Enabling Cyber Analytics using IoT Clusters and Containers

Soin Abdoul Kassif Traore*, Maria Valero†, Hossain Shahriar†, Liang Zhao†, Sheikh Ahamed‡ and Ahyoung Lee*
*Computer Science Department, Kennesaw State University
‡Computer Science Department, Marquette University
†Information Technology Department, Kennesaw State University
Email: straore3@students.kennesaw.edu, {mvalero2, hshahria, lzhao10, alee146}@kennesaw.edu, sheikh.ahamed@marquette.edu

***Abstract*—Many tech stars like Netflix, Amazon, PayPal, eBay, and Twitter are evolving from monolithic to a microservice (containerization) architecture due to the benefits for Agile and DevOps teams. Microservices architecture can be applied to multiple industries, like IoT, using containerization. Since the IoT industry has exponential growth, universities' responsibility is to teach IoT with hands-on labs to minimize the gap between what the students learn and what is on-demand in the job market. There are many approaches in the containerization field, but they can be challenging to use without depth knowledge in virtualization and code encapsulation. After a deep analysis of the containerization challenges, in this paper, we present a cyberinfrastructure based on containers to solve the virtualization and code-encapsulation problems. The cyberinfrastructure will provide the necessary tools for data collection and code development and testing using an IoT Cluster. It is a web-based platform that allows users to securely go into containerization without spending time learning virtualization. Results show that our proposed cyberinfrastructure allows the creation and deployment of microservices in multiple IoT devices and ensures easy data collection for posterior cyberanalysis.**

***Index Terms*—Containerization, web-based cyberinfrastructure, Cyberanalytics, IoT Cluster.**

I. Introduction

During the past decade, we have seen the adoption of microservices (containers) in the developers' community. According to IT researchers at O'Reilly, "more than 77% of businesses have now embraced microservices. The main reason for deploying microservices is to transform existing monolithic IT environments, systems, architectures, and applications into more flexible applications and solutions [1]." Microservice is mostly an architecture style, but recently, there have been significant improvements of lighter virtualization technologies, such as container-based virtualization and container orchestration solutions, providing portability for microservices [2]. Adopting a microservice architecture approach in IoT devices development will deliver multiple benefits to data collection and cyberanalytics. For example, using containers in IoT devices deployment and testing can help to gain some time in the development of applications for both industry and academia [3].

The continuous development of IoT demands more scalability, extensibility, and maintainability. Unfortunately, monolithic applications cannot offer such much-needed features because of their architectural structure [4]. So, implementing frameworks that provide automation of research processes can help adopt an adequate IoT architecture. Containers are a great application of the microservices concept, and they allow us to implement innovative frameworks to ease the research process around this technology. Thus, a framework that automatically generates microservices designed for specific research needs (e.g. data collection for data analytics) will give more time to develop essential modules without compromising others. Having such a framework can be a game-changer in the process of IoT research.

In this paper, we start exploring the concept and history of microservices implementation. We explore the microservices architectural concept and its emergence Service-Oriented Architecture (SOA), emphasizing self-management and lightness. We also discuss different implementation ideas of the architectural concept of microservices, among others, containerization, service discovery, container orchestration, continuous delivery, and service mesh. Then, we propose an approach to automatic generations of containers and their deployment in a IoT cluster with the idea of improving the data collection process and the cyberanalytics. We explain the details of the proposed framework, and how we can incorporate it into a research project. We focus on different implementations of containerization in general and especially on Docker [5]. The main contributions of this paper are: (1) we review the microservice architectural concept in Service-Oriented Architecture (SOA) and Containers; (2) we discuss potential approaches of cyberinfrastructure implementation using Docker containers. Also, we explore difficulties in IoT research and IoT testing using cloud-based platforms, and we introduce the idea of a dedicated platform for IoT development, testing and data collection; (3) we propose a novel automatic microservice generator for IoT clusters; and (4) we conduct experiments on a cluster-based IoT infrastructure.

