**Detection of DoS/DDoS attacks in Micro-services using Machine and Deep Learning**

**FYP– I REPORT**

**BS(CS/SE/CYS/AI) Fall 2024**

Asher Siddique:

21k-3860

Rija Anwar:

21k-3069

Mahnoor Amjad:

21k-3884



**Supervisor: Mr. Shoaib Raza**

**Department of Software Engineering**

**FAST-National University of Computer & Emerging Sciences, Karachi**

Contents

Introduction..................................................................................................................... 4 1. Related Work............................................................................................................... 5 2. Methodology................................................................................................................8 3. Testing and Results.................................................................................................. 12 4. System Diagram....................................................................................................... 15 5. Goals for FYP-II....................................................................................................... 15 6. Conclusion................................................................................................................ 15 References.................................................................................................................... 16

1. Introduction

With the rising adoption of cloud native applications & micro-service architectures, Kubernetes has gained sufficient notability as a container orchestration platform that deploys micro-services as per user/customer demand. However, it is prone to DoS and DDoS attacks made vulnerable by its scalability and flexibility issues just like any other online system is, but what makes it harder to detect is its distributed nature. This leads to severe service disruptions, loss of revenue, & compromised customer trust.

Classical machine learning has been employed for anomaly detection in cloud and containerized environments. Deep learning has been tried in detecting DoS/DDoS attack patterns. Challenges, however, remain in terms of real-time detection, scalability, minimizing false positives, and adapting to evolving attack patterns.

Our efforts aim to combine the logic of ML and DL to detect anomalous behavior that is indicative of DoS/DDoS attacks on cloud native applications through a feature collection technique tailored to Kubernetes.

2. Related Work / (Literature Review)

The following section explores the methodologies and techniques that were detailed in our selected research papers for detecting DoS/DDoS as well as anomalies in Kubernetes environment and what we have learnt from them.

#### ****1. Machine Learning Approaches****

**Hierarchical Systems**

Anemogiannis proposed a hierarchical anomaly detection method that uses baseline metrics for Kubernetes resources, scoring anomalies based on severity for efficient resolution.

**Supervised Learning**

Du et al. employed supervised ML models (e.g., SVM, Random Forest, k-NN) to classify system behavior, improving anomaly detection through fault injection and time-series analysis.

**Optimized Isolation Forests**

Zou et al. enhanced anomaly detection by weighting resource metrics and adjusting monitoring intervals, improving real-time detection while reducing overhead.

#### ****2. Deep Learning Approaches****

**Hybrid Models**

Tien et al. combined CNNs and RNNs to identify both spatial and temporal patterns in Kubernetes resource metrics, achieving 96% detection accuracy.

**Auto-encoders**

Lin et al. introduced a classified distributed learning model using auto-encoders to detect attacks through reconstruction errors, tailored to application-specific behaviors.

**Self-Organizing Maps**

Forsberg applied unsupervised learning for dimensionality reduction and root-cause analysis, enhancing anomaly detection through dependency graphs and anomaly alignment.

#### ****3. Combining ML and DL****

Harlicaj proposed a two-stage approach: classical ML for baseline anomaly detection and DL (e.g., RNN or CNN) for identifying complex, multi-step attack patterns, leveraging the strengths of both techniques.

#### ****4. Future Directions****

The paper by Darwesh et al. forms the basis for this project, introducing a novel feature collection technique and hierarchical anomaly detection system tailored to Kubernetes. It combines Isolation Forests for efficient detection of high-dimensional anomalies and LSTM networks for time-series analysis. Building upon this framework, our project aims to enhance real-time monitoring and accuracy while reducing false positives, aligning with Kubernetes' layered architecture.

3. Methodology

To address the research gap identified in our review, we required system call (syscall) data and resource metrics from Kubernetes clusters. However, no publicly available dataset existed for syscalls and resource metrics in Kubernetes environments. Additionally, using managed Kubernetes clusters was not feasible due to the high costs associated with generating DDoS traffic, which significantly increases resource usage.

We deployed a local Kubernetes cluster comprising four nodes (1 master and 3 worker nodes) using an Ansible script. Each worker node was equipped with 16GB DDR5 RAM and a 16-core Intel Core i7 (12th Gen) processor. The cluster utilized Calico as the networking plugin and MetalLB for load balancing.

A three-node botnet, controlled by a command-and-control (C2) server, was configured to simulate various types of traffic. The C2 server dispatched commands to the botnet, enabling it to dynamically alternate between attack types. The botnet maintained process IDs of launched binaries to terminate previous attacks before initiating new ones.

