## PART 1: Infrastructure Provisioning with Terraform on Azure.

* 1. **Overview.**

**AWS (Given) → Azure (Provisioned)**

| **AWS Requirement** | **Azure Equivalent Provisioned using Terraform** |
| --- | --- |
| **VPC with public and private subnets** | ✅ Azure Virtual Network (azurerm\_virtual\_network) with: - aks-subnet for AKS - postgres-subnet with VNet delegation |
| **EC2 instances or ECS/EKS for application hosting** | ✅ Azure Kubernetes Service (AKS) cluster: - azurerm\_kubernetes\_cluster with a system node pool |
| **RDS for PostgreSQL database** | ✅ Azure PostgreSQL Flexible Server: - azurerm\_postgresql\_flexible\_server with private DNS and subnet |
| **Security groups with appropriate rules** | ✅ Azure Network Security Group (azurerm\_network\_security\_group): - Allows port 443 for Kubernetes API |
| **Load Balancer for the frontend** | ✅ Azure Load Balancer (auto-created with AKS): - Type: Standard LB  - Integrated via network\_profile |

**1.2 Tools & Prerequisites**

* **Azure Subscription**
* [**Azure CLI**](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli)
* **Terraform**
* **Logged in via: az login**
  1. **Terraform Project Structure**

aks-postgres-infra/

├── main.tf → Core infrastructure resources

├── variables.tf → Input values (region, names, passwords)

├── outputs.tf → Useful output values

├── provider.tf → Azure provider configuration

├── backend.tf → Remote state backend (optional)

* 1. **Steps to Create Infrastructure.**
     1. **Resource Group.**

Creates the resource group:

|  |
| --- |
| resource "azurerm\_resource\_group" "main" {  name = var.resource\_group\_name  location = var.location  } |

**1.4.2 Virtual Network and Subnets**

**VNet with two subnets:**

* aks-subnet (for AKS nodes)
* postgres-subnet (with delegation for PostgreSQL)

|  |
| --- |
| resource "azurerm\_virtual\_network" "vnet" { |
| depends\_on = [ azurerm\_resource\_group.main ] |
| name = var.vnet\_name |
| address\_space = ["10.0.0.0/16"] |
| location = var.location |
| resource\_group\_name = var.resource\_group\_name |
| } |
| resource "azurerm\_subnet" "aks" { |
| name = "aks-subnet" |
| resource\_group\_name = var.resource\_group\_name |
| virtual\_network\_name = azurerm\_virtual\_network.vnet.name |
| address\_prefixes = ["10.0.1.0/24"] |
| } |
| resource "azurerm\_subnet" "postgres" { |
| resource "azurerm\_resource\_group" "main" {  name = var.resource\_group\_name  location = var.location  } |
| resource\_group\_name = azurerm\_resource\_group.main.name |
| virtual\_network\_name = azurerm\_virtual\_network.vnet.name |
| address\_prefixes = ["10.0.2.0/24"] |
| delegation { |
| name = "postgres-delegation" |
| service\_delegation { |
| name = "Microsoft.DBforPostgreSQL/flexibleServers" |
| actions = [ |
| "Microsoft.Network/virtualNetworks/subnets/action" |
| ] |
| } |
| } |
| } |

**1.4.3 Network Security Group for AKS**

* **Allows access to AKS control plane:**

|  |
| --- |
| resource "azurerm\_network\_security\_group" "aks\_nsg" { |
| depends\_on = [ azurerm\_resource\_group.main ] |
| name = "aks-nsg" |
| location = var.location |
| resource\_group\_name = var.resource\_group\_name |
| security\_rule { |
| name = "AllowKubeAPI" |
| priority = 100 |
| direction = "Inbound" |
| access = "Allow" |
| protocol = "Tcp" |
| source\_port\_range = "\*" |
| destination\_port\_range = "443" |
| source\_address\_prefix = "\*" |
| destination\_address\_prefix = "\*" |
| } |
| } |

**1.4.4 AKS Cluster**

**Configured with:**

* **Non-overlapping service CIDR**
* **Subnet from VNet**
* **System-assigned identity**

|  |
| --- |
| resource "azurerm\_kubernetes\_cluster" "aks" { |
| name = var.aks\_name |
| location = var.location |
| resource\_group\_name = var.resource\_group\_name |
| dns\_prefix = "aks-${var.resource\_group\_name}" |
| default\_node\_pool { |
| name = "system" |
| node\_count = 2 |
| vm\_size = "Standard\_B2s" |
| vnet\_subnet\_id = azurerm\_subnet.aks.id |
| } |
| identity { |
| type = "SystemAssigned" |
| } |
| network\_profile { |
| network\_plugin = "azure" |
| load\_balancer\_sku = "standard" |
| service\_cidr = "10.1.0.0/16" # ✅ does NOT overlap with 10.0.1.0/24 |
| dns\_service\_ip = "10.1.0.10" # ✅ must be inside the service\_cidr range |
| } |
| } |

**1.4.5 Private DNS Zone for PostgreSQL**

|  |
| --- |
| resource "azurerm\_private\_dns\_zone" "postgres\_dns" { |
| name = "privatelink.postgres.database.azure.com" |
| resource\_group\_name = azurerm\_resource\_group.main.name |
| } |
| resource "azurerm\_private\_dns\_zone\_virtual\_network\_link" "link" { |
| name = "postgres-dns-link" |
| resource\_group\_name = azurerm\_resource\_group.main.name |
| private\_dns\_zone\_name = azurerm\_private\_dns\_zone.postgres\_dns.name |
| virtual\_network\_id = azurerm\_virtual\_network.vnet.id |
| } |

**1.4.6 PostgreSQL Flexible Server**

|  |
| --- |
| resource "azurerm\_postgresql\_flexible\_server" "postgres" { |
| name                   = var.pg\_name |
| resource\_group\_name    = var.resource\_group\_name |
| location               = var.location |
| administrator\_login    = "pgadmin" |
| administrator\_password = var.pg\_password |
| sku\_name               = "B\_Standard\_B1ms" |
| storage\_mb             = 32768 |
| version                = "13" |
| delegated\_subnet\_id = azurerm\_subnet.postgres.id |
| private\_dns\_zone\_id    = azurerm\_private\_dns\_zone.postgres\_dns.id |
| public\_network\_access\_enabled = false |
| } |

**1.5 variables.tf for configurable parameters.  
 1.5.1** Using variable blocks in Terraform allows you to make your infrastructure code modular, reusable, and configurable — just like parameters in a function.

|  |
| --- |
| variable "location" { |
| default = "central india" |
| } |
| variable "resource\_group\_name" { |
| default = "octa-aks-rg" |
| } |
| variable "vnet\_name" { |
| default = "octa-aks-vnet" |
| } |
| variable "aks\_name" { |
| default = "octa-aks" |
| } |
| variable "pg\_name" { |
| default = "octadb" |
| } |
| variable "pg\_password" { |
| description = "PostgreSQL password" |
| default = "postgresql02" |
| } |
|  |

* 1. **Implement proper state management**
     1. Configure Remote Backend.  
        Create a Azure Storage account.

