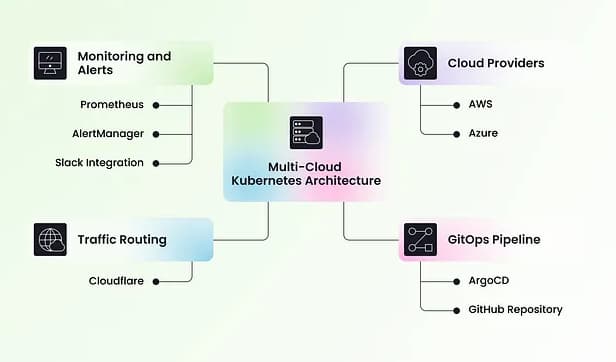
**# Multi-Cloud Kubernetes Deployment - Technical Documentation**

**Scenario**

**Imagine you’re running a global e-commerce site. Suddenly, the cloud provider goes down. This would cause chaos with angry users, lost revenue and stressed-out engineers. What’s the solution? The answer is a **strong, multi-cloud Kubernetes architecture with automatic failover.****

****

**Overview**

**I’ll explain how to build this system step-by-step. We’ll design a decentralized Kubernetes setup that uses two environments: one in **Amazon Web Services (AWS)** and another in **Microsoft Azure**. These clusters will work together, configured as either active/active or active/passive, with **Cloudflare** providing intelligent traffic routing and failover.**

**We’ll also set up a **GitOps pipeline** using **ArgoCD** so we can continuously deploy from a GitHub repository. To make sure everything runs smoothly, we’ll use **Prometheus and AlertManager** to monitor and alert you in real-time via **Slack**. By the end, you’ll have a practical, scalable, and resilient plan for dealing with outages without any downtime.**

**## 1. Introduction**

**This project implements a multi-cloud Kubernetes deployment using Terraform to set up Kubernetes clusters in AWS (EKS) and Azure (AKS). It integrates Cloudflare for failover management, ArgoCD for GitOps, and Prometheus for monitoring. The goal is to achieve high availability, automation, and monitoring for mission-critical applications.**

