Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online *Vol 7 Issue 1, January 2025*
(https://fcihib.journals.ekb.eg)

The Impact of using MLOps and DevOps on Container based Applications: A Survey

Zeinab Shoieb Elgamal[1], Laila Elfangary[2], Hanan Fahmy [3]
[1]Information System Dept., Faculty of Computers and Artificial Intelligence, Helwan University, Egypt
zeinab_elgamal@fci.helwan.edu.eg, lailaelfangary@fci.helwan.edu.eg, hanan.fahmy@fci.helwan.edu.eg

***Abstract*— Developing and implementing the machine learning applications as quickly as feasible is the aim of commercial machine learning (ML) initiatives. A lot of machine learning trials, however, fall short of their requirements and expectations because automating and operationalizing the machine learning systems is so challenging. MLOps, a model for machine learning operations, deals with this. There are several components to MLOps, including development culture, collections of concepts, and best practices. The consequences of MLOps for academics and professionals are, however, yet unknown. As a result, this study provides a detailed description of the underlying concepts, elements, as well architecture, to aid in the development of usable software based on ML methods that can make MLOps models simple to monitor, integrate, and scale securely to help save maintenance costs and improve the number of software deployments. The study draws attention to the remaining problems in the area before it is done.**

***Index Terms*— CI/CD, DevOps, MLOps, Container.**

I. Introduction

Machine learning operations (MLOps), the process of deploying and continuously integrating (CI/CD), machine learning-enabled systems, are receiving more attention due to the ML applications' explosive rise in popularity [1]. The process of automation for a typical CI/CD pipelines must be expanded to include monitoring retraining of the model during and after the production deployment because the modifications might alter not just the code but additionally the ML model attributes as well as the data itself [1]. Furthermore, the reusable, modular, and maybe shareable components are necessary for machine learning pipelines to facilitate developer repeatability by separating the deployment machine or environment of the execution time for a specific code, these components should preferably be containerized [2].
ML has become a vital element for unlocking data's capabilities and facilitating more innovative, effective, along with sustainable business practices [3]. Many ML applications, however, have not been as successful as anticipated in the real world. This is not unexpected from a research standpoint because the ML community has concentrated mostly on developing ML models rather than developing ML products that are ready for production and facilitating the essential coordination of the resulting ML systems [3]. In addition, these applications began to generate and store huge quantities of data as a result of their activities. These new advances necessitate real-time application operation monitoring [4]. If MLOps model selection and training are not regularly and consistently observed, The market worth of applications may decrease and organizations may suffer from a financial loss, but in the worst case scenario, they could harm their reputation [5].

The recently developed field of machine learning engineering called "Machine Learning Operations" (MLOps) is discussed in this research in order to effectively handle the issue of creating and monitoring effective ML. The study adopts an all-encompassing viewpoint to explain the related components, principles, responsibilities, and architectures. While recent studies cover a variety of specific MLOps topics, an absence of a comprehensive conception and explanation for the ML systems monitor appears. Various ways to interpret the phrase "MLOps" may result in misconceptions, which may result in setup errors for the entirety of the machine learning application.

The study's main contribution is as follows:

1- Creates a shared understanding of the Container, DevOps and MLOps phrases and associated ideas.
2- Elucidating the significance of MLOps principles, presenting and highlighting the key elements to successfully apply MLOps.
3- Highlights a variety of studies on machine learning-based container orchestration techniques.
4- Provides clear recommendations for MLOps practices and makes it possible to make predictions that are more accurate in real-world environments.
5- The research concludes by discussing the MLOps' potential future directions.

The remaining research is structured as follows. Sect. 2, Illustrates the necessary definitions. Sect. 3, presents the relevant research in the topic. Sect. 4, concludes the work with a short summary.

II. BACKGROUND

An overview of the fundamentals of DevOps, virtual machines, containers, microservices, Machine Learning and MLOps as a crucial stage in the software development life cycle is intended to be provided in this section.

A. DEVOPS

Software engineering has historically employed a variety of software process models and development techniques, including Waterfall and Agile. These approaches share the same goal of producing software applications that are ready for production [3]. The old waterfall methodology has been replaced by DevOps in software development teams recently because the traditional life cycle is insufficient for dynamic projects, which are necessary for the process for building Machine Learning (ML) applications to shorten the time that developers spend engaging in daily tasks [6], thus; ML pipeline is established in collaboration with DevOps with the objective of automating the ML lifecycle. Figure 1. Represents the differences between waterfall and DevOps and define the ML pipeline stages [7].

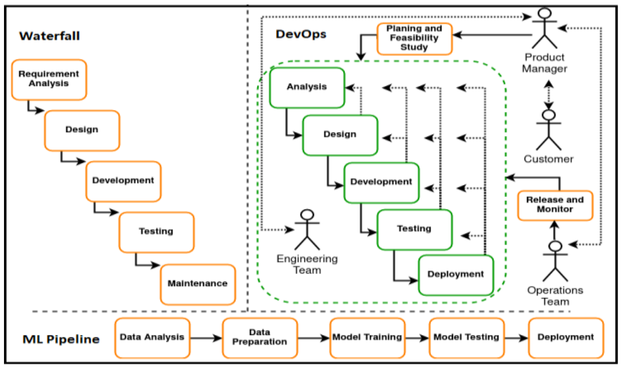

Figure 1. Distinction between the manual machine learning pipeline, DevOps, and Waterfall software development life cycles [7].

The recent software deployment methodology named as "**DevOps**" blends IT operations (Ops) with software development (Dev) [8]. Software applications can be automated, continuously integrated, continuously deployed, monitored, and teamwork grows with the aid of DevOps [9] [10]. The two primary DevOps techniques are Continuous Integration and Continuous Delivery in addition to the Continuous Testing, so DevOps pipeline, commonly referred to the CI/CD pipeline [7] [9] [11].

Software development with an emphasis on automating the development and combining of code from several developers is known as **Continuous Integration**, or **CI**. In this method, developers are asked to merge their changes on code more frequently to the main repository in order to shorten development cycles and enhance quality. This procedure's primary elements include automated software development, testing procedures, and version control systems [7] [12]. The main objective of **Continuous Delivery (CD)**, which seeks to deliver newly created features to the end user in the most expedient manner, is to build the software in a way that is continuously in a state suitable for production so that code modifications may be provided on request swiftly and safely [7] [13] [14] .

Continuous Testing, or **CT**, is the practice of running automated tests as component of the pipeline for software delivery with the goal of receiving prompt feedback on the business concerns associated with an upcoming software release [15].

