

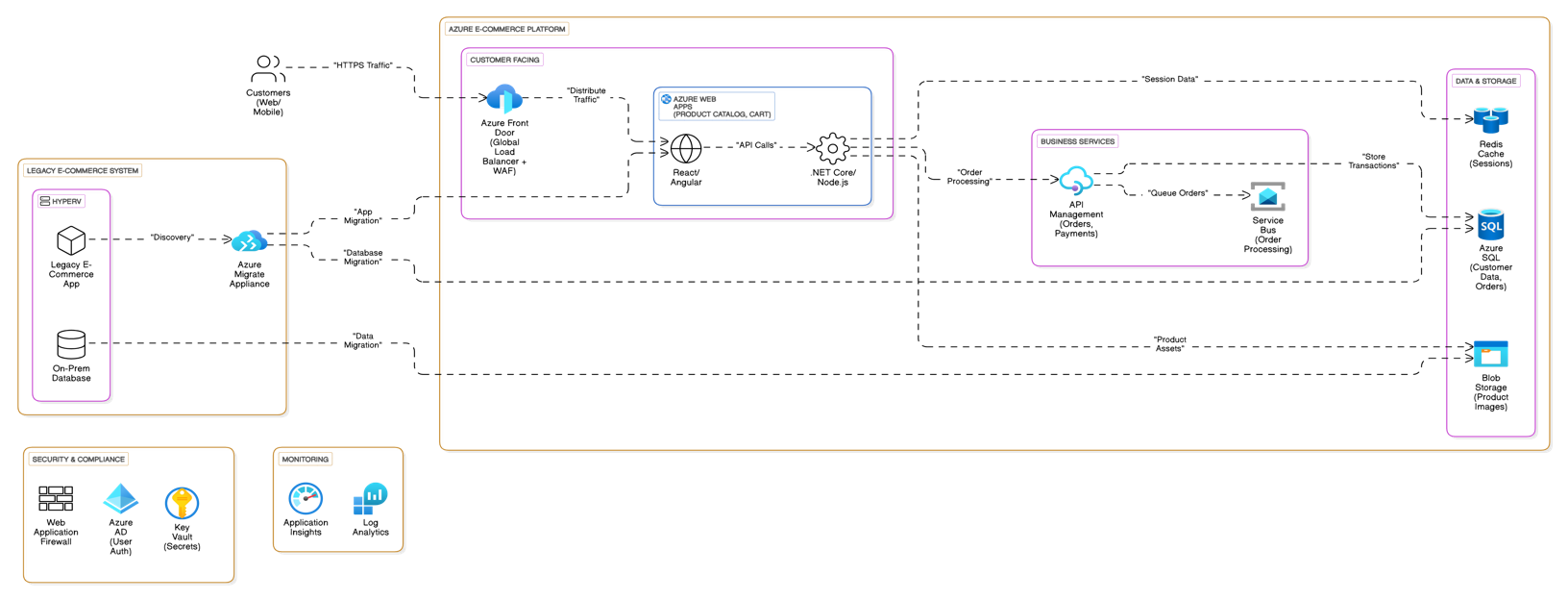
**Design a Migration of a Hyper-V based App to Azure by using Azure Migrate.**

This report outlines the migration of a real-time e-commerce application hosted on a Hyper-V environment to Microsoft Azure using Azure Migrate. The document includes an architectural diagram, the list of services used, an explanation of the application flow, operational aspects, and considerations under the Well-Architected Framework. Additionally, it ensures high availability, scalability, and security as a three-tier architecture.

**Application: Online Retail E-Commerce Platform**

We are migrating an on-premises e-commerce web application from Hyper-V to Azure. The platform allows users to browse products, add them to the cart, and complete online transactions.

**Architectural Diagram:**



The diagram above shows the components and flow of connections for the migration of our on-premises Hyper-V application for the E-commerce business to Azure cloud. The migration is done in phases. Let us break down the migration process:

**Phase 1: Discovery & Assessment**

Objective: Analyze existing systems and plan migration scope.

