Terraform Infra Provisioning

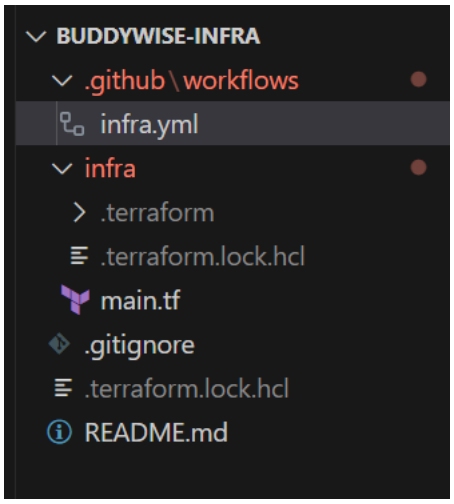
Terraform is an Infrastructure as Code (IaC) tool that allows you to provision, manage, and version your cloud infrastructure in a declarative way. In this project, Terraform is used to provision Azure infrastructure components like:

- **Resource Group**
- **Azure Container Registry (ACR)**
- **Azure Kubernetes Service (AKS)**
- **Role assignments and identity**
- **Networking (optional)**

By maintaining infrastructure in code, we achieve consistency, automation, and easy repeatability.

Terraform Directory:

You may choose to organize the Terraform infrastructure code in a separate repository (e.g., buddywise-infra) for better separation of concerns from your application code.



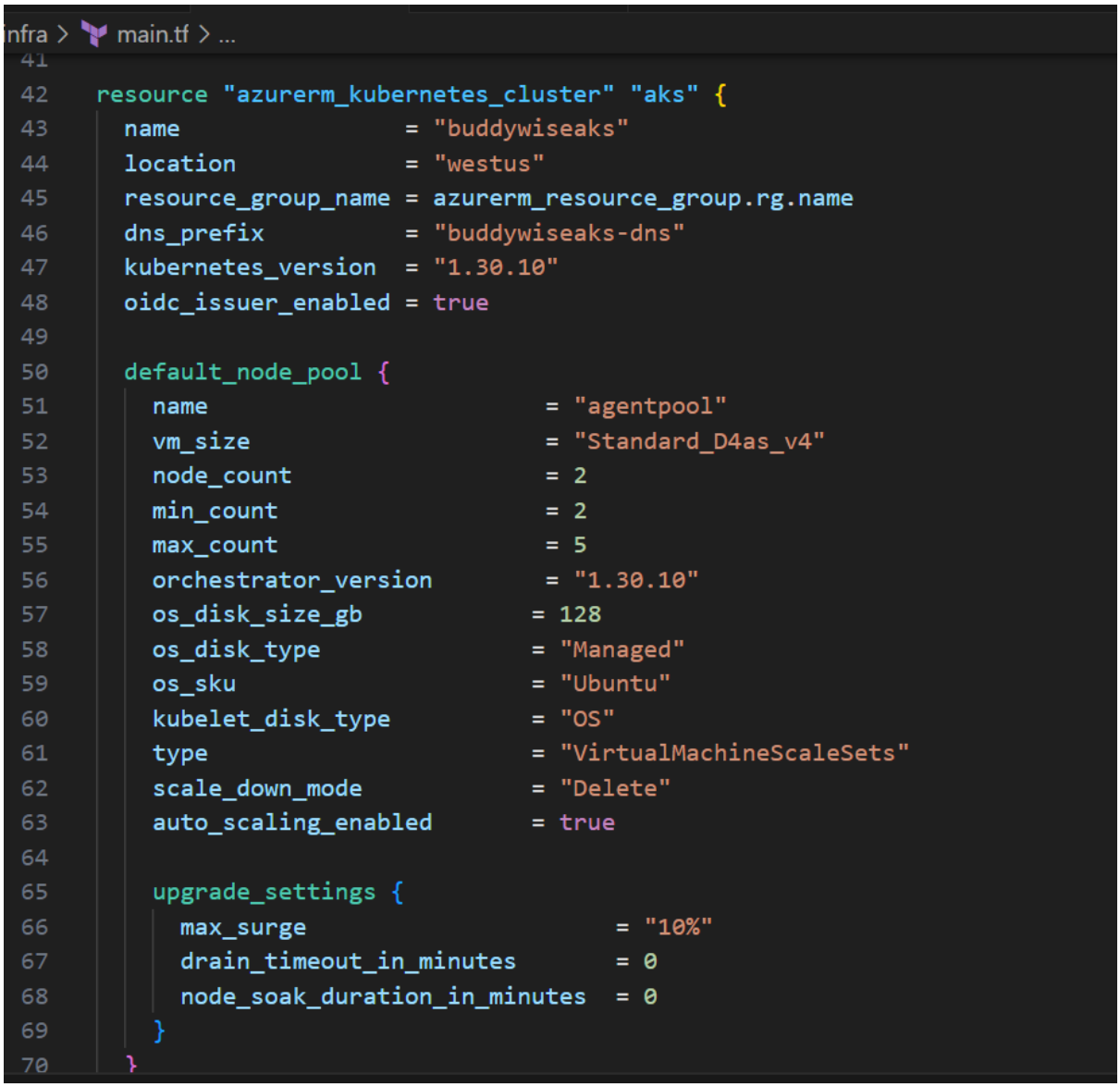
main.tf

This file defines **Azure resources** like:

