An Anomaly-based Detection System for Monitoring Kubernetes Infrastructures

Josue Genaro Almaraz-Rivera

Abstract—**Network monitoring is crucial to analyze infrastructure baselines and alert whenever an abnormal behavior is observed. However, human effort is limited in time and scope since many variables must be considered in real-time. In addition, infrastructures such as Kubernetes are complex by nature since they do not consider fixed equipment from which to gather data; instead, these infrastructures consider distributed, event-driven, and ephemeral containers that make it complicated to capture and track metrics. Artificial Intelligence models have demonstrated high detection rates for anomaly detection; therefore, there is a need to design and implement a global solution to collect complex data and orchestrate the whole Machine Learning Operations workflow. This document shares the findings and learnings from defining a cloud-native Artificial Intelligence infrastructure at Aligo to develop an anomaly-based detection system for monitoring on-premise Kubernetes infrastructures. After Chaos Engineering experiments, it is shown that the resulting deployed system is strong when alerting outliers and that an end-to-end infrastructure has been developed for conducting future Artificial Intelligence projects at the company.**

Index Terms—**Anomaly Detection, Cloud Native, Deep Learning, Kubernetes, LATAM-DDoS-IoT Dataset, Machine Learning, One-Class Classification, Online Learning**

I. Introduction

Machine fault detection is essential to trigger alarms whenever a device or equipment exhibits abnormal behavior. This monitoring task is notorious in critical production services, where an application must remain available for the users. Such a level of telemetry is necessary to be automatic and also based on Artificial Intelligence (AI) since Machine Learning (ML) and Deep Learning (DL) models have demonstrated high detection rates for anomaly detection [1].

Kubernetes (K8s) [2] is an open-source technology from the Cloud Native Computing Foundation (CNCF) [3]. It provides increased stability and autoscaling options [4], in addition to a better workloads orchestration of different instances, and it is going “under the hood” like Linux [5]. At Aligo [3], the infrastructure and applications management is done using K8s.

As part of a summer project, Aligo wanted to increase the visibility of its on-premise K8s infrastructure, adding AI to the operations to reduce the number of manual monitoring tasks. Measurement of the instances during normal operational state (i.e., the negative class) was easy to obtain; therefore, as a solution approach, it was decided to implement One-Class Classification (OCC) [6] techniques due to their advantages in dealing with imbalanced data and unknown positive or outlier classes [7].

Josue Genaro Almaraz-Rivera is with Aligo Defensores Informaticos, Medellin, Colombia e-mail: genaroalmaraz@exatec.tec.mx

This work is presented as a case study, showing the findings and learnings from defining an AI infrastructure inside of Aligo to create and implement an anomaly-based detection system for monitoring Kubernetes. It shows that AI systems are not just models since the building blocks include more tasks, such as infrastructure serving, configuration, and data preparation [8].

The experimentation part is divided into offline learning and online learning [9]. The offline learning experiment consisted of training OCC models using the novel LATAM-DDoS-IoT dataset [10], [11]; local data from Aligo’s on-premise K8s infrastructure was collected for online learning. These couple of tasks served for a robust classification performance evaluation of the created solution. Although a more established dataset could have been used for the online learning experiment, the data for this final test was aimed to be fully collected in real-time from Aligo’s infrastructure since it was the monitoring target.

The LATAM-DDoS-IoT dataset was created in 2022 in collaboration with Tecnologico de Monterrey and Universidad de Antioquia and contains 799,187 normal flows from one of Aligo’s production networks, along with millions of denial of service (DoS, DDoS) attacks [12] flows based on UDP, TCP, and HTTP protocols. Due to the data collection procedure and the advantage of having a labeled dataset to ease the evaluation of classification models, it was a good choice for initial testing.

Therefore, the main contributions of this work can be summarized as follows:

- A new detection system based on Artificial Intelligence models tested on the novel LATAM-DDoS-IoT dataset and local data to identify outliers on on-premise Kubernetes infrastructures.
- A tested cloud-native workflow proposal for conducting AI-based projects on Kubernetes environments.

The remaining sections of this document are structured as follows: the related work is presented in section II. The defined methodology can be read in section III. Section IV shows the results and discussion of the selected anomaly detection models and the created training and testing pipelines. Finally, in section V, the conclusions and future work are presented.

II. Related Work

One-Class Classification is a promising research area for anomaly detection [13]. This classification type focuses on modeling the objects of one class to learn a boundary that separates them from any other object.

OCC has been applied in network-based intrusion detection [14], in the protection against masquerade attacks [15], as well as in the profiling of anomalous behavior in industrial control systems [16]. Additionally, it is a promising strategy for detecting zero-day attacks [17].

Recent works regarding the deployment of Machine Learning on Kubernetes environments [18]–[22], suggest the implementation of end-to-end systems to alleviate the data scientists job when getting insights at production-level. Nevertheless, it seems that a classification-based solution is not available that provides an extensive benchmark of several AI orchestrators and data collection tools, nor step-by-step definition, implementation, and testing of a cloud-native solution for infrastructure monitoring.

Therefore, this work presents an OCC cloud-native approach for Kubernetes monitoring, starting with selecting the Artificial Intelligence and data tools required and ending with deploying and testing the created solution.

In addition, due to the One-Class Classification approach followed for anomaly detection, dataset balancing does not become a critical issue [23]. It is opposed to other works that perform binary or multi-class classification relying on potentially biased results toward the majority class due to the balancing process omitted [24], [25].

The following section provides more details regarding the proposed methodology for creating the anomaly-based detection system.

III. Methodology

This section presents the benchmark of different Artificial Intelligence orchestrators and data collection tools, the anomaly detection models selection for One-Class Classification, and the explanation of the hyperparameter tuning process and feature engineering for training and testing.

