TUNISIAN REUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage

National Institute of Applied Sciences and Technology

Deploying and monitoring a multi-container application to Azure Kubernetes Services automated with Azure DevOps

End of year project Report

Prepared By:

Mohamed Sadok MASGHOUNI
Youssef JOUILI
Haroun JAOUADI
Rania MIDAOUI

Supervised by:

M. Bassem BEN SALAH , INSAT

06-2023

Abstract

This project report presents our end of year project, and the different approaches and steps taken to complete it. As students at the National Institute of Applied Sciences and Technology, our objective was to develop and containerize an application, then to successfully deploy and monitor it using modern cloud technologies, and DevOps approaches.

Throughout the project, we focused on automating the build and release process, resulting on more reliable and consistent processes, and by that, reducing effort considerably and eliminating human errors due to misconfigurations and inconsistencies. Furthermore, we dedicated time to ensure the security of our application and deployment pipeline, monitoring and logging the solution's activities not only to detect errors and get alerts if our application fails, but also to detect and respond to any suspicious behavior.

The report aims to document our project journey by presenting the project's methodology and the approach employed to achieve the desired outcomes, while providing an overview of the project's infrastructure and deployment architecture.

Contents

List of Figures

Introduction

In recent years, there has been a significant shift in how organizations handle software applications, thanks to cloud computing and DevOps practices. Cloud platforms like Azure have become popular choices for businesses worldwide due to their scalability and efficiency. They provide enhanced reliability for application deployment and they most importantly, they are cost-efficient because, instead of investing in and maintaining physical infrastructure, organizations can leverage the pay-as-you-go model provided by cloud platforms.

With this shift, organizations kept looking for solutions to automate the deployment process, which offers numerous benefits such as accelerated deployment and scalability. As a result, application can now automatically adjust its resources to handle the load, ensuring a smooth user experience, while ensuring that businesses respond quickly to market demands, deliver new features and updates at a rapid pace, and gain a competitive edge at the same time.
In addition to automation, monitoring and log analysis are essential for improving the deployment process. Monitoring tools keep a close eye on how the system behaves, how users interact with it, and any potential security threats. By analyzing logs, organizations gain valuable insights that help enhance application reliability, detect issues in a timely manner, and troubleshoot effectively.

In this context, we worked on our end of year's project and we tried to leverage some of the advantages of these new technologies and cultures. Our project's main goal is to deploy and monitor a multi-container application to a Cloud solution which is, in our case, Azure using a managed container orchestration tool (Azure Kubernetes Services) and to automate the deployment process following DevOps tools and best practices.

This report is structured into four chapters. The first chapter provides an overview of project and its scope. It provides a theoretical study of the subject matter. The second chapter describes the deployment of an application and the automation process. It introduces the concepts and principles related to the services and tools used, and the steps to reach the goal. The third chapter focuses on monitoring the solution, log analysis and securing the pipeline. The fourth chapter is dedicated to the implementation and testing the deployments process and automation pipelines on a multi-container e-commerce application.

Chapter 1

Project scope for deploying a multi-container application

Introduction

In this section, we will introduce the project by highlighting its context, the problematic and the requirements it must follow to achieve the desired goals. Then we will give a general overview for the system's architecture.

1.1 Project Overview

We undertook this project to deploy a multi-container application to Azure Kubernetes Services. Our goal was to reduce effort put in the deployment process of an application, while minimizing the time, all of that, using DevOps approaches. This is done by implementing an automation solution, using a CI/CD (Continuous Integration/ Continuous Deployment) pipeline to integrate then deploy each change made to the application without the need to trigger the processes manually. Furthermore, we made sure to implement a monitoring solution on the Kubernetes Cluster and to secure the pipeline.

1.1.1 Problematic and Motivation

In recent years, the emergence of containerization, DevOps and cloud computing has brought significant advancements in the field of technology, surpassing the capabilities of traditional approaches. Engineers are facing the challenge of adapting to the changing landscape of technology because the traditional approach of managing on-premises infrastructure and deploying applications turned out to have many limitations in terms of scalability, resource utilization, and also availability challenge. There is a pressing need for engineers to embrace cloud computing and DevOps solutions to overcome these challenges and stay competitive in the IT industry. We aim to make deployment surpass all these challenges and obey to the DevOps culture, by creating automated, scalable, monitored and secure architectures.

To bring this project to life, we have to know the functional and non functional requirements for the solution:

- Functional Requirements:

1. The system should support the deployment of a multi-container application consisting of frontends, backends, and a database.
2. The application should be deployed using Docker containers.
3. The system should set up a Kubernetes cluster for managing and orchestrating the application's components.
4. Automatic scaling and load balancing should be implemented for efficient resource utilization.
5. The application should be accessible through an ingress controller for external traffic routing.
6. Continuous integration and continuous deployment (CI/CD) pipelines should be established for seamless application updates.
7. Monitoring and logging solutions should be integrated to track the performance and health of the application.

- Non-Functional Requirements:

1. Security measures should be implemented to protect the application and its data.
2. The system should be highly available and resilient to ensure minimal downtime.
3. Performance optimizations should be considered to achieve efficient resource utilization.
4. Scalability should be supported to handle increased workload and user traffic.
5. The system should be cost-effective by optimizing resource usage and minimizing operational expenses.

1.1.2 General architecture design

This figure 1.1 presents the general architecture of the project. It gives an overview of how the automated deployment process will be triggered, starting from a pushed change in the application code to any version control system, the flow of the metrics and logs that will be collected and visualized, and the redirection of the request sent by a client to the web application.

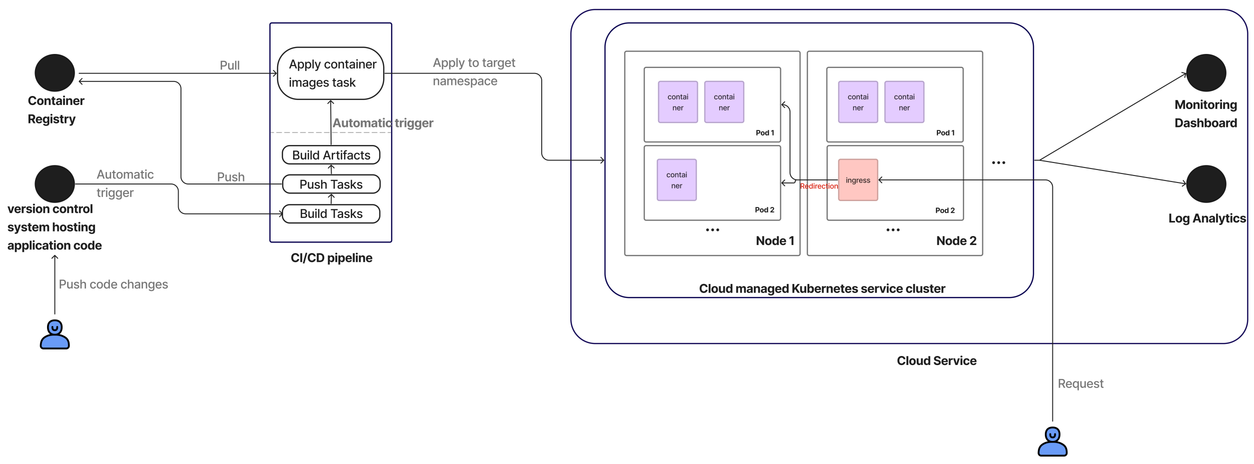


Figure 1.1: Deployment Architecture Overview

When a user sends a request for example to log into the application, it will begin with a Fully Qualified Domain Name (FQDN) followed by the path to the service (login, register ...) he wants to access, that domain name is resolved into the ingress' public IP address. Then, the ingress controller will forward the request to the right service based on the subdomain mentioned in the FQDN, which will forward it to the right pod. After receiving the request, the pod will treat it and generate a response that returns all the way back to the browser.

1.2 Deployment Workflow for a Multi-Container Web Applications

In our project, we have developed a solution for automating and monitoring deployment for a multi-container web applications. Let's walk through the steps involved in this process:

IDE: We used an integrated development environment (IDE) to write and manage our application code. It provided us with a user-friendly interface to create and edit the code for our web application.

Code Repository: The application code is stored in a version control system repository, in our case, GitHub. This repository serves as a central location to keep track of changes made to the code and collaborate with other team members.

CI/CD Pipeline: We used a continuous integration and continuous deployment (CI/CD) pipeline, which is managed by Azure DevOps. This pipeline automates the process of building, testing, and deploying our application to the Azure Kubernetes Services cluster.

Cluster Setup: Azure pipelines will trigger the set up for our Azure Kubernetes Services (AKS) cluster. This cluster acts as the foundation for running our application in a scalable and reliable manner.

Monitoring: Azure Monitor collects the metrics and logs from the AKS cluster and sends them to a Log Analytics Workspace. We used these metrics with a third party monitoring solution called Grafana for dashboard creation. We also added alert rules to receive emails upon critical or error events happening in the cluster.

Log Analysis: we utilized a Log Analytics workspace to gain valuable insights from our system logs. The Log Analytics workspace acts as a centralized repository where we collect and analyze logs from our cluster. By analyzing these logs, we extracted meaningful information about system behavior, audit, errors, auto scaling, etc.

1.2.1 Gantt Diagram

The project implementation took 3 months, below in figure 1.2 is the Gantt diagram that illustrates the project timeline:

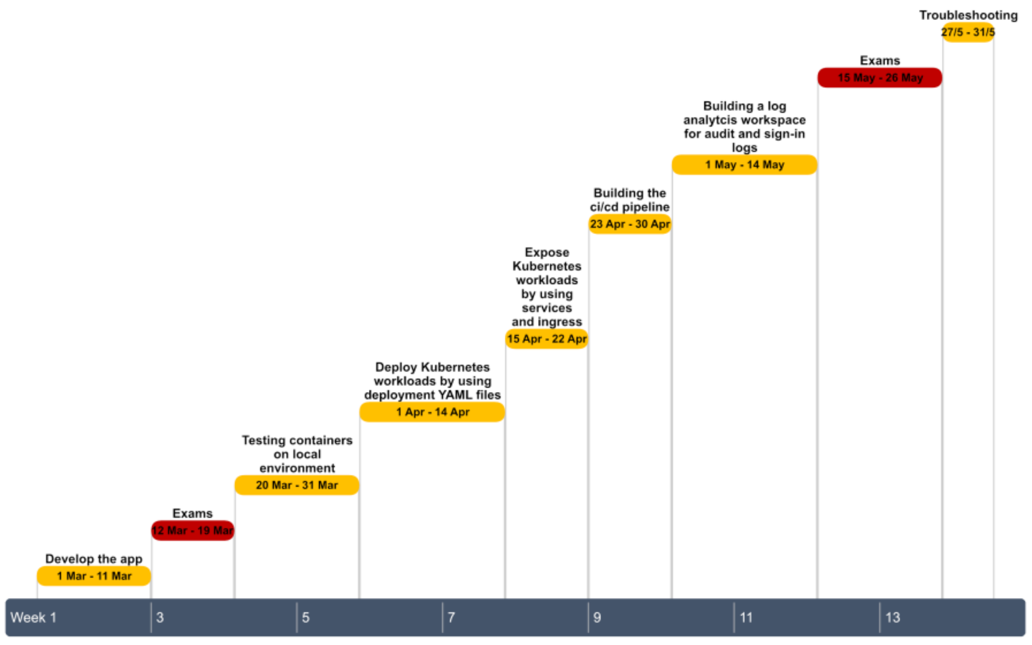


Figure 1.2: Gantt Diagram

Conclusion

In this section, we were able to explain briefly the problematic, motivation and context of this project. we gave an overview of the project's architecture and how it may overcome real-world obstacles in the constantly changing realm of technology. We also talked about the steps we followed to achieve our goal and the timeline of each task.

Chapter 2

Deployment to Azure Cloud Service

Introduction

In this chapter, we will cover the process of organizing our application into multiple containers, and deploying it using Docker and Kubernetes. Then, we will set up a Cloud provider managed Kubernetes cluster to manage these containers and ensure their good behavior. Next, we will discuss the CI/CD pipeline, which helped us automate the building, testing, and deployment of our application.

2.1 Containerizing the Application and Setting up a Kubernetes Cluster

In this section, we will explain how we can deal with the application into multiple containers and prepare them for deployment. Additionally, we will dive into the steps involved in establishing a Kubernetes cluster.

2.1.1 Containerizing the Application using Docker

2.1.1.1 Organizing and Preparing the Application for Deployment

The workflow was organized into some principal steps that we followed during the process of preparation of the application. The following steps, show the main work we went through.

2.1.1.2 Analyzing Application Structure

Analyzing Application Components: We began by carefully analyzing the different components of our application, understanding their functionalities, and identifying any dependencies or specific requirements they had.
Breaking Down the Application: We divided our application into smaller, manageable components based on their functionalities and to ensure scalability and maintainability.

2.1.1.3 Building and Configuring Docker Images for Application Components

What is Docker and Containers?
Docker is an open-source platform that allows us to automate the deployment and management of applications within containers.[1]
Now let's talk about the concept of containers, think of them as lightweight, portable and isolated virtual environments that contain everything needed to run an application, including the code, runtime, system tools, and libraries. Docker simplifies the process of packaging and distributing applications, making them portable and consistent across different environments.
Why using Docker architecture and not the traditional Virtual Machines architecture?

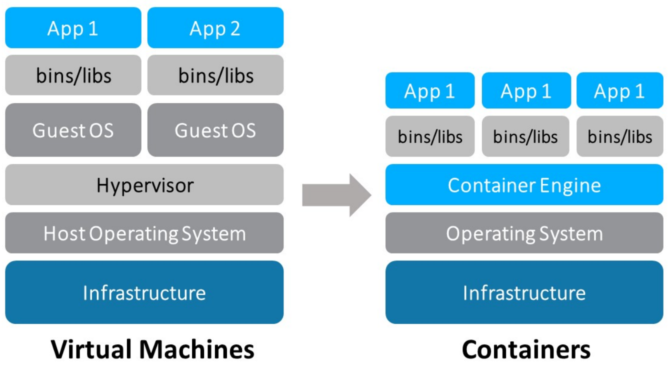


Figure 2.1: Virtual Machines to Docker and containers architecture

The figure 2.1 shows the two different architectures.

2.1.1.4 The steps of containerization

We will discuss the process of containerizing a multi-container application using Docker.

- **Writing Dockerfiles**: To build the Docker images, we created Dockerfiles. These files provided instructions on how to assemble the image.
- **Building Docker Images**: Using the Dockerfiles, we built the Docker images for each component. This process involved running the "docker build" command, which compiled the images based on the instructions provided in the Dockerfiles.
- **Local Testing**: We performed thorough testing on the locally built Docker images to ensure that each component functioned correctly.
- **Pushing Docker images to a a container registry**

2.1.2 Setting up a Kubernetes Cluster

Before Kubernetes, managing containerized applications required manual intervention and coordination. We had to ensure that each container was running correctly, handle scaling based on traffic demands, and manage the overall health of the application. This process was time-consuming and error-prone.