We developed a simulation script which alternated randomly between six traffic modes; HTTP Flood, UDP Flood, TCP Flood, ICMP Flood, Normal Traffic and High Traffic. Each mode was chosen as they produced the most varying changes in the system calls and resource metrics under different attack conditions which made it easier for the models to classify the attack type and were run for a randomly selected duration of 180 to 300 seconds. For attacks, we used hping3 to generate UDP, TCP, and ICMP floods, and goldeneye.py for HTTP floods. Normal traffic simulated 50 concurrent connections, while high traffic simulated 200 concurrent connections, with ramp-up, sustain, and cool-down phases.

To provide a realistic target for these attacks, we deployed an intentionally vulnerable, open-source microservices-based e-commerce website, Juice Shop, on the Kubernetes cluster. This service was the primary target of TCP and HTTP floods, specifically targeting port 80.

Metrics were collected every second from all three worker nodes, resulting in approximately 16MB of data, with over 55,000 rows. The collected metrics were categorized into Resource Metrics and System Calls:

**Resource Metrics**

CPU utilization: node\_cpu\_seconds\_total

Memory metrics: node\_memory\_MemAvailable\_bytes, node\_memory\_MemTotal\_bytes

Network metrics: node\_network\_receive\_bytes\_total, node\_network\_transmit\_packets\_total

Disk metrics: node\_disk\_read\_bytes\_total, node\_disk\_written\_bytes\_total

System load: node\_load1, node\_load5, node\_load15

TCP/UDP statistics: node\_sockstat\_TCP\_inuse, node\_sockstat\_UDP\_inuse

**System Calls**  
mmap, munmap, accept, brk, bind, connect, chdir, clone, close, kill, listen, mkdir, open, poll, read, rename, recvfrom, select, socket, sendto, write.

To facilitate metric collection, we configured the Kubernetes nodes with a custom agent service, node exporter, and Falco.

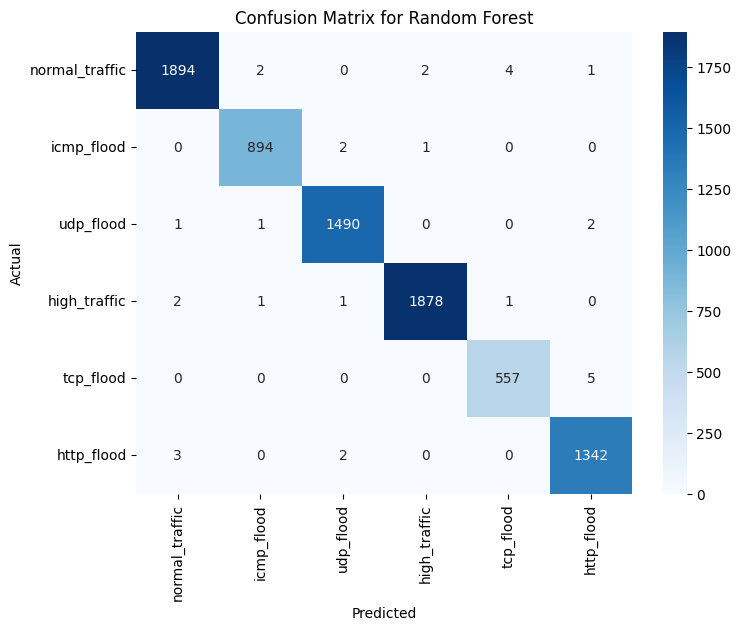
4. Testing and Results

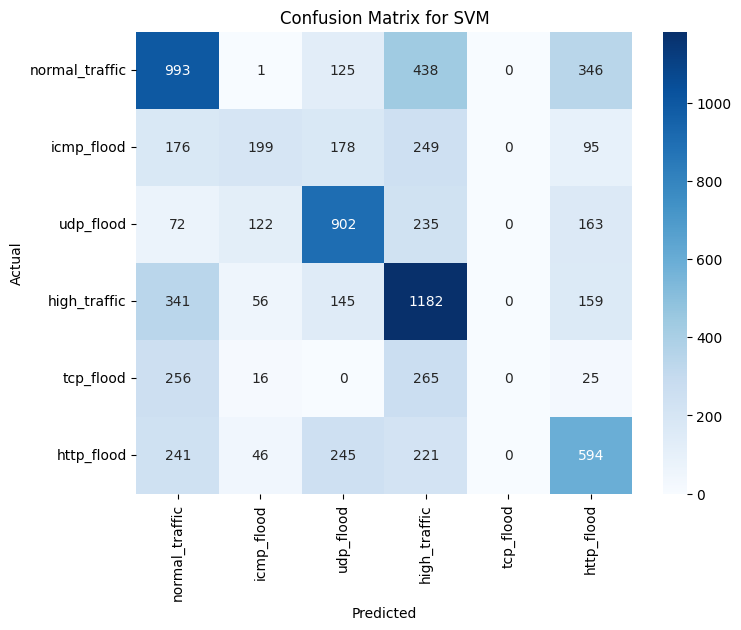
We implemented multiple machine learning models using a 90-10 train-test split and evaluated them with 5-fold cross-validation. Based on the research gap, the primary focus was on Random Forests, but additional models were implemented for the sake of comparison as the accuracy of Random Forests raised concerns about potential overfitting due to excessively high performance.

