# Development of Docker and Kubernetes Orchestration Platforms for Industrial Internet of Things Service Migration

Mohammed Saleh Ali Muthanna
*Institute of Computer Technologies and Information Security,*
*Southern Federal University*
Taganrog, Russian Federation
muthanna@sfedu.ru

Alexey Tselykh
*Institute of Computer Technologies and Information Security,*
*Southern Federal University*
Taganrog, Russian Federation
tselykh@sfedu.ru

2022 International Conference on Modern Network Technologies (MoNeTec) | 978-1-6654-7246-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/MONETEC55448.2022.9960769

*Abstract*—**This paper has considered an architecture for building the Industrial Internet of Things (IIoT) using edge computing technology. The levels of this architecture and their respective roles are described, as are the benefits of implementing Mobile Edge Computing (MEC) in IIoT, the model of the considered smart factory, and the case in which the application must be migrated from the cloud to the edge device. As part of the explanation for the selection of technologies used in coordinating migration for a network with edge computing, the concepts of designing software based on a microservice architecture were considered, as well as virtualization and containerization technologies. The Docker containerization technology and the Kubernetes orchestration system are explored in depth to facilitate the migration of cloud-based applications to the peripheral.**

***Keywords—Edge Computing, Fog Computing, Internet of Things, Industrial IoT, Docker, Kubernetes***

## I. Introduction

Smart industries powered by Internet of Things (IoT) are the future of the industrial revolution. The IoT ecosystem connects computers, sensors, actuators, and other IoT devices that communicate with each other over the Internet. Most IoT mobile devices, like smartphones, smart bands, cameras, virtual reality glass, etc., are commonly used every day. Many new IoT applications have occurred in the last few years and attracted tons of devotion, e.g. bike-sharing, intelligent and connected vehicles, and healthcare [1]. Emerging IoT technologies such as Narrowband IoT (NB-IoT) and IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) will definitely increase the penetration of IoT in different sectors.

Industry 4.0 is the fourth industrial revolution that started from the manufacturing units and then expanded to every aspect of the industry. It has begun to change the models used for business, relationships with the consumers, and even the strategies used in global innovations [2]. High-computer-intensive IoT applications in the industrial sector need a significant amount of computing power and energy to be processed locally on the devices [3].

Industrial Internet of Things (IIoT) is a concept that is widely used in the emerging industries [4]. The industry's primary purpose is the rapid expansion of industrial automation, which has been accomplished with the aid of IoT devices, sensors, and actuators. The technologies of radio frequency identification and wireless sensor networks have been widely implemented in IIoT [5]. In general, these technologies are computationally intensive, energy-conscious, and latency-sensitive. Consequently, they provide difficulties for the computational and processing skills of end-users.

To overcome these obstacles, researchers have created a new technology known as Multi-access Edge Computing (MEC), which enables the delivery of cloud services within end-users radio access networks [6]. The industry is researching the concept of MEC to improve latency and energy efficiency of future applications [7–9]. By installing edge nodes (ENs) in an industrial setting, end users offload computational activities to the ENs for execution [10]. With the enhanced performance of edge computing, computational offloading to the ENs may improve job execution performance, particularly energy consumption and end-user delay [11-13].

Artificial intelligence and machine learning are considered the driving force of smart factory revolution [14]. Machine learning algorithms are critical for the design and development of smart container schedulers to achieve a more efficient and personalized scheduling of microservices in cloud-fog-IoT networks [15].

The increasing number of IIoT devices faces numerous obstacles, one of which is addressing the issue of time delay during computational offloading. Existing research has provided solutions to this issue in various settings, including single-user, multi-user, and industrial networks. Cloud computing and Infrastructure-as-Code (IaC) supported by technologies such as Docker and Kubernetes [16], efficiently handle many of the challenges of IIoT by orchestrating containers for automated application deployment, elastic scaling, and service discovery.