|  |
| --- |
| * + 1. az group create -n tfstate-rg -l eastus   az storage account create --name <uniqueacct> --resource-group tfstate-rg --location eastus --sku Standard\_LRS   * + 1. az storage container create --name tfstate --account-name <uniqueacct> |

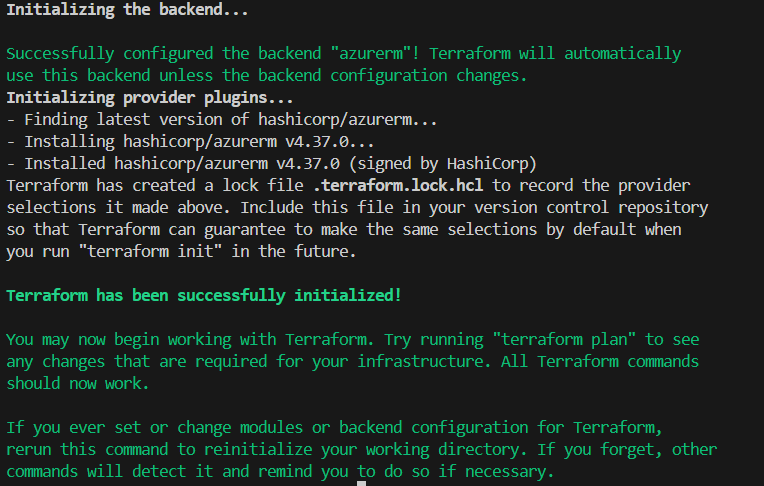
* + 1. Terraform state file is a JSON file that tracks the real-world infrastructure Terraform manages

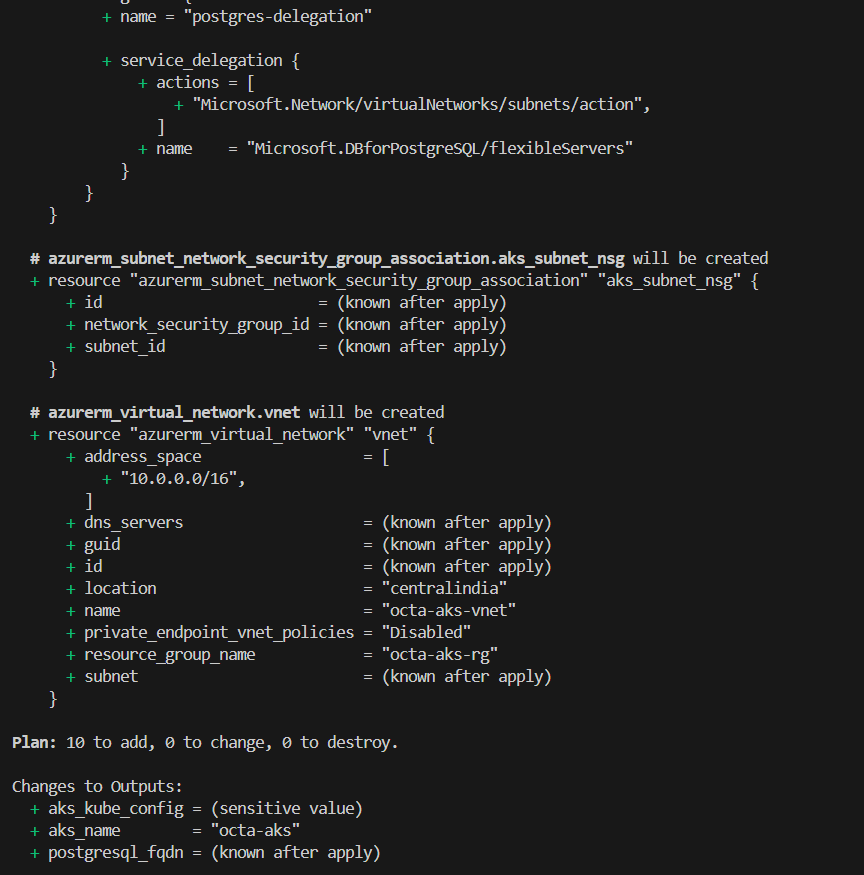
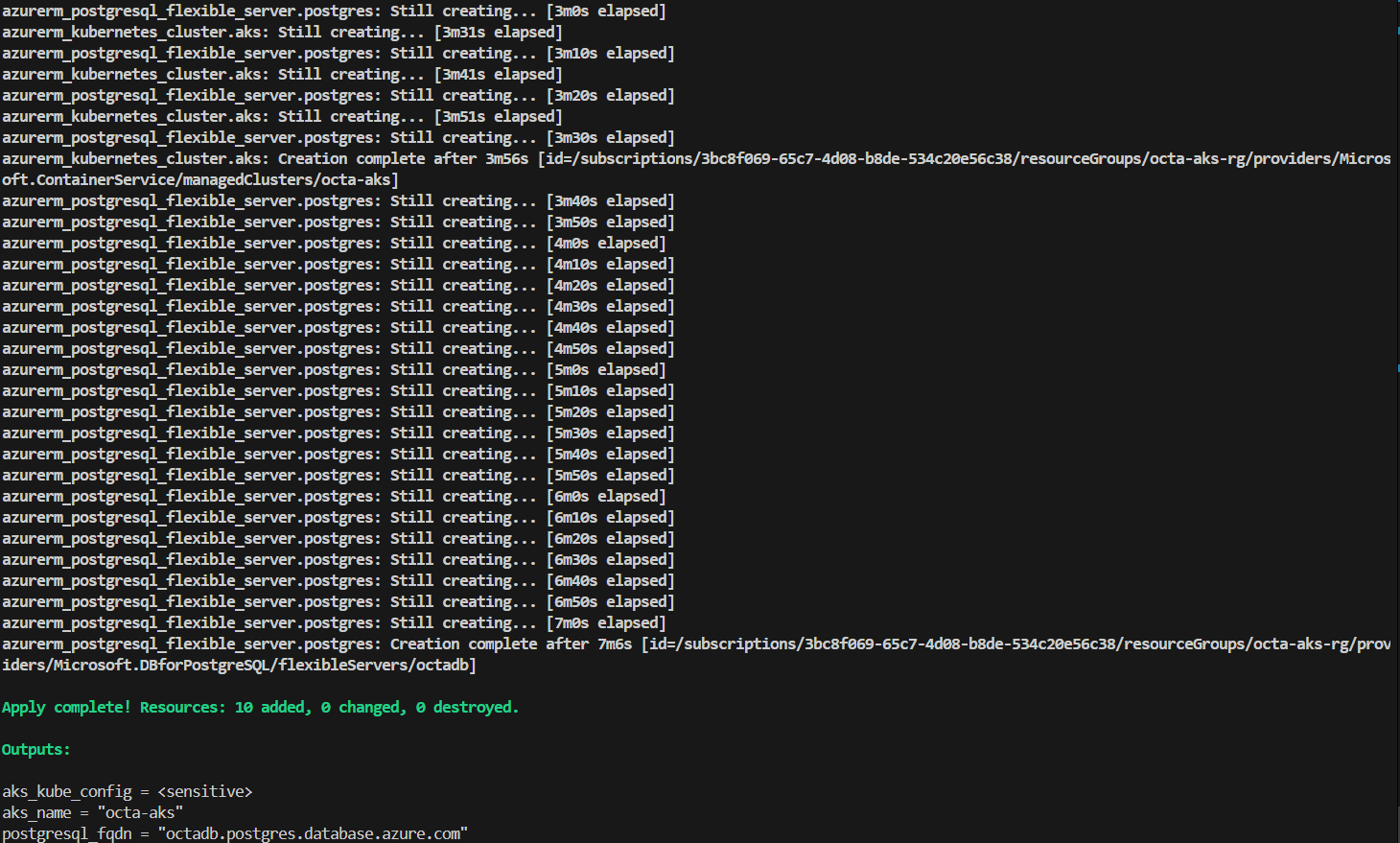
|  |
| --- |
| terraform { |
| backend "azurerm" { |
| resource\_group\_name  = "octa-rg" |
| storage\_account\_name = "tfstateaks123" |
| container\_name       = "tfstate" |
| key                  = "aks.tfstate" |
| } |
| } |

