A Machine Learning Operations (MLOps) Monitoring Model Using BI-LSTM and SARSA Algorithms

Zeinab Shoieb Elgamal, Laila Elfangary, Hanan Fahmy
Department of Information Systems-Faculty of Computers and Artificial Intelligence, Helwan University, Helwan, Egypt[1]

***Abstract*—Machine learning operations (MLOps) achieves faster model development, deliver higher machine learning models quality, and faster deployment cycle. Unfortunately, MLOps is still an uncertain concept with ambiguous research implications. Professionals and academics have focused only on creating machine learning models, rather than using sophisticated machine learning systems in practical situations. Furthermore, the monitoring system must have a comprehensive view over the system interactions. The need for a strong efficient monitoring system increases when it comes to use the multi container services. Therefore, this research provides a new proposed model called Multi Containers Monitoring (MCM) Model, based on multi container service and machine learning approaches which are bidirectional long short-term memory (BI-LSTM) and state-action-reward-state-action (SARSA). The proposed MCM model enables MLOps systems to be scaled and monitored efficiently. The proposed MCM model realizes and interprets the interactions between the containers. The proposed MCM model enhances the performance of the software release and increases the number of software deployments across different types of environments. Moreover, this research proposes four routines for each layer of the proposed MCM model that illustrates how each layer is going to be developed. This research also illustrates how the proposed MCM model achieves improvements ratio in software deployment cycles by using MLOps up to 24.55% and in build duration cycle up to 13%.**

Keywords*—Machine learning; MLOps; monitoring; container; model

I. Introduction

The fast and increasing popularity of machine learning (ML) applications has led to growing attention in Machine Learning Operations (MLOps), that is, the practice of continuous integration and deployment (CI/CD) of ML-enabled systems [1]. Since changes may not affect only the code but also the ML model parameters and the data themselves, the automation of traditional CI/CD needs to be extended to outspread to monitor model retraining in production [1]. ML has become a significant technique to leverage the potential of data and allows businesses to be more innovative, efficient, and sustainable [2] [3] [4] [5]. However, the success of many ML applications in the real world doesn't meet expectations as the ML community has focused extensively on the building of ML models not on building production-ready ML products and providing the necessary coordination of the resulting [2] [6] [7] [8]. Besides that, these applications started to produce and maintain a huge amount of data from their operations. Those new developments require monitoring the operations of applications in real-time [9]. If MLOps model selection and training are not closely and carefully monitored, applications may lose value in the market and organizations might be at risk of losing money, but the worst is to lose their reputation [10] [11] [12].

This research proposes a multi-container monitoring (MCM) model to monitor the communication and all containers' behavior for the software deployment cycles to help in more frequent releases and reduce production issues. Further, discusses the MLOps practice to effectively handle the issue of creating and monitoring effective ML. Furthermore, adopts a broad viewpoint to provide comprehension of the relevant principles, responsibilities, and architectural structures.

This research makes a significant contribution to the software industry by:

- Review all the previous studies in monitoring containerized software.
- Delineate what kind of problems MLOps practices may be best suited to apply, to help in reducing the re-developing and re-deploying.
- Monitor software performance at a finer granularity level.
- Monitor DevOps and MLOps pipelines and system infrastructure behavior.
- Reduce the build time of software systems.
- Improve the deployment rates of existing methods in software systems.

This research applies a new machine learning technique to monitor and learn more ML model features based on different software systems.

The remainder of this research is structured as follows. Section II presents the background. Section III illustrates the necessary definitions and related work in the field. Section IV, presents an overview of the utilized methodology and the proposed MCM model. It also presents the MCM model challenges and limitations. Section V and VI presents model dataset and model configuration. Results and discussion is

given in Section VII and Section VIII respectively. Finally, Section VI, concludes the work with a summary.

II. BACKGROUND

There are a variety of software process models and development methodologies used in software engineering such as waterfall and the agile manifesto. Those methodologies have similar objectives, which are to deliver production-ready software applications [2] [13] [14]. Recently software development teams have moved away from the traditional waterfall methodology to DevOps as the traditional life cycle is not suited for dynamic projects as needed in the ML development process, as shown in Fig. 1.

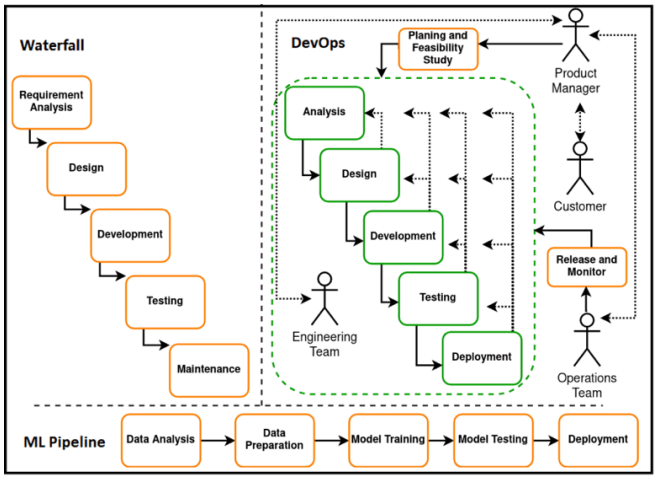

Fig. 1. Waterfall, DevOps SDLC, and the manual ML pipeline [15].

DevOps refers to the modern software deployment model that combines software development (Dev) and IT operations (Ops) [16]. DevOps aims to enable automation, continuous integration, continuous deployment, monitoring, and team collaboration of software applications in fast and small releases [17] [18]. The two primary DevOps practices are Continuous Integration and Continuous Delivery.

Continuous integration (CI) is a software practice that concentrates on automating the creation and integration of code from many developers. To enable quicker development cycles and enhance quality, developers are demanded to merge their code into the primary repository more frequently in this procedure. Version control systems (VCS), automated software development, and testing procedures are the key elements of this practice [15] [19].

Continuous delivery (CD) core purpose is to deliver newly created features to the end user as rapidly as possible by building the software in a way that is constantly in a production-ready state to ensure that code updates might be released on demand fast and safely [15] [20] [21].

Continuous deployment (CDE), which is frequently confused with CD, is a different technique. Continuous deployment is a technique where every software modification is automatically pushed to production. Even so, some businesses have procedures in place for obtaining outside approval before releasing new application's version to users. Thus, continuous delivery is considered necessary in certain circumstances; however, continuous deployment is optional and can be skipped [15] [19] as shown in Fig. 2.