1. Deploy Azure Migrate Appliance

* Install the Azure Migrate appliance in the on-premises Hyper-V environment.
* Use it to discover:

Legacy app dependencies.

Database schema and data volume.

**2. Analyze Compatibility**

* Identify which components can be "lift-and-shifted" (e.g. legacy app → Azure Web Apps) vs. those needing refactoring.
* Map dependencies (e.g. how the legacy app interacts with the on-prem database).

**Phase 2: Pilot Migration**

**Objective: Validate migration workflows with minimal risk.**

**1. Migrate Static Data**

* Use Azure Blob Storage tools to migrate non-critical data (e.g. product images) from Old Database to BlobStorage.

**2.Test App Migration**

* Rehost a small part of the legacy app (Old App) to Azure Web Apps (Web Frontend).
* Validate connectivity between migrated components:

Migrate --> Web Frontend (app migration).

Backend API --> Redis (session management).

3.Monitor Performance

* Use Application Insights to track the performance of the pilot.

**Phase 3: Full Database Migration**

Objective: Move transactional data to the cloud.

1. Prepare Azure SQL Database

* Create schema parity between Old Database and Azure SQL Database.

2. Use Azure Database Migration Service (DMS)

* Perform a cutover migration for customer/order data:

Old Database --> Azure SQL Database (via Migrate).

* Use transactional replication to minimize downtime.

**3. Sync Legacy and Cloud Data**

* Temporarily keep old database as read-only during cutover.

**Phase 4: Application Migration**

Objective: Transition of the legacy app to Azure.

**1. Rehost Legacy App**

* Migrate the full legacy app (Old App) to Azure Web Apps (Web Frontend).
* Update DNS to redirect traffic from on-premises to Azure Front Door.

**2. Reconfigure Middleware**

* Redirect legacy API calls to Azure API Management (APIMgmt).
* Connect Backend API to Service Bus for order queuing.

**3. Validate End-to-End Workflow**

* Test order placement:

Customer → Azure Front Door → Web Frontend → Backend API → APIMgmt → SQLDB

**Phase 5: Decommission On-Premises**

**Objective: Fully transition to Azure.**

**1. Retire Legacy Components**

* Shut down Old App and Old database after confirming all traffic flows through Azure.

**2. Optimize Cloud Resources**

* Enable auto-scaling for Azure Web Apps using Application Load Balancer.
* Configure Redis Cache for high-frequency session data.

**3. Enforce Security**

* Use Azure AD for centralized authentication.
* Rotate secrets via Key Vault.

**Phase 6: Monitoring & Optimization**

**Objective: Ensure reliability and scalability.**

**1. Track Metrics**

* Use Application Insights for APM (application performance monitoring).
* Aggregate logs via Log Analytics.

**2. Refine Architecture**

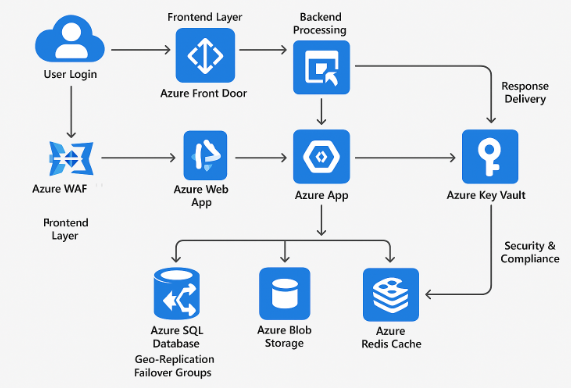
* Introduce serverless components (e.g. Azure Functions) for event-driven workflows.
* Explore Azure Kubernetes Service (AKS) if containerization is needed.

**Key Points:**

1. **Azure Migration Appliance:** The Azure Migrate appliance is a core part of Azure Migrate and a critical tool in Microsoft's cloud migration framework, designed to automate and streamline the process of moving on-premises workloads to Azure.
2. **Azure Front Door (AFD)**: Azure Front Door (AFD) is a global, scalable entry point for web applications, acting as a reverse proxy with built-in security, load balancing, and performance optimization.
3. **Redis:** Stores session data in the cloud.
4. **Azure SQL:** Stores transactional data.
5. **Azure Blob Storage:** Stores unstructured data like images.

