Tangent Solutions

assessment

|  |  |
| --- | --- |
| Raised hand with solid fill | The following document outlines the assessment requirements as put forward by Tangent Solutions for the Role:  Cloud Solutions Architect |

Table of Contents

[Overview 3](#_Toc130835813)

[Instructions 3](#_Toc130835814)

[Requirements 4](#_Toc130835815)

[Requirement 1 4](#_Toc130835816)

[Requirement 2 4](#_Toc130835817)

[Requirement 3 5](#_Toc130835818)

[Requirement 4 6](#_Toc130835819)

[Application architecture as it is understood. 7](#_Toc130835820)

[The application architecture as you understand it from the code and description. 7](#_Toc130835821)

[Original Architecture Based on Requirements 8](#_Toc130835822)

[The application architecture as you understand it from the code and description 8](#_Toc130835823)

[Recommended Solutions Diagrams 9](#_Toc130835824)

[The Azure infrastructure architecture you would recommend for the deployment. 9](#_Toc130835825)

[Recommended Solutions Documentation 11](#_Toc130835826)

[The Azure infrastructure architecture you would recommend for the deployment. 11](#_Toc130835827)

[AKS Cluster 12](#_Toc130835828)

[DevOps 13](#_Toc130835829)

[The DevOps processes and gates you would recommend. 13](#_Toc130835830)

[Security Recommendations 14](#_Toc130835831)

[Any security considerations that you would recommend for the deployment. 14](#_Toc130835832)

[Modernization 15](#_Toc130835833)

[Any modernization activities/projects which would improve the architecture. 15](#_Toc130835834)

[Inherent Original Design Issues 15](#_Toc130835835)

[Any possible issues with the architecture as you see it. 15](#_Toc130835836)

[Cost Management 16](#_Toc130835837)

[The following can be done to save on Azure costs. 16](#_Toc130835838)

# Overview

The following document contains Diagrams and Documentation based on the following:

* Assessment of the e-commerce application for an understanding of how it works.
* A list of requirements for the e-commerce application.
* Architecture Diagram based on the understanding of the application.
* Architecture Diagrams based on the original design with requirements.
* Architecture Diagrams based on the recommended solutions.
* Documentation containing, DevOps Processes, Security Recommendations, Modernization, Inherent Issues and Cost Management.

# Instructions

The client has created an ASP.NET web application and services for an e-commerce application. The code and description for this application can be found at <https://github.com/quintindk/workshop/tree/master/demos/asp-net-core>

The customer has a few additional requirements that are highlighted in the requirements section below.

# Requirements

|  |  |
| --- | --- |
| Raised hand with solid fill | The following is a list of requirements as requested by the client |

## Requirement 1

* Containerize the following projects.
* Divergent.Sales.API, Divergent.Shipping.API, Divergent.CompositionGateway, Divergent.Website
* Create a docker files
* These must build the Divergent.Sales.API, Divergent.Shipping.API, Divergent.CompositionGateway, Divergent.Website components for the backend APIs and website so that they can be built on a developer’s machine.
* Name the docker files and provide instructions for building the container images.
* Create a docker compose definition for the website.
* This is to build and run all components for the website on the developer’s local machine so that the developer can test the application.
* Name the docker compose files and provide instructions for running the docker compose to test the web site system.
* Create docker compose definitions for the APIs.
* These are to build and run all components for the backend APIs on the developer’s local machine so that the developer can test the composition API’s.
* Name the docker compose files and provide instructions for running docker compose to test the API system.

## Requirement 2

* Create Terraform scripts for deployments of the following:
* Creation of Azure Resource Group and Azure Container Registry.
* Creation of Azure Resource Group and AKS.
* Provide instructions for the running of the terraform scripts.
* Create Automated DevOps pipelines for the following:
* Terraform deployment of Resource Group and Container Registry by using a Service Principle.
* Container build and push the containers to the Azure Container Registry resource.
* Provide instructions registering the pipelines and any requirements for the service principal.

