* **Project Title**: Azure DevOps & Cloud Capstone Project
* **Name : jithendra.N**
* **Date Submitted: 25/06/25**
* **Mentor/Instructor** : vilas

**Agenda:**

This capstone demonstrates full DevOps lifecycle implementation using Azure DevOps, AKS, Terraform, and monitoring/security best practices. It includes multi-region HA/DR architecture, secure CI/CD pipelines, and infrastructure as code.

**3. Table of Contents**

* **Introduction**
* **DevOps Planning**
* **IaC Setup**
* **CI/CD Pipelines**
* **HA/DR & Connectivity**
* **Monitoring & Observability**
* **Security & Compliance**
* **Challenges & Troubleshooting**

**4. Introduction**

**Project Goals**

The goal of this Azure DevOps and Cloud Capstone Project is to simulate a real-world DevOps environment and implement a fully automated cloud-native solution. The project integrates modern DevOps principles, Git-based workflows, Infrastructure as Code (IaC), continuous integration and delivery pipelines, secure multi-region deployments, and disaster recovery strategies in Azure.

This capstone aims to:

* Design and deploy a production-grade architecture using Azure.
* Implement a complete CI/CD pipeline using Azure DevOps.
* Automate infrastructure provisioning using Terraform.
* Configure secure access, networking, and monitoring.
* Ensure high availability and disaster recovery through multi-region deployments.

**Scope and Objectives**

This project is designed to provide hands-on experience in:

* Managing separate codebases for frontend, backend, and infrastructure using Git.
* Creating Azure DevOps pipelines to build, test, scan, and deploy applications to AKS clusters.
* Provisioning AKS, ACR, VNets, and other Azure resources using reusable Terraform modules.
* Setting up ingress-based application routing, DNS with Traffic Manager, and VPN connectivity.
* Configuring monitoring, alerting, and observability with Azure Monitor and Application Insights.
* Implementing secure infrastructure with Azure Key Vault, NSGs, Defender, and Azure Policy.
* Defining and tracking SLO, SLA, and SLI metrics and simulating failover scenarios.

**Tools and Technologies Used**

| **Category** | **Tools/Technologies** |
| --- | --- |
| Cloud Platform | Microsoft Azure |
| Version Control | Git, Azure Repos |
| CI/CD | Azure DevOps Pipelines |
| Infrastructure as Code | Terraform |
| Containerization | Docker |
| Container Orchestration | Azure Kubernetes Service (AKS) |
| Registry | Azure Container Registry (ACR) |
| Monitoring & Logging | Azure Monitor, Log Analytics, Application Insights |
| Security | Azure Key Vault, Network Security Groups (NSGs), Azure Policy |
| Networking | Azure VNets, Azure Bastion, Site-to-Site VPN, Azure Traffic Manager |
| Code Analysis | SonarQube, Snyk, Trivy |
| App Stack | React (Frontend), Spring Boot (Backend) |

**5. DevOps Strategy & Planning**

**DevOps Culture and Values**

The project promotes a DevOps culture centered around automation, collaboration, and continuous delivery. Key values include:

* Automation of infrastructure provisioning, builds, testing, and deployments.
* Monitoring and observability integrated throughout the lifecycle.

**Comparison: DevOps vs Traditional SDLC**

|  |  |  |
| --- | --- | --- |
| Feature | Traditional SDLC | DevOps |
| Infrastructure Provisioning | Manual | Automated (IaC - Terraform) |
| Feedback Loops | Delayed | Immediate |
| Monitoring | After production issues | Continuous monitoring & alerts |

