Infrastructure as Code (IaC) - Complete Documentatio-1

Status	Not Started

Document Version: 7.0

Date: January 2026

Audience: All Technical Levels - Developers, DevOps, Architects

Executive Summary: The LEGO Analogy

Think of Infrastructure as Code as Building with LEGO:

Traditional IT	Infrastructure as Code	LEGO Analogy
Manual server setup	Code defines servers	Follow picture instructions
"Snowflake" servers	Identical servers every time	Identical LEGO sets
No documentation	Code IS documentation	Instruction manual
Hard to reproduce	Easy to recreate	Build same set anytime
Slow changes	Rapid deployment	Quick assembly

Core Idea: Define your entire infrastructure (servers, networks, databases) in code files, just like you define application features.

1. What is Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) is the practice of managing and provisioning computing infrastructure through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

Simple Explanation:

Imagine you could write a recipe that automatically builds your entire kitchen (ovens, counters, sinks) exactly the same way every time. IaC is that recipe for your IT infrastructure.

1.1 The Evolution of Infrastructure Management

Traditional (Manual/ClickOps):

```
# OLD WAY: Manual Process
1. Login to portal → Click "Create VM"
2. Select size → Choose image → Configure network
3. Install software → Configure firewall
4. Repeat for each environment...
5. Hope you remember all steps!
```

Infrastructure as Code:

hcl

```
# NEW WAY: Code Definition
resource "azurerm_virtual_machine" "web_server" {
  name                = "web-vm-01"
  location            = "eastus"
  resource_group_name = azurerm_resource_group.main.name
  vm_size             = "Standard_B2s"

  network_interface_ids = [azurerm_network_interface.main.id]

  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }
}
```

1.2 Why IaC Matters: The Business Impact

Before IaC (Costly):

- 2 hours to deploy a server
- 80% chance of human error
- 1 week to rebuild after disaster
- Inconsistent environments

After IaC (Efficient):

- 5 minutes to deploy a server
- 99.9% consistency
- 1 hour to rebuild everything
- Identical dev/test/prod environments

---

2. Core IaC Principles Explained

2.1 Principle 1: Declarative Infrastructure Modeling

Definition: Instead of writing step-by-step instructions (imperative), you declare the desired end state, and the IaC tool figures out how to achieve it.

What it means: Describe WHAT you want, not HOW to build it.

Imperative (HOW - Bad):

bash

```bash
# Manual steps - How to build
1. Create resource group
2. Create virtual network
3. Create subnet
4. Create NIC
5. Create VM
6. Install Nginx
7. Configure firewall
```

Declarative (WHAT - Good):

hcl

```
# Terraform - What you want
resource "azurerm_linux_virtual_machine" "web" {
  name                  = "web-server"
  size                  = "Standard_B2s"
  admin_username        = "adminuser"
  network_interface_ids = [azurerm_network_interface.web.id]

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
  }

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  custom_data = base64encode(templatefile("${path.module}/cloud-init.yaml", {
    server_role = "web"
  }))
}
```

Visual: Declarative vs Imperative

text

```
DECLARATIVE:                    IMPERATIVE:
┌─────────────────┐             ┌─────────────────┐
│ DESIRED STATE   │             │      STEPS      │
│ - 2 Web VMs     │             │ 1. Create VM1   │
│ - 1 Load Balancer│            │ 2. Create VM2   │
│ - 1 Database    │             │ 3. Install LB   │
└────────┬────────┘             │ 4. Configure DB│
         │                      └────────┬────────┘
         ▼                               ▼
┌─────────────────┐             ┌─────────────────┐
```

```
│   IaC TOOL      │               │   ADMIN/Script  │
│  Figures out    │               │  Follows steps  │
│    how to       │               │    exactly      │
│  make it happen │               │                 │
└─────────────────┘               └─────────────────┘
```

2.2 Principle 2: Idempotency

Definition: No matter how many times you run the same IaC code, the end result will always be the same. The operation produces the same result whether executed once or multiple times.

Simple Explanation:

Like a light switch - flipping it up always turns the light on, flipping it down always turns it off. It doesn't matter how many times you flip it.

What it means: Running the same code multiple times produces the same result.

Real-world analogy: A light switch

- Click ON → Light ON
- Click ON again → Still ON (not brighter)
- Click OFF → Light OFF
- Click OFF again → Still OFF

Non-idempotent (Dangerous):

bash

```
# BAD: Running twice creates duplicates
az vm create --name web-vm --resource-group my-rg ...
# Run again → Creates second VM with same name → ERROR
```

Idempotent (Safe):

hcl

```
# GOOD: Terraform ensures idempotency
resource "azurerm_virtual_machine" "web" {
  name = "web-vm"  # Unique identifier
}
# Run apply multiple times → Only one VM exists
```

Idempotency in Action:

bash

```
# First run: Creates resources
$ terraform apply
Plan: 10 to add, 0 to change, 0 to destroy.
Apply complete! Resources: 10 added.

# Second run: Nothing changes (idempotent!)
$ terraform apply
Plan: 0 to add, 0 to change, 0 to destroy.
Apply complete! Resources: 0 added, 0 changed.

# Modify code: Updates only what changed
$ terraform apply
Plan: 0 to add, 1 to change, 0 to destroy.
Apply complete! Resources: 0 added, 1 changed.
```

2.3 Principle 3: Version Control for Infrastructure

Definition: Store IaC code in version control systems (like Git) to track changes, collaborate, and maintain history of infrastructure evolution.

What it means: Store infrastructure code in Git, track changes, collaborate.

Traditional IT (No Version Control):

text

```
Monday:    Server configured by Alice
Tuesday:   Bob makes undocumented changes
Wednesday: Server crashes
Thursday:  Nobody knows what changed!
```

IaC with Git:

bash

```
# Every change is tracked
git log --oneline
# Output:
# a1b2c3d (HEAD → main) Add monitoring
# e4f5g6h Update VM size
# i7j8k9l Initial infrastructure commit

# See exactly what changed
git show a1b2c3d
# Shows added monitoring configuration
```

Git Workflow for Infrastructure:

text

```
        Developer Workflow
┌─────────────────────────────────────┐
│  1. git checkout -b feature/new-vm  │
│  2. Edit terraform files            │
│  3. terraform plan                  │
│  4. git commit -m "Add new VM"      │
│  5. git push                        │
│                                     │
│  6. Create Pull Request             │
│  7. Peer review                     │
│  8. Automated tests run             │
│  9. Merge to main                   │
│ 10. CI/CD pipeline deploys          │
└─────────────────────────────────────┘
```

2.4 Principle 4: Automation & Reproducibility

Definition: IaC enables automated deployment, creates reproducible environments, and provides disaster recovery through code.

What it means: Deploy identical infrastructure anywhere, anytime.

