

# Microservices caching for container-based IoT system in the edge and cloud

**Rawaa Qasha, Haleema Sulyaman**

Department of Computer Science, College of Mathematics and Computer Science, University of Mosul, Mosul, Iraq

## Article Info

### Article history:

Received Feb 1, 2025

Revised Mar 31, 2025

Accepted Jul 2, 2025

### Keywords:

Automated deployment

Container

DevOps

IoT

Microservices

## ABSTRACT

Microservices enable agile development by dividing internet of things (IoT) programs into autonomous components, ensuring fault tolerance and parallel operation for enhanced productivity. Their adaptability across diverse service types and applications improves IoT system performance. On the other hand, the container is the preferred solution for microservices-based enterprises. To improve the effectiveness of the deployment system presented in our paper 1, we developed a new caching technique to significantly optimize the performance of the deployment system and automate the sharing and re-using of ready-to-run microservices that have been packaged as Docker images. The new caching techniques are seamlessly integrated with our deployment system to optimize the microservices caching of the IoT application by utilizing Docker-based container virtualization and Redis for consistent data sharing. In addition, DevOps and versioning tools such as GOCD and GitHub are integrated into our system to enhance the automatic deployment of the microservices resulting in self-contained, portable, and repeatable IoT microservices. The effectiveness of the proposed techniques is evaluated via various experiments implemented in various working environments where the results show reduced deployment time and the effort required to re-execute the microservices, in addition to the reduction of burden and error that occur when adopting a manual deployment.

*This is an open access article under the [CC BY-SA](#) license.*



## Corresponding Author:

Rawaa Qasha

Department of Computer Science, College of Mathematics and Computer Science

University of Mosul

Mosul, Iraq

Email: rawa\_qasha@uomosul.edu.iq

## 1. INTRODUCTION

Microservices are a collection of small services that come as a result of the fragmentation of a larger project or process and perform a precisely defined function [1], [2]. Due to its widespread application in both social and commercial domains of life, the IoT is highly well-liked. With the advent of the IoT, everything in our environment is viewed as a smart object that communicates with other objects without the need for human intervention. These objects will be employed in a wide range of applications, including those related to industry, transportation, health, and smart cities [3]. When creating IoT applications, a huge amount of data will be produced. This data will need to be processed, and analyzed, in addition to store, so the need for services provided by cloud computing has emerged [4].

Microservices have been adopted as a working principle for organizing applications that rely on cloud computing, and to give the possibility to build and deploy IoT applications in a safe and reproducible manner that adapts to the changes and updates that occur during the application deployment process [5]. Adopting

microservices enables the IoT application to deal with huge amounts of data and improve the performance of the system that is deployed in multiple environments, including the Cloud environment [6]. On the other hand, container-based virtualization technology is used to produce lightweight microservices by packaging all the software resources, libraries, and other dependencies that the microservices need. Our previous work presented [7], shows how containers can be used to support the portability, dynamic deployment, and repeatability of IoT applications. Thus, using container-based virtualization, a group of microservices can be fully isolated. Therefore, we enable reserving or limiting the use of system resources for a specific microservices group to meet its needs in terms of caching, and by using Redis to store data and the possibility of sharing it among the container group, the caching process will become optimized and efficient [8], [9].

To improve our deployment system, we proposed a new approach for optimizing the performance of the system by developing microservices caching for the container-based IoT application by integrating the Docker container with Redis, versioning control, and GOCD to deploy the IoT microservices in portable and repeatable features. The main contributions of this research are:

- Enhance the automatic deployment framework on various environments presented in our previous work [1].
- Implementing multi-level Docker-image caching for the microservices of the IoT application to enhance the performance of the deployment system.
- Integration of Docker technology with Redis and versioning control to optimize Docker's image caching.

## 2. RELATED WORKS

In this section, we will present some details about the most important research and papers that have worked within the same field of research presented here, Alam *et al.* [10] presented a framework to support reproducibility, reusability, and on-demand provisioning by integrating physical and logical supplies, and this, in turn, will increase the reproducibility feature of the proposed system. In addition, the work presented a method for automatic management of system tasks and following up on changes that occur in the system. The optimization presented in this work was achieved based on the fact that the tasks performed by the workflow in the system share tasks, so adopting the caching principle will speed up the process of workflow provisioning.

