Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs

Pawan Kumar Sarika
Ericsson AB
Stockholm, Sweden
pawan.sarika@ericsson.com

Deepika Badampudi
Blekinge Institute of Technology
Karlskrona, Sweden
deepika.badampudi@bth.se

Sai Prashanth Josyula
Blekinge Institute of Technology
Karlskrona, Sweden
sai.prashanth.josyula@bth.se

Muhammad Usman
Blekinge Institute of Technology
Karlskrona, Sweden
muhammad.usman@bth.se

Abstract

Kubernetes is a free, open-source container orchestration system for deploying and managing Docker containers that host microservices. Kubernetes cluster logs help in determining the reason for the failure. However, as systems become more complex, identifying failure reasons manually becomes more difficult and time-consuming. This study aims to identify effective and efficient classification algorithms to automatically determine the failure reason. We compare five classification algorithms, Support Vector Machines, K-Nearest Neighbors, Random Forest, Gradient Boosting Classifier, and Multilayer Perceptron. Our results indicate that Random Forest produces good accuracy while requiring fewer computational resources than other algorithms.

Keywords: Kubernetes cluster logs, microservices, machine learning

ACM Reference Format:
Pawan Kumar Sarika, Deepika Badampudi, Sai Prashanth Josyula, and Muhammad Usman. 2023. Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3593434.3593472

1 Introduction

Ericsson, a leading Information and Communication Technology (ICT) service provider, is at the forefront of pioneering cloud RAN[1], a technology that handles network traffic in the cloud. To increase development efficiency, Ericsson developed the Application Development Platform (ADP) ecosystem [4], which among other things, includes a marketplace that hosts over 280 microservices. Among these, 50+ microservices are common microservices that can be reused and integrated across all applications within the organization. The microservices in the ADP ecosystem are hosted in Docker containers and managed by Kubernetes.

Some applications need a specific set of microservices that may differ from others. In addition, the applications may deploy microservices in different environments, such as Amazon Web Service (AWS), or Microsoft Azure Cloud. To ensure high quality, the ADP ecosystem includes a Continuous Integration and Continuous Deployment (CICD) team, which simulates the different application environments and performs various tests, including testing scalability, robustness, and resilience.

Whenever a test fails, the developers must classify the failure into the following categories - an issue in the cluster, artifactory, microservice, CICD tests itself, or environment. The classification is done to report the failure to the relevant team. For example, microservice bugs are assigned to the relevant microservice team, while CICD bugs are reported to the CICD team for resolution. In addition, knowing the failure reason can provide Ericsson insights on what they should improve. These tests generate an average of 450 MB of log data. Developers use their knowledge and experience to classify the failure, which may take up to two hours for initial analysis. As the number of microservices and teams using them grows, it becomes increasingly challenging and time-consuming for developers to classify these failures manually.

This paper reports the experience of automating the classification of failures using machine learning techniques at Ericsson. The goal of automation is to identify an efficient and cost-effective method to decrease the time required for manual analysis. To achieve this goal, we utilized developers' knowledge and experiences to better understand the workflows and information necessary for classifying the failure. Based on this understanding, we extracted important log

[1]https://www.ericsson.com/en/ran/cloud

ACM ISBN 979-8-4007-0044-6/23/06.
https://doi.org/10.1145/3593434.3593472

data to train a machine-learning model capable of accurately predicting the failure's cause.

2 Study Design

Figure 1 represents the steps followed in this study. In this section, we describe the process followed in each step.

