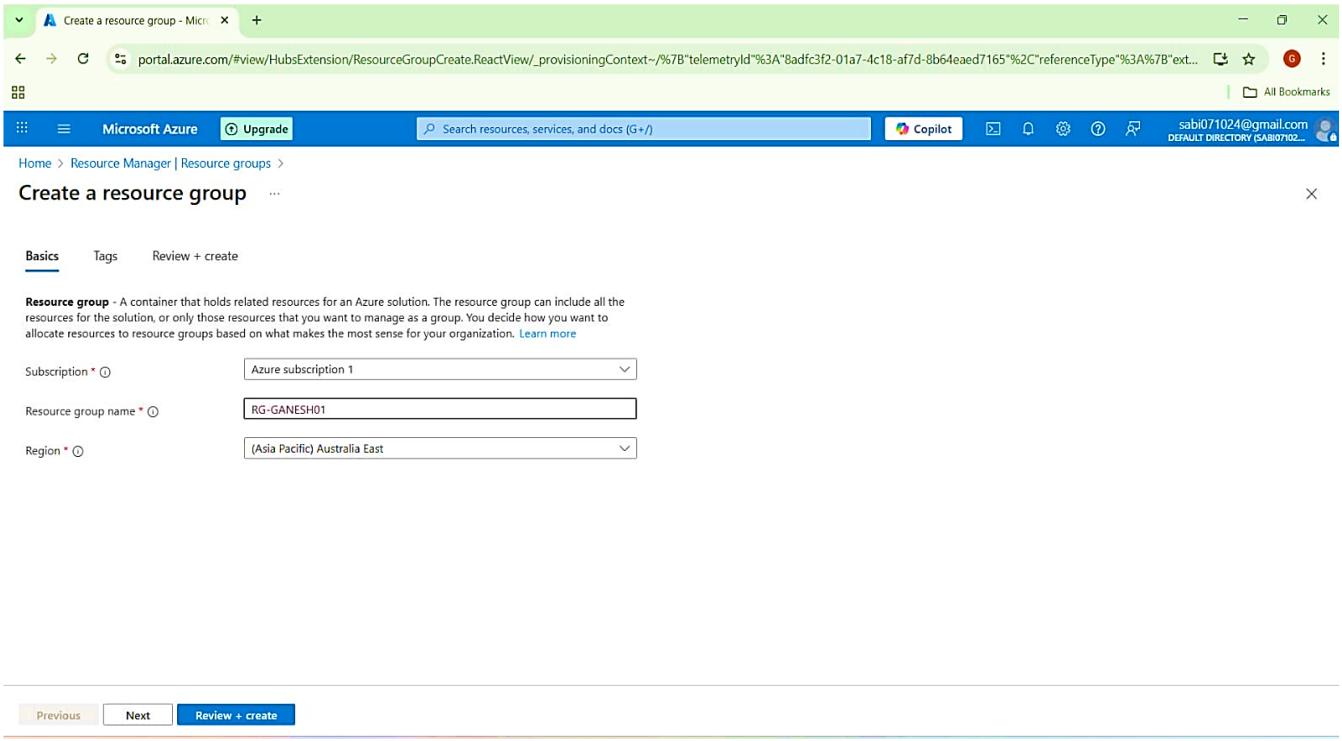
Virtual Network (VNet)

Subnets-WebServer , AppServer , DB server

Network Security Group (NSG)

Steps to build 3 – tier Architecture:

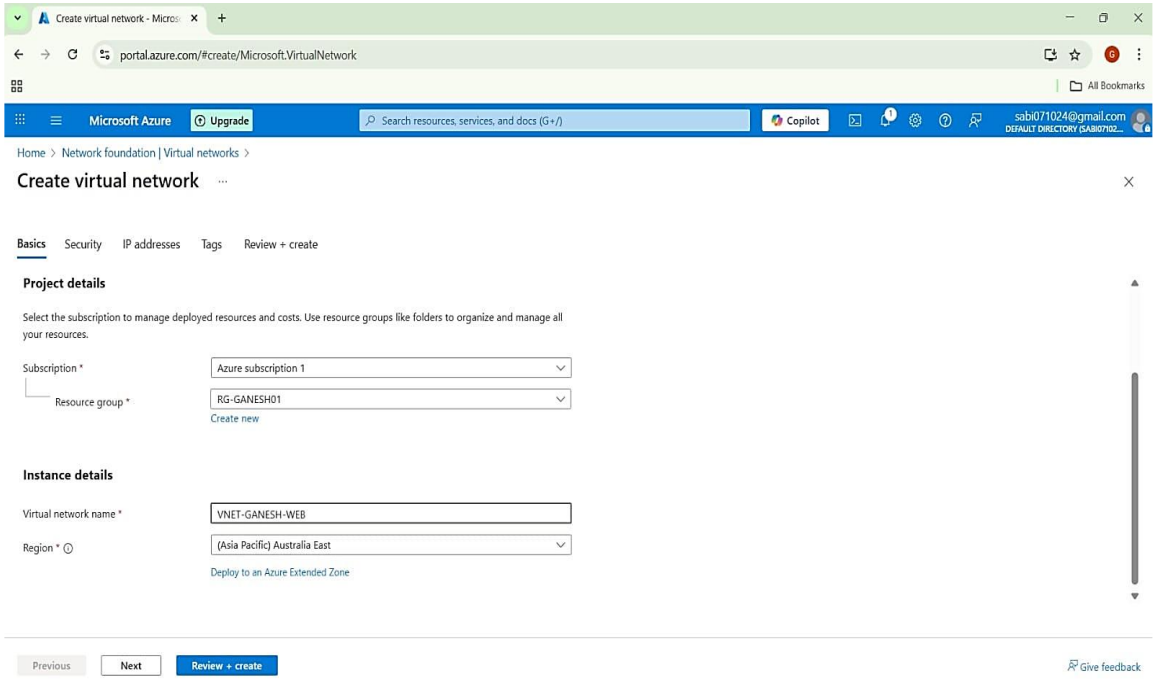
Step 1: Create a Resource group:



Select a region of your choice and assign a name to your Resource Group (RG) and then click on **Review + Create.**

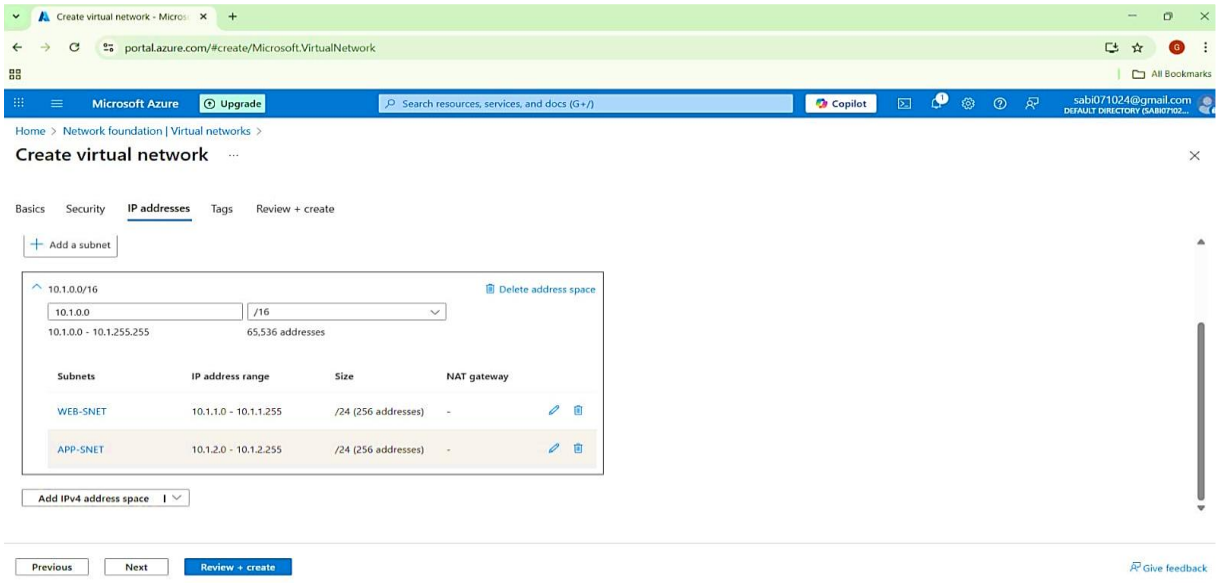
Step 2: Create a Virtual Network (VNet):

Type **Virtual Network** in the search box, then click **Create** to start setting it up.



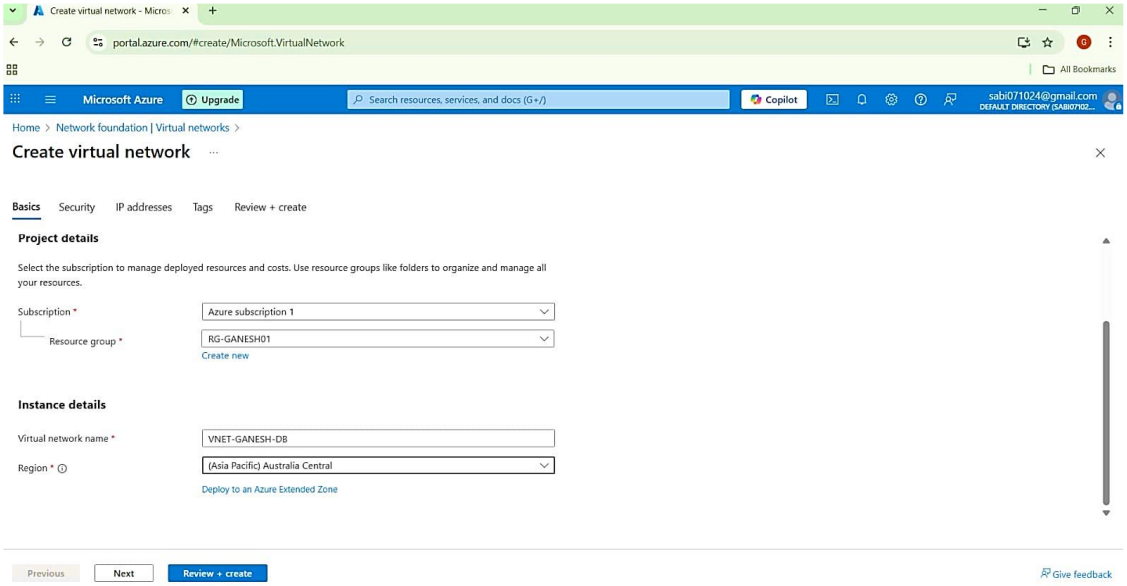
Select the Resource Group in which the Virtual Network will be deployed. Provide a name for the Virtual Network, choose a region, and then click **Next** to proceed to the IP Address section.

Note: The region of the Resource Group and the Virtual Network does not need to be the same. Using different regions does not impact architecture.

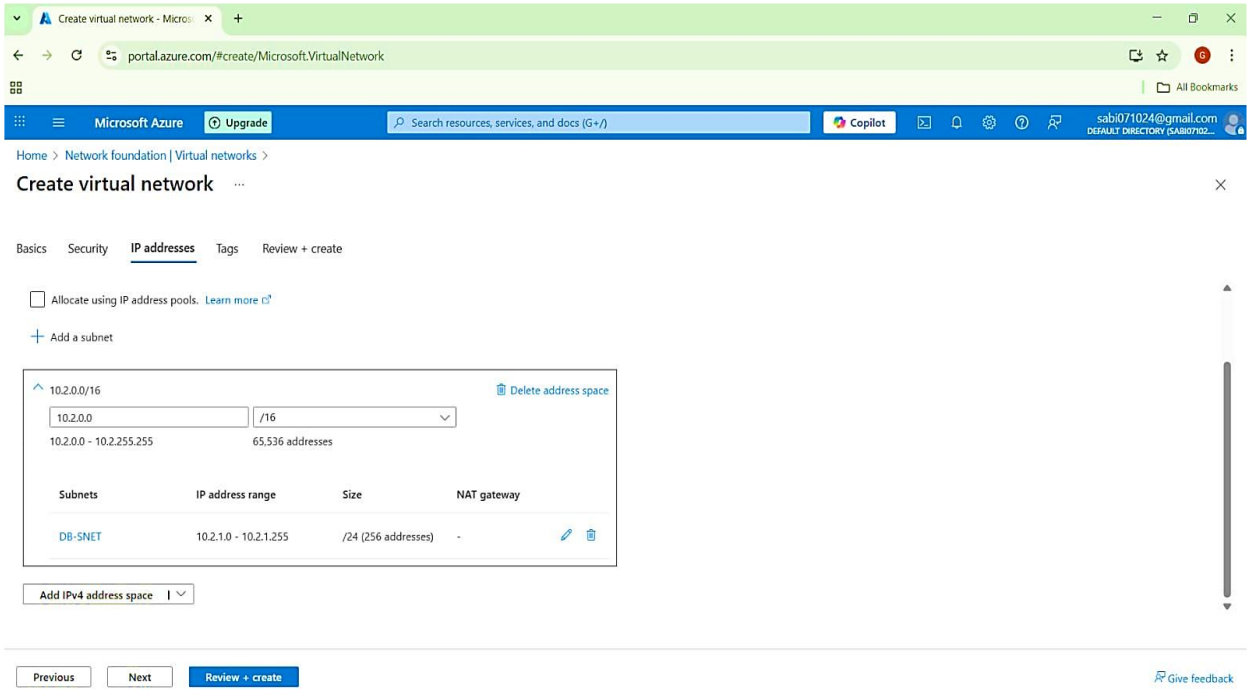


Configure the IP address space and CIDR range as per the architecture requirements. Update the subnet name and settings accordingly.

In above image we configured two subnets, one for webserver and another one for appserver.



Choose a region and click **Next** to navigate to the IP Address configuration page.



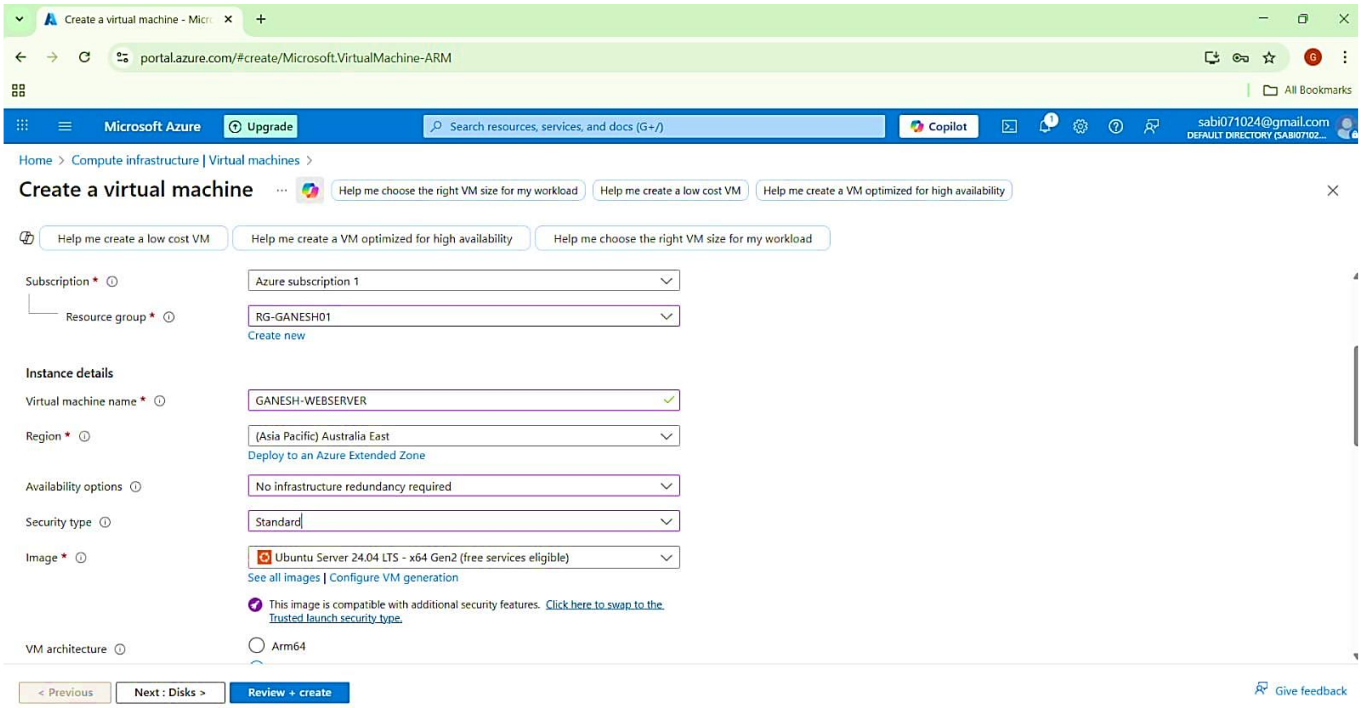
Note: We are creating to VNet's in two different region because, Azure Free trail doesn't provide **1vcpu** service (Total of 4vcpu's). And here we need three Servers so we use to different VNets in two different regions

