A COMPARATIVE STUDY OF CLOUD CONTAINER-BASED ORCHESTRATION FRAMEWORKS FOR DEVOPS

by

Somesh Rao Coka

A research study report submitted in partial fulfilment of the requirements for the degree of Master of Engineering in Computer Science

Examination Committee:	Dr. Chantri Polprasert (Chairperson)
	Dr. Chaklam Silpasuwanchai
	Dr. Chutiporn Anutariya.
Nationality:	Indian
Previous Degree:	Bachelor in Electronics & Communications
	Jawaharlal Nehru Technological University
	Hyderabad, Telangana, India
Scholarship Donor:	AIT Fellowship

Asian Institute of Technology

School of Engineering and Technology

Thailand

Inter-Semester 2023

ACKNOWLEDGEMENT

I want to express my profound gratitude to my advisor Chantri Polprasert for his valuable support and guidance. The door to Prof. Chantri's office was always open whenever I ran into a trouble spot or had a question about my research or writing.

For their unwavering support and constant encouragement during my years at university and the preparation of this report, I am eternally grateful to my parents C.V.K Nageshwara Rao and Coka Karuna Devi. Without them, I could not have completed this task. I am truly grateful for it.

The faculty, and staff in the Department of Computer Science and Information Management were all invaluable resources as I worked to complete my one-of-a-kind research project. I'm thankful to have been a part of the institute that provided me with such top-notch study resources.

SOMESH RAO COKA

ABSTRACT

DevOps is one of these huge trends. It is a combination or say unification of two major traditionally separate worlds of software development and operations into one common cycle. This report presents an in-depth review of the market's leading container orchestrators tools, with a particular emphasis on Kubernetes, Docker and Mesos. Kubernetes is recognized by industry experts as an exceptionally versatile orchestrator due to its capacity to scale and adjust to diverse operational demands. In certain circumstances, the inherent complexities of Kubernetes' architecture may result in substantial overhead, regardless of the platform's widespread adoption. On the other hand, Mesos is emphasized as a resilient alternative for companies that have already implemented legacy systems, providing enhanced orchestration capabilities for diverse workloads. To assess the effectiveness of these orchestrators in streamlining the deployment and administration of applications in cloud-based container environments and data centres, the architectural frameworks of these platforms are investigated in this document.

A critical element of the study comprises a comparative assessment of the durations required to provision clusters using Kubernetes, Mesos, and Docker Swarm. This comparative assessment furnishes organisations with quantifiable metrics to contemplate when choosing an orchestration tool. The primary purpose of the report is to provide organisations with the guidance they need to make well-informed decisions regarding their long-term technology strategy, operational team expertise, and current technical infrastructure.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Over the previous 5-10 years, the Agile technique and culture have grown at a faster pace, exposing the need for a more robust approach to the current software development life cycle. Many say, DevOps is an offspring of agile methodology which was born to keep up with the needs and demands of increased software delivery speeds.

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high quality. DevOps is achieved by breaking down the silos between development and operations teams and automating the processes between them.

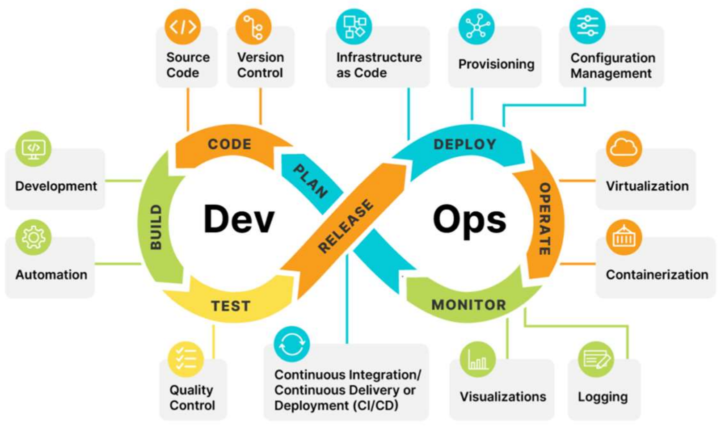

Figure 1 Different stages of DevOps

Most common DevOps use cases includes:

- *Continuous integration and continuous delivery (CI/CD):* This is a process of automating the building, testing, and deployment of code.
- *Infrastructure as code (IaC):* This is the practice of managing all infrastructure as code. It makes it easier to automate the deployment and management of infrastructure.
- *Containerization:* This is the practice of packaging software and its dependencies into a single unit called a container. Containers make it easier to deploy and manage applications in the cloud.
- *Microservices architecture*: This is an architectural style that breaks down applications into small, independent services. Microservices make it easier to develop, deploy, and scale applications.

Container orchestration is the process of deploying containers on a compute cluster consisting of multiple nodes. Orchestration tools extend lifecycle management capabilities to complex, multi-container workloads deployed on a cluster of machines. By abstracting the host infrastructure, container orchestration tools allow the users deploying to entire cluster as a single deployment target.

Figure 2 Different container orchestration tools

Container orchestration tools are software that help to automate the deployment, scaling, and management of containerized applications. They provide a way to manage multiple containers at the same time, and to ensure that they are running correctly.

1.2 PROBLEM STATEMENT

I would like to explain problem statements with a scenario as follows:

A DevOps engineer working for a large company that is migrating its applications to the cloud is responsible for deploying and managing a new containerized application that is critical to the company's business. The application is a microservices architecture, and it is composed of hundreds of containers. The engineer needs to choose a cloud container-based orchestration framework to deploy and manage the application. They are considering Kubernetes, Docker Swarm, and Apache Mesos.

This study will help any individual to overcome the above situation since this study comprises a detailed comparison between different orchestration tools so that he/she can choose the right tool for the right situation.

1.3 OBJECTIVES

Cloud container-based orchestration frameworks are becoming increasingly popular for deploying and managing containerized applications. However, there are a number of benefits and drawbacks to using these frameworks, and it is not always clear how to use them effectively.

- What are the different components of an orchestration tool, and how do they interact with each other?
- How can DevOps teams use these frameworks to ensure successful deployments of new software solutions?
- To perform a performance analysis to choose the optimum orchestration tool for quick deployment.

This study is significant because it will provide valuable insights into the use of cloud container-based orchestration frameworks. This study’s findings will interest DevOps teams, cloud architects, and software developers.

