drones

Article

Software Aging Effects on Kubernetes in Container Orchestration Systems for Digital Twin Cloud Infrastructures of Urban Air Mobility

Jackson Costa [1], Rubens Matos [1,2,*], Jean Araujo [1,3], Jueying Li [4], Eunmi Choi [5], Tuan Anh Nguyen [4,6,*], Jae-Woo Lee [6,*] and Dugki Min [4,*]

1 Departamento de Computação, Universidade Federal de Sergipe, São Cristóvão 49100-000, Brazil
2 Coordenação de Redes de Computadores, Instituto Federal de Sergipe, Lagarto 49400-000, Brazil
3 UNAME Group, Universidade Federal do Agreste de Pernambuco, Garanhuns 55292-270, Brazil
4 Department of Computer Science and Engineering, College of Engineering, Konkuk University, Seoul 05029, Republic of Korea
5 School of Software, College of Computer Science, Kookmin University, Seoul 02707, Republic of Korea
6 Konkuk Aerospace Design-Airworthiness Institute (KADA), Konkuk University, Seoul 05029, Republic of Korea
* Correspondence: rubens.junior@ifs.edu.br (R.M.); anhnt2407@konkuk.ac.kr (T.A.N.); jwlee@konkuk.ac.kr (J.-W.L.); dkmin@konkuk.ac.kr (D.M.)

Abstract: It is necessary to develop a vehicle digital twin (DT) for urban air mobility (UAM) that uses an accurate, physics-based emulator to model the statics and dynamics of a vehicle. This is because the use of digital twins in the operation and control of UAM vehicles is essential for the UAM operational digital twin infrastructure (UAM-ODT). There are several issues that need to be addressed in this process: (i) the lack of digital twin engines for the digitalization (twinization) of the dynamics and control of UAM vehicles at the core of UAM-ODT systems; (ii) the lack of back-end system engineering in the development of UAM vehicle DTs; and (iii) the lack of fault-tolerant mechanisms for the DT cloud back-end system to run uninterrupted operations 24/7. On the other hand, software aging and rejuvenation are becoming increasingly important in a variety of computing scenarios as the demand for reliable and available services increases. With the increasing use of containerized systems, there is also a need for an orchestrator to support easy management and reduce operational costs. In this paper, an operational digital twin (ODT) of a typical urban air mobility (UAM) infrastructure is developed on a private cloud system based on `Kubernetes` using a proposed cloud-in-the-loop simulation approach. To ensure the ODT can provide uninterrupted operational control and services in UAM around the clock, we propose a methodology for investigating software aging in `Kubernetes`-based containerized clouds. We evaluate the behavior of `Kubernetes` software using the `Nginx` and `K3S` tools while they manage pods in an accelerated lifetime experiment. We continuously execute operations for creating and terminating pods, allowing us to observe the utilization of computing resources (e.g., CPU, memory, and I/O), the performance of the `Nginx` and `K3S` environments, and the response time of an application hosted in those environments. In some conditions and for specific metrics, such as virtual memory usage, we observed the effects of software aging, including a memory leak that is not fully cleared when the cluster is stopped. These issues could lead to system performance degradation and eventually compromise the reliability and availability of the system when it crashes due to memory space exhaustion or full utilization of swap space on the hard disk. This study helps with the deployment and maintenance of virtualized environments from the standpoint of system dependability in digital twin computing infrastructures where a large number of services are running under strict continuity requirements.

Keywords: operational digital twin; urban air mobility; cloud-in-the-loop simulation; software aging; software rejuvenation; `Kubernetes`; `Nginx`; `K3S`

check for updates

Citation: Costa, J.; Matos, R.; Araujo, J.; Li, J.; Choi, E.; Nguyen, T.A.; Lee, J.-W.; Min, D. Software Aging Effects on Kubernetes in Container Orchestration Systems for Digital Twin Cloud Infrastructures of Urban Air Mobility. *Drones* **2023**, *7*, 35. https://doi.org/10.3390/drones7010035

Academic Editors: Ivana Semanjski, Antonio Pratelli, Massimiliano Pieraccini, Silvio Semanjski, Massimiliano Petri and Sidharta Gautama

Received: 19 October 2022
Revised: 11 December 2022
Accepted: 22 December 2022
Published: 3 January 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Digital twin (DT) technology is a cutting-edge innovation that has the potential to revolutionize various industries. DT involves creating a virtual replica of a physical object or system, and using data-driven analysis and decision-making to continuously update and improve it. The virtual replica, or digital twin, is made up of computational models that evolve and change over time, reflecting the structure, behavior, and environment of the physical object or system they represent [1,2]. Digital twin systems are digital representations of physical systems, such as vehicles, buildings, or manufacturing processes. They are used to simulate the behavior and performance of the physical system, and to predict its behavior or performance under different conditions. This can be useful for a variety of applications, such as planning for maintenance, optimizing the operation of the physical system, or analyzing the impact of changes to the system's design or operation.

The development of an operational vehicle digital twin system for urban air mobility (UAM-ODT) includes the following fundamental modules: (i) neural digital twin dynamic engines (DTDE), (ii) neural digital twin control engines (DTCE), (iii) digital twin control frame (DTCF), and (iv) digital twin cloud infrastructure (DTCI) as shown in Figure 1. The DTDE module is responsible for creating a virtual replica of the aerodynamics of UAM vehicles using learning-based techniques. The DTCE module performs control tasks, such as robust control, optimal control, and adaptive control, to ensure the safety of the vehicle. These two modules digitalize the dynamics and control of the vehicle to ensure that the operations of the vehicle in the digital space are identical to those in the physical space. The DTCF module serves as a bridge between the digital twin and the physical twin of the vehicle. It can provide teleoperation services, fault-tolerant control, or traffic prediction and management, with the belief that if the dynamics and control of the physical vehicle are accurately captured in the digital space along with the digital environment (e.g., city, region, country), the operations in the digital space can be effectively transferred to the physical space. The DTCI module is the common computing platform that hosts the entire UAM-ODT system, running constantly to create a virtual space of the real-world UAM physical infrastructure. Due to the stringent requirements for the high availability of the digital twin system, the DTCI must handle any failures and maintain constant digital operations and services in the long run. Particularly, if a digital twin runs all day and night, it can be subject to a phenomenon known as "software aging". Software aging is the gradual deterioration of the performance and reliability of software over time, due to factors such as changes in the operating environment, errors and defects in the software, or the accumulation of wear and tear on the software. If a digital twin runs continuously, it can experience software aging more quickly than if it were run only intermittently. This can cause the digital twin to become less accurate and less reliable over time, which can affect the quality of the predictions and decisions it makes. In this work, we investigate the software aging problems in the digital twin cloud infrastructure which is developed upon `Kubernetes`-based cloud environment using a cloud-in-the-loop simulation approach.

Software aging is a phenomenon that occurs when software systems become less reliable and less efficient over time. This can happen for a variety of reasons, such as changes in the environment, changes in the software itself, or the accumulation of errors and defects. When software ages, it can become less accurate and less reliable, which can affect the performance and behavior of the systems that it is used to control or manage. The software aging phenomenon occurs in operating software systems, causing sudden failures such as crashes and continuous performance degradation, which can be circumvented by a proactive strategy such as software rejuvenation to avoid abrupt system interruptions [3,4]. The relevance of such a phenomenon is remarkable, considering that the demand for availability and reliability in the provision of services in practically all areas has increased in order to have quality and competitiveness in each field of activity. Considering high availability requirements, the services of computing, health, security, financial system, geolocation, and routing are examples that can be cited. In order to meet such service demands without the unwanted effects of software aging, it is necessary to use an architec-

ture capable of maintaining its offer without huge operational costs of employing several redundant servers with high computational power, requiring human resources for their handling and management, and also incurring higher energy costs. Using virtual machines in contexts such as these has been an alternative because they provide functionalities of a physical server based on the same traditional computational architecture. Thus, it is possible to create several virtual machines on a single server, and each virtual machine can run different environments allowing the execution of heterogeneous systems [5]. The scalability and flexibility of IT (Information Technology) can be increased through virtualization, in addition to generating significant savings in operational costs. Thus, IT administration becomes easier to manage by obtaining better availability, operability, performance, and greater workload mobility through virtualization [6].

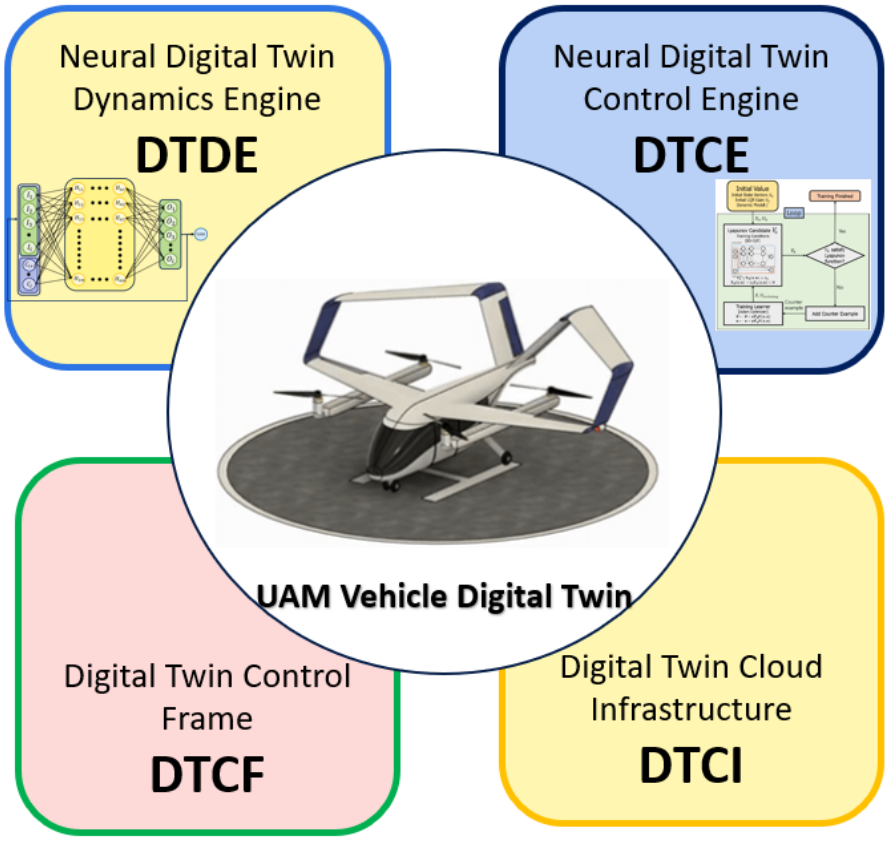


Figure 1. Operational Digital Twin for Urban Air Mobility (UAM-ODT).