Beltrão *et al.* [11] highlighted the scheduling methods used for the purpose of scheduling a group of containers in systems that depend on containers in their work, and among the most important tools used for the purpose of managing these containers in such systems are swarm, k8s, and mesos, and then comparing their efficiency in terms of their energy consumption, operating time, access time, time-latency, and load balancing for the purpose of obtaining the best way to provisioning the microservices used in Internet of Things applications.

Kałaska and Czarnul [12] relied on K8s for the purpose of deployment between internet of things devices and the cloud. Two scenarios were adopted: the first is to use the http protocol and the second is to use the amqp protocol to transfer Internet of Things data. It turns out that using K8s may make it difficult to deploy an IoT application with regard to the worker requirements of the cluster and master nodes, so it may not be suitable for all IoT applications, but if it covers all the required requirements, it will overcome the complexities and improve bootstrap time when deploying an IoT application between cloud environment and smart Internet of Things devices.

Ling *et al.* [13] presented an architecture to achieve modularity and scalability based on container technology. By combining the principle of modularity with the coordination provided by Docker Swarm, it has become possible to deploy an Internet of Things application in a distributed manner, thus obtaining a system capable of adapting to the dynamic changes that occur in the system. In addition, to achieve availability by providing several replicas distributed over different environments.

Neto [14] relied on the FAAS method to implement the set of microservices presented in the system, where each microservice of the system was designed as a FAAS. Then, the researchers in this work experimented and implemented these microservices and compared them with the case of their implementation as one integrated part. In addition to making a comparison of the cost of using both methods.

De Prado *et al.* [15] presented a method for designing an architecture based on docker container. The designed architecture was used to develop serverless applications that operate in parallel, are event-driven, and operate on custom environments in which Docker images can be built and executed, such as the AWS LAMBADA environment. This serverless architecture was used with container-based technology. Applying optimization based on the principle of caching for the purpose of reducing cost and achieving higher throughput during the execution of system tasks, especially tasks that need to be executed in parallel.

Qasha *et al.* [16] presented a set of tests with a number of clients, considering that they are a group of sensors that will send the collected data to the IoT device hive, which will represent the IoT middleware. The test was conducted in two different scenarios, the first is deployment based on docker container and the second is deployment without using docker container technology. Thus, it became clear through the

experiments carried out in this work that using the first scenario led to less burden compared to following the second scenario, especially when used with a group of clients in the system. In addition to testing to measure performance when expanding it by increasing the number of nodes in the system, taking into account the memory needs of the different nodes. Tian *et al.* [17] presented a model that supports temporary caching of microservices at the edge of the Internet of Things network for time-sensitive applications that operate on the principle of mobile computing. This model was applied based on "Markov Decision Process MDP" to improve the percentage of information presence in microservices and reduce the resulting delay, in addition to using the "Distributed Double Dueing Deep Q Network D3QN" algorithm, which was combined with the "Double DQN" and "Dueling DQN" algorithms to solve the equation resulting from "MDP" to produce an Internet of Things system with appropriate efficiency.

Finally, Tang *et al.* [18], the researchers compared the application of genetic algorithm with Cost\_greedy, time\_greedy, PSO, and Random techniques to solve the problem of deploying IoT applications in cloud and bus environments. The goal was to reduce the delay when there are cost constraints. The algorithms in this system were applied to "Shanghai Telecom data set".

### 3. MICROSERVICES CACHING FOR IOT USING DOCKER IMAGE

The main goal of the proposed approach is to enhance the performance of the provisioning framework by optimizing container caching to achieve portability and self-contained reproducibility for the system microservices. In addition, the automatic provisioning of the containerized microservices is enhanced by achieving a significant reduction in the effort required by a user to re-execute the microservices and the overall deployment and enactment time. The main aim of adding caching to the provisioning process is to automate the sharing and re-usability of the container-based IoT microservices.

Our caching techniques tackle the following situations: i) repeatedly using the microservices or other IoT components by the same IoT system or by a different one and ii) re-running instances of an IoT system in the same environment or on different ones. To realize and explore the optimization of our provisioning system, we have implemented the caching techniques such that they are seamlessly integrated into our existing system, while the essential components remain unchanged.