II. Microservices Technology

During the last ten years, software engineers came out with a new trend of software design that aims to facilitate the development and delivery of software infrastructure. The

new software design approach is based on independent development units. It is based on Service-Oriented Architecture (SOA) [6]. Microservice is an architectural style inspired by Service-Oriented Computing (SOC) and SOA. It is a relatively simple idea that consists of breaking down the software into an ensemble of executable artifacts. Each of these artifacts is a moonlight that is modelized to communicate with others and share resources, but each artifact is isolated. Microservices rely on REST (Representation State Transfer) and HTTP formats. According to James Lewis and Martin Fowler, the term "microservices" was first discussed in the May 2011 software architecture workshop to denote a common architectural approach the workshop participants had been exploring [7]. From a technical perspective, microservice applications were influenced by new generation software development, deployment, and management tools. Also, the architecture technique was used longtime before the term "microservice." The early architecture also included containers that Docker is used to implementing. In this section, we describe the main components of the Docker architecture and the process behind the use of Dockers.

A. *Microservices and Containers*

Today, software developers have to heed constraints that did not exist a couple of decades ago. Software architecture has to support specific deployment contexts, and it has to be cloud-oriented. There are principles constraints such as services, adaptivity, or models at runtime. These constraints involve taking into account modularity, superposition, and loose coupling. Pattern constraints like deployment specifics such as virtualization aspects which are: shared resources, portable containers, and controller-based feedback loops [8].

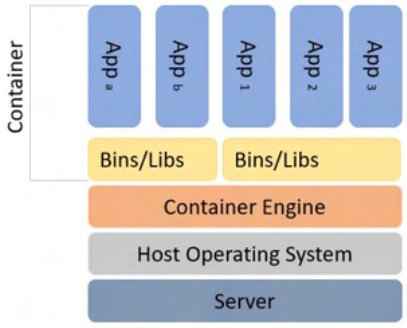

Fig. 1. Container architecture by Docker

Containers are an abstract unit of software inspired by the microservices architecture. They are independent executable units that include code, runtime, system tools, and system libraries to run applications. Containers provide a lightweight virtual environment that groups and isolates a set of processes and resources such as memory, CPU, and disks from the host and any other containers [9]. Processes and resources are inside the container, and thus they provide operating-system-level virtualization by abstracting the userspac As shown in Figure 1. So, containers can have virtual IP addresses, SSH access, OS images, and resources management. These features are possible because Linux Kernel functionalities give an abstraction layer on top of an existing kernel instance to create isolated environments similar to virtual machines.

A container is a group of processes isolated from the rest of the system. They execute their workload from a specific image that provides all files necessary to support the operations. The developer often uses a container engine to do these operations successfully and as an excellent matter. We will focus on the Docker engine, an open platform for developing, shipping, and running applications. Docker enables you to separate your applications in multiple isolated units called a container so that you can deliver software quickly[10]. It is an open-source implementation of the engine that powers dot Cloud, a popular Platform-as-a-Service. Docker architecture is composed of:

- The Docker Client is a Command Line (CLI) Interface used to configure, manage, and interact with Docker.
- Docker Daemon is the server that runs as the daemon. It listens to API requests and manages Docker objects (images, containers, networks, and volumes).
- Images are the read-only template used to create Docker containers. Docker images can be from public Docker repositories using pull and push commands.
- Docker File is configuration files used to build and configure images.
- Containers are instances of images that run applications. CLI manages containers, and new images can be created from custom containers, enabling developers to have safe image samples of their infrastructures.
- Docker Registries is a repository of Docker images, and it is composed of public repositories such as Docker Hub and Docker Cloud. We can pull official software images from the repositories and configure them to our convenience.
- Docker Engine: Combination of Docker daemon, Rest API, and CLI tool.

The process of creating a container with Docker is simple. The commands used by docker build come from a Dockerfile and a context. So, a context is a built set of files located through a specific PATH or an URL. That idea brings us to a build process that can refer to any of the files in a context; it can also use a copy of instruction to reference any file. From the website docs.docker.com, a URL in Docker is a parameter that can refer to three kinds of resources: Git, pre-packaged tarball context, and plain text. Once a container is created, it uses a remote filesystem provided by a container image. Thus, the container image filesystem contains everything needed by the application to run. Also, Docker has a specific tool named Docker Container that manages the container of Docker software. All of those features and functionalities discussed above make Docker the best Container creation and management candidate for the framework implemented in this research paper.