Disaster Recovery Scenario:

Without IaC (Days/Week):

text

```
Datacenter fails!
1. Panic
2. Try to remember all servers
3. Manual recreation (different configs)
4. Testing issues
5. 5 days downtime
```

With IaC (Hours):

text

```
Datacenter fails!
1. Run: terraform apply
2. Wait 30 minutes
3. Infrastructure restored
4. Run automated tests
5. 2 hours downtime
```

Reproducibility Example:

bash

```bash
# Deploy to development
terraform workspace select dev
terraform apply  # Creates dev environment

# Deploy to staging (identical but different scale)
terraform workspace select staging
terraform apply  # Creates staging environment

# Deploy to production (identical but larger)
terraform workspace select prod
terraform apply  # Creates production environment
```

```
# All identical in configuration, different in scale!
```

3. IaC Tools Comparison

3.1 Popular IaC Tools

Tool	Type	Language	Best For	Azure Integration
Terraform	Declarative	HCL (Hashicorp)	Multi-cloud	Excellent
Azure ARM	Declarative	JSON	Azure-only	Native
Bicep	Declarative	Bicep DSL	Azure-only	Excellent (ARM improved)
Pulumi	Imperative/Declarative	Python/TS/Go	Developers	Good
Ansible	Imperative	YAML	Configuration	Good
Chef/Puppet	Imperative	Ruby/Domain	Configuration	Good

3.2 Choosing the Right Tool

Decision Matrix:

text

```
What's your primary cloud?
├── Multi-cloud → Terraform
├── Azure only → Bicep or ARM
└── On-prem + cloud → Ansible

Who will write/maintain?
├── DevOps/Infra team → Terraform/Bicep
├── Developers → Pulumi
└── Ops team → Ansible/Chef

Need existing skills?
├── Know JSON/YAML → ARM/Ansible
├── Know programming → Pulumi
└── Willing to learn → Terraform/Bicep
```

4. Hands-On: Terraform for Azure

4.1 Setup Terraform Environment

bash

```
#!/bin/bash
# File: setup-terraform.sh
# Description: Complete Terraform setup for Azure

echo "=== TERRAFORM SETUP FOR AZURE ==="

# 1. Install Terraform (Linux/Mac)
echo "1. Installing Terraform..."
wget https://releases.hashicorp.com/terraform/1.5.0/terraform_1.5.0_linux_amd64.zip
unzip terraform_1.5.0_linux_amd64.zip
sudo mv terraform /usr/local/bin/
terraform version

# 2. Install Azure CLI
echo "2. Installing Azure CLI..."
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
az version

# 3. Login to Azure
echo "3. Logging into Azure..."
az login

# 4. Set subscription
echo "4. Setting subscription..."
SUBSCRIPTION_ID=$(az account show --query id -o tsv)
az account set --subscription $SUBSCRIPTION_ID
```

```
# 5. Create Service Principal for Terraform
echo "5. Creating Service Principal..."
az ad sp create-for-rbac \
  --name "terraform-sp" \
  --role "Contributor" \
  --scopes "/subscriptions/$SUBSCRIPTION_ID" \
  --years 2

# Save credentials to terraform.tfvars
cat > terraform.tfvars <<EOF
subscription_id = "$SUBSCRIPTION_ID"
tenant_id       = "$(az account show --query tenantId -o tsv)"
client_id       = "$(az ad sp list --display-name terraform-sp --query [0].appId -o tsv)"
client_secret   = "REPLACE_WITH_SECRET"
EOF

echo "✅ Terraform setup complete!"
echo "📋 Next: Replace client_secret in terraform.tfvars"
```

4.2 Basic Terraform Project Structure

text

```
my-iac-project/
├── providers.tf         # Cloud provider configuration
├── variables.tf         # Input variables
├── terraform.tfvars     # Variable values (gitignored)
├── outputs.tf           # Output values
├── main.tf              # Main infrastructure
├── modules/             # Reusable modules
│   ├── networking/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   └── compute/
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
├── environments/       # Environment configurations
│   ├── dev/
│   │   └── terraform.tfvars
│   ├── staging/
│   │   └── terraform.tfvars
│   └── prod/
│       └── terraform.tfvars
└── scripts/           # Helper scripts
    └── deploy.sh
```

4.3 Complete Terraform Example: 3-Tier Architecture

Step 1: providers.tf

hcl

```
# File: providers.tf
terraform {
  required_version = ">= 1.5.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.71.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.5.1"
    }
  }

  # Store state remotely in Azure Storage
  backend "azurerm"{
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstate${replace(lower(substr(md5("myproject"),0,
8)), "-", "")}"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
  }
}

provider "azurerm"{
  features {
    resource_group {
      prevent_deletion_if_contains_resources = false
    }
  }

  # Credentials from environment variables or terraform.tfvars
  subscription_id = var.subscription_id
  tenant_id       = var.tenant_id
  client_id       = var.client_id
  client_secret   = var.client_secret
}
```

Step 2: variables.tf

hcl

```
# File: variables.tf

# Azure Authentication
variable "subscription_id"{
  description = "Azure subscription ID"
  type        = string
  sensitive   = true
}

variable "tenant_id"{
  description = "Azure tenant ID"
  type        = string
  sensitive   = true
}

variable "client_id"{
  description = "Service Principal Client ID"
  type        = string
  sensitive   = true
}

variable "client_secret"{
  description = "Service Principal Client Secret"
  type        = string
  sensitive   = true
}

# Project Variables
variable "project_name"{
  description = "Project name used for resource naming"
  type        = string
  default     = "myapp"
}

variable "environment"{
  description = "Environment (dev, staging, prod)"
  type        = string
  default     = "dev"

  validation {
    condition     = contains(["dev", "staging", "prod"], var.environment)
    error_message = "Environment must be dev, staging, or prod."
  }
}

variable "location"{
  description = "Azure region"
  type        = string
  default     = "eastus"
}
```

```
variable "tags"{
  description = "Tags to apply to all resources"
  type        = map(string)
  default = {
    Project     = "MyApp"
    Environment = "Development"
    ManagedBy   = "Terraform"
    CostCenter  = "IT"
  }
}

# Network Configuration
variable "vnet_address_space"{
  description = "VNet address space"
  type        = list(string)
  default     = ["10.0.0.0/16"]
}

variable "web_subnet_cidr"{
  description = "Web tier subnet CIDR"
  type        = string
  default     = "10.0.1.0/24"
}

variable "app_subnet_cidr"{
  description = "App tier subnet CIDR"
  type        = string
  default     = "10.0.2.0/24"
}

variable "db_subnet_cidr"{
  description = "Database tier subnet CIDR"
  type        = string
  default     = "10.0.3.0/24"
}
```

```
# VM Configuration
variable "web_vm_count"{
  description = "Number of web tier VMs"
  type        = number
  default     = 2
}

variable "app_vm_count"{
  description = "Number of app tier VMs"
  type        = number
  default     = 2
}

variable "vm_size"{
  description = "Virtual machine size"
  type        = map(string)
  default = {
    web = "Standard_B2s"
    app = "Standard_B2s"
  }
}

variable "admin_username"{
  description = "Admin username for VMs"
  type        = string
  sensitive   = true
  default     = "azureadmin"
}

variable "admin_password"{
  description = "Admin password for VMs"
  type        = string
  sensitive   = true
  default     = ""
}
```

Step 3: Create Resource Group

hcl

```hcl
# File: main.tf (Part 1 - Resource Group)

# Generate random suffix for unique resource names
resource "random_string" "suffix" {
  length  = 8
  special = false
  upper   = false
}

# Create Resource Group
resource "azurerm_resource_group" "main" {
  name     = "rg-${var.project_name}-${var.environment}-${random_string.suffix.result}"
  location = var.location
  tags     = var.tags

  lifecycle {
    prevent_destroy = var.environment == "prod" ? true : false
  }
}
```