If the flight control software of an unmanned aerial vehicle (UAV) experiences software aging, it can affect the performance and behavior of the UAV. As the software ages, it can become less accurate and less reliable, which can cause the UAV to behave in unexpected or unsafe ways. To address software aging in the flight control software of a UAV, it is important to periodically update and maintain the software. This can involve installing patches and updates, fixing errors and defects, and re-tuning or re-calibrating the software to account for changes in the environment or the UAV itself. Regular maintenance and updates can help to ensure that the flight control software remains accurate and reliable over time, and can help to prevent or mitigate the effects of software aging. In some cases, software aging can cause the flight control software to become unstable or unreliable. If this happens, it may be necessary to take the UAV out of service temporarily in order to perform maintenance or repairs. This can involve replacing or upgrading the flight control software, or making other changes to the UAV in order to improve its performance and reliability. So, software aging in the flight control software of a UAV can affect the performance and behavior of the UAV. To address this problem, it is important to periodically update and

maintain the flight control software, and to take the UAV out of service if necessary in order to perform maintenance or repairs. This can help to ensure that the UAV remains safe and reliable over time.

To create a digital twin, a mathematical model of the physical system is created using data about the system's behavior and performance. This model is then used to simulate the behavior of the physical system under different conditions, and to make predictions about its performance. In order to create a reliable and accurate digital twin, it is important to use accurate and reliable software to create the model and simulate the system's behavior. However, software aging can be a problem for digital twin systems. As the software used to create and simulate the digital twin ages, it can become less accurate and less reliable. This can affect the accuracy and reliability of the digital twin, and can cause it to produce incorrect or inconsistent predictions. In some cases, this can lead to incorrect or sub-optimal decisions or actions based on the digital twin's predictions. To address software aging problems in digital twin systems, it is important to periodically update and maintain the software used to create and simulate the digital twin. This can involve installing patches and updates, fixing errors and defects, and re-tuning or re-calibrating the software to account for changes in the environment or the system being modeled. Regular maintenance and updates can help to ensure that the digital twin remains accurate and reliable over time, and can help to prevent or mitigate the effects of software aging.

Digital twin systems can experience a variety of errors, depending on the specific characteristics of the system and the software being used. Some common types of errors that can occur in digital twin systems include:

- *Data errors:* Digital twin systems are typically based on data about the behavior and performance of the physical system being modeled. If the data are incorrect or inconsistent, it can cause errors in the digital twin. For example, if the data contain missing or invalid values, or if the data are not properly pre-processed or cleaned, it can affect the accuracy and reliability of the digital twin.
- *Modeling errors:* Digital twin systems are based on mathematical models of the physical system being modeled. If the model is incorrect or incomplete, it can cause errors in the digital twin. For example, if the model does not accurately represent the underlying physical principles or behaviors of the system, or if the model is not properly calibrated or validated, it can affect the accuracy and reliability of the digital twin.
- *Software errors:* Digital twin systems are implemented using software, and software can contain errors or defects. If the software used to create or simulate the digital twin contains errors, it can cause the digital twin to behave in unexpected or incorrect ways. For example, if the software contains bugs or syntax errors, or if the software is not properly designed or implemented, it can affect the accuracy and reliability of the digital twin.

Overall, digital twin systems can experience a variety of errors, including data errors, modeling errors, and software errors. To address these errors and improve the accuracy and reliability of the digital twin, it is important to carefully collect and pre-process the data, to create accurate and well-calibrated models, and to use high-quality software that is free of errors and defects.

When using virtualization, it is possible to implement many servers in a smaller number of hosts (physical servers), which consequently implies the gain of physical spaces and energy cost reduction. However, once the virtual machine is initialized, all the hardware on which the Operating System (OS) is running is loaded and not just a copy of the OS, resulting in the consumption of many system resources, making virtualization very expensive from a computational point of view [7]. The use of containerization mitigates the operational cost of traditional virtualization, as stated by [5], in which the author addresses host-level virtualization known as container, which is another type of virtualization. This type of virtualization acts on top of the physical server offering support to several independent systems since the physical server already has an OS installed, not needing to load all the host hardware or its copy. Container-based virtualization has recently gained

much attention [8,9]. This virtualization makes an application run efficiently in the most varied computing environments through its encapsulation and its dependencies [10]. This virtualization technique is said by the author of [11] to be lightweight, as the system significantly decreases workloads by sharing OS resources from host. Containers provide an isolated environment for system resources such as processes, file systems and networks to run at the host OS level, without having to run a Virtual Machine (VM) with its own OS on top of virtualized hardware. By sharing the same OS kernel, containers start much faster using a small amount of system memory compared to booting an entire virtualized OS like in [10] VMs. `Kubernetes` is a widely used tool for managing containers, configure, maintain and manage solutions that have containers as an approach to the detriment of VMs. Thus, this work aims to evaluate the effects of software aging and the performance of `Kubernetes` when undergoing a high-stress load, characterized by creating replicas of pods to maintain service availability in the `Nginx` and `K3S` environments. Furthermore, the aging problem on an unmanned vehicle refers to the degradation of the vehicle's performance over time, due to factors such as wear and tear, corrosion, and obsolescence. As an unmanned vehicle ages, its components may become less reliable and less capable of performing their intended functions, which can affect the vehicle's ability to operate safely and effectively. The aging problem can be particularly challenging for unmanned vehicles, as they often operate in harsh or hostile environments, and they may be subjected to high levels of stress and strain. For example, an unmanned aerial vehicle (UAV) may experience high levels of vibration and air turbulence during flight, which can cause its components to wear out faster. Similarly, an unmanned underwater vehicle (UUV) may be exposed to corrosive saltwater, which can cause its components to corrode and deteriorate over time. In this work, our focus is on the investigation of a digital twin cloud infrastructure in which a `Kubernetes`-based cloud environment is investigated regarding software aging phenomenon of the cloud if hosting the UAM-ODT with no downtime.

The study in this work extends the related research area on software aging in virtualized environment through the following *key contributions:*

- proposed a cloud based simulation platform with provisioning for the development of UAM-ODT infrastructures
- proposed a methodology for measurement and assessment of software aging in a container-based environment with a `Kubernetes` cluster in the digital twin cloud.
- performed comprehensive test-bed experiments and observations of software aging phenomena along with software rejuvenation in `Kubernetes` clusters based on `Minikube` and `K3S` environments.
- Findings and impacts:
  - It is important to stress that aging events found in test-bed experiments indicate the threats of system failures and performance degradation due to software aging symptoms. However, the time that those events will occur depends on the characteristics and intensity of the workload that the system needs to process, as well as the hardware and software specification of that `Kubernetes` system.
  - If the system has more resources available or less workload than those employed in this experiment, the aging phenomenon would be slower, and subsequently, the failures due to resource exhaustion would take longer to occur. This fact does not reduce the importance of evaluating software aging in those systems as well as planning actions for their mitigation.

To the best of our knowledge, this work contributes to the practical implementation and maintenance of virtualized environment on the perspectives of system dependability in digital twin computing infrastructures in which a huge amount of services are running with a stringent requirement of continuity. The findings of this study bring about the comprehension of software aging phenomena in digital twin computing infrastructures developed on top of `Kubernetes`, which is at very early stage of current research on software aging problems for a high level of dependability and fault-tolerance in digital twin computing infrastructures.

In order to facilitate the understanding of this work, the paper is organized as follows. Section 2 addresses the related works that inspired this study on software aging assessment; Section 3 presents the fundamental concepts and system design used in this work; Section 4 deals with the methodology used in the research; the objective and planning, covering the context in which it was produced, the tools selected, variables involved, scripts for reproduction and the hardware used are discussed in Section 5. The results are presented and discussed in Section 6. In Section 7 are the remarks arising from our research results.

2. Related Work

The work described in [10] analyzes the performance of running containers with services hosted on them, carrying out experiments with containers monitoring system resources, including network, memory, disk, and `CPU`. The testbed environment consists of a `Kubernetes` cluster manually deployed to carry out the evaluation, considering the Microsoft Azure Kubernetes Service (AKS), Google `Kubernetes` Engine (GKE), or Amazon Elastic Container Service for `Kubernetes` (EKS).

The authors in [12] evaluated the memory utilization, network overhead of containers, storage, and `CPU` using Docker, comparing them with KVM hypervisors. They exposed in their experiments that the containers obtained, in the worst case, similar or superior performance when compared to the VMs.

The work presented in [13] conducted a similar study, however, comparing the performance obtained from containers when monitoring the number of requests an application server could handle in relation to the same application deployed in a VM and the results showed that the VMs had significantly outperformed the containers.

The research reported in [14] performed application experiments for HPC (high-performance computing), using benchmarking tools to evaluate memory, network, disk, and `CPU` performance in Linux Container (LXC) related virtualization implementations, along with OpenVZ and Linux VServer, showing that all containerized apps performed similarly to a native system.

The authors of [15,16] showed improvements obtained related to performance isolation for MapReduce workloads. However, when evaluating disk workloads, LXC failed to fully isolate resources, opposite behavior to that of hypervisor-based systems.