Our key contributions to solving this issue are as follows:

- Cloud storage for a microservice that will migrate to the network for edge computing;
- A migration technique based on the evaluated criteria and specifications;
- A series of tests for the experiment.

The remaining sections of this work are structured as follows. Section II provides the background knowledge of the microservices. In Section III, the architecture of the network and the major problem statements are described. Section IV describes the research methodologies involved in the proposed approach in a brief manner. In Section V simulation composition is then undertaken. Section VI finally closes the paper.

## II. Preliminaries

Currently, new software development designs are being proposed as a result of the exponential rise of various types of applications, platforms (Internet-based software products), and customer demand for them. The necessity for the product to be able to evolve rapidly and give new functions to users is the driving force behind the need for fundamentally new and efficient software architectures. Currently, the paradigm for developing final goods (for the user) is mostly based on an examination of the user's requirements. And a product that is not user-friendly has no place on the market. When updating any business logic, the software modifications should be modest and should not influence the existing (embedded) system processes. In addition, due to the complexity (in the engineering sense) of software solutions, the software of current and promising services must meet the following requirements: (1) flexibility in customization; sustainability; (2) easy and fast scalability (both individual functions and the entire system as a whole); (3) independence from the environment in which it will be deployed; (4) security; (5) fast completion and updating; (6) modularity; (7) parallel development/testing by multiple teams.

A microservice architectural approach to the creation of highly loaded applications meets all these needs. This architecture is intended to overcome the issue of developing complicated systems by decomposing them into separate modules. The development team determines the level of decomposition (servers according to Model-View-Controller (MVC) model, server modules, classes, objects, databases, or even individual functions-methods) in each instance. Nevertheless, any of the above-described modules can be represented as a common object: a container or a virtual computer. For instance, if the decomposition is not so low-level, the software architecture produced in accordance with the microservice architecture can be depicted as shown in Fig. 1.

Individual independent software modules can be developed in their own "isolated" environment and offered as virtual machines or containers among all the architectural approaches to the creation of server software. Using edge devices as a computing platform brings up the potential of using less powerful devices. Even clever IoT gadgets can't be used to their full potential if their microservice requirements aren't as high as they could be.

## III. Problem Statement and Proposed Work

The primary objective of this work is to investigate the concept of edge computing as it pertains to networks for the Industrial Internet of Things to determine whether it would be feasible to move microservices from the cloud to the edge computing node. The means to complete the migration and continue operating the application must be made available. Correctly designing a strategy for moving applications to edge computing networks involves identifying consistent system needs and related functions based on use cases, upon which to base thoughts about suitable tools for development and construction. According to the assigned tasks, it became important to research and select a viable architectural solution from among all possibilities. Select software for the implementation of this architecture, as well as deploy the solution and its operating components.

### *A. Model description*

As a foundation for the experiment, a model of a smart factory with a functional infrastructure was chosen as shown in Fig. 2. The migration method will be designed for the case of attaching a new manipulator to an operational cluster of devices, which requires a new control software (microservice) that is not present in the cluster controller's local storage.