Step 4: Create Networking Module

hcl

```hcl
# File: modules/networking/main.tf

# Create Virtual Network
resource "azurerm_virtual_network" "main" {
  name                = "vnet-${var.project_name}-${var.environment}"
  resource_group_name = var.resource_group_name
  location            = var.location
  address_space       = var.vnet_address_space
  tags                = var.tags
}
```

```
# Create Web Tier Subnet
resource "azurerm_subnet" "web" {
  name                 = "snet-web"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [var.web_subnet_cidr]

  # Service Endpoints for PaaS services
  service_endpoints = ["Microsoft.Storage", "Microsoft.KeyVault"]
}

# Create App Tier Subnet
resource "azurerm_subnet" "app" {
  name                 = "snet-app"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [var.app_subnet_cidr]

  # Delegate to AKS if needed
  # delegations {
  #   name = "aks-delegation"
  #   service_delegation {
  #     name = "Microsoft.ContainerService/managedClusters"
  #   }
  # }
}

# Create Database Tier Subnet
resource "azurerm_subnet" "db" {
  name                 = "snet-db"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [var.db_subnet_cidr]

  # Enable Service Endpoint for Azure SQL
  service_endpoints = ["Microsoft.Sql"]
```

```
}

# Create Network Security Group for Web Tier
resource "azurerm_network_security_group" "web" {
  name                = "nsg-web"
  resource_group_name = var.resource_group_name
  location            = var.location
  tags                = var.tags

  # Allow HTTP from Internet
  security_rule {
    name                       = "AllowHTTP"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "80"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }

  # Allow HTTPS from Internet
  security_rule {
    name                       = "AllowHTTPS"
    priority                   = 110
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "443"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }

  # Allow SSH from specific IPs
  security_rule {
    name                       = "AllowSSH"
    priority                   = 120
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = var.admin_ip
    destination_address_prefix = "*"
  }

  # Deny all other inbound
  security_rule {
    name                       = "DenyAllInbound"
    priority                   = 4096
    direction                  = "Inbound"
    access                     = "Deny"
    protocol                   = "*"
    source_port_range          = "*"
    destination_port_range     = "*"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}
```

Step 5: Create Compute Module

hcl

```
# File: modules/compute/main.tf

# Create Public IP for Load Balancer
resource "azurerm_public_ip" "web_lb" {
  name                = "pip-web-lb"
  resource_group_name = var.resource_group_name
  location            = var.location
  allocation_method   = "Static"
  sku                 = "Standard"
  tags                = var.tags
}

# Create Load Balancer
resource "azurerm_lb" "web" {
  name                = "lb-web"
  resource_group_name = var.resource_group_name
  location            = var.location
  sku                 = "Standard"
  tags                = var.tags

  frontend_ip_configuration {
    name                 = "frontend"
    public_ip_address_id = azurerm_public_ip.web_lb.id
  }
}

# Create Load Balancer Backend Pool
resource "azurerm_lb_backend_address_pool" "web" {
  loadbalancer_id = azurerm_lb.web.id
  name            = "backend-pool"
}

# Create Availability Set for Web VMs
resource "azurerm_availability_set" "web" {
  name                         = "as-web"
  resource_group_name          = var.resource_group_name
  location                     = var.location
  platform_fault_domain_count  = 2
  platform_update_domain_count = 5
  managed                      = true
  tags                         = var.tags
}

# Create Network Interface for Web VMs
resource "azurerm_network_interface" "web" {
  count               = var.vm_count
  name                = "nic-web-vm${count.index + 1}"
  resource_group_name = var.resource_group_name
  location            = var.location
  tags                = var.tags

  ip_configuration {
    name                          = "internal"
    subnet_id                     = var.subnet_id
    private_ip_address_allocation = "Dynamic"
  }
}

# Create Virtual Machines
resource "azurerm_linux_virtual_machine" "web" {
  count                           = var.vm_count
  name                            = "vm-web-${count.index + 1}"
  resource_group_name             = var.resource_group_name
  location                        = var.location
  size                            = var.vm_size
  admin_username                  = var.admin_username
  admin_password                  = var.admin_password
  disable_password_authentication = false
  availability_set_id             = azurerm_availability_set.web.id
  tags                            = var.tags

  network_interface_ids = [
    azurerm_network_interface.web[count.index].id
  ]

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
    version   = "latest"
  }

  # Cloud-init configuration
  custom_data = base64encode(templatefile("${path.module}/cloud-init.yaml", {
    hostname = "web-${count.index + 1}"
    role     = "web"
  }))

  # Auto-shutdown to save costs (not in production)
  provisioner "local-exec"{
    command = <<-EOT
      az vm auto-shutdown \
        --resource-group ${var.resource_group_name} \
        --name vm-web-${count.index + 1} \
        --time 1900 \
        --email "admin@company.com" \
        --webhook "https://hooks.slack.com/services/..."
    EOT
    when    = create
  }
}
```

Step 6: Database Module

hcl

```
# File: modules/database/main.tf

# Create PostgreSQL Server
resource "azurerm_postgresql_server" "main" {
  name                        = "psql-${var.project_name}-${var.environment}"
  resource_group_name          = var.resource_group_name
  location                    = var.location
  version                     = "11"
  administrator_login          = var.db_admin_username
  administrator_login_password = var.db_admin_password

  sku_name   = "GP_Gen5_2"
  storage_mb = 5120

  backup_retention_days        = 7
  geo_redundant_backup_enabled = false
  auto_grow_enabled            = true

  ssl_enforcement_enabled          = true
  ssl_minimal_tls_version_enforced = "TLS1_2"

  tags = var.tags
}

# Create Database
resource "azurerm_postgresql_database" "app" {
  name                = "appdb"
  resource_group_name = var.resource_group_name
  server_name         = azurerm_postgresql_server.main.name
  charset             = "UTF8"
  collation           = "English_United States.1252"
}

# Configure VNet Rules
resource "azurerm_postgresql_virtual_network_rule" "app_subnet" {
  name                = "app-subnet-rule"
  resource_group_name = var.resource_group_name
  server_name         = azurerm_postgresql_server.main.name
  subnet_id           = var.app_subnet_id
}

# Configure Firewall Rules
resource "azurerm_postgresql_firewall_rule" "app_tier" {
  name                = "app-tier"
  resource_group_name = var.resource_group_name
  server_name         = azurerm_postgresql_server.main.name
  start_ip_address    = var.app_subnet_start_ip
  end_ip_address      = var.app_subnet_end_ip
}

# Deny all other IPs (default)
resource "azurerm_postgresql_firewall_rule" "deny_all" {
  name                = "deny-all"
  resource_group_name = var.resource_group_name
  server_name         = azurerm_postgresql_server.main.name
  start_ip_address    = "0.0.0.0"
  end_ip_address      = "0.0.0.0"
}
```