Through memory, network, and disk metrics, the authors of [17] evaluated the performance of LXC, Docker, and KVM running many benchmarking tools to measure the performance of these components and concluded that the overhead caused by container-based virtualization technologies could have its weight considered irrelevant, despite the performance being compensated by safety.

Our main focus is on software aging investigation on a private cloud system hosting an operational digital twin of an eVTOL vehicle flying in a virtualized urban air mobility. Operational digital twins of vehicles in urban air mobility are digital representations of real-world vehicles that can be used for a variety of purposes. Some potential uses of operational digital twins in urban air mobility include:

- *Performance modeling and simulation:* Operational digital twins can be used to model and simulate the performance of vehicles in urban air mobility systems, including their flight dynamics, propulsion systems, and control systems. This can help to optimize the design and operation of vehicles, to improve their performance and efficiency, and to identify potential issues or risks.
- *Fleet management and maintenance:* Operational digital twins can be used to monitor the condition and performance of vehicles in real time, and to provide information about their current state and status. This can be used to support fleet management and maintenance operations, by providing timely and accurate data about the health and safety of vehicles, and by enabling proactive maintenance and repair.
- *Traffic management and control:* Operational digital twins can be used to support traffic management and control in urban air mobility systems, by providing information about the location, orientation, and velocity of vehicles. This can help to coordinate

the movement of vehicles, to avoid collisions and other hazards, and to optimize the flow of traffic in urban airspace.

- *Emergency response and rescue:* Operational digital twins can be used to support emergency response and rescue operations in urban air mobility systems, by providing real-time information about the location and status of vehicles. This can help to quickly and accurately identify the location and condition of vehicles in distress, and to coordinate rescue and recovery efforts.

Operational digital twins of vehicles in urban air mobility can be used for a variety of purposes, including performance modeling and simulation, fleet management and maintenance, traffic management and control, and emergency response and rescue. Due to such constant operational services, the UAM-ODT cloud system is inevitable to suffer software aging problems. In this study, we specifically investigate the software aging problems of a UAM-ODT cloud system based on `Kubernetes` virtualization environment.

3. System Design

3.1. Cloud in the Loop Simulation (CILS):

The ability to simulate a wide range of heterogeneous personal aerial vehicles (PAV) in the same virtual environment is critical and required to verify a variety of AI control algorithms even before their practical implementation on physical twin vehicles in digital twin infrastructures of future urban air mobility (UAM). One may deploy the AI control algorithms on actual vehicles and train them through practical flight testing in actual surroundings in order to improve the precision and calibre of neural network-based AI control algorithms (for example, neural Lyapunov control [18], deep reinforcement learning [19]). However, it frequently takes a lot of work and a considerable amount of time to collect enough flight test data for developing a competent AI control model for autonomous PAVs. As a result, one of the popular approaches is to develop an operational digital twin system for UAM (abbreviated as UAM-ODT) to replicate the actions of UAM vehicles in real-world settings within a shared virtual environment [20].

Inspired by the idea, we propose to adopt cloud-based solutions to develop a UAM-ODT system for a specific eVTOL PAV, called eVTOL KADA-UAM vehicle, under development by Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul, Republic of Korea as shown in Figure 2 and its virtual environment of UAM-ODT as shown in Figure 3.

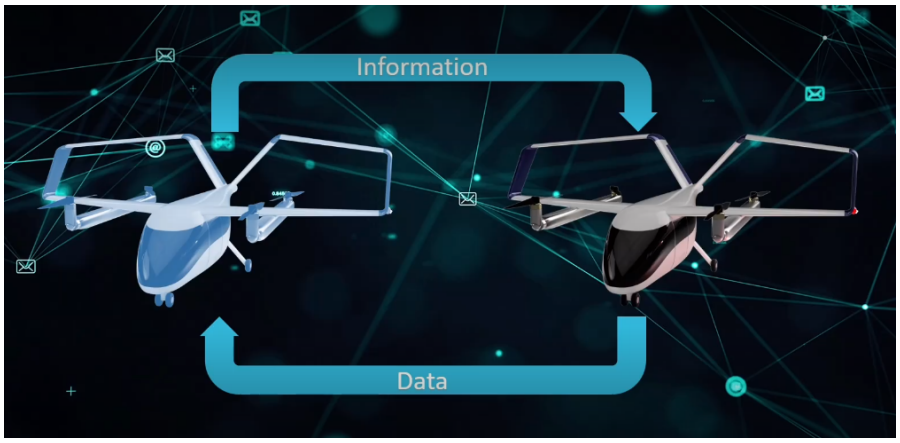
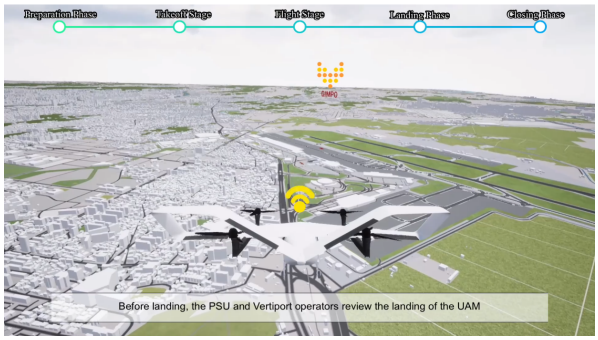


Figure 2. A digital replica of UAM vehicle in UAM-ODT infrastructure.

(**a**) In Flight



(**b**) Landing

(**c**) Vertiport

Figure 3. A visualization of UAM-ODT infrastructure. (The figures are excerpts from a video at https://blog.naver.com/yy8661 provided by Hyeon Jun Lee, Konkuk Aerospace Design-Trustworthiness Institute, Konkuk University, Seoul, Republic of Korea (rain9138@gmail.com)).

The proposal is the overall cloud-in-the-loop simulation (CILS) framework that can simulate the operations of a multitude of heterogeneous UAM vehicles with completely different aerodynamics in a UAM-ODT system, and thus can be used for verification and training of AI control algorithms in virtual world before practical implementation. The overall conceptual CILS architecture is designed as in Figure 4. To simulate multi-mode operations of heterogeneous environment which consists of multi-vehicles with different dynamics and configuration, we adopted the virtualization concept in cloud computing paradigm to separate a multitude of SILS processes onto different VMs. A single SILS process encompasses a PX4-based autopilot multi-mode AI control module (abbreviated as PX4) and a JSBsim based aerodynamic module called Konkuk Flight Simulation-Digital Twin Dynamics module (KFS-DT). The encapsulation of these two modules is called a dynamics-control SILS package, which is deployed on a multitude of VMs. The KFS-DT module guarantees the concept of digital twin framework for K-UAM vehicles in which it captures a high-fidelity CFD dynamics model of each physical vehicle. The autopilot control PX4 module transfer controls of each vehicle **u** to the dynamic module KFS-DT. The KFS-DT module computes new vehicle states **s** and returns them to the PX4 module.

The PX4 updates the current states **s** to a VM controller module. On the other hand, the PX4 receives updated sensor data from the VM controller module to generate new controls **u**. On each VM, there is a VM controller module to handle the data transactions between the VMs with a physical server for operational control management and for the visualization of the virtual environment (called environment control center). The VM controller module in each VM transmits vehicle states s receiving from a control PX4 module in the same VM and receives sensor data or mission data to/from the visualization center using Airsim [21] for AI application and Unity™for visualization. While an environment controller module in the control center is designed to handle the operations of all vehicles in the simulated environment upon the ground control module, AI module, Airsim server and Unity client module. The scalability of this cloud-based simulation framework is guaranteed by an auto-provisioning cloud system.

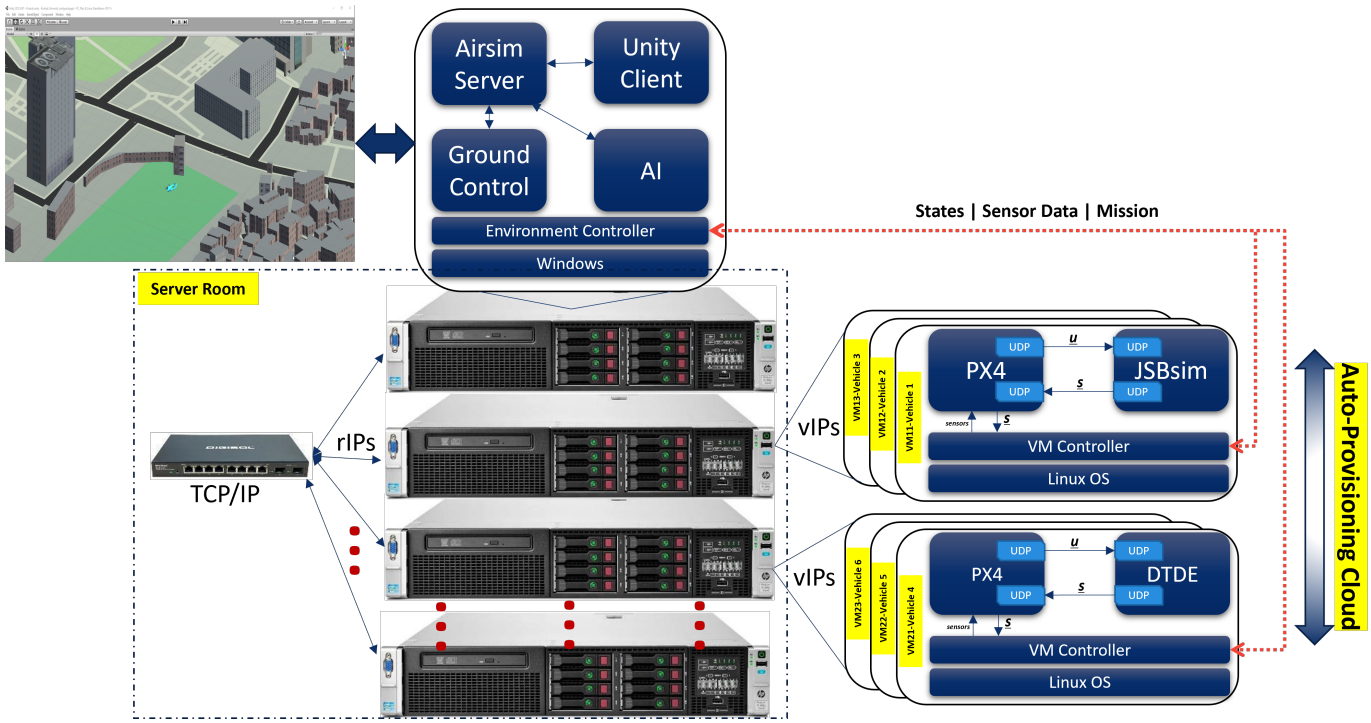


Figure 4. Cloud in the loop simulation framework.

