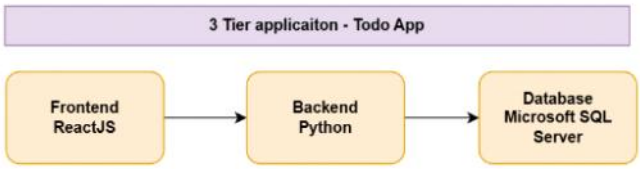
# Todo 3 Tier Monolithic-Application Deployment on Azure Cloud

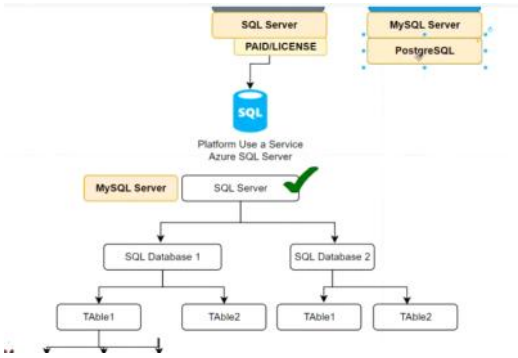
01 June 2025 10:00

**Create The Resource Group In which we are going to Deploy our 3 Tier Application.**

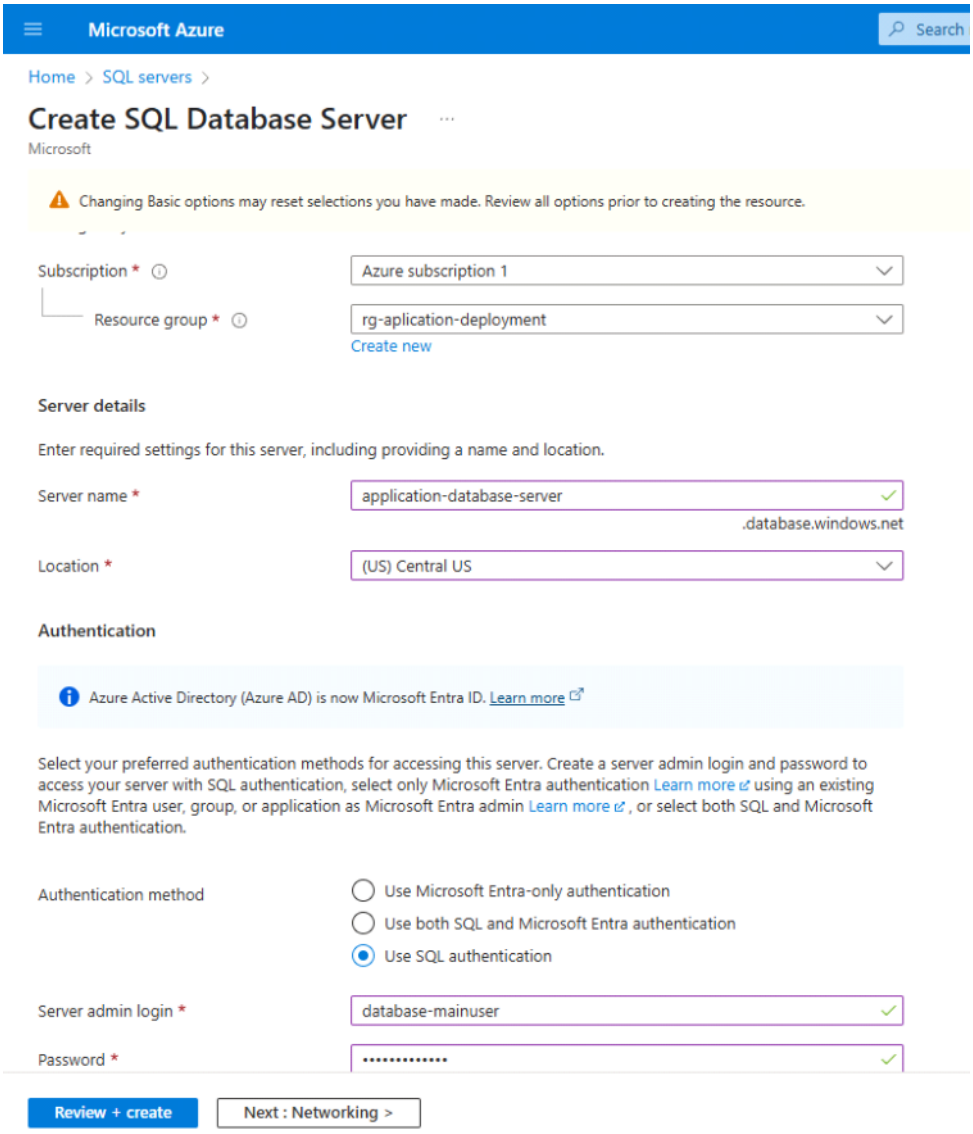


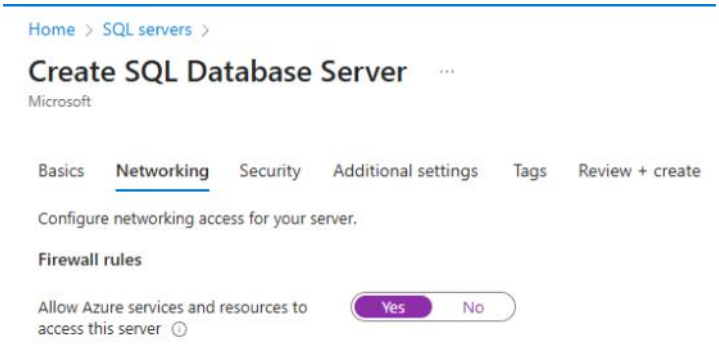

## 1. Database Set Up.

We will set up SQL Database using **Azure** in the Resource Group that we can as Azure SQL database to insert the coming Load from the Application UI. (To avoid whole setup that we are required to setup SQL Database on the local machine and Management, Backup, License.)

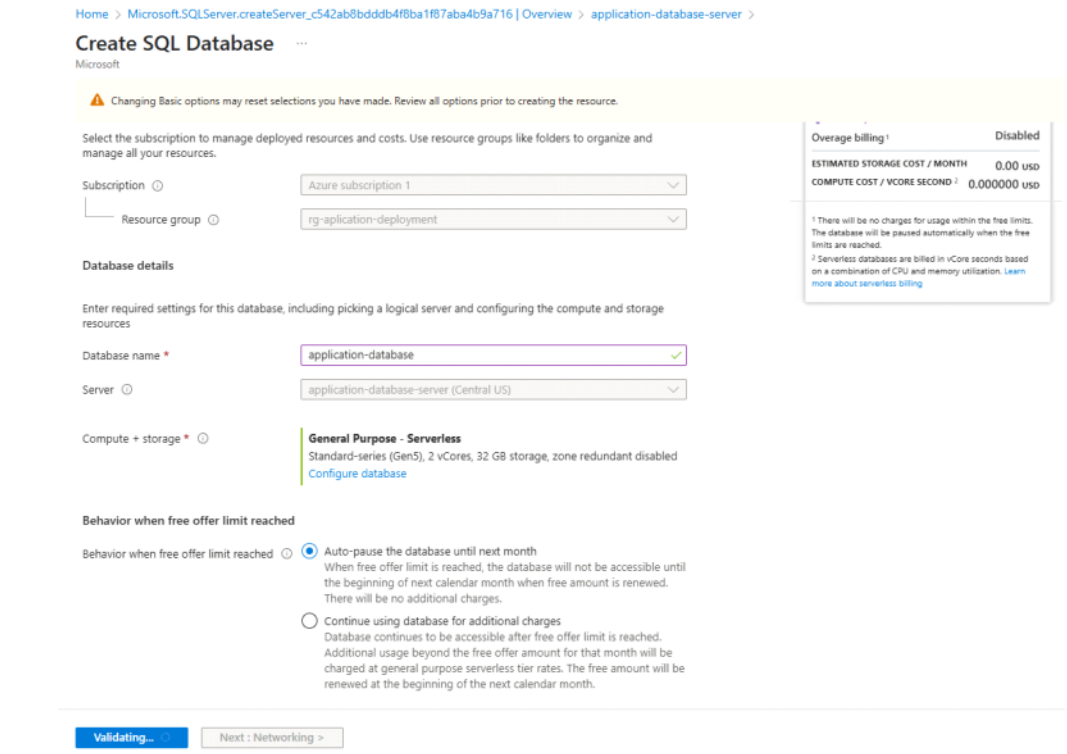



**Create SQL Database Server.**

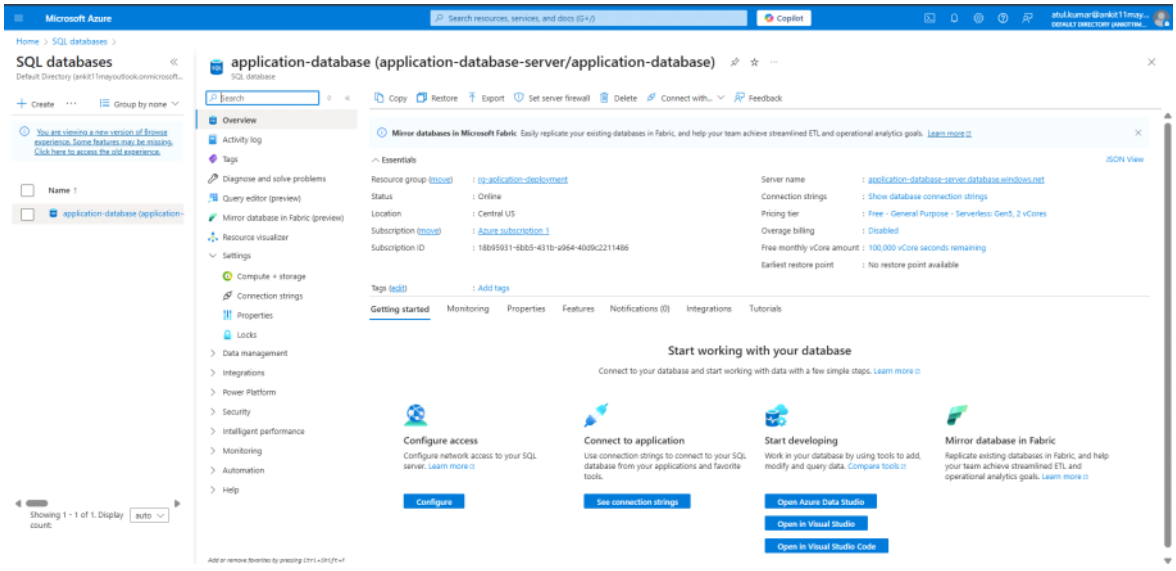


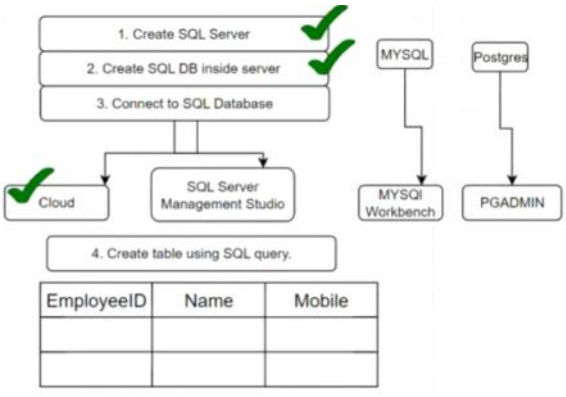

**Create SQL Database.**



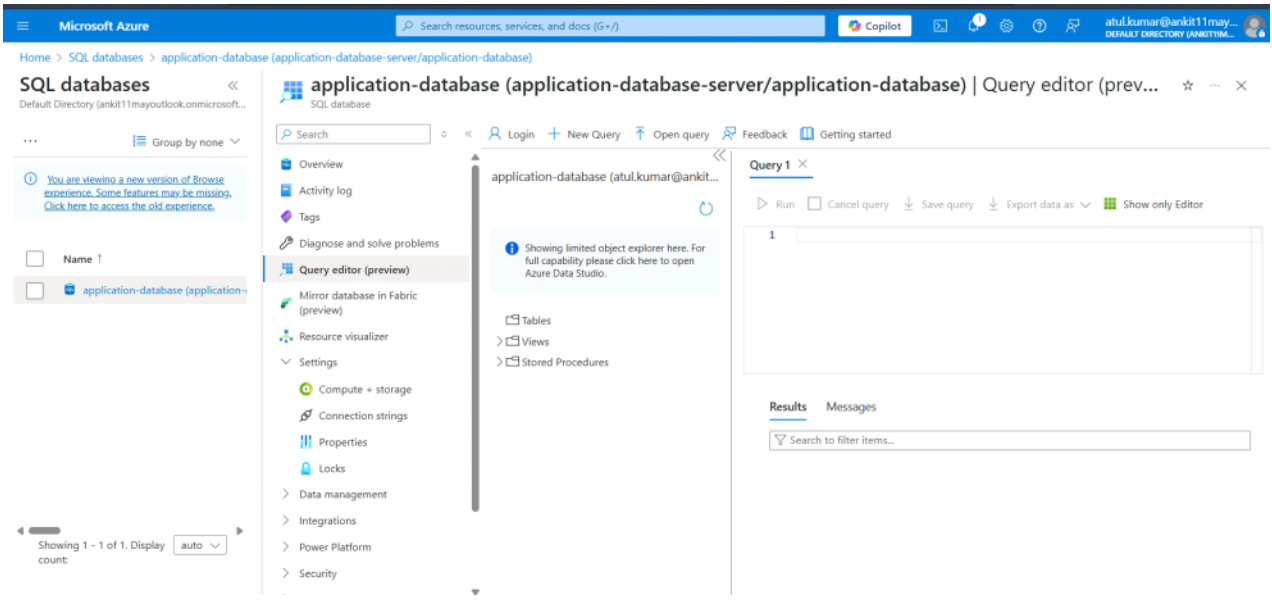

**SQL Database is created.**




You Can Access the Database from Cloud and MySQL Workbench, PGADMIN(Postgres Admin), SSMS(SQL Server Management Studio).