Step 7: Putting It All Together

hcl

```
# File: main.tf (Complete)

# Local variables for naming
locals {
  resource_prefix = "${var.project_name}-${var.environment}"
}

# Call Networking Module
module "networking"{
  source = "./modules/networking"

  resource_group_name = azurerm_resource_group.main.name
  location            = var.location
  project_name        = var.project_name
  environment         = var.environment
  tags                = var.tags

  vnet_address_space = var.vnet_address_space
  web_subnet_cidr    = var.web_subnet_cidr
  app_subnet_cidr    = var.app_subnet_cidr
  db_subnet_cidr     = var.db_subnet_cidr
  admin_ip           = "203.0.113.1" # Replace with your IP
}

# Call Web Tier Compute Module
module "web_tier"{
  source = "./modules/compute"

  resource_group_name = azurerm_resource_group.main.name
  location            = var.location
  project_name        = var.project_name
  environment         = var.environment
  tags                = var.tags

  tier                = "web"
  vm_count            = var.web_vm_count
  vm_size             = var.vm_size["web"]
  subnet_id           = module.networking.web_subnet_id
  admin_username      = var.admin_username
  admin_password      = var.admin_password
}

# Call App Tier Compute Module
module "app_tier"{
  source = "./modules/compute"

  resource_group_name = azurerm_resource_group.main.name
  location            = var.location
  project_name        = var.project_name
  environment         = var.environment
  tags                = var.tags

  tier                = "app"
  vm_count            = var.app_vm_count
  vm_size             = var.vm_size["app"]
  subnet_id           = module.networking.app_subnet_id
  admin_username     = var.admin_username
  admin_password     = var.admin_password
}

# Call Database Module
module "database"{
  source = "./modules/database"

  resource_group_name = azurerm_resource_group.main.name
  location            = var.location
  project_name        = var.project_name
  environment         = var.environment
  tags                = var.tags

  db_admin_username = "psqladmin"
  db_admin_password = random_password.db_password.result
  app_subnet_id     = module.networking.app_subnet_id

  # Calculate subnet IP range
  app_subnet_start_ip = cidrhost(var.app_subnet_cidr, 4)
  app_subnet_end_ip   = cidrhost(var.app_subnet_cidr, 254)
}

# Generate random password for database
resource "random_password" "db_password" {
  length           = 16
  special          = true
  override_special = "!@#$%&*()-_=+[]{}<>:?"
}

# Store secrets in Azure Key Vault
resource "azurerm_key_vault" "secrets" {
  name                = "kv-${local.resource_prefix}"
  resource_group_name = azurerm_resource_group.main.name
  location            = var.location
  tenant_id           = var.tenant_id
  sku_name            = "standard"
  access_policy {
    tenant_id = var.tenant_id
    object_id = var.client_id # Terraform Service Principal

    secret_permissions = [
      "Get", "List", "Set", "Delete", "Recover", "Backup", "Restore"
    ]
  }

  tags = var.tags
}

# Store database password in Key Vault
resource "azurerm_key_vault_secret" "db_password" {
  name         = "db-admin-password"
  value        = random_password.db_password.result
  key_vault_id = azurerm_key_vault.secrets.id

  depends_on = [azurerm_key_vault.secrets]
}
```

Step 8: outputs.tf

hcl

```
# File: outputs.tf

output "resource_group_name"{
  description = "Resource Group name"
  value       = azurerm_resource_group.main.name
}

output "web_lb_public_ip"{
  description = "Public IP of Web Load Balancer"
  value       = module.web_tier.lb_public_ip
}
```

```
output "web_vm_private_ips"{
  description = "Private IPs of Web VMs"
  value       = module.web_tier.vm_private_ips
}

output "app_vm_private_ips"{
  description = "Private IPs of App VMs"
  value       = module.app_tier.vm_private_ips
}

output "database_fqdn"{
  description = "Fully Qualified Domain Name of PostgreSQL"
  value       = module.database.postgresql_fqdn
}

output "key_vault_name"{
  description = "Name of the Key Vault containing secrets"
  value       = azurerm_key_vault.secrets.name
}

output "admin_username"{
  description = "Admin username for VMs"
  value       = var.admin_username
  sensitive   = true
}

# Generate inventory file for Ansible
resource "local_file" "ansible_inventory" {
  filename = "${path.module}/inventory.ini"
  content = templatefile("${path.module}/templates/inventory.tmpl", {
    web_ips = module.web_tier.vm_private_ips
    app_ips = module.app_tier.vm_private_ips
    db_fqdn = module.database.postgresql_fqdn
  })
}
```

```
# Generate README with connection info
resource "local_file" "readme" {
  filename = "${path.module}/DEPLOYMENT_INFO.md"
  content = <←EOT
    # Deployment Information

    ## Environment: ${var.environment}
    ## Created: ${timestamp()}

    ## Connection Information:

    ### Web Application
    URL: http://${module.web_tier.lb_public_ip}

    ### Virtual Machines
    Web Tier VMs:
    %{ for ip in module.web_tier.vm_private_ips ~}
    - ${ip} (SSH: azureadmin@${ip})
    %{ endfor ~}

    App Tier VMs:
    %{ for ip in module.app_tier.vm_private_ips ~}
    - ${ip} (SSH: azureadmin@${ip})
    %{ endfor ~}

    ### Database
    PostgreSQL: ${module.database.postgresql_fqdn}
    Database: appdb

    ### Secrets
    Key Vault: ${azurerm_key_vault.secrets.name}

    ## Next Steps:
    1. Configure DNS for load balancer
    2. Install SSL certificate
    3. Configure monitoring
    4. Set up backups
  EOT
}
```

5. Terraform Workflow in Practice

5.1 Complete Terraform Workflow Script

bash

```
#!/bin/bash
# File: terraform-workflow.sh
# Description: Complete Terraform workflow

echo "=== TERRAFORM WORKFLOW ==="

# 1. INITIALIZE
echo "1. Initializing Terraform..."
terraform init -upgrade

# 2. FORMAT CODE
echo "2. Formatting Terraform code..."
terraform fmt -recursive

# 3. VALIDATE CODE
echo "3. Validating Terraform code..."
terraform validate

# 4. SECURITY SCAN
echo "4. Running security scan..."
# Install tfsec if not available
if ! command -v tfsec &> /dev/null; then
    echo "Installing tfsec..."
    brew install tfsec  # or appropriate install for your OS
fi
tfsec .
```

```
# 5. PLAN
echo "5. Creating execution plan..."
terraform plan -out=tfplan.binary

# Convert binary plan to JSON for review
terraform show -json tfplan.binary > tfplan.json

# Show human-readable plan
echo "=== PLAN SUMMARY ==="
terraform show tfplan.binary

# Ask for confirmation
read -p "Apply this plan? (yes/no): " confirm
if [ "$confirm" != "yes" ]; then
  echo "Plan cancelled."
  exit 0
fi

# 6. APPLY
echo "6. Applying infrastructure..."
terraform apply tfplan.binary

# 7. OUTPUT INFORMATION
echo "7. Deployment complete!"
echo ""
echo "=== DEPLOYMENT OUTPUTS ==="
terraform output -json | jq -r 'to_entries[] | "\(.key): \(.value.value)"'

# 8. GENERATE DOCUMENTATION
echo "8. Generating documentation..."
terraform-docs markdown table --output-file README.md .

echo "✅ Terraform workflow complete!"
```

5.2 Git-Based Collaboration Workflow

bash

```
#!/bin/bash
# File: git-terraform-workflow.sh
# Description: Git-based Terraform collaboration

echo "=== GIT-BASED TERRAFORM WORKFLOW ==="

# Developer creates feature branch
git checkout -b feature/add-monitoring

# Make changes to Terraform files
cat >> main.tf << 'EOF'
# Add monitoring resources
resource "azurerm_monitor_action_group" "alerts" {
  name                = "CriticalAlerts"
  resource_group_name = azurerm_resource_group.main.name
  short_name          = "critalerts"

  email_receiver {
    name          = "sendtoadmin"
    email_address = "admin@company.com"
  }
}
EOF

# Format and validate
terraform fmt
terraform validate

# Run tests (if any)
echo "Running tests..."
# Add your test commands here

# Commit changes
git add .
git commit -m "feat: Add monitoring and alerting
```

```
- Add Azure Monitor Action Group
- Configure email alerts
- Update documentation"

# Push to remote
git push origin feature/add-monitoring

echo "✅ Changes pushed. Create Pull Request for review."
```

