Week-17 | Deploy an E-Commerce Project on Azure Kubernetes Service | Step-by-Step Guide Project AKS

Introduction

Deploying an e-commerce project on **Azure Kubernetes Service (AKS)** is a critical step in ensuring scalability, high availability, and efficient resource management for modern cloud applications. This guide provides a **detailed step-by-step approach** to setting up an AKS cluster, deploying an application, configuring networking and storage, implementing monitoring, security best practices, and optimizing performance.

By following this guide, you will:

- ☑ Set up an **Azure Kubernetes Service (AKS) cluster**.
- ☑ Deploy an **e-commerce application** on the cluster.
- ☑ Configure **networking and security** for efficient traffic management.
- ☑ Integrate **Azure Container Registry (ACR)** for seamless container management.

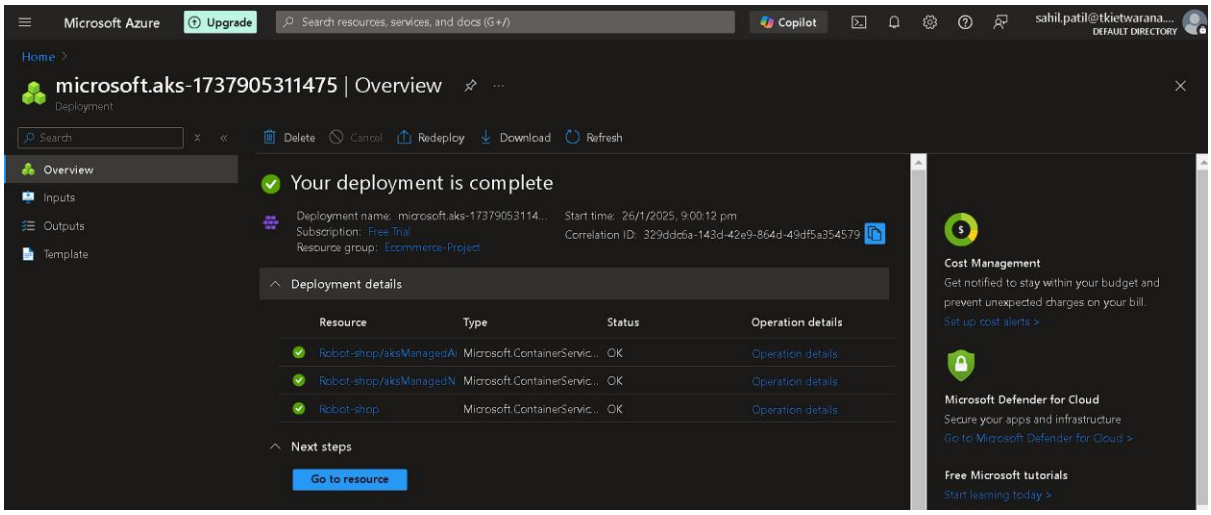
☑ Set up **monitoring and logging** for performance tracking.
☑ Optimize the deployment with **auto-scaling**.

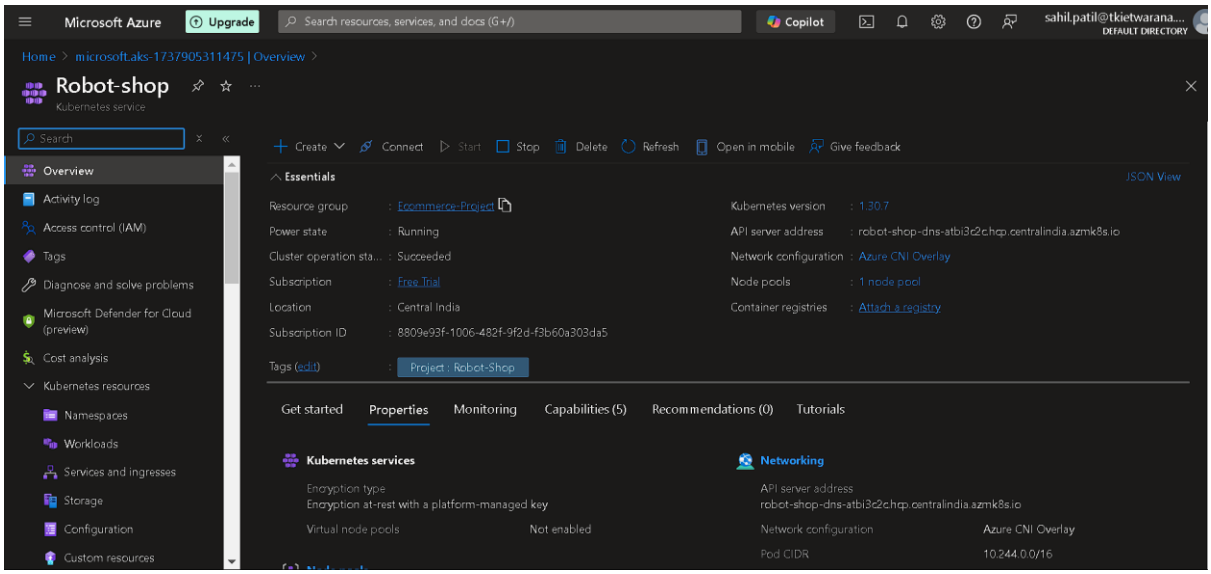
Why Use Azure Kubernetes Service (AKS)?