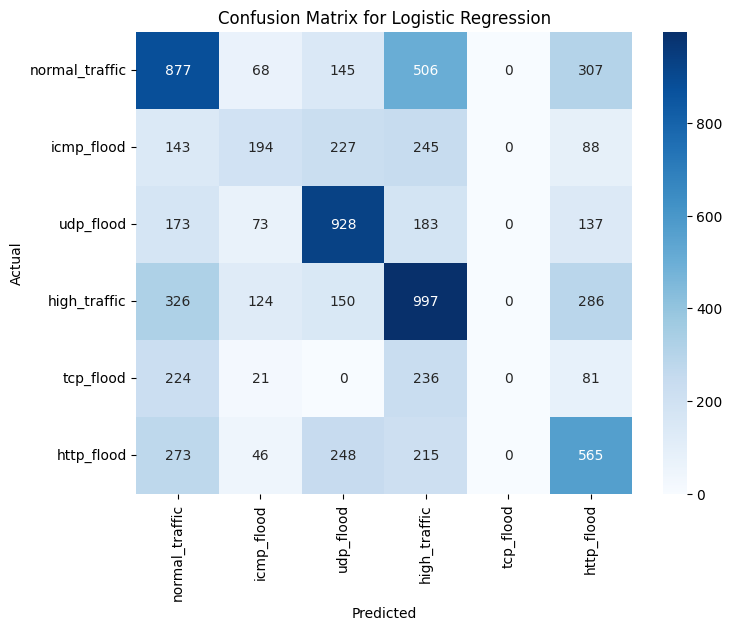
Random Forests achieved 99% accuracy. Decision Trees achieved 95% accuracy. Support Vector Machines (SVM) and Logistic Regression scored an accuracy below 50%. Results indicated that tree-based models (Random Forests and Decision Trees) outperformed non-tree-based models, aligning with the hierarchical nature of the collected metrics and the complex relationships between features.

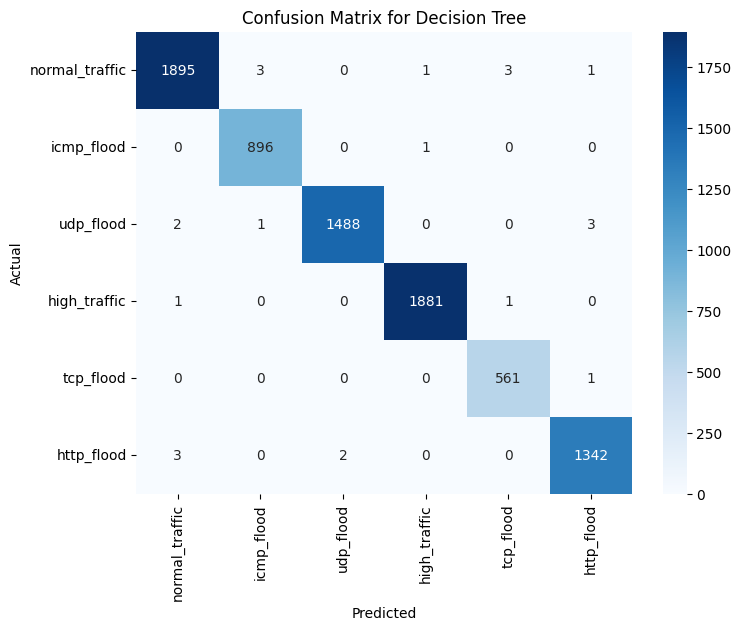
The tree-based models significantly outperformed other approaches, with Random Forest achieving the highest accuracy at 99%. This was validated through multiple metrics including F1-score, recall, and testing time. The performance of tree-based models suggests they are good for DDoS attack classification using system calls and resource metrics.

Below are given confusion matrices for each of the models mentioned. They clearly show that Random Forest matrix performs well, with minimal errors, in detecting and classifying different traffic conditions in the dataset. The same can be said for Decision Trees. While the SVM and Logistic Regression models demonstrates poor classification performance, with significant overlap and misclassifications between traffic types. It fails to capture the patterns necessary for effective classification in this dataset.

