IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 August 2022, accepted 24 September 2022, date of publication 19 October 2022, date of current version 25 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3216080

RESEARCH ARTICLE

Detection of Cluster Anomalies With ML Techniques

JOANNA KOSIŃSKA, (Member, IEEE), AND MACIEJ TOBIASZ
Faculty of Computer Science, Electronics and Telecommunications, Institute of Computer Science, AGH University of Science and Technology, 30-059 Krakow, Poland

Corresponding author: Joanna Kosińska (kosinska@agh.edu.pl)

This work was supported in part by the funds assigned to the AGH University of Science and Technology by the Polish Ministry of Science and Higher Education, and in part by the PLGrid Infrastructure.

ABSTRACT Due to the increasing complexity of computing clusters, it becomes more challenging to identify erroneous behavior inside them. Monitoring systems collect large amounts of data to provide analysis of cluster behavior. Their manual study is expensive and sometimes even impossible. Hence, a system capable of analyzing the gathered data and then, based on those data, detecting anomalies within the cluster is essential. Our paper proposes a system for detecting an anomaly in a Kubernetes cluster (KAD - Kubernetes Anomaly Detector). KAD uses machine learning techniques to diagnose problems that may arise. The novelty of our solution lies in proposing the concept of using various machine learning models that facilitate the detection of different types of anomalies. The user can choose which model to use for anomaly detection in the given situation. The KAD system can also automatically select the appropriate model. The selection is based on scoring procedures. The system trains the models using historical data that describe the usual behavior of the cluster. Then it detects anomalies based on the predictions of the trained model or signal reconstructions. Finally, in two groups of experiments, we evaluate the proposed concepts. The experiments confirm the importance of selecting the proper ML model to detect anomalies in different situations. Furthermore, the experiments assess the latency of the responses in a production-ready cluster.

INDEX TERMS Anomaly detection, cluster anomaly, kubernetes, learning model, ML, training dataset.

I. INTRODUCTION

The way of designing and deploying software has changed rapidly in recent years. Before Cloud Computing (CC) has become a trend, most enterprise applications were modeled according to a monolithic architecture. This approach suffered from several drawbacks, such as problematic scaling and maintenance [1]. To overcome the limitations of monolithic architectures, microservices have emerged [2], [3]. This approach distributes the application logic into multiple loosely coupled services that communicate through lightweight protocols. With the expansion of containerization [4], [5], [6] and CC technologies, the developed software has overcome the limitations caused by monoliths. However, in addition to the numerous benefits, there are also new challenges that container-based microservices present. Having multiple independent services within an application increases the complexity of tasks related to its configuration and management [7]. Manual administration is not efficient in this case. The need for automatic service management is addressed by an orchestrator. The container orchestrator, like Kubernetes [8], manages the deployment, scaling, and interactions between containers. Although orchestrators offer functionalities that significantly facilitate container management, cluster environments are still sophisticated. Only interaction between all parts of the system enables the final functionality to be delivered. It is necessary to monitor all components of the environment to ensure that the entire cluster is healthy. Furthermore, under constant observation [9] must be not only all components of the environment, but also the cooperation between different parts of the system. Therefore, cluster monitoring is essential. As can be seen, the

The associate editor coordinating the review of this manuscript and approving it for publication was Filbert Juwono.

monitoring system collects a large amount of data that need to be processed.

Accurate acquisition and interpretation of monitoring data require experienced human specialists. It would be beneficial to have a fully automated solution to detect unusual behavior in the cluster environment. The observations show how demanding cluster error detection is. Fortunately, existing anomaly detection techniques address this issue. In particular, the proposed systems use machine learning (ML) algorithms. Their ML models can be trained with data collected by deployed monitoring systems.

In recent years, both cluster monitoring and machine learning have been evolving dynamically. There are always new challenges to face. However, they are solved quickly. Most solutions address all scenarios with a common concept (in this case, with a common ML model). However, we claim that cluster anomaly detection with ML techniques can gain even greater accuracy if the technique is customized (properly selected ML model) to a given scenario.

The contribution of our research is the proposition of a solution to detect real-time anomalies in the Kubernetes cluster with ML techniques. The novelty of our solution lies in proposing the concept of supporting multiple complementary machine learning models to maximize detection accuracy. Depending on the scenario, the appropriate ML model is used. To evaluate the correctness of our concept, we implemented a system named Kubernetes Anomaly Detector (KAD). It has been evaluated in two groups of experiments. The first group emphasizes the usage of different machine learning algorithms in different cases. The second group measures the latency of the responses after reconfiguration.

The structure of the paper is as follows. The first section introduces the background and motivation of the work. Then, in the next section, an overview of existing approaches related to the problem is presented. Section III describes the machine learning models that we use. Section IV describes the proposed architecture of the KAD system. The following section evaluates our concept. It is divided into two subsections, each presenting a different experiment. Both subsections begin with an illustration of the environment prepared to test our proposition. These subsections end with a conclusion of the results achieved using different machine learning algorithms. Finally, Section VI summarizes our work and describes potential areas of further research.

II. RELATED WORK

The paper [10] proposes to divide existing solutions for cluster error detection into two groups. These groups are based on policies and anomalies. Policy-based solutions [11], [12], [13] are built using a predefined set of rules. These rules consider various aspects of the system. Sysdig's Falco [11] project allows us to define a set of rules to detect malicious activity in applications. The usability of policy-based systems is strongly limited, as they detect errors that are well defined and taken into account before creating a set of rules. This knowledge is impossible in containerized environments because they change rapidly. Therefore, the types of errors also change (for example, a new error related to horizontal scalability [14] occurs), limiting the usability of policy-based cluster error detection systems.

The second category [15], [16], [17] of cluster error detection systems copes with the frequent changes mentioned. Anomaly-based detection systems infer patterns from collected data and can easily adapt to a constantly changing container environment [10]. The paper [18] presents an example of an anomaly-based system. This paper shows that machine learning classification algorithms can correctly detect simulated faults in the CPU, memory, and network. Another example of an anomaly-based system is KubAnomaly [10]. Notable is the fact that its aim is similar to ours, but there are subtle differences that should be pointed out. It uses deep learning techniques to detect container security risks. The system runs on Kubernetes and uses custom agents to collect data from the nodes. This research shows that efficient anomaly detection in the Kubernetes cluster is possible without significant performance overhead. The KubAnomaly system detects anomalies related only to container security, as opposed to our solution, where we do not restrict the types of anomaly. The second difference is that KubAnomaly uses log-based monitoring with a custom data processing mechanism. The custom log parser may significantly limit the usability and portability of the system. Our solution does not impose such limitations.

Most papers use various, but always a single, machine learning model. Model examples are Isolation Forests [19] or HHMM (Hierarchical Hidden Markov Models) [20]. The HHMM-based solution, DLA (Detection and Localization System for Anomalies), is particularly worth noting since it proposes a system that localizes a component that causes the anomaly. However, the experiments were carried out only for several scenarios and metrics. The algorithm may not be optimal in different use cases. There also are some approaches ([21], [22]) that, with success, do not use ML at all. The paper [23] evaluates different anomaly detection methods (OC-SVM [24], LOF [25], LSTMs [26], and autoencoders [27]) in the context of Kubernetes self-healing and autoscaling mechanisms. All the papers presented show that the analyzed methods, particularly LSTM, may help to manage the Kubernetes cluster [28].