1.4 LIMITATIONS AND SCOPE

This study focuses specifically on three container orchestration tools: Docker Swarm, Kubernetes, and Apache Mesos. While other orchestration tools are available, this study is limited to these three due to their widespread use and popularity. The study aims to provide a comprehensive overview of each tool, including its use cases, DevOps practices, and performance metrics.

The scope of this study is limited to introducing the orchestration tools and comparing them based on the aforementioned factors. The study aims to provide an in-depth analysis of each tool's capabilities to recommend a specific tool for a particular use case. Additionally, while the study provides an overview of DevOps methodologies, it does not aim to provide an exhaustive analysis of all available methodologies.

This study is intended for readers who are interested in learning about container orchestration tools and their use in DevOps processes. The study may be particularly useful for software developers, system administrators, and IT professionals who are responsible for managing and deploying containerized applications.

1.5 ORGANIZATION OF THE REPORT

Following chapters are discussed in this report.

Chapter 1: Introduction

Chapter 2: Related Work

Chapter 3: Orchestration Tools: Description and Comparison

Chapter 4: Results

Chapter 5: Conclusion

CHAPTER 2
RELATED WORK

This chapter provides an overview of the existing works that are relevant to the chosen topic. It also outlines the procedure followed for conducting a thorough literature review to identify and extract the most pertinent literature.

2.1 REVIEW METHODOLOGY

The literature review process involves following a systematic procedure to gain a clear understanding of the current research landscape. It helps identify research trends, gaps, and potential areas for further investigation. This involves discussing the search Strategy and Selection Process, and inclusion and exclusion criteria in the subsequent sub-sections.

2.2. SEARCH STRATEGY

Digital libraries such as IET, IEEE Xplore, Wiley, google scholar, and science direct are used to find various research papers related to this study. As this study focuses on various concepts on the orchestration tools and devops practices which will be discussed in the subsequent chapters, there are various keywords used to collect all the necessary articles. The primary search strings used were "Kubernetes", “docker”, “Mesos”. Based on these search strings, few articles are identified and more search strings. The search string " Cloud Computing”, “Container Orchestration” AND “devops", search strings are used to find articles related to my study.

Other than research articles selected from the various search strings mentioned above different project websites and web blogs are also referred to understand the topic.

5.1 SUMMARIES OF RELATED PUBLICATIONS

DevOps is a set of principles and practices that enable better communication and collaboration between relevant stakeholders to specify, develop, continuously improve, and operate software and systems products and services. DevOps is suitable for most life cycle process models, and particularly appropriate when teams adopt agile methodologies. DevOps can be just as valuable in an iterative waterfall approach. DevOps demands higher levels of integration, collaboration, automation, and established feedback systems for continuous life cycle process model improvement. Higher-level process capabilities specifically bring security into prominence in the activities and tasks.

This research paper [1] attempts to clarify the requirements and offer comprehensive instructions for the implementation of DevOps, a methodology that incorporates software development and IT operations to expedite the delivery of software products and services with enhanced dependability and efficiency. In recent years, the DevOps approach has gained significant popularity and recognition within the field of software engineering. This approach has proven to be highly effective, as it facilitates the seamless integration, delivery, testing, monitoring, and feedback processes throughout the entire software life cycle. Nevertheless, the implementation of DevOps methodologies presents notable challenges and potential risks that warrant careful consideration. These include concerns pertaining to security, compliance, quality assurance, and collaboration.

In cloud computing, virtualization of the operating system is done Container-based virtualization is a type of OS-level virtualization that enables multiple isolated user-space instances to run on a single host kernel. In contrast to hypervisor-based virtualization, which necessitates a distinct guest OS for each virtual machine, container-based virtualization shares the host OS's resources with the containers, resulting in less overhead and greater efficiency. by an approach known as Container. The deployment and management of software can be achieved with the help of containers. It runs the software applications in a remote environment by packing all the application's dependencies. The applications that1 are containerized can easily drift towards the cloud. The APIs given by the orchestration can be used to deploy, modify or clone the container easily and efficiently.[5]

The paper [5] seeks to compare the efficiency of hardware resource utilisation and the scalability of applications using these two virtualization technologies. It employs four open source benchmark tools to measure the CPU, RAM, network and disc performance of Docker and VMs based on VirtualBox. Additionally, the paper discusses the benefits and drawbacks of each technology and their implications for cloud computing.

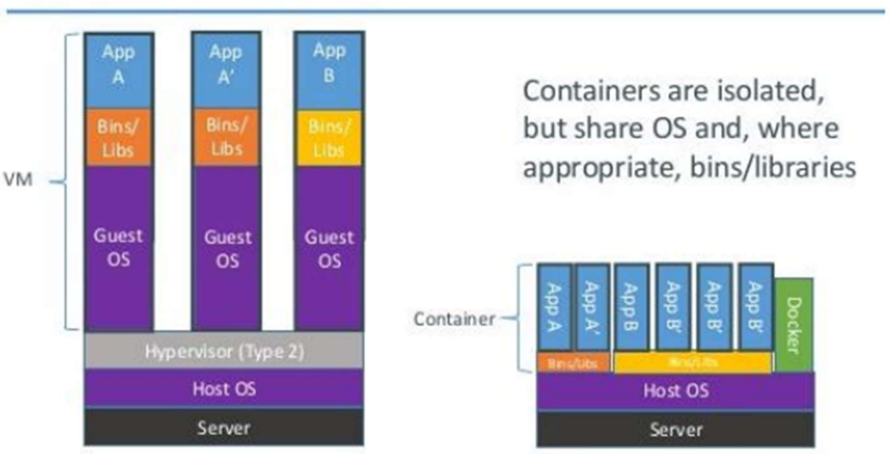

Figure 3 Virtual machine vs Docker

Authors decided to uses four benchmark tools to test the performance of Docker and VMs: HPL for CPU performance, sysbench for RAM performance, iperf for network performance and Bonnie++ for disk performance. The tests were performed on a HP Proliant server with a single Intel Xeon processor, 16 GB RAM, 1 TB Hard Disk and a Gigabit LAN network card6. The host operating system was Debian 8 for both Docker and VMs.

The paper [3] focuses on the efficacy of Singularity, an HPC-specific container-based virtualization system. On a cluster of 16 nodes connected by InfiniBand, they compared the performance of Singularity, Docker, and bare-metal. They used a suite of benchmarks and applications from NPB, HPL, HPCC, Graph500, LAMMPS, GROMACS, and TensorFlow to measure the CPU, memory, network, and disc performance. Except for disc performance, they discovered that Singularity had near-native performance on all metrics.