**## 2. Architecture Overview**

**### 2.1 Components**

**- \*\*Terraform\*\*: Infrastructure as Code (IaC) tool for provisioning cloud resources.**

**- \*\*AWS EKS & Azure AKS\*\*: Kubernetes clusters managed by AWS and Azure.**

**- \*\*Cloudflare\*\*: DNS and load balancing to manage failover across cloud providers.**

**- \*\*ArgoCD\*\*: GitOps tool to automate application deployments.**

**- \*\*Prometheus & AlertManager\*\*: Stack for monitoring and alerting.**

**### 2.2 High-Level Workflow**

**1. \*\*Provision Infrastructure\*\*: Deploy Kubernetes clusters in AWS and Azure using Terraform.**

**2. \*\*Configure Networking\*\*: Set up VPCs, subnets, and security policies for inter-cluster communication.**

**3. \*\*Set Up Cloudflare\*\*: Configure DNS and load balancing to enable automatic failover.**

**4. \*\*Deploy ArgoCD\*\*: Automate Kubernetes application deployments from a Git repository.**

**5. \*\*Deploy Monitoring\*\*: Install Prometheus and AlertManager to track system health.**

**6. \*\*Failover Testing\*\*: Simulate failures to validate that traffic reroutes seamlessly.**

**## 3. Setup Guide**

**### 3.1 Prerequisites**

**Ensure the following are installed:**

**- Terraform (>= 1.0.0)**

**- AWS CLI & Azure CLI configured with credentials**

**- Kubernetes CLI (`kubectl`)**

**- Helm**

**- ArgoCD CLI**

**- Cloudflare account with API access**

**### 3.2 Repository Setup**

**```sh**

**git clone https://github.com/your-repo/multi-cloud-k8s.git**

**cd multi-cloud-k8s**

**```**

**### 3.3 Terraform Deployment**

**#### Initialize Terraform**

**```sh**

**cd terraform**

**terraform init**

**```**

**#### Deploy Infrastructure**

**```sh**

**terraform apply -auto-approve**

**```**

**#### Retrieve Kubernetes Contexts**

**```sh**

**aws eks --region <region> update-kubeconfig --name <cluster\_name>**

**az aks get-credentials --resource-group <resource\_group> --name <cluster\_name>**

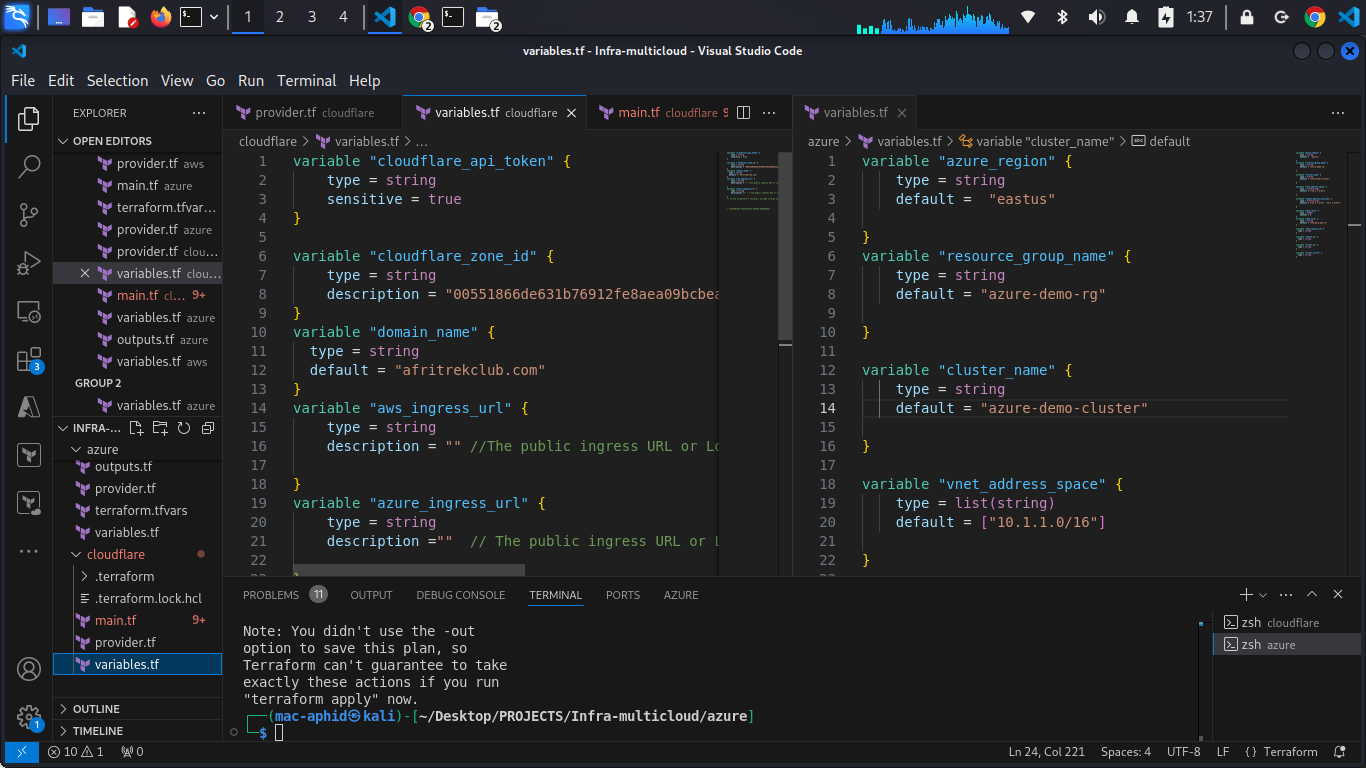