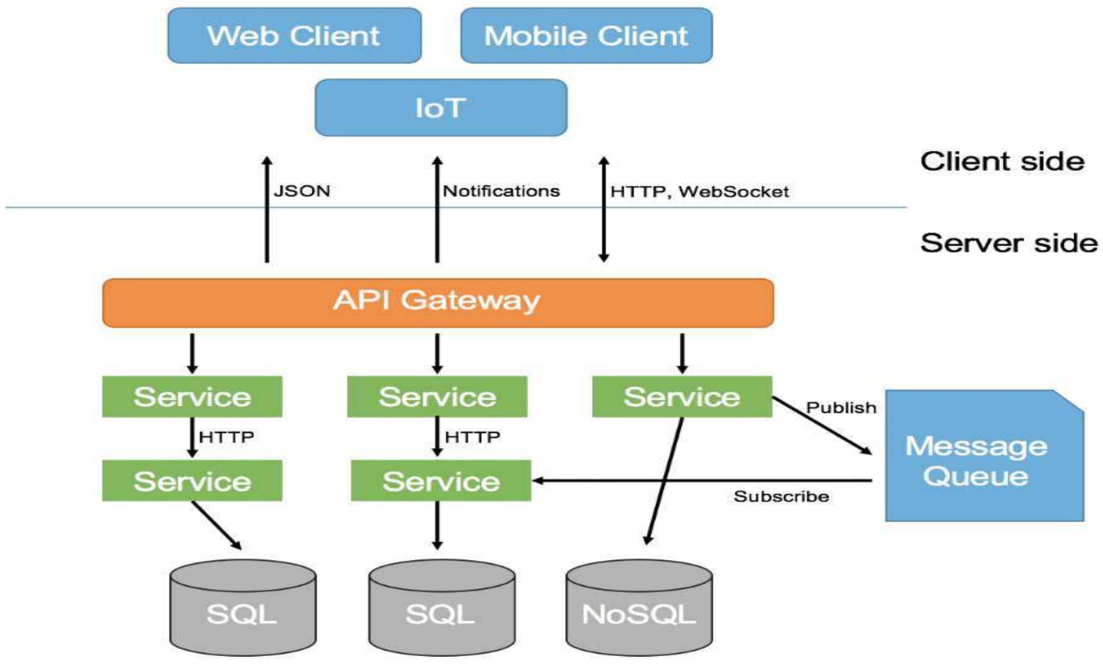

Fig. 1. Microservice software architecture

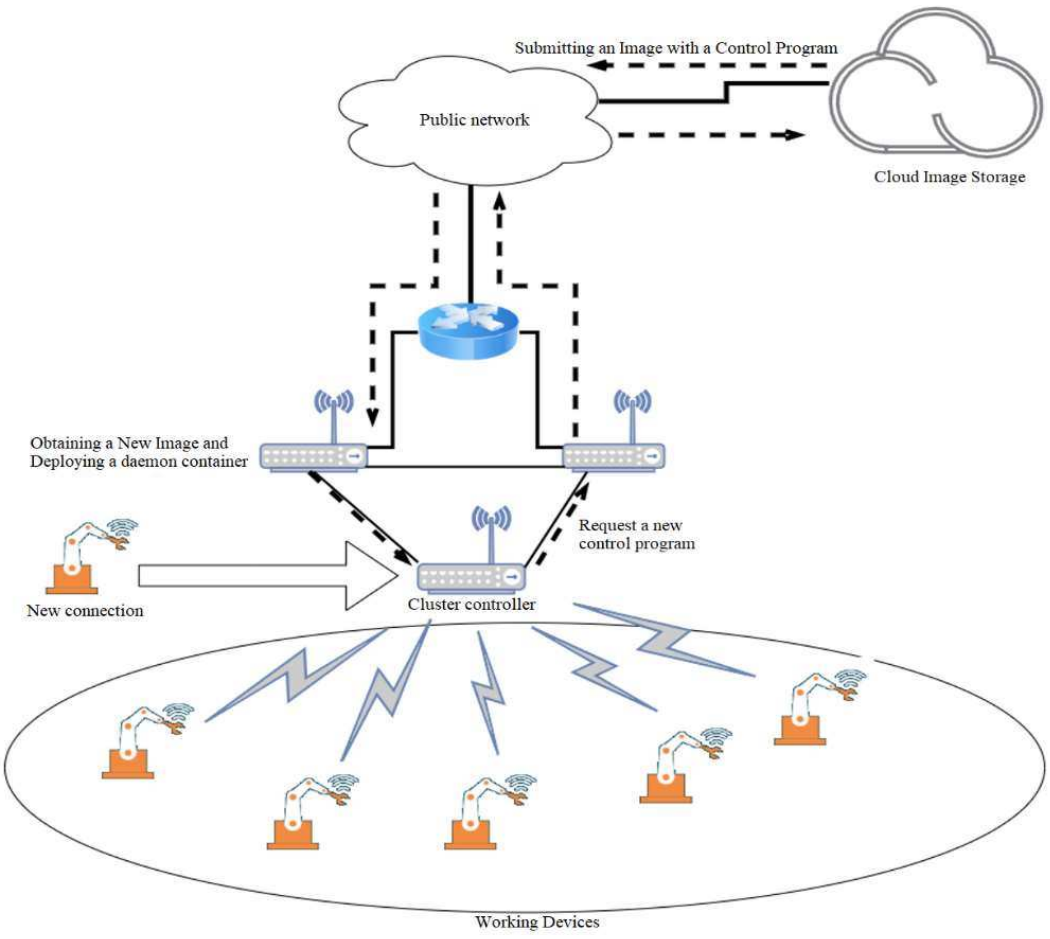
Submitting an Image with a Control Program
Public network
Cloud Image Storage
Obtaining a New Image and Deploying a daemon container
Request a new control program
New connection
Cluster controller
Working Devices