Step 3: Create Virtual Machines

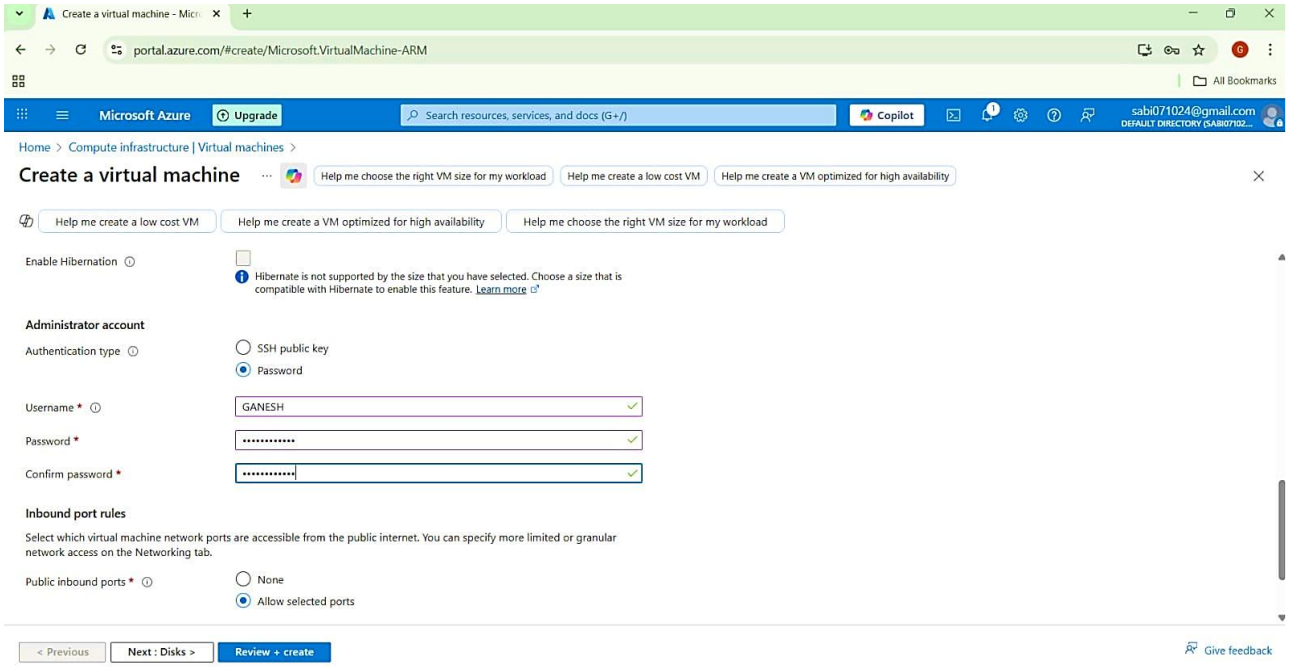
Now its time to create VM's.

WEB-SERVER(VM):

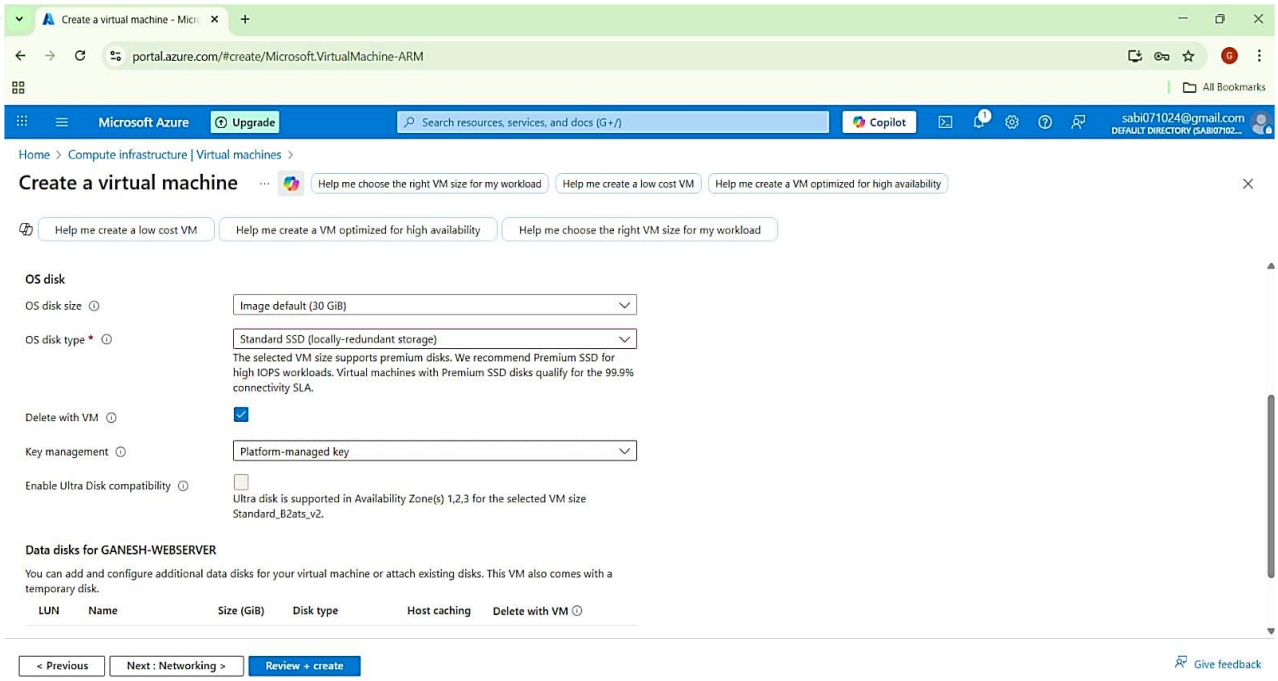
Search for Virtual Machines and click on create and select first option (virtual machine).



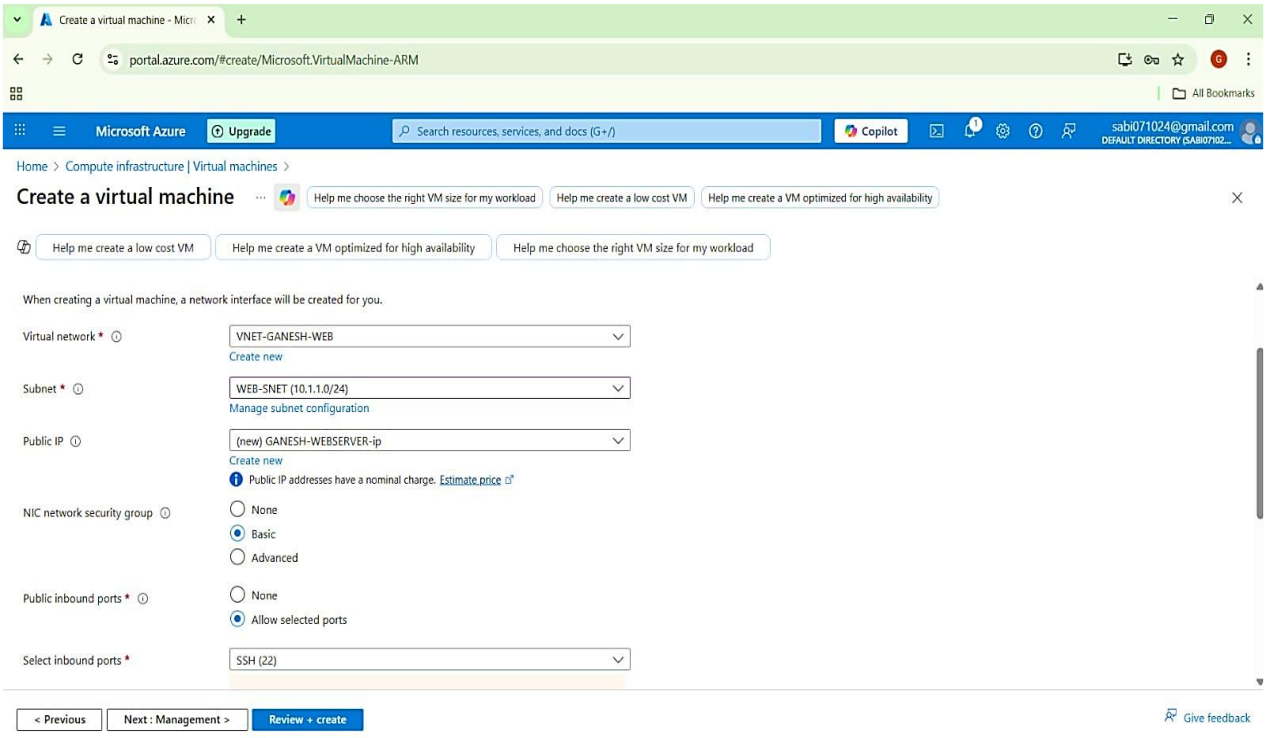
Choose the Resource Group, assign a name to the virtual machine, and select the region in which the initial Virtual Network (GANESH-web-vnet) was created. Configure the necessary availability options, select the Ubuntu image, and scroll down to proceed.



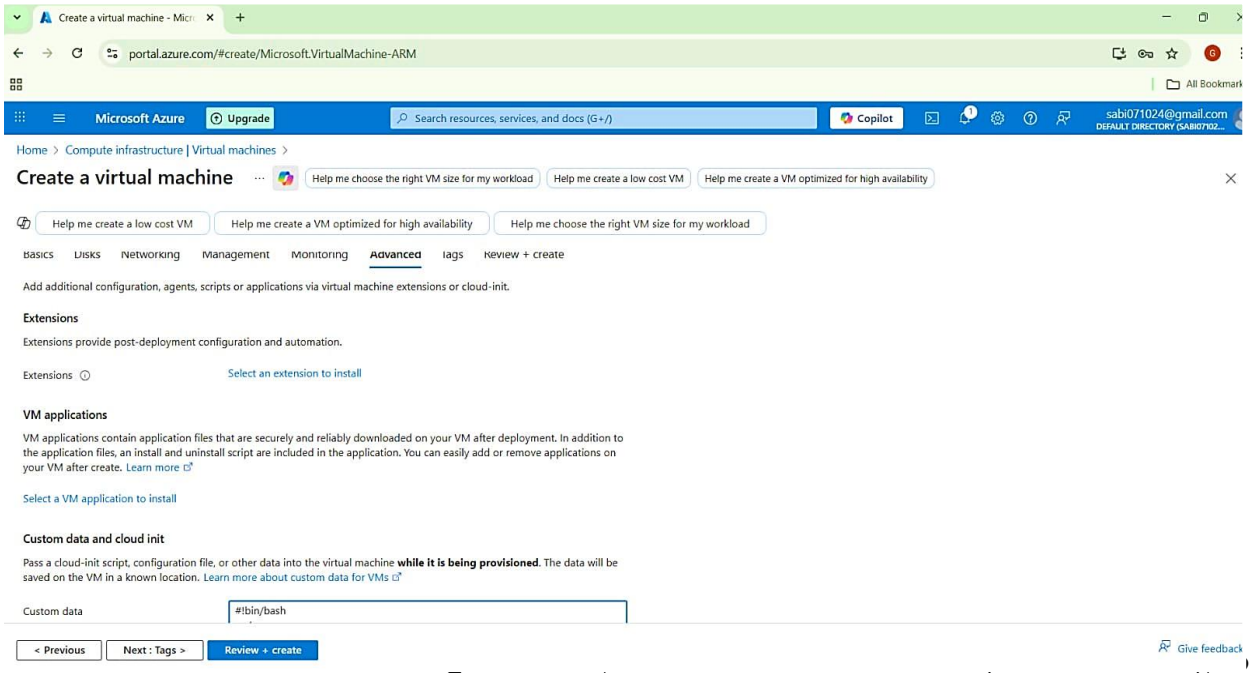
Choose the virtual machine size with at least 2 vCPUs. Select the preferred authentication method—SSH or password (password authentication is used here with a configured username and password). Allow inbound access on the SSH port to connect to the server, and then navigate



Choose the required disk type (SDD is selected here) and click **Next** to proceed.



Select the **VNet** named VNET-GANESH-WEB (created for the web server), choose the appropriate **subnet range**, then click **Next** to proceed to the **Advanced settings** page.



write the commands as shown in custom data:

