Supervised Classification of Cloud Workload Behavior Using Out-of-Band Performance Metrics

Maureen S. Van Devender, Thanh Le, Ryan Benton, Angela Buie, Ralph Mouawad and Zoe Steele
University of South Alabama, Mobile, AL, USA

mvandevender@southalabama.edu
tdl2321@jagmail.southalabama.edu
rbenton@southalabama.edu
aeb1323@jagmail.southalabama.edu
rlm2221@jagmail.southalabama.edu
zms2021@jagmail.southalabama.edu

Abstract: Technological advances have significantly improved the flexibility, scalability, and efficiency of computing resource utilization. The adoption of orchestration systems to manage virtual containers is one such example. In these environments, containers can be deployed for the duration of a task and then removed to release resources back to the system. While orchestrated containerization allows for efficient and flexible use of computing resources, concerns have been raised about the ability to detect anomalous behavior and to conduct forensics investigations in the environment. Monitoring temporal readings of system performance metrics offers a potential solution to anomalous behavior detection, and storing the performance readings away from the transient containers could be a solution to support forensic investigations. However, the resultant storage can become expansive over time, making it an expensive and often-impractical solution. In this research, we analyze temporal readings of out-of-band performance metrics gathered from various layers of the technology stack while trials of four distinct benchmarking workloads were running. Our objective was to determine if machine learning (ML) techniques could reliably distinguish between the running workloads based on the performance metrics. After conducting proof-of-concept experiments using various ML methods, we applied a random forest classifier to all readings and metrics in our datasets. The classifier was able to identify with a high degree of accuracy the workload that was running on the system based upon the readings. Furthermore, we found that a relatively small subset of the performance metrics was significant for accurate classification. This indicates that the problem of extensive storage and processing requirements could be improved. Our results indicate that a ML model trained on patterns of normal behavior could be used to monitor live metrics for the purpose of anomaly detection. These findings support the feasibility of using continuously collected performance metrics to enable real-time anomaly detection and improve forensic readiness in environments where logging may be transient or incomplete such as in orchestrated container systems.

Keywords: Cloud performance metrics, Containerized workloads, Machine learning, Workload classification, Digital forensics

1. Introduction

Container technology offers flexibility, scalability, and efficiency over virtual machines (VMs) in computing resource utilization and application deployment. As a result, containers have become the primary compute units powering cloud-native applications. (Susnjara, 2024). The global container technology market is growing - valued at $850 million in 2024 and expected to reach $4.48 billion by 2033 - representing an expected compound annual growth rate of 19.8 percent (Business Research Insights, 2025).

Parallelling the increase in cloud computing utilization, is an increase in cyber attacks in cloud environments. Kaspersky recently discovered a sophisticated malware attack targeting containerized environments (Kaspersky, 2025). Siloscape, discovered by Palo Alto Networks, is a heavily obfuscated malware that targets Kubernetes and is designed to escape the container and compromise the entire Kubernetes cluster (Prizmant, 2021). Several studies on container security have been conducted revealing the prevalence of attacks and security vulnerabilities in containerized environments (Spahn et al., 2023) (Sultan et al., 2019).

The transient nature of containers and other factors in cloud stack environments (Damshenas et al., 2012); (Jayalath et al., 2024) can diminish the ability to recognize anomalous behavior or to conduct forensic investigations when events occur (Sultan et al., 2019). ARMO reports that system administrators have difficulty recognizing anomalous behavior in containerized environments, and organizations report an increase in the frequency of cyber attacks (ARMO, 2025).

The growing adoption of containers, combined with difficulty in recognizing anomalous behavior and the rise in cyber attacks targeting cloud environments presents significant challenges for organizations. A key obstacle

is the ability to accurately characterize and detect normal system behavior within dynamic, containerized infrastructures. This research addresses that challenge by proposing a method for modeling normal behavior in cloud environments using performance metrics. Specifically, we analyze out-of-band performance metrics collected across multiple layers of two cloud stack deployments during the execution of four distinct workloads. Our investigation was guided by two primary objectives. First, we evaluate whether supervised ML can be applied to these metrics to accurately identify and distinguish between workloads. Second, we assess whether the number of metrics required for accurate classification can be reduced, making the approach more practical in real-world scenarios. Reducing the volume of metrics has the potential to decrease the storage and computational overhead required for ongoing monitoring and analysis, thereby improving scalability and efficiency.

The contributions of this work are:

- We demonstrate that supervised ML can accurately classify workloads based on out-of-bound performance metrics collected during execution, using two distinct cloud stack environments.
- We show that only a subset of performance metrics is necessary for effective workload classification, enabling a more efficient and scalable analysis.
- We compare results across the two environments to identify common metrics that contribute to workload differentiation, highlighting features that may generalize across deployment contexts.