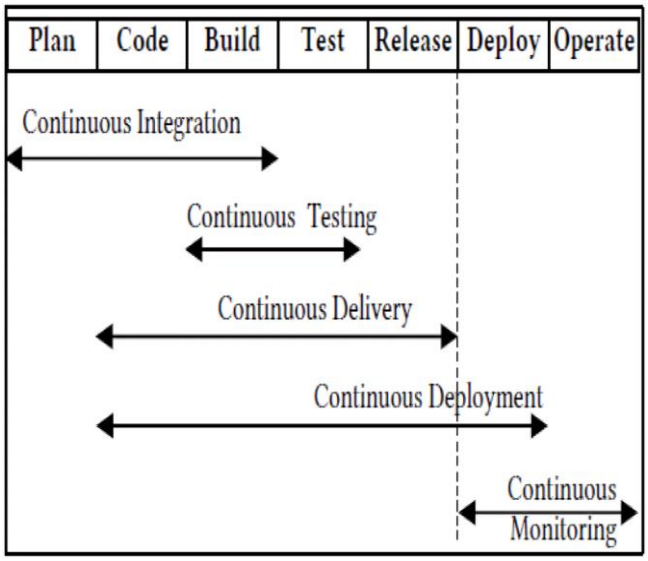

Fig. 2. Deployment pipelines [18].

Continuous monitoring is an automated procedure that uses cloud services to assess a deployed application's operational functionality against business criteria as it is being used [18] [22].

The DevOps pipeline, also known as the CI/CD pipeline, enables greater support for the deployment of applications to the cloud and utilizing a wide range of tools [15] [17] [23].

ML pipeline is defined as an automation of the ML life cycle by minimizing human interaction in routine processes [24].

MLOps refers to the complete vision of best practices and procedures from the design of the training data through the final deployment lifecycle [16] as shown in Fig. 3. MLOps can alternatively be considered as the integration of DevOps with machine learning techniques [25]. By another word, MLOps is the artificial intelligence (AI) equivalent of DevOps [16]. Furthermore, MLOps places a strong emphasis on automation while monitoring each step of the machine learning process, much like DevOps [26].

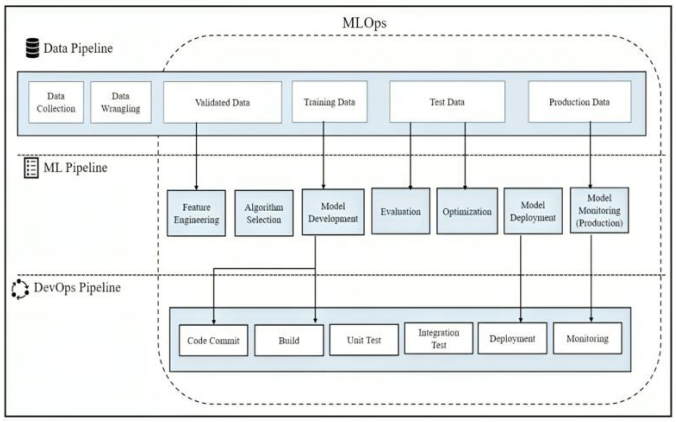

Fig. 3. High-level process perspective of MLOps [24].

MLOps phases are typically related to various roles and concepts, such as containerization and dockerizing [24] [27]. Containerization has become a standard approach for environments, which support on-demand, short-lived execution of computational tasks such as Function-as-a-Service (FaaS) platforms [28] [29].

Running applications in containers enables automatic orchestration and agile DevOps practices, in cloud-native platforms, the design of container and objects in software object-oriented programming (OOP) is similar: each container has a particular duty to carry out effectively [30] [31]. Cloud-native containers are easier to grow horizontally and replace, reuse, and update transparently [31], this has been termed the "Single Concern Principle" [32].

Dockerizing facilitates the hosting and execution of any kind of software applications, platforms, middleware, databases, packaged, in-house, and custom-built software. Moreover, the quicker maturation and reliability of the Docker platform have made it much easier to develop, distribute, deliver, and deploy software [33]. Furthermore, dockerizing offers a simple approach to isolate the network and restrict how much resource the containers can use [34].

There is another design concept that helps in monitoring the ML projects if it had been used during the project development phase which is the "Microservices" design rather than the monolithic design. Microservice architecture (MSA) is suggested to divide single-component applications into numerous loosely linked and independent microservice components [35] [36]. Microservices are applications broken down into their core functionalities. Each function operates as its own "service" inside a container and interacts with other containers across the network [37]. Microservices have several benefits over monolithic applications, such as autonomous update cycles, fine-grained resource control, and high elasticity [37]. The monolithic design is only appropriate for small-scale systems with straightforward internal structures since these monolithic applications adhere to an all-in-one architecture in which all functional modules are created and configured into precisely one deployment unit, namely, one container [38].

III. Related Work

Raúl Minin, et al., introduced a tool named Pangea that generates adequate execution settings for deploying analytic pipelines automatically. These pipelines are broken down into several stages so that each can be executed in the edge, fog, cloud, or on-premises environment which will minimize latency and make the best use of available hardware and software resources. Pangea is focused on achieving three specific goals: (1) creating the required infrastructure if it doesn't already exist; (2) providing it with the components needed to run the pipelines (i.e., configuring each host operating system and software, installing dependencies, and downloading the executable code); and (3) deploying the pipelines [39]. Raúl introduced a complex tool that takes a lot of work to conceptualize and build. Although the first version of the tool is sufficiently developed to demonstrate some of its potential advantages, further use cases and technology and connection compatibility must be added before it can be used in more situations. The web client requires to be improved to assist the management of users, pipelines, and infrastructure since Pangea isn't built to support the description and deployment of analytical pipelines in the training stage. Moreover, Pangea doesn't support monitoring pipelines and infrastructure behavior.

Matteo Testi, et al., provided a literature review on MLOps to illustrate the present difficulties in developing and sustaining an ML system in a production context. The literature review revealed that the utilization of MLOps in the workplace and the application of DevOps principles to machine learning are still under-discussed issues in academia. Furthermore, organizations will need to conduct experimental work to test the ML pipeline as they attempt to apply an ML approach to an end-to-end use case, going through each step and demonstrating what results if certain phases are skipped [40].

Sergio Moreschini, et al., offered a better illustration of MLOps by integrating ML development stages into the established DevOps practices. The research suggested a MLOps pipeline that concentrated on the duality between software engineers and machine learning developers and their roles [41]. Sergio's roadmap increased adoption of ML-based software generated a demand for ML developers who need to perform tasks in parallel to software developers and produced two extra loops for both the ML and software sides.

Pinchen Cui focused on providing security for containerization through secure monitoring of containerized applications to give better simulation of actual application behaviors and greater coverage of attacks with extended feature space [35]. Pinchen's research has not been put through an online evaluation. The elements of the security monitoring target must be enhanced to allow the framework to automatically determine what to monitor in an unsupervised manner. Meanwhile, the framework did not support scaling the dataset with various application architectures, such as multi-container applications, where a service is composed of several containers, and Docker Swarm's distributed monitoring.

Holger Gantikow, et al., suggested integrating containerized environments with rule-based security monitoring. The suitability of the method is investigated for both (1) a variety of undesirable behaviors that may point to abuse and attacks of workloads running inside a container and (2) misconfigurations and attempts to increase privileges and weaken isolation safeguards at the container runtime level [36]. The article does not cover the security monitoring of distributed workloads because shared workloads interact strongly across host borders.