2.1.2.1 What is Kubernetes

Kubernetes is an open-source container orchestration platform that helps us manage and deploy our Docker images effectively.[2]
The figure 2.2 shows the architecture of the kubernetes clusters, the master node and the worker nodes.

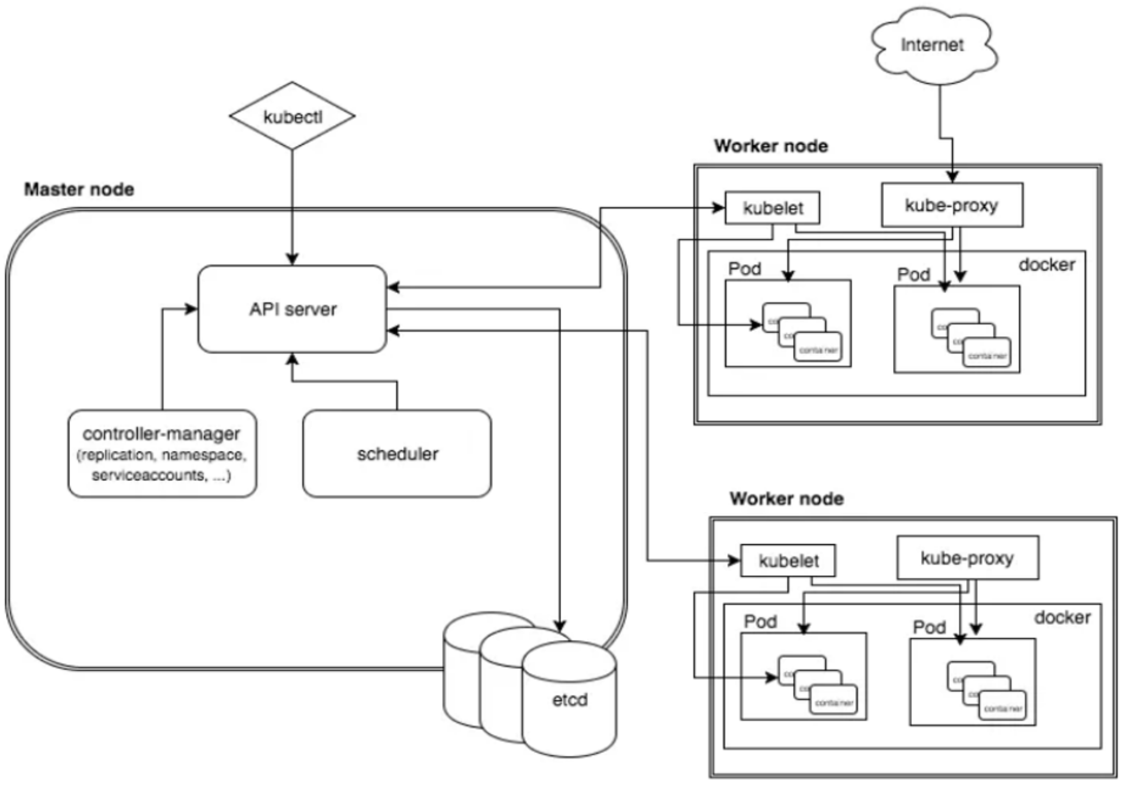


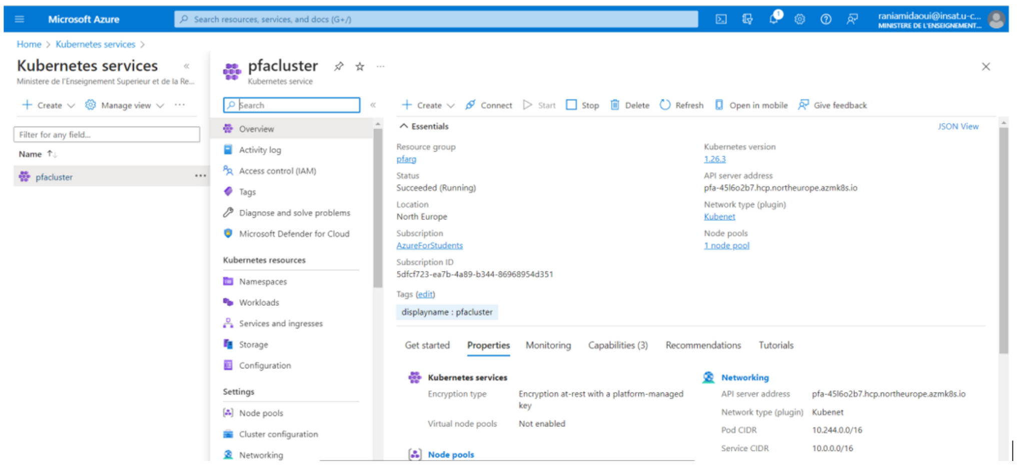
Figure 2.2: High level Kubernetes architecture diagram showing a cluster with a master worker nodes

With Kubernetes, we can automate the tasks of handling our containers and maintaining them. It provides a centralized control system that takes care of managing and monitoring the containers for us. We define the desired state of our application, including the number of containers, their configurations, and any scaling requirements, all this is done in configuration YAML files.

2.1.2.2 Deployment on Azure Kubernetes Service

We started by setting up a Kubernetes cluster in Azure Kubernetes Services. This involved configuring the necessary infrastructure, such as determining the number of machines (nodes) and establishing network connections.

Once the Kubernetes cluster was ready, we applied the deployments and services YAML files to the right namespaces in our Cluster, we used the default namespace for all our containers. During this phase, Docker images containing all the components needed for our application, like frontends, backend, and database were deployed. Kubernetes automatically manages and scaled these containers, taking care of the complex management tasks and allowing us to focus on developing and maintaining our application while AKS took care of the underlying infrastructure. In Figure 2.3 you can see an illustration of our Kubernetes cluster deployed in AKS.



Resource group: pfarg
Status: Succeeded (Running)
Location: North Europe
Subscription: AzureForStudents
Subscription ID: 5dfcf723-ea7b-4a89-b344-86968954d351
Tags (edit): displayname : pfacluster
Kubernetes version: 1.26.3
API server address: pfa-45l6o2b7.hcp.northeurope.azmk8s.io
Network type (plugin): Kubenet
Node pools: 1 node pool

Kubernetes services
Encryption type: Encryption at-rest with a platform-managed key
Virtual node pools: Not enabled

Networking
API server address: pfa-45l6o2b7.hcp.northeurope.azmk8s.io
Network type (plugin): Kubenet
Pod CIDR: 10.244.0.0/16
Service CIDR: 10.0.0.0/16

Figure 2.3: Kubernetes cluster deployed in the AKS

AKS Ingress is a Kubernetes resource that acts as a traffic gateway, allowing external access to services within the cluster. It manages the routing and load balancing of incoming traffic to the appropriate services, providing a centralized entry point for external communication with the previously deployed AKS cluster.In Figure

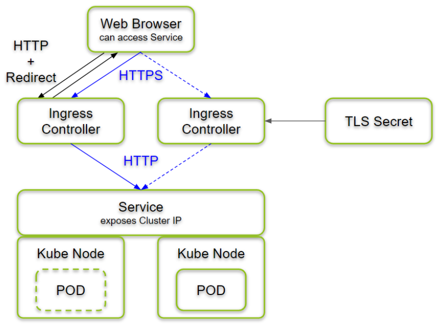
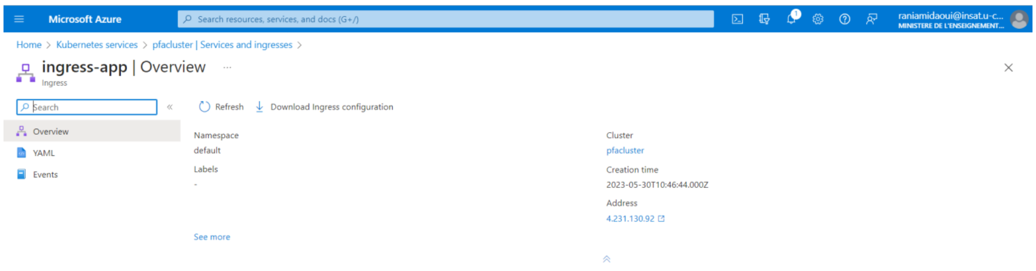


Figure 2.4: Ingress architecture diagram

2.5, you can see an illustration of the ingress service in the AKS cluster.


Microsoft Azure
Search resources, services, and docs (G+/)
raniamidaoui@insat.u-c...
MINISTERE DE L'ENSEIGNEMENT...
Home > Kubernetes services > pfacluster | Services and ingresses >
Search
Refresh
Download Ingress configuration
Overview
YAML
Events
See more

ingress-app | Overview
Ingress

Namespace
default
Labels
-

Cluster
pfacluster
Creation time
2023-05-30T10:46:44.000Z
Address
4.231.130.92

Figure 2.5: ingress service in the AKS cluster

2.1.3 Continuous Integration and Continuous Delivery (CI/CD) Pipeline with Azure DevOps

In software development, having an efficient and reliable process for building, testing, and deploying applications is essential. This is where the Continuous Integration and Continuous Deployment (CI/CD) pipeline comes into play. It allows teams to automate these processes, leading to faster delivery and better quality software. We will explore how we implemented a CI/CD pipeline using Azure DevOps to deploy a multi-container application to Azure Kubernetes Services (AKS). Azure DevOps, provided by Microsoft, is a set of tools and services that help manage the CI/CD pipeline.

2.1.3.1 What is CI/CD pipeline

A Continuous Integration/Continuous Delivery (CI/CD) pipeline is a set of predefined steps that are followed to deliver a new version of software. It helps improve the process of software delivery by using a DevOps approach. While it's possible to manually perform each step of the pipeline, this doesn't fully utilize its capabilities. With a CI/CD pipeline, we can automate the steps and enhance integration, testing, delivery, and deployment stages. This automation unlocks the true value of application development and ensures more efficient software delivery.

2.1.3.2 Continuous Integration

Continuous integration (CI) is the process of automatically combining code changes from multiple developers into a single software project. It is a fundamental DevOps practice that involves regularly merging code changes into a shared code repository for building and testing. To ensure the new code is valid and to prevent easily detectable conflicts, automated tools are commonly used. Figure 2.6 provides an overview of the CI workflow and demonstrates how it operates.

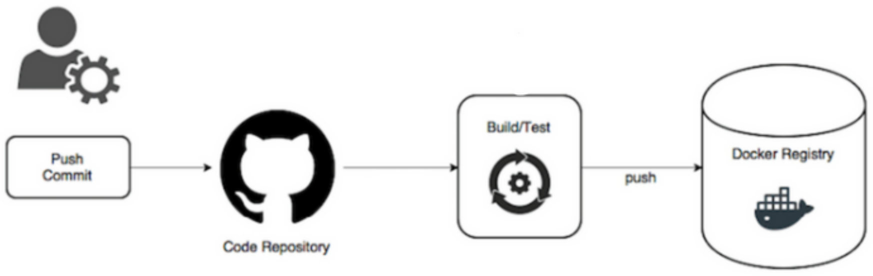


Figure 2.6: Continious Delivery workflow

2.1.3.3 Continuous Delivery

Continuous delivery (CD) is a practice that enables the creation of an automated release process. It involves an iterative feedback loop that focuses on delivering software updates to end users as quickly as possible. This allows for learning from their experiences and incorporating their feedback into future releases and provides a visual representation of the CD workflow.

2.1.3.4 Benefits

Integrating a CI/CD pipeline into a project's workflow offers the following advantages:

- Improved focus for developers: Developers can concentrate on enhancing the software, creating new code, and monitoring its behavior in production.
- Reduced development cost and complexity: Constant product updates become less stressful as the CI/CD pipeline streamlines the development process, resulting in cost and complexity reduction.
- Enhanced traceability: The CI/CD pipeline maintains clear logs of changes, tests, and deployments, ensuring traceability and facilitating troubleshooting.
- Easy rollback in case of production failure: If a failure occurs in the production environment, reverting back to a stable previous version is straightforward.
- Cultivation of adaptability and responsibility: The fast feedback cycle provided by the CI/CD pipeline encourages a culture of adaptation and accountability throughout the organization.

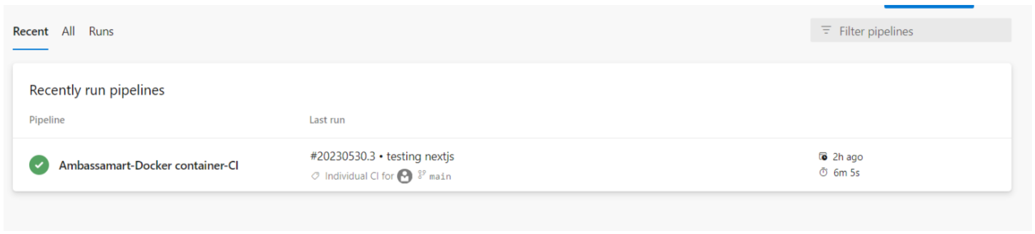
2.1.3.5 Pipeline stages

A CI/CD pipeline is composed of various components that organize specific sets of tasks into stages, which are as follows:

- **Source stage**: This stage is responsible for triggering the pipeline. It is commonly triggered by a source code repository, but it can also be triggered by a scheduler or the completion of other pipelines.
- **Build stage**: In this stage, the application is compiled to create a runnable version.
- **Test stage**: Here, automated tests are executed to ensure that the code functions as expected. This helps identify and prevent easily reproducible bugs.
- **Deploy stage**: This final stage deploys the code to the appropriate environment, such as development, staging, or production.

2.1.3.6 Streamlining Docker Image Building and Testing

To automate the creation of Docker images, we created a service connection from the Azure DevOps Project to Docker Hub to allow the CI/CD pipeline to push and pull container images. After each code change, the CI/CD pipeline automatically built and pushed the Docker images to the Docker Hub repository. This ensured that the latest version of our application was readily available for deployment.
We also created a service connection to Github, giving the CI/CD pipeline the right permissions to the repository containing the application code.. Whenever new code was pushed to the repository, the CI/CD pipeline automatically triggered the Docker image building and testing processes, enabling a continuous integration and deployment workflow.
This ensured that our application remained up-to-date, tested, and ready for deployment with minimal manual intervention.Figure 2.7 shows the pipeline we made in our project.



Recently run pipelines

Pipeline	Last run	
✓ Ambassamart-Docker container-CI	#20230530.3 • testing nextjs Individual CI for main	2h ago 6m 5s

Figure 2.7: Azure Devops pipeline

Conclusion

In this chapter, we have seen how containerizing the application and setting up the Kubernetes cluster, along with integrating a CI/CD pipeline using Azure DevOps, have significantly improved our software development process. Docker helped us package the application and its dependencies into portable containers, ensuring consistent deployments.

Chapter 3

Security, Monitoring and Logging

Introduction

In this chapter, we will discuss how monitoring and security play a crucial role in ensuring optimal performance and delivering the best user experience. We will detail how we applied these concepts on our Kubernetes Cluster and the application it's running.

3.1 Securing the CI/CD Pipeline

Scanning code and images for vulnerabilities is vital for securing the CI/CD pipeline. By using special tools, we can find and fix vulnerabilities or detect the presence of a malware before its too late. Regular scanning guarantees that only safe and trusted code and images are used.