The remainder of the paper is organized as follows: We review background and related work in Section 2. We describe the methodology in Section 3. Results are described in Section 4. In Section 5 we conclude and discuss future work.

2. Background and Related Work: Background Information and Related Work are Described Here

2.1 Background

Containerization is a virtual machine (VM) concept that differs from VM in that while both may utilize hardware virtualization, VMs include an operating system, whereas containers are executable units that package application code with its libraries and dependencies. This allows multiple containers to run in an environment where they share a single operating system kernel (Bui, 2015).

Container orchestration evolved from the work of Google as it sought to improve computer resource utilization and efficiency (Burns et al., 2016). Containers provide a means of deploying services, and orchestration is the means for managing them. Kubernetes (The Linux Foundation, n.d.), first developed by Google, now supported by an open-source community, is a popular container orchestration system. YARN (Yet Another Resource Negotiator) (*Apache Hadoop YARN*, n.d.), was also developed by Google to meet scheduling a resource management specific of the Hadoop environment (*Apache Hadoop*, n.d.). They are similar in concept, with Kubernetes being general purpose, and YARN being more specific to the Hadoop environment.

Monitoring and detecting anomalies in virtual environments, both VMs and containers, can take place using introspection (Garfinkel et al., 2003). Introspection provides the ability to observe and inspect the internal state and behavior of an environment from outside of the environment without having to modify the virtual environment itself. A commonly used tool for introspection is Prometheus (Prometheus, n.d.), an open source monitoring and alerting toolkit that allows for the collection and storage in a time-series data model of performance metrics at all levels of a cloud stack.

2.2 Related Work

Abed et al. (2015) used supervised learning to detect anomalous behavior in Linux containers by analyzing system calls. System calls are the typical way Linux containers communicate with the kernel. Using the Bag of System Calls (Kang et al., 2005) technique, they were successfully able to detect all tested anomalous behaviors with a very low false positive rate.

Cui and Umphress (2021) proposed an unsupervised introspection framework for containerized applications and services that enables behavioral anomaly detection by analyzing system calls with a focus on detecting malicious attacks. They chose the neural network LSTM (Long Short-Term Memory) autoencoder (Mirza and Cosan, 2018) as their classifier. The goal of the framework was to accurately classify benign and malicious behaviors using discrete system call traces. They were successful in doing so using various datasets of benign monitoring data as the classifier.

Watts et al. (2021) conducted an exploratory case study to evaluate whether introspection tools could acquire forensically viable data from multiple layers of a multi-server container orchestration environment. They successfully collected performance metrics across these layers. Furthermore, they demonstrated the ability to store the data outside of the running system.

Watts later extended this research with a comparison of two additional container orchestration platforms (Watts, 2020). This study aimed to assess whether performance metrics could be captured from other orchestration environments and to explore the utility of visualization techniques for analyzing resource utilization across platforms. Although the first study was published after the second, it was conducted earlier and laid the groundwork for this analysis that followed (Watts et al., 2021);(Watts, 2020).

Samir and Paul (Samir and Pahl, 2019) proposed a framework for operators of third party-party cloud services, where visibility into underlying performance metrics is limited. Their approach uses Hidden Markov Models (HMM) to map observed performance anomalies to potential root causes. This method is particularly relevant in environments where internal performance data is not directly accessible.

Previous research has applied both supervised and unsupervised learning techniques to detect anomalies by analyzing system calls in containerized environments. HMMs have been proposed as a solution when internal performance parameters are not observable. Other studies have demonstrated the feasibility of collecting extensive performance metrics using introspection tools, as well as using visualization techniques to assess performance in cloud-based container orchestration platforms. However, to our knowledge, no prior research has investigated the use of introspection data specifically to characterize the normal behavior of workloads in containerized environments. This study evaluates whether large volumes of such data can be effectively leveraged for this purpose.

3. Methodology

This study investigates the use of ML to recognize the normal behavior of workloads running within a cloud stack environment by analyzing out-of-band performance metrics. In addition to demonstrating the feasibility of this approach, we aim to optimize the set of metrics required to accurately distinguish between workloads. The datasets used in this study were created in prior research focused on visualizing performance data to support human analysis (Watts, 2020). The data consists of hundreds of performance metrics collected from multiple layers of two distinct cloud stack environments while four different computational workloads were executed at different times.

Both cloud stacks shared a common architecture consisting of three nodes incorporating Apache Spark for job management, Docker for containerization, and the Hadoop File System for file system storage. The difference was the orchestration layer, where the first environment used Kubernetes, and the second used YARN. Each environment ran four computational workloads from HiBench benchmarking suite (Huang et al., 2010): WordCount (WC), TeraSort (TS), Singular value Decomposition (SVD), and Random Forest (RF).

