KubernetesProject Terraform Overview

Generated: November 17, 2025

Repository: `/home/atw/Desktop/kubernatesProject`

1. Root-Level Configuration

`provider.tf`

- Pins the AWS provider to `~> 6.20`.
- Configures Terraform to use an S3 backend (`assemsamirziad` bucket, key `state/terraform.tfstate`, region `eu-north-1`, encryption enabled) so state is shared remotely.
- Sets the default AWS region from `var.region`.

`variables.tf`

- Declares all top-level inputs, including project metadata (`project_name`, `environment`), networking (`vpc_cidr`, subnet CIDRs and names for AZs A & B), EKS cluster settings (`cluster_name`, `kubernetes_version`), and node-group sizing/instance type defaults (`t3.small`, desired/min/max = 3).
- Also defines the S3 bucket name used for additional resources (`bucket_name`).

`main.tf`

- Composes the infrastructure by instantiating the `vpc` and `eks` modules located in subdirectories.
- Passes all networking variables (names/CIDRs for both AZs) into the VPC module.
- Wires the EKS module to the created VPC by supplying `vpc_id`, `vpc_cidr`, and the two private subnet IDs (one per AZ) so the control plane and node group span multiple AZs.
- Forwards node-group sizing and instance-type variables to the EKS module.

`outputs.tf`

- Exposes identifiers for the VPC, subnets, routing components, cluster, and node group so they can be referenced externally or from other Terraform workspaces.

`s3.tf`

- Provisions an S3 bucket used by the project (separate from the backend bucket).
- Enforces public access blocking and (currently disabled) versioning.

`terraform.tfstate`

- Present locally as part of Terraform's bookkeeping. Once the backend is initialized, the authoritative copy resides in the S3 bucket configured in `provider.tf`.

2. VPC Module (`./vpc`)

`vpc.tf`

• Creates the base VPC with tags, DNS support, and DHCP options (if any). This is the networking foundation for all other resources.

`public-subnet.tf` / `private-subnet.tf`

• Define two public and two private subnets (`public_a`, `public_b`, `private_a`, `private_b`) mapped to the first two availability zones discovered via `data.aws_availability_zones`.

• Public subnets enable public IP assignment; private subnets disable it for worker isolation.

`route-table-*.tf` and `route-table-associations.tf`

• Build and associate public/private route tables.

• Public subnets route internet-bound traffic through an Internet Gateway; private subnets route through NAT (see `nat-gateway.tf`).

`internet-gateway.tf` / `nat-gateway.tf`

• Provide outbound connectivity: the Internet Gateway serves public subnets, while the NAT Gateway enables private subnets to reach the internet for updates/images without exposing them publicly.

`data.tf`

• Fetches available AZs and stores them in `local.availability_zones` for consistent indexing across all subnet resources.

`outputs.tf`

• Returns IDs and CIDRs for each subnet (both AZs), plus gateway IDs, enabling the root module to consume them.

`variables.tf`

• Mirrors the root networking inputs, keeping the VPC module configurable and reusable.

3. EKS Module (`./eks`)

`cluster.tf`

- Declares the `aws_eks_cluster` resource.
- Configures API endpoints (private-only access), attaches the cluster IAM role, security group, and enables control-plane logging (`api`, `audit`, `authenticator`, `controllerManager`, `scheduler`).

`iam-cluster.tf` / `iam-node.tf`

- Provision IAM roles and policy attachments for the control plane and worker nodes.
- Node role trusts EC2 and attaches the standard EKS node policies (WorkerNode, CNI, ECR read-only).

`security-groups.tf`

- Defines the control-plane security group plus ingress/egress rules to allow communication with worker nodes within the VPC CIDR.

`node-group.tf`

- Creates a managed node group named `${cluster_name}-ubuntu-ng`.
- Uses the standard Amazon Linux 2 AMI for `t3.small` instances, runs on the supplied private subnets, and enforces `desired = min = max = 3`.
- Disk size set to 20 GiB and updates limited to one unavailable node at a time.

`variables.tf`

- Accepts inputs passed from the root module: cluster metadata, VPC/subnet info, node sizing, etc.

`outputs.tf` (if present)

- Would surface cluster and node-group attributes (not shown in snippets, but typically included).

4. Supporting Files

`s3.tf`

- Already covered above; ensures the project-specific bucket is tagged and locked down from public access.

`terraform.tfstate` / `.terraform.lock.hcl`

- Generated automatically by Terraform; lock file pins provider versions for reproducible installs.

5. Execution Flow Summary

1. **Initialize** (`terraform init`): installs the AWS provider and configures the S3 backend (prompting to migrate state if needed).

2. **Plan** (`terraform plan`): evaluates all modules, ensuring VPC resources, IAM roles, EKS cluster, node group, and S3 bucket are in sync.

3. **Apply** (`terraform apply`): provisions the full stack—networking, IAM, EKS control plane, and managed nodes.

4. **Destroy** (`terraform destroy`): tears down everything tracked in state (cluster, node group, VPC, IAM, S3 bucket, etc.)—useful for cleanup when finished.

6. Key Design Choices

- **Private Control Plane Access**: `endpoint_public_access = false` enforces access over private networking only.
- **Multi-AZ Resilience**: Two private subnets supply the EKS cluster and node group, satisfying AWS best practices for high availability.
- **Managed Node Group**: Simplifies lifecycle management (no self-managed autoscaling groups), while still allowing customization for instance size and disk capacity.
- **Remote State**: S3 backend centralizes state, preventing drift between collaborators and enabling eventual locking/DynamoDB integration if desired.

7. Next Steps / Enhancements

- Enable versioning or lifecycle rules on the S3 bucket for better state backups.
- Add DynamoDB state locking to prevent concurrent apply/destroy operations.
- Parameterize the backend bucket/key via CLI configuration or a separate backend config file if multiple environments are needed.
- Expand logging/monitoring (CloudWatch log groups, GuardDuty integration, etc.) for production readiness.

End of document.