**```**

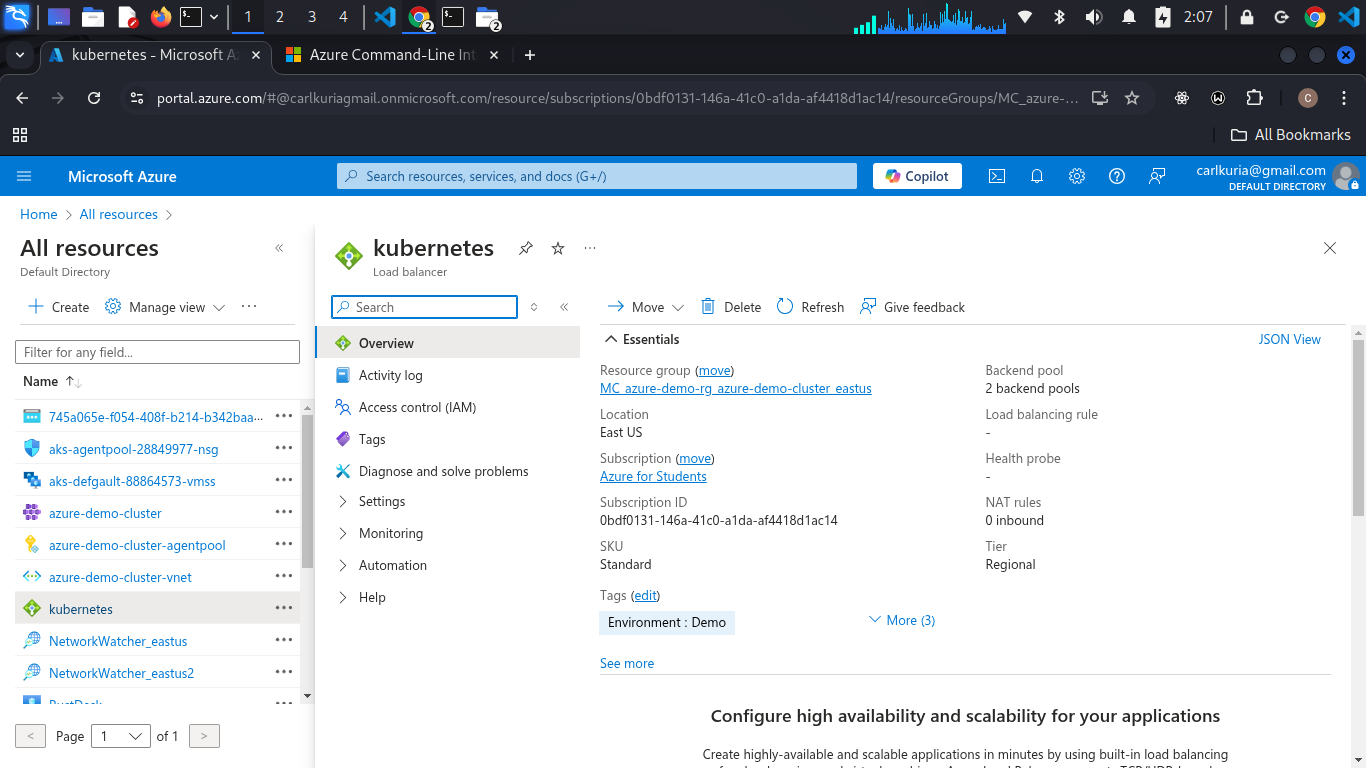
**#### Verify Clusters**

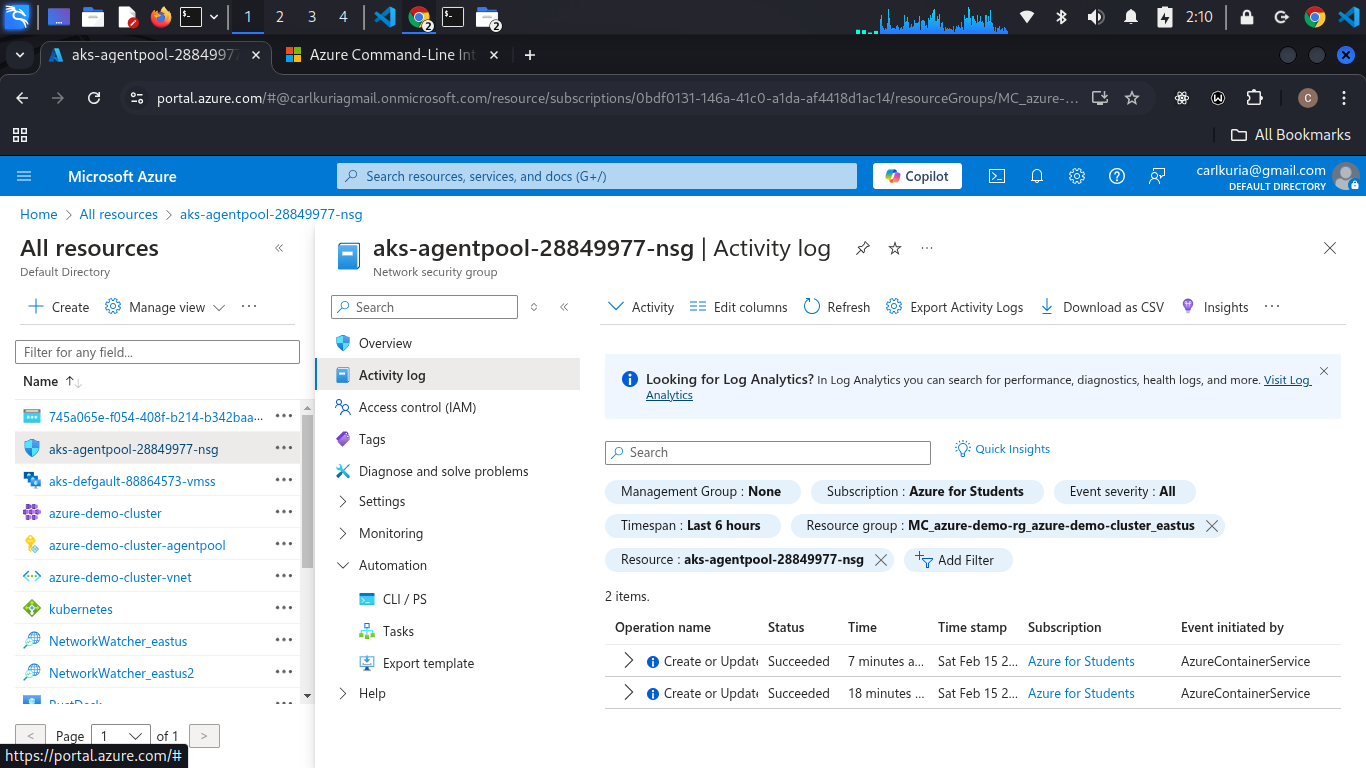
**```sh**

**kubectl get nodes**

**```**

****

****

****

**### 3.4 Configuring Cloudflare Failover**

**1. Add the external IPs of both clusters to Cloudflare DNS.**

**2. Set up load balancing rules to direct traffic to the healthiest cluster.**

**3. Configure automatic failover policies to ensure minimal downtime.**

**#### Problem Encountered: DNS Server Configuration**

**\*\*Problem\*\*: Faced difficulties in setting up the Cloudflare DNS server correctly, leading to failures in resolving traffic between clusters.**

**\*\*Solution\*\*:**

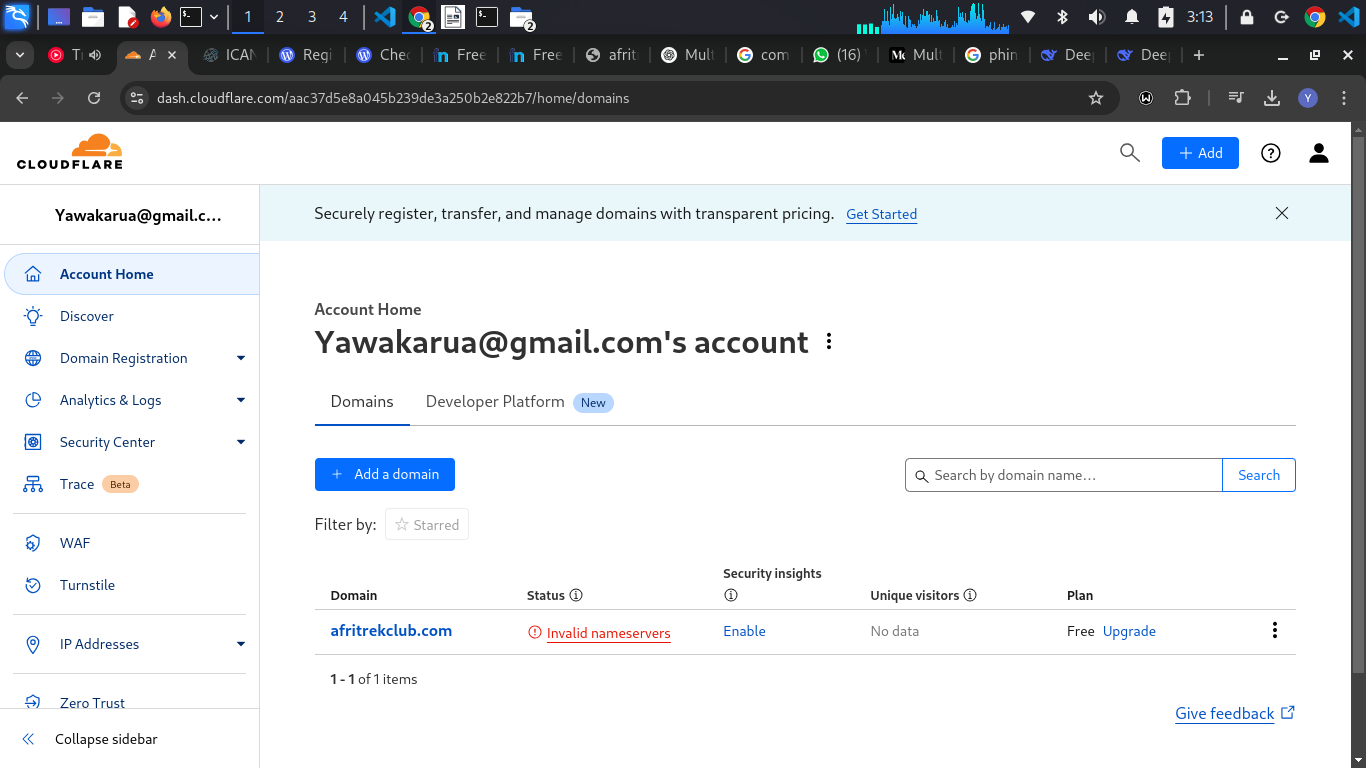
**- Verified the DNS records were correctly pointing to the external IPs of the Kubernetes clusters.**

