**Containerization with AKS – Modernization Approach**

Cloud Architecture Pattern

Table of Contents

[1 Introduction 3](#_Toc197474427)

[2 Infrastructure Overview 3](#_Toc197474428)

[2.1 Key Components 3](#_Toc197474429)

[3 Autoscaling Strategy 4](#_Toc197474430)

[3.1 Node Autoscaling 4](#_Toc197474431)

[3.1.1 Best Practices 4](#_Toc197474432)

[3.2 Pod Autoscaling 5](#_Toc197474433)

[3.2.1 Best Practices 5](#_Toc197474434)

[4 Enterprise Security Considerations 5](#_Toc197474435)

[4.1 Network Security 5](#_Toc197474436)

[4.2 Identity and Access Management 5](#_Toc197474437)

[4.3 Data Protection 5](#_Toc197474438)

[4.4 Threat Detection 5](#_Toc197474439)

[5 IaC (Terraform Module) for AKS Enterprise Architecture 6](#_Toc197474440)

[6 Business Continuity and Disaster Recovery (BCDR) 6](#_Toc197474441)

[6.1 High Availability 6](#_Toc197474442)

[6.2 Backup and Recovery 6](#_Toc197474443)

[6.3 Failover Strategy 6](#_Toc197474444)

[7 Application Deployment Flow 7](#_Toc197474445)

[7.1 Tool Stack for Pipelines 7](#_Toc197474446)

[7.2 Deployment Strategies 8](#_Toc197474447)

[7.2.1 Blue-Green Deployment 8](#_Toc197474448)

[7.2.2 Canary Deployment 8](#_Toc197474449)

[7.2.3 Rolling Updates 8](#_Toc197474450)

[7.3 Tools for Advanced Rollouts 8](#_Toc197474451)

[8 Configuration Management 8](#_Toc197474452)

[8.1 Flexible Deployment 8](#_Toc197474453)

[8.2 Environment Separation 9](#_Toc197474454)

[8.3 GitOps 9](#_Toc197474455)

[8.4 Use Case Differentiation 9](#_Toc197474456)

[8.5 Security and Maintainability 9](#_Toc197474457)

[9 Implementation Notes 9](#_Toc197474458)

[9.1 Infrastructure 9](#_Toc197474459)

[9.2 Application Deployment 10](#_Toc197474460)

[9.3 Cost Optimization Strategies 10](#_Toc197474461)

[10 Use Cases 10](#_Toc197474462)

[10.1 Complex Medical Conditions (CMC) 11](#_Toc197474463)

[10.1.1 Current On-Prem State 11](#_Toc197474464)

[10.1.2 Target Cloud State 12](#_Toc197474465)

# Introduction

Azure Kubernetes Service (AKS) is a managed container orchestration service provided by Microsoft Azure. It simplifies deploying, managing, and scaling containerized applications using Kubernetes. For enterprise-grade applications, it is essential to incorporate advanced features such as autoscaling, robust security, and Business Continuity and Disaster Recovery (BCDR).

This document outlines a comprehensive AKS setup for enterprise environments, covering infrastructure components, scaling strategies, and security best practices, supplemented with a real-world use case.

# Infrastructure Overview

An enterprise AKS setup typically involves multiple Azure services working together to provide a secure, resilient, and scalable environment.