The possibility of container innovation traces all the way back to 1992. At the lower part of container innovation, there is an idea of Linux namespace and cgroup to know the scene of container technology. Many frameworks have arisen as of late, which are in a steady development as they begin presenting new elements.[8]

Nowadays, several industries have adopted the concept of containers rather than virtual machines because of their lightweight, portability, and agility to handle virtualization. Containers can be divided into four categories:

- System container,
- Application container,
- Container manager,
- and Orchestrator.[2][3][4]

Kubernetes, Docker Swarm, Marathon, and Cloudify are some of the orchestrators. LXC, OpenVZ, WHC are examples of System containers, and Docker, LXC, WSC are examples of the application container. Container manager is further divided into on premise and managed. Docker, OpenVZ, rkt are applications of On-premises and ECS, GCE, ACS come under Managed container manager [2][4]. Docker is a lightweight virtualization technology that allows applications to run in isolated environments called containers, which share the host operating system's kernel.

The paper Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm [4] seeks to address the challenges and opportunities of designing distributed systems in multiple clouds using Docker Swarm, a container-based clustering tool. This article presents a simulation of the development and evaluation of a Docker Swarm-based distributed system that can be replicated across multiple clouds. It provides an overview of the key characteristics and constituents of Docker Swarm, including swarm mode, services, tasks, nodes, managers, workers, networks, load balancers, secrets, configs, stacks, and orchestration. The authors of this paper measured and analysed the high availability and fault tolerance of the system by simulating node failures and service disruptions. They also evaluate the automatic scalability, load balancing, and maintainability of services by simulating service updates and scaling operations.

VMs are software emulations of physical machines that run on top of a hypervisor, which provides an abstraction layer between the hardware and the guest operating systems. The paper [2] builds on the existing literature that has explored the advantages and disadvantages of containers and VMs for cloud computing. Some previous studies have focused on the implementation and deployment aspects of containers and VMs, such as Kominos et al., who present an overview of different container platforms in OpenStack, and Higgins et al., who discuss the orchestration of Docker containers in the high-performance computing environment. Other studies have conducted empirical evaluations of the performance and resource consumption of containers and VMs, such as Morabito et al., who compare the CPU, memory, disk, and network performance of Docker, LXC, and OSv containers with KVM VMs, and Rad et al., who analyze the CPU, memory, disk I/O, and network I/O performance of Docker containers with VMware VMs.

The paper also contributes to the literature by providing a comprehensive and updated performance comparison of Docker containers and KVM VMs using standard benchmark tools such as Sysbench, Phoronix, and Apache benchmark4. It also includes two operation speed measurements based on the eight queen problem and the eight puzzle problem, which are not commonly used in previous studies. Authors finds that Docker containers outperform KVM VMs in every test, which is consistent with most of the previous findings. However, they also acknowledges some limitations of its methodology, such as the use of only one host machine, one guest operating system, and one base image for Docker containers.

In contrast with many other open-source container cluster management technologies, Docker Swarm functionality is still evolving. With so many people using Docker, the swarm will soon have access to all the best capabilities that various devices have to offer. The docker swarm ongoing is a useful production system provided by Docker.[4]

Within this landscape, cloud container-based orchestration frameworks play a pivotal role. Notable contenders include Kubernetes, Docker Swarm, and Apache Mesos. These frameworks facilitate the management and coordination of containerized applications, enabling seamless deployment and scaling. Kubernetes is the ongoing widely accepted container orchestration framework which is open source in nature. This is an orchestration framework system for scaling up the applications and automating the applications deployments for the management purpose [7]. This framework makes the development process simpler by reducing the downtime of applications, increasing the security of the application and also to automate the scaling of the applications. Kubernetes includes six levels of abstraction. The Deployment specifies the required strategies to restart the pods when it fails. It creates a replica set which in turn creates the scale pods depending on the triggers that have been specified in the deployment. Kubernetes includes a basic building block called Pod. They handle the volumes and configurations for the specified containers because they live on the following worker nodes [10].

The aim to solve the scaling problem was given by the University of California in 2010. The solution for this is widely known as Apache Mesos. Mesos abstracts the CPU, memory and the essential disk resource in a way that allows the data centers to function as if they were on one specific large machine. Mesos itself creates a single underlying cluster to provide applications with the essential resources that are required without the overhead of the virtual machines and operating system. Mesos provides a feature of fault tolerance and also enables resource sharing for the distributed applications. The architecture of Mesos includes three components; they are: master, slaves and the applications that are going to run on them. Also, Mesos basically relies on the Apache Zookeeper for leader election within the ecosystem cluster and it also detects the leader for the Mesos slaves, masters and the other relying frameworks such as Marathon [7].

To evaluate and benchmark the performance of the container orchestration tools: especially from the perspective of their associated overheads, we adopt a two- step strategy. In the first step we use a well-known benchmarking test suite to measure the performance of a single container running on a Single worker node- This allows us to subsequently assess the overheads that are introduced at the worker node by each container orchestration tool. Then: we use an application that is implemented by multiple containers running over a cluster of Cloud-based worker nodes. This allows us to also assess the overheads that are introduced when containers need to communicate with each other. In both cases: the experiments are also executed in a system that is manually configured using only the underlying container technology, i.e, Docker, without any orchestration support. By comparing the performance of this baseline setting against the performance that is achieved when a given orchestration tool is used, it is possible to estimate the performance overheads associated with the tool. The next two subsections give more details about the benchmarks that were executed. [12]

Failover is a backup operational mode in which a secondary component takes over a system component's functions when the primary component is rendered unavailable due to failure or planned downtime. Mission-critical systems must include failover. Restoring the container service is allowed by rescheduling and it has been natively supported by container orchestration frameworks like Kubernetes and Apache Mesos. time with respect to the container and the host. The x-axis refers to the time. Fig. 4 depicts that the failover time for Cattle and Docker swarm remains the same but in terms of the remaining container orchestration tools such as Mesos and Kubernetes, Kubernetes shows a lesser failover time compared to Mesos because the Kubernetes agent here guarantees the availability of running the specified containers, and additionally, a replication controller is in charge of ensuring the rescheduling of unsuccessful pods. It can also be analysed that Mesos stands next to Kubernetes.

