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. **Scalability and Performance Optimization**

The conversation highlights the use of Redis for caching OTP-related data and metrics, as well as Nginx as a load balancer to efficiently manage web traffic. These components are designed to optimize authentication processes, reduce database load, and ensure high performance under varying workloads. Migrating to Azure provides access to scalable infrastructure, enabling Contoso to handle increased traffic and user demands seamlessly.

2. **Enhanced Security and Authentication**

Security requirements such as two-factor authentication, authorization, and caching OTP information in Redis are critical for the application. Azure's robust security features, including identity management, encryption, and compliance certifications, align with these needs, ensuring secure handling of user data and authentication processes.

3. **Modernization of Application Architecture**

The application is deployed in a three-tier architecture (web layer, app layer, and database layer) using Kubernetes pods, Nginx, Django-based web applications, Redis, and Postgres. Migrating to Azure allows Contoso to leverage modern cloud-native services, such as Azure Kubernetes Service (AKS) and Azure Database for PostgreSQL, to streamline operations and adopt a more agile, containerized architecture.

4. **Operational Efficiency and DevOps Enablement**

Recommended DevOps practices, such as deploying applications in a three-tier architecture and using Redis for caching, align with Azure's capabilities for automation, CI/CD pipelines, and infrastructure management. Azure's integrated tools and services can help Contoso improve operational efficiency, reduce manual intervention, and accelerate development cycles.

5. **Cost Optimization and Resource Management**

While cost considerations are not explicitly mentioned, the use of Redis for caching and Postgres for data storage implies a focus on optimizing resource utilization and reducing database load. Azure's pay-as-you-go pricing model and resource scaling capabilities can help Contoso minimize costs while ensuring efficient use of compute, storage, and networking resources.

These drivers collectively highlight the strategic benefits of migrating to Azure, enabling Contoso to modernize its infrastructure, enhance security, improve performance, and achieve operational efficiency.

# 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:

**Refactor** - Refactor the application to fully leverage Azure's cloud-native services while maintaining the existing architecture and functionality.

**Rationale:**

The current technology stack is already cloud-ready, containerized, and deployed on Azure Kubernetes Service (AKS). Refactoring allows the application to optimize its use of Azure's managed services, such as Azure Database for PostgreSQL, Azure Cache for Redis, and Azure Application Gateway, while retaining the n-tier architecture and Kubernetes-based deployment model. This approach balances modernization with minimal disruption, ensuring long-term scalability, performance, and cost efficiency.

**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 Django application with Azure-managed services like Azure Database for PostgreSQL.

• Optimize Kubernetes configurations to integrate with Azure-native tools like Azure Monitor and Azure DevOps.

• Plan for seamless migration of Redis to Azure Cache for Redis without data loss or downtime.

**Recommended Migration Phases:**

1. Phase 1: Assessment and Planning - Analyze the current architecture, dependencies, and workloads. Design the migration plan and identify Azure services to replace existing components.

2. Phase 2: Migration and Integration - Migrate the database to Azure Database for PostgreSQL, Redis to Azure Cache for Redis, and configure AKS to integrate with Azure-native tools like Azure Monitor and Azure Application Gateway.

3. Phase 3: Optimization and Validation - Optimize the application for Azure services, conduct performance testing, and validate the system's functionality and scalability in the new environment.

## 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) for storing user information.

- **Caching**: Redis for caching OTP (One-Time Password) information and metrics.

- **Compute**: Kubernetes pods running:

- Nginx (as a load balancer).

- Django-based web applications.

- Redis and Postgres within Kubernetes namespaces (e.g., Contoso and Watman).

- **Security**: Two-factor authentication (2FA) with OTP caching in Redis.

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

- **Monitoring**: Redis and Postgres are used for operational monitoring and caching metrics.

---

## 2. **Migration Strategy Recommendation**

### Recommended Migration Approach:

- **Replatform**: Move the existing workloads to Azure PaaS services where possible to reduce operational overhead while maintaining compatibility with the current stack. This includes migrating Postgres to Azure Database for PostgreSQL and Redis to Azure Cache for Redis.