A. Selection of Artificial Intelligence and Data Tools

Machine Learning workflows are essential when integrating ML Operations (MLOps) [26] into a technology ecosystem. That can be explained as these workflows outline the steps to collect and process data and build, deploy, and manage Machine Learning and Deep Learning models. Different workflow configurations exist, but not every project requires the same steps. See Fig. 1 for the defined workflow for the training process and Fig. 2 for the one defined for testing. In Fig. 1, the training dataset is created and preprocessed for then fitting the anomaly detection models in parallel; Fig. 2 shows the workflow to detect outliers in real-time, where the *Condition* node is the voting system explained in the next subsection.

These training and testing workflows can be orchestrated, and there are different tools on the market to do so, namely Kubeflow [27], Apache Airflow [28], and MLflow [29] [30]. See Table I for a benchmark of these options. Regarding the five criteria used for this evaluation, it is important to clarify some of these aspects:

1) Free to use: it does not consider infrastructure costs for operability.
2) Metric-level analysis support: it considers the integration of the orchestrator with other tools for data injection.
3) E2E ML workflow orchestration: it includes the end-to-end process, from data collection to model versioning and deployment. To do so, the orchestrators can use third-party services.

Based on Table I, both Kubeflow and Airflow are useful for this project; however, Kubeflow focuses on Machine Learning tasks, while Airflow is a generic orchestration platform [31]. In addition, Kubeflow architecture [32] includes native support for Jupyter Notebooks [33], data tools such as Istio [34] and Prometheus [35], as well as metadata management for the models. This explains the thoughts behind choosing Kubeflow as the AI orchestrator.

Several options were also analyzed for data collection, including Istio and Prometheus, but also InfluxDB [36] and OpenTSDB [37], due to previous personal experience with them. See Table II for this benchmark. A few criteria definitions are also worth clarifying in this table:

1) Kubernetes monitoring: distributed, event-driven, and ephemeral K8s containers [38] make it complicated to capture and track metrics, so a data monitoring tool must be able to handle this nature.
2) Free to use: as with the AI orchestrators benchmark, this aspect does not consider infrastructure costs for operability.
3) Metrics support: the counterpart is logging monitoring. Metrics are needed to create a set of features to help model the baseline of the K8s instances and train AI algorithms.
4) Time-series format: tracking the metrics' variations across time is needed to understand their behavior better.
5) GitHub stars: data up to July 1st, 2022, was considered and used as a popularity metric (i.e., with more popularity, more community around the tool).

Based on Table II, it was concluded that Prometheus is a suitable and popular tool for Kubernetes monitoring. Furthermore, Prometheus is a CNCF graduated project, indicating a high maturity level that reflects strong adoption and a secure code base [43].

B. Anomaly Detection Models

For the selection of the anomaly detection models for OCC, it was considered the capabilities the algorithms offer for fast prediction times and dealing with high-dimensional problems, as well as the proven application from state of the art. Regarding Machine Learning, One-class Support Vector Machines (OSVMs) [44] and Isolation Forest (IF) [45] were chosen; for Deep Learning, Autoencoders (AEs) [46] were tested. PyTorch [47] and scikit-learn [48], [49] are the APIs selected for coding the aforementioned models. PyTorch is a popular and flexible Deep Learning framework for architecture designs [50], and scikit-learn is an optimized and widely used framework for building Machine Learning models [51].

OSVMs use hyperplanes to maximize the separation of the anomalies from the normal instances [52], and they are also applicable for cases where the density of the data distribution is not well-defined [44]; IF constructs a profile of the instance space using the path length of different trees, where those

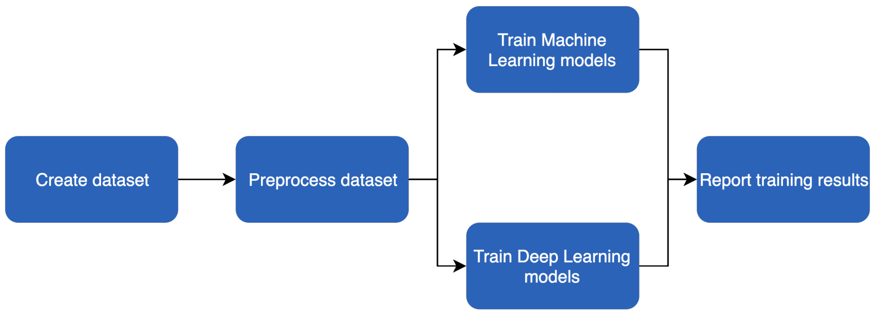


Fig. 1. Training workflow.

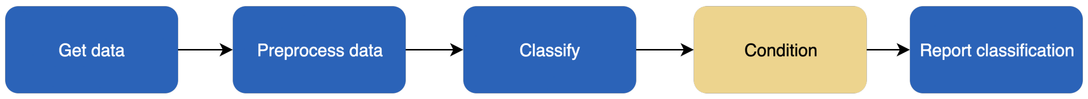


Fig. 2. Testing workflow.

TABLE I
ARTIFICIAL INTELLIGENCE ORCHESTRATORS BENCHMARK.

	Kubeflow	**Kedro**	**Airflow**	**MLflow**	**ZenML**	**Opni**
Native infrastructure served via Kubernetes	✔	✗	✔	✗	✗	✔
Open source	✔	✔	✔	✔	✔	✔
Free to use	✔	✔	✔	✔	✔	✔
Metric-level analysis support	✔	✔	✔	✔	✔	✔
E2E ML workflow orchestration	✔	✔	✔	✔	✔	✗

TABLE II
DATA COLLECTION TOOLS BENCHMARK.