CDE, or **Continuous Deployment**, is a unique method. A continuous deployment approach is used to automatically release each software change to production. A majority of companies have procedures in place that require external clearance before sharing information with users. While continuous deployment is optional and can be avoided, continuous delivery is often deemed mandatory [7] [12].

Continuous Monitoring is an automated procedure to evaluate a deployed product's performance in real time against business requirements [10] [16]. Figure 2. Represents the different between each software delivery pipelines [10].

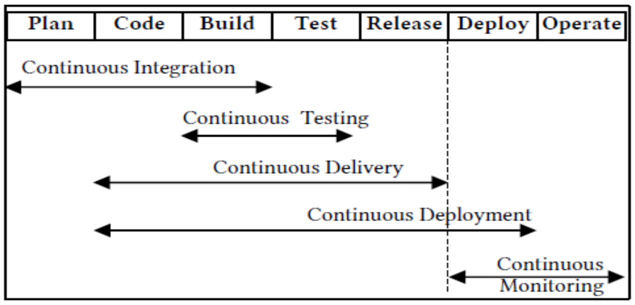

Figure 2. Software Delivery pipelines [10].

There are numerous problems associated with the DevOps methodology that the organization must overcome during the DevOps transformation phase. Table 1, lists these challenges.

TABLE I
DEVOPS CHALLENGES [10]

Qualified Participants	It can be difficult to integrate DevOps capable individuals into a team.
Management	Getting management support for DevOps procedures is difficult.
Build process	The DevOps build process is becoming more sophisticated.
Legacy system	Making DevOps and legacy systems work together.
Security	Realizing pipeline security in the DevOps development process.
Expert guidance	Locating skilled professionals to assist with DevOps operations.
Persistent cultural practices	Shifting established corporate culture to facilitate the implementation of DevOps.
Tool utilization	Integrate with a huge number of tools.
Process and mentality adaptation	Modifying some organization procedures and mentality in order to implement DevOps successfully.
Cooperation	Establishing cooperation between Operations and Development.
Monitoring	The DevOps monitoring process is becoming more essential at every step on the pipeline.
Expenses	Determining the hidden expenses of adopting DevOps

B. Containers and DevOps

Before Docker containers were invented, large, technologically-dependent businesses like Walmart and Chase Bank employed servers and added a lot of them, which resulted in over-allocation, to manage the growing volume of customer requests. The drawback of over-allocating servers was that it was very costly and that if it didn't scale effectively, the servers would burn out and the company would fall apart. Then, a concept known as **virtualization** emerged, which made it possible to run multiple operating systems on the same host. This was revolutionary for industries because it allowed any application to be run in isolation on the same server and infrastructure. It was like having a completely different computer running on the same machine [17].

Containerization is a modern paradigm that is characterized as an operating system (OS) virtualization technique that permits programs to run in distinct user spaces while sharing a single OS. Simply, containers are a considerably fewer resource and time intensive virtualization principle [17] [18] [19]. For systems that enable the temporary, on demand execution of computing processes, like Function As A Service (FaaS) platform, Containerization is a popular technique [20] [21]. For cloud-native platforms, container design is similar to object design in object-oriented (OO) software systems in that it allows automated orchestration and agile DevOps methods: Every container will be assigned a particular task to complete effectively [22] [23]. The "Single Concern Principle" has been used to describe how cloud-native containers can grow dynamically and replace, re-use, and update seamlessly by focusing on a single issue, which is equivalent to the single responsibility concept in OO-languages [23] [24].

Dockerizing aims to fully execute the DevOps paradigm [25]. Docker enables the hosting and execution of any kind of software, including packaged, in-house, custom-built, databases, platforms, middleware, and applications. Furthermore, it is now much easier to design, publish, deliver, and deploy software quickly and effectively [26]. Moreover, dockerizing provides an easy way to limit the amount of resources that the containers can utilize [27], and ensures that application stacks running on the same Docker layer are isolated from one another [28].
Organizing the configuration interface and optimizing machine setup are two benefits of Docker. As CI/CD establishes the relationship between Docker and DevOps, Docker can be utilized to improve the company's DevOps operations and maintains consistency between the production and testing environments [17]. The development process is improved by the usage of DevOps and container deployment together [29]. With containers, developers may specify exactly how to build a software environment that works for a certain piece of code. This covers environment variables, system libraries, third-party libraries, the basic operating system type, as well as the process of compiling a software application [29].
Figure 3 illustrates how Docker containers can operate on both tiny and large devices, with an average speed difference of 26x between them and virtual machines (VMs) [18] [30].

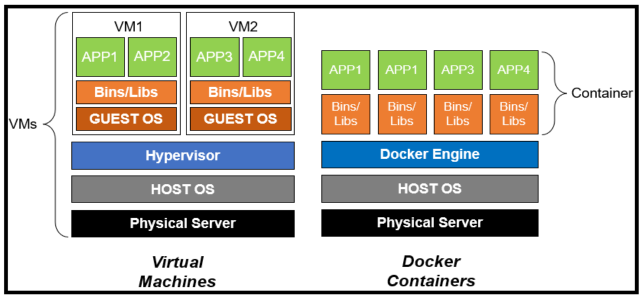

Figure 3. Virtual machines vs. Containers [18]

Table 2, provides an overview of the advantages of using containers for software deployment compared to virtual machines.

TABLE II

Advantages of Virtual Machines vs Containers [19]

Comparative element	VM	Container
Operating System	Guest OS	Shared-Host OS
Starting up time	Take a while to boot up	Extremely quickly
Standardization	OS-specific	Depending on the application
Movability	Less portable	More portable
Server's Demand	Additional servers are required.	Fewer servers are needed.
Security	Any operating system that operates in a VM has the ability to secure the date of the user. Security is also dependent on the hypervisor.	Because the kernel is shared by all of the apps, security may suffer. Therefore, it doesn't have any security measures.
Low redundancy	More redundant information	Few overlapping details
Hardware access	Absence of direct hardware access	Direct hardware access
Distribution of Resources	Significant number of resources needed	Less resources required
Memory requirement	High memory needs	Less memory
Sharing files and library	Isn't possible	Is possible by using Linux commands

Using a virtual machine or a container is an option available to DevOps experts throughout the software development process' deployment phase. Virtual machines are more dependable and safer since they enable total resource isolation and high adaptability. They need more resources than containers, though, and are heavier. On the other hand, applications and their dependencies are the main emphasis of containers. They are simple to deploy and manage and just require a single kernel to operate. It is also possible to deploy numerous containers on a single host or virtual machine; but, in the event that the host goes down, there is a greater chance that all of the containers would fail as well. Consequently, the decision between virtual machines and containers is based on the particular requirements of the project, including the degree of security, the necessary stability and the availability of resources.

