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 Development Logical Architecture..................................12  
4 Application Network Flow..................................................13  
4.1 Development Application Network Flow..............................14  
5 Proposed Architecture in Azure........................................15  
5.1 Development Proposed Architecture.................................16  
6 Architecture Heatmap.......................................................17  
7 Decision Matrix............................................................18  
8 Application Allocation and Scheduling...................................19  
9 Appendix......................................................................20  
9.1 Additional Backlog Items.............................................20  
9.2 Application and Infrastructure RBAC Information....................20  
9.2.1 Development Application and Infrastructure RBAC.................21  
9.3 Azure Services RBAC Information...................................22  
9.3.1 Development Azure Services RBAC.................................23  
9.4 Azure Tagging............................................................24  
9.4.1 Development Azure Tagging.......................................25  
9.5 Source Migration Delivery Information..............................26  
9.5.1 Development Source Delivery Information.........................27  
9.6 Target Migration Delivery Information..............................28  
9.6.1 Development Target Delivery Information.........................29

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. **Cost Optimization** - The conversation highlights the use of Redis for caching metrics and user information, such as OTP generation and expiration, to reduce database load and optimize costs. This approach minimizes resource consumption and improves efficiency, aligning with cost-saving goals.

2. **Security and Compliance** - Security requirements such as two-factor authentication, authorization, and caching OTPs with expiration policies are explicitly mentioned. Azure's robust security features and compliance certifications make it an ideal platform to meet these requirements while ensuring data protection and regulatory adherence.

3. **Scalability and Performance** - The use of Kubernetes pods for compute, Nginx as a load balancer, and Redis for caching demonstrates a focus on scalable architecture and performance optimization. Azure's ability to dynamically scale resources and handle high traffic loads supports these needs effectively.

4. **Modernization Objectives** - The deployment of applications in a three-tier architecture (web layer, app layer, and database layer) and the use of modern technologies like Kubernetes, Redis, and Postgres indicate a push toward modernizing the application stack. Migrating to Azure enables access to cutting-edge cloud-native services and tools.

5. **Operational Efficiency** - The conversation emphasizes DevOps practices such as load balancing with Nginx, caching with Redis, and structured data storage in Postgres. These practices streamline operations, improve authentication efficiency, and enhance overall system reliability, which are key drivers for migration to Azure.

While disaster recovery and business continuity are not explicitly addressed, the migration to Azure inherently provides access to robust disaster recovery solutions, which could be a secondary driver inferred from the context.

# 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 cloud-native application by leveraging managed Azure services while maintaining the current architecture and containerized deployment model.

**Rationale:**

The current technology stack is already cloud-native, containerized, and deployed on Azure Kubernetes Service (AKS). A Replatform approach allows the organization to optimize its use of Azure services, such as managed databases and caching, without requiring significant changes to the application code or architecture. This balances speed, cost, and long-term benefits by reducing operational overhead and improving scalability and reliability.

**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

**Key Migration Considerations:**

• Ensure compatibility of the Postgres database with Azure Database for PostgreSQL.

• Migrate Redis to Azure Cache for Redis to reduce operational complexity.

• Optimize AKS configurations for cost and performance, including autoscaling and monitoring.

**Recommended Migration Phases:**

1. Phase 1: Assessment and Planning - Evaluate the current AKS setup, database, and caching configurations. Identify dependencies and create a migration plan.

2. Phase 2: Migration of Managed Services - Migrate Postgres to Azure Database for PostgreSQL and Redis to Azure Cache for Redis. Test the application for compatibility and performance.

3. Phase 3: Optimization and Monitoring - Optimize AKS configurations, implement Azure Monitor for observability, and fine-tune the application for cost and performance.

## 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**

The current technology stack for the Contoso application includes the following components:

### **Technologies and Frameworks**

- **Web Framework**: Django-based web applications

- **Web Server**: Nginx (used as a load balancer)

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

- **Database**: PostgreSQL (used for storing user information)

### **Infrastructure Components**

- **Compute**: Kubernetes pods running the application components (Nginx, Django, Redis, and Postgres)

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

- **Storage**: Redis for caching and PostgreSQL for persistent data storage

### **Security Requirements**

- Two-factor authentication (2FA)

- Authorization mechanisms

- Caching of OTPs with expiration policies for secure authentication

### **Monitoring Tools**

- Redis (for caching metrics and OTPs)

- PostgreSQL (for storing user-related metrics)

### **Compliance Requirements**

- Implementation of 2FA and secure OTP management

- Secure storage of user information in PostgreSQL

---

## 2. **Migration Strategy Recommendation**

Based on the current stack and requirements, the recommended migration approach is **Replatforming** with selective **Refactoring** for modernization. This approach allows leveraging Azure's managed services to reduce operational overhead while introducing cloud-native capabilities for scalability, security, and cost optimization.