## Key Components

**Azure Kubernetes Service (AKS):** Core container orchestration platform.

**Azure SQL Database / CosmosDB:** Persistent storage for structured data.

**Azure Blob Storage:** Object storage for unstructured data.

**Azure Front Door:** Global HTTP load balancer and application firewall.

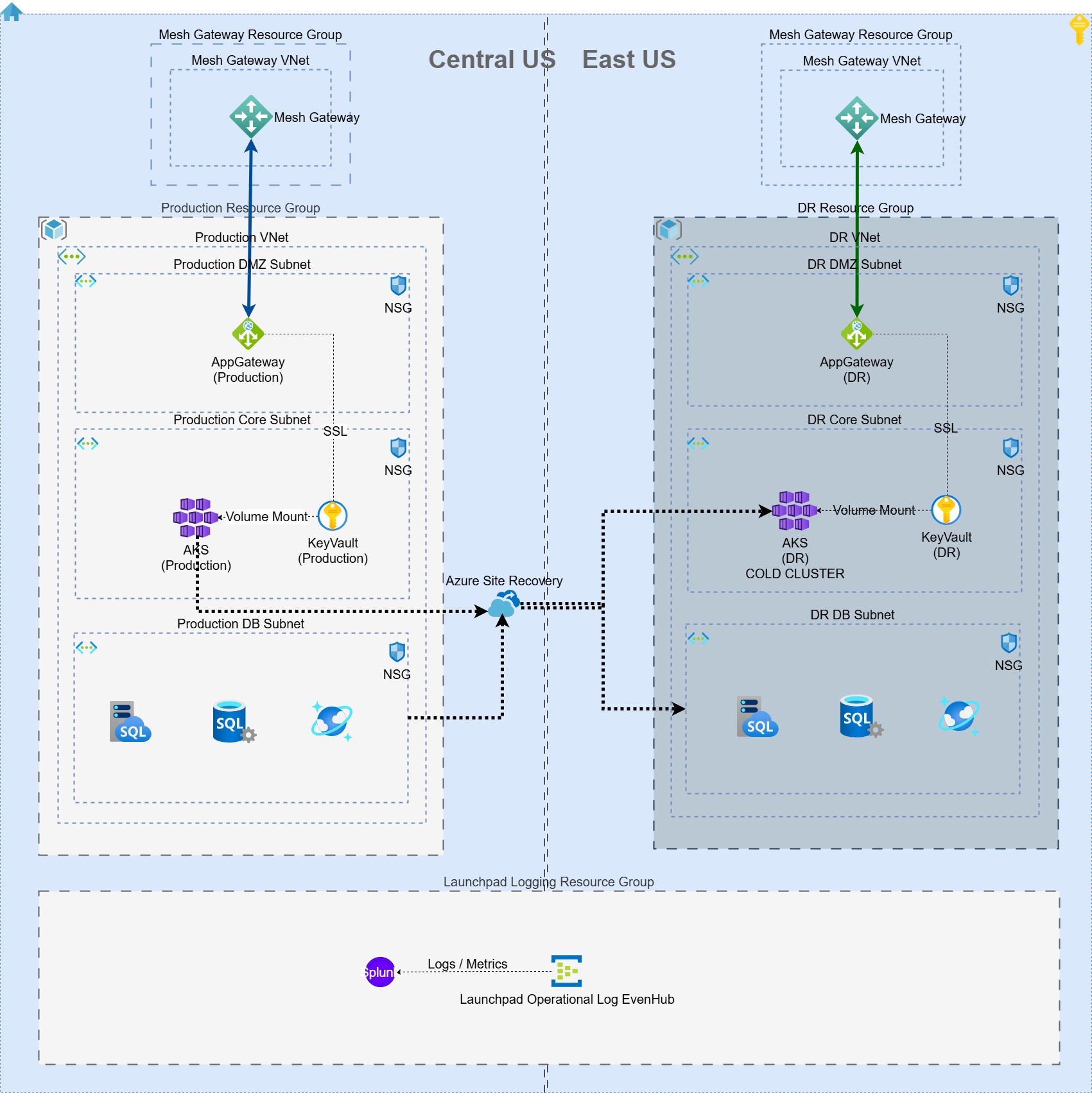
**Azure Application Gateway or Load Balancer:** Internal/external traffic management with Web Application Firewall (WAF).

