## **1. CALMS Framework**

**CALMS** is a conceptual model used to assess an organization’s DevOps maturity. It stands for:

| **Component** | **Description** |
| --- | --- |
| **C - Culture** | Emphasizes collaboration, shared responsibility, and breaking down silos between Development and Operations. |
| **A - Automation** | Focuses on automating repetitive tasks such as testing, deployment, infrastructure provisioning to increase speed and reduce human error. |
| **L - Lean** | Applies Lean principles to eliminate waste, improve flow, and focus on delivering customer value faster. |
| **M - Measurement** | Promotes collecting metrics (e.g., lead time, deployment frequency, failure rate) to make data-driven decisions. |
| **S - Sharing** | Encourages open communication, sharing knowledge, tools, and feedback across teams to foster trust and collaboration. |

## **2. The Three Ways (from "The Phoenix Project" by Gene Kim)**

These are guiding principles for the DevOps journey:

| **Way** | **Description** |
| --- | --- |
| **1. The First Way: Flow** | Focus on improving the flow of work from Development to Operations to the customer. Includes CI/CD, small batch sizes, and fast feedback loops. |
| **2. The Second Way: Feedback** | Enable and amplify feedback loops to fix issues early and improve quality. Includes automated testing, monitoring, and postmortems. |
| **3. The Third Way: Continual Learning and Experimentation** | Foster a culture of learning, experimentation, and risk-taking. Encourages blameless postmortems and innovation. |

## **Terraform Folder Structure**

my-terraform-project/

│

├── main.tf

├── variables.tf

├── outputs.tf

├── terraform.tfvars

├── provider.tf

├── backend.tf

└── modules/

└── ec2-instance/

├── main.tf

├── variables.tf

└── [outputs.tf](http://outputs.tf)

**Provider**:

provider "aws" {

region = var.region

profile = var.aws\_profile

assume\_role {

role\_arn = var.role\_arn

session\_name = "terraform-session"

}

}

**Backend:**

terraform {

backend "s3" {

bucket = "my-terraform-state-bucket" # S3 bucket name

key = "env/dev/terraform.tfstate" # Path to state file

region = "us-east-1" # AWS region

profile = "dev-profile" # AWS CLI profile

encrypt = true # Encrypt the state at rest in S3

dynamodb\_table = "terraform-locks" # DynamoDB table for state locking

acl = "bucket-owner-full-control" # Optional S3 ACL

}

}

**Main:**

# ✅ Fetch the most recent official Ubuntu 20.04 AMI

data "aws\_ami" "ubuntu" {

most\_recent = true # Get the latest available image

owners = ["099720109477"] # Canonical's AWS account ID

filter {

name = "name" # Filter by name pattern

values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-\*"]

}

}

**# Create an EC2 instance using a reusable module**

module "ec2\_instance" {

source = "./modules/ec2-instance" # Source path of the EC2 instance module

ami\_id = data.aws\_ami.ubuntu.id # Use the fetched Ubuntu AMI ID

instance\_type = var.instance\_type # Define instance type from variables

subnet\_id = var.subnet\_id # Subnet to launch the instance in

key\_name = var.key\_name # SSH key for access

vpc\_security\_group\_ids = var.security\_group\_ids # List of security groups for the instance

tags = {

Name = "UbuntuEC2" # Tag to name the instance

Environment = var.env # Tag to set the environment (e.g., dev, prod)

}

}

**Variable.tf**

variable "region" {

default = "us-east-1"

}

variable "instance\_type" {

description = "The EC2 instance type"

type = string

default = "t2.micro"

}

**Terraform.tfvars**

region = "us-east-1"

instance\_type = "t2.micro"

**outputs.tf**

output "ec2\_public\_ip" {

value = module.ec2\_instance.public\_ip

}

#togetOutput terraform output ec2\_public\_ip

**modules/ec2-instance/main.tf**

resource "aws\_instance" "this" {

ami = var.ami\_id

instance\_type = var.instance\_type

tags = {

Name = "TerraformExample"

}

}

**Security Group:**

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

name = "web-sg"

description = "Allow SSH and HTTP"

vpc\_id = aws\_vpc.main\_vpc.id

ingress {

description = "Allow SSH"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # Caution: open to all

}

ingress {

description = "Allow HTTP"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

description = "Allow all outbound traffic"

from\_port = 0

to\_port = 0

protocol = "-1" # all protocols

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "web-sg"

}

}

(Attach with Instance: vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id] )

## **Kubernates with Terraform:**

**# 1. Configure Kubernetes Provider**

provider "kubernetes" {

config\_path = "~/.kube/config"

config\_context = "my-cluster-context"

}

**# 2. Create a Kubernetes Namespace**

resource "kubernetes\_namespace" "example" {

metadata {

name = "example-namespace"

}

}

**# 3. Deploy a Kubernetes Deployment**

resource "kubernetes\_deployment" "nginx" {

metadata {

name = "nginx-deployment"

namespace = kubernetes\_namespace.example.metadata[0].name

}

spec {

replicas = 3

selector {

match\_labels = {

app = "nginx"

}

}

template {

metadata {

labels = {

app = "nginx"