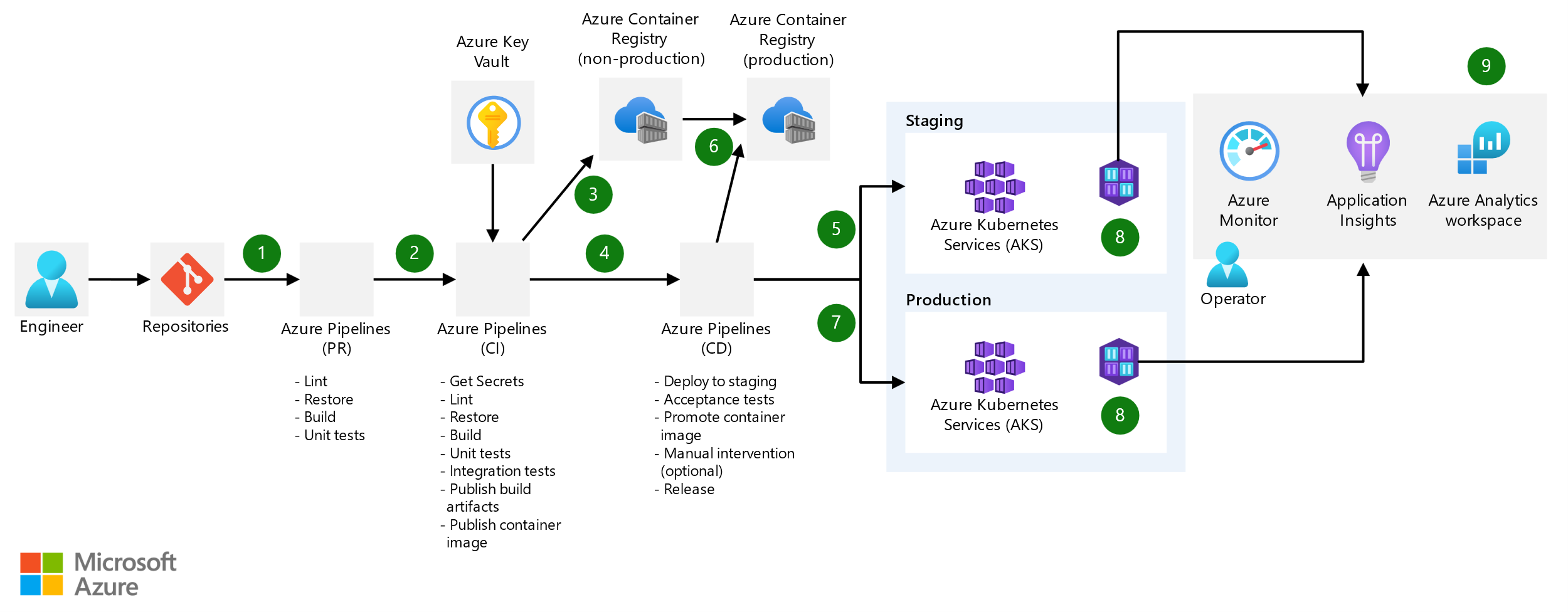
**Defined KPIs**

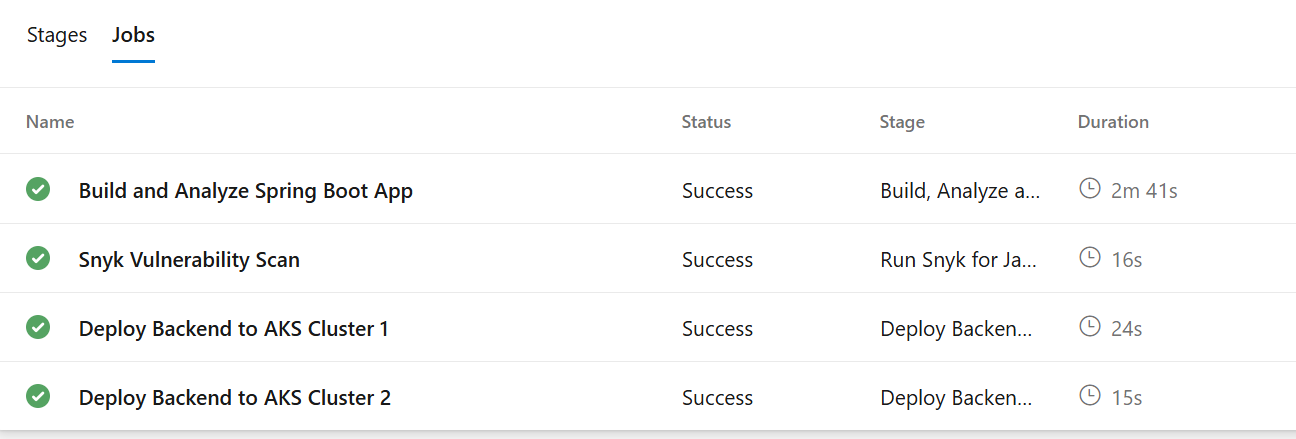
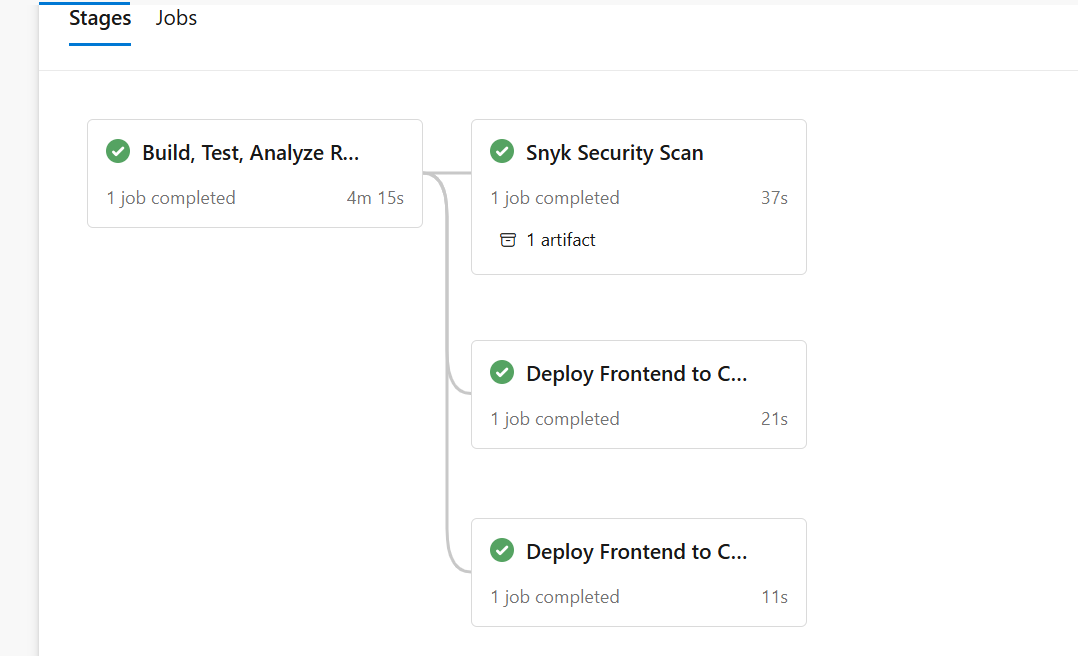