**Azure Key Vault:** Secure secrets and key management.

**Azure Monitor & Log Analytics:** Observability and monitoring.

**Azure Policy & Defender for Cloud:** Compliance and security monitoring.

**Backup Vault & Site Recovery:** For BCDR.



# Autoscaling Strategy

Scalability is crucial to handle varying workloads efficiently. AKS supports both node auto scaling and pod auto scaling.

## Node Autoscaling

AKS supports native node autoscaling through:

* **Cluster Autoscaler:** Dynamically adjusts the number of nodes in a node pool based on pending pods.
* **Karpenter:** An open-source, flexible, and high-performance Kubernetes cluster autoscaler originally developed by AWS, now available for use in AKS. Karpenter helps rapidly provision nodes in response to unschedulable pods, optimizing cost and performance.

### Best Practices

* Use multiple node pools for workload separation (e.g., system vs. user workloads).
* Combine with taints and tolerations for isolation.
* Use Karpenter for fine-grained control over instance provisioning, especially in heterogeneous workloads.
* Enable spot node pools for cost-effective batch workloads.

## Pod Autoscaling

* Horizontal Pod Autoscaler (HPA) adjusts pod replicas based on metrics like CPU, memory, or custom metrics.
* Vertical Pod Autoscaler (VPA) adjusts resource requests and limits.

### Best Practices

* Use HPA for dynamic workloads.
* Combine with KEDA for event-driven scaling (e.g., Azure Queue).

# Enterprise Security Considerations

Security is paramount for enterprise deployments. AKS offers deep integration with Azure’s identity and security stack.

## Network Security

* Deploy AKS in Azure Virtual Network (VNet) with private cluster configuration.
* Use Network Policies to control traffic between pods.
* Use Istio to enforce mutual TLS (mTLS), providing encrypted, authenticated communication between micro services and enabling zero-trust network security inside the AKS cluster.
* Integrate Application Gateway with WAF for Layer 7 security.

## Identity and Access Management

* Integrate AKS with Azure Active Directory (AAD).
* Use Azure RBAC to control access to Kubernetes resources.
* Limit permissions using Kubernetes RBAC.

## Data Protection

* Use Azure Key Vault for secrets, certificates, and keys.
* Enable encryption at rest and in transit for data.
* Use Transparent Data Encryption (TDE) or Azure SQL.

## Threat Detection

* Enable Microsoft Defender for Container.
* Monitor logs and metrics with Azure Monitor and Log Analytics.
* Apply Azure Policy for compliance enforcement.

# IaC (Terraform Module) for AKS Enterprise Architecture

Terraform module to provision and manage AKS clusters following enterprise-grade architecture principles.

**The module encapsulates**

* Cluster provisioning with Azure CNI, private clusters, RBAC, and managed identities.
* Node pool configuration includes taints, labels, auto-scaling, and spot instances.
* Networking (custom VNet, subnet segmentation, NSG associations).
* Monitoring and observability via Azure Monitor, Log Analytics, and optional Prometheus/Grafana integration.
* Security components such as Azure Key Vault, Defender for Cloud, pod identity integration, and network policies.
* It follows a modular structure supporting environment-specific overrides through input variables and backend state separation.
* Designed for reusability and compliance, it integrates with Azure DevOps pipelines and supports policy-as-code (e.g., Azure Policy, Sentinel).
* Ensures repeatable, scalable, and secure AKS deployments with built-in tagging, naming conventions, and cost controls.

# Business Continuity and Disaster Recovery (BCDR)

Ensuring availability and data integrity in the face of disruptions is vital.

## High Availability

* Use multiple availability zones.
* Distribute workloads across multiple node pools.

## Backup and Recovery

* Use Azure Backup for persistent volumes.
* Enable SQL Geo-replication and ZRS for Blob Storage.

## Failover Strategy

* Use Azure Site Recovery for region-to-region failovers.
* Keep infrastructure-as-code (IaC) templates ready for quick redeployment.
* Regular DR Drills are essential.

# Application Deployment Flow

A diagram of a network

AI-generated content may be incorrect.

