REAL TIME ANOMALY DETECTION FOR CONTAINERIZED ENVIRONMENTS

Nirothipan Megalingham
219375K

Degree of MSc in Computer Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

June 2024

REAL TIME ANOMALY DETECTION FOR CONTAINERIZED ENVIRONMENTS

Nirothipan Megalingham
219375K

Thesis submitted in partial fulfillment of the requirements for the degree
Degree of MSc in Computer Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

June 2024

DECLARATION

I declare that this is my own work and this Thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date: 01/09/2024

The supervisor should certify the Thesis with the following declaration.

The above candidate has carried out research for the Degree of MSc in Computer Science Thesis under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. Sunimal Rathnayake

Signature of the Supervisor: Date: 23/09/2024

ACKNOWLEDGEMENT

I extend my deepest and most sincere gratitude to Dr. Sunimal Rathnayake of the University of Moratuwa, Faculty of Computer Science and Engineering, for his invaluable mentorship during this research endeavor. Without his unwavering support, navigating through this journey would have been nearly insurmountable. Dr. Rathnayake not only provided continuous guidance but also mentored me during the thesis passing the time zone barriers, offering assistance remotely and helped me maintain the momentum during the challenging phases.

A heartfelt thank you is also due to my family for their enduring patience and encouragement throughout my research program. Their unwavering support has been a constant source of strength, from the inception of my studies to the culmination of this endeavor. Furthermore, I express my deepest appreciation to my parents for their steadfast support throughout my academic journey.

I am also indebted to the University of Moratuwa for granting me the opportunity to participate in the MSc program and for furnishing the requisite resources to see this research to fruition. Additionally, I am grateful to my former workplace, WSO2, for accommodating my pursuit of this part-time MSc and research alongside my professional responsibilities. Their support has been indispensable in achieving this milestone.

ABSTRACT

Anomalies in containerized environments pose a significant threat, given their potential to escalate small failures into catastrophic outcomes. These anomalies, which can manifest in various forms, possess the capability to disrupt service level agreements and tarnish an organization's reputation irreversibly. Thus, it becomes imperative to detect and address these anomalies promptly to minimize their adverse effects on business operations. In this study, we delve into the landscape of anomalies prevalent in containerized environments, focusing on understanding their diverse nature and the substantial impact they can have.

In this comprehensive survey, our focus lies in the real-time detection of anomalies within Kubernetes environments, a critical aspect in ensuring the robustness and stability of modern containerized systems. To achieve this, we conducted an extensive literature review, delving into existing research and methodologies pertinent to anomaly detection within Kubernetes ecosystems.

Furthermore, we conducted practical experiments by deploying applications on Azure Kubernetes Services, leveraging the inherent Kubernetes metrics API and Prometheus API to gather pertinent data. Employing sophisticated feature selection techniques, we curated datasets and trained a decision tree model capable of discerning anomalous patterns in real-time metrics streams. Through rigorous experimentation, we validated the efficacy of our approach, achieving a remarkable accuracy rate of 95% in anomaly prediction. This research underscores the significance of real-time anomaly detection in Kubernetes environments and offers tangible insights for enhancing the resilience of containerized infrastructures.

Keywords: Real Time Anomaly Detection, Machine Learning, Containerization, Kubernetes

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
AKS	Azure Kubernetes Service
CICD	Continuous Integration and Continuous Deployment
HHMM	Hierarchical Hidden Markov Models
K8s	Kubernetes
LSTM	Long short-term memory
MSA	Microservice Architecture
NFV	Network Function Virtualization
S3	Amazon Simple Storage Service
VM	Virtual Machines

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Terminologies

- **Containerization:** The process of packaging applications and their dependencies into containers that can run consistently across different environments.
- **Anomaly detection:** The process of identifying unexpected or abnormal behavior in a system, such as a spike in resource usage or a sudden increase in errors.
- **Docker:** A popular open-source platform that enables the creation, deployment, and management of containers.
- **Kubernetes:** An open-source platform that automates the deployment, scaling, and management of containers.
- **Machine Learning** Machine learning is a field of AI that teaches computers to learn from data and make predictions based on patterns. It allows computers to make decisions without explicit programming. Different types of machine learning exist for various data types and use cases. It is used in image recognition, speech recognition, natural language processing, and predictive modeling.
- **Cloud Infrastructure** Cloud infrastructure refers to the hardware and software components required to deliver cloud computing services. It typically includes a network of servers, storage systems, and databases, as well as virtualization software, security measures, and management tools, all housed in data centers and accessed over the internet.
- **Decision Tree** A decision tree is a hierarchical model used in machine learning for classification and regression tasks. It partitions the input space into regions based on feature values and assigns a predictive value to each region. Decision trees are composed of nodes representing decision points and edges representing possible outcomes, making them interpretable and easy to visualize.

In the context of this research, it is important to clarify the distinction between anomaly detection and prediction. Anomaly detection refers to the process of identifying data points or patterns that deviate significantly from the expected behavior within a system, such as a sudden spike in CPU usage or a network latency anomaly. This approach focuses on recognizing irregularities or outliers in real-time as they occur, enabling immediate response to potential issues. On the other hand, prediction involves forecasting future states or events based on historical data, such as predicting the future load on a server based on past trends.

This research specifically focuses on anomaly detection rather than prediction. We are developing a system that continuously monitors containerized environments to detect anomalies as they happen. The goal is to identify unexpected behaviors that could indicate performance degradation, security breaches, or other operational issues in real-time, allowing for prompt corrective actions to maintain system stability and reliability.

1.2 Overview

This chapter provides an overview of containers, their importance in modern application deployment and management, and the concept of containerization. It highlights the advantages of using containers, such as increased efficiency and portability. The chapter also addresses the challenges and anomalies that can occur in containerized environments, such as resource contention, network issues, and security breaches.

1.3 Background

Containerization is a widely adopted technology in the software development industry due to its numerous benefits. The main reason for its popularity is the ability to isolate applications from one another and from the underlying system. This isolation makes it easier to manage dependencies and avoid conflicts between different applications. Additionally, containers can be easily moved from one environment to another, making it simple to deploy and manage applications across various platforms. This makes it an ideal solution for organizations that need to quickly scale their applications to

meet changing demands. Furthermore, containers are designed to be lightweight and efficient, making it possible to run many containers on a single machine. This increases resource utilization, which reduces costs and improves performance. Overall, containerization provides an efficient and scalable solution for managing applications, making it a popular choice for organizations of all sizes.

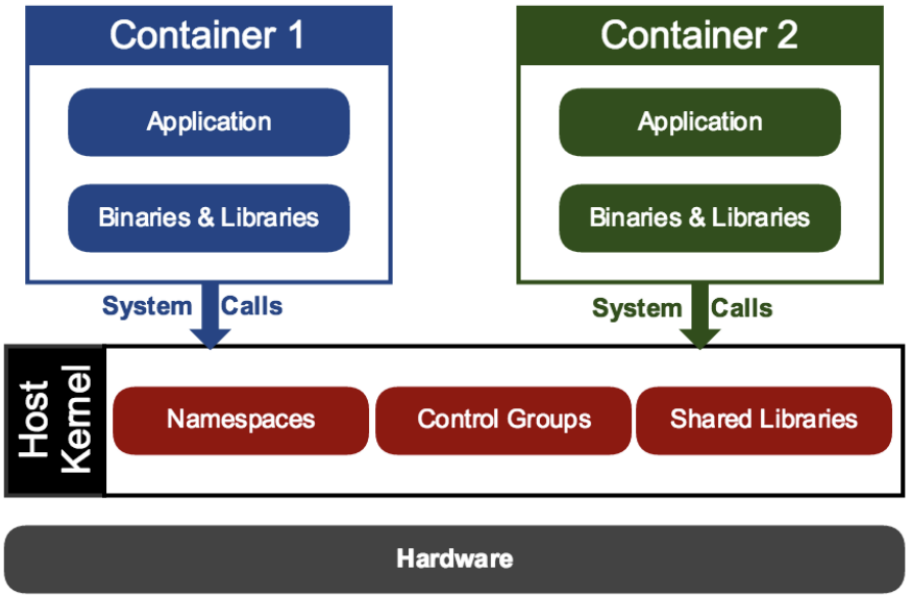


Fig. 1.1: Containers provide encapsulation environment for applications and dependencies

The modern applications are mostly deployed in containers or virtual environments. Containerization is a method of packaging software in a way that allows it to run consistently across different environments. It is a form of operating system virtualization that allows multiple isolated applications to run on a single host system, by providing each application with its own lightweight, self-contained environment called a container.

Containers are built from images, which are snapshots of the entire file system and runtime environment of an application, including the application code, libraries, and system dependencies. Because images are self-contained, they are highly portable and can be easily moved between different environments, such as development, testing, and production. This makes containerization a popular option for deploying and managing microservices, and for developing and testing software in a consistent and repeatable

way.

Monitoring containers for performance is important because it allows for early detection of any issues that may negatively impact the performance of the containerized applications, such as resource constraints, network problems, or other types of errors. This information can then be used to make adjustments to the container configuration, or to the underlying infrastructure, to improve the performance of the applications and to ensure that they are running smoothly. Additionally, monitoring can help identify and diagnose issues that may lead to downtime or data loss. In short, monitoring containers for performance can help to ensure that the containerized applications are operating as expected and that any issues are identified and addressed quickly.

1.4 Motivation

The container environments are susceptible to different anomalies. Anomalies in a container environment refer to any deviations from the expected behavior of containers, applications, and the systems they run on. Some common examples of anomalies in a container environment include:

1. **Resource utilization spikes:** A sudden increase in resource usage, such as CPU, memory, or network bandwidth, can indicate an anomaly, especially if it is not caused by a known event.

2. **Application crashes:** Unexpected crashes or errors in applications running in containers can be a sign of an anomaly, such as a bug or a conflict with another application.

3. **Network connectivity issues:** Slow or lost network connectivity, such as a drop in network traffic or slow response times, can be a sign of an anomaly, such as a network configuration issue or a security issue.

4. **Container failures:** Unexpected failures or crashes of containers, such as failing to start or stop, can indicate an anomaly in the system, such as a problem with the host machine or a problem with the container configuration.

These anomalies can have various root causes and can affect the performance, reliability, and security of the container environment. It is important to monitor and detect these anomalies so they can be resolved quickly to maintain the stability and efficiency of the system.

1.5 Research Problem

Different anomalies, which occur with the containerized applications, affect the performance and other multiple factors of the application. Anomalies in containerized applications are a significant concern for organizations as they can negatively impact the performance, reliability, and security of applications. Anomalies can take many different forms and can be caused by a range of factors, including software bugs, system configurations, resource limitations, or external factors such as network issues.

1. Errors or exceptions are a common type of anomaly in containerized applications. They are typically caused by unexpected conditions within the system and can cause the application to fail or behave in unexpected ways. Errors can also lead to resource leaks or other performance issues, making it crucial to detect and resolve them as soon as possible.

2. Unexpected or unusual system performance can also indicate an anomaly in a containerized application. Slow response times or high resource usage can be caused by various factors, such as resource constraints, incorrect configurations, or issues with the application code itself. Monitoring and analyzing system performance metrics can help identify these anomalies and diagnose their root cause.

3. Inconsistencies in the data processed by the application can also be indicative of an anomaly. This can result in incorrect results being returned or can impact the accuracy and reliability of the application as a whole. To address these anomalies, it is important to implement robust data validation and integrity checks to ensure the consistency of data processed by the application.

4. Finally, security vulnerabilities or breaches within the system are another type of anomaly that can pose a significant risk to organizations. This can include issues such as unauthorized access to sensitive data, malicious attacks on the system, or other security-related incidents. To address these anomalies, it is crucial to implement security best practices and to continuously monitor the system for potential security threats.

In conclusion, anomalies in containerized environments can take many forms and can have a significant impact on the performance, reliability, and security of applications. Therefore, it is crucial to implement robust and effective anomaly remediation systems to detect and resolve these anomalies in real-time.

1.6 Research Questions

Anomaly detection in containerized environments presents significant challenges and opportunities for enhancing the reliability and performance of applications. This research aims to address these challenges through the following research question:

1.6.1 Questions

How can a robust and efficient framework be developed for real-time anomaly detection in containerized environments using machine learning techniques?

To support this primary inquiry, the study explores several specific sub-questions:

1. **Study of Anomalies**

    - **What types of anomalies commonly occur in containerized environments?**
    - **How do these anomalies impact the performance and reliability of applications?**

    The research will begin by identifying and understanding the various anomalies that can affect containerized applications, focusing on their frequency and impact.

2. **Study of Anomaly Detection Strategies**

    - **Which statistical and machine learning strategies are most effective for detecting anomalies in containerized environments?**
    - **What preprocessing steps are necessary to enhance the performance of these anomaly detection models?**

    This segment will explore various anomaly detection methods, including statistical approaches, machine learning algorithms, and hybrid techniques. The study will also investigate the importance of data preprocessing.

3. **Anomaly Detection Framework**

    - **How can a scalable and integrative framework for real-time anomaly detection be modeled for containerized environments?**

    The research aims to develop a framework that can seamlessly integrate with existing systems and scale to manage large containerized environments efficiently.

4. **Data Collection and Model Building**

    - **What are the best practices for collecting and simulating data to train an anomaly detection model in a containerized environment?**

    Data collection and simulation are critical for training effective models. This question will guide the processes of gathering data from Prometheus, Kubernetes APIs, and other sources.

5. **Real-Time Anomaly Detection**

    - **How can real-time data be processed and analyzed to detect anomalies proactively in containerized environments?**

    This question focuses on the deployment of trained models as Python microservices to analyze real-time data and detect anomalies promptly.

1.6.2 Rationale

The research questions are designed to systematically address the complexities of anomaly detection in containerized environments. By examining both the types of anomalies and the detection strategies, the study aims to build a comprehensive understanding that informs the development of a scalable, integrative framework. Collecting and simulating data will ensure the model is well-trained and effective, while the focus on real-time detection aims to provide practical, actionable insights for maintaining application performance and reliability.