C. MICROSERVICES

Applications that have been divided into their essential functions (or activities) are called **Microservices**. Within a container (like a Docker container), each activity functions as a separate "service" and uses the network to interact with other containers [31].
Compared to conventional, monolithic programs, microservices have a number of benefits, such as great flexibility, extremely good resource management, and independent updating cycles [31]. Since these monolithic applications adhere to the paradigm of "all in one" in which every functional components are produced followed by configuration to precisely single deployment entity, specifically, single container, the monolithic design is only appropriate for tiny scale systems with straightforward inner structures [32]. If "Microservices" architecture, as opposed to monolithic design, had been employed during the project development phase, it would have assisted in monitoring ML initiatives. It is recommended to divide single-module systems across several connected and independent microservice modules using **Microservice Architecture** (MSA) [33] [34].

D. MACHINE LEARNING

Among the newest and most widely adopted technologies of the fourth wave of industrialization is machine learning, or ML. It gives systems the capacity to automatically grow and learn from the lessons learned out of having to be manually designed [35]. Building computer programs with data-driven learning capabilities is the aim of ML. The knowledge that has been acquired by a machine learning system must be stored in a knowledge representation structure known as an inductive hypothesis, which is usually in the form of a model. Figure 4, presents a general approach to machine learning [36].

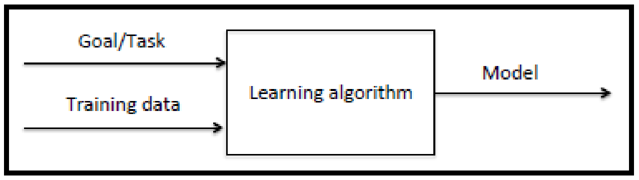

Figure 4. ML Approach [36].

Machine learning models are frequently applied to voice and facial recognition applications. But creating and utilizing machine learning (ML) techniques is more intricate and challenging than assessing and validating models using normal methodologies [37].

There are four phases for the machine learning deployment, and there are various factors including law, ethics, security, and end user trust that can affect one or more of these phases. These phases include: Model verification, which includes formal and test-based verification, requirements encoding; Model learning, which includes selection and training the model and hyperparameter selection; Model deployment, which includes monitoring, updating, and integrating; and data management, which includes gathering, preparing, expanding, and interpreting data. The difficulties and concerns for every phase at each step during ML deployment are shown in Table 3 [38].

TABLE III

ML DEPLOYMENT CONSIDERATIONS AND ISSUES [38]

Deployment Stage	Deployment Step	Considerations and Issues
Data Management	Data collection	Data finding.
	Data preprocessing	Data fragmentation. Data cleansing.
	Data expansion	Classifying the substantial amounts of data. Expertise availability. Inadequate data with a significant variance.
	Data interpretation	Profiles of data.
Model learning	Model selection	Model challenges. Limited resources. Model's interpretability.
	Training	Computational expenses. Environmental effect. Privacy conscious instruction.
	Hyper-parameter selection	Resource needs. Unknown search capability. Hardware enhancement.
Model verification	Requirements encoding	Performance indicators. Business driven indicators.
	Formal verification	Standard structures.
	Test-based verification	Testing by simulation. Data validation procedures.
Model deployment	Integration	Operational support. Reusing models and code. Software engineering anti-patterns. Varied team dynamics.
	Monitoring	Continuous feedback. Identification of outliers. Tools for custom design.
	Updating	Ideas drifting. Continuous delivery.
Intersecting elements	Ethics	Amplification of opinions. Accountability and equity. Authorship. Decision making.
	Law	National laws. Following existing rules. Concentrate on technical fix.
	End users' trust	End user participation. User experience. Describe ability record.
	Security	Data tainting. Model stealing. Model reversal.

E. CONTAINERS AND MLOPS

DevOps is expanded into the machine learning domain by MLOps [39]. MLOps relates to the complete set of best practices and procedures from designing the training data to the end deployment lifecycle [8]. To put it simple, MLOps is DevOps's AI equivalent [8]. Figure 5. Shows the integration of MLOps [6] [37] and Figure 6. Shows An overview of the MLOps Flow [37].

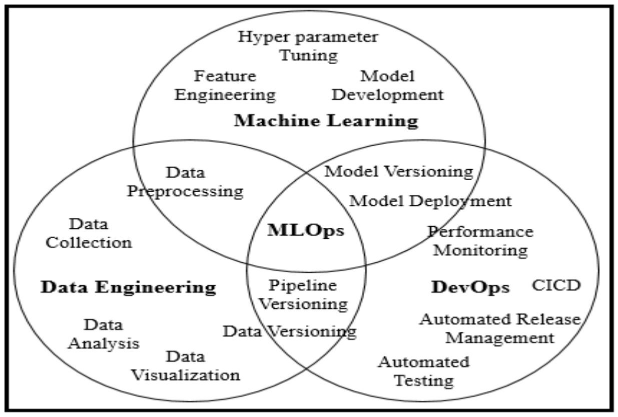

Figure 5. MLOps Integration [6][37].

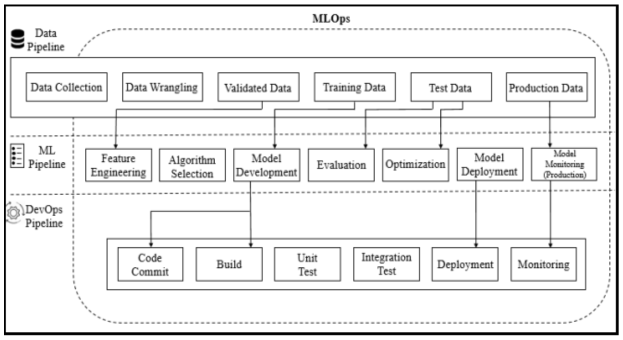

Figure 6. Overview of the MLOps Flow [37].

Like DevOps, MLOps sets a major emphasis on automation while maintaining focus on each step of the ML pipeline through the use of independent, modular components designed to prevent implementation problems and automatically optimize models that may be utilized by other configurations [40], by involve containerization and dockerizing in MLOps phases [6]. Figure 7. Shows the phases of MLOps [37].

