ABC COMPANY

Cloud modernization solution approach

Abstract

The document intents to provide the solution approach and the key consideration to move the inventory system of ABC company to Azure.

Prepared for Amdocs

Varun Kumar

me@varunkumar.in

Table of Contents

1. Problem Scenario:

Company ABC has a monolithic app that they've built over 10 years. They deliver a first-class inventory system that enables their customers to track inventories across chain stores. This distributed model of stores sends their inventory to the monolith application via SOAP. The company has been struggling exposing new features to customers due to constraints in the monolith design and quarterly code releases.

The hardware stack is as old as the code and will need to be replaced soon. All the inventory data is stored in a Relational Database per customer. In order to stay competitive in the market they need to consolidate their customers into a multi-tenant solution on Azure, GCP, or AWS. The company is very sensitive to security and in some ways afraid of the public cloud for that reason. You've been tasked with creating a comprehensive architecture for this application to exist in the cloud. Your architecture must highlight how we are securing this application to give the CIO / CTO / CISO piece of mind.

In addition, the CIO is looking for recommendations for application strategy with a Cloud Native approach that they can bring to the review board which consists of (Enterprise Architects, Application Architects, Business Owners, Product Owners and Finance).

2. Solution Overview

2.1. Solution Approach

1. Convert the monolithic architecture of the custom application into microservices and deploy on Azure App service and Azure Kubernetes Service (AKS).
2. Deploy the application in two regions (both active) load balanced by the Azure Front door.
3. Deploy the components in each region in a Virtual network with the Network Security Group to allow traffic only from the Azure Front door.
4. Deploy SQL databases in the SQL Elastic pool in a separate Virtual network. VNet peering will be done between each App and the database Virtual networks.
5. All the authentication will be done through Azure Active directory and the service-to-service communication though Managed identities.
6. Deploy the application through Infrastructure as code on Azure Devops with the Rolling deployment option.
7. Use sharded multi-tenant databases approach to consolidate the stores data on Azure.