1.6.3 Scope and Boundaries

The research will focus exclusively on containerized environments using technologies such as Docker and Kubernetes. It will explore both statistical and machine learning-based anomaly detection strategies, but it will not extend to non-containerized virtualization technologies or unrelated anomaly detection techniques.

1.6.4 Connection to Research Objectives

The research questions are directly aligned with the objectives of studying anomaly types, exploring detection strategies, developing a real-time detection framework, and ensuring practical application through robust data collection and model deployment. These questions will guide the research design, methodology, and analysis to achieve the goal of creating an efficient real-time anomaly detection system for containerized environments.

1.7 Research Objectives

Anomaly Detection in Containerized Environments is a crucial area of research that aims to tackle the challenges of detecting and diagnosing anomalies in containerized environments. The following are the objectives of this research:

1. **Study of Anomalies:** The first objective of this research is to study the different types of anomalies that can occur with applications in a containerized environ-

ment. This involves identifying the most common anomalies and understanding their impact on the performance and reliability of applications.

2. **Study of Anomaly Detection Strategies:** Anomaly detection strategies encompass a variety of approaches aimed at identifying unusual patterns or outliers in data. These strategies typically involve statistical analysis, machine learning algorithms, or a combination of both. Statistical methods, such as z-score analysis or moving averages, focus on deviations from expected behavior based on historical data. Machine learning techniques, such as clustering, classification, or autoencoders, leverage patterns and relationships within the data to detect anomalies. Hybrid approaches may integrate multiple methods to enhance detection accuracy and robustness. Additionally, anomaly detection strategies often involve preprocessing steps such as feature engineering, normalization, or dimensionality reduction to improve model performance. Ultimately, the choice of strategy depends on the characteristics of the data, the nature of the anomalies, and the specific requirements of the application.

3. **Anomaly Detection Framework:** Another objective of this research is to model a framework that can detect anomalies in real-time. This framework should be able to integrate with existing systems and be scalable to accommodate large containerized environments.

4. **Data Collection and Model Building:** Data collection entails gathering data from Prometheus and Kubernetes' built-in metrics APIs. Additionally, anomalies are simulated through external APIs and various sources to enrich the dataset. This collected data serves as the foundation for training the anomaly detection model.

5. **Real Time Anomaly Detection:** For real-time anomaly detection, data is collected from Prometheus and Kubernetes' built-in metrics APIs. This data is then fed into the trained model deployed as a Python microservice. The model analyzes the data to detect anomalies, enabling proactive identification of abnormal system behavior.

By achieving these objectives, this research aims to develop a robust and efficient real-time anomaly detection system for containerized environments that can detect anomalies in real-time with minimal human intervention.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

There have been multiple studies in the areas of performance monitoring of applications in the past in containerized environment. This chapter will provide an insight into the previous studies carried out in the field of monitoring applications and anomaly detection. This chapter has been divided based on the anomaly detection techniques which has been suggested on the studies.

Many studies have examined how anomaly detection methods can meet the specific needs of cloud-based networks. A recent comprehensive review of this field provides a systematic evaluation of 215 publications, representing the last decade of scientific development in anomaly detection[3]. This review categorizes methodologies into three main areas: machine learning, deep learning, and statistical approaches, summarizing how these techniques are employed for anomaly detection. Furthermore, it identifies three primary application areas—intrusion detection, performance monitoring, and failure detection—each with its sub-areas, and highlights the public datasets commonly used for evaluations.

In the context of anomaly detection, anomalies can be categorized based on their characteristics. For example, a collection of related data instances that are anomalous compared to the rest of the dataset is termed a collective anomaly. Approaches to anomaly detection vary in whether they rely on labeled data or if they operate independently of such prior knowledge, focusing instead on the inherent similarities and dependencies in the data to identify normal structures and deviations. The comprehensive review analyzed 215 papers from three different scientific databases, providing detailed insights into the application of various anomaly detection techniques in cloud computing environments [3].

2.2 Anomaly Detection with Machine Learning or Modeling Techniques

The usage of cloud environments, particularly OpenStack, is increasing across various fields due to its ability to improve service availability and reduce costs. One significant challenge addressed in these environments is anomaly detection through machine learning techniques such as LSTM models. In 2021, Rao [1] explored anomaly detection in cloud environments using artificial intelligence techniques. Specifically, Network Function Virtualization Network Function Virtualization (NFV) is used to virtualize network services on proprietary hardware. Researchers have leveraged LSTM models to develop automatic detection systems for anomalies. In experiments, Stress-ng commands were employed to induce stress on virtual machines for failure prediction. The proposed LSTM model achieved detection accuracies of 94.61% for the training set and 93.98% for the test set. Unlike traditional NFV monitoring approaches that rely on fixed rules and thresholds for system metrics such as CPU and memory, this method uses multivariate time series data to predict node failures. The experimental setup included a dataset with nine features and a multi-layer model with stacked bidirectional Long short-term memory Long short-term memory (LSTM) layers, tested in an OpenStack cloud environment comprising eight virtual machines and three physical machines, demonstrating high accuracy with various loss functions.

The paper "A Spatiotemporal Deep Learning Approach for Unsupervised Anomaly Detection in Cloud Systems" by Zilong He et al., published in IEEE Transactions on Neural Networks and Learning Systems [2], introduces TopoMAD, an unsupervised anomaly detection model tailored for cloud environments. TopoMAD leverages the topological information of cloud systems using a combination of graph neural networks (GNN) and long short-term memory networks (LSTM) within a variational auto-encoder (VAE) framework, allowing it to capture both spatial and temporal dependencies in the data. This method does not rely on labeled training data, making it suitable for real-world applications where such data is often unavailable. The model is robust, capable of training on contaminated data, and has demonstrated superior performance compared to state-of-the-art methods. Evaluations were conducted us-

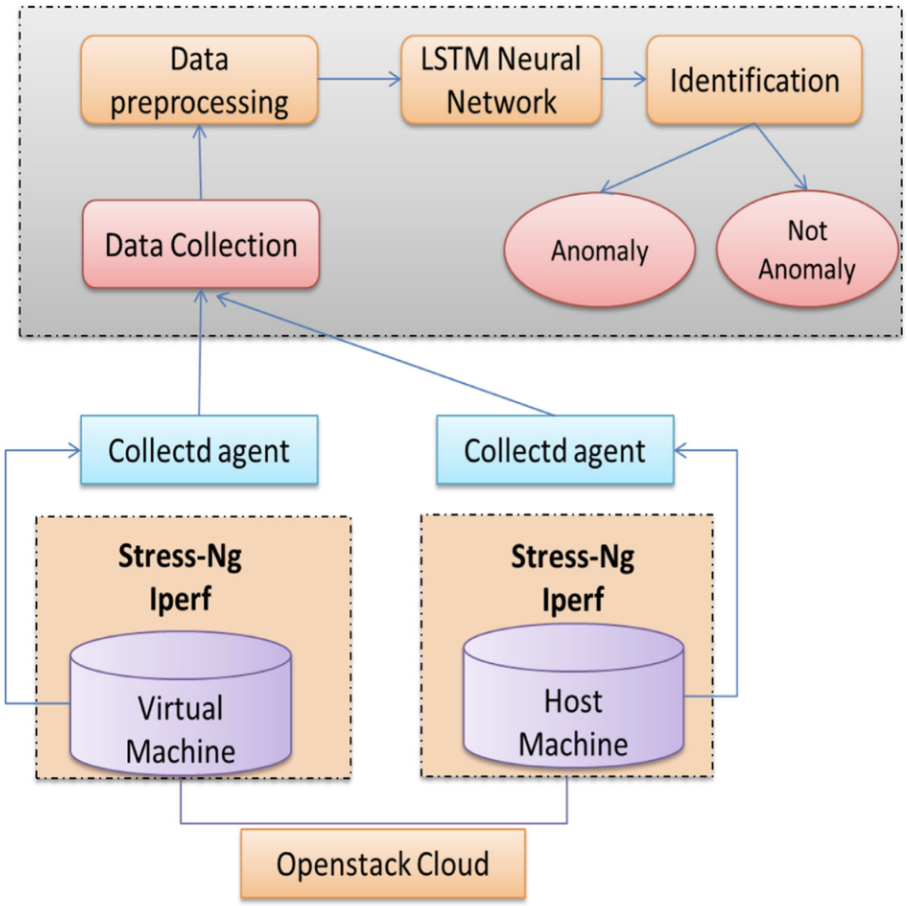


Fig. 2.1: Overall Methodology Using LSTM by Rao[1]

ing runtime performance data from big data batch processing and microservice-based systems, with the authors providing open-source datasets for further research. The study highlights the effectiveness of TopoMAD in complex cloud environments and suggests future work on online learning techniques and the integration of additional deep anomaly detection methods.

In 2018 Kardani et al [4] conducted a study on the Performance-Aware Management of Cloud Resources. The focus of the study was on the difficulties faced in managing distributed resources and maintaining quality of service in the dynamic environment of the cloud. The study emphasized the importance of performance-aware resource management methods, as opposed to traditional static capacity planning solutions. The paper discussed the significance of continuous monitoring of system attributes and performance metrics in detecting performance-related issues in applica-

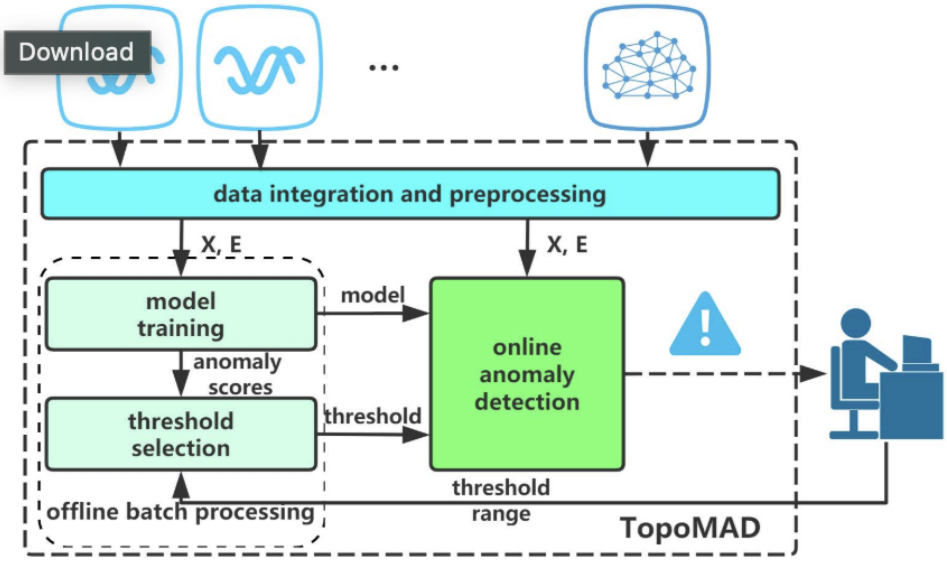


Fig. 2.2: Overall structure of TopoMAD[2]

tions and the utilization of data analytic such as statistical and machine learning approaches to analyze data and enhance system performance. In this study, a taxonomy of existing research in cloud resource management was presented and gaps for future research were identified.

Carla Sauvanaud et al [5] conducted a study in 2018 on anomaly detection and diagnosis in cloud services. The study aimed at developing a monitoring module for virtual machine (VM) applications that leverages both hypervisor monitoring and operating system (OS) monitoring. The hypervisor monitoring provided information on each individual VM, including its memory utilization, while the OS monitoring offered in-depth information about each application running within the VM. In order to gather training data, the research team added a fault injection component to the system. The collected data was then used in a supervised machine learning process. This study was particularly noteworthy for its broad range of error types included in the training, such as CPU, memory, disk, and network errors, leading to a more precise diagnostic output for the end user.

In 2019, a research team led by Chin-Wei Tien et al [6] from the Cybersecurity Technology Institute at the Institute for Information Industry Taipei Taiwan reported on a study focused on enhancing the security of containerized environments using

the Kubernetes platform. The team introduced KubAnomaly, a system designed to enhance security monitoring through anomaly detection within the Kubernetes orchestration platform. Utilizing neural network techniques and a container monitoring module for Kubernetes, the system effectively identified irregular behaviors, such as web service attacks. The study tested KubAnomaly on three different datasets and benchmarked its performance against other machine learning algorithms. Findings revealed that KubAnomaly achieved a 96% overall accuracy in anomaly detection and successfully identified four genuine hacker attacks from September 2018. The researchers suggested that using fully labeled data in future studies could improve accuracy. Additionally, they plan to integrate KubAnomaly with 5G network function virtualization platforms to bolster its cybersecurity capabilities.

The virtualization of computing resources through containers has been a focal point of recent research, particularly in the context of intrusion detection systems (IDS) for multi-tenant applications. Various machine learning classifiers, including AdaBoost, Multi-layer Perceptron (MLP), Support Vector Machine (SVM), and Random Forest, have been evaluated for their efficacy in this domain. Notably, while AdaBoost, MLP, SVM, and Random Forest classifiers achieved perfect AUC scores of 1.0, the MLP classifier demonstrated the highest F-Measure at 99.6%, closely followed by AdaBoost and SVM with F-Measures of 99.4% [7].

The study further investigated the implementation of the Bag of System Calls (BoSC) algorithm alongside a sliding window technique to assess eight machine learning algorithms for classifying system calls as benign or malicious. The findings indicated that the Decision Tree and Random Forest algorithms delivered the highest F-Measure of 99.8%, especially with a sliding window size of 30 combined with the BoSC algorithm. Moreover, an analysis of resource usage showed that despite both algorithms achieving high accuracy, the Decision Tree algorithm outperformed Random Forest in terms of speed and efficiency, using less CPU and memory. This indicates a significant advantage in deploying Decision Tree algorithms for IDS in containerized environments.