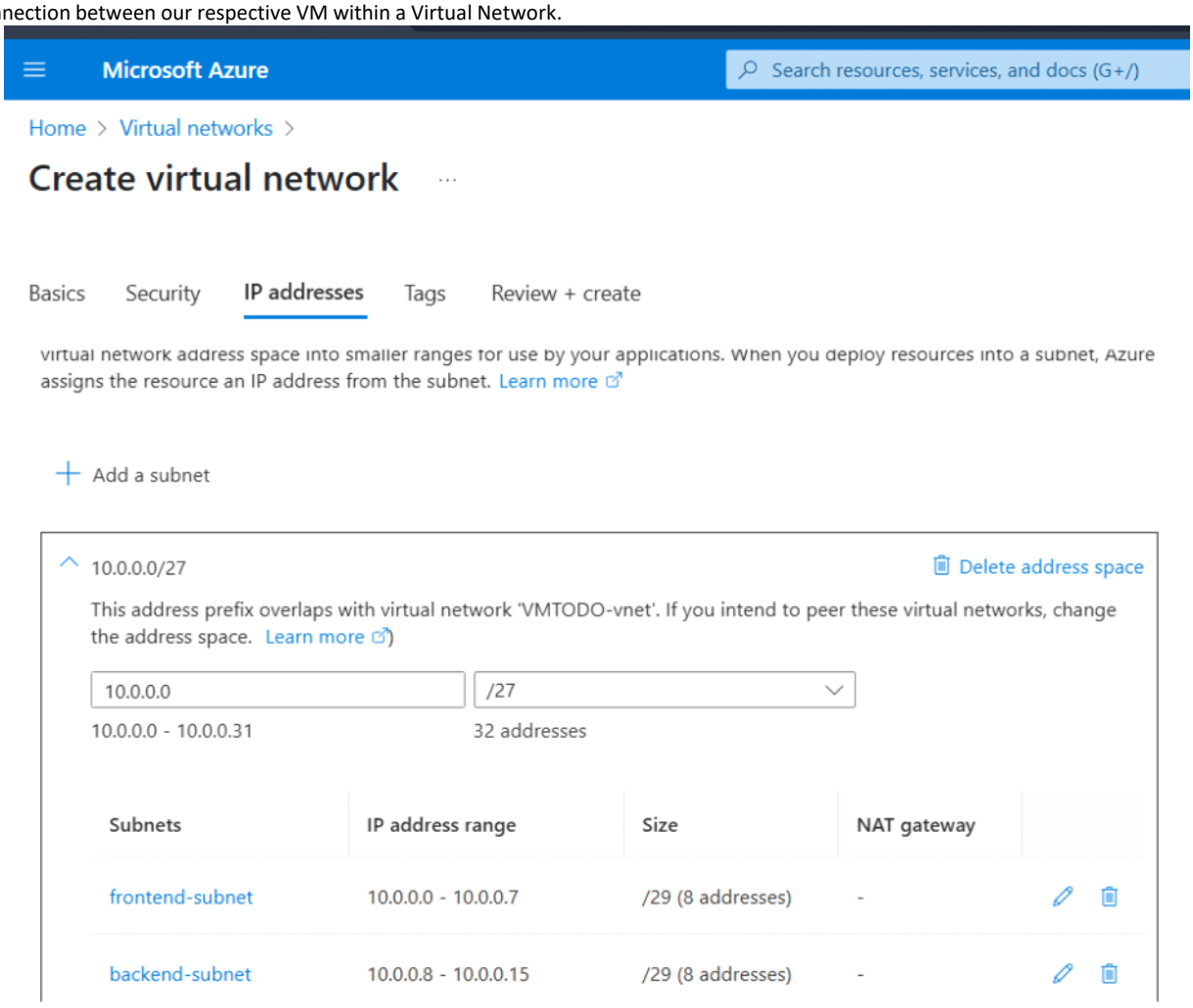



When you are accessing the Azure SQL Database from Cloud.
Then Allow IP to the AllowList to access the database and for querying it.
SQL Database is now Online and ready for use.




## Now To Set Up Backend Application and Frontend Application.

Create Virtual Network and Define the Subnet For both Frontend and Backend within the same virtual Network to establish a smooth connection between our respective VM within a Virtual Network.




## 2. Backend Set Up

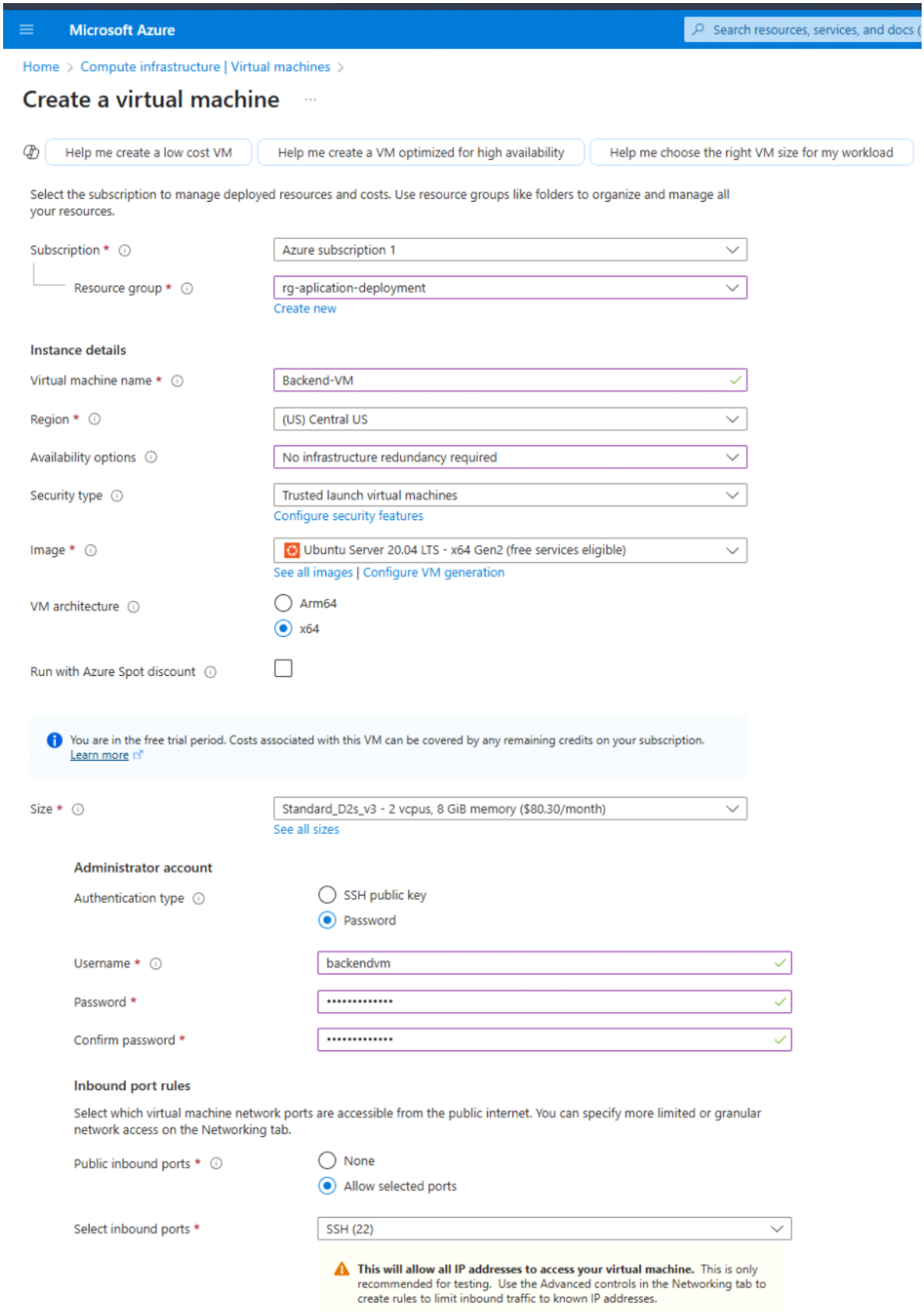
Follow the Readme.md to Check for the specific image need to be pull to set up infrastructure for Backend.