While recent studies cover a variety of specific MLOps topics, a comprehensive conception, generalization, and explanation of ML systems monitor are still lacking. Different interpretations of the phrase "MLOps" may result in misconceptions, which may result in setup errors for the entire ML system. MCM model will work on the monitoring of the development for the multi-container "distributed" ML systems resulting in the ability to improve the software build and deployment cycles in real-world settings.

IV. THE MCM PROPOSED MODEL

The proposed Multi Container Monitoring (MCM) Model consists of four different layers as shown in Fig. 4. The MCM's model layers are the "Development layer, MLOps and container layer, and Monitoring layer with the support of different tools in the Tools and Automation layer".

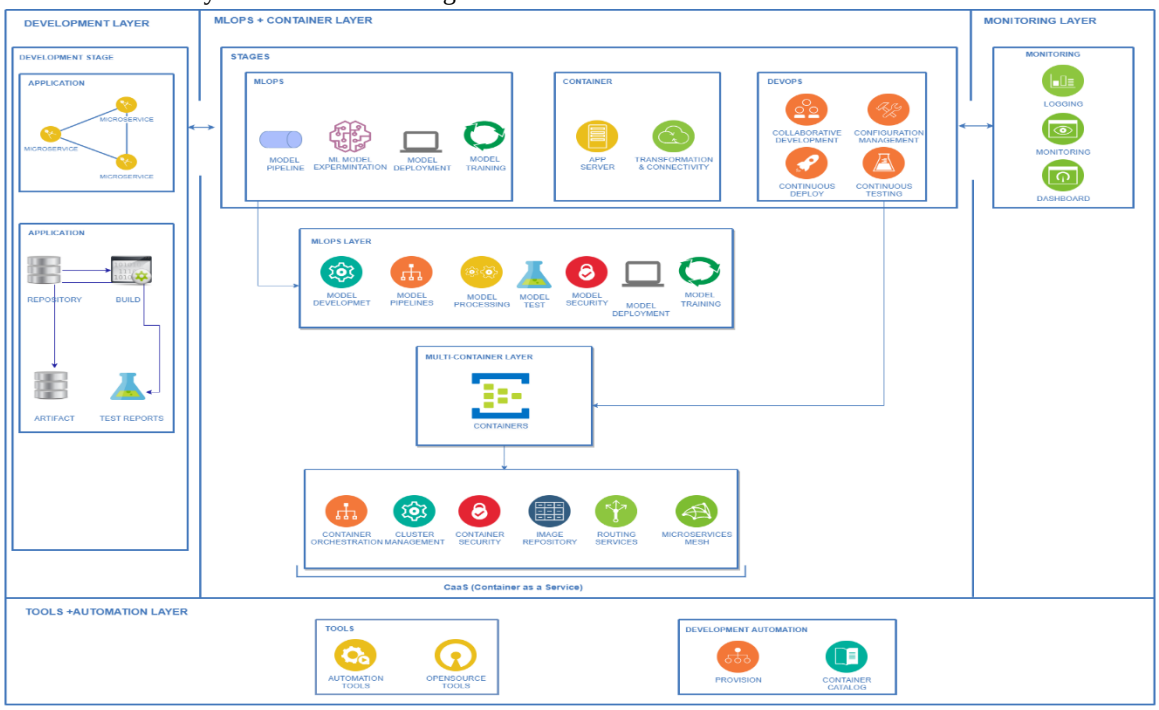

Fig. 4. Multi Containers Monitoring (MCM) model.

The MCM Model starts from the developer's commitment of a specific piece of code and ends with its deployment and monitoring on various environments. The next sub-sections will introduce a detailed description of the MCM's model layers and describe the components of the MCM model.

A. Development Layer

The initial part of the proposed MCM model is the development layer, which is further separated into two main sublayers: the application structure layer (microservices) and the code lifecycles layer. The microservice layer focuses on designing a good structure between each part of the application. The microservice layer seems to be as a pre-requisite for the code lifecycles. The development layer focuses on managing code changes and adopting CI principles to minimize code conflicts.

The code lifecycle layer consists of four steps. The first step verifies that the code is versioned correctly and that a stable version of the code has always been kept up to date in the main repository after the appropriate tagging. Developing a stable version of the code is the second step. The third step involves executing multiple test scenarios, starts after the build step is being successful. At this point, it is confirmed that the code satisfies the necessary functional and needed requirements. When the system's artifact is prepared for release, the last step in the build and validation software process is called "artifact preparation" This involves gathering the output required for the next release and deployment phases.

Routine 1 discusses the high-level routine for the code lifecycle sublayer.

Routine 1: Implement CI/CD Pipeline

Notations: SW // Software, VCS // Version control system, Repo // Repository, VAL // Validation, PR // Pull request, CONF // Confirmation, CI // Continuous integration, PL // Pipeline, TC // Test cases, Pkg // Package, Exec // Execute, CD // Continuous deployment
Input: SW code, Different types of TCs
Output: Deploy the succeeded container images to the target env
Steps:

```
1) Select VCS
2) Repo VAL
3) Code merge
If (PR= Yes) then
    While (review = accepted) do
        For (every merge CONF) do
            1) Push code
            2) Run CI PL
            3) Exec unit TCs
            4) Pkg the build artifact
            5) Exec extra TCs
            6) If (build pkg = accepted) then
                    Pkg the container image
                End
            7) If (CD env>=1) then
                    deploy pkg
                End
        End
    End
```

The procedures of this layer are concentrated on giving code and tests a transparent control structure. Selecting the most appropriate VCS is the initial stage, followed by defining the structure of the repository and its validation guidelines. Then get ready to create the different test cases required to get the CI/CD pipeline started. Finally, store the tested package and make it available for usage in deployment.

B. MLOps and Container Layer

The second layer of the MCM model is MLOps and Container layer. The MLOps and container layer is divided into two primary sublayers: the layer of MLOps comes first, and the layer of containers comes second.

MLOps layer includes the development of the MCM model, the pipelines construction, test preparation, configuring and adding some security gates throughout the MCM model.

Container layer, container as a service (CaaS), is the framework that relies on the idea of multiple containers. Therefore, each container in this layer will go through the same steps. The steps include container orchestration, cluster management, container security, image repository, routing services, and microservices mesh. The container's network configuration, security, and resource allocation are among the processes in this process, along with deployment, auto-scaling, health monitoring, migration, and load balancing.

The high-level routine for the MLOps layer is discussed in Routine 2. Routine 3 presents the container layer.