Fig. 2. Smart factory model

All control programs on the controller run in a Docker containerization environment. For the correct connection of a new manipulator, the controller must receive details about the device's model and any necessary software. After receiving and analyzing the received data, the controller sends a request to the cloud storage of images specifying the control container's required image. After acquiring the necessary picture, the container containing the control program will be launched and a network connection will be established to the new device.

## IV. Proposed System

To conduct the experiment, we selected two techniques named methods utilizing Docker functionality and Kubernetes functionality to migrate and install an application from the cloud to the device's edge.

### A. *Method Using Docker Functionality*

This solution presupposes solely using Docker's native capability, as shown in Fig. 3. Docker CLI will be used for management, Docker Hub will serve as remote cloud storage, and Docker daemon will organize the network, download the image, and deploy the container.

The following script in Table 1 will be used for the experiment.

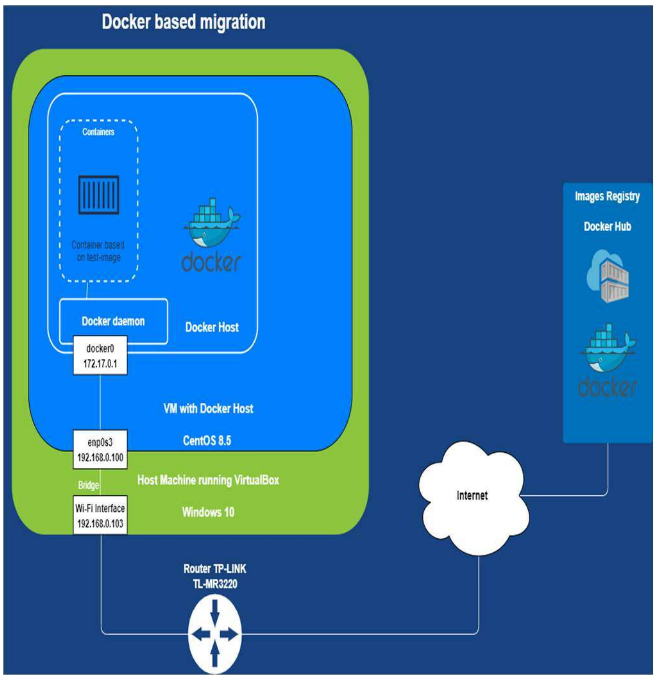

Fig. 3. The method using Docker functionality

TABLE I. LISTING OF THE SCRIPT UTILIZED TO MIGRATE WITH DOCKER CLI

**Bash Script**

```
#!/bin/bash

# Record the start time of the migration

start_time=`date +%s.%N`

# Running a microservice deployment using the missing # image locally, which will be downloaded from Docker Hub

docker run -d -p 5000:22 mruporot/test-image sleep 100

# Elapsed time output

echo -e "\nTime elapsed: $(echo "`date +%s.%N` - $start_time"|bc) seconds.\n"

# Getting the ID of the created container
id=`docker ps -a | grep mruporot | awk '{print $1}'`

# Removing the container and downloaded image to simplify the process of repeating the experiment

docker stop $id
docker rm $id
docker rmi mruporot/test-image
```