Each workload was run five times, with the average runtimes being approximately one hour. During each run, performance metrics were collected at 15-second intervals using Prometheus (Prometheus, n.d.), an open-source introspection tool. Metrics were stored separately for each run in an open-source version of the InfluxDB database (*InfluxDB*, 2022), resulting in 40 distinct databases - 20 from each cloud environment.

This process produced over 82 million readings in the Kubernetes environment and over 16 million readings in the YARN environment, capturing the state and behavior of systems throughout the execution period. A detailed breakdown of the number of readings per workload and run is provided in Table 1.

Our first step with the datasets was to conduct a data assessment and exploration. Using a combination of Python and InfluxQL, we assessed the data in the following manner. First, we made a count of the number of measures that were represented in each of the datasets. We determined that there were 918 measures in the Kubernetes cloud stack workload runs, and there were 460 measures in the YARN cloud stack workload runs. In analyzing the data, we discovered some runs were missing a few measures. For example, Kubernetes SVD run 1 contains four less measures than expected. Table 1 provides the number of measures found in each of the workloads and runs.

Our first two experiments served as proofs of concept, using subsets of performance metrics from the Kubernetes environment, to determine feasibility of using ML to classify the workload type. The third experiment expanded the analysis to include all available metrics from both the Kubernetes and YARN

environments. In each case, supervised ML models were applied to determine whether distinguishing characteristics in the data that would be useful to distinguishing the workloads.

Table 1: Data Collection Databases & Characteristics

Workload	Run	Kubernetes Workloads		YARN Workloads	
		# of Measures In Database	Total Number of Readings	# of Measures In Database	Total Number of Readings
Random Forest	1	918	5,020,897	460	1,688,268
Random Forest	2	918	5,051,652	460	1,538,698
Random Forest	3	918	5,059,807	459	1,691,722
Random Forest	4	918	5,724,634	460	1,558,258
Random Forest	5	918	5,753,480	460	1,705,080
Singular value Decomposition	1	914	1,601,589	460	251,544
Singular value Decomposition	2	917	1,361,938	460	151,355
Singular value Decomposition	3	918	2,098,395	460	94,564
Singular value Decomposition	4	918	1,589,882	460	109,694
Singular value Decomposition	5	918	1,578,196	460	77,693
TeraSort	1	914	7,725,992	460	914,430
TeraSort	2	918	7,240,915	460	739,023
TeraSort	3	918	7,588,652	460	715,044
TeraSort	4	918	7,629,585	459	700,745
TeraSort	5	918	8,148,981	460	2,206,421
WordCount	1	917	1,837,069	459	486,884
WordCount	2	914	1,859,804	459	472,124
WordCount	3	918	1,719,260	460	511,818
WordCount	4	918	1,788,422	461	406,300
WordCount	5	913	1,726,899	459	482,633
	Total Readings		82,106,049		16,502,298

3.1 Experiment One – Feature Prediction

In our first experiment, we focused on a subset of performance data specifically categorized as CPU-related metrics. These 19 metrics, identified through our initial data assessment, were selected from the original pool of over 900. (See Table 5 in the Appendix for a complete list.) The goal in this experiment was to determine whether ML could be used to predict the value of each feature using the remaining 18 during ordinary workload execution. The idea is that if features could be predicted from others, it would support the idea there is information within the features that could be used for workload prediction.

For each feature, A neural network-based machine learning model was developed to predict its value given the remaining 18 CPU metrics across the four benchmark workloads. To prepare the data, JavaScript is used to extract the data from InfluxDB and converted to JSON format, including statistical summaries for each of the metrics by workload. Regression analysis was applied to the JSON-formatted data to produce these summaries. The model was trained on the first four of the five databases associated with workload and tested on the fifth database.

3.2 Experiment Two – Predicting Workload Types Using a Subset of Features

In the second experiment, we investigated whether classification models could effectively distinguish between the workloads based on network metrics, and if so, which model performed best. We tested six commonly used classification models, as shown in Table 2. This experiment focused on a subset of metrics categorized as network-related during the data exploration phase. This refinement reduced the number of considered measurements from over 900 to 51. The measures used in this analysis are listed in **Table 6** in the Appendix. A non-random results with this approach support the idea of examining all measures for workload prediction.

