Terraform Deep Dive

Status	Not Started

Complete Mastery Guide with Hands-on Examples

Document Version: 8.0

Date: January 2026

Audience: All Levels - From Beginners to Advanced Users

Executive Summary: The Construction Site Analogy

Think of Terraform as a Construction Project Manager:

Terraform Concept	Construction Analogy	Purpose
Provider	Construction Company	Knows how to build specific things
Resource	Building Material	Actual things being built
Data Source	Site Survey	Information about existing things
State File	Blueprint + Progress Tracker	What's built and its current state
Variables	Project Specifications	Customizable project details
Outputs	Project Handover Document	Important information after build
Graph	Construction Schedule	Order of operations
Backend	Secure Document Storage	Where blueprints are safely stored

1. Terraform Core Concepts Explained

1.1 What is Terraform?

Official Definition: Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

Simple Explanation: Terraform is like a **universal remote control** for your cloud infrastructure. You write what you want (code), and Terraform makes it happen across AWS, Azure, Google Cloud, etc.

1.2 The Terraform Workflow

text

```
Three Command Workflow:
1. terraform init    → Downloads providers, sets up backend
2. terraform plan    → Shows what will be created/changed
3. terraform apply   → Actually creates/modifies resources
4. terraform destroy → Cleans up everything
```

2. Providers: The Construction Companies

2.1 What are Providers?

Definition: Providers are plugins that Terraform uses to interact with cloud providers, SaaS providers, and other APIs. Each provider adds a set of resource types and data sources that Terraform can manage.

Simple Analogy: Each cloud provider (Azure, AWS, Google) is like a different construction company that speaks its own language. Terraform providers are the translators.

hcl

```
# main.tf - Provider Configuration
terraform {
  required_version = ">= 1.0.0"

  required_providers {
    # Azure Provider
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }

    # AWS Provider
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }

    # Random Provider (for generating random values)
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

# Configure Azure Provider
provider "azurerm"{
  features {
    resource_group {
      prevent_deletion_if_contains_resources = false
    }
  }

  # Authentication - multiple methods
  # Option 1: Environment Variables
  # export ARM_CLIENT_ID="..."
  # export ARM_CLIENT_SECRET="..."
  # export ARM_SUBSCRIPTION_ID="..."
  # export ARM_TENANT_ID="..."

  # Option 2: Azure CLI Authentication (auto-detected)
  # Just run: az login
  # Option 3: Service Principal in code (NOT RECOMMENDED for production)
  # client_id       = var.client_id
  # client_secret   = var.client_secret
  # subscription_id = var.subscription_id
  # tenant_id       = var.tenant_id
}

# Configure AWS Provider
provider "aws"{
  region = "us-east-1"

  # Authentication options similar to Azure
}
```

2.2 Provider Aliases (Multiple Instances)

hcl

```
# Deploying to multiple regions
provider "azurerm"{
  features {}
  alias = "eastus"

  subscription_id = var.subscription_id
  tenant_id       = var.tenant_id
  client_id       = var.client_id
  client_secret   = var.client_secret
}

provider "azurerm"{
  features {}
  alias = "westus"

  subscription_id = var.subscription_id
  tenant_id       = var.tenant_id
  client_id       = var.client_id
  client_secret   = var.client_secret
}

# Use aliased providers
resource "azurerm_resource_group" "east" {
  provider = azurerm.eastus

  name     = "rg-eastus-resources"
  location = "eastus"
}

resource "azurerm_resource_group" "west" {
  provider = azurerm.westus

  name     = "rg-westus-resources"
  location = "westus"
}
```

2.3 Provider Initialization in Action

bash

```
#!/bin/bash
# terraform-init-demo.sh
echo "=== TERRAFORM INIT DEMONSTRATION ==="

# Create a basic Terraform configuration
cat > main.tf << 'EOF'
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
EOF

echo "1. Before terraform init:"
ls -la .terraform* 2>/dev/null || echo "No .terraform directory"

echo ""
echo "2. Running terraform init:"
terraform init

echo ""
echo "3. After terraform init:"
ls -la .terraform/
echo ""
echo "Contents of .terraform directory:"
find .terraform -type f | sort

echo ""
echo "4. Check provider installation:"
terraform version
```

Expected Output:

text

```
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "~> 3.0"...
- Finding hashicorp/random versions matching "~> 3.0"...
- Installing hashicorp/azurerm v3.85.0...
- Installed hashicorp/azurerm v3.85.0 (signed by HashiCorp)
- Installing hashicorp/random v3.5.1...
- Installed hashicorp/random v3.5.1 (signed by HashiCorp)
```

3. Resources: The Building Blocks

3.1 What are Resources?

Definition: Resources are the most important element in the Terraform language. Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components.

Simple Analogy: Resources are like **building materials** - bricks, windows, doors. Each resource block tells Terraform to create and manage that specific piece of infrastructure.

hcl

```
# Basic Resource Syntax
resource "resource_type" "resource_name" {
  # Required arguments
  parameter1 = value1
  parameter2 = value2

  # Optional arguments
  optional_param = optional_value

  # Meta-arguments (available for all resources)
  count      = 2           # Create multiple instances
  for_each = var.items  # Create based on map/set
  depends_on = [other_resource]
  lifecycle {
    create_before_destroy = true
    prevent_destroy       = false
    ignore_changes        = [tags]
  }
}
```

3.2 Resource Creation Examples

hcl

```
# Single Resource
resource "azurerm_resource_group" "main" {
  name     = "rg-main-resources"
  location = "eastus"

  tags = {
    Environment = "Production"
    ManagedBy   = "Terraform"
  }
}

# Resource with Count (Multiple Instances)
resource "azurerm_virtual_network" "main" {
  count = 3  # Creates 3 VNets

  name                = "vnet-${count.index}"
  address_space       = ["10.${count.index}.0.0/16"]
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  # Each gets unique name and address space
}

# Resource with For Each (Dynamic Instances)
variable "environments"{
  type = map(object({
    location = string
    cidr     = string
  }))
  default = {
    dev = {
      location = "eastus"
      cidr     = "10.1.0.0/16"
    }
    prod = {
      location = "westus"
      cidr     = "10.2.0.0/16"
    }
  }
}

resource "azurerm_virtual_network" "env" {
  for_each = var.environments

  name                = "vnet-${each.key}"
  address_space       = [each.value.cidr]
  location            = each.value.location
  resource_group_name = azurerm_resource_group.main.name
}

# Complex Resource with Nested Blocks
resource "azurerm_virtual_machine" "web" {
  name                  = "vm-web-server"
  location              = azurerm_resource_group.main.location
  resource_group_name   = azurerm_resource_group.main.name
  vm_size               = "Standard_B2s"
  network_interface_ids = [azurerm_network_interface.main.id]

  # Nested block: Storage
  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }

  # Nested block: OS Disk
  storage_os_disk {
    name              = "osdisk-web"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

  # Nested block: OS Profile
  os_profile {
    computer_name  = "webserver"
    admin_username = "adminuser"
    admin_password = "P@ssw0rd123!"  # In production, use Key Vault!
  }

  # Nested block: OS Profile Linux Config
  os_profile_linux_config {
    disable_password_authentication = false
  }
}
```