	Prometheus	**Istio**	**InfluxDB**	**OpenTSDB**
Handles the complex Kubernetes monitoring nature	✔	✔	✔	✔
Open source	✔	✔	✔	✔
Free to use	✔	✔	✗	✔
Metrics support	✔	✔	✔	✔
Time-series format	✔	✔	✔	✔
GitHub stars (popularity)	43.2k [39]	30.7k [40]	23.7k [41]	4.7k [42]

instances closer to the root are considered anomalies [45]; AEs learn the distribution from the normal input instances and considering the distribution difference between normal and anomaly data, detect outliers based on the reconstruction error (since the reconstruction error of anomaly data could be higher than that of normal data) [46].

With these three models, two experiments were conducted to evaluate the performance of the algorithms in supervised and unsupervised settings. The supervised setting applied offline learning and used the LATAM-DDoS-IoT dataset; the unsupervised setting implemented online learning to query real-time data from Aligo's K8s infrastructure. These experiments helped to better approximate how the final voting system would behave in production. This final and deployed voting system follows Eq. 1, where the variables x, y, and z indicate the Autoencoder, Isolation Forest, and One-class Support Vector Machine predictions. The values 0.6, 0.3, and 0.1 are the arbitrarily assigned weights to each prediction after observing the models' performance (i.e., a higher detection rate meant a higher weight, therefore giving more importance to that prediction). The metrics used to measure these detection rates are specified in the paragraph below.

$$outlier\_confidence\_score = 0.6x + 0.3y + 0.1z \tag{1}$$

$$flow = \begin{cases} anomaly & \text{if } outlier\_confidence\_score \geq 0.9 \\ inlier & \text{if } outlier\_confidence\_score < 0.9 \end{cases}$$

TABLE III
SELECTED KUBERNETES METRICS GROUPED BY THE FOUR GOLDEN SIGNALS OF MONITORING.

Category	Metrics	Question to answer
Latency	• API Server Request Latency • Scheduler E2E Scheduling Latency	How long does it take to service a task?
Traffic	• API Server Request Rate • Ingress Controller Connections • Scheduler Preemption Attempts • Pod Network Packets	How much demand is placed on the environment?
Errors	• Scheduling Failed Pods	How many fails exist?
Saturation	• Workqueue Depth • Pod CPU Utilization • Pod Memory Utilization • Pod Network I/O • Pod Disk I/O	How much is the environment being utilized?

For the supervised setting, the classification performance was measured by computing the area under the Receiver Operating Characteristic (ROC) curve (AUC) [53]. Unlike other metrics, such as the F1 score, AUC is insensitive to changes in the distribution of the training dataset [54]. AUC evaluates the true positive detection rate (plotted on the y-axis) versus the false positive detection rate (plotted on the x-axis). Since many real-world problems have large numbers of negative instances, performance in the upper left triangle of the ROC graph is ideal for achieving [53]; for the unsupervised setting, since there were not labeled records as ground truth, the performance was estimated computing the accuracy assuming all the testing instances were inliers, to indicate if the models were able to extract patterns from the input data.

C. Feature Engineering

The defined metrics for monitoring the K8s infrastructure health are in Table III. These metrics are grouped into the four golden signals of monitoring: latency, traffic, errors, and saturation [55]. See Table IV for the corresponding calculated features (a result of monitoring data sources such as NeuVector [56]). The features include the cumulative CPU time consumed by the container, the current working set of the container in bytes, the current depth of workqueue, and the total preemption attempts in the cluster [57].

Additionally, to help convergence and avoid data bias, there were performed experiments applying min-max normalization [58] and feature standardization [59] using scikit-learn. See Eq. 2–3, respectively.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{2}$$

$$x' = \frac{x - mean}{stddev} \tag{3}$$

In Eq. 2, each of the input features in x is scaled individually to a given range, by subtracting the smallest value in the numerator, and setting the difference between the largest and smallest values in the denominator; in Eq. 3, the input features are standardized by subtracting the mean and dividing by the standard deviation, producing data with zero mean and unit variance.

The results of the different experiments can be read in the next section.

IV. RESULTS AND DISCUSSION

This section shows the results of the offline and online learning experiments. For offline learning, a models benchmark by AUC value is presented; for online learning, the developed voting system (explained in the previous section) is deployed and tested to observe the real-time notifications of the anomalies detected.

A. Offline Learning

Before implementing Kubeflow pipelines for online learning, the anomaly detection models selection was tested using the LATAM-DDoS-IoT. From it, 5% of the total normal flows and 1,998 DDoS attack flows were taken. Overall, the resulting dataset size was 41,958 flows, with 95.24% normal traffic and 4.76% outliers, emulating a real-world scenario where negative instances are way more than positive ones [7].

Due to the high detection rates reported in [1], the same data split and hyperparameter tuning processes were performed: the dataset was separated into 80% for training, 10% for validation, and 10% for testing, additionally the second feature set proposed in the same work (i.e., the 15 best features) was used, and an iterative hyperparameter tuning was conducted, trying different configuration settings with the validation set. This iterative tuning process consisted in, for instance, measuring the classification performance of the OSVM using different kernels, where the Radial Basis Function ended up being the best one; for IF, different numbers of trees were tested, where 50 trees led to the best AUC value; and for AE, the number of nodes in the fully connected layers was changed, where the best architecture can be seen in Fig. 3 with 15 nodes in the input layer and six nodes for the latent space. See Table V for these best hyperparameters values for the Machine Learning and Deep Learning models. The contamination hyperparameter for the One-class Support Vector Machine and the Isolation Forest, refers to the proportion of outliers used.

