Capstone Final Report

Comparative Study of Cloud Native Application Development & Deployment Model

Introduction

Cloud-native applications transform software development by harnessing the scalability, flexibility, and efficiency of cloud platforms. Using containers, microservices, and orchestration tools like Kubernetes, they enable dynamic scaling and adaptability [1][3]. Integrating AI further enhances their functionality, enabling real-time data analysis, automation, and actionable insights.

Cloud-native adoption is transforming industries such as healthcare, finance, e-commerce, and logistics by enabling secure handling of high-demand workloads. However, organizations encounter challenges including seamless cloud integration, robust data security, and effective workflow automation. This project demonstrates cloud-native application development and deployment through a spam email detection application hosted on Azure Kubernetes Service (AKS), with automated updates powered by Azure Functions [2]. This use case underscores the dynamic scalability and adaptability essential for managing evolving datasets in real-world applications.

Challenges

- Lack of seamless integration with cloud-native workflows.
- Limited automation and scalability.
- Inadequate handling of dynamic dataset updates.

Project Description

This project explores the development and deployment of cloud-native applications with a focus on integrating Artificial Intelligence (AI) using **Azure's ecosystem**. The primary objective is to develop an efficient workflow for building, deploying, and automating ML models in Azure. The study revolves around containerization and orchestration using **Docker**, **Azure Kubernetes Service (AKS)**, and automation with **Azure Functions**.

The project is divided into two phases:

1. **Initial Deployment Using Kubernetes:**
    a. Implemented containerized deployment of a spam email detection model in Azure Kubernetes Service (AKS).
    b. Designed workflows for container orchestration and scaling using AKS.

2. **Automated Workflow Using Azure Functions:**
    a. Developed an automated data pipeline triggered by changes in the dataset stored in Azure Blob Storage.
    b. Automated retraining, rebuilding, and redeployment of the ML model using Azure Functions.

Phase 1: Initial Deployment Using Kubernetes

I. Initial Project Architecture

Components:

1. **Docker:** Package the ML model.
2. **Azure Container Registry (ACR):** Store and manage Docker images.
3. **Azure Kubernetes Service (AKS):** Deploy the containerized application.
4. **Nginx:** Expose the service externally via LoadBalancer.

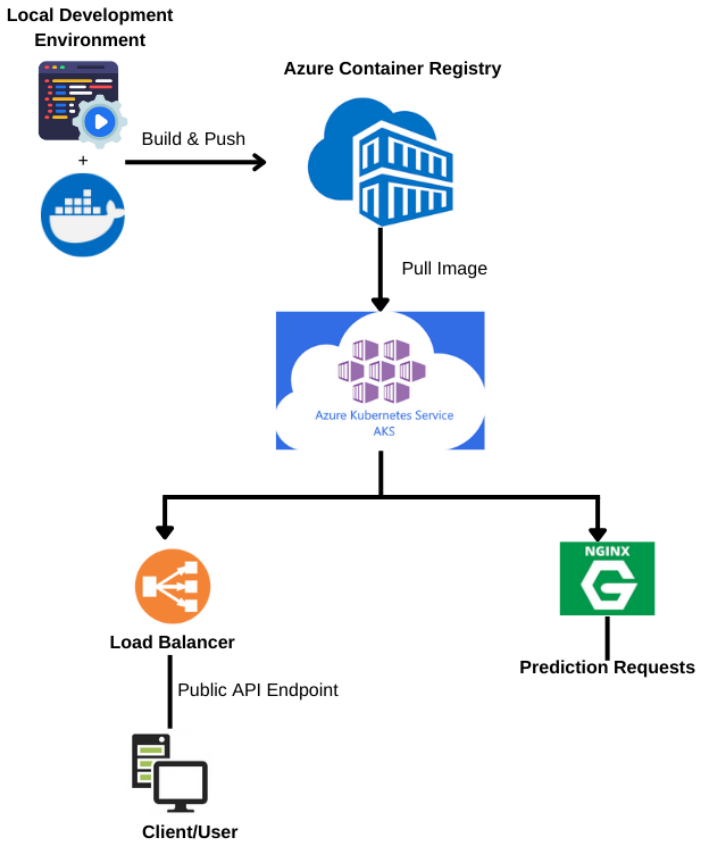

Fig 1: Kubernetes based Deployment

II. Development & Deployment Steps

1. **Selection of Model:**
    - Chose a **Spam Email Detection Model** using a pre-existing dataset. This model classifies emails as "spam" or "ham" using machine learning.
2. **Containerization of the ML Model:**
    - Created a container image of the spam detection model using Docker, ensuring portability and easy deployment across cloud platforms.
    - Configured Dockerfile for the model.

```
# Use official Python image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the application code
COPY . /app

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose port 5001 for Flask
EXPOSE 5001

# Set environment variable for Flask app
ENV FLASK_APP=app.py

# Run the application
CMD ["flask", "run", "--host=0.0.0.0", "--port=5001"]
```

