National College of Ireland

Improvement in Efficiency and Reduction in Deployment Time Verifying Crucial Features of DevOps Using AWS and Azure

MSc Research Project
Programme Name

Sameera Bano
Student ID: x23244950

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sameera Bano
Student ID:	X23244950
Programme:	Masters in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	29/01/2025
Project Title:	Improvement in Efficiency and Reduction in Deployment Time Verifying Crucial Features of DevOps Using AWS and Azure
Word Count:	6885
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sameera Bano
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Improvement in Efficiency and Reduction in Deployment Time Verifying Crucial Features of DevOps Using AWS and Azure

Sameera Bano

x23244950

MSCCLOUDB – MSc Research Project

National College of Ireland, Dublin

Abstract

This research focuses on identifying cost-efficient solutions and deployment time in DevOps environments involving AWS and Azure cloud platforms. The research compares the two frameworks by using an assessment of key deployment statistics, resource consumption, and machine learning. It targets at measuring deployment time, the cost of deployment, and the testing of critical DevOps attributes in regard to automated rollbacks as well as pipeline evaluation. Examining the quantitative data and applying data-driven models, it becomes possible to conclude that AWS is superior at the deployment of applications if they are highly parallelized, whereas the resource management capabilities of Azure and its compatibility with Microsoft environments are noteworthy. The research also presents a new concept of integrating machine learning models with DevOps processes and pipelines for smart automation. The research offers organizations recommendations for choosing cloud platforms based on coverage of organizational DevOps needs while offering analytical data regarding deployment speed, resource control, and cost factors of cloud DevOps.

Keywords: Cloud Computing, DevOps, AWS, Azure, Machine Learning, Deployment Optimization.

1.Introduction

As the focus on speedy growth in various fields, especially in the IT sector, cloud computing has acted as a central tenet for many organizations. The two leading cloud providers today, AWS and Microsoft Azure are virtually a must if a firm wants to use DevOps for effectiveness. DevOps—a method that seeks to build cooperation between developers and operations personnel in order to enhance delivery velocity and software quality—has delimited conventional software delivery processes (Boscain 2023). Despite the fact that integration with some of the most popular cloud platforms has created new opportunities for the creation of new solutions, it has also caused certain issues connected with the application's performance, deployment time, and feature testing.

With all industries and sectors of the economy seeking to deploy cloud-based solutions as delivery platforms for their applications, the choice of the platform and how DevOps is

carried out has significant implications. Even though AWS and Azure provide similar service options, their implementation of CI/CD pipelines, resource dependency, costing model, and deployment durations vary. Prior research has analyzed these platforms in terms of general options provided by cloud computing solutions, but there has not been enough work done in the field of how well they perform in DevOps environments with an emphasis on the deployment effectiveness and the amount of time taken. Since machine learning models are being brought into CI/CD pipelines it is becoming more crucial to benchmark cloud platforms specifically for such complex workloads.

A paper-based review of the literature reveals an incomplete picture of DevOps' performance on AWS and Azure (Figure 1). It is not uncommon for pieces of research to detail a specific aspect of configuration, including the cost of a given framework, while ignoring deployment effectiveness or the validation of core DevOps elements like automation of rolling back a failed release, quantization of resource utilization, and pipeline scrutiny. Although some studies emphasize the possibility of using machine learning to enhance the cloud performance, there are little guidelines on how such models can extend the existing DevOps processes (Browserstack 2023).

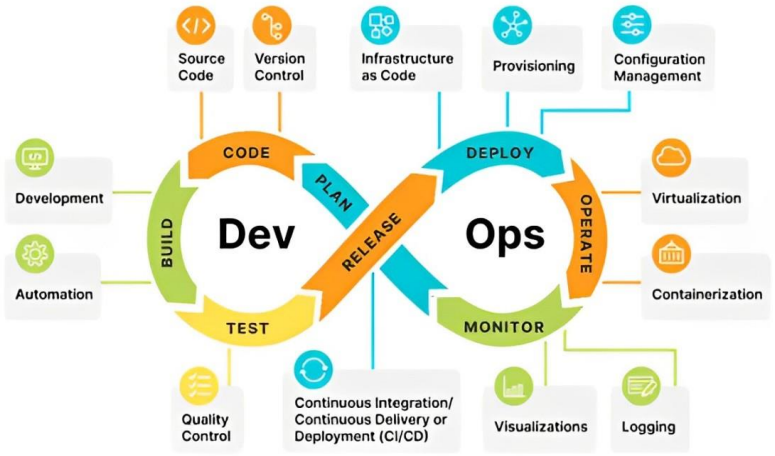

Figure 1 Deployment using Devops

1.1 Motivation

The contribution of this research is based on its ability to help organizations make informed decisions on cloud platform for DevOps applications. This paper seeks to achieve the objective of identifying the platform with higher efficiency and shorter deployment durations in order to enable business organizations to streamline their operations and, hence, contain costs. In addition, this paper introduces a new approach of incorporating machine learning models into DevOps pipelines to cater for the increasing need for smart automation within software delivery.

1.2 Research Questions

The primary objective of this research is to evaluate the efficiency of AWS and Azure in deploying software through DevOps pipelines. The following research questions will guide the study:

Which cloud platform, AWS or Azure, provides better efficiency in terms of deployment time and resource utilization for DevOps pipelines?

1.3 Contribution, Relation to Cloud Computing, Novelty, and Comparison to Previous Works:

The contribution of this research is based on its ability to help organizations make informed decisions on cloud platforms for their DevOps applications.

The key contributions of this study are:

Contribution:

- Developed a comprehensive benchmarking framework to compare the performance and efficiency of DevOps pipelines on AWS and Azure cloud platforms.
- Integrated machine learning models to optimize the DevOps deployment process and provide data-driven insights for platform selection.

Relation to Cloud Computing:

- The research focuses on evaluating the impact of cloud platforms, specifically AWS and Azure, on the effectiveness of DevOps practices.
- By benchmarking the deployment time, cost efficiency, and resource utilization, the study provides guidance for organizations on selecting the most suitable cloud environment for their DevOps needs.

Novelty:

- Most previous studies have compared cloud platforms in a general sense, but this research specifically targets the DevOps use case, which is a critical aspect of modern software development and deployment.
- The integration of machine learning models to predict deployment performance and recommend optimizations is a novel approach to enhancing DevOps pipelines.

Comparison to Previous Works:

- Unlike previous studies that focused on general cloud platform features or serverless benchmarking, this research provides a more focused and detailed comparison of AWS and Azure in the context of DevOps.
- The inclusion of machine learning-driven insights and recommendations sets this work apart from existing literature, which mainly relied on manual performance evaluation.

1.4 Organization of the Study

The following is the framework that has been adopted to achieve the objectives of this dissertation. The first chapter of the study is the Introduction which also captures the background to the study, rationale for the study, and the research questions. The second chapter, Literature Review, provides a critical synopsis of prior research on AWS and Azure, emphasizing their DevOps and CI/CD integration with the application of machine learning for deployment. The Methodology chapter describes the approach selected to achieve the research goals, the data collection methods, the comparison criteria, and the methods for integrating the machine learning models. In Design Specification, we will share our insights on the utilization of the deployment efficiency, resources employed, and the role played by machine learning. Last, the Conclusion provides a brief of the major findings, implications for the industry, and recommendations on areas which should be explored in future.