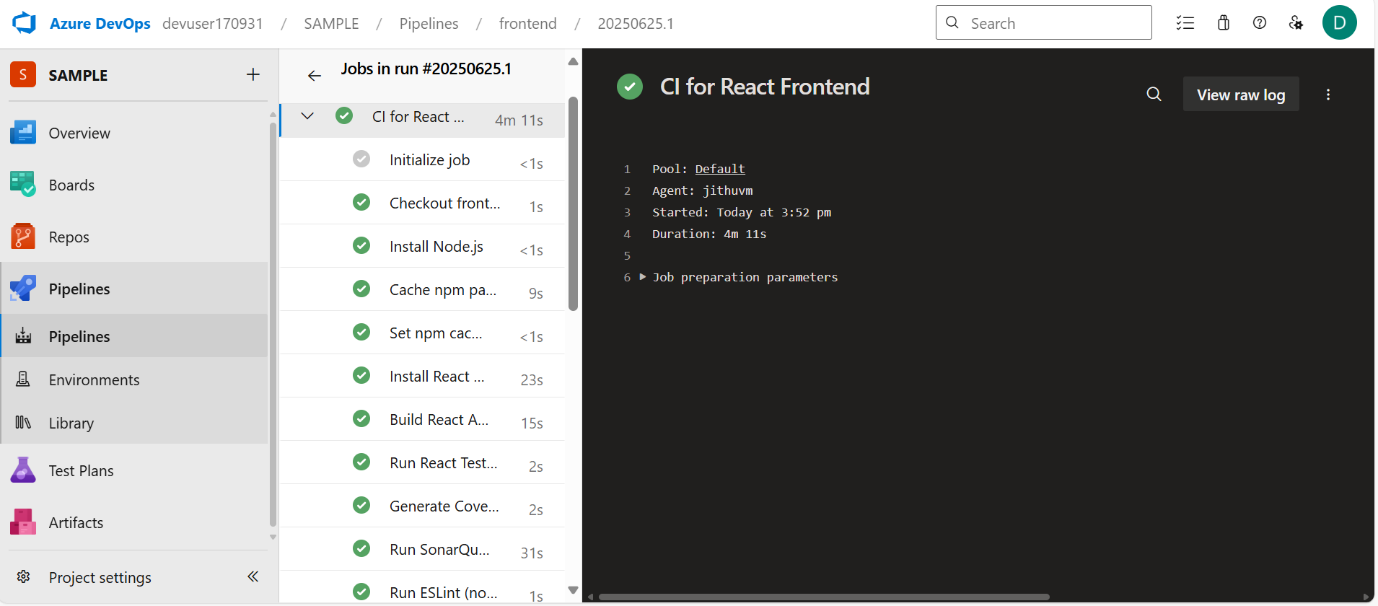
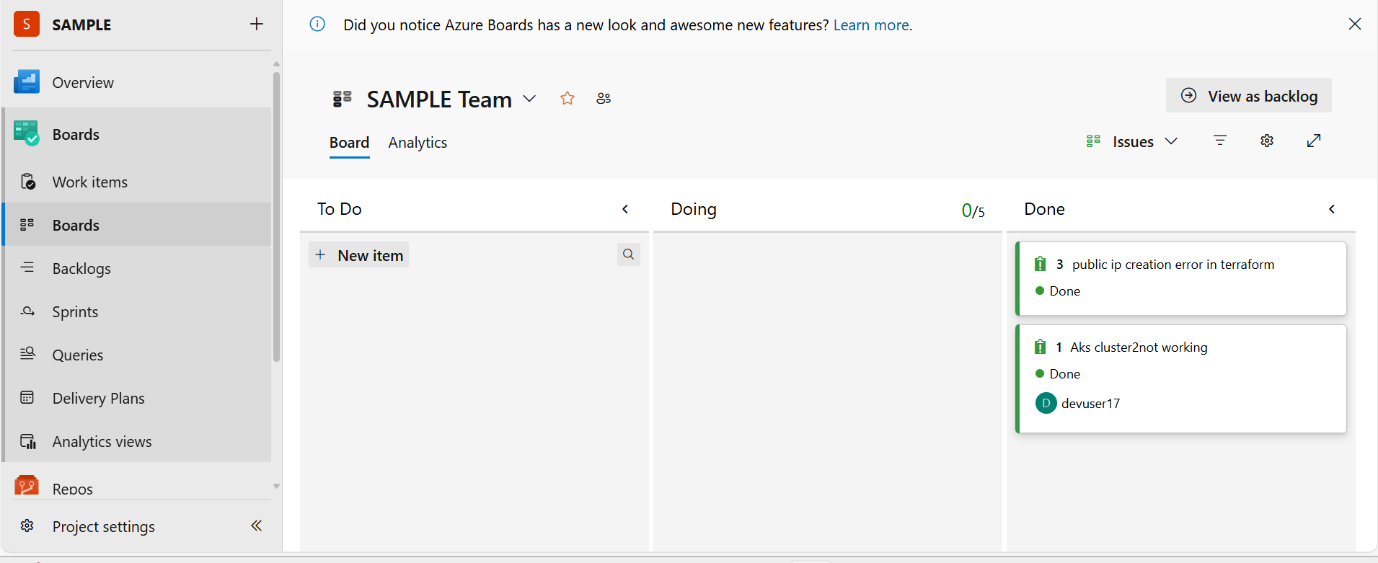
To measure DevOps effectiveness, the following KPIs were tracked:

* **Deployment Frequency**: Multiple times per day (via pipelines)
* **Change Failure Rate**: Minimized using automated testing and code scans
* **Lead Time for Changes**: Typically minutes to hours
* **Mean Time to Recovery (MTTR)**: Reduced through observability and DR strategies

**Azure DevOps Boards / Releases / Repos Structure**

* **Azure Boards**: Work items were created for user stories, tasks, and bugs. Linked with PRs for traceability.
* **Azure Repos**: Three repositories: frontend, backend, and infrastructure (Terraform).
* **Azure Pipelines**: YAML-based CI/CD pipelines defined in each repo.
* **Release Pipelines**: Deploying to multiple AKS clusters.





**6. Infrastructure as Code**

**Tools Used**

* **Terraform** for provisioning all infrastructure.
* **Azure CLI** for authentication and automation support within pipelines.

**Resource Breakdown**

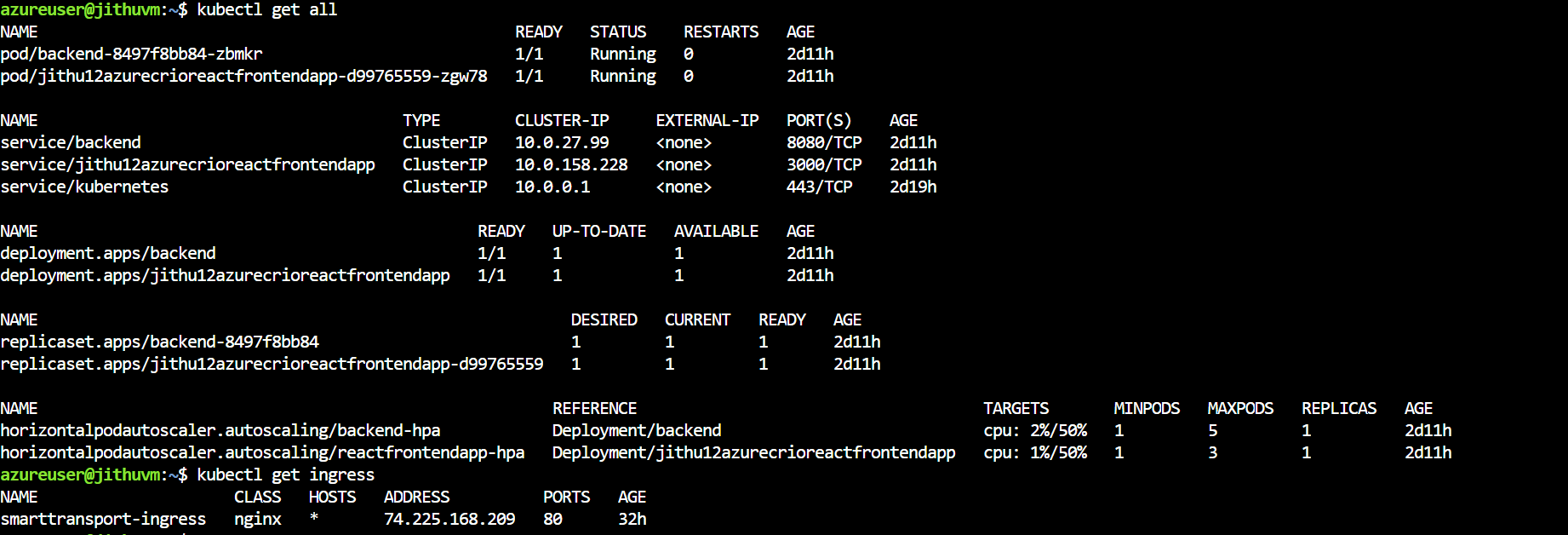
* **AKS Clusters (x2)**: Deployed in two regions to support active-passive DR architecture.
* **Azure Container Registry (ACR)**: Used to store and serve Docker images securely.
* **VNets/Subnets**: Multiple VNets created across regions with peering configurations.
* **Azure Bastion Host**: Deployed in a VNet to securely connect to VMs.
* **Windows VM**: Deployed in a separate VNet to simulate an on-prem environment.
* **Site-to-Site VPN**: Connected simulated on-prem VNet (with Windows VM) to Azure VNet using:
  + Virtual Network Gateway in Azure
  + Local Network Gateway configured with in azure itself to show them
  + Shared key authentication and IPSEC tunnel