}

}

spec {

container {

image = "nginx:1.21"

name = "nginx"

port {

container\_port = 80

}

}

}

}

}

}

**# 4. Expose Deployment with Service**

resource "kubernetes\_service" "nginx\_svc" {

metadata {

name = "nginx-service"

namespace = kubernetes\_namespace.example.metadata[0].name

}

spec {

selector = {

app = kubernetes\_deployment.nginx.spec[0].template[0].metadata[0].labels.app

}

port {

port = 80

target\_port = 80

}

type = "LoadBalancer"

}

}

**# 5. Create Kubernetes ConfigMap**

resource "kubernetes\_config\_map" "example" {

metadata {

name = "example-config"

namespace = kubernetes\_namespace.example.metadata[0].name

}

data = {

"example.key" = "example value"

}

}

**# 6. Create Kubernetes Secret**

resource "kubernetes\_secret" "db\_password" {

metadata {

name = "db-password"

namespace = kubernetes\_namespace.example.metadata[0].name

}

data = {

password = base64encode("supersecret123")

}

type = "Opaque"

}

**# 7. Provision a Managed Kubernetes Cluster (EKS example)**

module "eks" {

source = "terraform-aws-modules/eks/aws"

cluster\_name = "my-cluster"

cluster\_version = "1.24"

subnets = aws\_subnet.public[\*].id

vpc\_id = aws\_vpc.main.id

node\_groups = {

eks\_nodes = {

desired\_capacity = 2

max\_capacity = 3

min\_capacity = 1

instance\_type = "t3.medium"

}

}

}

**# 8. Kubernetes Provider Configuration with EKS Cluster Data**

data "aws\_eks\_cluster" "cluster" {

name = module.eks.cluster\_id

}

data "aws\_eks\_cluster\_auth" "cluster" {

name = module.eks.cluster\_id

}

provider "kubernetes" {

host = data.aws\_eks\_cluster.cluster.endpoint

token = data.aws\_eks\_cluster\_auth.cluster.token

cluster\_ca\_certificate = base64decode(data.aws\_eks\_cluster.cluster.certificate\_authority[0].data)

}

**# 9. Apply YAML manifest with kubernetes\_manifest resource**

resource "kubernetes\_manifest" "example" {

manifest = yamldecode(file("deployment.yaml"))

}

**# 10. Use kubectl local-exec provisioner to run commands**

resource "null\_resource" "kubectl\_apply" {

provisioner "local-exec" {

command = "kubectl apply -f k8s/deployment.yaml"

}

}

## **You have a local Kubernetes cluster running You want to apply them via Terraform.**

provider "kubernetes" {

config\_path = "~/.kube/config" # or point to your kubeconfig

}

resource "kubernetes\_manifest" "my\_deployment" {

manifest = yamldecode(file("${path.module}/deployment.yaml"))

}

resource "kubernetes\_manifest" "my\_service" {

manifest = yamldecode(file("${path.module}/service.yaml"))

}

**IF HELM**

provider "helm" {

kubernetes {

config\_path = "~/.kube/config"

}

}

resource "helm\_release" "nginx" {

name = "nginx"

repository = "https://charts.bitnami.com/bitnami"

chart = "nginx"

version = "15.2.0"

namespace = "default" }

## **Terraform with Digital ocean**

# This block specifies the required providers and their versions

# You can update the versions if needed, but these are stable versions

terraform {

required\_providers {

digitalocean = {

source = "digitalocean/digitalocean"

version = "~> 2.0"

}

kubernetes = {

source = "hashicorp/kubernetes"

version = "~> 2.0"

}

}

}

# DigitalOcean provider configuration

# This is where you'll need to provide your DigitalOcean API token

provider "digitalocean" {

token = var.do\_token

}

# This creates a Kubernetes cluster on DigitalOcean

# You can customize the following parameters:

# - name: Change "nextjs-cluster" to your preferred cluster name

# - region: Change "nyc1" to your preferred region (e.g., "ams3", "sgp1", "lon1")

# - version: Update to a newer Kubernetes version if needed

# - node\_pool:

# - size: Change "s-2vcpu-4gb" to other sizes like "s-1vcpu-2gb" or "s-4vcpu-8gb"

# - node\_count: Adjust the number of nodes (e.g., 1, 3, 4)

resource "digitalocean\_kubernetes\_cluster" "k8s\_cluster" {

name = "nextjs-cluster"

region = "nyc1"

version = "1.27.4-do.0"

node\_pool {

name = "default-pool"

size = "s-2vcpu-4gb"

node\_count = 2

}

}

# Kubernetes provider configuration

# This automatically configures the Kubernetes provider using the cluster details

# You don't need to modify this unless you have specific requirements

provider "kubernetes" {

host = digitalocean\_kubernetes\_cluster.k8s\_cluster.endpoint

token = digitalocean\_kubernetes\_cluster.k8s\_cluster.kube\_config[0].token

cluster\_ca\_certificate = base64decode(

digitalocean\_kubernetes\_cluster.k8s\_cluster.kube\_config[0].cluster\_ca\_certificate

)

}

[**variable.tf**](http://variable.tf)

variable "do\_token" {

description = "DigitalOcean API token"

type = string

sensitive = true

}