In response to executing this script, an image will be downloaded from Docker Hub, a container will be formed and launched on its basis, and the time passed from the beginning of the migration until the container is ready and the application within it begins will be displayed in the terminal. After the script stops the container, remove it and the image obtained from the cloud so the experiment can be repeated.

## *B. Method Using Kubernetes Functionality*

This method will take advantage of the capability of the Kubernetes orchestration system, specifically the declarative method of expressing deployment using a yaml file. To manage the cluster, we will use kubectl software and yaml is an input, while orchestration will be performed by the master node's components, and Docker will serve as the containerization environment, as shown in Fig. 4.

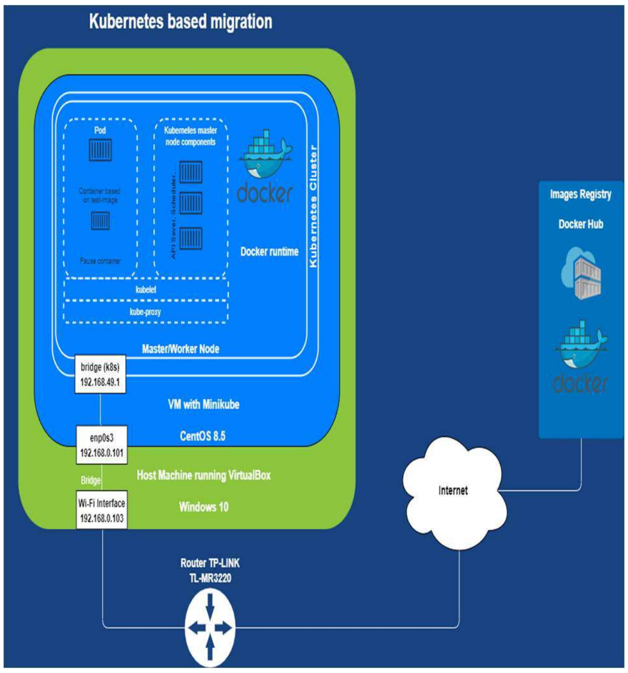

Fig. 4. The method using Kubernetes functionality

The following script in Table 2 will be used for the experiment.

This script will create a new test deployment, which will include an image downloaded from Docker Hub, a pod added, a container with the program and a pause container created and launched in it, a service created to connect the pod to the network, and the time displayed in the terminal, elapsed from the beginning of the migration to the moment the pod is ready and the program within it. It will cease and retire under the service that was deployed once the script deleted the deployment and service. To shorten the script, we repeated the experiment without the downloaded image stored locally on the Kubernetes.

TABLE II. LISTINGS OF THE SCRIPT AND YAML FILE USED TO PERFORM MIGRATION THROUGH KUBERNETES FUNCTIONALITY

**1. Bash Script**

```
#!/bin/bash

# Record the start time of the migration
start_time=`date +%s.%N`
# Start Deployment
kubectl apply -f deployment.yml
# 1 second delay to create a pod entry
sleep 1
# Get the ID of the created pod
pod=`kubectl get pods|grep test|awk '{print $1}'`
# Wait for the full launch of the program in the new field
kubectl wait --for=jsonpath='{.status.phase}'=Running pod/$pod
# Record the start time of the migration
echo -e "\nTime elapsed: $(echo "`date +%s.%N` - $start_time"|bc) seconds.\n"
# Delete the deployment to repeat the experiment
kubectl delete -f deployment.yml
```

**2. deployment.yml**

```
apiVersion: v1
kind: Service
metadata:
  name: test
spec:
  type: NodePort
  selector:
    app: test
  ports:
  - port: 5000
    targetPort: 22
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
      - name: test
        image: mruporot/test-image
        command: ["sleep"]
        args: ["100"]
        resources:
          limits:
            memory: "128Mi"
            cpu: "250m"
        ports:
        - containerPort: 22
```

## V. Experimental Results

To conduct the experiment, each technique was executed 15 times with and once without removing the downloaded image from the registry to estimate the time required to download the image.