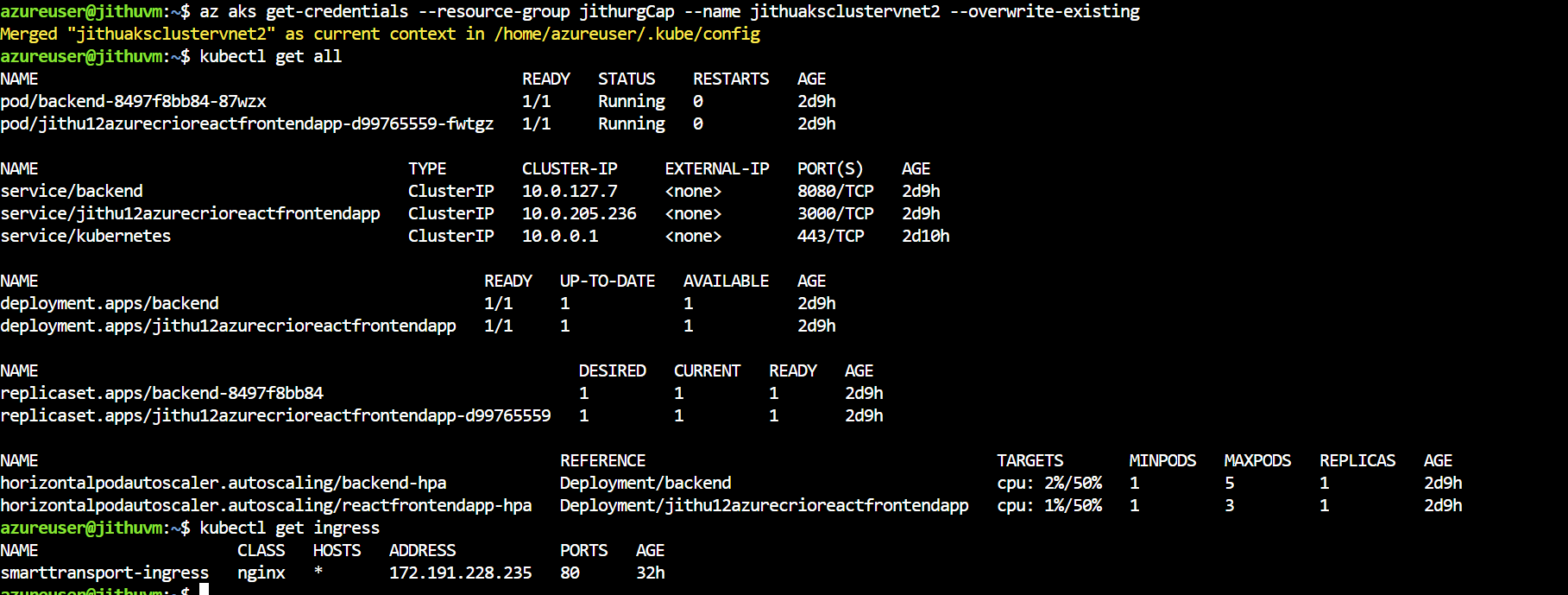
**Disaster Recovery (DR) Setup**

* **Active Cluster**: AKS in East US region
* **Passive Cluster**: AKS in West US region
* **Traffic Manager**: Configured to route traffic between the two clusters based on availability
* **Failover Simulation**: Validated switching to passive cluster in case of failure of the active one

**Diagrams Included:**

* **Architecture Diagram**: Showing AKS clusters, ingress IPs, Traffic Manager, VNet peering, site-to-site VPN

****

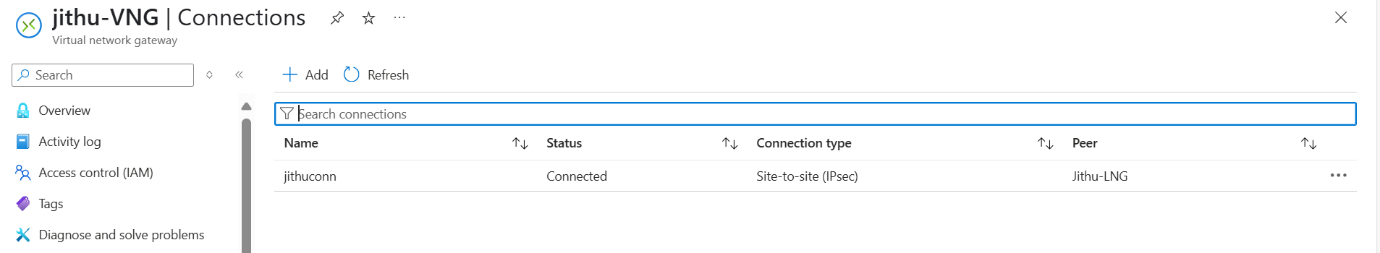
****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