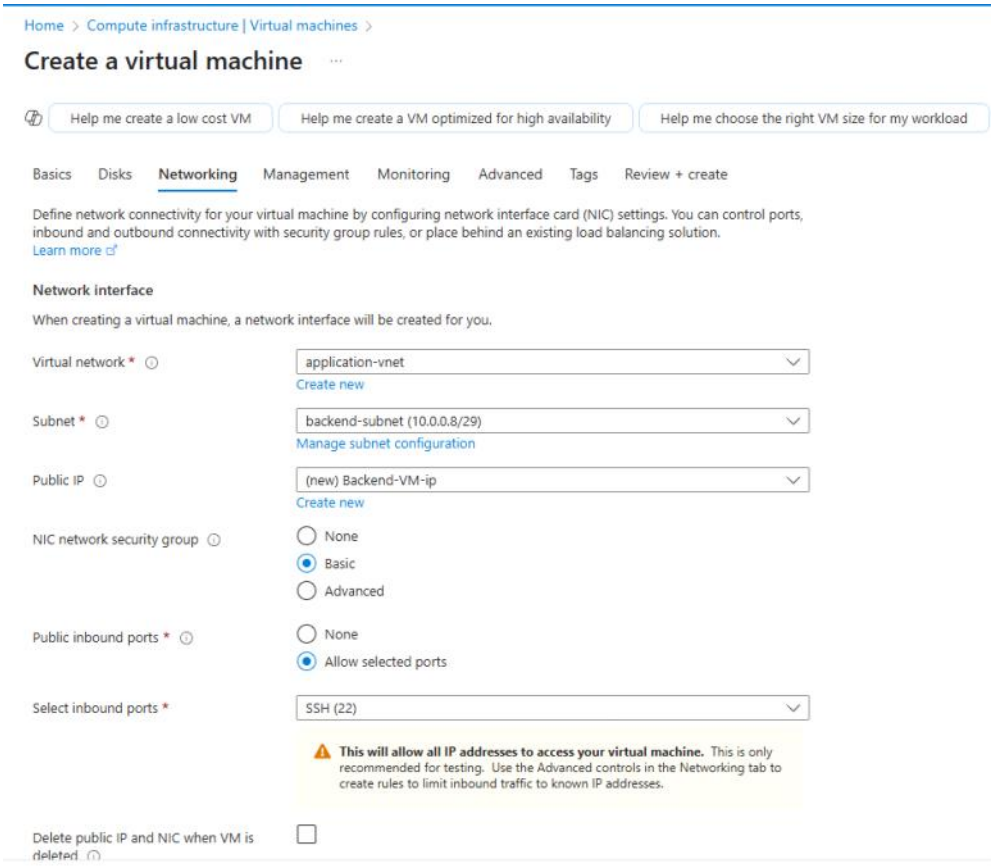
## Prerequisites

Before getting started, make sure you have the following prerequisites installed on your system:

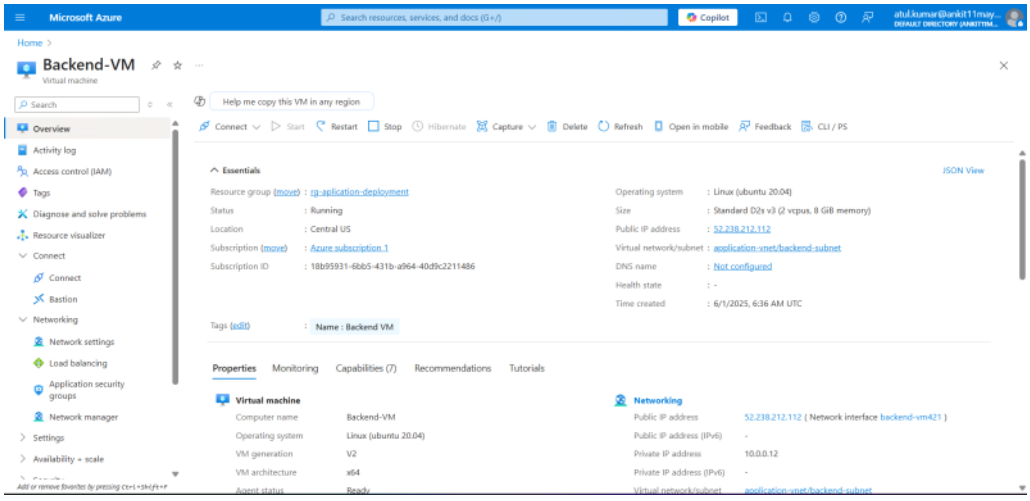
- source_image_reference = { publisher = "Canonical" offer = "0001-com-ubuntu-server-focal" sku = "20_04-lts" version = "latest" }
- Python
- pip

**Create Backend VM**

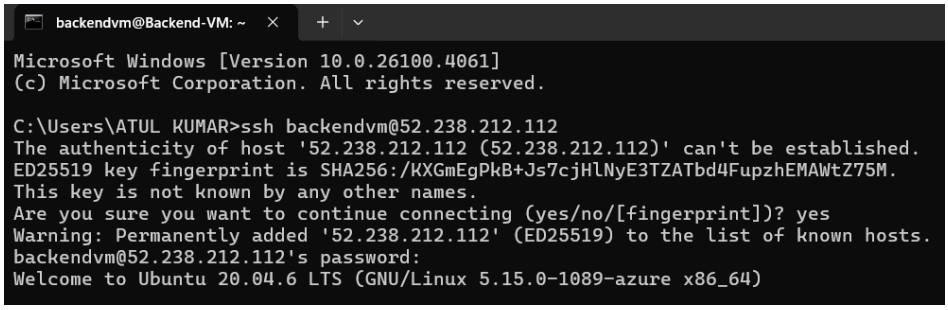





**Backend VM Created:**




**Connect to Virtual Machine using SSH**




Follow the requirement for backend application
Install PIP, Python

sudo apt install pip (It comes with Python3)

```
backendvm@Backend-VM:~$ python3 --version
Python 3.8.10
backendvm@Backend-VM:~$ pip --version
pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)
```

Clone the repository.