- **Refactor**: For components like Kubernetes, consider leveraging Azure Kubernetes Service (AKS) to simplify cluster management and integrate with Azure-native services.

- **Rehost**: For components that cannot be immediately replatformed or refactored, use Azure Virtual Machines or Azure Kubernetes Service (AKS) to host the existing workloads.

---

## 3. **Azure Services Recommendations**

### Database:

- **Azure Database for PostgreSQL - Flexible Server**:

- Fully managed Postgres database with high availability and automated backups.

- Supports scaling and compliance requirements.

- Use Azure Database Migration Service (DMS) for seamless migration of the existing Postgres database.

### Caching:

- **Azure Cache for Redis**:

- Fully managed Redis service for caching OTPs and metrics.

- Provides high availability, scalability, and integration with Azure monitoring tools.

### Compute:

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

- Migrate Kubernetes workloads to AKS for simplified cluster management.

- Use namespaces (e.g., Contoso and Watman) to maintain logical separation of workloads.

- Integrate with Azure Monitor for container insights.

### Networking:

- **Azure Application Gateway**:

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

- Provides SSL termination, URL-based routing, and DDoS protection.

### Security:

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

- Implement two-factor authentication (2FA) using Azure AD B2C or Azure AD MFA.

- Integrate with the Django application for user authentication and authorization.

- **Azure Key Vault**:

- Store sensitive information such as database connection strings, API keys, and certificates securely.

### Monitoring:

- **Azure Monitor**:

- Use Azure Monitor for end-to-end observability of AKS, Redis, and Postgres.

- Enable Application Insights for the Django application to monitor performance and diagnose issues.

- **Log Analytics**:

- Centralize logs from all components (AKS, Redis, Postgres) for better visibility and troubleshooting.

### Storage:

- **Azure Blob Storage**:

- Use for storing static assets (e.g., images, documents) if applicable.

- Integrate with Django for serving static and media files.

---

## 4. **Architecture Considerations**

### Design Patterns:

- **Three-Tier Architecture**:

- Web Layer: Azure Application Gateway (or Nginx in AKS).

- Application Layer: Django application running in AKS.

- Database Layer: Azure Database for PostgreSQL and Azure Cache for Redis.

### Scalability:

- Use AKS autoscaling to handle variable workloads.

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

### Security:

- Enforce network security using Azure Network Security Groups (NSGs) and Azure Firewall.

- Use private endpoints for Azure Database for PostgreSQL and Azure Cache for Redis to restrict access to internal networks.

- Implement role-based access control (RBAC) for AKS and other Azure resources.

### Integration:

- Use Azure DevOps or GitHub Actions for CI/CD pipelines to automate deployments to AKS.

- Leverage Azure Service Bus or Event Grid for asynchronous communication if needed in the future.

---

## 5. **Modernization Opportunities**

### Cloud-Native Enhancements:

- **Serverless Functions**:

- Use Azure Functions for OTP generation and expiration logic to reduce dependency on Redis for this functionality.

- **Managed Identity**:

- Use Azure Managed Identity to securely access Azure resources without managing credentials.

- **API Management**:

- Use Azure API Management to expose APIs securely and manage traffic.

### Performance Optimization:

- Enable Redis persistence in Azure Cache for Redis to ensure data durability.

- Use read replicas in Azure Database for PostgreSQL to offload read-heavy operations.

### Cost Optimization:

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

- Enable auto-scaling for AKS and Azure Cache for Redis to optimize resource usage.

---

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

### Phase 1: Assessment and Planning

- Inventory current workloads and dependencies.

- Set up Azure environment (subscriptions, resource groups, networking).

- Plan database migration using Azure Database Migration Service.

### Phase 2: Database and Caching Migration

- Migrate Postgres to Azure Database for PostgreSQL.

- Migrate Redis to Azure Cache for Redis.

### Phase 3: Compute Migration

- Deploy AKS and migrate Kubernetes workloads.

- Configure namespaces and deploy Nginx, Django, Redis, and Postgres workloads.

### Phase 4: Networking and Security

- Replace Nginx with Azure Application Gateway.