3.3 Resource Meta-Arguments Deep Dive

hcl

```
# 1. COUNT - Create multiple similar resources
resource "azurerm_public_ip" "web" {
  count = 3  # Creates 3 public IPs

  name                = "pip-web-${count.index}"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  allocation_method   = "Static"
  sku                 = "Standard"
```

```
}

# Access them: azurerm_public_ip.web[0], azurerm_public_ip.web[1], etc.

# 2. FOR_EACH - Create resources from a map or set
variable "subnets"{
  type = map(object({
    address_prefix = string
  }))
  default = {
    web = {
      address_prefix = "10.0.1.0/24"
    }
    app = {
      address_prefix = "10.0.2.0/24"
    }
    db = {
      address_prefix = "10.0.3.0/24"
    }
  }
}

resource "azurerm_subnet" "main" {
  for_each = var.subnets

  name                 = "snet-${each.key}"
  resource_group_name  = azurerm_resource_group.main.name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [each.value.address_prefix]
}

# Access them: azurerm_subnet.main["web"], azurerm_subnet.main["app"], e
tc.

# 3. DEPENDS_ON - Explicit dependency
resource "azurerm_network_interface" "web" {
  name                = "nic-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.main["web"].id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.web[0].id
  }

  # Explicitly depends on subnet and public IP
  depends_on = [
    azurerm_subnet.main["web"],
    azurerm_public_ip.web[0]
  ]
}

# 4. LIFECYCLE - Control resource lifecycle
resource "azurerm_storage_account" "data" {
  name                     = "stdata${random_string.suffix.result}"
  resource_group_name      = azurerm_resource_group.main.name
  location                 = azurerm_resource_group.main.location
  account_tier             = "Standard"
  account_replication_type = "LRS"

  lifecycle {
    # 1. Create before destroy (minimize downtime)
    create_before_destroy = true

    # 2. Prevent accidental destruction
    prevent_destroy = false  # Set to true in production!

    # 3. Ignore changes to tags (won't trigger recreation)
    ignore_changes = [
      tags,
      account_replication_type
    ]

    # 4. Replace when certain attributes change
    replace_triggered_by = [
      # Recreate if resource group location changes
      azurerm_resource_group.main.location
    ]
  }
}

resource "random_string" "suffix" {
  length  = 8
  special = false
  upper   = false
}
```

4. Data Sources: The Information Gatherers

4.1 What are Data Sources?

Definition: Data sources allow Terraform to use information defined outside of Terraform, or defined by another separate Terraform configuration. They are read-only views into pre-existing data.

Simple Analogy: Data sources are like **looking up information** in a reference book before building. They don't create anything, they just fetch information.

hcl

```
# Data Source Syntax
data "data_source_type" "reference_name" {
  # Arguments to query/filter
  filter = "criteria"

  # Optional constraints
}

# Use the data
resource "azurerm_resource" "example" {
  name = data.data_source_type.reference_name.attribute
}
```

4.2 Common Data Source Examples

hcl

```
# 1. Get information about existing resource group
data "azurerm_resource_group" "existing" {
  name = "existing-resource-group"
}

# 2. Get current Azure client config (subscription, tenant, etc.)
data "azurerm_client_config" "current" {}

# 3. Get existing virtual network
data "azurerm_virtual_network" "hub" {
  name                = "hub-vnet"
  resource_group_name = "hub-network-rg"
}

# 4. Get latest Ubuntu image
data "azurerm_platform_image" "ubuntu" {
  location  = "eastus"
  publisher = "Canonical"
  offer     = "UbuntuServer"
  sku       = "18.04-LTS"
}

# 5. Get SSH key from file
data "azurerm_ssh_public_key" "admin" {
  name                = "admin-key"
  resource_group_name = "ssh-keys-rg"
}

# 6. Get Key Vault secret (for passwords/connection strings)
data "azurerm_key_vault_secret" "db_password" {
  name         = "database-password"
  key_vault_id = azurerm_key_vault.secrets.id
}

# 7. Get information about built-in Azure policies
data "azurerm_policy_definition" "allowed_locations" {
  display_name = "Allowed locations"
}

# 8. Get existing network security group
data "azurerm_network_security_group" "bastion" {
  name                = "nsg-bastion"
  resource_group_name = "security-rg"
}
```

4.3 Practical Data Source Usage

hcl

```
# main.tf - Complete example with data sources
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "azurerm"{
  features {}
}

# Data source: Get existing resource group
data "azurerm_resource_group" "hub" {
  name = "hub-resources"
}

# Data source: Get client configuration
data "azurerm_client_config" "current" {}

# Use data sources in resources
resource "azurerm_key_vault" "secrets" {
  name                = "kv-${data.azurerm_client_config.current.subscription_id}"
  location            = data.azurerm_resource_group.hub.location
  resource_group_name = data.azurerm_resource_group.hub.name
  tenant_id           = data.azurerm_client_config.current.tenant_id
  sku_name            = "standard"

  # Add current client as admin
  access_policy {
    tenant_id = data.azurerm_client_config.current.tenant_id
    object_id = data.azurerm_client_config.current.object_id

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

# Data source depends on resource being created first
data "azurerm_key_vault_secret" "example" {
  name         = "example-secret"
  key_vault_id = azurerm_key_vault.secrets.id

  depends_on = [azurerm_key_vault.secrets]
}
```

```
# Output data source information
output "hub_location"{
  value       = data.azurerm_resource_group.hub.location
  description = "Location of the hub resource group"
}

output "current_subscription_id"{
  value       = data.azurerm_client_config.current.subscription_id
  description = "Current Azure subscription ID"
  sensitive   = false  # Subscription ID is not typically sensitive
}
```

5. Variables: The Configuration Parameters

5.1 What are Variables?

Definition: Variables in Terraform are parameters for Terraform modules that allow users to customize behavior without modifying the source code. They are defined in `.tf` files and can be set via multiple methods.

Simple Analogy: Variables are like **adjustable settings** on a machine. Same machine (code), different settings (variables) for different environments.