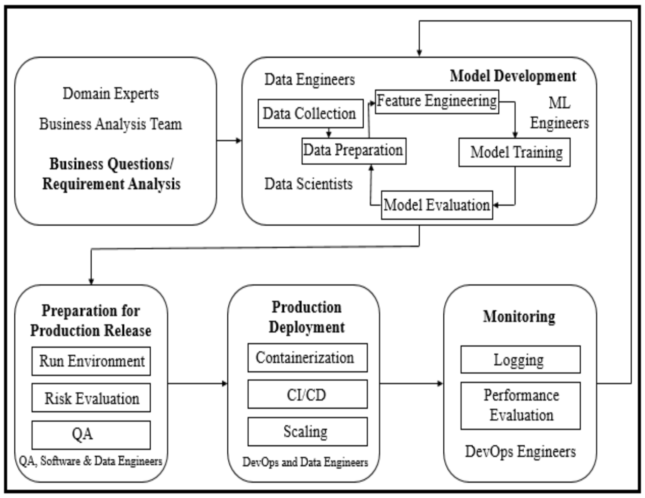

Figure 7. MLOps Phases [37].

MLOps has certain implementation issues. Distinct operational, machine learning system, and organizational challenges have been identified as these outstanding issues.

1) Distinct Operational Challenges

Because of the several stacks of hardware and software components and their interdependencies, machine learning is hard to operate manually. For this reason, reliable automation is needed [41] [42]. Furthermore, skill retraining is required because to an ongoing entry of new data. That being said, this recurring task calls for a high degree of automation [43]. Because of these recurring processes produce a large number of artworks, efficient governance is required [44] [45] [46]. In addition, versioning the code, model, and data is necessary to guarantee reproducibility and reliability [44] [3]. It may be difficult to handle a possible support request due to the intricacy of the situation (e.g., by identifying the real cause) [47], Therefore, in order to support in making informed judgements, it is imperative to monitor each step and collect as much information as possible.

2) Machine Learning System Challenges

Designing MLOps systems for demand fluctuations may be challenging, particularly in terms of ML training and monitoring protocols [41]. This may be the result of extensive and irregular data [48], thus, it is challenging to forecast with precision when the essential infrastructure components (CPU, RAM, and GPU) will be required, along with the capacity of the containers to scale requires a high degree of flexibility [41] [47].

3) Organizational Challenges

In organizational environments, data science practice mentality and culture are often problematic [49]. The study's conclusions imply that a shift in mindset distant from model-driven ML and towards the system-oriented field is necessary to effectively develop, implement, and operate machine learning systems. This can be accomplished by focusing more on the data-related procedures that take place before the machine learning model is developed. When developing ML solutions, roles specifically involved in these tasks should have a system-focused view. MLOps need to possess a wide range of specialized skills and duties. primarily in the fields of architecture, data engineering, machine learning, and DevOps engineering, as there aren't enough highly qualified professionals to occupy these roles [44] [50] [51]. MLOps is often left out of data science classes, therefore this is significant to the kind of training that workers of the future will require [41]. Apart from creating models, students also need to become proficient in the elements and methods required to build machine learning systems.
MLOps must therefore take a cooperative approach. This is difficult since teams typically work in conditions that are more solitary than collaborative and communication is made much more difficult by specialized terminologies and varying knowledge levels.

III. Strategies and Approaches

The methodologies and approaches used by recent studies to handle the subjects of containers and MLOps—with an emphasis primarily on the monitoring phase—will be the subject of the following section. Additionally, researches using MLOps algorithms and the classification of the majority ML methods employed in orchestration of containers domain will be presented in this part, demonstrating their efficacy in application analysis and monitoring.

A. Strategies

The results of the reviewed studies show that there are four primary categories for monitoring the containerized environments: Technical Characteristics, Applicability, Effect and Evaluation.

There are several subcategories within each of these main categories that offer more detailed information about the monitoring methods. The strategies for monitoring containerized environments categorized by category are displayed in Table 4.

TABLE IV

Monitoring Strategies for Containerized Environment.

Category	Sub- Category	Summary/Example
Technical Characteristics	Monitored object	Containerized apps, container engine, and/or host OS.
	Intrusiveness	Internal, external or both.
	Required resources	Hardware, software or both.
	Detection strategy	Anomaly or misuse based.
	Analysis strategy	ML and rule-based, statistical, rule-driven analysis, filtration and confirmation, and remote verification.
	Analysis duration	Instantaneous or offline.
	Analysis parameters	Configuration files, an app, system invocations, or traffic or sensor data, or.
	Measures	Resource usage for CPU, memory or network, communications with the operating system.
	Analysis process	Detailed steps for analysis.
	Response	Logging, setting up alarms, or thwarting attacks.
Application	Domain	IOT, cyber–physical, cloud or high-performance computing, general-purpose.
	Software category	Kubernetes, Linux or Docker.
Effect	Specifically aimed threats	Spoofing, manipulation, rejection, data exposure, denial of service, and privilege elevation.
	Directed assaults	Malware, unapproved entry, kernel exploits, and changes of binaries.
	Certain errors	Network latency, memory leaks, and log explosions.
	Pros and cons	Advantage: enhanced verification. Disadvantage: inaccurate anomaly prediction.
Evaluation	-	Metrics for evaluation: recall, accuracy, detection rate, and performance.

1) Technical Characteristics

Explains the essential actions, materials, and analysis findings, as well as the types of analysis and detection that the monitoring approach can apply. The following are ten subcategories within this category.

- Monitored object

The item under method monitoring in the container-based virtualization environment. The containerized apps, the container engine, and/or the host operating system (OS) can all be this.

- Intrusiveness

This describes whether the monitoring of the environment is conducted externally or inside.

- Required resources

The materials needed to run the monitoring method. These resources may consist of hardware, software, or both.

- Detection technique

In order to differentiate among two distinct approaches of incursion detection: anomaly and misuse-based.

- Analysis strategy

This is the approach used by the method for keeping an eye on and evaluating the activities occurring within the virtualized environment. ML-based, rule-based, statistical, rule-driven analysis, filtration and confirmation, and remote verification are some examples of analysis methodologies.

- Analysis time

Indicates whether offline or real-time mode monitoring is being done.

- Analysis input

The resources used as an input by the technique to do the analysis. These resources may include, for example, a series of system calls made in a predetermined duration, the possible app, traffic and sensor data, or configuration information.

- Measures

Measurements that were gathered, such as how frequently the OS was used and how much memory, CPU, and network was used.

- Analysis process

This is a thorough, step-by-step explanation of the analysis process.

- Response

What happens when an abnormality or violation is found by the monitoring mechanism. This can involve things like setting off an alarm, documenting information, or thwarting an attack. Furthermore, there are two types of monitoring: active and passive. The first type is event-driven, meaning it tracks the occurrence of events rather than being based on time. The second type is reliant on periodic status examination polling and is therefore capable of transitory attacks, or attacks that take place in between polling. Active monitoring, however, is subjected to attacks that do not fall inside the defined occurrences and hence evade detection systems.