Table 2: Machine Learning Classification Models used in Second Machine Learning Analysis

Logistic Regression (LR)
Linear Discriminant Analysis (LDA)
K-Nearest Neighbors (KNN)
Classification and Regression Trees (CART)
Gaussian Naive Bayes (NB)
Support Vector Machines (SVM)

To prepare the data for analysis, several preprocessing steps were performed. First, the data was extracted from the InfluxDB databases into a single CSV file using the InfluxQL query language. For each reading, the features of measurement name, measurement value, and timestamp were selected. A workload identifier was added to each record to label the workload from which the data was extracted. Finally, the timestamp values – originally recorded in 15-second intervals – were cleaned and formatted to provide the time in seconds and rounded to the nearest 15 second mark to ensure consistency across records. All models were tested on both unshuffled (temporal order) and shuffled (random order) data.

3.3 Experiment Three – Random Forest Classifier on all Metrics

In the third experiment, a comprehensive analysis was conducted for each of the two environments using the full set of metrics and all available readings. This included over 900 metrics from the Kubernetes environment and 460 metrics from the YARN environment. A Random forest classifier (Breiman, 2001), based on the CART methodology, was applied to assess whether workloads could be accurately classified based on the complete set of measurements, and to identify which metrics were most influential in the classification.

The significance of this analysis is twofold: first, it evaluates the accuracy of classification using the complete set of available metrics; second, it explores the potential to reduce the volume of introspection data required for accurate classification. Such a reduction could improve the efficiency and practicality of using introspection data to characterize normal system behavior in real-world settings.

To prepare the data for analysis, the following steps were taken separately for the Kubernetes and YARN measures. First, the data was extracted from the InfluxDB databases into CSV files to facilitate data analysis. The extraction was implemented using the Go programming language (*The Go Project - The Go Programming Language*, n.d.), which provided significant performance improvements over Python for this task. Next, the data was normalized and transformed using Python. Normalization involved scaling the measurement values to a range between zero and one.

The data transformation involved summarizing the 15-second readings into one-minute intervals. For each minute, the mean value of the metric was calculated. In addition, a workload identifier was added to each record. This resulted in two pivot tables - one for each environment - containing one record for each minute of execution for each workload, and each column (feature) representing a specific metric. Once the data was normalized and transformed, a Random Forest classifier was trained using Python. The model was configured with 100 estimators (trees), and the dataset was split randomly into 60% for training and 40% for testing.

4. Results: The Results of Each Experience are Described Here

4.1 Experiment One – Neural Network and Regression on CPU Metrics

The model successfully predicted the expected value ranges for each of the 19 CPU metrics across all four workloads, achieving an accuracy rate exceeding 99%. This high level of accuracy suggests that meaningful relationships exist among the measures, and that these relationships are stable within specific workload contexts. These results support the hypothesis that performance metrics can be used to characterize normal

system behavior and to potentially classify workloads as well. These findings justified proceeding with additional experiments aimed at evaluating classification performance using performance metrics. The system for building and testing a model for measurement prediction is available online (*Cloud Stack Regression Model for Predicting Normal CPU Measurement Ranges*, n.d.).

4.2 Experiment Two - Classification models with Network Metrics

Table 3 presents the results of applying six classification models to the network metrics dataset. All the models performed well above the 25% baseline expected for random guessing in a four-class classification problem. The Classification and Regression Trees (CART) model (Breiman et al., 1984) outperformed the others in both the shuffled and the unshuffled datasets, demonstrating robustness to the temporal ordering of data.

Table 3: Results of Test with Unshuffled Data.

	Unshuffled		Shuffled	
Classification Model	***Mean Fitness***	***St. Dev.***	***Mean Fitness***	***St. Dev.***
Logistic Regression (LR)	0.525761	0.016374	0.51955	0.030064
Linear Discriminant Analysis (LDA)	0.529154	0.015283	0.519553	0.03319
K-Nearest Neighbors (KNN)	0.424987	0.030443	0.482725	0.034742
Classification and Regression Trees (CART)	0.616275	0.028789	0.6122	0.038508
Gaussian Naive Bayes (NB)	0.474817	0.02506	0.481058	0.031984
Support Vector Machines (SVM)	0.440251	0.027811	0.49162	0.038854

This consistency suggest that the distinguishing characteristics of each workload are embedded in the metric values themselves, rather than the sequence of their occurrence. These results provided a strong rationale for proceeding to a more comprehensive analysis using the Random Forest algorithm, which is an ensemble method based on the CART framework.

4.3 Three – Random Forest Classifier on all Metrics

Using the random forest classifier, we were able to correctly classify the workloads with over 98% accuracy across 15 trials in both cloud stack environments.

Next, we analyzed which metrics contributed most to the classifier's performance. Figures 1 and 2 display the top 15 metrics by mean importance for Kubernetes and YARN environments based on 15 runs of the classifier. For clarity, only the top 15 metrics are shown in each plot. In both environments, importance scores drop sharply from the most important metric to the 15th metric, suggesting that only a small number of metrics significantly influence model accuracy. This implies that the model's efficiency might be improved by reducing the number of features evaluated.