The core functionality of our caching part is the automatic publishing and sharing of the IoT application and components in two levels: i) the local execution environment (on-host cache) and ii) public cache (online repository such as Docker Hub). Using the two levels of caching ensures the ability to share the IoT application and its components with any user either locally or publicly.

Complex interdependencies significantly complicate cache management and can lead to severe inconsistencies if not handled properly. In this research we developed a caching strategy that maintain data consistency across dependent microservices, preventing data corruption and application errors. It uses Redis as a valuable tool in implementing caching strategies that aim to maintain data consistency across dependent microservices. Redis strengthens microservice caching through:

- High speed: in-memory storage for rapid data access.
- Flexible data structures: versatile data handling for complex relationships.
- Real-time invalidation: pub/sub for efficient cache updates across services.
- Atomic operations: transactions for consistent multiple cache changes.

#### 3.1. Caching the workflow and task images

To simplify the sharing of our images, we automate the process of publishing the IoT components' images to ensure their immediate availability, consequently improving the performance of the IoT application. The ability to automatically share and re-use those components saves the required time for provisioning the full software stack of each microservice or IoT application. Further, the IoT system can easily run in other environments.

As presented in our previous work [1], in the single-container deployment scenario only one image will be created which is the IoT system image. Once the creation of this image ends, publishing the image will be started to the two cache levels. Specifically, in the single-container scenario, the IoT image is created after the provisioning of all components. Following that, the produced image will be pushed to the local and public cache to be available for any consequent use. Further, a search procedure has been implemented to automatically check the availability of the IoT image at the various cache levels. Therefore, for any consequent use in the future, the first step of IoT system deployment is the automatic search for a compatible image in the two levels. If the image is available, the framework will directly create a container and straightway proceed to task execution. Importantly, if no image is found then a base image is used. In the case of a multi-container deployment scenario, we use an approach very similar to cache microservice images. Instead of sharing the IoT system image, images corresponding to IoT microservices will be shared.

However, the caching of each microservice image starts immediately after the image creation and before the start of the next microservice provisioning. Redis is a powerful tool for caching in microservices, but it requires careful planning and implementation to ensure data consistency. It's crucial to understand that Redis itself doesn't magically solve all consistency problems. It provides the building blocks, and in future, we must implement appropriate patterns and strategies such as using Redis Monitoring Tools, e.g., RedisInsight, Prometheus, Grafana to track key metrics. In addition to use the Performance Tuning to Tune Redis configuration parameters, e.g., maxmemory, eviction policy based on monitoring data.

### 3.2. Enhancing container-based microservices deployment

This paper's goal of enhancing the IoT microservices deployment and implementing Docker-image caching has been achieved by adopting the following technologies:

- Running Docker Containers with Redis Redis is widely used for caching applications, which reduces the burden on servers and expedites loading times [19]. It can also act as a message broker to facilitate communication between different application components. Redis additionally supports transactions, enabling the atomic execution of several processes [20]. In our research, we explore mechanisms for verifying the integrity and authenticity of Docker images, such as digital signatures that is used between the different environments, content trust by using the CI/CD Pipeline, which has a static analysis, vulnerability scanning, and image signing.

In our previous work [7], the container technology has been integrated with TOSCA specification to develop a new on-demand and automatic deployment approach. By using Redis, a newly built Redis instance can be launched quickly and simply using Docker to launch the container. Adopting Redis raises some valuable features such as facilitating isolation, portability, fast setting up and Tear-Down, versioning, and cleaning development settings:

- Versioning control a system called version control, sometimes referred to as source management or versioning control keeps track of and controls changes made to files and software as time passes. Keeping track of various source code versions is a common practice in developing software, enabling multiple programmers to work together on the project and effortlessly control changes [21]).

In this work, we use versioning control as an essential tool for effective, modification tracking, code review, and code stability in developing our framework, It keeps track of all modifications in a well-organized manner:

- GoCD for Continuous Integration and Deployment For the “continuous integration and deployment” (CI/CD) procedures, GoCD is a wellliked open-source solution [22], [23]. GoCD has been utilized in this work to streamline software delivery automation. By exploiting the capabilities provided by it, we are capable of accelerating delivery time, the ability to cope with failures and to configure and flexibly deploy microservices. Microservice container development, testing, and deployment will be made possible by the seamless integration of Docker Container with GoCD [24]. In addition, GOCD allows coordinating the configuration process, deploying and updating all the microservices in the system in parallel, even in case of a conflict between the microservices tasks employed in the system [25]. In this work, we have integrated and utilized these technologies to achieve confidential delivery, configuration, and deployment efficiently and smoothly for all microservices used in the system.