#!bin/bash

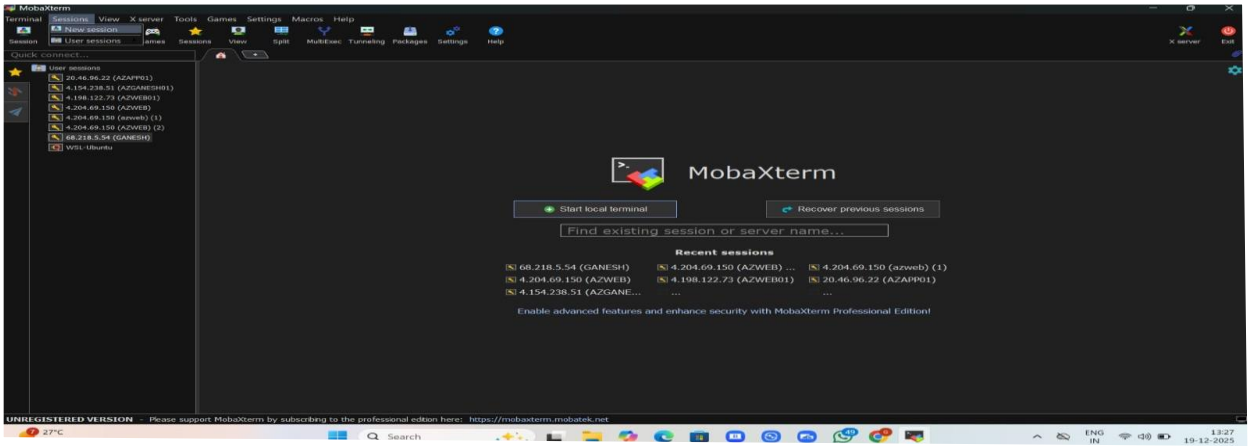
sudo su

apt update

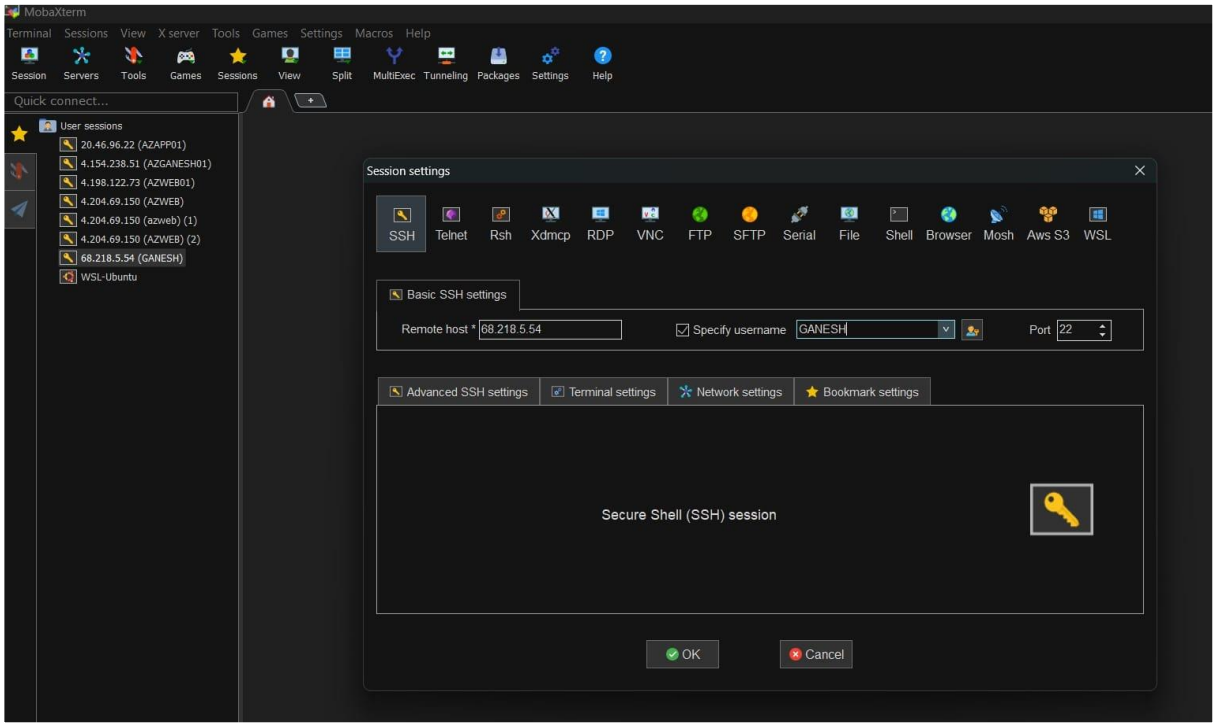
apt install nginx-y

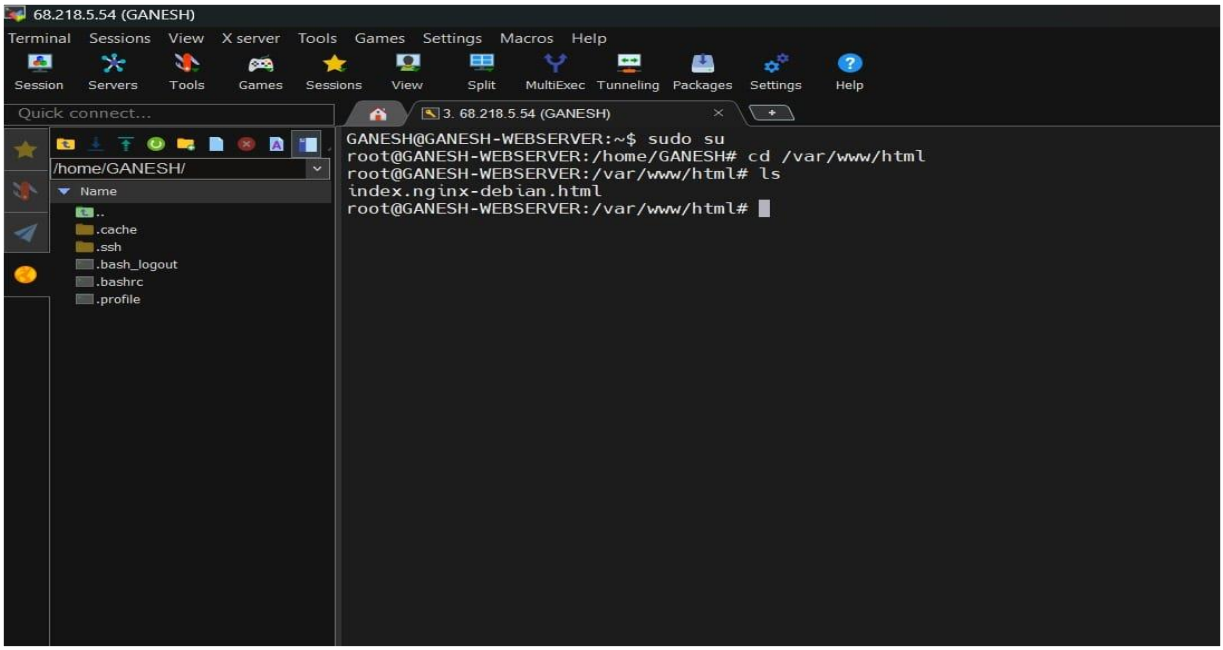
This will automatically install nginx server while creating the VM, and then click on review + create.

Now lets connect to the server and check whether the nginx server is installed or not. Here i used **Mobaxtream,** to connect to the Vm's using SSH and provide the **public ip** of the machine and enter password.



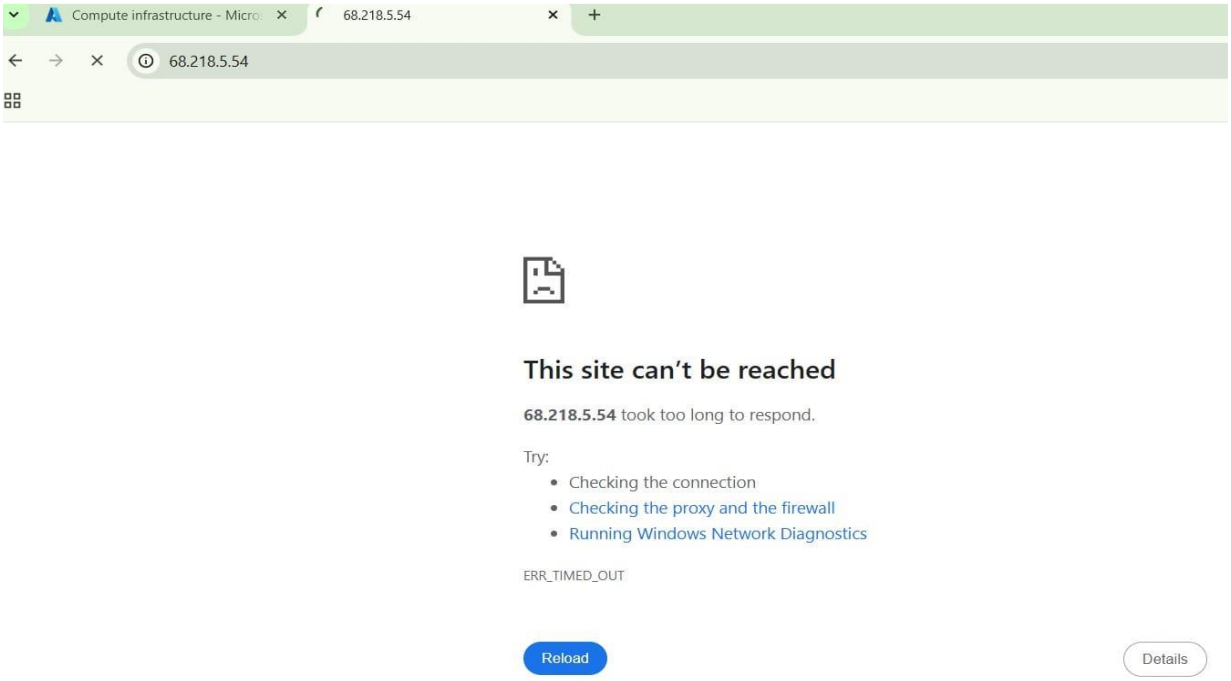
So, in mobaxtreme go to sessions →new session→session settings(click on ssh) and enter remote host and username.





Run the command cd /var/www/html to navigate to the web server's root directory, then use the ls command to verify whether Nginx is installed. As shown in the screenshot, the Nginx server files

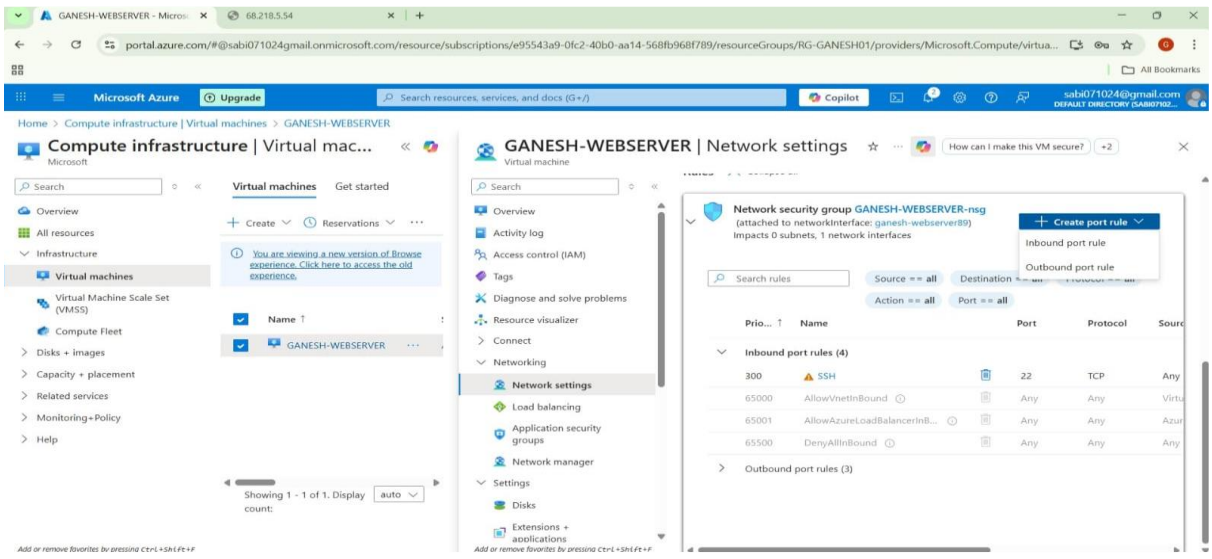
are present. Next, access the server by entering its **public IP address** in a web browser such as Chrome.



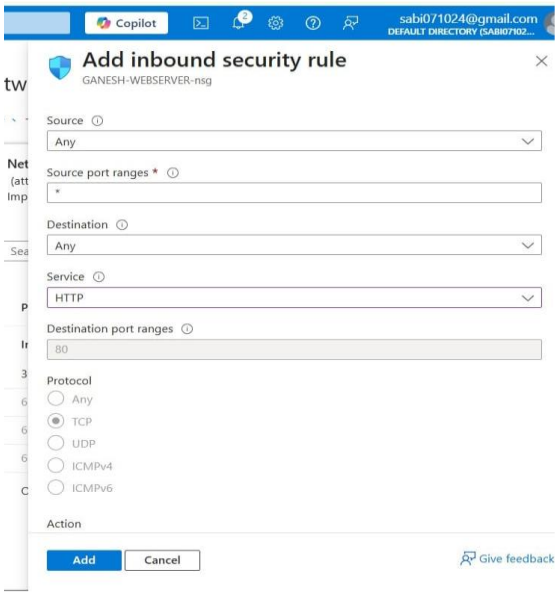
Since the server is currently inaccessible, it is necessary to configure **inbound rules** to allow traffic on **HTTP port 80**.

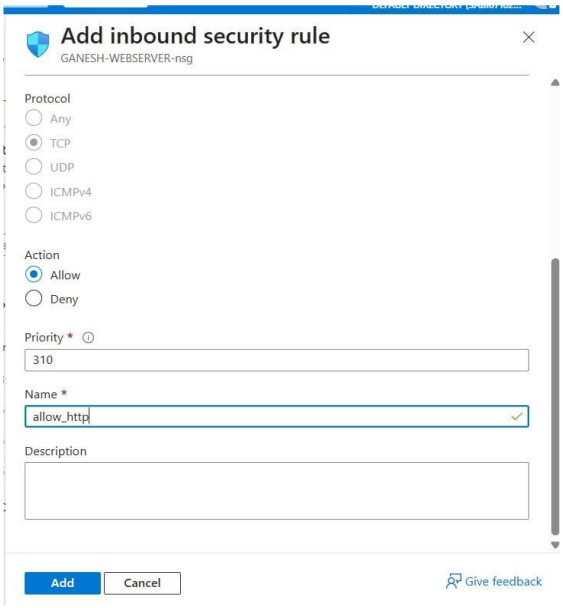
To do this:

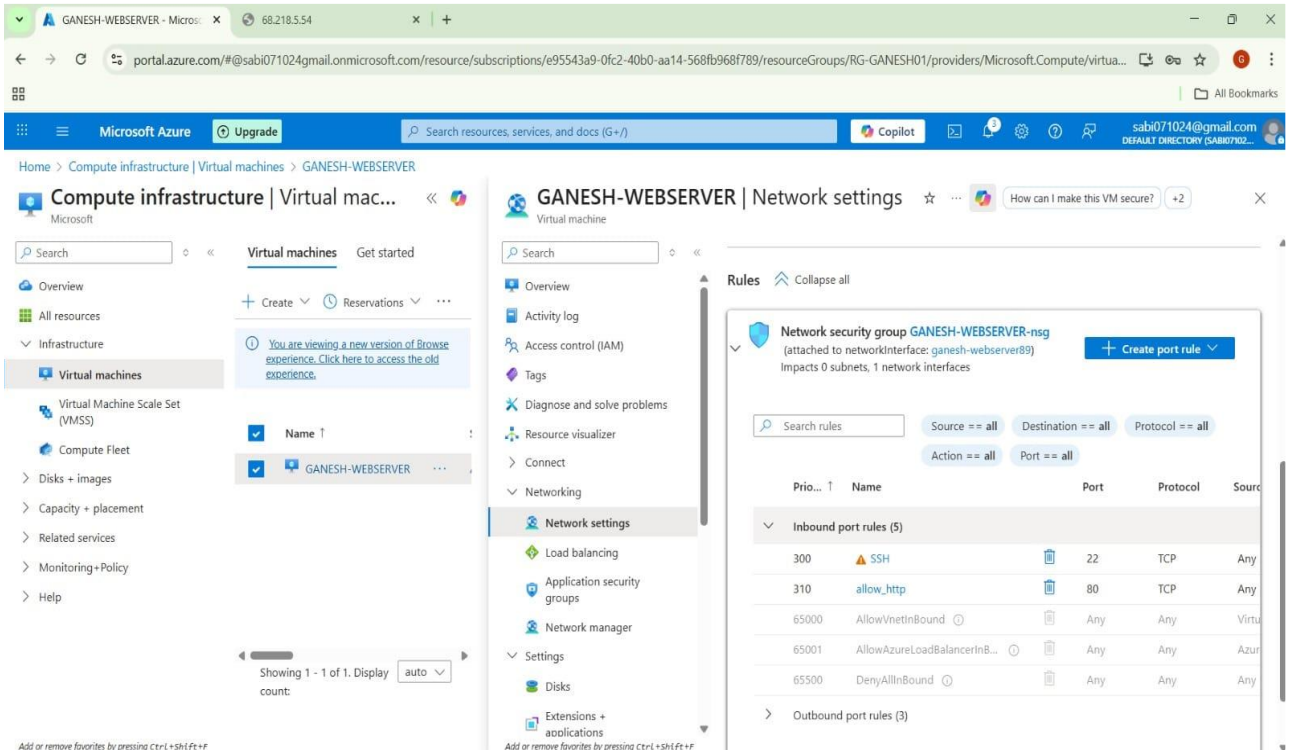
1. Navigate to the **web server** in the Azure portal.
2. Go to **Network Settings** and scroll down to the **Inbound and Outbound Rules** section.
3. Configure the inbound rule to allow **HTTP traffic on port 80** as required, then click **Add** to apply the rule.



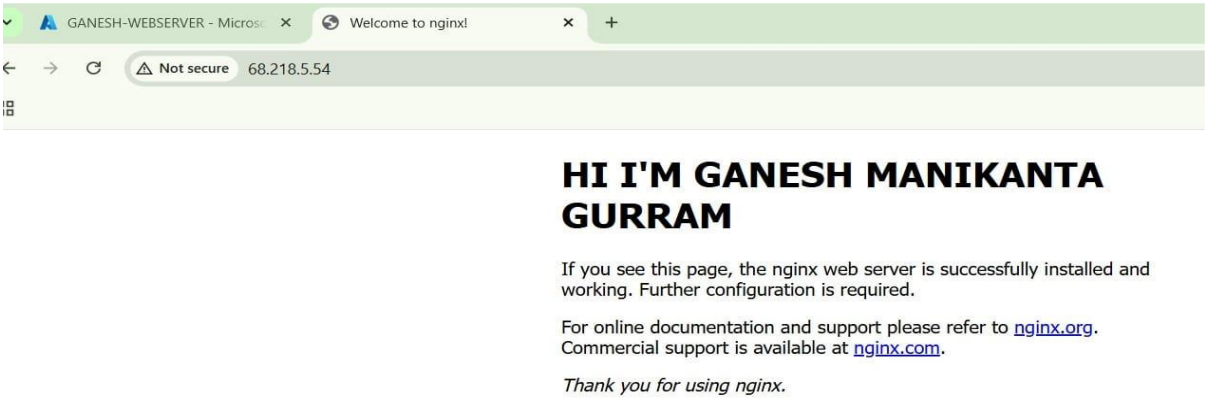
In web server virtual network→go to networking→networking settings→NSG(create port rule)→enter the rule and then save.







