International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 10 Issue 5 ǁ 2022 ǁ PP. 72-77

A Comprehensive Comparison Study on Container Orchestration Frameworks

M J Santhosh Kumar, Dr. Suma S
PG Student, Department of MCA, Dayananda Sagar College of Engineering College of Engineering®, Bengaluru, India[1]
Associate Professor, Department of MCA Dayananda Sagar College of Engineering College of Engineering®, Bengaluru, India[2]

Abstract- *As the businesses realize the dynamism of what can be done with their data, they are moving on from their existing resources to well-equipped Data Centers have become top priority for businesses. The deployment and management of applications in large-scale clustered environments are always a difficult task. Novel software architecture patterns, such as microservices, have evolved in the previous decade to increase program modularity and expedite development.Testing, scalability, and component replacement are all things that need to be done. To help with these,new techniques, such as DevOps methodology and tools supporting better software development and testing collaboration. While there are various new container orchestration engines on the market, such as Docker Swarm, Kubernetes, Apache Mesos, and Cattle, a comprehensive functional evaluation is required and performance evaluation to aid IT managers in the selection process. There is still a lack of the best suited orchestration solution.The purpose of this work is to close that gap and provide the experiment findings for the same.*

Keywords: *containerization; container orchestration engine; Docker Swarm; Kubernetes*

Date of Submission: 04-05-2022 Date of acceptance: 18-05-2022

I. INTRODUCTION

The need for frequent software delivery and development processes, known as "agile," as well as associated DevOps methodology and technologies to make software development and delivery a continuous lifecycle [1], has resulted from the requirement for business agility. The microservice architectural pattern has been adopted in line with these developments to break down the monolithic structure into separate components that are easier to build, maintain, and scale. In a nutshell, this design isolates each core application functionality in microservices and uses microservices as building blocks to createbigger systems.

Different frameworks have emerged only recently, and they are in constant evolution as new features are being introduced. This reality makes it difficult for practitioners and researchers to take care of a transparent view of the technology space. First, we isolate the foremost functional elements to research existing solutions which we apply to four main representative market players, namely, Docker Swarm,Kubernetes, Apache Mesos and Cattle. Second, we propose some performance metrics to benchmark them. Third, we show several experimental results to assess their performance behavior. managers within the design and migration to 5G fully-software-based infrastructures.

The rest of the paper is divided as follows. Section II provides the background about functional analysis and therefore the qualitative comparison. In Section III, we introduce main performance metrics, and in Section IV we use them for performance assessment. Section V and Section VI summarizes the related works and draws conclusions and future works.

II. BACKGROUND

In this section, we cover the background about the container orchestrator model, and then qualitatively compare a selection ofwidely diffused container orchestrators.

The idea of container technology dates back to 1992. At the bottom of container technology there's the concept of cgroup andLinux namespace to know the landscape of container technology.Container orchestration engines provide support for distributed applications. Different frameworks have emerged only recently, which they're in constant evolution as new features are being introduced. This reality makes it difficult for practitioners and researchers to take care of a transparent view of the technology space.

A. Container Orchestration Frameworks
1. Docker Swarm refers to Managers and Workers
2. Kubernetes refers to Masters and Nodes
3. Mesos refers to Masters and Agents.

The Docker ecosystem includes tools ranging from development to production deployment frameworks. Docker swarm is a component of cluster management in that list. For controlling a cluster of Docker containers, a combination of docker-compose, swarm, overlay network, and an outstanding service discovery tool like etcd or consul can be employed .The master(Managers) is that the node that is responsible for scheduling containers, whereas a slave (Workers) is liable for launching received containers. Both redundancy and placement are managed by the scheduling layer (see Table II)

In comparison to other open-source container cluster management technologies, Docker Swarm is still evolving in terms of functionality. Given the large number of docker contributors, it won't be long before the docker swarm has all of the finest features that other tools offer. Docker has a decent production strategy for utilizing the docker swarm in production described.

Kubernetes is a Google-developed open-source technology for managing containerized applications across a cluster of computers [8]. Kubernetes is built on a master/slave architectural pattern, in which a developer sends a list of apps to a master node, and the platform then distributes them between slave and master nodes. The master node is the cluster's control plane, and it may be duplicated to ensure high availability and fault tolerance by utilizing the scheduling layer. Slave nodes (also known as minions) are nodes that run application containers. A containerized application is provided by Kubernetes as a collection of containers, each of which is dedicated to a single microservice. Replication Controller is a Kubernetes component that ensures that a particular number of pods are currently providing a specific service. If the current status differs from expectations, such as in the event of an outage node, the replication controller starts arranging a new instance on a different slave node automatically. The heartbeat controller is developed with a subscriber notification system and is not overlyaggressive.

Apache Mesos, The University of California, Berkeley produced Apache Mesos, a major open source project. The architecture is based on a master/slave design paradigm, in which slave nodes handle job execution. The master process, which runs on a cluster manager node, is in charge of controlling and monitoring the whole cluster architecture. Mesos is another cluster management technology that excels in container orchestration. Twitter built it for its infrastructure and subsequently made it open source. eBay, Airbnb, and other businesses utilize it. Mesosis not a container-specific tool. Mesos is not a container-specific solution; instead, it may be used for VM or physical machine clustering for workloads other than containers (Big data, for example). It contains a Marathon framework for deploying and managing containers on a Mesos cluster that is very efficient. Mesos solutions are frequently created by layering an applicationon top of a Mesos cluster. Marathon is an application-level management system. Marathon communicates with the master component to provide orchestration services to the whole Mesos cluster. As a result, if a slave fails, Marathon creates a new instance to ensure fault tolerance. Mesos provides high-availability replicating master nodes to provide failover in the event of a master failure. It relies on Apache Zookeeper to accomplish this, which includes an election process that selects a new master node.

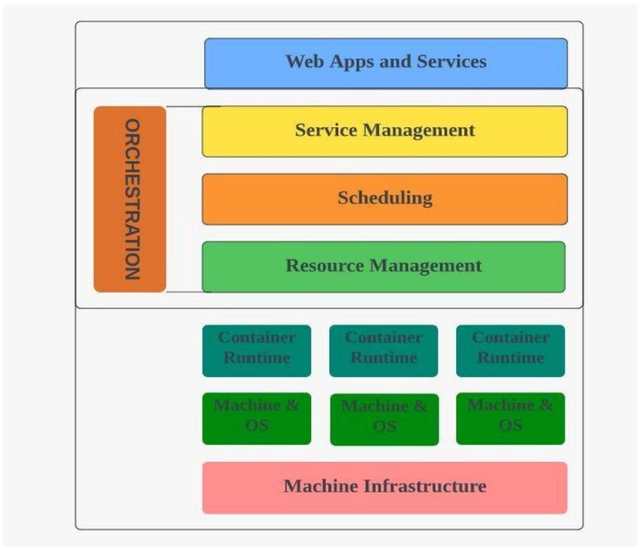


Fig. 1 Container Orchestration Layers

B. *Model and Functional Elements*

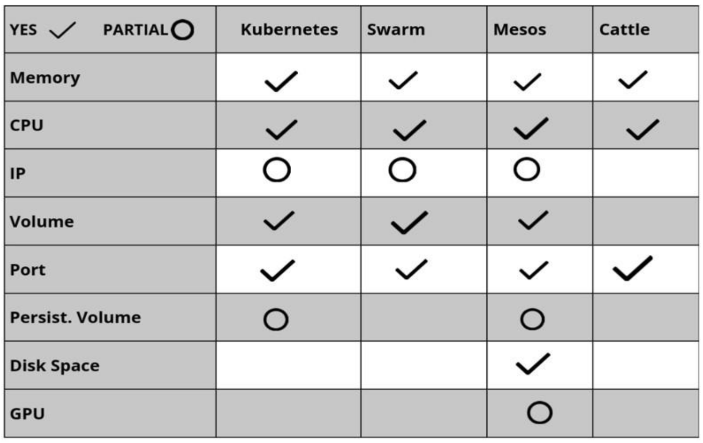
Container orchestration enables the creation of automated provisioning and change management procedures that ensure that agreed-upon policies and service standards are always met. Figure 1 shows our reference layered orchestration engine architecture, in which a collection of computers realise the support substrate via their kernel and container runtime. Three levels make up the orchestration engine structure: resource management, scheduling, and service management. Due to space constraints, we present a quick description of the important functional features below, which we then utilise for our qualitative analysis (see also Tables I, II, and III).

The resource management layer controls low-level resources; in this example, functional elements are the resources that may be managed/composed and include RAM, CPU/GPU, disc space, volumes (i.e., the ability to interface with the local hosting machine's file system), and persistent volumes. Its goal is to maximize usage while avoiding interference amongst competing containers for resources. The resources provided by the comparing solutions are listed in Table I.