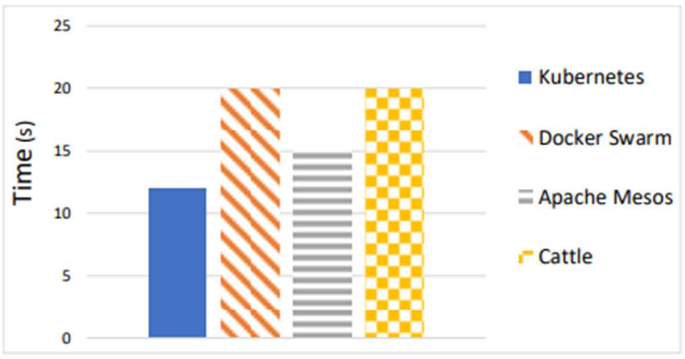

Figure 4 Container Failover Time

In terms of host failure, Fig. 5 is a clear indication that Kubernetes has the highest host failover time compared to the other orchestration tools. This is because Kubernetes has one of the most complex architectures in nature. This kind of orchestration methodology is event based and the Kubernetes controller will notify the objects whenever there is any change in the number of pods in the overall cluster.

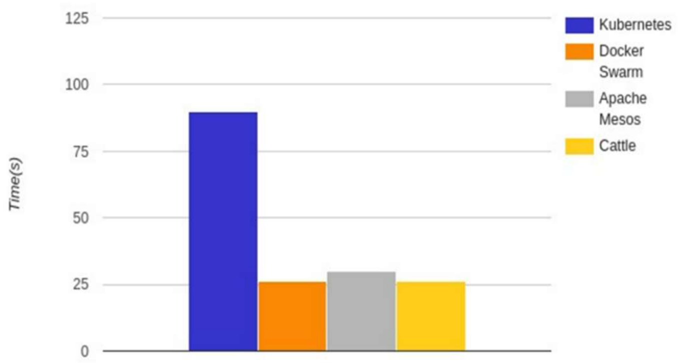

Figure 5 Host Failover Time

Elastic scalability is one of the core properties to be granted by cloud computing that has been boosted and facilitated by the advent of container-based technologies that enable fast bootstrapping. Along with that direction, the first performance metric evaluates the provisioning time required to provision a new cluster and to properly configure it with the required container orchestration support. Indirectly, this is also a measure of the container orchestrator's complexity.[13]

To compare their results, a comprehensive set of metrics has also been created. In terms of functional comparison, Kubernetes stands out as one of the modern market's most comprehensive orchestrators. That is why professionals choose to pick it over the alternative approaches. Additionally, due to specific instances of its high overhead, brought on by its sophisticated architecture, its performance may suffer. Both Kubernetes and Mesos aim to simplify the deployment and management of applications in data centres or cloud containers. They both provide support for businesses of various sizes.[8]

CHAPTER 3
ORCHESTRATION TOOLS: DESCRIPTION AND COMPARISON

3.1. CLOUD DEVOPS

Cloud DevOps is a software development methodology that blends cultural concepts, practices, and technologies to improve the speed and efficiency of producing apps and services in the cloud. It fosters collaboration and automation to integrate development and IT operations. It also facilitates the move from a product to a service economy by allowing for frequent upgrades and quick product innovation. Cloud DevOps requires developers and operators to collaborate and leverage the cloud as a common language.

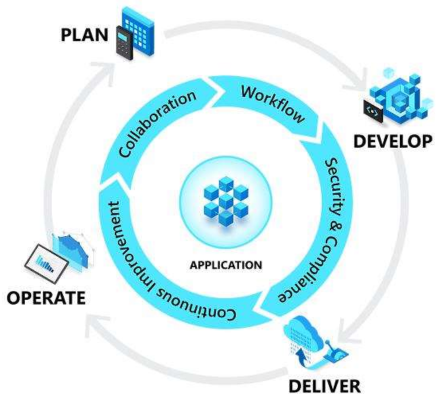

Figure 6 DevOps cycle to develop an application.

DevOps profoundly affects the phases of the application lifecycle, including planning, building, delivery, and operation. Each phase is dependent on the others, and the phases are not role specific. In a true DevOps culture, each role is involved in each phase to some extent.

While adopting DevOps practices automates and optimizes processes through technology, it all starts with the culture inside the organization—and the people who play a part in it. The task of developing a DevOps culture necessitates significant changes in how people work and collaborate. However, when organisations commit to a DevOps culture, they can foster the development of high-performing teams.

3.2 DEVOPS PRACTICES

DevOps teams can use multiple frameworks and approaches to improve the success of new software solution deployments. Here are some examples of commonly used frameworks and how they might be used:

CI/CD (Continuous Integration/Continuous Deployment):

- To deliver software more quickly and reliably, CI/CD advocates automating the development, testing, and deployment phases.
- DevOps teams can utilise CI/CD pipelines to ensure that every change to the code is completely tested and deployed in the same consistent manner by automating the building, testing, and deployment of software. As a result, any problems with the software can be fixed before it is released.
- Jenkins, GitLab CI/CD, and CircleCI are popular CI/CD tools/frameworks.

IaC (Infrastructure as Code):

- IaC is a paradigm for managing all of an organization's infrastructure as code, including servers, networks, and storage.
- DevOps teams can use IaC frameworks like Terraform and AWS CloudFormation to ensure that their infrastructure is consistent and reliable. This can help to lessen the chance of deployment mistakes.
- This makes it easy to automate infrastructure deployment and management, as well as trace changes and roll back to earlier configurations.

Configuration Management:

- Configuration management frameworks (for example, Ansible, Chef, and Puppet) aid in preserving consistency across several environments and automating software configuration setup.
- Configuration management solutions can help DevOps teams guarantee that the necessary software dependencies, configurations, and settings are applied correctly during deployments.
- These frameworks assist in reducing manual errors and enforcing uniform setups across multiple environments.