* 1. **Add outputs for key resources**
     1. The outputs.tf file defines values that Terraform will display after deployment.

|  |
| --- |
| output "aks\_name" { |
| value = azurerm\_kubernetes\_cluster.aks.name |
| } |
| output "aks\_kube\_config" { |
| value = azurerm\_kubernetes\_cluster.aks.kube\_config[0].host |
| sensitive = true |
| } |
| output "postgresql\_fqdn" { |
| value = azurerm\_postgresql\_flexible\_server.postgres.fqdn |
| } |

* 1. **Execution of Terraform**
     1. Initializes providers and sets up remote backend (if configured)
* **terraform init**

****

* + 1. Review the Plan
       - * **Terraform plan**
    2. Apply the Infrastructure
       - * **terraform apply**  
           

# 2. CI/CD Pipeline with GitHub Actions.

**2.1 Objective:**

Implement an end-to-end CI/CD pipeline that:

* Runs tests on pull requests
* Builds and pushes Docker images on merge to main
* Deploys to a staging environment
* Includes manual approval for production
* Scans for vulnerabilities
* Sends email notifications on failure

**2.2 Tools & Pre-requisites**

**2.2.1 GitHub Setup**

* A repository with your app code (React/Node)
* GitHub Actions enabled

**2.2.2 Docker**

* Dockerfile in your project root
* Azure Container Registry (ACR) created
* Docker login credentials (username/password) stored as GitHub secrets

**2.2.3 Kubernetes (AKS)**

* Staging and Production AKS clusters
* kubectl manifests for Deployment and Service inside /manifest/

**2.2.4 Secrets in GitHub**

Store the following under **Settings > Secrets and variables > Actions**:

* DOCKER\_USERNAME, DOCKER\_PASSWORD
* DOCKER\_REGISTRY (e.g., octaacrtestng.azurecr.io)
* AZURE\_CREDENTIALS (JSON for service principal)
* EMAIL\_USERNAME, EMAIL\_PASSWORD

**2.2.5 Email Alerts**

* Use Gmail App Password for EMAIL\_PASSWORD

**2.3 Project Structure.**

**├── Dockerfile**

**├── manifest/**

**│ ├── deployment.yaml**

**│ └── service.yaml**

**├── package.json**

**└── .github/**

**└── workflows/**

**└── ci-cd.yaml**

**2.4 Steps to create CICD GitHub Actions Pipeline.  
  
2.4.1 Testing on Pull Requests**

* Triggered when a PR is created against main
* Runs npm install, npm test, and optional test:integration.

|  |
| --- |
| name: CI/CD Pipeline |
| on: |
| pull\_request: |
| branches: [main] |
| push: |
| branches: [main] |
| workflow\_dispatch: |
| jobs: |
| test: |
| name: Run Unit & Integration Tests |
| runs-on: ubuntu-latest |
| steps: |
| - uses: actions/checkout@v4 |
| - name: Setup Node |
| uses: actions/setup-node@v4 |
| with: |
| node-version: 16 |
| - name: Install Dependencies |
| run: npm install |
| - name: Run Unit Tests |
| run: npm test |

**2.4.2 Build and Push Docker Image**

* Triggered on push to main
* Builds Docker image
* Tags it with commit SHA
* Pushes to Azure Container Registry.

|  |
| --- |
| build-and-push: |
| needs: test |
| runs-on: ubuntu-latest |
| steps: |
| - name: Checkout Code |
| uses: actions/checkout@v3 |
| - name: Docker Login to ACR |
| run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login octaacrtestng.azurecr.io -u ${{ secrets.DOCKER\_USERNAME }} --password-stdin |
| - name: Build Docker Image |
| run: docker build -t octaacrtestng.azurecr.io/myapptest:${{ github.sha }} . |
| - name: Push Docker Image |
| run: docker push octaacrtestng.azurecr.io/myapptest:${{ github.sha }} |
| - name: Scan Node Dependencies for Vulnerabilities |
| run: npm audit --audit-level=high || true |

**2.4.3 Security Scanning**

* npm audit to scan dependencies
* Trivy scans the Docker image for OS and library vulnerabilities
* Email alert if critical/high issues are found.

|  |
| --- |
| - name: Scan Docker Image with Trivy |
| uses: aquasecurity/trivy-action@master |
| with: |
| image-ref: 'octaacrtestng.azurecr.io/myapptest:${{ github.sha }}' |
| format: 'table' |
| exit-code: '1' |
| severity: 'CRITICAL,HIGH' |

**2.4.4 Deploy to Staging**

* Deploys the image to the AKS staging cluster
* Uses kubectl apply and set image.

|  |
| --- |
| deploy-aks: |
| needs: build-and-push |
| runs-on: ubuntu-latest |
| steps: |
| - name: Checkout Code |
| uses: actions/checkout@v4 |
| - name: Azure Login |
| uses: azure/login@v1 |
| with: |
| creds: ${{ secrets.AZURE\_CREDENTIALS }} |
| - name: Set AKS Context |
| uses: azure/aks-set-context@v1 |
| with: |
| creds: ${{ secrets.AZURE\_CREDENTIALS }} |
| cluster-name: octa-aks |
| resource-group: octa-aks-rg # ✅ Change if different |
| - name: Update Deployment Image |
| run: | |
| kubectl apply -f manifest/ |
| - name: Update Deployment Image  run: |  kubectl set image deployment/my-app my-app=octaacrtestng.azurecr.io/my-app:${{ github.sha }} -n default |

**2.4.5 Deployment to AKS**

**2.4.5.1 Manual Approval (Production)**

* Manual gate using environment: production
* Waits for approval from GitHub UI

**2.4.5.2 Deploy to Production**

* Once approved, deploys same image to production AKS
* Email alerts on any deployment failure

