**Terraform: Infrastructure as Code (IaC)**

Terraform is an open-source tool by HashiCorp that enables **infrastructure provisioning** using a declarative configuration language (HCL - HashiCorp Configuration Language). It allows automation, scalability, and state management of infrastructure.

**1. Key Types of Blocks in Terraform**

1. **Provider Block**
   * Defines the cloud/platform to be used (AWS, Azure, GCP, etc.).
   * Contains authentication details and region settings.
   * Example:

hcl

CopyEdit

provider "aws" {

region = "us-east-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

1. **Resource Block**
   * Defines what resources Terraform should create, such as VMs, databases, networks, etc.
   * Example:

h

CopyEdit

resource "aws\_instance" "my\_vm" {

ami = "ami-12345678"

instance\_type = "t2.micro"

}

1. **Variable Block**
   * Allows defining dynamic input values instead of hardcoding.
   * Example:

hcl

CopyEdit

variable "instance\_type" {

default = "t2.micro"

}

1. **Output Block**
   * Displays important information after deployment (like public IPs, IDs, etc.).
   * Example:

hcl

CopyEdit

output "instance\_ip" {

value = aws\_instance.my\_vm.public\_ip

}

1. **Module Block**
   * Reusable components that group multiple resources together.
   * Example:

hcl

CopyEdit

module "network" {

source = "./modules/network"

}

**2. Terraform vs. Ansible**

| **Feature** | **Terraform (IaC)** | **Ansible (Configuration Management)** |
| --- | --- | --- |
| Purpose | Provision resources (VMs, networks, storage, etc.) | Configure software and manage existing resources |
| Language | HCL (HashiCorp Configuration Language) | YAML |
| Mode | Declarative | Mostly Imperative |
| Use case | Creating infrastructure from scratch | Installing packages, configuring software, managing OS settings |
| State Management | Yes (Tracks state using .tfstate file) | No (idempotent runs, but doesn't store state) |

* **Terraform**: Creates & manages infrastructure.
* **Ansible**: Configures & maintains the deployed infrastructure.

**3. Purpose of Terraform**

✅ **Provisioning resources** (VMs, networks, databases, security groups, etc.).  
✅ **Automation & templating** (Infrastructure as Code - IaC).  
✅ **Scalability & efficiency** (Manage large environments easily).  
✅ **State management** (Tracks all resources in a .tfstate file).  
✅ **Reproducibility** (Deploy entire infra repeatedly without manual intervention).  
✅ **Multi-cloud support** (AWS, Azure, GCP, etc.).

**4. AWS Cloud Overview**

* **AWS launched in 2006** as a cloud computing platform.
* **Key Benefits**:
  + No need to buy physical servers, switches, routers, firewalls, disks, etc.
  + Cost-efficient: Pay-as-you-go model.
  + Scalable and flexible infrastructure.
* **Application Deployment Workflow**:
  + Developers write code → Compilation & Build → Creates an artifact (.exe, .jar, etc.).
  + Deploy to QA/Test environment.
  + Deploy to Production servers.

**5. Terraform Registry**

* **Terraform providers & modules are available here**:  
  🔗 [Terraform Registry](https://registry.terraform.io/)  
  🔗 [Browse Providers](https://registry.terraform.io/browse/providers)
* Terraform supports different **platforms** for resource management:
  + **GUI** (AWS Console, Azure Portal, etc.)
  + **CLI** (Command Line Interface)
  + **IaC scripts** (Terraform, CloudFormation, etc.)
  + **Templates & HCL scripts** (Terraform configuration files)

**Terraform Lab Guide: Creating an AWS EC2 Instance**

**Objective**

To provision an AWS EC2 instance using Terraform in the **us-east-1** (North Virginia) region.

**Prerequisites**

* AWS account
* IAM user with required permissions
* AWS CLI installed and configured (Optional)
* Terraform installed ([Installation Guide](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli))
* Basic knowledge of Terraform commands

**Step 1: Connect to AWS EC2 Instance**

1. **Login to AWS Console** → Go to **EC2 Dashboard**
2. **Select North Virginia (us-east-1) region**
3. **Launch an EC2 Instance** (Skip installation if already set up)
4. Connect to the instance:
   * Navigate to **Instances** → Select your instance → Click **Connect**
   * Choose **EC2 Instance Connect** → Click **Connect**
   * Run the following commands:
   * sudo -i
   * mkdir raman
   * cd raman

**Step 2: Create the Terraform Configuration File**

1. Create a new file:
2. vi res.tf
3. Add the following Terraform configuration:
4. provider "aws" {
5. region = "us-east-1"
6. }
7. resource "aws\_instance" "ec2" {
8. ami = "ami-04aa00acb1165b32a"
9. instance\_type = "t2.micro"
10. availability\_zone = "us-east-1a"
11. tags = {
12. Name = "raman-first-server"
13. project = "DevOps"
14. purpose = "training"
15. }
16. }

**Step 3: Initialize Terraform**

Run the following command to download required providers:

terraform init

**Step 4: Validate Terraform Configuration**

Validate the Terraform syntax:

terraform validate

**Step 5: Review Terraform Plan**

Before applying changes, review the execution plan:

terraform plan

**Step 6: Apply Terraform Configuration**

Execute the configuration and provision the EC2 instance:

terraform apply

* Type **yes** when prompted.

**Step 7: Verify EC2 Instance**

* Go to AWS Console → **EC2 Dashboard** → Verify that the instance is running.

**Step 8: Destroy Resources (Cleanup)**

If you want to delete all created resources:

terraform destroy

* Type **yes** when prompted.