The scheduling layer seeks to make effective use of cluster resources. It usually takes user-supplied input (e.g., placement restrictions, replication degree, etc.) and chooses how to arrange all of the containers that make up the applications. The following are the most significant capabilities (see Table II): placement, which allows you to directly manage scheduling decisions; replication/scaling, which allows you to describe the number of microservice copies.

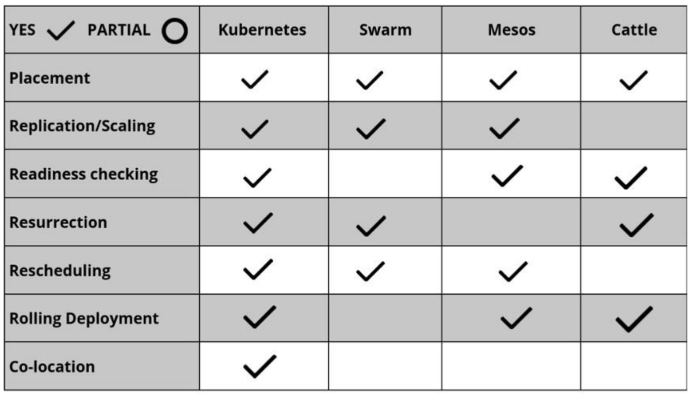
Finally, the service management layer gives you the tools you need to create and deploy (complicated) corporate applications. It is in charge of high-level aspects (see Table III), such as: Labels are used to associate metadata with container objects. Also the dependencies to express dependencies between containers; groups/namespaces to separate containers and facilitate multi-tenancy, Load balancing to distribute incoming traffic; and readiness checking to only make the application available online when it is ready to receive incoming traffic.

TABLE I. RESOURCE MANAGEMENT LAYER COMPARISON



YES ✓ PARTIAL ○	Kubernetes	Swarm	Mesos	Cattle
Memory	✓	✓	✓	✓
CPU	✓	✓	✓	✓
IP	○	○	○	
Volume	✓	✓	✓	
Port	✓	✓	✓	✓
Persist. Volume	○		○	
Disk Space			✓	
GPU			○	

TABLE II. SCHEDULING LAYER COMPARISON



YES ✓ PARTIAL ○	Kubernetes	Swarm	Mesos	Cattle
Placement	✓	✓	✓	✓
Replication/Scaling	✓	✓	✓	
Readiness checking	✓		✓	✓
Resurrection	✓	✓		✓
Rescheduling	✓	✓	✓	
Rolling Deployment	✓		✓	✓
Co-location	✓			

TABLE III. SERVICE MANAGEMENT LAYER COMPARISON

YES ✓ EXT/PART ○	Kubernetes	Swarm	Mesos	Cattle
Labels	✓	✓	✓	✓
Dependencies			✓	
Groups / Namespaces	✓		✓	
Load Balancing	✓		○	✓
Readiness Checking	✓		✓	

III. CONTAINER ORCHESTRATION ENGINEPERFORMANCE METRICS

Here comes the most crucial step for your research publication. Ensure the drafted journal is critically reviewed by your peers or any subject matter experts. Always try to get maximum review comments even if you are well confident about your paper.

This section focuses on the set of metrics that we used for comparing container orchestrator performances. First, we considered the time needed to finish the deployment of the container orchestration solution (Subsection III.A). Then, we concentrated on the performance analysis of the scheduling and service management layers (see Fig. 1) to gauge the time to make the appliance ready (for different scenarios, see Subsection III.B and III.C) and to gauge the time to reschedule and get over container/node faults (Subsection III.D).

A. Cluster provisioning time analysis

Elastic scalability is one among the core properties to be granted by cloud computing which has been boosted and facilitated by the advent of container-based technologies that enable fast bootstrap. In conjunction with that direction, the first performance metric evaluates the provisioning time needed for a new cluster and to properly configure it with the specified container orchestration support. Indirectly, this is also a measure of the container orchestrator complexity.

B. Failover Time

Failover mechanisms increase the reliability and availabilityof IT resources typically using clustering technologies to supply redundant implementations. For the analyzedcontainer orchestration solutions, failover was configured to automatically switch to a redundant resource instance whenever the currently active IT resources become unavailable. The failure can involve one container or the entire hosting cluster node; hence, in our analysis, we investigate both sorts of failures in terms of the timeto completely recover from the fault and make the appliance usable again.