Routine 2: Implement MLOps Pipeline

Notations: DAQ // Data acquisition, DS // Data source, PL // Pipeline, DA // Data analysis, KPI // Key performance indicators, PERF // Performance, OP // Operational
Input: Model Data
Output: Deploy the succeeded ML model to the target env
Steps:

```
While (Business understanding = true) do
    For each (DAQ) do
        Define DS
        Define PL
        Define deployment env
        Exploration and cleaning
        End
    Model Security
    Model deployment
    For each (Model training) do
        Exploratory DA
        Define feature engineering
        Model training
        End
    For each (Model evaluation) do
        Model KPIs (PERF and OP)
        Model retrain
        End
    Model logging
    End
```

The MLOps layer consists of seven steps and the development of the MLOps pipeline is the main objective at this layer. The identification of the application type is the most crucial step because it serves as the base for the subsequent steps. The second step is understanding the application type which enables the identification and analysis of the received data. The data transformation step is the third step and begins after the data is received. The fourth step is setting up some security validation over the model pipeline and configuring all necessary deployment environments for the pipeline. The fifth step is the MCM's model training phase. The sixth step is the validation step. Depending on the outcomes of the MCM model validation step, the last step at MLOps layer which is the retraining step. The retraining step begins with some modifications to the MCM model and/or data to maximize the benefits. Furthermore, this step depends on the data in the MCM's model log file to keep track of the used data to be able to take the necessary action.

Routine 3: Implement Container Structure

Notations: STRTG // Strategy, CONFIG // Configuration, SP // Security policies, DR // Disaster recovery, HA // High availability, RS // Routing services
Input: Container STRTG
Output: CONFIG container architecture
Steps:

```
Container orchestration
CONFIG the container registry
    For each (Container) do
        Set container SP
        Define PL
        End
    Push container image
```

Along with the MLOps layer, the container layer which contains three steps starts. The first step is establishing the rules for container's interaction with the other containers. This process is known as container management and orchestration. The identification of the deployment strategy for the containers is the second step. The last step is defining all the image registries that depend on the various deployment environments.

C. Monitor Layer

The third layer of the MCM model is monitor layer. This layer includes MCM's model-based event logging, step-by-step monitoring, and the creation of dashboards for convenient monitoring to aid in decision-making on enhancements. The high-level routine for this layer is discussed in Routine 4 and it has been highlighted also in Routine 2 steps 6 and 7.

Routine 4: Monitoring

Notations: INIT: Initialize, PL // Pipeline, ML // Machine learning, REC // Record, CONFIG // Configuration
Input: Monitoring matrices
Output: Monitor PL, container, and ML model
Steps:

```
INIT monitoring
While (Logging = true) do
    For each (PL, container, and ML model) do
```

```
|  |  REC external logs
|  |  REC internal logs
|  | End
| End
CONFIG dashboards
```

In order to validate, enhance, and maintain any abnormal behavior over MCM's model layers, the monitoring layer contains various dashboards to represent multiple perspectives at various stages. The monitor layer saved external logs that record which data the MCM model is applied to, and internal logs that check the inner workings of the ML pipeline and debug problems. The monitoring layer makes it easier to understand and identify any failures, and act quickly to keep everything under control and in good condition. Moreover, the monitoring layer would guarantee a system with fewer development, deployment, and monitoring issues.

D. Tools and Automation Layer

The fourth layer of the MCM model is the Tools and Automation layer. This layer is necessary to support the MCM's model layers with one or more different tools to obtain reliable statistics and facilitate the transition from one step to another. Specifically, for provisioning and container cataloging, this layer uses a set of open-source and development automation tools to help in all MCM's model building and monitoring.

V. MCM MODEL DATASET

The dataset used in the of the proposed MCM model consists of 286 pipelines. The pipelines are divided to continuous integration, continuous testing, and continuous delivery pipelines that use 100 microservices. The pipelines serve over seven million users and run on Windows and Linux operating systems. The dataset contains 122 production environments, seven test and pre-production environments and 158 attributes spread across each layer of the suggested MCM model. Therefore, the size of the dataset is sufficient to validate the proposed MCM model and to investigate possible future enhancements. The dataset used by the MCM model collects and stores unstructured data from a variety of sources, including emails, business papers, source control software, and software engineers' feedback. A sample of the dataset is shown in Fig. 5 and a sample of the dataset and docker file are shown in Fig. 6.

commit ID	369ddcdb	cc3c53cf
number of reviewers on pull request	4	2
source control system type	Git	Git
application type	Web application	API
application language	.Net	Java
number of databases	2	1
number of APIs	0	3
agent status	Idel	Idel
agent version	3.240.1	3.240.1
pipeline triggers	Enable continuous integration	Batch changes while a build is in progress
branch filters	master	dev
pipeline created date	10/22/2018 12:35 AM	02/12/2023 5:13PM
pipeline changed date	03/18/2024 02:55 PM	06/15/2023 02:55 PM
path to publish	$(build.artifactstagingdirectory)	$(build.artifactstagingdirectory)
build status	Success	Canceled
Docker image to deploy	WebImage	JavaAPI
max artifact size	5 GB	5 GB
code analysis type	SonarQube	SonarQube
retention policy (d)	30	15
number of containers	1	1
operating system	Windows	Linux
container app environment	Workload profile	Workload profile
image tag	release	integration

Fig. 5. Example of the dataset representation.

```
# Set the locale
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y
locales

RUN locale-gen en_US.UTF-8 &&\
DEBIAN_FRONTEND=noninteractive dpkg-reconfigure locales

RUN DEBIAN_FRONTEND=noninteractive apt-get install -y fontconfig fontconfig-
config
#libvirt-clients not found?
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y libvirt-bin virtinst
qemu-kvm libvirt-daemon libvirt-daemon-system qemu qemu-user wget shellcheck

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get -y upgrade

RUN modprobe kvm
RUN lsmod

EXPOSE 16509
#This didn't work?
RUN mknod /dev/kvm c 10 232

#Running in a container we can't access the socket?
RUN echo "listen_tls = 0" >> /etc/libvirt/libvirtd.conf; \
echo 'listen_tcp = 1' >> /etc/libvirt/libvirtd.conf; \
echo 'tls_port = "16514"' >> /etc/libvirt/libvirtd.conf; \
echo 'tcp_port = "16509"' >> /etc/libvirt/libvirtd.conf; \
echo 'auth_tcp = "none"' >> /etc/libvirt/libvirtd.conf

#RUN whoami
# root
#
#RUN pwd
# /
#
RUN ls -l /dev/
RUN ls -l /sys/module/
#RUN cat /sys/module/kvm_amd/parameters/nested
#RUN cat /sys/module/kvm_intel/parameters/nested
RUN cat /etc/issue

#setup ability to run kvm/qemu
VOLUME [ "/sys/fs/cgroup" ]
#RUN sed -i "/Service/a ExecStartPost=\/bin\/chmod 666 /dev/kvm"
/lib/systemd/system/libvirtd.service
RUN service libvirt-bin restart
RUN service qemu-kvm restart
RUN service libvirt-guests restart
RUN service virtlogd restart
RUN service --status-all

RUN mkdir tests

ADD docker-test.sh ./

#ADD /bin/chmod u+x docker-test.sh
#/bin/chmod no such file or directory
RUN ["chmod", "+x", "/docker-test.sh"]

#CMD ./docker-test.sh
```

Fig. 6. Example of the docker file.