During the experiment, both instances deployed a virtual machine in VirtualBox with CentOS 8.5, 4 GB RAM, 4 CPU Cores, and a Network interface in Bridge mode. A small alpine-based image containing 4 MB of disk space was utilized as a container image for testing. Docker version 20.10.15 and Minikube 1.25.2.

### A. Comparative analysis

As anticipated, the experiment revealed that the average migration time while utilizing solely Docker capabilities was shorter than when using Kubernetes: 4.7 sec versus 6.21 sec. Fig. 5 displays 15 measurements.

The time required to deploy the container without downloading the image from the registry was 1.18 sec using Docker and 4.15 sec using the alternative technique as shown in Table 3.

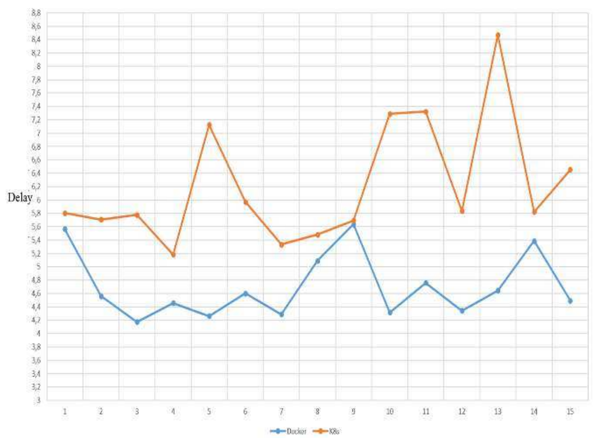

Fig. 5. Comparison of migration time when using two different methods

TABLE III. Comparison of migration time

| № | Docker | K8s |
|---|---|---|
| 1 | 5,565060244 | 5,805279816 |
| 2 | 4,562006411 | 5,706834534 |
| 3 | 4,173712762 | 5,776700607 |
| 4 | 4,456297254 | 5,181515526 |
| 5 | 4,261933743 | 7,124109717 |
| 6 | 4,601513353 | 5,972548791 |
| 7 | 4,289661139 | 5,33466648 |
| 8 | 5,092037997 | 5,48067136 |
| 9 | 5,635733797 | 5,693266123 |
| 10 | 4,316414177 | 7,287788165 |
| 11 | 4,759507059 | 7,321005146 |
| 12 | 4,342420071 | 5,834588486 |
| 13 | 4,643480073 | 8,472057266 |
| 14 | 5,388655502 | 5,819478237 |
| 15 | 4,49317179 | 6,453352564 |
| **Average** | 4,705440358 | 6,217590855 |
| **Without removal** | 1,180632928 | 4,155578596 |

## VI. Conclusion and Future work

In this proposed work, a model of the envisaged smart factory and the circumstance in which the application must be migrated from the cloud to the edge device are detailed. As part of the explanation for the selection of technologies used in planning migration for a network with edge computing, the concepts of designing software based on microservice architecture were considered, as well as virtualization and containerization technologies. In the paper, two approaches to migration, the Docker containerization platform and the Kubernetes orchestration system, were elaborated and compared. The relevant experiments were conducted, and their outcomes were analyzed.

Due to the difficulty to get open server/service failure data in actual cloud data centers, we intend to improve our failure injection technique by paying more attention to diverse cloud computing and Network Functions Virtualization (NFV) use cases. In addition, we intend to incorporate new technology in our future work (i.e., container orchestration, service mesh, monitoring tools, overlay networks, etc.). We will also enhance the predictive machine learning algorithms.