There have been several studies done based on neural networks for anomaly detection, for instance studies based on deep neural networks [8], [9], using one-class neural

networks [10], [11]. A research team from Furtwangen University, led by Holger Gantikow (2020), explored container anomaly detection using neural networks to analyze system calls[12]. Containers, known for their workload isolation and efficiency compared to virtual machines, sometimes face issues such as temporary directory problems in FLUKA containers. The study integrates file system path analysis with system call distribution analysis to prevent mimicry attacks. Anomaly detection using system call data has been a topic of research since the mid-1990s, and this study combines various models for improved detection. Evaluations using OpenFOAM and FLUKA applications with anomalies like cryptocurrency mining and ransomware demonstrated the effectiveness of the techniques. Future work aims to analyze the performance impact of tracing, optimize strategies using system call filters, and investigate the approach's applicability to additional workloads and anomalies, including distributed systems and performance variations in system call distribution.

A research team led by Yuhang Lin at North Carolina State University (2020) developed CDL, a framework designed to efficiently detect security attacks in containerized applications[13]. CDL addresses the challenge of limited training data for dynamic containers by integrating online application classification and anomaly detection, thereby reducing false positives and improving detection rates. Containers are popular in distributed computing due to their efficiency and low isolation overhead. The study found that CDL could detect 31 out of 33 attacks with low false positives. Implemented and tested on real-world vulnerabilities, CDL significantly reduced false positives and increased true positives compared to traditional methods. The framework enhances real-time security attack detection in containerized applications, with the random forest model being used for accurate application classification by reducing over-fitting errors.

A group led by Mohammad Islam at Toronto Metropolitan University (2021) [14] developed an automated monitoring system using deep learning neural networks to detect anomalies in real-time, addressing the challenges faced by Cloud service providers due to the increasing popularity of Cloud computing. The study focuses on the IBM Cloud platform and highlights the importance of proper monitoring for maintaining service reliability and uninterrupted operation. The proposed system utilizes a GRU-

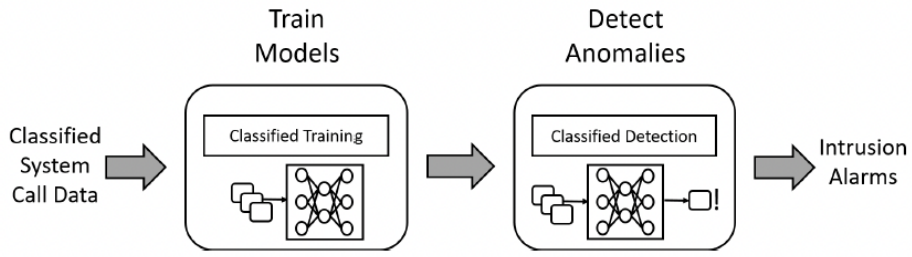


Fig. 2.3: Classified learning and anomaly detection of CDL.

based autoencoder model to detect anomalies in multi-dimensional time series data, sending alerts to DevOps teams via Slack channels. Emphasizing scalability, reliability, and interconnection feasibility, the system integrates with serverless architecture and third-party monitoring tools. The implementation in the IBM Cloud Platform has successfully improved detection accuracy and reduced false alarms, thereby freeing up human resources and mitigating the risk of Cloud outages. The study was conducted across 10 data centers.

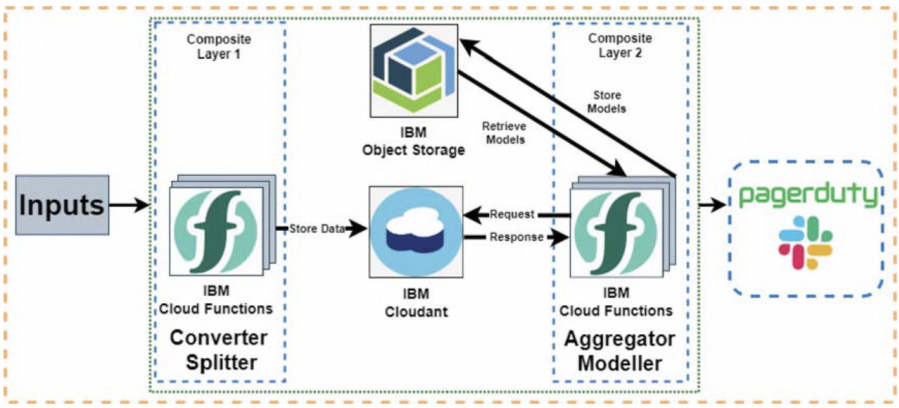


Fig. 2.4: The layered microservice-based architecture of the proposed solution.

Clinton Cao and a research group from Thales (France) (2022)[15] explored using state machine models to monitor and detect anomalies in Kubernetes (K8s) clusters. While neural networks are commonly used for anomaly detection, state machine models offer a more interpretable approach, particularly for understanding and detecting attacks in cloud environments running multiple microservice applications. Their novel passive-learning method for learning probabilistic state machines from NetFlow data

gathered from a K8s cluster achieved impressive results, with a balanced accuracy of 99.2% and an F1 score of 0.982. This approach successfully detected three attack scenarios. Additionally, they trained an isolation forest with 100 estimators on the same dataset, which helped to validate their findings. The study highlights the potential of state machine models in anomaly detection and discusses future research directions, including evaluating the generalizability of the approach to other systems and incorporating different types of log data for more accurate models.

Rupesh Karn and a research group at IBM (United States) (2021) [16] investigated the security threats posed by cryptomining malware targeting Kubernetes-managed containers in cloud computing. The study emphasizes the increasing use of containers, which run isolated systems on a single kernel, and the need for machine learning to detect and explain anomalies, aiding administrative decisions. They explored the use of explainable machine learning models, specifically focusing on syscall data collected from healthy and cryptomining-hijacked pods. The research demonstrated an automated pod anomaly detection setup within a Kubernetes cluster, finding that tree decision models provided the highest accuracy in detecting cryptomining applications. The system achieved over 78% aggregate anomaly prediction accuracy using syscall n-grams as features. The study also highlighted the importance of explainability, using visualizations to compare normal and compromised applications, which assists system administrators in making informed decisions. Challenges in implementing and managing rules in legacy cloud systems were also discussed, underscoring the role of machine learning in managing infected containers.

2.3 Anomaly Detection with Statistical Model

In 2016, Wang et al [17] conducted a study to address the challenges faced by web applications deployed on public cloud computing platforms, Wang and his colleagues proposed a Fault (F) Diagnosis (D) framework for Web applications in Cloud (C) Computing, referred to as FD4C. The authors introduced an online incremental clustering method to capture variations in workloads and identify access behavior patterns. The framework utilizes canonical correlation analysis to model the correlations between

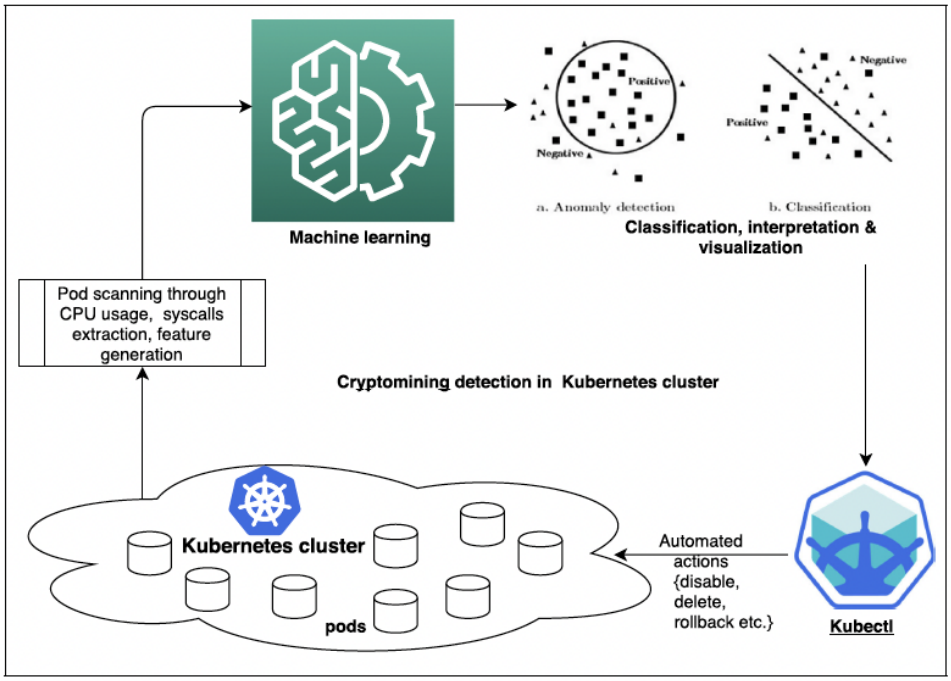


Fig. 2.5: Block diagram of cryptominer pod detection

workloads. The study involved 100 concurrent users. The results seem to reinforce existing knowledge in this field, with Wang stating that linear relationships exist among resource utilization metrics in each specific access behavior pattern. The proposed framework provides a useful solution for automatic fault diagnosis in web applications in cloud computing environments.

Du X and their team proposed [18], the use of time series analysis for detecting anomalies in a Microservice Architecture (MSA) (MSA). Their assumption is that, if the same microservice runs in multiple containers (which is a common scenario in this architecture), the performance data for each of these containers should be the same. To do this, they stored the time series performance data of each container in a database and used a data processing module to identify anomalous microservices. However, this approach has a limitation in that it assumes that only one container is anomalous at a given time and does not account for identifying anomalies if multiple containers are defective, which is a possible scenario in a Continuous Integration and Continuous Deployment Continuous Integration and Continuous Deployment (CICD) workflow.

Detecting and localizing performance anomalies in complex distributed systems,

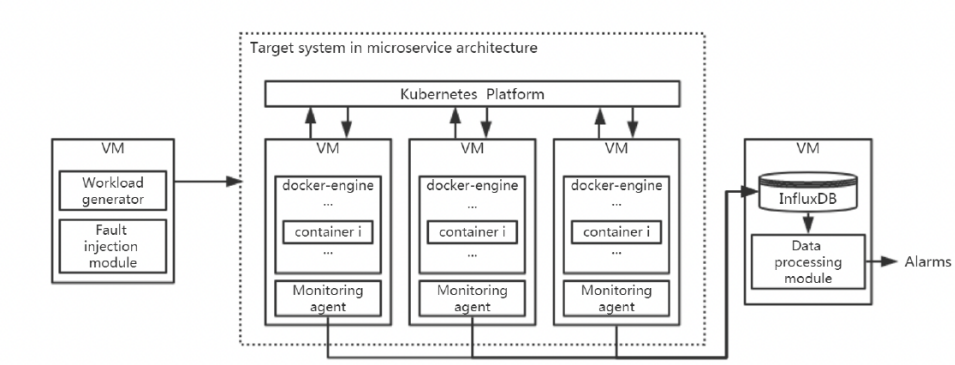


Fig. 2.6: Implementation of the Anomaly Detection System by Du X

especially those using short-lived container deployments, is essential for maintaining system operation. The framework utilizing hierarchical hidden Markov models Hierarchical Hidden Markov Models (HHMM) has proven effective in detecting and localizing anomalies to enhance system availability and performance. In "Detecting and Localizing Anomalies in Container Clusters Using Markov Models," Areeg Pahl (2020) addresses the challenges of monitoring and managing distributed software systems, focusing on detecting and identifying anomalous behavior in containers and nodes [19].

The research explores the application of HHMMs to measure response time and resource utilization, aiming to detect anomalies and pinpoint their root causes. The experimental results demonstrate the proposed framework's effectiveness in tracing anomalous behaviors[19]. Future work includes developing recovery actions for identified anomalies and expanding the framework to cover more anomaly scenarios and metrics while maintaining accuracy.

The study analyzed 1511 records and concluded that HHMMs improve root cause analysis in layered architectures compared to traditional Hidden Markov Models (HMMs). The authors acknowledge potential limitations and suggest that future discussions should address these. They recommend using HHMMs [19] to detect anomalous behavior in container and node workloads in managed distributed systems, emphasizing the importance of response time and resource utilization for reliability and performance management at runtime.

Dewi Sarno (2021) [20] investigated anomaly detection in port container handling

using control flow pattern analysis and fuzzy regression, which incorporates expert judgments. The study highlights that fuzzy regression outperforms support vector regression and multiple linear regression in detecting anomalies by achieving high sensitivity, specificity, and accuracy. This method is effective in preventing issues like fraud and inefficiency in business processes. The proposed approach involves a two-step method that significantly reduces false positives and false negatives.

2.4 Anomaly Detection with Feedback System

A study conducted by Thomas et al [18] in 2018 have created an anomaly detection system that is specifically designed for microservices running in containers. This system takes into account changes in the environment, which helps to eliminate false positive results and increase overall accuracy. The code was modified during the development process to include monitoring, and the Keiker tool was utilized to gather performance data. Additionally, the system includes an event-aware reviser that adjusts the anomaly threshold and reduces the number of false alarms.

In 2020, Areeg Pahl et al [21] proposed a self-healing model to auto-scale compute resources in a containerized cluster environment. The experiment platform consisted of multiple Virtual Machines (VM) (VM) running the Linux OS (Ubuntu 18.04.3 LTS x64) on a host PC with Intel i5 2.7GHz, 8 GB RAM, and 1 TB storage. Each VM was configured with 3 VCPU and 2 GB VRAM, and the virtual experiment was run on Xen 4.11.1. The self-healing model aimed to migrate overloaded containers and measure the performance of both the host and destination nodes. The recovery interval was set at 50 seconds to ensure that no problems would arise from any recovery actions taken. The study included 73 observations in the analysis.

2.5 Anomaly Detection by evaluating Infrastructure Utilization

The study conducted by Samir, Areeg, et al [22] in 2020, examines the methods for detecting anomalies in virtualized services by analyzing the performance of the infrastructure. The article focuses on three types of faults - CPU hog, Network Packet loss/latency, and performance anomalies due to workload congestion - and explains

how monitoring agents were installed on each virtual machine to collect data for anomaly detection. The article also compares containers, as a more lightweight form of virtualization, with traditional virtual machines and highlights their resource-saving advantages. Furthermore, the literature review explores how edge cloud infrastructure could benefit from containers to provide computational capabilities for IoT and other remote applications. The utilization measures, which indicate a resource's capacity in use, were discussed to understand resource workload and to prevent overloading. The results of the proposed architecture demonstrate its ability to detect and identify anomalous behavior with more than 96% accuracy, showing the solution's effectiveness.