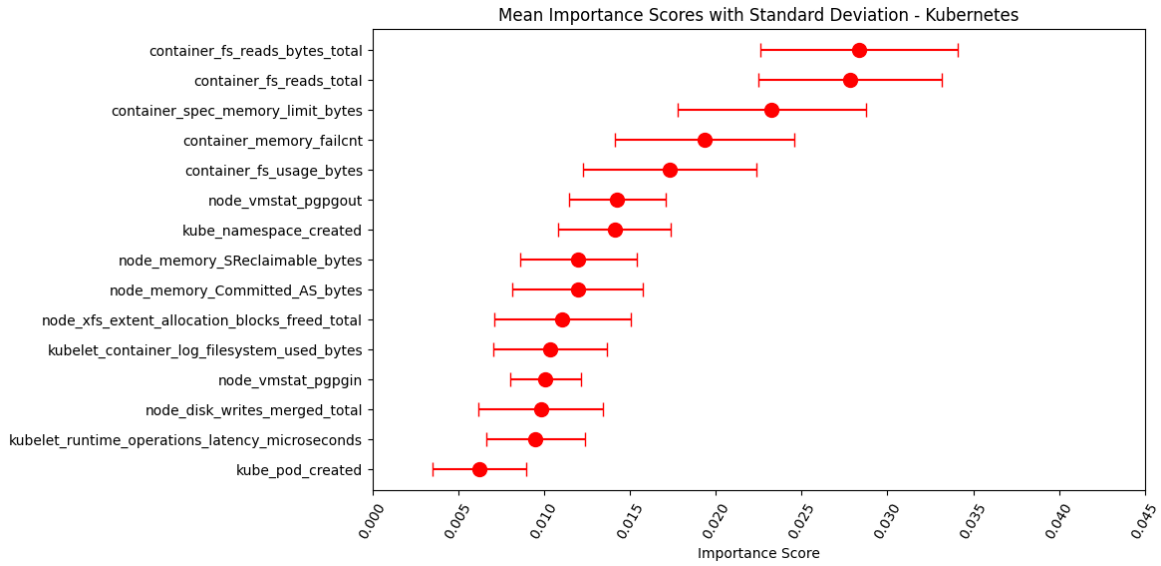

Figure 1: Kubernetes - Most Significant Measures.

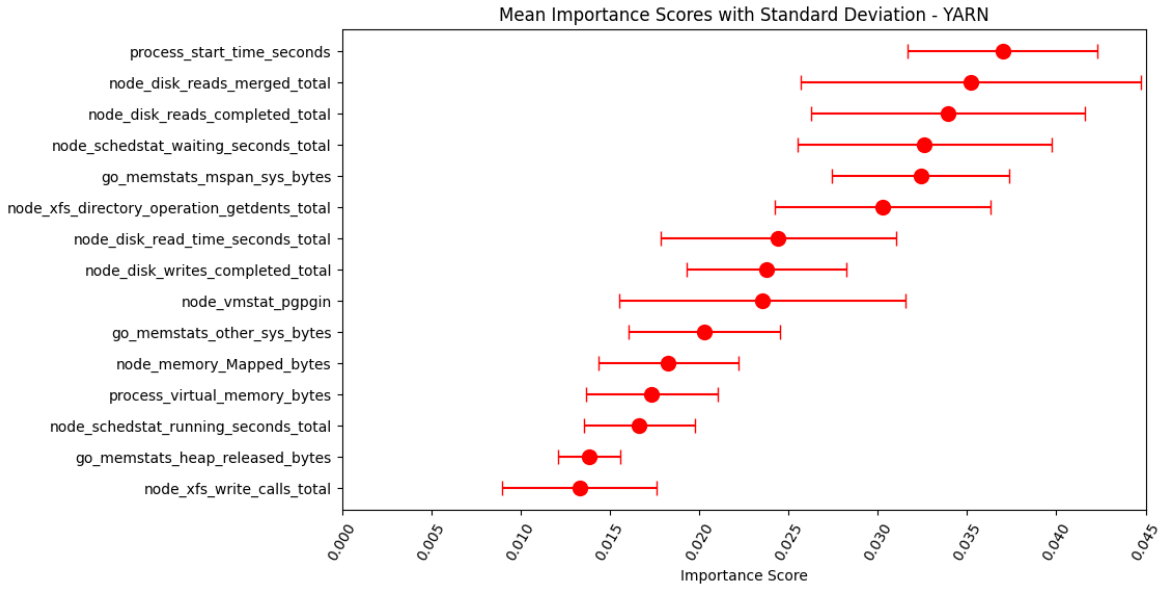

Figure 2: YARN - Most Significant Measures

We sought to determine if there were characteristics for comparison in the significant measures in the two environments. Because the measures were different for each environment, we chose to categorize the measures on two dimensions. The first is computing resource type, with categories including I/O, memory, CPU, storage, latency. The second is the abstraction layer of the system architecture (e.g., node, container, application). Each metric was assigned a weight based on its position in the importance ranking. Tables 7 and 8 in the Appendix show these metrics along with their assigned categories and weights. The metrics labeled “Not Useful” are those that did not contribute to workload differentiation. For example, the YARN metric process_start_time_seconds, indicates the epoch time when the job started, and does not meaningfully distinguish the workload.

