Contoso

**Application Assessment Report**

Table of Contents

Introduction......................................................................3  
1 Application Overview............................................................4  
1.1 Key Business Drivers.......................................................4  
1.2 Key Contacts.................................................................4  
1.3 Migration Strategy............................................................4  
1.3.1 Migration Pattern and Complexity........................................4  
1.3.2 Technology Selection..................................................5  
1.3.3 Indicative Azure Cost..................................................5  
1.4 Database Information.......................................................5  
1.5 Macro Dependencies.......................................................6  
1.6 Security Considerations..................................................6  
1.7 Resiliency Configuration..................................................6  
1.8 Network Access Requirements.............................................7  
1.9 Identity Providers.......................................................7  
1.10 Automation.................................................................7  
1.11 Customer Impact............................................................8  
1.12 Operational Concerns..................................................8  
1.13 Migration Acceptance Tests.............................................8  
1.14 Observability............................................................9  
2 Supporting Documents.......................................................10  
3 Current Logical Architecture.............................................11  
3.1 Production Logical Architecture...................................12  
3.2 QA Logical Architecture...........................................13  
3.3 Development Logical Architecture..................................14  
4 Application Network Flow..................................................15  
4.1 Production Application Network Flow...............................16  
4.2 QA Application Network Flow.......................................17  
4.3 Development Application Network Flow..............................18  
5 Proposed Architecture in Azure........................................19  
5.1 Production Proposed Architecture..................................20  
5.2 QA Proposed Architecture..........................................21  
5.3 Development Proposed Architecture.................................22  
6 Architecture Heatmap.......................................................23  
7 Decision Matrix............................................................24  
8 Application Allocation and Scheduling...................................25  
9 Appendix......................................................................26  
9.1 Additional Backlog Items.............................................26  
9.2 Application and Infrastructure RBAC Information....................26  
9.2.1 Production Application and Infrastructure RBAC..................27  
9.2.2 QA Application and Infrastructure RBAC..........................28  
9.2.3 Development Application and Infrastructure RBAC.................29  
9.3 Azure Services RBAC Information...................................30  
9.3.1 Production Azure Services RBAC..................................31  
9.3.2 QA Azure Services RBAC..........................................32  
9.3.3 Development Azure Services RBAC.................................33  
9.4 Azure Tagging............................................................34  
9.4.1 Production Azure Tagging........................................35  
9.4.2 QA Azure Tagging................................................36  
9.4.3 Development Azure Tagging.......................................37  
9.5 Source Migration Delivery Information..............................38  
9.5.1 Production Source Delivery Information..........................39  
9.5.2 QA Source Delivery Information..................................40  
9.5.3 Development Source Delivery Information.........................41  
9.6 Target Migration Delivery Information..............................42  
9.6.1 Production Target Delivery Information..........................43  
9.6.2 QA Target Delivery Information..................................44  
9.6.3 Development Target Delivery Information.........................45

Introduction

This Application Assessment Report for Contoso provides a comprehensive analysis of the current application architecture, requirements, and recommendations for migration to Microsoft Azure.

The assessment has been conducted based on customer interviews, technical documentation review, and application analysis. This document serves as the foundation for migration planning and Azure architecture design.

The key areas covered in this assessment include:

• Application overview and business drivers

• Current architecture and dependencies

• Security and compliance requirements

• Network access patterns

• Migration strategy and Azure service recommendations

• Risk assessment and mitigation strategies

1 Application Overview

# 1.1 Key Business Drivers

The key business drivers for this migration include:

### Top 5 Business Drivers for Azure Cloud Migration

1. **Scalability and Performance Optimization**

The use of Azure Kubernetes Service (AKS) for compute and Redis for caching indicates a need for scalable infrastructure to handle dynamic workloads efficiently. Redis caching improves authentication performance, while Nginx as a load balancer optimizes web traffic distribution across application layers, ensuring high availability and responsiveness.

2. **Security and Compliance Requirements**

The application has stringent security requirements, including two-factor authentication, OTP caching with expiration policies, and authorization mechanisms. Azure provides robust security features such as Azure Active Directory, managed identity, and compliance certifications, making it an ideal platform to meet these requirements.

3. **Modernization of Application Architecture**

The application is structured as a three-tier architecture (web layer, app layer, and database layer) and leverages containerized workloads in AKS. Migrating to Azure supports modernization efforts by enabling microservices-based architecture, container orchestration, and integration with cloud-native services.

4. **Operational Efficiency Goals**

The adoption of DevOps practices, such as deploying Nginx for load balancing and Redis for caching, reflects a focus on streamlining operations and improving application management. Azure's ecosystem supports automation, CI/CD pipelines, and infrastructure-as-code, enhancing operational efficiency.

5. **Strategic Technology Initiatives for Cloud Adoption**

The migration to Azure aligns with strategic goals to leverage cloud-native technologies like AKS, Redis, and Postgres. This transition enables the organization to stay competitive, adopt cutting-edge tools, and benefit from Azure's managed services, reducing the burden of on-premises infrastructure management.

### Additional Notes:

- **Business Continuity and Disaster Recovery Needs**: While not explicitly addressed in the conversation, disaster recovery is a critical consideration for cloud migrations. Azure's built-in DR capabilities (e.g., geo-redundancy, backup services) could be leveraged in future planning.

- **Cost Optimization**: Although cost considerations are not explicitly mentioned, migrating to Azure often reduces infrastructure costs through pay-as-you-go pricing and resource optimization.

# 1.2 Key Contacts

Key project contacts identified from the assessment:

* • Frank is identified as the application owner.

# 1.3 Migration Strategy

## 1.3.1 Migration Pattern and Complexity

Based on the application assessment, the recommended migration approach is:

**Replatform** - Optimize the existing application by leveraging Azure-managed services while retaining the core architecture and containerized deployment model.

**Rationale:**

The current technology stack is already cloud-native, containerized, and deployed on Azure Kubernetes Service (AKS), which indicates a high level of cloud readiness. A Replatform approach allows the organization to optimize the use of Azure services, such as managed databases and networking, without requiring a complete refactor or rebuild. This balances speed, cost, and long-term benefits by leveraging managed services like Azure Database for PostgreSQL and Azure Cache for Redis, reducing operational overhead while maintaining the existing Kubernetes-based deployment model.

**Complexity Assessment: Medium**

Moderate complexity requiring careful planning and phased approach. Some technology adaptations needed for optimal cloud deployment.

Key factors: Containerized deployment reduces complexity; Cloud-friendly technology stack; N-tier architecture requires careful tier migration planning; Multiple database technologies increase migration complexity

**Key Migration Considerations:**

• Ensure compatibility of the existing Postgres and Redis configurations with Azure-managed services.

• Evaluate the performance and scalability requirements of the application to configure AKS and managed services appropriately.

• Plan for minimal downtime during the migration to avoid disruption to users.

**Recommended Migration Phases:**

1. Phase 1 description: Assess and plan the migration by analyzing the current AKS setup, database configurations, and application dependencies. Identify Azure services to replace existing components (e.g., Azure Database for PostgreSQL, Azure Cache for Redis).

2. Phase 2 description: Migrate the databases to Azure-managed services (PostgreSQL and Redis) using tools like Azure Database Migration Service. Update application configurations to point to the new managed services.