2.6 Anomaly Detection using Rule Based System

In a recent study conducted by Shetty, Jyoti et al [23], the authors sheds light on the utilization of rule-based expert systems for enhancing the fault remediation process in cloud computing environments. The experiments carried out have shown the superiority of the proposed system in selecting the optimal remediation technique with minimal overhead and impact on the system compared to traditional methods. The research provides valuable insights into the development of efficient fault remediation systems for cloud computing and emphasizes the significance of adopting a structured approach to fault remediation in this field, with rule-based expert systems potentially playing a crucial role. The study adds to the existing literature on fault remediation in cloud computing and serves as a valuable resource for future researchers and practitioners.

A study led by Igor Kotenko et al. [24] explored an approach for detecting anomalies and attacks through the analysis of kernel logs obtained with enhanced Berkley Packet Filter (eBPF). The authors noted that the global market for application containers was valued at $2.1 billion in 2020, and was forecasted to reach $5 billion by 2023, growing at a rate of 33% annually. The authors proposed four methods for creating white and blacklists for detecting anomalies and attacks through eBPF kernel logs. In their study, the team used three datasets that included attacks and found that one of the main benefits of their approach is that the rules can be interpreted and adjusted

by operators, unlike machine learning models. They also stated their intentions to improve the proposed methods and enhance the accuracy and speed of anomaly and attack detection in the future.

2.7 Anomaly Detection using Forecasts

Mart et al. (2021) [25] present a new approach to detecting and alerting anomalies in systems. the authors explore the idea of observability and explain how it can be implemented in general and within the context of a Kubernetes Kubernetes (K8s) cluster. They noticed that current monitoring and anomaly detection methods require manual input from operations engineers in the form of hardcoded values. To address this limitation, they proposed a new solution that leverages forecasting instead. After evaluating various forecasting methods, they ultimately selected Facebook prophet due to its automated features. They then put forth a solution for forecasting Prometheus time series data that takes advantage of the scalability of Kubernetes.

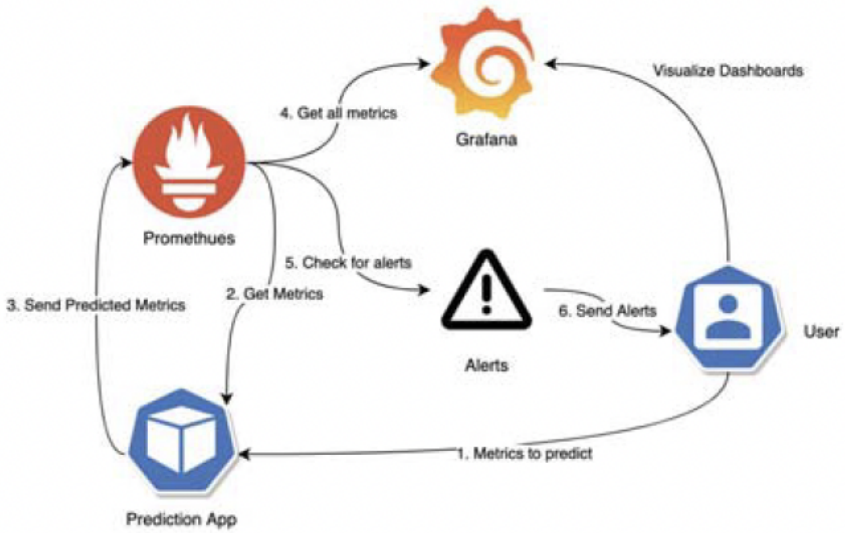


Fig. 2.7: Anomaly Detection workflow with Prometheus

2.8 Real Time Anomaly Detection System

ContainerGuard is a real-time detection system designed to identify Meltdown and Spectre attacks in container-based big data platforms while maintaining minimal performance overhead. A team from the Harbin Institute of Technology, led by Yulong Wang (2022), introduced ContainerGuard, highlighting its use of variational autoencoders to learn robust representations of normal patterns and ensemble networks to detect correlations between performance events [26]. This approach ensures excellent detection performance with minimal impact on system performance.

The paper reports extensive experiments conducted with 60 malicious samples to validate the effectiveness of ContainerGuard [26], focusing on Flush+Reload attacks. The authors proposed EnsembleVAE as an effective anomaly detection method and demonstrated that the system could be integrated into container-based platforms without requiring hardware or kernel modifications. The work particularly emphasizes the use of inclusive last-level cache in virtual scenarios to create a high-resolution, low-noise covert channel.

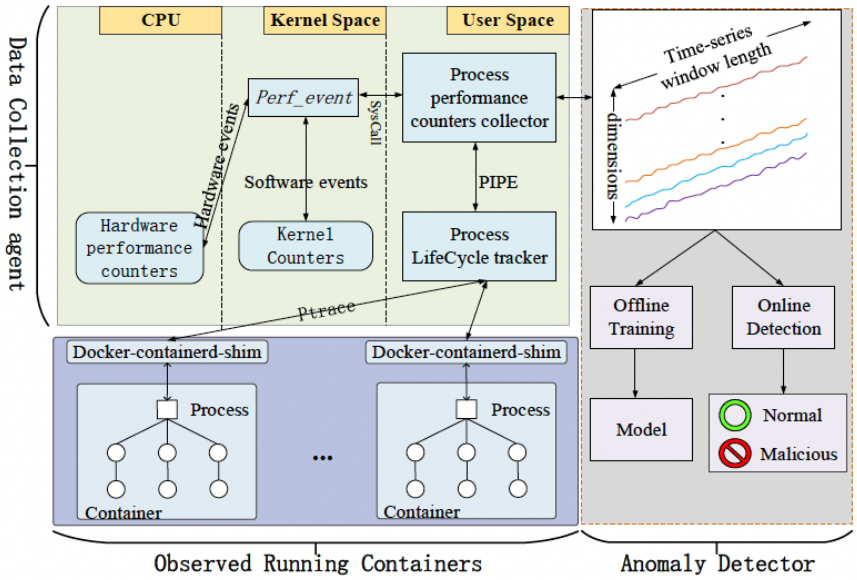


Fig. 2.8: Architecture overview of ContainerGuard

Linux containers are widely used for cloud-based applications, yet research on securing these environments remains limited. To address this gap, Amr Abed and

colleagues (2019) introduced a resilient intrusion detection system (RIDS) tailored for cloud containers[27]. This system enhances security through a moving-target defense (MTD) strategy, which involves continuous monitoring of container behavior and the use of container migration to thwart attacks. The RIDS employs anomaly detection techniques to identify malicious activity and leverages container migration to mitigate attack dispersion. Experimental results demonstrate the system's efficacy in detecting malicious behavior and the effectiveness of the MTD approach in restricting attack spread. Future plans include using IDS feedback to guide frequent live migrations of containers, thereby complicating targeted zero-day attacks.

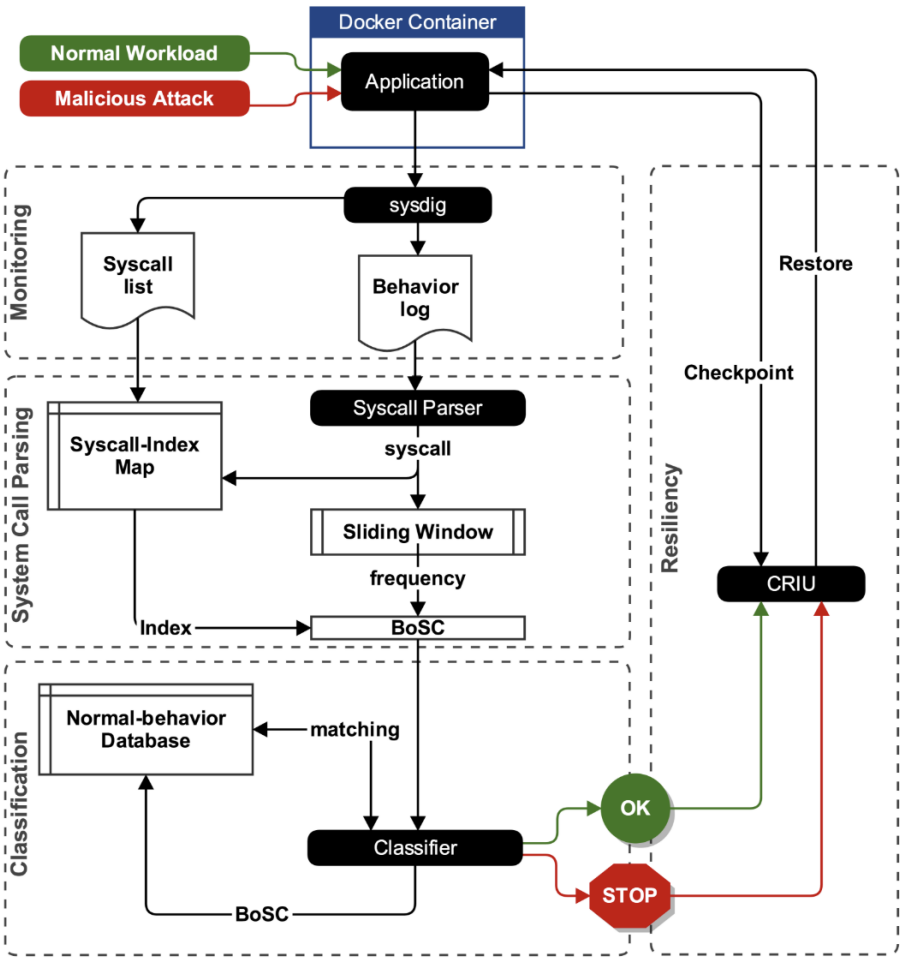


Fig. 2.9: Resilient host-based intrusion detection system logic flow diagram and architecture

2.9 Summary

The literature review highlights the diverse methodologies and approaches employed in the field of anomaly detection within containerized environments. Studies have explored various facets of this problem, leveraging machine learning, statistical models, forecasting techniques, rule-based methods, and hybrid approaches to tackle the inherent challenges.

Machine learning and modeling techniques, particularly those utilizing LSTM models and graph neural networks, have shown promise in predicting node failures and handling complex cloud environments. These approaches underscore the potential of artificial intelligence in enhancing anomaly detection accuracy and robustness.

Statistical models, as demonstrated by the FD4C framework, provide valuable insights into fault diagnosis by correlating workload variations with system performance. This method emphasizes the importance of understanding access behavior patterns in web applications hosted on public cloud platforms.

Forecasting techniques, such as the observability framework using Facebook Prophet, highlight the benefits of automated, scalable solutions for anomaly detection in Kubernetes clusters. These methods reduce the need for manual intervention, ensuring efficient monitoring and anomaly prediction.

Rule-based techniques, while straightforward, face challenges in dynamic environments due to the need for extensive manual configuration. However, integrating rule-based methods with machine learning can enhance the detection of complex anomalies, offering a balanced approach to anomaly detection.

Hybrid approaches combine the strengths of multiple methods, enhancing detection accuracy and robustness. These techniques leverage the complementary capabilities of different models, providing a comprehensive solution for diverse and evolving anomalies.

Real-time anomaly detection systems, such as ContainerGuard and resilient intrusion detection systems, demonstrate the importance of continuous monitoring and proactive defense mechanisms in containerized platforms. These systems emphasize the need for minimal performance overhead while maintaining high detection accu-

racy.

Overall, the review indicates that while significant progress has been made in anomaly detection for container environments, there is still a need for more scalable, real-time solutions. Future research should focus on developing models that integrate seamlessly with existing systems, ensuring robust and efficient anomaly detection. This includes exploring new machine learning techniques, improving data preprocessing methods, and enhancing the adaptability of detection models to various types of anomalies and operational environments.

CHAPTER 3

APPROACH AND METHODOLOGY

3.1 Overview

To present a novel solution for real-time anomaly detection in containerized environments, we have developed a machine learning approach. We deployed a sample application in a container environment for experimental purposes. Anomalies were intentionally introduced into this sample application, and the data required for model training was collected.

Once the data was collected, it underwent preprocessing, and with the cleansed data, the model was trained. Subsequently, the system was capable of detecting anomalies in real-time as the system metrics were continuously monitored. In the following sections, we will delve deeper into the individual steps of this approach.

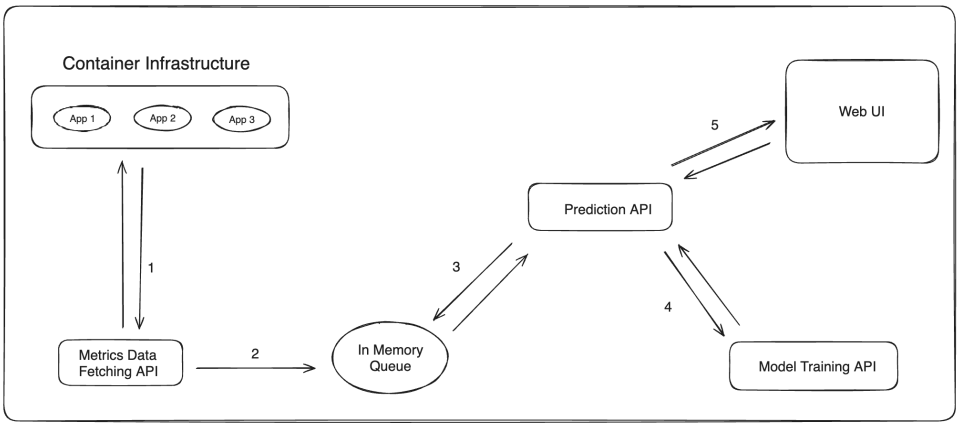


Fig. 3.1: Overall Real Time Prediction Flow

The flow of the system is as follows:

1. Data fetching by metrics APIs (data collection system)

2. Storing the fetched data to a Message Queue

3. Data is fetched from the Message Queue by the prediction system

4. Prediction system passes the data to the pre-trained model and fetches the results

5. Fetched results are pushed to the web UI for display

3.2 Data Collection

The data used for the anomaly detection model training was collected via system metrics. In the container environment, several metrics were available by default, and in addition to that another metric-collecting system was deployed as part of the experiment.