3.1.1 Vulnerability Scanning tools

Various scan tools are commonly used to secure the CI/CD pipeline. They help find possible security issues, misconfigurations, and other problems in the pipeline. By adding these tools to the pipeline, we can take action early on to address security concerns and make sure the software they deliver is secure. The figure 3.1 shows how we implemented security in our workflow.

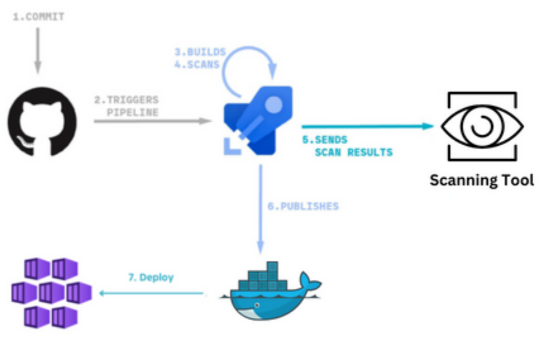
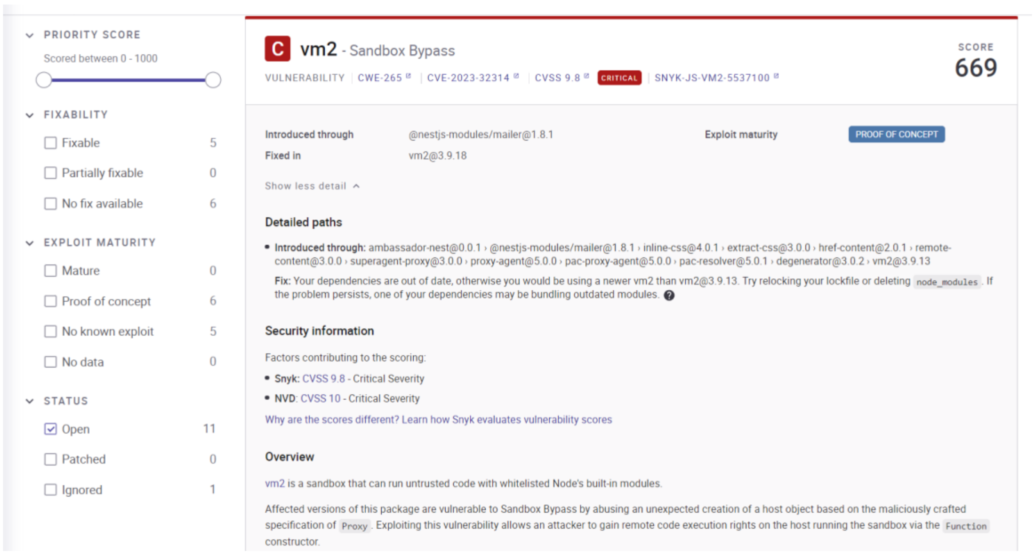


Figure 3.1: Injection of a Scanning Tool

3.1.2 Snyk

Snyk is a developer security platform. Integrating directly into development tools, workflows, and pipelines, Snyk simplifies the detection, prioritization, and remediation of security vulnerabilities in code, dependencies, containers, and infrastructure as code (IaC).[3] The results of a Snyk scan are typically presented in a clear and organized manner. Commonly, they are displayed in a user-friendly dashboard or report. This dashboard provides an overview of the security vulnerabilities detected, including details such as the affected components, severity levels, and suggested remediation steps. It may also include visual indicators or color-coded labels to highlight the criticality of each vulnerability.


PRIORITY SCORE
Scored between 0 - 1000

FIXABILITY

☐ Fixable	5
☐ Partially fixable	0
☐ No fix available	6

EXPLOIT MATURITY

☐ Mature	0
☐ Proof of concept	6
☐ No known exploit	5
☐ No data	0

STATUS

☑ Open	11
☐ Patched	0
☐ Ignored	1

C vm2 - Sandbox Bypass
VULNERABILITY | CWE-265 | CVE-2023-32314 | CVSS 9.8 CRITICAL | SNYK-JS-VM2-5537100
SCORE 669

Introduced through @nestjs-modules/mailer@1.8.1
Fixed in vm2@3.9.18
Exploit maturity PROOF OF CONCEPT

Show less detail

Detailed paths

- Introduced through: ambassador-nest@0.0.1 › @nestjs-modules/mailer@1.8.1 › inline-css@4.0.1 › extract-css@3.0.0 › href-content@2.0.1 › remote-content@3.0.0 › superagent-proxy@3.0.0 › proxy-agent@5.0.0 › pac-proxy-agent@5.0.0 › pac-resolver@5.0.1 › degenerator@3.0.2 › vm2@3.9.13

  Fix: Your dependencies are out of date, otherwise you would be using a newer vm2 than vm2@3.9.13. Try relocking your lockfile or deleting `node_modules`. If the problem persists, one of your dependencies may be bundling outdated modules.

Security information

Factors contributing to the scoring:

- Snyk: CVSS 9.8 - Critical Severity
- NVD: CVSS 10 - Critical Severity

Why are the scores different? Learn how Snyk evaluates vulnerability scores

Overview

vm2 is a sandbox that can run untrusted code with whitelisted Node's built-in modules.

Affected versions of this package are vulnerable to Sandbox Bypass by abusing an unexpected creation of a host object based on the maliciously crafted specification of `Proxy`. Exploiting this vulnerability allows an attacker to gain remote code execution rights on the host running the sandbox via the `Function` constructor.

Figure 3.2: Example of Snyk security report

This figure 3.2 shows an example of a critical issue detected by Snyk.

To add security to the deployment of our pipelines, we integrated Snyk to analyze the application code in the Github repository where it is stored, and the Docker images stored in Docker Hub. To make this possible, we followed these steps:
1- First we create an account on the official Snyk website.
2- From the projects blade, we add a new project, we choose Github as source for the vulnerability analysis, and connect Snyk to the Github repository, allowing it to access its content. Then, we add another project and this time, we link a Docker Hub account and give Snyk access to the images.
3- We trigger an analysis of it is not automatically triggered, and analyze the results.

3.2 Enabling secure HTTPS connections to the application

HTTPS(Hyper Text Transfer Protocol Secure) ensures that information shared between web servers and clients is encrypted .

3.2.1 Difference between HTTP and HTTPS

HTTPS is an enhanced version of the Hypertext Transfer Protocol (HTTP), where the "S" stands for "secure." It is used to secure communication on the internet by encrypting data exchanged between a web browser and a website. With HTTPS, user page requests and server responses are encrypted, ensuring the confidentiality and integrity of the transmitted data.
This figure 3.7 shows the main difference between HTTP protocol and HTTPS.

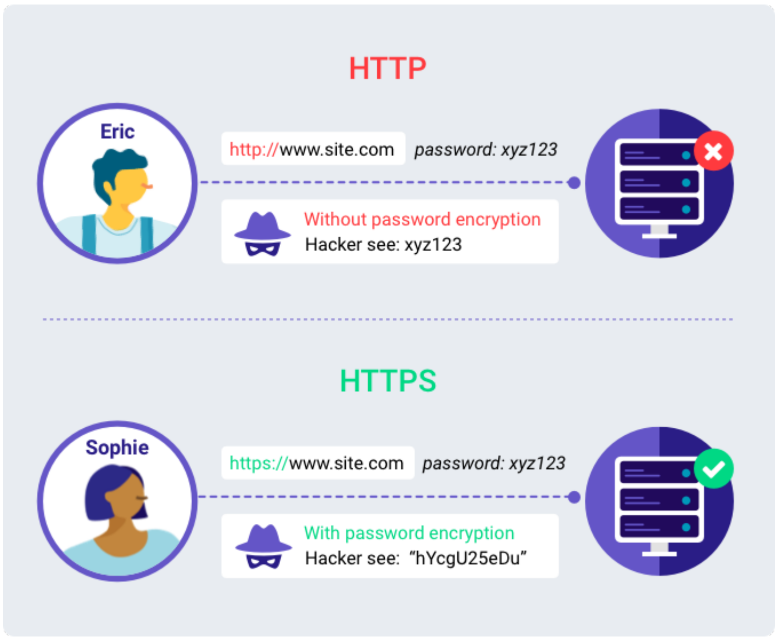


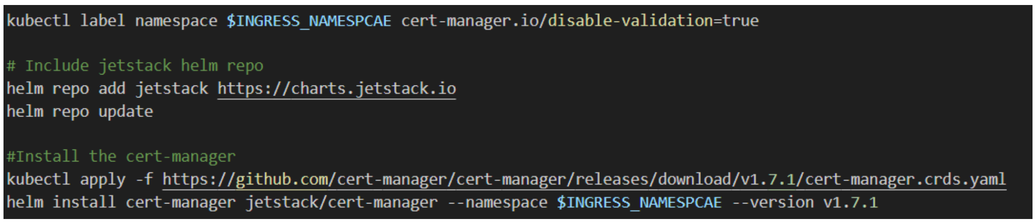
Figure 3.3: Difference between HTTP and HTTPS

3.2.2 Cert-manager and TLS certificates

An SSL/TLS(Secure Sockets Layer/Transport Layer) certificate is a digital object that allows systems to verify the identity subsequently establish an encrypted network connection to another system using the Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol. [4]

cert-manager adds certificates and certificate issuers as resource types in Kubernetes clusters, and simplifies the process of obtaining, renewing and using those certificates. It supports issuing certificates from a variety of sources, including Let's Encrypt (ACME), HashiCorp Vault, and Venafi TPP / TLS Protect Cloud, as well as local in-cluster issuance. cert-manager also ensures certificates remain valid and up to date, attempting to renew certificates at an appropriate time before expiry to reduce the risk of outages and remove toil.[5]

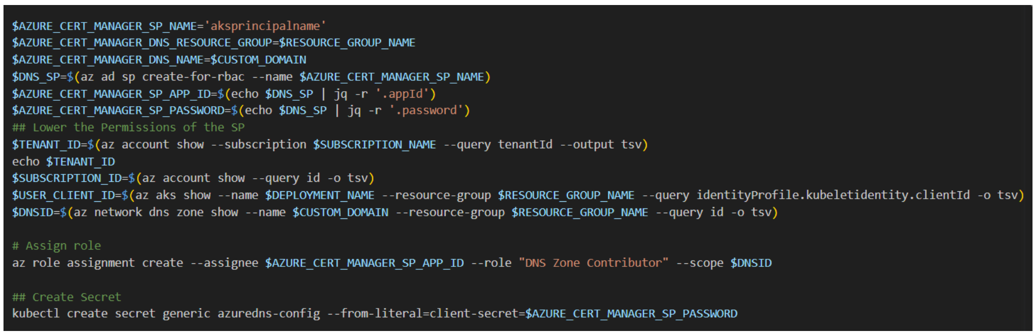
To switch from HTTP to HTTPS as the default protocol used for communicating with the application, several steps need to be followed: 1- Install cert-manager for SSL certificates in ingress-basic namespace using Helm.



```
kubectl label namespace $INGRESS_NAMESPCAE cert-manager.io/disable-validation=true

# Include jetstack helm repo
helm repo add jetstack https://charts.jetstack.io
helm repo update

#Install the cert-manager
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.7.1/cert-manager.crds.yaml
helm install cert-manager jetstack/cert-manager --namespace $INGRESS_NAMESPCAE --version v1.7.1
```

Figure 3.4: Difference between HTTP and HTTPS

2- Create a cluster issuer for issuing certificate. 3- Create an ingress route to configure the host based rules along with DNS record and TLS certificate that route traffic to one of the two applications.



```
$AZURE_CERT_MANAGER_SP_NAME='aksprincipalname'
$AZURE_CERT_MANAGER_DNS_RESOURCE_GROUP=$RESOURCE_GROUP_NAME
$AZURE_CERT_MANAGER_DNS_NAME=$CUSTOM_DOMAIN
$DNS_SP=$(az ad sp create-for-rbac --name $AZURE_CERT_MANAGER_SP_NAME)
$AZURE_CERT_MANAGER_SP_APP_ID=$(echo $DNS_SP | jq -r '.appId')
$AZURE_CERT_MANAGER_SP_PASSWORD=$(echo $DNS_SP | jq -r '.password')
## Lower the Permissions of the SP
$TENANT_ID=$(az account show --subscription $SUBSCRIPTION_NAME --query tenantId --output tsv)
echo $TENANT_ID
$SUBSCRIPTION_ID=$(az account show --query id -o tsv)
$USER_CLIENT_ID=$(az aks show --name $DEPLOYMENT_NAME --resource-group $RESOURCE_GROUP_NAME --query identityProfile.kubeletidentity.clientId -o tsv)
$DNSID=$(az network dns zone show --name $CUSTOM_DOMAIN --resource-group $RESOURCE_GROUP_NAME --query id -o tsv)

# Assign role
az role assignment create --assignee $AZURE_CERT_MANAGER_SP_APP_ID --role "DNS Zone Contributor" --scope $DNSID

## Create Secret
kubectl create secret generic azuredns-config --from-literal=client-secret=$AZURE_CERT_MANAGER_SP_PASSWORD
```

Figure 3.5: Difference between HTTP and HTTPS

4- Verify the automatic created certificate. 5- Test the applications using Custom Domain, it may take some time for the certificate to be applied.

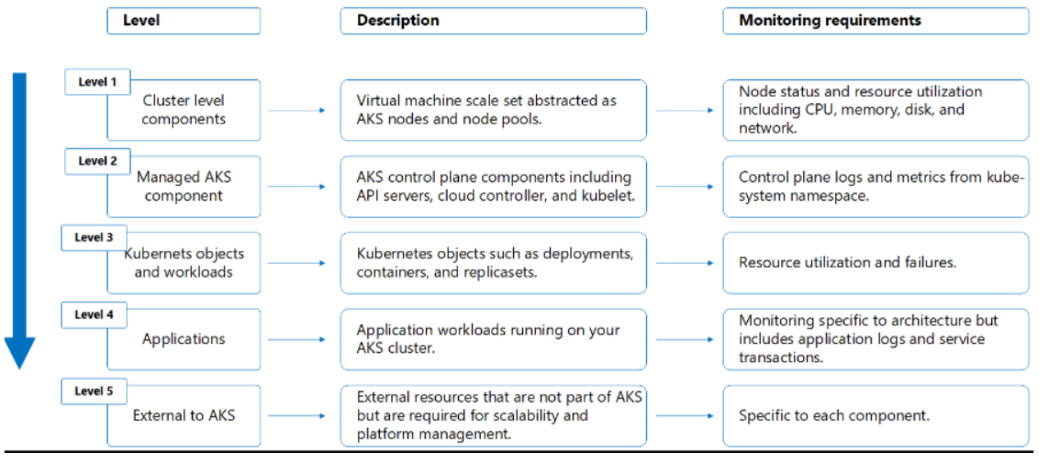
3.3 Monitoring and Metrics Visualization

In this section, we will discuss the importance of monitoring and metrics visualization in managing and improving systems, applications, and infrastructure.

