Terraform Setup and How to use Terraform Script

This document outlines the steps to deploy a virtual network with subnets, a web server, and an Azure SQL database using Terraform. The code is modularized for reusability and simplicity.

Directory Structure

The Terraform code is organized as follows:

```
AzureTaskResources/
├── main.tf
├── outputs.tf
├── providers.tf
├── variables.tf
├── terraform.tfvars
├── modules/
│   ├── network/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── vm/
│   │   ├── main.tf
│   │   ├── variables.tf
│   └── sql/
│       ├── main.tf
│       ├── variables.tf
```

Steps to Use

Step 1: Clone the Repository

Clone the repository containing the Terraform files.

```
git clone <repository-url>
cd AzureTaskResources
```

Step 2: Initialize Terraform

Run the following command to initialize Terraform and download the required providers:

terraform init

Step 3: Update Variables (Optional)

Edit the `terraform.tfvars` file to customize the deployment parameters:

```
region              = "East US"
resource_group_name = "example-resource-group"
web_subnet_name     = "web-subnet"
db_subnet_name      = "db-subnet"
vm_name             = "web-server"
admin_username      = "adminuser"
admin_password      = "StrongPassword123!"
vm_size             = "Standard_DS1_v2"
sql_server_name     = "example-sql-server"
sql_database_name   = "example-database"
sql_sku             = "S1"
```

Step 4: Plan the Deployment

Run the following command to view the resources that will be created:

```
terraform plan
```

Step 5: Apply the Deployment

Run the following command to create the resources in Azure:

```
terraform apply
```

Confirm the deployment by typing `yes` when prompted.

Step 6: Verify the Deployment

Once the deployment is complete, Terraform will output the following:

- Public IP of the web server.
- SQL Server name.

You can verify the resources in the Azure portal.

Outputs

After running `terraform apply`, the following outputs are available:

- **Web Server Public IP**: Use this to access the web server.

- **SQL Server Name**: Use this to connect to the SQL database.

Modules Breakdown

Network Module

- **Purpose**: Creates a virtual network with two subnets (web and database) and a network security group.
- **Path**: `modules/network/`

Web VM Module

- **Purpose**: Deploys a Linux-based virtual machine in the web subnet.
- **Path**: `modules/vm/`

SQL Module

- **Purpose**: Deploys an Azure SQL server and database in the database subnet.
- **Path**: `modules/sql/`

Cleaning Up Resources

To destroy all the resources created by Terraform, run:

```
terraform destroy
```

For cost management and optimization:

1. **Azure Pricing Calculator**:
    - Go to the Azure Pricing Calculator.
    - Add the services you're using (e.g., Virtual Machines, Database services).
    - Configure them with the correct specs (e.g., VM size, storage type, region, etc.) and get an estimate of the monthly cost.
    - After estimating the cost, Azure will often provide cost optimization recommendations, such as switching to reserved instances, adjusting VM sizes, or optimizing storage types.

For **Monitoring and Logging**:

1. **Azure Monitor Setup**:
    - Navigate to Azure Monitor in the Azure portal.
    - Under "Monitor," select **Metrics** to track resource health, performance, and utilization (like CPU, memory, etc.).
    - Set up the **Log Analytics Workspace** if you haven't already, and configure monitoring for both your Virtual Machines and Databases by linking them to this workspace.
2. **Alerts for CPU Usage**:
    - In **Azure Monitor**, go to **Alerts** and click **+ New alert rule**.
    - Select the **Virtual Machine** resource for which you want to monitor CPU usage.
    - Choose the **metric** (CPU utilization), set the threshold to 80%, and configure the alert actions (e.g., email notification).
    - Save and enable the alert rule to start receiving notifications whenever CPU usage exceeds the 80% threshold.

azure-pipeline.yml

```yaml
trigger:
  - main

pool:
  vmImage: 'ubuntu-latest'

variables:
  - group: MyAzureTask # Link your variable group

steps:
  # Step 1: Checkout Code
  - task: Checkout@1

  # Step 2: Set Up Java Environment
  - task: JavaToolInstaller@0
    inputs:
      versionSpec: '17'
      jdkArchitecture: 'x64'

  # Step 3: Build the Spring Petclinic Application
  - script: |
      ./mvnw clean package
    displayName: 'Build with Maven'

  # Step 4: Copy the JAR file to the Azure VM
  - task: CopyFilesOverSSH@0
    inputs:
      sshEndpoint: 'AzureVM' # Define this in the service connections
      sourceFolder: '$(System.DefaultWorkingDirectory)/target'
      contents: '*.jar'
      targetFolder: '/home/$(VM_USERNAME)/petclinic'

  # Step 5: SSH into the VM and Deploy the Application
  - task: SSH@0
    inputs:
      sshEndpoint: 'AzureVM' # Define this in the service connections
      runOptions: 'commands'
      commands: |
        cd /home/$(VM_USERNAME)/petclinic
        nohup java -jar *.jar > app.log 2>&1 &
    displayName: 'Start Application on VM'
```