VI. MCM MODEL CONFIGURATION

Every stage of the MCM's model preparation phase is covered in this section. This process consists of two primary preparation steps: building the application is the first main step, and setting up the ML model is the second.

A. Application Build

The migration phase and the automation phase are the two major stages of the application build process. The process depends on separating the application's functionality into discrete services according to their use cases or modules, and integration the DevOps with MLOps pipelines. Furthermore, the process focuses on using the containerization concept which makes it easier to manage and scale the system and makes it easier to automate the deployment of new features.

- Migration Phase from Monolithic Application to Containerized-Microservices Application

The goal of the migration process is to have a new, well-structured application to benefit from containers and DevOps tools in the next automation phase. Furthermore, the migration process identifies methods that will assist with securing the daily operation of the IT and reduce costs. Five primary stages made up the migration process, some of which had the subsequent sub-stages:

1) Evaluating the legacy application: The components of the MCM model were chosen with the need to address issues such as performance bottlenecks and reduce build and deployment durations.

2) Select a migration method: The Lift-and-Shift and Refactoring migration methods were chosen for the proposed MCM model.

3) Get the Legacy Application Ready for the Migration.

4) Data on migration: the GitOps tools, orchestration platform command-line interface (CLI), or orchestration platform web console can all be used to carry out deployment on the platform after migrating the application data. RedHat-OpenShift, and Kubernetes were chosen as the orchestration platforms for the proposed MCM model to scale resources for microservices and even to simply enable autoscaling. The platform CLI and Azure pipelines were used for the deployment execution.

5) Evaluate and deploy: once the application deployed on an orchestration platform, careful testing is also essential to confirm that the application performs as intended. The application can be launched to production if the testing is successful, which will complete the migration process.

- Automation Phase

This stage demonstrates how the refactored code "microservices" that were previously developed during the migration phase was built automatically based on Docker as shown in Fig. 7. One of the objectives of moving to microservices is to achieve the most benefit of the automated CI/CD pipeline, which enables a very seamless release process. In the case of an issue, the CI/CD pipeline system can initiate a fix or revert to an earlier version. Define checks at each stage using security scans, service level objectives (SLOs), service level indicators (SLIs), and service level agreements (SLAs).

Automate the changes between continuous integration, testing, delivery, and deployment.

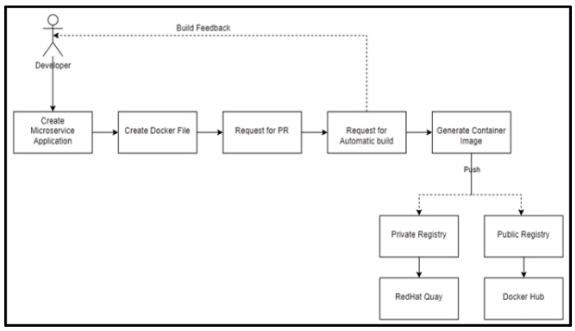

Fig. 7. Creation of container application and push the image to the registry.

The automation procedure was broken down into the following nine major phases:

1) Select a source control (SC) system. Git was the best SC system that met the requirements after several SC system types were compared.

2) Produce Docker files that specify each microservice's container contents.

3) Make a PR request.

4) Build and package every microservice, for Java projects, that have been developed and coded with Spring Boot and for .Net applications, which was created and developed with the .net framework versions 4.8.0, 4.8.1, and 6.

5) Build and produce Docker container images and verify if the image was locally created and accessible.

6) Launch the Docker containers and verify to see if the application within the container is operating properly.

7) Push the image to the registry, at this step, the locally produced image is pushed to the shared registry (RedHat Quay and/ or Docker Hub) using the image ID.

8) Initially, steps 3 through 8, are completed manually. Afterwards, CI/CD pipelines are set up using Azure DevOps. At this stage, each microservice has a CI pipeline to execute the code following an approved PR. The CT pipeline verifies the written code. The CD pipeline manages the deployment procedure for each environment, including integration, QC, security testing, UAT, load testing, packaging, and pre-production. A certain set of stakeholders must be informed and given permission before the created image is deployed on any deployment environment.

9) A new commit was added to the SC and step 3 was restarted to meet the additional requirements.

In conclusion, you can successfully containerize your application by following the previous steps, which include setting up the required tools, building the image, running the containers, and pushing the image to a registry. This will streamline the deployment and management process. Moreover, will encourage consistency and repeatability.

B. The Proposed MCM Model Build

The MCM model was built using a combination of recurrent neural network (RNN) architecture “Bidirectional LSTM (Bi- LSTM)” and a reinforcement learning algorithm “state–action–reward–state–action (SARSA)”. This combination integrates into application CI/CD pipelines and monitoring systems, enabling continuous optimization of container application performance. Further, by analyzing streaming data from container environments, the Bi-LSTM and SARSA can adapt to changing workload patterns and optimize resource usage dynamically. Combining the algorithms results in improved, robust, and shortened build and deployment times. The proposed MCM's model performance was evaluated using backpropagation through time (BPTT). The Adaptive Moment Estimation (Adam) optimizer was selected to improve the performance and accelerate the convergence of Bi-LSTM.

The proposed MCM’s model build process was divided into the seven main stages listed below:

1) Environment representation

a) Define the scope of the MCM’s model environment in terms of the performance and state of the containerized applications.

b) Normalize the metrics to make sure they were appropriate for input into MCM model and to guarantee consistent scaling.

2) Action scope

a) Specify a range of actions that the MCM model agent can accomplish to maximize container performance.

b) Discretization of the actions into a finite set.

3) Feedback function: provide the suggested MCM's agent a reward function that gives feedback based on the observed metrics, actions taken, and the obtained performance outcomes. The design of the reward function is broken down into three categories: positive, negative, and delayed feedback. Positive feedback rewards actions that improve container performance, like shorter build and deploy times, higher throughput, more efficient use of resources, and/or faster response times. Negative feedback penalizes actions that lead to inefficiency or performance degradation, like excessive resource consumption and downtime.

4) MCM's model agent: The proposed MCM's model agent depends on the SARSA's agent as shown in Fig. 8, to pick actions based on the current state and the observed feedback. MCM model learns an action-value function M (s, a) that estimates the expected cumulative feedback of taking an action (a) in the specified state (s) then a balanced exploration (random actions) and exploitation (best-known actions) strategy was combined with temporal difference learning in the SARSA algorithm to update the M-values.

5) Integrated MCM's model architecture: the MCM's model architecture is based on The Bi-LSTM design which consist of some LSTM layers, several hidden units per layer, and the size of input and output. System configurations, workload patterns, historical performance data, and other aspects were arranged into sequences and time-series representations to create the appropriately structured input data. The MCM model uses the input data to learn temporal patterns and anticipate future states. The SARSA agent then uses these predictions to make decisions on how best to optimize container performance and deployment durations. The schematic Bi-LSTM is shown in Fig. 9 and MCM's model integration architecture is shown in Fig. 10.