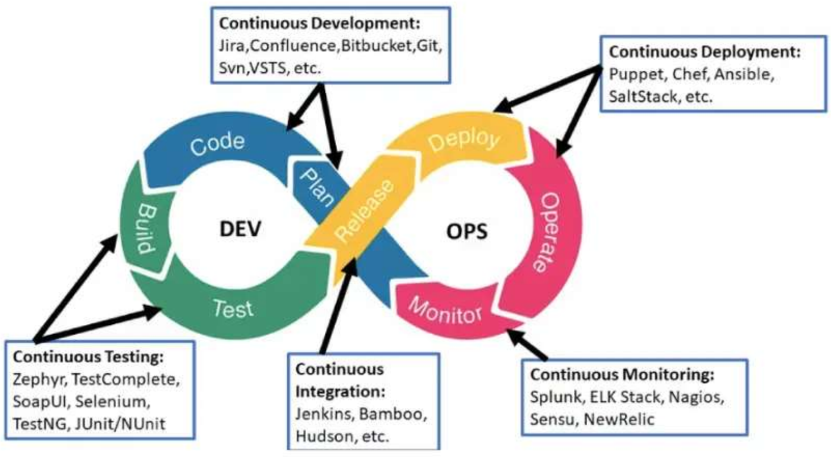
Continuous Development: Jira,Confluence,Bitbucket,Git, Svn,VSTS, etc.
Continuous Deployment: Puppet, Chef, Ansible, SaltStack, etc.
Code
Plan
Build
Test
DEV
Release
Deploy
OPS
Operate
Monitor
Continuous Testing: Zephyr, TestComplete, SoapUI, Selenium, TestNG, JUnit/NUnit
Continuous Integration: Jenkins, Bamboo, Hudson, etc.
Continuous Monitoring: Splunk, ELK Stack, Nagios, Sensu, NewRelic

Figure 7 DevOps Practices

Monitoring and Alerting:

- Monitoring frameworks like Prometheus, Nagios, and Datadog allow for the collection of metrics as well as the monitoring of the health of the deployed software.
- Monitoring and alerting systems can be set up by DevOps teams to proactively identify and address issues that may develop during or after deployments.
- Monitoring indicators like as response times, error rates, and resource utilization assists in ensuring the software's performance and reliability.

Incident Management:

- Incident management frameworks (for example, ITIL, Atlassian Jira Service Management) provide standardized processes for effectively handling issues and outages.
- DevOps teams can use incident management practices to quickly respond to and resolve issues that arise during or after deployments.
- Teams may minimise downtime and maintain service levels by having well-defined processes and clear communication routes.

Agile/Scrum:

- Scrum and other agile approaches emphasise iterative and incremental development, which promotes cooperation and adaptation.
- Agile principles can be used by DevOps teams to enable frequent product releases, continuous feedback loops, and strong communication between development and operations.
- Agile frameworks give a disciplined approach to software development and deployment, allowing teams to adjust to changing needs and reliably deliver value.

3.3 ORCHESTRATION TOOLS AND CONTAINERISATION

Orchestration tools are pieces of software that automate the configuration, coordination, integration, and data management procedures for a variety of applications and systems. Orchestration technologies can be used to increase the productivity of IT activities such as server provisioning, incident management, cloud orchestration, database administration, application orchestration, and many more. Orchestration tools like Docker Swarm, Kubernetes, and Apache Mesos help manage and scale containerized applications across multiple hosts.

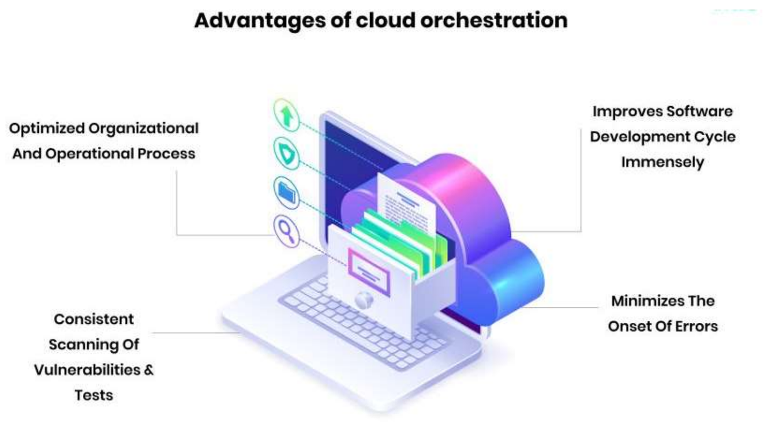
Advantages of cloud orchestration
Optimized Organizational And Operational Process
Improves Software Development Cycle Immensely
Minimizes The Onset Of Errors
Consistent Scanning Of Vulnerabilities & Tests

Figure 8 Advantages of cloud orchestration

A container is a standard unit of software that wraps up code and all its dependencies so that the programme can be moved from one computing environment to another quickly and reliably.

Containers are frequently compared with virtual machines (VMs). Containers, like virtual machines, allow you to package your application with libraries and other requirements, resulting in isolated environments for operating your software services. The similarities, however, end here, as containers provide a significantly more lightweight unit for developers and IT Ops teams to deal with, with a slew of advantages:

- Containers are substantially lighter than virtual machines.
- Containers virtualize at the OS level, whereas virtual machines (VMs) virtualize at the hardware level.
- Containers share the operating system kernel and use a fraction of the RAM that VMs do.

3.4 COMPARATIVE STUDY ON THE ORCHESTRATION TOOLS

To manage and automate the deployment, scaling, and management of containerized applications, orchestration tools are used in conjunction with containerization technology. Containers are lightweight virtualization instances that are self-contained and can be used to encapsulate an application and its dependencies into a single unit that can operate on any machine. They offer applications with a consistent runtime environment, allowing them to run in a range of environments without modification.

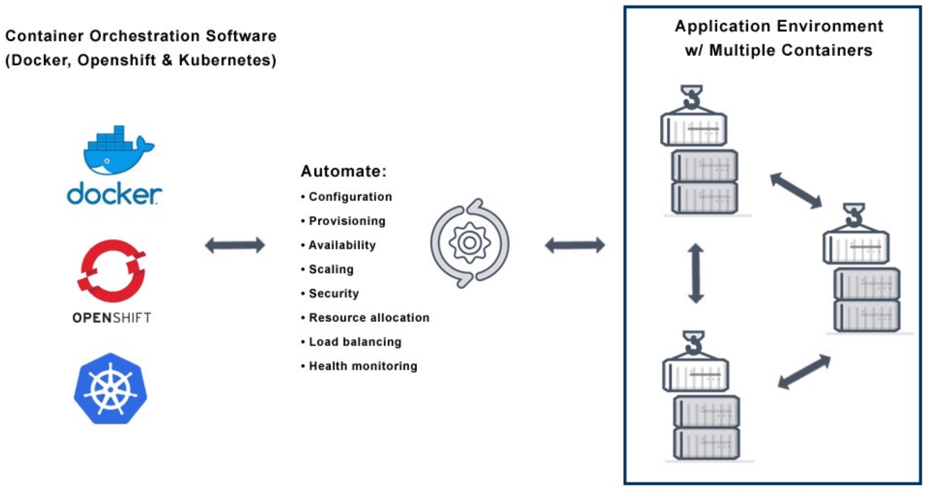

Figure 9 Orchestration tools process which they automate.