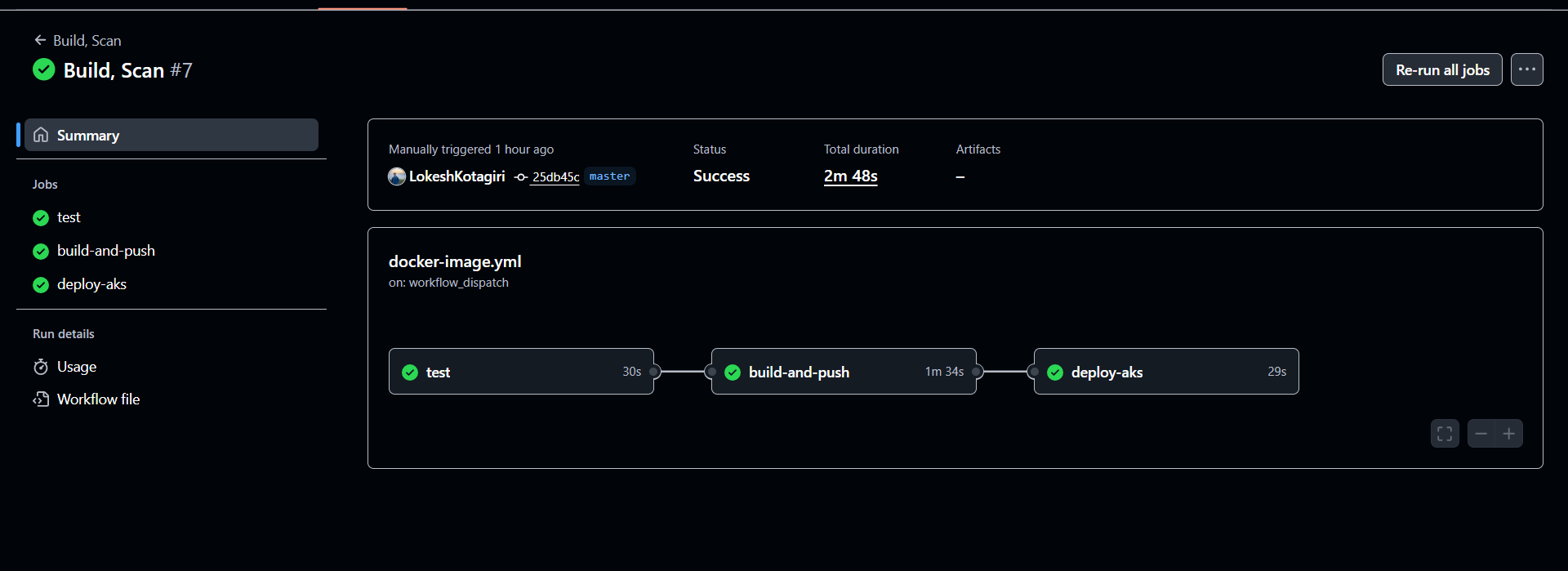
|  |
| --- |
| wait-for-approval: |
| name: Manual Approval for Production |
| needs: deploy-staging |
| runs-on: ubuntu-latest |
| environment: |
| name: production |
| url: https://your-production-url.com |
| steps: |
| - name: Await Approval |
| run: echo "Waiting for manual approval." |
| deploy-production: |
| name: Deploy to Production |
| needs: wait-for-approval |
| runs-on: ubuntu-latest |
| steps: |
| - name: Checkout Code |
| uses: actions/checkout@v4 |
| - name: Azure Login |
| uses: azure/login@v1 |
| with: |
| creds: ${{ secrets.AZURE\_CREDENTIALS }} |
| - name: Set AKS Context |
| uses: azure/aks-set-context@v1 |
| with: |
| creds: ${{ secrets.AZURE\_CREDENTIALS }} |
| cluster-name: prod-aks |
| resource-group: prod-rg |
| - name: Deploy to Production |
| run: | |
| kubectl apply -f manifest/ |
| kubectl set image deployment/my-app my-app=${{ secrets.DOCKER\_REGISTRY }}/myapp:${{ github.sha }} -n default |
|  |

**2.4.6 Email Notifications**

Sent using dawidd6/action-send-mail on:

* Test failure
* Vulnerability scan failure
* Deployment failure (staging/production)

|  |
| --- |
| - name: Send Email notification on failure |
| if: failure() |
| uses: dawidd6/action-send-mail@v3 |
| with: |
| server\_address: smtp.gmail.com |
| server\_port: 465 |
| username: ${{ secrets.EMAIL\_USERNAME }} |
| password: ${{ secrets.EMAIL\_PASSWORD }} |
| subject: ❌ Security Scan Failed - ${{ github.repository }} |
| body: | |
| The security scan for build ${{ github.sha }} failed. |
|  |
| Please review the logs in GitHub Actions for details. |
| Repository: ${{ github.repository }} |
| Workflow: ${{ github.workflow }} |
| Job: ${{ github.job }} |
| to: ktss.kotagiri@gmail.com |
| from: GitHub Security Bot <github@example.com> |