3.3.1 Monitoring approach

Monitoring is essential for maintaining the health, performance, and security of systems, applications, and infrastructure. A proactive monitoring approach helps in identifying and addressing issues before they impact the system. Key metrics are monitored to ensure optimal operation, and effective strategies are implemented to ensure continuous monitoring.[6]

In our project, we chose to use a common bottoms-up approach, starting from infrastructure up through applications. Each layer has distinct monitoring requirements.[7]



Level	Description	Monitoring requirements
Level 1 Cluster level components	Virtual machine scale set abstracted as AKS nodes and node pools.	Node status and resource utilization including CPU, memory, disk, and network.
Level 2 Managed AKS component	AKS control plane components including API servers, cloud controller, and kubelet.	Control plane logs and metrics from kube-system namespace.
Level 3 Kubernets objects and workloads	Kubernetes objects such as deployments, containers, and replicasets.	Resource utilization and failures.
Level 4 Applications	Application workloads running on your AKS cluster.	Monitoring specific to architecture but includes application logs and service transactions.
Level 5 External to AKS	External resources that are not part of AKS but are required for scalability and platform management.	Specific to each component.

Figure 3.6: Difference between HTTP and HTTPS

3.3.2 Azure monitor

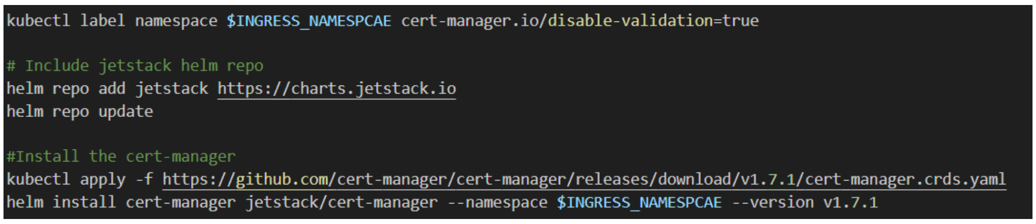
Azure Monitor is a powerful monitoring solution that gathers and analyzes data from your cloud and on-premises environments. It helps ensure the availability and performance of your applications and services. By collecting data from various components of our system and storing it in a centralized Log Analytics workspace, Azure Monitor allows for correlation and analysis using a set of shared tools. This data can be used to gain insights into application performance and automate responses to system events. Visualizations and analysis help you understand how your applications are functioning and make informed decisions. We used Azure monitor as our main solution to collect metrics and logs by enabling it at the cluster level from the insights and Diagnostic settings blades.[8]

3.3.3 Log Collection with Log Analytics Workspace

Log Analytics Workspace, a feature within Azure Monitor, facilitates centralized log collection, storage, and analysis. Logs from various sources can be collected and sent to a Log Analytics Workspace for further analysis. Configuration options are available for data sources and ingestion, and the query language helps in analyzing logs and extracting insights.[9] We analyzed the logs collected using Kusto Query Language (KQL) queries and stored on the Log Analytics Workspace.

3.3.4 Creating Grafana Dashboards

We used Grafana, an open-source platform, is used to create interactive and customizable dashboards. By leveraging Azure Monitor Logs, Grafana can be integrated to visualize metrics and logs. Azure Monitor Logs serve as a data source for Grafana, and the dashboards can be designed and configured to display important information and trends. The following steps were required to achieve this:



```
kubectl label namespace $INGRESS_NAMESPCAE cert-manager.io/disable-validation=true

# Include jetstack helm repo
helm repo add jetstack https://charts.jetstack.io
helm repo update

#Install the cert-manager
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.7.1/cert-manager.crds.yaml
helm install cert-manager jetstack/cert-manager --namespace $INGRESS_NAMESPCAE --version v1.7.1
```

Figure 3.7: Difference between HTTP and HTTPS

To start using the installed Grafana, we had to decode the generated password from base64 and connect on http://localhost:3000 to the admin account using that password. We then added a new data source, an Azure Monitor data source, provided the application ID, tenant ID and client password to successfully receive the logs from Azure Monitor, then created the Dashboard to visualize whatever we needed leveraging the collected data.

3.3.5 Alerting

Alerting is a critical component of monitoring, allowing timely detection of critical events or performance issues. Azure Monitor provides alerting capabilities based on specific conditions and thresholds. Alert rules are defined to trigger notifications via email, SMS, or other channels, ensuring that the right stakeholders are informed promptly. We leveraged this capability to set up rules and get notifications whenever our application encounters a problem. To do this, we created custom alert rules from the Alert section on the AKS cluster blade.

Conclusion

In this chapter, we discussed the security features we added to the implementation of our project, we saw how Azure Monitor can be utilized to implement a robust monitoring approach, create visual dashboards, set up alerting mechanisms, and collect logs for analysis.

Chapter 4

Testing the solution on an e-commerce application

Introduction

At this stage, we will go into the implementation of the solution we have developed. We will begin by discussing the application code we used in the testing and software environment and tools that were utilized throughout the implementation process. We will provide insights into the practical implementation of our solution and present the results obtained from testing the deployed web application on AKS.

4.1 Introducing the web application

The Ambassamart web application is an e-commerce platform consisting of three frontend and one backend, with each website dedicated to one of our user types: Admin, Ambassador, and Client.
Firstly, we have the Admin Application, a React-based app, which allows administrators (owners) to manage their products, ambassadors' accounts, and orders.
Secondly, we have the Ambassador Application, also a React-based app, where ambassadors can select products and generate a link to share with clients. This link is the URL that will redirect the client to the checkout application.
Finally, we have the Checkout Application, built using Next.js, which enables clients to view the products selected by the ambassador, choose the desired quantity for each item, and complete their order.
After the order completion, 90 per cent of the total amount will be the admin's revenue, while the remaining 10 per cent will be the ambassador's revenue.
Our backend server is NestJS-based, which is linked to our PostgreSQL database using package named TYPEORM, and we utilize Redis as a caching tool to deliver fast performance and reduce response time by storing data in memory and utilizing efficient data structures.

4.1.1 Use case Diagram

In this section, we will present the different diagrams we used to model our project. The Figure 4.1 below, showcases the Use Case Diagram to better explain how the user may interact with the tool.

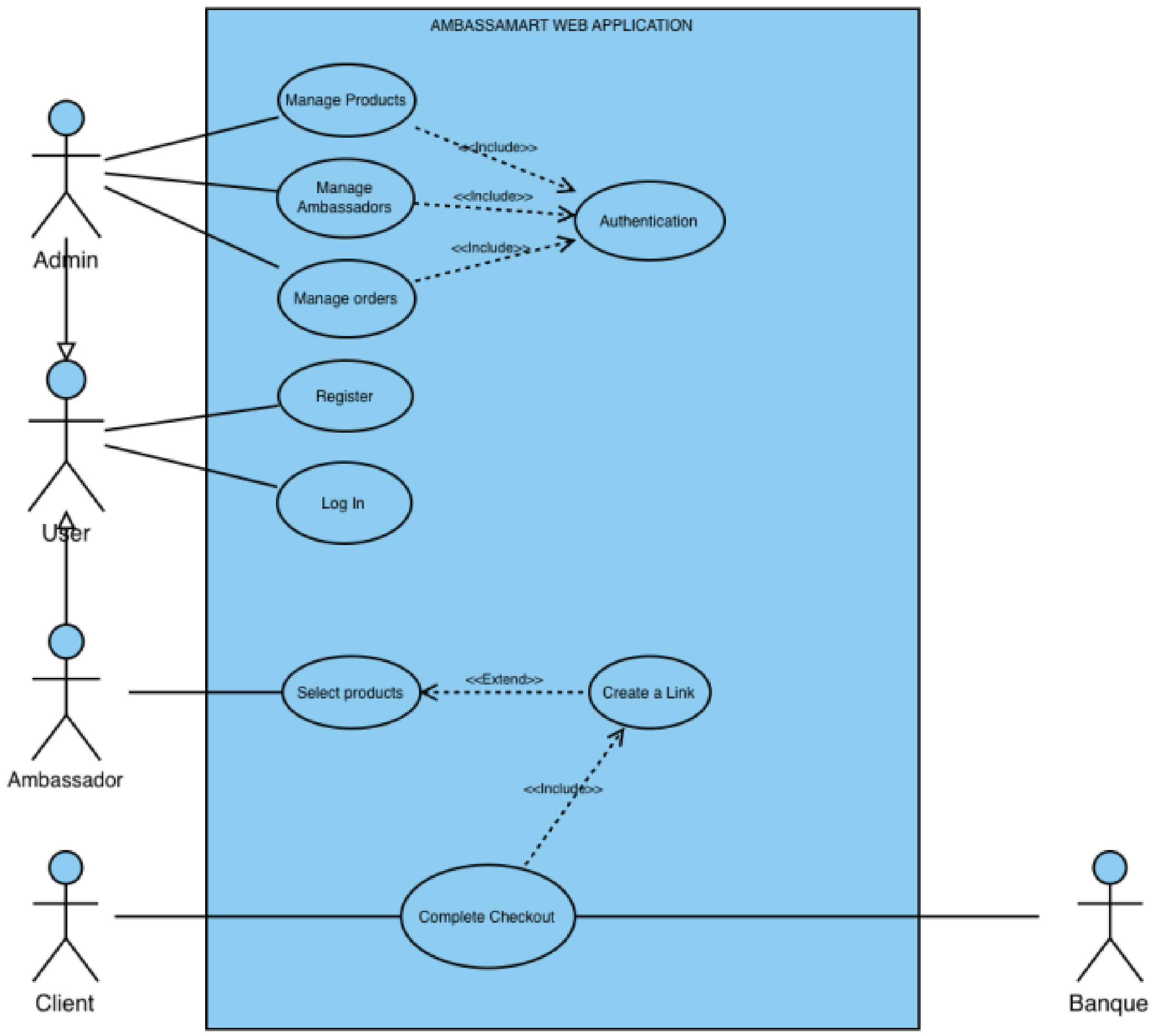


Figure 4.1: Use case Diagram

4.1.2 Sequence Diagram of the checkout action

This diagram , figure 4.2 represents how the checkout process is done in the system

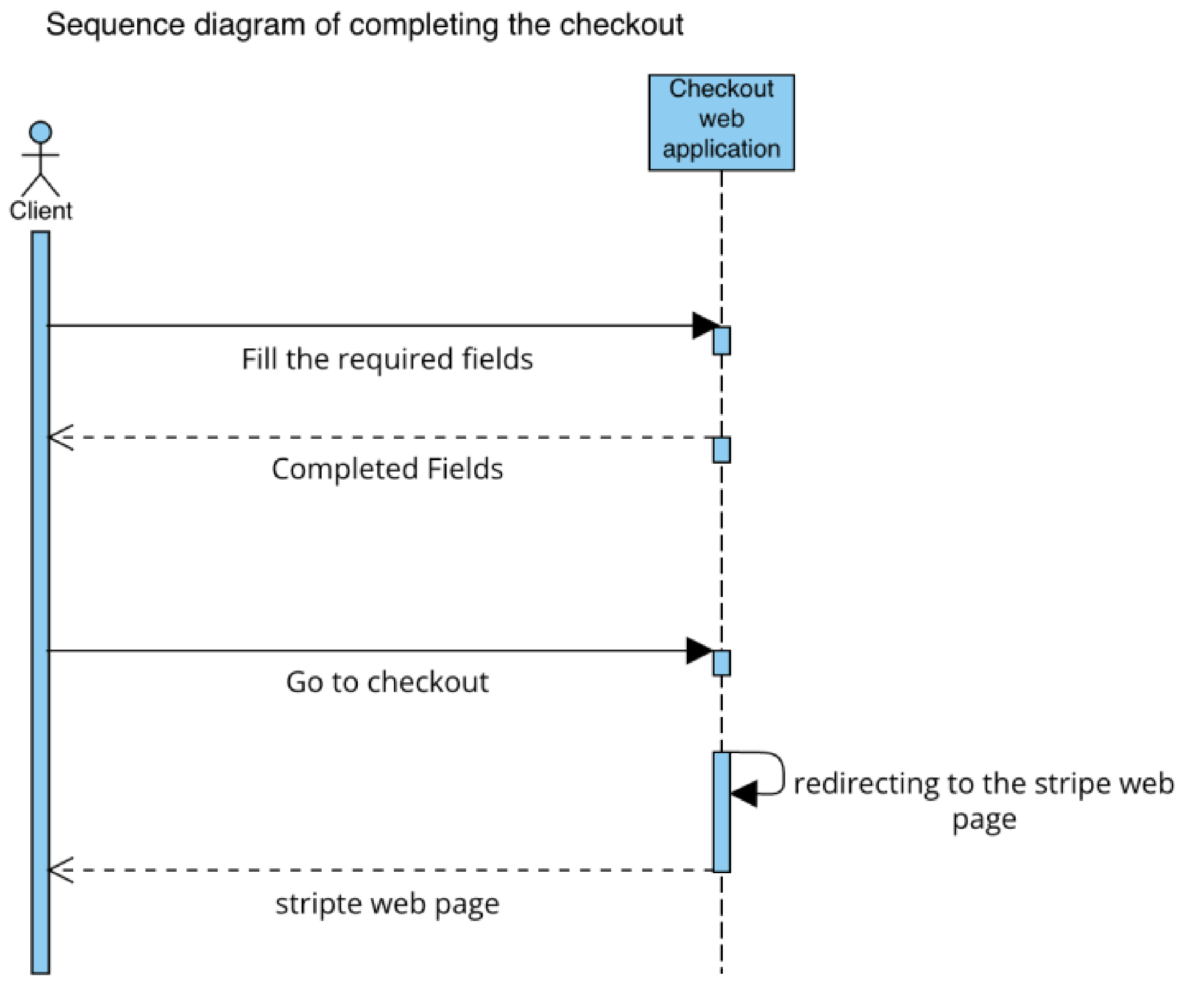


Figure 4.2: Sequence Diagram of the link generation

4.1.3 Sequence Diagram of the link generation

This diagram , figure 4.3 represents how the the links are generated when the selection of products is done.