Cluster error detection can be categorized as a particular problem for time series anomaly detection. This point of view allows for the benefit of both advanced metric-based monitoring systems and time series analysis models. The authors of [29] compare different existing time series analysis models in the context of anomaly detection. In addition, they propose a novel deep learning approach that outperforms the other methods. Although that paper does not directly concern cluster error detection, it is worth noting because it describes practices that can be used to solve the problem of cluster error detection. Additionally, the paper uses the same dataset for

evaluation as in our work: the Numenta anomaly benchmark dataset.

Our research overcomes most of the limitations mentioned above. The novelty of the proposed concept results from the support of multiple complementary machine learning models to achieve accurate results in different scenarios. The relevant ML model is chosen manually by the user or automatically according to the scenario. The concept assumes generality, allowing us to incorporate various ML models, particularly those recommended by the research described in this section.

III. BASICS OF CLUSTER ANOMALIES IN ML-CONTEXT

The paper [30] defines an anomaly as something unusual, unexpected, or different from what normally happens, but with a neutral overtone. An anomaly may mean a serious problem, a great opportunity, or something completely different. Accurate anomaly detection is often a benefit as it allows for an in-time reaction to changing circumstances. In machine learning, an anomaly refers to a deviation from the expected pattern. Consequently, the definition of anomaly detection is that it is a problem of finding data points that do not follow the expected pattern [31].

There are several categories of anomaly detection methods, distinguished by the existence of labels and anomalous points in the training set. Their taxonomy is presented in [31].

A. MODELS

To achieve high accuracy and elasticity, we propose the use of various statistical and machine learning models. The models are selected in a way that covers complementary approaches. Each model is based on different assumptions and represents a distinct approach to the problem. Our solution uses four types of models. The first two models (SARIMA and HMM) come from traditional time-series analysis and represent the family of statistical models. In contrast, the last two models (LSTM and Autoencoder) are deep learning models. Each model we chose was based on studies in the literature (Section II) and their recommendations.

Fig. 1 shows the anomaly-based error detection procedure. In general, to detect errors using ML, two subsequent phases are necessary. These are (i) the training phase and then (ii) the proper anomaly detection phase. A Data Source (e.g. a monitoring system) is responsible for providing data for further processing. During the training phase, the selected model learns from the available data. Then, during the anomaly detection phase, the model classifies a batch of data points in a loop.

The algorithm 1 describes the general procedure for anomaly detection used by all chosen models. First, the algorithm adjusts the model parameters (line 5) with a model-specific training procedure. The training procedure returns errors (residuals) that establish a threshold (line 6). This threshold is used in more detail in the anomaly detection phase. The anomaly detection phase begins with a model-specific procedure, that is, prediction or reconstruction of the anomaly detection dataset (line 7). On the basis of the prediction or reconstruction of the model, the anomaly detection error is calculated (line 8). Then the calculation of the anomaly score follows (line 9).

The procedures used to train and detect anomalies using concrete models are a specialization of the general anomaly detection procedure (the general procedure was presented in the Algorithm 1). The following subsections briefly describe each of the concrete models, focusing on the differences in the model-specific steps of the general procedure.

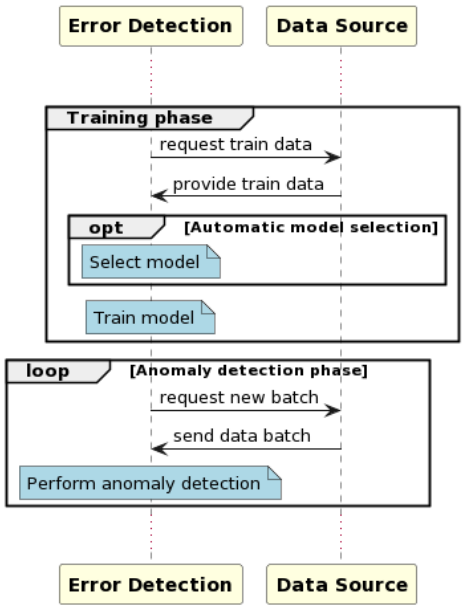


FIGURE 1. Sequence diagram of anomaly-based error detection.

Algorithm 1 General Procedure of Anomaly Detection

Input:

1: M – model,
2: TR – training dataset,
3: AD – anomaly detection dataset.

Output:

4: sAD – scored AD dataset.
5: $train\_err \leftarrow train(M, TR)$ ▷ variable *train_err* stores errors (residuals) computed on validation dataset. It is further used to establish the threshold.
6: $thr \leftarrow max(train\_err)$ ▷ *thr* denotes the error threshold. Stored in the purpose of anomaly detection phase.
7: $pred \leftarrow predict(M, AD)$ ▷ *pred* variable stores predictions or reconstructions of the anomaly detection dataset that the trained model provides.
8: $detect\_err \leftarrow abs(pred - get\_next\_batch(AD))$ ▷ Based on the results stored in *pred* variable and the actual anomaly detection dataset, errors *detect_err* are calculated.
9: $sAD \leftarrow detect\_err / thr$ ▷ sAD enhances AD by anomaly score, i.e. a value between 0 and 1, which denotes how anomalous (according to the system predictions) a data point is.

1) SARIMA

SARIMA [32] is a statistical model, often used in time-series analysis, to learn the dynamics of the signal and the relationships between current and previous samples. The detection of anomalies using the SARIMA model begins with the selection of hyperparameters. Examples of hyperparameters include the order of seasonal and non-seasonal autoregressive (AR) components, the moving average (MA) parameters, the differencing order, and the seasonal period which must be selected. We propose choosing hyperparameters in a grid-search process that maximizes AIC and BIC criteria [33] and avoids overfitting. Then the model should be trained using nonanomalous data (semi-supervised learning), during which the model learns (non)seasonal autoregressive and moving average parameters. Finally (line 7 of Algorithm 1), it is necessary to calculate the predictions of the model.

2) HMM

HMM [34] assumes that the observed data depend on latent variables. This assumption allows for the modeling of hidden patterns that often occur in time series. We first propose training the model on a dataset composed of non-anomalous data using the EM algorithm [35]. After establishing the parameters of the model, it determines the most probable sequence of latent variables that generated the training data. On the basis of this sequence, it generates samples (reconstructions). The reconstructions are then compared to the actual time series in order to return the training errors. Similarly to the training phase, the anomaly detection phase of the HMM model uses reconstruction errors, in contrast to the SARIMA model, which uses prediction errors.

3) LSTM

If this ML model is enabled, we propose to have two layers: a dropout layer to prevent overfitting [36] and a dense layer. During the training procedure, the LSTM uses the backpropagation algorithm. The idea behind using LSTM in the anomaly detection system is similar to that described in the subsection that presents the SARIMA model. The LSTM model uses prediction errors to classify samples as anomalous.