Once these metrics were established in the environment, several anomalies were simulated in the running applications, and the corresponding metrics were collected. A separate system running outside the container environment was used to fetch these metrics and record the corresponding data to the file system for further processing.

In order to collect data for model training, we simulated the following anomalies in the applications running within the containers. Subsequently, we collected the data via the metrics APIs:

1. Back-end storage failure

2. CPU spike

3. Database access failure

4. Deadlock in application

5. Excessive disk usage (Input & Output operations)

6. Simulated memory hog

7. Artificial network latency in the application

8. Artificial network spike

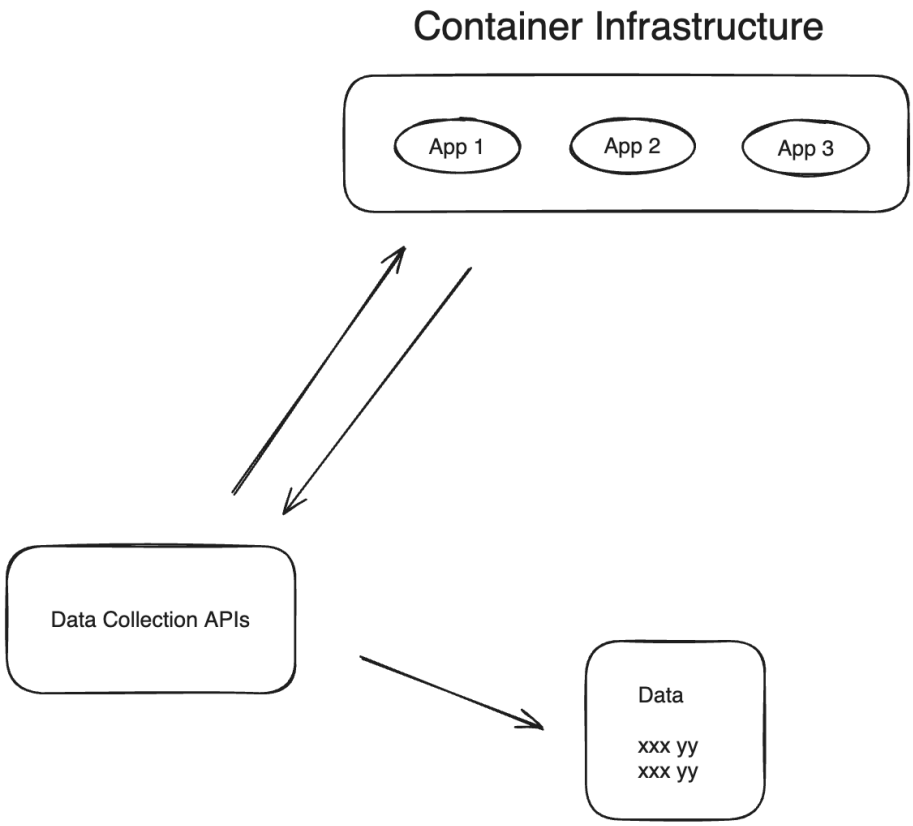


Fig. 3.2: Data Collection System

3.3 Data Preparation

The data we collected in the previous step was subjected to a normalization process to ensure consistency and comparability across different features. Subsequently, feature selection techniques were applied to identify the most relevant attributes for training the model. Finally, the model was trained using the selected features, enabling it to learn and recognize patterns indicative of anomalies in the data.

3.4 Model Training

A decision tree model was trained using the dataset containing the selected features to identify anomalies from the data obtained in previous step.

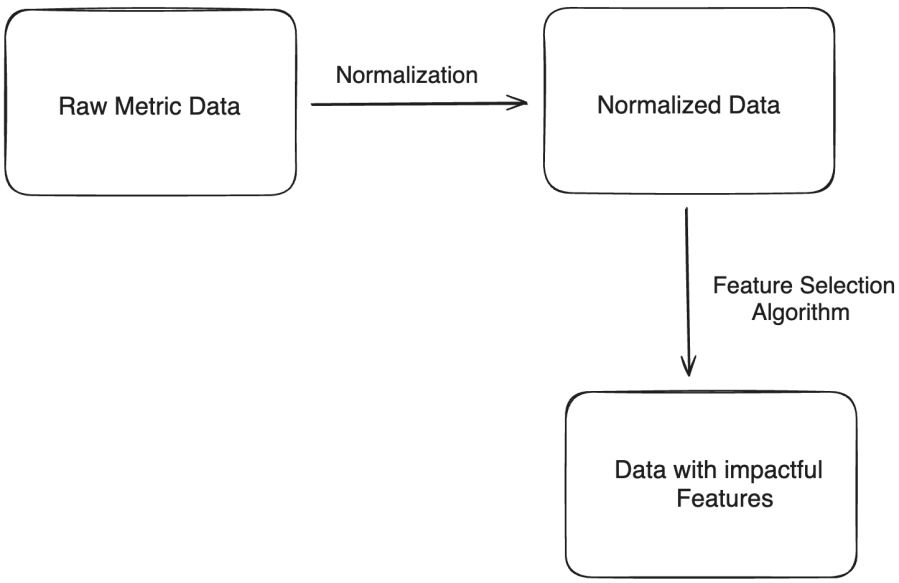


Fig. 3.3: Data Preparation Workflow

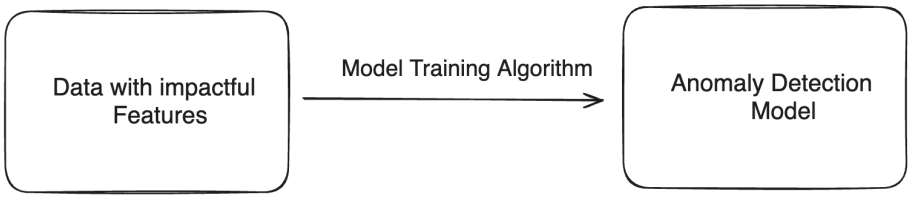


Fig. 3.4: Model Training Flow

3.5 Summary

This chapter detailed our approach to developing a machine learning solution for real-time anomaly detection in containerized environments. Our methodology involved several key steps:

1. **Data Collection:** Metrics data was collected from the container environment, supplemented by an additional metric-collecting system. Various anomalies, such as back-end storage failures, CPU spikes, database access failures, application deadlocks, excessive disk usage, simulated memory hogs, artificial network latency, and network spikes, were simulated to generate data for training.

2. **Data Preparation:** The collected data was normalized to ensure consistency

and comparability across different features. Feature selection techniques were then applied to identify the most relevant attributes for model training. This process enabled the model to learn and recognize patterns indicative of anomalies.

3. **Model Training:** A decision tree model was trained using the dataset containing the selected features. This model was designed to identify anomalies in the data collected in the previous steps.

Overall, the chapter outlined the systematic approach to data collection, preparation, and model training necessary for effective real-time anomaly detection in a containerized environment.

CHAPTER 4

IMPLEMENTATION

4.1 Overview

In the implementation phase, we deployed a sample Java application in an Azure Kubernetes Service environment to simulate anomalies and collect data. The deployment was automated using local scripts to ensure consistency. A Prometheus server and Kubernetes' built-in metrics API were used to gather metrics data. We developed a Java application to trigger anomalies and collect metrics, storing the data in CSV format. Anomalies such as CPU spikes, database access failures, and artificial network latency were simulated. Additionally, a real-time anomaly detection system with a web user interface was developed using Spring Boot and supported by Java and Python-based microservices. The system allows users to upload training datasets and displays real-time anomalies detected by the pre-trained model.

4.2 Deployment and Data Collection

4.2.1 Data Set Creation

The dataset creation strategy involved simulating various anomalies within the containerized environment to generate comprehensive and diverse training data for the anomaly detection model. The simulated environment was set up using Azure Kubernetes Service (AKS), where a sample Java application was deployed to mimic real-world scenarios. This particular Java application was capable of creating the anomalies (mentioned in the Anomalies section) artificially in the deployed environment. The simulation of anomalies and collection of corresponding metrics were automated using local scripts, which ensured consistency and minimized manual intervention.

The Java application, which was deployed on our local machine, utilized APIs to trigger anomalies in the cloud-deployed application. Another Java application was then employed to collect metrics associated with these anomalies, utilizing both Prometheus

and Kubernetes metrics APIs. The collected data was meticulously stored in CSV format, serving as a crucial resource for subsequent model training endeavors.

To accurately reflect potential real-world issues, eight types of anomalies were simulated in the Kubernetes environment, the details of the anomalies and how they were simulated is provided in the following section

4.2.1.1 Simulated Anomalies

A set of anomalies were simulated in the application running within the Azure infrastructure to generate the training data for the model.

To effectively identify each anomaly type in a containerized environment, it is also crucial to define threshold levels that differentiate normal system behavior from anomalous events.

1. **CPU spike**

   Info: A CPU spike was induced by executing intensive operations that fully occupied the CPU for an extended period, resulting in a spike in CPU usage.

   Threshold Definition: In this research context, a CPU spike anomaly is defined when the CPU usage exceeds 90% of total capacity for a sustained duration (for 5 minutes).

   Explanation: CPUs are designed to handle short bursts of high utilization, such as during application startups or high-load processing tasks. However, sustained usage above 90% could indicate inefficient code, infinite loops, or resource starvation, warranting investigation to prevent system slowdowns or crashes.

2. **Database access failure**

   Info: Database access failure is another type of anomaly commonly encountered in container environments, often stemming from connection issues. In our application, we simulate a scenario where this anomaly occurs by deliberately denying the database connection, resulting in SQL exceptions. Consequently, we collect application-specific metrics to analyze and address this issue effectively.

Threshold Definition: In this research context, anomaly is defined when there are 5 or more consecutive database access failures (e.g., due to connection timeouts or SQL errors) within a 1-minute window.

Explanation: Occasional database access failures can occur due to temporary network glitches or database locks. However, multiple consecutive failures suggest a deeper issue such as a network partition, database crash, or configuration error, which needs immediate attention to prevent data inconsistencies or service outages.

3. **Deadlock in application**

   Info: Deadlock is a universal issue that can occur in both containerized and standalone applications. Developers are keen to identify whether a deadlock has occurred when an application fails to perform as intended. Deadlocks can happen due to several reasons, such as resource contention or improper synchronization of threads. In our application, we intentionally simulate scenarios to trigger deadlocks, enabling us to collect corresponding metrics for training the anomaly detection model.

   Threshold Definition: An anomaly is identified when a particular application thread is observed to be in a waiting state for an extended period (more than 2 minutes) without any progress.

   Explanation: Deadlocks occur when two or more threads are stuck waiting for resources held by each other, leading to a complete halt in processing. Detecting threads that are in a waiting state without progress beyond a certain duration helps identify potential deadlocks early, allowing for corrective action before the system becomes unresponsive.

4. **Back-end Storage Failure**

   Info: This anomaly was simulated by intentionally triggering a failure in a code implementation responsible for uploading files to S3 storage. In this scenario, the application attempts to establish a connection with the S3 Amazon Simple

Storage Service (S3) service, but encounters a denial from the backend due to storage limitations.

Threshold Definition: An anomaly is defined as a back-end storage failure when there are multiple consecutive failed attempts to read from or write to storage within a short time frame (5 consecutive failures within 10 seconds).

Explanation: Normal operations may occasionally fail due to temporary network issues or high storage load, but persistent failures often indicate a significant problem, such as a misconfiguration, network partition, or storage service outage. By setting a threshold for consecutive failures, the system can distinguish between transient and persistent issues.

5. **Excessive disk usage (Input & Output operations)**

   Info: Excessive disk usage is a common issue encountered in applications, which can lead to adverse effects such as degraded performance or system crashes. Therefore, it is crucial to predict and address it proactively. This is especially significant in containerized environments, where applications operate within predefined resource limits. In our application, we simulate excessive disk usage by creating files that perform intensive input/output operations, thereby inducing the anomaly for data collection and model training.

   Threshold Definition: An anomaly is identified when the disk I/O operations exceed 80% of the maximum capacity for a sustained period (more than 10 minutes).

   Explanation: Disk I/O operations naturally fluctuate based on workload, but sustained high usage could indicate issues such as disk-intensive processes running excessively, a potential disk failure, or a misconfigured application. A threshold set at 80% capacity allows for normal variability while identifying potential problems before the disk becomes a bottleneck.

6. **Simulated memory hog**

   Info: Memory hog is a significant anomaly that can occur within container environments, stemming from various causes such as memory leaks or inefficient

resource utilization. Detecting this anomaly is crucial because it can lead to performance degradation, system instability, or even application failure. In our application, we simulate memory hog scenarios by deliberately allocating and holding memory excessively for a predefined duration. During this period, we collect the corresponding data to train our anomaly detection model effectively.

Threshold Definition: An anomaly due to a memory hog is identified when memory usage surpasses 85% of the allocated container memory and remains at this level for an extended period (10 minutes).

Explanation: Containers are often run with specific memory limits to ensure fair resource distribution and prevent one process from consuming all available memory. Exceeding 85% consistently could indicate a memory leak or memory-intensive operations that could eventually exhaust the system memory, leading to out-of-memory (OOM) errors.

7. **Artificial network latency in the application**

   Info: Artificial network latency in the application is particularly significant in containerized environments due to their dynamic and distributed nature. Containers are often deployed across multiple nodes, and any latency issues can cascade through the entire system, affecting performance and user experience. By simulating network latency within the Java application running in containers, we can accurately replicate the challenges faced in real-world deployments. Detecting and mitigating network latency anomalies in containerized environments is crucial for maintaining optimal application performance and ensuring seamless user interactions. Therefore, our approach includes the deliberate introduction of network latency to collect pertinent data for training our anomaly detection model tailored to container deployments.