The versioned microservices are inherently dynamic. Changes in code, APIs, and dependencies mean that cached data can quickly become obsolete or incompatible. This creates a tension between the performance benefits of caching and the need for data accuracy and consistency. In this research, we overcome these issues by adopting the following points:

- Store version metadata alongside cached data, allowing services to perform runtime compatibility checks. This would enable services to detect and handle incompatible cached data.
- Integrate caching considerations into deployment pipelines. This would involve automatically invalidating or migrating caches during deployments.

## 4. EVALUATION

### 4.1. System setup

The work involved conducting multiple experiments to test the deployment of the suggested IoT microservices, which use DHT11, Gaz, sound, and flame sensors to monitor the surrounding environment. To gather the necessary IoT data, we use a Raspberry Pi 4 with 4 Giga storage. It runs Docker 20.10.7 and the Ubuntu distribution versioning 20.4 LTS of Linux. The researcher also utilizes an AWS EC2 instance running Ubuntu distribution of Linux version 22.4 LTS and Docker 20.10.7 in the cloud-based environment.

#### 4.2. Caching for different provisioning scenarios

The testing experiments utilized in this work are carried out in two scenarios: Monolithic approach, in which every sensor task is handled by a single monolithic program. Microservices approach, in which one microservice is used for every sensor task. In both experiments, the implemented IoT system is hosted on various environments: Raspberry Pi for data collection, Edge node for data preprocessing, and Cloud node for data visualization.

Container-based virtualization is used to implement the suggested IoT system from data collection to data visualization in the hosted environments for the two cases (monolithic and microservices). In the IoT application deployment with a monolithic approach, the deployment scenario requires fewer computer resources such as disk storage and memory, since only one Docker image is required to build a single container that hosts all of the IoT components and their dependencies. When employing a monolithic approach, every task from all sensors used is bundled into a single container. A single basic Python image is used, and the sensor tasks for management and their associated dependencies are integrated into the container. Figure 1 illustrates how to create this container, beginning with acquiring the basic image and any necessary tools, libraries, and artifacts. All tasks within the container go through the same procedure of downloading these prerequisites again. To conserve space, the container will be completely erased or destroyed once all of the actions inside of it have been accomplished.

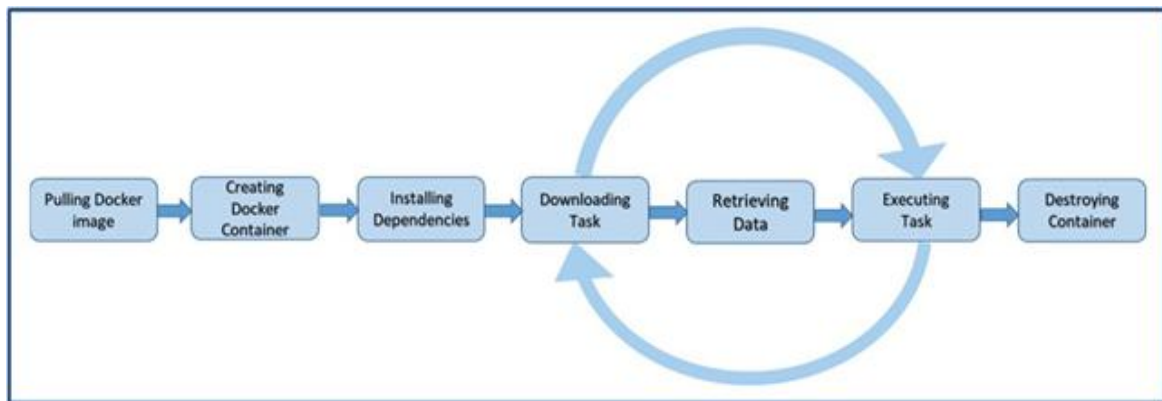


Figure 1. Monolithic approach for implementing the system containers

The second method packages each sensor's tasks into a self-contained Docker container and is considered a microservice. Therefore, each container includes all necessary activities, dependencies, and tools to manage the data from a single sensor. Every IoT application task will adhere to the particular deployment strategy. First, the basic image was downloaded from the internet repository. Next, all the software libraries, artifacts, and tools required for this work were downloaded. In general terms, Docker containers are transient; without using a volume, data is lost when containers are removed.

