End-to-End Solution for AKS Deployment and Observability

|  |  |  |
| --- | --- | --- |
| Date | Topic | Author |
| 16-10-2024 | Added project details | Siva Shankar |
| 18-10-2024 | Added architect diagrams | Siva Shankar |

### Project Index

1. **Introduction**
2. **Infrastructure Provisioning with Terraform**
   * Step 1: Setting Up Terraform for Azure
   * Components
   * Terraform Files
   * Step 2: Running Terraform
3. **Setting Up GitHub Codebases and GitHub Actions**
   * Step 3: GitHub Repository Setup
   * GitHub Actions Workflow
4. **Deploying Applications with Helm**
   * Step 4: Create Helm Charts for the Application
   * Helm Chart Files
5. **Implementing GitOps with Argo CD**
   * Step 5: Setting Up Argo CD
   * Argo CD Sync
6. **Observability with Azure Monitor, Log Analytics, and Application Insights**
   * Step 6: Set Up Monitoring
7. **Architecture Diagram**
8. **Conclusion**

# 1. Introduction

This document provides a step-by-step guide to deploying an application on **Azure Kubernetes Service (AKS)** using a variety of tools and technologies for infrastructure provisioning, continuous deployment, GitOps, and observability. Below are the key areas covered in this solution:

1. **Terraform**:
   * Provisioning Azure accounts, resource groups, and AKS clusters.
   * Configuring **SSO/OIDC** authentication and **Service Principal** setup for secure access management.
   * Automating the creation of resources, such as **Network Security Groups (NSG)** and **Virtual Networks**.
2. **AKS (Azure Kubernetes Service)**:
   * Setting up **AKS clusters** with **Role-Based Access Control (RBAC)** enabled.
   * **Azure Active Directory (AAD)** integration for identity and access management within the cluster.
3. **GitHub and GitHub Actions**:
   * Managing **GitHub repositories** for Terraform configurations, Helm charts, and Argo CD manifests.
   * Automating **CI/CD pipelines** using **GitHub Actions** for Terraform deployments and Kubernetes app releases.
4. **Helm Charts**:
   * Using **Helm** to define, package, and deploy Kubernetes applications on AKS.
   * Managing application configurations and dependencies with Helm for scalable, repeatable deployments.
5. **Argo CD for GitOps**:
   * Implementing **GitOps** with **Argo CD** to continuously synchronize the desired state in Git with the actual state of the Kubernetes cluster.
   * Leveraging declarative application management for automated rollouts and rollbacks.
6. **Observability**:
   * Integrating **Azure Monitor** for performance monitoring of the AKS cluster.
   * Setting up **Log Analytics** to collect, analyze, and query logs from the cluster and its applications.
   * Configuring **Application Insights** to provide detailed metrics and diagnostics for application health monitoring.

### 2. ****Infrastructure Provisioning with Terraform****

Terraform automates the provisioning of Azure infrastructure, ensuring repeatable and consistent environments. The setup includes creating essential resources like resource groups, AKS clusters, and authentication mechanisms.  
**Components Involved:**