**- Ensured that Cloudflare’s proxy mode was disabled initially to diagnose DNS resolution.**

**- Configured TTL values appropriately to prevent long cache times in case of updates.**

**- Used `dig` and `nslookup` commands to debug DNS propagation issues.**

**- Cross-checked Cloudflare API settings to confirm correct load balancing rules were applied.**

****

**### 3.5 ArgoCD Deployment**

**```sh**

**kubectl create namespace argocd**

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

**```**

**#### Access ArgoCD Dashboard**

**```sh**

**kubectl port-forward svc/argocd-server -n argocd 8080:443**

**```**

**Visit `https://localhost:8080` to log in.**

**### 3.6 Monitoring Setup**

**#### Install Prometheus Stack**

**```sh**

**helm repo add prometheus-community https://prometheus-community.github.io/helm-charts**

**helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace**

**```**

**## 4. Failover Testing**

**1. \*\*Manually take down a cluster\*\*:**

**- Simulate a failure by stopping the Kubernetes control plane or deleting nodes.**

**- Verify that Cloudflare reroutes traffic to the remaining healthy cluster.**

**2. \*\*Node Failure Test\*\*:**

**- Terminate an individual node and check if the cluster auto-scales correctly.**

**- Ensure Prometheus and AlertManager detect the failure and trigger alerts.**

**3. \*\*Application Deployment Test\*\*:**

**- Deploy an application update via ArgoCD and confirm the changes propagate correctly across clusters.**

**## 5. Problems Encountered & Solutions**

**### 5.1 Cloudflare Failover Delays**

**\*\*Problem\*\*: Initial failover took longer than expected.**

**\*\*Solution\*\*: Adjusted Cloudflare health checks to use a shorter timeout and increased polling frequency.**

**### 5.2 ArgoCD Sync Issues**

**\*\*Problem\*\*: ArgoCD was not pulling updates from Git.**

**\*\*Solution\*\*: Verified repository access permissions and manually synced the application to debug issues.**

**### 5.3 Terraform State Management**

**\*\*Problem\*\*: Terraform state file conflicts when multiple developers applied changes.**

**\*\*Solution\*\*: Used remote state storage in AWS S3 with state locking via DynamoDB.**

**### 5.4 Prometheus High Resource Consumption**

**\*\*Problem\*\*: Prometheus and AlertManager consumed excessive CPU and memory.**

**\*\*Solution\*\*: Tuned Prometheus retention settings and limited metric collection for non-essential services.**

**## 6. Business Benefits**

**- \*\*High Availability\*\*: Ensures uninterrupted service across cloud providers.**

**- \*\*Automated Deployments\*\*: ArgoCD seamlessly synchronizes code changes.**

**- \*\*Proactive Monitoring\*\*: Prometheus and AlertManager provide real-time alerting.**

**- \*\*Reduced Downtime\*\*: Cloudflare load balancing enhances reliability.**

**- \*\*Scalability\*\*: Kubernetes clusters in multiple clouds ensure resilience.**

**## 7. Conclusion**

**This multi-cloud architecture enhances operational confidence by providing failover mechanisms, automation, and monitoring. It is suitable for businesses requiring robust, scalable, and highly available infrastructure.**