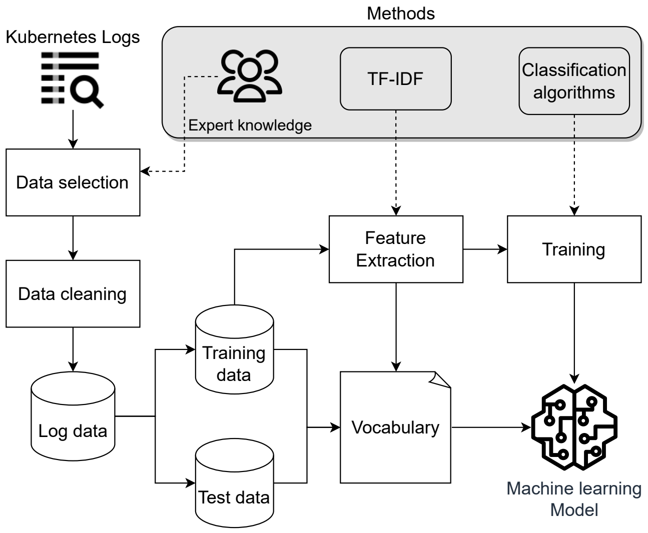

Figure 1. Overall research process.

2.1 Data Selection

As depicted in Figure 1, expert knowledge was utilized to identify the most informative sections of Kubernetes cluster logs, which reduced the overall data processing requirements. Moreover, before automation, it is important to comprehend the manual process of failure classification. The first author, a team member responsible for testing and deploying microservices, noted that each team member had their own approach to analyzing failures. To establish a consensus, we conducted four in-depth interviews to better understand the manual failure classification process. The interviewees had varying levels of experience in identifying reasons for failure, ranging from 1.5 to 4 years. Furthermore, the interviewees have specialized expertise in several areas related to manual failure analysis, such as test environments, end-to-end testing, and coordination with microservice development teams. As a result, the participants have diverse skills and knowledge that were valuable in the failure classification process.

We started the interview by explaining the study objective. We then presented a scenario in which a failure occurred and asked the interviewees to classify the failure using the same steps they would use in a real-world situation. The interviewee is free to use any log as an example. An experienced developer typically examines the sections of the log that are likely to contain the failure reason. This step allowed us to identify the most frequently visited Kubernetes cluster logs.

The interviewees were asked to walk through the different steps they followed, along with the inputs considered and the outputs of each step. Through this process, we identified the crucial log parts for failure classification and how developers identified failure reasons.

2.2 Data Cleaning

In this study, we utilized important data identified from the interview for classification. Following data selection, data cleaning was conducted (Figure 1). We ensured data cleanliness by removing inaccurate, incomplete, or irrelevant data. Human-readable elements (e.g., punctuation, newline characters) were removed to reduce computational costs. We converted all log elements to lowercase letters to avoid ambiguities, and unique elements (e.g., timestamps, IP addresses, line numbers) were removed. Stopwords and stemming methods were not used as they rely on the English dictionary and may not work for non-English words like 'requesthandler-class.'

2.3 Feature Extraction

After cleaning the Kubernetes cluster logs data, the text format must be converted into a numerical form using Feature Extraction (FE) for algorithms to process it. Although word-to-vector (word2vec) is a popular method for feature extraction, it was not considered appropriate due to the wide variation in the size of each log. Instead, we utilized the TF-IDF method, a common feature extraction procedure for sentences and documents. TF-IDF was more suitable for logs since the words related to the error generally occur once or twice in the entire log. This method highlights words distinctive from the text, allowing the extraction of essential data features required for classification. Using TF-IDF, we generate vocabulary, which is used to extract the feature vectors of the log data.

2.4 Classification

Finally, the most crucial step, i.e., log classification, involves using classification algorithms to construct models that can make accurate predictions. This study employs supervised learning, as it is necessary to make predictions for specified classes. Non-parametric algorithms are selected for this study because they do not require prior knowledge and are particularly suitable when working with large datasets [2]. The chosen algorithms for this study include Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Random Forest Classifier, Gradient Boosting Classifier, and Multilayer Perceptron (MLP).

2.5 Training

We proceeded with training the selected classification algorithms using the feature vectors of the training data. We used the `scikit-learn` library to obtain the trained models. The `SVC()` is trained with a linear kernel (SVM). The `RandomForestClassifier()` is trained with 1000 trees. All other function parameters of `KNeighborsClassifier()` and `GradientBoostingClassifier()` are set to default values used by `scikit-learn`. MLP was built using keras sequential model consisting of three dense layers and two Dropout layers. The F1 Score weighted metric is used for evaluation along with the Adam optimizer set to a learning rate of 0.001, and categorical cross-entropy is used as the loss metric.