5.3 CI/CD Pipeline Example (GitHub Actions)

yaml

```
# File: .github/workflows/terraform.yml
name: Terraform

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  terraform:
    name: Terraform Plan/Apply
    runs-on: ubuntu-latest

    env:
      ARM_CLIENT_ID: ${{ secrets.ARM_CLIENT_ID }}
      ARM_CLIENT_SECRET: ${{ secrets.ARM_CLIENT_SECRET }}
      ARM_SUBSCRIPTION_ID: ${{ secrets.ARM_SUBSCRIPTION_ID }}
      ARM_TENANT_ID: ${{ secrets.ARM_TENANT_ID }}

    steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        terraform_version: 1.5.0

    - name: Terraform Init
      run: terraform init

    - name: Terraform Format
      run: terraform fmt -check -recursive

    - name: Terraform Validate
      run: terraform validate

    - name: Terraform Security Scan
      uses: aquasecurity/tfsec-action@v1.0.0

    - name: Terraform Plan
      if: github.event_name == 'pull_request'
      run: terraform plan -out=tfplan

    - name: Terraform Apply
      if: github.ref == 'refs/heads/main' && github.event_name == 'push'
      run: terraform apply -auto-approve tfplan

    - name: Update Outputs
      if: github.ref == 'refs/heads/main' && github.event_name == 'push'
      run: |
        terraform output -json > terraform_outputs.json
        echo "DEPLOYMENT_OUTPUTS<<EOF" >> $GITHUB_ENV
        cat terraform_outputs.json >> $GITHUB_ENV
        echo "EOF" >> $GITHUB_ENV
```

6. Advanced IaC Concepts

6.1 State Management

Why State Matters:

hcl

```
# Terraform State tracks reality
# File: terraform.tfstate (simplified)
{
  "resources": [
    {
      "type": "azurerm_virtual_machine",
      "name": "web_server",
      "instances": [
        {
          "attributes": {
            "id": "/subscriptions/.../virtualMachines/web-vm-01",
            "name": "web-vm-01",
            "private_ip_address": "10.0.1.4"
          }
        }
      ]
    }
  ]
}
```

Remote State with Azure Storage:

hcl

```
# backend.tf
terraform {
  backend "azurerm"{
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstate12345"
    container_name       = "tfstate"
    key                  = "production.terraform.tfstate"

    # State locking to prevent concurrent modifications
    use_azuread_auth     = true
  }
}
```

State Commands:

bash

```
# Show current state
terraform state list
terraform state show azurerm_virtual_machine.web

# Import existing resources
terraform import azurerm_virtual_machine.web /subscriptions/.../virtualMachines/existing-vm

# Move resources (refactoring)
terraform state mv \
  azurerm_virtual_machine.old_name \
  module.web_tier.azurerm_virtual_machine.new_name

# Remove from state (not destroy)
terraform state rm azurerm_virtual_machine.unmanaged
```

6.2 Workspaces for Multiple Environments

bash

```
# Create workspaces for different environments
terraform workspace new dev
terraform workspace new staging
terraform workspace new prod

# List workspaces
terraform workspace list
# Output:
#   default
# * dev
#   staging
#   prod

# Switch between environments
terraform workspace select staging

# Use workspace-specific variables
# terraform.tfvars
environment = terraform.workspace

# Or separate variable files
terraform plan -var-file="environments/${terraform.workspace}.tfvars"
```

6.3 Dynamic Configuration with Data Sources

hcl

```
# Get information about existing resources
data "azurerm_subscription" "current" {}

data "azurerm_client_config" "current" {}

data "azurerm_virtual_network" "existing" {
  name                = "existing-vnet"
  resource_group_name = "existing-rg"
}

# Use in resources
resource "azurerm_subnet" "new" {
  name                 = "new-subnet"
  resource_group_name  = data.azurerm_virtual_network.existing.resource_group
_name
  virtual_network_name = data.azurerm_virtual_network.existing.name
  address_prefixes     = ["10.0.100.0/24"]
}

# Get latest Ubuntu image
data "azurerm_platform_image" "ubuntu" {
  location  = "eastus"
  publisher = "Canonical"
  offer     = "0001-com-ubuntu-server-jammy"
  sku       = "22_04-lts"

  filter {
    name   = "properties.storageProfile.osDisk.diskSizeGB"
    values = ["30"]
    regex  = false
  }
}

resource "azurerm_virtual_machine" "web" {
  # ... other config ...

  storage_image_reference {
    id = data.azurerm_platform_image.ubuntu.id
  }
}
```

6.4 Advanced Module Patterns

Module Composition:

hcl

```
# File: modules/application/main.tf
module "network"{
  source = "../networking"
  # ... inputs ...
}

module "web_tier"{
  source = "../compute"

  tier              = "web"
  subnet_id         = module.network.web_subnet_id
  security_group_id  = module.network.web_nsg_id
  # ... other inputs ...
}

module "app_tier"{
  source = "../compute"

  tier              = "app"
  subnet_id         = module.network.app_subnet_id
  security_group_id  = module.network.app_nsg_id
  # ... other inputs ...
}

module "database"{
  source = "../database"

  subnet_id         = module.network.db_subnet_id
  allowed_subnet_id  = module.network.app_subnet_id
  # ... other inputs ...
}
```

Conditional Resources:

hcl

```
# Create bastion host only in production
resource "azurerm_bastion_host" "main" {
  count = var.environment == "prod" ? 1 : 0

  name                = "bastion-${var.environment}"
  resource_group_name = azurerm_resource_group.main.name
  location            = azurerm_resource_group.main.location

  ip_configuration {
    name                 = "configuration"
    subnet_id            = azurerm_subnet.bastion[0].id
    public_ip_address_id = azurerm_public_ip.bastion[0].id
  }
}
```

For-Each vs Count:

hcl

```
# Using count (simpler, but less flexible)
resource "azurerm_virtual_machine" "web_count" {
  count = 3

  name = "web-vm-${count.index}"
  # ...
}

# Using for_each (more flexible, better for updates)
locals {
  web_vms = {
    web1 = { size = "Standard_B2s", zone = 1 }
    web2 = { size = "Standard_B2s", zone = 2 }
    web3 = { size = "Standard_B4ms", zone = 3 }
  }
}

resource "azurerm_virtual_machine" "web_foreach" {
  for_each = local.web_vms

  name = "web-vm-${each.key}"
  size = each.value.size
  zone = each.value.zone
  # ...
}
```

7. Testing Infrastructure as Code

7.1 Unit Testing with Terratest

go

```go
// File: test/terraform_test.go
package test

import (
    "testing"
    "github.com/gruntwork-io/terratest/modules/terraform"
    "github.com/stretchr/testify/assert"
)

func TestTerraformAzureInfrastructure(t *testing.T) {
    t.Parallel()

    terraformOptions := &terraform.Options{
        TerraformDir: "../",
        Vars: map[string]interface{}{
            "environment": "test",
            "web_vm_count": 1,
            "app_vm_count": 1,
        },
    }

    // Clean up after test
    defer terraform.Destroy(t, terraformOptions)

    // Deploy infrastructure
    terraform.InitAndApply(t, terraformOptions)

    // Test outputs
    resourceGroupName := terraform.Output(t, terraformOptions, "resource_group_name")
    assert.Contains(t, resourceGroupName, "rg-myapp-test")

    webLBIP := terraform.Output(t, terraformOptions, "web_lb_public_ip")
    assert.NotEmpty(t, webLBIP)
    // Test HTTP connectivity
    url := fmt.Sprintf("http://%s", webLBIP)
    http_helper.HttpGetWithRetry(t, url, 200, "Welcome to", 30, 5*time.Second)
}
```

