🚀Complete Azure Real-Time Project: Flask App + PostgreSQL with Load Balancer

🏢Architecture Overview

```
Internet
  |
  v
Azure Load Balancer (Public IP)
  |
  +--- App-VM1 (Web Tier + Flask)
  +--- App-VM2 (Web Tier + Flask)
         |
         v
    DB-VM (Private VM running PostgreSQL)
```

🏫Step-by-Step Deployment

🪚Step 1: Create Resource Group & VNet

```
az group create --name RG-WebApplication --location eastus

az network vnet create \
  --resource-group RG-WebApplication \
  --name WebVNet \
  --address-prefix 10.0.0.0/16 \
  --subnet-name WebSubnet \
  --subnet-prefix 10.0.1.0/24

az network vnet subnet create \
  --resource-group RG-WebApplication \
  --vnet-name WebVNet \
  --name AppSubnet \
  --address-prefix 10.0.2.0/24
```

🚜Step 2: Create NSGs and Associate to Subnets

WebNSG

```
az network nsg create --resource-group RG-WebApplication --name WebNSG

az network nsg rule create --resource-group RG-WebApplication --nsg-name WebNSG
\
  --name AllowHTTP --protocol Tcp --direction Inbound --priority 100 \
  --source-address-prefixes Internet --destination-port-ranges 80 --access Allow

az network nsg rule create --resource-group RG-WebApplication --nsg-name WebNSG
\
  --name AllowSSH --protocol Tcp --direction Inbound --priority 110 \
  --source-address-prefixes Internet --destination-port-ranges 22 --access Allow
```

AppNSG

```
az network nsg create --resource-group RG-WebApplication --name AppNSG

az network nsg rule create --resource-group RG-WebApplication --nsg-name AppNSG
\
  --name AllowPostgresFromWeb --protocol Tcp --direction Inbound --priority 100
\
  --source-address-prefixes 10.0.1.0/24 --destination-port-ranges 5432 --access
Allow

az network nsg rule create --resource-group RG-WebApplication --nsg-name AppNSG
\
  --name AllowSSHFromWeb --protocol Tcp --direction Inbound --priority 110 \
  --source-address-prefixes 10.0.1.0/24 --destination-port-ranges 22 --access
Allow
```

Associate NSGs to Subnets

```
az network vnet subnet update \
  --resource-group RG-WebApplication \
  --vnet-name WebVNet \
  --name WebSubnet \
  --network-security-group WebNSG

az network vnet subnet update \
  --resource-group RG-WebApplication \
  --vnet-name WebVNet \
```

```
  --name AppSubnet \
  --network-security-group AppNSG
```

🚀Step 3: Create App VMs (Web Tier)

```
az vm create \
  --resource-group RG-WebApplication \
  --name App-VM1 \
  --image UbuntuLTS \
  --size Standard_B1s \
  --vnet-name WebVNet \
  --subnet WebSubnet \
  --admin-username dbuser \
  --generate-ssh-keys \
  --nsg WebNSG \
  --public-ip-sku Basic \
  --priority Spot --eviction-policy Deallocate

az vm create \
  --resource-group RG-WebApplication \
  --name App-VM2 \
  --image UbuntuLTS \
  --size Standard_B1s \
  --vnet-name WebVNet \
  --subnet WebSubnet \
  --admin-username dbuser \
  --generate-ssh-keys \
  --nsg WebNSG \
  --public-ip-sku Basic \
  --priority Spot --eviction-policy Deallocate
```

Step 4: Create DB-VM (Private PostgreSQL VM)

```
az vm create \
  --resource-group RG-WebApplication \
  --name DB-VM \
  --image UbuntuLTS \
  --size Standard_B1s \
  --vnet-name WebVNet \
  --subnet AppSubnet \
  --admin-username dbuser \
  --generate-ssh-keys \
```

```
  --public-ip-address "" \
  --priority Spot --eviction-policy Deallocate
```

Then, use Azure Portal > DB-VM > Reset password (SSH Public Key) to paste your public key for SSH access from App-VM1.