6) MCM model training: to ensure that gradients and errors are transmitted appropriately throughout time and to update the model weights while taking into consideration the sequential nature of the data, the MCM model was trained using BPTT. Further, Adam's optimizer was selected to modify the Bi-LSTM weights during training to minimize the loss function. Fig. 11 shows the flowchart for the training loop.

7) Evaluation and deployment

a) A validation dataset and container assessment measures, such as response times, throughput, resource utilization, and build/deploy durations, were utilized to assess the MCM model algorithm's performance.

b) Embed the trained MCM's model algorithm in the CI/CD pipelines and containerized application deployment pipelines to automate the entire deployment process and guarantee a smooth integration with the current DevOps workflows.

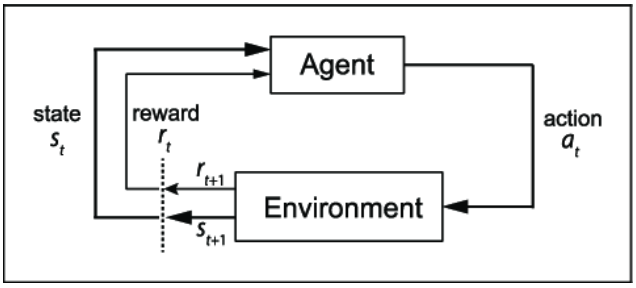

Fig. 8. MCM's model agent environment interaction.

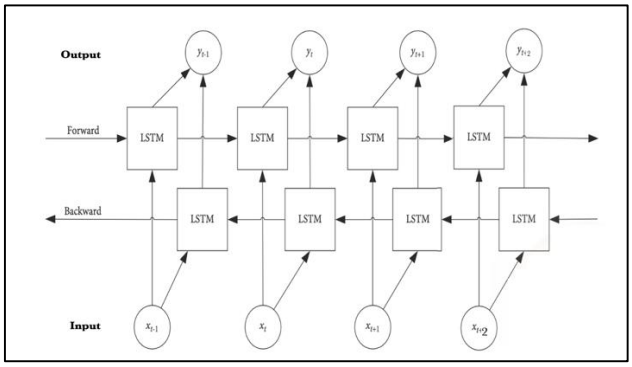

Fig. 9. Schematic Bi-LSTM's architecture.

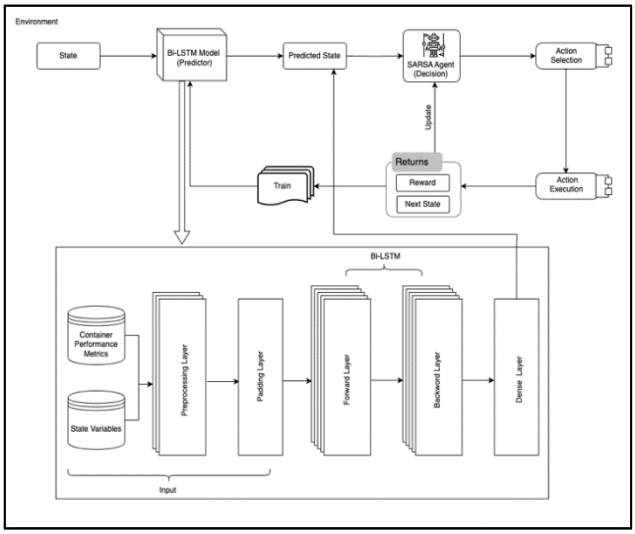

Fig. 10. Integrated MCM's model architecture

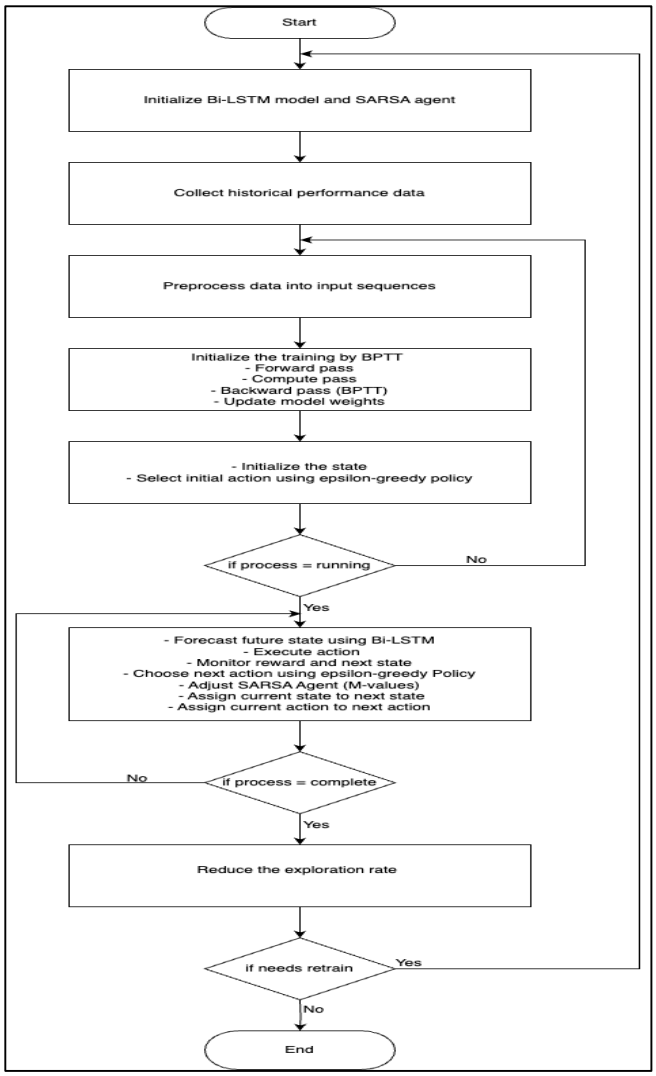

Fig. 11. MCM's model training flowchart.

VII. MCM Model Results

The outcomes of a successful progress visualization during MCM model training to improve the deployment time are presented in Fig. 12. The improvements in build and deployment durations are shown in Fig. 13.

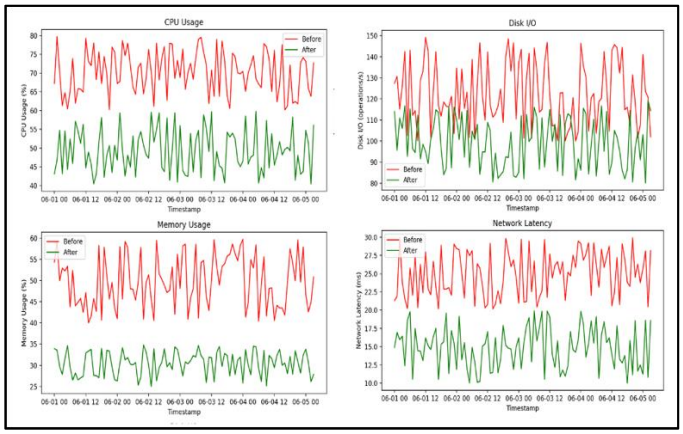

Fig. 12. MCM's model resource usage optimization.