7.2 Integration Testing

bash

```
#!/bin/bash
# File: test/integration-test.sh

echo "=== INTEGRATION TESTS ==="

# Test 1: Verify resources created
echo "1. Verifying resource creation..."
az resource list --resource-group $(terraform output -raw resource_group_name) --query "[].{Name:name, Type:type}" --output table

# Test 2: Verify network connectivity
echo "2. Testing network connectivity..."
WEB_LB_IP=$(terraform output -raw web_lb_public_ip)
curl -s -o /dev/null -w "%{http_code}" http://$WEB_LB_IP | grep 200

# Test 3: Verify database connectivity
echo "3. Testing database connectivity..."
DB_FQDN=$(terraform output -raw database_fqdn)
DB_PASSWORD=$(az keyvault secret show --vault-name $(terraform output -raw key_vault_name) --name db-admin-password --query value -o tsv)
psql "host=$DB_FQDN port=5432 dbname=appdb user=psqladmin password=$DB_PASSWORD sslmode=require" -c "SELECT 1"

# Test 4: Verify security rules
echo "4. Verifying security rules..."
az network nsg rule list --resource-group $(terraform output -raw resource_group_name) --nsg-name nsg-web --query "[].{Name:name, Access:access, Port:destinationPortRange}" --output table
```

```
echo "✅ All integration tests passed!"
```

7.3 Compliance Testing

hcl

```
# File: policies/compliance.tf
# Azure Policy for compliance

resource "azurerm_policy_definition" "require_tags" {
  name         = "require-tags"
  policy_type  = "Custom"
  mode         = "Indexed"
  display_name = "Require tags on resources"

  policy_rule = <<POLICY_RULE
{
  "if": {
    "field": "[concat('tags[', parameters('tagName'), ']')]",
    "exists": "false"
  },
  "then": {
    "effect": "deny"
  }
}
POLICY_RULE

  parameters = <<PARAMETERS
{
  "tagName": {
    "type": "String",
    "metadata": {
      "displayName": "Tag Name",
      "description": "Name of the tag, such as 'environment'"
    }
  }
}
PARAMETERS
}

# Assign policy
resource "azurerm_policy_assignment" "tag_compliance" {
  name                 = "tag-compliance"
  scope                = azurerm_resource_group.main.id
  policy_definition_id = azurerm_policy_definition.require_tags.id

  parameters = <<PARAMETERS
{
  "tagName": {
    "value": "environment"
  }
}
PARAMETERS
}
```

8. Disaster Recovery with IaC

8.1 Complete DR Strategy

bash

```
#!/bin/bash
# File: disaster-recovery.sh
# Description: Complete disaster recovery with IaC

echo "=== DISASTER RECOVERY PROCEDURE ==="

# 1. Backup current state
echo "1. Backing up current state..."
az storage blob copy start \
  --account-name $(terraform output -raw state_storage_account) \
  --container-name tfstate \
  --source-blob production.terraform.tfstate \
  --destination-blob production.terraform.tfstate.backup-$(date +%Y%m%d-%
H%M%S)

# 2. Restore from backup if needed
echo "2. Available backups:"
az storage blob list \
  --account-name $(terraform output -raw state_storage_account) \
  --container-name tfstate \
  --query "[?contains(name, 'backup')].name" \
  --output table

# 3. Deploy to DR region
echo "3. Deploying to DR region..."
cd terraform/dr-region

# Initialize with DR state
terraform init \
  -backend-config="resource_group_name=terraform-state-rg-dr" \
  -backend-config="storage_account_name=tfstatedr12345" \
  -backend-config="container_name=tfstate" \
  -backend-config="key=dr.terraform.tfstate"

# Deploy DR infrastructure
terraform apply -auto-approve -var="location=westus"

# 4. Update DNS to DR region
echo "4. Updating DNS..."
az network dns record-set a update \
  --resource-group dns-rg \
  --zone-name mycompany.com \
  --name app \
  --target-resource $(terraform output -raw dr_lb_public_ip_id)

echo "✅ Disaster recovery procedure complete!"
```

8.2 Blue-Green Deployment

hcl

```
# File: blue-green-deployment.tf

# Blue environment (current)
module "blue"{
  source = "./modules/environment"

  environment = "blue"
  color       = "blue"
  dns_prefix  = "app-blue"
}

# Green environment (new)
module "green"{
  source = "./modules/environment"

  environment = "green"
  color       = "green"
  dns_prefix  = "app-green"
}

# Traffic Manager for blue-green switching
resource "azurerm_traffic_manager_profile" "app" {
  name                   = "app-tm"
  resource_group_name    = azurerm_resource_group.main.name
  traffic_routing_method = "Weighted"

  dns_config {
    relative_name = "app"
    ttl           = 60
  }

  monitor_config {
    protocol                     = "HTTPS"
    port                         = 443
    path                         = "/health"
    interval_in_seconds          = 30
    timeout_in_seconds           = 10
    tolerated_number_of_failures = 3
  }
}

# Blue endpoint (100% traffic initially)
resource "azurerm_traffic_manager_endpoint" "blue" {
  name                = "blue-endpoint"
  resource_group_name = azurerm_resource_group.main.name
  profile_name        = azurerm_traffic_manager_profile.app.name
  type                = "azureEndpoints"
  target_resource_id  = module.blue.lb_public_ip_id
  weight              = 100
}

# Green endpoint (0% traffic initially)
resource "azurerm_traffic_manager_endpoint" "green" {
  name                = "green-endpoint"
  resource_group_name = azurerm_resource_group.main.name
  profile_name        = azurerm_traffic_manager_profile.app.name
  type                = "azureEndpoints"
  target_resource_id  = module.green.lb_public_ip_id
  weight              = 0
  enabled             = false
}
```

Switch Traffic Script:

bash

```
#!/bin/bash
# File: switch-traffic.sh

# Switch from blue to green
```

```
echo "Switching traffic from blue to green..."

# Enable green endpoint
az network traffic-manager endpoint update \
  --name green-endpoint \
  --profile-name app-tm \
  --resource-group my-rg \
  --type azureEndpoints \
  --enabled true

# Gradually shift traffic
for weight in 10 25 50 75 100; do
  echo "Setting green weight to ${weight}%..."

  az network traffic-manager endpoint update \
    --name blue-endpoint \
    --profile-name app-tm \
    --resource-group my-rg \
    --type azureEndpoints \
    --weight $((100 - weight))

  az network traffic-manager endpoint update \
    --name green-endpoint \
    --profile-name app-tm \
    --resource-group my-rg \
    --type azureEndpoints \
    --weight $weight

  sleep 300  # Wait 5 minutes between changes
done

# Disable blue endpoint
az network traffic-manager endpoint update \
  --name blue-endpoint \
  --profile-name app-tm \
  --resource-group my-rg \
  --type azureEndpoints \
  --enabled false

echo "✅ Traffic switched to green environment!"
```