## Requirement 3

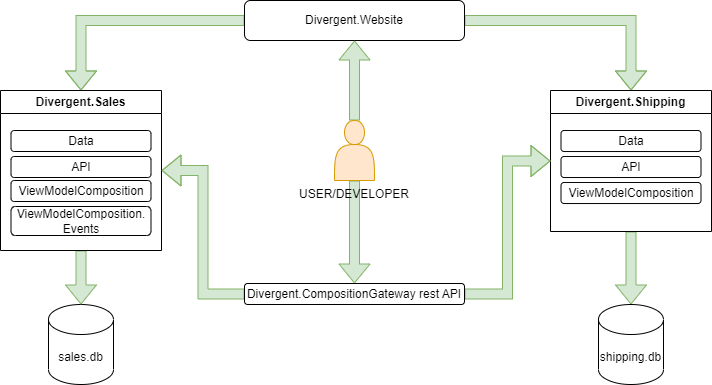
* Create a Helm chart for the deployment of the entire solution to AKS including:
* 4 Containers as deployments
* Applicable Services and Ingresses
* Provide instructions for running the helm command for the installation.
* Setup DevOps Pipeline
* To run the install task of the Helm chart installation using the same Service Principle.
* Provide instructions for registering the pipeline and any requirements for the service principal and service connection.

## Requirement 4 (CSA)

* Diagram and document the architectural design for the application deployment including:
* The application architecture as you understand it from the code and description.
* The Azure infrastructure architecture you would recommend for the deployment.
* The DevOps processes and gates you would recommend.
* Any security considerations that you would recommend for the deployment.
* Any modernization activities/projects which would improve the architecture.
* Any possible issues with the architecture as you see it.

# Application architecture as it is understood.

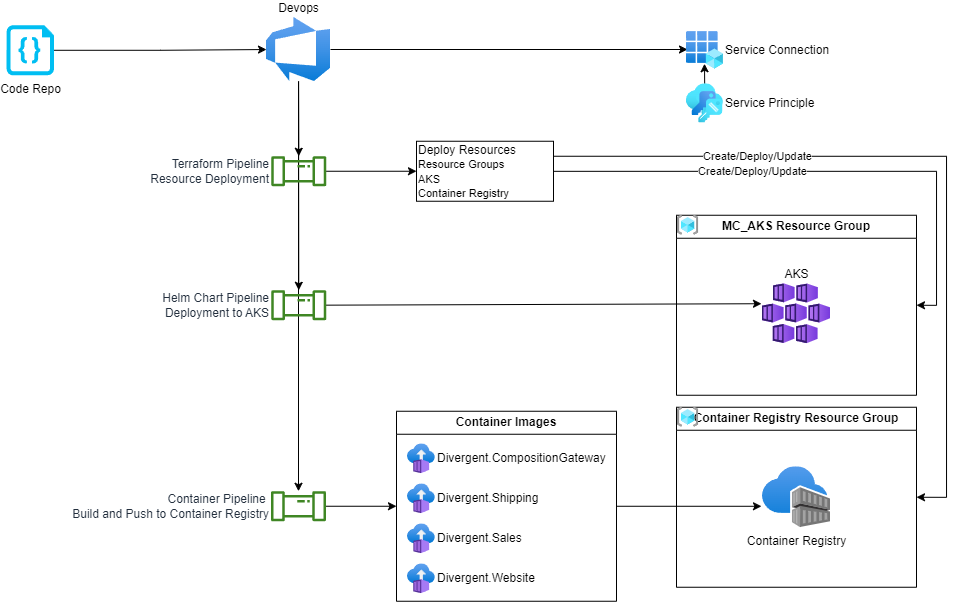
## The application architecture as you understand it from the code and description.



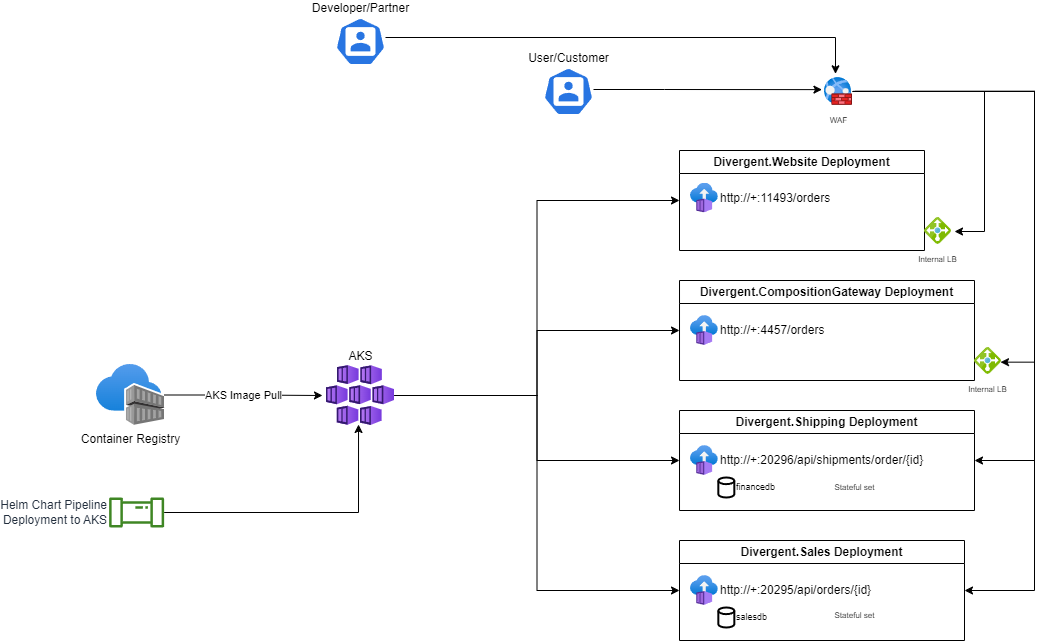
# Original Architecture Based on Requirements