3. Phase 3 description: Optimize and validate the AKS deployment by leveraging Azure-native features such as Azure Monitor for Kubernetes, Azure Container Registry, and Azure Networking services. Perform end-to-end testing and monitor performance post-migration.

## 1.3.2 Technology Selection

**Azure Technology Selection and Architecture Strategy:**

# Azure Technology Selection and Architecture Recommendations for Contoso Migration

## 1. Current Technology Stack Analysis

### Technologies, Frameworks, and Infrastructure Components:

- **Application Name**: Contoso

- **Database**: PostgreSQL (Postgres)

- **Caching**: Redis (used for caching OTPs and other metrics)

- **Compute**: Azure Kubernetes Service (AKS) hosting application components in pods

- Components: Nginx (load balancer), Contoso application, Redis, Postgres

- **Networking**: Nginx as a load balancer in front of the web layer

- **Security**: Two-factor authentication (2FA) with OTP caching and expiration policies

- **Architecture**: Three-tier architecture (web layer, app layer, database layer)

### Observations:

- The application is containerized and deployed on AKS, indicating a modern microservices-based architecture.

- Redis is used for caching, which improves performance for authentication workflows.

- Postgres is the primary database for user information storage.

- Disaster recovery and monitoring strategies are not addressed in the current setup.

---

## 2. Migration Strategy Recommendation

### Recommended Migration Approach:

- **Replatform**: The current architecture is already containerized and running on AKS, which aligns well with Azure's cloud-native services. The migration will focus on optimizing the existing setup for Azure-native services without significant code changes.

- **Enhancements**: Introduce Azure-native services for security, monitoring, disaster recovery, and cost optimization.

### Migration Phases:

1. **Assessment and Planning**:

- Evaluate the current AKS cluster configuration and workloads.

- Assess Postgres and Redis configurations for compatibility with Azure Database for PostgreSQL and Azure Cache for Redis.

2. **Replatforming**:

- Migrate Postgres to Azure Database for PostgreSQL.

- Migrate Redis to Azure Cache for Redis.

- Optimize AKS workloads for Azure-native integrations.

3. **Enhancements**:

- Implement Azure Monitor and Azure Log Analytics for monitoring.

- Configure Azure Active Directory (AAD) for identity and access management.

- Set up disaster recovery and backup strategies.

4. **Validation and Optimization**:

- Conduct performance testing and fine-tune configurations.

- Optimize costs using Azure Advisor recommendations.

---

## 3. Azure Services Recommendations

### Compute:

- **Azure Kubernetes Service (AKS)**:

- Retain AKS for container orchestration.

- Use Azure Container Registry (ACR) for container image storage and management.

- Enable AKS features like Azure Policy for governance and Azure Defender for Kubernetes for security.

### Database:

- **Azure Database for PostgreSQL**:

- Use the Flexible Server deployment option for better control over scaling and high availability.

- Enable automated backups and geo-redundancy for disaster recovery.

- Configure performance tiers based on workload requirements.

### Caching:

- **Azure Cache for Redis**:

- Use the Premium tier for enhanced security (VNet integration) and persistence.

- Configure Redis data eviction policies to optimize memory usage.

### Networking:

- **Azure Application Gateway**:

- Replace Nginx with Azure Application Gateway for load balancing and web application firewall (WAF) capabilities.

- Enable end-to-end SSL encryption for secure communication.

- **Azure Virtual Network (VNet)**:

- Deploy all resources within a VNet for secure communication.

- Use Network Security Groups (NSGs) to control traffic flow.

### Security:

- **Azure Active Directory (AAD)**:

- Integrate AAD for user authentication and role-based access control (RBAC).

- Enable Multi-Factor Authentication (MFA) for enhanced security.

- **Azure Key Vault**:

- Store sensitive information like database connection strings and API keys securely.

- **Azure Defender**:

- Enable Azure Defender for threat protection across resources.

### Monitoring:

- **Azure Monitor and Log Analytics**:

- Use Azure Monitor for real-time performance monitoring.

- Configure Log Analytics for centralized logging and diagnostics.

- **Azure Application Insights**:

- Enable Application Insights for application performance monitoring and telemetry.

### Disaster Recovery:

- **Azure Site Recovery**:

- Implement Azure Site Recovery for AKS disaster recovery.

- **Geo-Replication**:

- Enable geo-replication for Azure Database for PostgreSQL and Azure Cache for Redis.

---

## 4. Architecture Considerations

### Design Patterns:

- **Microservices Architecture**:

- Retain the current three-tier architecture and optimize for scalability.

- **Caching Strategy**:

- Use Redis for caching frequently accessed data to reduce database load.

- **Load Balancing**:

- Replace Nginx with Azure Application Gateway for better integration with Azure services.

### Scalability:

- Enable horizontal scaling for AKS pods based on CPU/memory usage.

- Configure auto-scaling for Azure Database for PostgreSQL and Azure Cache for Redis.

### Security:

- Enforce network isolation using VNets and NSGs.

- Use Azure Policy to enforce compliance and governance.

### Integration:

- Use Azure DevOps or GitHub Actions for CI/CD pipelines.

- Integrate Azure Key Vault with AKS for secure secret management.

---

## 5. Modernization Opportunities

### Cloud-Native Services:

- Replace Nginx with Azure Application Gateway for better integration and security.

- Use Azure Database for PostgreSQL and Azure Cache for Redis for managed database and caching services.

### Enhanced Security:

- Implement Azure Active Directory for centralized identity management.

- Use Azure Key Vault for secure secret storage.

### Monitoring and Observability:

- Leverage Azure Monitor, Log Analytics, and Application Insights for end-to-end monitoring.

### Disaster Recovery:

- Implement geo-redundancy and Azure Site Recovery for high availability and disaster recovery.

---

## 6. Cost Optimization Opportunities

- Use Azure Reserved Instances for predictable workloads to reduce compute costs.

- Enable auto-scaling for AKS and database services to optimize resource usage.

- Use Azure Advisor to identify underutilized resources and optimize costs.

---

## 7. Implementation Phases and Priorities

### Phase 1: Assessment and Planning

- Evaluate current workloads and dependencies.

- Plan migration strategy and identify Azure services.

### Phase 2: Replatforming

- Migrate Postgres to Azure Database for PostgreSQL.

- Migrate Redis to Azure Cache for Redis.

- Optimize AKS workloads.

### Phase 3: Enhancements

- Implement Azure Monitor, Log Analytics, and Application Insights.

- Configure Azure Active Directory and Key Vault.

- Set up disaster recovery and backup strategies.

### Phase 4: Validation and Optimization

- Conduct performance testing and fine-tune configurations.

- Optimize costs using Azure Advisor recommendations.

---

By following this comprehensive migration and modernization plan, the Contoso application can leverage Azure's cloud-native capabilities to achieve scalability, security, and cost efficiency while maintaining high performance and reliability.