1. **Azure Resource Group**:
   * **Purpose**: Logical container for Azure resources.
   * **Explanation**: All resources related to the project (VNet, AKS, Key Vault, etc.) are housed in a resource group, allowing for centralized management and access control.
2. **Virtual Network (VNet) and Subnets**:
   * **Purpose**: Networking layer for resource communication.
   * **Explanation**: VNets provide secure communication between the AKS cluster and other Azure services. Subnets are allocated within the VNet to manage and isolate traffic.
3. **Azure Kubernetes Service (AKS)**:
   * **Purpose**: Managed Kubernetes service for containerized applications.
   * **Explanation**: AKS simplifies Kubernetes management, providing features like automatic upgrades, scaling, and built-in integration with Azure services. It is provisioned with RBAC enabled for secure access and Azure Active Directory (AAD) integration for identity management.
4. **Azure Key Vault**:
   * **Purpose**: Secure storage for secrets, keys, and certificates.
   * **Explanation**: Key Vault is used to securely manage sensitive information such as Service Principal credentials, API keys, and other configuration secrets required by applications deployed in AKS.
5. **Service Principal or OIDC for Authentication**:
   * **Purpose**: Secure authentication for deploying resources.
   * **Explanation**:
     + **Service Principal**: Acts as a non-interactive identity with assigned roles to deploy Azure resources via Terraform.
     + **OIDC**: Used in conjunction with GitHub Actions to authenticate workflows securely with Azure, eliminating the need for storing secrets in plain text.
6. **Network Security Groups (NSG)**:
   * **Purpose**: Control inbound and outbound traffic at the subnet or NIC level.
   * **Explanation**: NSGs allow fine-grained control over network traffic to and from the AKS cluster, improving security and ensuring only approved traffic flows.
7. **Role-Based Access Control (RBAC)**:
   * **Purpose**: Manage access to the AKS cluster based on roles.
   * **Explanation**: RBAC ensures that users and services have appropriate permissions to perform specific actions within the cluster, enhancing security and limiting over-permissioned roles.
8. **Azure Active Directory (AAD) Integration**:
   * **Purpose**: Centralized identity management for AKS.
   * **Explanation**: AAD is integrated with AKS to provide secure, centralized user authentication and authorization, leveraging existing enterprise identities for seamless access.
9. **GitHub Actions Integration**:
   * **Purpose**: CI/CD automation for deployments.
   * **Explanation**: GitHub Actions automates Terraform provisioning and application deployments on AKS. OIDC is used for secure, token-based authentication between GitHub and Azure.
10. **Helm Charts**:
    * **Purpose**: Manage Kubernetes deployments.
    * **Explanation**: Helm is used to package and deploy Kubernetes applications to the AKS cluster, ensuring consistency and scalability. Terraform also handles Helm chart installations.
11. **Argo CD (GitOps)**:
    * **Purpose**: Continuous delivery and GitOps for Kubernetes.
    * **Explanation**: Argo CD monitors Git repositories for changes and automatically applies them to the AKS cluster, ensuring that the live state of applications matches the desired state declared in Git.
12. **Azure Monitor and Log Analytics**:
    * **Purpose**: Monitoring and log aggregation.
    * **Explanation**: Azure Monitor tracks the performance of the AKS cluster and applications, while Log Analytics collects logs for troubleshooting and operational insights. These tools provide end-to-end observability for better decision-making.

### 3. Terraform Files:

**.gitignore**: Specifies which files and directories Git should ignore.   
**aad\_group.tf**: Manages the creation of Azure Active Directory (AAD) groups, used for RBAC and access.  
**aks\_config.tf**: Configures the AKS cluster, including node pools, authentication, and network integration.  
**backend.tf**: Configures Terraform remote state management using Azure Blob Storage.  
**keyvault.tf**: Defines the Azure Key Vault configuration to securely store and manage secrets.  
**locals.tf**: Defines local values for Terraform configurations.  
**main.tf**: Defines the core components, including the Azure provider configuration.   
**monitoring.tf**: Configures Azure Monitor and Log Analytics to track the performance and health. **network.tf**: Sets up the Azure Virtual Network (VNet) and subnets that the AKS cluster.   
**outputs.tf**: Defines the outputs of the Terraform deployment.  
**providers.tf**: Configures the providers that Terraform will use, including the Azure provider.   
r**bac.tf**: Configures Role-Based Access Control (RBAC) within the Azure environment.  
**README.md**: Provides comprehensive documentation for the project.  
**role\_assignments.tf**: Assigns roles within the Azure environment, linking Azure Active Directory **service\_principle.tf**: Creates and manages Azure service principals, which are used by applications.  
**statemgmt.tf**: Configures additional state management options for Terraform.  
**terraform.tfvars**: Stores values for the variables defined in variables.tf, such as region, resource names.  
**variables.tf**: Defines input variables for Terraform, like AKS cluster name, region, and node count.

# 4. Setting Up GitHub Codebases and GitHub Actions

To implement Continuous Integration and Continuous Deployment (CI/CD) for the Terraform configuration, we will use GitHub Actions. The first step is to create a repository for the project and push the Terraform files into it. Once the files are added, a GitHub Actions workflow will be created to automate the deployment process.

1. **Create a GitHub Repository**:
   * Create a new repository in your GitHub account.
   * Clone the repository to your local machine.
   * Add all the Terraform configuration files (e.g., main.tf, network.tf, variables.tf, etc.) to the repository.
2. **Add GitHub Actions Workflow**:
   * In the root of your repository, create a directory named .github/workflows.
   * Inside the workflows directory, create a new file called ci-cd-pipeline.yml.
3. **Configure the GitHub Actions Workflow**: The GitHub Actions workflow is responsible for automating the validation and deployment of the Terraform infrastructure. The workflow will have the following steps:
   * **Linting**: Validates the Terraform code syntax using a linter like tflint to ensure best practices are followed.
   * **Validation**: Runs terraform validate to verify that the configuration files are syntactically valid and the values provided are appropriate.
   * **Plan and Apply**: Executes terraform plan to create a preview of the changes to be applied, and then applies the Terraform configuration to deploy the infrastructure on Azure.