AKS provides a **fully managed Kubernetes environment**, allowing developers and DevOps engineers to deploy, manage, and scale containerized applications with minimal operational overhead. Some of the key benefits include:

- **High Availability**: Ensures that applications remain available during failures.
- **Scalability**: Auto-scales applications based on traffic and resource utilization.
- **Integrated Security**: Provides role-based access control (RBAC), Azure Active Directory integration, and network policies.
- **Cost Optimization**: Eliminates the need for managing Kubernetes control planes, reducing operational costs.

Step 1: Setting Up the AKS Cluster





To deploy an e-commerce application, the first step is setting up an **Azure Kubernetes Service (AKS) cluster**. AKS handles the orchestration of containerized applications by managing Kubernetes nodes, networking, and scaling requirements.

Key Considerations Before Setup

- Choose the **right Azure region** for optimal latency and cost efficiency.
- Decide the **number of nodes** required based on application workload.
- Enable **Azure Monitor** for observability.
- Implement **role-based access control (RBAC)** for secure access.

Once the AKS cluster is set up, it serves as the **infrastructure** where the application will be deployed, providing a scalable and resilient foundation for the e-commerce project.

Step 2: Configuring Azure Container Registry (ACR)

Containerized applications require a **secure and efficient way to store and manage container images**. **Azure Container Registry (ACR)** is a private container registry that integrates with AKS, ensuring smooth deployment.

Benefits of Using ACR

- Secure storage for **Docker images**.
- Seamless **integration with AKS** for automated deployments.
- **Role-based access control (RBAC)** to manage who can push or pull images.
- Supports **geo-replication** for global availability.

Once ACR is created, the e-commerce application's Docker images need to be **built, tagged, and pushed** to the registry for deployment.

Step 3: Deploying the E-Commerce Application on AKS

```
sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project (main)
$ kubectl create ns robot-shop
namespace/robot-shop created

sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project (main)
$ kubectl get ns
NAME              STATUS   AGE
default           Active   4m56s
kube-node-lease   Active   4m56s
kube-public       Active   4m56s
kube-system       Active   4m57s
robot-shop        Active   11s
```

```
sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project/AKS/helm (main)
$ helm install robot-shop --namespace robot-shop .
NAME: robot-shop
LAST DEPLOYED: Sun Jan 26 21:05:49 2025
NAMESPACE: robot-shop
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

```
sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project/AKS/helm (main)
$ kubectl get deployments -n robot-shop
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
cart        1/1     1            1           5m15s
catalogue   1/1     1            1           5m15s
dispatch    1/1     1            1           5m15s
mongodb     1/1     1            1           5m15s
mysql       1/1     1            1           5m15s
payment     1/1     1            1           5m15s
rabbitmq    1/1     1            1           5m15s
ratings     1/1     1            1           5m15s
shipping    1/1     1            1           5m15s
user        1/1     1            1           5m15s
web         1/1     1            1           5m15s

sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project/AKS/helm (main)
$ kubectl get svc -n robot-shop
NAME        TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)                       AGE
cart        ClusterIP      10.0.88.33      <none>         8080/TCP                      5m18s
catalogue   ClusterIP      10.0.218.30     <none>         8080/TCP                      5m18s
dispatch    ClusterIP      None            <none>         55555/TCP                     5m18s
mongodb     ClusterIP      10.0.86.49      <none>         27017/TCP                     5m18s
mysql       ClusterIP      10.0.210.118    <none>         3306/TCP                      5m18s
payment     ClusterIP      10.0.202.171    <none>         8080/TCP                      5m18s
rabbitmq    ClusterIP      10.0.49.78      <none>         5672/TCP,15672/TCP,4369/TCP   5m18s
ratings     ClusterIP      10.0.194.188    <none>         80/TCP                        5m18s
redis       ClusterIP      10.0.56.63      <none>         6379/TCP                      5m18s
shipping    ClusterIP      10.0.17.13      <none>         8080/TCP                      5m18s
user        ClusterIP      10.0.49.53      <none>         8080/TCP                      5m18s
web         LoadBalancer   10.0.115.128    4.188.64.10    8080:31348/TCP                5m18s

sahil@SAHI-SNEH MINGW64 /h/DevOps-Projects/Robot-Shop-Project/AKS/helm (main)
$ kubectl get pods -n robot-shop
NAME                           READY   STATUS    RESTARTS   AGE
cart-78dbff49b-psfmr           1/1     Running   0          5m25s
catalogue-7b4b777975-jh6vf     1/1     Running   0          5m25s
dispatch-7d4ff989d7-xwr6s      1/1     Running   0          5m25s
mongodb-b487b86b6-b2lxj        1/1     Running   0          5m25s
mysql-7c9bcd9464-vfxvd         1/1     Running   0          5m25s
payment-7474f4f69f-h72vq       1/1     Running   0          5m25s
rabbitmq-7bc9649444-lc5kg      1/1     Running   0          5m25s
ratings-8c68dd6c5-n92nz        1/1     Running   0          5m25s
redis-0                        1/1     Running   0          5m24s
shipping-5c899bdb6c-5p9jb      1/1     Running   0          5m25s
user-596968bd87-s88j9          1/1     Running   0          5m24s
web-6545b6c677-tcl8b           1/1     Running   0          5m25s
```

Once the cluster and container registry are set up, the next step is **deploying the application**. The deployment process includes:

Key Components of Deployment

- **Pods**: The smallest deployable units that run the application containers.
- **Deployments**: Define the desired state of the application and ensure it remains operational.
- **Services**: Manage communication between different components of the application.

The e-commerce application is deployed by defining **Kubernetes manifest files** for deployment and services. These configurations ensure that the application runs smoothly, handles requests efficiently, and can be updated or rolled back without downtime.