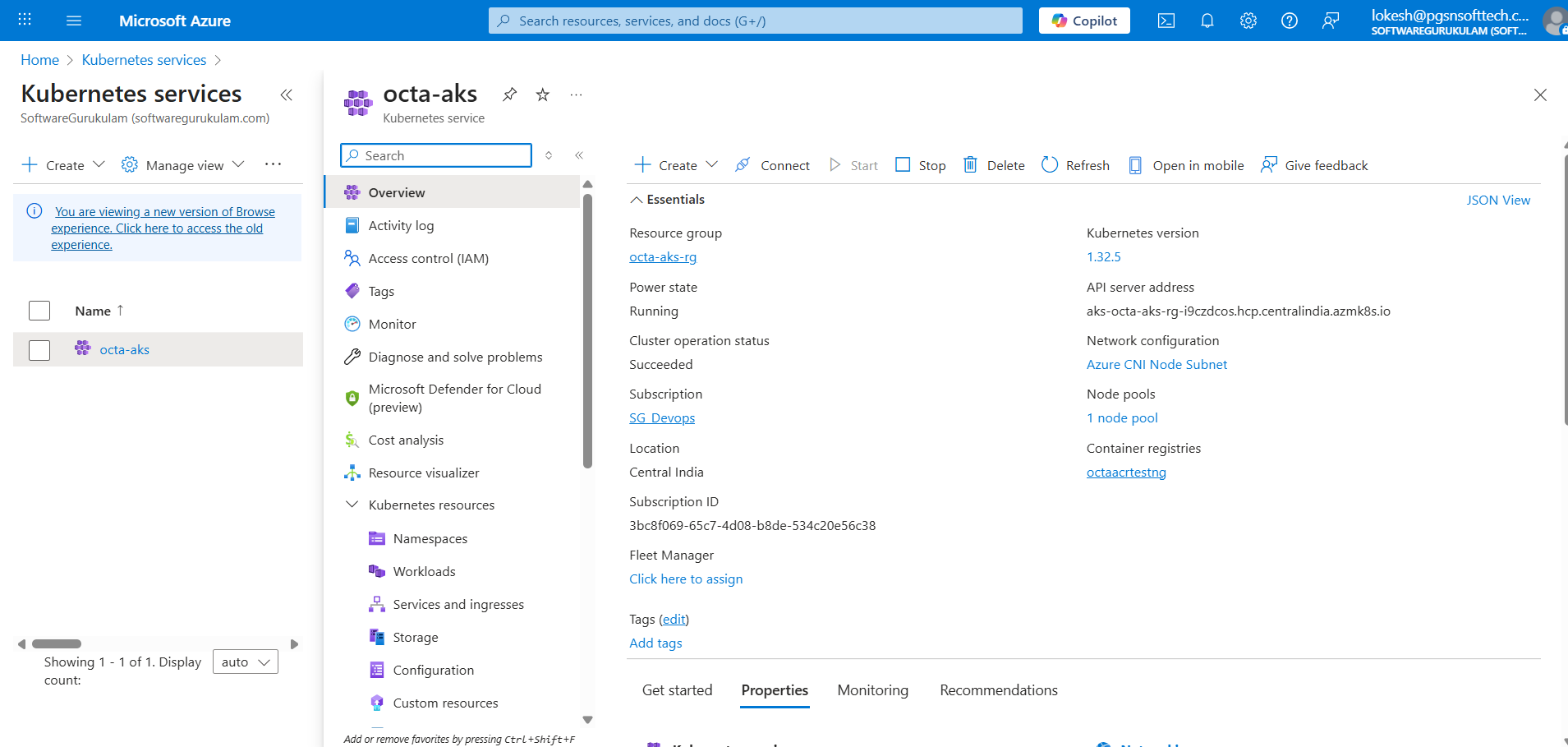
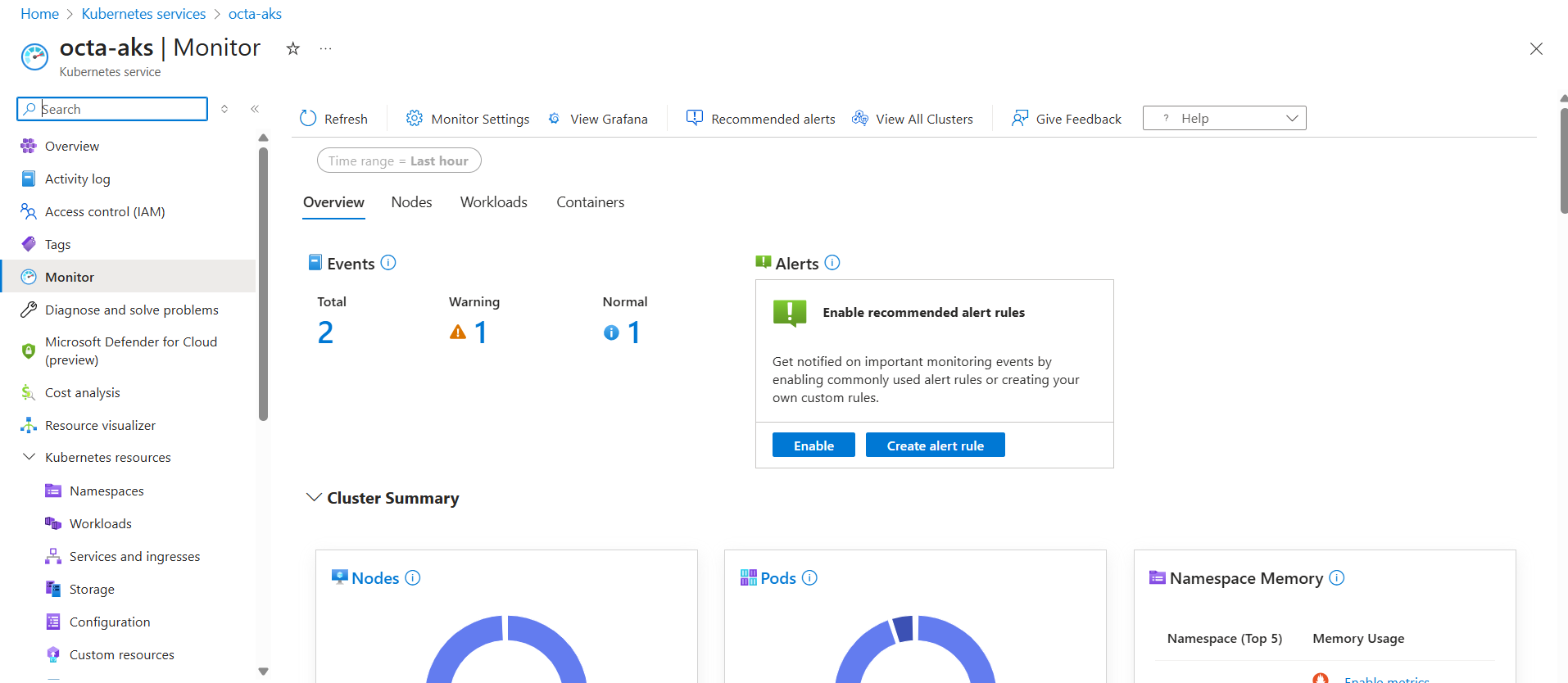
**2.4.7 Deployment Status in GUI GitHub.**  
  


3. Monitoring and Logging **Step 1: Enable Monitoring on AKS (Azure Monitor Integration)**

Azure Monitor collects:

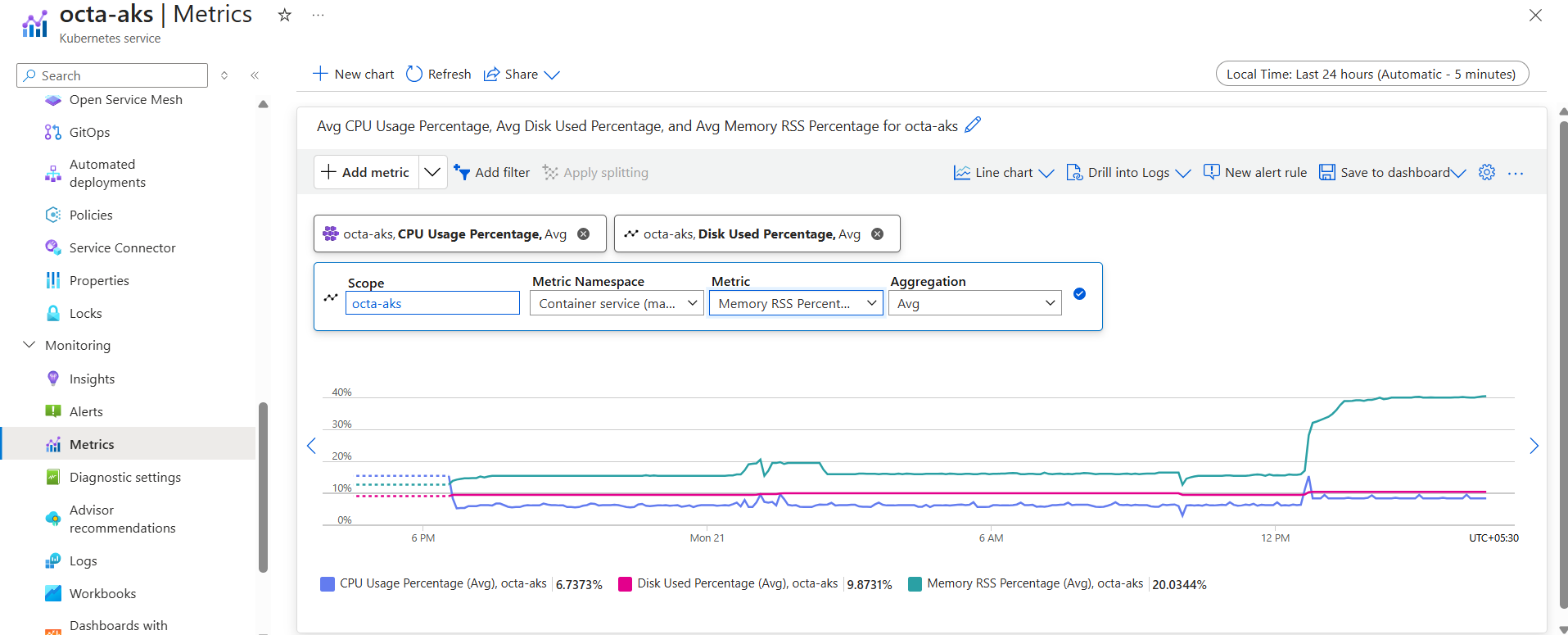
* Infrastructure metrics (CPU, memory, disk)
* Container metrics
* Logs from stdout/stderr and kubelet.

