**🔬 Lab Guide 6: Provisioning an AWS EC2 Instance with Terraform**

**🎯 Learning Objectives**

By the end of this lab, participants will:

* Understand Terraform workflow (init → plan → apply → destroy).
* Write a Terraform configuration for AWS.
* Provision an **EC2 VM** (with networking + security group).
* Validate deployment and destroy cleanly.

**📘 Background for Trainees**

* **Terraform** = Infrastructure as Code (IaC) tool.
* Benefits: reproducible infra, version-controlled configs, safe changes.
* Workflow:
  1. **Write** Terraform code (.tf files).
  2. **Init** → download providers.
  3. **Plan** → preview changes.
  4. **Apply** → deploy.
  5. **Destroy** → clean up resources.

**🛠️ Prerequisites**

* AWS account with IAM user + programmatic access.
* Access key + secret key configured locally via:
* aws configure
* Installed:
  + Terraform
  + AWS CLI
* Basic knowledge of EC2 (VMs in AWS).

**🛠️ Step-by-Step Instructions**

**Step 1 – Create Project Directory**

mkdir terraform-aws-lab && cd terraform-aws-lab

**Step 2 – Define Provider**

Create provider.tf:

provider "aws" {

region = "sa-east-1" # or ap-south-1 for Mumbai

}

**Step 3 – Create EC2 Instance**

Create main.tf:

resource "aws\_instance" "sre\_vm" {

ami = "ami-0d8c29083d0808e2d" # Amazon Linux 2 AMI (us-east-1)

instance\_type = "t2.micro"

key\_name = "my-existing-key" # 👈 use your AWS key pair

vpc\_security\_group\_ids = [aws\_security\_group.sre\_sg.id]

tags = {

Name = "raman-sre-training-vm"

}

}

**Step 4 – Add Networking (Security Group)**

In security.tf:

resource "aws\_security\_group" "sre\_sg" {

name = "raman-sre-sg"

description = "Allow SSH inbound traffic"

ingress {

description = "SSH"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # WARNING: open to all (ok for lab)

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

**Step 5 – Initialize Terraform**

terraform init

* Downloads AWS provider plugin.

**Step 6 – Plan Deployment**

terraform plan

* Shows what Terraform will create (EC2 + Security Group).

**Step 7 – Apply Deployment**

terraform apply -auto-approve

* Provisions VM in AWS.
* Output will show public\_ip.

**Step 8 – Connect to VM**

ssh -i my-key.pem ec2-user@<public\_ip>

* Verify login works.

**Step 9 – Destroy Resources**

terraform destroy -auto-approve

* Cleans up infra to avoid charges.

**📄 Deliverables**

* Terraform config files (provider.tf, main.tf, security.tf).
* Screenshot of:
  + Terraform apply output.
  + AWS Console → EC2 instance running.
* Proof of SSH login to VM.

**🔬 Lab Guide 7 : Manage a Service Config Change with Ansible**

**🎯 Learning Objectives**

By the end of this lab, participants will:

* Understand how Ansible manages servers without agents.
* Write a simple Ansible playbook.
* Push a configuration change (Nginx config update) to a remote VM.
* Verify the service is updated and restarted automatically.

**📘 Background for Trainees**

* **Terraform** → builds infrastructure (VMs, networks, cloud resources).
* **Ansible** → configures and manages those VMs (packages, files, services).
* Both together = full Infra as Code pipeline.

Ansible connects to machines over **SSH**, executes **playbooks** (YAML), and ensures systems are in the desired state.

**🛠️ Prerequisites**

* A Linux VM provisioned (from previous Terraform lab).
* SSH access with key (e.g., ec2-user@<public\_ip>).
* Control node (your laptop or another VM) with Ansible installed:
* sudo apt install ansible -y # Ubuntu/Debian
* Basic knowledge of YAML.

**🛠️ Step-by-Step Instructions**

**Step 1 – Prepare Inventory File**

Create hosts.ini:

[sre\_vms]

<public\_ip\_of\_vm> ansible\_user=ec2-user ansible\_ssh\_private\_key\_file=~/my-key.pem

**Step 2 – Install Nginx (Playbook)**

Create install\_nginx.yml:

- name: Install and configure Nginx

hosts: sre\_vms

become: yes

tasks:

- name: Install Nginx

ansible.builtin.yum:

name: nginx

state: present

- name: Ensure Nginx is running

ansible.builtin.service:

name: nginx

state: started

enabled: yes

Run it:

ansible-playbook -i hosts.ini install\_nginx.yml

NOTE : MAKE SURE TO DO chmod 400 my-key.pem

Verify:

Browse on browser: http://<public\_ip>

You should see Nginx welcome page.