- Implement Azure AD for 2FA and secure access.

- Configure private endpoints and NSGs.

### Phase 5: Monitoring and Optimization

- Enable Azure Monitor and Application Insights.

- Optimize performance and costs using scaling and reserved instances.

### Phase 6: Modernization

- Implement serverless functions for OTP management.

- Introduce API Management for secure API exposure.

---

## Conclusion

The proposed migration strategy and Azure services align with Contoso's current technology stack and requirements. By leveraging Azure PaaS and cloud-native services, the solution will achieve improved scalability, security, and operational 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:**

**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
* • 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 metrics and OTP-related data.

**Recommended Database Migration Strategy:**

**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

**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

**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 services being used for compute include Kubernetes pods running Nginx, Django-based web applications, Redis, and Postgres within namespaces such as Contoso and Watman.

**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:**

**BCDR Requirements**: N/A - Specific BCDR requirements not detailed 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 Contoso Development Environment

The Contoso application in the Development environment is designed with a modern, scalable, and secure architecture leveraging Azure cloud services and open-source technologies. The architecture follows a three-tier design, comprising a web layer, an application layer, and a database layer, with a focus on performance optimization, security, and scalability.

#### Current Architecture Overview

The application is deployed on Azure Kubernetes Service (AKS), where Kubernetes pods host the core components of the system. The web layer is managed by **Nginx**, which functions as a load balancer to efficiently distribute incoming traffic to the application layer. The application layer is built using a **Django-based web application**, which handles business logic and user interactions. The database layer consists of **Postgres**, which serves as the primary data store for user information, and **Redis**, which is used for caching metrics and OTP-related data to enhance performance and reduce database load.

#### Key Components and Technologies

1. **Compute**: Kubernetes pods running Nginx, Django, Redis, and Postgres.

2. **Database**: Postgres for persistent storage of user data.

3. **Caching**: Redis for caching OTP generation and expiration details, as well as other metrics.

4. **Networking**: Nginx as a load balancer to manage web traffic.

5. **Monitoring**: Redis and Postgres are utilized for monitoring application performance and data integrity.

#### Environment-Specific Requirements

The Development environment emphasizes security and performance. Two-factor authentication is implemented, with OTPs cached in Redis to ensure proper expiration and validation. User data is securely stored in Postgres. The architecture also supports modularity by organizing components into Kubernetes namespaces, such as "Contoso" and "Watman," to isolate workloads and simplify management.

#### Scalability, Availability, and Performance

To ensure scalability, the application leverages Kubernetes' ability to scale pods dynamically based on traffic and resource demands. Nginx enhances availability by distributing traffic across multiple instances of the web application, preventing single points of failure. Performance is optimized through Redis caching, which reduces the load on Postgres by handling frequent, transient data like OTPs. This caching mechanism also accelerates authentication processes, improving user experience.

In summary, the Contoso Development environment is a robust, cloud-native architecture that integrates Azure services, open-source technologies, and best practices in DevOps to deliver a secure, scalable, and high-performing application.

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 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 web layer is fronted by **Nginx**, which acts as a load balancer to manage incoming HTTP/HTTPS traffic efficiently. Nginx routes requests to Kubernetes pods hosting Django-based web applications within the **Contoso** and **Watman** namespaces. The application layer communicates with **Redis** for caching OTP-related data and metrics, and with **Postgres** for storing user information. Both Redis and Postgres are deployed within the same Kubernetes cluster, ensuring low-latency internal communication.

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

No external systems or third-party integrations were explicitly mentioned in the transcript. However, the application relies on Azure Kubernetes Service (AKS) for compute resources, which may involve external connectivity for Azure APIs and management endpoints.

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

1. **User Authentication:** Incoming user requests are routed through Nginx to the Django application. The application queries Redis for cached OTP information and Postgres for user data during authentication.

2. **Caching and Storage:** Redis handles OTP generation and expiration, reducing load on Postgres by caching frequently accessed data. Postgres serves as the primary database for persistent user information.

3. **Monitoring:** Redis and Postgres are also utilized for monitoring purposes, with Redis caching metrics and Postgres storing logs or user-related data.