4) AUTOENCODER

The neural network of the autoencoder [37], [38], [39] consists of two parts: encoder and decoder. The autoencoder learns the representation of the data and reconstructs the original data. We propose to use an autoencoder consisting of a single LSTM (appropriate for time-series modeling) and a dropout layer [36] in both the encoder and decoder parts.

In this work, the autoencoder performs anomaly detection based on the reconstruction error, similarly to the HMM model.

IV. KUBERNETES ANOMALY DETECTOR

We implemented a system named the Kubernetes Anomaly Detector (KAD) to evaluate the correctness of our concept of supporting various complementary machine learning models to maximize the detection accuracy. It is publicly available at https://github.com/emerelte/kad. The system we propose uses semi-supervised anomaly detection, which assumes that the training data set is clean. This assumption is justified if the training dataset does not contain anomalies. In this case, it is not necessary to label the dataset.

KAD consists of several components that cooperate to detect anomalies in the Kubernetes cluster. As depicted in Fig. 2 it consists of six components:

System Core – manages the internal state of the system and interacts with the user through exposed APIs (depicted as flow ⑪). KAD exposes two endpoints. The first exposes the anomaly detection results, and the second is related to the configuration settings. The endpoint named *results endpoint* contains the processed time series and the associated anomaly score. The data available at the results endpoint also contain target anomaly labels (binary values that indicate whether the processed sample is classified as anomalous) and the predictions/reconstructions of the signal that the system produced. The second endpoint allows system reconfiguration during runtime through the ConfigMap [40] abstraction. Table 1 gives a brief description of all parameters of the system configuration.

After the system starts, it loads the desired configuration. During the training phase, it retrieves the training data from the Data Source component (depicted as flow ①) and transfers them to the Model Selector component (flow ⑤) to perform model selection. The training phase ends after the model is selected and trained. The Model Selection Component notifies the System Core when the training phase ends and returns the trained model (flow ⑥). Then the System Core periodically queries the Data Source Module to fetch a new portion of the anomaly detection data. It passes the data to the Model component, which performs anomaly detection. Finally, the appropriate endpoint exposes the results.

Data Source – its role is to provide data needed to train machine learning models and perform online anomaly detection. It obtains raw monitoring data from a given period (flow ②) and transforms them into a form that other components can process.

Data Processing – transforms the input data to an appropriate format and processes them to achieve the needed characteristics. The Data Source Component delivers the input data (depicted as the flow ③). Next, the Data Processing Component responds with the preprocessed data (see flow ④). The appropriate preprocessing technique is selected when the time series characteristics (seasonality, stationarity, noise, etc.) are known.

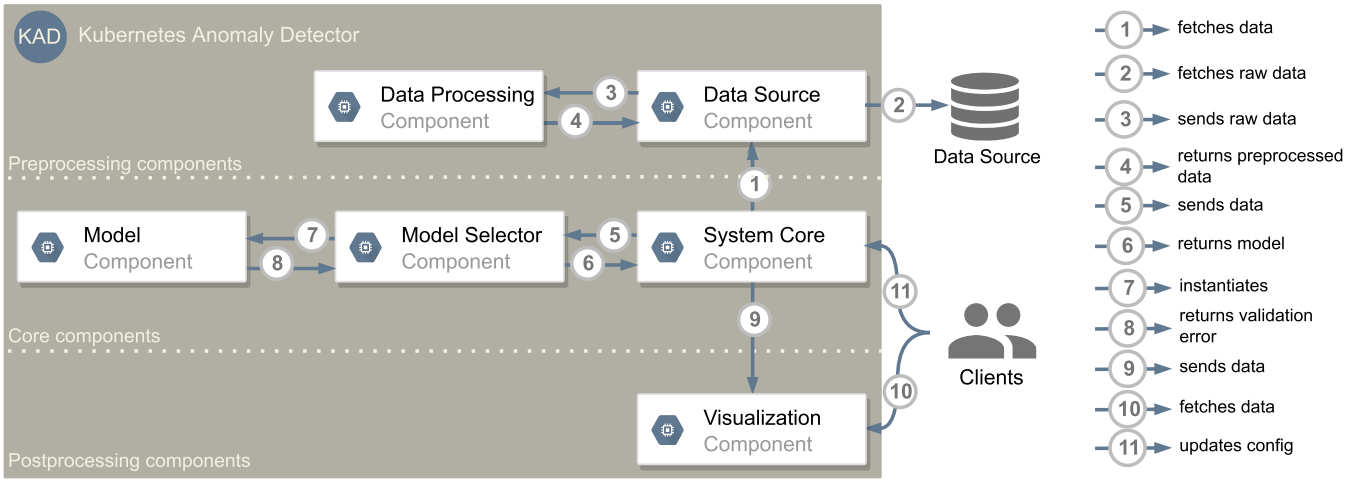


FIGURE 2. Simplified architecture of the KAD.

TABLE 1. System configuration parameters.

Parameter name	Description	Modifiable (at runtime)	Mandatory
Start time	Start of training phase	✓	✓
End time	End of training phase	✓	✓
Query	Query for a desired time-series (format depending on the data source, in particular this is a Prometheus query, e.g. $rate(req\_dur\_seconds\_count[1m])$)	✓	✓
Update interval	Time between consecutive queries		✓
Application URL	URL where the system will be accessible		✓
Results endpoint	System results endpoint (containing original time-series, anomaly score and labels in JSON format)		✓
Config endpoint	Endpoint for system configuration updates		✓
Model	The selected model (if empty, automatic model selection is performed)	✓	

These techniques include resampling, differencing, FFT, etc.

Model – a valid model has to implement methods `train` and `detect`. The `train` method returns the validation error (depicted as the flow ⑧) that is needed to select an appropriate model using the Model Selector component. On the other hand, the output of the `detect` method contains information about the results of anomaly detection, such as the anomaly score or the label.

Model Selector – realizes our concept of using a proper ML model in different cases. Model Selector component selects the model in one of the available modes:

- Manual, where the user directly chooses a particular model. Inside the selected instance of the model, it determines all its necessary parameters based on the time-series characteristics (like period or stationarity) or a grid-search procedure through the parameter space.
- Automatic, where the system decides which model is the most suitable for the given data. The model selection procedure instantiates (see flow ⑦) all available models and performs a training procedure on each of them. After completion of the training, it compares the validation errors of different models. The automatic model selection procedure returns the model with the lowest validation error. The validation error calculations are the same for each type of model and are based on the mean squared error (MSE).

Visualization – presents the anomaly detection results to the users (depicted as flow ⑩). It uses plotting libraries to create graphs related to the analyzed time-series. The resulting graphs contain the actual time-series predictions made by the model and the detected anomalies. Additionally, a separate graph presents the anomaly score calculated by the system. The System Core component provides all data required to present the results (flow ⑨).