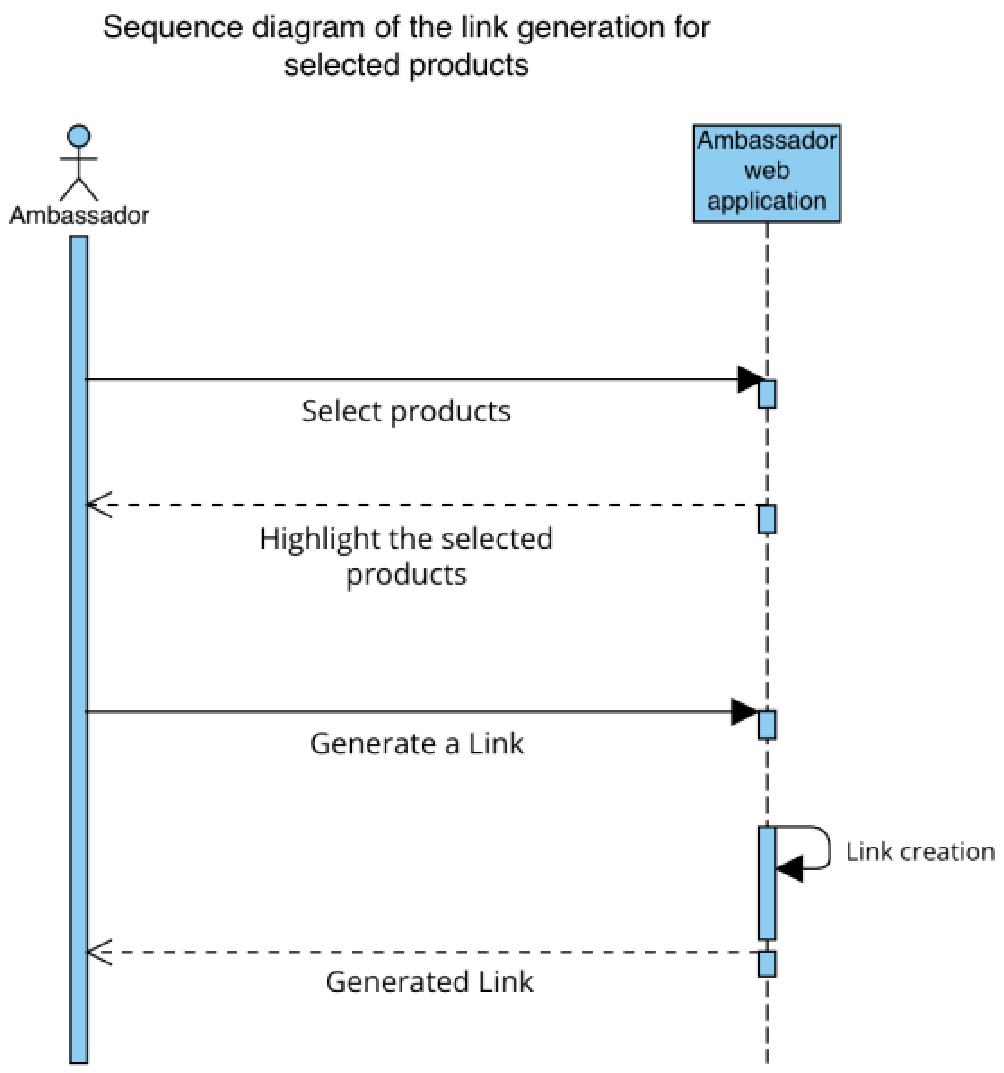

Sequence diagram of the link generation for selected products
Ambassador
Ambassador web application
Select products
Highlight the selected products
Generate a Link
Link creation
Generated Link

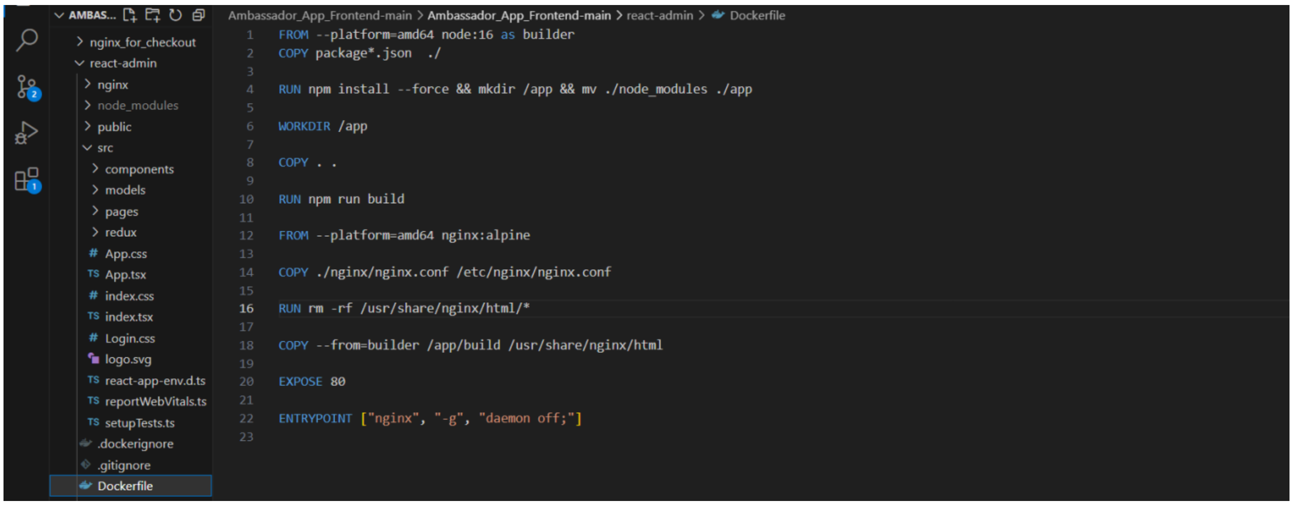
Figure 4.3: Sequence Diagram of the link generation

4.2 Implementation and Results Showcase

In this section, we will present the results that we got through the most important stages of our project's implementation.

4.2.1 Dockerizing the application

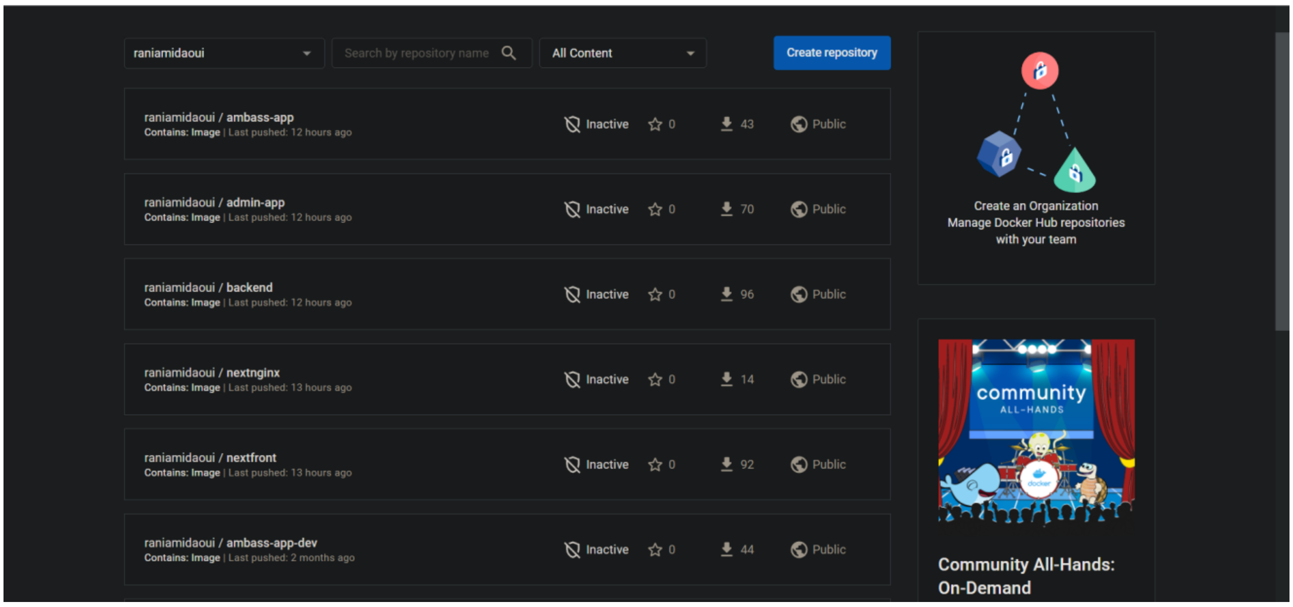
To deploy the application to Azure Kubernetes Cluster, we discussed how it should be containerized first. We chose to create a container for each frontend on its own, a container for the backend, a database container and a redis container. In the figures 4.4 , 4.5 we can see an example of the Docker file used to build the container image of the admin frontend.



```
FROM --platform=amd64 node:16 as builder
COPY package*.json  ./

RUN npm install --force && mkdir /app && mv ./node_modules ./app

WORKDIR /app

COPY . .

RUN npm run build

FROM --platform=amd64 nginx:alpine

COPY ./nginx/nginx.conf /etc/nginx/nginx.conf

RUN rm -rf /usr/share/nginx/html/*

COPY --from=builder /app/build /usr/share/nginx/html

EXPOSE 80

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Figure 4.4: Docker file

After creating all the container images, and we them pushed Docker Hub. These will be used in the first deployment of the application.

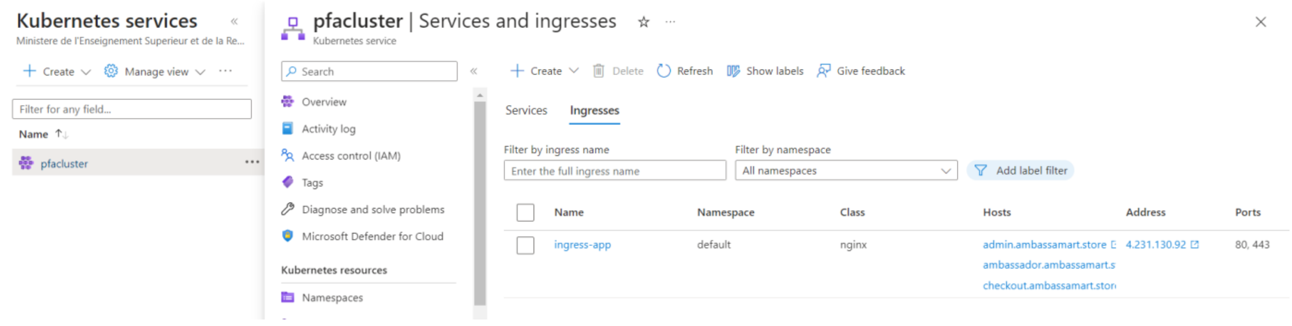


Repository	Status	Stars	Downloads	Visibility
raniamidaoui / ambass-app Contains: Image | Last pushed: 12 hours ago	Inactive	0	43	Public
raniamidaoui / admin-app Contains: Image | Last pushed: 12 hours ago	Inactive	0	70	Public
raniamidaoui / backend Contains: Image | Last pushed: 12 hours ago	Inactive	0	96	Public
raniamidaoui / nextnginx Contains: Image | Last pushed: 13 hours ago	Inactive	0	14	Public
raniamidaoui / nextfront Contains: Image | Last pushed: 13 hours ago	Inactive	0	92	Public
raniamidaoui / ambass-app-dev Contains: Image | Last pushed: 2 months ago	Inactive	0	44	Public

Figure 4.5: Dockerhub repository of the application

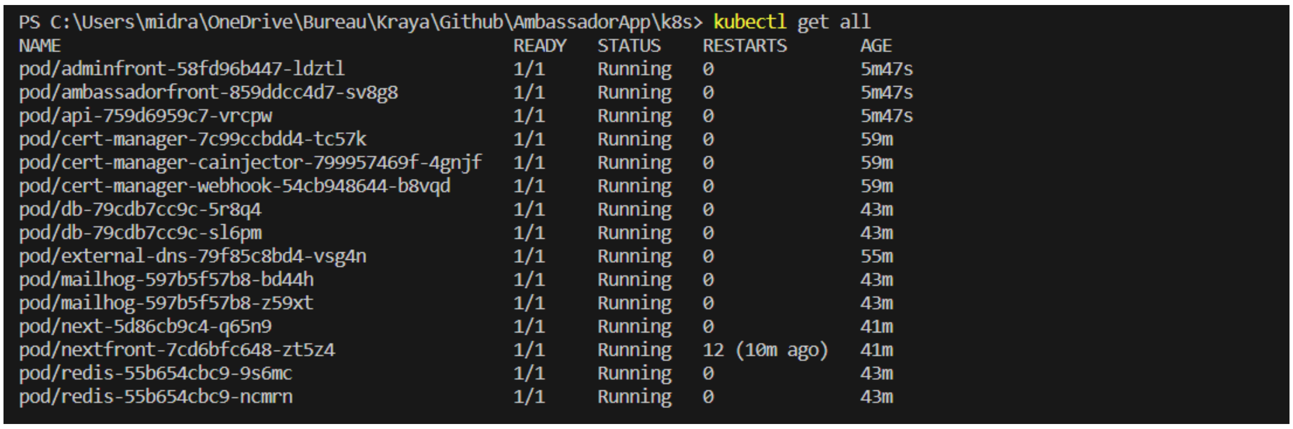
4.2.2 Creating a Kubernetes Cluster on Azure Kubernetes Services (AKS)

In the figures 4.6 , 4.7 and 4.8 We show the state of the cluster after we created the Kubernetes cluster and set auto scating to automatic. We didn't forget the ingress controller too.



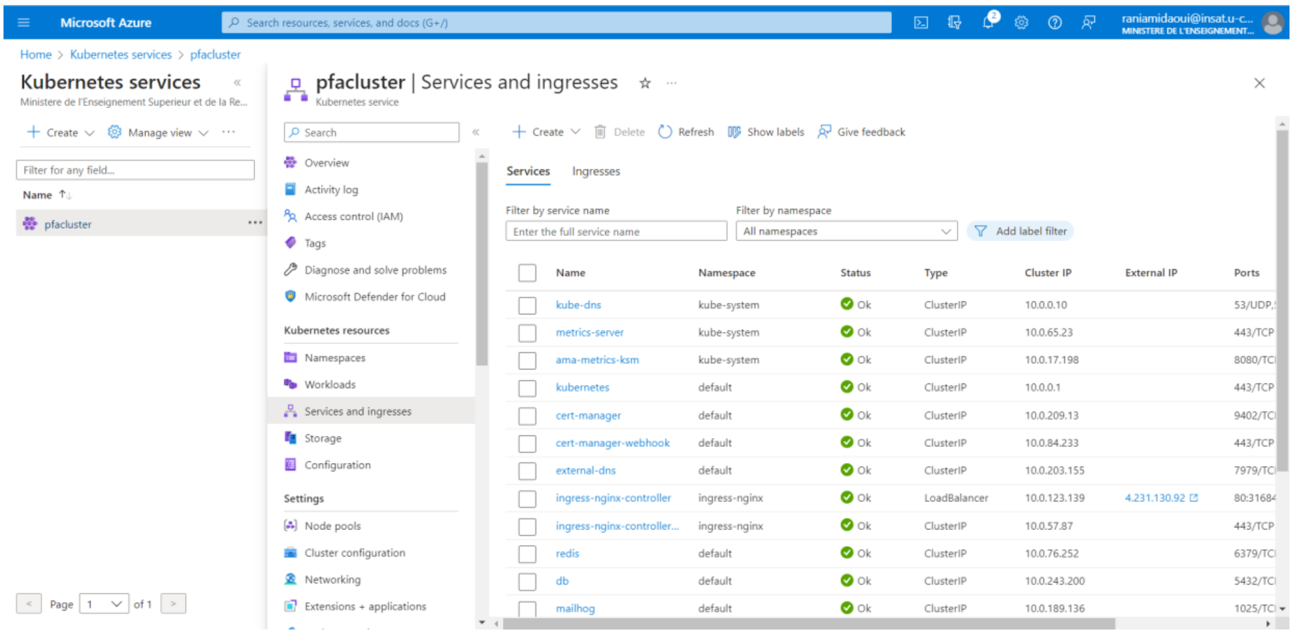
	Name	Namespace	Class	Hosts	Address	Ports
☐	ingress-app	default	nginx	admin.ambassamart.store ambassador.ambassamart.s checkout.ambassamart.stor	4.231.130.92	80, 443

Figure 4.6: Nginx ingress