Following the training of the algorithms, we utilized them to predict class labels for every log in the test data. We collected various metrics, such as accuracy, F1-score, training time, and prediction time, to compare the classification algorithms' performances for predicting failure reasons. We measured training and prediction times to assess computational cost, which can be a better metric for real-world performance than time complexity [3]. Stratified 10-fold cross-validation was employed as the sampling strategy. This approach helped identify the algorithm with the best predictive accuracy and computational cost for a given hardware and software configuration.

3 Expert opinion findings on analyzing failure reasons

During the interviews, developers scrutinized a failed test case and analyzed the Kubernetes cluster logs to classify the failure. Specifically, they searched for logs associated with the microservice under test within the `pods/container` or `pods/describe` directories. If no relevant logs were found, the logs of dependent microservices were consulted. Our analysis revealed that the error or relevant information could usually be found in these directories. Even when it was not, there were clues indicating potential failure type. As a result, we concluded that logs located in `pods/containers` and `pods/describe` directories are crucial in most instances. To further strengthen this conclusion, we also examined the tickets submitted by the developers. We found that the root directories for 43 of the 50 reported log snippets were located in `pods/container` or `pods/describe`, or both directories.

The `pods/containers` and `pods/describe` directories were determined to be important during the interview phase. The logs were pre-processed, and the logs' size was significantly reduced (up to 96.08%) through data selection and cleaning, reducing computational resources.

4 Developing prediction models

The classification algorithms were used to process the training data and generate a classification model, which was then used to predict the test data's class labels. The performance of the algorithms was evaluated based on their accuracy, F1-score, training time, and prediction time.

Table 1. Model accuracy and F1-score for each algorithm

Algorithm	Accuracy	F1-score
SVM	0.6905	0.6580
KNN	0.6576	0.6580
Random Forest	0.7279	0.7106
Gradient Boosting	0.7302	0.7071
MLP	0.7086	0.6911

Table 2. Pairwise P-values for accuracy

	SVM	**KNN**	**Random Forest**	**Gradient Boosting**	**MLP**
SVM	1.0000	0.3517	0.4374	0.2418	0.9000
KNN		1.0000	**0.0037**	**0.0010**	0.1143
Random Forest			1.0000	0.9000	0.7633
Gradient Boosting				1.0000	0.5626
MLP					1.0000

We compared the chosen classification algorithms statistically based on the recommendations by Demšar [1]. We applied the Friedman test to test the null-hypothesis that all algorithms are equivalent. If this hypothesis was rejected, we conducted a post-hoc Nemenyi test to compare the algorithms pairwise.

Table 1 presents the accuracy and F1-scores of each algorithm. The Friedman test was performed to evaluate the statistical significance of the differences between the algorithms, resulting in a p-value of 0.00074 and a Q value of 19.12, confirming significant differences in accuracy. We subsequently conducted a Nemenyi test, and the p-values of the combinations of algorithms are shown in Table 2 with P-values less than α (0.05) in bold. Random Forest and Gradient Boosting were statistically better than KNN.

Similarly, for F1-scores (Table 3), the Friedman test yielded a p-value of 0.00056 and a Q value of 19.76, indicating significant differences. The Nemenyi test showed that Random Forest outperformed SVM and KNN with statistically significant differences. Furthermore, KNN's performance was significantly lower than Gradient Boosting's. Regarding both F1-score and accuracy, Random Forest, Gradient Boosting, and MLP performed similarly.

We perform the Nemenyi test to determine which algorithms are statistically different based on F1-scores. Table 2 represents the p-value of F1-score for each combination of algorithms. The p-value in bold is less than α (0.05).