III. Cyberanalytics framework with Dockers

The proposed framework has multiple challenges from the design side to the implementation side. Its primary purpose is to help users get familiar with microservices and offer a reliable and secure space to execute them, and create a platform where students and researchers can easily collect data for cyberanalytics from IoT devices. Thus the framework users can create microservices containers blueprint called Docker Files using a remote a Docker Engine and eventually run containers remotely in IoT devices without any prior resource configuration or download. Also, the cybereanalytics framework has an additional physical part: an IoT cluster for data collection using sensors and Raspberry Pis. The idea is to offer a complete containerized environment capable of supporting cyberanalitycs on IoT devices from software to hardware. Indeed, the design has three main components: a Linux server as the foundation for the framework. Then, we have the Docker Engine that will provide necessary resources for container creation and management. Finally, we have an IoT cluster used for data collection for cyberanalysis purposes.

A. Platform foundation: The Linux Server

The server side is composed of a Linux server, the host Operating System for this framework, and a web server for the user interface. It also includes a Docker Engine (DE) that is the microservices part of the framework. The client-side is a user-friendly website that allows users to set up a test environment with tools, libraries, and dependencies ready for multiple testing scenarios using containers. Various containers can be created quickly and share data and resources if needed for testing through their configuration.

We design an environment to enabling easy user-framework interaction, so they can easily manage containers in the Docker environment. The server has a particular setup and for security purposes. Not only we have to take the regular server security steps, but also, we have to allow users to interact with the servers safely. A user through Docker must create, configure, run, modify and delete containers interpreted in our Linux systems as files. Enabling users to modify files on a server remotely can bring security concerns. So the challenge is to give enough permission to the framework's user to interact with the containers. Users cannot have privileges on the system kernels and critical server files. The idea is to maintain the infrastructure applications and secondary servers in a Docker environment by running them in special containers only accessible by the server administrator. Thus, regular users will be granted some limited privileges on the framework docker engine that will allow them to work with containers safely without compromising the framework or other users' containers.

B. The Docker Engine

Our Docker Engine includes one class with an Apache server image configured to manage the user interface, which is basically a website with unique features. Also, it has a Maria DB container, the database software used to store information accessible from the website. Those first two containers have their own IP addresses. The last container in this class is the PHPmyAdmin software image used to handle the administration of Maria DB over the Web. For efficient management for these two types of containers, we will use Kubernetes to manage the containers. Also, Kubernetes has namespaces and subnamespaces that will offer restrictions necessary for the system hierarchy.

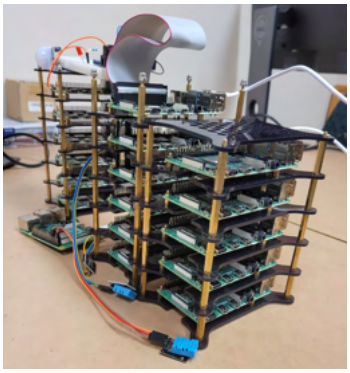

Fig. 2. IoT Clusters Platform for Data Collection

The second class of containers is containers created by the system users for their utilization. Those containers will be accessible by their owners within the system and authorized people they choose to interact with. They are pullable out of the system for personal use, and they can be published so everyone using the framework can have access.

Creating a new container with its configuration is done by using a file called Docker File. In contrast with the containers and images of the framework, Docker Files need to be run from the server to pull images from the Docker Hub or our database. Those will be the only authorized framework user files to run on the Linux server, and we have to handle their privileges. They have only privileges to use Docker daemon to pull or to push container images using the Linux server.

C. IoT Cluster for Data Collection and Cyberanalytics

The IoT data collection platform is part of the physical part of the framework. It adds more flexibility and provides an opportunity to collect, processes, analyze, and visualize data using our framework. The IoT clusters are composed of 20 Raspberry Pis 4 (RP4) mounted in clusters of 6 devices. As shown in Figure 2, each RP4 is connected to a server and has a secure data transfer preconfigured. We have 2 RP4 servers that host a web server for virtual hosts. We also propose 20 Arduinos and 150 sensors, from temperature sensors to cameras and motion sensors. One RP4 cluster has been configured and is ready to use. All servers are implemented with microservices and are part of our integrated framework. The whole platform has a monitoring system using Apache Server instance with a website that collects RP4 and Arduino running statistics using Websockets and HTTPS protocols. As shown in Figure 3, the monitoring websites used JavaScript and PHP to manage the connected devices. The primary purpose of this platform is to have a physical data collecting platform that can load code for

sensors, run the code and transfer the data to the framework for advanced cyberanalytics.