As commented in [23], the creation of a state-of-the-art AI-based intrusion detection system needs adequate hyperparameter tuning to improve the efficiency of the predictions. Although this optimization process may lead to overfitting, conducting this tuning in a separate validation set is relevant

TABLE IV
FEATURES DESCRIPTION FOR THE CREATED KUBERNETES DATASET.

Feature	**Description**
timestamp	Record measurement time.
api_server_request_latency	Response latency.
api_server_request_rate	Summation of the per-second rate of apiserver requests over the last 5 minutes.
scheduling_failed_pods	Total number of failed scheduled pods.
workqueue_depth	Current depth of workqueue.
scheduler_e2e_scheduling_latency	99th percentile of the E2E scheduling latency.
scheduler_preemption_attempts_total	Summation of the per-second rate of preemption attempts until now over the last 5 minutes.
active_client_connections	Current number of client connections with active state.
reading_client_connections	Current number of client connections with reading state.
waiting_client_connections	Current number of client connections with waiting state.
writing_client_connections	Current number of client connections with writing state.
container_cpu_cfs_throttled_seconds_total	Summation of the per-second rate duration the container has been throttled over the last 5 minutes.
container_cpu_usage_seconds_total	Summation of the per-second rate of the CPU time consumption over the last 5 minutes.
container_cpu_system_seconds_total	Summation of the per-second rate of the system CPU time consumption over the last 5 minutes.
container_cpu_user_seconds_total	Summation of the per-second rate of the user CPU time consumption over the last 5 minutes.
container_memory_working_set_bytes	Current working set in bytes.
container_network_receive_packets_total	Summation of the per-second rate of the number of packets received over the last 5 minutes.
container_network_receive_packets_dropped_total	Summation of the per-second rate of the number of packets dropped while receiving over the last 5 minutes.
container_network_receive_errors_total	Summation of the per-second rate of the number of errors encountered while receiving over the last 5 minutes.
container_network_transmit_packets_total	Summation of the per-second rate of the number of packets transmitted over the last 5 minutes.
container_network_transmit_packets_dropped_total	Summation of the per-second rate of the number of packets dropped while transmitting over the last 5 minutes.
container_network_transmit_errors_total	Summation of the per-second rate of the number of errors encountered while transmitting over the last 5 minutes.
container_network_receive_bytes_total	Summation of the per-second rate of the number of bytes received over the last 5 minutes.
container_network_transmit_bytes_total	Summation of the per-second rate of the number of bytes transmitted over the last 5 minutes.
container_fs_reads_bytes_total	Summation of the per-second rate of the number of bytes read over the last 5 minutes.
container_fs_writes_bytes_total	Summation of the per-second rate of the number of bytes written over the last 5 minutes.

to leave the testing set only for reporting the final classification results with the models finished.

The One-class Support Vector Machine and the Autoencoder present their best result after min-max normalization, while Isolation Forest presents it after data standardization. See Fig. 4 for the AUC of all these three models. Autoencoder is the best-evaluated algorithm and detects 88% of the outliers. Furthermore, Isolation Forest is the best ML model, and One-class Support Vector Machine is still better than random guessing (i.e., the blue dashed line). Nevertheless, higher true positive rates may be achieved with a more robust hyperparameter tuning.

B. *Online Learning*

Two Kubeflow pipelines were coded and deployed: one for training and the other for testing. The training pipeline collected data from Aligo's on-premise K8s infrastructure for ten days using the Prometheus HTTP API. This data is per minute and adds up to 11,846 records, divided into 90% to build the AI models and 10% to estimate their classification

TABLE V
SUMMARY OF THE BEST HYPERPARAMETERS VALUES FOR THE DIFFERENT MACHINE LEARNING AND DEEP LEARNING MODELS.

Model	Hyperparameters
One-class Support Vector Machine	• Kernel: Radial Basis Function • Kernel coefficient: 1 / number of features • Contamination: 5%
Isolation Forest	• Number of trees: 50 • Samples to train each tree: 256 • Contamination: 1%
Autoencoder	• Epochs: 10 • Batch size: 256 • Activation function: ReLU • Loss function: Mean Squared Error • Optimizer: Adam • Learning rate: 0.001 • See Fig. 3 for the architecture.

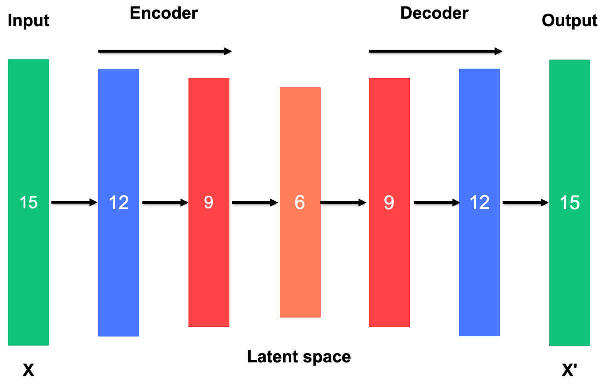


Fig. 3. Best Autoencoder architecture.