The system that is built during the work supports both cases. In both cases, GOCD relies on completely automating the publishing process, starting with the process of searching and obtaining the required images, then implementing the necessary steps to obtain the required results from the IoT application, and completing all the containers within the system. The optimization process here is done by relying on Redis to cache the images present in the system. Usually, preparing the images and uploading them to Docker Hub is preferable, as it is easier than building new images from scratch. Then, by using Redis, all containers in the system will access these images and the data that deals with them. By using containers and adopting the monolithic method, the time taken to implement and deploy the IoT application is less compared to using microservices, especially when all application tasks share all dependencies, including the libraries and programs used, provided that no conflict occurs between the needs of the tasks of the proposed application.

#### 4.3. The impact of multi-caching cases on the iot deployment with different scenarios

In this section, the effect of our caching techniques is highlighted on the deployment of an IoT application on three environments: data-gateway (Raspberry Pi device), edge, and cloud with two deployment scenarios: monolithic and microservices. The IoT applications have been deployed in separate environments with three different cases:

- Deployment with base-image: in this case, the IoT application has been deployed with an empty cache, and a base image is used to create the containers for the IoT components on Monolithic and Microservices scenarios. In this deployment case, the images are created automatically from the IoT containers and pushed to the Docker Hub.
- Deployment with on-host cache: in which the IoT application is deployed using images available from the on-host cache in the execution environment.
- Deployment with Public cache: in this case, the IoT is deployed using images in Docker-Hub.

Figure 2 presents the results of deploying time for the IoT application with the two scenarios and the three cases. The results show the impact of the caching on the deployment time. In detail, the case of deployment with an on-host cache has the most significant influence on the deployment time which can be recognized as the shortest time compared with the full deployment case. The deployment with a public cache also significantly impacts the deployment time with slight differences between the two deployment scenarios. Deployment with on-host cache the two scenarios require less time than using the deployment with public-cache because of the time required to download the images from the Docker-Hub servers.