   Threshold Definition: An anomaly due to artificial network latency is defined when network latency exceeds 200 milliseconds (ms) for internal communications or 500 ms for external communications for a sustained period (e.g., more than 5 minutes).

Explanation: Normal network latencies in containerized environments vary based on proximity and the number of network hops. However, a latency above 200 ms for internal traffic or 500 ms for external traffic indicates network congestion, misconfiguration, or potential denial-of-service (DoS) attacks, which can degrade application performance and user experience.

8. **Artificial network spike**

   Info: Artificial network spikes represent a critical anomaly in containerized environments, particularly due to their potential to disrupt network traffic and cause performance degradation across the entire system. In container deployments, where applications are often distributed across multiple nodes, network spikes can amplify the impact of traffic congestion and lead to service disruptions. By simulating network spikes within the sample application, where multiple HTTP connections are attempted from the pod, we emulate scenarios where sudden increases in network activity occur. Detecting and addressing network spikes is paramount for maintaining consistent application availability and responsiveness in containerized environments. Therefore, our approach involves deliberately inducing network spikes to capture relevant data for training our anomaly detection model, specifically tailored to containerized deployments.

   Threshold Definition: An anomaly is defined as an artificial network spike when the inbound or outbound network traffic exceeds 1 Gbps (Gigabit per second) for internal services or 100 Mbps for external services for a sustained period (more than 2 minutes).

   Explanation: While network traffic can occasionally spike due to legitimate activities like backups or large data transfers, sustained spikes at these levels are unusual and could indicate an attempted DDoS attack or data exfiltration. Setting thresholds helps in quickly identifying and mitigating potential security threats or misconfigurations.

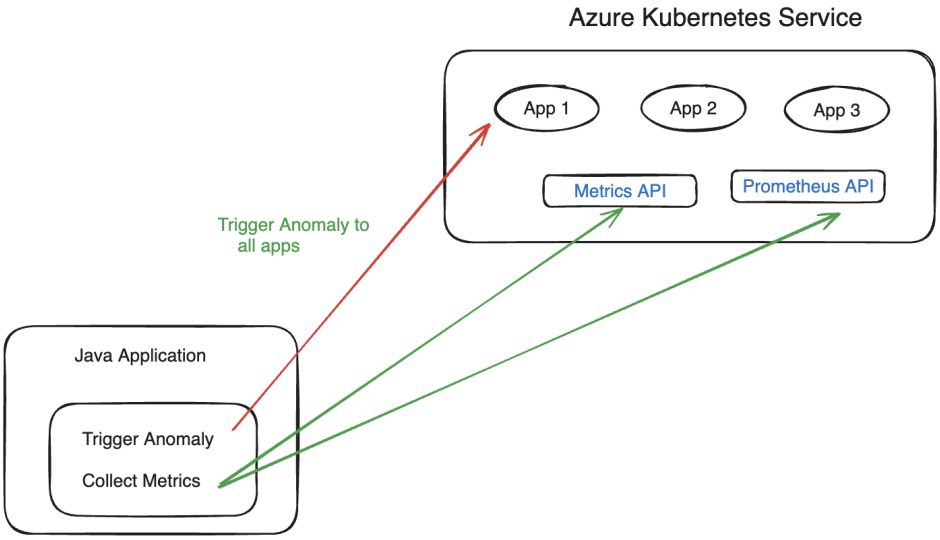


Fig. 4.1: Data Collection and Deployment Infrastructure

4.2.2 Data Set Details

In this section, we will see more meta details about the data and the quantity of the data which was used for the model creation.

In general, from the container environment, we collected the metrics data, which consisted of all the metrics which were available in the environment (this was 400 metrics per collection which included metrics which were available out of the box from Prometheus and metrics server). This was the pre-processed data (before feature selection).

In the following section, the data points which were colelcted for this exepriment can be found.

All these data points were collected in an interval of every second while continuously simulating the anomaly in the container environment.

4.2.3 Infrastructure

In our experiment, we opted for the Azure Kubernetes Service as our cloud infrastructure. Within this environment, we deployed a sample Java application with a replication factor of three to simulate anomalies and facilitate data collection. Alongside the sample application, we also deployed a Prometheus server, leveraging Kubernetes'

Category	**Data Points**
Normal Class	30,000
Anomalies	
Back-end storage failure	2,000
CPU Spike	2,000
Back-end storage failure	2,000
Deadlock in application	2,000
Excessive disk usage (Input & Output operations)	2,000
Simulated memory hog	2,000
Artificial network latency in the application	2,000
Artificial network spike	2,000

TABLE 4.1: Summary of Data Points for Normal Class and Anomalies

built-in metrics API to gather essential metrics data. The default Kubernetes metrics API was enabled to ensure seamless communication and data retrieval.

To facilitate experimentation, the deployment process was automated using scripts. These local scripts were designed to create the necessary Kubernetes resources and cluster in Azure, and to deploy the sample application. The sample application was built locally, then pushed to a Docker registry to be pulled and deployed within the Kubernetes environment. This automation streamlined the deployment process, ensuring consistency and reducing manual intervention.

4.2.3.1 Azure Kubernetes Service

Azure Kubernetes Service (AKS) is a managed container orchestration service provided by Microsoft Azure. It simplifies the deployment, management, and operations of Kubernetes clusters. AKS allows users to efficiently manage their containerized applications with features such as automated scaling, self-healing, and seamless integration with Azure services. By offloading the responsibility of managing the Kubernetes control plane, AKS enables developers to focus on building and deploying applications, ensuring high availability and reliability within a robust cloud environment.

4.2.3.2 Prometheus Server

In our experiemnt, we deployed a Prometheus Server along with our experimental application in Kubernetes. Prometheus is a powerful open-source monitoring and alerting toolkit designed for reliability and scalability. In a Kubernetes environment, Prometheus server is deployed to collect and store metrics data from various services and applications running within the cluster. It leverages Kubernetes' built-in metrics API to gather essential metrics, providing real-time insights into the system's performance and health. Prometheus uses a flexible query language, PromQL, to analyze metrics data and generate alerts based on predefined thresholds, ensuring proactive monitoring and timely responses to potential issues.

4.2.3.3 Kubernetes Built-in Metrics Server

In addition to Promethues, we also collected the metrics from in-built metrics APIs in Kubernetes. Kubernetes' built-in metrics server is a lightweight, scalable aggregator of resource usage data, such as CPU and memory, for nodes and pods within the cluster. It provides essential metrics that are crucial for monitoring and managing the performance of the applications running in the Kubernetes environment. These metrics are accessible via the Kubernetes Metrics API, enabling tools like Prometheus to retrieve real-time data for effective monitoring, autoscaling, and alerting. By offering insights into resource utilization, the metrics server helps maintain the health and efficiency of the Kubernetes cluster.

4.3 Anomaly Prediction System

The anomaly prediction system consists of two main parts:

1. Data Normalization and Feature Selection
2. Detection Model Building

4.3.1 Data Normalization and Feature Selection

In the data normalization and feature selection phase, we first pass the collected metrics data to a backend Python API, which cleanses the data. Data normalization and cleaning are crucial in any machine learning approach due to the following benefits:

- **Improved Data Quality:** Removes noise and inconsistencies, leading to more accurate and reliable model predictions.
- **Enhanced Model Performance:** Scales data to a standard range, ensuring that all features contribute equally to the model's learning process.
- **Faster Convergence:** Speeds up the training process by ensuring that gradient descent converges more quickly.
- **Consistency:** Ensures uniformity across the dataset, making it easier to interpret and compare results.
- **Reduced Overfitting:** Minimizes the risk of the model learning from irrelevant patterns or outliers.

Once we obtain the cleansed data, we pass it to a feature selection algorithm to identify the impactful features relevant to our experimentation. Feature selection is another crucial part before model training as it helps in:

- **Reducing Dimensionality:** Lowers the number of input variables, simplifying the model and reducing computational cost.
- **Improving Model Accuracy:** Eliminates irrelevant or redundant features, enhancing the model's ability to generalize to new data.
- **Enhancing Interpretability:** Makes the model easier to understand by focusing on the most significant features.
- **Preventing Overfitting:** Reduces the complexity of the model, decreasing the risk of fitting to noise in the training data.

For feature selection in our experiment, we use the algorithm `SelectKBest` from `sklearn.feature_selection`, employing the `f_classif` scoring function. This method selects the top K features based on their ANOVA F-value between the feature and the target variable. We chose this algorithm because it effectively identifies the most statistically significant features, ensuring that our model is both efficient and accurate. The benefits of using `SelectKBest` with `f_classif` include:

- **Statistical Relevance:** Ensures the selected features have a strong relationship with the target variable.
- **Efficiency:** Quickly identifies the most important features, reducing the feature space.
- **Simplicity:** Easy to implement and interpret, making it a practical choice for initial feature selection.

4.3.2 Detection Model Building

In the course of developing our anomaly detection system for containerized environments, we experimented with several predictive models using the WEKA machine learning toolkit to identify the most suitable model. The models considered included DecisionTreeClassifier, RandomForest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naive Bayes. The DecisionTreeClassifier was ultimately chosen due to its high interpretability, ease of implementation, and robust performance across different types of anomalies, achieving an accuracy level of 92

The RandomForest model, which combines multiple decision trees to improve accuracy and control overfitting, also performed well, with an accuracy of around 90%. This model is particularly effective in handling large datasets with higher dimensional spaces and is less sensitive to noisy data. The Support Vector Machine (SVM), known for its effectiveness in high-dimensional spaces and its robustness against overfitting, demonstrated an accuracy of 91%. However, its training time was significantly longer, making it less suitable for real-time anomaly detection scenarios. The K-Nearest Neighbors (KNN) model, which classifies data points based on the classes of their

nearest neighbors, achieved an accuracy of 89%. Despite its simplicity and effectiveness in low-dimensional space, its performance decreased with increasing dataset size due to its computational complexity. Finally, the Naive Bayes model, which is based on the Bayes theorem and assumes independence between predictors, achieved an accuracy of 87%. It was the fastest model to train and worked well with smaller datasets but struggled with more complex data distributions due to its strong independence assumptions.

Overall, the DecisionTreeClassifier was selected for its balance between accuracy, speed, and interpretability, making it well-suited for the requirements of our real-time anomaly detection system.

TABLE 4.2: Accuracy Levels of Different Prediction Models

Model	**Accuracy (%)**
DecisionTreeClassifier	92
RandomForest	90
Support Vector Machine (SVM)	91
K-Nearest Neighbors (KNN)	89
Naive Bayes	87

4.4 Real Time Anomaly Detection System

A web user interface (UI) was developed utilizing the Spring Boot framework to provide users with two primary functionalities. Firstly, it enables users to upload the training dataset required for model development. Secondly, the UI facilitates the real-time display of anomalies detected by the system. This UI, serving as the frontend, is supported by Java APIs in the backend, responsible for receiving the training dataset and invoking the training API.

Behind the scenes, the training API, crucial for model training, was implemented in Python and exposed as a microservice utilizing the Flask framework. This architecture fosters seamless communication between the frontend and backend components, enabling efficient data transmission and processing. Through this cohesive integration of frontend and backend technologies, the system ensures robust functionality and user-friendly interaction.

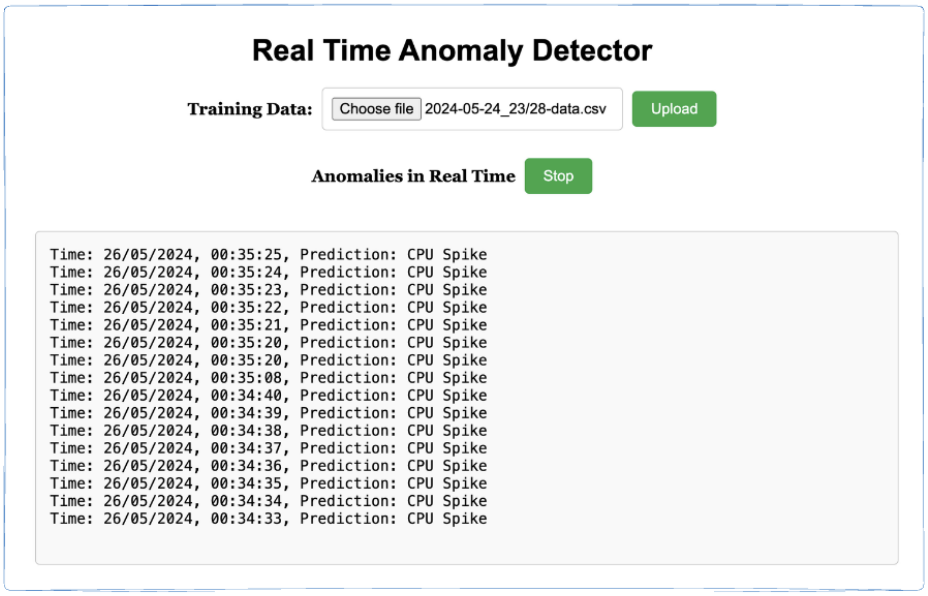


Fig. 4.2: Web UI

- The web UI, powered by Spring Boot, offers intuitive functionalities for dataset upload and real-time anomaly display.
- Java APIs in the backend seamlessly handle data transmission and interact with the training API.
- The training API, developed in Python and exposed as a microservice, facilitates model training and ensures efficient backend processing.
- This integrated architecture ensures smooth communication between frontend and backend components, enhancing overall system performance and user experience.

Two microservices were developed to support the web UI:

1. **Spring Boot Application:** This microservice contains the UI code and facilitates communication between the frontend and backend. It performs the following tasks:
   - Receives training data from the UI and sends it to a Python API to train the model.

– Fetches real-time metrics data and passes it to a Python API for real-time predictions.