**Authentication Methods in Terraform**

Terraform supports multiple authentication methods:

1. **AWS CLI Authentication:**
2. aws configure
3. **IAM Role-Based Authentication (Recommended for EC2)**
4. **Direct Credentials in Provider Block (Not Secure - Avoid in Production)**
5. provider "aws" {
6. region = "us-east-1"
7. access\_key = "your-access-key"
8. secret\_key = "your-secret-key"
9. }

**Conclusion**

You have successfully created, managed, and destroyed an EC2 instance using Terraform. This guide provides a foundation for using Infrastructure as Code (IaC) efficiently with Terraform.

**Next Steps:**

* Experiment with different instance types and configurations.
* Use variables to make your scripts more dynamic.
* Explore Terraform modules to modularize your infrastructure.

End of Lab Guide.

**Terraform Lab Guide: VPC and EC2 Instance Deployment**

**Objective**

To provision a custom AWS Virtual Private Cloud (VPC) with public and private subnets and launch an EC2 instance in both environments using Terraform.

**Prerequisites**

* AWS account
* IAM user with required permissions
* AWS CLI installed and configured (Optional)
* Terraform installed ([Installation Guide](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli))
* Basic knowledge of Terraform commands

**Step 1: Connect to AWS EC2 Instance**

1. **Login to AWS Console** → Go to **EC2 Dashboard**
2. **Select North Virginia (us-east-1) region**
3. **Launch an EC2 Instance** (Skip installation if already set up)
4. Connect to the instance:
   * Navigate to **Instances** → Select your instance → Click **Connect**
   * Choose **EC2 Instance Connect** → Click **Connect**
   * Run the following commands:
   * sudo -i
   * mkdir raman
   * cd raman

**Step 2: Create the Terraform Configuration File**

1. Create a new file:
2. vi res.tf
3. Add the following Terraform configuration to create a **Private VPC with a Subnet**:
4. provider "aws" {
5. region = "us-east-1"
6. }
7. resource "aws\_vpc" "vpcrk" {
8. cidr\_block = "10.0.0.0/16"
9. enable\_dns\_support = true
10. enable\_dns\_hostnames = true
11. tags = {
12. Name = "manual-vpc-raman"
13. }
14. }
15. resource "aws\_subnet" "rksub" {
16. vpc\_id = aws\_vpc.vpcrk.id
17. cidr\_block = "10.0.0.0/24"
18. map\_public\_ip\_on\_launch = false
19. tags = {
20. Name = "private-raman-subnet"
21. }
22. }
23. resource "aws\_instance" "ec2" {
24. ami = "ami-04aa00acb1165b32a"
25. instance\_type = "t2.micro"
26. subnet\_id = aws\_subnet.rksub.id
27. tags = {
28. Name = "raman-first-server"
29. project = "Devops"
30. purpose = "training"
31. network = "manual"
32. }
33. }

**Step 3: Initialize Terraform**

Run the following command to download required providers:

terraform init

**Step 4: Validate Terraform Configuration**

Validate the Terraform syntax:

terraform validate

**Step 5: Review Terraform Plan**

Before applying changes, review the execution plan:

terraform plan

**Step 6: Apply Terraform Configuration**

Execute the configuration and provision the EC2 instance:

terraform apply

* Type **yes** when prompted.

**Step 7: Enhance Configuration with Public Subnet and Internet Access**

Modify **res.tf** to include an **Internet Gateway, Public Subnet, and Route Table**:

provider "aws" {

region = "us-east-1"

}

resource "aws\_vpc" "vpcrk" {

cidr\_block = "10.0.0.0/16"

enable\_dns\_support = true

enable\_dns\_hostnames = true

tags = {

Name = "manual-vpc-raman"

}

}

resource "aws\_internet\_gateway" "gw" {

vpc\_id = aws\_vpc.vpcrk.id

tags = {

Name = "manual-vpc-raman-gw"

}

}

resource "aws\_route\_table" "public\_rt" {

vpc\_id = aws\_vpc.vpcrk.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.gw.id

}

tags = {

Name = "public-route-table"

}

}

resource "aws\_subnet" "public\_rksub" {

vpc\_id = aws\_vpc.vpcrk.id

cidr\_block = "10.0.1.0/24"

map\_public\_ip\_on\_launch = true

tags = {

Name = "public-raman-subnet"

}

}

resource "aws\_route\_table\_association" "public\_assoc" {

subnet\_id = aws\_subnet.public\_rksub.id

route\_table\_id = aws\_route\_table.public\_rt.id

}

resource "aws\_instance" "public\_ec2" {

ami = "ami-04aa00acb1165b32a"

instance\_type = "t2.micro"

subnet\_id = aws\_subnet.public\_rksub.id

tags = {

Name = "raman-public-server"

project = "Devops"

purpose = "training"

network = "manual"

}

}

**Step 8: Destroy Resources (Cleanup)**

If you want to delete all created resources:

terraform destroy

* Type **yes** when prompted.

**Authentication Methods in Terraform**

Terraform supports multiple authentication methods:

1. **AWS CLI Authentication:**
2. aws configure
3. **IAM Role-Based Authentication (Recommended for EC2)**
4. **Direct Credentials in Provider Block (Not Secure - Avoid in Production)**

**Conclusion**

You have successfully created a VPC, public and private subnets, an Internet Gateway, a route table, and launched EC2 instances using Terraform. This guide provides a foundation for implementing custom networking solutions in AWS.

**Next Steps:**

* Experiment with security groups and NAT Gateways.
* Use variables to make your scripts more dynamic.
* Explore Terraform modules to modularize your infrastructure.

End of Lab Guide.