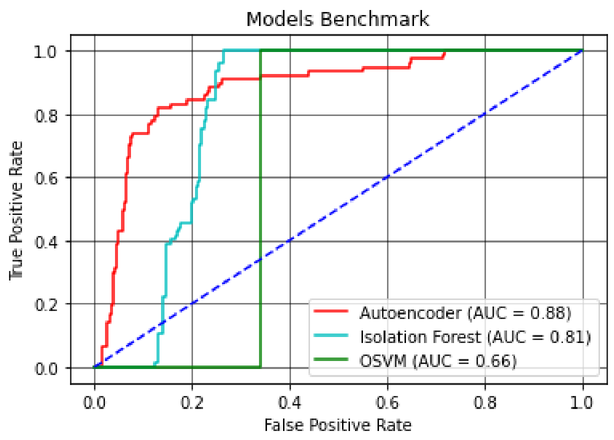
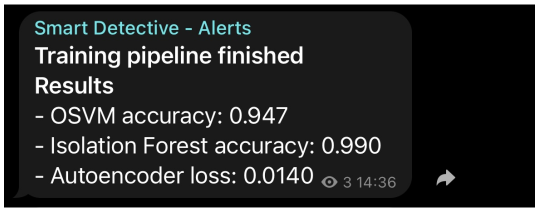


Fig. 4. LATAM-DDoS-IoT dataset results.



Smart Detective - Alerts
Training pipeline finished
Results
- OSVM accuracy: 0.947
- Isolation Forest accuracy: 0.990
- Autoencoder loss: 0.0140 3 14:36

Fig. 5. Telegram message with the training pipeline results.

performance. The duration of one minute was decided after considering it a reasonable time to see variations of the features presented in Table IV. Regarding the data split, since the same best hyperparameters values obtained from the offline learning experiment were used, no extra set for validation was maintained (i.e., for further hyperparameter tuning), adding up that potential 10% to the 80% of training data.

The training results are received via chat through a Telegram bot. See Fig. 5 for these estimated results, which indicate a substantial pattern extraction from the input data (i.e., a high accuracy and a low training loss).

For passing data through each step of the pipelines, MinIO [60] was used, and Amazon S3 [61] was used for passing data between pipelines. Amazon S3 allowed the storage and organization of the resulting models and datasets, retaining valuable information to maintain a timeline of different runs and enabling further detailed comparisons.

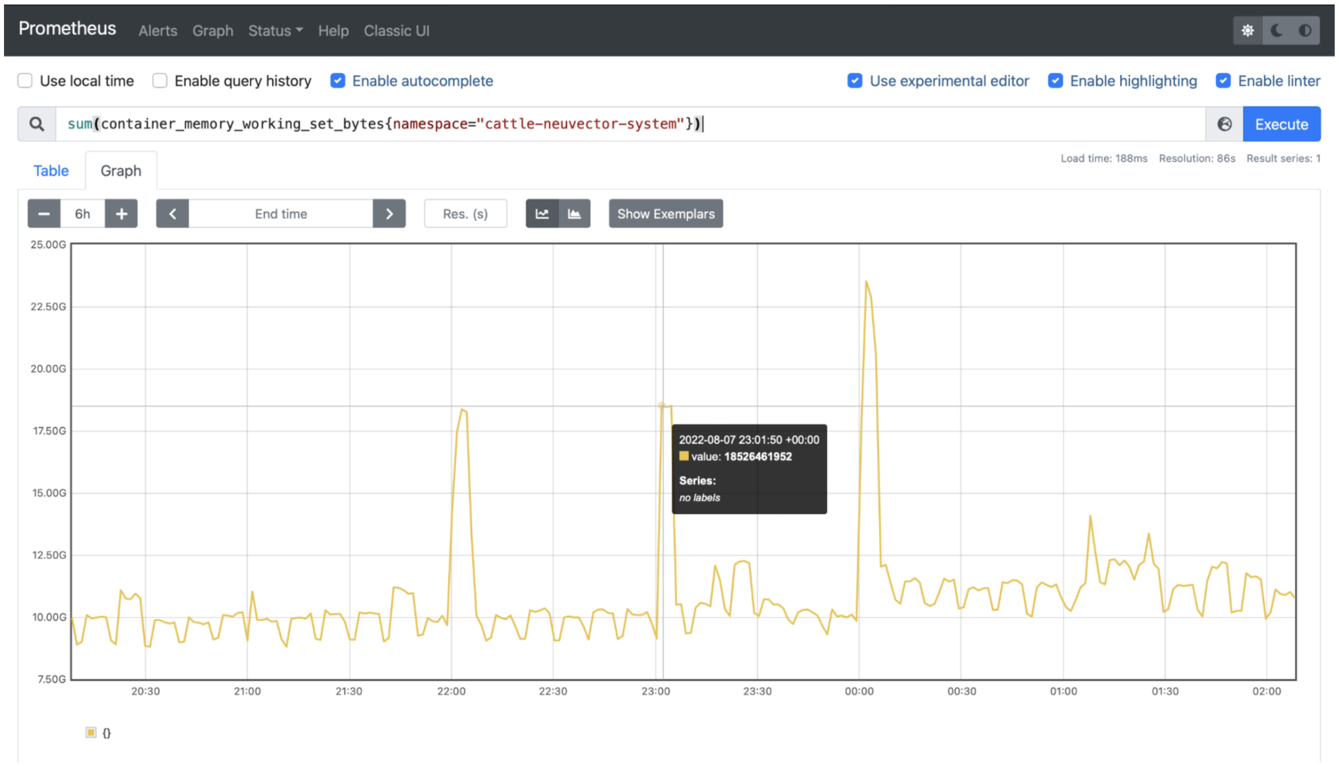
Chaos Engineering experiments [62] were run to test the trained models, similar to what Netflix has done in the past [63], [64]. Chaos Mesh [65] was used, a CNCF project personally considered more straightforward than Litmus [66]. A stress scenario was created where a memory saturation experiment was automated against a set of pods in a target namespace. These pods are randomly stressed for five minutes, at the start of every hour, by four threads that occupy 4,096 MB. See Fig. 6 for the results of running this saturation.