**Enable from Azure Portal**

1. Go to your AKS cluster.  
   
2. Under **Monitoring**, click **Insights**.
3. Click **Enable** and select or create a Log Analytics Workspace.  
   
4. Click **Apply**.

**Infrastructure Metrics**

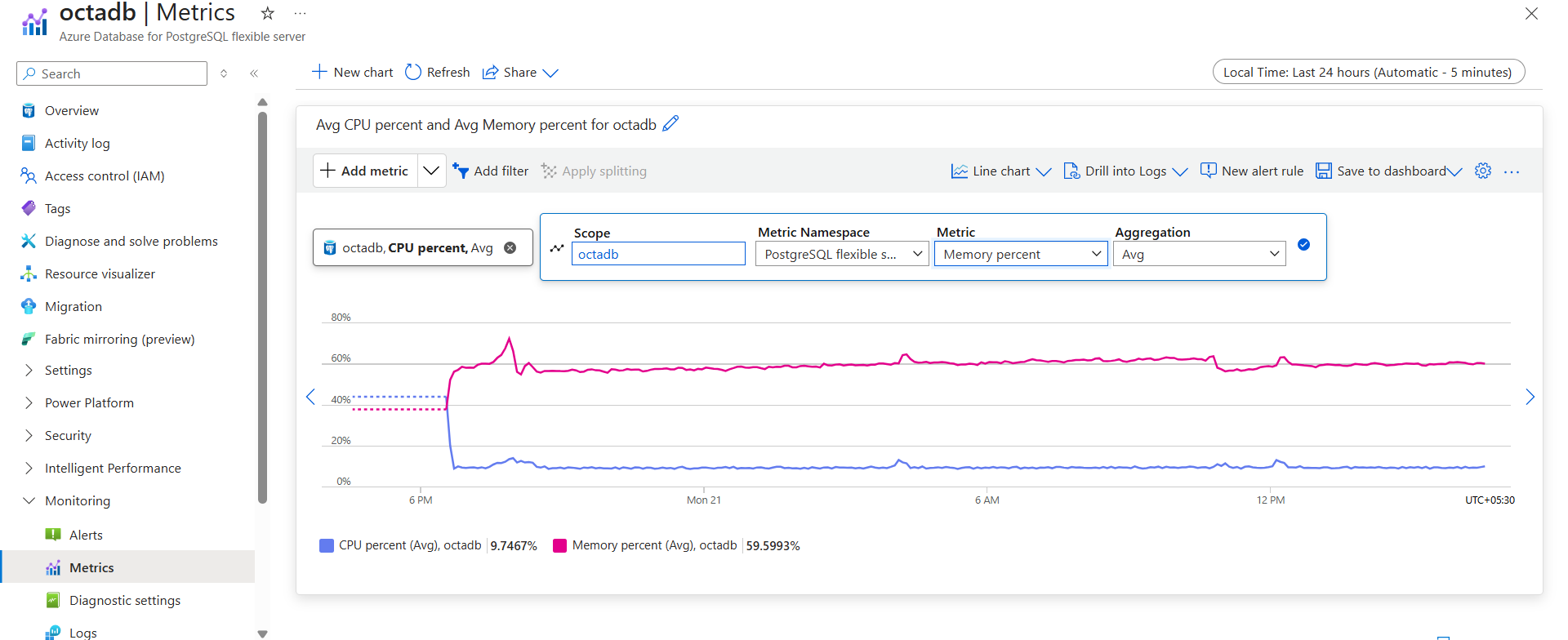
Monitor CPU, memory, and disk for AKS.



**Database Metrics**

**Azure Database Monitoring (PostgreSQL)**

* Go to **Azure Portal > Database Resource > Monitoring > Metrics**
* Track:
  + CPU/Memory
  + Connections
  + Deadlocks