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

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

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

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

- Internal communication between Kubernetes pods occurs over the cluster's virtual network, ensuring secure and isolated traffic.

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

To meet security requirements, **two-factor authentication** is implemented, with OTP caching in Redis and user data stored in Postgres. Proper authorization mechanisms are enforced at the application layer. Additionally, Redis and Postgres are deployed within the Kubernetes cluster to minimize exposure to external networks. Secure protocols (HTTPS) are recommended for external communication, and internal traffic is isolated within the cluster's virtual network.

This setup ensures robust security, optimized performance, and efficient data flow for the Contoso application in the Development environment.

Figure: Development Application Network Flow Diagram

|  |  |
| --- | --- |
| Step | Details |
| 1 | The planned networking components include Nginx as a load balancer in front of the web layer, Redis for caching OTP-related information, and Postgres for storing user information. |

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, while ensuring scalability, security, and operational efficiency.

---

#### **Compute Resources**

The application leverages Kubernetes pods running Nginx, Django-based web applications, Redis, and Postgres. To support this, the following Azure services are recommended:

- **Azure Kubernetes Service (AKS):** AKS provides a managed Kubernetes environment to deploy and manage the Nginx load balancer, Django-based web applications, Redis, and Postgres pods. AKS supports namespaces, allowing logical separation of resources such as `Contoso` and `Watman`.

- **Azure Container Registry (ACR):** Use ACR to store and manage Docker images for the Nginx and Django-based web applications, ensuring seamless integration with AKS.

- **Azure Virtual Machine Scale Sets (VMSS):** For additional compute flexibility, VMSS can be used to scale workloads dynamically based on demand, particularly for the web and app layers.

---

#### **Database**

The application uses Postgres for storing user information. To meet this requirement:

- **Azure Database for PostgreSQL – Flexible Server:** This fully managed Postgres service provides high availability, automated backups, and scaling capabilities. It is ideal for storing user data securely and efficiently.

- **Azure Cache for Redis:** Redis is used for caching OTP-related information and metrics. Azure Cache for Redis offers a managed Redis service with built-in monitoring, scaling, and security features.

---

#### **Networking**

The application requires a robust networking setup to manage connectivity between components:

- **Azure Application Gateway:** Replace Nginx as the load balancer with Azure Application Gateway, which provides Layer 7 load balancing, SSL termination, and Web Application Firewall (WAF) capabilities for enhanced security.

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

- **Azure Private Link:** Enable secure access to Azure Database for PostgreSQL and Azure Cache for Redis using Private Link, ensuring traffic remains within the Azure backbone.

---

#### **Security**

Security is a critical requirement for the application, including two-factor authentication and OTP management:

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

- **Azure Firewall:** Deploy Azure Firewall to protect the environment from external threats and enforce network security policies.

- **Azure Security Center:** Use Security Center to monitor and manage security posture across all Azure resources.

---

#### **Monitoring**

Operational monitoring is essential for Redis and Postgres, as well as the overall application:

- **Azure Monitor:** Use Azure Monitor to collect and analyze metrics and logs from AKS, Redis, and Postgres. Configure alerts for critical events such as high CPU usage or database connection failures.

- **Azure Log Analytics:** Integrate Log Analytics with Azure Monitor to centralize log data and enable advanced querying and visualization.

- **Azure Application Insights:** Enable Application Insights for the Django-based web applications to monitor performance, detect anomalies, and trace user interactions.

---

### **Additional Considerations**

- **Scalability:** AKS and Azure Database for PostgreSQL support auto-scaling to handle increased workloads during development and testing phases.

- **Compliance:** While compliance requirements were not explicitly mentioned, Azure services such as Key Vault and Security Center help meet industry standards for data protection and security.

- **Cost Optimization:** Use Azure Cost Management to monitor and optimize resource usage, ensuring the development environment remains cost-effective.

This architecture leverages Azure's managed services to meet the application's requirements while ensuring scalability, security, and operational efficiency. It provides a solid foundation for the Contoso development environment and can be adapted for production use with minimal changes.

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, 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 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 |