Self Healing Microservices Architecture: A case study in Docker Swarm Cluster

Basel Magableh
Dublin Institute of Technology
Dublin, Ireland
basel.magableh@dit.ie

Luca Longo
Dublin Institute of Technology
Dublin, Ireland
luca.longo@dit.ie

ABSTRACT

Each service running in Microservices cluster could be scaled in/out based on the demand issued by end-users, orchestration algorithm, or load balancer running on the swarm manager. This means each service performance and behaviour continuously changes overtime, which makes it a challenge to use a statistical model to identify and detect anomalous behaviour in Microservices architecture. A sudden peak of the CPU some of the time could be considered to be as an anomaly as it might be an action issued by the cluster manager to meet recent high demand. The performance of the cluster nodes could fluctuate around the demand to accommodate scalability, orchestration and load balancing issued by cluster managers. This requires a model that is able to detect anomalies in real-time and generate a high rate of accuracy in detecting any anomalies and a low rate of false alarms. At the same time, it requires dynamic policy configuration that can be used to adapt the recent changes in the operational environment. This research focuses on proposing a self-healing architecture, that is the able to continuously monitor the operational environment, detects and observes anomalous behaviour, and provides a reasonable adaptation policy for self-scaling, self-healing, and self-tuning the computational resources to adapt a sudden changes in its operational environment dynamically at rune-time without human intervention.

CCS CONCEPTS

• **Theory of computation → Unsupervised learning and clustering**; • **Computer systems organization → Cloud computing**; • **Software and its engineering** → Software infrastructure;

KEYWORDS

Self Healing, MicoroServices Architecture, Anomaly detection, Runtime configuration

ACM Reference Format:
Basel Magableh and Luca Longo. 2019. Self Healing Microservices Architecture: A case study in Docker Swarm Cluster. In *Proceedings of ACM SAC Conference (SAC'19).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

SAC'19, April 8-12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

A service oriented architecture (SOA) is a type of architecture for building applications as a set of loosely coupled software components which can be orchestrated to guarantee a high level of interoperability and scalability in distributed infrastructures [18]. The software components could encapsulate an implementation of a microservice which can deliver a well-defined level of functionality to meet business requirements [27]. A Microservices architecture could be defined in the context of a service-oriented architecture composed by tiny fine-grained building blocks of software applications [55]. This allows a software architecture to be composed from a set of distributed loosely coupled microservices [55]. Microservices improve software modularity and make the application easy to develop and maintain. However, With the rapid development of cloud infrastructures and virtualisation techniques, a high demand for building SOA architectures in a complete virtualised environment has emerged. This need was met by introducing container engine like Docker [41], CoreOS [14] as well as cluster management framework such as Docker swarm [2] and Kubernetes [33]. The performance of Microservices running in cluster mode could fluctuate around the demand to accommodate scalability, orchestration and load balancing offered by the cluster manager [55]. In their daily base tasks, many Dev-Ops face an issue: defining and identifying a threshold which can be used to identify a network, system or user activity as normal behaviour. Another challenge that exist in Microservices clusters is the ability to dynamically scale horizontally (Adjusting the number of nodes participating in the cluster) or vertically (Adjusting the computational resources available for the services) without human intervention. In addition to this, it is not possible to configure auto scaling policy that could be used by the cluster manager to perform resilient and autonomous reasoning to achieve the desired QoS of the architecture.

Nowadays, cluster management technology does not embed a component that can guarantees continuous monitoring and adaptation of the operation environment and that can offer the architecture dynamic and self-adaptive capability to perform changes at runtime. To achieve such a high level of scalability, a swarm cluster, for instance, should have a component for continuously monitoring the cluster and a component for adaptation that can implement a reasonable reaction/scaling policy to accommodate the changes in the operating environment. This presents a challenge to build a self-healing microservices architecture that can dynamically adjust its own behaviour and heal itself against anomalous behaviour detected at real-time. Self-healing refers to a property of a self-adaptive software to have the capability of discovering, diagnosing and reacting to disruptions. It can also anticipate potential problems and, accordingly, take proper actions to prevent a failure [50].

To achieve a high level of self-healing it is necessary to have four major functionalities: 1) Monitoring and detecting events (context), 2) Context reasoning, 3) Adaptation strategies, 4) validation and verification of the adaptation action [30]. This research focuses on the proposal of a model that can continuously observe and monitor microservices architectures and can detect anomalous behaviour with high accuracy with a minimal low rate of false alarms. At the same time, this model is envisioned to respond to true positive alarms by suggesting a set of adaptation policies (adaptation strategy) that can be deployed in a cluster to achieve a high degree of self-healing in response to changes in the operating environment.

This paper is structured as follows: Section 2 provides an overview of self-healing architectures and surveys, a description of approaches for anomaly detection, run-time configurations and cluster management. Section 3 presents a model that can continuously observe microservice architectures with Self-healing capabilities. Section 3.1 proposes a strategy for analysing and evaluating the capability of the model to detect anomalous behaviours and to trigger suitable adaptation actions. Section 4 is focused on results followed a by a critical discussion. Section 5 summarises this research, highlighting its contribution and setting future work.

2 RELATED WORK

Self-adaptive software is characterised by a number of properties best referred to as autonomic [29]. These the 'self-* properties' include Self-organisation, Self-healing, Self-optimisation and Self-protection [26]. Self-organising is the capability of a software to self-reconfiguring itself automatically and dynamically in response to changes including installing, updating, integrating, and composing/decomposing software entities [50]. Self-healing is the capability of a software of discovering, diagnosing and reacting to disruptions. It can also anticipate potential problems and, accordingly, take suitable actions to prevent a failure [26]. Self-healing aspects of Microservices architectures requires a decision-making strategy that can work in real-time. This is essential for the software to reason about its own state and its surrounding environment in a closed control loop model and act accordingly [12]. Typically, a self-adapting system implements the following main stages:

(1) Gathering of data related to the surrounding context (Context Sensing);
(2) Context observation and detection;
(3) Dynamic decision making;
(4) Adaptation execution to achieve the adaptation objectives defined as QoS;
(5) Verification and validation of the applied adaptation actions in terms of its ability to meet the adaptation objectives and achieve the desired QoS and at the same time maintaining the architecture state.

This section focuses on discussing the related work of anomaly detection and dynamic decision making based on multi-dimensional utility-based model. However, there is many approaches are used for achieving high level of self-adaptability though Context sensing involving context collection, observation and detection of contextual changes in the operational environment. Also, the ability of the system to dynamically adjust its behaviour can be achieved using parameter-tuning [13], component-based composition [**?**], or Middleware-based approaches [**?**]. Another important aspect of self-adaptive system is related to its ability to validate and verify the adaptation action at runtime based on Game theory [**?**], Utility theory as in [**? ?**], or Model driven approach as in [**?**].

Context information (1) refers to any information that is computationally accessible and upon which behavioural variations depend [24]. Context observation and detection approaches (2) are used to detect abnormal behaviour within the microservices architecture at run-time. Related work in context modelling [54], context detection and engineering self-adaptive software system are discussed in [12, 49, 50]. In dynamic decision making and context reasoning (3), the architecture should be able to monitor and detect normal/abnormal behaviour by continuously monitoring the contextual information found in the microservices cluster. There are two phases for detecting anomalies in a software system: a training phase which involves profiling the normal behaviour of the system; a second phase aimed at testing the learned profile of the system with new data and employing it to detect normal.abnormal behaviours [46]. Three major techniques for anomaly detection have emerged from the literature: statistical anomaly detection, data-mining and machine-learning based techniques [46]. Within the statistical methods, a system observes the activity of the system and generates profiles of system metrics to represent its behaviour. The system profile includes performance measures of the system resources such as CPU and Memory. For each measure, a separate profile is stored. Then, the current readings of the system are profiled and compared against the memorised past profile to calculate an anomaly score. This score is calculated by comparing all measures within the profile against a threshold specified by the developer. Once the system detects that the current readings of the system are higher than this threshold, then these will be automatically categorised as intrusions thus triggering an alert [32, 39]. Various statistical anomaly detection systems have been proposed and they have some advantages [7, 16, 37, 40, 48]. One of this is that they can detect an anomaly without prior knowledge of the system. This can mitigate the common problem of a cold start found in machine learning techniques. Additionally, statistical anomaly detection provides accurate notifications of malicious attacks that occurred over long periods of times and it performs better in detecting denial-of-service attacks [46]. However, a disadvantage is that a skilled attacker might train a statistical anomaly detection system to accept the abnormal behaviour as normal. It is difficult to determine the thresholds that make a balance between the likelihood of a false negative (the system fails to identify an activity as an abnormal behaviour) and the likelihood of a false positive (false alarms). Statistical methods need an accurate model with a precise distribution of all measures. In practice, the behaviour of virtual machines/computers cannot be entirely be modelled using solely statistical methods.

Data mining is about finding insights which are statistically reliable, unknown previously, and actionable from data [47]. The dataset must be available, relevant, adequate, and clean. The data mining process involves discovering a novel, distinguished and useful data pattern in large datasets to extract hidden relationships and information about the data. In general, there are two issues involved in the use of data mining in an intrusion detection system.

First, there is a lack of a large dataset to be used by the algorithm containing lots of information about the Microservices architecture. Second, few approaches were targeting the Intrusion Detection System in Microservices architecture [47]

Data mining based intrusion detection systems have three major difficulties which prevent them from being widely adopted in Microservices architecture [37, 46]. Firstly, the low accuracy of detecting an anomaly [21, 46], as the data mining process would require large dataset with longer time interval to be able to improve the accuracy of detection Most data mining techniques are very heavy on the computational resources, so makes the efficiency has high impact on the computational resources through the training, monitoring and detection phase.As most data mining techniques heavily rely on computational resources, this negatively influeence their adoption in a Microservices architecture [46]. Additionally, usually a data mining method used to classify an attack within a specific system cannot be successfully employed within another system for the same purpose. This because the process of training, testing the model and performing classification of anomalies needs to be repeated with different data or architecture [9].

Machine learning, in the context of anomaly detection, can allow the creation of software system able to learn and improve its detection accuracy over time [10]. Machine learning-based anomaly detection models aims to detect anomalies similar to statistical and data mining approaches. However, unlike them latter which tend to focus on understanding the process that generated the data, the former are data-driven and are mainly focus on training a model based exclusively on past data [46]. This means that, when additional and new data is provided they can intrinsically change their detection strategy and classify significant deviations from the normal behaviour of an underlying software program. An application of Machine Learning which enables the Microservices cluster to distinguish between normal and abnormal behaviour in the data can be found in [9]. In general, Intrusion Detection Systems (IDS) uses a combination of clustering and classification algorithms to detect anomalies. The clustering algorithm is used to cluster the dataset and label them. Then, a decision tree algorithm can be used to distinguish between normal and abnormal behaviour. Golmah [20] suggested the use of an effective classification model to identify normal and abnormal behaviour in network-based IDS. The usage of Machine Mearning algorithm in this context can be found in [6, 9, 17, 20, 22]. Due to the opening deployment and limited resources found in a microservices cluster, it is very important to use a lightweight approach to data clustering and classification as suggested in [36, 48, 52]. Other Machine Learning-based techniques have been employed in [19, 25, 45]. Most of these techniques are targeting network based attacks in distributed environments. However, the techniques described in [19, 25, 45] pay less attention to induce a model of IDS that can optimise the computational resources and has less impact on the deployment mechanisms.

Due to this issue, this research focuses on proposing an anomaly detection mechanism that is more suitable for the Microservices architecture and can be easily deployed with less footprints on the limited resources found in the tiny container.

Several machine learning algorithms have been used in clustering and classifying the network events in IDS. For distributed systems and a wireless sensor network a different approach was used, as shown in [35, 36, 42]. Al-Yaseen, Othman, and Nazri [5] proposed a hybrid mechanism built upon a modified K-means and the C4.5 learning algorithm. Solanki and Dhamdhere [53] used K-means jointly with a Support Vector Machine classifier. The main limitation of K-means clustering is the need to perform initial selection of the data before the clustering can be started.

Numenta Platform for Intelligent Computing (NUPIC) is based on the Hierarchical Temporal Memory (HTM) model proposed in [23]. HTM has been experimentally applied to image recognition [51], natural language processing [8], and most importantly in anomaly detection [4, 34, 56]. A novel approach for anomaly detection in real-time streaming data proposed in [4, 34]. The proposed system based on the HTM model claimed to be efficient and tolerant to noisy data. Most importantly it offers continuous monitoring of real-time data and adapts the changes of the data statistics. It also detects very subtle anomalies with a very minimum rate of false positives. In a recent study, Ahmad et al. [3] proposed an updated version of the anomaly detection algorithm with the introduction of the anomaly likelihood concept. The anomaly score calculated by the NUPIC anomaly detection algorithm represents an immediate calculation of the predictability of the current input stream. This approach works very well with predictable scenarios in many practical applications. As there is no noisy and unpredictable data found the raw anomaly score was gives an accurate prediction of false negatives. However, the changes in predictions would lead to revealing anomalies in the system's behaviour. Instead of using the raw anomaly score, Ahmad et al. [3] proposed a method for calculating the anomaly likelihood by modelling the distribution of anomaly scores and using the distribution to check the likelihood of the current state of the system to identify anomalous behaviour. The anomaly likelihood refers to a metric which defines how anomalous the current state is based on the prediction history calculated by the HTM model. So, the anomaly likelihood is calculated by maintaining a window of the last raw anomaly scores and then calculating the normal distribution over the last obtained values, then the most recent average of anomalies is calculated using the Gaussian tail probability function (Q-function) [15].

I did speak about each of them briefly at the beginning of related work, it is well know in this filed that the literature for each approach is so huge so it is not possible to discuss all of them in one paper

Each service running in the Microservices cluster could be scaled in/out based on the demand issued by end-users, the orchestration algorithm, or the load balancer running on the cluster manager [55]. This means each service performance and behaviour continuously changes overtime, which makes it a challenge to use a statistical model to identify and detect anomalous behaviour. A sudden peak of the CPU some of the time could be considered to be as an anomaly as it might be an action issued by the cluster manager to meet recent high demand. The performance of the cluster nodes could fluctuate around the demand to accommodate scalability, orchestration and load balancing issued by cluster managers. This requires a model that is able to detect anomalies in real-time and generate a high rate of accuracy in detecting any anomalies and a low rate of false alarms. In addition, there will be a set of variations that can be used by the system to adapt the changes in its operational environment. This requires a dynamic decision making that

can calculate the utility of all possible adaptation actions based on the architecture constraints (i.e. number of replicas, number of nodes, desired objectives, metrics thresholds), anomaly score of the detected conditions (CPU, Memory, DISK I/O, Network I/O), and the confidence and accuracy of the anomaly score of the detected abnormal behaviour, and the desired/predicted cluster state. Then, the adaptation manager will execute the adaptation action and verifies its successfulness over the cluster architecture. Also, the adaptation manager will be able to self-tune and self-adjust the architecture parameters to meet high/low demand for services. Finally, the architecture will preserve the cluster state through the adaptation cycle (monitoring, observing, detecting, , reacting, and verifying). This research focuses on finding a method to continuously observe and monitor the swarm cluster and be able to detect anomalous behaviour with a high accuracy and generate a low rate of false alarms. Then provide the architecture with adaptation strategies with high utility to reason about the detected anomalies and be able to self-adjust the architecture parameters and verifies the adaptation actions at runtime without human intervention.

3 DESIGN AND METHODOLOGY

For this aim, This research focuses on proposing a model that can continuously observe and monitor the microservices architecture and be able to detect anomalous behaviour with high accuracy and generate low rate of false alarms. At the same time, the architecture should be able to respond to True positive alarms by suggesting a set of adaptation policies (adaptation strategy), that can be deployed in the cluster to achieve high level of self-healing in response to changes in its operating environment. Because of the uniqueness of streaming data found in microservices cluster, the design of self-healing microservices architecture should meets the following requirements:

(1) The system should be able to operate over real-time data (no look-ahead).
(2) The algorithm must continuously monitor and learn about the behaviour of the cluster.
(3) The algorithm must be implemented with an automatic unsupervised learning technique, so it can continuously learn new behaviour and anomalies in real-time.
(4) The algorithm must be able to adapt the changes of the operating environment and provides adaptation strategy that can be orchestrated over the cluster nodes.
(5) The algorithm should be able to detect anomalies as early as possible before the anomalous behaviour is interrupting the functionality of the running services in the cluster.
(6) The proposed model should minimises the false positives (False Alarms) rate and the false negatives rate. If the system identifies a normal behaviour as an attack, this attempt should be classified as a False Positives (False Alarm).
(7) The proposed model should offer a high detection rate, better accuracy and a lower false alarm rate.
(8) The proposed model should offer consistence adaptation strategy, and preserve the cluster state and it should offer the architecture with a roll back strategy in case the adaptation action failed.
(9) One important aspect of a self-healing Microservices architecture is the ability to continuously monitor the operational environment, detect and observe anomalous behaviour, and provide a reasonable policy for self-scaling, self-healing, and self-tuning the computational resources to adapt a sudden changes in its operational environment dynamically at runetime.

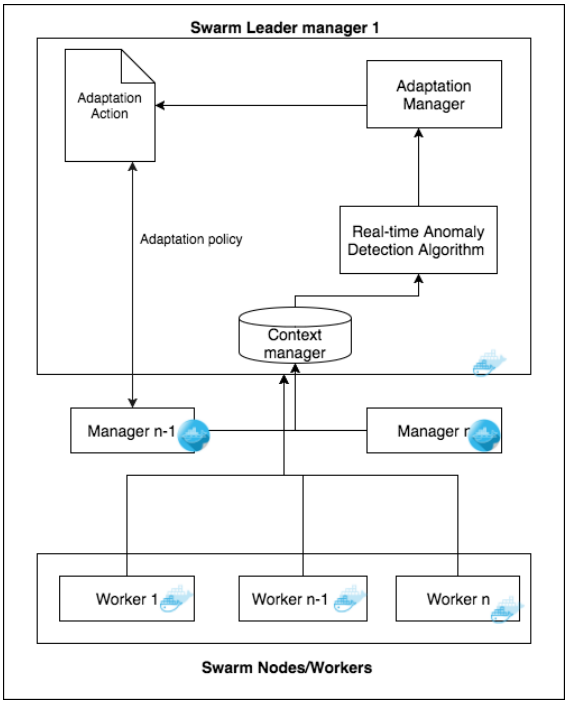

Figure 1: Microservices Architecture implemented in Docker Swarm

To satisfy these requirements of self-adaptability in general and self-healing in specific, each cluster leaders is implemented to include three major components.

The Microservices architecture was implemented in Docker swarm [1] as shown in Figure 2. Docker swarm refers to a cluster management and orchestration functionality embedded in Docker engine [1]. Docker is an open source project aim to build a container platform to facilitate the development and deployment of an application across cloud infrastructure. The main building components of Docker is containers. A container is a lightweight, stand-alone, executable package of a software or service. The container includes everything needed to run an application in the virtualised cloud environment [1].

First, the context manager, this component is responsible for continuously observing and monitoring the behaviour of the architecture and collecting fine-grained metrics about the running services in the containers. The collected data are stored in an influxDB database as shown in Figure 1. InfluxDB is an open-source time series database developed by InfluxData. It is meant to be used as a backing store for any use case involving large amounts

of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytic [28].

The second component, is the a real-time anomaly detection algorithm. This component is responsible for running unsupervised machine learning algorithm based on NUPIC framework [3]. The collected real-time data is feed on the fly to NUPIC machine learning algorithm, which provides two features: first, contentious detection of anomalous behaviour with high accuracy. second, it also provide predictions about the architecture performance based on the collected historic data. This can alerts the architecture about incoming spike on resources demand which can be used by the adaptation manager to schedule a proactive adaptation strategy ahead of time. As shown in Figure 1, the cluster consisted of many managers and many workers. To meet scalability and availability, the cluster manager distributed the work load between the workers based on Raft Consensus Algorithm [44]. This means that each service could be executed by assigning multiple containers across the cluster.

The third component is the adaptation component. The adapter is responsible for executing adaptation strategies based on the anomaly score and the predictions generated by the NUPIC algorithm. Once the adaptation action is completed by the adaptation manager, a set of adaptation actions are deployed in the architecture. To avoid, conflicts between multiple adaptation variations, the adapter allow the adaptation actions to be fully completed and verified by the cluster leader, then it will put a cool off timer before initiating new adaptation actions. This technique is used to avoid resources thrashing and preserving the cluster state for auto-recovery.

The adaptation manager uses dynamic parameters tuning, which makes the Microservices more adaptive to different scenarios. The adaptation manager will use a preconfigured policy issued by the DevOp and reconfigure them on the flay based on the visibility to execute the adaptation action by the consensus algorithm running in the cluster. In different words, the DevOps define the policy parameters that suits the Microservice architecture, but the adaptation manager at runtime adjusts those polices on the flay based on the demand and the runtime requirements. The adjustment of the adaptation policy is a major challenge as there might a finite set of configurations and parameter tuning over unlimited number of contextual conditions. For this aim, the adaptation manager extended a multi-dimensional utility function as in [**?**] to elect the best adaptation action. In this case, the adaptation action refers to any parameter-based or compositional-based configuration of the architecture, maintaining its original functional properties [**?**]. The adaptation manager uses a utility function to calculate and priorities the requirements that need to be meet in the adaptation action. It is assumed that the adaptation manager provides automatic monitoring and detection of the cluster environmental condition. Once an alert is received this will triggers the adaptation manager to calculate the utility function for the context of the operational environment. From utility theory, a von Neumann-Morgenstern [**?**]. Utility function $U_i : X_i \rightarrow \mathbb{R}$ assigns a real number to each quality dimension i, which we can normalize to the range [0, 1]. Across multiple dimensions, we can attribute a percentage weight to each dimension to account for its relative importance compared to other dimensions. These weights form the utility preferences. The overall utility is then given by the utility preference function $fitness_i(V_j, C_m)$ is calculated using the utility function in **??**.