In this work, our focus is on the digital twin version of an eVTOL vehicle to capture flight operations in an urban air mobility for further studies on performance modeling and simulation of vehicle, fleet management and maintenance, traffic management and control, and emergency response and rescue. Thus, the detailed design of the overall cloud in the loop simulation framework running on a private cloud platform is presented in a comprehensive manner while the detailed design of the eVTOL vehicle in consideration regarding circuit and IT problem in the individual engine control systems is out of scope of the study. We consider how the dynamics of the vehicle and its corresponding control are simulated in a virtualized environment of urban air mobility to mimic the real-world flight operations for air traffic managements in urban areas.

3.2. Cloud Provisioning Hardware System:

The hardware infrastructure of CILS is designed as shown in Figure 5, consists of main two components: one is virtual cluster (VC) disk image provisioning, and the other is auto provisioning virtual instance creator. Virtual machines existing in the same VC subgroup can be used by sharing the virtual disk image with homogeneous S/W as read-only. The numerous existing cloud systems provides GUI where users can build instances. It seems that this function simplifies the manipulation of creating instances on the cloud

system whereas it just repeats useless operation to prepare requisites and build VMs. The auto provisioning virtual instance creator based on infrastructure as a code (IaC) provides a consistent CLI workflow to manage hundreds of cloud services and customize simulation environments.

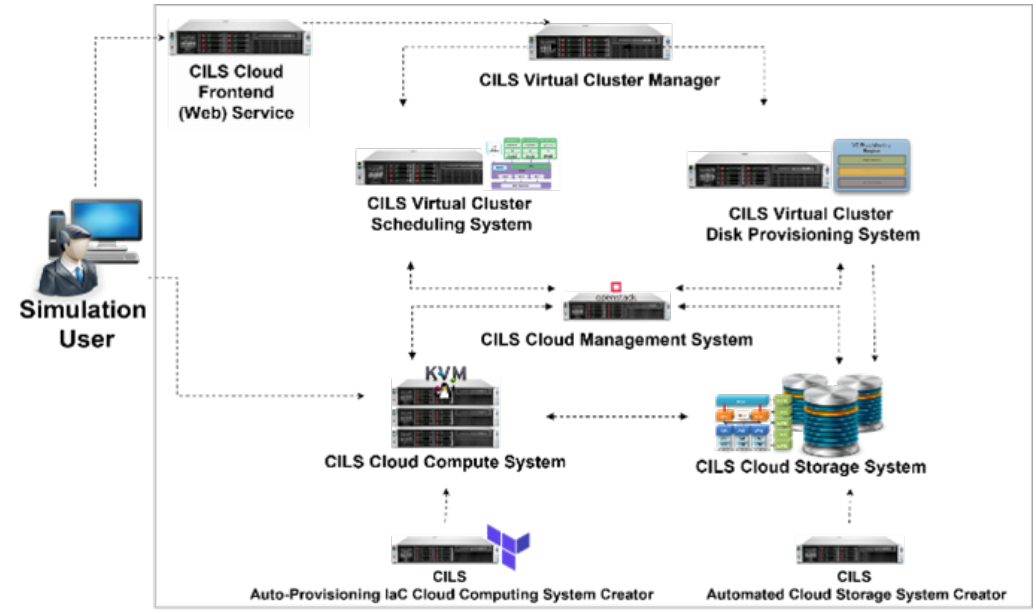


Figure 5. Cloud provisioning hardware system architecture.

Figure 6 shows the provisioning technology for virtual cluster images. The VC disk image provisioning based on union mounting technique can integrates one instance with diversified simulation virtual disk images depending on the simulation requirement such as PX4-Autopilot, JSBSim, Airsim, FlightGear and so on. Due to the orchestration of the cloud management, it can implement the communication via layer 2 or layer 3 between instances. Thanks to the capabilities of cloud provisioning and orchestration as designed in Figures 5 and 6, the UAM management can be maintained and stored on CILS Cloud Compute System complying the CILS framework in Figure 3.

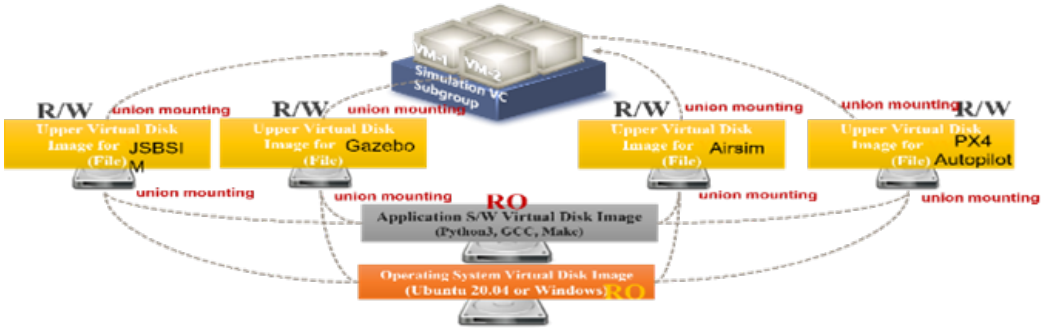


Figure 6. Virtual cluster image provisioning technology.

3.3. Software Aging and Rejuvenation

For urban air mobility (UAM), it is necessary to create a vehicle digital twin (DT) that uses a precise, physics-based emulator to characterise a vehicle's statics and dynamics. As a result, the UAM operational digital twin infrastructures need the deployment of the digital twin in vehicle operations and control (UAM-ODT). The problems are, (i) the absence of digital twin engines for the digitalization (twinization) of dynamics and control of UAM vehicles running at the core of UAM-ODT systems; (ii) the absence of back-end system engineering in the development of UAM vehicles; and (iii) the absence of fault-tolerant mechanisms for the DT cloud back-end systems running 24/7 uninterrupted operations.

Unmanned vehicles, also known as drones, are a relatively new technology and there are still many challenges and limitations associated with their use. One of the main causes of errors in the management of unmanned vehicles is the lack of a reliable and robust communication system. This can make it difficult for the operator to control the drone and receive accurate information about its location and status. Additionally, the complex algorithms used to control the drone's movements can sometimes produce unexpected or unpredictable behavior, leading to errors in the management of the vehicle. Other potential causes of errors in the management of unmanned vehicles include software bugs, hardware malfunctions, and interference from external sources such as radio waves or other electromagnetic signals. Software aging refers to the gradual degradation of a software system's performance over time. In the context of unmanned vehicles, software aging can be caused by a number of factors, including the accumulation of data, changes in the operating environment, and the introduction of new features or updates. As the software continues to be used, it may become slower, less reliable, and more prone to errors. This can affect the performance of the unmanned vehicle and make it more difficult to manage. Other potential causes of software aging in the management of unmanned vehicles include the use of outdated or inefficient algorithms, inadequate testing and debugging, and the lack of proper maintenance and support.

Software aging and rejuvenation has been an active line of research since 1995 when it was proposed by Huang et al., then at AT&T Bell Labs [22]. The reasons that lead to software aging include data loss, accumulated operating system error, resource consumption, and sudden crashes, for example. These phenomena, which accumulate gradually over time, can lead to software performance degradation, which can lead to a sudden crash or shutdown of software systems [23]. A fault tolerance prevention strategy, called software rejuvenation, aims at circumventing the negative effects of software aging, thus making it an important issue for systems reliability by avoiding sudden system failures caused by software aging, providing security and availability [24,25]. Companies such as Amazon and Google have increased interest in adopting technology architecture based on microservices (which usually rely on containerization) [26]. The reason for such an adoption is that application systems based on microservices architecture have the advantage of being easier to develop, deploy, and scale compared to monolithic architecture systems [27]. Containerization systems allow the configuration of the environment for software deployment in the shortest possible time, solving problems of integration of the most diverse applications [28].

When using containers, the application code in any offered service is involved in containerization along with its libraries, all dependencies, and configuration files necessary for its execution in the most diverse types of environments. This containerization becomes autonomous and portable, as it is abstracted from the host OS and can be executed on any computing platform [29]. This approach has been widely applied in the computing industry and demonstrated in several studies. Refs. [12,17] report overall cost reduction and overall application performance optimization in containers. The applications' microservices are generated by dividing them into small units and independently, increasing the scalability and portability of the services and the containers [10]. Nonetheless, the increase in the use of containers implies the need for tools capable of managing, through the control of tasks such as the operation of applications in containers throughout the infrastructure, scaling, and automation of application deployment [30]. An example of a container orchestration tool that is increasingly needed and widespread is `Kubernetes`, open-source and made available by Google. Container management tools are at the peak of expectations in the Hype Cycle for Cloud Computing from Gartner [31].

Such expectations give signs that the field in container orchestration technologies is on the rise, attractive, and very competitive, and should continue at an increasing pace as several organizations consider adopting the container-based approach [9,32,33]. A fair amount of tools as solutions for the execution and orchestration of containers emerged and quickly became solution standards in this context, among them Docker and `Kubernetes`.