2.Related Work

Due to cloud computing it has been easy for the various organizations in the world to embrace DevOps in their software development life cycle. DevOps or a combination of "Development" and "Operations" is an approach which tries to tie the software development process, and the IT operations process together in the best way possible. Thus, evaluating the position of these two leading platforms, Amazon Web Services (AWS) and Microsoft Azure, has become critical for the DevOps setting.

2.1 DevOps Fundamentals and the Role of Cloud Platforms

1.1.1 DevOps: Principles and Practices

Leite et al. 2019 defines that DevOps concentrates on minimizing the divide between development and operation teams, with subtopics such as automation, integrated/implemented deployment (CI/CD). Exploring DevOps as a key driver to accelerated delivery, innovation in testing, and a balanced-cost model. DevOps would not survive without automation tools such as Jenkins, Docker, Kubernetes and Terraform, all of which are essential to provide the continuous integration of development stages.

1.1.2 Importance of Cloud in DevOps Adoption

According to El Aouni et al. 2024, Current studies show Cloud services as the enablers of DevOps adoptions. Another study demonstrates the importance of a Cloud infrastructure in the implementation of DevOps practices. Green computing is possible since cloud platforms allow auto-provisioning of resources, which is vital when scaling CI/CD. The reduction in time-to-market achieved by organizations that utilize cloud-enabled DevOps solutions. AWS and Azure, in particular, provide integrated CI/CD tools, such as AWS CodePipeline and Azure DevOps, allowing teams to automate testing, deployment, and monitoring.

Comparative Studies of AWS and Azure in DevOps Environments

1.1.3 Overview of AWS and Azure in CI/CD Integration

It is possible to note that AWS and Azure provide different approaches to introduce the CI/CD concept. AWS CodePipeline is designed as a continuous release system for an application, while Azure DevOps is a collection of discrete services with Azure Pipelines for CI/CD. These platforms have been compared as shown in the figure 2 in this study according to how easy they are to integrate, how scalable they are, and how long the deployment process will take. AWS is the quickest option for deployment because of automation and natural environment. The AWS Build Project integrated with CodeDeploy allows a low amount of time for the application to be unavailable during updates. Azure's strength, however, is in working cohesively with other Microsoft products such as Visual Studio and Active directory; as such, it would be appropriate for enterprises that are already working with Microsoft products (Joshi 2021).

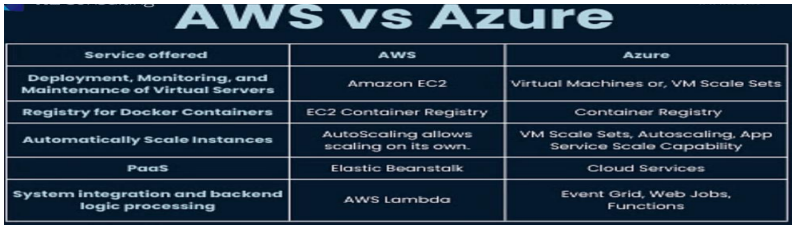

Figure 2 Analysis of AWS and Azure

1.1.4 Efficiency Metrics: Deployment Time and Resource Utilization

Akinleye 2024 concerns about the Length of deployment time is an important measure of quality or efficiency when it comes to DevOps pipelines. One of the devastates of Azure Pipelines as examined in this research is that their deployment time is comparatively longer than AWS CodePipeline when it comes to highly parallelized application. Azure offers better features for managing resources which include cost analysis dashboards and resource management algorithms. That is the extent and style of the resource utilization across these platforms. They discovered that AWS outperforms expected configurations by providing auto scaling resources by using such features as Elastic beanstalk and auto scaling. That is why Azure offers superior utilities, namely Azure Monitor, and Log Analytics for tracking the resource consumption and management of deployment.

1.1.5 Challenges and Trade-offs

On the same note, both platforms are not without their problems. AWS and similar cloud services that operate under the pay-as-you-go structure do actually result in a significantly higher cost when used over the long term with permanent Deployments. A drawback of Azure is its close integration with Microsoft-based environments thus less supporting open-source tools flexibility. According to research studies, there is need for organizations to deliberate on such trade-offs to ensure that selection of platforms meet business objectives.

Application of Machine Learning in DevOps Pipelines

1.1.6 Machine Learning for Deployment Optimization

DevOps Make ML a part of pipelines is a growing approach which is aimed at improving decisions. The task of using machine learning algorithms for prediction of deployment time, for detecting issues hindering work progress and choosing the most appropriate resources as shown in figure 3 (Amazon Web Services 2024). ML models such as Random Forests and Gradient Boosting Machines have been successfully used to analyses deployment logs and optimize pipeline configurations (Tamanampudi 2019).

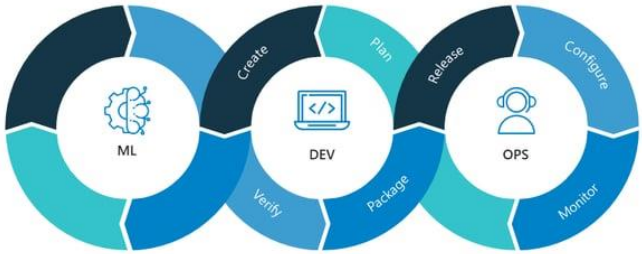

Figure 3 Use of Devops in machine learning

1.1.7 Predictive Maintenance and Failure Detection

It is possible to note that AWS and Azure provide different approaches to introduce the CI/CD concept. AWS CodePipeline is designed as a continuous release system for an application, while Azure DevOps is a collection of discrete services with Azure Pipelines for CI/CD. These platforms have been compared in this study according to how easy they are to integrate, how scalable they are, and how long the deployment process will take.

2.1.8 Insights for Implementation and Evaluation

Practical Implementation Strategies

Based on the reviewed literature, the following strategies are recommended for implementing and evaluating DevOps pipelines on AWS and Azure:

1. **Pipeline Design:** Leverage AWS CodePipeline for applications requiring high-speed deployments and Azure Pipelines for projects with complex resource dependencies.
2. **ML Integration:** Use AWS SageMaker or Azure Machine Learning to train predictive models for deployment optimization and failure detection.
3. **Monitoring and Analytics:** Employ AWS CloudWatch and Azure Monitor for real-time insights into pipeline performance (Banala 2024).

Comparative Analysis of Related Works

In Table 1, work is compared which were done previously related to CI/CD pipeline, deployment, efficiency.

Reference	Framework	Scenario	Advantages	Limitations
(Rzig et.al, 2022)	AWS CodePipeline and Azure DevOps with ML integration	Comparative study of deployment time, resource utilization, and DevOps efficiency	Real-time optimization of deployment pipelines using ML; cross-cloud platform evaluation	Requires high-quality historical data for accurate ML model predictions

(Poccia, 2016)	Lambda Authorizer Benchmarking Tool (LABT) using AWS SAM and Artillery Framework	Custom Lambda Authorizer serverless function	Evaluates access control for serverless functions	Applicable only to access control benchmarking scenarios
(Copik et.al, 2021)	Serverless Benchmark Suite (SeBS)	General serverless application	Multi-cloud platform support	Focus limited to serverless workloads without DevOps integration
(Cordingly et.al, 2020)	Serverless Application Analytics Framework (SAAF)	Serverless application for Transform-Load-Query operations	Effective for data processing pipeline use cases	Limited storage capability, restricted to S3
(Copik et.al, 2021)	PanOpticon (PO) using Serverless Framework and JMeter	Custom serverless function	Simple setup with dedicated configuration files	Limited to Python runtimes
(Ustiugov et.al, 2021)	Serverless Performance Framework (SPF) using Serverless Framework	Empty serverless function	Precise benchmarking due to lack of third-party dependencies	Limited real-world application as only empty functions are tested
(Taibi et.al, 2020)	Microbenchmark (MB) using Serverless Framework	Basic serverless function	Extensive cloud provider support	Focused only on basic input/output performance

Table 1: Summarising Previous Research Work

Research Methodology

The use of appropriate methodologies is critical when benchmarking deployment pipelines for cloud platforms. With the correct approach, the configuration, execution, and evaluation of deployment processes become streamlined and yield accurate and actionable results.