2) Applicability

Clarifies explains the areas where the monitoring methods may be applied. The following are two subcategories within this category.

- Domain

The area may fall within the categories of IoT, cyber-physical systems, general purpose, cloud or high-performance computing.

- Software category

This indicates the kind of software that can be used with the monitoring technique, such as Kubernetes, Docker, and Linux containers.

3) Effect

Outlines the results and the objective of the monitoring methods. The following are four subcategories within this category.

- Expected threats

The STRIDE term or the security risks that the monitoring method focuses on. Spoofing, manipulation, rejection, data exposure, denial of service, or privilege elevation are some examples of the dangers.

- Expected attacks

These are the specific security attacks that the monitoring method is intended to stop. Malware, unauthorized entry, kernel exploits, traffic congestion on the network as well binary alteration are a few examples of the attack types.

- Expected faults

The errors that the method aims to correct, such as memory leaks, CPU overuse, delay in networks, and log explosions.

- Benefits and drawbacks

An explanation for the monitoring approach's benefits (like improved verification) and drawbacks (like erroneous anomaly prediction).

4) Evaluation

Explains the kind of assessment that is carried out using the monitoring approach; it may involve testing, case studies, or experiments. It also covers the evaluation process, evaluation results, and evaluation metrics like performance.

B. Approaches

VM orchestration has been implemented using machine learning (ML). For instance, various publications have addressed early attempts to auto-configure virtual machines (VMs) using reinforcement learning (RL) methods. Container orchestrators in conventional cloud computing platforms are typically built with heuristic policies that ignore the variety of workload circumstances and demands for quality of service (QoS).

With the goal of automating deployment of application as well enabling flexible setup adjustments during runtime of a wide variety of workload types, orchestration of the container gives cloud service suppliers the authority to ascertain the configuration, deployment, and maintenance procedures for containerized applications in cloud-based computing environments [52] [32]. A ML frame work of container orchestration is shown in Figure 8 [32].

The following is the primary shortcomings of container orchestration.

1)Metrics level

The existing container orchestration approaches mostly focus on assessing infrastructure-level metrics, with little attention paid to application-level metrics and particular QoS needs. When it comes to tasks like task completion times, communication delays, and delays in task deployment, containerized workloads could have more stringent time limitations than standard cloud workloads.

2) Policies

The majority of the approaches are small-scale, static heuristic procedures that are configured offline based on specific workload scenarios. For example, threshold-based auto scaling schemes may only be appropriate for workload management within a pre-specified range. These policies are unable to manage workloads that are extremely dynamic and require programs to be scaled in or out at runtime in accordance with predetermined behavior patterns.

3) Performance

When the system scales up, heuristic techniques may perform significantly worse. Algorithms for bin packing, such as least fit and best fit, are commonly employed to address issues related to resource allocation and task scheduling. On the other hand, these techniques could not function well in large-scale computing clusters when job scheduling delays are considerable.

4) Resources

Typically, co-located apps disregard resource contention and performance interruption. Application performance degradation, increased maintenance expenses along with the breaches of service-level agreements (SLAs) could result from co-located apps competing for shared resources.

5) Dependency structures

The machine learning models are very simple for containerized workload scenarios, although they are only useful for conventional cloud apps; When resource provisioning, the dependencies among the components of a containerized application are not considered. For example, as compared to typical monolithic programs, Microservice apps that are containerized are more lightweight and decentralized. Within an application, various microservice units are connected internally. The dependencies of the majority of other microservices are essential elements of a microservice architecture and are more prone to experience the breaches of service level objectives (SLO) because of increased the requirements for resources as well as communication expenses.

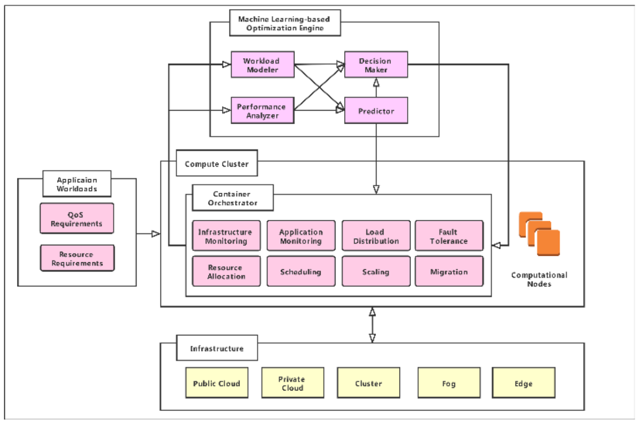

Figure 8. A high-level ML container orchestration framework [32].

As per the studied researches the taxonomy of container orchestration based on machine learning can be classified into five main classifications. Summary of the primary categories together with their corresponding subcategories are displayed in Table 5.

TABLE V
CONTAINER ORCHESTRATION CATEGORIES

Category	Sub-Category	Summary
Application Architecture	Monolithic	Adhere to an all-in-one design in which every functional module is created and set up inside of a single deployment unit, or container.
	Microservices	For various functionalities and corporate goals, every microservice component can be installed, deployed as well run separately.
	Serverless	A paradigm for event-driven applications that performs stateless computing operations is defined by the serverless architecture. Functions are often brief segments of code with particular configurations and constrained execution times that are hosted in function containers and are intended to carry out certain user-defined capabilities.
Infrastructre	Single cloud	Constructed using resources from a single cloud service provider, either public or private, to host and offer services for every application.
	Multi cloud	Incorporates a variety of cloud services, such as private, public, or a combination of both. Numerous factors, including resource configurations, pricing, network latency, and geographic locations, might vary throughout cloud service providers, providing greater options for application deployment optimization.
	Hybrid cloud	Is made up of several cloud components, such as fog, edge, or private clouds. Deploying all of the data and apps to public or private clouds isn't always efficient in this approach, however at fog or edge devices, the hybrid cloud can allow data and apps to be installed and processed.
Optimization objectives	Resource efficiency	Energy and cost efficiency measures for resource utilization at the infrastructure level.
	SLA assurance	Auto scaling is typically used to respond to the constantly shifting dynamic and unpredictable demands by automating application maintenance with SLA assurance.
Behavior Modeling and Prediction	Performance analysis	Identifies the relationship between application-level or infrastructure-level statistics to represent the performance of the application and the system statuses.

