**Devops Exercises**

Please pick just 2 of the exercises below and complete them. Once you have completed your

submission, place it in a github repo that you share.

We will schedule a time to review your solution with you after we have had a chance to review it

internally.

**AWS VPC Exercise**

You have been assigned a project to create a new terraform module for creating a staging VPC

in AWS. The following requirements have been given to you.

- The network supernet of 172.16.0.0/16

- The VPC should provide an endpoint for AWS Systems Manager (SSM), internally.

- The VPC should provide an internal endpoint for S3.

- The VPC should have two different availability zones, with two private subnets and two

public subnets for each availability zone.

- The VPC should contain appropriate routing tables.

- The VPC should include appropriate NAT gateways and an internet gateway.

Create module code to fulfill this request utilizing terragrunt/terraform. Show how you would use

this module code to bring up the given architecture.

**A Small EC2 App**

Using the following technologies:

- Packer

- Ansible

- Terraform/terragrunt.

- Utilizing a pack/fry idiom in ansible, create an Amazon Machine Image (AMI) which runs

a small nginx webserver. (A static page loading here is just fine).

- It should be based on the latest Amazon Linux 2023 AMI.

- Ensure there are both pack and fry roles and that the appropriate variables

needed for the ami are injectable via userdata in the frying role. (if applicable)

- Create a terraform stack which creates the following configuration:

- An autoscaling group of two Ec2 Instances

- Launch template

- Instance profile role along with manageable policy to be attached to the

EC2 instances.

- An application load balancer (ALB)

- A target group where you will place the ec2 instances.

- Security groups for the load balancer and ec2 instances with the proper rules to

ensure ssh access to the instances and communication between the instances

and the load balancer.

- Place the autoscaling group on a private subnet.

- Bonus points if you add transport layer security support via an SSL cert through AWS Certificate Manager (ACM) but not necessary for this exercise.

**What I understood from the exercise related to AWS VPC Exercise**

* A **staging VPC** on **172.16.0.0/16**
* **Two AZs**, and **two private + two public subnets per AZ** (→ 4 private + 4 public)
* **Routing tables**, **IGW**, **NAT gateways** (one per AZ)
* **Private endpoints**: **SSM (interface)** and **S3 (gateway)**
* Show **module code** and **how to use it**; optionally via **Terragrunt**; and show **plan only for VPC** resources.

**Structure of the files and folder in the mono repository**

1. modules/vpc
   * Creates the VPC (cidr = 172.16.0.0/16) with two AZs.
   * 4 private + 4 public subnets (you pass explicit CIDRs).
   * **IGW** for public subnets.
   * **NAT gateways per AZ** and private route tables with default route via NAT.
   * **VPC endpoints submodule**:
     + s3 **gateway** endpoint associated to the route tables for internal S3.
     + ssm **interface** endpoint (in subnets) with private DNS for internal SSM.
2. modules/s3\_bucket\_state
   * Bootstraps (first run) the **S3 state bucket** and **DynamoDB lock table**.
   * Has **creation toggles**: create\_bucket / create\_lock\_table.
   * On later runs we flip them to false and feed back the existing names.
3. modules/iam\_tf\_policies
   * Creates/uses IAM policies the pipeline needs:
     + RW to the state bucket + lock table.
     + Minimal VPC “apply” permissions (Describe + CreateVpc/CreateTags etc.).
   * Skips creation if an **existing policy ARN** is detected/passed.
4. modules/github\_oidc
   * Uses an **existing** GitHub OIDC provider ARN (or can create one),
   * Creates an **assumable role** for the workflow and **attaches** the policies from (3).