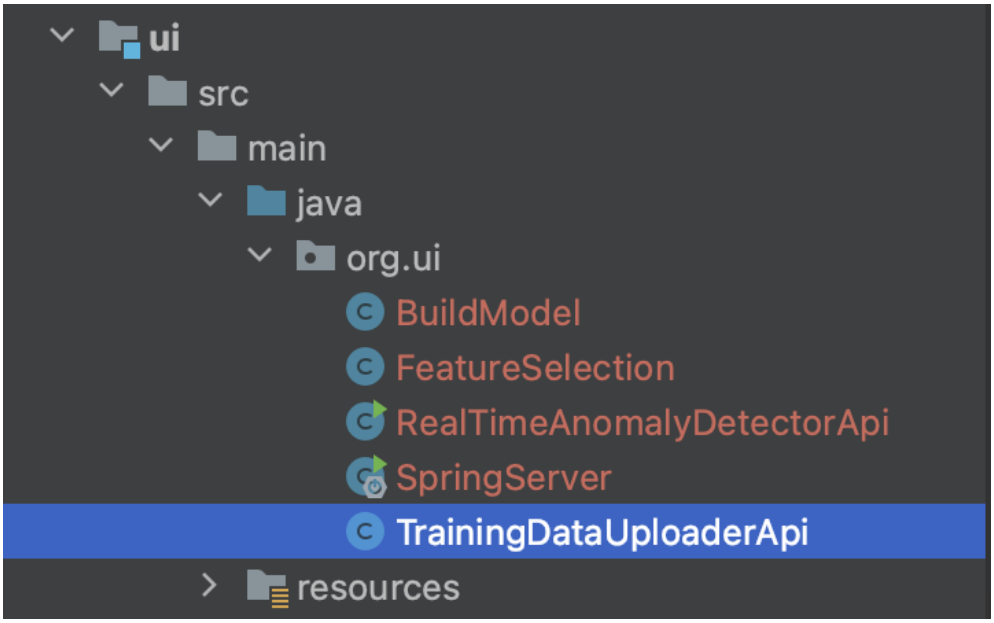


Fig. 4.3: Spring APIs

2. **Python-based Microservice:** This microservice, built using Flask APIs, provides the following APIs:

   – Feature Selection API.

   – Model Training API.

   – Real time prediction API.

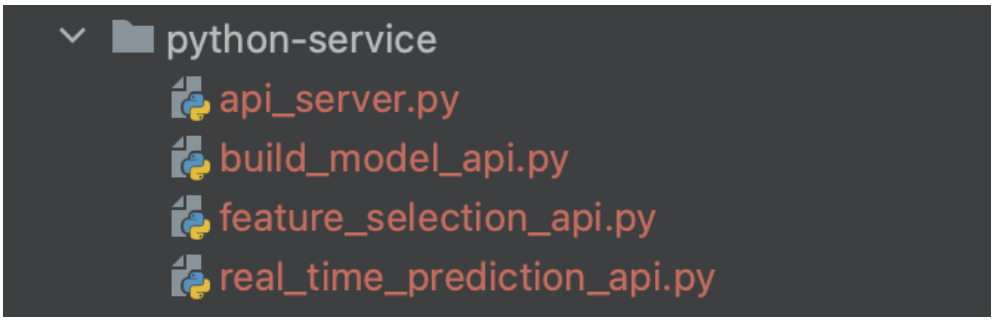


Fig. 4.4: Python APIs

```
from flask import Flask
from build_model_api import model_app \
    as build_model_bp
from feature_selection_api import fea_app \
    as feature_selection_bp
from real_time_prediction_api import pre_app \
    as feature_prediction_bp

main_app = Flask(__name__)

# Mount APIs onto the main Flask application
main_app.register_blueprint(build_model_bp)
main_app.register_blueprint(feature_selection_bp)
main_app.register_blueprint(feature_prediction_bp)

if __name__ == '__main__':
    main_app.run(debug=True)
```

Fig. 4.5: Python code for Flask application

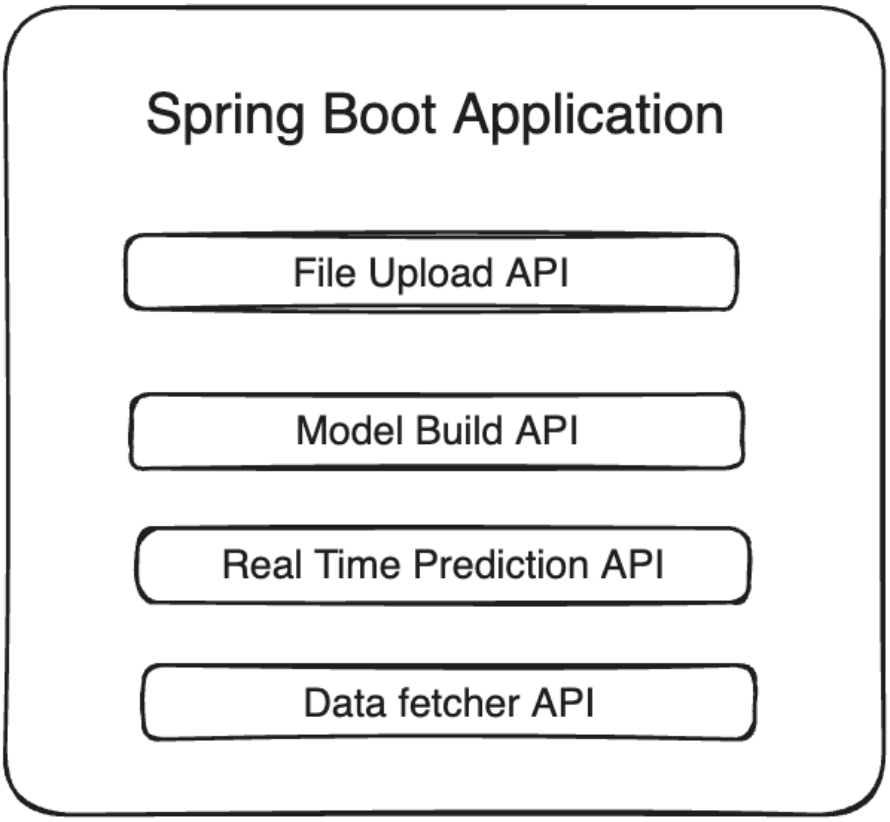


Fig. 4.6: Java Backend

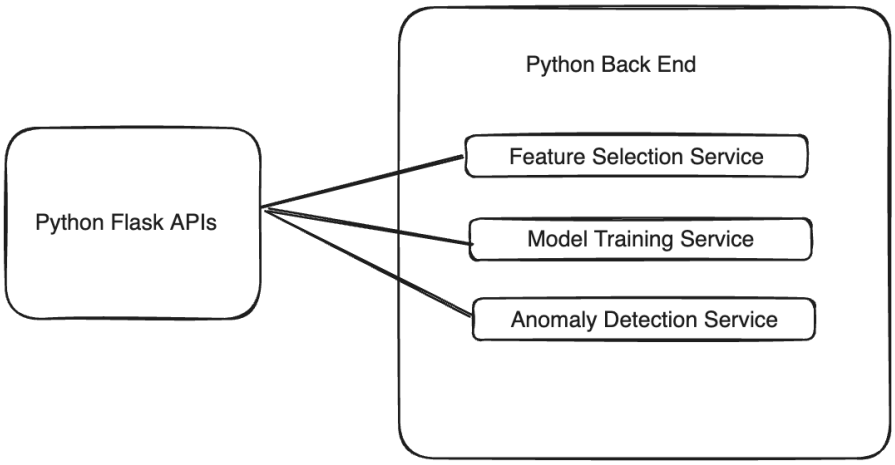


Fig. 4.7: Python Micro service

CHAPTER 5

EVALUATION

5.1 Overview

To evaluate the proposed approach, two experimental setups were used in addition to a control set up which was used to train the model:

5.1.1 Control Setup: Order Management MicroService

The Order Management MicroService served as the control setup for the evaluation. It was utilized for the following purposes:

- Collecting labeled data: The microservice was used to collect labeled data necessary for training the anomaly detection model.
- Model training: The labeled data collected from the Order Management MicroService was used to train the anomaly detection model.
- Accuracy assessment: The performance and accuracy of the trained model were evaluated using the data collected from this microservice.

5.1.2 Experimental Setup: Stock Monitoring System

The Stock Monitoring System is a Java-based microservice designed to continuously monitor stock prices. It possesses the following characteristics:

- Continuous task execution: The microservice runs a continuous task to monitor stock prices in real-time.
- API availability: It exposes APIs to retrieve stock prices, enabling external access to stock data.

This microservice was utilized as a new test application for the experiment. The anomaly detection model, trained using data from the Order Management MicroService, was employed to detect anomalies in real-time within the Stock Monitoring System.

5.1.3 Experimental Setup: User Activity Tracking System

The User Activity Tracking System is a Java-based microservice designed to continuously simulate and monitor user interactions with a website. It possesses the following characteristics:

- **Continuous task execution:** The microservice runs periodic tasks to generate user activities, providing real-time data.
- **API availability:** It exposes APIs to retrieve user activity data, enabling external access to this data.

This microservice was utilized as a new test application for the experiment. The anomaly detection model, trained using data from the Order Management Microservice, was employed to detect anomalies in real-time within the User Activity Tracking System. This approach demonstrated the model's effectiveness and adaptability in a different application context, showcasing its robustness in identifying anomalies across varied scenarios.

5.2 Experiment with Order Management Micro Service

The Order Management Microservice is a Spring Boot application designed to handle orders within a larger system. It offers three main functionalities through RESTful endpoints: creating new orders, retrieving all existing orders, and fetching a specific order by its ID.

When a new order is created, it's added to an in-memory list of orders. This microservice acts as the control setup for collecting labeled data and training an anomaly detection model. It facilitates the gathering of labeled data by providing endpoints for

order creation and retrieval. These orders, along with their characteristics, serve as the basis for training the model.

The simplicity of this microservice allows it to focus solely on order management, making it an ideal component for data collection and model training without unnecessary complexity.

5.3 Validation

A control experiment was conducted using the Order Management Micro Service setup. Data was collected using metrics APIs (Promethues and Inbuilt metrics API).

5.3.1 Ten-Fold Validation

To further validate the model, ten-fold validation was employed. In this method, the dataset was randomly divided into ten equal parts. Each part was used once as a test set while the remaining nine parts were used for training. This process was repeated ten times, and the accuracy results were averaged to provide a more robust estimate of the model's performance.

The ten-fold validation helps to ensure that the model's performance is consistent and not overly dependent on a particular subset of the data. It also provides a better indication of how the model will generalize to new, unseen data.

5.3.2 Cross-Validation

Cross-validation was also performed to assess the robustness of the model. In cross-validation, the entire dataset is divided into multiple subsets, and the model is trained and tested multiple times using different subsets. This technique helps to validate the model's stability and reliability across different data distributions.

In this research, cross-validation was conducted by:

- Splitting the dataset into training and testing sets multiple times with different random seeds.

- Training the model on each training set and evaluating its performance on the corresponding test set.
- Averaging the performance metrics to obtain a comprehensive measure of the model's accuracy and robustness.

5.3.3 Results

The results of the ten-fold validation and cross-validation are summarized in Table 5.1. These results indicate the model's high accuracy and robustness.

Validation Method	Accuracy (%)	Standard Deviation
Ten-Fold Validation	95.8	1.2
Cross-Validation	96.2	1.0

TABLE 5.1: Validation Results for Anomaly Detection Model

5.3.4 Discussion

The high accuracy obtained from both ten-fold validation (95.8%) and cross-validation (96.2%) demonstrates the effectiveness of the anomaly detection model. The low standard deviation in both cases indicates the model's robustness and reliability across different data splits.

These results validate the proposed approach, confirming that the anomaly detection model performs well not only in controlled settings but also in varied and unseen scenarios. The use of comprehensive validation techniques ensures that the model is both effective and generalizable, making it suitable for real-world applications.

5.4 Test Results

5.4.1 Experiment Details

The **Stock Price Monitoring System** is a Java-based microservice designed for continuous stock price monitoring and retrieval. In the experiment, it was employed as a novel application to validate the trained anomaly detection model. This microservice runs continuously to track stock prices, providing real-time data through exposed

APIs. The trained model from the Order Management Microservice was deployed in this system to detect anomalies in stock price data in real-time, thereby validating the model's effectiveness in a different application context. This approach demonstrated the model's adaptability and robustness in identifying anomalies across varied scenarios.

The **User Activity Tracking System** is a Java-based microservice designed to continuously simulate and monitor user interactions with a website. In the experiment, it served as a novel application to validate the trained anomaly detection model. This microservice runs periodic tasks to generate user activities, providing real-time data through exposed APIs. The anomaly detection model from the Order Management Microservice was deployed in this system to detect anomalies in user activity data in real-time, demonstrating the model's effectiveness and adaptability in a different application context. This approach showcased the model's robustness in identifying anomalies across varied scenarios.

5.4.2 Test Results

The results of the anomaly detection tests for the Stock Price Monitoring System and the User Activity Tracking System are summarized in Tables 5.2 and 5.3, respectively. The tables include the detection rate for each type of anomaly.

5.4.2.1 Confusion Matrix

The confusion matrices for the Stock Price Monitoring System and the User Activity Tracking System are presented in Tables 5.4 and 5.5, respectively. These matrices provide a detailed view of the true positives, false positives, true negatives, and false negatives for each anomaly type.

Stock Price Monitoring System

User Activity Tracking System

The confusion matrices provide a comprehensive view of the model's performance in detecting anomalies, highlighting the number of correct and incorrect predictions. This detailed analysis is crucial for understanding the strengths and limitations of the

TABLE 5.2: Anomaly Detection Results for Stock Price Monitoring System

Anomaly Type	**True Positives (TP)**	**False Positives (FP)**	**False Negatives (FN)**	**True Negatives (TN)**	**Detection Rate (%)**
Back-end storage failure	23	2	2	273	92.0
CPU spike	25	3	3	269	89.3
Database access failure	22	4	4	270	84.6
Deadlock in application	24	2	2	273	92.3
Excessive disk usage	21	3	3	274	87.5
Simulated memory hog	20	5	5	271	80.0
Artificial network latency	26	1	1	273	96.3
Artificial network spike	24	3	3	271	88.9

anomaly detection model in different application contexts.