## 1.3.3 Indicative Azure Cost

**Indicative Monthly Azure Costs:**

**Estimated costs based on recommended Azure services:**

|  |  |
| --- | --- |
| **Service Category** | **Estimated Monthly Cost** |
| Azure Kubernetes Service (AKS) | $300 - $800 |
| Azure SQL Database (Default) | $200 - $500 |
| Azure Cache for Redis | $100 - $250 |
| Application Gateway + VNet | $150 - $300 |
| Key Vault + Security Center | $50 - $150 |
| Azure Monitor + App Insights | $100 - $250 |
| Blob Storage + File Storage | $50 - $150 |
| **Total Estimated** | **$950 - $2,400** |

**Cost Analysis Based on Current Technology Stack:**

* • Containerization Advantage: Existing containers reduce migration costs and enable efficient resource utilization
* • Cloud-Ready Architecture: Modern technology stack reduces migration costs and speeds up migration timeline
* • Database Migration: Leverage Azure Database Migration Service for cost-effective database transitions
* • Replatform Strategy: Moderate migration costs with significant long-term operational savings

**Cost Optimization Opportunities:**

* • Reserved Instances: 30-50% savings for predictable workloads
* • Azure Hybrid Benefit: Leverage existing licenses for Windows/SQL Server
* • Auto-scaling: Optimize resource utilization based on demand
* • Spot Instances: Up to 90% savings for development/testing environments
* • Azure Cost Management: Continuous monitoring and optimization
* • AKS Cost Optimization: Node auto-scaling and resource quotas

*Note: Costs are indicative and based on recommended Azure services from technology analysis. Actual costs may vary based on usage patterns, data transfer, and specific service configurations. A detailed Azure Pricing Calculator assessment will be performed during planning phase.*

# 1.4 Database Information

**Database Configuration and Requirements:**

From the assessment, the following database information was identified:

• What type of database is being used?: The database being used is Postgres.

• What storage solutions are being used?: The storage solutions being used are Postgres for storing user information and Redis for caching information such as OTP generation details.

**Recommended Database Migration Strategy:**

**Redis Migration:**

• **Target Platform**: Azure Cache for Redis Premium tier

• **Migration Method**: Redis data migration using MIGRATE command or backup/restore

• **High Availability**: Zone redundancy and geo-replication support

• **Performance**: In-memory performance with persistence options

• **Security**: SSL encryption, virtual network isolation, access policies

**PostgreSQL Migration:**

• **Target Platform**: Azure Database for PostgreSQL Flexible Server

• **Migration Method**: Azure Database Migration Service or pg\_dump/pg\_restore

• **High Availability**: Built-in high availability with zone redundancy

• **Backup Strategy**: Automated daily backups with point-in-time recovery (up to 35 days)

• **Security**: SSL/TLS encryption, Azure AD integration, Advanced Threat Protection

**Database Migration Best Practices:**

• Perform thorough compatibility testing in non-production environments

• Implement robust backup and rollback procedures

• Plan for minimal downtime using online migration techniques

• Establish performance baselines before and after migration

• Configure monitoring and alerting for database health and performance

• Document connection string changes and application configuration updates

# 1.5 Macro Dependencies

**System Dependencies and Integration Architecture:**

The following dependencies and integrations were identified:

• What Azure services are being used for compute?: Azure Kubernetes Service (AKS) is being used for compute, as the application components (e.g., Nginx, Contoso, Redis, Postgres) are running in pods within namespaces.

**Recommended Integration Architecture for Azure:**

**Database Integration Strategy:**

• **Data Architecture**: Implement database per service pattern for microservices

• **Data Synchronization**: Azure Data Factory for ETL processes

• **Event Sourcing**: Azure Event Store or Cosmos DB for event-driven architecture

• **CQRS Pattern**: Separate read/write databases using Azure SQL and Cosmos DB

• **Data Security**: Row-level security and column encryption

**Migration Integration Strategy:**

• **Phase 1**: Establish Azure backbone services (Service Bus, API Management)

• **Phase 2**: Migrate applications with maintained integration points

• **Phase 3**: Modernize integration patterns using cloud-native services

• **Phase 4**: Implement monitoring and observability across all integrations

**Best Practices:**

• Implement circuit breaker patterns for resilient integrations

• Use Azure Monitor for end-to-end distributed tracing

• Design for eventual consistency in distributed systems

• Implement proper retry policies with exponential backoff

• Use managed identities for secure service-to-service authentication

# 1.6 Security Considerations

• Identity and Access Management: Implement comprehensive identity management for Azure migration

• Data Encryption and Protection: Ensure data encryption in transit and at rest

# 1.7 Resiliency Configuration

**Business Continuity and Disaster Recovery Strategy:**

From the assessment, the following BCDR requirements were identified:

• What disaster recovery strategy is planned?: "The disaster recovery strategy is not addressed in the transcript."

**Recommended Azure BCDR Architecture:**

**Container-Based BCDR Strategy:**

• **Multi-Region AKS**: Deploy AKS clusters across multiple Azure regions

• **Container Registry Replication**: Geo-replicate container images for disaster recovery

• **Persistent Volume Backup**: Azure Backup for Kubernetes persistent volumes

• **Application State**: Implement stateless design with external state management

• **Traffic Routing**: Azure Traffic Manager for automatic failover between regions

**Database BCDR Strategy:**

• **Azure Cache for Redis**:

- Zone redundancy for high availability within region

- Geo-replication for cross-region disaster recovery

- Data persistence options for recovery scenarios

**Recommended Recovery Objectives:**

• **Recovery Time Objective (RTO)**: 4 hours - Maximum acceptable downtime

• **Recovery Point Objective (RPO)**: 1 hour - Maximum acceptable data loss

• **Availability Target**: 99.9% uptime - Approximately 8.76 hours downtime per year

**Azure Native BCDR Services:**

• **Azure Site Recovery**: Automated disaster recovery orchestration

• **Azure Backup**: Centralized backup management and monitoring

• **Azure Traffic Manager**: DNS-based traffic routing with health monitoring

• **Azure Monitor**: Continuous monitoring and alerting for BCDR events

• **Azure Resource Manager**: Infrastructure as Code for rapid environment recreation

**BCDR Implementation Phases:**

1. **Assessment**: Define RTO/RPO requirements and document dependencies

2. **Design**: Create multi-region architecture with appropriate Azure services

3. **Implementation**: Deploy BCDR infrastructure and configure replication

4. **Testing**: Regular disaster recovery drills and failover testing

5. **Documentation**: Maintain runbooks and escalation procedures

6. **Monitoring**: Continuous monitoring of backup and replication health

# 1.8 Network Access Requirements

• Load Balancing and High Availability: Implement load balancing for application availability and performance

• Secure Network Connectivity: Establish secure communication between application components

# 1.9 Identity Providers

• Azure Active Directory: Configuration to be determined

• Multi-Factor Authentication: Configuration to be determined

# 1.10 Automation

• Automation Assessment Required: Unable to extract automation details from transcript

# 1.11 Customer Impact

• Customer Impact Assessment Required: Unable to extract customer impact from transcript

# 1.12 Operational Concerns

• Operational Assessment Required: Unable to extract operational concerns from transcript

# 1.13 Migration Acceptance Tests

NA - Migration acceptance testing strategy to be defined during planning phase

# 1.14 Observability

Monitoring: Azure Monitor setup required - unable to extract specific requirements from transcript

Alerts: Azure Monitor alerting configuration needed

Events: Azure Event tracking to be configured

2 Supporting Documents

The following table provides a summary of the supporting documents to support the planning and migration of the application.

|  |  |
| --- | --- |
| Artefact | Information Location |
| Application Information Form | Generated from Q&A analysis |
| Azure Migrate Assessment | Azure Migrate Portal |
| Network Architecture Diagram | To be created |
| Security Requirements Document | To be created |
| Migration Plan Document | To be generated |

3 Current Logical Architecture

The following section provides a view of the logical architecture of the application per environment.