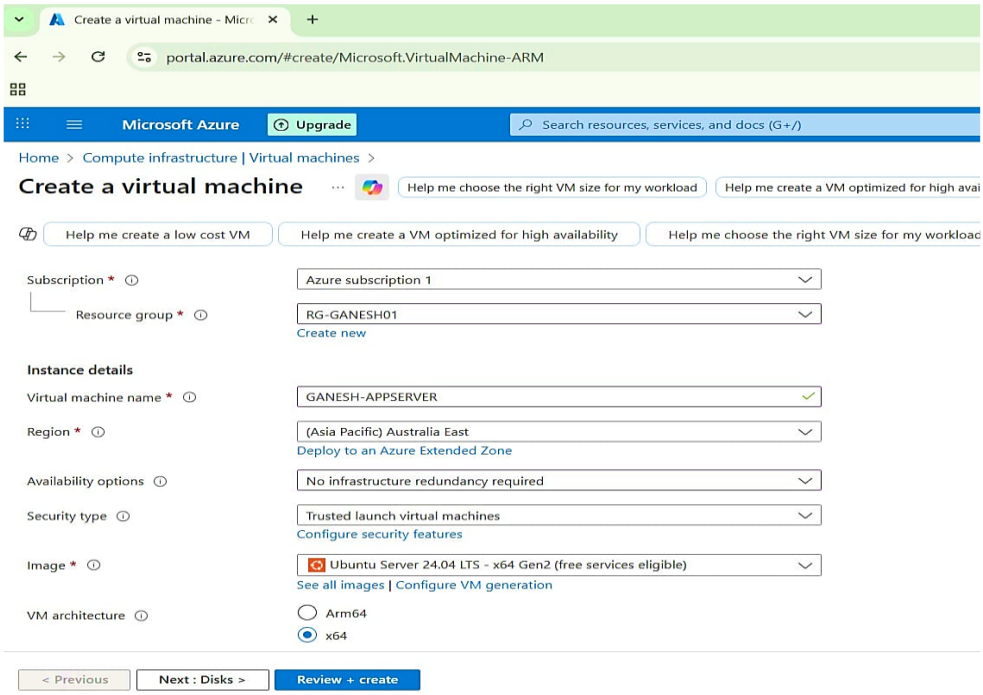
The Nginx server is now publicly accessible. To verify, open a web browser (e.g., Chrome) and enter the server's **public IP address** in the address bar. The default Nginx welcome page should appear, confirming successful access to the server.

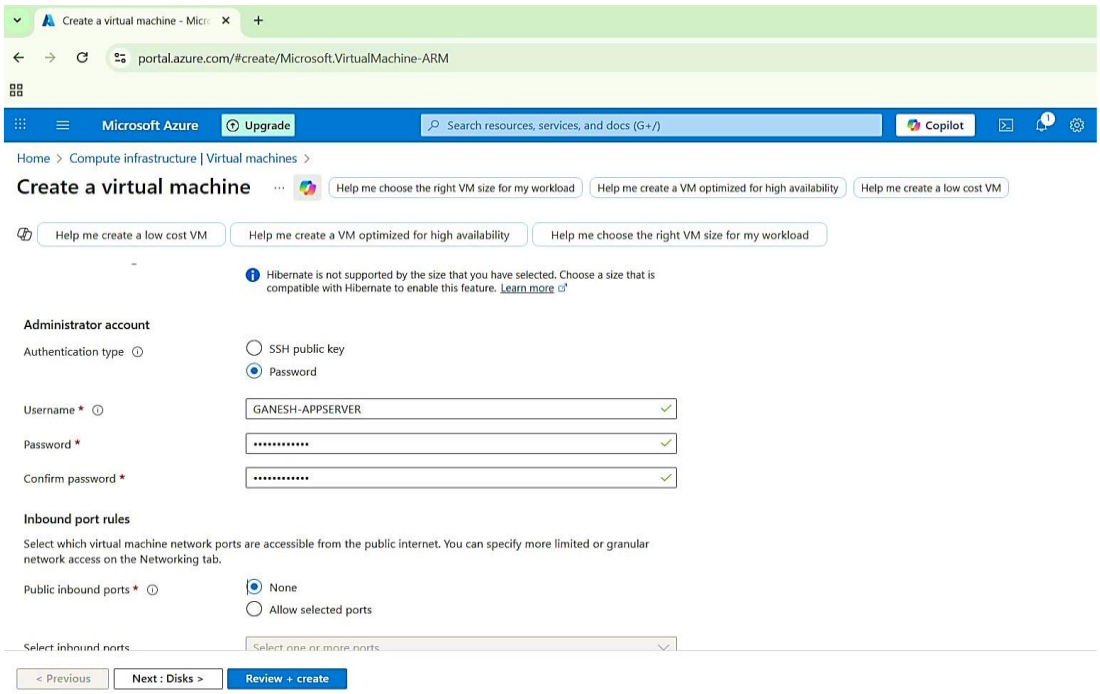


The Nginx server is now accessible.

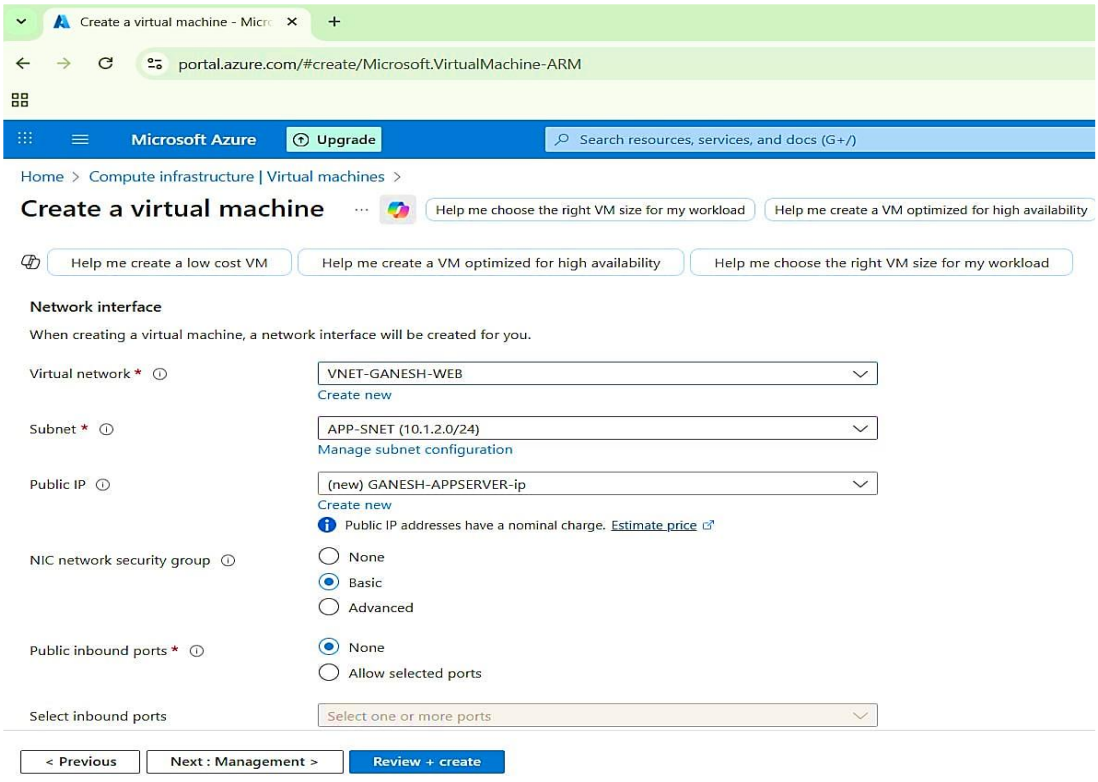
APP SERVER:

Next, we will create the App Server. Search for Virtual Machines in the Azure portal, click Create, and select the first option (Virtual Machine). The creation process is similar to that of the Web Server VM, The region and VNet are also same for the App-server, region(RG-GANESH-01),virtual network(VNET-GANESH-WEB).



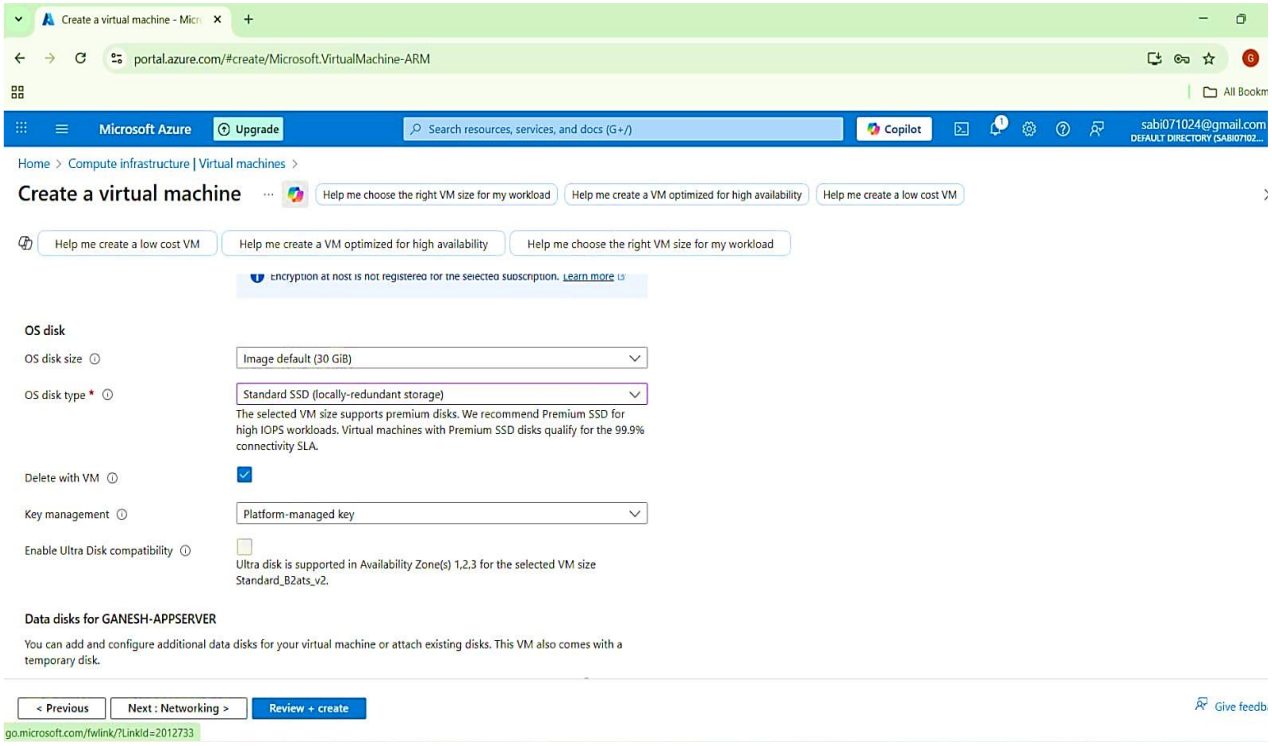


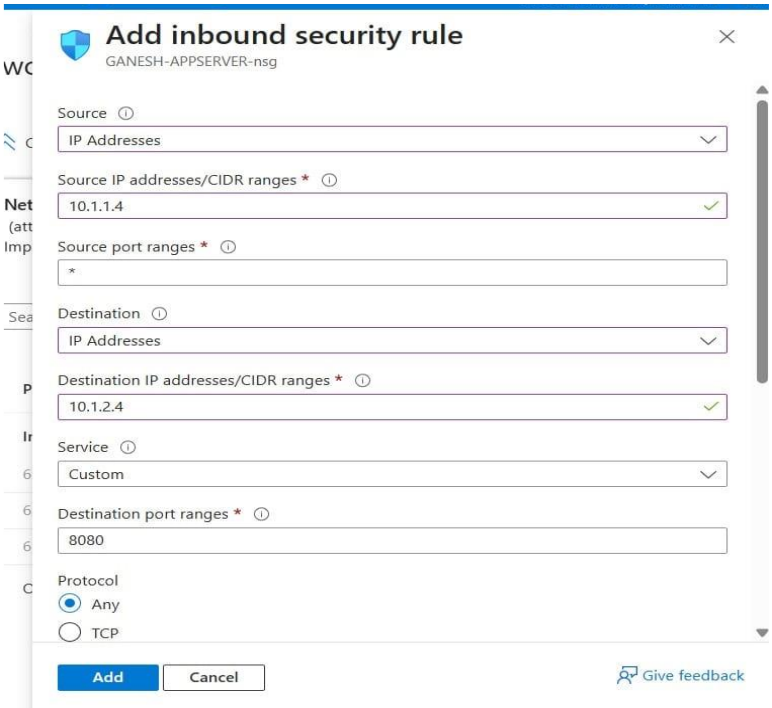
At this stage, all unnecessary ports should be **denied**. Specific inbound rules will be added later as required to allow controlled access. This ensures that the server remains secure during the initial setup.



Select the **VNET-GANESH-WEB** Virtual Network for this server, choose the **APP-SNET(10.1.2.0/24)** subnet, and then click **Review + Create** to proceed with the VM deployment.

Here also I’m selecting my disk as standard sdd. Because it is enough for me.

