Mitigating Resource Management and Continuous Integration Obstacles in Heavy Traffic Systems Using Containerization and Orchestration Tools

Tural Ahmadov, Leyla Muradkhanli

Graduate School of Science, Art and Technology,
Khazar University, Azerbaijan
Corresponding Author. Email: ahmadov.tural@khazar.org

Abstract

In this paper I covered virtualization technologies, including Virtual Machines and Containerization engines analyzing their advantages and drawbacks. After reviewing possible container application areas, I introduce Kubernetes, an orchestration tool that manages a cluster of containers on autopilot given correct configurations. After that, I give more details on how an orchestration framework functions, communicates with containers, and keeps the containers up. Moreover, I propose CI/CD tools and compare their efficiency through a designed experiment built using a cluster of nodes provided by the accessible version of Google Cloud Platform and deployed containers via Google Kubernetes Engine. At the end of the experiment, I give a comprehensive analysis of the results. To summarize, both containers and virtual machine technologies allow users to describe and develop their software environments before running them on top of multiple resources in a portable, repeatable manner. With containers, it is possible to construct scalable architecture composed of a large number of services (microservices). Also, the integration of new features can be done more efficiently if deployed in a continuous manner using the proposed CI/CD tools. Nevertheless, there are open questions to be researched, such as how the various tools respond when something goes wrong in the pipelines and the best policies for reverting to previous versions to ensure high availability.

Keywords: VM; containerization; orchestration; CI; CD; pipeline.

Introduction

For high traffic systems any optimization means that the project may use less resources which means lowering the costs. I am going to look through and compare existing containerization and orchestration solutions which can assist with solving the issue.

Moreover, for real-time 24-hour active systems integration of new features can be done only in continuous way as down time has to be as close to zero as possible. CI/CD tools with Kubernetes are my primary target of research for the challenge. Given the wide selection of alternatives available, it can be very difficult for an organization that wants to adopt Kubernetes to decide on which CI/CD tools to adopt. This thesis will investigate the advantages and disadvantages of some of the most popular tools, as well as different types of CI/CD pipelines in Kubernetes, using a combination of literature studies and experiments.

This subsection has two subsections

The aims and objectives for the project are:

- A1 Research containerization technologies
  - O1 Compare container-based state-of-the-art virtualization engines
- A2 Review orchestration tools
  - O2 Contrast advantages and drawbacks of the existing container orchestration frameworks
- A3 Create an infrastructure to test the different CI / CD tools.
  - O3 Identify and setup example Kubernetes applications that will be used for testing.
  - O4 Set up pipelines with tools that combine CI / CD, separate CI and CD tools,

Kubernetes-specific tools and generic tools.

- A4 Establish evaluation criteria and use the testing infrastructure to evaluate CI / CD

pipelines.

  - O5 Identify the appropriate evaluation criteria.
  - O6 Characterize each pipeline according to the criteria.

➢ A5 Evaluate which CI/CD technologies are most suitable for a microservices application

on Kubernetes.

- O7 Investigate Kubernetes-specific tools compared to generic CI/CD tools.
- O8 Investigate integrated compared to standalone CI/CD tools.

Background and Related work

This section gives a short-term clarification of the associated work studies in resource management, containerization, orchestration and CI/CD.

Researchers in this study (MinSu Chae, 2017) examined the effectiveness of KVM and Docker. According to them, three different techniques were employed to gauge performance: (a) comparing the host operating system's CPU and memory utilization, (b) measuring idle CPU and memory usage and IO performance via massive file copying, and (c) comparing Web server performance using JMeter. The measured findings revealed a 3.6–4.6 times difference in memory consumption. When you launch a virtual computer using KVM, the operating system must start. To execute the program in containers, Docker needs the absolute least resources. A performance comparison reveals that Docker utilizes CPU, HDD, and RAM more quickly and effectively than KVM. In fact, even when no action is carried out, KVM wastes extra resources for the operating system. Additionally, while using KVM, the process of creating a new VM is time-consuming. When building a distributed system, it takes some time to generate a new VM for load balancing if a VM suddenly suffers a load. To do more processing on the same PM, utilize the Docker's Container as opposed to a VM. KVM and Docker are only contrasted when set up on a single physical machine in the study. Based on the placement technique, a clustering environment influences the performance of a virtual machine and containers. As a result, further research is required to compare the effectiveness of KVM and Docker in a clustering setting.