# 3.1 Production Logical Architecture

The following provides the logical architecture view of the Production environment.

### Logical Architecture for Contoso Production Environment

The Contoso application is structured as a three-tier architecture, comprising a web layer, application layer, and database layer. The production environment leverages Azure Kubernetes Service (AKS) for compute, where application components are deployed as pods within namespaces. These components include Nginx (acting as a load balancer), the Contoso application, Redis, and Postgres. This architecture ensures modularity, scalability, and efficient resource utilization.

#### Key Components and Technologies

1. **Web Layer**: Nginx serves as the load balancer, distributing incoming traffic across the Contoso application pods. This ensures optimal traffic management and high availability.

2. **Application Layer**: The Contoso application pods handle the core business logic and user interactions. These pods are deployed within AKS namespaces for isolation and scalability.

3. **Database Layer**: Postgres is used as the primary database for storing user information, ensuring data persistence and reliability. Redis is employed as a caching layer to store transient data, such as OTP (One-Time Password) generation details, to enhance authentication performance.

4. **Security**: The application enforces two-factor authentication (2FA) with OTP caching in Redis. Expiration policies are implemented to ensure secure and time-bound OTP validation.

5. **Storage**: Postgres handles persistent data storage, while Redis provides in-memory caching for performance-critical operations.

#### Environment-Specific Requirements

The production environment emphasizes security, scalability, and performance. Two-factor authentication is a critical requirement, with Redis caching OTPs to reduce database load and improve response times. The use of AKS ensures that the application can scale horizontally by adding more pods as demand increases. Nginx, as the load balancer, ensures high availability by distributing traffic evenly across the application pods.

#### Scalability, Availability, and Performance Considerations

- **Scalability**: AKS enables dynamic scaling of pods to handle varying workloads. The modular architecture allows independent scaling of the web, application, and database layers.

- **Availability**: Nginx ensures high availability by balancing traffic and preventing single points of failure. The use of AKS further enhances availability through its managed Kubernetes infrastructure.

- **Performance**: Redis significantly improves performance by caching frequently accessed data, such as OTPs, reducing the load on Postgres. Nginx optimizes traffic distribution, ensuring efficient utilization of application resources.

This logical architecture aligns with the discussed requirements, leveraging Azure services and open-source technologies to deliver a secure, scalable, and high-performing production environment for the Contoso application.

Figure: Production Current Logical View

# 3.2 QA Logical Architecture

The following provides the logical architecture view of the QA environment.

### Logical Architecture for Contoso QA Environment

The Contoso application in the QA environment is structured as a three-tier architecture, comprising a web layer, application layer, and database layer. The architecture leverages Azure Kubernetes Service (AKS) for compute, with application components deployed as pods within namespaces. Key components include Nginx as a load balancer, Redis for caching, and Postgres for persistent data storage. This setup ensures modularity, scalability, and efficient resource utilization.

#### Key Components and Technologies

1. **Web Layer**: Nginx serves as the load balancer, distributing incoming traffic across Contoso pods in the web layer. This ensures optimized traffic management and supports high availability.

2. **Application Layer**: The Contoso application pods handle core business logic and user interactions. Redis is integrated into this layer to cache OTP-related data, such as generation details and expiration policies, improving authentication efficiency.

3. **Database Layer**: Postgres is used for storing user information and other persistent data. It provides robust data management capabilities and supports the application's security requirements.

#### Environment-Specific Requirements

The QA environment mirrors production in terms of architecture to ensure consistency during testing. Security requirements include two-factor authentication, authorization mechanisms, and caching of OTPs with expiration policies. Redis plays a critical role in meeting these requirements by providing fast, in-memory caching for OTP data. Additionally, the architecture supports modular testing of individual components, such as Nginx, Redis, and Postgres, within the AKS cluster.

#### Scalability, Availability, and Performance Considerations

Scalability is achieved through AKS, which allows dynamic scaling of pods based on workload demands. Nginx ensures high availability by distributing traffic evenly across application pods, preventing bottlenecks. Redis enhances performance by reducing database load through caching, particularly for frequently accessed data like OTPs. This caching mechanism also accelerates authentication processes, improving user experience.

#### Disaster Recovery and Monitoring

While disaster recovery strategies and monitoring tools were not explicitly addressed in the transcript, the use of AKS provides inherent resilience through container orchestration and automated failover capabilities. Future considerations for QA could include implementing Azure Monitor or Prometheus for real-time observability and defining a disaster recovery plan to ensure continuity during unexpected failures.

In summary, the Contoso QA environment is designed to test a robust, scalable, and secure application architecture. It leverages modern technologies such as AKS, Nginx, Redis, and Postgres to meet performance, security, and availability requirements while maintaining consistency with production environments.

Figure: QA Current Logical View

# 3.3 Development Logical Architecture

The following provides the logical architecture view of the Development environment.

### Logical Architecture for the Contoso Application – Development Environment

The Contoso application in the Development environment is structured as a three-tier architecture, comprising a web layer, an application layer, and a database layer. The architecture leverages modern cloud-native technologies and services to ensure scalability, performance, and security.

#### Key Components and Technologies

1. **Compute**: The application runs on **Azure Kubernetes Service (AKS)**, where components such as Nginx, Contoso application pods, Redis, and Postgres are deployed in separate namespaces. This containerized approach ensures modularity and scalability.

2. **Networking**: **Nginx** serves as the load balancer, distributing incoming web traffic efficiently across the web layer. This ensures optimal utilization of resources and high availability.

3. **Database**: **Postgres** is used as the primary database for storing user information, ensuring data persistence and reliability.

4. **Caching**: **Redis** is employed for caching, specifically for storing OTP (One-Time Password) generation details. This improves authentication performance by reducing database load and enabling faster access to frequently used data.

5. **Security**: The application enforces robust security measures, including **two-factor authentication (2FA)**, authorization mechanisms, and caching of OTPs with expiration policies to ensure secure and efficient authentication workflows.

#### Environment-Specific Requirements

The Development environment mirrors the logical structure of other environments (e.g., QA, Production) to maintain consistency. However, it is optimized for iterative development and testing. Key requirements include:

- **Rapid Deployment**: The use of AKS and containerized components facilitates frequent updates and testing cycles.

- **Security Testing**: The 2FA implementation and OTP caching mechanisms are critical features that require thorough validation in this environment.

- **Performance Validation**: Redis caching and Nginx load balancing are tested to ensure they meet performance benchmarks before deployment to higher environments.

#### Scalability, Availability, and Performance Considerations

- **Scalability**: AKS provides horizontal scaling capabilities, allowing the application to handle increased workloads by dynamically adjusting the number of pods.

- **Availability**: Nginx ensures high availability by distributing traffic across multiple Contoso pods, preventing single points of failure.

- **Performance**: Redis caching significantly reduces latency for authentication-related operations, while Nginx optimizes traffic distribution to maintain consistent response times.