**Step 1: Clone the Repository**

Clone the application's source code from your version control system or download it as a zip archive and extract it to your local machine.

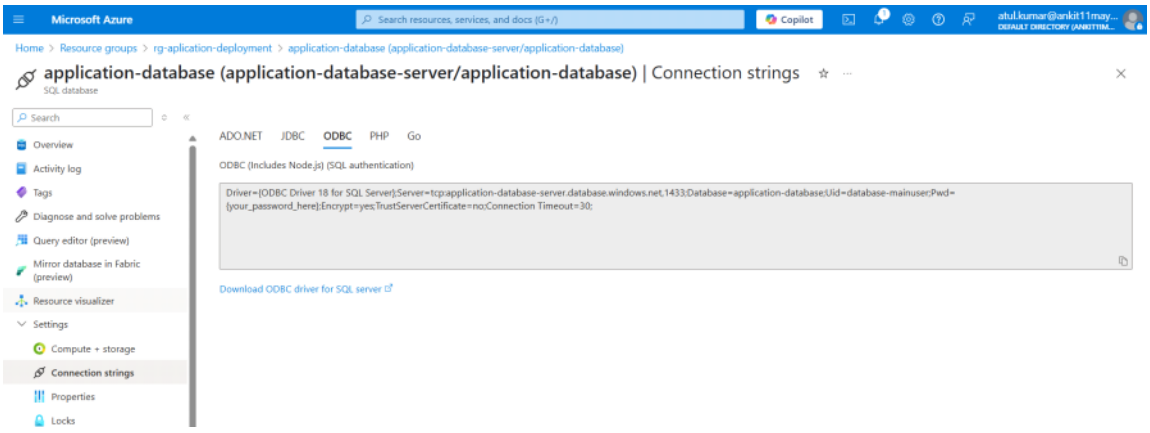
```
git clone <repository_url>
cd <repository_directory>
```

Make the required changes in the specified Files.

**Step 2: Update Connection String**

Edit the `app.py` file to update the `connection_string` variable with the appropriate connection details for your SQL Server database. Update ODBC Version to 17

Retrieve the connection string URL from the created application-database and make changes in the Backend code according to the Mentioned by developer in the Readme.md file(app.py)




Driver={ODBC Driver 17 for SQL Server};Server=tcp:application-database-server.database.windows.net,1433;Database=application-database;Uid=database-mainuser;Pwd=Adminuser@123;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;

```
backendvm@Backend-VM:~/PyTodoBackendMonolith$ nano app.py
backendvm@Backend-VM:~/PyTodoBackendMonolith$ cat app.py
import pyodbc
import uvicorn
import os
from fastapi import FastAPI
from pydantic import BaseModel
from fastapi.middleware.cors import CORSMiddleware
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

connection_string = "Driver={ODBC Driver 17 for SQL Server};Server=tcp:application-database-server.database.windows.net,1433;Database=application-database;Uid=database-mainuser;Pwd=Adminuser@123;Encrypt=yes;TrustServerCertificate=no;Connection Timeout=30;"

app = FastAPI()

# Configure CORSMiddleware to allow all origins (disable CORS for development)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # This allows all origins (use '*' for development only)
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Define the Task model
class Task(BaseModel):
    title: str
    description: str

# Create a table for tasks (You can run this once outside of the app)
@app.get("/api")
def create_tasks_table():
    try:
        conn = pyodbc.connect(connection_string)
        cursor = conn.cursor()
        cursor.execute("""
            CREATE TABLE Tasks (
                ID int NOT NULL PRIMARY KEY IDENTITY,
```

**Note: We don't hard code the Connection string in the source code. We should keep it in Azure Key Vault and access from there.**

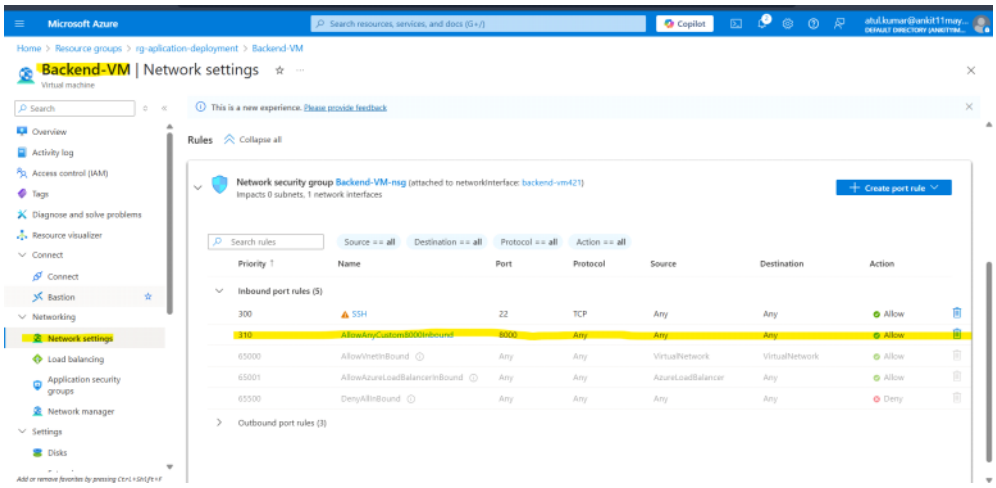
Run The Application by running the code lines one by one. It will be running on Port specified.

**Step 3: Run Below Commands to make the application running**

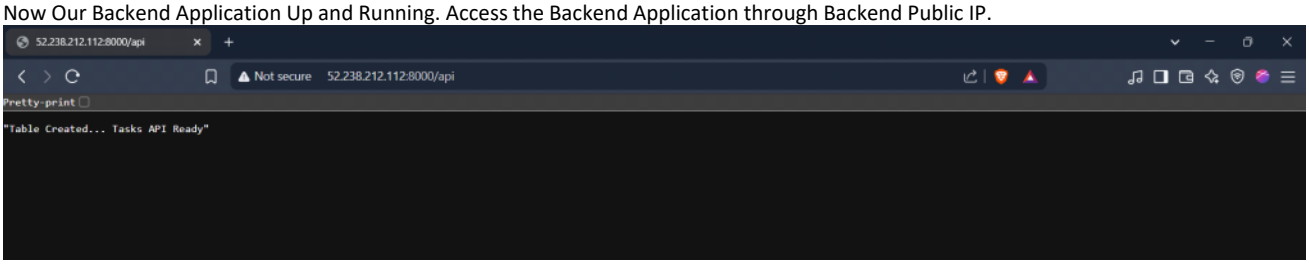
To Run the Application, open a terminal, navigate to the project directory, and run the following command:

```
sudo su
apt-get update && apt-get install -y unixodbc unixodbc-dev
curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
curl https://packages.microsoft.com/config/debian/10/prod.list > /etc/apt/sources.list.d/mssql-release.list
apt-get update
ACCEPT_EULA=Y apt-get install -y msodbcsql17
pip install -r requirements.txt
uvicorn app:app --host 0.0.0.0 --port 8000
```

**Set Inbound Rule for the Port on which our VM is running.**




Now Our Backend Application Up and Running. Access the Backend Application through Backend Public IP.