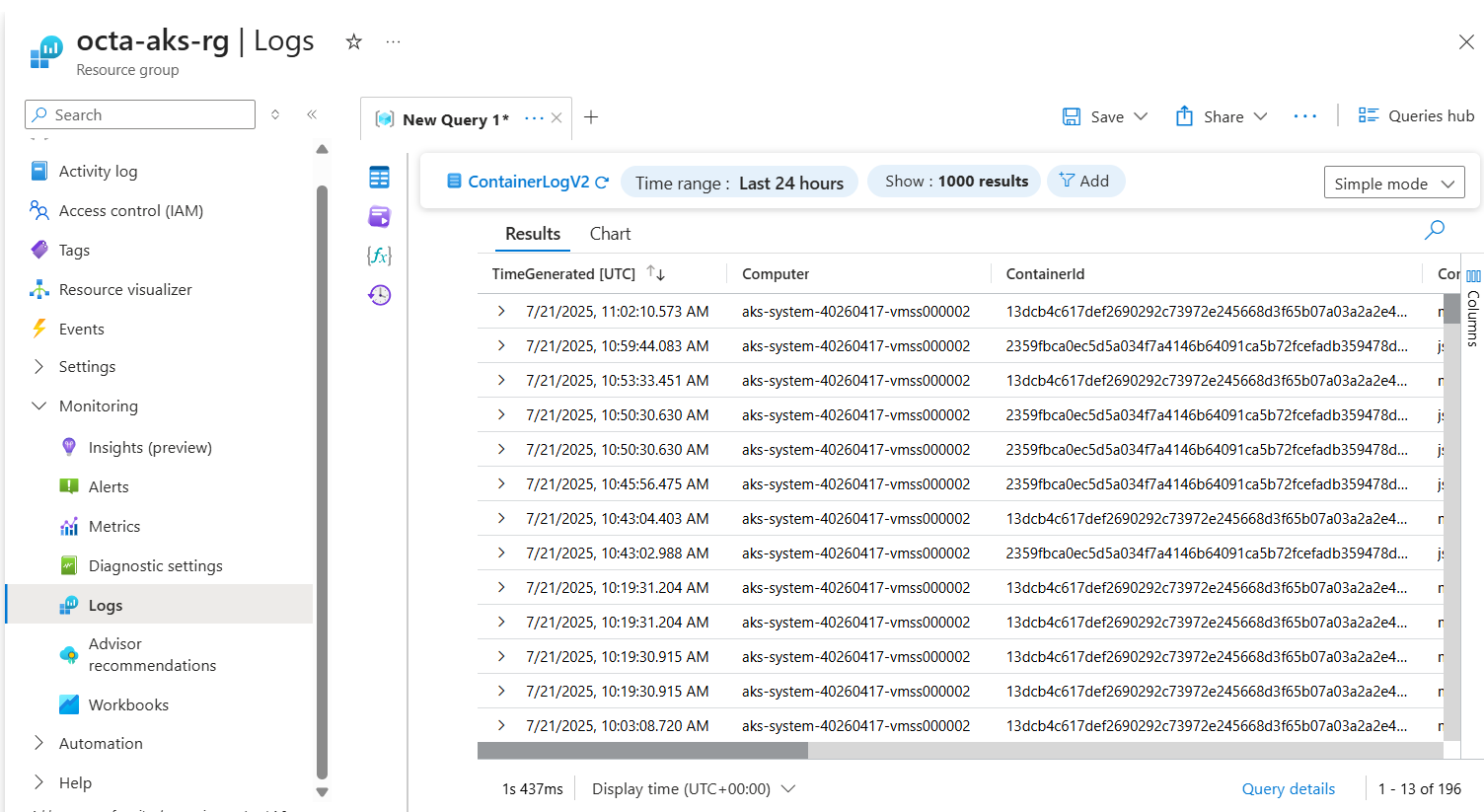


**Configure Centralized Logging**

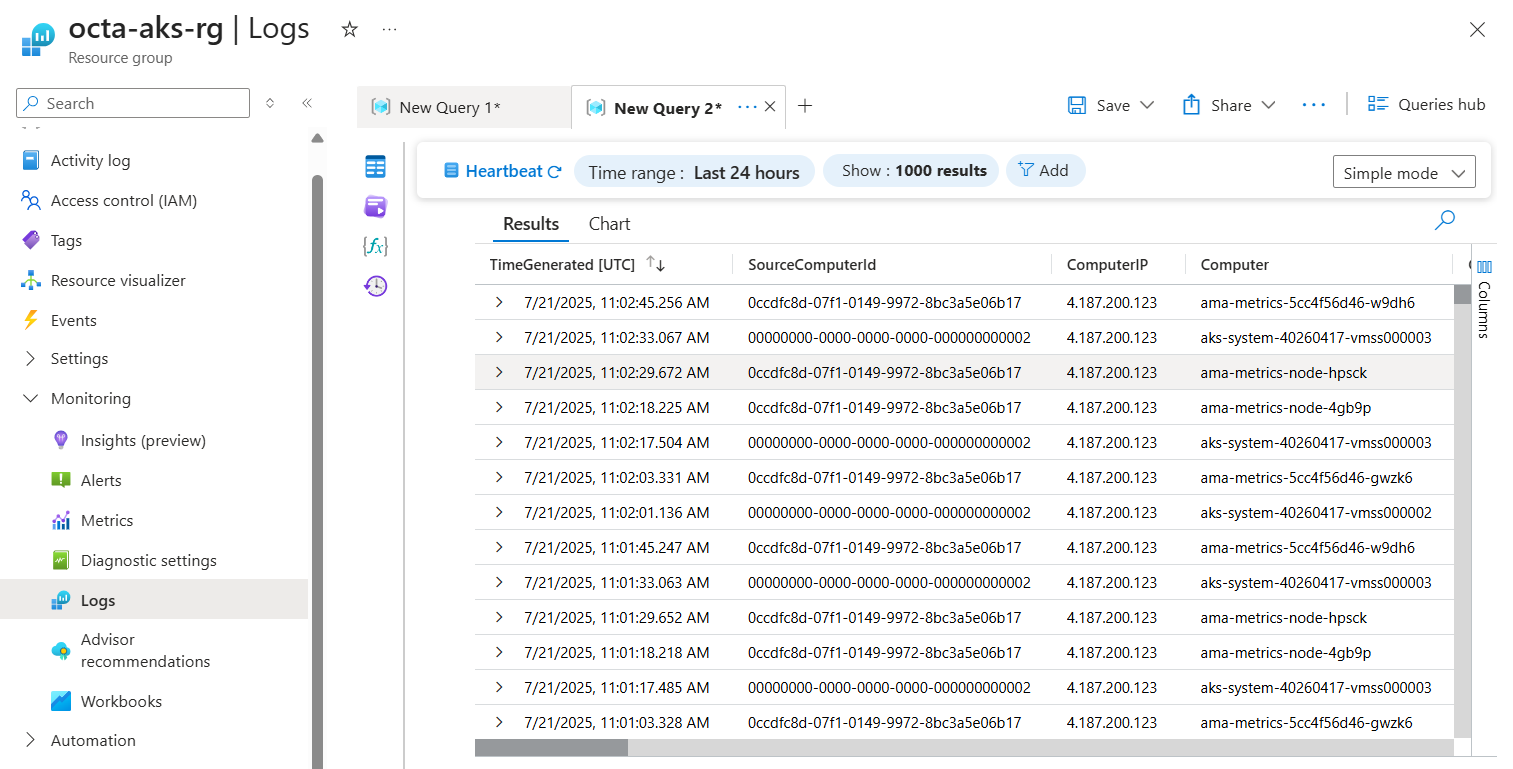
|  |  |  |
| --- | --- | --- |
| **Application Logs** | **App Insights SDK or container stdout** | **AppTraces, ContainerLog** |

|  |  |  |
| --- | --- | --- |
| **System Logs** | **From AKS nodes or VMs** | **Syslog, Heartbeat** |