The machine learning model adjusts its parameters during the anomaly detection phase. Only the hyperparameters remain unchanged. The rest parameters (e.g. neural network weights) can be refined in the training or anomaly detection phases. It makes it possible to detect anomalies in a rapidly changing

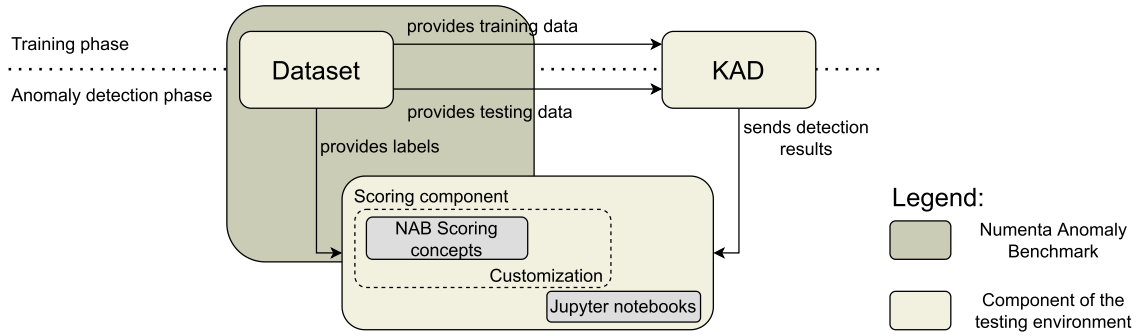


FIGURE 3. Components of the evaluation environment of Experiment 1.

TABLE 2. NAB dataset characteristics.

Data	Source	Size [points]	Exemplary content	Contains anomalies
artificialNoAnomaly	Artificially generated	20160	Square wave, flat line	
artificialWithAnomaly	Artificially generated	24192	Square wave with an unexpected peak	✓
realAWSCloudwatch	AWS server metrics	67740	CPU utilization, network traffic	✓

cluster, where the system needs to continuously monitor the environment.

KAD solutions are unique among existing anomaly detection systems. This system offers an elastic interface that allows the use of various machine learning models. Different models can be matched to different types of data. KAD tries to select the most suitable model for each new data to maximize accuracy.

V. EXPERIMENTAL EVALUATION

When evaluating our concept of offering various complementary machine learning models, we divided them into two groups of experiments. We divided the experiments according to the kind of data (input) that we used to train the models and then detect the anomalies. The distinguished experiments present what follows:

1) The selection of anomaly detection model,
2) The latency of responses in the production-ready cluster.

In addition, we present the evaluation environment with its configuration. The description is divided into two subsections. Each subsection is for a particular experiment group.

A. EXPERIMENT 1: THE SELECTION OF ANOMALY DETECTION MODEL

The evaluation environment is depicted in Fig. 3. In this scenario, the models described in Section III-A were trained on the Numenta Anomaly Benchmark (NAB) dataset [41]. The dataset served as a Data Source for KAD. For the experiment, we implemented a special kind of KAD Client, named a Scoring Component (Fig. 3). The Scoring Component's role is to calculate and visualize metrics of different anomaly detection models. It was implemented in Python, using Jupyter.

The NAB dataset consists of 58 data streams belonging to different domains. Some of the streams are artificial, but most of the streams are real-world examples. Some of the data streams contain anomalies with a known root cause. Table 2 presents the general characteristics of the dataset. The paper [42] presents a detailed explanation of the characteristics of the NAB dataset, particularly the labeling procedure.

In addition to the dataset, the Numenta Anomaly Benchmark delivers its environment and a procedure [43] to score the models. The KAD's interface is not compliant with the NAB scoring system. Thus, the Scoring Component calculated customized metrics based on the NAB concepts (this depicts Fig. 3).

1) CUSTOMIZED ANOMALY DETECTION SCORING PROCEDURE

Paper [42] presents the foundations of the NAB dataset. The paper states that the dataset is useful for anomaly detection systems based on streaming data. However, KAD's models rely on batch processing, which does not conform to the NAB DUT (Detector Under Test) interface. Therefore, we customized the scoring procedure as described below.

The procedure is based on the concepts introduced by the authors of the NAB dataset. We defined three components of the final algorithm's score that reflect the ideas of NAB in the batch-processing environment:

- accuracy component – rewards models that can detect the exact anomalous points,
- collective component – rewards a model for detecting a collective anomaly,
- precision component – punishes a model for false positive alerts.

Each scoring component returns a number between 0 and 1, where 0 indicates the worst score, and 1 is the best possible result. The final score is a weighted average of the three components. The idea of a weighted average reflects the concept of an application profile presented in NAB [42].

The accuracy component detects the closest anomaly to the ground-truth anomaly. If the model finds the exact anomalous timestamp, it is rewarded with the maximal score – 1. The reward decreases linearly with the distance between the ground-truth anomaly and the closest alert. If the detection came from outside the anomaly window, the accuracy component returns 0:

$$acc\_score = max\{0, 1 - \frac{\sum_{k=1}^{K} \frac{2d_k}{W_k}}{K}\} \quad (1)$$

where:

K - number of ground truth anomalies
d_k - distance between k^{th} ground truth anomaly and its closest prediction
W_k - width of k^{th} anomaly window

In the NAB scoring procedure, the calculations are based on the first sample from the anomaly window to award the earliest detection in the streaming data. Taking into account the characteristics of KAD, we decided to maximize the accuracy score in the middle of the anomaly window. Adjusting the original NAB scoring procedure for KAD is justified, as it uses batch processing and takes multiple data points as a single input. The procedure is also capable of finding the anomaly as accurately as possible.

The accuracy component rewards accurate classifications. However, it does not consider the characteristics of the anomaly. It is sufficient for point anomalies, but does not provide appropriate scores for collective anomalies, frequently present in the NAB corpus. This characteristic led us to introduce the second component. It measures the density of alerts raised in the anomaly window. In the case of collective anomalies, we expect alerts in multiple places across the anomaly window. The following equation reflects these expectations:

$$coll\_score = min\{1, \frac{\sum_{n=1}^{N} max\{0, f(n)\} * p_n}{\sum_{n=1}^{N} max\{0, f(n)\}}\} \quad (2)$$

where:

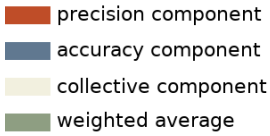
n - sample number
N - number of samples
$f(n)$ - sigmoidal weight function
p_n - n^{th} element of model predictions array (1 - anomaly, 0 - normal)

The sigmoidal weight function is defined as follows:

$$f(n) = \frac{2}{1 + \exp(|n - m| - \frac{W}{2})} - 1 \quad (3)$$

n - sample number (absolute)
m - index of the closest ground truth anomaly
W - width of the closest anomaly window

The collective component has no counterpart in the NAB scoring procedure, which takes into account only the first point of the anomaly window. We have decided to incorporate it into the KAD scoring framework to verify whether machine learning models can detect collective anomalies.



The weighted average (that we finally calculate as an arithmetic average) of scoring components facilitates visual comparison of the different models.

FIGURE 4. Legend of all graphs in Experiment 1.