# 5. Deploying Applications with Helm

#### ****Creating Helm Charts for the Application****

Helm is used to package and manage Kubernetes applications by simplifying deployments, upgrades, and rollbacks. A Helm chart is a collection of files that define a related set of Kubernetes resources.

1. **Create the Helm Chart**:
   * Helm organizes Kubernetes objects into a reusable package (a Helm chart). To create a Helm chart, use the following command.  
     helm create my-app
2. **Modify the Helm Chart Files**: The following key files are part of the Helm chart, and they should be customized for your application:
   * **Chart.yaml**: Contains metadata about the chart, including the name and version of the chart.

apiVersion: v2

name: my-app

description: A Helm chart for my Kubernetes application

version: 1.0.0

* + **values.yaml**: Contains the default configuration values for the chart, such as the Docker image to use, replica counts, service types, and port configurations.

replicaCount: 3

image:

repository: my-app-image

tag: "latest"

pullPolicy: Always

service:

type: LoadBalancer

port: 80

* + **templates/**: Contains Kubernetes manifests like deployment.yaml, service.yaml, and optionally ingress.yaml. These are used to define how Kubernetes resources will be created.
    - **deployment.yaml**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: {{ .Values.replicaCount }}

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

ports:

- containerPort: 80

* + - **service.yaml**:

apiVersion: v1

kind: Service

metadata:

name: my-app

spec:

type: {{ .Values.service.type }}

ports:

- port: {{ .Values.service.port }}

selector:

app: my-app

1. **Deploy the Application using Helm**: After configuring your Helm chart, you can deploy the application to your AKS cluster:

helm install my-app ./my-app

1. **Helm Upgrade and Rollback**:
   * To upgrade your application, update the Helm chart and run:

helm upgrade my-app ./my-app

* + To roll back to a previous version in case of failure:

helm rollback my-app <revision>

### ****6. Implementing GitOps with Argo CD****

Argo CD is a declarative GitOps continuous delivery tool for Kubernetes, which helps automate the synchronization of your Kubernetes cluster with your Git repository. It monitors your Helm chart repository and automatically updates the AKS cluster with changes.

1. **Install Argo CD on AKS**: To install Argo CD on the AKS cluster, follow these steps:

kubectl create namespace argocd

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

1. **Access the Argo CD UI**: To access the Argo CD UI, port-forward the Argo CD API server:

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

Then, navigate to http://localhost:8080 in your browser to access the UI. Use the default admin user with the initial password fetched from:

kubectl get pods -n argocd

kubectl get secret argocd-initial-admin-secret -n argocd --template={{.data.password}} | base64 -d

1. **Create an Argo CD Application**: To synchronize your GitHub repository (which contains your Helm chart) with your AKS cluster, define the Argo CD application in argocd-app.yaml:

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: my-app

namespace: argocd

spec:

project: default

source:

repoURL: 'https://github.com/yourusername/yourrepo'

targetRevision: HEAD

path: charts/my-app

destination:

server: 'https://kubernetes.default.svc'

namespace: default

syncPolicy:

automated:

prune: true

selfHeal: true

1. **Apply the Argo CD Application**: Once the argocd-app.yaml is ready, apply it to create the application in Argo CD:

kubectl apply -f argocd/argocd-app.yaml

1. **Sync Argo CD with the Repository**: Argo CD will automatically monitor the repository for changes. To manually trigger a sync:

argocd app sync my-app

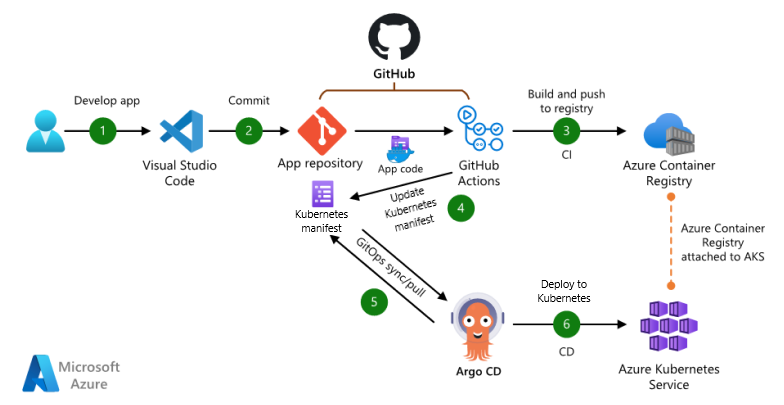
# 6. Observability with Azure Monitor, Log Analytics, and Application Insights