5.2 Variable Types and Definitions

hcl

```
# variables.tf
# 1. String Variable
variable "environment"{
  type        = string
  description = "Environment name (dev, staging, prod)"
  default     = "dev"

  validation {
    condition     = can(regex("^(dev|staging|prod)$", var.environment))
    error_message = "Environment must be dev, staging, or prod."
  }
}

# 2. Number Variable
variable "vm_count"{
  type        = number
  description = "Number of VMs to create"
  default     = 2

  validation {
    condition     = var.vm_count > 0 && var.vm_count <= 10
    error_message = "VM count must be between 1 and 10."
  }
}

# 3. Boolean Variable
variable "enable_monitoring"{
  type        = bool
  description = "Enable Azure Monitor"
  default     = true
}

# 4. List Variable
variable "locations"{
  type        = list(string)
  description = "Azure regions to deploy to"
  default     = ["eastus", "westus"]

  validation {
    condition     = length(var.locations) <= 3
    error_message = "Maximum 3 locations allowed."
  }
}

# 5. Map Variable
```

```
variable "tags"{
  type = map(string)
  description = "Resource tags"
  default = {
    Environment = "dev"
    ManagedBy   = "Terraform"
    CostCenter  = "IT"
  }
}

# 6. Object Variable (Complex)
variable "network_config"{
  type = object({
    vnet_address_space = list(string)
    subnet_prefixes    = map(string)
    enable_ddos        = bool
  })
  default = {
    vnet_address_space = ["10.0.0.0/16"]
    subnet_prefixes = {
      web = "10.0.1.0/24"
      app = "10.0.2.0/24"
      db  = "10.0.3.0/24"
    }
    enable_ddos = true
  }
}

# 7. Tuple Variable (Fixed-length list with specific types)
variable "vm_sizes"{
  type = tuple([string, string, string])
  default = ["Standard_B2s", "Standard_B4ms", "Standard_D4s_v3"]
}

# 8. Set Variable (Unique values only)
variable "admin_users"{
  type    = set(string)
  default = ["user1", "user2", "user3"]
}

# 9. Any Type (Dynamic)
variable "custom_config"{
  type        = any
  description = "Custom configuration object"
  default     = null
}

# 10. Sensitive Variable
variable "db_password"{
  type        = string
  description = "Database password"
  sensitive   = true  # Won't show in logs/outputs
  default     = ""
}
```

5.3 Setting Variable Values

bash

```
# Multiple ways to set variables:

# 1. terraform.tfvars file (automatically loaded)
# terraform.tfvars
environment = "prod"
vm_count    = 5
locations   = ["eastus", "westus2"]

# 2. *.auto.tfvars files (also auto-loaded)
# prod.auto.tfvars
environment = "prod"

# 3. Command line flags
```

```
terraform apply -var="environment=prod" -var="vm_count=3"

# 4. Environment variables (TF_VAR_ prefix)
export TF_VAR_environment="prod"
export TF_VAR_db_password="P@ssw0rd123!"
terraform apply

# 5. Variable definition files
terraform apply -var-file="prod.tfvars"

# 6. UI input (if not defined elsewhere)
# terraform will prompt for input
```

Example variable files:

hcl

```
# dev.tfvars
environment       = "dev"
vm_count          = 2
enable_monitoring = false
locations         = ["eastus"]

tags = {
  Environment = "Development"
  CostCenter  = "R&D"
}

# prod.tfvars
environment       = "prod"
vm_count          = 5
enable_monitoring = true
locations         = ["eastus", "westus2", "centralus"]

tags = {
  Environment = "Production"
  CostCenter  = "Operations"
  SLA         = "99.9%"
}
```

5.4 Variable Precedence

text

```
HIGHEST PRECEDENCE
1. -var command line flag
2. -var-file command line flag
3. *.auto.tfvars files (alphabetical order)
4. terraform.tfvars
5. Environment variables (TF_VAR_name)
6. Variable defaults in variables.tf
LOWEST PRECEDENCE
```

6. Locals: The Internal Variables

6.1 What are Locals?

Definition: Local values assign a name to an expression, allowing it to be used multiple times within a module without repeating it. They are like variables but only available within the module where they're defined.

Simple Analogy: Locals are like **temporary sticky notes** you use while working. They help organize calculations but aren't exposed outside your workspace.

hcl

```
# Syntax
locals {
  # Simple value
  environment_prefix = "env-${var.environment}"

  # Complex calculations
  vnet_name = "${local.environment_prefix}-vnet"
  # Conditional values
  vm_size = var.environment == "prod" ? "Standard_D4s_v3" : "Standard_B2s"

  # Transformations
  upper_tags = { for k, v in var.tags : upper(k) => upper(v) }

  # Merged maps
  default_tags = {
    CreatedBy   = "Terraform"
    CreatedDate = formatdate("YYYY-MM-DD", timestamp())
  }

  all_tags = merge(local.default_tags, var.tags)

  # List comprehensions
  subnet_names = [for name, config in var.subnet_configs : "snet-${name}"]
}

# Usage
resource "azurerm_virtual_network" "main" {
  name                = local.vnet_name
  address_space       = ["10.0.0.0/16"]
  location            = var.location
  resource_group_name = azurerm_resource_group.main.name

  tags = local.all_tags
}
```

6.2 Practical Locals Examples

hcl

```
# main.tf - Complete locals example
locals {
  # 1. Naming conventions
```

```
naming_prefix = "${var.project_name}-${var.environment}"

# 2. Resource names using naming convention
resource_names = {
  rg   = "${local.naming_prefix}-rg"
  vnet = "${local.naming_prefix}-vnet"
  kv   = "${local.naming_prefix}-kv"
  sa   = "${local.naming_prefix}sa"  # Storage account has special naming
}

# 3. Conditional configuration
vm_config = {
  count = var.environment == "prod" ? 3 : 1
  size  = var.environment == "prod" ? "Standard_D4s_v3" : "Standard_B2s"
  zones = var.environment == "prod" ? ["1", "2", "3"] : null
}

# 4. Network CIDR calculations
vnet_cidr = "10.${var.network_number}.0.0/16"

subnet_cidrs = {
  web = cidrsubnet(local.vnet_cidr, 8, 1)   # 10.x.1.0/24
  app = cidrsubnet(local.vnet_cidr, 8, 2)   # 10.x.2.0/24
  db  = cidrsubnet(local.vnet_cidr, 8, 3)   # 10.x.3.0/24
}

# 5. Tag merging
mandatory_tags = {
  Environment     = var.environment
  ManagedBy       = "Terraform"
  TerraformModule = basename(abspath(path.module))
}

all_tags = merge(local.mandatory_tags, var.custom_tags)

# 6. Data transformation
```

```
  vm_admin_users = toset(concat(
    var.default_admins,
    var.environment_admins[var.environment]
  ))

  # 7. Complex validation
  is_valid_environment = contains(["dev", "staging", "prod"], var.environment)

  # 8. File operations
  user_data = templatefile("${path.module}/templates/cloud-init.yaml", {
    hostname = "webserver"
    packages = ["nginx", "nodejs"]
  })
}

# Use locals throughout configuration
resource "azurerm_resource_group" "main" {
  name     = local.resource_names.rg
  location = var.location

  tags = local.all_tags
}

resource "azurerm_virtual_network" "main" {
  name                = local.resource_names.vnet
  address_space       = [local.vnet_cidr]
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  tags = local.all_tags
}

resource "azurerm_subnet" "web" {
  name                 = "snet-web"
  resource_group_name  = azurerm_resource_group.main.name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [local.subnet_cidrs.web]
}

# Conditional resources using locals
resource "azurerm_monitor_action_group" "alerts" {
  count = var.environment == "prod" ? 1 : 0

  name                = "${local.naming_prefix}-alerts"
  resource_group_name = azurerm_resource_group.main.name
  short_name          = "alerts"

  email_receiver {
    name          = "admin"
    email_address = var.admin_email
  }
}
```

7. Outputs: The Exposed Results

7.1 What are Outputs?

Definition: Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use. Outputs are the return values of a Terraform module.