**Application Flow**

**User Interaction with the Application:**



**User Interaction & Data Flow in a Three-Tier Architecture**

1. **User Login:** A customer accesses the e-commerce website via a browser.
2. **Frontend Layer:** The request first reaches **Azure Front Door**, which directs it to the appropriate **Azure Web App (Frontend UI)**.
3. **Load Balancing:** The request is routed through **Azure Application Gateway** and then to the Azure web app via **Azure Load Balancer**.
4. **Backend Processing:** The backend API, hosted on **Azure App Service**, processes the request and fetches product details from **Azure SQL Database and Blob Storage**.
5. **Database Interaction:** The backend service interacts with **Azure SQL Database**, utilizing **Geo-Replication and Failover Groups** for high availability.
6. **Caching Layer:** Frequently accessed data is retrieved from **Azure Redis Cache** to reduce database load.
7. **Security & Compliance:** All sensitive data is managed using **Azure Key Vault**, and web traffic is protected by **Azure Web Application Firewall (WAF)**.
8. **Response Delivery:** The processed information (products, cart details, or payment confirmation) is sent back to the user's browser.

**Operational Aspects of the Application**

**Development Environment:**

* Developers use **Azure DevOps** for CI/CD pipelines.
* Code is stored in **Azure Repos (Git)**.
* Infrastructure is managed using **Terraform & Ansible**.

**Testing Environment:**

* Automated testing is performed using **Azure Test Plans**.
* Pre-production testing on **Azure App Service (Staging Environment)**.

**Production Environment:**

* Live traffic is served via **Azure Front Door, Load Balancer, and App Service**.
* Application monitoring using **Azure Monitor and Application Insights**.
* Disaster recovery is managed with **Azure Backup and Azure Site Recovery**.

**Well-Architected Framework Considerations**

**A. High Availability Strategies:**

* Multi-Region Deployment with Azure Front Door for global traffic routing.
* Availability Zones for Azure VMs & Load Balancers to prevent single points of failure.
* Active Geo-Replication in Azure SQL Database for automatic failover.
* Azure Site Recovery for disaster recovery planning.

**B. Scalability Strategies:**

* Azure Front Door and Application Load Balancer automatically adjusts the number of App Service instances based on demand.
* Horizontal Scaling of Azure VMs ensures increased backend processing capacity.
* Azure Redis Cache speeds up frequent database queries.

**C. Security Strategies:**

* Azure Web Application Firewall (WAF) in Azure Application Gateway protects against threats like SQL injection & XSS attacks.
* Azure Key Vault secures API keys and credentials.
* DDoS Protection is enabled via Azure Front Door.
* RBAC (Role-Based Access Control) and Multi-Factor Authentication (MFA) enforce strong access controls.

**7. Well-Architected Framework Considerations**

### **a) Reliability:**

* Azure Site Recovery ensures failover capability.
* Load balancing across multiple VMs ensures uptime.

### **b) Security:**

* Azure Security Center continuously monitors security threats.
* Role-based access control (RBAC) restricts unauthorized access.

### **c) Performance Efficiency:**

* Auto-scaling in Azure App Services ensures efficient resource utilization.
* Azure Redis Cache reduces database load and improves response times.

**d) Cost Optimization:**

* Reserved VM instances reduce operational costs.
* Pay-as-you-go model for storage and compute services optimizes expenditure.

**e) Operational Excellence:**

* Centralized monitoring through Azure Monitor and Log Analytics.
* Automated deployment via Azure DevOps.

**Conclusion**

This document outlined the migration of a Hyper-V-based e-commerce application to Azure. By leveraging Azure Migrate and various Azure services, the migration process ensures high availability, scalability, and security in a three-tier architecture. The next steps include testing the migrated workloads, optimizing cloud costs, and implementing continuous monitoring and auto-healing strategies.