Current research relies on modest, custom-built test beds because Edge/Fog computing is still in its early phases of industrial adoption. However, they lack support for large-scale trials, preventing them from capturing crucial Micro Service Architecture characteristics such as distributed, location-aware deployments, load balancing, dependability, security, and interoperability of services within the large-scale IIoT ecosystem.

## Acknowledgment

The research is supported by postdoc fellowship granted by the Institute of Computer Technologies and Information Security, Southern Federal University, project № P.D./22-01-KT.

## References

[1] Y .S. Jeong, and S. S. Shin, “An IoT healthcare service model of a vehicle using implantable devices,” Cluster Comput, 2018, 21, pp. 1059–1068.

[2] P. W. Khan and Y. Byun, “A Blockchain-Based Secure Image Encryption Scheme for the Industrial Internet of Things,” Entropy 2020, 22, 175.

[3] A. Riofrío and G. Caiza, “Cloud Computing Architecture with Dockers for an Industrial Process,” 2021 International Conference on Engineering and Emerging Technologies (ICEET), 2021, pp. 1–6.

[4] A. Rafiq, W. Ping, W. Min, and M. S. A. Muthanna, “Fog Assisted 6TiSCH tri-layer network architecture for adaptive scheduling and energy-efficient offloading using rank-based Q-learning in smart industries,” IEEE Sensors Journal, 2021, 21(22), pp. 25489–25507.

[5] M. S. A. Muthanna, A. Muthanna, A. Rafiq, M. Hammoudeh, R. Alkanhel, S. Lynch, and A. A. Abd El-Latif, “Deep reinforcement learning based transmission policy enforcement and multi-hop routing in QoS aware LoRa IoT networks,” Comput. Commun, 2021, 183, pp. 33–50.

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing: a key technology towards 5G,” ETSI White Paper, vol. 11, Sep. 2015.

[7] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, “Orchestration of containerized microservices for IIoT using Docker,” 2017 IEEE International Conference on Industrial Technology (ICIT), 2017, pp. 1532-1536.

[8] A. B. M. Adam, X. Wan, and Z. Wang, “Energy Efficiency Maximization in Downlink Multi-Cell Multi-Carrier NOMA Networks With Hardware Impairments,” in IEEE Access, vol. 8, pp. 210054–210065, 2020.

[9] E. M. Dogo, A. F. Salami, C. O. Aigbavboa, and T. Nkonyana, "Taking Cloud Computing to the Extreme Edge: A Review of Mist Computing for Smart Cities and Industry 4.0 in Africa," EAI/Springer Innovations in Communication and Computing, pp. 107–132, 2018.

[10] W. Lee, J. Koo, Y. Park, and S. Choi, "Transfer time, energy, and quotaaware multi-rat operation scheme in smartphone," IEEE Transactions on Vehicular Technology, vol. 65, no. 1, pp. 307–317, 2016.

[11] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," IEEE Communications Surveys and Tutorials, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[12] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration," IEEE Communications Surveys and Tutorials, vol. 19, no. 3, pp. 1657–1681, 3rd Quart. 2017.

[13] A. B. M. Adam, M. S. A. Muthanna, A. Muthanna, T. N. Nguyen, and A. A. A. El-Latif, "Toward Smart Traffic Management With 3D Placement Optimization in UAV-Assisted NOMA IIoT Networks," in IEEE Transactions on Intelligent Transportation Systems, 2022.

[14] R. Cioffi, M. Travaglioni, G. Piscitelli, A. Petrillo, and F. De Felice, "Artificial intelligence and machine learning applications in smart production: Progress, trends, and directions," Sustainability (Switzerland), 2020, 12 (2), article № 492.

[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 (Switzerland), 2020, 20 (6), article № 1714.

[16] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing Docker and Docker-Compose Specifications: A Developers' Survey," IEEE Access, 2022, vol. 10, pp. 2318–2329.