2.2. Solution Architecture

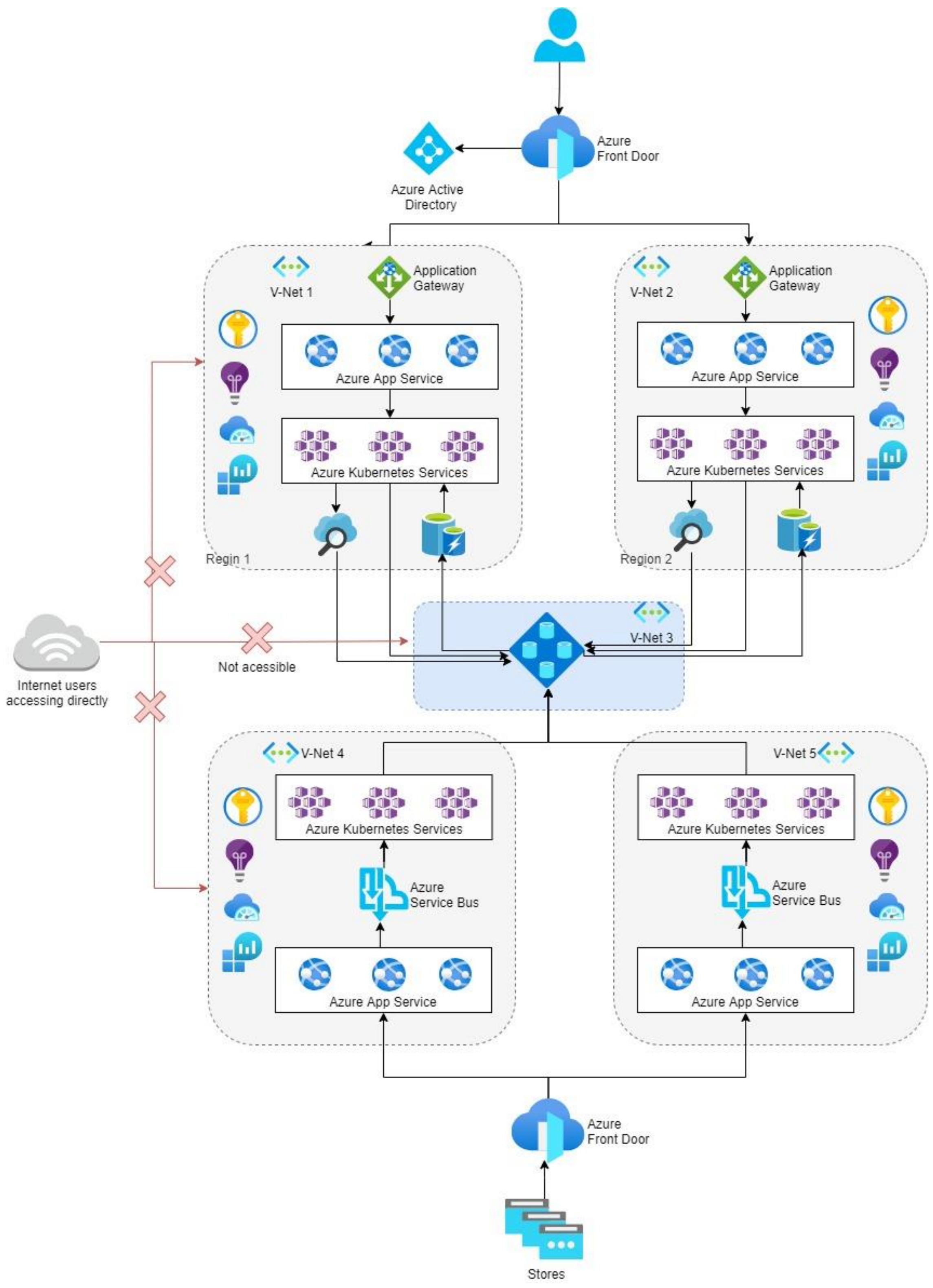

Figure 1: Architecture Diagram

1. Azure Front Door handles a few initial tasks:
    - Processing the initial request.
    - Load balancing across the regions.
    - SSL(HTTPS) termination and offloading.
    - Fail over if there's a regional outage.
2. The architecture uses **Azure Active Directory** (Azure AD) as the identity provider for authentication.
3. Once routed to the appropriate region, Application Gateway routes and load balances, directing requests to the appropriate App Service.
4. Configure App Service to scale up and out automatically. It makes App Service a good fit for scaling a host of tenant HTTP-driven requests on demand.
5. The data access layer services are also independently scaled based on load. You can deploy and scale these data services using the **Azure Kubernetes Service** (AKS). AKS can implement a microservice architecture, which features a series of containers that each encapsulate specific functionality within the cluster. It allows for a high degree of abstraction and de-coupling within the code.
6. Store and manage relational data outside of the application framework. Doing so provides a single point of data entry for either region. Replication, availability, scalability, and security are achievable by leveraging the strength of **Azure SQL Elastic Pools**. Provision each tenant a database in a pool. Allocate the resources available in the pool to databases on-demand based on load.

2.3. Solution Components

The primary components suggested for the architecture in this solution.

- **Azure Front Door**: A regional load balancer that routes client traffic to the correct region. It can fail over to the second region if region failure happens and it can secure the internet-facing entry point via Azure Web Application Firewall.
- **Azure Active Directory (Azure AD):** Acts as the identity provider for the entire application, enforcing authentication and end-to-end authorization of the request in the application.

- **Application Gateway**: Routes and load-balances traffic internally in the application to the various services that satisfy client business needs. Azure Front Door and Application Gateway combine to provide complex load-balancing at all levels in a multitenant solution.
- **App Service:** Azure's premier service for web applications and web-based APIs. Security integrates with services like Azure AD and Azure Key Vault. You can configure scaling to happen automatically. App Service can also leverage integrated DevOps capabilities for continuous integration and deployment to multiple environments.
- **Azure Kubernetes Service (AKS):** Orchestrates instances of container images deployed to a cluster. Managing multiple clients' data often involves implementing a suite of components to manage. Developing these many smaller components as container-based microservices creates an ideal scenario for the deployment to an AKS cluster. Built into the framework are tools for autoscaling, load balancing, and upgradeability. AKS integrates well with a continuous integration/continuous delivery (CI/CD) strategy using the available DevOps features and Azure Container Registry.
- **Azure SQL Elastic Pools**: Provides a solution for managing a set of databases flexibly with a pool of resources. The service allocates resources on demand to the databases. It helps reducing the budget and overhead of maintaining multiple SQL Servers with large chunks of unused compute resources.
- **Azure Cognitive Search** (formerly known as Azure Search): Provides powerful indexing and query engine to the application. It gives clients access to strong query functionality.
- **Azure Cache for Redis**: An in-memory managed cache to reduce latency and increase performance for the clients. High throughput allows for a high volume of requests to handle multiple tenants accessing the system. Can be scale the service as application loads increase. Supports encryption at rest to protect and isolate cached tenant data.