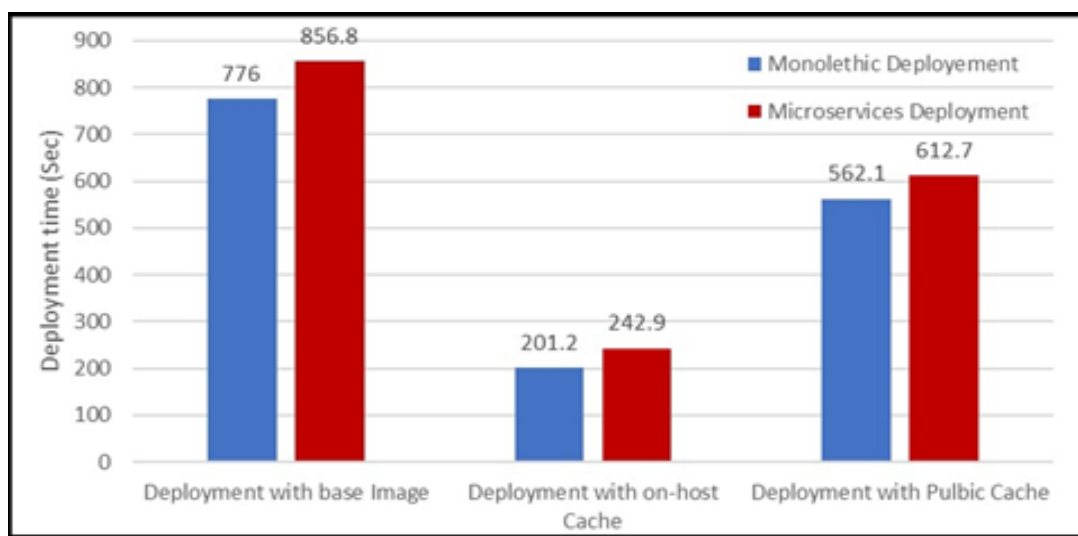


Figure 2. The execution time for monolithic and microservices scenarios

#### 4.4. STORAGE RESOURCES FOR THE DEPLOYMENT SCENARIOS

As we have mentioned in section 3.2, building and deploying containers to implement operations using microservices will take more time than using the monolithic method. However, in the monolithic method there is no type of isolation for the tasks involved in the application, and this in turn will lead to unwanted security problems. However, using this method, the researchers will need fewer system resources to deploy the proposed application, especially regarding the use of memory and hard disk, in addition to that all the offices and software used that are shared between the application tasks will be downloaded and installed once and then used by the tasks. By adopting the microservices method, the deployment environment will be secure and isolated for each task by using a dedicated container for each microservice to pack its dependencies. Thus, this method enables us to execute more than one task at the same time, even in the event of a conflict between tasks. In any case, using a dedicated container for each microservice may lead to an increase in system resource consumption compared to using the monolithic method. Figure 3 shows the storage consumption for the two methods.

#### 4.5. Using relevant reliability metrics for effectiveness evaluation of the deployment scenarios

For The effectiveness evaluation of the deployment scenarios applied in this research, a comprehensive assessment that includes metrics like mean time between failures (MTBF), which is used to track the time between failures and calculate the average. We use it to compare different deployment scenarios of the system. The second metric is mean time to recovery (MTTR), which is used to measure the average time it takes to restore the system after a failure. A lower MTTR indicates faster recovery and less downtime. Finally, the third metric is Availability, which is used to measure the percentage of time the

system is operational. A higher availability percentage indicates a more dependable system. The results of these evaluation is illustrated in the Figure 4.

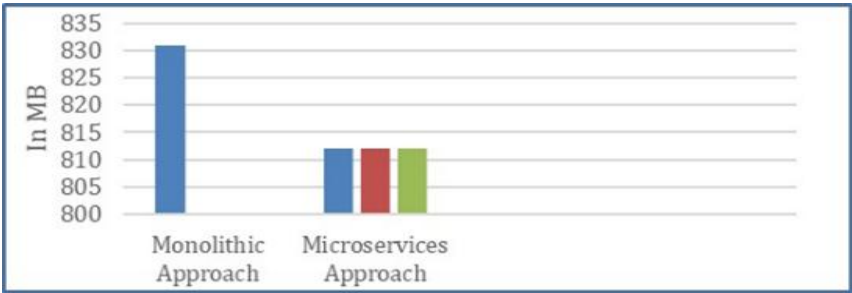


Figure 3. Storage consumption for monolithic and microservices approaches

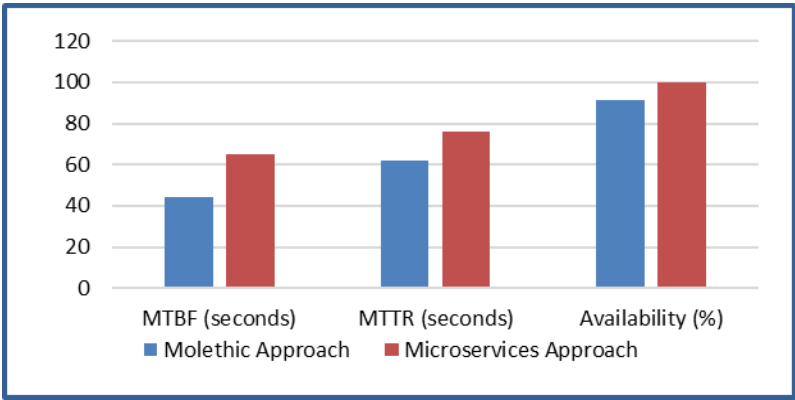


Figure 4. Effective evaluation using reliability metrics

5. CONCLUSION

This paper introduces an optimized microservices deployment system, enhancing performance and automation by adopting Docker-image caching for the IoT system. It proposes a novel framework that merges version control, automated deployment, and Docker technology to combat IoT application decay. The system improves performance, repeatability, and the sharing and reusing of applications by automatically recognizing tasks and container images. Our tests validate the methodology and highlight the benefits of the proposed optimizations, showing that image caching and automated microservices deployment accelerates provisioning in various scenarios.