- Resource Group (azurerm_resource_group)

```
resource "azurerm_resource_group" "rg" {
  name     = "buddy-wise"
  location = "westus"
}
```

- Azure Kubernetes Cluster (azurerm_kubernetes_cluster)



- Azure Container Registry (azurerm_container_registry)

```
resource "azurerm_container_registry" "acr" {
  name                = "acrbuddywise"
  location            = "westus"
  resource_group_name = "buddy-wise"
  sku                 = "Standard"

  admin_enabled                 = true
  anonymous_pull_enabled        = false
  data_endpoint_enabled         = false
  export_policy_enabled         = true
  public_network_access_enabled = true
  quarantine_policy_enabled     = false
  trust_policy_enabled          = false
  zone_redundancy_enabled       = false
  network_rule_bypass_option    = "AzureServices"

  tags = {
    environment = "production"
  }
}
```

You can modularize the code later using Terraform Modules.

Backend Configuration for State Management:

To support collaborative deployments and maintain state across environments, configure a **remote backend** using Azure Storage:

```
terraform {
  backend "azurerm" {
    resource_group_name  = "buddy-wise"
    storage_account_name = "saakstfstateprod"
    container_name       = "tfstate-prod"
    key                  = "terraform.tfstate"
  }
}
```

Ensure the storage account and blob container are created **before running terraform init**.

Commands to Deploy Infrastructure:

Navigate to the Terraform directory

cd terraform/

Initialize Terraform backend and provider plugins

terraform init

Preview infrastructure changes

terraform plan

Apply and provision resources in Azure

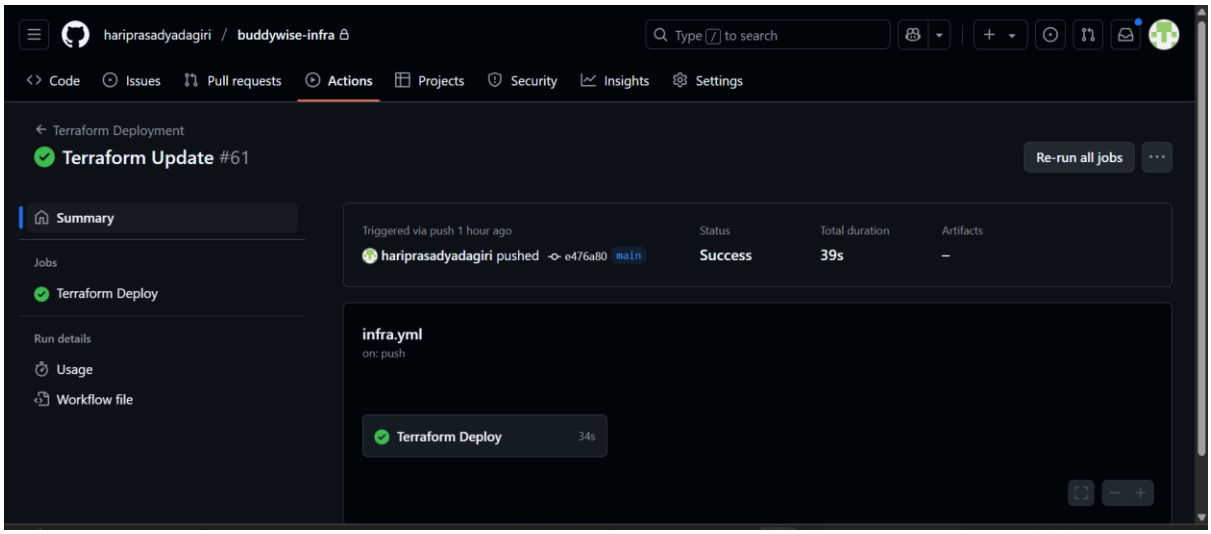
terraform apply

Use the commands below from the CLI to deploy from git

:> git add.

:> git commit -m "Deployment infra using Terraform" (you use your own massage while committing)

:> git push origin main



Verify Kubernetes Deployment:

Once the infrastructure is deployed and the GitHub Actions pipeline has run:

Before know the status for nodes and deployment status please run below command

$> az aks get-credentials --resource-group buddywise-rg --name buddywise-aks

kubectl get deployments -n prod # Verify Deployment Status

```
PS C:\Users\yadag\Buddywise> kubectl get deployments -n prod
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
myapp-backend    2/2     2            2           2d6h
myapp-frontend   2/2     2            2           2d6h
PS C:\Users\yadag\Buddywise>
```

kubectl get svc # Check service exposure and IPs

```
PS C:\Users\yadag\Buddywise> kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   3d
```

kubectl get nodes # See the Nodes Status

```
PS C:\Users\yadag\Buddywise> kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
aks-agentpool-40028103-vmss000000   Ready    <none>   3d    v1.30.10
aks-agentpool-40028103-vmss000001   Ready    <none>   3d    v1.30.10
PS C:\Users\yadag\Buddywise>
```

Best Practices & Notes

- All credentials (Client ID, Tenant ID, etc.) should be stored in **GitHub Secrets**.
- Use a **remote backend** for state management in production.
- Use az aks get-credentials to configure kubectl locally: