# Synthetic Time Series for Anomaly Detection in Cloud Microservices

Mohamed Allam, Noureddine Boujnah, Noel E. O'Connor, and Mingming Liu

Insight SFI Research Centre for Data Analytics, Dublin City University
mohamed.allam@insight-centre.org
{noureddine.boujnah,noel.oconnor,mingming.liu}@dcu.ie

**Abstract.** This paper proposes a framework for time series generation built to investigate anomaly detection in cloud microservices. In the field of cloud computing, ensuring the reliability of microservices is of paramount concern and yet a remarkably challenging task. Despite the large amount of research in this area, validation of anomaly detection algorithms in realistic environments is difficult to achieve. To address this challenge, we propose a framework to mimic the complex time series patterns representative of both normal and anomalous cloud microservices behaviors. We detail the pipeline implementation that allows deployment and management of microservices as well as the theoretical approach required to generate anomalies. Two datasets generated using the proposed framework have been made publicly available through GitHub.

**Keywords:** Anomaly Detection · Cloud Monitoring · Distributed Systems · Microservice Applications · Time Series Analysis

## 1 Introduction

The microservice architecture has emerged as a dominant paradigm for building scalable and flexible software systems. In contrast to monolithic architectures, microservices decompose applications into small, independently deployable services, each responsible for a specific business function. This architectural style offers numerous benefits, including enhanced agility, scalability, and fault isolation [11]. Consequently, the adoption of microservices has witnessed a significant surge in recent years, driven by the growing demand for cloud-native and distributed systems.

Despite the advantages of microservice architectures, effectively monitoring and ensuring the reliability of microservice-based applications pose substantial challenges [12]. The dynamic nature of microservices, characterized by their distributed nature and high degree of interdependence, makes traditional monitoring approaches impractical. Manual inspection becomes infeasible due to the sheer volume and complexity of services. Moreover, relying solely on anomaly detection methods focusing on single components often proves inadequate, as they fail to capture the contextual dependencies and correlations inherent in microservice environments and might lead to a very high number of false alarms [13].

To address these challenges, recent research efforts have focused on developing anomaly detection techniques tailored specifically for microservice architectures. These approaches leverage multivariate analysis and incorporate diverse data modalities, combining telemetry with logs and traces, to enhance anomaly detection accuracy.

Studies focusing on anomaly detection in microservice applications or similar distributed systems environments often utilize existing datasets [6] [18] or create their own by injecting anomalies [13] into simulated environments and collecting observability data. However, the scarcity of suitable datasets, compounded by the diverse array of potential anomalies and deployment configurations, can render existing datasets inadequate. Crafting an original dataset through simulating a microservice environment and intentionally introducing anomalies presents a significant challenge. This complexity arises from the need to deploy the environment, simulate a realistic load, and inject anomalies effectively. Each step requires different frameworks to interact seamlessly, demanding a high degree of integration. Additionally, while it is possible to simulate various types of anomalies, these are primarily based on assumptions and may not accurately reflect real-world production environments. Obtaining production data from cloud providers is further complicated by GDPR and other constraints. To address these challenges, we have engaged with industry experts from a world-leading cloud provider, using their insights to design the generation of load patterns and anomalies.

In light of these challenges, we propose an anomaly generation platform tailored specifically for microservice applications leveraging Amazon Web Services (AWS) [1] in addition to some open-source tools. Our platform encompasses a comprehensive pipeline that includes detailed descriptions of microservice deployment, load simulation, observability instrumentation, and data collection mechanisms. By providing a holistic framework for generating realistic anomaly scenarios, our platform aims to address the limitations of existing methodologies and facilitate more robust assessments of anomaly detection techniques in microservice environments through the generation of labelled multivariate datasets. In this paper, we present the design and implementation of our open-source microservice platform for anomaly generation, monitoring, and data collection. We believe that our platform will serve as a valuable tool for researchers and practitioners in the field of microservice application monitoring and anomaly detection, enabling more accurate and reliable evaluations of anomaly detection techniques in real-world settings. Furthermore, we make available two open-source labelled multivariate datasets [1] containing some targeted anomaly scenarios. Finally, we note that the main focus of this paper is on the task of synthetic dataset generation. Due to space limitations, the application of various anomaly detection techniques on the generated datasets will be addressed in our future work.

The remainder of the paper is as follows: Section 2 provides an overview of the current solutions in this field and highlights the key areas where further improvements are needed. Section 3 introduces our approach to address the

[1] https://github.com/Mohamed164/AD-microservice-app

identified gaps. In Section 4, we offer an illustrative example demonstrating the generation of a multivariate labelled dataset using our proposed approach. Section 5 summarizes our findings, discusses the limitations of this work, and suggests directions for future research.

## 2 Related Work

In the domain of microservice anomaly detection, researchers rely on two main approaches to train data-driven machine learning algorithms and evaluate anomaly detection techniques: the creation of datasets via simulation frameworks and the utilization of pre-existing datasets. In this section, we describe these two approaches and highlight some of their advantages and shortcomings.

### 2.1 Simulation Frameworks

Microservice application simulation entails the creation of synthetic environments that closely replicate the operational characteristics of real-world microservice architectures. These simulations encompass the generation of artificial workload patterns, service interactions, and anomaly scenarios, in addition to data collection, to facilitate the evaluation of anomaly detection algorithms. A salient advantage of simulation-based methodologies lies in their ability to include controlled experimental settings, enabling researchers to systematically manipulate parameters, introduce specific anomalies, and assess algorithmic performance under diverse conditions. Notably, Nobre et al. [13] proposed a simulation framework for synthesizing time-series data representative of microservice system behavior. However, since the main focus is on the anomaly detection model the approach does not document the platform nor detail the load function used to simulate user traffic which is crucial for simulating scenarios similar to those found in the real world.

### 2.2 Existing Datasets

An alternative approach involves the utilization of publicly available datasets. These datasets typically comprise real-world operational data collected from production microservice architectures or simulated environments. Additionally, some datasets are collected from other types of distributed system architectures that are not necessarily microservices but bear significant similarities with them. These datasets are often utilized in this domain due to their relevance and applicability. By leveraging such datasets, researchers can assess algorithmic performance on authentic data and corroborate findings in real-world settings.

Jun Huang et al. [6] trained and evaluated the performance of their anomaly detection model using two publicly available datasets. The MSDS (Multi-modal Dataset for System Anomaly Detection) [2] provides a rich resource for evaluating anomaly detection algorithms in distributed systems. The MSDS consists of

[2] https://zenodo.org/records/226060

distributed traces, application logs, and metrics collected from a complex distributed system (Openstack) [15] used for AI-powered analytics. The dataset includes metrics data from 5 physical nodes, each containing 7 metrics such as RAM and CPU usage, as well as log files distributed across the infrastructure with a total of 23 features. Notably, MSDS also offers a JSON file containing ground-truth information for injected anomalies, including start and end times and corresponding anomaly types, facilitating accurate evaluation of anomaly detection techniques. The second dataset is the AIOps-Challenge 2 dataset [7]. This dataset is derived from a simulated e-commerce system operating on a microservice architecture, with 40 service instances deployed across 6 physical nodes. It encompasses metrics recorded by each service instance, including 56 metrics, with 25 utilized in this study, covering aspects such as RAM and CPU usage. Additionally, log files are recorded for each service instance, containing a collective set of 5 features, including timestamps and original logs. The dataset traces scheduling information among service instances, capturing timestamps, types, status codes, service instance names, span IDs, parent IDs, and trace IDs. The dataset includes intentionally injected anomalies at service, pod (service instance), and node levels, accompanied by start times, levels, service names, and types. The ratio of normal to abnormal data is 120:1. A significant drawback of this dataset is that the end time of injected anomalies is not provided.

Chenyu Zhao et al. [18] used the GAIA dataset [3]. It is a multimodal dataset collected from a system consisting of 10 instances. It consists of more than 0.7 million metrics, 87 million logs, and 28 million traces collected in a two-week period. It includes real-world injected failures alongside with their ground truth (timestamp of injection). This work also utilizes another larger dataset that is not open-source. The Server Machine Dataset (SMD) [4] proposed by [16] and utilized by [10], is not a microservice dataset but rather a distributed system dataset. Spanning five weeks, it comprises data collected from 28 online service systems, each distributed across various servers. These systems offer a range of services including searching, ranking, and data processing.

While publicly available datasets confer the advantage of real-world relevance, they also pose certain challenges. A prevalent issue pertains to the restricted diversity and coverage of anomaly types within the datasets, which may not adequately encapsulate the spectrum of anomalies encountered in practice. Additionally, challenges such as data labeling inaccuracies or absent ground truth annotations can pose impediments to algorithmic evaluation and comparison.

## 3 Proposed Approach

In addressing the gaps highlighted in Section 2, our framework offers a practical approach. Firstly, it involves simulating a realistic load function that accurately mirrors the complex seasonal and trend patterns observed in real-world loads.

[3] `https://github.com/CloudWise-OpenSource/GAIA-DataSet`
[4] `https://github.com/NetManAIOps/OmniAnomaly`

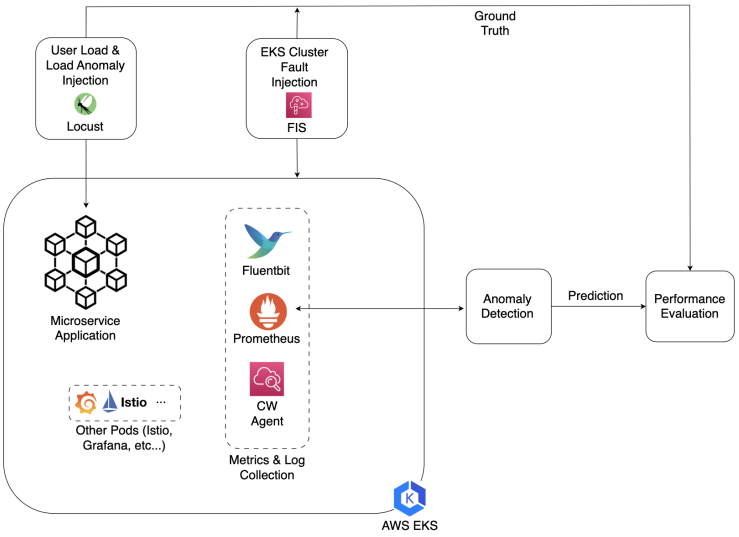



Fig. 1: Experimental Setup Architecture

Secondly, it enables the injection of various anomalies, each with different parameters related to user traffic load, the microservice cluster, and the underlying infrastructure. Thirdly, we propose a method to automate the labelling of these injected anomalies. Finally, we describe how metrics are collected to create a multivariate dataset, and how additional observability data, such as logs, can enhance this dataset to support multimodal models. This section details the architecture and setup of our proposed framework.

### 3.1 Architecture and Main Components

Our setup, depicted in Figure 1, involves deploying a microservice application on Amazon Elastic Kubernetes Service (EKS) [3], a managed Kubernetes service provided by Amazon Web Services (AWS) [1] that simplifies deployment, management, and scaling of containerized applications. The EKS cluster is deployed on several AWS EC2 instances (Elastic Compute Cloud), which are virtual servers within the AWS cloud serving as cluster nodes. We employ two open-source applications, widely used in literature: Sock-Shop [17] and Online Boutique [5], both of which mimic e-commerce platforms with multiple microservices written in different languages and include databases.

To inject anomalies simulating cluster malfunctions, we use AWS Fault Injection Service (FIS) [4], for ease of integration with other AWS services deployed. For observability, we use Istio [8], a service mesh that manages traffic flows and collects metrics within the EKS cluster. AWS CloudWatch, a monitoring and observability service within the AWS ecosystem, is used to gather metrics from cluster nodes and pods. Additionally, the CloudWatch Fluent Bit agent is used

to collect microservice logs. Prometheus [14], an open-source monitoring and alerting tool, is integrated to scrape Istio metrics, providing additional insights.

To simulate load, we utilize Locust [9], a flexible load testing tool, deployed on a test server due to its ability to handle custom load functions. This setup allows us to introduce anomalies related to the application traffic.

### 3.2 Load Simulation and Load Anomaly Injection

To simulate realistic user traffic patterns, we utilize the Locust framework, employing a custom function that dynamically generates users over time based on the desired number of users and their creation or spawn rate. We then explore how manipulating the user load and spawn rate functions enables the introduction of load anomalies, facilitating comprehensive testing of the system under varying conditions. Finally, within this framework, users engage in predefined scenarios that encapsulate diverse behaviors and interactions with the application, adding further realism to our simulation.

**Normal User Load** To simulate the number of users, we propose an additive model incorporating multiple seasonal components, each with its own noise term. Arbitrary seasonal functions can be employed, but we suggest utilizing a sine square function. As per our conversations with experts of a leading cloud provider, this function closely mimics access patterns to services observed in some real-world datasets. The model is represented as Equation (1).

$$\mathrm{N(t)} = (1 + trend(t)) \times \mathrm{base\_load} + \sum_{i=1}^{n} A_i \times ((\sin(\frac{2\pi t}{T_i}))^2 + wn_i(t)) \tag{1}$$

where:

- $A_i$ is the amplitude of the $i^{th}$ sine squared function,
- $T_i$ is the periodicity of the $i^{th}$ sine squared function, and
- $wn_i(t)$ is the noise component sampled from white noise with variance $\sigma_i$.

and $trend(t)$ is defined by (2).

$$trend(t) = \sum_{i=1}^{m} f_i(t) \tag{2}$$

where $m$ is the total number of intervals, and $f_i(t)$ is the piecewise linear function for the $i^{th}$ interval, as defined by (3).

$$f_i(t) = \begin{cases} \mathrm{slope}_i, & \text{if sudden shift occurs} \\ \mathrm{slope}_{i-1} \cdot (1 - \frac{t - \mathrm{start}_i}{\mathrm{end}_i - \mathrm{start}_i}) + \mathrm{slope}_i \cdot \frac{t - \mathrm{start}_i}{\mathrm{end}_i - \mathrm{start}_i}, & \text{otherwise} \end{cases} \tag{3}$$

where:

- $\text{slope}_i$ is the slope of the trend within the $i^{th}$ interval,
- $\text{start}_i$ is the start of the $i^{th}$ interval, and
- $\text{end}_i$ is the end of the $i^{th}$ interval.

The occurrence of a sudden shift within each interval $i$ is modeled by a Bernoulli distribution with parameter $p = 0.01$ (probability of a sudden shift in each interval), i.e., $Y \sim \text{Bernoulli}(n, 0.01)$.

The spawn rate, $R(t)$, is defined as the gradient of $N(t)$ with respect to time, indicating how quickly new users are introduced into or subtracted from the system, as shown by Equation (5). When the spawn rate is negative, it indicates that the number of users is decreasing as virtual users are being removed. As we are dealing with continuous and non-differentiable discrete functions, the gradient with respect to $t$ is the mean average of the left and right sides of the derivative, and provided by Equation (4).

$$R(t) = \frac{N'_r(t) + N'_l(t)}{2} \tag{4}$$

Where, $N'_r(t)$ is the right hand derivative of $N(t)$ and $N'_l(t)$ is the left hand derivative of $N(t)$, for the sake of computation the spawn rate can be written as Equation (5).

$$R(t) = \lim_{h \to 0} \frac{N(t+h) - N(t-h)}{2h} \tag{5}$$

**Load Anomaly Generation** Load anomalies are introduced into the load function by augmenting its values probabilistically, as depicted by Algorithm 1. With a given probability $p$, the algorithm generates a load anomaly during an interval characterized by a duration sampled from a uniform distribution $U(a, b)$, where $a$ and $b$ represent the lower and upper bounds of the duration range, respectively. Additionally, the magnitude of the anomaly is determined by a multiplier sampled from a uniform distribution $U(c, d)$, where $c$ and $d$ denote the lower and upper bounds of the multiplier range, respectively. This process ensures that anomalies are introduced into the load function in a controlled manner, enabling comprehensive analysis and adaptive responses to fluctuations in user activity.

Figure 2 illustrates an example of a load function generated using the proposed framework. The trend and periodic components are decomposed. It includes increasing and decreasing trend intervals in addition to a sudden shift. The spawn rate is also depicted in the figure.

**User Scenarios** To effectively simulate user load on both applications, which are e-commerce websites with similar functionalities, we define three distinct user scenarios:

- Visitor User: a user that visits the homepage and catalogue
- New Shopper User: user that registers, visits catalogue, adds a random item to the cart, and makes the order.

**Algorithm 1** Load Anomaly Generation

```
anomaly_end_time ← None
anomaly_multiplier ← None
for (timestamp, value) in user_load do
    if anomaly_end_time and timestamp ≤ anomaly_end_time then
        Mark timestamp as anomaly
        Adjust value using anomaly_multiplier
    else if random number < p then
        anomaly_duration ← U(a, b)                    ▷ Sample anomaly duration
        anomaly_end_time ← timestamp + anomaly_duration
        Mark timestamp as anomaly
        anomaly_multiplier ← U(c, d)                  ▷ Sample anomaly multiplier
        Adjust value using anomaly_multiplier
    end if
end for
```

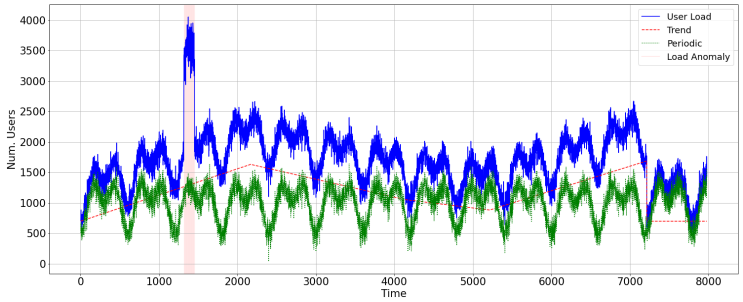
– Returning Shopper User: a similar flow to the new shopper users but the user is already registered so performs only a login.

Figure 3 illustrates a scenario that combines the actions of New Shopper User and Returning Shopper User. The probability of a New Shopper User workflow is denoted by $x$, while the probability of a Returning Shopper User workflow is 1-$x$.

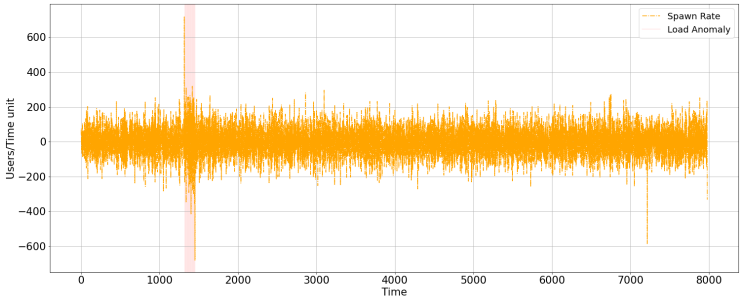
### 3.3 EKS Cluster Anomaly Injection

FIS is employed to systematically introduce anomalies within our EKS cluster. These scenarios are defined as templates and encompass various anomaly types at both the node and pod levels. At the pod level, anomalies include pod deletion, CPU stress, memory stress, increased network latency, packet dropping, and I/O stress. Similarly, at the node level, anomalies involve CPU stress, memory stress, increased network latency, packet dropping, I/O stress, and instance rebooting. Table 1 summarizes these anomalies and their parameters.

Anomalies are deployed by defining several anomaly templates on AWS FIS. Each anomaly template requires the definition of an action, for example, CPU stress, with its parameters, and a target, for example, an EC2 instance or a pod. We define a total of 12 templates for every anomaly and target type in Table 1. The targets are defined as "ec2-target" or "pod-target" depending on the target type and are left as placeholders. In the case of EC2 anomalies, we use the Amazon Resource Name (ARN) of the target instance, and in the case of pods, we use a label selector to identify the pods to be targeted. We use an IAM role with policies allowing access to the EKS cluster, EC2 instances and CW for logging. We also setup log forwarding to CW as these logs will serve as ground truth for anomaly labelling in a later stage. We then use a Python script, using the Boto3 library [2], to deploy these experiments according to a probability using a uniform distribution. Deployment consists of two steps: updating the




(a) Load Function




(b) Spawn Rate

Fig. 2: Load Function and Spawn Rate

FIS experiment template by substituting the placeholder with a random target or multiple targets, and then deploying the experiment. Algorithms 2 summarizes these steps.

### 3.4 Metrics Collection

To collect metrics we utilize the Boto3 library and the Prometheus API, in the highlighted step in Figure 4. Node and pod-level metrics are obtained by querying CloudWatch, while service-level metrics (coming from Istio) are retrieved from Prometheus. Load anomaly ground truth is logged by the Locust custom script, while FIS experiments' information and ground truth are retrieved from its CloudWatch logs. By combining these datasets, we obtain a comprehensive, multivariate dataset describing the state of the cluster, its nodes, pods, and services. This dataset includes anomaly injection ground truths, making it a valuable resource for training and evaluating anomaly detection algorithms.

The final dataset comprises node-level metrics, pod-level metrics and service-level metrics. In addition to the anomaly ground truth composed of FIS experiment information (experiment deployed, start and end time), and load anomaly injection information (start time, end time).

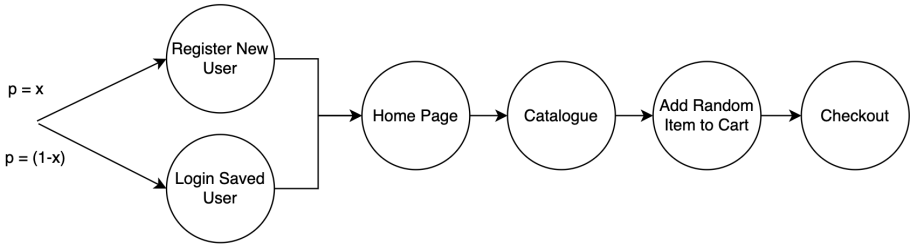



Fig. 3: User Scenarios: New Shopper User and Returning Shopper User

Table 1: Parameters of FIS experiments

| **Anomaly** | **Target** | **Parameters** |
|---|---|---|
| Pod Deletion | pod | target |
| CPU Stress | pod/node | target, duration, load % |
| Memory Stress | pod/node | target, duration, load % |
| Network Latency | pod/node | target, duration, latency magnitude |
| Packet Dropping | pod/node | target, duration, packet drop rate |
| I/O Stress | pod/node | target, duration, I/O space % |
| Node Reboot | node | target |