In the analysis by computing resource category, I/O and memory metrics emerged as the most influential in workload prediction for both environments (see Table 3).

Table 3: Metric Importance by Resource Category

	Kubernetes		Yarn	
Metric Compute Resource Category	**Category Weight**	**% of Total**	**Category Weight**	**% of Total**
I/O	43	35.83%	55	45.83%
Memory	41	34.17%	35	29.17%
CPU	0	0.00%	3	2.50%
Latency	2	1.67%	12	10.00%
Storage	11	9.17%	0	0.00%
Not Useful	23	19.17%	15	12.50%
Total	120	100.00%	120	100.00%

In the analysis by abstraction layer, we found environment-specific trends. For the Kubernetes environment, the most significant metrics came from the node and container layers. In contrast, in the YARN environment, the most important metrics were found at the node and application layers. Despite these differences, node-level metrics were consistently the most important across both environments. Table 4 presents the results of this analysis.

Table 4: Metric Importance by Abstraction Layer Category

	Kubernetes				Yarn			
Abstraction Layer	**Number of Measures**	**% of Total Measures**	**Category Weight**	**% of Weight**	**Number of Measures**	**% of Total Measures**	**Category Weight**	**% of Weight**
Node	6	40.00%	38	31.67%	10	66.67%	82	68.33%
Application	0	0.00%	0	0.00%	3	20.00%	19	15.83%
Container	5	33.33%	65	54.17%	1	6.67%	4	3.33%
Platform	4	26.67%	17	14.17%	0	0.00%	0	0.00%
Not Useful	0	0.00%	0	0.00%	1	6.67%	15	12.50%

5. Conclusion and Future Work

This study demonstrates that supervised ML can be used to distinguish workloads operating within containerized orchestration environments through the analysis of out-of-band performance measures. In addition to accurate classification, we identified a small subset metrics that contributed most significantly to workload differentiation.

While the specific metrics found to be effective in our analysis may not generalize across all environments or workload types, our results suggest that a comparably limited set of distinguishing features may be derivable in other contexts. In particular, I/O and memory-related metrics are most influential in our models. In addition, the importance of metrics across multiple abstraction layers suggests the need for multi-layer observability in anomaly detection systems, where anomalies may manifest at different levels depending on the workload or threat model.

Modeling performance characteristics of normal workload behavior, as we have done, is a promising approach to detecting anomalous behavior - both benign and malicious – in containerized systems. Such detection capabilities may prove valuable in real-time monitoring as well as forensic analysis. The ability to detect deviations from expected behavior could assist system operators in early intervention and aid investigators in reconstructing events that preceded an incident.

Future research will explore the generalizability of our findings to a broader range of environments and workload types. In addition, more analysis of feature importance should be conducted to validate and test the results. Finally, our goal is to extend this work toward the detection of anomalous workloads in operational containerized systems.

Partial funding provided by the Center for Advanced Research in Forensics Science.

Ethics declaration: Ethical clearance was not required for this research.

AI declaration: AI tools were not used in the creation of this paper.

References

Abed, A. S., Clancy, T. C. and Levy, D. S. (2015) 'Applying Bag of System Calls for Anomalous Behavior Detection of Applications in Linux Containers.' *In 2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–5.

Apache Hadoop (n.d.). [Online] [Accessed on 29th May 2025] https://hadoop.apache.org/.

Apache Hadoop YARN (n.d.). [Online] [Accessed on 29th May 2025] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

ARMO (2025) *The State of Cloud Runtime Security, 2025 Edition*.

Breiman, L. (2001) 'Random forests.' *Machine learning*. Springer, 45 pp. 5–32.

Breiman, L., Friedman, J., Olshen, R. A. and Stone, C. (1984) *Classification and regression trees*. New York, NY, USA: Chapman and Hall/CRC.

Bui, T. (2015) 'Analysis of docker security.' *arXiv preprint arXiv:1501.02967*.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016) 'Borg, Omega, and Kubernetes.' *Queue*. ACM New York, NY, USA, 59(5) pp. 50–57.