$$fitness_i(V_j, C_m) = \sum_{i}^{k} W_i U_i \tag{1}$$

Example, if three objectives, u1, u2, u3, are given decreasing importance as follows: the first is twice as important as the second, and the second is three times as important as the third. Then the utility fitness would be quantified as $[w1 : 0.6, w2 : 0.3, w3 : 0.1]$. As soon as there is contextual change is detected in the architecture, the adaptation manager computes the utility of all variants related to the operational conditions and decides if an adaptation is required by checking if the current variant is still the one offering the highest utility. So, the utility function is calculated as in **??** proposed in [**?**]. Konstantinos et al. uses weight *wi* to capture the user preferences. In this research we uses the anomaly likelihood calculated by NUPIC for each metric as the weight of the preferences of each context change scaled from 0 to 1. Meaning if the *wi* for the metric *CPU Usage by Node* is 1 and the CPU usage is 70% then this will give the adaptation manager high utility for performing horizontal scaling action i.e. adding new nodes to the cluster.

$$Utility(V_j, C_m) \equiv \frac{\sum_{i=1}^{k}(W_i \cdot fitness_i(V_j, C_m))}{\sum_{i=1}^{k} W_i} \tag{2}$$

Mathematically, if we assume that the new context is $Context_m$ and the system selects variants as the optimal one, then it is implied that the selected variant has a higher (or at least equal) utility, compared to any other variant.

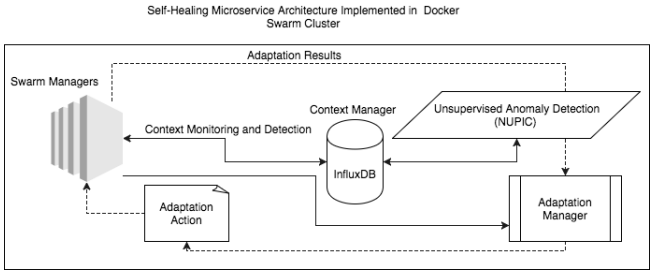

Figure 2: Microservices Architecture implemented in Docker Swarm

4 EXPERIMENTAL EVALUATION

To validate the ideas presented in the paper, we design and develop a working prototype of Microservice architecture in Docker swarm [1] The process can be summarised into three main steps: Feature selection: which involve collecting the data generated from the swarm cluster and use influxdb feature of acyclic graph implementation to filter noisy data and forward to NUPIC only specific metric including (CPU usage, Memory, Disk Reads Bytes/sec, Network Read/s, network write/s and Disk Writes Bytes/sec). Model Training: The second step is to run NUPIC Anomaly detection algorithm over the streamed data collected in Step 1. This will generate a model that will be used to detect anomaly for the incoming data streams from the swarm cluster.

[1] https://docs.docker.com/engine/swarm/

After running the anomaly detection algorithm for each metric. The algorithm provides a value for the anomaly score. The anomaly score varies between 0 to 1. This gives an overall score of how confident the algorithm about the detected anomaly after running over the dataset. In addition, the algorithm provides another indicator called Anomaly Likelihood. The anomaly likelihood refers to a metric which defines how anomalous the current state is based on the prediction history calculated by the HTM model. So, the anomaly likelihood is calculated by maintaining a window of the last raw anomaly scores and then calculating the normal distribution over the last obtained values, then the most recent average of anomalies is calculated using the Gaussian tail probability function (Q-function) [15]. Once their is an anomalous behaviour detected with high score of anomaly likelihood. The adaptation manager will calculate the confidence of the triggered anomaly by calculating the QOS of the anomaly score. Then, the adaptation manager will provides a set of actions like: scale in/out, add/remove nodes, add/remove managers, run security batch, or open/block ports.

The adaptation manager will tune the adaptation policy actions, resources, desired state on the fly as it use a preconfigured template for the adaptation policy. Then, the adaptation manager issues the adaptation policy to the cluster manager for execution. At this stage, the leader and all managers in the cluster will vote based on the consensus algorithm to validate and verifies the adaptation action. If the adaptation action won the votes, the policy will be executed. If the adaptation action lost the vote, then the adaptation manager will re-initialised a new adaptation cycle.

4.1 Evaluation

The evaluation of this model will come in two folds: First, evaluating the accuracy of the anomaly detection algorithms using confusion matrix [31]. Second, evaluating the constancy of the adaptation action and evaluating the state of the swarm after executing the adaptation action. The rate between successful adaptation against failed adaptation is calculated.

4.2 Evaluation Anomaly Detection Accuracy

A famous technique for evaluating the anomaly detection system is the use of a confusion matrix [31]. Kohavi et al. [31] show the criteria for the confusion matrix which represents the total number of records inspected by the intrusion detection system as shown in Table 1. The matrix, which represents the relationship between the actual levels of normal behaviour and the detected level of normal behaviour by the algorithm. Also, it represents the relationship between the actual anomaly recorded in the system and the predicted/identified anomaly that the algorithm identified as anomalous behaviour. In addition, the matrix provides a summary of the evaluation parameters including True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

In case of our proposed model, the NUPIC algorithm will be used to inspect the data set. If NUPIC considered a specific record as anomalous and it was an actual anomaly then this attempt is classified as a **True Positive**. If NUPIC classifies the data as normal behaviour and it was normal data then this attempt is classified as **True Negative**. If NUPIC classifies an anomalous behaviour as normal behaviour this means that NUPIC failed to detect anomaly. In the case, this attempt is classified as **False Negative**. If NUPIC classifies the data as anomalous behaviour but the data was normal behaviour then this attempt is considered a False Alarm/**False Positive**. Both True Positive and False Positive are important benchmark to measure the accuracy of intrusion detection systems.

After calculating the values of TP, TN, FP, and FN, for our proposed algorithm, the confusion matrix will be used to calculate the model detection rate, false positive, and accuracy. Those results will be compared against a well-known anomaly detection system called SNORT as shown in 3.

A manual inspection of a sample data of 1,528 records are used to evaluate the accuracy of anomaly detection in this experiment.