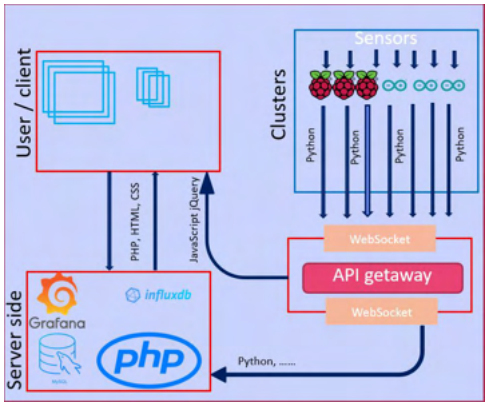

Fig. 3. IoT Clusters Platform Architecture

IV. Framework Implementation

The overall system is a microservices-based framework using Docker to allow students and researchers to take advantage of all functionalities and features proposed by containerization without spending time and energy learning and setting up a personal container database and software. They can also collected data from different sensors for cyberanalytics. The framework provides multiple benefits due to its layered architecture. Layering a system simplifies maintenance and future developments. Thus, we can intervene on different layers without compromising the whole infrastructure. Also, it eases resource allocation. Each layer checks requests and data for errors before routing workflow to the next layer. Entries and data are reviewed twice every time, first by the concerned layer and second by the following layer. Finally, users are free to implement containers responding to unique configurations; the system is flexible and designed for container customization. So it guarantees a certain degree of elasticity compared to actual physical computers or devices.

A. System Architecture

The primary goal of the framework is to help researchers and students in the deployment and testing of codes and collected data for posterior analysis. We want to help them adopt microservices in their process and help them make the transition by offering a framework easy to use with no prior or little knowledge in containers in general and Docker in particular. Thus the framework architecture is user-friendly, and it has information on Docker containers, how to use them efficiently and how to take advantage of them.

We propose an architecture divided into three layers passing sharing information to give users a pleasant experience using the framework. The first layer is the user interface composed of a website. The second and third layers are inside the Docker engines consisting of algorithms and software needed to create and manage containers. We propose an architecture divided into three layers passing sharing information to give users a pleasant experience using the framework. The first layer is the user interface composed of a website. The second and third layers are inside the Docker engines consisting of algorithms and software needed to create and manage containers. Details of the described architecture are shown in Figure 4.

Fig. 4. Proposed Framework Architecture

The user interface is a dynamic website host in a webserver container and can access resources stored in a MariaDB container. The server-side of the website is in PHP, and it handles multiple user functions and interactions with the rest of the framework. PHP scripts run the creation of a new user or connections to user accounts. It also allows users to access containers. Suppose the user wants to create or generate a new container, the Docker file module is used through a docker file generation form. In that case, he has access to a container creation form that includes a list of images present in the framework, a list of image chosen libraries and dependencies, and finally, a list of possible configurations scenarios on how the future container will run. Also, a returning user can access a list of containers created in the preview sessions and a list of containers shared with him by other users. This part of the website is accessible from the User Management Unit. Finally, a user with a container or multiple containers can interact with those instances using the command line tool integrated into the website with a container download option. HTML, JavaScript, and CSS assist the PHP webpages on the client-side to give the user an excellent and clean user interface.

The second layer in the docker engine gathers user information to execute the user workload. When a user requests a docker file generation at the lowe-level, the container building module uses the future container information to pull the needed image from the database. It starts the container creation process by downloading or pulling libraries and dependencies from the local database or the internet. In this part, a PHP code sent to the docker engine handles all processes explained above. Finally, it finishes running the docker file by adding the user code and configurations to the container. Once done, the container is passed to the User management Unit to store the container under user availability and give to the container management unit to access it. This layer offers high scalability

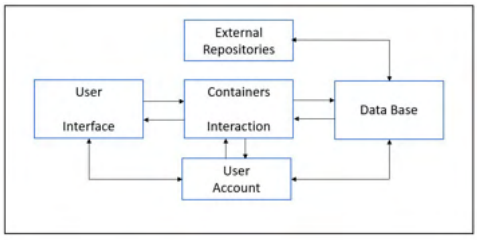

Fig. 5. Framework Overview

to the user. The workflow can be divided between multiple containers providing total control on the instances' runtime scheduling. The third layer, composed of databases run in containers, is saved by information about images and users. It depends on the second layer and executes commands from the three modules that compose that layer. It is also the layer that has internet access to download files needed for container creation. Figure 5 shows a complete overview of how a workflow is handled and processed by different framework modules.