## Tool Stack for Pipelines

|  |  |  |
| --- | --- | --- |
| Layer | Tool | Purpose |
| CI/CD | GitHub Actions | Code integration, build, test, deployment pipelines |
| GitOps | ArgoCD | Declarative deployment to AKS, Git is source of truth |
| Code Quality | SonarQube | Static code analysis (bugs, smells, duplications) |
| Container Scanning | Trivy, Snyk | Detect vulnerabilities in images & dependencies |
| Dependency Check | OWASP DC | Identifies vulnerable libraries (CVE scanning) |
| Secrets Management | Azure Key Vault | Secure storage and access of secrets and credentials |
| Runtime Protection | Azure Defender for Kubernetes | Security posture management and threat protection |
| Monitoring & Alerts | HCP Observability, Azure Monitor, Grafana | Observability with alerts, dashboards, and metrics |

## Deployment Strategies

Enterprise-grade deployments to AKS should follow controlled rollout strategies to reduce risk and enable safe deployment. Key deployment strategies include:

### Blue-Green Deployment

* Maintain two environments: Blue (live) and Green (staging).
* Route traffic to Green after successful validation, then decommission Blue.
* Minimal downtime, easy rollback.

### Canary Deployment

* Gradually route traffic to new version while monitoring key metrics.
* Use tools like Argo Rollouts or Flagger for traffic shifting.
* Useful for catching issues early before full rollout.

### Rolling Updates

* Update pods incrementally, maintaining availability.
* Kubernetes default rollout strategy.
* Works well for stateless applications.

## Tools for Advanced Rollouts

* **Argo Rollouts:** Progressive delivery controller for Kubernetes, integrates with ArgoCD.
* **Azure DevOps Pipelines:** Supports rolling deployments via deployment strategies and gates.
* **Azure Traffic Manager:** Can be used with blue-green scenarios across regions.

# Configuration Management

## Flexible Deployment

Helm provides templated, reusable Helm charts, Kustomize allows overlay-based customization without templates.

## Environment Separation

Helm uses different values.yaml files; Kustomize separates configurations using base and overlays directories.

## GitOps

GitOps is fully supported by ArgoCD and Azure DevOps pipelines, enabling automated, Git-driven deployments.

## Use Case Differentiation

Use Helm for complex, configurable apps, use Kustomize for simpler apps requiring clean YAML and environment overlays.

## Security and Maintainability

Both Helm and Kustomize help maintain DRY (Don't Repeat Yourself) configurations and ensure secure, auditable IaC practices for AKS workloads.

# Implementation Notes

## Infrastructure

* **IaC:** Use a standardized custom Terraform module for AKS to ensure repeatable, secure, and scalable deployments.
* **Private AKS Cluster:** Deploy AKS with private API server access and Azure Bastion/JumpBox for secure management.
* **Azure AD & RBAC Integration:** Enable Azure Active Directory and Azure RBAC for secure and auditable cluster access control.
* **Node Pool Strategy:** Use separate node pools for system and workloads, with auto-scaling and spot instance support.
* **Storage & Network Policies:** Use Azure Disk/Files for stateful workloads, and enforce Kubernetes Network Policies for intra-cluster traffic control.
* **High Availability & BCDR:** Deploy AKS across availability zones, with backup policies for etcd and app state using Velero or Azure Backup.

## Application Deployment

* **CI/CD and GitOps:** Use GitHub Actions pipelines for build/test, and ArgoCD for GitOps-based deployment into AKS.
* **Security Scanning Tools:** Integrate SonarQube (code quality), Trivy/Snyk (image & dependency scanning), and OWASP Dependency-Check in pipeline stages.
* **Secrets Management:** Use Azure Key Vault for secrets, integrated via CSI driver and terraform, with RBAC and access policies.
* **Observability Integration:** Deploy Prometheus, Grafana, and Alertmanager; integrate with Azure Monitor and route alerts to Slack/Teams.
* **Helm/Kustomize Strategy:** Standardize deployments using Helm (for templating) or Kustomize (for overlays), depending on team structure and use case.