Table 1: Results of the proposed anomalies detection model on confusion matrix

X= 1528	Predicted anomalies	Predicted normality
Actual anomalies (TP + FN) (55)	TP = 49	FN = 6
Actual normalies (FP + TN) (1473)	FP/ False Alarm = 38	TN = 1435

Table 2: Results of the SNORT anomaly detection on confusion matrix

X= 1528	Predicted anomalies	Predicted normality
Actual anomalies (TP + FN) (55)	TP = 32	FN = 23
Actual normalies (FP + TN) (1473)	FP/ False Alarm = 62	TN = 1411

After inspecting the sample data, NUPIC identified 49 True Positives (TP) out of 55 simulated DDOS attacks. The False Positive (FP) was significantly low and it was 38 out of 1473 as shown in Table 1. 1435 records were identified by NUPIC as normal behaviour and they were indeed normal traffic generated from accessing the services running on the Docker swarm. However, NUPIC identified 6 records as False Negatives (FN), which means there was an anomalous attack but NUPIC failed to detect it. The overall accuracy refers to how often the model is correctly identifying anomalous behaviour and it can be computed using the following formula:

$$Accuracy = (TP + TN)/x$$

Based on the values outlined in Table 1, the accuracy of the proposed model equals to $Accuracy = (49+1435)/1528 = 0.97$, and this indicates that the proposed model is highly accurate in detecting anomalies. This value of accuracy is very significant compared with the anomaly detection system of SNORT as shown in Table 2. The misclassification of the proposed model can be calculated by the following formula:

$$Misclassification = (FP + FN)/x$$

So, the proposed model misclassification can be calculated using the above formula and it equals $6 + 38/1528$, $Misclassification = 6.0287$. The True Positive (TP) rate refers to the sensitivity of the model and it can be calculated using the following formula:

$$TP\ rate = TP/(numberofactualanomaliesrecordsinx)$$

Based on the above equation the $TP\ rate = 49/55 = 0.89$. This means the proposed model is significantly accurate in detecting

anomalies in the Docker swarm. The False Positive rate of the model can be computed using the following formula:

$$FP\ rate = FP/(numberofactualnormalrecordsinx)$$

So, the $FP\ rate = 38/1437 = 0.0264$.. This means the model has very low false alarms when detecting anomalies. The FP rate also indicates the model has a high level of accuracy in detecting anomalous behaviour. The specificity of the model-or True Negative- refers to the ratio of correct alarms or anomaly detection rate and it can be computed using the following formula:

$$TN\ rate = TN/(numberofactualnormalrecordsinx)$$

So, the ratio of correct alarms of the proposed model can be calculated using the above formula and it is equal to $TN\ rate = 1435/1437 = 0.998$. This is the model detection rate and it is a great indicator that the model is significantly accurate. The proposed model for precision can be computed using the following formula:

$$Precision = TP/(numberofpredictedanomaliesinx)$$

The precision model equals $Precision = 9/55 = 0.89$. Finally, the prevalence of the model can be computed using the following formula:

$$Prevalence = (Actualanomalies)/x$$

So, the prevalence equals $55/1528 = 0.03599$ and this makes it equal to 0.03599

5 DISCUSSION

Based on the above calculation of SNORT and NUPIC anomaly detection, it was found that the proposed model of using NUPIC performs significantly better in terms of the detection rate, false alarms, and accuracy. The false alarm rate in the model (0.0264) is lower than SNORT (1,12). In addition, the model provides a high rate of true alarms and anomaly detection as summarised in Table 3 The

Table 3: Comparison between SNORT Anomaly Detection and the Proposed Model

Rate	NUPIC	SNORT	Result
Accuracy	0.97	0.944	Implementation of NUPIC provides better accuracy over the SNORT anomaly detection.
Misclassification	0.0287	0.0554	The rate of false alarms is lower in the proposed model compared with SNORT.
True Positive Rate (TP)	0.89	0.58	The rate of true alarms in the model is significantly better than SNORT.
False Positive (FP)/ False Alarm	0.0264	1.12	The proposed model has a lower rate of incorrect classification of attacks, but SNORT has a higher rate of not detecting attacks and considering them as normal behaviour
True Negative (TN	0.998	0.981	Both models- SNORT and the proposed model- have a good rate of detecting the normal behaviour of the system. However, the proposed model provides better results.
Precision	0.89	0.58	The precision of the proposed model is far better than SNORT. The gap between the two models is clear.
Prevalence	0.03599	0.03599	Both models have the same prevalence rate as the DDOS was simulated as part of the experiment.

values in 3 clearly show that the proposed model performs better in terms of false alarms, detection rate, and accuracy. A further calculation, as shown below, could provide a better idea of the performance of the proposed model. The detection rate of SNORT is $32/((32+62)) = 0.340$ and this indicates that the proposed model has a better detection rate, as the proposed model used a machine-learning algorithm to build a model about the data and provides a prediction for each value. Those factors improve the detection rate compared with SNORT, which used a rule-based engine to match the incoming traffic and pre-configured rules.

The False Alarm of the proposed model is 0.0257. This indicates that the proposed model has a very low rate of providing a false alarm or identifying normal behaviour as anomalous. In the contrast, the false alarm for SNORT is $62/((62+1411)) = 0.042$. This value is a clear indication that SNORT would provide more false alarms about normal traffic in the system. This is due to the fact that SNORT considers one factor of identifying an anomaly which is the network traffic. In contrast, the proposed model continuously observed and learned the behaviour of so many performance criteria about the cluster in addition to the network traffic.

The overall accuracy of the proposed model is 0.97. The accuracy of SNORT is 0.944, due to the fact that SNORT is continuously monitoring the system network, but it has no means of understanding the behaviour of the system. The proposed model builds an accurate model of the system behaviour so it has better insight into any fluctuations or spikes in the data.