B. Work in progress

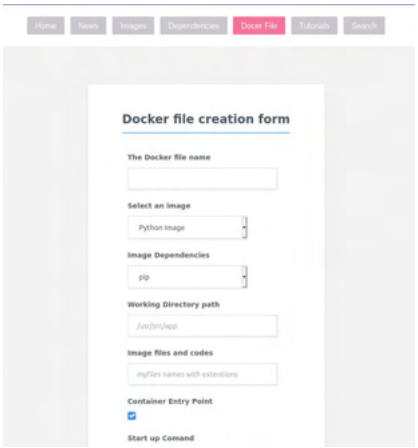

Fig. 6. Docker file Creation From

We have a framework prototype composed of the Linux server, the Docker engine, and The database. A user has access to a websiteand can create Docker files from the system. The Docker file pulls an image from an external repository using the framework and create a container with particular specifications set during the docker file creation process. The website has three main functions: information on containers and how to use them, a docker file creation form, a CLI to interact with the framework. Information provided is about different images and configurations of containers that can be created on the framework. It also has data on particular images, their version, and their dependencies and libraries. So no prior research is needed before starting using the system. The Dockerfile generation form has three main sections handled by an HTML form shown in Figure 6. The first section is important because it contains the name of the future container, and it is essential to have unique names for each container to avoid confusion or collisions. The second section lists images, dependencies, and libraries from which the user can choose. The last area is about different configurations scenarios that a user wants to set within the container, for example, running a specific code at the start of a container running time. The lists are subjective because they are updated in terms of the database. Also, in this part, the user will upload code files to be encapsulated in the container. It is essential because it makes it easier to test developing code. Also, an encapsulated code can be used to test the container functionalities before starting using it. The code file path in the future container is on the form, which is /usr/src/app. Docker recommends that the user use the app folder for their project and different code and script files depending on the type of container; this folder can change. Data and instructions filled in the form are processed in the back using PHP and SQL queries. The user also has access to a CLI; it provides an interface that the user can use to interact with a Docker file previously created. Thus, a user can generate a container from a Docker file and test it to ensure that it has configurations and libraries needed for its work or research. Commands entered in CLI are executed on the Linux Server, and the results are sent back to the web page using JavaScript and PHP. At this point, the actual database stores names of official images from Docker Hub, Dependencies, and libraries related to images and information about possible containers configurations. As shown in Figure 6, a user will first select an image name stored in the database during a Docker file creation process. Then, he has access to a list of dependencies and libraries related to the image name selected. Regarding what is chosen in the second part of the process, the user will access possible configurations built and stored in the database. Also, the user can add custom configurations that are not in the system, but it has to be done in a particular way to avoid misconfigurations. The prototype does not store any created file for the container, and the user management database is still under development. So the prototype's primary purpose is to generate and downloadable Docker files, and it provides a platform to test a container generated from the Docker file. For user privacy, there is an automatic deletion of all containers and Docker files created. At this point, the actual database stores names of official images from Docker Hub, Dependencies, and libraries related to images and information about possible containers configurations. As shown in Figure 7, a user will first select an image name stored in the database during a Docker file creation process. Then, he has access to a list of dependencies and libraries related to the image name selected. Regarding what is chosen in the second part of the process, the user will access possible configurations built and stored in the database. Also, the user can add custom configurations that are not in the system, but it has to be done in a particular way to avoid misconfigurations.

C. Application of the Propose Framework on a Project

We explain how our proposed framework could help implement concrete a capstone originally developed in 2019 at ********* University. The research project was to implement

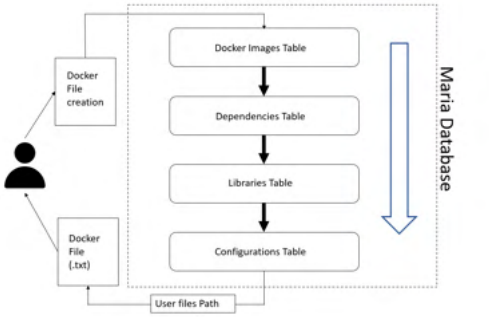

Fig. 7. Docker file Creation UML

a non-invasive computer vision for fall-detection using IoT and artificial intelligence. This project objective was to implement a prototype that could help detect older people's falls subject to different health conditions and prevent by SMS appropriate staff or caregivers on time. Thus, it can be a lifesaver technology for people that need the help of others to take care of themselves. They used multiple tools for the implementation of this project which are: (1) a raspberry pi connected to RapsCam for the data collection part; (2) a python script using sockets for data transmission that stream live videos from the Raspberry pi to a server; and (3) a OpenPose model to feed images and train the artificial intelligence model.