**Step 3 – Manage a Config Change**

Now simulate a config update (e.g., custom HTML page).

Create update\_nginx\_config.yml:

- name: Update Nginx homepage

hosts: sre\_vms

become: yes

tasks:

- name: Deploy custom index.html

ansible.builtin.copy:

dest: /usr/share/nginx/html/index.html

content: |

<html>

<head><title>SRE Lab</title></head>

<body>

<h1>Hello from Ansible-managed service!</h1>

<p>This page was deployed by Ansible at {{ ansible\_date\_time.iso8601 }}</p>

</body>

</html>

- name: Restart Nginx

ansible.builtin.service:

name: nginx

state: restarted

Run it:

ansible-playbook -i hosts.ini update\_nginx\_config.yml

Verify:

Browse on browser: http://<public\_ip>

Now you should see the **custom HTML page** with a timestamp.

**Step 4 – Change Config Again**

Edit the playbook content message → re-run → see updated homepage.  
This demonstrates **idempotency**: Ansible re-applies config until it matches desired state.

**Step 5 – Clean Up (Optional)**

Remove Nginx if needed:

- name: Remove Nginx

hosts: sre\_vms

become: yes

tasks:

- name: Uninstall Nginx

ansible.builtin.yum:

name: nginx

state: absent

**📄 Deliverables**

* Ansible inventory (hosts.ini).
* Two playbooks (install\_nginx.yml, update\_nginx\_config.yml).

**🔬 Lab Guide 8: Script to automate service restarts**

**🎯 Learning Objectives**

By the end of this lab, participants will:

* Understand how to detect and remediate a failed service using a shell script.
* Learn to use systemctl commands to monitor service status.
* Automate recovery using **cron jobs** for periodic execution.
* Compare this basic self-healing with monitoring-driven automation (Prometheus/Alertmanager).

**📘 Background**

* **Service reliability** depends on minimizing downtime.
* Traditionally, engineers check services manually:
* systemctl status nginx
* **Problem**: Manual checks don’t scale.
* **Solution**: Automate service monitoring and restarting with a simple script + cron.
* This is an example of *eliminating toil* in SRE practices.

**🛠️ Prerequisites**

* Ubuntu VM with nginx installed (sudo apt install nginx -y).
* systemd available (default on Ubuntu 18.04+).
* Basic knowledge of Linux shell scripting and cron jobs.

**🛠️ Step-by-Step Instructions**

**🛠️ Install Nginx on Ubuntu**

**Step 1 – Update package list**

sudo apt update

sudo apt install nginx -y

sudo systemctl start nginx

sudo systemctl enable nginx

systemctl status nginx

**Step 2 – Write the Monitoring Script**

Create a file named **check\_nginx.sh**:

#!/bin/bash

# Script to check and restart Nginx if down

if systemctl is-active --quiet nginx; then

echo "$(date): Nginx is running"

else

echo "$(date): Nginx is NOT running, restarting..."

sudo systemctl restart nginx

# Verify restart

if systemctl is-active --quiet nginx; then

echo "$(date): Nginx restarted successfully"

else

echo "$(date): Failed to restart Nginx!"

fi

fi

* Update permissions of the check\_nginx file

chmod +x /home/ubuntu/check\_nginx.sh

**Step 4 – Automate with Cron**

Edit the crontab:

sudo crontab -e

sudo crontab -l

Add this entry to check Nginx every minute:

\* \* \* \* \* /home/ubuntu/check\_nginx.sh >> /var/log/nginx\_monitor.log 2>&1

Save and exit. (cntrl+O), ( cntrl +X)

Check logs:

tail -f /var/log/nginx\_monitor.log

**Step 5 – Simulate Failure**

1. Stop Nginx manually:
2. sudo systemctl stop nginx
3. Wait up to 1 minute (cron schedule).
4. Check log file:
5. <timestamp>: Nginx is NOT running, restarting...
6. <timestamp>: Nginx restarted successfully
7. Confirm Nginx is active:
8. systemctl status nginx

**Step 6 – Extend (Optional)**

* Add **email/Slack notification** when a restart occurs.