6 CONCLUSIONS AND FUTURE WORK

The outcomes from this study can be summarised into the following findings:

(1) The introduction of Containers technology advances the implementation of Microservices architecture
(2) Continuous Delivery and Monitoring are great advantages of implementing Microservices architecture in Docker swarm.
(3) Anomaly Detection using statistical approach is not suitable for detecting the anomaly in Docker swarm because:
(4) It is difficult to determine the thresholds of the computational resources in a complete virtualised environment such as Docker swarm.
(5) It is difficult to have an accurate profile of the Docker swarm that has an accurate distribution among all measures
(6) Anomaly detection is one of the most significant, current applications for machine learning in the Internet-of-things and Microservices architecture.
(7) Streaming data introduces big challenges for machine learning models as there is a lack of datasets that can be used for training the model. Most importantly, a new model for training is required when the dataset changes or the environment has changed.
(8) Unlike batch models, where the full dataset is available, streaming models require processing and learning with each data point.
(9) Additionally, real-time data streams are often found with massive volumes and high velocities. This leaves little room for using an unsupervised learning approach with human experts to perform labelling over the data

REFERENCES

[1] 2017. Docker Containers. (2017). https://www.docker.com/

[2] 2017. Docker Swarm. (Dec. 2017). https://docs.docker.com/swarm/swarm_at_scale/deploy-app/

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262, Supplement C (2017), 134–147.

[4] Subutai Ahmad and Scott Purdy. 2016. Real-Time Anomaly Detection for Streaming Analytics. *arXiv preprint arXiv* abs/1607.02480 (2016).

[5] Wathiq Laftah Al-Yaseen, Zulaiha Ali Othman, and Mohd Zakree Ahmad Nazri. 2017. Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system. *Expert Systems With Applications* 67 (2017), 296–303.

[6] J Amudhavel, V Brindha, B Anantharaj, P Karthikeyan, B Bhuvaneswari, M Vasanthi, D Nivetha, and D Vinodha. 2016. A Survey on Intrusion Detection System: State of the Art Review. *Indian Journal of Science and Technology* 9, 11 (March 2016), 1–9.

[7] Debra Anderson, Thane Frivold, and Alfonso Valdes. 1995. Next-generation intrusion detection expert system (NIDES): A summary. (1995).

[8] Itamar Arel, Derek C Rose, and Thomas P Karnowski. 2010. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *IEEE computational intelligence magazine* 5, 4 (2010), 13–18.

[9] Anna L Buczak and Erhan Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials* 18, 2 (May 2016), 1153–1176.

[10] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. 2012. A method for classification of network traffic based on C5. 0 Machine Learning Algorithm. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*. IEEE, 237–241.

[11] F Chang and V Karamcheti. 2000. Automatic configuration and run-time adaptation of distributed applications. *hpdc* (2000), 11.

[12] BHC Cheng, R de Lemos, H Giese, P Inverardi, J Magee, R M Malek, H Müller, S Park, M Shaw, and M Tichy. 2008. Software Engineering for Self-Adaptive Systems: A Research Road Map (Draft Version). *Dagstuhl Seminar Proc. 08031* (2008).

[13] S W Cheng, D Garlan, and B Schmerl. 2009. Evaluating the effectiveness of the rainbow self-adaptive system. (2009).

[14] coreos. [n. d.]. rkt, a security-minded, standards-based container engine. ([n. d.]). https://coreos.com/rkt/

[15] John W Craig. 1991. A new, simple and exact result for calculating the probability of error for two-dimensional signal constellations. In *Military Communications Conference, 1991. MILCOM'91, Conference Record, Military Communications in a Changing World., IEEE*. IEEE, 571–575.

[16] Dorothy Denning and Peter G Neumann. 1985. *Requirements and model for IDES-a real-time intrusion-detection expert system.* SRI International.

[17] Frank Doelitzscher, Christoph Reich, Martin Knahl, Alexander Passfall, and Nathan Clarke. 2012. An agent based business aware incident detection system for cloud environments. *Journal of Cloud Computing: Advances, Systems and Applications* 1, 1 (2012), 9.

[18] Thomas Erl. 2005. *Service-oriented architecture: concepts, technology, and design.* Pearson Education India.

[19] MF Md Fudzee and J Abawajy. 2008. A classification for content adaptation system. *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services* (2008), 426–429.

[20] Vahid Golmah. 2014. An efficient hybrid intrusion detection system based on C5. 0 and SVM. *International Journal of Database Theory and Application* 7, 2 (2014), 59–70.

[21] Dikshant Gupta, Suhani Singhal, Shamita Malik, and Archana Singh. 2016. Network intrusion detection system using various data mining techniques. In *Research Advances in Integrated Navigation Systems (RAINS), International Conference on*. IEEE, 1–6.

[22] Nutan Farah Haq, Abdur Rahman Onik, Md Avishek, Khan Hridoy, Musharrat Rafni, Faisal Muhammad Shah, and Dewan Md Farid. 2015. Application of machine learning approaches in intrusion detection system: a survey. *International Journal of Advanced Research in Artificial Intelligence* (2015).

[23] Jeff Hawkins and Sandra Blakeslee. 2007. *On Intelligence.* Macmillan.

[24] R Hirschfeld, P Costanza, and O Nierstrasz. 2008. Context-oriented Programming. *Journal of Object Technology* (Jan. 2008).

[25] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Pierre-Louis Dubouilh, Ephraim Iorkyase, Christos Tachtatzis, and Robert Atkinson. 2016. Threat analysis of iot networks using artificial neural network intrusion detection system. In *Networks, Computers and Communications (ISNCC), 2016 International Symposium on*. IEEE, 1–6.