IV. EXPERIMENTAL RESULTS

This section discusses the experimental setup of the testbed that is used to collect the experimental results and then reports a thorough performance evaluation and discussion structured according to the performance metrics introduced within the previous section.

A. Experimental Setup

The setup comprises 8 physical servers, namely, MicroServer HPE Gen8. Also it includes 2x Intel (R) Celeron(R) CPU G16610T @ 2.30 GHz processors with 12 GB of RAM. Each server is provided with 2 hard drives of 750GB. Both devices work at 7200 rpm.

Since customizing and fixing orchestrators may be a daunting task, we used Rancher [10]. Rancher includes everything required for managing containers performing at a staggering level compared to orchestration perspective. It allows to neglect orchestration-specific configurations and automates the setup of container orchestrator environments and facilitates the update and (re)configuration of new stable releases.

We have directly installed Rancher on top of four physical servers, each operating on the Ubuntu system. And have used Rancher UI tool and also its monitoring internal system for measuring performance test's times. To reduce error factor, each experiment has been repeated 33 times which shows average values; we aren't reporting standard deviations that are typically rather limited.

B. Results and Discussion

This section shows performance results collected for four representative container orchestration engines: Docker Swarm, Kubernetes, Mesos, and Cattle.

1) Cluster provisioning time

The first set of results inspects the required cluster provisioning time for deploying one orchestration tool via Rancher, Cattle requires the shortest time to deploy a single cluster since it's the Rancher native orchestration tool, and so, the whole architecture is optimized to deploy the container orchestrator that's provided by default. Kubernetes requires very high provisioning time.This is because of its architecture whichis the most complex one. In fact, Kubernetes is the solution that serves the most capabilities to manage and deploy production-level services (see Tables I, II, III). Docker Swarm and Apache Mesos,instead, showed a shorter provisioning time with reduced complexity.

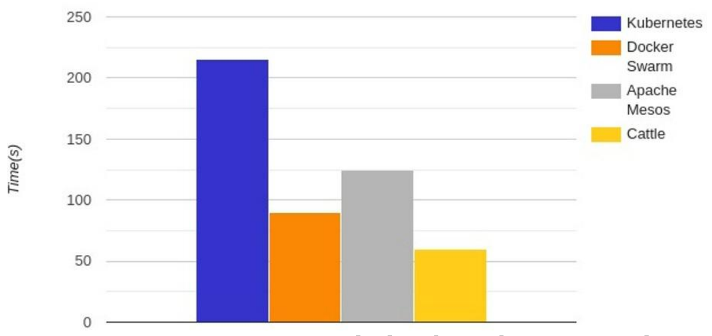


Fig. 2. Provisioning time to deploy the orchestration tools

2) Failover Time

Rescheduling allows restoring container services and it is natively supported by Kubernetes and Docker Swarm.However, Rancher introduces the likelihood to define a minimum number of running containers for every microservice. Hence, there is a dedicated Rancher functionality that provides this feature, even though the underlying container orchestrator doesn't offer this capability. Hence, for this set of experiments,the native features for Kubernetes and Docker Swarm were used, whereas for Cattle and Apache Mesos we exploited the Rancher capability.

As we can see in Fig. 3a, Kubernetes is one of the best solutions to the failure of the single container.On the contrary, as shown in Fig. 3b, in case of a node failure, it is the worst. This is not surprising whereas in the first case failure is detected by a local agent of Kubernetes that guarantees instead. the availability of running containers. In the second case,there is a replication controller which is responsible for guaranteeing the rescheduling of failed pods. This approach is event-based, and the Kubernetes controller notifies the Kubernetes object when there is a change in the number of pods . That chain effect is the main reason for the highest failover time value with Kubernetes. The Docker Swarm and Cattle, exploit the heartbeat functionality of the Docker architecture which allows them to provide the shortest failover time

.

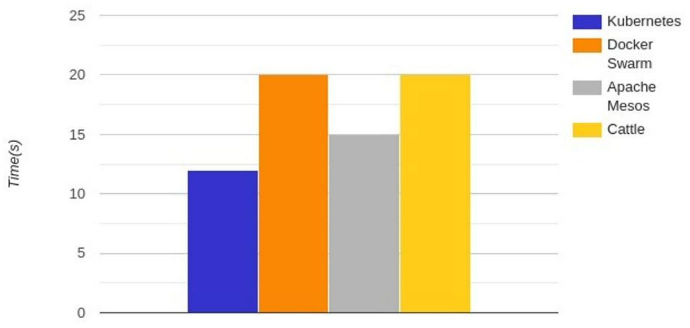


Fig. 3a. Failover Time - Container Failure.