### **Key Migration Steps**

1. **Replatform**:

- Migrate PostgreSQL to Azure Database for PostgreSQL (managed service).

- Migrate Redis to Azure Cache for Redis (managed service).

- Deploy the application on Azure Kubernetes Service (AKS) for container orchestration.

2. **Refactor**:

- Replace Nginx with Azure Application Gateway for load balancing and enhanced security features.

- Introduce Azure Active Directory (Azure AD) for implementing two-factor authentication and authorization.

---

## 3. **Azure Services Recommendations**

The following Azure services are recommended to map the current technology stack and meet the application's requirements:

### **Compute**

- **Azure Kubernetes Service (AKS)**: Migrate the Kubernetes pods to AKS for containerized application deployment. AKS provides scalability, high availability, and integration with other Azure services.

- **Azure Functions** (optional): For OTP generation and expiration logic, consider using serverless Azure Functions to reduce the load on Redis and improve scalability.

### **Database**

- **Azure Database for PostgreSQL**: Use the managed PostgreSQL service for storing user information. This service provides automated backups, scaling, and high availability.

### **Caching**

- **Azure Cache for Redis**: Migrate Redis to Azure Cache for Redis to handle OTP caching and other metrics. This managed service ensures high performance and reliability.

### **Networking**

- **Azure Application Gateway**: Replace Nginx with Azure Application Gateway for load balancing, SSL termination, and Web Application Firewall (WAF) capabilities.

- **Azure Front Door** (optional): For global load balancing and enhanced performance, consider Azure Front Door in conjunction with Application Gateway.

### **Security**

- **Azure Active Directory (Azure AD)**: Implement two-factor authentication and authorization using Azure AD. Azure AD B2C can be used for customer-facing applications.

- **Azure Key Vault**: Store sensitive information such as API keys, connection strings, and secrets securely.

### **Monitoring and Observability**

- **Azure Monitor**: Use Azure Monitor for end-to-end observability, including metrics, logs, and alerts for AKS, Redis, and PostgreSQL.

- **Azure Application Insights**: Integrate Application Insights for performance monitoring and diagnostics of the Django application.

### **Disaster Recovery**

- **Azure Site Recovery**: Implement a disaster recovery plan using Azure Site Recovery for AKS and other critical components.

- **Geo-Replication**: Enable geo-replication for Azure Database for PostgreSQL and Azure Cache for Redis to ensure data availability in case of regional outages.

---

## 4. **Architecture Considerations**

### **Design Patterns**

- **Three-Tier Architecture**: Maintain the separation of web, application, and database layers for modularity and scalability.

- **Microservices**: Consider breaking down the Django application into smaller microservices for better scalability and maintainability.

### **Scalability**

- Use AKS's auto-scaling capabilities to handle varying workloads.

- Enable scaling for Azure Cache for Redis and Azure Database for PostgreSQL based on performance metrics.

### **Security**

- Enforce network security using Azure Network Security Groups (NSGs) and private endpoints for Redis and PostgreSQL.

- Use Azure AD for role-based access control (RBAC) and two-factor authentication.

### **Integration**

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

- Use Azure Event Grid or Service Bus for asynchronous communication between components, if needed.

---

## 5. **Modernization Opportunities**

### **Cloud-Native Enhancements**

- **Serverless OTP Management**: Replace Redis-based OTP generation with Azure Functions to reduce dependency on Redis and improve scalability.

- **Managed Load Balancing**: Replace Nginx with Azure Application Gateway for enhanced security and simplified management.

- **Identity Management**: Use Azure AD B2C for a seamless and secure user authentication experience.

### **Cost Optimization**

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

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

- Use Azure Cost Management to monitor and control spending.

---

## 6. **Implementation Phases and Priorities**

### **Phase 1: Assessment and Planning**

- Conduct a detailed assessment of the current Kubernetes deployment.

- Identify dependencies and prepare a migration plan.

### **Phase 2: Database Migration**

- Migrate PostgreSQL to Azure Database for PostgreSQL using Azure Database Migration Service.

- Enable geo-replication and automated backups.

### **Phase 3: Caching Migration**

- Migrate Redis to Azure Cache for Redis.

- Configure Redis with appropriate expiration policies for OTP caching.

### **Phase 4: Compute Migration**

- Deploy the application on AKS.

- Replace Nginx with Azure Application Gateway.

### **Phase 5: Security and Compliance**

- Implement two-factor authentication using Azure AD.

- Secure sensitive data using Azure Key Vault.