The Kubeflow testing pipeline was scheduled to run every minute. See Fig. 7 for the real-time alerts received on Telegram for the three peaks noticed from Fig. 6. Prometheus time is in UTC, while Telegram messages show the local time zone (i.e., UTC-5). The defined and deployed voting system is 100% confident when detecting these anomalies. This saturation experiment may be a good starting point to measure the AI system performance since memory issues like SYN scanning [67], NOP sled [68], and resource exhaustion [69] are crucial to be detected on time.

The next section presents the conclusions and future work that can be carried out.

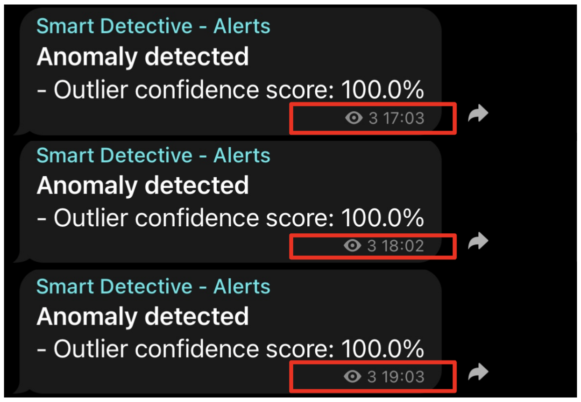
V. CONCLUSIONS AND FUTURE WORK

This paper shows the definition, implementation, and testing of a cloud-native infrastructure to develop AI projects at Aligo (see Fig. 8). It went from features definition and data collection to pipeline deployment and Chaos Engineering experiments. The created anomaly-based detection system presents strong performance and aims to reduce the time network engineers spend in front of a computer looking at any outliers.



```
sum(container_memory_working_set_bytes{namespace="cattle-neuvector-system"})
```

Fig. 6. Chaos Engineering experiment for memory saturation.



Smart Detective - Alerts
Anomaly detected
- Outlier confidence score: 100.0%

Smart Detective - Alerts
Anomaly detected
- Outlier confidence score: 100.0%

Smart Detective - Alerts
Anomaly detected
- Outlier confidence score: 100.0%

Fig. 7. Alerts received on Telegram.

However, more data may be needed to model a more robust baseline. In future work, the retention period of Prometheus (which by default is 15 days [70]) is planned to be extended to collect more records from the target instances. Furthermore, the One-Class Classification approach helps only in detecting outliers, so there is now the need to add at least another layer of classification, for instance, to indicate if the detected anomaly corresponds to a specific attack [71] and, with this, ease the troubleshooting of the problem.

Additionally, since the same tuned hyperparameters values from the LATAM-DDoS-IoT dataset were used for online learning, the integration of an automated hyperparameter tuning technique, such as Random Search [72], is now required. It could reduce the amount of human effort when training by automatically adjusting the values of the hyperparameters to ensure the same or similar classification performance results whenever the input training data changes.

Although the Kubeflow infrastructure was deployed on-premise, the solution proposal presented in this work combines cloud elements, namely the communication with Amazon S3 and the Telegram bot. These elements may fail, and network interruptions may exist. Therefore, as future work, alternative internal storage must be implemented for the resulting models and datasets from the online learning process, as well as look for another communication channel to alert about detected anomalies.

Acknowledgment

The author would like to thank Francisco Muñoz and Brian Martinez for their valuable support in the development of this project.