## The application architecture as you understand it from the code and description

**DevOps**



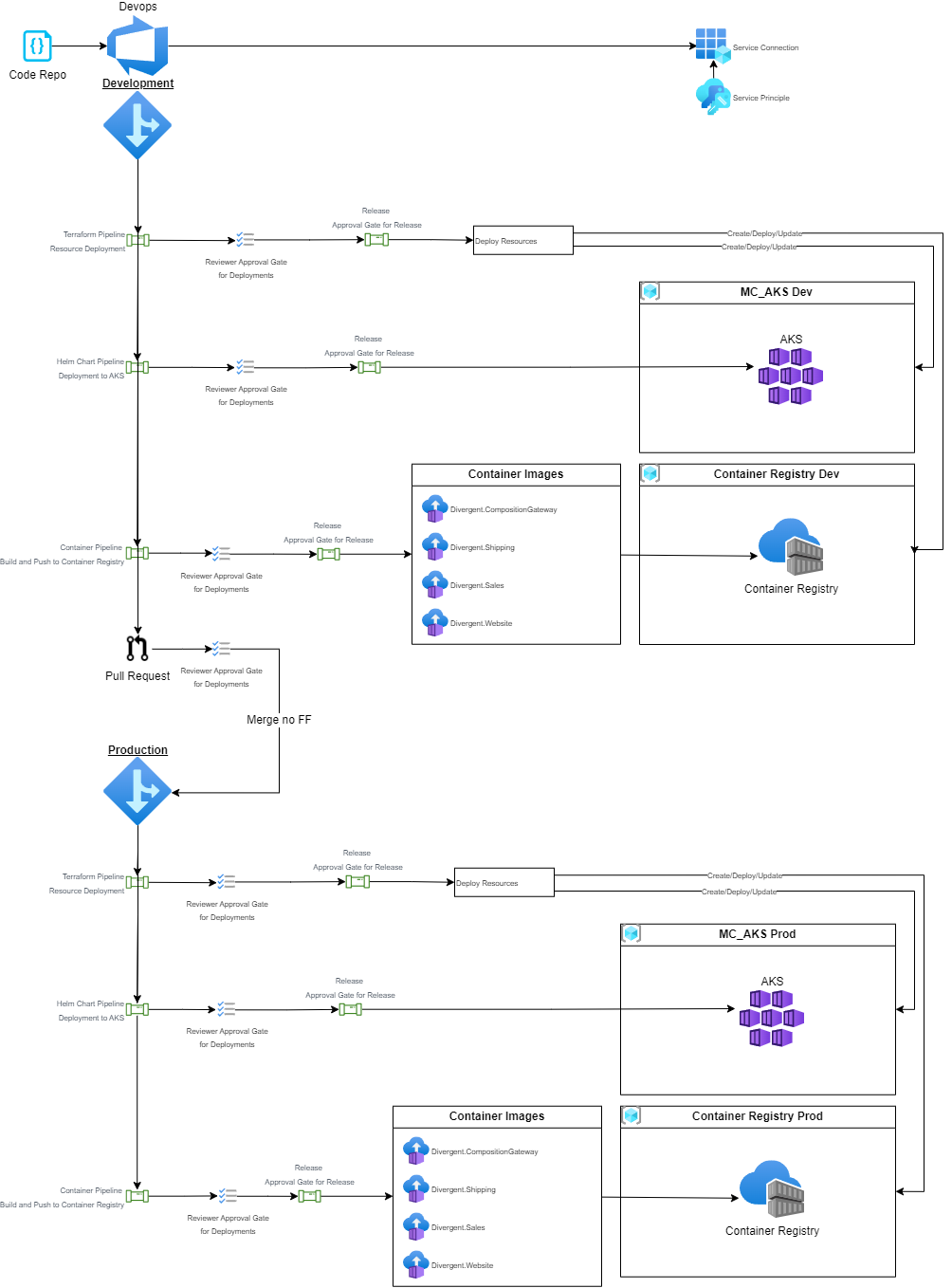
**Operational**

****

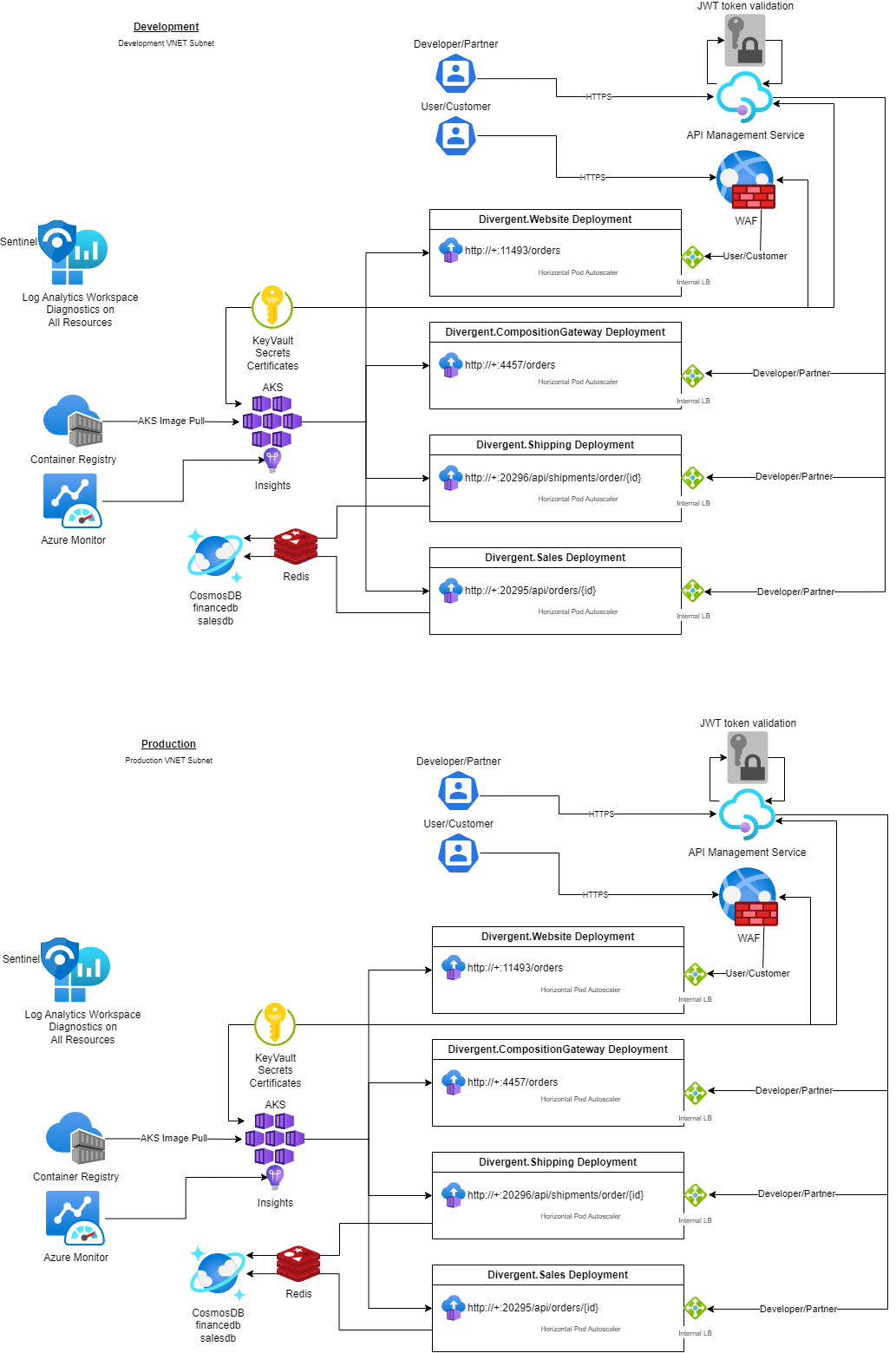
# Recommended Solutions Diagrams

## The Azure infrastructure architecture you would recommend for the deployment.

**DevOps**



**Operational**

****

# Recommended Solutions Documentation

## The Azure infrastructure architecture you would recommend for the deployment.

1. Use two separate AKS clusters, one for development and another for production, each deployed in a separate virtual network (VNET) subnet.
2. Use Azure Container Registry to store and manage container images for the applications and APIs.
3. Use Azure API Management to manage the APIs and control access to them. The API Gateway service can be deployed on the AKS cluster and exposed through API Management.
4. Use Azure Cosmos DB to store the sales and shipping databases, and configure the databases to be accessible only from the internal network.
5. Use Azure Redis Cache to cache data for the MVC web application service and Composition API Gateway service to improve performance.
6. Use Azure Key Vault to store secrets and sensitive configuration data, such as database connection strings and authentication keys.
7. Use Azure WAF to protect the MVC web application service and Composition API Gateway service from common web-based attacks, such as SQL injection and cross-site scripting.
8. Implement logging and monitoring using Azure Monitor, Log Analytics Workspace and Sentinel to detect and respond to security events and incidents.