Simple Analogy: Outputs are like **receipts or completion certificates**. They tell you what was created and important information about it.

hcl

```
# Output Syntax
output "output_name"{
  value       = expression
  description = "Human-readable description"
  sensitive   = false  # or true for sensitive data
  depends_on  = []     # rarely needed, but available
}
```

7.2 Output Examples

hcl

```
# outputs.tf - Complete output examples
# 1. Simple string output
output "resource_group_name"{
  value       = azurerm_resource_group.main.name
  description = "Name of the main resource group"
}

# 2. Complex object output
output "virtual_network"{
  value = {
    id   = azurerm_virtual_network.main.id
    name = azurerm_virtual_network.main.name
    cidr = azurerm_virtual_network.main.address_space
  }
  description = "Virtual network configuration"
}

# 3. Output with transformation
output "web_vm_public_ips"{
  value = {
    for vm in azurerm_linux_virtual_machine.web :
    vm.name => vm.public_ip_address
  }
  description = "Public IP addresses of web VMs"
}

# 4. Conditional output
output "monitoring_dashboard_url"{
  value = var.enable_monitoring ? azurerm_application_insights.main.connecti
```

```
on_string : null
  description = "Application Insights connection string if monitoring is enable
d"
}

# 5. Sensitive output (won't show in CLI)
output "database_connection_string"{
  value       = "Server=${azurerm_mssql_server.main.fully_qualified_domain_na
me};Database=${azurerm_mssql_database.main.name};User Id=${var.db_use
rname};Password=${var.db_password}"
  description = "SQL Server connection string"
  sensitive   = true  # Won't be displayed in terraform output
}

# 6. Computed output
output "web_load_balancer_url"{
  value       = "http://${azurerm_public_ip.web_lb.ip_address}"
  description = "URL to access the web load balancer"
}

# 7. Output from count resources
output "all_vm_names"{
  value = azurerm_linux_virtual_machine.web[*].name
  description = "Names of all web VMs"
}

# 8. Output from for_each resources
output "subnet_ids"{
  value = {
    for k, v in azurerm_subnet.main : k ⇒ v.id
  }
  description = "Map of subnet names to their IDs"
}

# 9. Data source output
output "current_subscription_id"{
  value       = data.azurerm_client_config.current.subscription_id
  description = "Current Azure subscription ID"
}

# 10. Local value output
output "full_naming_convention"{
  value       = local.naming_prefix
  description = "Full naming convention used in deployment"
}

# 11. Formatted output
output "deployment_summary"{
  value = <<EOT
Deployment Summary:
===================
Environment: ${var.environment}
Location: ${var.location}
Resource Group: ${azurerm_resource_group.main.name}
Virtual Network: ${azurerm_virtual_network.main.name}
Web VMs: ${join(", ", azurerm_linux_virtual_machine.web[*].name)}
Load Balancer IP: ${azurerm_public_ip.web_lb.ip_address}
EOT
  description = "Human-readable deployment summary"
}
```

7.3 Using Outputs

bash

```
# View all outputs
terraform output

# View specific output
terraform output resource_group_name

# View output as JSON
```

```
terraform output -json

# Use outputs in scripts
RG_NAME=$(terraform output -raw resource_group_name)
echo "Resource Group: $RG_NAME"

# Sensitive outputs won't show by default
terraform output database_connection_string
# Output: <sensitive>

# But you can force display (be careful!)
terraform output -json database_connection_string | jq -r '.database_connection_string.value'
```

---

8. Terraform Graph: The Dependency Visualizer

8.1 What is Terraform Graph?

Definition: Terraform builds a dependency graph from your configurations, and uses that graph to determine the correct order of operations. The `terraform graph` command produces a visual representation of the dependency graph.

Simple Analogy: Terraform graph is like a **project Gantt chart** showing what needs to be built first, what depends on what, and the critical path.

8.2 Understanding Implicit vs Explicit Dependencies

hcl

```
# main.tf - Demonstrating dependencies
# IMPLICIT DEPENDENCIES (Recommended)
# Terraform automatically detects dependencies through references

resource "azurerm_resource_group" "main" {
  name     = "rg-main"
  location = "eastus"
}
```

```
# Implicit dependency: VNet references resource group
resource "azurerm_virtual_network" "main" {
  name                = "vnet-main"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.main.location          # ← Implicit
  resource_group_name = azurerm_resource_group.main.name              # ← Implicit
  # Terraform knows VNet depends on RG
}

resource "azurerm_subnet" "web" {
  name                 = "snet-web"
  resource_group_name  = azurerm_resource_group.main.name             # ← Implicit
  virtual_network_name = azurerm_virtual_network.main.name            # ← Implicit
  address_prefixes     = ["10.0.1.0/24"]
  # Subnet depends on both RG and VNet
}

# EXPLICIT DEPENDENCIES (Use when needed)
# Sometimes dependencies aren't visible in code

resource "azurerm_network_interface" "web" {
  name                = "nic-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.web.id
    private_ip_address_allocation = "Dynamic"
  }

  # Explicit dependency: Even though we reference subnet.id,
  # we might need to ensure NSG is created first
  depends_on = [
    azurerm_network_security_group.web
  ]
}

resource "azurerm_network_security_group" "web" {
  name                = "nsg-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  # This doesn't reference the NIC, so no implicit dependency
  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 = "*"
  }
}

# Circular Dependency Example (PROBLEM!)
# resource "azurerm_virtual_machine" "web" {
#   name                  = "vm-web"
#   location              = azurerm_resource_group.main.location
#   resource_group_name   = azurerm_resource_group.main.name
#   network_interface_ids = [azurerm_network_interface.web.id]
#   vm_size               = "Standard_B2s"
#
#   storage_image_reference {
#     publisher = "Canonical"
#     offer     = "UbuntuServer"
#     sku       = "18.04-LTS"
#     version   = "latest"
#   }
#
#   storage_os_disk {
#     name              = "osdisk-web"
#     caching           = "ReadWrite"
#     create_option     = "FromImage"
#     managed_disk_type = "Standard_LRS"
#   }
# }
#
# resource "azurerm_network_interface" "web" {
#   name                = "nic-web"
#   location            = azurerm_resource_group.main.location
#   resource_group_name = azurerm_resource_group.main.name
#
#   ip_configuration {
#     name                          = "internal"
#     subnet_id                     = azurerm_subnet.web.id
#     private_ip_address_allocation = "Dynamic"
#     public_ip_address_id          = azurerm_public_ip.web.id
#   }
#
#   # Circular! VM depends on NIC, but NIC wants to attach public IP from VM
#   # This will fail
#   depends_on = [azurerm_virtual_machine.web]
# }
```