	Dependency analysis	Examines the inherent dependencies between containerized application components. Comprehending the distribution of workload and pressure among application components in order to accurately configure resources.
	Anomaly detection	Helps prevent SLA breaches and system crashes by classifying and identifying abnormal system behaviors, such as security risks, performance degradation, workload spikes, instance failure and resource overloading.
	Workload characteristics	By assigning resources precisely in response to incoming workloads, to improve the caliber of resource provisioning choices, a comprehension of workload patterns is necessary.
Resource Provisioning	Scheduling	The overall performance of the application and the efficiency of its resources are directly impacted by the quality of scheduling decisions.
	Scaling	The process of adjusting the size of compute nodes or containerized applications in response to possible workload variations. This process makes sure that the applications are supplied with sufficient resources to reduce the likelihood of SLA breaches.
	Migration	Transferring a task or set of tasks from one node to another.

The studied researches, evaluating the studies that published between 2016 and 2023, discussed a range of algorithms for machine learning which have been used to container orchestration, encompassing tasks such as workload modelling and reinforcement learning decision-making. The growing usage of machine learning solutions aims to integrate many current of machine learning techniques to create a full orchestration pipeline, which includes resource provisioning and multi-dimensional behavior modelling, in order to increase the accuracy of prediction as well as the efficiency of computing. The expansion of different cloud infrastructures and application designs is additionally facilitated by the advancement of machine learning models. The objectives and matrices for each of these algorithms are listed in Tables 6 and 7.

TABLE VI
CONTAINER ORCHESTRATION OBJECTIVES BASED ON ML- APPROACH

Objective	performance analysis	dependency analysis	scaling	scheduling	estimate task arrival rates	forecast resource utilization	forecast resource needs	estimate request arrival rates	anomaly detection	computation offloading	Ref
CNN + BT	√	√	√		√						[53]
K-means	√		√	√		√			√	√	[54]
Model-based RL	√		√	√		√			√	√	[55] [56] [57]
GRU	√		√	√		√			√	√	[58] [59]
LASSO	√		√	√		√			√	√	[57]
DT	√		√	√		√			√	√	[60]
PR	√		√	√		√			√	√	[54]
DRL	√		√	√		√			√	√	[61] [62]
Q-Learning		√	√				√		√		[63]
SARSA	√			√				√	√		[64]
MDP + Q-Learning	√		√	√		√			√	√	[65] [66]
MDP + SARSA	√	√	√								[55]
SVM	√			√				√	√		[57] [67]
Actor-Critic	√		√	√		√			√	√	[62]
SVM + Actor-Critic	√	√	√								[57] [62]
LSTM		√	√				√		√		[68] [59]
BI-LSTM	√			√				√	√		[69]
BI-LSTM + SARSA	√	√	√		√						[70]

Gradient Boosting Regression		√	√				√		√		[71]
ANN	√			√				√	√		[72] [62] [67]
NN						√					[73] [59]
NB	√			√				√	√		[67]
ARIMA						√					[74]
SVR	√			√				√	√		[72]
RF	√			√				√	√		[67]
LR	√			√				√	√		[72]

TABLE VII
CONTAINER ORCHESTRATION MATRIX BASED ON ML- APPROACH

Matrix	resource demands	application latency	propagation timeouts	image's size of the container	power usage	duplicate sizes	time of response	request arrival rate	request type	package loss rate	task completion time	Ref
CNN + BT	√	√										[53]
K-means	√		√	√	√	√	√					[54]
Model-based RL	√		√	√	√	√	√					[55] [56] [57]
GRU	√		√	√	√	√	√					[58] [59]
LASSO	√		√	√	√	√	√					[57]
DT	√		√	√	√	√	√					[60]
PR	√		√	√	√	√	√					[54]
DRL	√		√	√	√	√	√					[61] [62]
Q-Learning	√											[63]
SARSA	√									√	√	[64]
MDP + Q-Learning	√		√	√	√	√	√					[65] [66]
MDP + SARSA	√					√	√	√	√			[55]
SVM	√									√	√	[57] [67]
Actor-Critic	√		√	√	√	√	√					[62]
SVM + Actor-Critic	√					√	√	√	√			[57] [62]
LSTM	√											[68] [59]
BI-LSTM	√									√	√	[69]
BI-LSTM + SARSA	√	√										[70]
Gradient Boosting Regression	√											[71]
ANN	√									√	√	[72] [62] [67]
NN	√											[73] [59]
NB	√									√	√	[67]
ARIMA	√											[74]
SVR	√									√	√	[72]
RF	√									√	√	[67]
LR	√									√	√	[72]

In addition to the earlier investigation, there have been further studies that, without concentrating on the algorithms employed, explored MLOps and DevOps with container-based applications. These studies can be summarized as follows.

A program called Pangea was invented by Raúl Miñón et al. to automatically provide suitable execution settings for the deployment of analytic pipelines. Pipelines for analysis are divided into multiple phases, allowing for the optimal usage of software and hardware resources while minimizing latency, whether they are executed in an on-premises, cloud, edge, or fog environment. Pangea aims to accomplish three distinct objectives: firstly, building the necessary infrastructure in case it doesn't already exist; secondly, supplying it with the elements required to execute the pipelines (such as configuring the executable code, installing dependencies, and setting up every host software and operating system); thirdly, implementing the pipelines [75]. Raúl presented a sophisticated tool that requires extensive planning and development. The tool's initial iteration is developed enough to show off some of its potential benefits, but before it can be applied in additional scenarios, further use cases, technology, and connection compatibility need to be added. Since Pangea is not made to allow the definition and deployment of analytical pipelines at the training step, the web client needs to be upgraded to help with user, pipeline, and infrastructure management. Furthermore, Pangea does not provide infrastructure behavior and pipeline monitoring.

A survey of the literature on MLOps was presented by Matteo Testi et al. to highlight the current challenges associated with creating as well as maintaining a production-level machine learning system. The literature review indicated that there is still a lack of discussion in academics regarding the use of MLOps in the workplace and the integration of DevOps principles with machine learning. Organizations attempting to implement a machine learning approach in a complete use case will also need to conduct experimental work to test the ML pipeline, going through each step and demonstrating what happens if key phases are ignored [33].

Sergio Moreschini et al. provided a more comprehensive example of MLOps by incorporating the stages of ML development into the well-established DevOps procedures. The study proposed an MLOps pipeline that focused on the responsibilities and duality of software engineers and machine learning developers [76]. Sergio's vision accelerated the introduction of machine learning software, creating a requirement for ML developers to work concurrently with software developers and creating two additional loops for the ML and software domains.