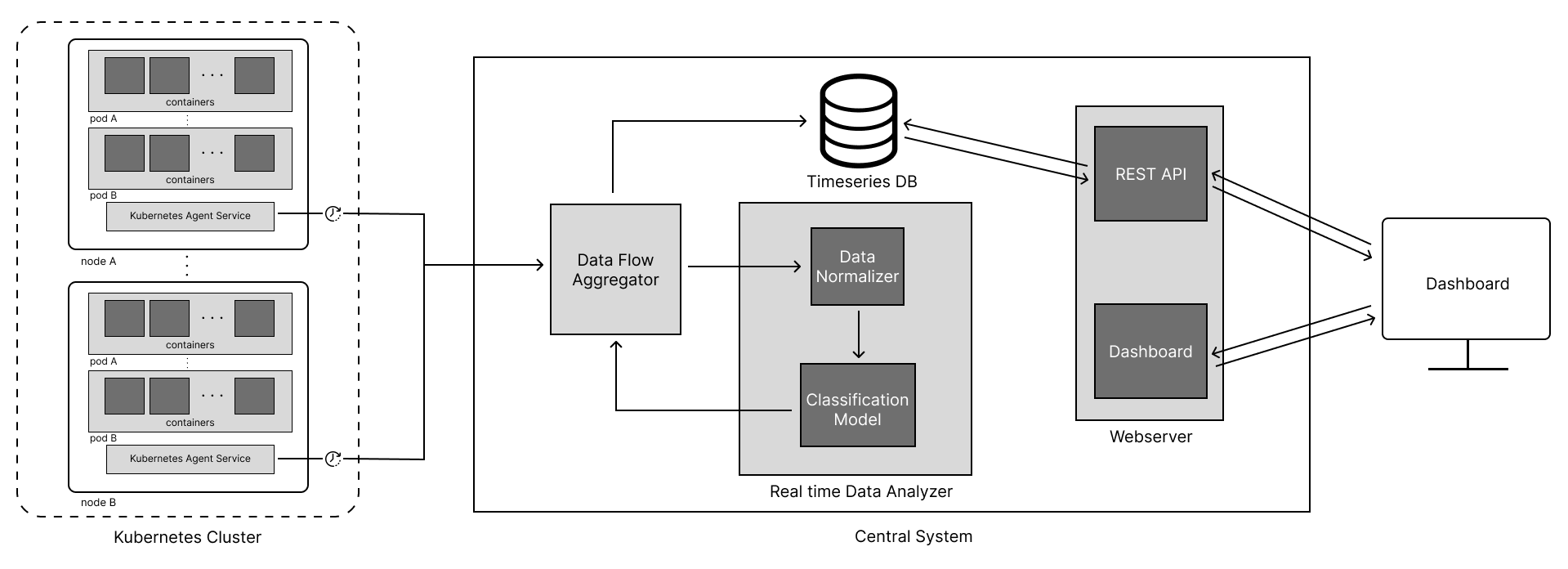








5. System Diagram



6. Goals for FYP-II

FYP-II will move towards finalization of the FYP, as well as the delivery of the following objects:

1. Codebase for the following
   1. Kubernetes Agent Service
   2. Webserver and Dashboard
   3. Central Module
2. Full Working System
3. Machine Learning and Deep Learning Trained Models
4. Research Paper

7. Conclusion

To conclude, we performed an elaborate review on our set of research papers to identify the gaps in research and practice in developed and deployed systems for detection of anomalies and DoS/DDoS attacks in Kubernetes environment. We also selected our base paper upon which we would further build in this regard.

Based on that paper’s research we worked on setting up a local cluster in Kubernetes to prepare our dataset comprised of syscalls and resource metrics. That dataset was then used to train the Random Forests model which incurred 99% results. Three more models were trained to settle the doubt that the initial results were accurate.

For the coming semester, we will be focusing on training the remaining models and creating a codebase for a Kubernetes Agent Service and deploying it as per our base paper’s research. A research paper documenting our research and practice would also be published. Along with an entirely completed working system, our project for final year will come to completion.

References

1. Anemogiannis, M. (2023). Hierarchical anomaly detection strategy for Kubernetes resources.
2. Du, M., et al. (2018). DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning.
3. Zou, H., et al. (2019). Docker container anomaly monitoring system based on optimized isolation forest algorithm.
4. Tien, C. W., Liao, J. W., Chang, S. C., & Kuo, S. Y. (n.d.). KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches.
5. Lin, Y., Tunde-Onadele, O., & Gu, X. (n.d.). CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized App.
6. Forsberg, J. (2022). Automatic anomaly detection and root cause analysis for microservice clusters.
7. Harlicaj, E. (n.d.). Anomaly Detection of Web-Based Attacks in Microservices.
8. Darwesh, G., Hammoud, J., & Vorobeva, A. A. (n.d.). A novel approach to feature collection for anomaly detection in Kubernetes environment and agent for metrics collection from Kubernetes nodes.