# 🔁 Task: Optimize Terraform Infrastructure for Scalability and Reusability

## 🎯 Objective

Refactor and organize the Terraform codebase to support scalable, modular, and environment-agnostic infrastructure, making it easy to:

- Deploy multiple environments (e.g., dev, staging, prod)

- Support multiple cloud providers (AWS, Azure, GCP, etc.)

- Add or scale infrastructure components (VMs, load balancers, networks)

- Reuse code across services or teams

- Terraform to provision infrastructure

- CI/CD pipelines to automate deployment

- Docker to containerize the NGINX app

- OpenSSL to enable HTTPS support within the containers

- Application Load Balancers (AWS ALB & Azure App Gateway) to distribute traffic

- Optional: route global traffic via DNS failover or load balancing

---

## Key Subtasks

### 1. Create Reusable Terraform Modules

Build modular components for:

- Compute (VMs)

- Networking (VPCs, subnets, security groups)

- Load Balancers (ALB, App Gateway)

- Docker + App deployment logic

Each module should:

- Be parameterized using \*\*variables.tf\*\*

- Output key values like IPs, DNS, certs, etc.

\*\*Example module structure:\*\*

```

modules/

├── compute/

├── networking/

├── loadbalancer/

├── nginx-app/

```

---

### 2. 🏗️ Support Multiple Environments

Define environment-specific configurations:

- dev, staging, prod

- Use workspaces or separate directories for each

- Use shared modules with environment-specific variables

\*\*Example structure:\*\*

```

environments/

├── dev/

│ └── terraform.tfvars

├── prod/

│ └── terraform.tfvars

```

---

### 3. 🌍 Multi-Cloud Support

- Use separate provider blocks and modules for \*\*AWS\*\* and \*\*Azure\*\*

- Enable toggling cloud deployments via variables or workspaces

- AWS:

- Create a VPC with public and private subnets

- Deploy the NGINX VMs with EC2 instance (Ubuntu) into private subnets

- Add a NAT Gateway in the public subnet

- Route private subnet outbound traffic through the NAT Gateway

- Ensure no public IPs are attached directly to backend VMs

- Security Group (Allow 22, 80, 443)

- Application Load Balancer (ALB)

- Forward to EC2 on port 443

- Azure:

- Create a Virtual Network with:

- A public subnet (for bastion or load balancer)

- A private subnet (for NGINX VM)

- Use Azure NAT Gateway or Load Balancer Outbound Rules for internet access

- Ensure no public IPs are attached directly to backend VMs

- Virtual Machine (Ubuntu)

- NSG (Allow 22, 80, 443)

- Application Gateway (HTTPS listener)

- Backend pool → Azure VM on port 443

\*\*Example:\*\*

```hcl

variable "enable\_aws" { type = bool }

variable "enable\_azure" { type = bool }

```

---

### 4. 🔄 Scalable Compute Design

- Allow module to deploy multiple VMs using `count` or `for\_each`

- Output public/private IPs dynamically

- Group VMs under auto-scaling or target groups where applicable

---

### 5. 🗂️ Remote State Management

- Store state in a centralized backend (e.g., \*\*S3 + DynamoDB\*\* or \*\*Azure Storage Account\*\*)

- Use \*\*locking\*\* to avoid conflicts during concurrent deployments

---

### 6. ⚙️ CI/CD Integration with Multi-Environment Support

Extend the pipeline to:

- Deploy the infra using both Jenkins & Azure Devops YAML pipelines

- Accept `env` as input

- Run `terraform apply` for chosen environment

- Use different state files / workspaces per environment

\*\*Example:\*\*

```yaml

env:

ENVIRONMENT: dev # or staging, prod

```

### 7. 🔐 Dockerized NGINX App with SSL (OpenSSL)

Inside the Docker container:

- Generate self-signed SSL certs using OpenSSL

- Configure NGINX to serve HTTPS using:

- /etc/nginx/certs/cert.pem

- /etc/nginx/certs/key.pem

- Redirect HTTP → HTTPS

📌 Docker should expose port 443, and optionally port 80 for redirect.

### 8. 📶 Load Balancing

- AWS ALB:

- Listener on port 443

- Forwarding to EC2 on port 443

- Target group: EC2 instance with health check

- Azure App Gateway:

- Listener on port 443

- Backend pool: Azure VM

- Health probe on / (HTTPS)

### 9. 🌍 (Optional) Global DNS Routing

Use Route 53 / Cloudflare to:

- Point nginx.example.com to both:

- AWS ALB DNS

- Azure App Gateway DNS

Use failover, latency-based, or weighted routing

---

## ✅ Success Criteria

- Infra code is \*\*modular, DRY, and easy to extend\*\*

- Modules can be reused across multiple environments and teams

- Can deploy:

- Multiple VMs with one config

- To AWS, Azure (and more)

- CI/CD pipeline supports different environments and scales infra cleanly using ADO & Jenkins.

- State is managed safely and centrally

- Both AWS and Azure infrastructures are provisioned via Terraform

- Load balancers route HTTPS traffic to backend NGINX containers

- SSL works with self-signed certs (browser warning expected)

- Application is available on:

- https://<aws-alb-dns>

- https://<azure-appgw-dns>

- Document the whole process with Github README.md file

---

## 💡 Benefits

- Easier collaboration across teams

- Rapid environment provisioning

- Cloud-agnostic and ready for multi-region/cloud scaling

- Better separation of concerns (infra logic vs. env config)

---