****

* **Terraform Folder Structure**: Modules separated by resource (aks, networking, vpn, etc.) and environment

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**7. CI/CD Pipeline Architecture**

**Tools Used**

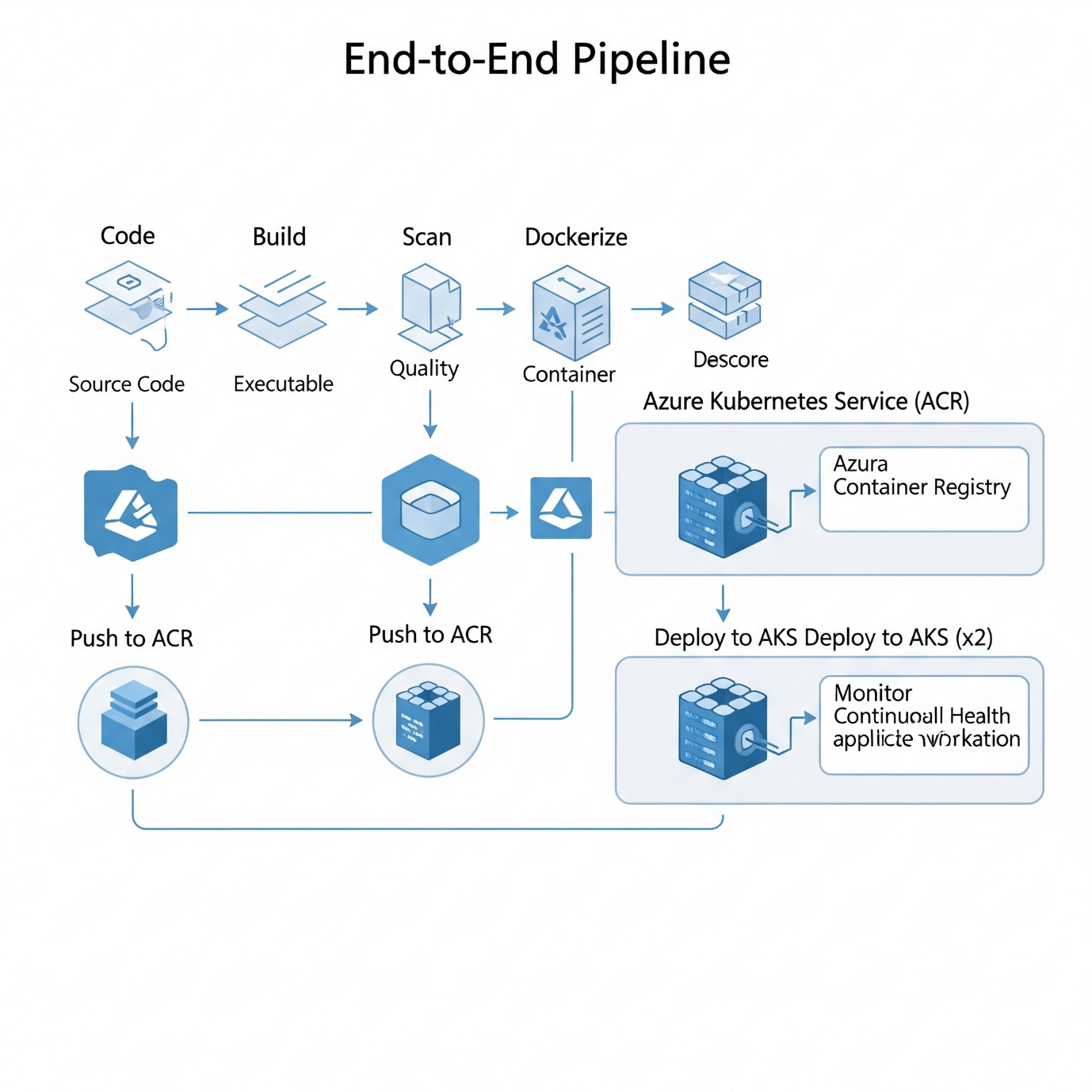
* **Azure DevOps Pipelines** for all CI/CD processes.

**Self-hosted Runners**

* **Microsoft-hosted agents** used in most pipelines. Self-hosted agents were explored for scalability but not used.

**Pipeline Breakdown**

* **App Build**:
  + Frontend: Built using npm ci for React app.
  + Backend: Built using mvn clean install for Spring Boot app.
* **Image Build**:
  + Dockerfiles were used to build images for frontend and backend.
  + Images were tagged with commit hash and pushed to Azure Container Registry (ACR).
* **Security & Quality Scans**:
  + **SonarQube**: Static code analysis.
  + **Snyk**: Dependency vulnerability scanning.
  + **Trivy**: Container vulnerability scanning post Docker build.
* **Deployment to AKS**:
  + Kubernetes manifests applied to both clusters (region A and B) via kubectl apply.
  + Each cluster used separate ingress resources to expose apps using static ip that I created in the terraform cluster and attached that to the ingress.
  + Deployments were made to /api/employees in backend and / in frontend.