Step 4: Configuring Networking and Ingress

To make the application accessible over the internet, it is necessary to **set up networking** in AKS. This involves configuring:

1. Load Balancing

- A **Kubernetes Service of type LoadBalancer** ensures external access to the application.
- Azure Load Balancer automatically distributes incoming traffic.

2. Ingress Controller

- An **Ingress Controller** is used to route external HTTP/S traffic to the correct application services.
- It supports **SSL/TLS termination**, ensuring secure connections.

3. Custom Domain Integration

- The application can be accessed via a **custom domain**.
- **DNS records** are configured to point to the Ingress Load Balancer.

This networking setup ensures **high availability, security, and efficient traffic routing** for the e-commerce application.

Step 5: Configuring Persistent Storage

E-commerce applications often require **persistent storage** for managing product data, user sessions, orders, and payment transactions. AKS supports **various storage solutions** to meet different application needs.

Storage Options in AKS

- **Azure Disks**: Best for high-performance, block-level storage.
- **Azure Files**: Used for file-based shared storage between multiple pods.
- **Persistent Volumes (PVs)**: Abstracts storage resources, allowing easy attachment to Kubernetes workloads.

Choosing the right storage solution depends on **application requirements, performance, and cost considerations**.

Step 6: Implementing Auto-Scaling for Performance Optimization

To handle **varying traffic loads**, AKS provides **auto-scaling mechanisms**:

1. Horizontal Pod Autoscaler (HPA)

- Adjusts the number of application pods based on CPU and memory utilization.
- Ensures that the application scales up during peak traffic and scales down during low demand.

2. Cluster Autoscaler

- Dynamically **adds or removes nodes** in response to demand.
- Optimizes **cost efficiency** by preventing over-provisioning.

Configuring auto-scaling ensures that the e-commerce platform remains **responsive, cost-effective, and high-performing**.

Step 7: Setting Up Monitoring and Logging

Observability is crucial for maintaining application health and performance. **Azure Monitor** and **Log Analytics** provide comprehensive insights into AKS workloads.

Monitoring Features

- **Live metrics** tracking CPU, memory, and network usage.
- **Container logs and event tracking** for debugging.
- **Alerting** based on predefined thresholds.

Logging with Azure Log Analytics

- Captures **real-time application logs**.
- Helps **troubleshoot issues** quickly.
- Stores logs for **audit and compliance**.

Monitoring tools ensure proactive issue resolution and maintain application stability.

Step 8: Implementing Security Best Practices

Securing an e-commerce platform on AKS involves multiple layers of protection.

1. Network Security

- Restrict **public API access** to prevent unauthorized access.