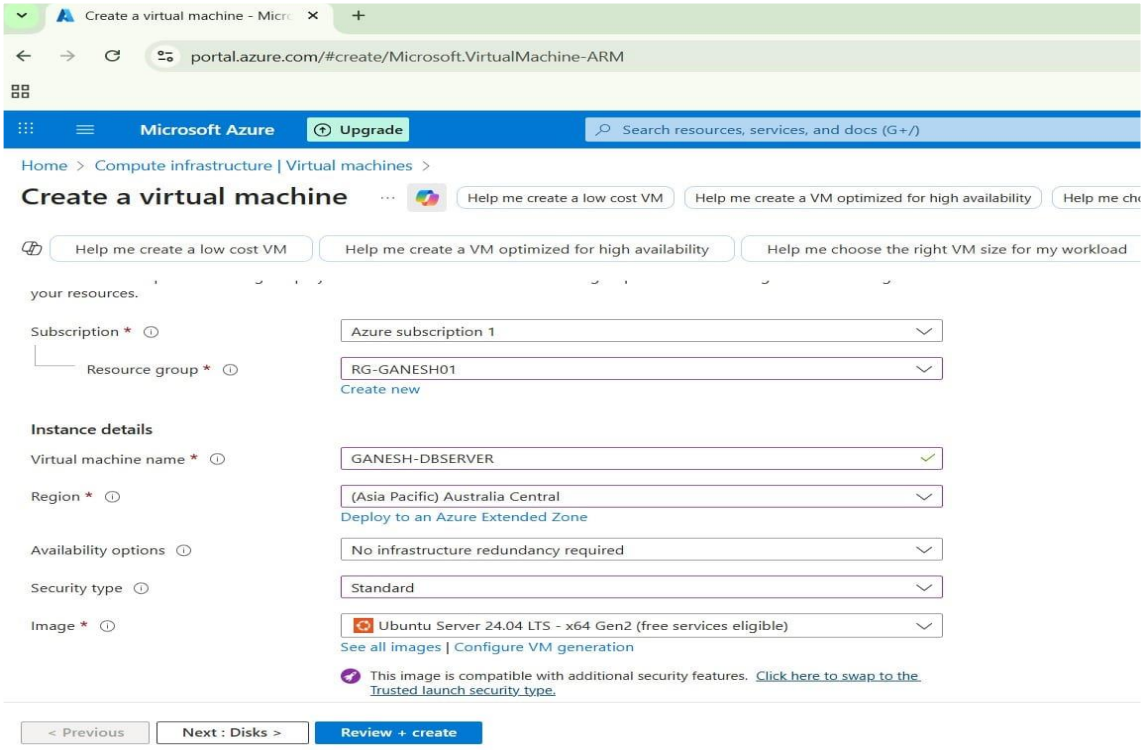


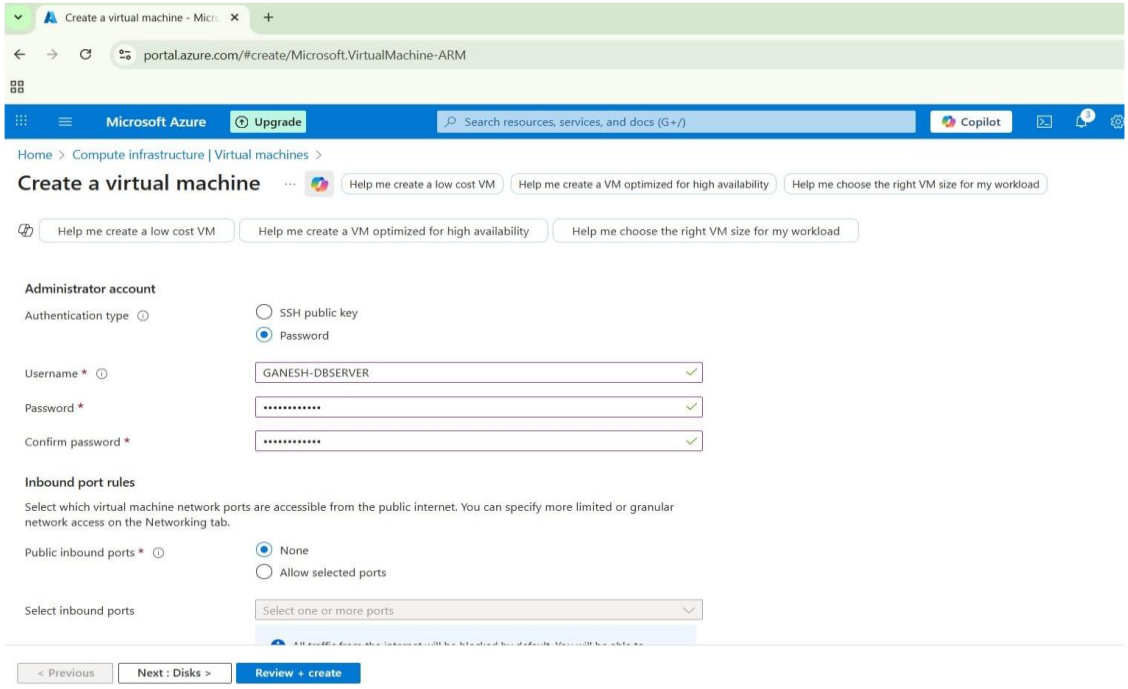


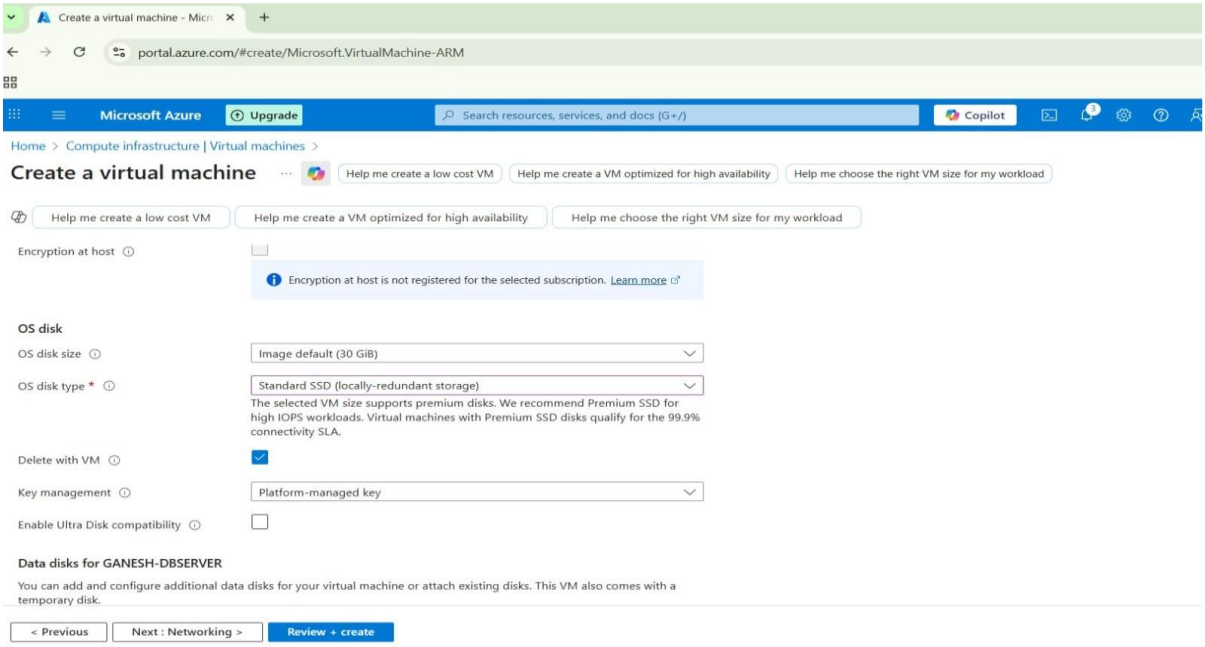
DB SERVER:

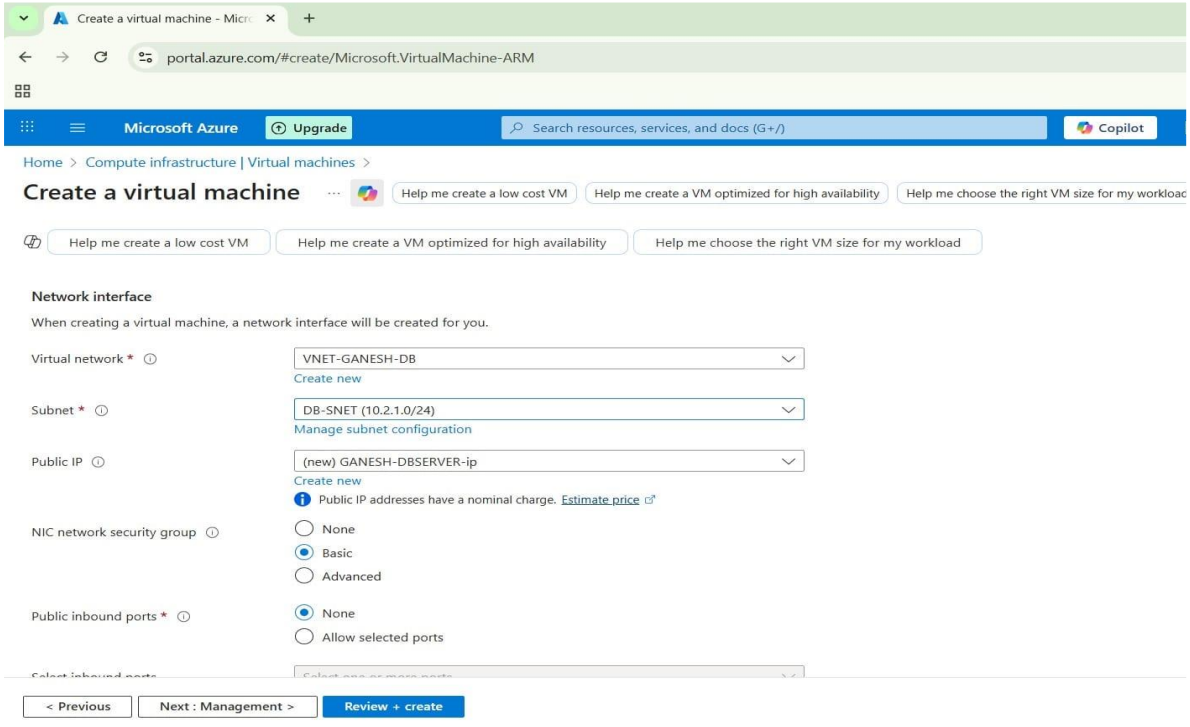
Next, create the **Database Server (DB Server)** VM following the same steps as the App Server. Select the **VNET-GANESH-DB** and assign it to a **DB-SNET(subnet), 10.2.1.0/24**.

Here we are choosing the another VNET rather than to WEB-SERVER AND APP-SERVER because in our free trail we are not able to create multiple(3-machines)machines.So,we choose another virtual network in that region and then we create our machine and later we do virtual network peering to connect.



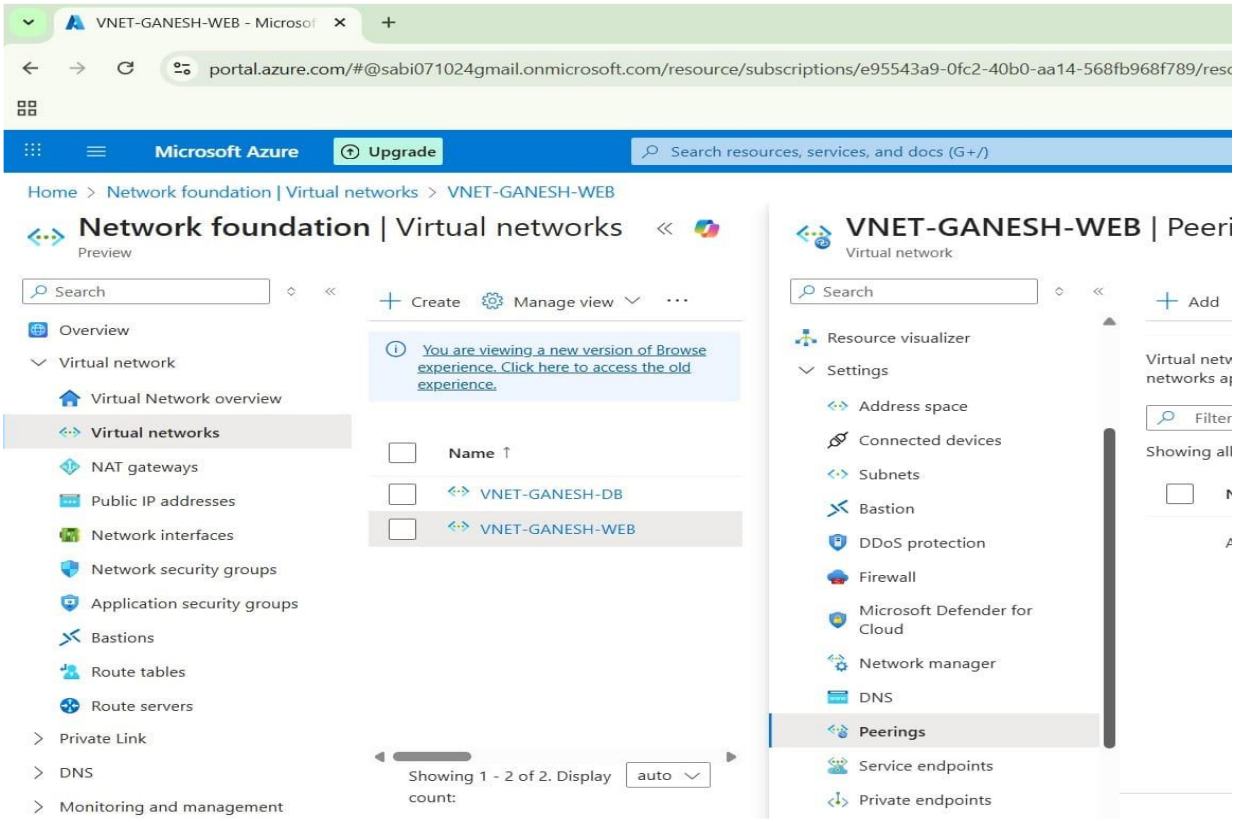






Here we done creating our machine(GANESH-DBSERVER)

PEERING :

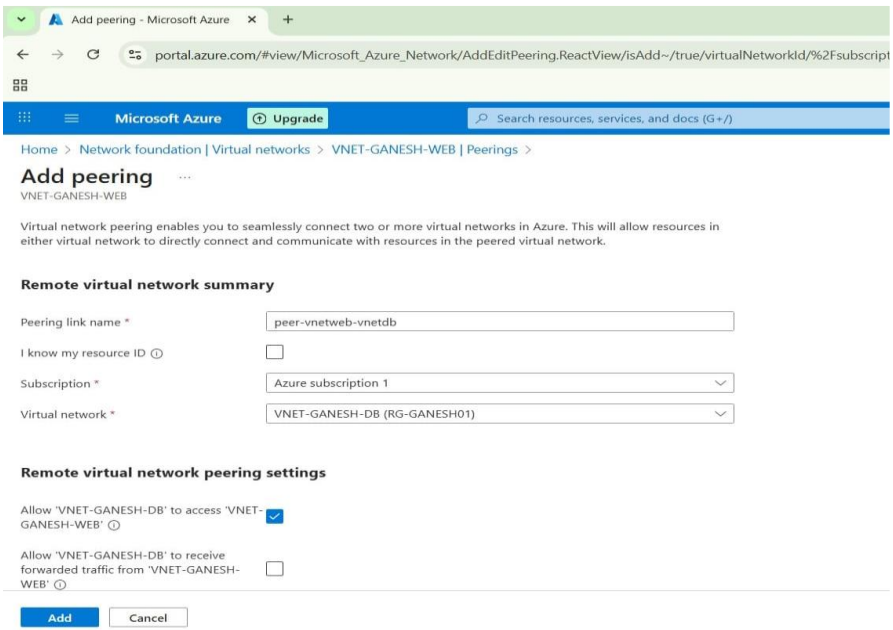


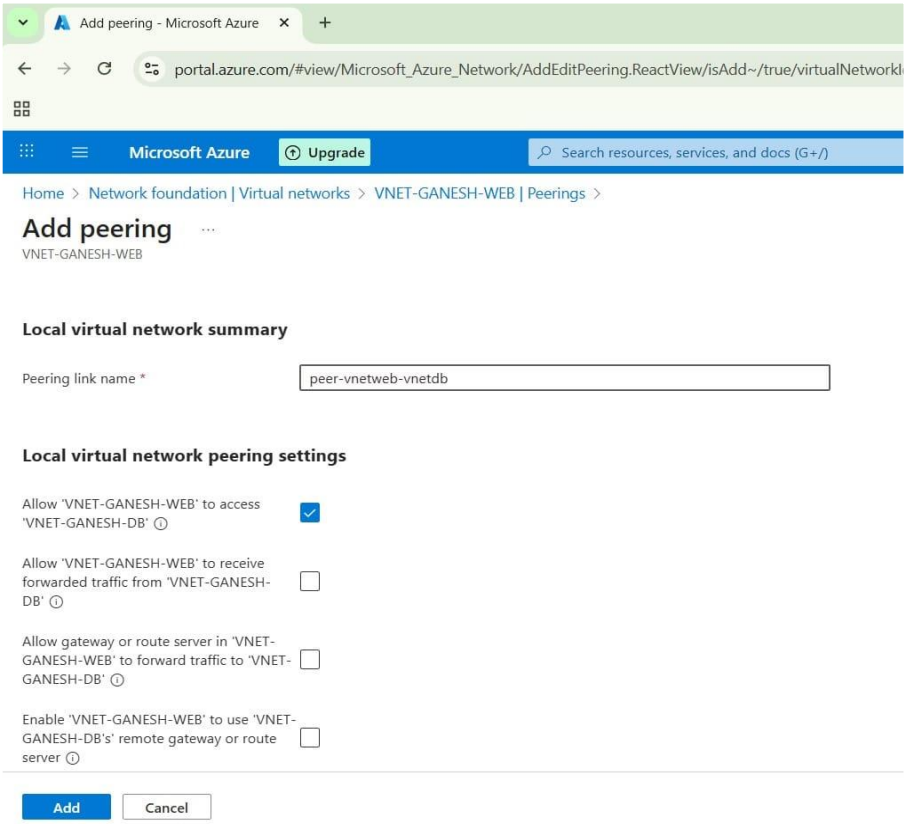
Setting up Peering:

Since the servers are deployed across **two different VNets**, it is essential to enable communication between all three servers. To achieve this, we configure **Network Peering**.