Container orchestration platforms such as Docker Swarm, Kubernetes, and Apache Mesos aid in the management and scaling of containerized workloads across multiple hosts. These technologies make it easy to deploy and maintain containerized applications at scale while simultaneously delivering containerization's flexibility and portability.

Docker Swarm

Docker Swarm is Docker's native clustering and orchestration solution. Initially, the docker came with a system of cluster management with a specific communication protocol called Beam. Later, more APIs were added and renamed as a swarm. Thus, the first-generation swarm is known as swarm v1.

It enables the creation and management of a swarm of Docker nodes, transforming them into a single virtual Docker host. Docker Swarm makes it effortless to deploy, scale, and manage containerized applications across several hosts for high availability and scalability.

Companies that adopted docker swarm:

- **Udemy** uses Docker Swarm to deploy its online learning platform. Docker Swarm helps Udemy to ensure that its platform is always available and that it can be easily scaled up or down as needed.
- **Pinterest** uses Docker Swarm to deploy its image sharing platform. Docker Swarm helps Pinterest to improve the performance and reliability of its platform.
- **Lockheed Martin** uses Docker Swarm to deploy its Orion platform, which is a cloud-based platform for managing military operations. Orion uses Docker Swarm to ensure that its applications are always available, even in the event of a node failure.
- **Spotify** uses Docker Swarm to deploy its music streaming service. Docker Swarm helps Spotify to ensure that its service is always available and that it can be easily scaled up or down as needed.

There are several components in the docker swarm each of them as follows:

Docker is a software platform that allows software developers to quickly include container use in the programme development process.

The application serves as a command and control interface between the host operating system and containerized applications.

Figure 10 Docker and Docker swarm icon logo

The docker application's major focus is on containers and their use and administration in the software development process. Containers enable developers to bundle apps with all the code and dependencies required to work in any computing environment. A container is launched in the Docker application by running an image.
An image is a collection of executable files that includes all of the code, libraries, runtime, binaries, and configuration files required to operate an application. A container can be thought of as an image's runtime instance.
There are two types of services in docker swarm: replicated and global.

Replicated services: Swarm mode replicated services functions based on a specified number of replica tasks for the swarm manager to assign to available nodes.
Global services: Global services function by using the swam manager to schedule one task to each available node that meets the services constraints and resource requirements.
Working Architecture of Docker Swarm

Docker Swarm follows a master-worker architecture where the master node manages the worker nodes. Docker Swarm's architecture consists of the following components:

Manager Nodes: These are the nodes in charge of managing the worker nodes and orchestrating service deployment across the cluster. Manager nodes keep the cluster running and handle failover in the event of a node failure.

Worker Nodes: These are the nodes that carry out the tasks that the manager node assigns to them. They are in charge of running the containers and supplying the resources required to operate the services.

Service: A service is a collection of tasks that are distributed across the cluster. Multiple replicas of service can run on different nodes.

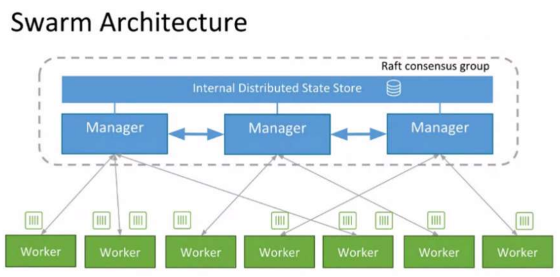

Figure 11 Swarm architecture diagram

Features of Docker Swarm

- Load Balancing Built-in: Docker Swarm includes load balancing capabilities. It distributes incoming traffic across the service's containers, removing the need for external load balancers.
- Self-healing: Docker Swarm regularly monitors the health of the cluster's containers and nodes. When a container or node becomes unhealthy, it is immediately rescheduled or replaced, ensuring that the application stays operational.
- Ease of Use: Docker Swarm is part of the Docker ecosystem and integrates seamlessly with Docker CLI and Compose. It makes use of current Docker principles, making it simple for users who are already familiar with Docker to get started with orchestration.
- Scalability and Availability: Docker Swarm provides your applications scalability and high availability by automatically dividing workload among available nodes and duplicating services across many nodes. This makes sure your applications remain functional even if one or more nodes in the cluster fail.

Use case scenario.

Docker swarm is an easy-to-use orchestration solution its for companies that prefer a straightforward way to manage containerized applications without the complexity of other orchestrators like Kubernetes.

Docker Swarm is ideal for smaller deployments or applications that don't need the broad functioning and scalability of larger orchestrators. It can manage a few to hundreds of nodes efficiently.

Docker Swarm is also preferred when we need to quickly deploy and manage containerized applications without spending significant time on configuration and setup, Docker Swarm provides a streamlined approach.

Kubernetes

Kubernetes, often known as K8s, is an open-source container orchestration technology that automates containerized application deployment, scaling, and management. Google created it, and the Cloud Native Computing Foundation (CNCF) now maintains it.

Companies that adopted Kubernetes:

- **The New York Times**: The New York Times uses Kubernetes to run their website and mobile apps. Kubernetes helps it to improve the performance and availability of their applications.
- **Netflix**: Netflix uses Kubernetes to manage their streaming service. Kubernetes helps Netflix to embed features; swipe, skip forward, log in, pay bills all these actions are handled by different dedicated Micro services. A concept that Netflix pioneered and all of them are held in Virtual Containers.
- **Uber**: Uber uses Kubernetes to manage their ride-hailing service. Kubernetes helps Uber to scale their service to meet the demands of their riders and drivers.
- **Robinhood**: Robinhood uses Kubernetes to manage their trading platform. Kubernetes helps Robinhood to scale their platform to meet the demands of their users.