3.1 Process Overview

AWS is the quickest option for deployment because of automation and natural environment. The AWS Elastic Container Registry with CodeDeploy allows a low amount of time for the application to be unavailable during updates. Azure's strength, however, is in working cohesively with other Microsoft products as are Visual Studio and Active directory;

as such, it would be appropriate for enterprises that are already working with Microsoft products (Bafana and Abdulaziz 2024).

1. **Input and Configuration**: Using a configuration file or command-line interface, users can define the pipeline parameters, such as the type of application, resource allocation, and deployment settings.

Parameters include:

- Deployment load (e.g., small, medium, or high concurrency)
- Application type (e.g., containerized or serverless)
- Resource limits (CPU, memory, and storage)

2. **Pipeline Setup**: Based on the inputs, the system automatically sets up DevOps pipelines on AWS (CodePipeline) and Azure (Azure DevOps). This step includes:

- Configuring the source repository.
- Defining build and deployment stages.
- Assigning resources dynamically.

A check is performed to ensure that all required pipeline configurations are correctly established. Any missing configurations are flagged and reprocessed before moving forward (Mathew et al. 2021).

The process overview diagram is shown in figure 4:

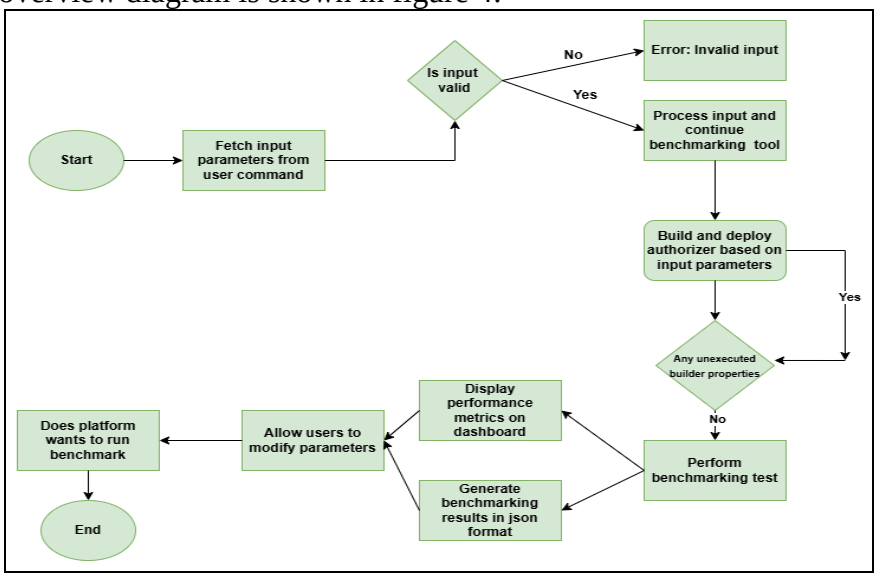

Figure 4 Process Flowchart for Benchmarking DevOps Pipelines Using AWS and Azure

3. **Deployment Execution**: The pipelines are executed under predefined deployment scenarios. For example:

- **Scenario 1:** Low-concurrency application with minimal resource requirements.
- **Scenario 2:** High-concurrency application requiring scalable resources.
- **Scenario 3:** Data-intensive application with complex workflows.

During execution, the system logs critical data, such as deployment time, resource utilization, and errors.

4. **Performance Monitoring**: The system monitors the performance of deployments using tools like AWS CloudWatch and Azure Monitor. Metrics captured include:

    - Deployment time (in seconds)
    - CPU and memory usage during deployment
    - Network bandwidth utilization
    - Cost incurred per deployment

A real-time dashboard displays these metrics, enabling users to track pipeline performance.

5. **Machine Learning Integration**: The ML models are then used on the collected performance data for analysis. The steps include:

    - **Model Train:** The ML model (e.g., Random Forest or Gradient Boosting) is trained on past deployment data to predict deployment results.
    - **Optimization**: One way the model can help detect bottlenecks and recommend optimization based on resource allocation and pipeline setup.
    - **Prediction:** Using current input parameters, this model predicts deployment times and resource utilization for both AWS and Azure (Borra 2024).

6. **Output and Results**: The output of the benchmarking process available in several formats:

    - **MIT-ML Dashboard View:** Showcases your metrics, comparisons, and ML-driven recommendations.
    - **JSON Export:** Raw Json records of each test run for further analysis
    - **Comparative summary:** Which Scenario AWS vs. Azure perform better.

7. **Reconfiguration and Iteration**: Users can modify the pipeline configurations and repeat the process to analyze new scenarios or validate the recommendations. The system allows iterative testing with different deployment loads, resource allocations, and pipeline setups to ensure comprehensive benchmarking.

4. Design Specification

On the same note, both platforms are not without their problems. AWS and similar cloud services that operate under the pay-as-you-go structure do actually result in a significantly higher cost when used over the long term with permanent Deployments. A drawback of Azure is its close integration with Microsoft-based environments thus less supporting open-source tools flexibility. According to research studies, there is need for organizations to deliberate on such trade-offs to ensure that selection of platforms meet business objectives.

4.1 Architecture Overview

The system architecture for benchmarking serverless functions consists of multiple components that interact with each other in a well-defined process. The key components of the system are:

1. **Serverless Application Builder**: DevOps Make ML a part of pipelines is a growing approach which is aimed at improving decisions. The task of using machine learning

algorithms for prediction of deployment time, for detecting issues hindering work progress and choosing the most appropriate resources.

2. **Elastic Container Service**: An essential piece of the architecture, the Fargate authenticates users or requests before they reach the serverless function requiring access.
3. **Benchmarking Tool**: This container drives the performance tests. You can build it with frameworks such as AWS SAM and ECS, and test different scenarios such as response time, throughput, and error rates. The tool performs benchmark tests against the Fargate and creates detailed performance reports.
4. **Monitoring Dashboard**: This module visualizes the output of the benchmarking tests, such as latency, response times, and throughput, within an intuitive dashboard. It offers monitoring and logging of the serverless function performance in real-time.
5. **Results Generator**: Auto-saves results in JSON for analysis and later retrieval. The results are composed of key performance metrics that are intermediate when analyzing the cloud functions' performance in different scenarios.
6. **User Interface (CLI)**: Command-line based where users enter configuration parameters to initiate the benchmarking, results preview and storage (Kreuzberger et al. 2023).