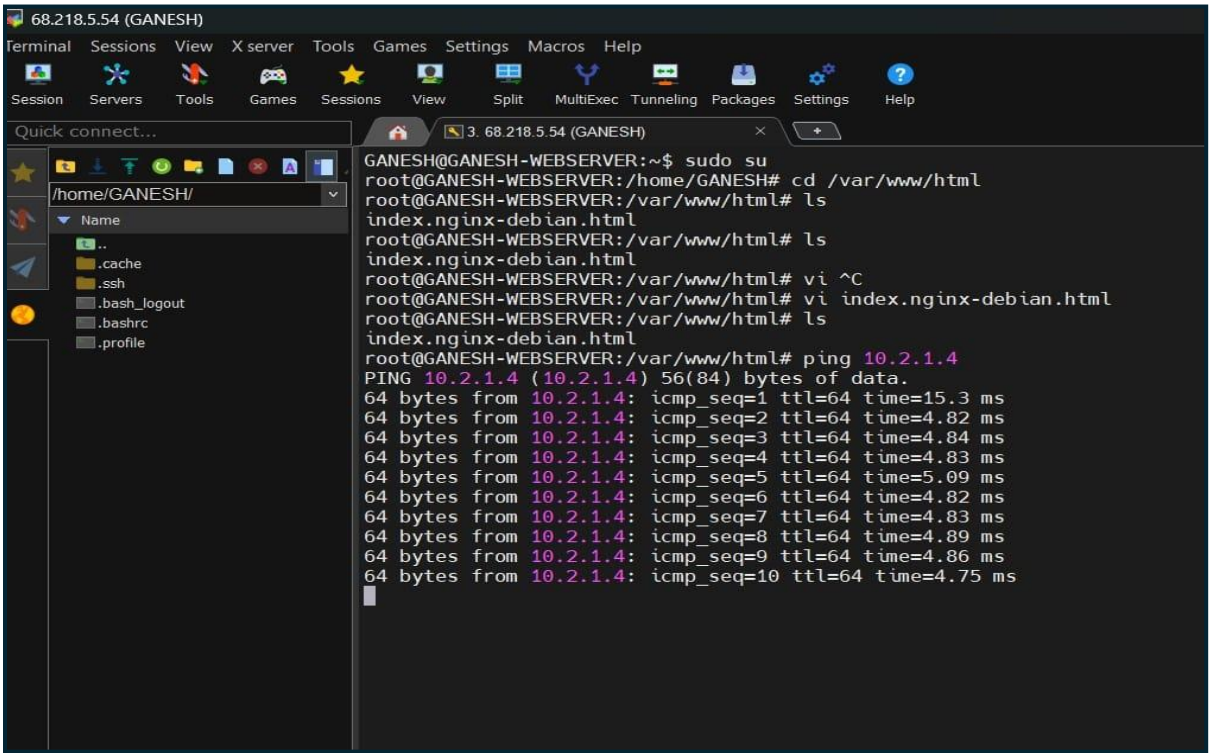
Network Peering:

Network Peering acts as a **bridge between two virtual networks**, allowing resources in different VNets to communicate securely and efficiently as if they were part of the same network.



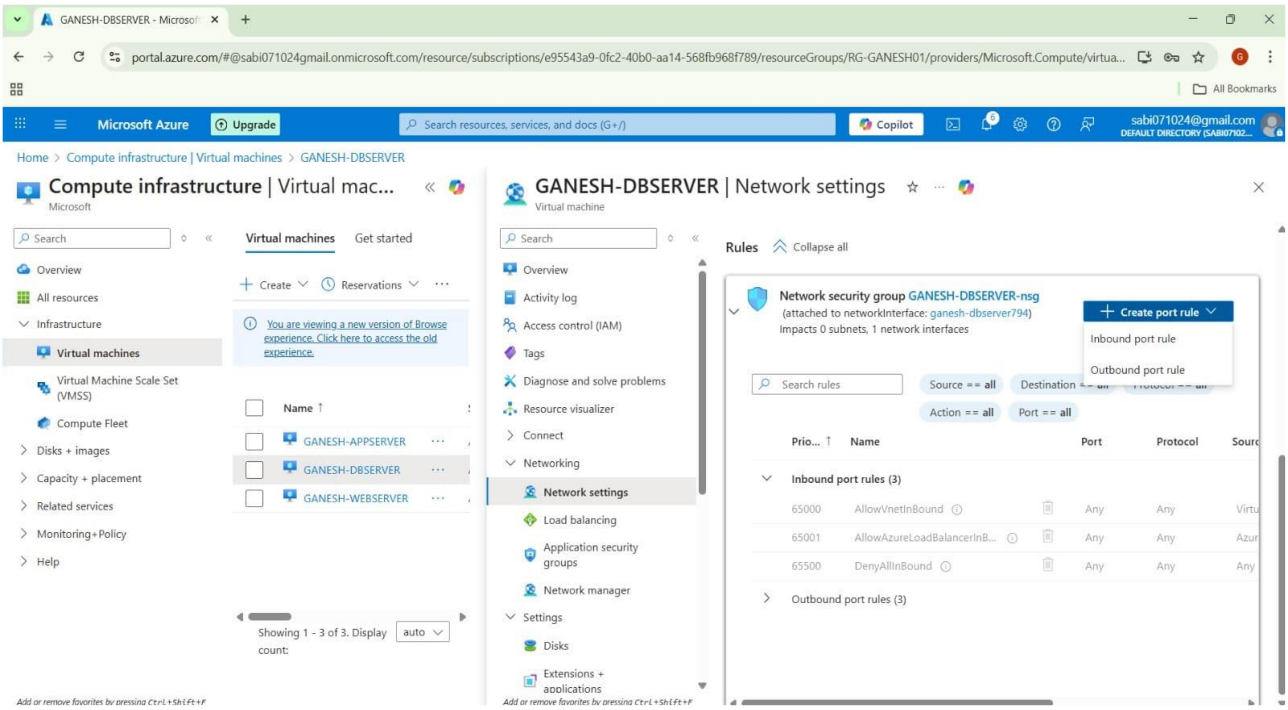


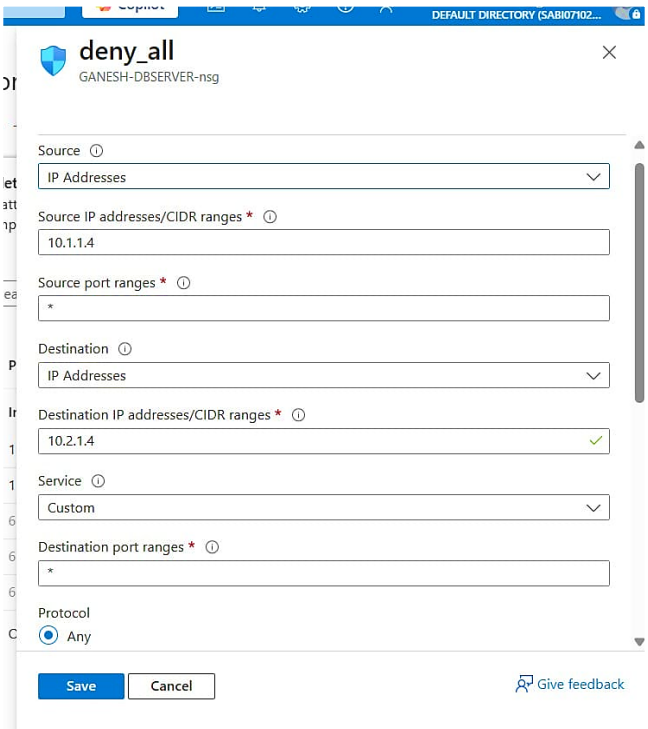
To establish communication between the servers deployed across two different VNets, we configure VNet peering. For example, navigate to ganesh-web-vnet and select the Peering option from the left-hand menu. Provide a name for the peering connection and select the destination VNet, vnet-ganesh-db, then click Add to create the peering. Once the peering is established, connectivity between the servers can be verified using the ping command, ensuring that all servers can communicate across the VNet's.



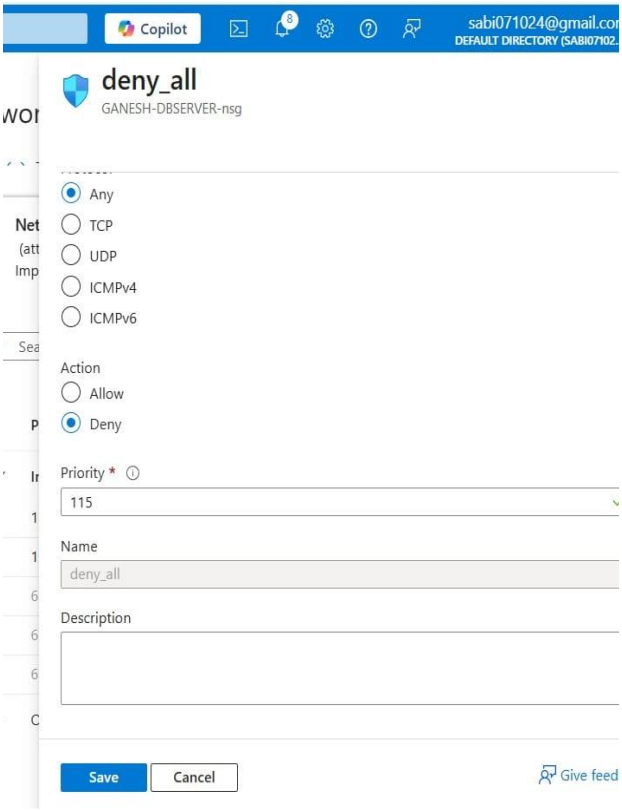
Adding inbound and outbound rules:

Inbound and outbound rules are configured based on the role of each server. The Web Server (Frontend) is intended to be publicly accessible, as it hosts the frontend web page, and therefore the required inbound rules have already been added. The App Server and db server (Backend), should remain private and must not be accessible from the public internet. It should only communicate based on the rules. This can be done as follows:

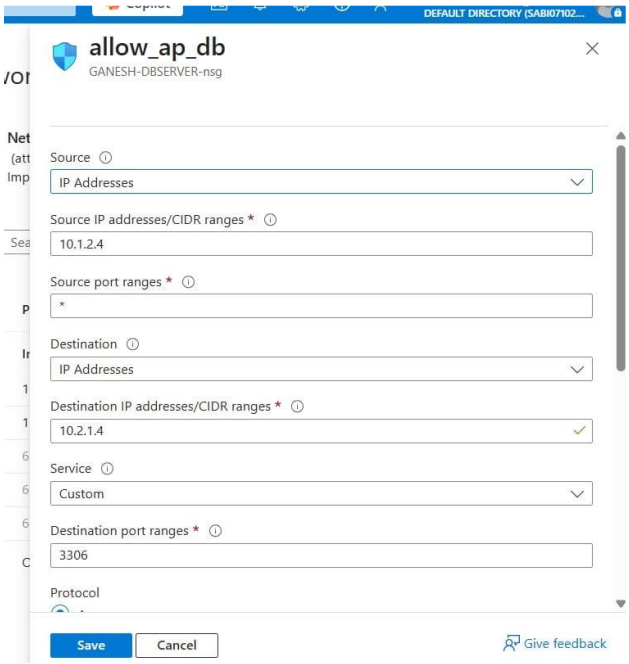


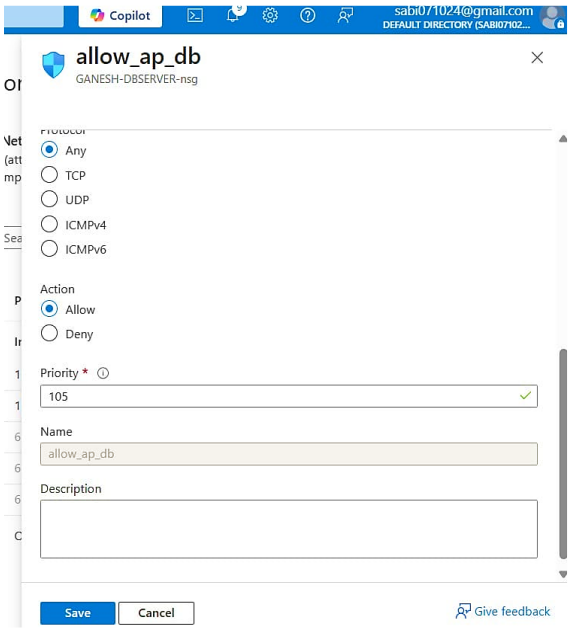


s



Since the **Database Server** communicates only with the **App Server**, inbound and outbound rules must be configured accordingly. The database handles requests and responses from the App Server; therefore, access to the **MySQL port (3306)** is allowed by specifying the **App Server's private IP address as the source** and the **Database Server's private IP address as the destination**. This configuration ensures that only the App Server can communicate with the Database Server over port 3306.



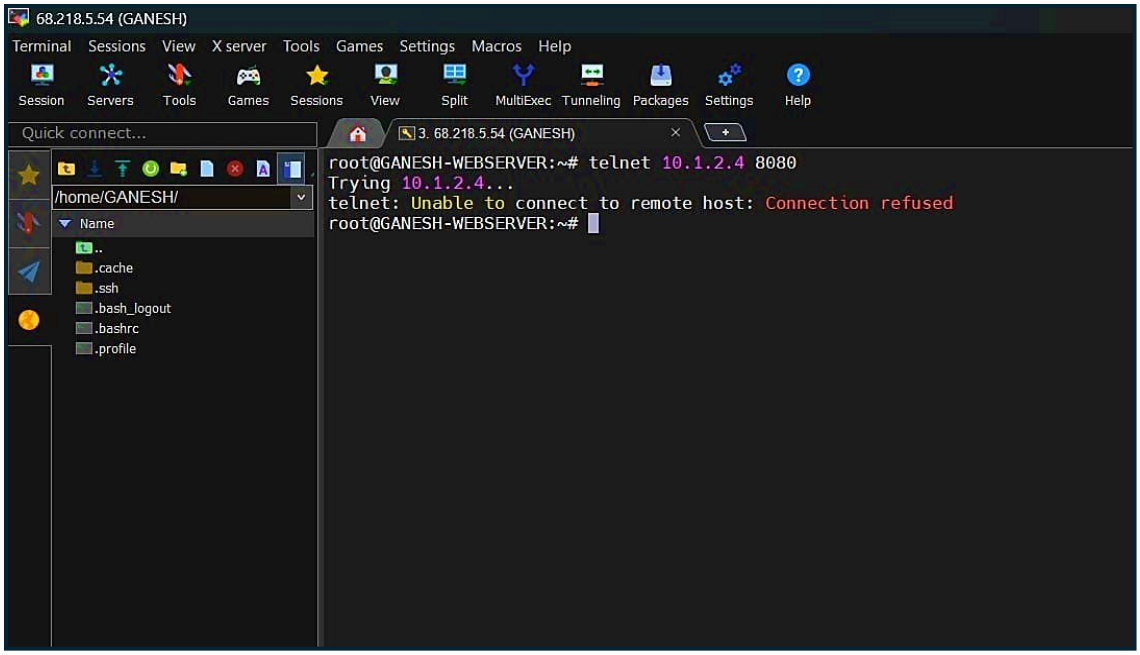


Installing and Configuring Tomcat on the App Server:s

With the setup of all servers completed, the next step is to verify connectivity between the Web Server and the App Server. This can be tested using the command:

telnet 10.1.2.4 8080

At this stage, the connection attempt will fail because Tomcat has not yet been installed on the App Server. This confirms that the application service is not currently running on port 8080 and must be installed and configured before communication can be established.