### **Phase 6: Monitoring and Optimization**

- Set up Azure Monitor and Application Insights for observability.

- Optimize costs using Azure Cost Management.

---

## Conclusion

The proposed migration strategy and Azure services recommendations align with Contoso's current technology stack and requirements. By leveraging Azure's managed services and cloud-native capabilities, the solution will achieve improved scalability, security, and cost efficiency. The phased implementation plan ensures a smooth transition with minimal disruption to existing operations.

## 1.3.3 Indicative Azure Cost

**Indicative Monthly Azure Costs:**

Based on the assessment discussion:

* • "Caching user information and metrics in Redis to optimize OTP generation and expiration processes" is mentioned as a cost optimization strategy.

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

|  |  |
| --- | --- |
| **Service Category** | **Estimated Monthly Cost** |
| Azure Kubernetes Service (AKS) | $300 - $800 |
| Azure SQL Database (Default) | $200 - $500 |
| 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** | **$850 - $2,150** |

**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

**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 Redis for caching information such as OTP generation and expiration, and Postgres for storing user information.

**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?: The Azure services being used for compute are Kubernetes pods running Nginx, Django-based web applications, Redis, and Postgres.

**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:**

**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 Development Logical Architecture

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

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

The Contoso application’s development environment is designed as a three-tier architecture, leveraging Azure services for compute, storage, and networking. The architecture is optimized for scalability, security, and performance, with specific technologies and frameworks tailored to meet the application's requirements.

#### Current Architecture Overview

The application is deployed on Kubernetes pods in Azure, with Nginx serving as the load balancer in front of the web tier. The web tier hosts Django-based web applications, which handle user interactions and business logic. Redis is utilized for caching critical information, such as One-Time Password (OTP) generation and expiration, while Postgres serves as the primary database for storing user information. This architecture ensures separation of concerns between the web layer, application layer, and database layer, promoting modularity and maintainability.

#### Key Components and Technologies

1. **Compute**: Kubernetes pods running Nginx and Django-based web applications.

2. **Storage**: Redis for caching OTPs and metrics, and Postgres for persistent user data storage.

3. **Networking**: Nginx as a load balancer to distribute traffic across the web tier and ensure high availability.

4. **Security**: Two-factor authentication and authorization mechanisms, with OTP caching in Redis and expiration policies to enforce secure authentication practices.

#### Environment-Specific Requirements

The development environment emphasizes security and compliance, requiring two-factor authentication and proper authorization mechanisms. Redis is configured to cache OTPs with expiration policies, ensuring secure and efficient authentication workflows. Postgres is used to store user information, adhering to compliance requirements for data integrity and security. Monitoring tools, such as Redis and Postgres, are integrated to track metrics and ensure system health during development.

#### Scalability, Availability, and Performance Considerations

The architecture is designed to scale horizontally by deploying additional Kubernetes pods as traffic increases. Nginx ensures availability by distributing traffic across the web tier, preventing bottlenecks. Redis enhances performance by caching frequently accessed data, such as OTPs, reducing database load and improving response times. Postgres is optimized for transactional workloads, ensuring reliable data storage and retrieval.

#### Summary

The Contoso application’s development environment leverages a robust technology stack, including Kubernetes, Nginx, Django, Redis, and Postgres, to meet the requirements for security, compliance, scalability, and performance. The use of Redis for caching and Postgres for persistent storage ensures efficient data management, while the three-tier architecture promotes modularity and maintainability. This environment is well-suited for iterative development and testing, with monitoring tools integrated to ensure system reliability and performance.

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 Development Application Network Flow

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

### Network Flow Description for Contoso Development Environment

The Contoso application in the Development environment leverages a three-tier architecture comprising a web layer, application layer, and database layer. The network connectivity patterns involve communication between Kubernetes pods hosting Nginx, Django-based web applications, Redis, and Postgres. Nginx acts as a load balancer at the web layer, handling incoming HTTP/HTTPS traffic and distributing requests to the Django application pods. The web layer communicates with Redis and Postgres in the application and database layers, respectively, over internal network connections.

Redis is used for caching OTP generation and expiration data, while Postgres stores user information. Redis communicates with the Django application layer over TCP, typically on port 6379, while Postgres uses TCP on port 5432 for database queries. Both Redis and Postgres are internal components, ensuring secure communication within the virtual network.

External integrations are not explicitly mentioned in the transcript, but the application requires secure access for users, necessitating two-factor authentication. This involves OTP generation and caching in Redis, with expiration policies enforced to meet compliance requirements. Security measures include authorization mechanisms and caching sensitive data in Redis to minimize database load and optimize performance.

