Azure Kubernetes Service (AKS) Architecture Documentation

Overview

This document provides a detailed explanation of the Azure Kubernetes Service (AKS) architecture depicted in the diagram. The architecture represents a modern cloud-native application deployment using AKS with various supporting Azure services for CI/CD, security, monitoring, and data storage.

Architecture Components

Core Infrastructure

Component	Description
Azure Kubernetes Service (AKS)	Managed Kubernetes service that simplifies container orchestration
Kubernetes Cluster	Orchestrates containerized applications and manages their lifecycle
Virtual Network	Isolated network environment containing all AKS components

Client Access Path

Component	Description
Client Apps	Frontend applications or services that connect to the system
Azure Load Balancer	Distributes incoming traffic to maintain high availability

Deployment Pipeline

Component	Description
Azure Pipelines	CI/CD service for automating build and deployment workflows
Helm	Package manager for Kubernetes that simplifies application deployment
Container Registry	Stores and manages container images for deployment
Docker Push/Pull	Commands to upload and download container images

AKS Namespaces

Namespace	Purpose
Front-end	Contains user-facing components including Ingress controller
Back-end Services	Houses microservices and business logic components
Utility Services	Holds supporting services like monitoring and search

Infrastructure Components

Component	Description
Ingress Controller	Manages external access to services, acting as API gateway
Pod Autoscaling	Automatically adjusts the number of pods based on demand
Elasticsearch	Search and analytics engine for logs and application data
Prometheus	Monitoring and alerting system for collecting metrics

External Services

Service	Purpose
External Data Stores	External databases (SQL) and cloud storage solutions
Azure Active Directory	Identity and access management service
Azure Monitor	Performance monitoring, alerting, and diagnostics service
Azure Key Vault	Secure storage for secrets, certificates, and encryption keys

Security

Component	Description
RBAC (Role-Based Access Control)	Kubernetes and Azure AD integration for access control
Azure Key Vault	Securely stores and manages sensitive information
Virtual Network	Network isolation for AKS resources

Data Flow

User Request Path

1. **Client Applications**:
   - Send requests to the Azure Load Balancer
2. **Load Balancer**:
   - Distributes traffic to the Ingress controller in the Front-end namespace
3. **Ingress Controller**:
   - Routes requests to appropriate back-end services based on URL paths and rules
4. **Back-end Services**:
   - Process requests, potentially calling other microservices
   - Connect to external data stores when needed

    - Scale automatically based on demand via Pod autoscaling

Deployment Flow

1. **Development and Code Commits**:
    - Developers push code to a repository connected to Azure Pipelines
2. **CI/CD Pipeline**:
    - Azure Pipelines triggers build process for committed code
    - Builds Docker images and pushes them to Container Registry
    - Uses Helm for packaging and deploying applications to AKS
3. **Deployment to AKS**:
    - Helm upgrades deploy new versions to the Kubernetes cluster
    - Docker images are pulled from Container Registry to AKS nodes

Security and Management Flow

1. **Access Control**:
    - Authentication and authorization via Azure Active Directory
    - RBAC for fine-grained access control within Kubernetes
2. **Monitoring and Operations**:
    - Azure Monitor collects metrics and logs from AKS
    - Prometheus gathers application-specific metrics
    - Dev/Ops teams access monitoring dashboards and administrative tools

Key Features and Benefits

High Availability and Scalability

- **Load Balancing**: Azure Load Balancer distributes traffic evenly
- **Pod Autoscaling**: Automatically adjusts resources based on demand
- **Multiple Back-end Services**: Distributed architecture for resilience
- **Kubernetes Orchestration**: Self-healing capabilities to maintain desired state

DevOps and CI/CD Integration

- **Azure Pipelines**: Automated build and deployment workflows
- **Helm Integration**: Standardized application packaging and deployment
- **Container Registry**: Central repository for container images

- **Infrastructure as Code**: Deployment templates for consistent environments

Security and Compliance

- **Azure Active Directory Integration**: Enterprise-grade identity management
- **RBAC**: Fine-grained access control at Kubernetes level
- **Key Vault**: Secure secret management
- **Network Isolation**: Virtual network segmentation

Monitoring and Observability

- **Azure Monitor**: Platform-level monitoring and alerting
- **Prometheus**: Application-specific metrics collection
- **Elasticsearch**: Log aggregation and search capabilities
- **End-to-End Visibility**: From infrastructure to application performance

Implementation Considerations

Namespace Organization

The architecture uses namespaces for logical separation of components:

1. **Front-end Namespace**:
    - Contains Ingress controller
    - Entry point for external traffic
    - API gateway functionality
2. **Back-end Services Namespace**:
    - Contains business logic microservices
    - Multiple independent services that can scale individually
    - Service-to-service communication
3. **Utility Services Namespace**:
    - Contains supporting services like Elasticsearch and Prometheus
    - Separated to allow different scaling and security policies

Deployment Strategy

The architecture supports:

1. **Blue/Green Deployments**: Using Helm for controlled rollouts

2. **Canary Releases**: Gradual traffic shifting to new versions
3. **Rollbacks**: Quick reversion to previous versions if issues are detected

Scaling Considerations

1. **Horizontal Pod Autoscaling**: Based on CPU, memory, or custom metrics
2. **Cluster Autoscaling**: AKS node pool scaling based on pod scheduling demands
3. **Manual Scaling**: For predictable load patterns or special circumstances

Security Best Practices

1. **Network Policies**: Limiting pod-to-pod communication
2. **Service Principals**: Least privilege access for AKS cluster operations
3. **Secret Management**: Using Azure Key Vault integration
4. **Container Security**: Scanning images for vulnerabilities

Operational Guide

Monitoring and Alerts

1. **Key Metrics to Monitor**:
   - Cluster health and node status
   - Pod resource utilization
   - Application response times
   - Error rates and exception counts
2. **Alert Configuration**:
   - Critical service availability
   - Resource thresholds (80% CPU, memory)
   - Error rate spikes
   - Deployment status changes

Disaster Recovery

1. **Backup Strategies**:
   - Persistent volume snapshots
   - Database backups for external stores
   - Configuration backups via Azure Pipelines
2. **Recovery Procedures**:

- AKS cluster recreation from template
- Data restoration from backups
- Service redeployment via CI/CD pipeline

Cost Optimization

1. **Resource Management**:
    - Right-sizing node pools based on workload requirements
    - Leveraging spot instances for non-critical workloads
    - Autoscaling to match demand patterns
2. **Storage Optimization**:
    - Appropriate storage class selection
    - Log retention policies
    - Image cleanup and lifecycle management

Conclusion

This Azure Kubernetes Service architecture provides a robust, secure, and scalable platform for deploying containerized applications. The design incorporates Azure best practices for container orchestration, CI/CD integration, security, and monitoring. The multi-namespace approach with clearly defined responsibilities ensures that the system remains maintainable and can evolve over time.

The use of managed services like AKS, Azure Monitor, and Azure Active Directory reduces operational overhead while improving reliability and security. The integrated CI/CD pipeline with Azure Pipelines and Helm enables rapid, consistent deployments supporting modern DevOps practices.

Overall, this architecture represents a production-ready solution suitable for mission-critical applications requiring high availability, security, and scalability in a containerized environment.