The work of Pinchen Cui concentrated on securing containerized applications through secure monitoring. Better application behavior simulation and wider attack coverage using an expanded feature area are required [77]. Pinchen's work has not been subjected to an online assessment. Enhancing the capabilities of the security monitoring target is necessary to allow the framework to decide what needs to be monitored automatically in an unsupervised manner. Furthermore, the framework did not enable the scaling of a dataset featuring various application architectures, including distributed monitoring with Docker Swarm and applications using multi-containers, in which a service is made up of multiple containers.

Rule-based security monitoring should be integrated with containerized environments, according to Holger Gantikow et al. The method's suitability is examined for two things: firstly, a range of undesired behaviors that could indicate misuse or assaults on workloads operating inside of containers; and secondly, incorrect setups as well as efforts to weaken isolation measures and enhance privileges at the level of container runtime [34]. As shared tasks engage in significant interactions beyond host boundaries, security monitoring of distributed workloads is not covered in Holger's work.

IV. Conclusion

More than ever, machine learning systems are being developed and only a tiny percentage of these proofs of concepts, meanwhile, are implemented and put into production. Furthermore, the academic community has concentrated a great deal on developing machine learning models but not nearly as much on applying sophisticated real-world applications of machine learning. In reality, data specialists still do the majority of ML operations manually. The Machine Learning Operations (MLOps) methodology aims to address these problems. To put it simple, ML model combine data and code but MLOps is about combine ML model (Algorithms, Weights and Hyperparameters) and software (Scripts, Libraries, Infrastructure and DevOps). By another word, MLOps integrates with SDLC (Source control repositories, etc.) for code and integrates with DevOps for Automation, Scale and Collaboration.

The machine learning solution's efficiency and effectiveness are determined only by its features and nature. The research took a substantial amount of time and effort before being able to obtain the necessary information to find answers to various opened areas, especially MLOps monitoring and constraints, which leads to believe that the documentation is insufficiently detailed, particularly for those who are unfamiliar with the technology. The goal of this initiative is to clarify MLOps by examining the literature and instruments already in use in order to derive a comprehensive definition of MLOps and its associated ideas and procedures such as containers.

The findings illustrate that even there are many benefits of using containers, there are a few drawbacks that can be difficult for companies whose IT architecture and procedures demand formal approaches and standardization. Because the concept of containers is linked to automation in various fields and DevOps, which means the process automating of generating containers and then deploying them to the platform as a service environment like RedHat OpenShift, Amazon Web Services or IBM BlueMix by extending current architecture modelling standards, this introduce one of the many challenges in maintaining machine learning systems and DevOps as well which is Monitoring.

Moreover, not many new machine learning models have been adopted by the container orchestration in the past few years and the monitoring system for the container service needs to be examined very specifically for every container and its entirety and to understand the relationships that exist between the containers which will leads by its role to the areas of monitoring the ML model by utilizing MLOps to automatically monitor the ML model's characteristics and features and identify what kinds of issues MLOps techniques could be most appropriate for; in order to increase the frequency of software deployments have a big challenge.

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] S. Garg, P. Pundir, G. Rathee, P. K. Gupta, S. Garg, and S. Ahlawat, "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," *Proc. - 2021 IEEE 4th Int. Conf. Artif. Intell. Knowl. Eng. AIKE 2021*, no. Ci, pp. 25–28, 2021, doi: 10.1109/AIKE52691.2021.00010.

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] F. Innlandet and R. Colomo-palacios, "DevSecOps : A Multivocal Literature Review DevSecOps : A Multivocal Literature Review," *Commun. Comput. Inf. Sci.*, no. October, 2017, doi: 10.1007/978-3-319-67383-7.

[13] 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.

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

[15] M. A. Mascheroni and E. Irrazábal, "Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review," *Comput. y Sist.*, vol. 22, no. 3, pp. 1009–1038, 2018, doi: 10.13053/CyS-22-3-2794.

[16] B. Fitzgerald and K. Stol, "Continuous software engineering : A roadmap and agenda," vol. 000, pp. 10–12, 2015.

[17] P. Bellishree and Deepamala N, "A Survey on Docker Container and its Use Cases," *Int. Res. J. Eng. Technol.*, no. July, pp. 2716–2720, 2020.

[18] B. S. Kim, S. H. Lee, Y. R. Lee, Y. H. Park, and J. Jeong, "Design and Implementation of Cloud Docker Application Architecture Based on Machine Learning in Container Management for Smart Manufacturing," *Appl. Sci.*, vol. 12, no. 13, pp. 1–16, 2022, doi: 10.3390/app12136737.

[19] A. K. Yadav, M. L. Garg, and Ritika, *Docker containers versus virtual machine-based virtualization*, vol. 814, no. January. Springer Singapore, 2019. doi: 10.1007/978-981-13-1501-5_12.

[20] 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.

[21] E. Oakes *et al.*, "SOCK : Rapid Task Provisioning with Serverless-Optimized Containers SOCK : Rapid Task Provisioning with Serverless-Optimized Containers," 2018.

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

[23] 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.

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

[25] M. Fokaefs, C. Barna, R. Veleda, M. Litoiu, J. Wigglesworth, and R. Mateescu, "Enabling DevOps for Containerized Data-Intensive Applications: An Exploratory Study," *28th Mod. Artif. Intell. Cogn. Sci. Conf. MAICS 2017*, pp. 189–190, 2017, doi: 10.1145/1235.

[26] 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.

[27] 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.

[28] "Engineering DevOps from Chaos to Continuous Improvement and Beyond," 2019.

[29] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2017-Decem, pp. 74–81, 2017, doi: 10.1109/CloudCom.2017.40.

[30] C. Anderson, "Docker," 2015.

[31] G. Cusack *et al.*, "Efficient microservices with elastic containers," *Conex. 2019 Companion - Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol. Part Conex. 2019*, no. June 2020, pp. 65–67, 2019, doi: 10.1145/3360468.3368180.

[32] 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.

[33] 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.

[34] 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.

[35] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Comput. Sci.*, vol. 2, no. 3, pp. 1–21, 2021, doi: 10.1007/s42979-021-00592-x.

[36] A. Ławrynowicz and V. Tresp, "Introducing machine learning," *Perspect. Ontol. Learn.*, vol. 18, no. January, pp. 35–50, 2014, doi: 10.1007/978-3-030-67626-1_8.

[37] S. Wazir, G. S. Kashyap, and P. Saxena, "MLOps : A Review".

[38] A. Paleyes, R. G. Urma, and N. D. Lawrence, "Challenges in Deploying Machine Learning: A Survey of Case Studies," *ACM Comput. Surv.*, vol. 55, no. 6, 2022, doi: 10.1145/3533378.