```
PS C:\Users\midra\OneDrive\Bureau\Kraya\Github\AmbassadorApp\k8s> kubectl get all
NAME                                          READY   STATUS    RESTARTS      AGE
pod/adminfront-58fd96b447-ldztl               1/1     Running   0             5m47s
pod/ambassadorfront-859ddcc4d7-sv8g8          1/1     Running   0             5m47s
pod/api-759d6959c7-vrcpw                      1/1     Running   0             5m47s
pod/cert-manager-7c99ccbdd4-tc57k             1/1     Running   0             59m
pod/cert-manager-cainjector-799957469f-4gnjf  1/1     Running   0             59m
pod/cert-manager-webhook-54cb948644-b8vqd     1/1     Running   0             59m
pod/db-79cdb7cc9c-5r8q4                       1/1     Running   0             43m
pod/db-79cdb7cc9c-sl6pm                       1/1     Running   0             43m
pod/external-dns-79f85c8bd4-vsg4n             1/1     Running   0             55m
pod/mailhog-597b5f57b8-bd44h                  1/1     Running   0             43m
pod/mailhog-597b5f57b8-z59xt                  1/1     Running   0             43m
pod/next-5d86cb9c4-q65n9                      1/1     Running   0             41m
pod/nextfront-7cd6bfc648-zt5z4                1/1     Running   12 (10m ago)  41m
pod/redis-55b654cbc9-9s6mc                    1/1     Running   0             43m
pod/redis-55b654cbc9-ncmrn                    1/1     Running   0             43m
```

Figure 4.7: The cluster pods



	Name	Namespace	Status	Type	Cluster IP	External IP	Ports
☐	kube-dns	kube-system	✓ Ok	ClusterIP	10.0.0.10		53/UDP,
☐	metrics-server	kube-system	✓ Ok	ClusterIP	10.0.65.23		443/TCP
☐	ama-metrics-ksm	kube-system	✓ Ok	ClusterIP	10.0.17.198		8080/TC
☐	kubernetes	default	✓ Ok	ClusterIP	10.0.0.1		443/TCP
☐	cert-manager	default	✓ Ok	ClusterIP	10.0.209.13		9402/TC
☐	cert-manager-webhook	default	✓ Ok	ClusterIP	10.0.84.233		443/TCP
☐	external-dns	default	✓ Ok	ClusterIP	10.0.203.155		7979/TC
☐	ingress-nginx-controller	ingress-nginx	✓ Ok	LoadBalancer	10.0.123.139	4.231.130.92	80:3168
☐	ingress-nginx-controller...	ingress-nginx	✓ Ok	ClusterIP	10.0.57.87		443/TCP
☐	redis	default	✓ Ok	ClusterIP	10.0.76.252		6379/TC
☐	db	default	✓ Ok	ClusterIP	10.0.243.200		5432/TC
☐	mailhog	default	✓ Ok	ClusterIP	10.0.189.136		1025/TC

Figure 4.8: The cluster services and ingress view in the azure portal

4.2.3 The web application

In the figures 4.9 , 4.10 we are show the results of our deployment, the web application used in our demo is up and running.

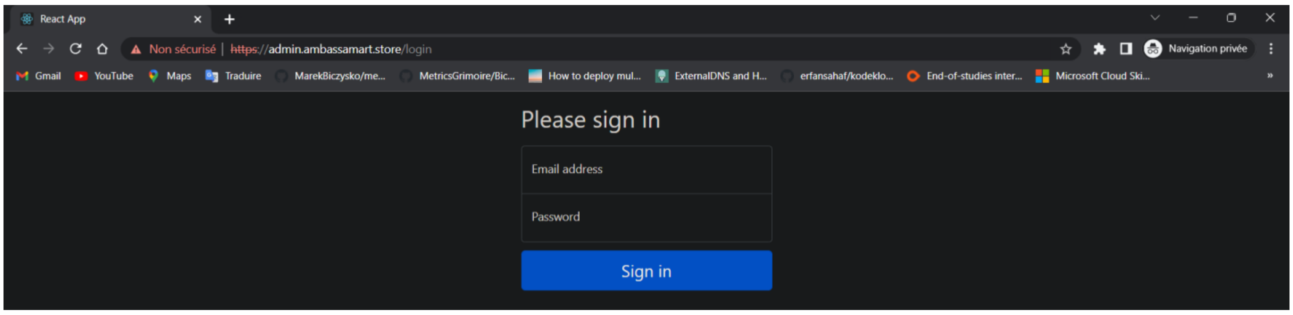


Figure 4.9: Admin login frontend

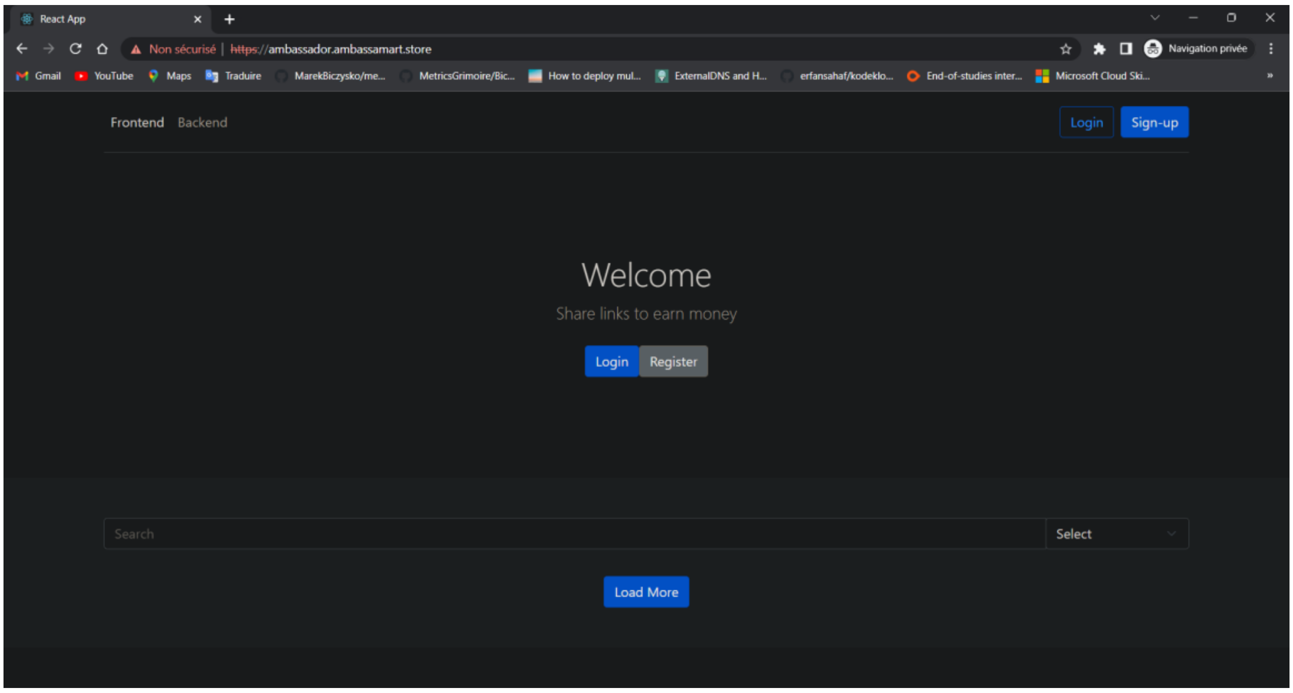
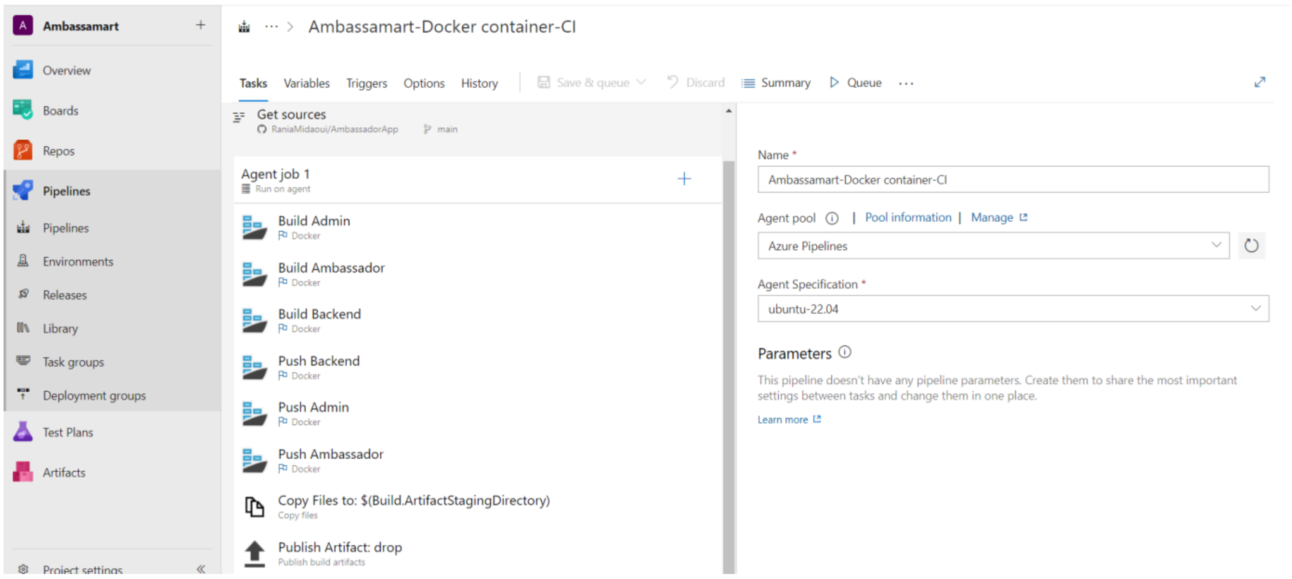


Figure 4.10: Ambassador frontend

4.2.4 CI/CD Pipeline using Azure DevOps

In the figures 4.11 , 4.11 and 4.13, we are showing the pipelines related to our code and Kubernetes cluster. The first figure shows the tasks used in the CI pipeline, the Build* tasks build the container images upon trigger, the Push* tasks push the container images to Docker Hub, the Copy Files task copies the Kubernetes YAML files to a build artifact, to be used in the CD pipeline in the next stage. The Release pipeline (CD) will then deploy the new images.



Ambassamart-Docker container-CI

Get sources
RaniaMidaoui/AmbassadorApp main

Agent job 1
Run on agent

Build Admin
Docker

Build Ambassador
Docker

Build Backend
Docker

Push Backend
Docker

Push Admin
Docker

Push Ambassador
Docker

Copy Files to: $(Build.ArtifactStagingDirectory)
Copy files

Publish Artifact: drop
Publish build artifacts

Name *
Ambassamart-Docker container-CI

Agent pool | Pool information | Manage
Azure Pipelines

Agent Specification *
ubuntu-22.04

Parameters
This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.
Learn more

Figure 4.11: Agent jobs

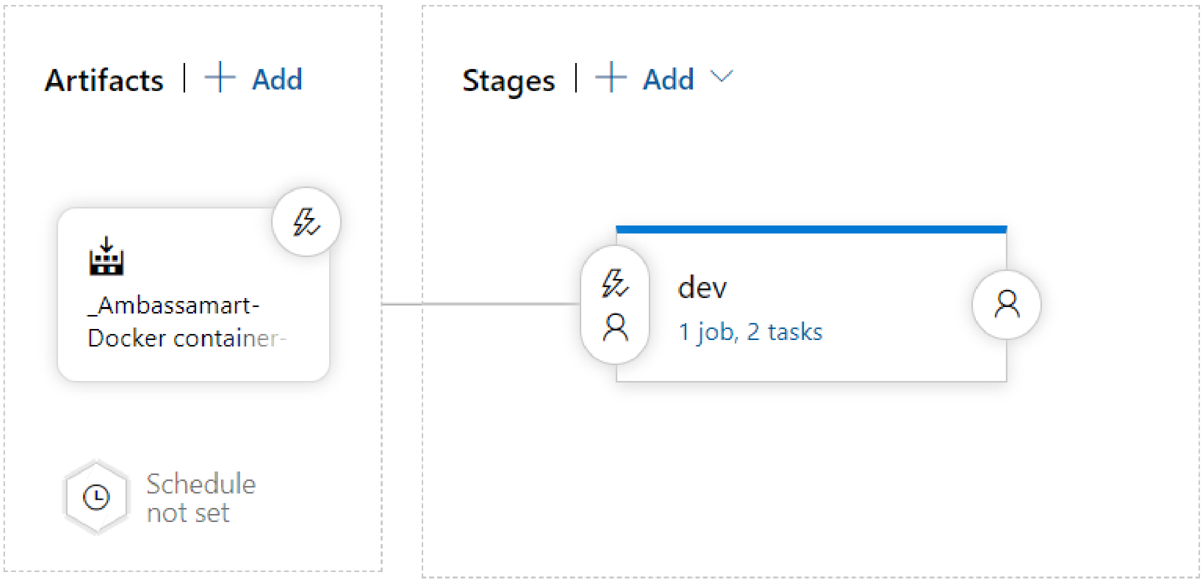
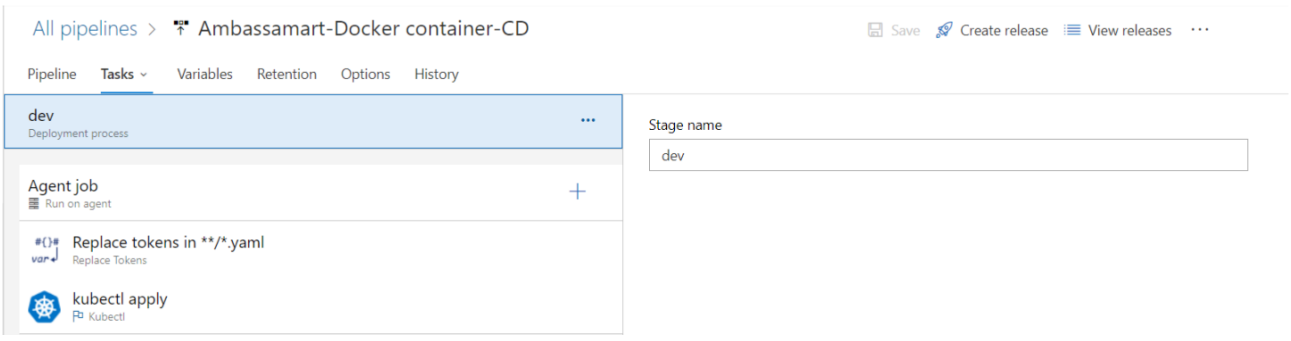


Figure 4.12: CI/CD linking diagram



Ambassamart-Docker container-CD

dev
Deployment process

Agent job
Run on agent

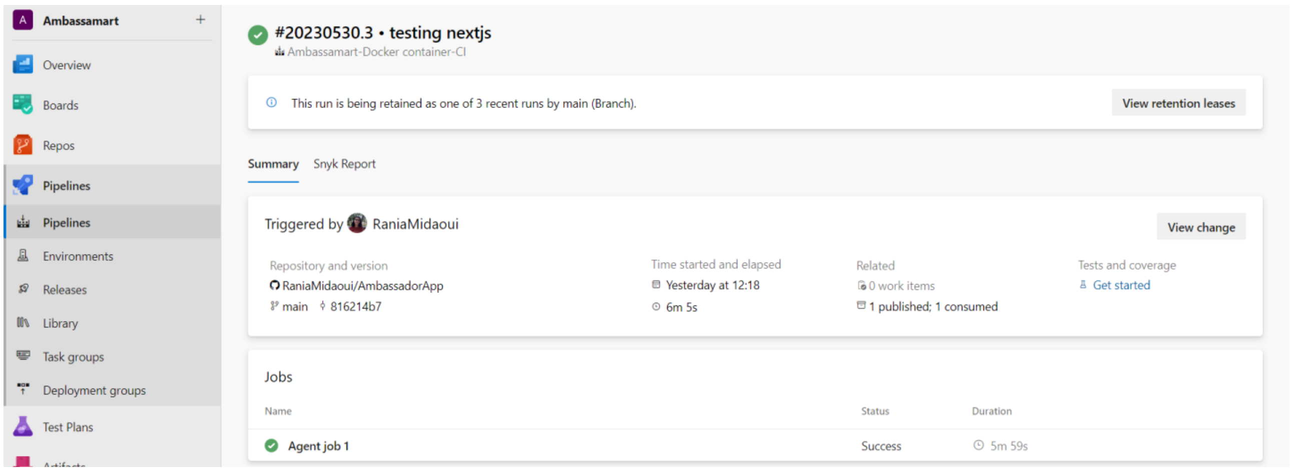
Replace tokens in **/*.yaml
Replace Tokens

kubectl apply
Kubectl

Stage name
dev

Figure 4.13: Continuous Delivery agent job

In the figures 4.14, we triggered our pipelines with a push to Github, we can see how the CI/CD pipeline finished all its tasks successfully.



#20230530.3 • testing nextjs
Ambassamart-Docker container-CI

This run is being retained as one of 3 recent runs by main (Branch).

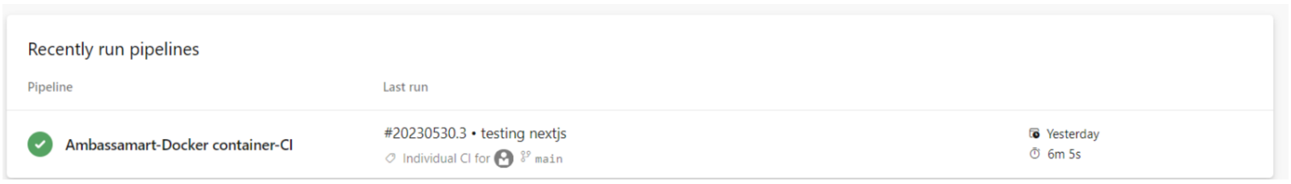
Triggered by RaniaMidaoui

Repository and version: RaniaMidaoui/AmbassadorApp, main, 816214b7
Time started and elapsed: Yesterday at 12:18, 6m 5s
Related: 0 work items, 1 published; 1 consumed
Tests and coverage: Get started

Jobs

Name	Status	Duration
Agent job 1	Success	5m 59s

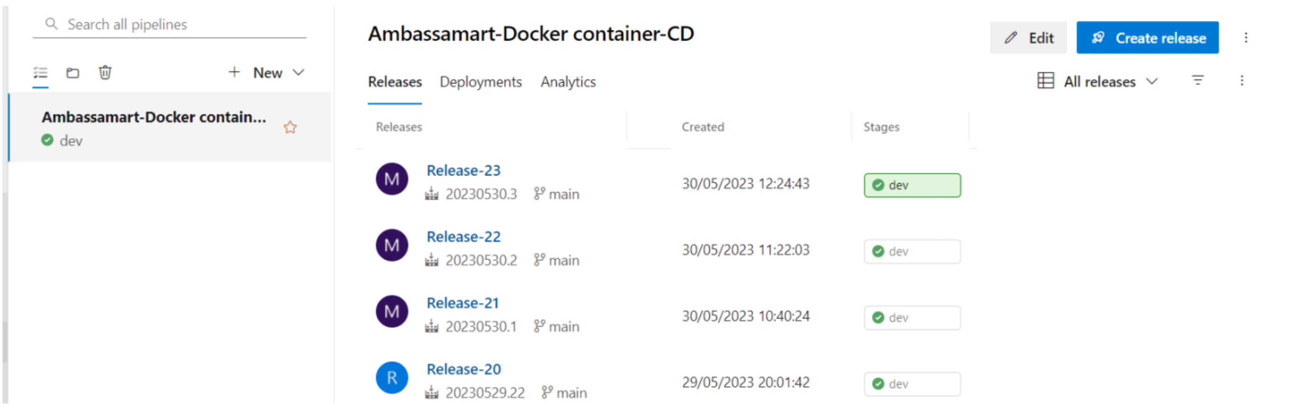
Figure 4.14: Application Continuous integration job agent triggering



Recently run pipelines

Pipeline	Last run	
Ambassamart-Docker container-CI	#20230530.3 • testing nextjs Individual CI for main	Yesterday 6m 5s

Figure 4.15: Continuous integration triggering



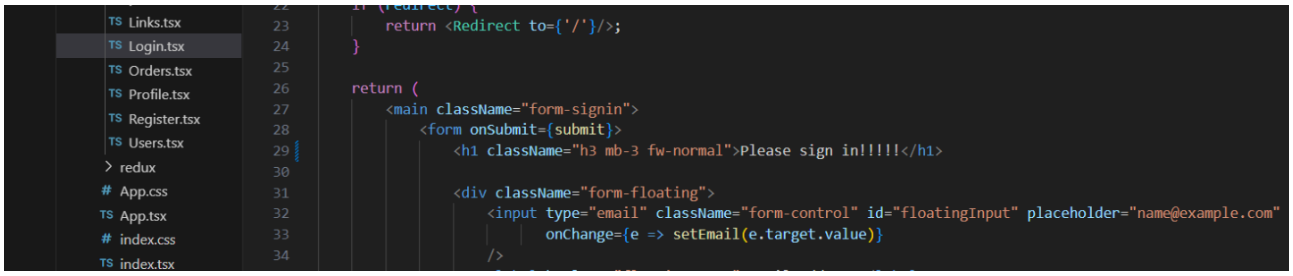
Ambassamart-Docker container-CD

Releases	Created	Stages
Release-23 20230530.3 main	30/05/2023 12:24:43	dev
Release-22 20230530.2 main	30/05/2023 11:22:03	dev
Release-21 20230530.1 main	30/05/2023 10:40:24	dev
Release-20 20230529.22 main	29/05/2023 20:01:42	dev

Figure 4.16: Continuous Delivery Releases

4.2.5 Pipelines trigger results shown on the website

We transformed the message shown in the sign in to "Please sing in!!!!!" instead of "Please sing in", and the pipeline triggered and changed the message.