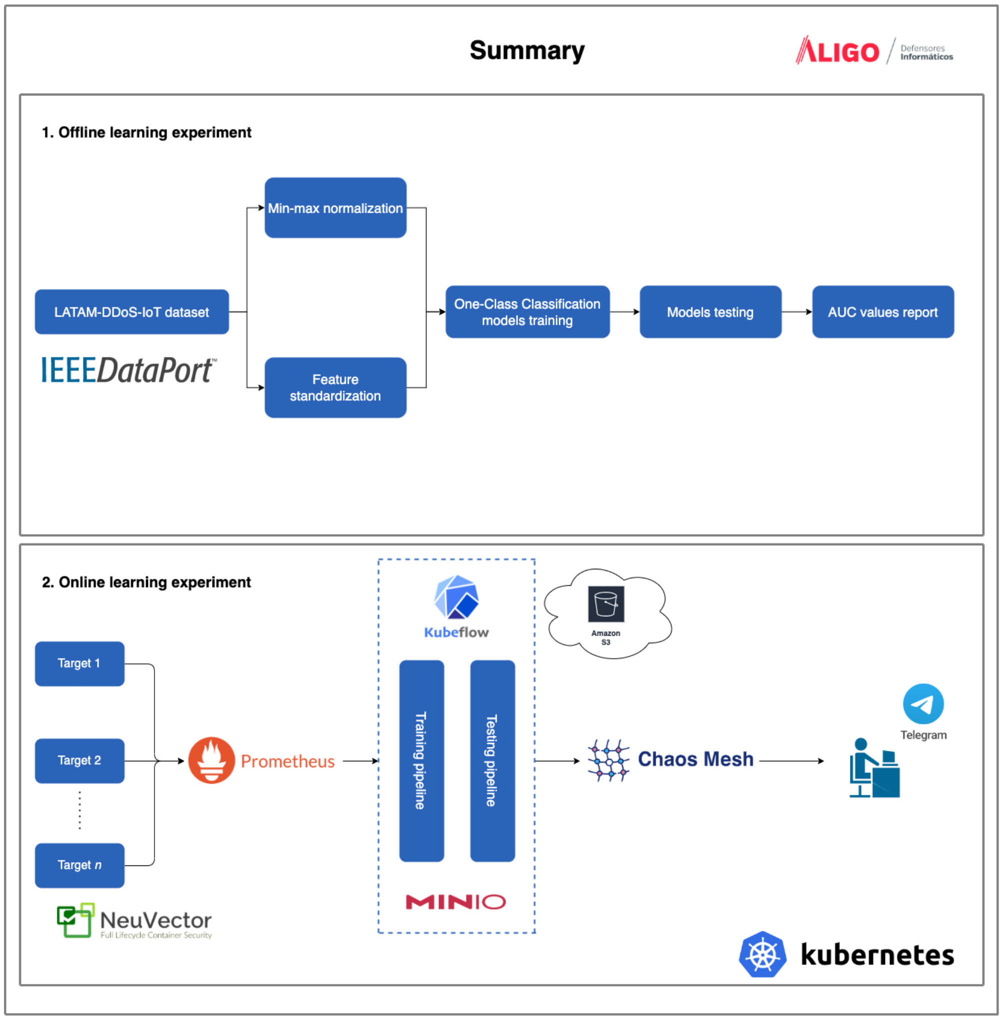


Fig. 8. Summary of the different work presented in this paper.

References

[1] J. G. Almaraz-Rivera, J. A. Perez-Diaz, and J. A. Cantoral-Ceballos, “Transport and application layer ddos attacks detection to iot devices by using machine learning and deep learning models,” *Sensors*, vol. 22, no. 9, 2022.

[2] “Kubernetes. https://kubernetes.io/.”

[3] “Aligo. https://aligo.com.co/.”

[4] “Apache airflow and kubernetes.” https://airflow.apache.org/docs/apache-airflow/stable/kubernetes.html. Accessed on 4 July 2022.

[5] “Cncf annual survey 2021.” https://www.cncf.io/reports/cncf-annual-survey-2021/. Accessed on 17 August 2022.

[6] S. S. Khan and M. G. Madden, “One-class classification: taxonomy of study and review of techniques,” *The Knowledge Engineering Review*, vol. 29, no. 3, p. 345–374, 2014.

[7] N. Seliya, A. Abdollah Zadeh, and T. M. Khoshgoftaar, “A literature review on one-class classification and its potential applications in big data,” *Journal of Big Data*, vol. 8, no. 1, p. 122, 2021.

[8] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, “Hidden technical debt in machine learning systems,” in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[9] D. Sahoo, Q. Pham, J. Lu, and S. C. H. Hoi, “Online deep learning: Learning deep neural networks on the fly,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI’18, p. 2660–2666, AAAI Press, 2018.

[10] J. G. Almaraz-Rivera, J. A. Perez-Diaz, J. A. Cantoral-Ceballos, J. F. Botero, and L. A. Trejo, “Toward the protection of iot networks: Introducing the latam-ddos-iot dataset,” *IEEE Access*, vol. 10, pp. 106909–106920, 2022.

[11] J. G. Almaraz-Rivera, J. A. Perez-Diaz, J. A. Cantoral-Ceballos, J. F. Botero, and L. A. Trejo, “Latam-ddos-iot dataset.” https://dx.doi.org/10.21227/rwtj-dd43, 2022.

[12] S. T. Zargar, J. Joshi, and D. Tipper, “A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks,” *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[13] P. Perera, P. Oza, and V. M. Patel, “One-class classification: A survey,” *CoRR*, vol. abs/2101.03064, 2021.

[14] P. Arregoces, J. Vergara, S. A. Gutiérrez, and J. F. Botero, “Network-based intrusion detection: A one-class classification approach,” in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2022.

[15] J. B. Camiña, M. A. Medina-Pérez, R. Monroy, O. Loyola-González, L. A. P. Villanueva, and L. C. G. Gurrola, “Bagging-randomminer: a one-class classifier for file access-based masquerade detection,” *Machine Vision and Applications*, vol. 30, no. 5, pp. 959–974, 2019.

[16] G. Ahmadi-Assalemi, H. Al-Khateeb, G. Epiphaniou, and A. Aggoun, “Super learner ensemble for anomaly detection and cyber-risk quantification in industrial control systems,” *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13279–13297, 2022.

[17] L. Bilge and T. Dumitraş, “Before we knew it: An empirical study of zero-day attacks in the real world,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS ’12, (New York, NY, USA), p. 833–844, Association for Computing Machinery, 2012.

[18] “Red hat emerging technologies. prometheus anomaly detection.” https://next.redhat.com/2019/11/18/prometheus-anomaly-detection/. Accessed on 15 September 2022.

[19] L. P. Dewi, A. Noertjahyana, H. N. Palit, and K. Yedutun, "Server scalability using kubernetes," in *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, pp. 1–4, 2019.

[20] Y. Zhou, Y. Yu, and B. Ding, "Towards mlops: A case study of ml pipeline platform," in *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pp. 494–500, 2020.

[21] H. Cai, C. Wang, and X. Zhou, "Deployment and verification of machine learning tool-chain based on kubernetes distributed clusters," *CCF Transactions on High Performance Computing*, vol. 3, no. 2, pp. 157–170, 2021.

[22] J. George and A. Saha, "End-to-end machine learning using kubeflow," in *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*, CODS-COMAD 2022, (New York, NY, USA), p. 336–338, Association for Computing Machinery, 2022.

[23] M. Pawlicki, R. Kozik, and M. Choraś, "A survey on neural networks for (cyber-) security and (cyber-) security of neural networks," *Neurocomputing*, vol. 500, pp. 1075–1087, 2022.

[24] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[25] R. Biswas and S. Roy, "Botnet traffic identification using neural networks," *Multimedia Tools and Applications*, vol. 80, no. 16, pp. 24147–24171, 2021.