## Cost Optimization Strategies

* **Azure Cost Management + Billing:** Use Azure Cost Management to monitor, analyze, and forecast AKS cluster. Set budgets and alerts to proactively track overages and optimize resource usage based on historical trends.
* **Use Azure Advisor for Cost Recommendations:** Azure Advisor provides real-time recommendations to reduce AKS and other infrastructure costs — like resizing underutilized node pools, deleting unused resources, and switching to reserved instances for predictable workloads.
* **Enable Cluster Autoscaler & Node Pool Rightsizing:** Implement Cluster Autoscaler and optimize node sizing to scale AKS workloads dynamically based on demand. i.e., If need will use spot node pools or burstable VM SKUs (like B-series) for dev/test environments to lower compute costs.
* **Use Azure Monitor + Container Insights for Waste Detection:** Monitor idle pods, over-provisioned resources, and underutilized deployments using Azure Monitor and Container Insights. This helps fine-tune resource requests/limits to avoid wasteful over-allocation.
* **Implement CI/CD Optimization in Azure DevOps:** Streamline pipelines in ADO using self-hosted agents on demand, limit unnecessary pipeline runs with path filters, and reuse templates to avoid redundant builds — reducing pipeline execution costs.

# Use Cases

## Complex Medical Conditions (CMC)

CMC is a suite of applications designed for insurance providers, employers, and users. CMC consists of six portal applications and batch jobs to support business operations. The application tech stack is Java, Spring Framework, JSP, Javascript, Tomcat, Apache HTTPD, Oracle Database, and Microsoft SQL Server.

### Current On-Prem State

**Infrastructure & Configuration**

* The portal applications are deployed to Tomcat Servers which are facaded by HTTPD.
* Tomcat and HTTPD updates are manually handled.
* No Infrastructure as Code (IaC) or Configuration as Code (CaC) practices are implemented.

**Application Characteristics**

* Applications are stateless but rely on sticky sessions.
* Hosted on Linux; batch jobs generate files stored on NFS/SFTP and accessed locally.
* Batch jobs pull data from Oracle Managed DB and MS SQL Server.
* Authentication via OHID with MFA for customer-facing apps.