9. Real-World IaC Project Structure

9.1 Enterprise-Grade Structure

text

```
terraform-enterprise/
├── README.md
├── .gitignore
├── .pre-commit-config.yaml
├── .terraform-version
│
├── modules/
│   ├── _templates/          # Module templates
│   ├── networking/          # Network resources
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   ├── README.md
│   │   └── examples/
│   ├── compute/             # Compute resources
│   ├── database/            # Database resources
│   ├── security/            # Security resources
│   ├── monitoring/          # Monitoring resources
│   └── iam/                 # IAM resources
│
├── environments/
│   ├── shared/              # Shared resources
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── terraform.tfvars
│   │
│   ├── dev/                 # Development
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── terraform.tfvars
│   │   └── backend.tf
│   │
│   ├── staging/             # Staging
│   └── prod/                # Production
│
├── scripts/
│   ├── bootstrap/           # Bootstrap scripts
│   ├── validation/          # Validation scripts
│   └── utilities/           # Utility scripts
│
├── policies/                # Policy definitions
│   ├── security/
│   ├── compliance/
│   └── cost/
│
├── tests/
│   ├── unit/                # Unit tests
│   ├── integration/         # Integration tests
│   └── e2e/                 # End-to-end tests
│
├── docs/
│   ├── architecture/        # Architecture diagrams
│   ├── operations/          # Operational guides
│   └── decisions/           # Architecture Decision Records
│
└── .github/
    ├── workflows/           # GitHub Actions
    └── dependabot.yml       # Dependency updates
```

9.2 GitOps Workflow with Terraform

yaml

```
# File: .github/workflows/gitops-terraform.yaml
name: GitOps Terraform

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

permissions:
  id-token: write
  contents: read

jobs:
  terraform-gitops:
    runs-on: ubuntu-latest
    environment: production

    steps:
    - name: Checkout
      uses: actions/checkout@v3

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        terraform_version: 1.5.0

    - name: Terraform Init
      run: terraform init

    - name: Terraform Plan
      id: plan
      run: |
        terraform plan -out=tfplan
        terraform show -json tfplan > tfplan.json
    - name: Create Plan Summary
      uses: actions/github-script@v6
      with:
        script: |
          const fs = require('fs');
          const plan = JSON.parse(fs.readFileSync('tfplan.json', 'utf8'));

          let summary = '## Terraform Plan Summary\\n\\n';
          summary += `**Resource Changes:** ${plan.resource_changes.length}\\n
\\n`;

          const changes = {
            create: 0,
            update: 0,
            delete: 0
          };

          plan.resource_changes.forEach(change ⇒ {
            changes[change.change.actions[0]]++;
          });

          summary += `- Create: ${changes.create}\\n`;
          summary += `- Update: ${changes.update}\\n`;
          summary += `- Delete: ${changes.delete}\\n`;

          github.rest.issues.createComment({
            issue_number: context.issue.number,
            owner: context.repo.owner,
            repo: context.repo.repo,
            body: summary
          });

    - name: Auto Apply (Approved Changes)
      if: github.event_name == 'push' && github.ref == 'refs/heads/main'
      run: terraform apply -auto-approve tfplan

    - name: Update Deployment Status
```

```
      if: github.event_name == 'push' && github.ref == 'refs/heads/main'
      run: |
        echo "DEPLOYMENT_TIME=$(date -u +'%Y-%m-%dT%H:%M:%SZ')" >>
$GITHUB_ENV
        echo "COMMIT_SHA=${{ github.sha }}" >> $GITHUB_ENV

    - name: Send Notification
      uses: actions/github-script@v6
      with:
        script: |
          github.rest.repos.createDispatchEvent({
            owner: context.repo.owner,
            repo: context.repo.repo,
            event_type: 'deployment_complete',
            client_payload: {
              environment: 'production',
              commit_sha: '${{ env.COMMIT_SHA }}',
              deployment_time: '${{ env.DEPLOYMENT_TIME }}'
            }
          });
```

10. Monitoring and Maintenance

10.1 Drift Detection

bash

```
#!/bin/bash
# File: detect-drift.sh
# Description: Detect infrastructure drift

echo "=== INFRASTRUCTURE DRIFT DETECTION ==="

# 1. Check for manual changes
echo "1. Checking for configuration drift..."
terraform plan -detailed-exitcode
```

```
DRIFT_EXIT_CODE=$?
if [ $DRIFT_EXIT_CODE -eq 2 ]; then
    echo "⚠️  DRIFT DETECTED!"
    echo "Changes found between Terraform state and actual infrastructure."

    # Send alert
    curl -X POST -H 'Content-type: application/json' \
      --data "{\"text\":\"Infrastructure drift detected in $(terraform workspace show)\"}" \
      $SLACK_WEBHOOK_URL
elif [ $DRIFT_EXIT_CODE -eq 0 ]; then
    echo "✅ No drift detected."
else
    echo "❌ Error checking drift."
    exit 1
fi

# 2. Check for security drift
echo "2. Checking for security drift..."
az policy state summarize \
  --resource-group $(terraform output -raw resource_group_name)

# 3. Generate drift report
echo "3. Generating drift report..."
terraform show -json > current_state.json
terraform plan -out=drift_plan
terraform show -json drift_plan > planned_state.json

# Compare states
python3 -c "
import json
with open('current_state.json') as f:
    current = json.load(f)
with open('planned_state.json') as f:
    planned = json.load(f)
```

```
print('Drift Report:')
print('=' * 50)
for resource in planned['resource_changes']:
    if resource['change']['actions'] != ['no-op']:
        print(f\"{resource['type']}.{resource['name']}: {resource['change']['action
s']}\")
"

echo "✅ Drift detection complete!"
```

10.2 Cost Monitoring

hcl

```
# File: modules/cost-management/main.tf

# Create budget
resource "azurerm_consumption_budget_subscription" "monthly" {
  name            = "monthly-budget"
  subscription_id = data.azurerm_subscription.current.id

  amount     = 1000
  time_grain = "Monthly"

  time_period {
    start_date = "2023-01-01T00:00:00Z"
  }

  notification {
    enabled        = true
    threshold      = 80.0
    operator       = "GreaterThan"
    threshold_type = "Actual"

    contact_emails = [
      "finance@company.com",
      "devops@company.com"
    ]
  }

  notification {
    enabled        = true
    threshold      = 100.0
    operator       = "GreaterThan"
    threshold_type = "Actual"

    contact_emails = [
      "finance@company.com",
      "devops@company.com"
    ]
  }
}

# Cost analysis query
resource "azurerm_cost_management_export" "daily" {
  name                         = "daily-cost-export"
  resource_group_id            = azurerm_resource_group.main.id
  recurrence_type              = "Daily"
  recurrence_period_start_date = "2023-01-01T00:00:00Z"

  export_data_storage_location {
    container_id     = azurerm_storage_container.cost_reports.id
    root_folder_path = "/costs"
  }

  export_data_options {
    time_frame = "WeekToDate"
    type       = "Usage"
  }
}
```

11. Migration to IaC

11.1 Migrating Existing Infrastructure

bash

```bash
#!/bin/bash
# File: migrate-to-terraform.sh
# Description: Migrate existing Azure resources to Terraform

echo "=== MIGRATING EXISTING INFRASTRUCTURE TO TERRAFORM ==="

# 1. Discover existing resources
echo "1. Discovering existing resources..."
RESOURCE_GROUP="existing-rg"

# Export existing resources
az resource list --resource-group $RESOURCE_GROUP --query "[].{Name:name,
Type:type, Id:id}" --output table > existing-resources.txt

# 2. Generate Terraform configuration
echo "2. Generating Terraform configuration..."
for resource in $(az resource list --resource-group $RESOURCE_GROUP --query
"[].id" -o tsv); do
  echo "Processing: $resource"

  # Get resource type and name
  resource_type=$(echo $resource | awk -F/ '{print $(NF-1)}')
  resource_name=$(echo $resource | awk -F/ '{print $NF}')

  # Generate import block
  cat >> import.tf <<EOF
import {
  to = azurerm_${resource_type//Microsoft./}.${resource_name//-/}
  id = "$resource"
}
EOF

  # Generate resource block template
  cat >> main.tf <<EOF
resource "azurerm_${resource_type//Microsoft./}" "${resource_name//-/}" {
  # TODO: Populate with actual configuration
  name                = "$resource_name"
  resource_group_name = "$RESOURCE_GROUP"
}
EOF
done

# 3. Import resources
echo "3. Importing resources into Terraform state..."
terraform init
terraform plan -generate-config-out=generated.tf
terraform apply -auto-approve

# 4. Refactor and clean up
echo "4. Refactoring generated code..."
echo "✅ Migration complete! Review generated.tf and refactor as needed."
```