### 3.5 Logs Collection

Logs are collected by the CloudWatch agent[5] and forwarded to CloudWatch where they are saved. These logs can be queried by utilizing the Boto3 library. There are four types of log groups generated by the CloudWatch agent:

- /**aws**/**containerinsights**/<**cluster_name**>/**dataplane**: This log group contains logs related to the data plane of your Amazon EKS cluster. It includes information about network traffic, load balancer activity, and communication between nodes.
- /**aws**/**containerinsights**/<**cluster_name**>/**application**: Logs relevant to the applications running on the cluster are stored in this log group. It includes application-level logs, such as logs generated by services or applications deployed within the cluster.
- /**aws**/**containerinsights**/<**cluster_name**>/**performance**: This log group captures performance-related logs, providing insights into the resource utilization, latency, and other performance aspects of the cluster.
- /**aws**/**containerinsights**/<**cluster_name**>/**host**: Logs related to the underlying hosts or nodes of the cluster are stored in this log group. It

[5] https://docs.aws.amazon.com/eks/latest/userguide/eks-observe.html

**Algorithm 2** Deploying Anomaly Experiments

**Require:** Probability of anomaly $p_{\text{anomaly}}$, List of anomaly templates

```
while True do
    if random() < p_anomaly then
        Pick a random anomaly template
        if Target type is EC2 then
            Use ARN of random target instance(s)
        else
            if Target type is pod then
                Use label selector to target random pod(s)
            end if
        end if
        Update experiment template
        Deploy experiment
    end if
    Wait for next minute
end while
```

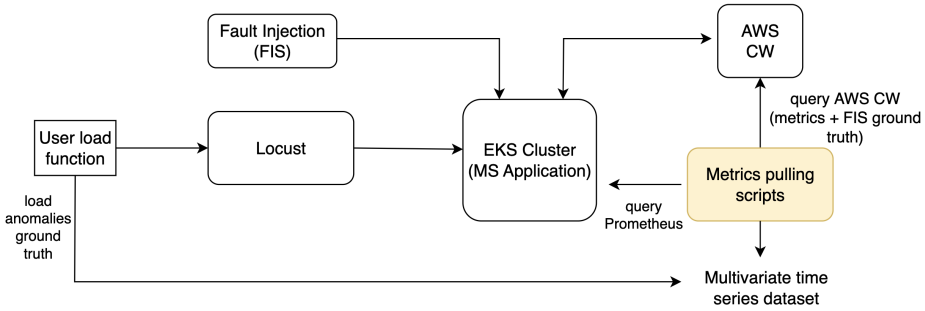



Fig. 4: Metrics Collection

includes system-level logs, such as kernel messages, hardware events, and other host-specific information.

## 4 Example Dataset

This section presents a dataset created using our proposed framework. The parameters below were used as inputs in our dataset generation pipeline for both published datasets.

The following parameters were used as input to the Locust load function, $N(t)$, as defined by Equation 1: the periodic functions $(600, 50, 0.1)$, $(320, 40, 0.05)$, $(30, 9, 0.01)$, and $(60, 16, 0.05)$; a total duration of 7 days (duration = 604800); the number of intervals for trend computation (num_intervals = 30); the range of mean slopes (slope_mean_range = $(-0.98, 2.5)$); the probability of sudden shifts (prob_sudden_shift = 0.0001); the baseline load (base_load = 20.0); the

anomaly probability (anomaly_prob = 0.00025); and the range of anomaly multipliers (anomaly_multiplier_range = (1, 2)). Table 2 provides a summary of the experiment parameters used to inject EKS cluster anomalies.

Table 2: FIS Experiment Parameters

| **Anomaly** | **Parameters** |
|---|---|
| CPU Stress | Load perc.: 100%, Duration: 4 mins |
| Memory Stress | Load perc.: 100%, Duration: 4 mins |
| Network Latency | Delay: 400 ms, Duration: 5 mins |
| Packet Dropping | Loss Perc.: 40%, Duration: 2 mins |
| I/O Stress | Load perc.: 80%, Duration: 5 mins |
| Pod Deletion | - |
| Node Reboot | - |

The structure and dimensionality of the metrics dataset are summarized in Table 3. This dataset includes all collected metrics along with the labeled ground truth. Additionally, we provide a separate dataset containing information about injected FIS anomalies, which includes their types, targets, and start and end timestamps.