Figure 12 Kubernetes icon logo

Kubernetes objects are entities that persist in the Kubernetes system. These entities are used by Kubernetes to reflect the state of your cluster. A Kubernetes object is a "record of intent"--once created, the Kubernetes system will work continuously to guarantee the object exists. You're effectively telling the Kubernetes system what you want your cluster's workload to look like by generating an object; this is your cluster's desired state. To work with Kubernetes objects, whether creating, modifying, or deleting them, you must use the Kubernetes API. When you use the kubectl command-line interface, for example, the CLI handles all of the necessary Kubernetes API calls.

Working Architecture of Kubernetes

A Kubernetes Cluster consists of Worker Machines known as Nodes and a Control Plane. There is at least one worker node in a cluster. The Kubectl CLI interacts with the Control Plane, which manages the Worker Nodes.

Master Node: The master node is in charge of monitoring the Kubernetes cluster. It is made up of various parts they are as follows:

API Server: The API server is the cluster's central control point for administering and controlling it. It makes the Kubernetes API available to users and other components, allowing them to communicate with the cluster.

Scheduler: Workloads (containers) are assigned to worker nodes by the scheduler based on resource availability, limitations, and other policies.

Control Manager: Controller Manager: The controller manager manages the cluster-level functions of node and pod lifecycle management, scaling, and monitoring.

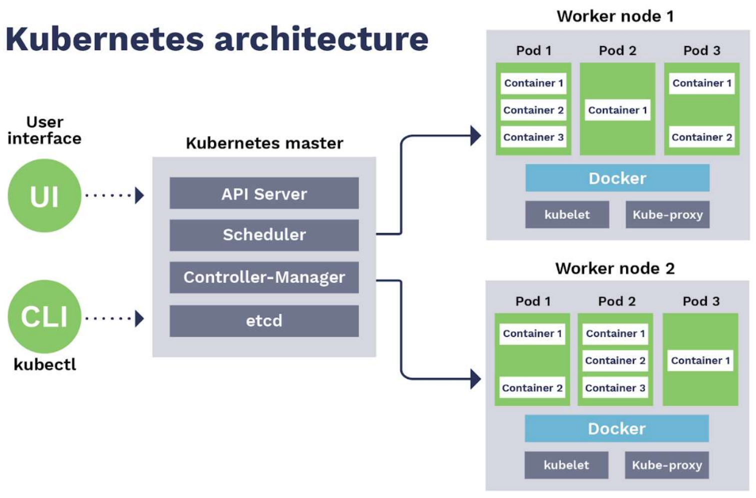

Figure 13 Kubernetes cluster architecture

etcd: etcd is a distributed key-value store that serves as the data store for the cluster. It stores the cluster's setup, state, and other metadata.

Worker Nodes: Worker nodes, also known as minions or worker machines, are the machines that deploy and run containers. Components of the worker node as follows:

Kubelet: A kubelet is a worker node agent that communicates with the master node. It is in charge of container management, initiating and stopping pods, and reporting the node's status.

Container Runtime: Kubernetes supports many container runtimes, including Docker, containerd, and CRI-O. The container runtime is in charge of fetching and running container images.

Kube-proxy: kube-proxy is in charge of network proxying and distributing load inside the cluster. It controls network rules and ensures service communication.

Pods: The smallest and most fundamental unit of Kubernetes. A pod is a collection of one or more containers that are co-located and share resources such as network and storage. The Kubernetes master schedules and manages pods.

Features of Kubernetes

Apart from the feature that docker swarm holds here are some extra features that Kubernetes was introduced with:

- Service Discovery and Load Balancing: Kubernetes assigns each service a stable IP address and DNS name, allowing other services to discover and communicate with them. Traffic is distributed across service replicas using the built-in load balancer.
- Rolling updates and rollbacks: Kubernetes allows you to change containers and applications without downtime. It progressively swaps out old containers with new ones. In the event of a problem, Kubernetes allows for simple rollbacks to prior versions.
- Declarative Configuration: To define the desired state of the system, Kubernetes uses declarative YAML or JSON configuration files. It constantly reconciles the current state with the desired state, making complicated applications easier to manage and maintain.

While Kubernetes is powerful and feature-rich, it isn't always the best choice for every scenario. Its complexity might be overkill for small projects or teams without the bandwidth to manage its intricacies. In such cases, simpler solutions like Docker Swarm or platform-as-a-service (PaaS) offerings might be more appropriate. However, for enterprises looking for a scalable, robust, and versatile container orchestration solution, Kubernetes often stands out.

Use case Scenario

Kubernetes is used in various scenarios depending on the company and application scale some of the use cases are mentioned below:

Kubernetes is especially proficient at handling complicated applications and large-scale deployments. Kubernetes provides the necessary capabilities for a distributed system with several services that require scalability, fault tolerance, and load balancing.

Kubernetes is well-suited for microservices-based architectures, where applications are broken down into smaller, independently deployable services. It enables you to manage and scale each service independently, allowing for greater flexibility and isolation.

Kubernetes integrates well with continuous delivery pipelines and allows for rolling updates, canary deployments, and automatic scalability. It helps to foster a DevOps culture by allowing teams to automate application deployment, testing, and monitoring.

Kubernetes is interoperable across multi-cloud and hybrid environments, which allows you to deploy applications across several cloud providers or on-premises infrastructure. It offers a consistent administrative interface and assists in avoiding vendor lock-in.

Apache Mesos

Apache Mesos is an open-source cluster manager that handles workloads in a distributed environment through dynamic resource sharing and isolation. Mesos brings together the existing resources of the machines/nodes in a cluster into a single pool from which a variety of workloads may utilize. Also known as node abstraction, this removes the need to allocate specific machines for different workloads. Mesos is suited for the deployment and management of applications in large-scale clustered environments.

Companies that adopted Mesos:

- **Apple** uses Apache Mesos to run its Siri backend.
- **The Walt Disney Company** uses Apache Mesos to run its streaming media infrastructure.
- **Twitter** uses Apache Mesos to run its Hadoop and Spark clusters. Hadoop and Spark are used by Twitter to process large amounts of data, and Mesos helps to ensure that these clusters can scale to meet the demands of Twitter's massive user base.
- **eBay** uses Apache Mesos to run its Docker containers. Mesos provides a unified API for managing both containers and traditional applications, making it a good choice for organizations that want to adopt a container-based infrastructure.
- **Netflix** uses Apache Mesos to run its Mantis real-time event processing system. Mantis is responsible for processing billions of events per day, and Mesos helps to ensure that Mantis can scale to meet the demands of Netflix's ever-growing user base.