    - **Docker Build and Push Commands**

```
docker build -t flask-spam-detector .
docker tag flask-spam-detector
capstonecncf.azurecr.io/flask-spam-detector:v1
az acr login --name capstonecncf
docker push capstonecncf.azurecr.io/flask-spam-detector:v1
```

3. **Setup of Azure Container Registry (ACR):**
    - Created a **container registry** named **capstonecncf** in Azure to host the Docker images.
    - Configured the ACR to store and manage the container images for seamless integration with Azure Kubernetes Service (AKS) [1][3].

4. **Deployment of Kubernetes Cluster (AKS):**
    - Created an **AKS cluster** named **CNCF** in Azure with one master node and two nodes (an agent pool and a user pool) [1].

- Configured the deployment of the model to AKS, using Kubernetes deployment and service manifests [3].

Kubernetes Deployment and Service YAML files:

Deployment (deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-spam-detector
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-spam-detector
  template:
    metadata:
      labels:
        app: flask-spam-detector
    spec:
      containers:
      - name: flask-spam-detector
        image: capstonecncf.azurecr.io/flask-spam-detector:v1
        ports:
        - containerPort: 5001
```

Service (service.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: flask-spam-detector
spec:
  selector:
    app: flask-spam-detector
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5001
  type: LoadBalancer
```

Commands to Apply Kubernetes Configuration:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
kubectl get svc # To retrieve external IP of the service
```

5. **Hosting and Testing the Application:**
   - The Spam Detection application is successfully deployed at http://172.210.63.81, hosted via Nginx and running in the AKS cluster.
   - Verified the application by sending test requests to the API endpoint using the following command:

```
curl -X POST -H "Content-Type: application/json" -d '{"message": "Free entry in 2 a weekly competition to win FA Cup tickets."}' http://20.242.210.78:80/predict
```

Phase 2: Automating Using Azure Functions

I. Automated Workflow Architecture

Components:

1. **Azure Blob Storage**:
   a. **Dataset**: Stores the dataset (spam.csv) in the dataset container.
   b. **Model**: Stores the trained model (model.pkl) in the models container.
2. **Azure Functions**:
   a. **PredictFunction**: Handles HTTP requests to predict if a message is spam or not using the trained model.
   b. **RetrainFunction**: Automates the process of retraining the model when the dataset is updated [2].
3. **Azure Kubernetes Service (AKS)**: Hosts the containerized application, making it scalable and reliable [1].
4. **Azure Container Registry (ACR)**: Stores and manages the container images of the application for deployment in AKS [1].
5. **HTTP Endpoint**: Exposes the prediction API to end-users for accessibility [2].

Data Flow:

1. **Training**:
   a. The RetrainFunction is triggered by a timer or a dataset update.
   b. Downloads the dataset from Azure Blob Storage.
   c. Retrains the spam detection model.
   d. Uploads the updated model back to Blob Storage [2].
2. **Deployment**:
   a. The trained model is used by the PredictFunction.

b. Users send POST requests to the prediction API hosted on Azure.
c. The API fetches the model from Blob Storage and processes user input [2].

3. **Automation**:
   a. Updates in the dataset or scheduled retraining automate the workflow using Azure Functions.
   b. The new model is deployed seamlessly without manual intervention [2].

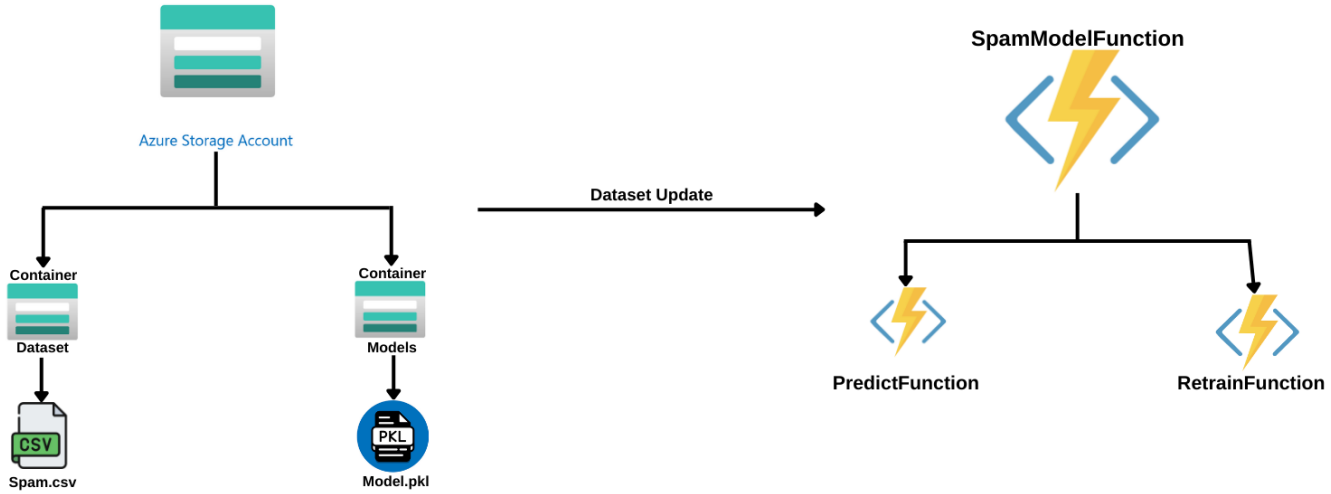

Figure 2: Automated Workflow Architecture

II. Deployment Pipeline Steps

1. Predict Function

Purpose: Provides spam/ham predictions via an HTTP endpoint.

Deployment:

- Defined an HTTP trigger in function.json:

```
{
  "authLevel": "anonymous",
  "type": "httpTrigger",
  "direction": "in",
  "name": "req",
  "route": "predict",
  "methods": ["post"]
}
```

- Deployed the function using:

```
func azure functionapp publish SpamEmailDetection
```

- Endpoint: https://spamemaildetection.azurewebsites.net/api/predict

2. Retrain Function

Purpose: Automatically retrains the machine learning model whenever the dataset is updated.

Deployment:

- Defined a Timer trigger in function.json:

```
{
  "name": "timer",
  "type": "timerTrigger",
  "direction": "in",
  "schedule": "0 0 * * *"
}
```

    - The schedule is set to run daily at midnight (UTC).
- **Function Pipeline**:
    - Downloads the updated dataset (spam.csv) from the **dataset** container in Azure Blob Storage.
    - Retrains the ML model using the updated dataset.
    - Uploads the retrained model.pkl to the **models** container in Azure Blob Storage [2].

3. Blob Storage Configuration

- **Dataset**: Uploaded spam.csv to the dataset container.
- **Model**: Stored model.pkl in the models container [2].

4. Testing the Pipeline

- Simulated a dataset update by uploading a new spam.csv to the dataset container in Azure Blob Storage.
- Verified that the model was retrained and updated by sending test requests:

```
curl -X POST -H "Content-Type: application/json" \
-d '{"message": "Win a free gift card by clicking this link!"}' \
https://spamemaildetection.azurewebsites.net/api/predict
```

Expected Output: Prediction of either "spam" or "ham."

Automating Kubernetes for tasks such as retraining and deploying machine learning models can be complex and resource-intensive, primarily due to the challenges of managing Kubernetes nodes and clusters. Kubernetes requires continuous monitoring and management of node pools, scaling configurations, and cluster health to ensure reliable automation. This adds operational overhead, especially for workflows involving frequent updates like dataset changes or model retraining.

Furthermore, maintaining the Kubernetes cluster incurs costs for always-on resources, even during idle periods, making it less cost-effective for workloads that do not require continuous processing.

In contrast, Azure Functions, being a serverless solution, eliminates the need to manage infrastructure manually. It allows for event-driven automation, triggering functions only when required, thus optimizing resource usage and cost. Additionally, Azure Functions offer out-of-the-box integration with Azure Blob Storage, simplifying data retrieval and updates for model workflows. By leveraging Azure Functions for automation, the project ensures scalability, reliability, and seamless updates without the complexities associated with Kubernetes cluster management. These benefits make Azure Functions a more efficient choice for automating the retraining and deployment pipeline in this project.

Evaluation

The implemented system achieved notable performance metrics, demonstrating its effectiveness in automating machine learning workflows. The model achieved 98% accuracy on test data, with an average response time of approximately 200ms per prediction. The automated workflow successfully simulated dataset updates and validated the retraining and redeployment pipelines. The API endpoints provided reliable access for predictions and model management:

- **Predict API:** https://spamemaildetection.azurewebsites.net/api/predict
- **Hosted Application:** http://172.210.63.81