[26] D. A. Tamburri, "Sustainable mlops: Trends and challenges," in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 17–23, 2020.

[27] "Kubeflow. https://www.kubeflow.org/."

[28] "Apache airflow. https://airflow.apache.org/."

[29] "Mlflow. https://mlflow.org/."

[30] "Machine learning operations." https://www.run.ai/guides/machine-learning-operations. Accessed on 5 December 2022.

[31] "Kubeflow vs airflow." https://hevodata.com/learn/kubeflow-vs-airflow/. Accessed on 5 July 2022.

[32] "Kubeflow architecture." https://www.kubeflow.org/docs/started/architecture/. Accessed on 5 July 2022.

[33] "Jupyter. https://jupyter.org/."

[34] "Istio. https://istio.io/."

[35] "Prometheus. https://prometheus.io/."

[36] "Influxdb. https://www.influxdata.com/."

[37] "Opentsdb. http://opentsdb.net/."

[38] "Kubernetes architecture for ai workloads." https://www.run.ai/guides/kubernetes-architecture. Accessed on 5 July 2022.

[39] "Prometheus repository: https://github.com/prometheus/prometheus/."

[40] "Istio repository: https://github.com/istio/istio/."

[41] "Influxdb repository: https://github.com/influxdata/influxdb/."

[42] "Opentsdb repository: https://github.com/opentsdb/opentsdb/."

[43] "Cncf cloud native interactive landscape." https://landscape.cncf.io/. Accessed on 6 July 2022.

[44] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Computation*, vol. 13, pp. 1443–1471, 07 2001.

[45] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, mar 2012.

[46] C. Yin, S. Zhang, J. Wang, and N. N. Xiong, "Anomaly detection based on convolutional recurrent autoencoder for iot time series," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 1, pp. 112–122, 2022.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[49] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[50] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, sep 2018.

[51] M.-A. Zöller and M. F. Huber, "Benchmark and survey of automated machine learning frameworks," *J. Artif. Int. Res.*, vol. 70, p. 409–472, may 2021.

[52] D. M. J. Tax and R. P. W. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, 2004.

[53] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. ROC Analysis in Pattern Recognition.

[54] N. Japkowicz, *Assessment Metrics for Imbalanced Learning*, ch. 8, pp. 187–206. John Wiley & Sons, Ltd, 2013.

[55] N. Murphy, B. Beyer, C. Jones, and J. Petoff, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

[56] "Neuvector. https://neuvector.com/."

[57] "Kubernetes metrics reference." https://kubernetes.io/docs/reference/instrumentation/metrics/. Accessed on 2 December 2022.

[58] Y. Kayode Saheed, A. Idris Abiodun, S. Misra, M. Kristiansen Holone, and R. Colomo-Palacios, "A machine learning-based intrusion detection for detecting internet of things network attacks," *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 9395–9409, 2022.

[59] M. A. Pirbonyeh, M. A. Shayegan, G. Sotudeh, and S. Shamshirband, "Heterogeneous domain adaptation by features normalization and data topology preserving," *Knowledge-Based Systems*, vol. 257, p. 109536, 2022.

[60] "Minio. https://min.io/."

[61] "Amazon s3. https://aws.amazon.com/es/s3/."

[62] "Principles of chaos engineering." https://principlesofchaos.org/. Accessed on 8 August 2022.

[63] "The netflix simian army." https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116. Accessed on 8 August 2022.

[64] "Netflix chaos monkey upgraded." https://netflixtechblog.com/netflix-chaos-monkey-upgraded-1d679429be5d. Accessed on 8 August 2022.

[65] "Chaos mesh. https://chaos-mesh.org/."

[66] "Litmus. https://litmuschaos.io/."

[67] A. Rodríguez and L. Castillo, "A first step towards a general-purpose distributed cyberdefense system," in *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection* (Y. Demazeau, B. An, J. Bajo, and A. Fernández-Caballero, eds.), (Cham), pp. 237–247, Springer International Publishing, 2018.

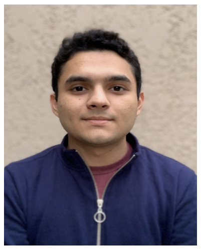
[68] X. Huang, F. Yan, L. Zhang, and K. Wang, "Honeygadget: A deception based approach for detecting code reuse attacks," *Information Systems Frontiers*, vol. 23, no. 2, pp. 269–283, 2021.

[69] J. Antunes, N. F. Neves, and P. Veríssimo, "Detection and prediction of resource-exhaustion vulnerabilities," in *19th International Symposium on Software Reliability Engineering (ISSRE 2008), 11-14 November 2008, Seattle/Redmond, WA, USA*, pp. 87–96, IEEE Computer Society, 2008.

[70] "Prometheus storage." https://prometheus.io/docs/prometheus/latest/storage/. Accessed on 8 August 2022.

[71] "6 common kubernetes and container attack techniques and how to prevent them." https://www.paloaltonetworks.com/blog/prisma-cloud/6-common-kubernetes-attacks/. Accessed on 8 August 2022.

[72] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, p. 281–305, feb 2012.



Josue Genaro Almaraz-Rivera received the B.S. degree in computer science from Universidad Autonoma de Nuevo Leon, Mexico, in 2020, and the M.S. degree in computer science from Tecnologico de Monterrey, Mexico, in 2022, where he is currently pursuing the Ph.D. degree in computer science.

He has worked at companies such as Apple and Meta. Also, he was a TEDx speaker at two events at Universidad Autonoma de Nuevo Leon, in 2017 and 2018, with the Social Hacking and Artificial Intelligence topics.

His current research interests include Self-Supervised Learning and its potential application to cybersecurity.