11.2 Phased Migration Strategy

Phase 1: Assessment

bash

```
# Inventory current state
az graph query -q "Resources | project name, type, location, resourceGroup" --output table

# Identify dependencies
az graph query -q "Resources | where type in~ ('Microsoft.Network/virtualNetworks', 'Microsoft.Network/networkInterfaces') | project name, type, dependencies" --output json
```

Phase 2: Pilot Project

hcl

```
# Start with non-critical resources
module "pilot_networking"{
  source = "./modules/networking"

  # Migrate networking first (least risky)
}

# Import existing VNet
import {
  to = module.pilot_networking.azurerm_virtual_network.main
  id = "/subscriptions/.../resourceGroups/existing-rg/providers/Microsoft.Network/virtualNetworks/existing-vnet"
}
```

Phase 3: Gradual Migration

text

```
Week 1-2: Networking (VNet, Subnets, NSGs)
Week 3-4: Storage (Storage Accounts, Disks)
Week 5-6: Compute (VMs, Scale Sets)
Week 7-8: Database (SQL, PostgreSQL)
Week 9-10: Monitoring & Security
```

12. Best Practices Checklist

12.1 Security Best Practices

- **Never store secrets in code** - Use Azure Key Vault
- **Enable audit logging** - Log all Terraform operations
- **Use remote state with locking** - Prevent concurrent modifications
- **Implement least privilege** - Service Principal with minimal permissions
- **Regularly rotate credentials** - Automate credential rotation
- **Scan for vulnerabilities** - Use tfsec, checkov regularly

- **Enable Azure Policy** - Enforce compliance

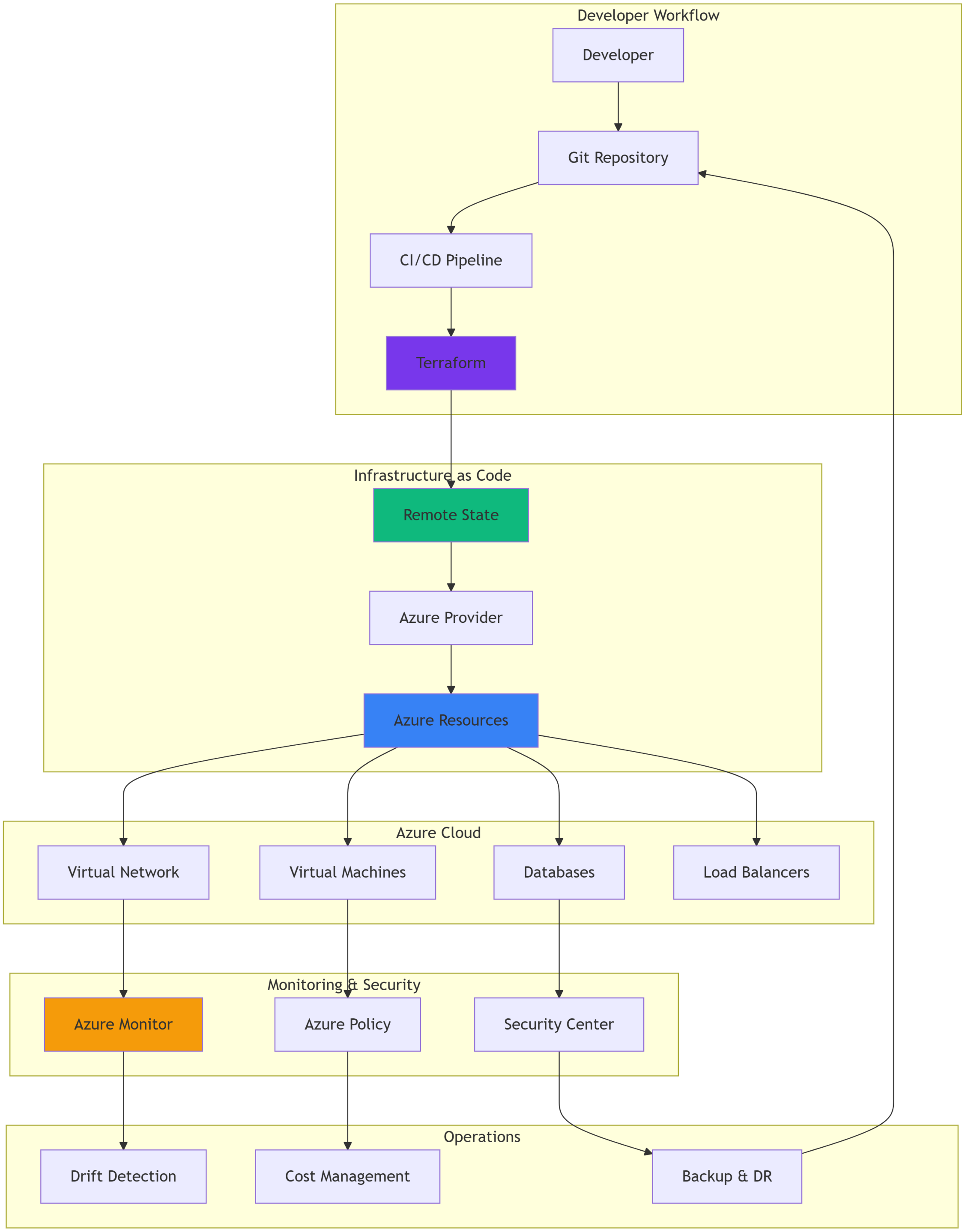
12.2 Operational Excellence

- **Version control everything** - All code in Git
- **Modular design** - Reusable, composable modules
- **Comprehensive testing** - Unit, integration, compliance tests
- **Documentation as code** - READMEs, ADRs, diagrams
- **CI/CD pipeline** - Automated testing and deployment
- **Monitoring and alerting** - Drift detection, cost alerts
- **Regular updates** - Keep Terraform and providers updated

12.3 Cost Optimization

- **Use terraform plan** - Preview costs before deployment
- **Implement tagging** - Track costs by project/environment
- **Set budgets and alerts** - Prevent cost overruns
- **Clean up unused resources** - Regular cleanup scripts
- **Use appropriate SKUs** - Right-size resources
- **Implement auto-shutdown** - For non-production environments
- **Leverage reservations** - For predictable workloads

Final Architecture Diagram



Key Takeaways

1. **IaC is NOT optional** - It's essential for modern cloud operations

2. **Start small, think big** - Begin with a single resource, plan for enterprise
3. **Automate everything** - From testing to deployment to cleanup
4. **Security first** - Never compromise on security practices
5. **Collaborate effectively** - Use Git workflows, code reviews, documentation
6. **Plan for disaster** - Have backup, recovery, and migration strategies
7. **Monitor and optimize** - Continuous improvement is key