AWS DevOps Terraform Infrastructure Guide

Shramish Kafle

May 24, 2025

Overview

This document provides a complete guide to deploying a professional-grade infrastructure using Terraform, GitHub Actions, and Packer on AWS, specifically tailored for training and automation.

1 Features

- Remote backend via S3 + DynamoDB for Terraform state
- Modular structure: VPC, RDS, EC2, IAM, Security Groups, ELB + ASG
- GitHub Actions CI/CD pipeline
- Packer custom AMI for Windows 2025 with:
  - Power BI Desktop and Gateway
  - Adoptium Java
  - Python
  - GoCD
- RDS in private subnet with PostgreSQL + SQL init
- S3 via VPC endpoint
- IAM best practices with scoped access

2 Setup Guide

Step 1: Clone the Repo

```
git clone https://github.com/your-org/aws-terraform-devops.git
cd aws-terraform-devops
```

Step 2: Set GitHub Secrets

Go to `Settings > Secrets > Actions` and add:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `TF_VAR_account_id`
- `TF_VAR_db_password`

Step 3: Bootstrap Terraform Backend (Run once)

```
cd bootstrap
terraform init
terraform apply
```

This provisions the S3 bucket and DynamoDB for remote state management.

Step 4: Build Custom AMI with Packer

```
cd packer
packer init .
packer build .
```

Update `terraform.auto.tfvars` with your AMI ID:

```
windows_ami_id = "ami-0abc1234def56789"
```

Step 5: Push Code and Trigger Pipeline

```
git add .
git commit -m "Initial infra push"
git push origin main
```

3 Terraform Code Walkthrough

3.1 main.tf: Root Terraform Configuration

The `main.tf` file connects all the infrastructure modules. Here's a line-by-line walkthrough:

Local Tags Block

```
locals {
  common_tags = {
    Project = "DevOps-Training"
    Environment = var.environment
    Owner = "Shramish Kafle"
  }
}
```

This defines shared tagging metadata used across all AWS resources. Tags aid in cost reporting, search, and environment scoping.

VPC Module

```
module "vpc" {
  source = "./modules/vpc"
  vpc_cidr = "10.0.0.0/16"
  aws_region = var.aws_region
  availability_zones = var.availability_zones
  environment = var.environment
  tags = local.common_tags
}
```

Provisions the main VPC with:

- Public + private subnets (across 2 AZs)
- Internet Gateway
- S3 Gateway endpoint

Security Groups

```
module "sg" {
  source = "./modules/sg"
  vpc_id = module.vpc.vpc_id
  admin_cidr = var.admin_cidr
  environment = var.environment
  tags = local.common_tags
}
```

Creates security groups for EC2, RDS, and ELB — allowing:

- RDP from your IP to EC2
- Port 1433 access from EC2 to RDS
- HTTP/HTTPS access for ELB

IAM Module

```
module "iam" {
  source = "./modules/iam"
  environment = var.environment
  account_id = var.account_id
  user_name = "tf-manager"
  tags = local.common_tags
}
```

Creates:

- An IAM role with limited access to tagged EC2, RDS, S3
- A user `tf-manager` that can assume this role

RDS Module

```
module "rds" {
  source = "./modules/rds"
  subnet_ids = module.vpc.private_subnet_ids
  sg_id = module.sg.rds_sg_id
  db_username = var.db_username
  db_password = var.db_password
  db_name = "trainingdb"
  environment = var.environment
  aws_region = var.aws_region
  secret_arn = var.secret_arn
  run_init_sql = var.run_init_sql
  tags = local.common_tags
}
```

Creates a multi-AZ PostgreSQL DB instance with:

- Automatic subnet group
- Optional schema initialization (via script)
- Secrets Manager integration
- Protected (not publicly accessible)

EC2 Module

```
module "ec2" {
  source = "./modules/ec2"
  ami_id = var.windows_ami_id
  subnet_id = module.vpc.public_subnet_ids[0]
  sg_id = module.sg.ec2_sg_id
  key_name = "my-keypair"
  user_data_file = "${path.module}/scripts/user_data.ps1"
  rds_endpoint = module.rds.rds_endpoint
  db_username = var.db_username
  db_password = var.db_password
  db_name = "trainingdb"
  environment = var.environment
  tags = local.common_tags
}
```

