Terraform AWS Mono-Repo Walkthrough

Repository: `notoriousjayy-terraform-aws-infra`

Jordan Suber – Software Engineer

June 20, 2025

Agenda

1. Project Structure
2. Terraform Workflow
3. Module Layout
4. Network & Security
5. SSH Key & Access
6. Conclusion

1. Repository Layout

High-level tree

```
notoriousjayy-terraform-aws-infra/
|-- README.md
|-- terraform.tfvars.example
|-- environments/
|   `-- dev/
|       |-- backend.tf
|       |-- main.tf
|       `-- ...
`-- modules/
    |-- ec2/
    `-- vpc/
```

- **modules/** – reusable building blocks (VPC, EC2, ...)
- **environments/** – per-environment compositions (dev, stg, prod)
- **terraform.tfvars.example** – team template for variable input
- **backend.tf** – S3 remote state + DynamoDB locking

2. Day-to-Day Workflow

1. **`cd environments/dev`**
2. Configure values:

```
cp terraform.tfvars.example terraform.tfvars
# edit: aws_region, key_name, ...
```

3. Initialise plugins & back-end `$ terraform init`
4. Plan changes `$ terraform plan -out=tfplan`
5. Apply infrastructure `$ terraform apply tfplan`

Why this matters

Remote state prevents drift and the saved plan supports CI/CD gates.

3. Module Highlights

VPC Module

```
resource "aws_vpc" "this" { ... }
resource "aws_subnet" "public" { ... }
resource "aws_internet_gateway" "this" { ... }
```

- Parametrised by CIDR, subnet list, AZ list.
- Outputs: `vpc_id`, `public_subnet_ids`.

EC2 Module

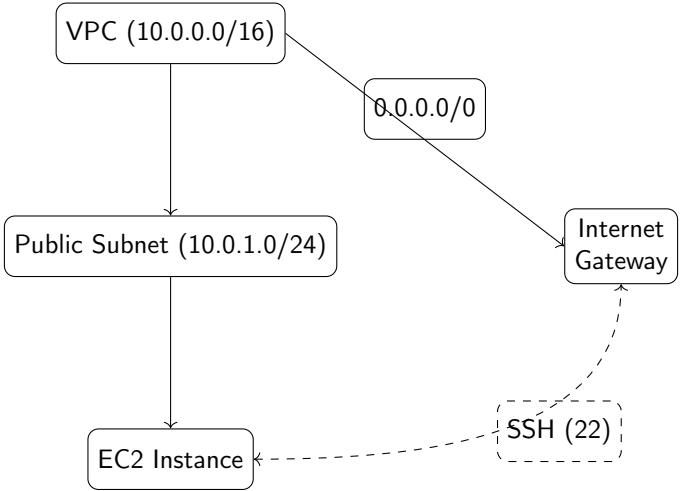
```
data "aws_ami" "debian" { ... }      # latest Debian 11
resource "aws_security_group" "ssh" { ... }
resource "aws_instance" "this" { ... }
```

Inter-Module Wiring

```
module "vpc" {                  # (1) Network foundation
  source            = "../../modules/vpc"
  cidr_block        = var.vpc_cidr
  ...
}

module "ec2" {                  # (2) Compute layer
  source            = "../../modules/ec2"
  vpc_id            = module.vpc.vpc_id
  public_subnet_ids = module.vpc.public_subnet_ids
  key_name          = aws_key_pair.debian.key_name
}
```

4. VPC Topology



Security-Group Definition

```
resource "aws_security_group" "ssh" {
  name   = "${var.instance_name}-ssh"
  vpc_id = var.vpc_id

  ingress {
    from_port  = 22
    to_port    = 22
    protocol   = "tcp"
    cidr_blocks = [var.ssh_ingress_cidr] # lock to VPN in prod
  }

  egress {
    from_port  = 0
    to_port    = 0
    protocol   = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

5. Key-Pair Automation

```
resource "tls_private_key" "ssh" {
  algorithm = "RSA"
  rsa_bits = 4096
}

resource "aws_key_pair" "debian" {
  key_name  = var.key_name
  public_key = tls_private_key.ssh.public_key_openssh
}
```

SSH into the Instance

After ‘terraform apply‘:

```
instance_public_ip = "3.145.128.42"
private_key_pem  = "<redacted>"
```

1. Save the key and lock permissions `$ echo "$TF_PRIV_KEY" > dev.pem && chmod 600 dev.pem`
2. Connect (user `admin`):

```
$ ssh -i dev.pem admin@3.145.128.42
```

Tip ↓

Add an entry in `/.ssh/config` to shorten future log-ins.

Key Takeaways

- Mono-repo split into **modules** and **environments**.
- Remote S3 back-end + locking keeps state safe.
- Clean outputs wire VPC $\rightarrow$ EC2 without duplication.
- Key-pair generated at apply-time; never committed.
- One-command onboarding: `terraform init && apply`.

Welcome aboard – Happy provisioning!

Questions?