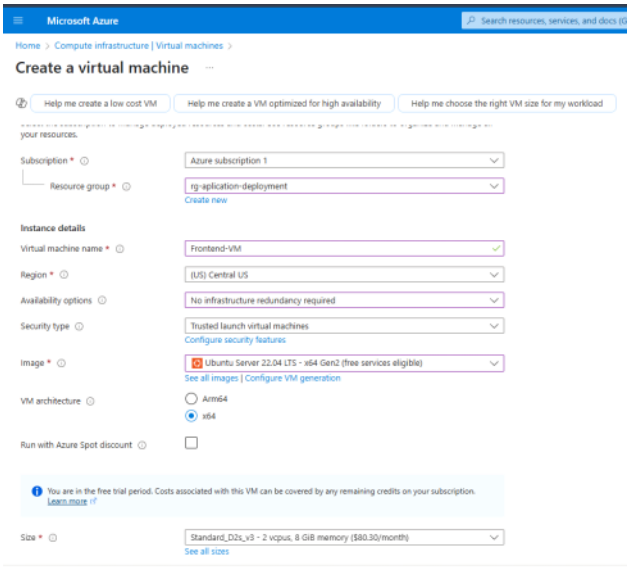



To Troubleshoot Backend Based Application you can use Backend VM connection console and see where it is failing and which all request requests is success and failure.

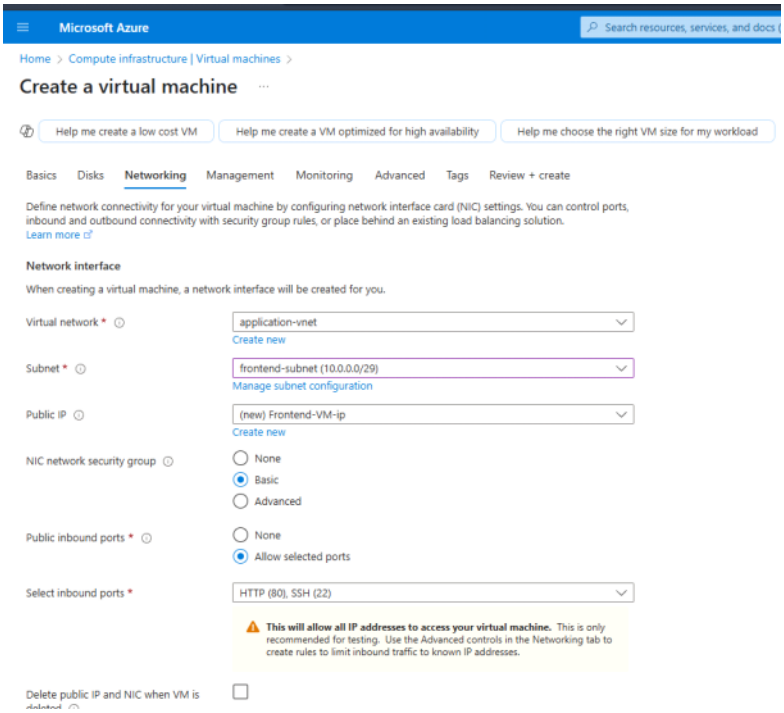
## 3. Frontend Set Up

Start Following the Frontend Application Readme.md File in the repository and create VM if there is any specific image Infrastructure requirements.

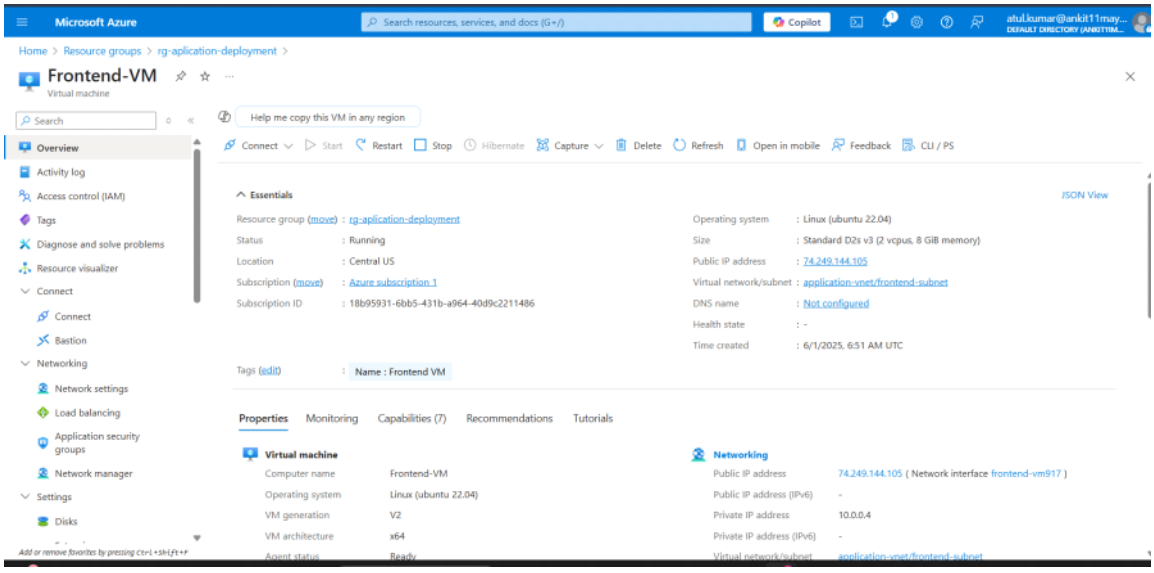
**Create Frontend VM**



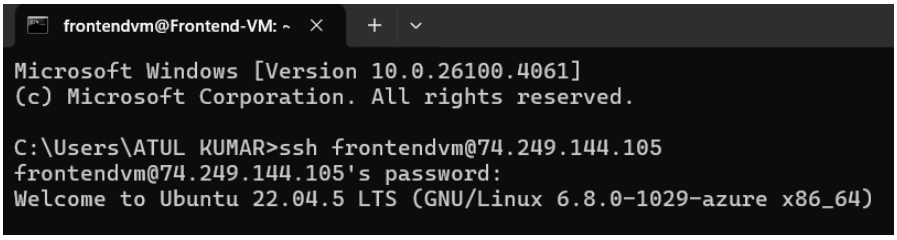

Enabled Port No. 80 to access Nginx Server and by keep the by default ssh Port No. 22 enabled.