The scoring function on points outside the anomaly window is used for calculations of the last scoring component, the precision component. The lower its value, the severer punished would a model be for a false positive in a given timestamp. The formula for the precision component is as follows:

$$prec\_score = min\{1, 1 - \frac{\sum_{n=1}^{N} min\{0, f(n)\} * p_n}{\sum_{n=1}^{N} min\{0, f(n)\}}\} \quad (4)$$

where:

n - sample number
N - number of samples
$f(n)$ - sigmoidal weight function
p_n - n^{th} element of model predictions array (1 - anomaly, 0 - normal)

The precision component rewards a model for not making inappropriate alerts when there are no anomalies.

The NAB scoring procedure also penalizes models for false positive predictions. The sigmoidal weight function (Equation 3) used in Equation 4 is conceptually close to the scoring function defined in the NAB paper [42].

The described calculations score KAD models according to their anomaly detection processes.

2) RESULTS

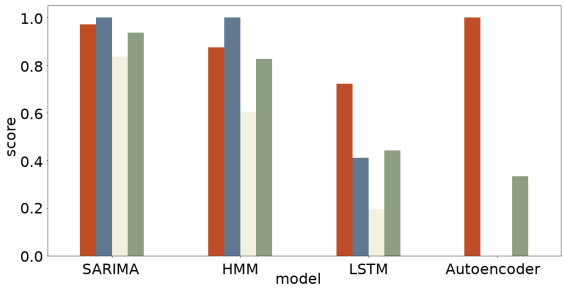
The following subsection presents the results of the chosen machine learning models gained on the NAB dataset.

Both Figs. 5a and 5b depict the comparison of the chosen models. The bars represent the three components of the customized anomaly detection scoring procedure. The Y-axis shows the scores obtained for the calculations.

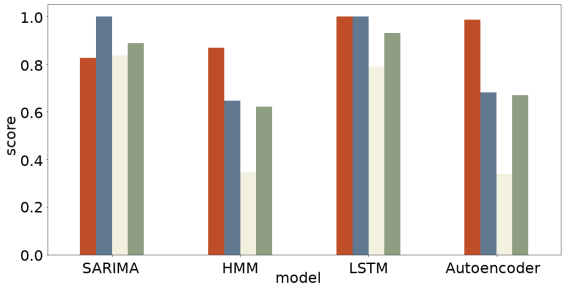
The graph shows (particularly Fig. 5b) that the SARIMA model achieved a score significantly higher than 0.8 in each of the scoring components.

3) DISCUSSION OF RESULTS

Fig. 5a shows that the statistical models (SARIMA and HMM) perform better on the trivial art_daily_flatmiddle.csv example. On the other hand, deep learning models (autoencoder and LSTM) seem to have problems learning the system dynamics. On another example of a trivial artificially created time-series – art_daily_jumpsup.csv – the neural



(a) Chart for art_daily_flatmiddle.csv
In the data, there is one collective anomaly. It is different from the rest of the time-series. The signal is a regular square signal.



(b) Chart for art_daily_jumpsup.csv
Dataset contains one anomaly window that covers an unexpected peak.

All files are from the NAB corpus and contain artificially generated data.

FIGURE 5. Comparison of latency of chosen anomaly detection models.

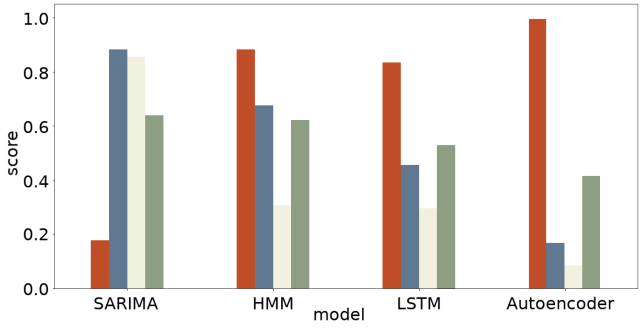


FIGURE 6. Average scores achieved by the analyzed models on artificially generated data from NAB corpus.

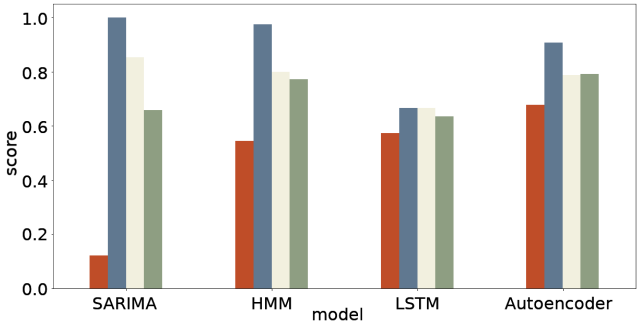


FIGURE 7. Average scores achieved by the analyzed models on data presenting EC2 CPU utilization collected by AWS Cloudwatch.

network models achieve higher scores on this example than on 5a – see Fig. 5b.

To summarize the results obtained, we once again tested the KAD models. We selected two types of data streams. Both are from the NAB dataset. The first stream is artificially generated (named artificialWithAnomaly), and the second contains data presenting CPU utilization collected from AWS Cloudwatch (this stream is named realAWSCloudwatch). Figs. 6 and 7 depict the scores achieved by the KAD models. The results show that the models detect anomalies accurately both in artificial data and in real-world data. It is worth noting that the statistical models (SARIMA and HMM) achieve higher scores on the artificial data. The artificial examples from the NAB corpus contain trivial patterns from which we can deduce hyperparameters of the statistical models. On the other hand, neural network-based models, particularly the autoencoder, perform better on AWS Cloudwatch data that contain more complex patterns. However, the drawback of the autoencoder is its poor interpretability compared to the statistical models.

B. EXPERIMENT 2: THE LATENCY OF RESPONSES IN THE PRODUCTION-READY CLUSTER

The configuration depicted in Fig. 8 of the evaluation environment served to carry out Experiment 2.

We based the research infrastructure on an OpenStack cloud software. Within the cloud, we deployed multiple virtual servers. The infrastructure was partially supported by PLGrid [44]. The OpenStack's Compute Nodes built up a clustered environment. The deployed Kubernetes cluster consisted of 20 nodes. That is, there are 19 workers and one master node. In the cluster, we have installed the KAD system and an example of a microservice application. For testing purposes, we used a demo open-source Sock Shop application developed by Weaveworks (https://www.weave.works). The application consists of 14 microservices. In Fig. 8 they are denoted as multiple microservices at the back of the Sock Shop Demo Application component. The application is available under Apache License 2.0. It exposes metrics for scraping by a Prometheus [45] instance. Additionally, we developed a Python client application that served as a front-end for KAD.