## Step 6: Set Up Monitoring

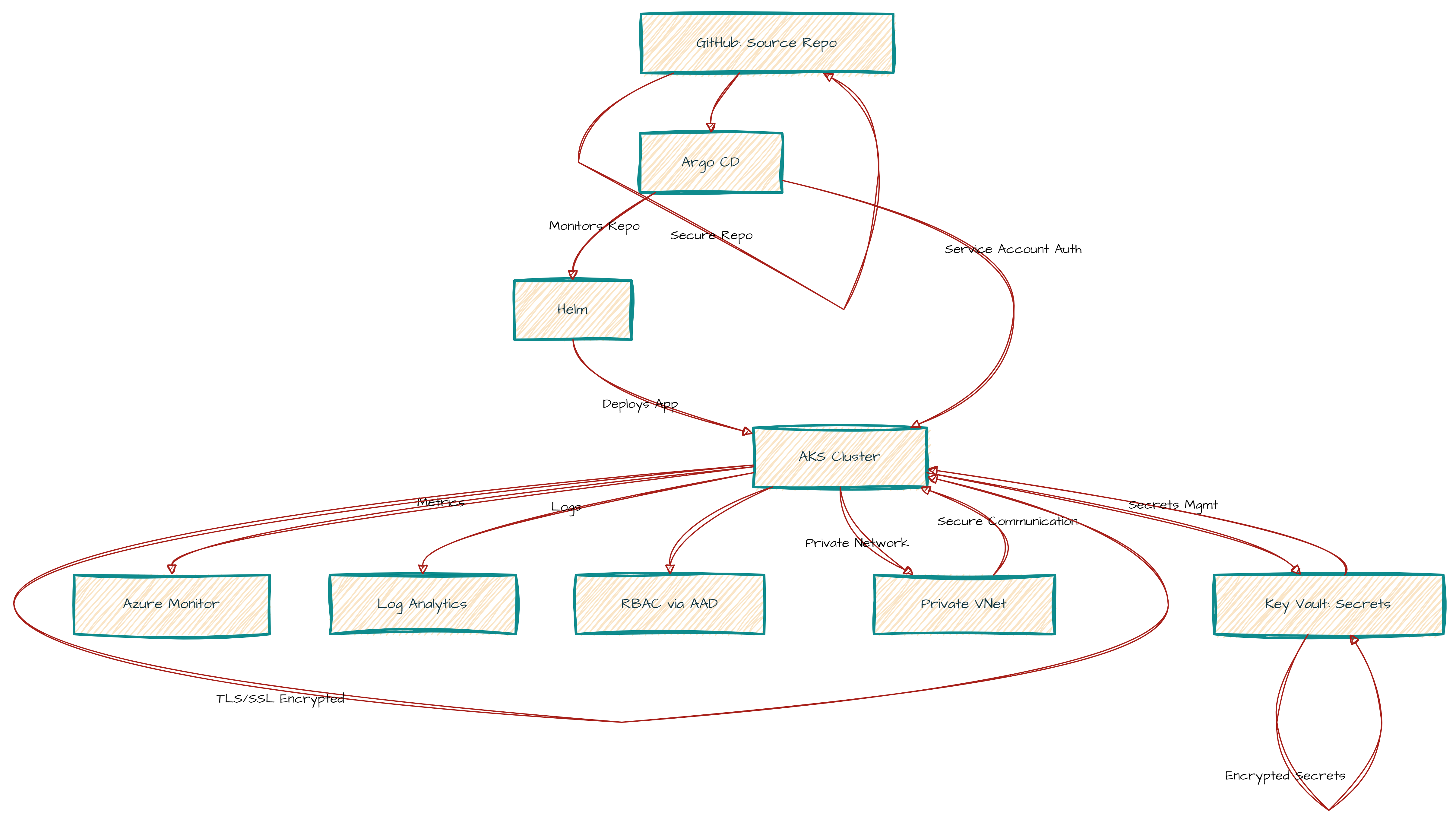
Azure Monitor: Tracks infrastructure performance.  
Log Analytics: Centralized logging for your AKS cluster.  
Application Insights: Tracks application-level metrics like latency, request rates, and failures.

Add monitoring resources to Terraform configuration:  
1. `monitoring.tf`: Configure Log Analytics and Application Insights.

# 7. Architecture Diagram



**The architecture includes:**GitHub for source control and CI/CD.  
Terraform for infrastructure provisioning.  
Helm for application deployment.  
Argo CD for GitOps and continuous delivery.  
Azure Monitor,   
Log Analytics,  
Application Insights for observability.



**Data/Authentication Flow diagram**

# 8. Conclusion

This end-to-end solution covers provisioning Azure infrastructure with Terraform, deploying applications using Helm, automating GitOps with Argo CD, and integrating monitoring solutions for observability.