Our future work aims to incorporate container orchestration tools and expand to larger IoT applications, leveraging AI technologies for data processing, analysis, and visualization.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Rawaa Qasha	✓	✓			✓	✓		✓		✓		✓	✓	
Haleema Sulyaman		✓	✓	✓	✓	✓		✓	✓		✓			

C : Conceptualization	I : Investigation	Vi : Visualization
M : Methodology	R : Resources	Su : Supervision
So : Software	D : Data Curation	P : Project administration
Va : Validation	O : Writing - Original Draft	Fu : Funding acquisition
Fo : Formal analysis	E : Writing - Review & Editing	

## CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

## DATA AVAILABILITY

The data that support the findings of this study will be available in [https://github.com/HaleemaEssa/rpi-blynk/blob/main/Advanced\\_IoT\\_Dataset.csv](https://github.com/HaleemaEssa/rpi-blynk/blob/main/Advanced_IoT_Dataset.csv) following a [6-month] embargo from the date of publication to allow for the commercialization of research findings.

## REFERENCES




- [1] H. E. Solayman and R. P. Qasha, "Seamless integration of DevOps tools for provisioning automation of the IoT application on multi-infrastructures," in *2023 3rd International Conference on Intelligent Communication and Computational Techniques, ICCT 2023*, IEEE, Jan. 2023, pp. 1–7. doi: 10.1109/ICCT56969.2023.10075814.
- [2] A. H. Ibrahim, M. Eliemy, and A. A. Youssif, "An enhanced adaptive learning system based on microservice architecture," *Future Computing and Informatics Journal*, vol. 8, no. 1, pp. 1–40, 2023.
- [3] M. Chadha, V. Pacyna, A. Jindal, J. Gu, and M. Gerndt, "Migrating from microservices to serverless: an IoT platform case study," in *WoSC 2022 - Proceedings of the 8th International Workshop on Serverless Computing, Part of Middleware 2022*, New York, NY, USA: ACM, Nov. 2022, pp. 19–24. doi: 10.1145/3565382.3565881.
- [4] L. Farhan and R. Kharel, "Internet of things: vision, future directions and opportunities," in *Smart Sensors, Measurement and Instrumentation*, vol. 29, 2019, pp. 331–347. doi: 10.1007/978-3-319-99540-3\_17.
- [5] M. I. Rahman, S. Panichella, and D. Taibi, "A curated dataset of microservices-based systems," in *SSSME-2019: Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution*, CEUR Workshop Proceedings, 2019. [Online]. Available: <http://urn.fi/urn:nbn:de:0074-2520-6>
- [6] R. K. Naha *et al.*, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018, doi: 10.1109/ACCESS.2018.2866491.
- [7] H. E. Solayman and R. P. Qasha, "On the use of container-based virtualisation for IoT provisioning and orchestration: a survey," *International Journal of Computing Science and Mathematics*, vol. 18, no. 4, pp. 299–311, 2023, doi: 10.1504/IJCSM.2023.135042.
- [8] S. V. Amanuel and I. M. Ahmed, "A review of the various machine learning algorithms for cloud computing," in *ICOASE 2022 - 4th International Conference on Advanced Science and Engineering*, IEEE, Sep. 2022, pp. 124–129. doi: 10.1109/ICOASE56293.2022.10075592.
- [9] M. Al-Dabbagh and A. K. Ali, "Employing light fidelity technology in health monitoring system," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 26, no. 2, p. 989, May 2022, doi: 10.11591/ijeecs.v26.i2.pp989-997.
- [10] H. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT Using docker and edge computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, Sep. 2018, doi: 10.1109/MCOM.2018.1701233.
- [11] A. C. Beltrão, B. B. N. de França, and G. H. Travassos, "Performance evaluation of kubernetes as deployment platform for IoT devices," in *23rd Iberoamerican Conference on Software Engineering, CIBSE 2020*, 2020.
- [12] R. Kałaska and P. Czamul, "Investigation of performance and configuration of a selected IoT System-middleware deployment benchmarking and recommendations," *Applied Sciences*, vol. 12, no. 10, p. 5212, May 2022, doi: 10.3390/app12105212.
- [13] W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: a dynamic and efficient serverless and faas framework for private cloud," in *Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019*, IEEE, Dec. 2019, pp. 1416–1421. doi: 10.1109/CSCI49370.2019.00265.
- [14] N. M. L. Neto, "A Container-based architecture for accelerating software tests via setup state caching and parallelization," Universidade do Porto (Portugal) ProQuest Dissertations & Theses, 2019.
- [15] R. P. De Prado, S. García-Galán, J. E. Muñoz-Expósito, A. Marchewka, and N. Ruiz-Reyes, "Smart containers schedulers for microservices provision in cloud-fog-IoT networks. challenges and opportunities," *Sensors*, vol. 20, no. 6, p. 1714, Mar. 2020, doi: 10.3390/s20061714.
- [16] R. Qasha, Z. Wen, J. Cała, and P. Watson, "Sharing and performance optimization of reproducible workflows in the cloud," *Future Generation Computer Systems*, vol. 98, pp. 487–502, Sep. 2019, doi: 10.1016/j.future.2019.03.045.
- [17] H. Tian *et al.*, "DIMA: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, vol. 25, no. 5, pp. 1769–1792, Sep. 2022, doi: 10.1007/s11280-021-00939-7.
- [18] B. Tang, X. Zhang, Q. Yang, X. Qi, F. Alqahtani, and A. Tolba, "Cost-optimized Internet of Things application deployment in edge computing environment," *International Journal of Communication Systems*, vol. 38, no. 1, Jan. 2025, doi: 10.1002/dac.5618.
- [19] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version Control System: A Review," *Procedia Computer Science*, vol. 135, pp. 408–415, 2018, doi: 10.1016/j.procs.2018.08.191.
- [20] A. Sokolov, A. Larionov, and A. Mukhtarov, "Distributed system for scientific and engineering computations with problem containerization and prioritization," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 14123 LNCS, 2024, pp. 68–82. doi: 10.1007/978-3-031-50482-2\_6.
- [21] S. M. Sumukh, A. Bhadauria, K. Nandy, and S. Upadhyay, "Lightweight data storage and caching solution for MQTT broker on edge - a case study with SQLite and redis," in *Proceedings - IEEE 21st International Conference on Software Architecture Companion, ICSA-C 2024*, IEEE, Jun. 2024, pp. 368–372. doi: 10.1109/ICSA-C63560.2024.00066.