```
        return <Redirect to={'/'}/>;
    }

    return (
        <main className="form-signin">
            <form onSubmit={submit}>
                <h1 className="h3 mb-3 fw-normal">Please sign in!!!!!</h1>

                <div className="form-floating">
                    <input type="email" className="form-control" id="floatingInput" placeholder="name@example.com"
                        onChange={e => setEmail(e.target.value)}
                    />
```

Figure 4.17: Code modification

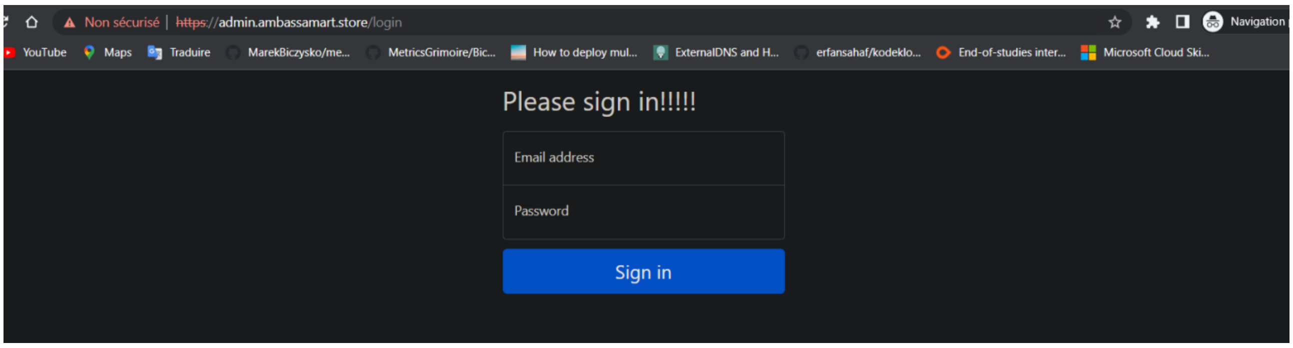
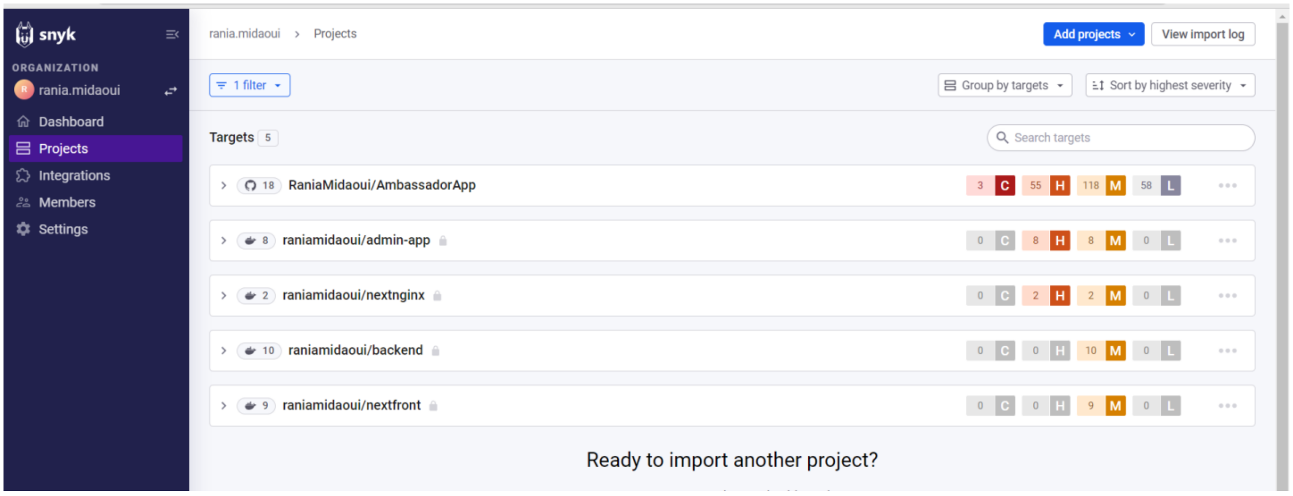


Figure 4.18: Showcasing the modification in the application

4.2.6 Securing Pipeline

In the figures 4.19 we see an analysis done with Snyk on our application code and container images.



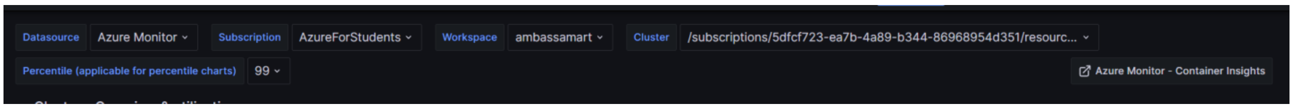
Targets 5

		C		H		M		L
18	RaniaMidaoui/AmbassadorApp	3	C	55	H	118	M	58	L
8	raniamidaoui/admin-app	0	C	8	H	8	M	0	L
2	raniamidaoui/nextnginx	0	C	2	H	2	M	0	L
10	raniamidaoui/backend	0	C	0	H	10	M	0	L
9	raniamidaoui/nextfront	0	C	0	H	9	M	0	L

Figure 4.19: Synk front interface

4.2.7 Monitoring

In the figures 4.20 , 4.21 and 4.22 we are showing a practice to monitor our Kubernetes cluster using Grafana Dashboards.



Datasource: Azure Monitor
Subscription: AzureForStudents
Workspace: ambassamart
Cluster: /subscriptions/5dfcf723-ea7b-4a89-b344-86968954d351/resourc...
Percentile (applicable for percentile charts): 99

Figure 4.20: Monitoring with Grafana

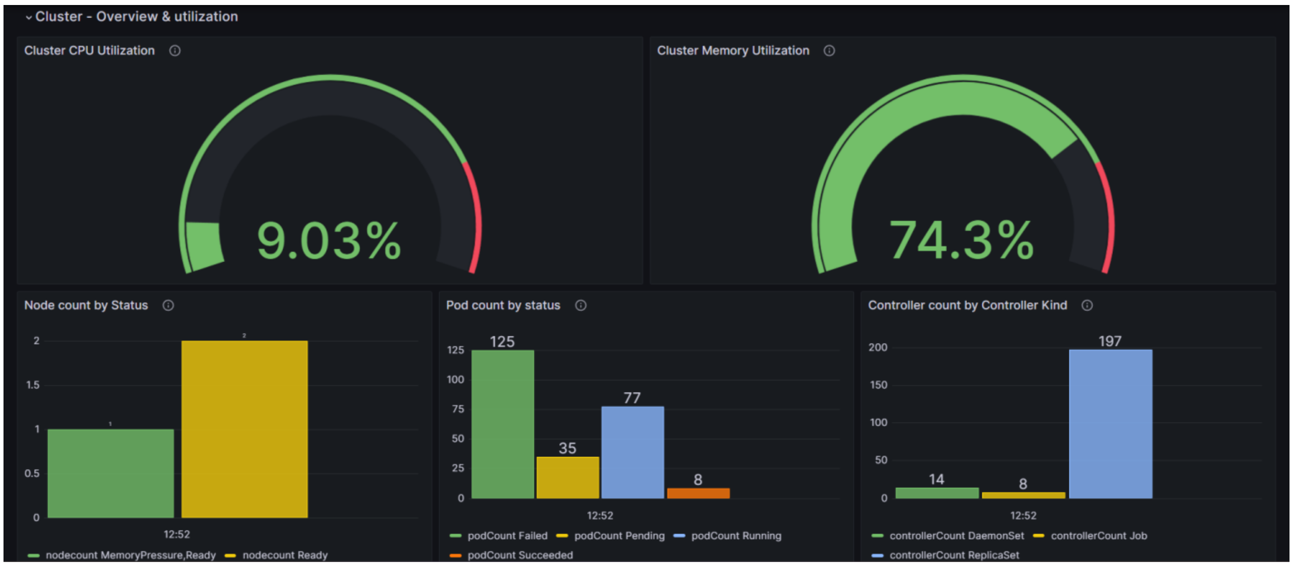


Figure 4.21: Monitoring with Grafana

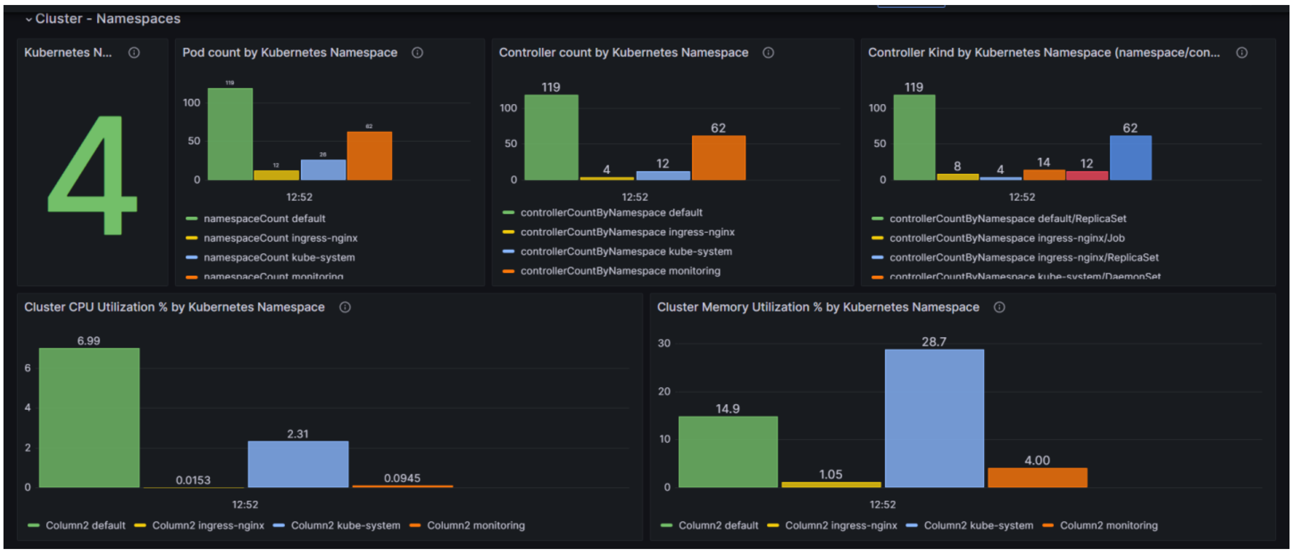
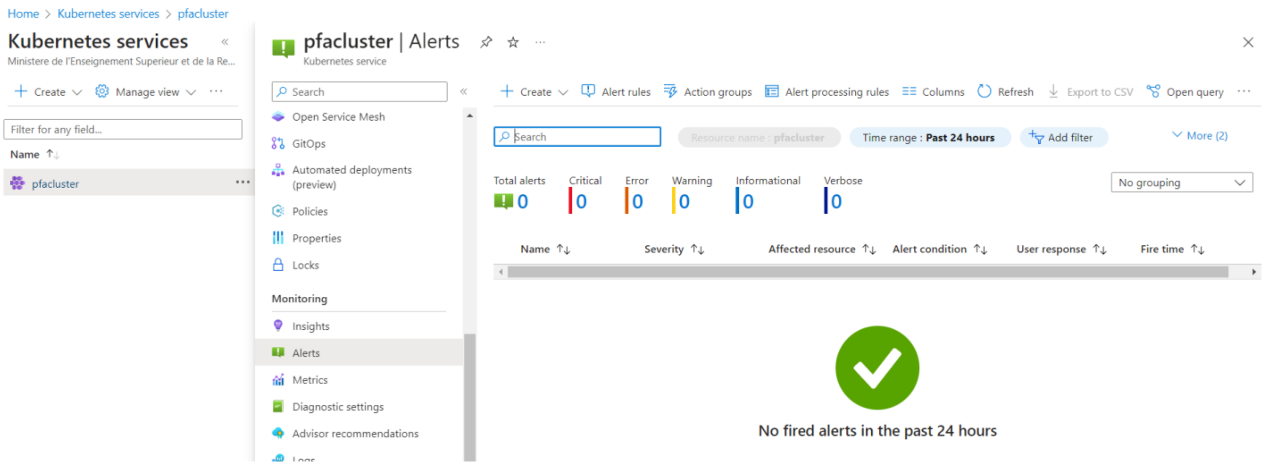

Cluster - Namespaces
Kubernetes N...
4
Pod count by Kubernetes Namespace
119
12
26
62
12:52
namespaceCount default
namespaceCount ingress-nginx
namespaceCount kube-system
namespaceCount monitoring
Controller count by Kubernetes Namespace
119
4
12
62
12:52
controllerCountByNamespace default
controllerCountByNamespace ingress-nginx
controllerCountByNamespace kube-system
controllerCountByNamespace monitoring
Controller Kind by Kubernetes Namespace (namespace/con...
119
8
4
14
12
62
12:52
controllerCountByNamespace default/ReplicaSet
controllerCountByNamespace ingress-nginx/Job
controllerCountByNamespace ingress-nginx/ReplicaSet
controllerCountByNamespace kube-system/DaemonSet
Cluster CPU Utilization % by Kubernetes Namespace
6.99
0.0153
2.31
0.0945
12:52
Column2 default
Column2 ingress-nginx
Column2 kube-system
Column2 monitoring
Cluster Memory Utilization % by Kubernetes Namespace
14.9
1.05
28.7
4.00
12:52
Column2 default
Column2 ingress-nginx
Column2 kube-system
Column2 monitoring

Figure 4.22: Monitoring with Grafana

4.2.8 Alerts

In this figure 4.23 we can see that the alerts have been successfully set but no alert have triggered



Total alerts 0 | Critical 0 | Error 0 | Warning 0 | Informational 0 | Verbose 0

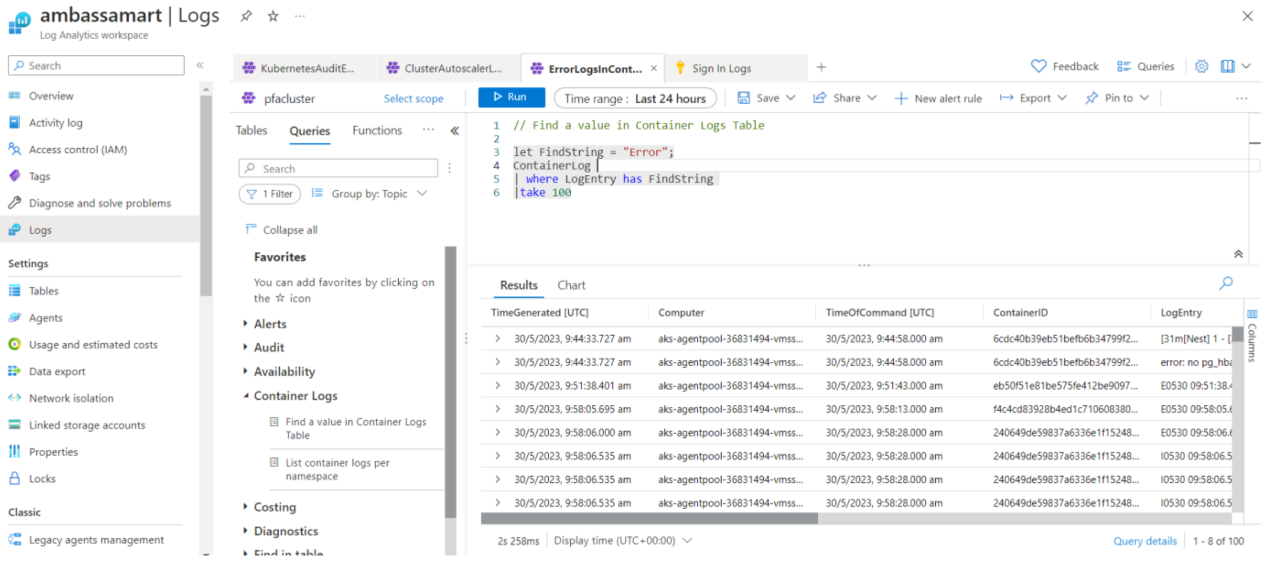
Time range : Past 24 hours

No fired alerts in the past 24 hours

Figure 4.23: Showcasing the alerts in the cluster

4.2.9 Log Analytics

In the figures 4.24 , 4.25 we are showing some logs sent by our cluster.



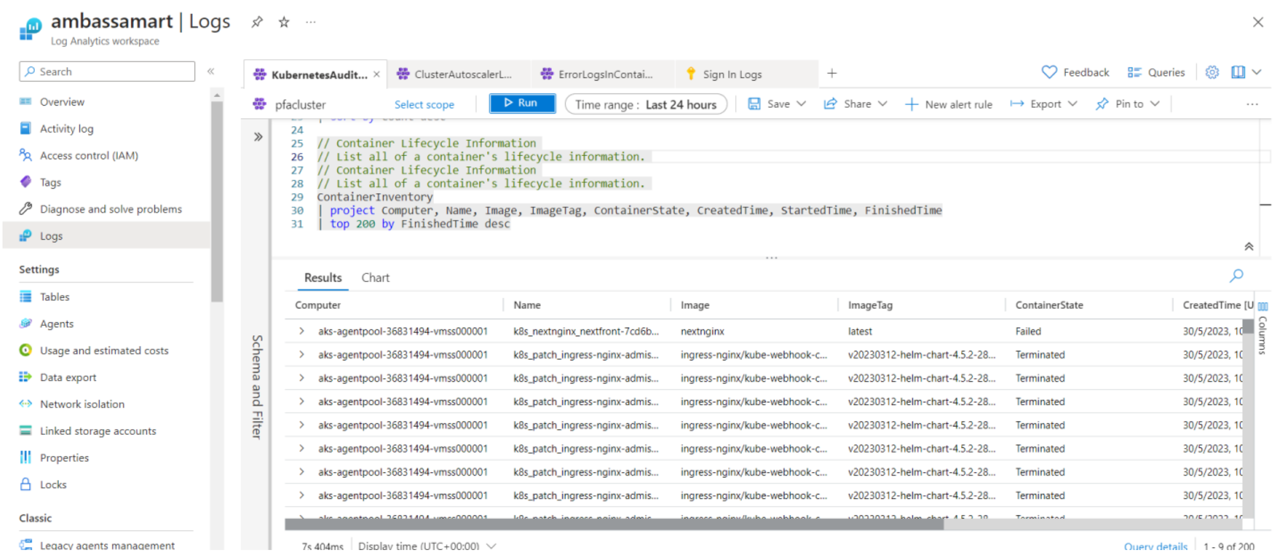
Time range : Last 24 hours

```
// Find a value in Container Logs Table

let FindString = "Error";
ContainerLog
| where LogEntry has FindString
|take 100
```

TimeGenerated [UTC]	Computer	TimeOfCommand [UTC]	ContainerID	LogEntry
30/5/2023, 9:44:33.727 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:44:58.000 am	6cdc40b39eb51befb6b34799f2...	[31m[Nest] 1 - [
30/5/2023, 9:44:33.727 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:44:58.000 am	6cdc40b39eb51befb6b34799f2...	error: no pg_hba
30/5/2023, 9:51:38.401 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:51:43.000 am	eb50f51e81be575fe412be9097...	E0530 09:51:38.4
30/5/2023, 9:58:05.695 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:58:13.000 am	f4c4cd83928b4ed1c710608380...	E0530 09:58:05.6
30/5/2023, 9:58:06.000 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:58:28.000 am	240649de59837a6336e1f15248...	E0530 09:58:06.6