Another research examined the outcomes of several Kubernetes resource management tools, including the Horizontal Pod Autoscaler and resource allocation through request and limit settings. Experiments demonstrate that identifying appropriate requests boosts cost-efficiency in contexts with few applications without significantly affecting other factors. This was confirmed for a Cassandra-based application, a made-up SaaS service, and workloads that were both seasonal and bursty. Regardless of the scaling technique chosen, scaling Cassandra in Kubernetes

hurts performance rather than increases it because of an overhead added by running Cassandra on Kubernetes. Even when pods are co-located and the workload is seasonal, the HPA works effectively for an artificial SaaS application. Other strategies could be used for workloads that come in bursts. In conclusion, despite certain drawbacks, Kubernetes' scaling capabilities show considerable promise for preventing SLA breaches and improving resource cost-efficiency in settings focused on containers (Stef Verreydt, 2019).

The case study in this research addressed a variety of adoption issues, according to studies on continuous integration [3], with the following conclusions standing out: 1) A key element for successful CI implementation is mentality. In order to convert skeptics, one must take into account their resistance to the introduction of a new procedure. 2) In order to make the everyday activities involved in the CI process easier, testing tools and the infrastructure supporting it must be mature. To enable more frequent and effective integrations, continuous integration promotes the use of automated technologies. (3) Like Agile, the assumptions behind the CI concept could not hold true for all businesses, goods, or projects, particularly those with broader scopes. Some of the difficulties with the shift to continuous integration have been recognized, such as testing, infrastructure maturity, tools, and attitude. Software needs, however, were also mentioned in this survey as a barrier to CI adoption.

Knowing how to overcome the obstacles that an organization may have while implementing CI gives practitioners degree of understanding that they may not have had before. Companies who are going to implement CI might utilize these problems as a checklist.

Resource management using containerization

IBM (Zhang et al., 2018) presented containerization technology for the first time in 1979. Implemented in the UNIX operating system V7, with the addition of a chroot (Ltd, 2022) system call. This was the first step toward isolation, with segregated groups functioning on a single host. This separation relied on numerous underlying technologies included into the Linux kernel, including namespaces and cgroups (Chiang, 2022). Namespace support was introduced in Linux kernel version 2.4.19, whereas cgroups, often known as control group technology, was published in Linux kernel version 2.6.24.

The introduction of microservices architecture (Campeanu, 2018) based on containers technology, including Linux containers (LXC) (Zhang et al., 2018), OpenVZ (Openvz, 2022), Docker (Inc, 2022), Singularity (Sylabs 2022), and uDocker (2022), produced a change in the way we construct applications, from

operations to programming. Container lifecycle management was utilized by several major orchestration systems, including Kubernetes (2019), Docker Swarm (2019), and Apache Mesos (2022). These orchestrators offered frameworks for container management inside a microservices architecture. Furthermore, these frameworks include capabilities ideal for scheduling containers with limited resources, fault tolerance, and auto-scaling. Kubernetes and Open-Shift are two orchestration platforms that are becoming more popular in computer systems, particularly those in the industrial and scientific areas. Following that, Rancher-compliant orchestration management solutions arose to manage orchestrators while maintaining efficiency characteristics that assure performance throughout the computing infrastructure.

Although cloud computing is the most common setting for application containerization (Pahl et al., 2017), containerization technique is also applicable to various application domains other than cloud services, such as scientific computing, big data processing, high performance computing, and development operation (devops).

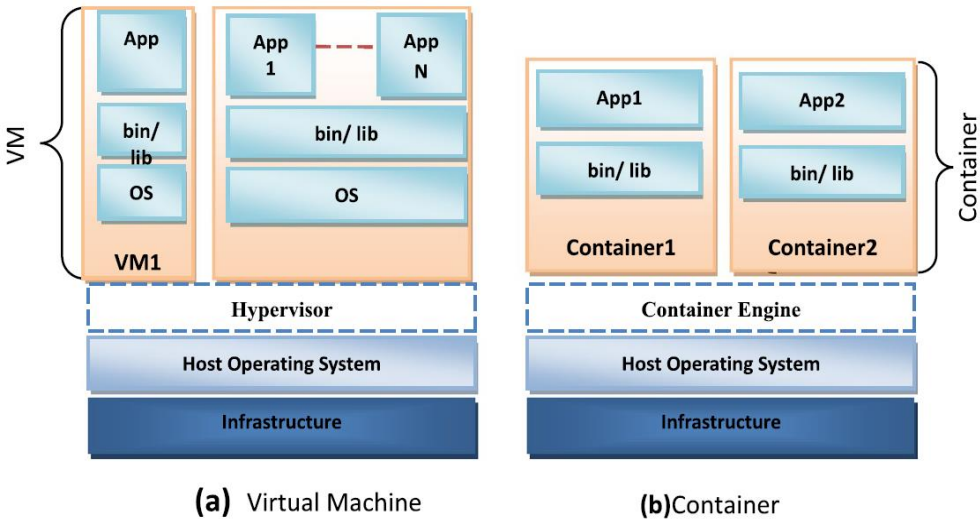

Figure 1. Comparison of system architecture-based virtualization.

