EKS Cluster Creation with Terraform – Full Production Guide (Code Included)

1. Purpose

This document provides a **complete, production-ready EKS setup using Terraform**, including:

- EKS cluster creation
- Networking & IAM
- IRSA (IAM Roles for Service Accounts)
- AWS Load Balancer Controller
- GitOps with Argo CD
- Multi-region DR
- FinOps cost optimization

This is designed for **real-world environments**, not demos.

---

2. Prerequisites

- AWS account
- AWS CLI configured
- Terraform >= 1.x
- kubectl
- Helm

---

3. Terraform Project Structure

```
eks-terraform/
├── backend.tf
├── provider.tf
├── variables.tf
├── vpc.tf
├── iam.tf
├── eks.tf
├── nodegroup.tf
├── irsa.tf
├── autoscaler.tf
├── externaldns.tf
├── secrets.tf
├── outputs.tf
```

---

4. Terraform Backend (Remote State)

```
terraform {
  backend "s3" {
    bucket         = "eks-terraform-state"
    key            = "eks/terraform.tfstate"
    region         = "us-east-1"
    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
}
```

---

5. Provider Configuration

```
provider "aws" {
  region = var.region
  default_tags {
    tags = {
      Environment = "production"
      Owner       = "platform"
      CostCenter  = "eks"
    }
  }
}
```

---

6. VPC and Networking (vpc.tf)

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  name    = "eks-vpc"
  cidr    = "10.0.0.0/16"

  azs             = ["us-east-1a", "us-east-1b"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24"]

  enable_nat_gateway = true
  single_nat_gateway = true
}
```

Best Practice: **Worker nodes always in private subnets**.

---

7. IAM Roles (iam.tf)

EKS Cluster Role

```
resource "aws_iam_role" "eks_cluster_role" {
  name = "eks-cluster-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = { Service = "eks.amazonaws.com" }
      Action = "sts:AssumeRole"
    }]
  })
}
```

Attach policies:

- AmazonEKSClusterPolicy
- AmazonEKSVPCResourceController

---

8. EKS Cluster (eks.tf)

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  cluster_name    = "prod-eks"
  cluster_version = "1.29"

  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets

  enable_irsa = true
}
```

---

9. Managed Node Groups (nodegroup.tf)

```
module "eks" {
  eks_managed_node_groups = {
    default = {
      instance_types = ["t3.medium"]
      desired_size   = 2
      max_size       = 5
      min_size       = 1
      capacity_type  = "ON_DEMAND"
    }
    spot = {
      instance_types = ["t3.large"]
      desired_size   = 1
      max_size       = 3
      min_size       = 0
      capacity_type  = "SPOT"
    }
  }
}
```

---

10. Configure kubectl

```
aws eks update-kubeconfig --region us-east-1 --name prod-eks
kubectl get nodes
```

---

11. IRSA – IAM Roles for Service Accounts (irsa.tf)

```
module "alb_irsa" {
  source = "terraform-aws-modules/iam/aws//modules/iam-role-for-service-accounts-eks"

  role_name = "alb-controller"

  attach_load_balancer_controller_policy = true

  oidc_providers = {
    main = {
      provider_arn               = module.eks.oidc_provider_arn
      namespace_service_accounts = ["kube-system:aws-load-balancer-controller"]
    }
  }
}
```

---

12. AWS Load Balancer Controller (Helm)

```
helm repo add eks https://aws.github.io/eks-charts
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=prod-eks \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

---

13. GitOps with Argo CD

Install Argo CD

```
kubectl create namespace argocd
kubectl apply -n argocd \
  -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

---

14. GitOps Repository Structure

```
gitops-repo/
├── apps/
│   ├── frontend/
│   ├── backend/
│   └── database/
└── argocd-apps/
```

---

15. Argo CD Application Manifest

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: frontend
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/org/gitops-repo
    path: apps/frontend
    targetRevision: main
  destination:
    server: https://kubernetes.default.svc
    namespace: frontend
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

---

16. Production Best Practices

- Private API endpoint for EKS
- IRSA for all controllers
- Image scanning before deploy
- Resource limits on pods
- HPA + Cluster Autoscaler
- Multi-region DR planning
- FinOps cost controls

---

17. Multi-Region Disaster Recovery (DR)

Strategy

- Active–Passive EKS clusters in two regions
- Route 53 health-check–based failover
- GitOps keeps clusters synchronized automatically

Primary: `us-east-1`
Secondary: `us-west-2`

Route 53 Failover

```
resource "aws_route53_health_check" "primary" {
  fqdn              = "app.example.com"
  port              = 443
  type              = "HTTPS"
  resource_path     = "/health"
  failure_threshold = 3
  request_interval  = 30
}

resource "aws_route53_record" "primary" {
  zone_id = var.zone_id
  name    = "app.example.com"
  type    = "A"
  set_identifier = "primary"
  failover_routing_policy { type = "PRIMARY" }
  alias { name = aws_lb.primary.dns_name; zone_id = aws_lb.primary.zone_id;
evaluate_target_health = true }
}

resource "aws_route53_record" "secondary" {
  zone_id = var.zone_id
  name    = "app.example.com"
  type    = "A"
  set_identifier = "secondary"
  failover_routing_policy { type = "SECONDARY" }
  alias { name = aws_lb.secondary.dns_name; zone_id = aws_lb.secondary.zone_id;
evaluate_target_health = true }
}
```

Argo CD Multi-Cluster

```
argocd cluster add primary-context
argocd cluster add secondary-context
```

ApplicationSet for multi-region deployments ensures manifests are synced in both clusters automatically.

---

18. FinOps Cost Controls

Node Groups & Scaling

- Use **mixed instance types**
- Use **Spot instances** for non-critical workloads
- HPA + Cluster Autoscaler reduces waste

```
capacity_type = "SPOT"
```

Network & Storage Costs

- Prefer **S3 over EFS**
- Monitor NAT Gateway traffic
- Use VPC endpoints where possible

CI/CD Costs

- Enforce image size limits
- Remove unused images from ECR

- Scan images before build

Observability for FinOps

- Prometheus/Grafana dashboards for utilization and idle resources
- Track underutilized workloads and over-provisioned nodes

---

19. Final Architecture Outcome

- Multi-region DR with automatic failover
- GitOps sync across clusters
- Autoscaling at pod and node layers
- Secure secrets via Secrets Manager + CSI Driver
- ALB ingress with ExternalDNS
- Full FinOps visibility and cost optimization