After verifying connectivity between the Web Server and the App Server, the next step is to install and configure **Apache Tomcat** on the App Server. The App Server can be accessed directly from the Web Server using the SSH command:

ssh username & ip address(private ip) (ssh GANESH-APPSERVER@10.1.2.4)

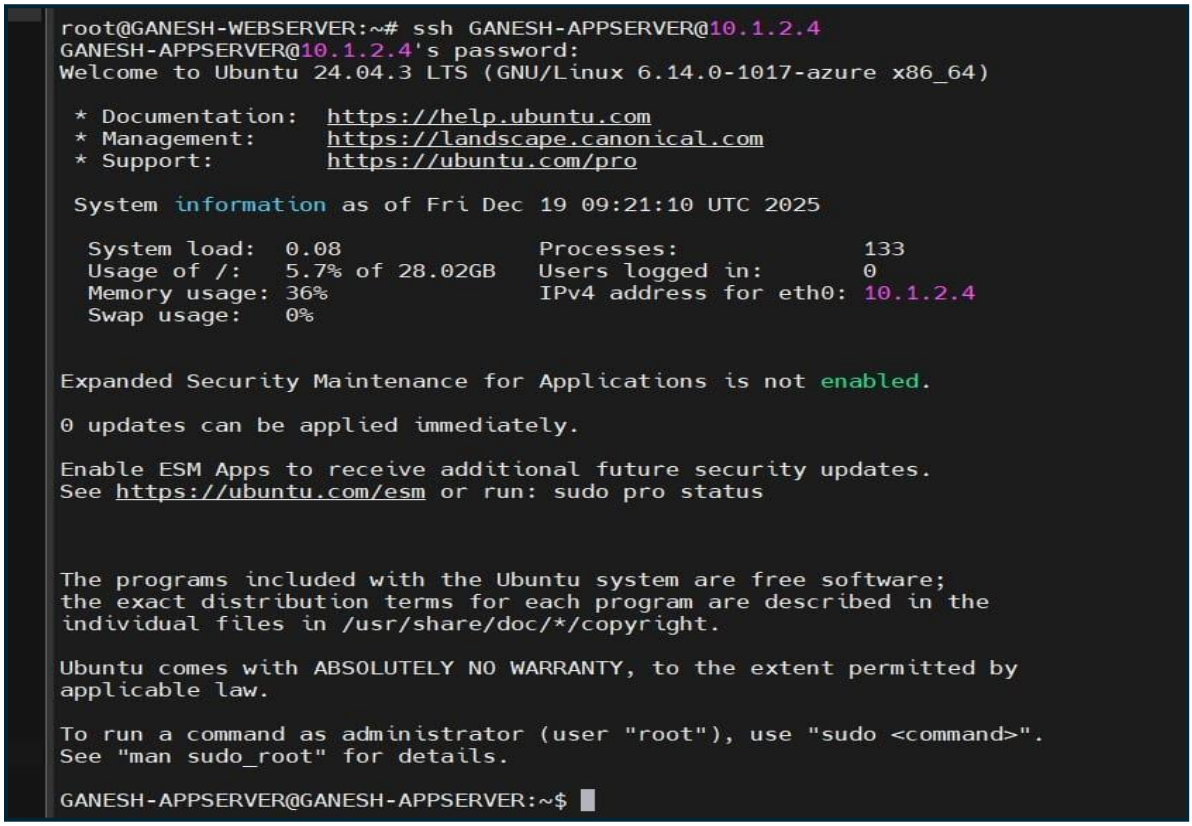
Once connected, switch to the **root user** and execute the following commands to install and configure Tomcat:

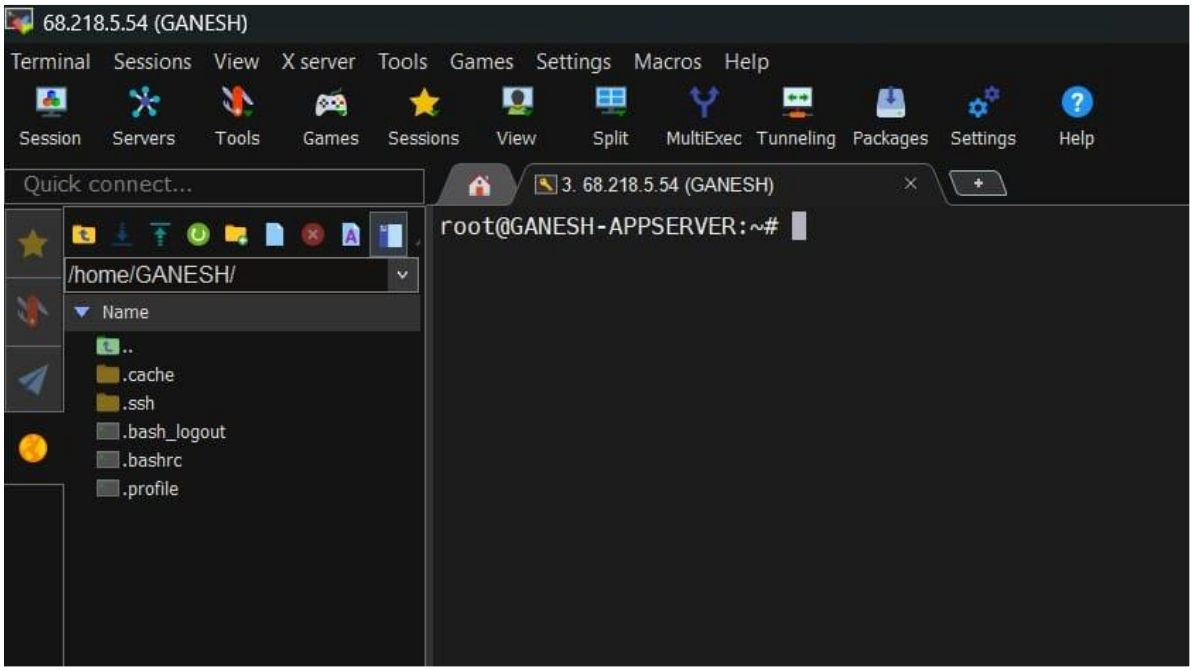
1. Update the package list: apt update
2. Install Java (OpenJDK 11): apt install openjdk-11-jdk -y
3. Create a Tomcat user: useradd -m -U -d /opt/tomcat -s /bin/false tomcat

   This command creates a dedicated Tomcat user for security purposes.

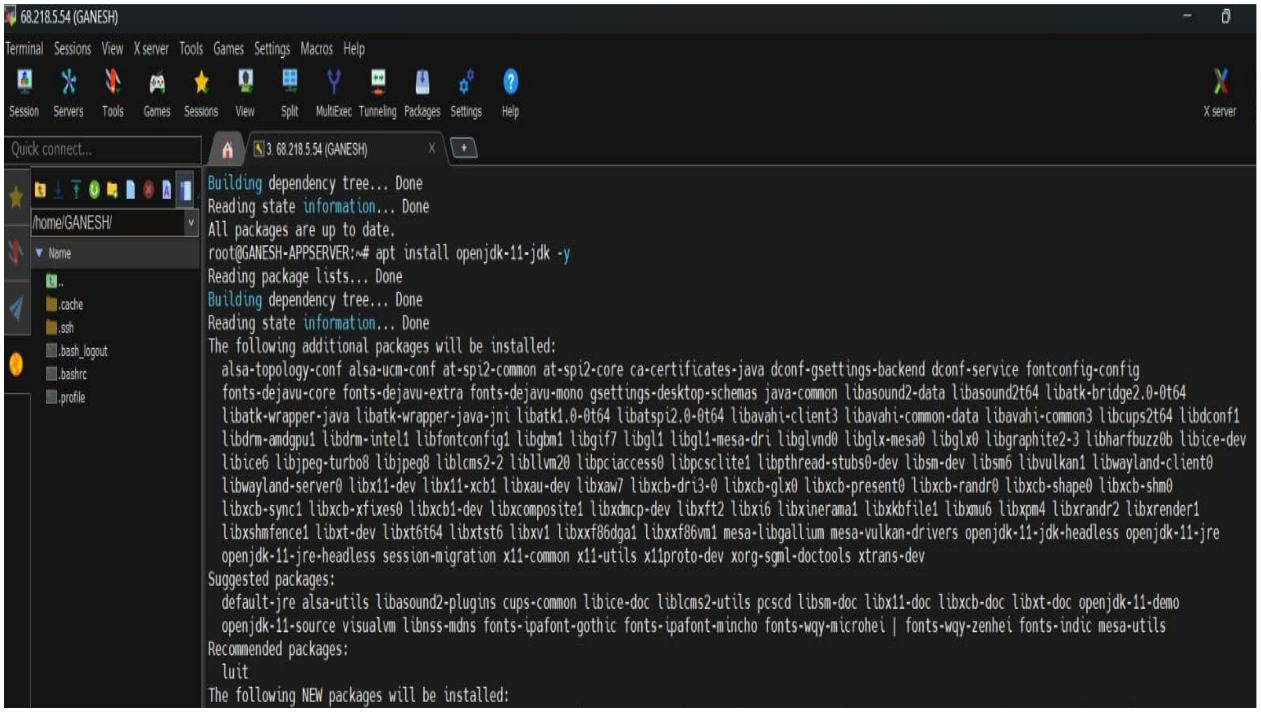
4. Download Apache Tomcat: cd /tmp
   wget https://archive.apache.org/dist/tomcat/tomcat10/v10.1.19/bin/apache-tomcat-10.1.19.tar.gz
5. Create the Tomcat installation directory: mkdir -p /opt/tomcat
6. Extract Tomcat files: tar -xzf /tmp/apache-tomcat-10.1.19.tar.gz -C /opt/tomcat --strip-components=1

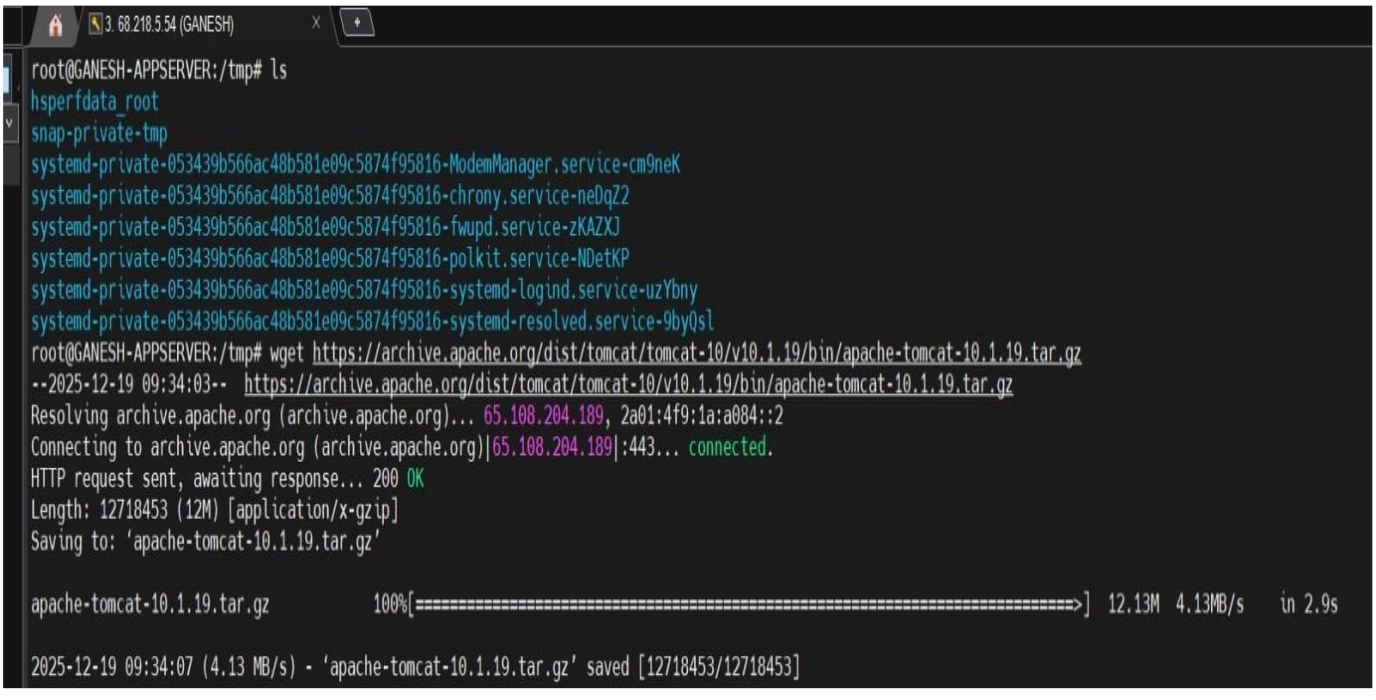
This extracts the downloaded Tomcat files into the newly created /opt/tomcat directory.



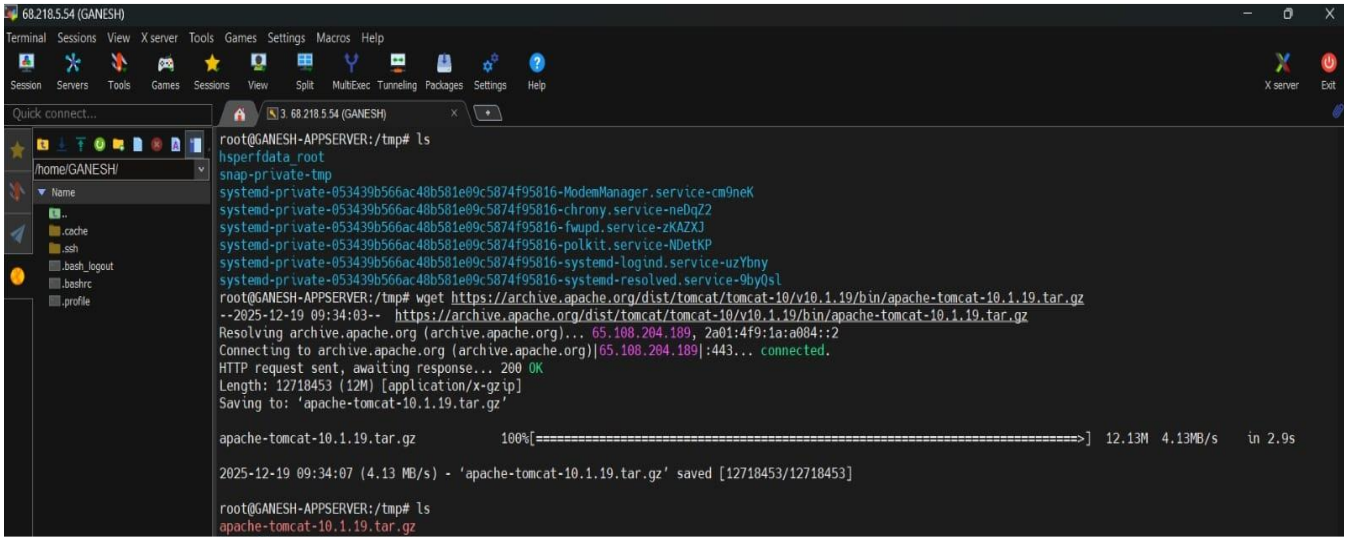


Now we are in app server, and here we need to install tomcat.





7. ls /opt/tomcat - must see (bin conf lib logs webapps work temp)




```
root@GANESH-APPSERVER:~# cd /tmp
root@GANESH-APPSERVER:/tmp# ls
apache-tomcat-10.1.19.tar.gz
hsperfdata_root
snap-private-tmp
systemd-private-053439b566ac48b581e09c5874f95816-ModemManager.service-cm9neK
systemd-private-053439b566ac48b581e09c5874f95816-chrony.service-neDqZ2
systemd-private-053439b566ac48b581e09c5874f95816-fwupd.service-zKAZXJ
systemd-private-053439b566ac48b581e09c5874f95816-polkit.service-NDetKP
systemd-private-053439b566ac48b581e09c5874f95816-systemd-logind.service-uzYbny
systemd-private-053439b566ac48b581e09c5874f95816-systemd-resolved.service-9byQsl
root@GANESH-APPSERVER:/tmp# mkdir -p /opt/tomcat
root@GANESH-APPSERVER:/tmp# tar -xzf /tmp/apache-tomcat-10.1.19.tar.gz -C /opt/tomcat – strip-components=1
tar: –: Not found in archive
tar: strip-components=1: Not found in archive
tar: Exiting with failure status due to previous errors
root@GANESH-APPSERVER:/tmp# ^C
root@GANESH-APPSERVER:/tmp# tar -xzf /tmp/apache-tomcat-10.1.19.tar.gz -C /opt/tomcat --strip-components=1
root@GANESH-APPSERVER:/tmp# ls /opt/tomcat
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  README.md  RELEASE-NOTES  RUNNING.txt  bin  conf  lib  logs  temp  webapps  work
root@GANESH-APPSERVER:/tmp#
```