The most significant advantage of virtualization is that it abstracts the hardware. It does, however, provide an isolated working environment for programs by aggregating logical resources such as CPU, memory, network, and storage. As demonstrated in Figure 1-a, the virtual machine (VM) instance's whole guest OS operates as a single process on the host. This results in high resource needs, which cause the VM to start slowly.

I/O routing is used in virtualization to coordinate requests between virtual devices and shared physical hardware. Instead of controlling resources, virtual machine migration between real computers created security risks. This vulnerability makes the system more insecure, and installing virtualized systems has become considerably more difficult. Operating system-level virtualization is the most popular form of virtualization, which allows for the use of isolation methods. The isolation method offers users with virtual environments that are comparable to those seen on dedicated servers. Container refers to the isolated virtual environment seen in Figure 1-b.

Container orchestration

Container orchestration enables cloud and application providers to describe how multi-container packaged applications in the cloud are selected, deployed, monitored, and dynamically configured. It is a framework that provides a collection of APIs for managing the container's whole life cycle (cf., Figure 2).

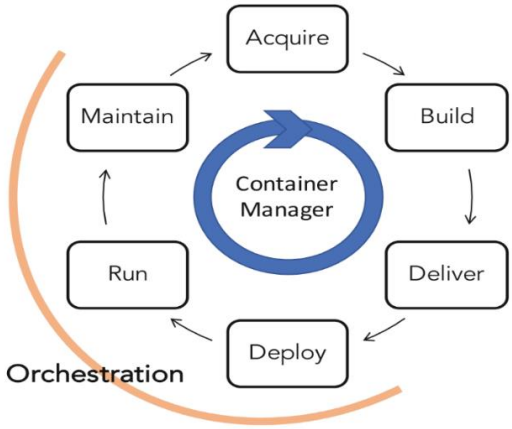

Figure 2. The life cycle of a container.

Container managers may be on-premise (to be deployed, configured, and maintained on private datacenters or in the cloud) or managed (offered by cloud providers as a service). Docker was created as a container management solution; however, the container management ecosystem is constantly evolving. Docker, for example, can handle both Windows Server containers and Hyper-V containers. rkt

also provides APIs for simple application container management. Google Container Engine, Microsoft Azure Container Service, and Amazon ECS are three cloud platform managed container managers (usually they support Docker and LXC). LXD is the manager for LXC in terms of system containers. OpenVz also offers APIs for container management.

Material and method

Implementation

This section depicts the experiment's execution. Figure 3 depicts the implementation that will be used in this investigation.

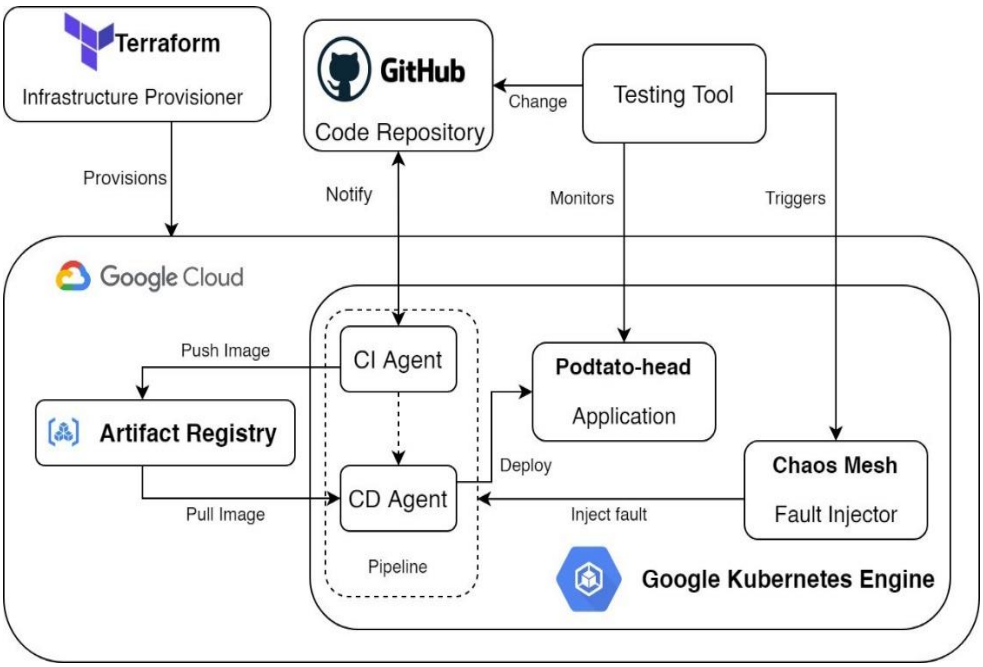

Figure 3. Implementation overview.