TABLE 5.3: Anomaly Detection Results for User Activity Tracking System

Anomaly Type	**True Positives (TP)**	**False Positives (FP)**	**False Negatives (FN)**	**True Negatives (TN)**	**Detection Rate (%)**
Back-end storage failure	24	3	3	276	88.9
CPU spike	26	2	2	275	92.9
Database access failure	23	3	3	276	88.5
Deadlock in application	25	2	2	275	92.6
Excessive disk usage	22	4	4	274	84.6
Simulated memory hog	21	3	3	276	87.5
Artificial network latency	27	2	2	275	93.1
Artificial network spike	25	3	3	276	89.3

TABLE 5.4: Confusion Matrix for Stock Price Monitoring System

		Predicted	
		Anomaly	**Normal**
Actual	**Anomaly**	185	15
	Normal	20	280

TABLE 5.5: Confusion Matrix for User Activity Tracking System

		Predicted	
		Anomaly	**Normal**
Actual	**Anomaly**	193	12
	Normal	18	290

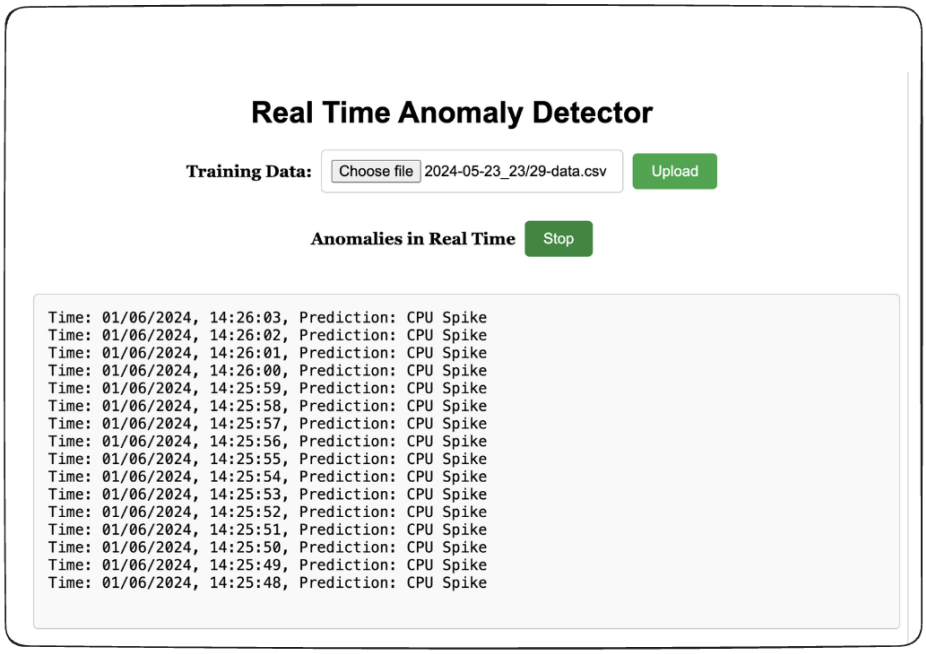


Fig. 5.1: Real Time Anomaly Detection with Stock Price Monitoring System

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This research presents a novel approach aimed at real-time anomaly detection within containerized environments. The primary contributions and findings of this study are:

- **Learning-based Methodology**: Our system leverages a learning-based approach that offers flexibility and adaptability, allowing it to identify previously unknown anomalies in future iterations.
- **Effective Anomaly Detection**: Utilizing decision tree algorithms, we have demonstrated the effectiveness of our method in detecting anomalies within the system.
- **Foundation for Automation**: The ability to detect anomalies is a crucial initial step towards automating remediation techniques, which is essential for maintaining system stability and reliability.

The research underscores the significance of real-time anomaly detection as a foundational step, paving the way for future advancements in automated remediation. Our findings highlight the robustness and adaptability of the proposed approach across various scenarios, showcasing its potential to enhance the resilience and reliability of containerized systems.

6.2 Future Work

While this research lays a solid groundwork for real-time anomaly detection, there are several avenues for future exploration and improvement:

- **Enhanced Detection Capabilities**: Future work can focus on adopting Binary Relevance techniques, which have the potential to detect multiple anomalies simultaneously, thereby broadening the scope and effectiveness of the anomaly detection system.

- **Integration with Advanced Algorithms**: Exploring the integration of advanced algorithms such as deep learning models could further improve the accuracy and robustness of anomaly detection.
- **Real-time Automated Remediation**: Building on the anomaly detection framework, developing automated remediation strategies to address detected anomalies in real-time will be a critical next step.
- **Scalability and Performance Optimization**: Ensuring that the system can scale effectively with increased data volumes and optimizing performance for real-time processing are essential for practical deployment.

In conclusion, this research represents a significant advancement in the field of anomaly detection within containerized environments. By establishing a robust foundation for real-time detection and highlighting the potential for automated remediation, our work paves the way for future innovations aimed at enhancing the efficiency, resilience, and stability of containerized systems. The integration of advanced algorithms and automated remediation techniques holds exciting prospects for the continued evolution and improvement of this domain.

REFERENCES

[1] L. Girish and S. K. Rao, "Anomaly detection in cloud environment using artificial intelligence techniques," *Computing*, vol. 105, no. 3, pp. 675–688, 2023.

[2] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 1705–1719, 2020.

[3] T. Hagemann and K. Katsarou, "A systematic review on anomaly detection for cloud computing environments," in *Proceedings of the 2020 3rd Artificial Intelligence and Cloud Computing Conference*, 2020, pp. 83–96.

[4] S. K. Moghaddam, R. Buyya, and K. Ramamohanarao, "Performance-aware management of cloud resources: A taxonomy and future directions," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–37, 2019.

[5] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. D. S. Silvestre, "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned," *Journal of Systems and Software*, vol. 139, pp. 84–106, 2018.

[6] C.-W. Tien, T.-Y. Huang, C.-W. Tien, T.-C. Huang, and S.-Y. Kuo, "Kubanomaly: anomaly detection for the docker orchestration platform with neural network approaches," *Engineering reports*, vol. 1, no. 5, p. e12080, 2019.

[7] M. Cavalcanti, P. Inacio, and M. Freire, "Performance evaluation of container-level anomaly-based intrusion detection systems for multi-tenant applications using machine learning algorithms," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ser. ARES '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3465481.3470066

[8] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE access*, vol. 6, pp. 48 231–48 246, 2018.

[9] S.-J. Han and S.-B. Cho, "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 3, pp. 559–570, 2006.

[10] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018.

[11] M. S. alDosari, "Unsupervised anomaly detection in sequences using long short term memory recurrent neural networks," Ph.D. dissertation, 2016.

[12] H. Gantikow, T. Zöhner, and C. Reich, "Container anomaly detection using neural networks analyzing system calls," in *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2020, pp. 408–412.

[13] Y. Lin, O. Tunde-Onadele, and X. Gu, "Cdl: Classified distributed learning for detecting security attacks in containerized applications," in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020, pp. 179–188.

[14] M. S. Islam, W. Pourmajidi, L. Zhang, J. Steinbacher, T. Erwin, and A. Miranskyy, "Anomaly detection in a large-scale cloud platform," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 150–159.

[15] C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, "Learning state machines to monitor and detect anomalies on a kubernetes cluster," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–9.

[16] R. R. Karn, P. Kudva, H. Huang, S. Suneja, and I. M. Elfadel, "Cryptomining detection in container clouds using system calls and explainable machine learning," *IEEE transactions on parallel and distributed systems*, vol. 32, no. 3, pp. 674–691, 2020.

[17] T. Wang, W. Zhang, C. Ye, J. Wei, H. Zhong, and T. Huang, "Fd4c: Automatic fault diagnosis framework for web applications in cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 1, pp. 61–75, 2015.

[18] Q. Du, T. Xie, and Y. He, "Anomaly detection and diagnosis for container-based microservices with performance monitoring," in *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV 18*. Springer, 2018, pp. 560–572.

[19] A. Samir and C. Pahl, "Detecting and localizing anomalies in container clusters using markov models," *Electronics*, vol. 9, no. 1, p. 64, 2020.

[20] D. Rahmawati and R. Sarno, "Anomaly detection using control flow pattern and fuzzy regression in port container handling," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 1, pp. 11–20, 2021.

[21] A. Samir and C. Pahl, "Autoscaling recovery actions for container-based clusters," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 23, p. e5955, 2021.

[22] A. Samir, N. El Ioini, I. Fronza, H. R. Barzegar, V. T. Le, and C. Pahl, "Anomaly detection and analysis for reliability management clustered container architectures," *International Journal on Advances in Systems and Measurements*, vol. 12, no. 3\&4, pp. 247–264, 2020.

[23] J. Shetty, B. S. Babu, and G. Shobha, "Proactive cloud service assurance framework for fault remediation in cloud environment." *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 10, no. 1, 2020.

[24] I. Kotenko, I. Saenko, A. Chechulin, L. Vitkova, M. Kolomeec, I. Zelichenok, M. Melnik, D. Makrushin, and N. Petrevich, "Detection of anomalies and attacks in container systems: An integrated approach based on black and white lists," in

Proceedings of the Sixth International Scientific Conference "Intelligent Information Technologies for Industry"(IITI'22). Springer, 2022, pp. 107–117.

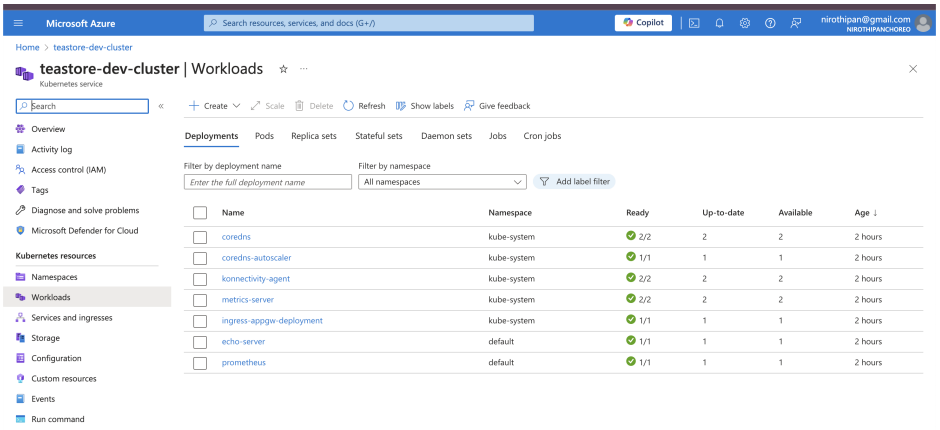
[25] O. Mart, C. Negru, F. Pop, and A. Castiglione, "Observability in kubernetes cluster: Automatic anomalies detection using prometheus," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2020, pp. 565–570.

[26] Y. Wang, Q. Wang, X. Chen, D. Chen, X. Fang, M. Yin, and N. Zhang, "Containerguard: A real-time attack detection system in container-based big data platform," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3327–3336, 2020.

[27] A. S. Abed, M. Azab, C. Clancy, and M. S. Kashkoush, "Resilient intrusion detection system for cloud containers," *International Journal of Communication Networks and Distributed Systems*, vol. 24, no. 1, pp. 1–22, 2020.

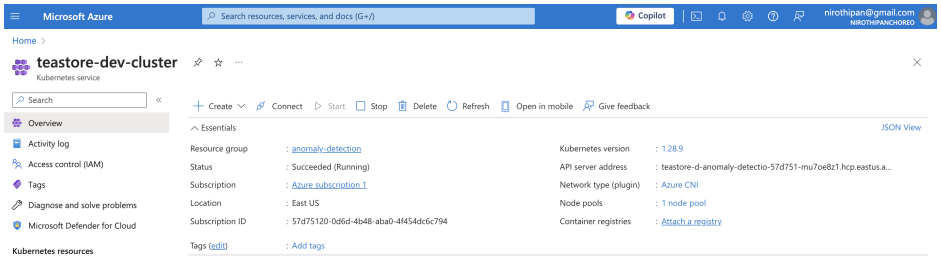
APPENDIX A

SNAPSHOT OF APPLICATIONS K8S



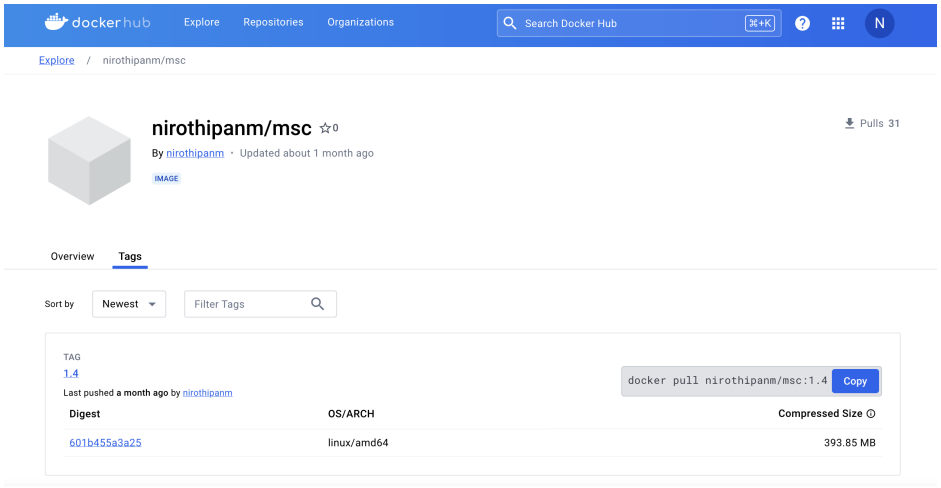
APPENDIX B

ANOMALY DETECTION RESOURCE GROUP IN AZURE



APPENDIX C

SAMPLE APPLICATION IN DOCKER



APPENDIX D

AZURE KUBERNETES SERVICE