8.3 Generating and Understanding Graphs

bash

```
#!/bin/bash
# graph-demo.sh
echo "=== TERRAFORM GRAPH DEMONSTRATION ==="
```

```
# Create a complex configuration
cat > main.tf << 'EOF'
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {
  features {}
}

resource "random_pet" "name" {
  length = 2
}

resource "azurerm_resource_group" "main" {
  name     = "rg-${random_pet.name.id}"
  location = "eastus"
}

resource "azurerm_virtual_network" "main" {
  name                = "vnet-${random_pet.name.id}"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
}
```

```
resource "azurerm_subnet" "web" {
  name                 = "snet-web"
  resource_group_name  = azurerm_resource_group.main.name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = ["10.0.1.0/24"]
}

resource "azurerm_network_security_group" "web" {
  name                = "nsg-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
}

resource "azurerm_network_interface" "web" {
  name                = "nic-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.web.id
    private_ip_address_allocation = "Dynamic"
  }

  depends_on = [azurerm_network_security_group.web]
}

resource "azurerm_linux_virtual_machine" "web" {
  name                = "vm-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  size                = "Standard_B2s"

  admin_username = "adminuser"
  admin_ssh_key {
    username   = "adminuser"
    public_key = file("~/.ssh/id_rsa.pub")
  }

  network_interface_ids = [azurerm_network_interface.web.id]

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

  source_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }
}
EOF

echo "1. Initialize Terraform..."
terraform init

echo ""
echo "2. Generate graph in DOT format..."
terraform graph > graph.dot

echo "3. Generate graph in PNG format (requires Graphviz)..."
if command -v dot &> /dev/null; then
  terraform graph | dot -Tpng > graph.png
  echo "Graph saved as graph.png"
  echo "Open graph.png to see the dependency graph"
else
  echo "Graphviz not installed. Install with:"
  echo "  Ubuntu: sudo apt-get install graphviz"
  echo "  macOS: brew install graphviz"
  echo "  Windows: choco install graphviz"
```

```
fi

echo ""
echo "4. Show simplified graph (just important dependencies)..."
terraform graph -type=plan | head -20

echo ""
echo "5. Analyze the graph structure:"
echo "   - Each box is a resource"
echo "   - Arrows show dependencies"
echo "   - Dashed lines: data sources"
echo "   - Solid lines: resources"
echo "   - Arrow direction: depends on"
```

8.4 Graph Output Interpretation

text

```
Typical Graph Structure:

random_pet.name
      │
      ▼
azurerm_resource_group.main
      │
      ▼
azurerm_virtual_network.main
      │
      ▼
azurerm_subnet.web ──────────┐
      │            │
      ▼            ▼
azurerm_network_security_group.web
      │            │
      └──────────────────────┘

           │
```

```
         ▼
azurerm_network_interface.web
         |
         ▼
azurerm_linux_virtual_machine.web

Key Insights:
1. Resource Group depends on random_pet (for unique name)
2. VNet depends on Resource Group
3. Subnet depends on VNet
4. NIC depends on both Subnet and NSG (explicit depends_on)
5. VM depends on NIC
```

9. State Management: The Single Source of Truth

9.1 What is Terraform State?

Definition: Terraform state is a **JSON file** that maps real-world resources to your configuration, keeps track of metadata, and improves performance for large infrastructures. It's stored in a `terraform.tfstate` file.

Simple Analogy: Terraform state is like a **building registry** that records:

- What buildings exist
- Who owns them
- What they look like
- When they were built

9.2 Why State is Critical

json

```
// Example terraform.tfstate (simplified)
{
  "version": 4,
  "terraform_version": "1.5.0",
  "serial": 1,
  "lineage": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "outputs": {
    "resource_group_name": {
      "value": "rg-myapp-prod",
      "type": "string"
    }
  },
  "resources": [
    {
      "mode": "managed",
      "type": "azurerm_resource_group",
      "name": "main",
      "provider": "provider[\"registry.terraform.io/hashicorp/azurerm\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "id": "/subscriptions/.../resourceGroups/rg-myapp-prod",
            "location": "eastus",
            "name": "rg-myapp-prod",
            "tags": {
              "Environment": "prod"
            }
          },
          "sensitive_attributes": [],
          "private": "..."
        }
      ]
    }
  ]
}
```

What State Tracks:

1. **Resource Mapping:** Links resources in code to real cloud resources

2. **Metadata:** Resource dependencies and relationships
3. **Performance:** Cache of resource attributes
4. **Synchronization:** Team collaboration enabler

9.3 Local State vs Remote State

bash

```bash
# LOCAL STATE (Default - for learning only)
# .terraform/terraform.tfstate
terraform {
  # No backend configuration = local state
}

# Problems with local state:
# 1. Not shared with team
# 2. Lost if machine dies
# 3. No locking (two people can run terraform apply)
# 4. No version history

# REMOTE STATE (Production requirement)
terraform {
  backend "azurerm" {
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstate12345"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"

    # Optional: Use SAS token or Managed Identity
    # Use environment variables for secrets:
    # export ARM_SAS_TOKEN="..."
    # Or use Azure AD authentication
  }
}
```

9.4 Setting Up Remote State Backend

bash

```bash
#!/bin/bash
# setup-remote-state.sh
echo "=== SETTING UP AZURE REMOTE STATE BACKEND ==="

# Configuration
RESOURCE_GROUP="terraform-state-rg"
STORAGE_ACCOUNT="tfstate$(openssl rand -hex 5)"
CONTAINER="tfstate"
LOCATION="eastus"

echo "1. Creating Resource Group for state..."
az group create \
  --name $RESOURCE_GROUP \
  --location $LOCATION

echo "2. Creating Storage Account..."
az storage account create \
  --name $STORAGE_ACCOUNT \
  --resource-group $RESOURCE_GROUP \
  --location $LOCATION \
  --sku Standard_LRS \
  --encryption-services blob \
  --allow-blob-public-access false \
  --min-tls-version TLS1_2

echo "3. Creating Blob Container..."
az storage container create \
  --name $CONTAINER \
  --account-name $STORAGE_ACCOUNT \
  --auth-mode login

echo "4. Enabling Soft Delete (recommended)..."
```

```
az storage account blob-service-properties update \
  --account-name $STORAGE_ACCOUNT \
  --enable-delete-retention true \
  --delete-retention-days 30

echo "5. Enabling Versioning (recommended)..."
az storage account blob-service-properties update \
  --account-name $STORAGE_ACCOUNT \
  --enable-versioning true

echo "6. Generating backend configuration..."
cat > backend.tf << EOF
terraform {
  backend "azurerm" {
    resource_group_name  = "$RESOURCE_GROUP"
    storage_account_name = "$STORAGE_ACCOUNT"
    container_name       = "$CONTAINER"
    key                  = "terraform.tfstate"
  }
}
EOF

echo ""
echo "✅ Remote state backend configured!"
echo ""
echo "Storage Account: $STORAGE_ACCOUNT"
echo "Container: $CONTAINER"
echo "Key: terraform.tfstate"
echo ""
echo "Next steps:"
echo "1. Run: terraform init -reconfigure"
echo "2. Terraform will migrate state to Azure Storage"
```