9. (Optional) Grafana can be used for AKS cluster monitoring by providing real-time insights into your cluster's performance, resource utilization, and health. Grafana integrates with several data sources, including Prometheus, which is an open-source monitoring system that gathers metrics from various sources and stores them in a time-series database.
10. Use the least privilege principle: When creating resources such as service principals and RBAC roles, grant only the minimum permissions necessary to perform their intended function. This helps to reduce the attack surface and improve security.
11. Monitor and manage costs: Use Azure Monitor to monitor resource utilization and cost, and implement cost management practices to optimize resource usage and minimize costs.
12. Ensure high availability: Configure multiple availability zones for the AKS cluster to ensure high availability and resilience against infrastructure failures.
13. Use Cluster auto-scaling and Pod auto-scaling (Horizontal Pod Autoscaler) to handle the increased load.
14. Follow best practices for container images: Use secure and up-to-date container images, and scan them for vulnerabilities before deployment.
15. Implement data protection and privacy: Use Azure Key Vault or other secure storage solutions to protect sensitive data such as secrets, passwords, and keys.
16. Implement security measures such as role-based access control (RBAC), network security groups (NSGs), and virtual network service endpoints to ensure secure access to the resources and limit exposure to potential security threats.
17. Implement continuous integration and delivery (CI/CD) using DevOps to automate the deployment of Azure resources, applications, and APIs to the AKS clusters as well as control the entire development and deployment lifecycle by using reviewer and approval gates.

Overall, the above architecture recommendation follows the Microsoft Well-Architected Framework and is designed to provide a secure, reliable, efficient, and cost-effective system for the e-commerce website.

## DevOps

## The DevOps processes and gates you would recommend.

1. Use a continuous integration and delivery (CI/CD) pipeline: Implement a CI/CD pipeline to automate the build, test, and deployment of your application to AKS. This pipeline should include automated testing, code quality checks, and gates to ensure that only high-quality code is deployed to production.
2. Use infrastructure as code (IaC) tools: Use IaC tools such as Azure Resource Manager (ARM) templates and Terraform to define your AKS infrastructure as code. This makes it easier to manage and replicate your infrastructure across environments.
3. Use source control management (SCM): Use SCM tools such as Git to version control your code and infrastructure definitions. This enables you to track changes and collaborate with other developers.
4. Use release gates: Use release gates such as automated testing, code quality checks, and security scans to prevent low-quality or insecure code from being deployed to production.
5. Use Pull Requests to merge the development branch into the production branch protected by review/approval stage gates to ensure that change control is adhere to and that the live Production system is protected.
6. Monitor your deployments: Monitor your deployments using tools such as Azure Monitor or Grafana to detect and diagnose issues. This enables you to quickly respond to issues and minimize downtime.