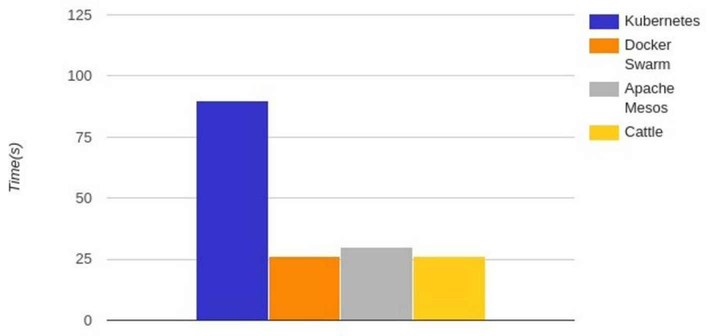


Fig. 3b. Failover Time - Host Failure.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we have analyzed and compared features and services offered by container orchestrators. Also we have designed a cohesive set of metrics to compare the performances at scheduling and service management layers and believe that those metrics are reusable for all kinds of container orchestrators. Finally, we chose four representative container orchestrators, namely, Docker Swarm, Kubernetes, Apache Mesos, and Cattle, that were deeply inspected and assessed. In terms of functional comparison, It is noticed that Kubernetes is one of the most complete orchestrators nowadays on the market. That explains why practitioners gravitate toward preferring it to other ones. At the same time, its complex architecture introduces, in some cases, a significant overhead that may hinder its performances.

The results obtained are stimulating our further research activities in the field. Kubernetes and Mesos are both aimed at making it simpler to deploy and manage applications in containers in the data center or cloud. Both offer a level of support to businesses of various sizes. In the end, it's all about choosing the correct cluster management solution for the company's unique requirements, both now and in the future.

ACKNOWLEDGMENT

We would like to show our gratitude towards all who have directly or indirectly contributed towards the success of the study and helping us to reach a final conclusion. .

REFERENCES

[1]. Canonical, "For CTOs: the no-nonsense method to accelerate your business using containers," white paper from February 2017.
[2]. Bernstein, "Container Orchestration Wars," by Karl Isenberg, in Velocity 2016, Santa Clara, USA
[3]. Karl Isenberg, “Container Orchestration Wars”, in Velocity 2016, Santa Clara, US
[4]. Z. Li et al., “Performance Overhead Comparison between Hypervisor and Container Based Virtualization”, in IEEE Int. Conf. on Advanced Information Networking and Applications (AINA), 2017, pp. 955-962.
[5]. R. Dua, A. R. Raja, and D. Kakadia, “Virtualization vs Containerization to Support PaaS”, in 2014 IEEE Int. Conf. on Cloud Engineering, 2014.
[6]. R. Morabito et al., “Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge”, IEEE Internet of Things Journal, vol. 4, pp. 1019-1030, 2017.
[7]. M. Rouse, “What is Docker Swarm?”, TechTarget, August 2016. http://searchitoperations.techtarget.com/definition/Docker-Swarm.
[8]. J. Ellingwood, “An Introduction to Kubernetes”, Digital Ocean, 14 October 2016. https://www.digitalocean.com/community/tutorials/an- introduction-to-kubernetes.
[9]. R. Howard, “Orchestration With Kubernetes, Docker Swarm, and Mesos”, Dzone, July 2017. https://dzone.com/articles/orchestration-with-kubernetes-docker-swarm-and-mesos.
[10]. “Rancher”, http://rancher.com/docs/rancher/v1.6/en/rancher- services/scheduler/.
[11]. Yang Zhao et al., “Performance of Container Networking Technologies”, in Proceedings of the Workshop on Hot Topics in
[12]. Container Networking and Networked Systems (HotConNet '17), pp.1-6.
[13]. E Datsika et al. “Software defined network service chaining for OTT service providers in 5G networks”, IEEE Communications Magazine 55 Nov2017.
[14]. Vincent Reniers, “The Prospects for Multi-Cloud Deployment of SaaS Applications with Container Orchestration
[15]. R. Morabito, “A performance evaluation of container technologies on Internet of Things devices”, in 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS), pp. 999-1000.
[16]. Z. Nikdel et al., “DockerSim: Full-stack simulation of container-based Software-as-a-Service (SaaS) cloud deployments and environments”, in 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2017, pp. 1-6.