Business Research Insights (2025) *Container Technology Market Size, Share - [2025 To 2033]*. [Online] [Accessed on 9th June 2025] https://www.businessresearchinsights.com/market-reports/container-technology-market-106710.

Cloud Stack Regression Model for Predicting Normal CPU Measurement Ranges (n.d.). [Online] [Accessed on 16th July 2024] https://ralphato.github.io/TensorFlow/.

Cui, P. and Umphress, D. (2021) 'Towards Unsupervised Introspection of Containerized Application.' *In Proceedings of the 2020 10th International Conference on Communication and Network Security*. New York, NY, USA: Association for Computing Machinery (ICCNS '20), pp. 42–51.

Damshenas, M., Dehghantanha, A., Mahmoud, R. and bin Shamsuddin, S. (2012) 'Forensics investigation challenges in cloud computing environments.' *In Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*. IEEE, pp. 190–194.

Garfinkel, T., Rosenblum, M., and others (2003) 'A Virtual Machine Introspection Based Architecture for Intrusion Detection.' *In Ndss*. San Diega, CA, pp. 191–206.

Huang, S., Huang, J., Liu, Y., Yi, L. and Dai, J. (2010) 'Hibench: A representative and comprehensive hadoop benchmark suite.' *In Proc. ICDE Workshops*, pp. 41–51.

InfluxDB (2022) InfluxData. [Online] [Accessed on 11th July 2024] https://www.influxdata.com/home/.

Jayalath, R., Ahmad, H., Goel, D., Shuja Syed, M. and Ullah, F. (2024) 'Microservice Vulnerability Analysis: A Literature Review With Empirical Insights.' *IEEE Access*, 12 pp. 155168–155204.

Kang, D.-K., Fuller, D. and Honavar, V. (2005) 'Learning classifiers for misuse and anomaly detection using a bag of system calls representation.' *In Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, pp. 118–125.

Kaspersky (2025) *Kaspersky uncovers Dero crypto miner spreading via exposed container environments*. /. [Online] [Accessed on 10th June 2025] https://www.kaspersky.com/about/press-releases/kaspersky-uncovers-dero-crypto-miner-spreading-via-exposed-container-environments.

Mirza, A. H. and Cosan, S. (2018) 'Computer network intrusion detection using sequential LSTM Neural Networks autoencoders.' *In 2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4.

Prizmant, D. (2021) 'Siloscape: First Known Malware Targeting Windows Containers to Compromise Cloud Environments.' Unit 42. 7th June. [Online] [Accessed on 10th June 2025] https://unit42.paloaltonetworks.com/siloscape/.

Prometheus (n.d.) *Prometheus*. [Online] [Accessed on 11th July 2024] https://prometheus.io/.

Samir, A. and Pahl, C. (2019) 'Anomaly detection and analysis for clustered cloud computing reliability.' *CLOUD COMPUTING*, 2019 p. 120.

Spahn, N., Hanke, N., Holz, T., Kruegel, C. and Vigna, G. (2023) 'Container orchestration honeypot: Observing attacks in the wild.' *In Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 381–396.

Sultan, S., Ahmad, I. and Dimitriou, T. (2019) 'Container Security: Issues, Challenges, and the Road Ahead.' *IEEE Access*, 7 pp. 52976–52996.

Susnjara, S. (2024) *What Are Containers? | IBM*. [Online] [Accessed on 9th June 2025] https://www.ibm.com/think/topics/containers.

The Go Project - The Go Programming Language (n.d.). [Online] [Accessed on 17th July 2024] https://go.dev/project.

The Linux Foundation (n.d.) *Kubernetes*. Kubernetes. [Online] [Accessed on 29th May 2025] https://kubernetes.io/.

Watts, T., Benton, R., Shropshire, J. and Bourrie, D. (2021) 'Insight from a Containerized Kubernetes Workload Introspection.' *In 2021 54th Hawaii International Conference on System Sciences*. Maui, Hawaii.

Watts, T. H. (2020) *Visualization of Orchestrated Containerized Workloads*. PhD Thesis. University of South Alabama.