Table 3: Dataset Structure Summary (G.T.: ground truth)

| **Entity Type** | **Metrics** | **#Entities** | **Total** |
|---|---|---|---|
| Node | 25 | 5 | 125 |
| Pod | 18 | 14 | 252 |
| Service | 9 | 10 | 90 |
| Load anomaly G.T. | 1 | - | 1 |
| FIS anomaly G.T. | 1 | - | 1 |
| Anomaly G.T. | 1 | - | 1 |
| **Total** | | | **470** |

Figure 5 illustrates the impact of various anomalies deployed on the Front-end on its key metrics. The network-latency anomaly prominently increases both the average and 95th percentile response times of the Front-end service metrics. Additionally, there is a slight decrease in the received bytes metric and a drop in the request rate. However, metrics such as CPU and memory utilization remain unaffected. The CPU stress anomaly significantly elevates the CPU percentage metric. Despite the CPU percentage occasionally hitting 100%, container or pod restarts are not observed in this scenario. On the other hand, the memory stress anomaly results in heightened memory and CPU percentages but does not cause failures or impact user experience. In contrast, the packet loss anomaly leads to

substantial failures, manifesting in increased p95 response times and the number of failed requests. Ultimately, this anomaly necessitates a container restart.

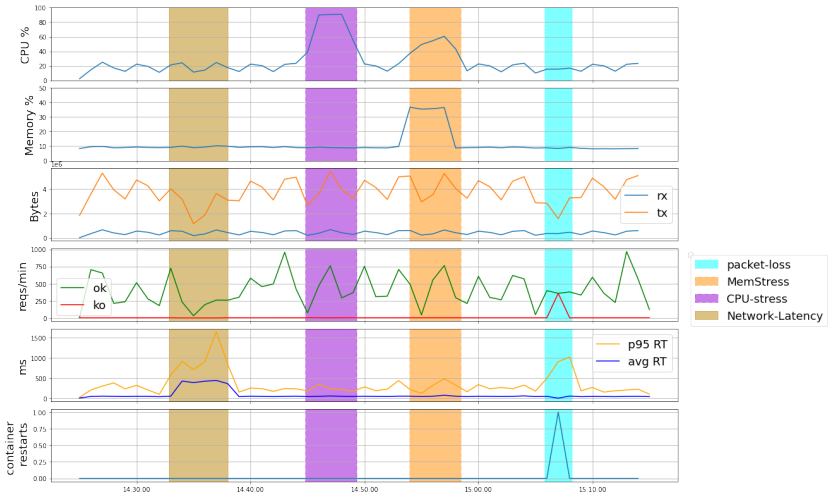



Fig. 5: Effect of various anomalies deployed on the Front-end pod on key metrics

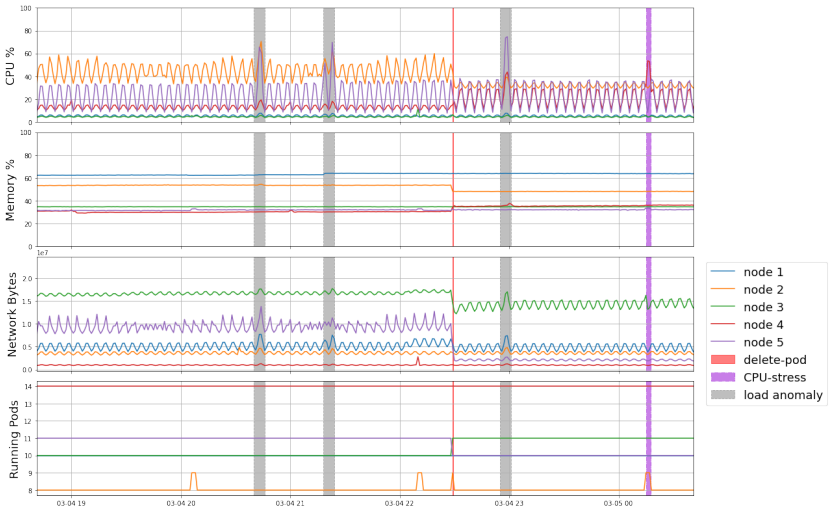



Fig. 6: Effect of various anomalies on key cluster node metrics [load anomalies, CPU stress on Front-end pod, Front-end pod deletion]

Figure 6 illustrates the effects of three types of anomalies on key EC2 metrics of the 5 nodes of the EKS cluster. Load anomalies cause an evident increase in the CPU metrics of various cluster nodes. The same applies to the network bytes metric, while memory metrics are relatively unaffected. The CPU stress anomaly, deployed on the Front-end pod, causes a spike in the CPU utilization metric of the node on which the pod is deployed (node 4). Pod deletion causes significant changes in all metrics, as the pod is restarted on a different node. Thus, we notice a drop in the metrics of node 2 and an increase in those of node 4. Pod deletion is a case of how an error on a certain cluster node affects metrics of a different cluster node.