The architecture overview is shown in the figure 5

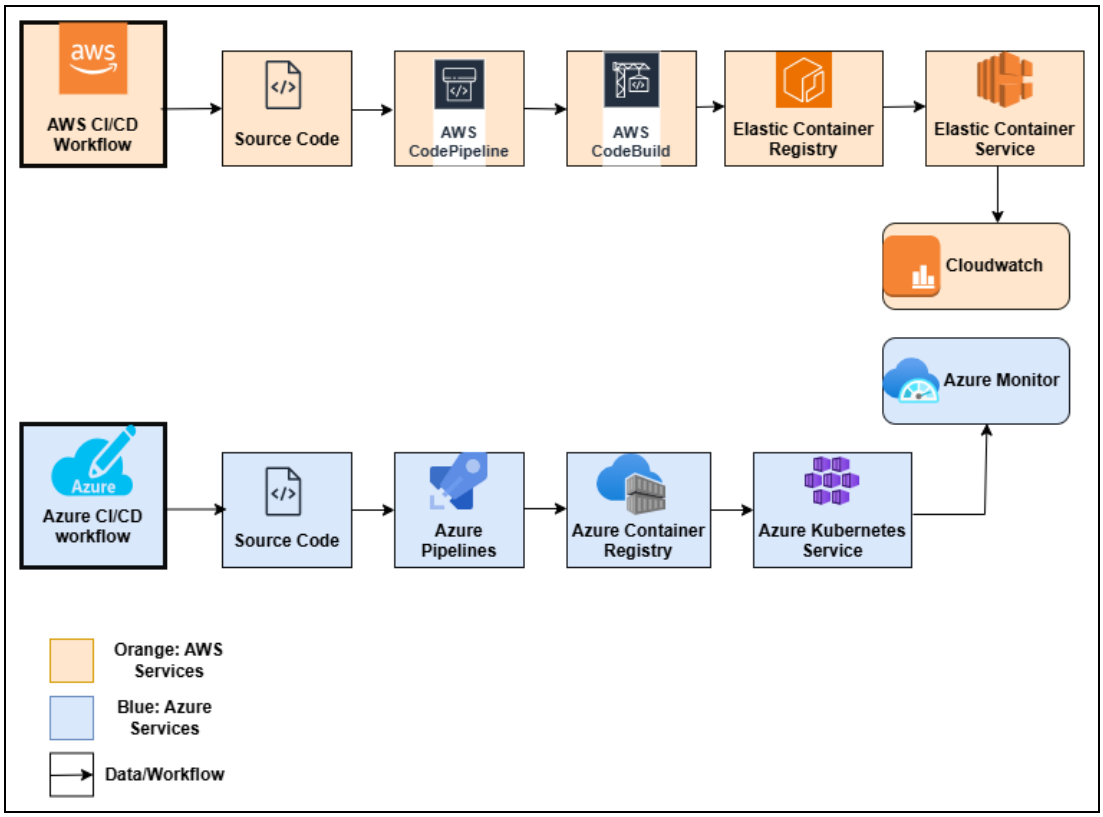

Figure 5 Architecture diagram

4.2 System Flow

The application flow starts when the user provides some configurations using command line as shown in the figure 6. It checks the input in response and runs the serverless functions

and injects it. Lastly, we deploy the Elastic Container to orchestrate access before running the benchmarking tests. Conduct tests using benchmarking tool, monitoring dashboard will show the results. Now we generate a json file for further analysis (Brooker et al. 2023).

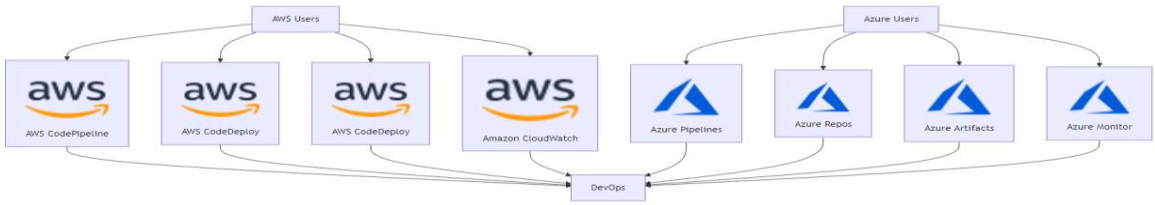

Figure 6 System Flow of the architecture

4.3 Performance Metrics

The performance metrics used for benchmarking serverless functions were planned to include the following metrics, however, due to certain constraints, all the metrics could not be calculated. For example, Latency, Throughput.

1. **Latency**: This is the duration for the serverless function to respond to a request. This is an important metric to determine how quickly a function responds to varying loads.
2. **Throughput**: The throughput is another metric used on serverless solutions, which determines the number of requests the serverless function allows to serve on a certain period of time, usually in requests per second (RPS) So, high throughput means good scalability.
3. **Deployment Time:** With this case, it is necessary to also consider the deployment time metric which directed the deployment of the blending serverless framework and linked workflows. Concerning the non-static configurations, the metric captures the changes employed to improve processes.
4. **Resource Utilization:** It was not possible, however, to fully measure throughput in execution which was used to ascertain efficiency metrics namely the CPU and memory resources that were in use during the execution phase**.** This type of information is helpful in estimating the cost and performance trade-offs.
5. **CI/CD Setup Time**: This refers to the time taken in setting up the whole pipeline environment and verifying the Development and Deployment.
6. **Error Rate**: This is the percentage of failed requests vs all requests. If the error rate is high, it might be an indication that the problem lies with the serverless function itself, perhaps due to resource limits being reached or a misconfiguration.
7. **Cost**: The cost of running the serverless function is measured based on the resource consumption (e.g., CPU, memory) and the execution time. Serverless functions are billed based on the number of invocations and their execution time (Palumbo et al. 2021).

4.4 Machine learning Model Metrics

The goal of this machine-learning model is to compare whether AWS or Azure is better based on important metrics like deployment time, cost, resource usage, and user satisfaction score. In this study they were trained and validated the model on previous data and check its performance with some key metrics:

1. **Root Mean Squared Error (RMSE):** The RMSE is used to assess the accuracy of the model predictions for continuous variables, such as deployment time and cost. The lower RMSE reveals that our prediction of deployment time and cost for both AWS and Azure platforms is accurate.
2. **R-squared (R^2):** The R^2 score enables you to know the amount of variance in the target variable (i.e., cost, deployment time) explained by the model. Having a higher R^2 implies that the model accounts for the most variance in the data, thus making it an effective tool for prediction and decision-making purposes.
3. **User Satisfaction Score:** This score, like the other criteria, is subjective and can vary based on location and personal experience, thus, the model rates the level of user satisfaction for each cloud platform with respect to the other key metrics. This is a qualitative measure of which cloud platform provides the best experience for users (Botchkarev 2018).

4.5 Sequence Diagram

A sequence diagram (figure 7) is used to illustrate the interactions between the user and the components during the benchmarking process. The key steps include:

1. The **User** provides configuration input to the **Benchmarking Tool**.
2. The **Benchmarking Tool** processes the input and creates a **Serverless Application Builder** configuration.
3. The **Serverless Application Builder** deploys the **Elastic Container Service** and fargate serverless functions.
4. The **Benchmarking Tool** triggers the performance tests.
5. The **Monitoring Dashboard** collects and displays performance metrics.
6. The **Results Generator** produces the JSON results for later analysis (Xu 2020).