30/5/2023, 9:58:06.535 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:58:28.000 am	240649de59837a6336e1f15248...	I0530 09:58:06.5
30/5/2023, 9:58:06.535 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:58:28.000 am	240649de59837a6336e1f15248...	I0530 09:58:06.5
30/5/2023, 9:58:06.535 am	aks-agentpool-36831494-vmss...	30/5/2023, 9:58:28.000 am	240649de59837a6336e1f15248...	I0530 09:58:06.5

2s 258ms | Display time (UTC+00:00) | Query details | 1 - 8 of 100

Figure 4.24: Kubernetes cluster audit



```
// Container Lifecycle Information
// List all of a container's lifecycle information.
// Container Lifecycle Information
// List all of a container's lifecycle information.
ContainerInventory
| project Computer, Name, Image, ImageTag, ContainerState, CreatedTime, StartedTime, FinishedTime
| top 200 by FinishedTime desc
```

Computer	Name	Image	ImageTag	ContainerState	CreatedTime [U
aks-agentpool-36831494-vmss000001	k8s_nextnginx_nextfront-7cd6b...	nextnginx	latest	Failed	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10
aks-agentpool-36831494-vmss000001	k8s_patch_ingress-nginx-admis...	ingress-nginx/kube-webhook-c...	v20230312-helm-chart-4.5.2-28...	Terminated	30/5/2023, 10

Figure 4.25: Error container logs

Conclusion

This chapter covered the realization of the project. It started by presenting the software environment then it moved to the technologies used for the development of the project explaining each choice by reference to the requirements of the application. finally it presented the different modules developed within the scope of the project.

Conclusion and perspectives

Throughout this project, we successfully deployed a multi-container web application to Azure Kubernetes Service (AKS) using Azure DevOps. We started by understanding the deployment and automation process and the benefits it offers in terms of high availability, scalability, and time saving.

Next, we dived into Azure Kubernetes Services (AKS) and explored how it provides a managed Kubernetes environment for seamless deployment and orchestration of our containers. We leveraged AKS to create and manage our Kubernetes cluster, ensuring high availability and efficient resource utilization, after developing and containerizing an e-commerce application.

To make our development process smoother, we set up a CI/CD pipeline using Azure DevOp. This pipeline helped us automate tasks like building, testing, and deploying our application, ensuring that changes were integrated and delivered continuously.

We also implemented monitoring and logging mechanisms using Azure Monitor and Azure Log Analytics Workspace to gain deeper insights into the performance and health of our application.

In the future, we can explore using more Azure services to make our solution better. For example, we can use Azure Key Vault to keep our application secrets safe and secure. We can also think about using Azure Logic Apps to automate our work and make different systems work together smoothly.

Bibliography

[1] *Docker*. [visited on 15/02/2023]. URL: `https://docs.docker.com/get-started/overview/`.

[2] *Kubernetes*. [visited on 10/03/2023]. URL: `https://kubernetes.io/docs/concepts/overview/`.

[3] *Snyk*. [visited on 26/05/2023]. URL: `https://snyk.io/`.

[4] *Certification Manager Amazon*. [visited on 6/05/2023]. URL: `https://aws.amazon.com/fr/what-is/ssl-certificate/`.

[5] *Certification manager github*. [visited on 6/05/2023]. URL: `https://github.com/cert-manager/cert-manager`.

[6] *Monitoring*. [visited on 15/05/2023]. URL: `https://www.loggly.com/use-cases/proactive-monitoring-definition-and-best-practices/`.

[7] *Monitoring Approach*. [visited on 24/05/2023]. URL: `https://learn.microsoft.com/en-us/azure/aks/monitor-aks`.

[8] *Azure Monitor*. [visited on 28/04/2023]. URL: `https://learn.microsoft.com/en-us/azure/azure-monitor/overview`.

[9] *Azure log analytics workspace*. [visited on 15/05/2023]. URL: `https://learn.microsoft.com/en-us/azure/azure-monitor/logs/log-analytics-workspace-overview`.

[3] [4] [5]