They enable the creation of new containers and pods (a computational unit in `Kubernetes` comprising one or more containers) with their deployments in an agile way when an increase in application workload is detected, or even a pod drops due to excessive consumption of its resources, such as memory or `CPU` (Central Processing Unit), through monitoring [34]. However, its various components and related complexity have a very costly learning curve, which may not be easy to manage even with its proven efficiency in scaling, configuring, and maintaining services. Therefore, managing a `Kubernetes` infrastructure is a complex task. This has given rise to a new market for managing Containers, such as hosted `Kubernetes` solutions.

4. Methodology

The methodology adopted in this work followed the flow shown in Figure 7, which in summary is based on an experimental evaluation applied by measuring the use of system resources and performance in a container-based environment with a `Kubernetes` cluster. The evaluation was carried out in different scenarios using the `Nginx` or `K3S` tool to manage the cluster.

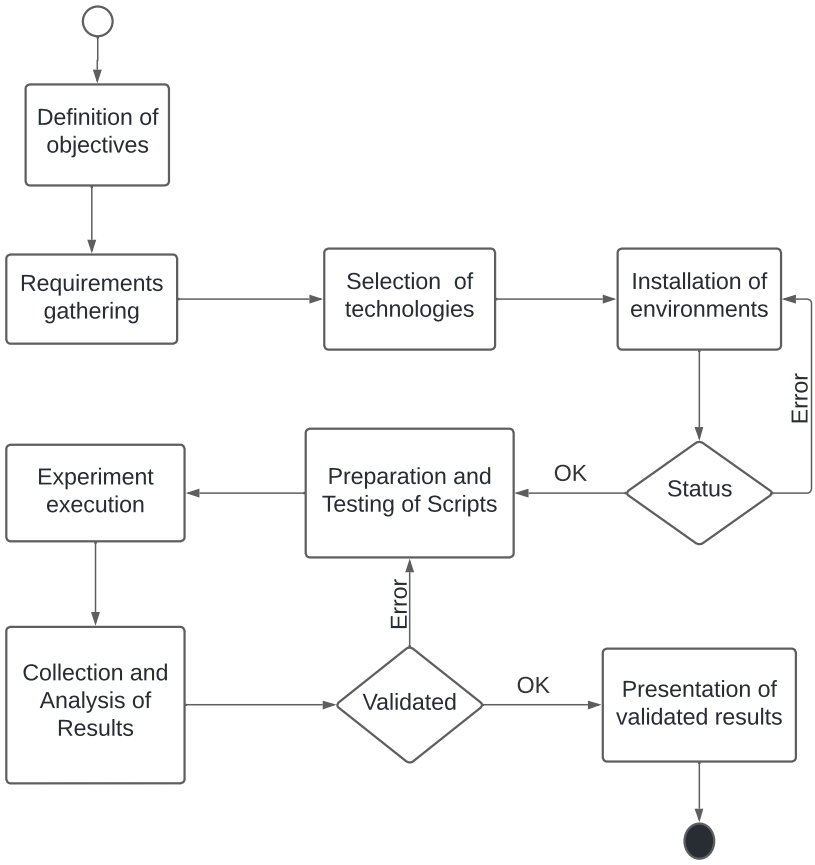


Figure 7. Methodology of the software aging measurement and assessment in `Kubernetes` environment.

Kubernetes is an open-source platform for managing and orchestrating containerized applications. It allows you to deploy and manage multiple containers, such as those created with Docker, across a cluster of machines, and it provides many features and tools to help you automate, scale, and manage applications and their dependencies. One of the main benefits of Kubernetes is that it helps to simplify and automate the process of deploying and managing applications in a distributed environment. This can save time and effort, and it can help to improve the reliability and scalability of applications. Additionally, Kubernetes provides many features and tools that can help you manage and monitor applications, such as: (i) Service discovery and load balancing: Kubernetes can automatically assign unique IP addresses to each of containers, and it can automatically

distribute incoming traffic across the containers in cluster. (ii) Configuration management: Kubernetes allows you to define application's configuration in a declarative manner, using YAML files or other configuration formats. This can make it easier to manage and update the configuration of applications. (iii) Health checking: Kubernetes can monitor the health of containers and applications, and it can automatically restart or replace containers that are not functioning properly. (iv) Self-healing: Kubernetes can automatically detect and recover from failures in application, such as when a container crashes or when a node in cluster goes down. Therefore, he benefits of Kubernetes include improved automation, scalability, and reliability for applications, as well as a rich set of features and tools for managing and monitoring applications in a distributed environment [35].

In this research, the experiments were carried out using scripts developed for these scenarios to simulate a service's distribution using a `Kubernetes` cluster, which can be accessed externally through the Internet, receiving a high-stress load by performing requests. We have also developed scripts to monitor software aging metrics, such as `CPU` utilization, memory consumption, and disk utilization, among others, in order to measure the performance of a service hosted in `Kubernetes` by checking the time of requests correctly fulfilled.

The environments of `Nginx` and `K3S` adopted in this experimental evaluation are composed of a cluster containing `5` Pods and `1` Service—that allows communication between the Pods. One of the Pods was configured as a Deployment of an `Nginx` web server, which enabled testing the performance of an application hosted in `Kubernetes`, responding to user requests from anywhere connected to the Internet.

`CPU` utilization, memory consumption, disk utilization, and total response time were some of the metrics used for this study, based on the metrics used in [10,14]. The results of these measures were captured by scripts developed for this purpose and, finally, evaluated through analysis of their behavior.

The proposed methodology actually can be applied for typical operational digital twin version of heterogeneous UAM vehicles including rotary aircrafts such as drones or helicopters, fixed-wing aircrafts or hybrid aircrafts such as eVTOL vehicles. Since the UAM-ODT platform is designed to run on a private cloud computing system based on cloud-in-the-loop simulation paradigm with heterogeneous digital twin modules of dynamics and controls as shown in Figure 4.

5. Experimental Planning

5.1. Goal Definition

To guarantee the mitigation of software aging emergence in the UAM-ODT platform with proper operational management and maintenance, this work presents a developed methodology for the investigation of software aging phenomenon by measuring the use of system resources and performance. We use different tools for the measurement in different experiments. The objective of this work was formally defined when using the Goal Question Metric (`GQM`) method [36] to verify the emergence of the effects of Software Aging as well as the performance of the `Kubernetes` Cluster in `Minikube` and `K3S` through the response time of the service when responding to requests.

5.2. Planning

The independent variables in the experiment are the number of simultaneous pod replication requests made to the service, overloading the `Nginx` application server, and emulating `Kubernetes`' autoscaling so that the service continues to be available under high workloads. The dependent variables were: `CPU` utilization, memory consumption, disk utilization, and average response time to requests.

Following the `GQM` method, the following research questions were designed to broadly cover the scope of this work:

- `Q1?:` Have indicators of software aging been found?

- `Q2?`: Which environment had the best performance in controlling the consumption of resources related to software aging?
- `Q3?`: Was there a similarity in behavior between the results obtained with `Minikube` and `K3S`?

In order to answer `Q1` and `Q2`, the following metrics were evaluated: `CPU` utilization, memory consumption, and disk utilization. In order to answer `Q3`, we conducted a comparative analysis of the results obtained from metrics that have been used to answer `Q1` and `Q2`.

5.3. Object Selection

Samples of `125` h of monitoring in the `Minikube` environment and `95` h in the `K3S` environment were considered to evaluate the system's performance and verify the aging effects.

5.4. Experimental Design

The following steps were developed for the execution of the experiment:

- `Step 1`: Survey of the requirements for its realization in the `Minikube` and `K3S` environment.
- `Step 2`: Development and analysis of monitoring scripts, execution of the environment and its stress.
- `Step 3`: The experiment execution script, both in `Minikube` and in `K3S`, followed the following general script:
  - `Step 3.a`: Execute the monitoring script for `2` h without any workload before the cluster is started.
  - `Step 3.b`: Run script that starts the cluster with the container orchestrator and keeps monitoring for initial `2` h without stress.
  - `Step 3.c`: Run the high workload emulating the auto-scaling `420` times in a loop.
  - `Step 3.d`: After the end of stress, wait `2` h and execute a script that ends the container orchestrator as a possible software rejuvenation action.
  - Repeat steps `3.a`, `3.b`, `3.c`, and `3.d` until completing `five` cycles.
- `Step 4`: Generate graphs of the results obtained and analyze them.

To reinforce the understanding of our experiment, Figure 8 depicts a diagram that represents the sequence of operations performed by the general script we just described.

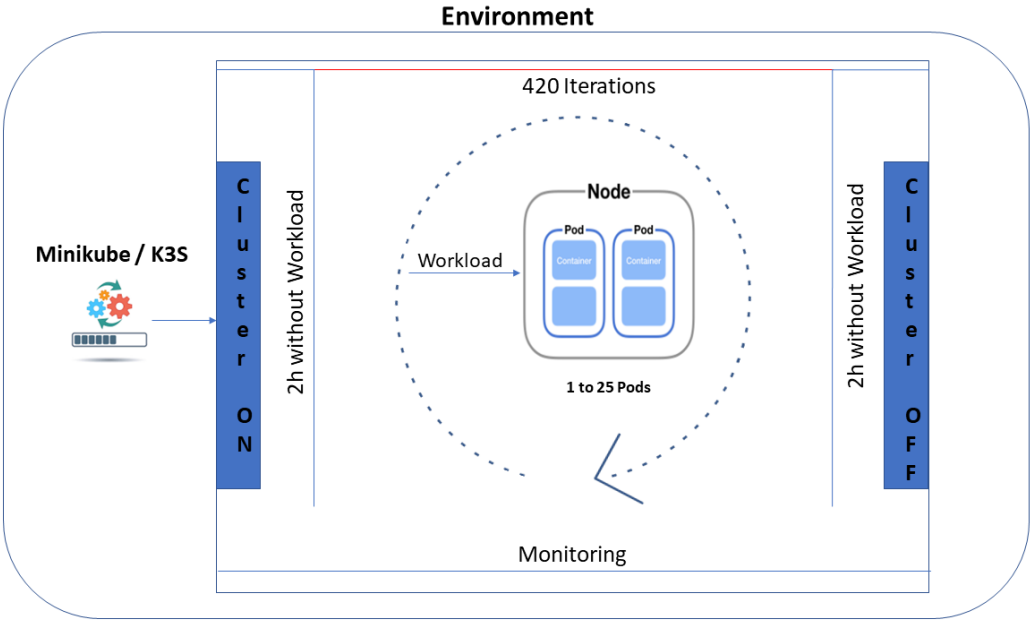


Figure 8. Diagram for cycles of operations performed by the experiment script.