**Technology Stack**

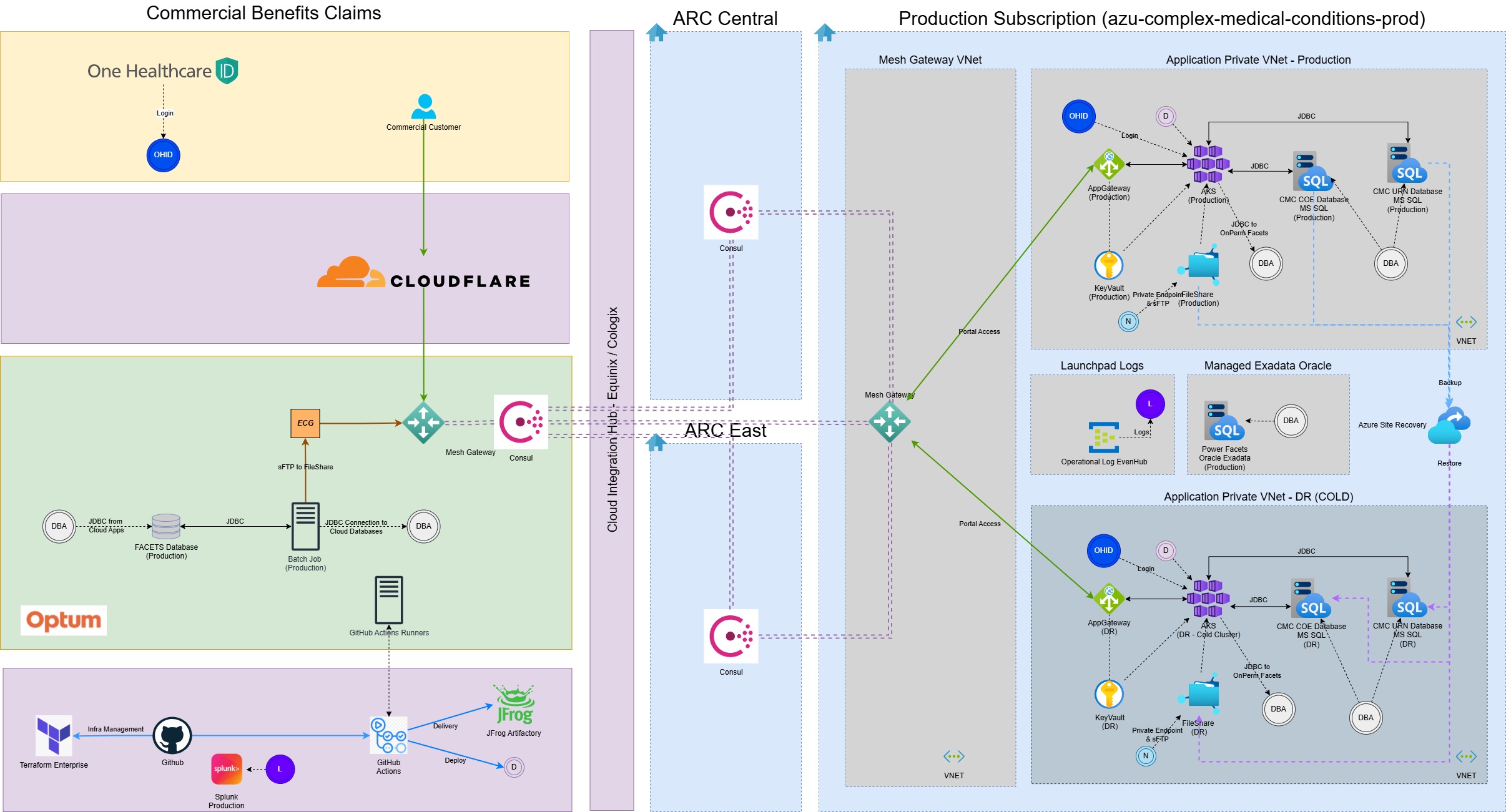
* Backend: RESTful APIs using Hibernate with EH Cache.
* Frontend: Uses TinyMCE for content editing and HTTP API calls.
* Search powered by Apache Lucene.

**Security Operations**

* Static security scans (SCA/SAST) are performed as part of existing CI

**Monitoring & Orchestration**

* Monitoring tools include Grafana and Dynatrace.
* IBM Tivoli manages batch job orchestration.



### Target Cloud State

**Containerization & Deployment**

* All CMC portal applications are containerized and deployed to Azure Kubernetes Service (AKS) for scalability. Implement shared session management to support horizontal scaling without sticky sessions.

**Environment Segregation**

* Separate production and non-production environments using dedicated VNets with private endpoint to enhance security and isolation.

**Infrastructure Monitoring**

* HCP recommended tools for comprehensive observability, monitoring, and compliance.

**Traffic Management & Security**

* Integrate Azure Load Balancer, Front Door, and WAF to improve traffic distribution, security, and global accessibility.

**CI/CD & Security**

* GitHub Actions with self-hosted runners for production to ensure isolation.
* Conduct regular network/application penetration tests and runtime vulnerability scans.

**Application Modernization**

* Re-architect apps into modular, containerized services (frontend/backend separation) for better scalability and orchestration.

**Azure Native Adoption**

* Leverage Azure-native services (Load Balancer, API Gateway, Monitoring, etc.) to boost performance, security, and reduce manual operations.