Appendix 1

Table 5: Subset of Metrics Used in Initial Machine Learning Analysis

CPU Metrics
attachdetach_controller_forced_detaches
bootstrap_signer_rate_limiter_use
container_cpu_cfs_periods_total
container_spec_cpu_period
container_spec_cpu_quota
container_spec_cpu_shares
machine_cpu_cores
node_cpu_core_throttles_total
node_cpu_seconds_total
node_hwmon_chip_names
node_hwmon_temp_max_celsius
process_cpu_seconds_total
scheduler_binding_duration_seconds_bucket
scheduler_binding_latency_microseconds_sum
scheduler_e2e_scheduling_duration_seconds_bucket
scheduler_e2e_scheduling_latency_microseconds_sum
scheduler_scheduling_algorithm_duration_seconds_bucket
scheduler_scheduling_latency_seconds_sum

Table 6: Subset of Metrics Used in Second Machine Learning Analysis

Network Metrics	
aggregator_openapi_v2_regeneration_count	kubelet_docker_operations_total
apiserver_admission_controller_admission_duration_seconds_bucket	kubelet_http_inflight_requests
apiserver_watch_events_total	kubelet_http_requests_total
container_network_receive_bytes_total	kubelet_network_plugin_operations_duration_seconds_bucket
container_network_transmit_packets_total	kubelet_network_plugin_operations_latency_microseconds_sum
coredns_build_info	kubelet_pleg_relist_duration_seconds_bucket
coredns_plugin_enabled	kubelet_pleg_relist_latency_microseconds_sum
etcd_object_counts	kubelet_pod_start_duration_seconds_bucket
etcd_request_latencies_summary_sum	kubelet_pod_worker_start_latency_microseconds_sum
kube_daemonset_created	kubelet_running_container_count
kube_daemonset_updated_number_scheduled	kubelet_runtime_operations_total
kube_deployment_created	net_conntrack_dialer_conn_attempted_total
kube_deployment_status_replicas_updated	net_conntrack_listener_conn_closed_total
kube_node_created	node_netstat_Icmp_InErrors
kube_node_status_condition	Network Metrics (continued)
kube_pod_completion_time	node_netstat_UdpLite6_InErrors

Network Metrics	
kube_pod_status_scheduled_time	node_network_address_assign_type
kube_replicaset_created	node_network_up
kube_replicaset_status_replicas	node_sockstat_FRAG_inuse
kube_statefulset_created	node_sockstat_UDPLITE_inuse
kube_statefulset_status_update_revision	prometheus_http_request_duration_seconds_bucket
kubelet_cgroup_manager_duration_seconds_bucket	prometheus_http_response_size_bytes_sum
kubelet_cgroup_manager_latency_microseconds_sum	prometheus_sd_consul_rpc_duration_seconds_count
kubelet_container_log_filesystem_used_bytes	prometheus_sd_updates_total
kubelet_containers_per_pod_count_sum	prometheus_target_interval_length_seconds
kubelet_docker_operations	prometheus_target_sync_length_seconds_sum

Table 7: Kubernetes Top 15 Importance Measures - Categories & Weights

Importance Ranking	Measure Name	Computing Resource	Measure Weight	Abstraction Layer
1	container_fs_reads_bytes_total	I/O	15	Container
2	container_fs_reads_total	I/O	14	Container
3	container_spec_memory_limit_bytes	not useful	13	Container
4	container_memory_failcnt	memory	12	Container
5	container_fs_usage_bytes	I/O	11	Container
6	node_vmstat_pgpgout	memory	10	Node
7	kube_namespace_created	not useful	9	Platform
8	node_memory_SReclaimable_bytes	memory	8	Node
9	node_memory_Committed_AS_bytes	memory	7	Node
10	node_xfs_extent_allocation _blocks_freed_total	storage	6	Node
11	kubelet_container_log_filesystem_used_bytes	storage	5	Platform
12	node_vmstat_pgpgin	memory	4	Node
13	node_disk_writes_merged_total	I/O	3	Node
14	kubelet_runtime_operations_latency_microseconds	latency	2	Platform
15	kube_pod _created	not useful	1	Platform

Table 8: YARN Top 15 Importance Measures - Categories & Weights

Importance Ranking	Measure Name	Computing Resource	Measure weight	Abstraction Layer
1	process_start_time_seconds	not useful	15	Not Useful
2	node_disk_reads_merged_total	I/O	14	Node
3	node_disk_reads_completed_total	I/O	13	Node
4	node_schedstat_waiting_seconds_total	latency	12	Node
5	go_memstats_mspan_sys_bytes	memory	11	Application
6	node_xfs_directory_operation_getdents_total	I/O	10	Node
7	node_disk_read _time_seconds_total	I/O	9	Node
8	node_disk_ writes_completed_total	I/O	8	Node
9	node_vmstat_pgpgin	memory	7	Node

Importance Ranking	Measure Name	Computing Resource	Measure weight	Abstraction Layer
10	go_memstats_other_sys_bytes	memory	6	Application
11	node_memory_Mapped_bytes	memory	5	Node
12	process_virtual_memory_bytes	memory	4	Process
13	node_schedstat_running_seconds_total	CPU	3	Node
14	go_memstats_heap_released_bytes	memory	2	Application
15	node_xfs_write_calls_total	I/O	1	Node