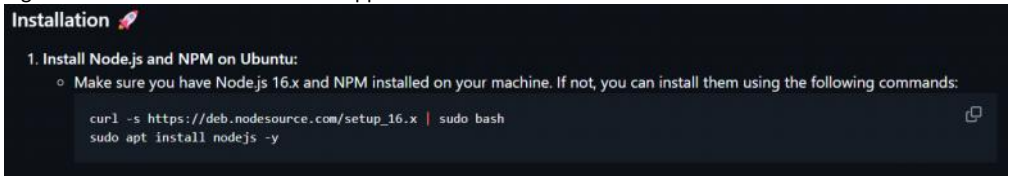

**Frontend VM Created.**




**Now connect to the Frontend VM using SSH.**




Start Following the Readme.md File in Frontend Application source code.




Clone the Frontend Application Repository and change to the application repository.
Make the required changes in the specified Files.

## Configuration ⚙

2. **Update Backend URL:**
   - Open the `src/TodoApp.js` file.
   - Locate the variable storing the backend URL and update it with the appropriate value. (* See Below for PrivateIp Configuration)

```
frontendvm@Frontend-VM:~$ cd ReactTodoUIMonolith/
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ ls
README.md  package-lock.json  package.json  public  src
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ cd src
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/src$ ls
App.css  App.js  App.test.js  TodoApp.js  index.css  index.js  logo.svg  reportWebVitals.js  setupTests.js  theme.js
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/src$ nano TodoApp.js
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/src$ cat TodoApp.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, TextField, Container, Typography, Grid, Card, CardContent, IconButton } from '@mui/material';
import { Delete } from '@mui/icons-material';
import { Box } from '@mui/material';

const API_BASE_URL = 'http://52.238.212.112:8000/api';

const backgroundImage = process.env.PUBLIC_URL + '/background.jpg';

function TodoApp() {
    const [tasks, setTasks] = useState([]);
    const [newTask, setNewTask] = useState({ title: '', description: '' });

    const fetchTasks = async () => {
        try {
            const response = await axios.get(`${API_BASE_URL}/tasks`);
            setTasks(response.data);
        } catch (error) {
            console.error('Error fetching tasks', error);
        }
    };

    const createTask = async () => {
        try {
            await axios.post(`${API_BASE_URL}/tasks`, newTask);
            fetchTasks();
            setNewTask({ title: '', description: '' });
        } catch (error) {
            console.error('Error creating task', error);
        }
    };
```

Before installing the required node module mentioned in package.json file in React based Application.

```
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ ls
README.md  package-lock.json  package.json  public  src
```

3. **Install Dependencies:**
   - Run the following command to install project dependencies:

```
npm install
```

After npm install all the required node modules and there in the node module folder.

```
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ ls
README.md  node_modules  package-lock.json  package.json  public  src
```

After npm run build our build artifacts is ready and ready to deploy/copy and paste in the /var/www/html on nginx server.

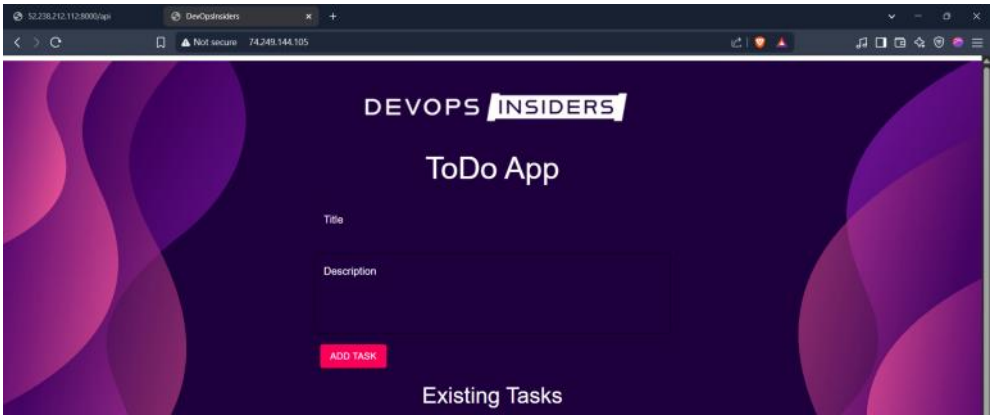
```
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ ls
README.md  build  node_modules  package-lock.json  package.json  public  src
frontendvm@Frontend-VM:~/ReactTodoUIMonolith$ cd build
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/build$ ls
asset-manifest.json  background.jpg  devopsinsiderslogo.png  favicon.ico  index.html  logo192.png  logo512.png  manifest.json  robots.txt  static
```

**Now Install the Nginx server** using -> sudo apt install nginx

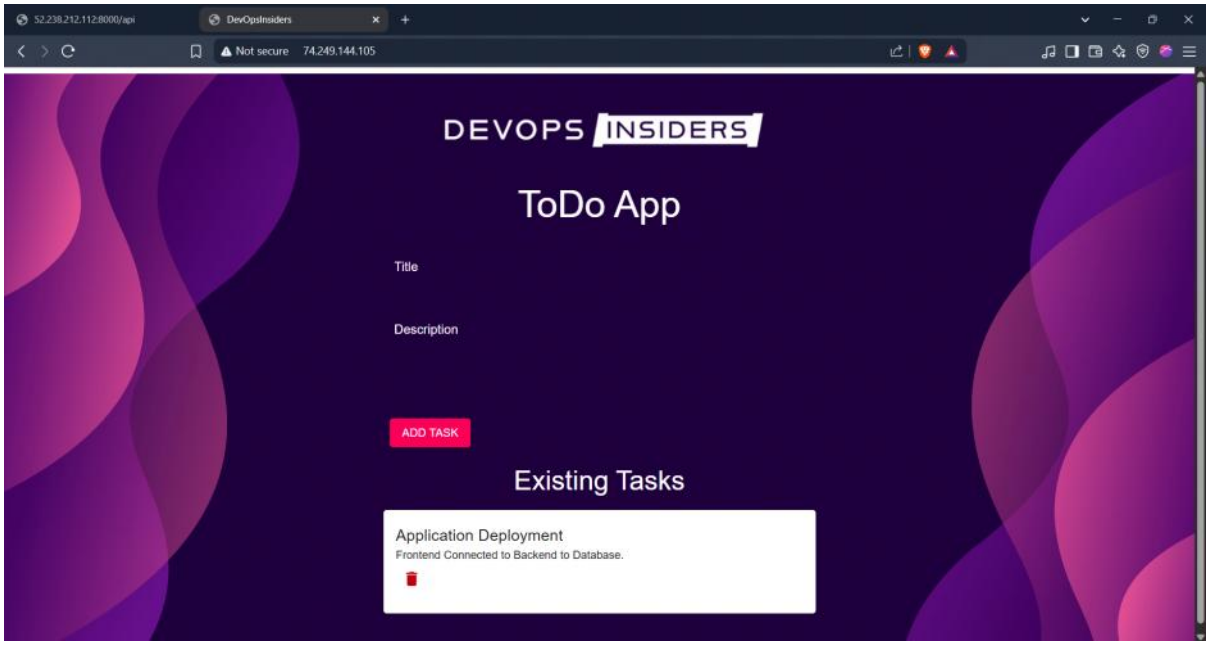
Copy the build folder files to nginx /var/www/html folder.

```
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/build$ sudo cp -r * /var/www/html
frontendvm@Frontend-VM:~/ReactTodoUIMonolith/build$ cd /var/www/html
frontendvm@Frontend-VM:/var/www/html$ ls
asset-manifest.json  devopsinsiderslogo.png  index.html              logo192.png  manifest.json  static
background.jpg       favicon.ico             index.nginx-debian.html  logo512.png  robots.txt
```