As seen from Figure 3, cluster was setup using Google Cloud environment. Terraform for infrastructure provisioning, custom testing tools, github as code repository.

Results and disscussions

According to the experiment findings:

❖ Kubernetes-specific tools have a quicker deployment time and can sustain pod defects with minor deployment time increases.

❖ Integrated tools take longer to deploy. However, this is most likely due to the tool's general nature. Because both the CI and CD tools must be installed in the application cluster, integrated tools are more difficult to set up in a pull fashion.

The standalone and Kubernetes-specific tools produced the greatest outcomes based on the metrics utilized in this research. However, integrated and generic tools may offer additional advantages, such as being quicker to set up or having plugins to facilitate the integration process.

Conclusion

Users may specify and create their software environments in containers and virtual machines and then execute them on top of multiple resources in a portable, repeatable manner. This article provided an in-depth examination of commonly used containerization technologies and their key characteristics. Moreover, I illustrated and discussed various aspects of application domains to define container architecture for computing systems. Furthermore, the research has proven that understanding the capabilities and methodologies available for a specific containers-based solution, as well as the characteristics of workloads, is critical for optimizing systems. The container technique is now at the core of contemporary computing infrastructure because it eliminates various issues associated with sophisticated execution environment requirements that are often in conflict with other components of application operations. Containers have been embraced by various efforts and are becoming a standard technology, such as Cloud Native and Dev/Ops. Containers allow for the creation of scalable architectures built of a large number of services (microservices). IT businesses such as Google, Microsoft, Netflix, and others already depend on container technology in their production environments.

There are no established comparison criteria to assess alternative CI / CD solutions in my cluster deployment since research on CI / CD tools and pipelines is few, and there are few big studies that compare different tools. Although the deployment time utilized in this experiment is useful, other considerations may be more significant in

selecting which CI/CD systems to employ.

More research is required to determine which criteria and characteristics are most relevant to consider when selecting tools to employ. More studies are also necessary to evaluate pipeline design security and how firms should pick between different CI / CD methodologies, such as a push or pull. Organizations who support the pull-style and GitOps pipeline approaches say that there are security advantages, but this has to be verified and explored more.

Other tools may have additional characteristics that make them more appropriate for certain areas or jobs. More research on similar instruments and their merits and limitations might be conducted. This might include tool functionality that is not completely integrated with the CI/CD process. For example, to automatically prune obsolete application installations and monitor for changes in the running application.

Another area that may need more research is how the different tools react when anything goes wrong in the pipelines. For example, if a new software deployment has defects, there may be a simple way to return to prior versions of the application to guarantee high availability.

Acknowledgments

Thanks to Google Cloud Platform for providing 90 days free service trial. That helped me in setting up infrastructure for conducting the experiment.

References

Adam Debbiche, M. D. R. B. S. (2015). "Challenges When Adopting Continuous Integration: A Case Study. Product-Focused Software Process Improvement, 17-32.

Beltre, A. M., Saha, P., Govindaraju, M., Younge A., & Grant, R. E. (2019). Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms. 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), 11-20.

Campeanu, G. (2018). A mapping study on microservice architectures of Internet of Things and cloud computing solutions. 2018 7th Mediterranean Conference on Embedded Computing (MECO), 1-4.

Chiang, E. (2022). Containers from Scratch. Available: https://ericchiang.github.io/post/containers-from-scratch/#container-file-system.

Inc, D. (2022). The Docker platform. Available: https://docs.docker.com/.

Ltd, C. (2022). Container and virtualization tools. Available: https://linuxcontainers.org/.

MinSu Chae, H. L. K. L. (2017). "A performance comparison of linux containers and virtual machines using Docker and KVM," *Cluster Computing*, p. 1765–1775.

Openvz. (2022). Open source container-based virtualization for Linux. Available: https://openvz.org/.

Pahl, C., Brogi, A., Soldani H., & Jamshidi, P. (2017). Cloud Container Technologies: A State-of-the-Art Review. IEEE Transactions on Cloud Computing, 677-692.

Stef Verreydt, E. H. B. (2019). "Leveraging Kubernetes for adaptive and cost-efficient resource management," *Association for Computing Machinery*, p. 37–42, 2019.

Sylabs. (2022). Singularity. Available: https://www.sylabs.io/docs/.

T. a. s. foundation. (2022). Apache Mesos. Available: https://mesos.apache.org/..

Tarek Menouer, P. D. (2019). Containers scheduling consolidation approach for cloud computing. Pervasive Systems, Algorithms and Networks, 178-192.

uDocker. (2022). uDocker. Available: https://github.com/indigo-dc/udocker.

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., & Zhou, W. (2018). A comparative study of containers and virtual machines in big data environment. 2018 IEEE 11th International Conference on Cloud Computing, 178-185.