9.5 State Locking

Why Locking is Essential:

text

```text
Scenario without locking:
10:00 AM: Alice runs terraform apply
10:01 AM: Bob runs terraform apply
10:02 AM: ❌ State corruption! Both modifying same resources

Scenario with locking:
10:00 AM: Alice runs terraform apply
10:01 AM: Bob tries terraform apply
10:02 AM: ✅ "Error: State is locked by Alice" - Bob waits
10:05 AM: Alice finishes, lock released
10:06 AM: Bob can now run terraform apply
```

Backends that Support Locking:

- Azure Blob Storage (with blob lease)
- AWS S3 + DynamoDB
- Google Cloud Storage
- HashiCorp Consul
- Terraform Cloud/Enterprise

Lock File Details:

bash

```bash
# When terraform apply runs:
# 1. Creates lock file: terraform.tfstate.lock.info
# 2. Contains: Who, When, Why, Operation ID
# 3. Released when operation completes
# 4. If crash occurs, manual intervention needed

# Force unlock (be careful!)
terraform force-unlock LOCK_ID
```

9.6 State Operations

bash

```bash
# 1. View state
terraform state list  # List all resources in state
terraform state show azurerm_resource_group.main  # Show specific resource

# 2. Modify state
terraform state mv azurerm_resource_group.old azurerm_resource_group.new  # Rename in state
terraform state rm azurerm_resource_group.old  # Remove from state (doesn't delete resource!)

# 3. Import existing resources
# Step 1: Add resource to configuration
# resource "azurerm_resource_group" "existing" {
#   name     = "existing-rg"
#   location = "eastus"
# }

# Step 2: Import
terraform import azurerm_resource_group.existing /subscriptions/.../resourceGroups/existing-rg

# 4. Refresh state (sync with real world)
terraform refresh

# 5. State pull/push (for advanced scenarios)
terraform state pull > state.json  # Download state
terraform state push state.json    # Upload state (dangerous!)

# 6. Backup state (always do this before operations!)
cp terraform.tfstate terraform.tfstate.backup
```

```
# 7. State inspection
terraform state list -id=*vnet*  # Filter resources
terraform state list | grep "azurerm_virtual_network"

# 8. Troubleshooting state
terraform state replace-provider hashicorp/azurerm hashicorp/azurerm2  # Provider migration
```

---

10. Complete Hands-on Example

10.1 Project Structure

text

```
terraform-mastery/
├── main.tf              # Primary configuration
├── variables.tf         # Input variables
├── outputs.tf           # Output values
├── terraform.tfvars     # Variable values
├── backend.tf           # Remote state configuration
├── providers.tf         # Provider configuration
├── locals.tf            # Local values
├── data-sources.tf      # Data sources
├── modules/             # Reusable modules
│   └── network/
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
└── scripts/
    ├── setup-backend.sh
    └── deploy.sh
```

10.2 Complete Configuration Example

hcl

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

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.85.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.5.0"
    }
    time = {
      source  = "hashicorp/time"
      version = "~> 0.9.0"
    }
  }

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

provider "azurerm"{
  features {
    resource_group {
      prevent_deletion_if_contains_resources = true
    }
    key_vault {
      purge_soft_delete_on_destroy = true
    }
  }

  # Authentication via environment variables or Azure CLI
}

provider "random"{}

provider "time"{}
```

hcl

```
# variables.tf
variable "environment"{
  type        = string
  description = "Deployment environment (dev, staging, prod)"
  default     = "dev"

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

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

variable "vm_count"{
  type        = number
  description = "Number of web VMs to deploy"
  default     = 2

  validation {
    condition     = var.vm_count >= 1 && var.vm_count <= 5
    error_message = "VM count must be between 1 and 5."
  }
}

variable "enable_monitoring"{
  type        = bool
  description = "Enable Azure Monitor resources"
  default     = true
}

variable "tags"{
  type        = map(string)
  description = "Tags to apply to all resources"
  default = {
    ManagedBy   = "Terraform"
    Environment = "dev"
  }
}
```

hcl

```
# locals.tf
locals {
  # Naming conventions
  naming_prefix = "app-${var.environment}"

  # Resource names
  resource_names = {
    rg   = "${local.naming_prefix}-rg"
    vnet = "${local.naming_prefix}-vnet"
    kv   = "${local.naming_prefix}-kv"
  }

  # Environment-specific configuration
  config = {
    dev = {
      vm_size      = "Standard_B2s"
      auto_shutdown = true
      backup_retention = 7
    }
    staging = {
      vm_size      = "Standard_B4ms"
      auto_shutdown = true
      backup_retention = 14
    }
    prod = {
      vm_size      = "Standard_D4s_v3"
      auto_shutdown = false
      backup_retention = 30
    }
  }

  current_config = local.config[var.environment]

  # CIDR calculations
  vnet_cidr = "10.${index(["dev", "staging", "prod"], var.environment) + 1}.0.0/1
6"

  subnet_cidrs = {
    web = cidrsubnet(local.vnet_cidr, 8, 1)
    app = cidrsubnet(local.vnet_cidr, 8, 2)
    db  = cidrsubnet(local.vnet_cidr, 8, 3)
  }

  # Tags
  mandatory_tags = {
    Environment     = var.environment
    DeploymentDate  = time_static.deployment_date.rfc3339
    TerraformModule = "complete-example"
  }
  all_tags = merge(local.mandatory_tags, var.tags)
}

# Static timestamp for consistent tagging
resource "time_static" "deployment_date" {}
```

hcl

```
# data-sources.tf
# Get current Azure client configuration
data "azurerm_client_config" "current" {}

# Get existing resource group for shared services
data "azurerm_resource_group" "shared" {
  name = "shared-services-rg"
}

# Get latest Ubuntu image
data "azurerm_platform_image" "ubuntu" {
  location  = var.location
  publisher = "Canonical"
  offer     = "UbuntuServer"
  sku       = "18.04-LTS"
}
```

hcl

```
# main.tf
# Create resource group
resource "azurerm_resource_group" "main" {
  name     = local.resource_names.rg
  location = var.location

  tags = local.all_tags
}
```

```
# Create virtual network
resource "azurerm_virtual_network" "main" {
  name                = local.resource_names.vnet
  address_space       = [local.vnet_cidr]
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  tags = local.all_tags
}

# Create subnets using for_each
resource "azurerm_subnet" "main" {
  for_each = local.subnet_cidrs

  name                 = "snet-${each.key}"
  resource_group_name  = azurerm_resource_group.main.name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = [each.value]
}

# Create network security group
resource "azurerm_network_security_group" "web" {
  name                = "nsg-web"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name