[26] Paul Horn. 2001. *Autonomic computing: IBM's Perspective on the State of Information Technology.* Technical Report.

[27] Judith Hurwitz, Robin Bloor, Marcia Kaufman, and Dr Fern Halper. 2009. Service Oriented Architecture For Dummies, 2nd IBM Limited Edition. (April 2009), 1–76.

[28] InfluxDB. [n. d.]. InfluxData Documentation. ([n. d.]). https://docs.influxdata.com/

[29] Ozalp Babaoglu Márk Jelasity, Alberto Montresor Christof Fetzer, Stefano Leonardi Aad van Moorsel, and Maarten van Steen. [n. d.]. Self-star Properties in Complex Information Systems. ([n. d.]).

[30] G M Kapitsaki, G N Prezerakos, N D Tselikas, and I S Venieris. 2009. Context-aware service engineering: A survey. *Journal of Systems and Software* 82, 8 (2009), 1285–1297.

[31] R Kohavi and F Provost. 1998. Confusion matrix. *Machine learning* 30, 2-3 (1998), 271–274.

[32] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 251–261.

[33] Kubernetes. 2018. Kubernetes. (Jan. 2018). https://kubernetes.io/

[34] Alexander Lavin and Subutai Ahmad. 2015. Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark. *CoRR* abs/1510.03336 (2015), 38–44.

[35] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 120–132.

[36] Yang Li, Bin-Xing Fang, You Chen, and Li Guo. 2006. A lightweight intrusion detection model based on feature selection and maximum entropy model. In *Communication Technology, 2006. ICCT'06. International Conference on*. IEEE, 1–4.

[37] Teresa F Lunt, Ann Tamaru, and F Gillham. 1992. *A real-time intrusion-detection expert system (IDES).* SRI International. Computer Science Laboratory.

[38] B Magableh. 2012. *Context-Oriented Component-based Software Development.* Ph.D. Dissertation. Trinity College Dublin, Dublin.

[39] Constantine Manikopoulos and Symeon Papavassiliou. 2002. Network intrusion and fault detection: a statistical anomaly approach. *IEEE Communications Magazine* 40, 10 (2002), 76–82.

[40] Roy A Maxion and Frank E Feather. 1990. A case study of ethernet anomalies in a distributed computing environment. *IEEE transactions on Reliability* 39, 4 (1990), 433–443.

[41] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[42] A Mishra and A K Misra. 2009. Component assessment and proactive model for support of dynamic integration in self adaptive system. *ACM SIGSOFT Software Engineering Notes* 34, 4 (2009), 1–9.

[43] Aitor Murguzur, Rafael Capilla, Salvador Trujillo, Óscar Ortiz, and Roberto E Lopez-Herrejon. 2014. Context variability modeling for runtime configuration of service-based dynamic software product lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. ACM, 2–9.

[44] Diego Ongaro and John Ousterhout. 2015. The raft consensus algorithm. (2015).

[45] Hamed Haddad Pajouh, Reza Javidan, Raouf Khayami, Dehghantanha Ali, and Kim-Kwang Raymond Choo. 2016. A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IoT backbone networks. *IEEE Transactions on Emerging Topics in Computing* (2016).

[46] Animesh Patcha and Jung-Min Park. 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *COMPUTER NETWORKS* 51, 12 (Aug. 2007), 3448–3470.

[47] Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. 2010. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119* (2010).

[48] Martin Roesch. 1999. Snort: Lightweight intrusion detection for networks. In *Lisa*. 229–238.

[49] Holger Giese Hausi A Müller Mary Shaw Jesper Andersson Luciano Baresi Basil Becker Nelly Bencomo Yuriy Brun Bojan Cukic Ron Desmarais Schahram Dustdar Gregor Engels Kurt Geihs Karl M Goeschka Alessandra Gorla Vincenzo Grassi Paola Inverardi Gabor Karsai Jeff Kramer Marin Litoiu Antonia Lopes Jeff Magee Sam Malek Serge Mankovskii Raffaela Mirandola John Mylopoulos Oscar Nierstrasz Mauro Pezzè Christian Prehofer Wilhelm Schäfer Rick Schlichting Bradley Schmerl Dennis B Smith João P Sousa Gabriel Tamura Ladan Tahvildari Norha M Villegas Thomas Vogel Danny Weyns Kenny Wong Jochen Wuttke Rogério de Lemos. 2011. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap (Draft Version of May 20, 2011). (May 2011), 1–16.

[50] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *Transactions on Autonomous and Adaptive Systems (TAAS* 4, 2 (May 2009).

[51] R Škoviera and I Bajla. 2013. Image classification based on hierarchical temporal memory and color features. *Slovak Academy of Sciences, MEASUREMENT* (2013).

[52] Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, L Todd Heberlein, Che-Lin Ho, Karl N Levitt, Biswanath Mukherjee, Stephen E Smaha,

Tim Grance, and others. 1991. DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype. In *Proceedings of the 14th national computer security conference*. Washington, DC, 167–176.

[53] Meghana Solanki and Vidya Dhamdhere. 2013. Intrusion Detection System by using K-Means clustering C 4.5 FNN SVM classifier. (2013).

[54] T Strang and C Linnhoff-Popien. 2004. A context modeling survey. *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp* (2004).

[55] Joe Stubbs, Walter Moreira, and Rion Dooley. 2015. Distributed systems of microservices using docker and serfnode. In *Science Gateways (IWSG), 2015 7th International Workshop on*. IEEE, 34–39.

[56] SM Mohammadzadeh Ziabary and Shahram Khadivi. 2010. HLMT: Human-Like Machine Translation Inspired by Bilinguals' Cortex Activity and Translation Behavior. (2010).