Provisions a Windows 2025 instance from the Packer AMI with:

- User data script that connects to RDS
- Runs SQL init on first boot
- Encrypted root volume

ELB/ASG Module

```
module "elb_asg" {
  source = "./modules/elb_asg"
  subnet_ids = module.vpc.public_subnet_ids
  vpc_id = module.vpc.vpc_id
  ami_id = var.windows_ami_id
  sg_id = module.sg.elb_sg_id
  key_name = "my-keypair"
  user_data_file = "${path.module}/scripts/user_data.ps1"
  environment = var.environment
  tags = local.common_tags
}
```

Sets up:

- Application Load Balancer (HTTP + HTTPS)
- Target group for EC2
- Auto-scaling group with launch template

4 Terraform Modules Explained

1. `modules/vpc`

- Provisions a VPC with CIDR `10.0.0.0/16`
- Two public and two private subnets across `us-east-1a` and `us-east-1b`
- Internet Gateway and S3 VPC endpoint (for isolated S3 access)
- Public route table with internet access; private subnets do not have direct internet

2. `modules/sg` (Security Groups)

- EC2 SG: RDP (3389) from admin CIDR, full outbound
- RDS SG: Allows port 1433 from EC2 SG
- ELB SG: Public ingress on HTTP (80) and HTTPS (443)

3. `modules/iam`

- IAM role with scoped permissions to EC2, RDS, S3
- IAM user that can assume the role via `sts:AssumeRole`
- IAM policy includes RDS-Data and SecretsManager for DB init

4. `modules/rds`

- Multi-AZ PostgreSQL 17.2 with private subnet group
- Not publicly accessible
- Init SQL script runs via `psql` in local-exec or via EC2 startup
- Lifecycle block for safe upgrades

5. `modules/ec2`

- EC2 Windows Server 2025 (AMI built via Packer)
- Bootstraps with RDS DB credentials and runs schema init via user data
- Custom volume size, encrypted root disk

6. `modules/elb_asg`

- Application Load Balancer with Auto-Scaling Group
- Launch Template for Windows EC2 instance
- Health checks, target group, propagation of instance tags

5 CI/CD Pipeline: GitHub Actions

File: `.github/workflows/terraform.yml`

The CI pipeline runs on every push or PR to `main`. It performs:

- Checkout
- Terraform setup (version 1.4.6)
- AWS credentials injection via GitHub secrets
- Terraform linting and validation
- `terraform init`, `plan`, and optionally `apply`

Only the main branch will trigger `terraform apply`.

Required GitHub Secrets

Go to `Settings → Secrets → Actions` and add:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `TF_VAR_account_id`
- `TF_VAR_db_password`

6 Building the Windows Server AMI (via Packer)

File: `packer/windows_server_2025.pkr.hcl`

Uses the `amazon-ebs` builder to:

- Use official Windows Server 2022 base image
- Create a new AMI with:
  - Power BI Desktop + Gateway
  - Adoptium Java
  - Python
  - GoCD Server and Agent
- Communicator: WinRM over port 5986 (SSL, insecure = true)
- Public IP used for provisioning

How to Build

```
cd packer
packer init .
packer build .
```

Copy the resulting `ami-{id}` into `terraform.auto.tfvars`.

7 Secrets and Initialization

Database Initialization

- Either run via `null_resource` with psql
- Or let EC2 run init using `scripts/user_data.ps1`

Secrets are stored in AWS Secrets Manager and accessed during provisioning.

8 Best Practices and Notes

- All resources are tagged with common tags: Project, Owner, Environment
- EC2 and RDS are deployed in separate subnets for security
- RDS has `deletion_protection = true` to avoid accidental deletes
- Use separate workspaces or directories for dev/staging/prod if scaling
- Avoid modifying existing subnet groups directly — create new ones if needed

9 Conclusion

This infrastructure is production-ready, modular, and secure. It's an excellent foundation for learning DevOps IaC practices with real-world AWS services. The CI/CD workflow ensures reproducible and auditable changes.

For questions or improvements, feel free to reach out.

Maintained by: Shramish Kafle