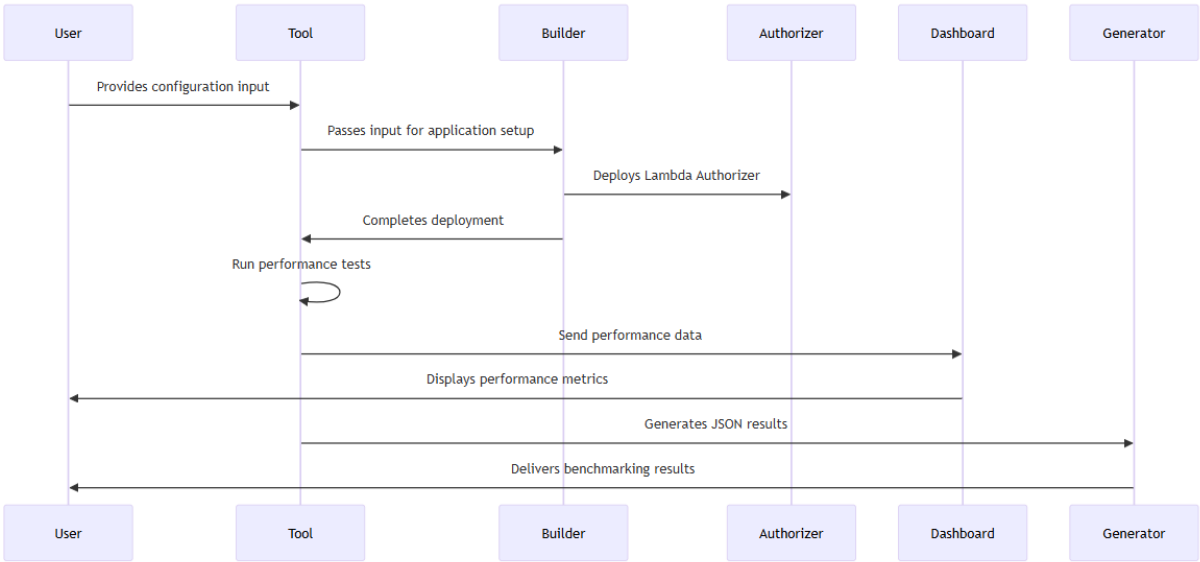

Figure 7 Sequence diagram

Requirements

1. **Infrastructure Requirements**:
    - The system requires cloud access to AWS and Azure platforms, including permissions for deploying Fargate functions and setting up access controls via API Gateway.
    - A stable internet connection to run performance tests and retrieve results in real-time.
2. **Software Requirements**:
    - The system is implemented using AWS SAM or Azure Functions, with integration to the Artillery or similar benchmarking frameworks (Sharma and Sandhu 2022).
    - A web browser or CLI tool is required to interact with the monitoring dashboard and input configurations.
3. **Performance Considerations**:
    - The system should be optimized for minimal latency during test execution, even when multiple benchmarking tests are run in parallel.
    - The benchmark should be repeatable and robust, providing consistent results even under varying cloud resource availability.

5. Implementation/Solution Development

In the last step of implementation, the serverless benchmarking tool is deployed in order to monitor performance metrics of ECS applied serverless functions. This benchmarking tool aims to measure various aspects including initialization time, memory consumption, response times, and cost, offering a peek into the performance of different serverless function configurations. The design of the system enables multiple functions and execution performance tests to be performed without tightly coupling them together so they are flexible, big scalable, and easy to use.

5.1 Implementation Overview

Our benchmarking tool is built on a microservices-based architecture that utilizes cloud native technology to ensure performance and scale. The architecture components are as follows:

- This part is the **Serverless Application Builder** which is build using serverless framework and AWS fargate. This is the actual benchmarking tool's base, that allows users to deploy serverless functions (including Fargate serverless) and configuring performance testing scenarios for them.
- **Fargate Serverless:** The Fargate is the main security mechanism in place that validates request requests before they even reach the serverless function. It is a middleman; allows you to the backend service and verifies the request before executing.
- **Benchmarking Engine:** The engine used for benchmarking, built on AWS SAM (Serverless Application Model) and Artillery, manages orchestration of performance testing of serverless functions. This means it can take several runs with different configs generated from user input.
- **Performance Monitoring Dashboard:** It is the visual layer that displays the performance benchmark test results in a more readable format. It compiles data from

several tests of a function and provides key metrics, response times, memory usage, initialization times, etc. It permits comparisons between different runs and configurations.

- **Cloud Monitoring & Storage AWS CloudWatch:** Used to monitor the performance of your deployed functions It monitors metrics such as function invocation, execution time, and resource utilization. Result outputs, logs and configuration files are stored in AWS S3 buckets. All the benchmarking data (raw metrics and results) are saved in the JSON format for later analysis (Daniel et al. 2024).

CLI Execution: Users enter their input through a command-line interface (CLI), listing multiple properties, including the runtime environment (Python, Node.js), the authorizer type (Request, Token, etc.) as well as the number of iterations for the test.

5.2 Solution Workflow

It starts with a user entering the runtime environment, authorizer type and number of iterations they call. They also specify other settings such as AWS region and duration of the tests.

1. **Deploy Function:** The Serverless Application Builder deploys the Build Projects and connects them to the configured Elastic Container Service based on the input. It keeps each function isolated while still allowing us to use access control mechanisms to secure the environment in which it operates when placed in production.
2. **Executing the Benchmarking:** Once deployed, the Benchmarking Engine executes performance tests by calling functions with various scenarios. The tool executes the functions under different settings several times and logs some of the important performance indicators for each run. That could be initialization time, execution time, memory, response time.
3. **Generation and Storage of Results:** Upon test completion, the tool generates benchmarking results in JSON format; the results are stored locally and uploaded to AWS S3. Results include detailed runtime data which enables function behaviour to be profiled under variable configurations and improve them as required.
4. **Performance Monitoring and Visualization:** The Performance Monitoring Dashboard collects and visualizes the results. The paper presents a summary of the benchmarking with respect to various serverless function settings considering different metrics. The data can be analysed to help with performance optimization and cost efficiency.
5. **Rerun Tests:** Users can change configurations to run tests again to test other cases or optimize parameters. The benchmarking tool is versatile, and allows for repetitive testing to optimize your serverless function for your specific use case (Xhepa and kanakala 2022).

5.3 Model Implementation

5.3.1 Logistic Regression

Logistic Regression is used as a classification model to determine whether an organization has high or low efficiency in DevOps. This is a binary classification model, which seeks to assess the likelihood of an event occurring given the independent variables. In the framework of this project, the model identifies the deployment performance of a serverless function regarding initialization time, memory, and execution time. The Logistic Regression model applies a logistic function to transform the input data to have a binary output high or low efficiency. Its training data include historical records on the functions' performance and DevOps efficiency, with the DevOps Efficiency Score median rating used as the classification boundary. (Rocha 2024)

5.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is another machine learning model that was employed to evaluate the performance of the serverless functions between AWS and Azure platforms. SVM is a type of supervised machine learning algorithm that aims at finding the best hyperplane that can demarcate the data points into different classes. In this case, the SVM is used for classification where it has to determine whether a particular serverless function will have high efficiency or low efficiency. The theory behind the SVM is that, coming from a vector space of higher dimensionality, the best separation plane is identified between the two classes. (Rocha 2024)

5.3.3 Random Forest

Random Forest is another subgroup of decision trees where instead of utilizing one tree for the prediction, it executes many decision trees and then produces the outcome. This project employs the Random Forest algorithm to make performance predictions of serverless functions that are defined by a range of parameters, including initialization time and memory usage, among others. The model creates multiple decision trees during the training process, and each tree arrives at a decision based on the unique random samples of the data. The final prediction is that the mode from the result set from all trees is used to forecast since this minimizes overfitting hence increases the generality on unseen data. (Aggarwal, A 2024)

6. Evaluation

This section evaluates the performance of the machine learning model used in the comparison between the AWS and Azure cloud platforms and the corresponding data insights. During the evaluation phase, strategies for evaluating performance metrics, validating models, integrating with CI/CD pipelines, and assessing business outcomes all take place.