**8. High Availability / Disaster Recovery**

**Architecture: Active-Passive AKS Clusters**

Two separate Azure Kubernetes Service (AKS) clusters were provisioned in different regions to support high availability and disaster recovery:

* **Active AKS Cluster**: Hosts live traffic in the primary region.
* **Passive AKS Cluster**: Synchronized deployment in secondary region, idle unless failover occurs.

**Azure Traffic Manager Configuration**

* Azure Traffic Manager was used with **Priority routing method**.
* The frontend services of both clusters are exposed via ingress and linked to Traffic Manager endpoints.
* Priority routing ensures traffic is served by the primary (active) cluster unless it becomes unavailable.

**DR Simulation Steps**

1. Deployed both clusters with same app manifests.
2. Tested accessibility via dynamic DNS from Traffic Manager.
3. Manually stopped the primary cluster's frontend service.
4. Verified automatic failover to secondary region within seconds.
5. Restored primary and observed failback behavior.

**9. Monitoring & Observability**

**Tools Used**

* **Azure Monitor**: Centralized monitoring solution for collecting, analyzing, and acting on telemetry from Azure and on-premises environments.
* **Log Analytics**: Used to query logs, visualize trends, and identify anomalies across AKS, VM, and other services.
* **Application Insights**: Used for collecting performance and telemetry data from the Spring Boot backend and React frontend apps.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a graph

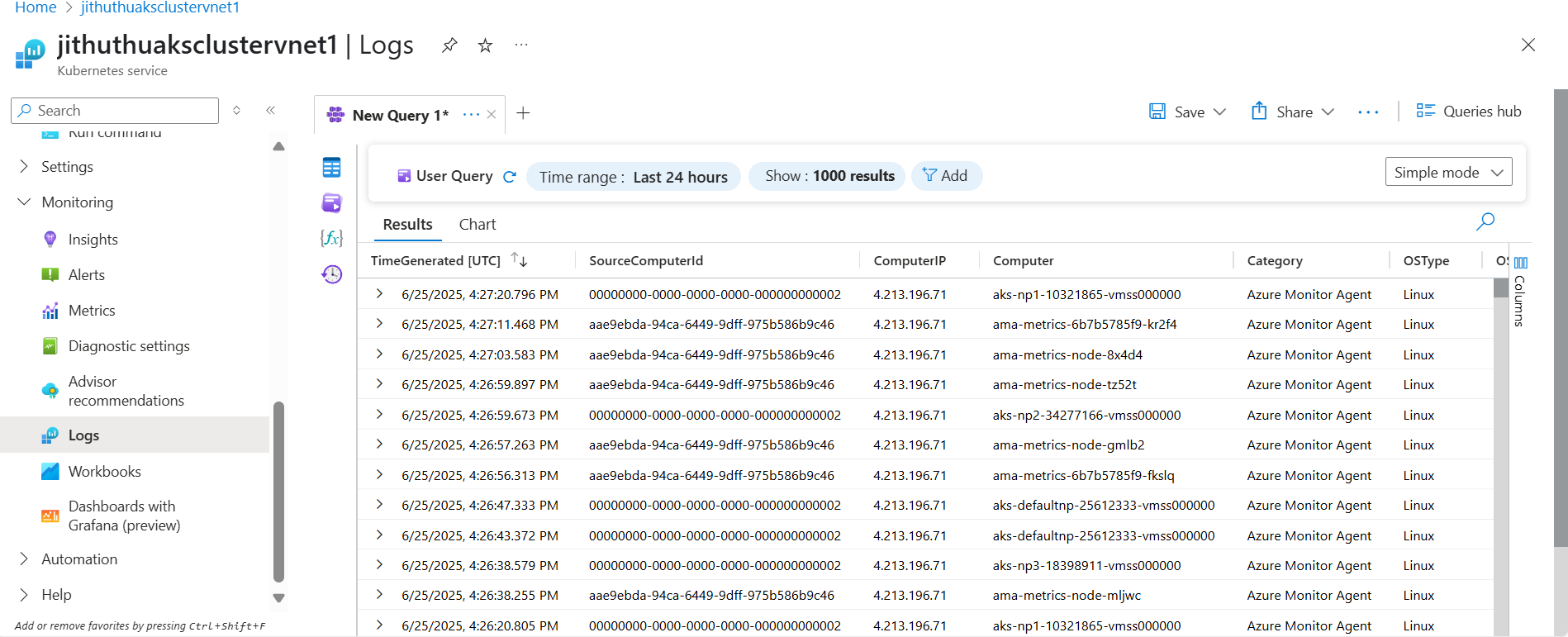
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



**Dashboards Created**

* Custom Azure Monitor dashboards were created to display:
  + AKS node and pod status
  + CPU and memory usage
  + Application response time and failure rates
  + Request count and HTTP response codes
  + Live Metrics Stream for Application Insights

**Alerts and Thresholds**

* + Alerts configured when:
    - Error rate exceeds 2% in 5-minute window
    - Pod crash loops detected
  + Alerts triggered via Azure Monitor and integrated with email and Azure DevOps for incident tracking.