# Security Recommendations

## Any security considerations that you would recommend for the deployment.

1. Secure communication: Ensure that all communication between the e-commerce website and other resources, including the AKS clusters, CosmosDB databases, and the WAF, is encrypted using secure protocols such as HTTPS and TLS. CORS Policies should not be left “loose” (allowall), Methods (GET, POST, DELETE etc) should be secured and only to allow methods intended for the primary function of the API’s.
2. Access control: Implement role-based access control (RBAC) to control access to resources based on the principle of least privilege. Ensure that only authorized personnel have access to sensitive resources and data.
3. API security: Implement API security best practices, such as limiting access to APIs based on API keys or tokens, monitoring API usage and access, and limiting access to sensitive APIs.
4. Network security: Configure network security groups (NSGs) to restrict traffic to and from Development and Production AKS clusters, CosmosDB databases, and other resources. (Consider using DDoS protection to protect against potential DDoS attacks that can disrupt availability of the e-commerce website, This can be achieved with Azure solutions or Public DNS providers like Cloudflare.)
5. Authentication and authorization: Implement Azure AD authentication and authorization for AKS and other resources. Use strong passwords and multi-factor authentication (MFA) to further secure access to resources.
6. Data encryption: Ensure that all sensitive data stored in databases is encrypted at rest using Azure encryption technologies. (CosmosDB encryption at rest and in transit is "on" by default)
7. Container security: Implement container security best practices, such as running containers with non-root users, limiting container privileges, and scanning container images for vulnerabilities using Azure Security Center Container Registry scanning or integrate container image scanning into your CI/CD pipeline.
8. Patch management: Keep all resources, including AKS clusters, up-to-date with the latest patches and security updates to address known vulnerabilities.
9. Logging and monitoring: Implement logging and monitoring solutions, such as Azure Monitor, Azure Log Analytics and Sentinel to detect security incidents and respond quickly to potential threats.
10. Disaster recovery: Implement a disaster recovery plan to ensure business continuity in the event of a security incident or outage.
11. Compliance: Ensure that your deployment meets industry and regulatory compliance standards, such as PCI DSS, HIPAA, and GDPR, as applicable.

# Modernization

## Any modernization activities/projects which would improve the architecture.

1. Implement Azure Functions for serverless computing: Azure Functions allow you to execute code on-demand without managing the infrastructure. You can use them for event-driven scenarios like processing orders, sending emails, or updating databases.
2. Upgrade runtime components to the latest version. Upgrading runtime components is important for several reasons. It can improve the performance, stability, and security of the application, and can also enable the application to take advantage of new features and capabilities. Additionally, upgrading runtime components can ensure that the application remains compatible with other software components and systems it interacts with.  
     
   !Caveat

When upgrading runtime components, it's important to ensure that the new components are compatible with the existing application and that any necessary changes are made to the application to support the new components. It's also important to thoroughly test the application after the upgrade to ensure that it functions correctly and to identify any issues that may have been introduced during the upgrade process.

# Inherent Original Design Issues

## Any possible issues with the architecture as you see it.

1. The original design with requirements did not include segmentation of environments I have included this into the Recommended Solutions.
2. The original design with requirements had a flaw where the sales and shipping databases would lose their data when deployments where patched or new images pulled/deployed in AKS due to the way the application is designed but was addressed by applying the well-architected framework and think about resilience, therefore making the sales and shipping API’s statefulsets solved the problem and then improving on this was to recommend the use of a database service outside the cluster like CosmosDB which is also NoSQL like LiteDB, but would require development to use.

# Cost Management

## The following can be done to save on Azure costs.

1. Optimize AKS nodes: You can optimize your AKS nodes to run at the minimum required size for your application workload. This can be done by setting up auto-scaling rules that scale up and down based on application demand. You can also consider using low-priority VMs to reduce costs.
2. Use reserved instances: Reserved instances are a great way to save money on compute costs in Azure. You can purchase reservations for VMs and other Azure services, which can offer significant discounts compared to on-demand pricing.
3. Right-size Azure Cache for Redis: Azure Cache for Redis is a fully managed, in-memory caching service that can help improve application performance. However, it can also be expensive if not right-sized for your application. You should monitor cache usage and consider scaling down or using a lower-tier service if usage is low.
4. Optimize CosmosDB throughput: CosmosDB is a NoSQL database service that can scale globally and provide high availability. However, it can also be expensive if not optimized correctly. You should monitor database usage and consider adjusting throughput or using a lower-tier service if usage is low.
5. Use Azure Cost Management: Azure Cost Management is a free tool that can help you optimize your cloud spend. You can use this tool to monitor costs, set up alerts, and identify areas where you can save money.

By implementing these recommendations, you can optimize your Azure resources and reduce costs without compromising performance or security.

Thank you for the opportunity.

Delmain.