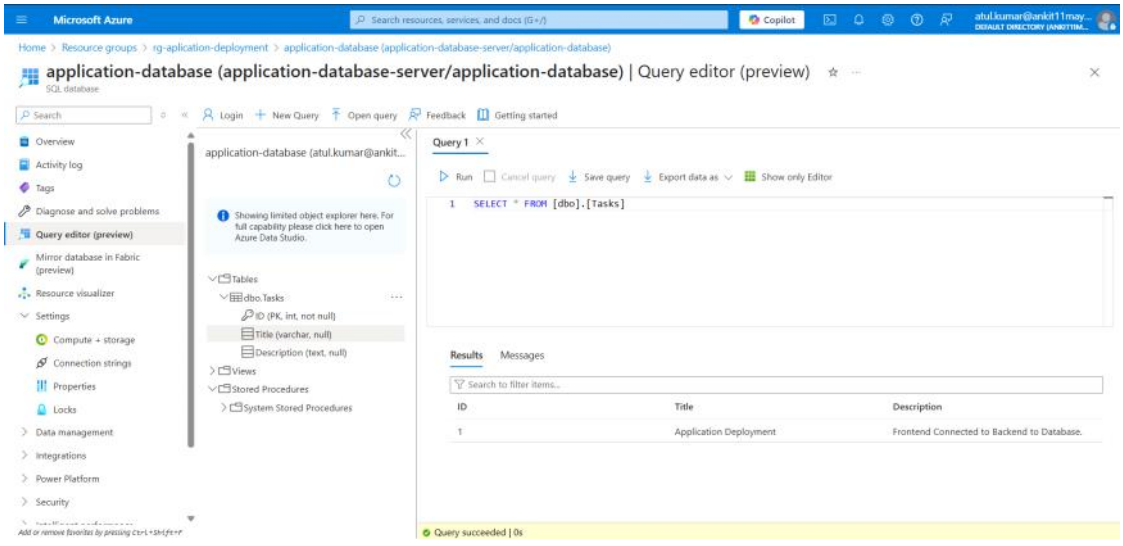
Now access the Frontend Application using the Public IP.



Now when you put the data in the application from Frontend it will be saved to Database.
To Troubleshoot Frontend Based Application you can use Browser Networking section and see where it is failing to send the request and getting the response.




You Can check the Data is now inserted into the Azure SQL Database and can see using the query by accessing the database in cloud using query editor.




Note:

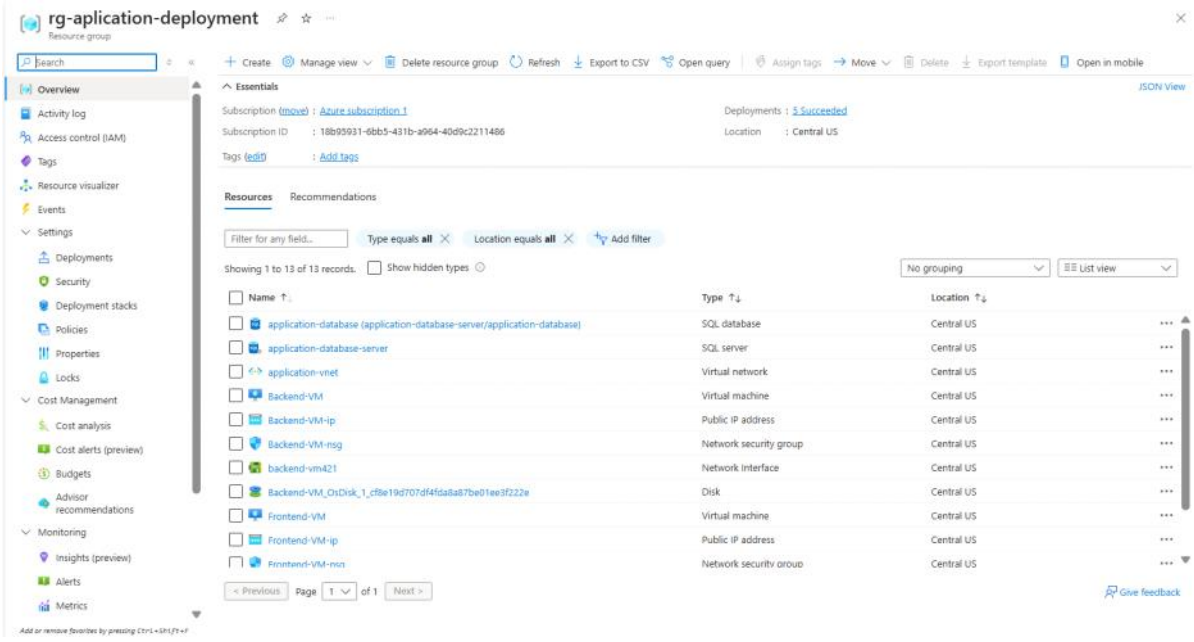
- Keep the status of All-Frontend VM, Backend VM and Azure SQL Database showing running/online to avoid any disconnection.
- We ca setup the whole Frontend Application build folder in the local and can use SCP to securely copy the code from the local to the virtual machine home folder

  scp -r myfolder username@vm_ip:/home/application

  and then can copy the all the files recusevly from the folder using the cp command and past into /var/www/html to access the application.

  sudo cp -r * /var/www/html

- Whole Set up in Resource Group Will Look like this.




- We need to run specific build commands depending on the type of application and the technology it's built with.

| Source Code | Dependencies | Build Tool | Artifacts Type | Server Type (deployment) |
|---|---|---|---|---|
| Angular/React JS | npm install/ package.json | tool = npm npm run build | .html, .css, .js | nginx |
| Java | pom.xml | tool = maven mvn clean | .jar, .war | tomcat |
| dotnet | csproj | dotnet build and publish, msbuild | .dll | iis, dotnet runtime |
| Python | requirements.txt | pip install | No Artifacts | Uvicorn |