  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 = "*"
  }

  security_rule {
    name                       = "AllowSSH"
    priority                   = 110
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = data.azurerm_client_config.current.object_id =
= "" ? "*" : "VirtualNetwork"
    destination_address_prefix = "*"
  }

  tags = local.all_tags
}

# Create public IP for load balancer
resource "azurerm_public_ip" "web_lb" {
  name                = "pip-web-lb"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  allocation_method   = "Static"
  sku                 = "Standard"

  tags = local.all_tags
}

# Create network interfaces for web VMs
resource "azurerm_network_interface" "web" {
  count = var.vm_count

  name                = "nic-web-${count.index}"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.main["web"].id
    private_ip_address_allocation = "Dynamic"
  }

  tags = local.all_tags
}

# Associate NSG with NIC
resource "azurerm_network_interface_security_group_association" "web" {
  count = var.vm_count

  network_interface_id      = azurerm_network_interface.web[count.index].id
  network_security_group_id = azurerm_network_security_group.web.id
}

# Create web VMs
resource "azurerm_linux_virtual_machine" "web" {
  count = var.vm_count

  name                = "vm-web-${count.index}"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  size                = local.current_config.vm_size
  admin_username      = "adminuser"

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

  admin_ssh_key {
    username   = "adminuser"
    public_key = file("~/.ssh/id_rsa.pub")
  }

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

  source_image_reference {
    publisher = data.azurerm_platform_image.ubuntu.publisher
    offer     = data.azurerm_platform_image.ubuntu.offer
    sku       = data.azurerm_platform_image.ubuntu.sku
    version   = "latest"
  }

  tags = local.all_tags

  # Auto-shutdown for non-production
  lifecycle {
    ignore_changes = [
      tags["DeploymentDate"]
    ]
  }
}

# Conditional monitoring resources
resource "azurerm_log_analytics_workspace" "monitoring" {
  count = var.enable_monitoring ? 1 : 0

  name                = "log-${local.naming_prefix}"
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  sku                 = "PerGB2018"
  retention_in_days   = 30

  tags = local.all_tags
}

resource "azurerm_monitor_diagnostic_setting" "vnet" {
  count = var.enable_monitoring ? 1 : 0
  name                       = "diagnostics-vnet"
  target_resource_id         = azurerm_virtual_network.main.id
  log_analytics_workspace_id = azurerm_log_analytics_workspace.monitoring
[0].id

  enabled_log {
    category = "VMProtectionAlerts"
  }

  metric {
    category = "AllMetrics"
    enabled  = true
  }
}
```

hcl

```
# outputs.tf
output "resource_group_name"{
  value       = azurerm_resource_group.main.name
  description = "Name of the main resource group"
}

output "virtual_network_name"{
  value       = azurerm_virtual_network.main.name
  description = "Name of the virtual network"
}

output "web_vm_private_ips"{
  value = {
    for idx, vm in azurerm_linux_virtual_machine.web :
    vm.name => vm.private_ip_address
  }
  description = "Private IP addresses of web VMs"
}
```

```
output "load_balancer_public_ip"{
  value       = azurerm_public_ip.web_lb.ip_address
  description = "Public IP address of the load balancer"
}

output "subnet_ids"{
  value = {
    for name, subnet in azurerm_subnet.main :
    name ⇒ subnet.id
  }
  description = "Map of subnet names to their IDs"
}

output "monitoring_workspace_id"{
  value       = var.enable_monitoring ? azurerm_log_analytics_workspace.monit
oring[0].id : null
  description = "Log Analytics Workspace ID if monitoring is enabled"
}

output "deployment_summary"{
  value = <<EOT
========================================
Deployment Complete!
========================================
Environment: ${var.environment}
Location: ${var.location}
Resource Group: ${azurerm_resource_group.main.name}
Virtual Network: ${azurerm_virtual_network.main.name}
Web VMs Deployed: ${var.vm_count}
Load Balancer IP: ${azurerm_public_ip.web_lb.ip_address}
Monitoring Enabled: ${var.enable_monitoring}
========================================
EOT
```

```
  description = "Human-readable deployment summary"
}
```

10.3 Deployment Script

bash

```
#!/bin/bash
# deploy.sh
set -e  # Exit on error

echo "=== TERRAFORM COMPLETE DEPLOYMENT ==="

# Environment selection
ENV=${1:-dev}
CONFIG_FILE="${ENV}.tfvars"

if [ ! -f "$CONFIG_FILE" ]; then
  echo "❌ Config file $CONFIG_FILE not found"
  echo "Available configs:"
  ls *.tfvars 2>/dev/null | sed 's/.tfvars//' || echo "No config files found"
  exit 1
fi

echo "Deploying environment: $ENV"
echo "Using config: $CONFIG_FILE"

# Initialize Terraform (with backend)
echo ""
echo "1. Initializing Terraform..."
terraform init -reconfigure

# Validate configuration
echo ""
echo "2. Validating configuration..."
terraform validate
```

```
# Plan deployment
echo ""
echo "3. Planning deployment..."
terraform plan -var-file="$CONFIG_FILE" -out=tfplan

# Ask for confirmation
read -p "4. Apply the plan? (yes/no): " confirmation
if [ "$confirmation" != "yes" ]; then
  echo "Deployment cancelled."
  exit 0
fi

# Apply deployment
echo ""
echo "5. Applying deployment..."
terraform apply tfplan

# Show outputs
echo ""
echo "6. Deployment outputs:"
terraform output

# Cleanup
rm -f tfplan

echo ""
echo "✅ Deployment completed successfully!"
echo ""
echo "Next steps:"
echo "1. Access your application: http://$(terraform output -raw load_balancer_public_ip)"
echo "2. Check monitoring: https://portal.azure.com"
echo "3. View state: terraform state list"
echo ""
echo "To destroy: terraform destroy -var-file='$CONFIG_FILE'"
```

11. Advanced Patterns & Best Practices

11.1 Workspace Management

bash

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

# Switch between workspaces
terraform workspace select dev

# List workspaces
terraform workspace list

# Each workspace has its own state file
# backend.tf
terraform {
  backend "azurerm" {
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstate12345"
    container_name       = "tfstate"
    key                  = "${terraform.workspace}.terraform.tfstate"
  }
}
```

11.2 State Import Strategy

bash

```
# Import existing infrastructure in phases
# Phase 1: Resource Groups
terraform import azurerm_resource_group.main /subscriptions/.../resourceGroups/my-rg

# Phase 2: Virtual Networks
terraform import azurerm_virtual_network.main /subscriptions/.../resourceGroups/my-rg/providers/Microsoft.Network/virtualNetworks/my-vnet

# Phase 3: Subnets
terraform import azurerm_subnet.web /subscriptions/.../resourceGroups/my-rg/providers/Microsoft.Network/virtualNetworks/my-vnet/subnets/web

# Create import script
cat > import.sh << 'EOF'
#!/bin/bash
set -e

echo "Starting import process..."

# Import resource group
terraform import azurerm_resource_group.main $RESOURCE_GROUP_ID

# Import virtual network
terraform import azurerm_virtual_network.main $VNET_ID

# Import subnets
terraform import azurerm_subnet.web $SUBNET_WEB_ID
terraform import azurerm_subnet.app $SUBNET_APP_ID
terraform import azurerm_subnet.db $SUBNET_DB_ID

echo "Import complete!"
EOF
```