This logical architecture ensures that the Contoso application in the Development environment is well-equipped for iterative development, testing, and validation, while adhering to the principles of scalability, security, and performance optimization.

Figure: Development Current Logical View

4 Application Network Flow

The following section provides the details for the application network flow required by the application.

# 4.1 Production Application Network Flow

The following diagram provides the application network flow for the Production environment.

### Network Flow Description for Contoso Production Environment

The Contoso application in the Production environment operates within a three-tier architecture, leveraging Azure Kubernetes Service (AKS) for compute. The network connectivity patterns are designed to ensure secure and efficient communication between components while adhering to the application's security and performance requirements.

#### **Current Network Connectivity Patterns**

The application uses **Nginx** as a load balancer to manage incoming traffic to the web layer, which consists of Contoso pods running within AKS namespaces. Nginx distributes traffic across the application layers, optimizing web traffic flow. The web layer communicates with the app layer, which handles business logic, and the database layer, which includes **Postgres** for persistent storage and **Redis** for caching OTP-related data. All components are containerized and deployed within AKS pods.

#### **External Systems and Integrations**

The application integrates with external systems for two-factor authentication, requiring OTP generation and caching. Redis is used to store OTPs temporarily, ensuring expiration policies are enforced. While the transcript does not explicitly mention external APIs or third-party services, the OTP caching mechanism suggests potential integration with external authentication providers.

#### **Data Flow and Communication Patterns**

1. **Inbound Traffic:** User requests enter the system via Nginx, which routes traffic to the appropriate Contoso pods in the web layer.

2. **Internal Communication:** The web layer interacts with Redis for caching OTPs and Postgres for user data storage. Redis ensures low-latency access to frequently used data, while Postgres handles persistent storage.

3. **Outbound Traffic:** No explicit outbound traffic patterns are mentioned, but OTP-related data may be sent to external authentication systems if integrated.

#### **Network Requirements, Ports, and Protocols**

- **Nginx Load Balancer:** Requires HTTP/HTTPS (ports 80/443) for inbound traffic.

- **Redis:** Operates on port 6379 for caching data.

- **Postgres:** Uses port 5432 for database communication.

- **AKS Pods:** Internal communication between pods occurs over secure Kubernetes networking, leveraging cluster-internal DNS for service discovery.

#### **Security and Connectivity Concerns**

The application enforces **two-factor authentication** and OTP caching with expiration policies to ensure secure user authentication. Network security is enhanced by isolating components within AKS namespaces, minimizing cross-component exposure. Additionally, secure protocols (e.g., HTTPS) should be used for all external-facing traffic to protect sensitive user data.

This network flow ensures scalability, security, and performance optimization for the Contoso application in the Production environment.

Figure: Production Application Network Flow Diagram

|  |  |
| --- | --- |
| Step | Details |
| 1 | The planned networking components include Nginx as a load balancer in front of the web layer, which consists of Contoso pods. This setup is consistent across environments (QA, production, etc.). |

# 4.2 QA Application Network Flow

The following diagram provides the application network flow for the QA environment.

### Network Flow Description for Contoso Application in QA Environment

The Contoso application in the QA environment operates within a three-tier architecture, with distinct web, application, and database layers. The network connectivity patterns are centered around Azure Kubernetes Service (AKS), where application components such as Nginx, Contoso, Redis, and Postgres are deployed in pods within dedicated namespaces.

#### Current Network Connectivity Patterns

Nginx serves as the load balancer at the web layer, managing inbound traffic and distributing it across Contoso application pods. This ensures efficient traffic handling and high availability. The application layer communicates with Redis for caching OTP-related data and with Postgres for persistent storage of user information. Redis and Postgres are also deployed as pods within the AKS cluster, ensuring low-latency communication within the cluster.

#### External Systems and Integrations

No external systems or integrations were explicitly mentioned in the transcript. However, the application relies on Azure Kubernetes Service for compute and internal networking.

#### Data Flow and Communication Patterns

1. **Inbound Traffic**: User requests are routed through Nginx, which acts as the entry point and load balancer.

2. **Web to App Layer**: Nginx forwards requests to Contoso pods in the application layer.

3. **App to Database Layer**: The application layer interacts with Postgres for user data storage and Redis for caching OTPs and other ephemeral data.

4. **Caching**: Redis is used to store OTPs with expiration policies, optimizing authentication performance.

#### Network Requirements, Ports, and Protocols

- **Nginx**: Listens on standard HTTP/HTTPS ports (80/443) for inbound traffic.

- **Redis**: Communicates over its default port (6379) for caching operations.

- **Postgres**: Operates on its default port (5432) for database queries.

- All communication within the AKS cluster occurs over secure internal networking.

#### Security and Connectivity Concerns

The application enforces two-factor authentication, requiring secure communication between components. OTPs are cached in Redis with expiration policies to prevent misuse. Proper namespace isolation within AKS ensures that pods are segregated by function, enhancing security. Additionally, the use of Nginx as a load balancer ensures secure and efficient traffic distribution.

This network flow ensures that the Contoso application in the QA environment is optimized for performance, security, and scalability.

Figure: QA Application Network Flow Diagram

# 4.3 Development Application Network Flow

The following diagram provides the application network flow for the Development environment.

### Network Flow Description for Contoso Application in the Development Environment

The Contoso application in the Development environment operates within a three-tier architecture, leveraging Azure Kubernetes Service (AKS) for compute. The network connectivity patterns are structured to ensure secure and efficient communication between components.

#### **Current Network Connectivity Patterns**

The application uses Nginx as a load balancer at the web layer, which routes incoming traffic to Contoso pods running within AKS namespaces. The web layer communicates with the application layer, which includes Redis for caching and Postgres for database operations. Redis is used to cache OTP (One-Time Password) generation details, while Postgres stores user information. All components are containerized and run within AKS pods, ensuring isolated and scalable deployments.

#### **External Systems or Integrations**

No external systems or third-party integrations were explicitly mentioned in the transcript. However, the application relies on Redis and Postgres, which are internal to the AKS cluster.

#### **Data Flow and Communication Patterns**

1. **Ingress Traffic**: External client requests are routed through Nginx, which acts as the ingress controller and load balancer. Nginx distributes traffic to the Contoso pods in the web layer.

2. **Web to App Layer**: The web layer communicates with Redis for caching OTP-related data and Postgres for user authentication and information retrieval.

3. **App to Database Layer**: The application layer interacts with Postgres for persistent storage of user data and Redis for temporary caching.

#### **Network Requirements, Ports, and Protocols**

- **Nginx**: Listens on standard HTTP/HTTPS ports (80/443) for incoming traffic.

- **Redis**: Communicates over port 6379 for caching operations.

- **Postgres**: Operates on port 5432 for database queries.

- All communication within the AKS cluster uses secure internal networking.

#### **Security and Connectivity Concerns**

The application enforces two-factor authentication, requiring secure communication between components. OTPs are cached in Redis with expiration policies to enhance security. Proper namespace isolation within AKS ensures that pods are segregated by environment. Additionally, Nginx provides a secure ingress point, and internal traffic between components is restricted to necessary ports and protocols.

This network flow ensures scalability, security, and performance optimization for the Contoso application in the Development environment.

Figure: Development Application Network Flow Diagram

5 Proposed Architecture in Azure

The following section details the proposed architecture per environment of the application when being migrated to Azure.