Throughout all routines in `step 3`, another script sends client requests to the service that is hosted in the cluster. Those requests are effectively serviced by any pods that might have been created throughout the stress workload. Figure 9 illustrates the interaction between a client and the service in the `Kubernetes` cluster both in the `Minikube` environment and in the `K3S` environment, in both, the infrastructure architecture is configured as in Figure 2, which defines a logical set of Pods and enables exposure external traffic, load balancing and service discovery for these Pods, which have `Nginx` as a lightweight HTTP server, which is represented with Other App.

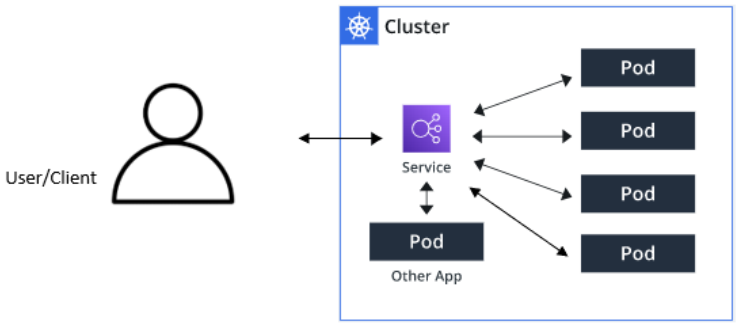


Figure 9. Cluster and Client Interaction Overview.

5.5. Instrumentation

The hardware used in this experiment was: a host with `8 GB` of RAM, a Core `i3` processor with a `3.1 GHz` clock, a WiFi module, and Ubuntu Linux OS version `20.04 64` bits. The software used were: Shell script, for experiment implementation, monitoring and collection of data generated as results through the bash command interpreter; `K3S` version `v1.22.5+k3s1`; `Minikube` version `1.15.1` commit: `23f40a012abb52eff365ff99a709501a61ac5876`; `Kubernetes v1.19.4` on Docker `19.03.13` for running the `Kubernetes` Cluster and Pods.

Metrics were collected with an interval of `60 s` for monitoring `CPU` utilization and memory consumption, while for the disk usage metric the interval was `5 s`, which we considered necessary to have enough samples, avoiding interference from monitoring activity. on actual system performance.

6. Experimental Results

In this section, the results collected from the experiment will be presented for both `Minikube` and `K3S` environments, considering the metrics of `CPU` utilization, memory consumption, disk utilization, and, finally, the requests made to the service. Each metric result is described in the following subsections. These are metrics for continuity and performance of the UAM-ODT cloud infrastructure. The data were collected from the cloud infrastructure rather than from the vehicle. The reason is we are investigating the software aging problems in a private cloud to host 24/7/365 operational digital twin services for UAM management.

It is worth highlighting that the experiments' total time differs in `Minikube` and `K3S` due to a difference in the average time to restart the pods within the auto-scaling process. This information was also measured and is presented in Table 1, showing the fastest execution of this action in the `K3S` environment, `25.4%` faster than `Minikube`, evidencing an improved efficiency in auto-scaling of `K3S` when compared to `Minikube`.

Table 1. Average Pod Reset Time.

Environment	Time (s)
`Minikube`	`97.56`
`K3S`	`72.80`

6.1. CPU Utilization

In the CPU utilization evaluation, data were collected from the following specific metrics: USR, which is the percentage of CPU used by the task during execution at the user level; SYS, which is the percentage of CPU used by the task during execution at the kernel level of the OS; WAIT is the percentage of CPU spent by the task while waiting to be executed; and finally, the CPU_TOTAL, which is the total percentage of CPU time used by the task monitored by Pidstat tool, which provides statistics report for the tasks on GNU/Linux systems.

Figure 10 shows a peak of 180% of CPU_TOTAL during the initialization of the Cluster, but with an average slightly above 100% during the entire experiment in the Minikube environment. It is also possible to notice in the graph a controlled behavior within Minikube about the metrics limits since the limit is only exceeded when starting the environment. Notice also that values of utilization higher than 100% in this context are related to the usage of more than one core of the processor by this process.

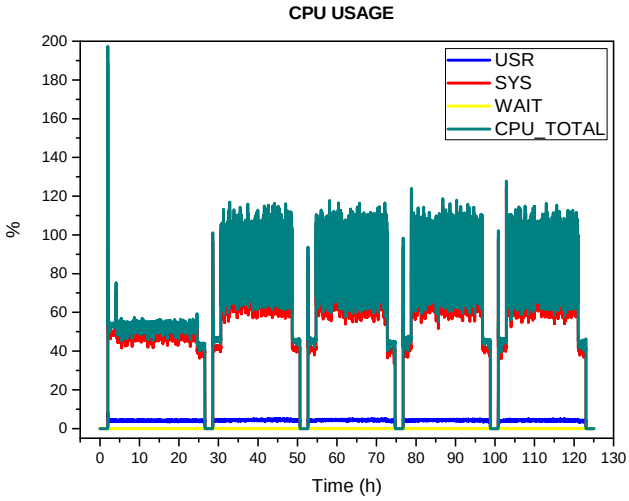


Figure 10. CPU utilization in Minikube.

Figure 11 shows a different behavior of K3S about Minikube regarding CPU utilization when we look at the CPU_TOTAL metric, which, unlike Minikube, it shows an increase in CPU_TOTAL utilization together with the USR metric over time, being interrupted when applying the cluster termination, which seems to act as a software rejuvenation technique for this situation. Although, during the entire experiment, the CPU_TOTAL did not exceed 60%.

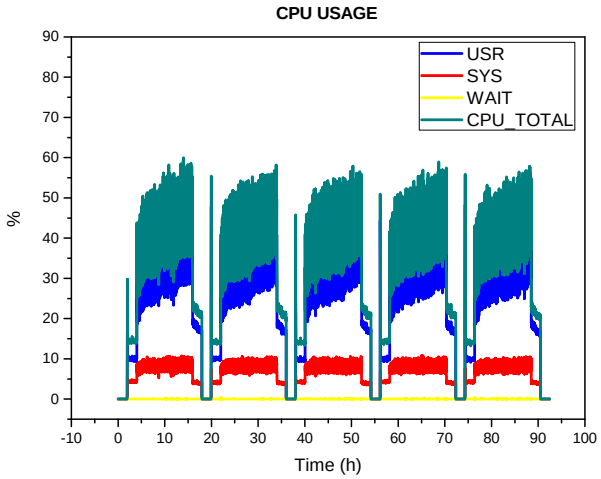


Figure 11. CPU utilization on K3S.

6.2. Disk-Related Metrics

In the evaluation of disk-related metrics, data were collected for the following metrics: `READ`, which represents the amount of kilobytes per second that the task took to be read; `WRITE`, which is the amount of kilobytes per second that the task sent to be written to the disk; and finally `CANCELLED`, which is the amount of kilobytes per second whose disk writing was canceled by the task, that can also occur when the task truncates some dirty page cache. All these metrics were monitored by the `Pidstat` tool in both `Minikube` and `K3S`.

In Figure 12, the `WRITE` and `CANCELLED` metrics have their behavior unchanged throughout the experiment, always walking close to `0 KB/s`. Although, the `READ` metric had a distinct behavior, holding the same value throughout a single cycle of the cluster stress, and presenting a linear growth among cycles until the fourth execution cycle, being interrupted abruptly when reaching about `4,000,000 KB/s` due to the limiting factor of the `Minikube` environment. Such behavior may be indicative of the software aging phenomenon in this environment.

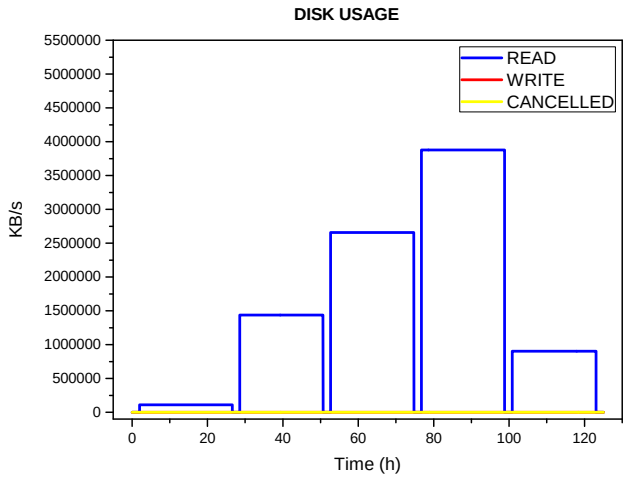


Figure 12. Disk-related metrics in `Minikube`.

In Figure 13, the `WRITE` and `CANCELLED` metrics also remain close to `0` KB/s throughout the experiment, similar to the execution in the `Minikube` environment. However, the behavior is different in the `READ` metric, which was not interrupted abruptly and had a linear growth from one cycle to another until the end of the experiment execution in `K3S`. It is important to mention that `K3S` presented smaller values of bytes read per second than `Minikube`, which might have prevented it from the abrupt fall observed there.