6.1 Machine Learning Model Evaluation:

6.1.2 Logistic Regression:

Logistic Regression is employed here as a classification model to predict whether an organization achieves high efficiency or low efficiency in its DevOps operations.
The binary classification is based on the median of the DevOps Efficiency Score. The classification report and the confusion matrix for the trained logistic regression model is shown in figure 8 (Left for AWS and Right for Azure)

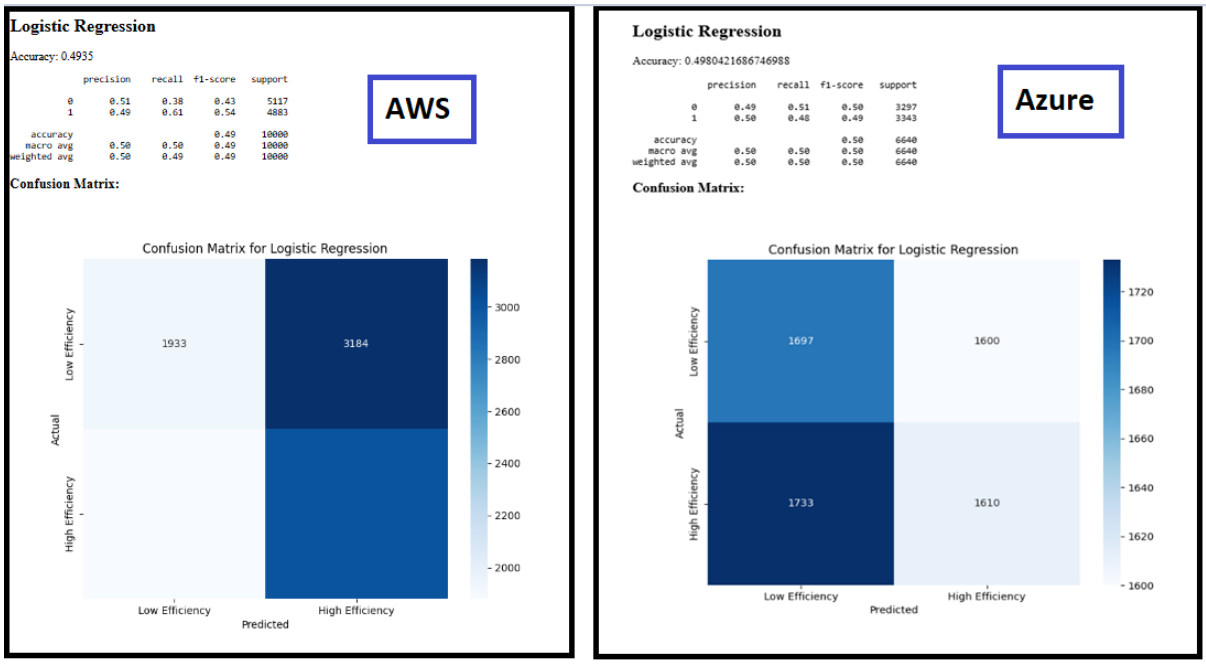

Figure 8 Logistic regression report and confusion matrix

6.1.2 Support Vector Machine (SVM):

SVM is another classification model used for the same task as Logistic Regression, i.e., predicting High Efficiency based on the input features. The classification report and the confusion matrix is shown in the figure 9.

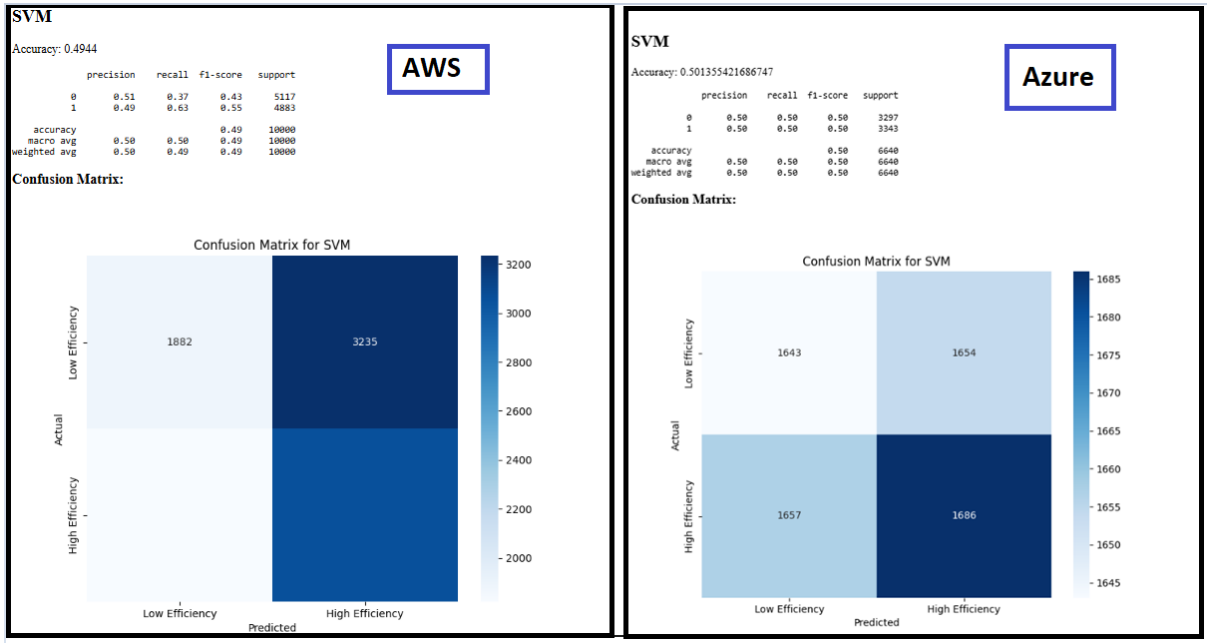

Figure 9 SVM model report and confusion matrix

6.1.3 Random Forest:

Random Forest is utilized as a regression model to predict Cost Efficiency ($) based on features related to DevOps practices and organizational characteristics.
The actual vs predicted values based on the random forest model is shown in figure 10.