# 5.1 Production Proposed Architecture

The following diagram represents the proposed architecture for the Production environment.

### Proposed Azure Architecture for Contoso Production Environment

Based on the application assessment, the following Azure architecture is recommended for the Contoso production environment. This architecture aligns with the requirements, constraints, and technology stack discussed in the transcript.

---

#### **Compute Resources**

The application components (Nginx, Contoso, Redis, Postgres) are containerized and run within namespaces. To support this architecture, **Azure Kubernetes Service (AKS)** is the ideal choice for compute resources. AKS provides a managed Kubernetes environment that simplifies deployment, scaling, and management of containerized applications. Key recommendations include:

- **AKS Cluster**: Deploy an AKS cluster with node pools optimized for production workloads. Use autoscaling to handle traffic spikes and ensure high availability.

- **Namespaces**: Organize application components into separate namespaces within the AKS cluster for logical isolation and management.

- **Pod Scaling**: Configure horizontal pod autoscaling for critical components like Nginx and Contoso pods to ensure scalability during peak loads.

---

#### **Database**

The application uses Postgres for storing user information and Redis for caching OTP-related data. Azure offers managed services for both technologies:

- **Azure Database for PostgreSQL**: Use the flexible server deployment option for high availability, automated backups, and scaling. Configure geo-replication for disaster recovery to ensure data resilience.

- **Azure Cache for Redis**: Deploy Redis as a managed service to handle caching requirements. Configure Redis with appropriate expiration policies for OTP caching and enable clustering for improved performance and scalability.

---

#### **Networking**

The application uses Nginx as a load balancer in front of the web layer. Azure services can enhance this setup:

- **Azure Application Gateway**: Replace or complement Nginx with Application Gateway, a Layer 7 load balancer that provides SSL termination, Web Application Firewall (WAF) capabilities, and traffic routing. This ensures secure and optimized traffic distribution across the web layer.

- **Azure Virtual Network (VNet)**: Deploy all resources within a VNet to ensure secure communication between components. Use subnets to isolate the web, application, and database layers.

- **Private Endpoints**: Configure private endpoints for Azure Database for PostgreSQL and Azure Cache for Redis to ensure secure access within the VNet.

---

#### **Security**

Security is a critical requirement for the Contoso application, including two-factor authentication and OTP caching. Azure services can address these needs:

- **Azure Active Directory (Azure AD)**: Implement Azure AD for user authentication and authorization. Use Azure AD Multi-Factor Authentication (MFA) to enforce two-factor authentication.

- **Azure Key Vault**: Store sensitive information such as API keys, connection strings, and certificates securely in Key Vault. Integrate Key Vault with AKS for secure access to secrets.

- **Network Security Groups (NSGs)**: Apply NSGs to VNet subnets to restrict inbound and outbound traffic based on security rules.

- **Azure Firewall**: Deploy Azure Firewall to provide centralized network security and protect against threats.

---

#### **Monitoring**

Although monitoring tools were not explicitly discussed, operational visibility is essential for production environments. Recommended Azure services include:

- **Azure Monitor**: Use Azure Monitor to collect and analyze metrics and logs from AKS, Postgres, Redis, and other resources. Configure alerts for critical events.

- **Azure Log Analytics**: Integrate Log Analytics with Azure Monitor to enable advanced querying and visualization of logs.

- **Azure Application Insights**: Enable Application Insights for real-time performance monitoring of the Contoso application. Use it to track response times, dependencies, and failures.

---

#### **Scalability and Disaster Recovery**

To ensure scalability and resilience:

- **AKS Autoscaling**: Configure AKS to scale pods and node pools automatically based on demand.

- **Geo-Replication**: Enable geo-replication for Azure Database for PostgreSQL to support disaster recovery.

- **Azure Traffic Manager**: Use Traffic Manager to distribute traffic across multiple regions for high availability and failover.

---

### Summary of Azure Services

- **Compute**: Azure Kubernetes Service (AKS)

- **Database**: Azure Database for PostgreSQL, Azure Cache for Redis

- **Networking**: Azure Application Gateway, Azure Virtual Network, Private Endpoints

- **Security**: Azure Active Directory, Azure Key Vault, Network Security Groups, Azure Firewall

- **Monitoring**: Azure Monitor, Azure Log Analytics, Azure Application Insights

This architecture ensures scalability, security, and operational efficiency while meeting the specific requirements of the Contoso production environment.

Figure: Production Proposed Architecture Diagram

# 5.2 QA Proposed Architecture

The following diagram represents the proposed architecture for the QA environment.

### Proposed Azure Architecture for Contoso QA Environment

Based on the application assessment, the following Azure architecture is proposed for the QA environment of the Contoso application. This architecture aligns with the technologies, requirements, and constraints discussed in the transcript.

---

#### **Compute Resources**

The Contoso application uses Azure Kubernetes Service (AKS) for compute, as the application components (Nginx, Contoso, Redis, Postgres) are containerized and run in pods within namespaces. For the QA environment:

- **Azure Kubernetes Service (AKS):** Deploy AKS to host the containerized application components. Use a cluster with node pools optimized for QA workloads, ensuring cost efficiency while maintaining performance.

- **Azure Container Registry (ACR):** Store and manage container images for the application. Integrate ACR with AKS for seamless deployment.

- **Azure Virtual Machine Scale Sets (VMSS):** Back the AKS cluster with VMSS to enable auto-scaling based on workload demands in the QA environment.

---

#### **Database**

The application uses Postgres for persistent data storage and Redis for caching. The following Azure services are recommended:

- **Azure Database for PostgreSQL - Flexible Server:** Use this fully managed Postgres service to store user information. Enable high availability and automated backups to ensure data durability in the QA environment.

- **Azure Cache for Redis:** Deploy Redis as a managed caching solution to store OTP generation details and other frequently accessed data. Configure expiration policies for OTPs to meet security requirements.

---

#### **Networking**

The application uses Nginx as a load balancer in front of the web layer. The following Azure networking components are recommended:

- **Azure Application Gateway:** Replace or complement Nginx with Azure Application Gateway, which provides a scalable and secure Layer 7 load balancer. Use the Web Application Firewall (WAF) feature to protect against common web vulnerabilities.

- **Azure Virtual Network (VNet):** Deploy all resources within a VNet to ensure secure communication between components. Use subnets to isolate the web, application, and database layers.

- **Azure Private Link:** Enable private connectivity between the AKS cluster, Postgres, and Redis services to enhance security.

---

#### **Security**

The application requires two-factor authentication, OTP caching, and secure communication. The following Azure services address these needs:

- **Azure Active Directory (Azure AD):** Implement Azure AD for user authentication and enable two-factor authentication (2FA) to meet security requirements.

- **Azure Key Vault:** Store sensitive information such as API keys, database credentials, and encryption keys securely.

- **Azure Firewall:** Deploy Azure Firewall to control and monitor traffic between the QA environment and external networks.

- **Network Security Groups (NSGs):** Apply NSGs to subnets and network interfaces to restrict traffic flow based on security rules.

---

#### **Monitoring**

Although monitoring tools were not explicitly discussed, the following Azure services are recommended to ensure operational visibility:

- **Azure Monitor:** Use Azure Monitor to collect metrics and logs from AKS, Postgres, Redis, and other resources. Set up alerts for critical events.