11.3 State Security

hcl

```
# Enable encryption at rest for state
# backend.tf
terraform {
  backend "azurerm"{
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstate12345"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"

    # Use customer-managed keys for encryption
    # storage_account_key = var.storage_account_key

    # Or use SAS token
    # sas_token = var.sas_token
  }
}

# Use Azure Key Vault for sensitive values
data "azurerm_key_vault_secret" "state_sas_token" {
  name         = "tf-state-sas"
  key_vault_id = azurerm_key_vault.secrets.id
}

# In CI/CD, use Managed Identity or Service Principal
```

12. Troubleshooting Common Issues

12.1 State Corruption

bash

```
# Symptoms: "Error: Failed to load state"
# Solution: Restore from backup

# 1. Always have backups
cp terraform.tfstate terraform.tfstate.backup.$(date +%Y%m%d)

# 2. If state is corrupted:
# a. Pull state from backend
terraform state pull > corrupted-state.json

# b. Create backup
cp corrupted-state.json corrupted-state.backup.json

# c. Manually edit (if you know what you're doing)
# Or restore from known good backup
terraform state push good-state.json

# 3. Use terraform state rm to remove problematic resources
terraform state rm azurerm_resource_group.corrupted
```

12.2 Provider Authentication

bash

```
# Common error: "Error: building AzureRM Client"
# Solutions:

# 1. Check Azure CLI login
az account show

# 2. Set environment variables
export ARM_CLIENT_ID="..."
export ARM_CLIENT_SECRET="..."
export ARM_SUBSCRIPTION_ID="..."
export ARM_TENANT_ID="..."
```

```
# 3. Use service principal file
export ARM_CLIENT_CERTIFICATE_PATH="/path/to/cert.pfx"
export ARM_CLIENT_CERTIFICATE_PASSWORD="..."

# 4. Debug authentication
export TF_LOG=DEBUG
terraform plan
```

12.3 Dependency Issues

bash

```
# Error: "Resource depends on non-existent resource"
# Check implicit vs explicit dependencies

# 1. Generate graph to visualize
terraform graph | dot -Tpng > graph.png

# 2. Check for circular dependencies
# Look for: A → B → A patterns

# 3. Add explicit depends_on where needed
# depends_on = [azurerm_resource_group.main]

# 4. Use terraform validate to catch issues early
terraform validate
```

---

13. Performance Optimization

13.1 Large State Files

bash

```
# State file getting too large (>100MB)

# 1. Remove old resources from state
terraform state rm $(terraform state list | grep -E "old_resource_pattern")

# 2. Use -target for specific operations
terraform apply -target=azurerm_virtual_network.main

# 3. Split infrastructure into multiple states
# Use Terraform Workspaces or separate directories

# 4. Use -parallelism to control concurrent operations
terraform apply -parallelism=5

# 5. Enable state file compression in backend
# Some backends support automatic compression
```

13.2 Module Optimization

hcl

```
# Use modules for reusability
module "network"{
  source = "./modules/network"

  # Pass only necessary variables
  environment = var.environment
  location    = var.location
}

# Keep modules focused and small
# modules/network/main.tf should only contain network resources

# Use data sources within modules to reduce duplication
```

Summary: Terraform Mastery Checklist

✅ Core Concepts Mastered:

- **Providers:** Configure and authenticate cloud providers
- **Resources:** Create, modify, and destroy infrastructure
- **Data Sources:** Query existing infrastructure
- **Variables:** Parameterize configurations
- **Locals:** Create internal computed values
- **Outputs:** Expose information to users and other configurations
- **Graph:** Understand and visualize dependencies
- **State:** Manage Terraform state effectively

✅ Advanced Skills:

- **Remote State:** Configure and use remote backends
- **State Locking:** Prevent concurrent modifications
- **Workspaces:** Manage multiple environments
- **Import:** Bring existing infrastructure under management
- **Modules:** Create reusable infrastructure components
- **Lifecycle:** Control resource creation/destruction behavior
- **Provisioners:** Execute scripts on resources (use sparingly)

✅ Production Readiness:

- **CI/CD Integration:** Automate Terraform in pipelines
- **Testing:** Validate configurations before apply
- **Security:** Manage secrets and access controls
- **Compliance:** Enforce policies and standards
- **Documentation:** Maintain clear, up-to-date docs

- **Backup & Recovery:** Protect state and configurations
- **Monitoring:** Track changes and performance

Final Architecture Diagram

text

```
┌──────────────────────────────────────────────────────────────────
────────┐
│               TERRAFORM ECOSYSTEM                    │
│               COMPLETE ARCHITECTURE                   │
├──────────────────────────────────────────────────────────────────
────────┤
│                                    │
│   ┌─────────────┐      ┌─────────────┐      ┌─────────────┐   │
│  │  PROVIDERS │      │  RESOURCES │      │   STATE   │  │
│  │  (Azure,   │───────▶│  (VMs,     │───────▶│  (JSON    │  │
│  │   AWS,     │      │   VNets)   │      │   File)   │  │
│  │   GCP)     │      │            │      │           │  │
│   └─────────────┘      └─────────────┘      └─────────────┘   │
│        │              │              │    │
│        └──────────────┬──────────────┴──────────────┬──────────┘     │
│              │            │          │
│          ┌──────▼──────┐    ┌──────▼──────┐          │
│          │  VARIABLES │     │  OUTPUTS   │        │
│          │  (Input)   │     │  (Output)  │        │
│          └─────────────┘    └─────────────┘         │
│              │            │          │
│          ┌──────▼──────┐    ┌──────▼──────┐          │
│          │   LOCALS   │     │   GRAPH    │        │
│          │ (Internal) │     │(Dependency)│        │
│          └─────────────┘    └─────────────┘         │
│                                    │
│   ┌──────────────────────────────────────────────────────────────
┐   │
```

```
│  │          BACKEND STORAGE            │   │
│  │  • Azure Storage      • AWS S3            │   │
│  │  • Google Cloud Storage• Terraform Cloud      │   │
│  │  • HashiCorp Consul    • HTTP             │   │
│  └────────────────────────────────────────────────────────────────
┘   │
│                                   │
│  ┌────────────────────────────────────────────────────────────────
┐   │
│  │           LOCKING MECHANISM          │   │
│  │  • Blob Lease (Azure) • DynamoDB (AWS)       │   │
│  │  • etcd (Consul)      • Terraform Cloud      │   │
│  └────────────────────────────────────────────────────────────────
┘   │
│                                   │
└─────────────────────────────────────────────────────────────────────
──────────┘
```