To simulate user behavior, we used the Locust load generator tool [46]. It provides a convenient way to generate a specified number of application users that perform a given number of queries at given endpoints. We used this mechanism to generate normal and anomalous application loads. We have periodically spawned a specific number of clients. Table 3 presents the configuration parameters that helped to generate the non-anomalous load. A Prometheus instance has been monitoring the number of requests to the application that was proportional to the number of spawned clients. Thus, the KAD instance received a simple signal with a visible seasonality. To generate anomalies, we have significantly modified the number of clients in specific periods. The simplicity of the input data allowed us to focus on the performance aspects of the individual models (since all of them recognized anomalies

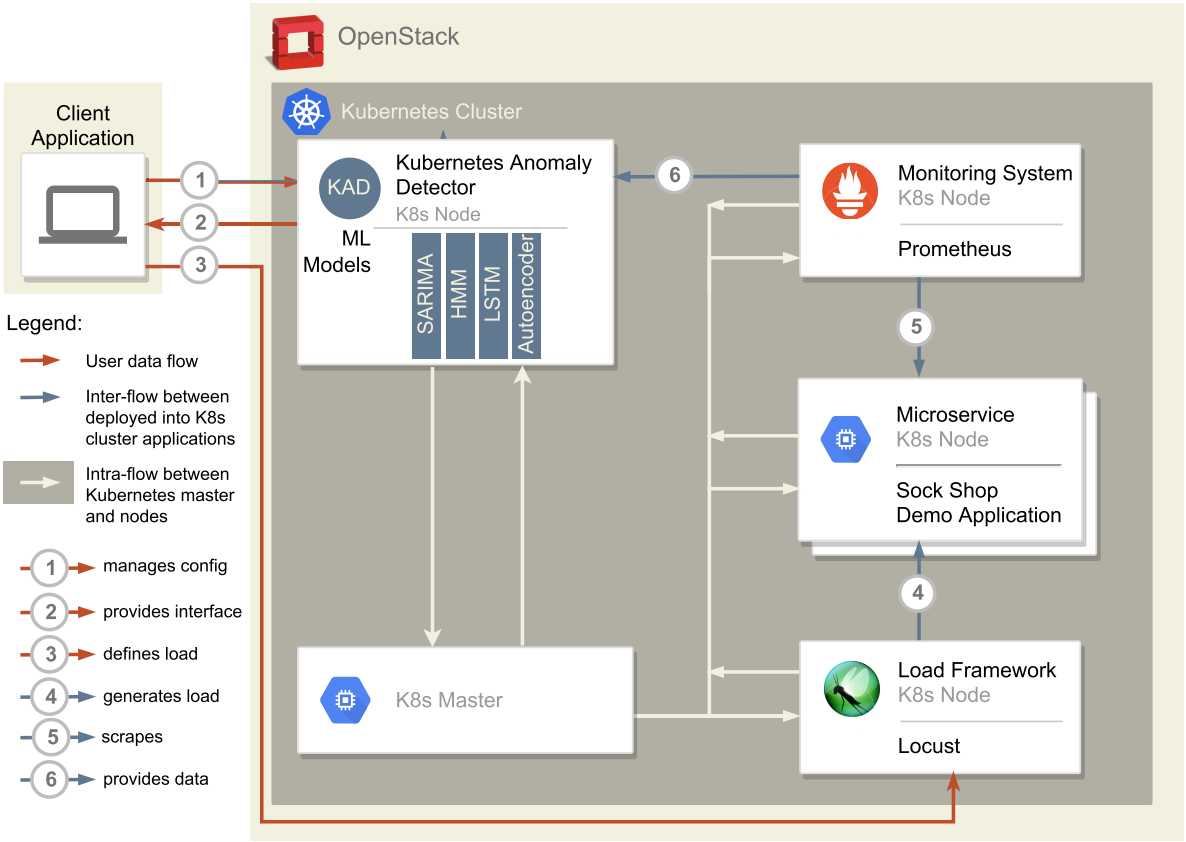


FIGURE 8. Evaluation environment of Experiment 2.

TABLE 3. Configuration of the normal load.

Key	Value
No clients	0 – 2000
Spawn rate	500 clients/second
Cycle period	500 seconds

TABLE 4. Flows of Experiment 2.

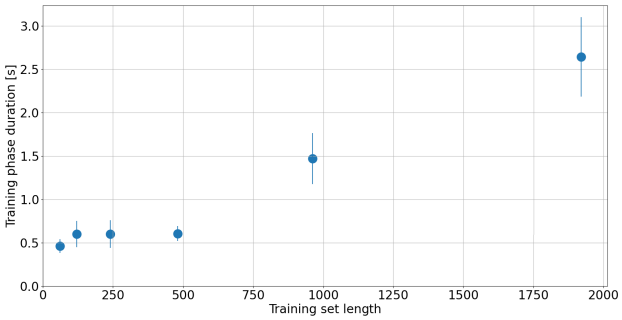
No	Name	Description
①	manages config	The client sets and updates system configuration parameters
②	provides interface	KAD exposes endpoints for configuration management and results obtaining.
③	defines load	The client prepares and sends the desired load pattern.
④	generates load	Locust Framework generates appropriate load in the microservices, according to the pattern defined by the client.
⑤	scrapes	Prometheus monitors the application, collecting the desired metrics.
⑥	provides data	Prometheus forwards the collected metrics to KAD.

with relatively similar and high accuracy), which was the primary purpose of the experiment.

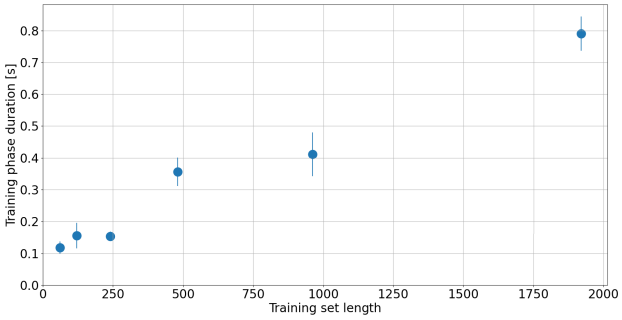
Fig. 8 shows three types of flow. They are depicted as different colors of lines: (i) red - denoting user data flow, (ii) dark blue - denoting internal flows between applications installed in the Kubernetes cluster, and (iii) creme - for intra communication among Kubernetes components (e.g. tasks of the control plane, tasks of the etcd key value store, etc.). From the perspective of Experiment 2, the first and second flows are important. All are described in Table 4.

The experiment's goal was to measure the training performance of the machine learning models, i.e. the relationship between the data set size and the time needed to train the model. To achieve the results, the client collected the latencies of the KAD responses for a set of queries. Each query triggered the KAD reconfiguration, causing the training procedure to begin with a dataset of a specified size. On the basis of the measured latencies, we prepared the visualizations shown in Fig. 9. Each figure relates to a different machine learning model. The presented characteristics result from averaging five consecutive runs of the experiment. By repeating its run, we intended to increase the reliability of the results.

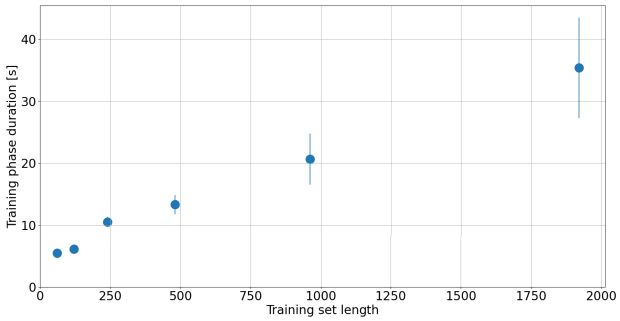
The delta between the two neighboring points is twice as large as the previous one. The results show that the duration of the training phase of different models differs significantly for the same data.



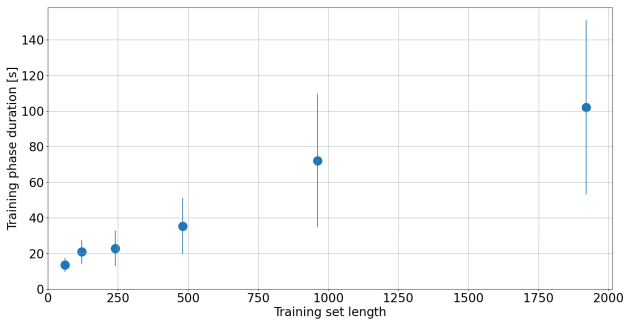
(a) Results for SARIMA model



(b) Results for HMM model



(c) Results for LSTM model



(d) Results for Autoencoder model

We prepared a Python client that assessed the KAD latency. The client measured the time needed to acquire results after a configuration change. It performed a set of 5 queries with the same configuration to estimate the time needed to process data of a specific size. The client repeated the experiment for different time ranges.

FIGURE 9. **Duration of the model training versus size of the training set.**

1) DISCUSSION OF RESULTS

Both statistical models (i.e. SARIMA and HMM) require little time to train the model compared to neural networks. However, the user should pass the correct parameters to the model (number of hidden states in case of HMM or seasonality regarding SARIMA, etc.). A grid-search procedure can select the parameters, but this significantly prolongs the time needed to train the statistical models correctly. However, the trivial data pattern (i.e. we can clearly distinguish different hidden states in the observed data) allows for manual parameter adjustment. As shown in Fig. 5a, HMM and SARIMA can perform with precision on such data. Both models can be the best choice because of their short training time.

C. EVALUATION SUMMARY

The acquired characteristics of particular models are as follows:

SARIMA – This model achieved the highest score when classifying artificial data from the NAB corpus (Fig. 6). It performed slightly worse on real data from AWS Cloudwatch (Fig. 7). The experiments show that SARIMA produces good results when the data contain a trivial pattern, such as a clear seasonality. Then, even if the size of the training set is limited, the SARIMA results can be accurate. Notable is the low precision of the SARIMA model. It means that the analyzed datasets produced many false-positives alarms. This may be a reason not to use the SARIMA model for applications where frequent notifications are not tolerable. The unquestionable advantages of the SARIMA model are its high interpretability [32] and its ability to learn efficiently even on small datasets.

HMM – This model accurately detected anomalies in the data of both types analyzed (Experiment 1). It achieved higher precision and lower average training time than the SARIMA model. Its interpretability is not always clear as the meaning of the hidden states is not always known. It can serve as an alternative to the SARIMA model when the data pattern consists of several separable states, e.g. when the time series contains several periods in which the data come from different distributions. Furthermore, the HMM model is preferred when performance is crucial.

LSTM – This model performed slightly worse than the HMM model on both datasets (Experiment 1). Furthermore, it took significantly longer to train when given the same data (Experiment 2). Its results are poorly interpretable, as it is based on neural networks. Its numerous parameters do not have as concrete meaning as the parameters of the statistical models. On the other hand, a large number of parameters make the LSTM model suitable for modeling a broader class of time-series data, as shown in Fig. 7, showing that LSTM also performs well in sophisticated examples.

Autoencoder – The autoencoder obtained the highest results in the data collected from the monitoring of EC2 instances (Fig. 7). As in the case of the LSTM model, the ability to learn multiple classes of patterns can compensate for the poor interpretability of the model.

The autoencoder needed the most training time among the other models (Experiment 2). Moreover, it needs a large dataset to produce valuable results. However, it can still be the best solution for scenarios where the data pattern is not transparent, i.e. where it is hard to describe data characteristics in terms of statistical parameters.

The results presented from the evaluation confirm our concept. The experiments show that the diversity of the models may positively impact the final predictions. Different machine learning models can be accurate in different scenarios. Thus, it can be beneficial to select from a range of complementary models.

VI. CONCLUSION

This paper describes our concept of anomaly detection that uses machine learning to diagnose problems within a Kubernetes cluster. Our concept benefits from the support of various complementary machine learning models. We propose to use different ML models in different scenarios. The evaluation of our concept is carried out by the developed Kubernetes Anomaly Detector (KAD) system. Currently, the system supports four models: SARIMA, HMM, LSTM and autoencoder. The possibility of selecting from various models can improve its overall performance in different scenarios. KAD uses a form of ensemble learning to maximize its accuracy. Additionally, KAD enables system reconfiguration at runtime.

Our concept was tested in two different environment configurations that built up the testbeds for the experiments. The first experiment used the Numenta Anomaly Benchmark dataset as a data source. The authors of this dataset provided the ground truth labels, helping to measure the accuracy of different models. We introduced a customized anomaly detection scoring algorithm that is based on the notions of anomaly windows and the sigmoidal weight function. The algorithm scores the total accuracy. This experiment proved the correctness of assuming the usage of different machine learning algorithms in different scenarios. In the second experiment, we measured the latency of the responses after reconfiguration. To carry out this experiment, we deployed a microservices application in the production environment. The Locust generator simulated a normal and anomalous load within the cluster. Meanwhile, the experiment showed that the developed KAD system can be successfully deployed in a Kubernetes cluster to perform real-time anomaly detection.

The experiments showed the potential of our concept and its usability. However, they also revealed areas that need further improvement. The current KAD implementation features only univariate models. This means that anomaly detection can be performed only on one metric at a given time. The introduction of multivariate models would allow us to address more complex cases.

Another notable improvement direction is the identification of failures in cluster components. If the system diagnoses the root cause of the faulty behavior, it could provide a self-healing mechanism allowing its autonomous [47], [48] recovery.