- **Azure Log Analytics:** Centralize log data for analysis and troubleshooting. Integrate with AKS and other services for end-to-end observability.

- **Azure Application Insights:** Enable Application Insights for the Contoso application to monitor performance, detect anomalies, and track user behavior.

---

#### **Scalability and Performance Optimization**

- **Horizontal Pod Autoscaler (HPA):** Configure HPA in AKS to scale pods based on CPU or memory usage, ensuring the application can handle varying QA workloads.

- **Redis Caching:** Use Azure Cache for Redis to reduce database load and improve response times for OTP-related operations.

- **Load Balancing:** Use Azure Application Gateway or Nginx to distribute traffic efficiently across the web layer.

---

This architecture ensures that the QA environment for the Contoso application is scalable, secure, and optimized for performance while leveraging Azure's managed services to reduce operational overhead.

Figure: QA Proposed Architecture Diagram

# 5.3 Development Proposed Architecture

The following diagram represents the proposed architecture for the Development environment.

### Proposed Azure Architecture for Contoso Development Environment

Based on the application assessment, the following Azure architecture is recommended for the Contoso development environment. This architecture aligns with the technologies, requirements, and constraints discussed in the transcript.

---

#### **Compute Resources**

The application components (Nginx, Contoso, Redis, Postgres) are containerized and run within namespaces. To support this architecture, **Azure Kubernetes Service (AKS)** is the ideal choice for compute resources. AKS provides a managed Kubernetes environment that simplifies deployment, scaling, and management of containerized applications. Key features include:

- **Namespace Isolation**: AKS supports namespace-based isolation for organizing and securing application components.

- **Pod Autoscaling**: AKS enables horizontal pod autoscaling to handle varying workloads in the development environment.

- **Integration with Azure DevOps**: AKS integrates seamlessly with CI/CD pipelines for automated deployments.

---

#### **Database**

The application uses Postgres for storing user information and Redis for caching OTP-related data. The following Azure services are recommended:

1. **Azure Database for PostgreSQL**: A fully managed relational database service that supports high availability, automated backups, and scaling. Key features include:

- Flexible server deployment for development workloads.

- Built-in security features like encryption at rest and in transit.

2. **Azure Cache for Redis**: A managed Redis service that provides low-latency data caching. It supports:

- High-performance caching for OTP generation and expiration policies.

- Integration with AKS for seamless communication between application layers.

---

#### **Networking**

The application uses Nginx as a load balancer to distribute web traffic across the web layer (Contoso pods). The following Azure networking components are recommended:

1. **Azure Application Gateway**: A managed load balancer that provides Layer 7 routing, SSL termination, and Web Application Firewall (WAF) capabilities. It can replace or complement Nginx for enhanced security and scalability.

2. **Azure Virtual Network (VNet)**: A secure network boundary for AKS, Postgres, and Redis. Subnets can be configured to isolate resources and enforce network security policies.

3. **Azure Private Link**: Ensures secure connectivity between AKS, Postgres, and Redis without exposing resources to the public internet.

---

#### **Security**

The application requires two-factor authentication, authorization, and caching of OTPs with expiration policies. The following Azure services address these requirements:

1. **Azure Active Directory (Azure AD)**: Provides identity and access management with support for two-factor authentication. Azure AD can be integrated with the application for secure user authentication.

2. **Azure Key Vault**: Manages sensitive information such as API keys, certificates, and secrets. It ensures secure storage and access control for OTP-related data.

3. **Azure Firewall**: Protects the development environment by enforcing network security rules and filtering traffic.

---

#### **Monitoring**

Although monitoring tools were not explicitly addressed in the transcript, the following Azure services are recommended to ensure operational visibility:

1. **Azure Monitor**: Provides comprehensive monitoring for AKS, Postgres, Redis, and other Azure resources. Features include:

- Metrics and logs collection for performance analysis.

- Alerts and dashboards for proactive issue resolution.

2. **Azure Log Analytics**: Enables centralized log management and querying for troubleshooting and auditing purposes.

3. **Container Insights**: Monitors AKS clusters, providing insights into pod performance, resource utilization, and health.

---

#### **Scalability**

The architecture supports scalability through:

- **AKS Autoscaling**: Automatically adjusts the number of pods based on workload demands.

- **Azure Database for PostgreSQL Scaling**: Allows vertical and horizontal scaling to accommodate growing data needs.

- **Redis Cache Scaling**: Ensures low-latency performance even during peak traffic.

---

### Summary

The proposed Azure architecture leverages managed services like AKS, Azure Database for PostgreSQL, Azure Cache for Redis, and Azure Application Gateway to meet the application's compute, database, networking, security, and monitoring requirements. This architecture ensures scalability, security, and operational efficiency for the Contoso development environment while aligning with the technology stack and frameworks discussed.

Figure: Development Proposed Architecture Diagram

6 Architecture Heatmap

Architectural heatmap is a high-level ranking of key concerns that are relevant to application migration to Azure.

|  |  |  |
| --- | --- | --- |
| Area | Notes | Ranking |
| Overall Complexity | Low complexity identified from Q&A analysis (score: 0) | Low |
| App Remediation | Application modifications required | Low |
| Data Migration | Standard data migration approach | Low |
| Network Configuration | Network and security setup | Medium |
| Integration Complexity | Standard integration approach | Low |

7 Decision Matrix

**Migration Decision Matrix**

The following matrix outlines the key decisions made during the assessment:

|  |  |  |  |
| --- | --- | --- | --- |
| **Decision Area** | **Options Considered** | **Selected Approach** | **Rationale** |
| Migration Strategy | Rehost vs Replatform vs Refactor | Assessment-based strategy (requires detailed analysis) | Strategy selection for Contoso requires analysis of technical complexity and business timeline from assessment |
| Compute Platform | Azure VMs vs App Service vs Container Apps | Platform selection based on architecture analysis | Compute platform recommendation depends on application architecture patterns identified in assessment |

**Key Decisions Rationale:**

* • Adopt Replatform migration approach: Replatform approach recommended based on existing modern technology stack and containerization readi...
* • Maintain compatibility with existing processes and tools during Replatform migration
* • Ensure minimal business disruption during transition
* • Enable future cloud optimization opportunities through Replatform approach

8 Application Allocation and Scheduling

The application allocation and scheduling cover the final decisions regarding the application to be migrated.

|  |  |  |  |
| --- | --- | --- | --- |
| Move Group | Wave Allocation | Scheduling | Migration Factory |
| Wave 1 - Core Applications | Wave 1 | Month 2-3 | Azure Migrate Service |

9 Appendix

# 9.1 Additional Backlog Items

List any additional work items that needs to be included to complete the migration

|  |  |
| --- | --- |
| Area | Final Decision |
| Migration Tooling | Azure Migrate + Azure Site Recovery |
| Planning Dependencies | Network and security configuration first |
| Resource Allocation | Dedicated migration team |
| Testing Strategy | Parallel testing environment |

# 9.2 Application and Infrastructure RBAC Information

The following tables provides the RBAC information for the application and infrastructure it's hosted on.