## 5 Conclusions and Future Work

This paper provides an in-depth overview of a platform designed for deploying microservice applications in a distributed cluster, simulating user load, and injecting various types of anomalies. The platform's effectiveness is demonstrated by generating a multivariate labeled dataset specifically tailored for anomaly detection tasks within a distributed microservice environment. We have described in detail the pipeline implementation that facilitates the deployment and management of microservices, along with the methods required to generate anomalies, label them, and collect observability data. Additionally, we made publicly available two datasets generated using the proposed framework.

Despite the platform's strengths, there are some limitations in this work. While we propose a modular framework capable of supporting various types of microservice applications, our current deployment is limited to e-commerce applications. Additionally, while we gather metrics and logs, we do not include traces in our observability data at present. Traces, which track request propagation across services, are sometimes crucial for detailed error analysis and will be incorporated in our future work. Moreover, our discussion currently focuses only on synthetic data generation for anomaly detection. Future efforts will explore how to leverage our framework and datasets for anomaly detection tasks, including a comparative analysis of different state-of-the-art data-driven anomaly detection algorithms.

## Acknowledgment

This research was conducted with the financial support of Science Foundation Ireland *12/RC/2289_P2* at Insight the SFI Research Centre for Data Analytics at Dublin City University. The data used in this work is fully anonymous. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

## References


1. Amazon Web Services (AWS). https://aws.amazon.com/, accessed: April 15, 2024
2. Boto3 Documentation. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html, accessed: April 18, 2024
3. Amazon Web Services: Amazon Elastic Kubernetes Service (Amazon EKS). https://aws.amazon.com/eks/
4. Amazon Web Services: AWS Fault Injection Simulator. https://aws.amazon.com/fis/
5. Google Cloud Platform: Online Boutique Microservices Demo. https://github.com/GoogleCloudPlatform/microservices-demo
6. Huang, J., Yang, Y., Yu, H., Li, J., Zheng, X.: Twin Graph-based Anomaly Detection via Attentive Multi-Modal Learning for Microservice System. arXiv preprint (2023)
7. Huang, J., Yang, Y., Yu, H., Li, J., Zheng, X.: Twin graph-based anomaly detection via attentive multi-modal learning for microservice system. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 66–78. IEEE (2023)
8. Istio: Istio Service Mesh. https://istio.io/
9. Locust: Locust Load Testing Tool. https://locust.io/
10. Ma, M., Zhang, S., Chen, J., Xu, J., Li, H., Lin, Y., Nie, X., Zhou, B., Wang, Y., Pei, D.: Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems. In: Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21). pp. 413–426. USENIX Association (July 2021), https://www.usenix.org/conference/atc21/presentation/ma
11. Nadareishvili, I., Mitra, R., McLarty, M., Amundsen, M.: Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc." (2016)
12. Nguyen, H.X., Zhu, S., Liu, M.: A survey on graph neural networks for microservice-based cloud applications. Sensors **22**(23), 9492 (2022)
13. Nobre, J., Pires, E.J.S., Reis, A.: Anomaly Detection in Microservice-Based Systems. Applied Sciences **13**(13), 7891 (2023). https://doi.org/10.3390/app13137891, https://www.mdpi.com/2076-3417/13/13/7891
14. Prometheus: Prometheus Monitoring System. https://prometheus.io/
15. Sefraoui, O., Aissaoui, M., Eleuldj, M., et al.: Openstack: toward an open-source solution for cloud computing. International Journal of Computer Applications **55**(3), 38–42 (2012)
16. Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., Pei, D.: Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19). pp. 2828–2837. KDD '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3292500.3330672, https://doi.org/10.1145/3292500.3330672
17. Weaveworks: Sock Shop: A Microservice Demo Application. https://github.com/microservices-demo/load-test (2016)
18. Zhao, C., Ma, M., Zhong, Z., Zhang, S., Tan, Z., Xiong, X., Yu, L., Feng, J., Sun, Y., Zhang, Y., Pei, D., Lin, Q., Zhang, D.: Robust Multimodal Failure Detection for Microservice Systems. arXiv preprint (2023)