Fig. 12 demonstrates optimization of the resource usage of the containers. It can be noticed that the containers enhancements made by the MCM model compared to the previously employed methodologies for CUP usage is up to 38.25%, for disk I/O is up to 39.20%, for memory usage is up to 50.77% and for network latency is up to 58.37%.

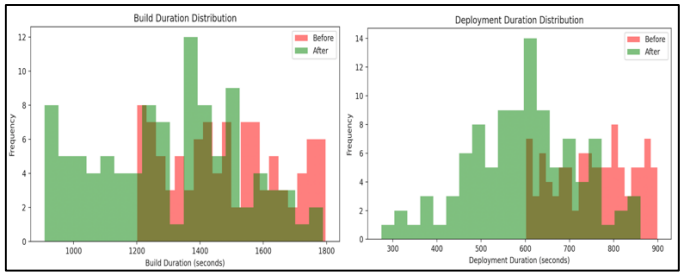

Fig. 13. MCM's model build and deployment duration.

Fig. 13 observes the improvements in seconds of the build and deployment frequencies across the different environments, which are for a build duration up to 13% and for a deployment duration up to 24.55%.

VIII. MCM Model Discussion

The literature research revealed several barriers to the use of MLOps. These unresolved concerns fall into three categories: organizational, ML system, and operational.

A. Organizational Challenges

A common issue in organizational settings is the mentality and culture of data science practice [31]. The study's findings indicate that to effectively develop, implement, and monitor machine learning systems, there require a culture shift away from model-driven machine learning and toward a discipline that is system-oriented. This can be accomplished by placing more emphasis on the data-related activities that take place before the creation of the ML model. Furthermore, when designing ML products, roles involved in these activities should have a system-focused view and must therefore be a group process as this is challenging since teams usually operate in silos rather than collaborative environments also the specialized terminologies and varying degrees of knowledge further complicate communication. Moreover, MLOps demands a wide range of skills and specialized roles. Because there aren't enough highly qualified professionals to fill these positions, particularly in the fields of architects, data engineers, ML engineers, and DevOps engineers [32] [33] [34]. As MLOps is often not included in data science courses, this is relevant to the training that future workers will need [35].

B. ML System Challenges

It might be difficult to develop MLOps systems to accommodate changing demands, especially when it comes to the ML training and monitoring procedures [35]. This is a result of potentially massive and unpredictable data [36], which makes it challenging to correctly forecast the necessary infrastructure resources (CPU, RAM, and GPU), and calls a high level of flexibility when it comes to the scalability of the containers [35] [37].

C. Operational Challenges

ML is challenging to execute manually because of the numerous software and hardware stacks and their interrelationships. Thus, reliable automation is required to produce numerous artifacts, which require solid governance [35] [38] [39] [32] [40] [41]. Versioning of the data, model, and code is also necessary to guarantee reliability and reproducibility [32] [2]. Finally, because there are so many parties and components involved, it can be difficult to handle a possible support request (for example, by identifying the root cause). Moreover, failures might result from a combination of ML infrastructure and software [37] [42], making it essential to monitor each phase and collect as much data as possible to aid in making timely decisions.

Therefore, this research proposed a model (MCM) which depends on the multi-container architecture and microservices principles that applied to solve the mentioned barriers by building and deploying the stages of the application development lifecycle. The MCM reduced the requirement for re-developing and re-deploying software applications while also improving the performance of software releases. The MCM model enhances the build duration cycles and software deployment cycles ratio by employing MLOps.

IX. Conclusions and Future Work

More machine learning systems than ever before are being developed as a result of the growing demand to innovate. Higher monitoring and analysis skills are required for ML models. However, only a few of these proofs of concept move forward to deployment to production. Furthermore, in the real world, data scientists are still managing ML operations largely manually. These issues are addressed by the Machine Learning Operations (MLOps) paradigm. Moreover, according to the linked publications, there are no studies that concentrate on monitoring MLOps applications, especially those that rely on multi-container and microservices design. Therefore, this research proposed a model (MCM) which depends on the multi-container architecture and microservices principles that applied to the build and deploy stages of the application development lifecycle. The developed MCM model used to increase the number of software deployments across a variety of environments. Further the proposed MCM improved the software release performance and decreased the need for re-developing and re-deploying software applications. By utilizing MLOps, the suggested MCM model improves the software deployment cycles ratio by up to 24.55% and build duration cycles by up to 13%. This was useful in directing different IT teams towards the areas of monitoring ML model's features by using MLOps. Furthermore, the research recommended four routines for each layer of the suggested MCM model, described how each layer will be developed. As future work, more experimental work is also needed to assess the MLOps pipelines and see how they might affect the overall software development cycle. The MCM model needs to be implemented on different data sets and monitor its efficiency. More experiments to compare the performance of the MCM model algorithm against baseline approaches or alternative optimization strategies are needed.

References

[1] F. Calefato, F. Lanubile, and L. Quaranta, A Preliminary Investigation of MLOps Practices in GitHub, vol. 1, no. 1. Association for Computing Machinery, 2022. doi: 10.1145/3544902.3546636.

[2] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine Learning Operations (MLOps): Overview, Definition, and Architecture," 2022, [Online]. Available: http://arxiv.org/abs/2205.02302.

[3] S. Makinen, H. Skogstrom, E. Laaksonen, and T. Mikkonen, "Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?," Proc. - 2021 IEEE/ACM 1st Work. AI Eng. - Softw. Eng. AI, WAIN 2021, pp. 109–112, 2021, doi: 10.1109/WAIN52551.2021.00024.

[4] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, "Applying devops practices of continuous automation for machine learning," Inf., vol. 11, no. 7, pp. 1–15, 2020, doi: 10.3390/info11070363.

[5] Y. Liu, Z. Ling, B. Huo, B. Wang, T. Chen, and E. Mouine, "Building A Platform for Machine Learning Operations from Open Source Frameworks," IFAC-PapersOnLine, vol. 53, no. 5, pp. 704–709, 2020, doi: 10.1016/j.ifacol.2021.04.161.

[6] L. Baier and S. Seebacher, "Challenges in the Deployment and," 27th Eur. Conf. Inf. Syst., no. May, pp. 1–15, 2019, [Online]. Available: https://aisel.aisnet.org/ecis2019_rp/163/

[7] D. A. Tamburri, "Sustainable MLOps: Trends and Challenges," Proc. - 2020 22nd Int. Symp. Symb. Numer. Algorithms Sci. Comput. SYNASC 2020, pp. 17–23, 2020, doi: 10.1109/SYNASC51798.2020.00015.

[8] O. Spjuth, J. Frid, and A. Hellander, "The machine learning life cycle and the cloud: implications for drug discovery," Expert Opin. Drug Discov., vol. 16, no. 9, pp. 1071–1079, 2021, doi: 10.1080/17460441.2021.1932812.

[9] E. Calikus, Self-Monitoring using Joint Human- Machine Learning : Algorithms and Applications, no. 69.