**Putting all the code centralized in the mono repository**

The root file is main.tf calling the modules and passes inputs:

module "vpc" { # ← the VPC with subnets, NAT, routes, endpoints

source = "./modules/vpc"

name = var.vpc\_name

cidr = "172.16.0.0/16"

azs = ["eu-west-1a", "eu-west-1b"]

private\_subnets = ["172.16.0.0/20","172.16.16.0/20","172.16.32.0/20","172.16.48.0/20"]

public\_subnets = ["172.16.64.0/20","172.16.80.0/20","172.16.96.0/20","172.16.112.0/20"]

enable\_nat\_gateway = true

one\_nat\_gateway\_per\_az = true

single\_nat\_gateway = false

}

module "s3\_bucket\_state\_oidc" { # ← state bucket + lock table (first run only)

source = "./modules/s3\_bucket\_state"

bucket\_prefix\_name = var.bucket\_prefix\_name

lock\_table = var.lock\_table

create\_bucket = var.create\_bucket

create\_lock\_table = var.create\_lock\_table

state\_key = "envs/${var.environment}/terraform.tfstate"

existing\_bucket\_name = var.existing\_bucket\_name

existing\_lock\_table = var.existing\_lock\_table

}

module "iam\_tf\_policies" { # ← policies used by the OIDC role

source = "./modules/iam\_tf\_policies"

bucket\_name = module.s3\_bucket\_state\_oidc.s3\_bucket\_id

lock\_table\_name = module.s3\_bucket\_state\_oidc.lock\_table\_name

region = var.region

depends\_on = [module.s3\_bucket\_state\_oidc]

}

module "github\_oidc" { # ← OIDC role that attaches those policies

source = "./modules/github\_oidc"

create\_oidc\_provider = false

oidc\_provider\_arn = local.existing\_provider\_arn

create\_oidc\_role = true

repositories = var.repository\_list

oidc\_role\_attach\_policies = [

module.iam\_tf\_policies.tf\_backend\_rw\_policy\_arn,

module.iam\_tf\_policies.tf\_vpc\_apply\_policy\_arn

]

depends\_on = [module.iam\_tf\_policies]

}

# **Short summary**

# The module vpc creates the VPC with subnets, NAT, routes, endpoints.

* The module iam\_tf\_policies creates the iam tf polcicy and identify required for github oidc
* The module github\_oidc is creating the OIDC role that attahes the polcies in the module iam\_tf\_polcies

The dependencies are as follows

1. “iam\_tf\_policies” module depending on “s3\_bucket\_state\_oidc”
2. “github\_oidc” depending on the “iam\_tf\_policies” module.

More information about the plan can be seen in [this link](https://github.com/mklmfane/yamasoft-assignment/actions/runs/18636865298/job/53129232850).

**How I split in CI/CD flow**

* **Bootstrap job** (long-lived AWS keys just once):
  1. init locally (backend disabled),
  2. create **state S3 + DDB**, **policies**, **OIDC role** (module targets),
  3. export bucket/table/role outputs.
* **Deploy job** (assume OIDC role):
  1. flip create\_bucket=false and create\_lock\_table=false,
  2. migrate TF backend to S3 + DDB,
  3. plan/apply the whole stack.

**How I plan the exercise**

Three practical ways (pick one):

* **Simple (demo)**: target the module

terraform plan -target=module.vpc -input=false -no-color

(Good for showcasing; do not use -target routinely for full lifecycles.It was used for cost reasons )

* **Clean separation**: run the VPC as its **own stack** (Terragrunt)

live/staging/vpc/terragrunt.hcl → source = ../../..//modules/vpc

Then:

cd live/staging/vpc

terragrunt plan

* I set inputs so non-VPC modules don’t create (e.g. create\_bucket=false, create\_lock\_table=false) and just terraform plan to create VPC resources.

**How the requirements for this exercise are met**

* CIDR and subnet layout match **172.16.0.0/16** having each **2 AZs × (2 private + 2 public)**.
* **IGW** for public subnets;
* **NAT per AZ** for private egress.
* **Route tables** were created and associated appropriately for public and the private subnets.
* **Endpoints**:
  + **S3**: **Gateway** endpoint for internal S3 access.
  + **SSM**: **Interface** endpoint with private DNS for internal SSM.

**Deploying An Application**

Helm

- Create a simple templated helm chart that loads up an nginx server and an appropriate

ingress. (assume either an nginx or alb controller is just fine)

- Create a values.yml file that fulfills the values for this helm chart.

- Show a command that dumps the template generated.

**Github Actions**

- Create an example github action that would use Github’s Open ID Connect (OIDC)

provider as a trusted AWS Identity to deploy the helm chart above into a staging EKS

cluster.