2.4. Multitenancy

A multitenant solution is the key consideration in this solution. The solution allocates enough resources to process all client requests effectively. Deploy applications to a pair of regions for high availability.

Many client databases can exist on the same Elastic Pool, isolated and secured by transparent data encryption (TDE). Assign replication policies to each Elastic Pool to provide backup and failover for client data.

Sharded multi-tenant database

This model allows packing large numbers of tenants into a single database, driving the cost-per-tenant down. Flexibility is available in this model, allowing to optimize for cost with multiple tenants in the same database, or optimize for isolation with a single tenant in a database. The choice can be made on a tenant-by-tenant basis, either when the tenant is provisioned or later, with no impact on the design of the application.

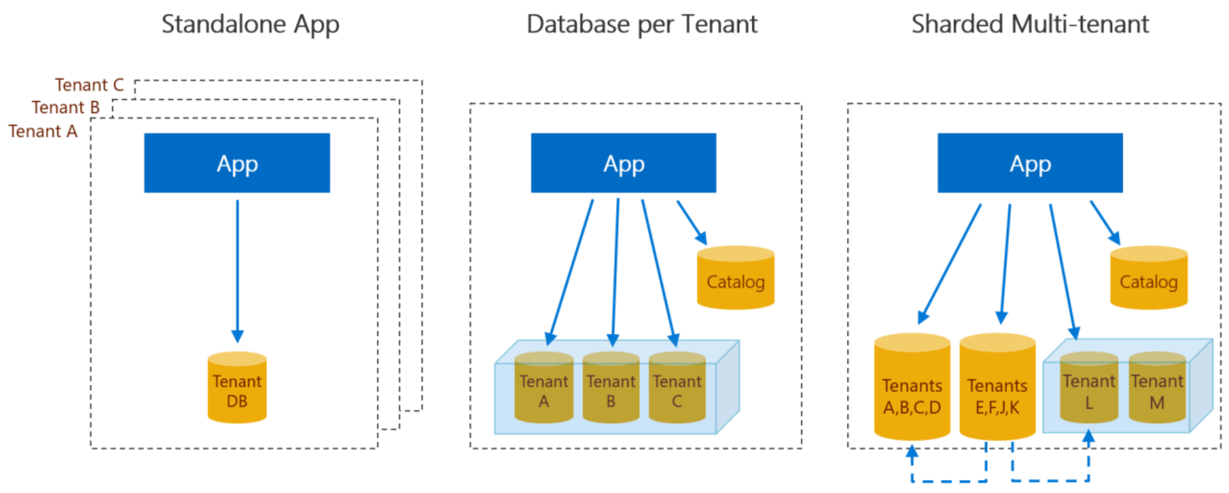

Figure 2: Multi-tenant database strategy

3. Zero Downtime Deployments

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources. The following diagram shows the rolling deployment for the application in AKS with the help of Azure DevOps build and release pipelines and Azure container registry.