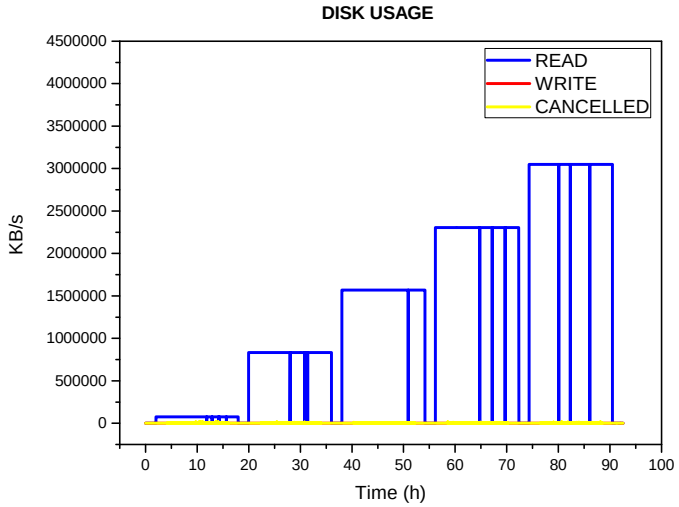


Figure 13. Disk usage in `K3S`.

6.3. Memory Usage

In the evaluation of memory consumption, data were collected for the metrics: MEM_USED, which represents the calculation of the total memory used; MEM_FREE, which is the memory that is not being used; MEM_AVAILABLE, which estimates how much memory is available to start new applications without swapping (it may include memory space that is being used for buffers or cache); MEM_SHARED, which is the memory mainly used by TMPFS which is the file system that keeps all files in virtual memory; MEM_BUFFERS_CACHED, which is the sum of the memory buffers and cache; SWAP_USED and SWAP_FREE metric, which represent respectively the used and free amount of virtual memory's swap space, that allows the system to use a part of the hard disk as physical memory. All these metrics were monitored using the "free" tool in both the Minikube and K3S environments.

In the evaluation of memory utilization in Minikube, the MEM_USED metric in Figure 14 has its behavior mirrored with that of the MEM_AVAILABLE metric, while the MEM_USED increases throughout the experiment, the MEM_AVAILABLE decreases in an inversely proportional trend. MEM_USED has a consumption increase of around 70% at the end of the experiment, even applying rejuvenation (i.e., cluster termination and restart between cycles). Such an action drops the memory usage temporarily, but when the cluster is started again, the system restores the same memory usage level observed at the end of the previous cycle. The MEM_FREE metric has a drop close to 48%. The MEM_BUFFERS_CACHED metric has a drop of around 41%. The SWAP_USED metric also behaves inversely to the SWAP_FREE metric, while the SWAP_USED has a 20% increase at the end of the experiment and SWAP_FREE a drop of 11%. The MEM_SHARED metric in both Minikube and K3S behave similarly, maintaining a regularity between 48 to 179 MB of consumption.

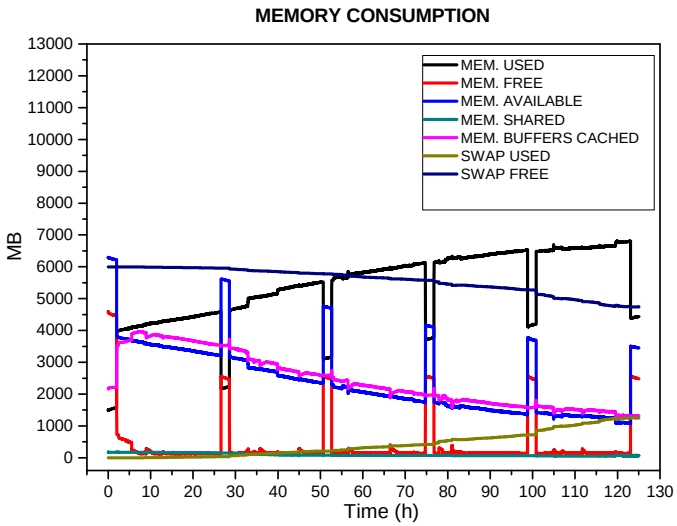


Figure 14. Memory consumption in Minikube.

In the evaluation of memory utilization in K3S, the MEM_USED metric in Figure 15 showed behavior similar to that observed in Minikube. MEM_USED has a consumption increase of around 61% at the end of the experiment, even applying rejuvenation. The MEM_FREE metric has a decrease of close to 79%. MEM_BUFFERS_CACHED has an increase of around 12%, which differs from the behavior in Minikube. The SWAP_USED has an increase of 8% when it reaches the end of the experiment and the SWAP_FREE a decrease of 8.5%.

For these memory consumption metrics, both in Minikube and in K3S, linear regression calculations on MEM_USED were performed to estimate the moment when the system would reach its upper limit for RAM usage, which in these cases is 8 GB. To confirm that estimate, we also computed the linear regression for MEM_FREE, which is another way to indicate the exhaustion of the resource, leading to system downtime and, consequently, the interruption of service provision. Similar regression estimates were carried out for the swap space usage.

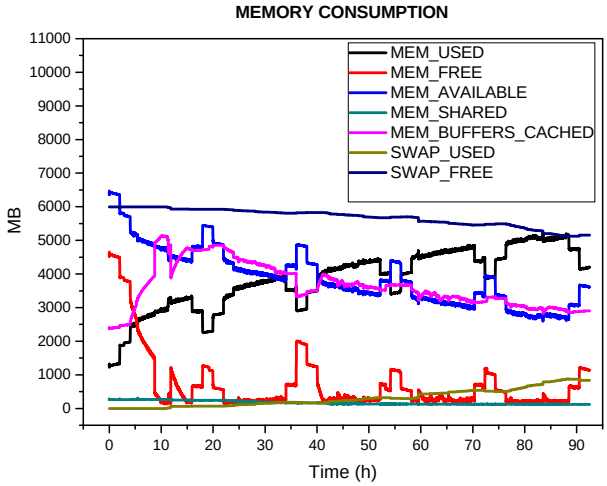


Figure 15. Memory consumption on K3S.

Equation (1) of the linear regression was obtained for the MEM_USED metric in the Minikube environment (MU_Minikube), shown in Figure 14. From this equation, it is possible to observe, as a function of MU_Minikube, that the 8 GB limit is reached after 170 h (i.e., 7 days and 2 h) of continuous execution of the workload used in the experiment. For the SWAP_USED metric, also exposed in Figure 14 for the Minikube environment, the linear regression Equation (2) was obtained.

$$MU_{Minikube} = 3900.84 + 23.98072 \times T_{stress} \quad (1)$$

In Equation (2), it is possible to observe that the upper limit of the SWAP_USED metric of the Nginx environment (SU_Minikube), which in this case is 5.8 GB, is reached after approximately 551 h of experiment, or 22 days of the same, so that the resource was completely exhausted.

$$SU_{Minikube} = -221.43413 + 10.37255 \times T_{stress} \quad (2)$$

For the MEM_USED metric of the K3S environment (MU_K3S), the linear regression Equation (3) was obtained which, through it, it is possible to observe that the upper limit of 8GB of resource for the MEM_USED metric is reached after 187 h (i.e., 7 days and 8 h) of workload execution.

$$MU_{K3S} = 2482.70 + 29.67105 \times T_{stress} \quad (3)$$

Finally, the SWAP_USED metric of the K3S environment (SU_K3S) had the linear regression Equation (4) obtained, which allows the visualization of resource exhaustion, which has a total of 5.8GB, after 603 h (i.e., 25 days and 1 h) of workload performed in the experiment.

$$SU_{K3S} = -120.24857 + 9.30782 \times T_{stress} \quad (4)$$

6.4. Evaluation and Discussions

When evaluating the results presented in Figures 10 and 11, it can be seen that most of the CPU consumption happens through the USR metric in the K3S environment, while the SYS metric does the highest consumption in the Nginx environment. This growth behavior of the USR metric in K3S was recurrent even after applying Software Rejuvenation every cycle, unlike the Nginx environment that maintains stability in the consumption of its CPU utilization metrics.

The results presented in Figures 12 and 13 show similar behavior in the use of disk usage metrics in the K3S and Nginx environment, differing only that in Nginx, the READ metric presents an interruption when it reaches 4,000,000 KB/s, returning in the fifth cycle with a total utilization close to 10%. In the K3S environment, this READ metric does

not suffer an interruption but presents a linear growth from one workload cycle to another. In both scenarios, the READ metric generally presents a linear growth representing a greater need for reading from disk at each cycle.

Figures 14 and 15 show similar behavior related to memory consumption metrics in the Nginx and K3S environments, respectively. In both, there is a linear growth of the MEM_USED metric and in the SWAP_USED metric, and the opposite behavior of the MEM_AVAILABLE and SWAP_FREE metrics. Thus, with Equations (1) and (2) obtained from linear regression, it is possible to glimpse the effects arising from software aging related to memory consumption even after the application of a potential software rejuvenation action, that is, the cluster termination.

The results presented in this section are the observation of software aging phenomenon for a private cloud system hosting a UAM-ODT platform for UAM management. It is crucial to emphasise that those findings point to the dangers of system breakdowns and performance declines brought on by signs of software ageing. However, the timing of those events relies on the nature and volume of the workload that the system must handle, in addition to the hardware and software requirements of particular Kubernetes system. The ageing phenomena would be delayed and the failures caused by resource exhaustion would follow if the system had more resources available or a lighter burden than that used in this experiment. This reality does not lessen the significance of assessing the software ageing in those systems and organising countermeasures. Evaluating these scenarios using other software rejuvenation approaches and complementary metrics related to software aging are the most promising steps that could be taken in future work.

Regarding how to avoid the observed software aging phenomenon in the UAM-ODT infrastructure, in general, there are several strategies that can be used to avoid or mitigate software aging. These can include: (i) regularly updating and patching the software to fix bugs and security vulnerabilities; (ii) monitoring the performance of the software and identifying potential problems before they occur; (iii) implementing automation and management tools to help manage the software and its dependencies; (iv) using modular, microservice-based architectures to make it easier to update and maintain individual components of the system; (v) using containerization technologies, such as Docker, to package the software and its dependencies into a self-contained environment that can be easily deployed and managed. These are just some examples of strategies that can be used to avoid software aging in a cloud system. Currently, the technique to avoid software aging is monitoring the performance of the software and identifying potential problems before they occur. Further investigation on how to adopt the software rejuvenation techniques in optimal and automatic manner will be an interesting extension for research into the UAM-ODT system in which the services for UAM management using ODT are constant and at zero downtime.