A screenshot of a computer

AI-generated content may be incorrect.

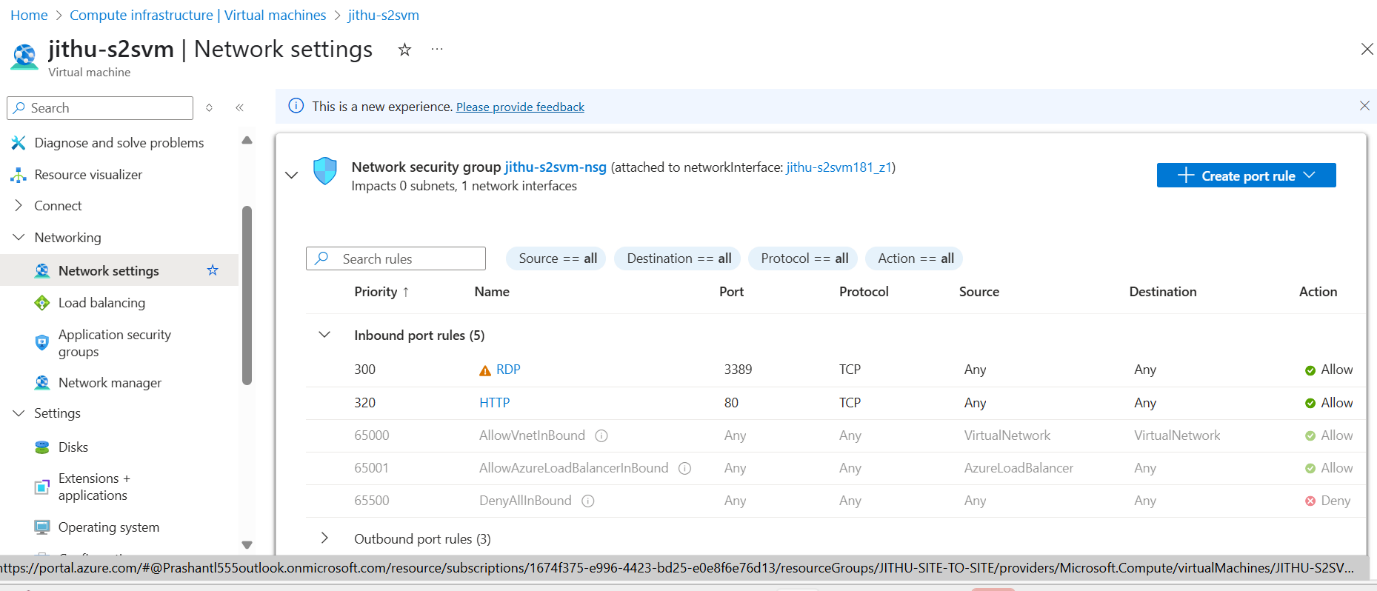
**10. Security & Compliance**

**Azure Key Vault**

* Secrets for application config, DB credentials, and ACR tokens stored securely.
* Azure Key Vault integrated into Azure Pipelines using service connection and AzureKeyVault@2 task.
* Secrets consumed at runtime without hardcoding sensitive data in pipeline YAML.

**NSG: Network Security Group Rules**

* Defined custom NSG rules for:
  + Allow SSH/RDP to Windows VM from known IPs only
  + Deny all other inbound traffic only 80 allow for connection for site to site connection



**11. Challenges & Troubleshooting**

* **Problem**: Ingress traffic not routing correctly between AKS clusters.  
   **Fix**: Corrected host config in Ingress and DNS target to nip.io address.
* **Problem**: Terraform plan failed due to missing backend state container.  
   **Fix**: Verified and created tfstate blob container in Azure Storage manually.
* **Problem**: VPN connection timeout.  
   **Fix**: Identified firewall blocking ports. Adjusted NSG and verified shared key match.
* **Problem**: SonarQube analysis not publishing results.  
   **Fix**: Updated token and ensured correct Quality Gate setup.
* **Problem**: Snyk scan exceeding pipeline timeout.  
   **Fix**: Moved Snyk scan to run in parallel job and adjusted timeout settings.
* **Problem**: Application Insights not receiving logs.  
   **Fix**: Verified instrumentation key and added proper telemetry SDK in Spring Boot app.
* **Problem**: React frontend build failing in CI pipeline.  
   **Fix**: Set NODE\_OPTIONS=--openssl-legacy-provider to handle Webpack compatibility issue with Node.js 18.
* **Problem**: Kubernetes manifests failing due to missing ingress class.  
   **Fix**: Added ingressClassName: nginx explicitly in Ingress YAML definition.
* **Problem**: ACR image pull failed in AKS during deployment.  
   **Fix**: Created proper role assignment between AKS managed identity and ACR scope.
* **Problem**: Log Analytics workspace not connected to AKS.  
   **Fix**: Enabled monitoring during AKS provisioning via Terraform and verified workspace linkage.

Application:

**A screenshot of a computer

AI-generated content may be incorrect.**