**# Azure IaaC with Terraform**

This project defines basic cloud infrastructure on *\*Microsoft Azure\** using *\*Terraform\**. The goal is to demonstrate how to deploy:

- Virtual Network

- Subnets

- Network Security Groupda3

- Windows Virtual Machine

- Load Balancer

**## Tools Used**

- Cloud Provider: Microsoft Azure

- IaaC Tool: Terraform v1.5+

- Modules: Native resources defined in HCL

- CI/CD: ADO pipeline

**## Resource Choices**

| Resource          | Purpose |

|---------          |--------|

| Virtual Network   | Isolate cloud resources securely |

| Subnet            | Logical subdivision of the network |

| NSG               | Control traffic to/from VM |

| Virtual Machine   | Compute resource for running applications |

| Load Balancer     | Distribute incoming traffic |

| Public IP         | Expose the load balancer to the internet |

**#   Directory Structure**

IaaC/

├── environments/                       # Environment-specific configs

│   ├── dev/

│   │   ├── backend.tf                   # Remote backend configuration for dev

│   │   ├── dev.tfvars                   # Variables specific to dev

│   │   ├── main.tf                      # References modules with dev-specific

│   │   ├── provider.tf                  # Provider and Terraform version

│   │   ├── variables                    # Variables specific to dev

│   │

│   └── prod/

│       ├── backend.tf

│   │   ├── dev.tfvars

│   │   ├── main.tf

│   │   ├── provider.tf

│   │   ├── variables

│

├── modules/                            # Reusable module definitions

│   ├── VirtualNetwork/

│   │   ├── main.tf

│   │   ├── variables.tf

│   │   ├── outputs.tf

│   │

│   ├── VirtualMachine/

│   │   ├── main.tf

│   │   ├── variables.tf

│   │   ├── outputs.tf

│   │

│   ├── NetworkSecurityGroup/

│   │   ├── main.tf

│   │   ├── variables.tf

│   │   ├── outputs.tf

│   │

│   └── LoadBalancer/

│       ├── main.tf

│       ├── variables.tf

│       ├── outputs.tf

│

├── scripts/                          # Optional: Custom scripts

│

├── pipeline/                         # CI/CD pipeline configurations

│   ├── ci.yml           # Azure DevOps pipeline definition

│

│   .terraform                         # Terraform metadata

└── .terraform.lock.hcl               # Terraform provider lock file (auto-generated)

**## How to Deploy**

Make sure you have Terraform and Azure CLI installed. Then run the following for manual provisioning:

**# Log in to Azure**

az login

**# Initialize Terraform**

terraform init

**# Preview the infrastructure**

terraform plan -var-file="dev.tfvars"

**# Deploy the infrastructure**

terraform apply -var-file="dev.tfvars"

---------

**# Pipeline Overview**

The pipeline consists of multiple stages:

**# Static Code Analysis – runCheckov**

    -   Installs Checkov, an open-source tool for static analysis of Terraform templates.

    -   Runs a Checkov scan on the source code to catch misconfigurations or security issues early.

    -   Publishes test results to Azure DevOps Test tab.

**# Terraform Validation – validate**

    -   Installs Terraform CLI.

    -   Initializes the backend to use remote state storage (Azure Blob).

    -   Runs terraform validate to ensure the configuration is syntactically correct.

**# Terraform Plan**

    -   Initializes Terraform.

    -   Executes terraform plan to preview changes.

    -   Converts the plan to JSON and runs a Checkov scan on the plan output to catch policy violations.

    -   Publishes results.

    -   Awaits manual validation/approval (maximum 3-day timeout).

**# Terraform Apply**

    -   Only runs after successful plan and manual approval.

    -   Executes terraform plan again to ensure the state hasn’t changed.

    -   Applies changes with terraform apply.

**#   Deployment Instructions**

**#   Step-by-step**

    -   Clone the Repository

    -   git clone https://github.com/kesirsre/terraform-azure-infra.git

    -   cd terraform-azure-infra

**#   Connect Azure DevOps to Git Repo**

    -   Navigate to Azure DevOps > Pipelines > New Pipeline

    -   Select your Git repository

    -   Create the Pipeline

    -   Choose YAML path: ci.yml

    -   Set Up Azure Service Connection

    -   Go to Project Settings > Service Connections

    -   Create a new ARM service connection with sufficient permissions

**# Commit and Push Changes**

    -   git add .

    -   git commit -m "Initial commit with pipeline"

    -   git push origin master

**#   Trigger the Pipeline**

    -   The pipeline triggers on every push to main.

    -   Or you can run it manually from Azure DevOps.

**# Screenshots for reference**

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.