Our proposed framework could help save time during the implementation of such innovative ideas. We have the hardware and software necessary to implement most IoT projects, and the architecture used by the framework gives flexibility to researchers and students. For the implementation of this project, the group would use different processes when using our framework. The IoT clusters explain the above functional raspberry pis, and we propose a variety of sensors for local users. RaspCam is part of our inventory, as well as motion detectors. These devices would help save some financial resources for the group, and they could focus more on the software part of the research, which was the primary focus. The clusters communicate directly to the proposed cyberinfrastructure using python script and Websockets. It also offers a python script of sockets enabling the Raspberry Pis to transfer collected data to a server without much configuration.

D. Future Work

We are implementing ways to store containers and enable users to interact with safely within the framework. We are exploring ideas of creating virtual clusters to store containers belonging to the same user or same group of users. Also, users have limited interaction with containers created on the actual implementations, and the containers are run for testing purposes. The current prototype does not allow any further interaction. Thus, further development of the container interaction module will enable users to run containers and interact with them without leaving the framework or without a need to download them locally for advanced interactions. So, the CLI needs more functionalities resulting in getting more permission to modify and interact with the Docker machine from the Linux Server. We are looking for a safe way to allow those interactions without creating a bridge that can be exploited later for hacking purposes.

V. Conclusion

The recent development and standardization of microservices which have huge application possibilities, has allowed us to consider a framework that will help researchers and students in the difficulty encountered during the development, testing, and deployment phases of their research. The framework that we are proposing could help them accelerate their research process, and it will allow them to accomplish the tasks cited earlier with a certain simplicity. Docker provides flexibility and modularity for the construction of complex software systems. The fact that it is open-source cloud-oriented and facilitates code virtualization, the academic sphere should take advantage of it and develop applications that can help boost research and ease its difficulties. The implementation discussed is still under development, but the current prototype can help people with limited knowledge of Docker and help them create and run their own Docker containers with a certain simplicity. We recognize a need for further work to improve the database, user interaction, and repository. The architecture present in this paper will not substantially change and, we discussed improvements in future work.

References

[1] S. S. Mike Loukides, "Microservices adoption in 2020," Jul 2020. [Online]. Available: https://www.oreilly.com/radar/microservices-adoption-in-2020/

[2] R. H. K. I. of Technology, R. Heinrich, K. I. o. Technology, A. van Hoorn University of Stuttgart, A. v. Hoorn, U. o. Stuttgart, H. K. K. University, H. Knoche, K. University, F. L. S. AG, and et al., "Performance engineering for microservices: Research challenges and directions," Apr 2017.

[3] F. Osses, G. Márquez, and H. Astudillo, "Exploration of academic and industrial evidence about architectural tactics and patterns in microservices." New York, NY, USA: Association for Computing Machinery, 2018.

[4] L. Sun, Y. Li, and R. A. Memon, "An open iot framework based on microservices architecture," *China Communications*, vol. 14, no. 2, pp. 154–162, 2017.

[5] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*, 2016, pp. 1–5.

[6] C. MacKenzie, K. Laskey, F. Mccabe, P. Brown, and R. Metz, "Reference model for service oriented architecture 1.0," *Public Rev. Draft*, vol. 2, 08 2006.

[7] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[8] A. S. Gillis, "What are containers (container-based virtualization or containerization)?" Mar 2020. [Online]. Available: https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

[9] A. Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, "Exploring container virtualization in iot clouds," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, May 2016, pp. 1–6.

[10] C. Boettiger, "An introduction to docker for reproducible research," vol. 49, no. 1, p. 71–79, Jan. 2015. [Online]. Available: https://doi.org/10.1145/2723872.2723882