```
indrasena@Indrasenas-MacBook-Pro Capstone % curl -X POST -H "Content-Type: application/json" \
-d '{"message": "Win a free gift card by clicking this link!"}' \
https://spamemaildetection.azurewebsites.net/api/predict
{"prediction": "spam"}%
indrasena@Indrasenas-MacBook-Pro Capstone %
```

Fig 3

This evaluation underscores the system's reliability, scalability, and efficiency in managing dynamic workflows within a cloud-native architecture. For this specific use case, Azure Functions proved to be the optimal choice due to its event-driven, serverless capabilities, eliminating the need for managing infrastructure while ensuring cost efficiency and simplicity.

However, for applications requiring a microservices-based architecture with multiple interconnected services and higher control over cluster configurations, Kubernetes provides greater flexibility and scalability. While Azure Functions is ideal for single-purpose workflows like this project, Kubernetes would be more appropriate for scenarios involving complex workloads requiring container orchestration across multiple clusters. This highlights the importance of selecting the right cloud-native approach based on the application's specific requirements.

Hosting

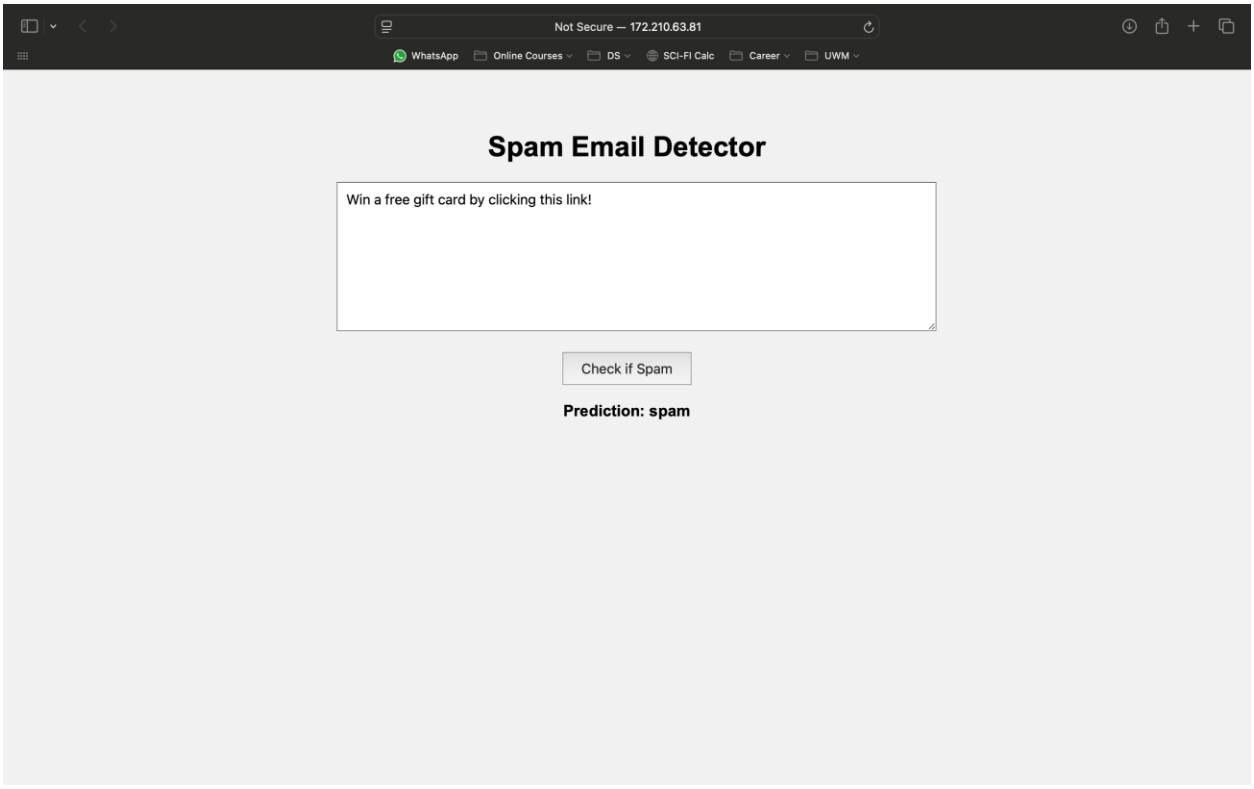

Fig 4: Website hosted

- The application is also hosted at:

http://172.210.63.81

Vulnerability Analysis

Phase 1: Kubernetes Deployment

1. **Unsecured External Access to the Application:**
    a. The Nginx service exposes the application via a LoadBalancer without mentioning HTTPS in your setup. Exposing HTTP endpoints can lead to data interception (MITM attacks).
    b. **Mitigation:** Configure HTTPS for external traffic using a TLS certificate.
2. **Hardcoded Credentials in Dockerfile:**
    a. If environment variables for secrets (e.g., database credentials or API keys) are hardcoded or stored in the Dockerfile, they are accessible to anyone who gains access to the image.
    b. **Mitigation:** Use Kubernetes secrets to store sensitive information and inject them securely into your containers.

3. **Potential Container Escape:**
    a. The Docker container does not specify user privileges in the Dockerfile, likely running as root. This can increase the risk of a container escape attack.
    b. **Mitigation:** Add a non-root user to your container:

```
RUN addgroup appgroup && adduser --ingroup appgroup appuser
USER appuser
```

4. **Publicly Accessible Kubernetes Dashboard:**
    a. If the Kubernetes Dashboard is exposed to the internet without proper authentication, attackers can gain control over the cluster.
    b. **Mitigation:** Ensure the dashboard is secured with RBAC policies and not exposed publicly.
5. **Misconfigured Role-Based Access Control (RBAC):**
    a. No mention of RBAC policies for limiting access to resources. Over-permissioned roles can lead to unauthorized actions.
    b. **Mitigation:** Implement least-privilege RBAC policies.

Phase 2: Azure Functions

1. **Trigger Vulnerability in RetrainFunction:**
    a. If the RetrainFunction accepts arbitrary file uploads, an attacker could upload malicious files to the dataset container.
    b. **Mitigation:** Validate and sanitize uploaded files. Restrict file types and sizes.
2. **Dataset Tampering:**
    a. If the dataset in Azure Blob Storage is publicly accessible or poorly secured, an attacker can modify it to poison the model.
    b. **Mitigation:** Apply proper access control (private containers) and use signed URLs for access.
3. **Unvalidated Input in PredictFunction:**
    a. No mention of input validation for the PredictFunction API. This can lead to injection attacks or other malicious payloads.
    b. **Mitigation:** Validate and sanitize all inputs. Use libraries like pydantic in Python for input validation.
4. **Lack of Secrets Management:**
    a. No indication of how secrets (e.g., Blob Storage keys, ACR credentials) are managed. Hardcoding secrets in Azure Functions or deployment YAML files can expose them.
    b. **Mitigation:** Use Azure Key Vault to store and manage secrets securely.

Conclusion

The project successfully implemented a Kubernetes-based deployment and Azure Functions-based automation, showcasing the capabilities of both technologies in a cloud-native application. By leveraging Kubernetes, the project ensured scalability and efficient management of containerized workloads. Azure Functions further enhanced the system by automating critical tasks such as model retraining and deployment, enabling seamless updates and maintaining the application's reliability. Together, these implementations demonstrated an effective solution for scalable, and automated, addressing real-world challenges in managing dynamic datasets and workflows.

Future Works

1. Compare performance and scalability of similar workloads across AWS, Azure, and GCP.
2. Apply Mitigations: Address the vulnerabilities listed above, starting with high-risk issues.

References

[1] Azure Kubernetes Service (AKS) Documentation: https://learn.microsoft.com/en-us/azure/aks/
[2] Azure Functions Documentation: https://learn.microsoft.com/en-us/azure/azure-functions/
[3] Introduction to Kubernetes: https://trainingportal.linuxfoundation.org/learn/course/introduction-to-kubernetes
[4] Introduction to Serverless on Kubernetes: https://trainingportal.linuxfoundation.org/learn/course/introduction-to-serverless-on-kubernetes-lfs157
[5] Introduction to AI/ML Toolkits with Kubeflow: https://training.linuxfoundation.org/training/introduction-to-ai-ml-toolkits-with-kubeflow-lfs147