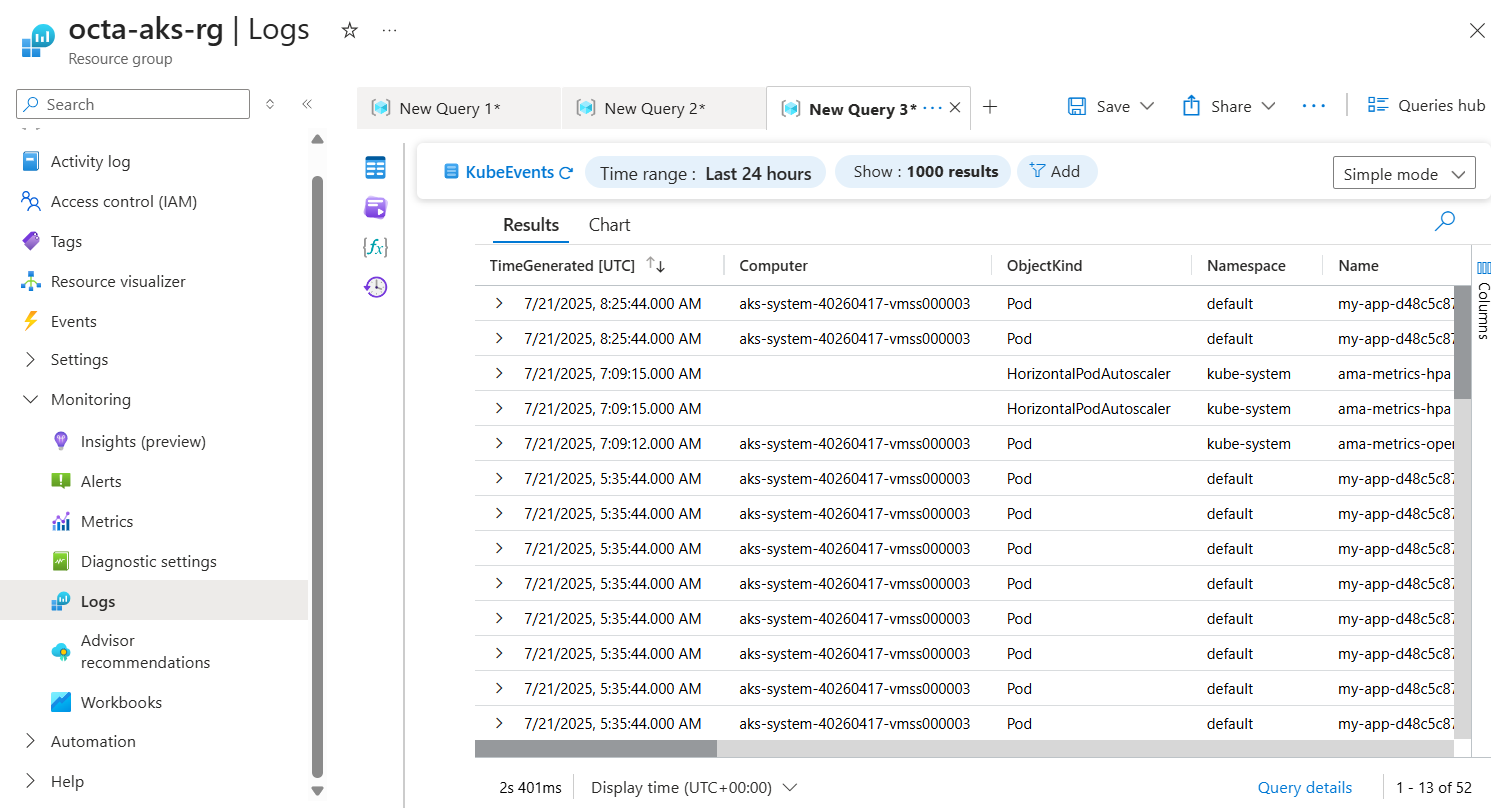
|  |  |  |
| --- | --- | --- |
| **Access Logs** | **Ingress / App Gateway diagnostics** | **AzureDiagnostics** |

**Application Logs:  
**

**System logs:**

****

**Access logs:**

****

Part 4: Documentation and Best Practices **Azure Infrastructure & CI/CD Pipeline using Terraform and GitHub Actions**

**Overview**

This project provisions a production-grade infrastructure on \*\*Microsoft Azure\*\* using \*\*Terraform\*\*, and configures a complete CI/CD pipeline using \*\*GitHub Actions\*\*. It replaces a legacy AWS setup with an Azure-native architecture including Kubernetes (AKS), PostgreSQL, and automated deployments with integrated security scanning and email notifications.

Infrastructure Setup with Terraform

**Prerequisites**

- Azure Subscription

- Azure CLI (logged in with `az login`)

- Terraform CLI

- GitHub repository with:

- Dockerfile

- K8s manifests (`/manifest`)

- `.github/workflows/ci-cd.yaml`

**Project Structure**

aks-postgres-infra/

├── main.tf

├── variables.tf

├── outputs.tf

├── provider.tf

├── backend.tf (for remote state)

**Setup Instructions**

1. \*\*Clone and Configure\*\*

git clone <repo-url>

cd aks-postgres-infra

Configure Remote Backend

az group create -n tfstate-rg -l eastus

az storage account create --name <uniqueacct> --resource-group tfstate-rg --sku Standard\_LRS

az storage container create --name tfstate --account-name <uniqueacct>

**Initialize Terraform**

terraform init

terraform plan

terraform apply

**Architecture Decisions**

|  |  |
| --- | --- |
| AWS Equivalent | Azure Provisioned |
| VPC with Subnets | Azure VNet with AKS and PostgreSQL Subnets |
| EC2 / EKS | Azure Kubernetes Service (AKS) |
| RDS (PostgreSQL) | Azure PostgreSQL Flexible Server |
| Security Groups | Azure Network Security Groups (NSG) |
| ELB (Load Balancer) | AKS-integrated Azure Standard Load Balancer |

Private Subnets: PostgreSQL is accessible only via VNet.

AKS Node Pool: Runs with system-assigned identity and isolated subnet.

Private DNS: PostgreSQL resolves only inside the VNet.

**Security Considerations**

* PostgreSQL runs with public access disabled
* AKS control plane access restricted to port 443 only
* Passwords and credentials stored as GitHub Secrets
* Manual approval required before deploying to production
* npm audit and Trivy scan all dependencies and container images

**Cost Optimization**

* Small VM size (Standard\_B2s) for AKS node pool
* PostgreSQL SKU: B\_Standard\_B1ms (burstable)
* Resource groups logically segmented
* Infrastructure auto-scales for staging and production separately

**CI/CD Pipeline (GitHub Actions)**

* Defined in .github/workflows/ci-cd.yaml

Pipeline Steps:

* Test on PR – Runs npm test on pull requests
* Build & Push Docker Image – Image pushed to Azure Container Registry (ACR)

Security Scans:

* npm audit
* Trivy for container vulnerabilities
* Deploy to Staging AKS – On merge to main
* Manual Approval for Production

Deploy to Production AKS

* Email Alerts – On test failure, vulnerability, or failed deployment
* Required GitHub Secrets
* DOCKER\_USERNAME, DOCKER\_PASSWORD
* DOCKER\_REGISTRY (e.g. octaacrtestng.azurecr.io)
* AZURE\_CREDENTIALS (JSON for service principal)
* EMAIL\_USERNAME, EMAIL\_PASSWORD (Gmail App Password recommended)

**Secret Management (Implemented)**

* Secrets are securely stored in GitHub Actions via:
* Settings > Secrets and Variables > Actions

Used in:

* + Docker login credentials
  + Azure login (AZURE\_CREDENTIALS)
  + Email credentials for failure alerts

**Monitoring & Logging**

AKS Monitoring

* Enabled via Azure Monitor & Log Analytics
* Tracks CPU, memory, container logs, and K8s metrics

PostgreSQL Monitoring

* Metrics: CPU, Memory, Connections, Deadlocks
* Enabled via Azure Metrics & Alerts

**Centralized Logging**

|  |  |
| --- | --- |
| Type | Example Tools |
| Application | stdout, App Insights |
| System | Syslog, Heartbeat |
| Access Logs | Ingress, App GW |

**Outputs (Post Deployment)**

output "aks\_name" {

value = azurerm\_kubernetes\_cluster.aks.name

}

output "aks\_kube\_config" {

value = azurerm\_kubernetes\_cluster.aks.kube\_config[0].host

sensitive = true

}

output "postgresql\_fqdn" {

value = azurerm\_postgresql\_flexible\_server.postgres.fqdn

}