[10] T. Schröder and M. Schulz, "Monitoring machine learning models: a categorization of challenges and methods," Data Sci. Manag., vol. 5, no. 3, pp. 105–116, 2022, doi: 10.1016/j.dsm.2022.07.004.

[11] L. Cardoso Silva et al., "Benchmarking Machine Learning Solutions in Production," Proc. - 19th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2020, no. March, pp. 626–633, 2020, doi: 10.1109/ICMLA51294.2020.00104.

[12] P. Liang et al., "Automating the training and deployment of models in MLOps by integrating systems with machine learning", Proceedings of the 2nd International Conference on Software Engineering and Machine Learning, 2024, doi: 10.54254/2755-2721/67/20240690.

[13] C. Wu, E. Haihong, and M. Song, "An Automatic Artificial Intelligence Training Platform Based on Kubernetes," ACM Int. Conf. Proceeding Ser., pp. 58–62, 2020, doi: 10.1145/3378904.3378921.

[14] B. Karlaš et al., "Building Continuous Integration Services for Machine Learning," Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., no. November, pp. 2407–2415, 2020, doi: 10.1145/3394486.3403290.

[15] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, "Demystifying mlops and presenting a recipe for the selection of open-source tools," Appl. Sci., vol. 11, no. 19, 2021, doi: 10.3390/app11198861.

[16] Y. Liu, "Understanding MLOps : a Review of " Practical Deep Learning at Scale with Understanding MLOps : a Review of ' Practical Deep Learning at Scale with MLFlow ' by Yong Liu," no. July, 2022, doi: 10.13140/RG.2.2.21031.83369.

[17] Z. Shoieb, L. Abdelhamid, M. Abdelfattah, "Enhancing Software Deployment Release Time Using DevOps Pipelines", IJSER, vol.11, no. 3, 2020, ISSN: 2229-5518.

[18] M. Rowse and J. Cohen, "A survey of DevOps in the South African software context," Proc. Annu. Hawaii Int. Conf. Syst. Sci., vol. 2020-Janua, pp. 6785–6794, 2021, doi: 10.24251/hicss.2021.814.

[19] A. Sajid et al., "AI-Driven Continuous Integration and Continuous Deployment in Software Engineering" 2nd International Conference on Disruptive Technologies (ICDT), 2024.

[20] R. Subramanya, S. Sierla, and V. Vyatkin, "From DevOps to MLOps: Overview and Application to Electricity Market Forecasting," Appl. Sci., vol. 12, no. 19, 2022, doi: 10.3390/app12199851.

[21] B. Mayumi, A. Matsui, and D. H. Goya, "Applying DevOps to Machine Learning Processes : A Systematic Mapping," 2019.

[22] T. Zheng et al., "NCCMF: Non-Collaborative Continuous Monitoring Framework for Container-Based Cloud Runtime Status", Computers, Materials & Continua, 2024, doi: 10.32604/cmc.2024.056141

[23] P. Agrawal and N. Rawat, "Devops, A New Approach to Cloud Development Testing," IEEE Int. Conf. Issues Challenges Intell. Comput. Tech. ICICT 2019, 2019, doi: 10.1109/ICICT46931.2019.8977662.

[24] N. Hewage and D. Meedeniya, "Machine Learning Operations: A Survey on MLOps Tool Support," no. February, 2022, doi: 10.48550/arXiv.2202.10169.

[25] S. Alla and S. K. Adari, Beginning MLOps with MLFlow. 2021. doi: 10.1007/978-1-4842-6549-9.

[26] G. Recupito et al., "A Multivocal Literature Review of MLOps Tools and Features," no. July, pp. 84–91, 2023, doi: 10.1109/seaa56994.2022.00021.

[27] L. E. L. B, I. Crnkovic, R. Ellinor, and J. Bosch, "From a Data Science Driven Process to a Continuous Delivery Process for Machine Learning Systems," Proceedings- PROFES- 21st Int. Conf., vol. 1, 2020.

[28] G. E. De Velp, E. Rivière, and R. Sadre, "Understanding the performance of container execution environments," WOC 2020 - Proc. 2020 6th Int. Work. Contain. Technol. Contain. Clouds, Part Middlew. 2020, no. 37, pp. 37–42, 2020, doi: 10.1145/3429885.3429967.

[29] C. Segarra et al., "Serverless Confidential Containers: Challenges and Opportunities" 2024.

[30] B. Burns, "Design patterns for container-based distributed systems".

[31] Z. Shen et al., "X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers," Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS, pp. 121–135, 2019, doi: 10.1145/3297858.3304016.

[32] E. Summary, "PRINCIPLES OF CONTAINER-BASED".

[33] R. Madhumathi, "The Relevance of Container Monitoring Towards Container Intelligence," 2018 9th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2018, pp. 1–5, 2018, doi: 10.1109/ICCCNT.2018.8493766.

[34] P. Liu and J. Guitart, "Performance comparison of multi-container deployment schemes for HPC workloads: an empirical study," J. Supercomput., vol. 77, no. 6, pp. 6273–6312, 2021, doi: 10.1007/s11227-020-03518-1.

[35] J. Brier and lia dwi jayanti, "DevSecOps of Containerization," vol. 21, no. 1, pp. 1–9, 2020, [Online]. Available: http://journal.um-surabaya.ac.id/index.php/JKM/article/view/2203

[36] H. Gantikow, C. Reich, M. Knahl, and N. Clarke, "Rule-Based Security Monitoring of Containerized Environments," Commun. Comput. Inf. Sci., vol. 1218 CCIS, pp. 66–86, 2020, doi: 10.1007/978-3-030-49432-2_4.

[37] A. Mahesar et al., "Efficient microservices offloading for cost optimization in diverse MEC cloud networks". *J Big Data, vol*. 11, no. 123, 2024. https://doi.org/10.1186/s40537-024-00975-w

[38] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," ACM Comput. Surv., vol. 54, no. 10s, pp. 1–35, 2022, doi: 10.1145/3510415.

[39] R. Miñón, J. Diaz-De-arcaya, A. I. Torre-Bastida, and P. Hartlieb, "Pangea: An MLOps Tool for Automatically Generating Infrastructure and Deploying Analytic Pipelines in Edge, Fog and Cloud Layers," Sensors, vol. 22, no. 12, 2022, doi: 10.3390/s22124425.

[40] M. Testi et al., "MLOps: A Taxonomy and a Methodology," IEEE Access, vol. 10, no. June, pp. 63606–63618, 2022, doi: 10.1109/ACCESS.2022.3181730.

[41] S. Moreschini, F. Lomio, D. Hastbacka, and D. Taibi, "MLOps for evolvable AI intensive software systems," Proc. - 2022 IEEE Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2022, no. January, pp. 1293–1294, 2022, doi: 10.1109/SANER53432.2022.00155.

[42] G. Bou Ghantous and A. Q. Gill, Evaluating the DevOps Reference Architecture for Multi-cloud IoT-Applications, vol. 2, no. 2. Springer Singapore, 2021. doi: 10.1007/s42979-021-00519-6.