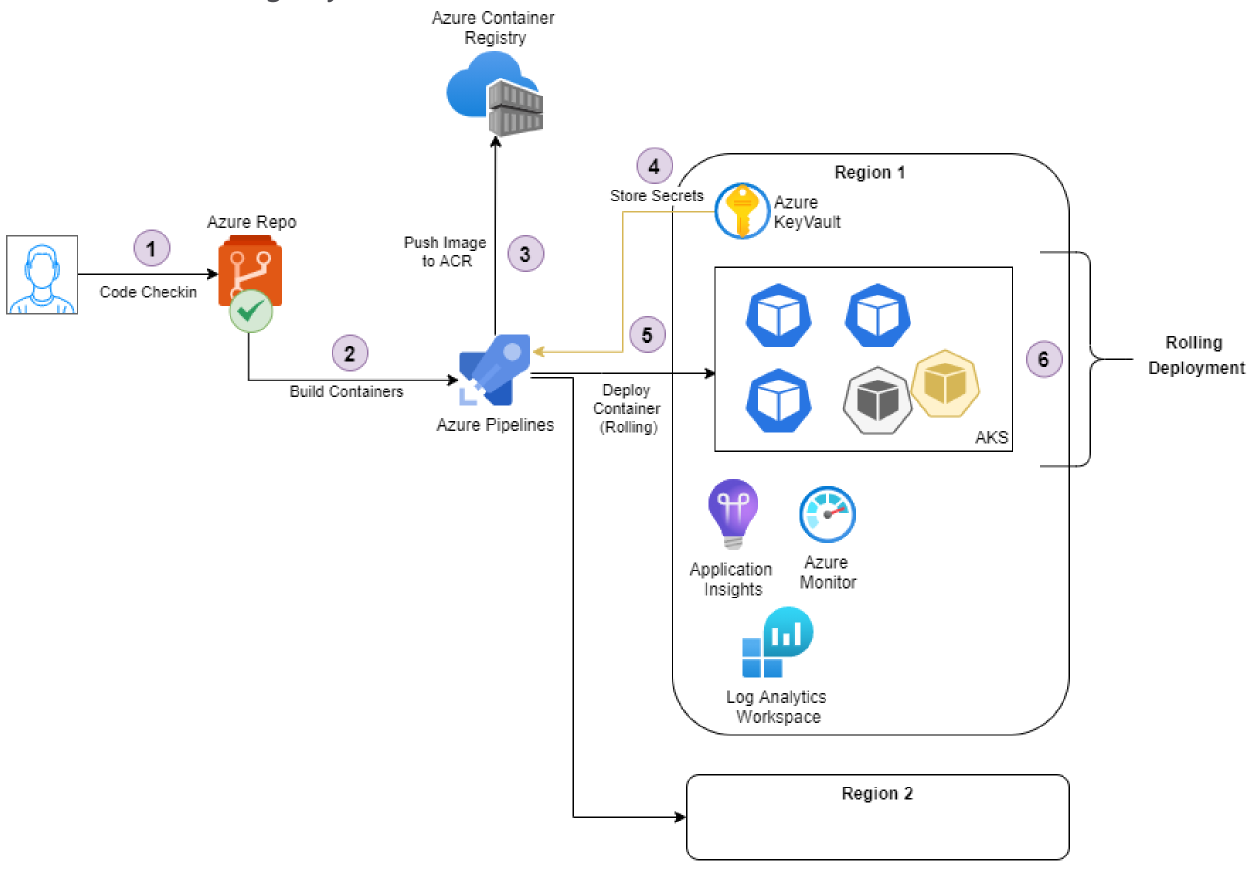

Figure 3: Zero downtime deployment using Azure DevOps

1. PR is raised by the developer to check-in the code in the repo.
2. The Build pipeline automatically runs and push the image to the Azure Container Registry.
3. The Release pipeline gets triggered and deploy the image from ACR on the Azure Kubernetes Service with the rolling update option.
4. The secrets are stored in the Azure Key vault and injected in the application on need basis.
5. New Pod in the AKS is created with the new image, AKS checks the health of the new Pod with the help of liveliness and readiness probes.
6. Once the new Pod is confirmed healthy, the old pod is removed.

4. Key Considerations

4.1. Security

The system addresses security from end-to-end at each level of the application:

- Each application stack in each region lies within an **Azure Virtual Network**. The system restricts traffic into the virtual network accepting requests from Azure Front Door, protecting all application services from external traffic.
- Azure Front Door provides built-in HTTPS support for its domains with TLS/SSL certificate. You can choose to use a certificate that is managed by Azure Front Door or use your own certificate. That means the system can encrypt all traffic to the application.
- Azure Front Door also implements Azure Web Application Firewall, protecting the SaaS stack from attacks at the edge, before the system routes requests to the application.
- All credentials, secrets, and connection strings can be securely managed by **Azure Key Vault**. This helps to protects client data by ensuring that a breach in code or man-in-the-middle attack wouldn't gain access to tenant databases.
- **Transparent data encryption (TDE) and just in time (JIT)** decryption protects data on the database. The system encrypts all data on the database at rest, and only decrypts it when requested by the tenant.
- Following the **Principal of least privilege** makes sure that only the right access is provided to the applications and the right people in the team. Example, the members that require read access will have read access only and not the write access, customer applications will have read access only to database and backend applications will have write access only.
- **Managed identities** eliminate the need for developers to manage credentials. Managed identities provide an identity for applications to use when connecting to resources that support Azure Active Directory (Azure AD) authentication. Applications may use the managed identity to obtain Azure AD tokens. For example, an application may use a managed identity to access resources like SQL database, key vault, storage etc.

4.2. Scalability and Availability

The solution is designed to account for a large number of tenants. It takes advantage of the large number of scalable components and services to grow based on load.

- Azure App service and AKS have the autoscaling capabilities to scale up and down or out and in based on the load on the system.
- The database is deployed in the elastic pool to optimize the resources beads on the load. The pool can be defined to auto scale up or down based on the load.
- For high availability the application is deployed in two regions giving the availability of 99.99% together.
- SQL red replicas can be created in one or more regions for the automatic failover in case of a region outage and BCDR.