8. chown -R tomcat:tomcat /opt/tomcat - change ownership

9. chmod +x /opt/tomcat/bin/*.sh - change permission

10. nano /etc/systemd/system/tomcat.service - to create systemed file

[Unit]
Description=Apache Tomcat
After=network.target

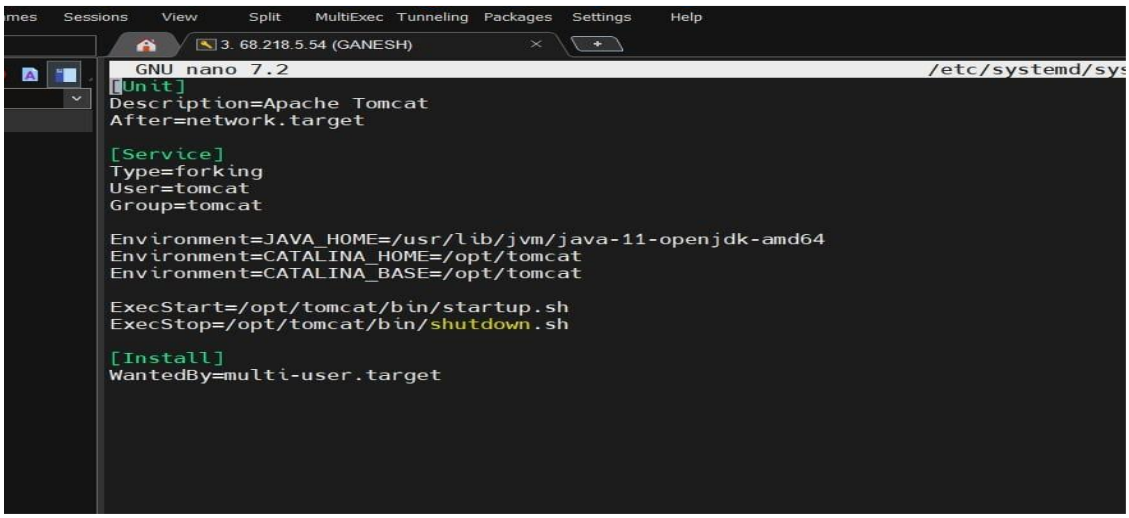
[Service]
Type=forking

User=tomcat
Group=tomcat

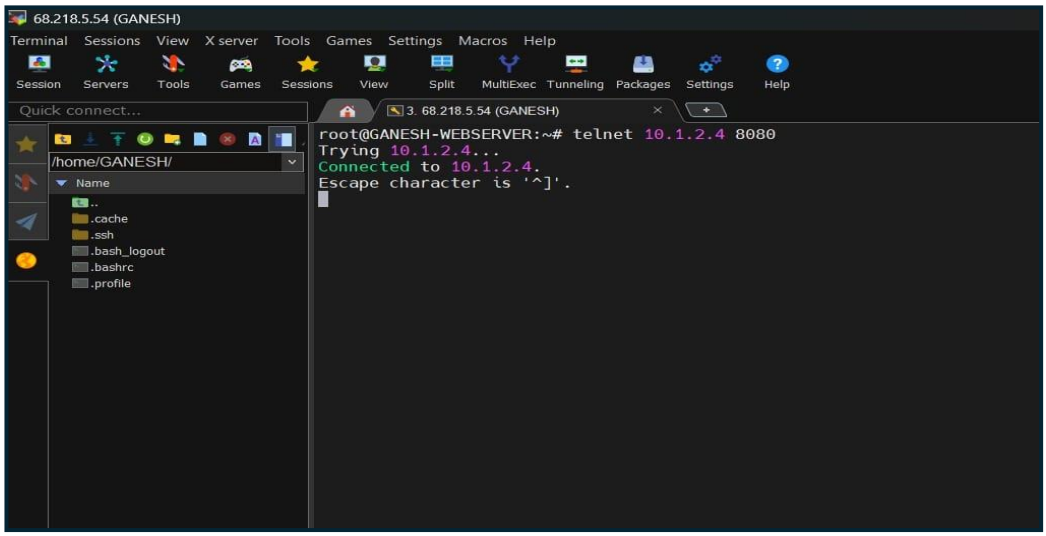
Environment=JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
Environment=CATALINA_HOME=/opt/tomcat
Environment=CATALINA_BASE=/opt/tomcat

ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

[Install]
WantedBy=multi-user.target



- systemctl daemon-reload
- systemctl start tomcat
- systemctl enable tomcat
- ss -tulnp | grep 8080 — to check 8080 port is open or not
- Now go to webserver root address and retype the telnet command: telnet 10.1.2.4 8080



Installing and Configuring MySql database in DB Server:

Now we need to install MySql database in DB Server and try to connect from appserver to dbserver using: telnet 10.2.1.4

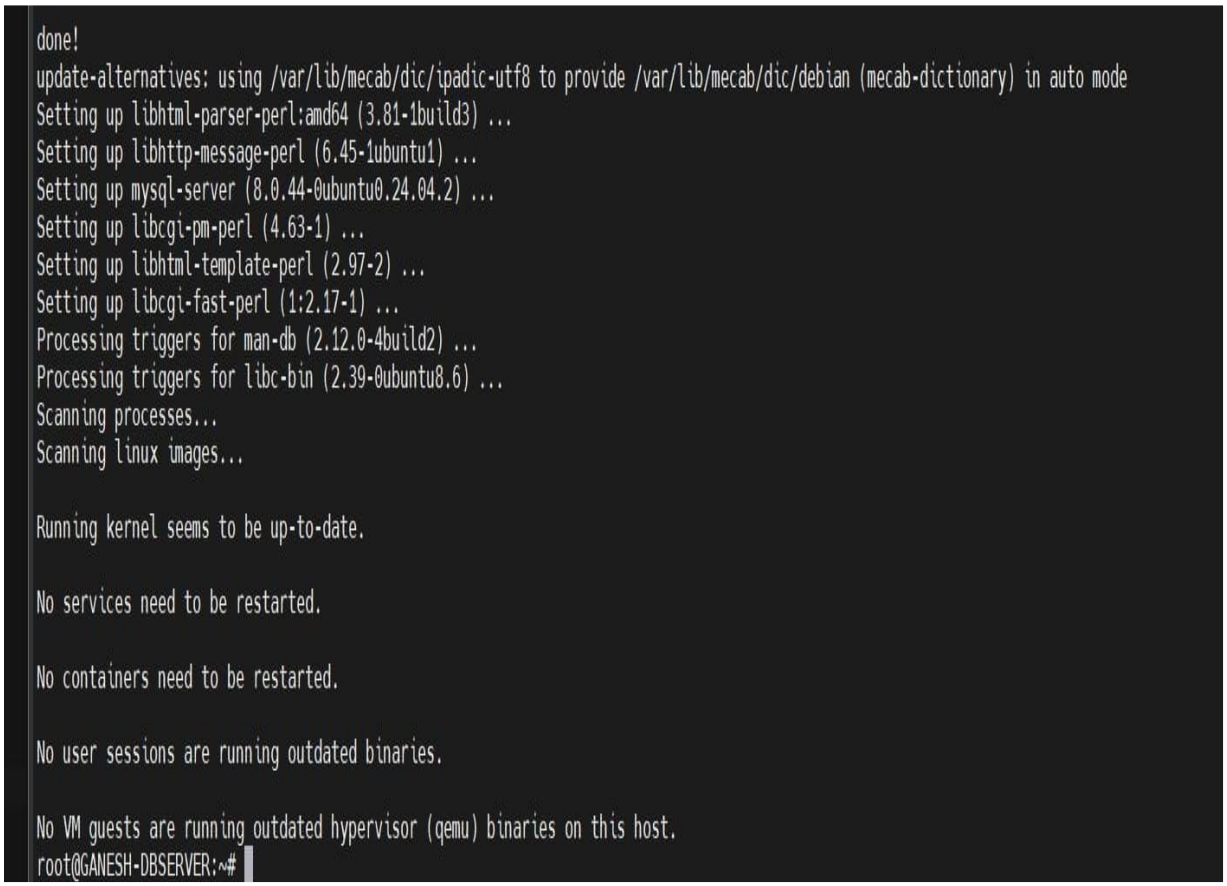
Now lets connect to app server from web server using SSH command and then from app server we connect to DB server using SSH command:

ssh ganesh@10.2.1.4

We cannot directly connect to dbserver from webserver because we denied all the ports communication of dbserver with webserver.

Execute the following commands to install and configure MySql database:

- apt update
- apt install mysql-server -y
- systemctl start mysql
- systemctl enable mysql
- systemctl status mysql



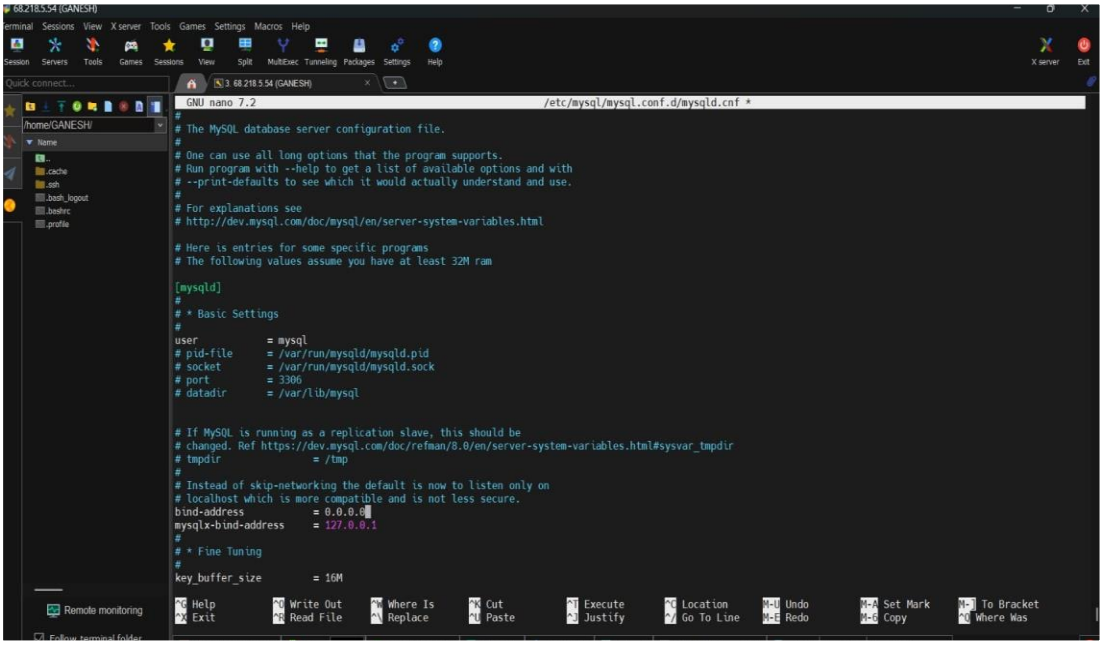
```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@GANESH-DBSERVER:~# systemctl start mysql
root@GANESH-DBSERVER:~# systemctl enable mysql
Synchronizing state of mysql.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable mysql
root@GANESH-DBSERVER:~# systemctl status mysql
● mysql.service - MySQL Community Server
     Loaded: loaded (/usr/lib/systemd/system/mysql.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-12-19 10:36:52 UTC; 1min 28s ago
   Main PID: 2997 (mysqld)
     Status: "Server is operational"
      Tasks: 37 (limit: 1037)
     Memory: 354.8M (peak: 378.9M)
        CPU: 1.096s
     CGroup: /system.slice/mysql.service
             └─2997 /usr/sbin/mysqld

Dec 19 10:36:51 GANESH-DBSERVER systemd[1]: Starting mysql.service - MySQL Community Server...
Dec 19 10:36:52 GANESH-DBSERVER systemd[1]: Started mysql.service - MySQL Community Server.
root@GANESH-DBSERVER:~#
```

- mysql_secure_installation - and press yes for all
- ss -tulnp | grep 3306:

  It is used to check whether MySQL (port 3306) is running and listening on the server.

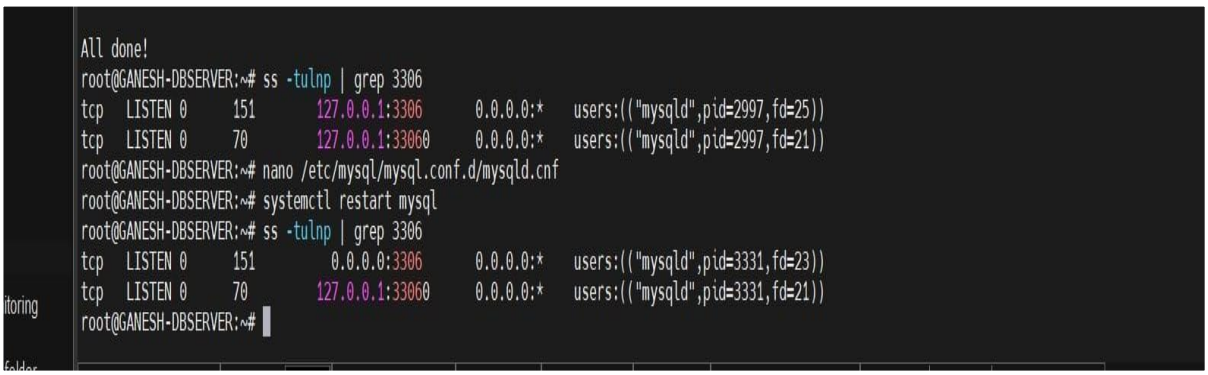
Now try to connect db server from app server, but the connection will be refused.Because in above grep command the ip is 127.0.0.1:3306, so it will connect to only that ip, so we need to chane it to 0.0.0.0:3306 to connect using any ip address. Go to db server again and execute.

- nano /etc/mysql/mysql.conf.d/mysqld.cnfex:
- ecute this cmd you will get THIS image



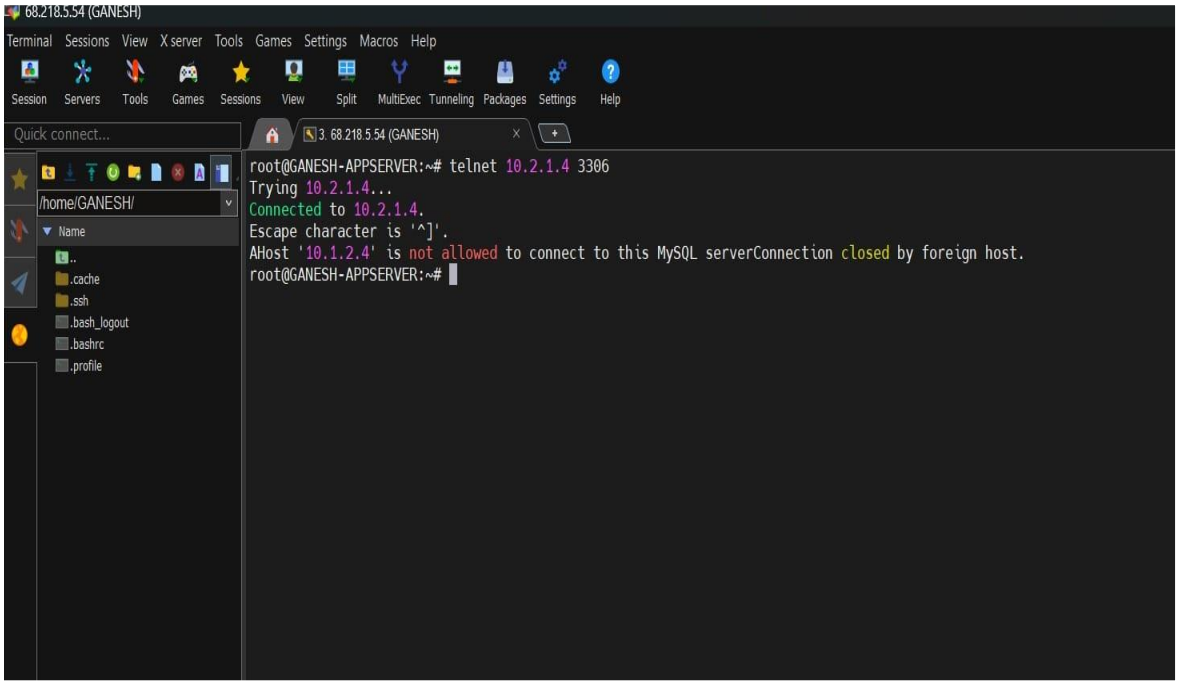
Here change the bind-address from 127.0.0.1 to 0.0.0.0

- again execute: ss -tulnp | grep 3306



127.0.0.1:3306 is changed to 0.0.0.0:3306(Now, we connect from any ip).

Now try to connect to database server from appserver,



So, the connection is established between appserver and dbserver.

CONCLUSION :

This project shows how a three-tier architecture is implemented on Microsoft Azure. The application is divided into three layers: Web Server, Application Server, and Database Server. This separation helps in better performance, easy management, and improved security.

Azure Virtual Networks (VNets), subnets, and Network Security Groups (NSGs) are used to allow secure communication between the servers and to block unauthorized access. The Web Server is used to handle user requests, the Application Server runs the business logic using Apache Tomcat, and the Database Server stores data securely using MySQL in a private subnet.

Overall, this project represents a real-world cloud architecture and provides a basic understanding of how multi-tier applications are deployed on Azure.