REFERENCES

[1] M. F. J. Lewis. (Mar. 2021). *Microservices—A Definition of This New Architectural Term*. [Online]. Available: https://Martinfowler.com/articles/microservices.html
[2] J. Scott, *A Practical Guide to Microservices Containers*. Santa Clara, CA, USA: MAPR, 2017.
[3] S. Newman, *Building Microservices*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2015.
[4] S. Maheshwari, S. Deochake, R. De, and A. Grover, "Comparative study of virtual machines and containers for DevOps developers," 2018, *arXiv:1808.08192*.
[5] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Apr. 2016, pp. 202–211.
[6] H. van Ham, J. van Ham, and J. Rijsenbrij, *Development of Containerization: Success Through Vision, Drive and Technology*. Amsterdam, The Netherlands: IOS Press, 2012. [Online]. Available: https://books.google.pl/books?id=CgQmkTczzPwC
[7] D. Ernst, D. Bermbach, and S. Tai, "Understanding the container ecosystem: A taxonomy of building blocks for container lifecycle and cluster management," in *Proc. 2nd Int. Workshop Container Technol. Container Clouds*, 2016, pp. 1–6.
[8] *Kubernetes Site*. (Feb. 2022). [Online]. Available: https://kubernetes.io
[9] J. Kosińska and K. Zieliński, "Autonomic management framework for cloud-native applications," *J. Grid Comput.*, vol. 18, no. 4, pp. 779–796, Dec. 2020.
[10] C. Tien, T. Huang, C. Tien, T. Huang, and S. Kuo, "KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches," *Eng. Rep.*, vol. 1, no. 5, Dec. 2019.
[11] (Feb. 2021). *Falco, the Open-Source Cloud-Native Runtime Security Project*. [Online]. Available: https://sysdig.com/opensource/falco/
[12] (Feb. 2022). *Holistic Kubernetes Security for the Enterprise*. [Online]. Available: https://www.aquasec.com/products/kubernetes-security/
[13] (Feb. 2022). *Twistlock: Container Security Product Overview and Analysis*. [Online]. Available: https://www.esecurityplanet.com/products/twistlock/
[14] J. Kosinska and K. J. Zielinski, "Experimental evaluation of rule-based autonomic computing management framework for cloud-native applications," *IEEE Trans. Services Comput.*, early access, Mar. 15, 2022, doi: 10.1109/TSC.2022.3159001.
[15] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48231–48246, 2018.
[16] S. A. Ajila, C.-H. Lung, and A. Das, "Analysis of error-based machine learning algorithms in network anomaly detection and categorization," *Ann. Telecommun.*, vol. 77, nos. 5–6, pp. 359–370, Jun. 2022.
[17] E. Eskin, "Detecting errors within a corpus using anomaly detection," in *Proc. 1st Meeting North Amer. Chapter Assoc. Comput. Linguistics*, New York, NY, USA: Association for Computational Linguistics, 2000, pp. 148–153.
[18] Q. Du, T. Xie, and Y. He, "Anomaly detection and diagnosis for container-based microservices with performance monitoring," in *Algorithms Architectures for Parallel Processing*, J. Vaidya and J. Li, Eds. Cham, Switzerland: Springer, 2018, pp. 560–572.
[19] Z. Zou, Y. Xie, K. Huang, G. Xu, D. Feng, and D. Long, "A Docker container anomaly monitoring system based on optimized isolation forest," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 134–145, Jan. 2022.
[20] A. Samir and C. Pahl, "DLA: Detecting and localizing anomalies in containerized microservice architectures using Markov models," in *Proc. 7th Int. Conf. Future Internet Things Cloud (FiCloud)*, 2019, pp. 205–213.
[21] T. F. Düllmann, "Performance anomaly detection in microservice architectures under continuous change," Tech. Rep., 2017.
[22] Z. Guan, J. Lin, and P. Chen, "On anomaly detection and root cause analysis of microservice systems," in *Service-Oriented Computing—ICSOC 2018 Workshops*, X. Liu, M. Mrissa, L. Zhang, D. Benslimane, A. Ghose, Z. Wang, A. Bucchiarone, W. Zhang, Y. Zou, and Q. Yu, Eds. Cham, Switzerland: Springer, 2019, pp. 465–469.
[23] Y. Matsuo and D. Ikegami, "Performance analysis of anomaly detection methods for application system on Kubernetes with auto-scaling and self-healing," in *Proc. 17th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2021, pp. 464–472.

[24] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, Oct. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320316300267
[25] T. Huang, Y. Zhu, Q. Zhang, Y. Zhu, D. Wang, M. Qiu, and L. Liu, "An LOF-based adaptive anomaly detection scheme for cloud computing," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf. Workshops*, Jul. 2013, pp. 206–211.
[26] S. Hochreiter and J. J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 80–1735, 1997.
[27] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proc. 2nd Workshop Mach. Learn. Sensory Data Anal. (MLSDA)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 4–11, doi: 10.1145/2689746.2689747.
[28] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for Kubernetes edge clusters," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 958–972, Mar. 2021.
[29] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "DeepAnT: A deep learning approach for unsupervised anomaly detection in time series," *IEEE Access*, vol. 7, pp. 1991–2005, 2019.
[30] (Mar. 2021). *Macmillan Dictionary*. [Online]. Available: https://www.macmillandictionary.com/dictionary/british/anomaly
[31] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.
[32] C. R. George Box, G. Jenkins, and G. Ljung, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2016.
[33] J. Kuha, "AIC and BIC: Comparisons of assumptions and performance," *Sociol. Methods Res.*, vol. 33, no. 2, pp. 188–229, Nov. 2004, doi: 10.1177/0049124103262065.
[34] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Germany: Springer-Verlag, 2006.
[35] G. Xuan, W. Zhang, and P. Chai, "EM algorithms of Gaussian mixture model and hidden Markov model," in *Proc. Int. Conf. Image Process.*, vol. 1. Oct. 2001, pp. 145–148.
[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html
[37] D. Bhowick, D. K. Gupta, S. Maiti, and U. Shankar, "Stacked autoencoders based machine learning for noise reduction and signal reconstruction in geophysical data," 2019, *arXiv:1907.03278*.
[38] K. Panguluri and K. Kamarajugadda, "Image generation using variational autoencoders," *IJITEE Int. J. Inf. Technol. Electr. Eng.*, Mar. 2020.
[39] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, Aug. 2017, p. 665, doi: 10.1145/3097983.3098052.
[40] (Mar. 2022). *Kubernetes Configmaps*. [Online]. Available: https://kubernetes.io/docs/concepts/configuration/configmap/
[41] (Jan. 2022). *The Numenta Anomaly Benchmark (NAB) Homepage*. [Online]. Available: https://github.com/numenta/NAB
[42] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," 2015, *arXiv:1510.03336*.
[43] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217309864
[44] (Mar. 2022). *PLGrid Home Page*. [Online]. Available: https://www.plgrid.pl
[45] B. Brazil, *Prometheus: Up & Running*. Sebastopol, CA, USA: O'Reilly Media, 2018.
[46] (Sep. 2021). *Locust Homepage*. [Online]. Available: https://locust.io/
[47] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003, doi: 10.1109/MC.2003.1160055.
[48] P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Armonk, NY, USA: IBM, 2001.

JOANNA KOSIŃSKA (Member, IEEE) received the Ph.D. degree in information and communication technology. She is currently an Assistant Professor at the Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland. Her research interests include distributed computing, specifically cloud-native computing and resource management.

Since 2020, she has been the CEO of the Try IT Foundation. The Foundation aims to promote the discipline of IT among girls and women. Since then, with the Institute of Computer Science, AGH University of Science and Technology, Krakow, and the Try IT Foundation, she has been the Director of the "Girls Go IT" Grant.

MACIEJ TOBIASZ received the M.Sc. degree in computer science from the AGH University of Science and Technology, Krakow, Poland. He is currently working as a software developer at telecommunications industry. His research interests include cloud computing and machine learning.