- [22] S. B. Capgemini, "DevOps essentials: key practices for continuous integration and continuous delivery," *International Numeric Journal of Machine Learning and Robots*, vol. 8, no. 8, pp. 1-14, 2024, [Online]. Available: <https://ijnmr.com/index.php/fewfewf/article/view/83>
- [23] D. B. Jagannadha Rao, Y. H. Reddy, and V. Polepally, "Scaling-Up jenkins with a single node instance in DevOps," *Intelligent Computing and Automation*, pp. 299-309, 2025, doi: 10.1007/978-981-96-0143-1\_24.
- [24] D. Pianini and A. Neri, "Breaking down monoliths with Microservices and DevOps: an industrial experience report," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2021, pp. 505-514. doi: 10.1109/ICSME52107.2021.00051.
- [25] F. El Aouni, K. Moumane, A. Idri, M. Najib, and S. U. Jan, "A systematic literature review on Agile, Cloud, and DevOps integration: Challenges, benefits," *Information and Software Technology*, vol. 177, p. 107569, Jan. 2025, doi: 10.1016/j.infsof.2024.107569.

## BIOGRAPHIES OF AUTHORS



**Rawaa Qasha**    received B.Sc. and M.Sc. In Computer Science from the University of Mosul (UoM) in 1997 and 2000, respectively. In 2000, she started working as a lecturer and researcher in Computer Science at College of Computer Science and Mathematics/UoM. She received a Ph.D. from School of Computing, Newcastle University/UK in 2017. Currently, she is an assistant professor in the College of Computer Science and Mathematics at University of Mosul and a visiting researcher at Newcastle University. Her research interests concentrate on distributed systems, Cloud Computing, Workflow Management, IoT and AI. She can be contacted at email: rawa\_qasha@uomosul.edu.iq.



**Haleema Essa Solayman**    received the B.Sc., MSc, and Ph.D. degrees in computer science from the University of Mosul (UoM), Iraq. In 2006, 2012, 2023 respectively. She started her academic career in the Department of Computer Science at the College of Computer Science and Mathematics in 2006. She taught many practical and theoretical subjects and supervised undergraduate students in the same department. Her research interests include Distributed Systems, Cloud Computing, Containerization, IoT, DevOps, and CI/CD. She can be contacted at email: haleema\_essa@uomosul.edu.iq.