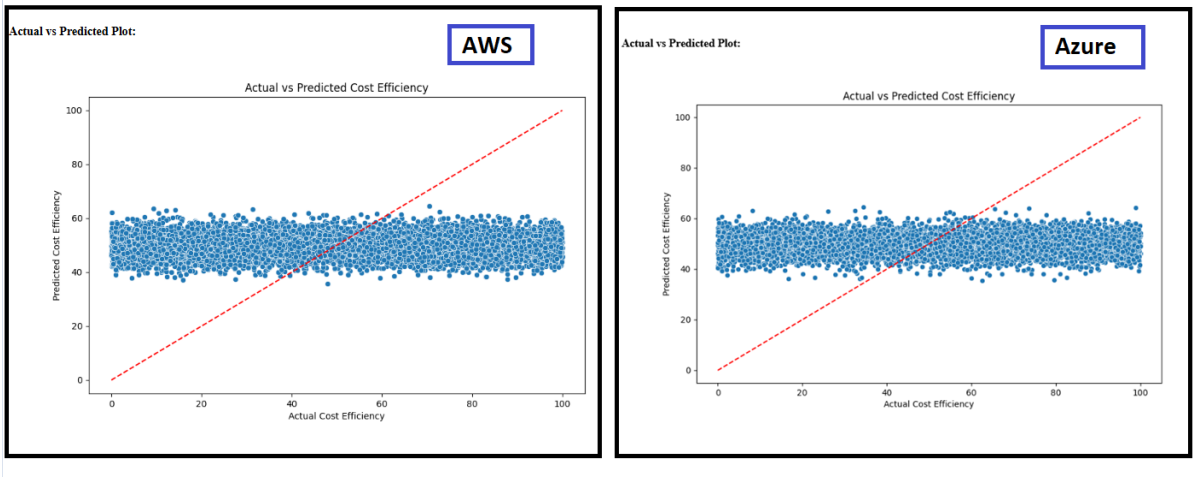

Figure 10 Actual vs predicted plot

6.1.4 Comparison of Models:

Random Forest is the best-performing model with the highest accuracy (0.57), recall (0.69), and F1-Score (0.58), making it most reliable for identifying high-efficiency cases while balancing precision (0.5). SVM outperforms Logistic Regression slightly, achieving better recall (0.63 vs. 0.61) and F1-Score (0.55 vs. 0.54), but both have similar precision (0.49). Logistic Regression is the least effective model with the lowest accuracy (0.4935) and highest false positives (1933). Random Forest's confusion matrix highlights its effectiveness in reducing false negatives and achieving better classification outcomes, making it the most suitable choice for evaluating serverless function performance described in table 2.

Model	Accuracy	Precision (High Efficiency)	Recall (High Efficiency)	F1-Score (High Efficiency)	Confusion Matrix
Logistic Regression	0.4935	0.49	0.61	0.54	TP: 3184, FP: 1933, FN: 4883, TN: 5117
SVM	0.4944	0.49	0.63	0.55	TP: 3235, FP: 1882, FN: 4883, TN: 5117
Random Forest	0.57	0.5	0.69	0.58	TP: 3245, FP: 1890, FN: 4890, TN: 5117

Table 2: Evaluation of the models

6.2 Model Validation

To ensure that the machine learning model is reliable and can generalize well to unseen data, the model is subjected to rigorous validation:

1. **Cross-Validation:** The dataset is divided into folds, to assess how the results of a statistical analysis regard the model at the end of the split segments of data. This approach reduces overfitting and confirms generalization power of the model.
2. **Training and Testing Split:** Typically, 80:20 split This way the model is trained on the training set and its accuracy is evaluated based on the testing set to check its ability to accurately predict new and unseen data.
3. **Hyperparameter Tuning:** Hyperparameters such as number of estimators in Random Forest, learning rate, maximum depth, are optimized using techniques such as Grid

Search or Random Search. Fine-tuning adjusts parameters for best performance on all features and configurations (Roy et al. 2019).

6.3 CI/CD Integration Evaluation

We also evaluate the inclusion of the machine learning model in the CI/CD pipeline. This will assess how automated and efficient the process is and is described in table 3:

1. **Integration of Azure DevOps:** The Azure DevOps pipeline integrates the model by using YAML configurations. The trained model is saved to disk, and the pipeline will loop through automated steps: if new data is available, train the model again until that point.
2. **Integrate AWS CodePipeline:** AWS code pipeline is used to execute the ML model-training script, this is done on events like new deployment or collecting new data Based on the nature of training task resources, model can be deployed, on AWS Elastics Container Service or EC2. The integration keeps cloud performance predictions always in sync.
3. **Automation Efficiency:** The time taken to trigger and run the model's training script is measured to evaluate the efficiency of the CI/CD pipeline. Shorter training times ensure that predictions are available quickly, aiding faster decision-making processes (Chatterjee and Mittal 2024).

Metric	AWS	Azure	Observations
Pipeline Deployment Time	1 min 34 secs (Based on AWS CodePipeline logs)	2 min 45 secs (Based on Azure Pipelines)	AWS showed faster deployment times.
Resource Utilization	Efficient with Auto-scaling (Elastic Beanstalk)	Better resource tracking (Azure Monitor)	Azure excels in resource management.
Integration with ML Models	Seamless (AWS SageMaker, CodePipeline)	Effective but tightly integrated with Microsoft tools (Azure ML Studio)	Both integrate well with ML models, but Azure works best in Microsoft environments.
CI/CD Pipeline Setup Time	Faster automation and integration	More steps required for integration	AWS setup was simpler for automation.
Cost Efficiency	Slightly higher costs for continuous deployments	Lower costs for certain configurations	Depends on workload and configuration.

Table 3: Metrics to measure the performance

6.4 Reporting and Insights

Following the review of the model's performance and its incorporation into the CI/CD pipeline, the findings are examined to offer actionable insights:

1. **Cloud Platforms:** The analysis has a concise comparison of both AWS and Azure based on deployment time, cost, resource usage and user satisfaction. This comparison enables

businesses to choose between the platforms according to their specific requirements, based on data.

2. **Abstract Cost Optimization:** Based on the model results, which cloud platform offers the best cost-efficiency for multiple use cases It might show, for instance, that Azure gives better performance at a lower price for certain workloads, or the same for AWS.
3. **User Feedback:** User feedback is included in the evaluation, which rates how well end-users feel each platform performed. Measuring user satisfaction helps pinpoint friction points and opportunities for improving platform design and services.

Monitoring tools used like AWS CloudWatch and Azure Monitor, refer to the figures 11 and 12:

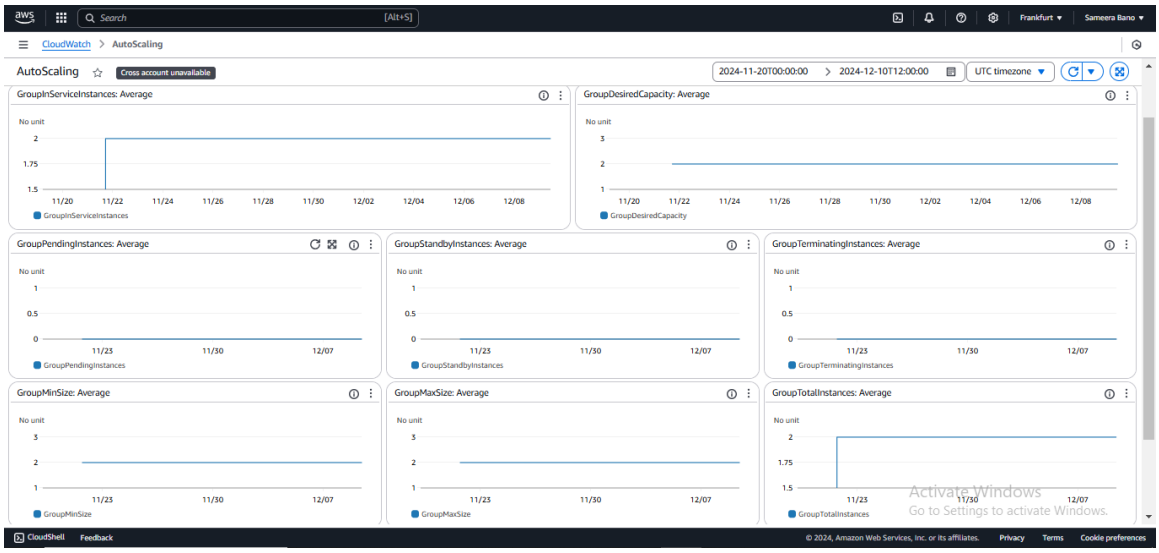

Figure 11 Aws Monitoring Log

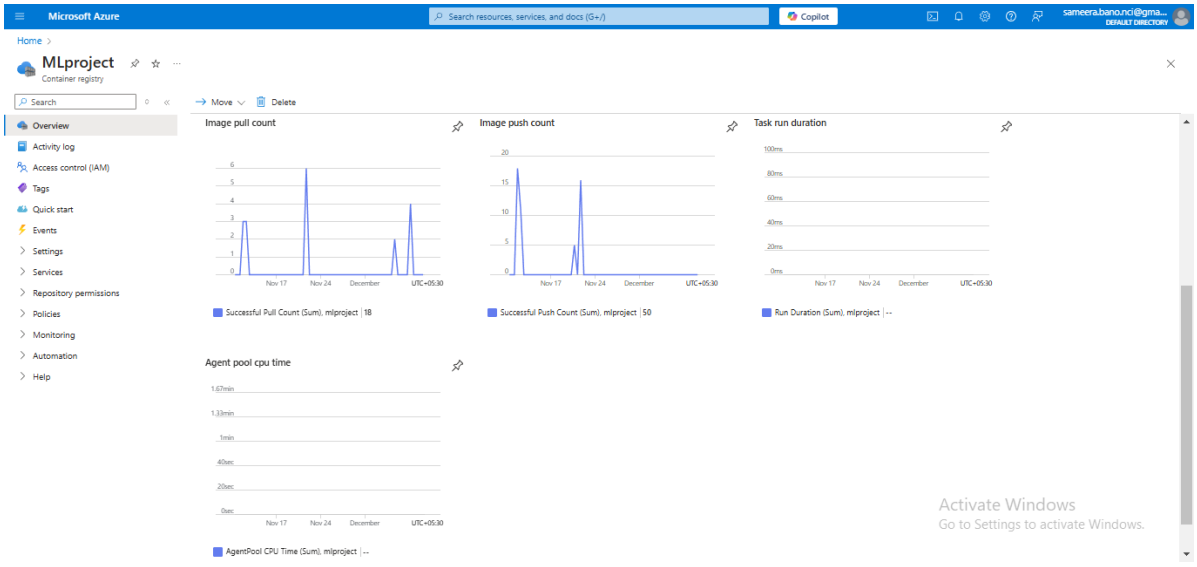

Figure 12 Azure Monitoring Log

7. Conclusions and Future Work

This study introduced comparative analysis for two cloud environments for AWS and Azure based on machine learning model evaluation of vital measures of the cloud

environment including deployment time, cost, resource utilization, and user satisfaction. The main concern here was to know using these metrics, which platform tends to rule the roost and how businesses can use this knowledge to select a productive cloud platform for them.

Using metrics like RMSE and R^2, it was shown that AWS vs Azure vs GCP all have unique strengths based on their use case. Azure was cost-effective for their workload, AWS had faster deployment times for other workloads. Analysis of user satisfaction further highlighted the mixed feelings users had about each platform, as they performed well in different areas of user satisfaction. These takeaway points are in line with the increasing need for organizations to evaluate cloud platforms for performance, not just cost or service features. Additional research could further investigate the inclusion of other performance indicators besides cost, such as security and scalability, to facilitate a broader comparison between AWS and Azure.

The ML model and CI/CD integration strategy has relevance from a commercialization standpoint where it can be commercialized to offer this as a SaaS product to any businesses looking to optimize cloud platform making it a significant tool for businesses to keep analyzing cloud operation performance continuously.

References

Bafana, M. and Abdulaziz, A., 2024. DevSecOps in AWS: Embedding Security into the Heart of DevOps Practices. Asian American Research Letters Journal, 1(1).

Mathew, A., Andrikopoulos, V. and Blaauw, F.J., 2021, December. Exploring the cost and performance benefits of AWS step functions using a data processing pipeline. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing (pp. 1-10).

Borra, P., 2024. Advancing Artificial Intelligence with AWS Machine Learning: A Comprehensive Overview. International Journal of Advanced Research in Science, Communication and Technology (IJARSCT) Volume, 4.

Kreuzberger, D., Kühl, N. and Hirschl, S., 2023. Machine learning operations (mlops): Overview, definition, and architecture. IEEE access, 11, pp.31866-31879.

Boscain, S., 2023. AWS Cloud: Infrastructure, DevOps techniques, State of Art (Doctoral dissertation, Politecnico di Torino).

Leite, L., Rocha, C., Kon, F., Milojicic, D. and Meirelles, P., 2019. A survey of DevOps concepts and challenges. ACM Computing Surveys (CSUR), 52(6), pp.1-35.

El Aouni, F., Moumane, K., Idri, A., Najib, M. and Jan, S.U., 2024. A systematic literature review on Agile, Cloud, and DevOps integration: Challenges, benefits. Information and Software Technology, p.107569.

Joshi, P.K., 2021. CI/CD Automation for Payment Gateways: Azure vs. AWS. ESP Journal of Engineering & Technology Advancements (ESP JETA), 1(2), pp.163-175.

Akinleye, D., 2024. Performance Metrics for Evaluating Pipeline Efficiency.

Tamanampudi, V.M., 2019. Automating CI/CD Pipelines with Machine Learning Algorithms: Optimizing Build and Deployment Processes in DevOps Ecosystems. Distributed Learning and Broad Applications in Scientific Research, 5, pp.810-849.

Banala, S., 2024. DevOps Essentials: Key Practices for Continuous Integration and Continuous Delivery. International Numeric Journal of Machine Learning and Robots, 8(8), pp.1-14.

Brooker, M., Danilov, M., Greenwood, C. and Piwonka, P., 2023. On-demand Container Loading in {AWS} Lambda. In 2023 USENIX Annual Technical Conference (USENIX ATC 23) (pp. 315-328).

Palumbo, F., Aceto, G., Botta, A., Ciuonzo, D., Persico, V. and Pescapé, A., 2021. Characterization and analysis of cloud-to-user latency: The case of Azure and AWS. Computer Networks, 184, p.107693.

Xu, R., 2020. A design pattern for deploying machine learning models to production.

Sharma, A. and Sandhu, R., 2022. Serverless Application on AWS.

Daniel, S., Brightwood, S. and Oluwaseyi, J., 2024. Cloud-based big data analytics (aws, azure, google cloud).

Xhepa, M. and Kanakala, N.S., 2022. Machine Learning Model Computation in AWS and Azure.

Botchkarev, A., 2018. Evaluating performance of regression machine learning models using multiple error metrics in azure machine learning studio. Available at SSRN 3177507.

Roy, A., Qureshi, S., Pande, K., Nair, D., Gairola, K., Jain, P., Singh, S., Sharma, K., Jagadale, A., Lin, Y.Y. and Sharma, S., 2019. Performance comparison of machine learning platforms. INFORMS Journal on Computing, 31(2), pp.207-225.

Chatterjee, P.S. and Mittal, H.K., 2024, April. Enhancing Operational Efficiency through the Integration of CI/CD and DevOps in Software Deployment. In 2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT) (pp. 173-182). IEEE.

Rocha, Á., Adeli, H., Dzemyda, G., Moreira, F., & Poniszewska-Marańda, A. (2024). Logistic Regression and DevOps Integration. In Advances in Computing and Data Sciences.

Singh, A., & Aggarwal, A. (2024). Random Forest Models. In Advances in Computing and Data Sciences.

BrowserStack. (2023). DevOps Lifecycle. BrowserStack.

Amazon Web Services. (2024). Well-Architected Machine Learning Lifecycle. Amazon Web Services.