Finally, the algorithms were evaluated based on their training and prediction times. The average training time in minutes for all ten folds is shown in Table 4. It was observed that Gradient Boosting took a significantly longer time for training when compared to other algorithms. On the other

Table 3. Pairwise P-values for F1-score

	SVM	KNN	Random Forest	Gradient Boosting	MLP
SVM	1.0000	0.9000	**0.0248**	0.1143	0.4374
KNN		1.0000	**0.0022**	**0.0160**	0.1143
Random Forest			1.0000	0.9000	0.6830
Gradient Boosting				1.0000	0.9000
MLP					1.0000

hand, MLP took more than 2x longer for training compared to the Random Forest. Regarding prediction time, all five algorithms finished predictions in less than a second, indicating no significant difference between them in real-world performance. Based on the analysis, it can be concluded that the Random Forest algorithm performed better in terms of both accuracy and training time.

Table 4. Training and prediction time in minutes

Algorithm	Training time	Prediction time
SVM	0.845	0.082
KNN	0.004	0.015
Random Forest	1.702	0.006
Gradient Boosting	110.773	0.002
MLP	4.111	0.005

5 Piloting the prediction models

To test the model in a production environment, we developed an end-to-end prototype. The results of the model were added to a dashboard for developer analysis. We received feedback from developers, who indicated that while the failure classification helps them understand the reason for the failure and where they should look in the logs, they would appreciate additional information beyond the failure reason. As part of future work, we will devise an additional solution to include the most similar log previously reported along with its diagnosis report. Reviewing diagnosis reports of similar logs will help developers quickly understand the cause of failure. This way, developers can precisely look at the specific logs increasing their efficiency and helping them close the cases faster.

In addition, developers expressed concerns about the trustworthiness of the models' predictions due to a lack of understanding of how the model arrived at its conclusions. Further effort is required to improve the models' interpretability and explainability.

Although Random Forest performed better, we need to tune the hyperparameters to truly capture the effectiveness of Random Forest compared to other algorithms.

6 Conclusion

Identifying the reason for a failed CICD test is important to assign the failure to the relevant team for resolution. The automatic identification of failure reasons minimizes the manual effort needed to analyze the failure. In this study, we report experiences of classifying large-scale Kubernetes cluster logs using machine learning classification algorithms: Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Random Forest Classifier, Gradient Boosting Classifier, and Multilayer Perceptron (MLP). Each log generates 450 MB of data on average, which was reduced to 91% by identifying relevant parts of the logs using expert opinion. We used the TF-IDF feature extraction method. Then we trained and evaluated the classification algorithms in terms of accuracy, F1 scores, training, and prediction time.

The results indicated that Random Forest outperforms SVM, KNN, Gradient Boosting, and MLP in terms of accuracy and computational cost. When evaluating the proof-of-concept, the developers expect additional information to strengthen the confidence in the prediction. For example, providing similar logs and diagnosis reports can help the developers to understand how previously failed tests were classified manually and arrived at the same classification predicted by the model. As part of future work, we aim to make the comparison of the algorithms more rigorous by applying hyperparameter tuning and further improving the model's interpretability and explainability.

As a prospect for future research, we intend to enhance the dimensionality of the input data. Currently, we concatenate logs from different services into a single file, creating a two-dimensional input. Our plan is to aggregate logs from different pods within each service, producing a three-dimensional input where each service has its own file. This approach would enable us to leverage deep learning algorithms, such as CNNs, which are proficient in processing 3D data. By doing so, our model would be able to identify the interdependencies among the services, leading to improved results.

Acknowledgments

The Knowledge Foundation supports this work through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden.

References

[1] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[2] Stuart J Russell and Peter Norvig. 2010. Artificial Intelligence A Modern Approach Third Edition.

[3] Author Daniel Lemire. 2021. Big-O notation and real-world performance. Retrieved February 20, 2023 from https://lemire.me/blog/2013/07/11/big-o-notation-and-real-world-performance/

[4] Muhammad Usman, Deepika Badampudi, Chris Smith, and Himansu Nayak. 2022. An Ecosystem for the Large-Scale Reuse of Microservices in a Cloud-Native Context. *IEEE Software* 39, 05 (2022), 68–75.