Step 5: Install PostgreSQL on DB-VM

```
sudo apt update
sudo apt install postgresql -y

# Setup PostgreSQL
sudo -i -u postgres
psql
CREATE DATABASE azuredb;
CREATE USER dbadmin WITH PASSWORD 'StrongP@ssw0rd';
GRANT ALL PRIVILEGES ON DATABASE azuredb TO dbadmin;
\q
exit

# Configure remote access
sudo nano /etc/postgresql/*/main/postgresql.conf
# Set: listen_addresses = '*'

sudo nano /etc/postgresql/*/main/pg_hba.conf
# Add: host all all 10.0.1.0/24 md5

sudo systemctl restart postgresql
```

Step 6: Install Flask App on App-VM1 & App-VM2

Run this on **both VMs**:

```
sudo apt update
sudo apt install python3-pip -y
sudo pip3 install flask psycopg2-binary

cat <<EOF > app.py
from flask import Flask
import psycopg2
import socket
app = Flask(__name__)
```

```
@app.route('/')
def home():
    try:
        conn = psycopg2.connect(
            host="10.0.2.4",
            database="azuredb",
            user="dbadmin",
            password="StrongP@sswOrd"
        )
        cur = conn.cursor()
        cur.execute("SELECT NOW();")
        result = cur.fetchone()
        cur.close()
        conn.close()
        hostname = socket.gethostname()
        return f"<h1>Connected to DB!</h1><p>DB Time: {result[0]}</p><p>Server:
{hostname}</p>"
    except Exception as e:
        return f"<h1>Connection Failed</h1><p>{e}</p>"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
EOF

sudo python3 app.py > flask.log 2>&1 &
```

🚧Step 7: Create Load Balancer and Backend Pool

```
az network public-ip create \
  --resource-group RG-WebApplication \
  --name WebApplication-LB-PublicIP \
  --sku Basic --allocation-method Static

az network lb create \
  --resource-group RG-WebApplication \
  --name WebApplication-LB \
  --public-ip-address WebApplication-LB-PublicIP \
  --frontend-ip-name LoadBalancerFrontEnd \
  --backend-pool-name WebApplicationBackendPool

az network lb probe create \
  --resource-group RG-WebApplication \
  --lb-name WebApplication-LB \
  --name httpProbe \
  --protocol tcp \
  --port 80
```

```
az network lb rule create \
  --resource-group RG-WebApplication \
  --lb-name WebApplication-LB \
  --name httpRule \
  --protocol tcp \
  --frontend-port 80 \
  --backend-port 80 \
  --frontend-ip-name LoadBalancerFrontEnd \
  --backend-pool-name WebApplicationBackendPool \
  --probe-name httpProbe
```

🚡 Step 8: Attach App VM NICs to Backend Pool

First, find NIC + IP config names:

```
az network nic list --resource-group RG-WebApplication \
  --query "[].{NIC:name, IPConfigs:ipConfigurations[].name}" -o json
```

Then attach each NIC:

```
az network nic ip-config address-pool add \
  --address-pool WebApplicationBackendPool \
  --ip-config-name ipconfig1 \
  --nic-name App-VM1VMNic \
  --resource-group RG-WebApplication \
  --lb-name WebApplication-LB

az network nic ip-config address-pool add \
  --address-pool WebApplicationBackendPool \
  --ip-config-name ipconfig1 \
  --nic-name App-VM2VMNic \
  --resource-group RG-WebApplication \
  --lb-name WebApplication-LB
```

Step 9: Access the App via Load Balancer

```
az network public-ip show \
  --resource-group RG-WebApplication \
```

```
  --name WebApplication-LB-PublicIP \
  --query ipAddress -o tsv
```

Then open your browser:

```
http://<LoadBalancer_Public_IP>
```

You should see DB time and the hostname (`App-VM1` or `App-VM2`).

🌐Success! You now have a Load Balanced Flask + PostgreSQL app on Azure using Spot VMs!

Let me know if you'd like to:

- Auto-script this with bash or Terraform
- Add autoscaling or backups
- Add HTTPS with Azure Application Gateway

Happy building!