## 9.2.1 Production Application and Infrastructure RBAC

|  |  |  |
| --- | --- | --- |
| Areas | Role | Access List |
| Application | Administrator | Contoso Admins |
| Application | User | Contoso Users |
| Infrastructure | Administrator | Infrastructure Admins |
| Database | Administrator | Database Admins |

## 9.2.2 QA Application and Infrastructure RBAC

|  |  |  |
| --- | --- | --- |
| Areas | Role | Access List |
| Application | Administrator | Contoso Admins |
| Application | User | Contoso Users |
| Infrastructure | Administrator | Infrastructure Admins |
| Database | Administrator | Database Admins |

## 9.2.3 Development Application and Infrastructure RBAC

|  |  |  |
| --- | --- | --- |
| Areas | Role | Access List |
| Application | Administrator | Contoso Admins |
| Application | User | Contoso Users |
| Infrastructure | Administrator | Infrastructure Admins |
| Database | Administrator | Database Admins |

# 9.3 Azure Services RBAC Information

The following tables provides the Azure RBAC information for the Azure services to be configured when hosting the application.

## 9.3.1 Production Azure Services RBAC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | User ID | User Email address | Access Type | Roles |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |

## 9.3.2 QA Azure Services RBAC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | User ID | User Email address | Access Type | Roles |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |

## 9.3.3 Development Azure Services RBAC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | User ID | User Email address | Access Type | Roles |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |
| To be determined | TBD | TBD | Reader Access | Application / Infra/ Testing |

# 9.4 Azure Tagging

The following tables provides the Azure tagging information to be used when applying the Azure Tags to the application components.

## 9.4.1 Production Azure Tagging

|  |  |  |  |
| --- | --- | --- | --- |
| Tag Name | Type | Description | Value |
| environment | Free text (3-15 char) | Cost allocation and reporting. | production |

## 9.4.2 QA Azure Tagging

|  |  |  |  |
| --- | --- | --- | --- |
| Tag Name | Type | Description | Value |
| environment | Free text (3-15 char) | Cost allocation and reporting. | qa |

## 9.4.3 Development Azure Tagging

|  |  |  |  |
| --- | --- | --- | --- |
| Tag Name | Type | Description | Value |
| environment | Free text (3-15 char) | Cost allocation and reporting. | development |

# 9.5 Source Migration Delivery Information

The following tables provide the source migration delivery information to support the migration per environment.

## 9.5.1 Production Source Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Server Specifications | Container runtime environment documentation for Contoso, including Docker/Kubernetes configurations and resource requirements |
| Authentication Systems | Current authentication mechanism documentation for Contoso, including user management and access control |
| Backup and Recovery | Production backup strategy documentation, including RTO/RPO requirements, backup schedules, and recovery procedures |
| Network Configuration | Current network architecture for Contoso, including firewall rules, load balancer configuration, and integration endpoints |
| Application Deployment | Container deployment documentation, including image repositories, CI/CD pipelines, and orchestration configurations |
| Configuration Management | Application configuration documentation for Production, including environment variables, connection strings, and feature flags |
| Data Migration Requirements | Database migration specifications for Postgres, Redis, including schema, data volume, and migration strategy |

## 9.5.2 QA Source Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Server Specifications | Container runtime environment documentation for Contoso, including Docker/Kubernetes configurations and resource requirements |
| Authentication Systems | Current authentication mechanism documentation for Contoso, including user management and access control |
| Backup and Recovery | Backup and recovery procedures for QA environment, including data protection and restoration processes |
| Network Configuration | Current network architecture for Contoso, including firewall rules, load balancer configuration, and integration endpoints |
| Application Deployment | Container deployment documentation, including image repositories, CI/CD pipelines, and orchestration configurations |
| Configuration Management | Application configuration documentation for QA, including environment variables, connection strings, and feature flags |
| Data Migration Requirements | Database migration specifications for Postgres, Redis, including schema, data volume, and migration strategy |

## 9.5.3 Development Source Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Server Specifications | Container runtime environment documentation for Contoso, including Docker/Kubernetes configurations and resource requirements |
| Authentication Systems | Current authentication mechanism documentation for Contoso, including user management and access control |
| Backup and Recovery | Backup and recovery procedures for Development environment, including data protection and restoration processes |
| Network Configuration | Current network architecture for Contoso, including firewall rules, load balancer configuration, and integration endpoints |
| Application Deployment | Container deployment documentation, including image repositories, CI/CD pipelines, and orchestration configurations |
| Configuration Management | Application configuration documentation for Development, including environment variables, connection strings, and feature flags |
| Data Migration Requirements | Database migration specifications for Postgres, Redis, including schema, data volume, and migration strategy |

# 9.6 Target Migration Delivery Information

The following tables provide the target migration delivery information to support the migration per environment.

## 9.6.1 Production Target Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Azure Compute Services | Azure Kubernetes Service (AKS) for Contoso, including node pools, auto-scaling, and container orchestration |
| Azure Database Services | Azure database service selection based on Contoso data requirements and performance needs |
| Networking and Security | Azure Application Gateway, Network Security Groups, Azure Firewall, and private endpoints for production security and traffic management |
| Monitoring and Management | Azure Monitor for containers, Application Insights, Log Analytics workspace, and Kubernetes monitoring solutions |
| Backup and Recovery | Azure Backup with policy-based backup, Azure Site Recovery for disaster recovery, and geo-redundant storage for critical data protection |
| Cost Management | Azure Cost Management with budget alerts, Reserved Instances for predictable workloads, and cost optimization recommendations |
| DevOps Integration | Azure DevOps with container-based CI/CD pipelines, Azure Container Registry, and Infrastructure as Code deployment |

## 9.6.2 QA Target Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Azure Compute Services | Azure Kubernetes Service (AKS) for Contoso, including node pools, auto-scaling, and container orchestration |
| Azure Database Services | Azure database service selection based on Contoso data requirements and performance needs |
| Networking and Security | Network Security Groups, Azure Load Balancer, and secure network configuration for QA environment |
| Monitoring and Management | Azure Monitor for containers, Application Insights, Log Analytics workspace, and Kubernetes monitoring solutions |
| Backup and Recovery | Azure Backup configuration for QA, automated backup policies, and point-in-time recovery capabilities |
| Cost Management | Cost monitoring and budget management for QA, appropriate SKU selection, and resource optimization |
| DevOps Integration | Azure DevOps with container-based CI/CD pipelines, Azure Container Registry, and Infrastructure as Code deployment |

## 9.6.3 Development Target Delivery Information

|  |  |
| --- | --- |
| Requirements | Comments |
| Azure Compute Services | Azure Kubernetes Service (AKS) for Contoso, including node pools, auto-scaling, and container orchestration |
| Azure Database Services | Azure database service selection based on Contoso data requirements and performance needs |
| Networking and Security | Network Security Groups, Azure Load Balancer, and secure network configuration for Development environment |
| Monitoring and Management | Azure Monitor for containers, Application Insights, Log Analytics workspace, and Kubernetes monitoring solutions |
| Backup and Recovery | Azure Backup configuration for Development, automated backup policies, and point-in-time recovery capabilities |
| Cost Management | Development cost controls including auto-shutdown policies, dev/test pricing, and resource lifecycle management |
| DevOps Integration | Azure DevOps with container-based CI/CD pipelines, Azure Container Registry, and Infrastructure as Code deployment |