- Enforce **Azure Policy** to define security rules.
- Implement **network policies** for controlled traffic flow.

2. Identity and Access Management (IAM)

- Use **Azure Active Directory (AAD)** for authentication.
- Enable **Role-Based Access Control (RBAC)** to limit permissions.

3. Data Protection

- Enable **encryption at rest and in transit**.
- Implement **secure secrets management** using Azure Key Vault.

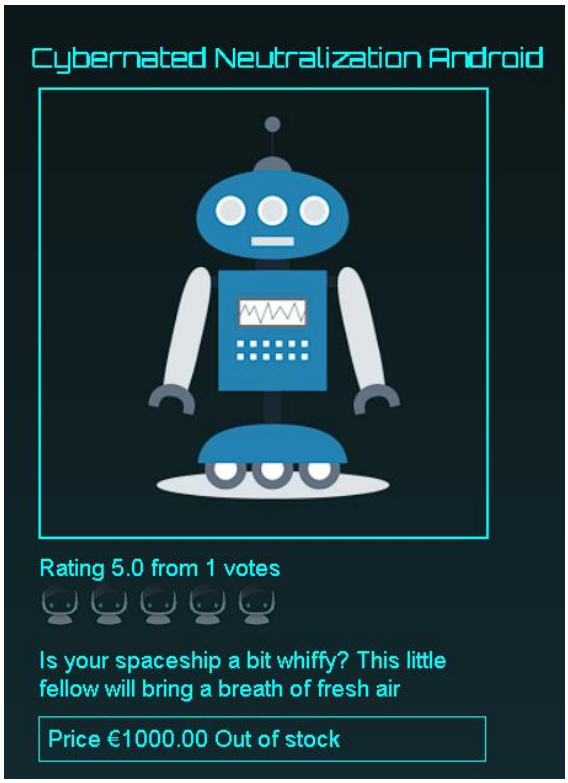
By applying security best practices, the application remains **protected against threats, unauthorized access, and data breaches**.

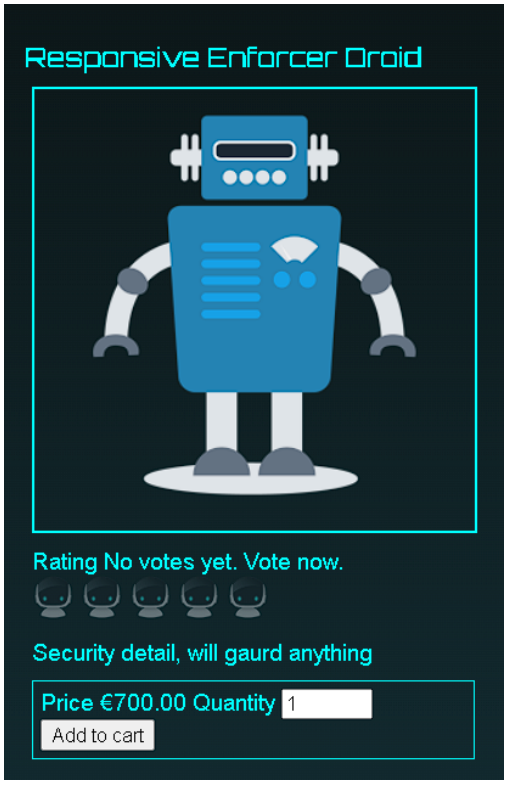
Step 9: Cleaning Up Resources

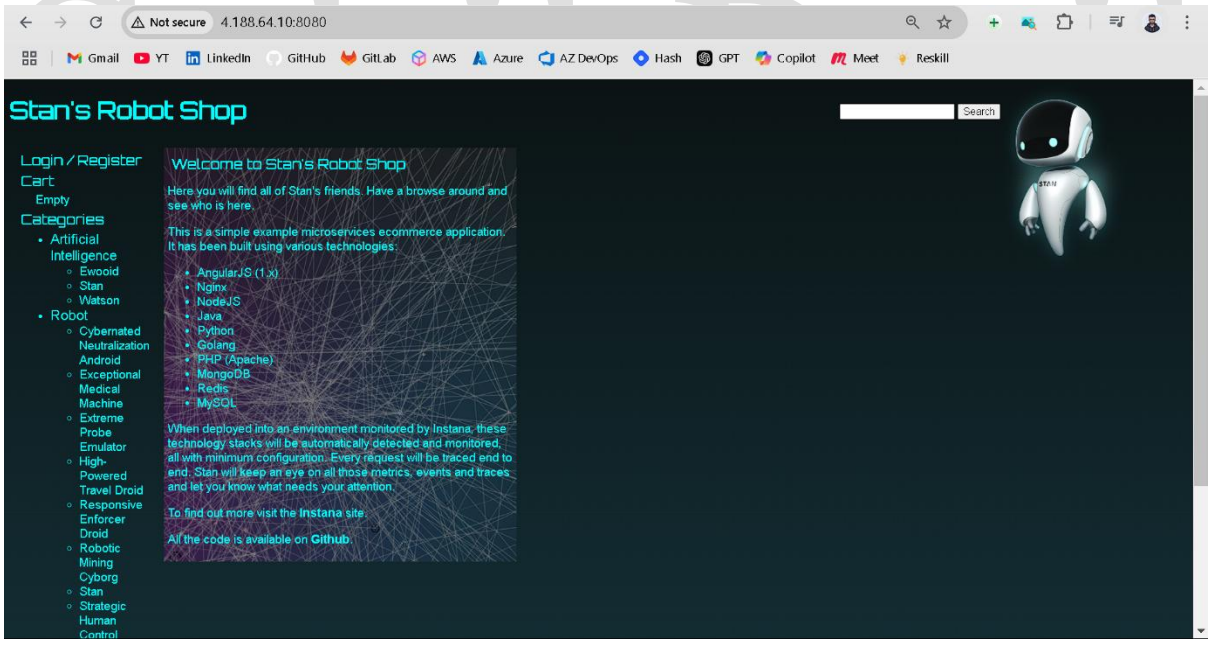
To **avoid unnecessary costs**, it is important to delete unused resources once the deployment is no longer required. This includes:

- **Deleting the AKS cluster** if not needed.
- **Removing storage and container registry** resources.
- **Cleaning up DNS records and ingress configurations**.

Efficient resource management ensures cost savings and prevents security vulnerabilities.


Cybernated Neutralization Android
Rating 5.0 from 1 votes
Is your spaceship a bit whiffy? This little fellow will bring a breath of fresh air
Price €1000.00 Out of stock


Responsive Enforcer Droid
Rating No votes yet. Vote now.
Security detail, will gaurd anything
Price €700.00 Quantity 1
Add to cart


Not secure 4.188.64.10:8080
Gmail YT LinkedIn GitHub GitLab AWS Azure AZ DevOps Hash GPT Copilot Meet Reskill
Stan's Robot Shop
Search
Login / Register
Cart
Empty
Categories
Artificial Intelligence
Ewooid
Stan
Watson
Robot
Cybernated Neutralization Android
Exceptional Medical Machine
Extreme Probe Emulator
High-Powered Travel Droid
Responsive Enforcer Droid
Robotic Mining Cyborg
Stan
Strategic Human Control
Welcome to Stan's Robot Shop
Here you will find all of Stan's friends. Have a browse around and see who is here.
This is a simple example microservices ecommerce application. It has been built using various technologies:
AngularJS (1.x)
Nginx
NodeJS
Java
Python
Golang
PHP (Apache)
MongoDB
Redis
MySQL
When deployed into an environment monitored by Instana, these technology stacks will be automatically detected and monitored, all with minimum configuration. Every request will be traced end to end. Stan will keep an eye on all those metrics, events and traces and let you know what needs your attention.
To find out more visit the Instana site.
All the code is available on Github.

Conclusion

This guide provided a **comprehensive roadmap** to **deploy an e-commerce project on AKS**. By following these steps, the application benefits from **scalability, security, and high availability** while leveraging **Azure's powerful Kubernetes ecosystem**.

With a properly configured **AKS cluster, networking, storage, monitoring, and security setup**, the e-commerce project is well-equipped for **seamless cloud operations and future scalability**.