7. Conclusions

This paper presented a comprehensive study on the effects of software aging problems on Kubernetes in container orchestration system in a digital twin cloud infrastructure for UAM-ODT systems. The behaviours of Kubernetes software were analysed in an accelerated lifespan experiment utilising both Nginx and K3S tools. The operations for establishing and terminating pods were carried out in real time, allowing us to monitor the usage of computational resources (such as CPU, memory, and I/O), the performance of the Nginx and K3S environments, and the response time of an application hosted in those environments. In particular settings and for specific metrics, such as virtual memory utilisation, software ageing effects were detected, indicating a memory leak that is not entirely cleansed when the cluster is halted. The study's findings help to understand the phenomenon of software ageing in digital twin computing infrastructures built on Kubernetes, which is at the very beginning of current research on software ageing issues for highly reliable and fault-tolerant digital twin computing infrastructures.

Author Contributions: Conceptualization, methodology, and supervision, R.M.; project administration, formal analysis and investigation, R.M., J.A., J.-W.L., D.M. and T.A.N.; resources, funding acquisition and investigation, E.C., J.-W.L. and D.M.; validation, J.C., J.L.; writing—original draft, J.C., R.M. and J.A.; writing—review and editing, J.A., R.M., J.L., E.C., J.-W.L., D.M. and T.A.N.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1A2C2094943). This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education (2020R1A6A1A03046811). This work is supported by the Korea Agency for Infrastructure Technology Advancement(KAIA) grant funded by the Ministry of Land, Infrastructure and Transport (Grant RS-2022-00143965).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nguyen, T.A.; Jeon, S.; Maw, A.A.; Min, D.; Lee, J.W. Toward dependable blockchain and AI engines of digital twin systems for urban air mobility. In Proceedings of the KSAS 2021 Spring Conference, The Korean Society for Aeronautical & Space Sciences, Samcheok-si, Gangwon-do, Republic of Korea, 7–9 July 2021; pp. 408–409.
2. Nguyen, T.A.; Li, J.; Jang, M.; Maw, A.A.; Pham, V.; Lee, J.W. Cloud-in-the-loop simulation: A cloud-based digital twin HW/SW framework for multi-mode AI control simulation of eVTOL KADA-UAM Personal Aerial Vehicles. In Proceedings of the KSAS 2022 Spring Conference, The Korean Society for Aeronautical & Space Sciences, Goseong-gun, Gangwon Province, Republic of Korea, 20–24 April 2022; pp. 138–139.
3. Jiang, L.; Peng, X.; Xu, G. Time and Prediction based Software Rejuvenation Policy. In Proceedings of the 2010 Second International Conference on Information Technology and Computer Science, Kiev, Ukraine, 24–25 July 2010; pp. 114–117. [CrossRef]
4. Nguyen, T.A.; Min, D.; Park, J.S. A Comprehensive Sensitivity Analysis of a Data Center Network with Server Virtualization for Business Continuity. *Contin. Math. Probl. Eng.* **2015**, *2015*, 521289. [CrossRef]
5. Dewi, L.P.; Noertjahyana, A.; Palit, H.N.; Yedutun, K. Server Scalability Using Kubernetes. In Proceedings of the 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON), Bangkok, Thailand, 11–13 December 2019; pp. 1–4. [CrossRef]
6. Chuang, C.F.; Chen, S.S. To Implement Server Virtualization and Consolidation Using 2P-Cloud Architecture. *J. Appl. Sci. Eng.* **2017**, *20*, 121–130.
7. Vaughan-Nichols, S.J. Containers vs. Virtual Machines: How to Tell Which Is the Right Choice for Your Enterprise. 2016. Available online: https://www.networkworld.com/article/3068392/containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterprise.html (accessed on 10 September 2022).
8. Hat, R. Automation, Cloud, Security Lead Funding Priorities. 2018. Available online: https://www.redhat.com/en/blog/redhat-global-customer-tech-outlook-2019-automation-cloud-securitylead-funding-priorities?source=bloglisting (accessed on 11 June 2019).
9. Portworx. 2018 Container Adoption Survey. 2018. Available online: https://portworx.com/wpcontent/uploads/2018/12/Portworx-Container-Adoption-Survey-Report-2018.pdf (accessed on 11 June 2019).
10. Pereira Ferreira, A.; Sinnott, R. A Performance Evaluation of Containers Running on Managed Kubernetes Services. In Proceedings of the 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, NSW, Australia, 11–13 December 2019; pp. 199–208. [CrossRef]
11. Education, I.C. Containerization. 2019. Available online: https://www.ibm.com/cloud/learn/containerization (accessed on 20 September 2022).
12. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and Linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 171–172. [CrossRef]
13. Joy, A.M. Performance comparison between Linux containers and virtual machines. In Proceedings of the 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 19–20 March 2015; pp. 342–346. [CrossRef]
14. Xavier, M.G.; Neves, M.V.; Rossi, F.D.; Ferreto, T.C.; Lange, T.; De Rose, C.A.F. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, UK, 27 February–1 March 2013; pp. 233–240. [CrossRef]

15. Xavier, M.G.; Neves, M.V.; Rose, C.A.F.D. A Performance Comparison of Container-Based Virtualization Systems for MapReduce Clusters. In Proceedings of the 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turin, Italy, 12–14 February 2014; pp. 299–306. [CrossRef]
16. Xavier, M.G.; De Oliveira, I.C.; Rossi, F.D.; Dos Passos, R.D.; Matteussi, K.J.; Rose, C.A.D. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, Finland, 4–6 March 2015; pp. 253–260. [CrossRef]
17. Morabito, R.; Kjällman, J.; Komu, M. Hypervisors vs. Lightweight Virtualization: A Performance Comparison. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015; pp. 386–393. [CrossRef]
18. Chang, Y.C.; Roohi, N.; Gao, S. Neural Lyapunov Control. In *Proceedings of the Advances in Neural Information Processing Systems*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Nice, France, 2019; Volume 32, pp. 3245–3254. [CrossRef]
19. Jiang, Z.; Lynch, A.F. Quadrotor Motion Control Using Deep Reinforcement Learning. *J. Unmanned Veh. Syst.* **2021**, *9*, 234–251. [CrossRef]
20. Glaessgen, E.; Stargel, D. The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles. In Proceedings of the 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA, Honolulu, HI, USA, 23–26 April 2012; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2012. [CrossRef]
21. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Springer: Cham, Switzerland, 2018.
22. Huang, Y.; Kintala, C.; Kolettis, N.; Fulton, N. Software rejuvenation: Analysis, module and applications. In Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers, Pasadena, CA, USA, 27–30 June 1995; pp. 381–390. [CrossRef]
23. Xu, J.; You, J.; Zhang, K. A neural-wavelet based methodology for software aging forecasting. In Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, HI, USA, 12 October 2005; Volume 1, pp. 59–63. [CrossRef]
24. Avritzer, A.; Weyuker, E.J. Monitoring Smoothly Degrading Systems for Increased Dependability. *Empir. Softw. Eng.* **1997**, *2*, 59–77. [CrossRef]
25. Nguyen, T.A.; Kim, D.S.; Park, J.S. A Comprehensive Availability Modeling and Analysis of a Virtualized Servers System Using Stochastic Reward Nets. *Sci. World J.* **2014**, 1–18. [CrossRef] [PubMed]
26. Singleton, A. The Economics of Microservices. *IEEE Cloud Comput.* **2016**, *3*, 16–20. [CrossRef]
27. Dragoni, N.; Lanese, I.; Larsen, S.; Mazzara, M.; Mustafin, R.; Safina, L. Microservices: How To Make Your Application Scale. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 11th ed.; Springer: Cham, Switzerland, 2017.
28. Trunov, A.S.; Voronova, L.I.; Voronov, V.I.; Ayrapetov, D.P. Container Cluster Model Development for Legacy Applications Integration in Scientific Software System. In Proceedings of the 2018 IEEE International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS), St. Petersburg, Russia, 24–28 September 2018; pp. 815–819. [CrossRef]
29. Ageyev, D.; Bondarenko, O.; Radivilova, T.; Alfroukh, W. Classification of Existing Virtualization Methods Used in Telecommunication Networks. In Proceedings of the 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), Kyiv, UKraine, 24–27 May 2018; pp. 83–86. [CrossRef]
30. Blog, G.C.P. An Update on Container Support on Google Cloud Platform. 2014. Available online: https://cloudplatform.googleblog.com/2014/06/an-update-on-container-support-on-google-cloud-platform.html (accessed on 8 October 2021).
31. Research, G. Hype Cycle for Cloud Computing. 2018. Available online: https://www.https://www.gartner.com/en/documents/3884671 (accessed on 14 October 2021).
32. Diamanti. 2018 Container Adoption Benchmark Survey. 2018. Available online: https://diamanti.com/wp-content/uploads/2018/07/WP_Diamanti_End-User_Survey_072818.pdf (accessed on 12 October 2021).
33. Forrester. The Forrester New waveTM: Enterprise Container Platform Software Suites. 2018. Available online: https://cloud.google.com/containers/ (accessed on 18 October 2021).
34. Kubernetes. Production-Grade Container Orchestration. 2019. Available online: https://kubernetes.io/ (accessed on 18 October 2021).
35. Nguyen, T.A.; Min, D.; Choi, E.; Lee, J.-W. Dependability and Security Quantification of an Internet of Medical Things Infrastructure Based on Cloud-Fog-Edge Continuum for Healthcare Monitoring Using Hierarchical Models *IEEE Internet Things J.* **2021**, *8*, 15704–15748. [CrossRef]
36. Basili, G.; Caldiera, V.R.; Rombach, H.D. The goal question metric approach. In *Encyclopedia of Software Engineering*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1994; pp. 528–532.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.