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] B. Derakhshan, A. R. Mahdiraji, T. Rabl, and V. Markl, “Continuous deployment of machine learning pipelines,” *Adv. Database Technol. - EDBT*, vol. 2019-March, no. March, pp. 397–408, 2019, doi: 10.5441/002/edbt.2019.35.

[49] 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/

[50] 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.

[51] 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.

[52] E. Casalicchio and S. Iannucci, “The state-of-the-art in container technologies: Application, orchestration and security,” *Concurr. Comput. Pract. Exp.*, vol. 32, no. 17, pp. 1–17, 2020, doi: 10.1002/cpe.5668.

[53] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, “Sinan: ML-based and QoS-aware resource management for cloud microservices,” *Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS*, pp. 167–181, 2021, doi: 10.1145/3445814.3446693.

[54] S. Venkateswaran and S. Sarkar, “Fitness-Aware Containerization Service Leveraging Machine Learning,” *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1807–1820, 2021, doi: 10.1109/TSC.2019.2898666.

[55] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, “FScaler: Automatic Resource Scaling of Containers in Fog Clusters Using Reinforcement Learning,” *2020 Int. Wirel. Commun. Mob. Comput. IWCMC 2020*, no. October, pp. 1824–1829, 2020, doi: 10.1109/IWCMC48107.2020.9148401.

[56] S. Zhang, T. Wu, M. Pan, C. Zhang, and Y. Yu, “A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning,” *Proc. - 2020 IEEE 13th Int. Conf. Web Serv. ICWS 2020*, pp. 489–497, 2020, doi: 10.1109/ICWS49710.2020.00072.

[57] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, “Firm: An intelligent fine-grained resource management framework for SLO-Oriented microservices,” *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation, OSDI 2020*, pp. 805–825, 2020.

[58] Y. Cheng, C. Wang, H. Yu, Y. Hu, and X. Zhou, “GRU-ES: Resource usage prediction of cloud workloads using a novel hybrid method,” *Proc. - 21st IEEE Int. Conf. High Perform. Comput. Commun. 17th IEEE Int. Conf. Smart City 5th IEEE Int. Conf. Data Sci. Syst. HPCC/SmartCity/DSS 2019*, pp. 1249–1256, 2019, doi: 10.1109/HPCC/SmartCity/DSS.2019.00175.

[59] R. Fu, Z. Zhang, and L. Li, “Using LSTM and GRU Neural Network Methods for Traffic Flow Prediction,” *IEEE*, pp. 5–9, 2016.

[60] J. E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, “Investigating Machine Learning Algorithms for Modeling SSD I/O Performance for Container-Based Virtualization,” *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1103–1116, 2021, doi: 10.1109/TCC.2019.2898192.

[61] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, “Migration Modeling and Learning Algorithms for Containers in Fog Computing,” *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 712–725, 2019, doi: 10.1109/TSC.2018.2827070.

[62] Y. Bao, Y. Peng, and C. Wu, “Deep Learning-Based Job Placement in Distributed Machine Learning Clusters With Heterogeneous Workloads,” *IEEE/ACM Trans. Netw.*, vol. 31, no. 2, pp. 634–647, 2023, doi: 10.1109/TNET.2022.3202529.

[63] Y. Xu, J. Yao, H. A. Jacobsen, and H. Guan, “Cost-efficient negotiation over multiple resources with reinforcement learning,” *2017 IEEE/ACM 25th Int. Symp. Qual. Serv. IWQoS 2017*, 2017, doi: 10.1109/IWQoS.2017.7969160.

[64] A. I. Orhean, F. Pop, and I. Raicu, “New scheduling approach using reinforcement learning for heterogeneous distributed systems,” *J. Parallel Distrib. Comput.*, vol. 117, pp. 292–302, 2018, doi: 10.1016/j.jpdc.2017.05.001.

[65] F. Rossi, M. Nardelli, and V. Cardellini, “Horizontal and vertical scaling of container-based applications using reinforcement learning,” *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2019-July, pp. 329–338, 2019, doi: 10.1109/CLOUD.2019.00061.

[66] H. C. Song, “An overview of underwater time-reversal communication,” *IEEE J. Ocean. Eng.*, vol. 41, no. 3, pp. 644–655, 2016, doi: 10.1109/JOE.2015.2461712.

[67] H. Dawid, *CORDIC Algorithms and Architectures*. 2018. doi: 10.1201/9781482276046-22.

[68] S. Y. Shah, Z. Yuan, S. Lu, and P. Zerfos, “Dependency analysis of cloud applications for performance monitoring using recurrent neural networks,” *Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017*, pp. 1534–1543, 2017, doi: 10.1109/BigData.2017.8258087.

[69] X. Tang, Q. Liu, Y. Dong, J. Han, and Z. Zhang, “Fisher: An efficient container load prediction model with deep neural network in clouds,” *Proc. - 16th IEEE Int. Symp. Parallel Distrib. Process. with Appl. 17th IEEE Int. Conf. Ubiquitous Comput. Commun. 8th IEEE Int. Conf. Big Data Cloud Comput. 11t*, pp. 199–206, 2018, doi: 10.1109/BDCloud.2018.00041.

[70] M. Yan, X. M. Liang, Z. H. Lu, J. Wu, and W. Zhang, “HANSEL: Adaptive horizontal scaling of microservices using Bi-LSTM,” *Appl. Soft Comput.*, vol. 105, p. 107216, 2021, doi: 10.1016/j.asoc.2021.107216.

[71] Y. L. Cheng, C. C. Lin, P. Liu, and J. J. Wu, “High resource utilization auto-scaling algorithms for heterogeneous container configurations,” *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, vol. 2017-Decem, pp. 143–150, 2017, doi: 10.1109/ICPADS.2017.00030.

[72] K. Ye, Y. Kou, C. Lu, Y. Wang, and C. Z. Xu, “Modeling Application Performance in Docker Containers Using Machine Learning Techniques,” *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, vol. 2018-Decem, pp. 1057–1062, 2018, doi: 10.1109/PADSW.2018.8644581.

[73] H. Zhang, H. Ma, G. Fu, X. Yang, Z. Jiang, and Y. Gao, “Container based video surveillance cloud service with fine-grained resource provisioning,” *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 0, pp. 758–765, 2016, doi: 10.1109/CLOUD.2016.103.

[74] Y. Meng, R. Rao, X. Zhang, and P. Hong, “CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis,” *PIC 2016 - Proc. 2016 IEEE Int. Conf. Prog. Informatics Comput.*, pp. 468–472, 2017, doi: 10.1109/PIC.2016.7949546.

[75] 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.

[76] 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.

[77] 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