Data flow begins with user requests entering through Nginx, which forwards them to the Django application layer. The application interacts with Redis for OTP validation and expiration checks, and with Postgres for user data retrieval and storage. All communication occurs over secure internal protocols, with no direct exposure of Redis or Postgres to external networks.

To address security concerns, the environment must enforce strict access controls, including network security groups (NSGs) to restrict access to Redis and Postgres. Two-factor authentication ensures secure user access, while caching in Redis reduces latency and optimizes resource utilization.

Figure: Development Application Network Flow Diagram

|  |  |
| --- | --- |
| Step | Details |
| 1 | Redis and Postgres are planned as networking components. Redis is used for caching information such as OTP generation and expiration, while Postgres is used for storing user information. Additionally, Nginx is mentioned as a load balancer in front of the web layer. |

5 Proposed Architecture in Azure

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

# 5.1 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 uses Kubernetes pods running Nginx and Django-based web applications. To support this, the following Azure services are recommended:

- **Azure Kubernetes Service (AKS):** AKS provides a managed Kubernetes environment to deploy and manage containerized applications. It supports scaling, monitoring, and integration with other Azure services. AKS will host the Nginx load balancer and Django-based web applications.

- **Azure Container Registry (ACR):** Use ACR to store and manage Docker images for the application components, ensuring seamless integration with AKS.

- **Azure Redis Cache (Azure Cache for Redis):** Redis is used for caching OTPs and metrics. Azure Cache for Redis provides a fully managed Redis service with high availability and scalability.

---

#### **Database**

The application uses Postgres for storing user information. The following Azure service is recommended:

- **Azure Database for PostgreSQL:** A fully managed PostgreSQL database service that offers built-in high availability, automated backups, and scaling options. It ensures compliance with security and operational requirements while supporting the application's data storage needs.

---

#### **Networking**

The application requires connectivity between components and a load balancer for the web layer. The following Azure services are recommended:

- **Azure Application Gateway:** Use Application Gateway as a Layer 7 load balancer in front of the web tier. It provides SSL termination, Web Application Firewall (WAF) for enhanced security, and routing capabilities.

- **Azure Virtual Network (VNet):** Create a VNet to securely connect AKS, Redis, and PostgreSQL. Subnets can be defined for the web layer, application layer, and database layer to ensure proper isolation and security.

- **Azure Private Link:** Use Private Link to securely connect Azure Cache for Redis and Azure Database for PostgreSQL to the VNet, ensuring data is transmitted securely without exposure to the public internet.

---

#### **Security**

The application has stringent security requirements, including two-factor authentication, authorization, and caching OTPs with expiration policies. The following Azure services are recommended:

- **Azure Active Directory (Azure AD):** Implement Azure AD for user authentication and authorization. Azure AD supports two-factor authentication and integrates seamlessly with the application.

- **Azure Key Vault:** Use Key Vault to securely store secrets, such as database connection strings and API keys, ensuring compliance with security best practices.

- **Azure Firewall:** Deploy Azure Firewall to protect the VNet and enforce security policies for inbound and outbound traffic.

---

#### **Monitoring**

Monitoring is critical for operational efficiency. The following Azure services are recommended:

- **Azure Monitor:** Use Azure Monitor to collect and analyze metrics and logs from AKS, Redis, and PostgreSQL. It provides insights into application performance and resource utilization.

- **Azure Log Analytics:** Integrate Log Analytics with Azure Monitor to query and analyze logs for troubleshooting and performance optimization.

- **Azure Application Insights:** Use Application Insights to monitor the Django-based web applications, track user behavior, and identify performance bottlenecks.

---

#### **Scalability**

To ensure scalability, the architecture leverages Azure's auto-scaling capabilities:

- **AKS Auto-Scaling:** Configure AKS to automatically scale pods based on resource utilization and traffic patterns.

- **Azure Cache for Redis Scaling:** Use Redis scaling options to handle increased caching demands during peak loads.

- **Azure Database for PostgreSQL Scaling:** Enable vertical and horizontal scaling for the PostgreSQL database to accommodate growing data storage and query requirements.

---

#### **Operational Needs**

The architecture supports DevOps practices and operational efficiency:

- **Azure DevOps:** Use Azure DevOps for CI/CD pipelines to automate application deployment and updates. Integrate with ACR for container image management.

- **Azure Backup:** Enable automated backups for Azure Database for PostgreSQL to ensure data recovery in case of failures.

---

### Summary

This proposed Azure architecture leverages managed services like AKS, Azure Cache for Redis, and Azure Database for PostgreSQL to meet the application's compute, database, networking, security, and monitoring requirements. It ensures scalability, operational efficiency, and compliance with security standards, making it well-suited for the Contoso development environment.

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 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 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 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 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, 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 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 |