Apache

MESOS™

Figure 14 Apache Mesos icon logo

Working Architecture of Apache Mesos

Apache Mesos follows a master-slave architecture, where the master node manages the cluster, and the slave nodes (also known as agents) execute tasks. Apache Mesos uses master-agent architecture in conjunction with frameworks to manage and isolate resource requests. The agent daemons run on each cluster node. Mesos also utilizes Apache Zookeeper, part of Hadoop, to synchronize distributed processes to ensure all clients receive consistent data and assure fault tolerance. Each framework consists of at least two crucial components: a scheduler and executor. Schedulers register with the Mesos master to get resources, and executors launch the command or program that runs tasks on the slaves.

Master Node: The master node is in responsible for monitoring the Mesos cluster. It is composed of up of various parts:

Mesos Master: The Mesos Master is the primary control point for the cluster, managing resources and scheduling tasks. It keeps track of available resources and decides how to allocate them among frameworks.

Zookeeper: Zookeeper is used for leader election and cluster state storage. It contributes to the Mesos master's high availability and fault tolerance.

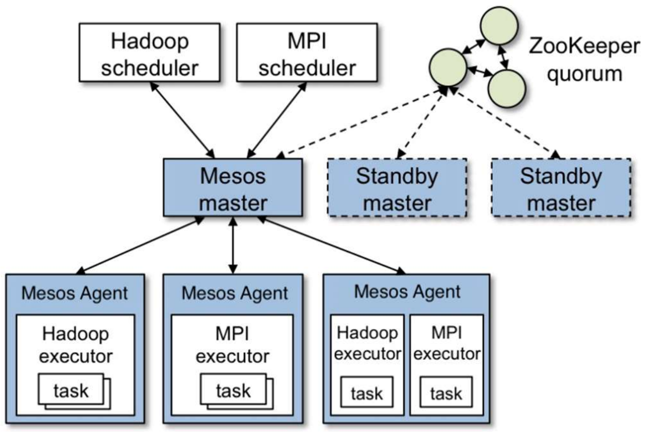

Figure 15 Mesos Architecture diagram

Slave Nodes: Slave nodes, also known as agents, are the machines in a cluster that perform tasks. Each slave node is equipped with the following components:

Mesos Agent: The Mesos Agent runs on each slave node and communicates with the Mesos Master. It provides the master with available resources and completes the duties assigned to it.

Containerizes: Mesos supports a variety of containerizes, including Docker and rkt, which enable task isolation. Containerizes enable tasks to execute in isolated contexts where they do not interfere with other tasks or the host system.

Frameworks: Frameworks are applications or services that operate on top of Mesos to make use of its resources. Frameworks can be built by users or supplied with Mesos. Frameworks can request resources from the Mesos Master, and if those resources are available, they can launch tasks on Mesos Agents to execute workloads.

Features of Apache Mesos

Apache Mesos provides several features that distinguish it from other resource management and cluster orchestration tools Some of Mesos' features include multi-resource scheduling.

- Dynamic Resource Allocation: Mesos supports dynamic resource allocation, which allows frameworks to scale their resource usage based on demand. It enables efficient resource utilisation by assigning resources to frameworks only when they are required.
- Heterogeneous Workload Support: Mesos can handle a wide range of workloads, including batch processing, data processing frameworks (such as Apache Spark), containerized applications (such as Docker containers), and more. It offers a single platform for handling various workloads.
- Fault Tolerance: Mesos includes built-in fault tolerance techniques. The use of Zookeeper for leader election and state storage guarantees that the master node is always available. Mesos can recover and reschedule work on healthy agents if a master or agent fails.
- Flexible Scheduling: Mesos supports flexible scheduling techniques, including support for various scheduling strategies. It enables frameworks to specify how tasks are assigned to agents based on resource needs and limits by using custom schedulers or leveraging Mesos' default scheduler.

Use case Scenario.

Apache Mesos is well-suited for applications that require dynamic scalability depending on varying workloads. Mesos can effectively manage and allocate resources in heterogeneous environments, such as clusters with machines of different CPU, memory, or storage capacities. It can also scale resources up or down based on demand, ensuring that applications have the necessary resources when needed. This makes Mesos a good choice for applications that require dynamic scaling based on fluctuating workloads.

Mesos is ideal for big data processing frameworks such as Apache Spark, Hadoop, and Flink. It can allocate resources and manage the execution of data-intensive workloads across a cluster efficiently.

Mesos allows multi-tenancy, which enables several frameworks or applications to live on the same cluster while remaining isolated from one another. It includes techniques for resource allocation and isolation to ensure equitable resource distribution among tenants.

CHAPTER 4
RESULTS

4.1 PERFORMANCE METRICS OF THE TOOLS

This section aims to provide a more detailed overview of the performance metrics for container orchestration tools, including the results and discussion of the provisioning time for deploying container orchestration tools such as Mesos, Kubernetes, and Docker swarm, as well as metrics for container provisioning.

Provisioning time of the cluster

Cluster provisioning time is the amount of time it takes to create a cluster of computing resources, such as virtual machines or containers. This includes the time it takes to launch the instances, install the software, and configure the cluster. The provisioning time can vary depending on the size of the cluster, the type of instances being used, and the availability of resources in the cloud.

The Test was done in a Cloud environment; AWS Cloud I used Virtual machines (EC2 instances) and configured them as follows: OS installed was Ubuntu 22.04 LTS, the Number of CPUs was 2, the Memory was 4 GB, the Storage was 15 GB and the Storage type was EBS.

During my testing, I discovered that Docker Swarm reduced complexity and reduced provisioning time. The time required to deploy container orchestration tools is depicted in Figure 13. The x-axis represents the amount of time these container orchestration tools are used for setup and deployment.

Because the entire architecture is optimized here to execute deploying, the Docker swarm tool takes less time to deploy an application. Mesos framework takes longer than Docker Swarm, but it takes less time to deploy an application than Kubernetes.

So, it can be concluded that Kubernetes requires a longer provisioning time than any other peer framework, this is because of its architecture, which is one of the most complicated in nature.

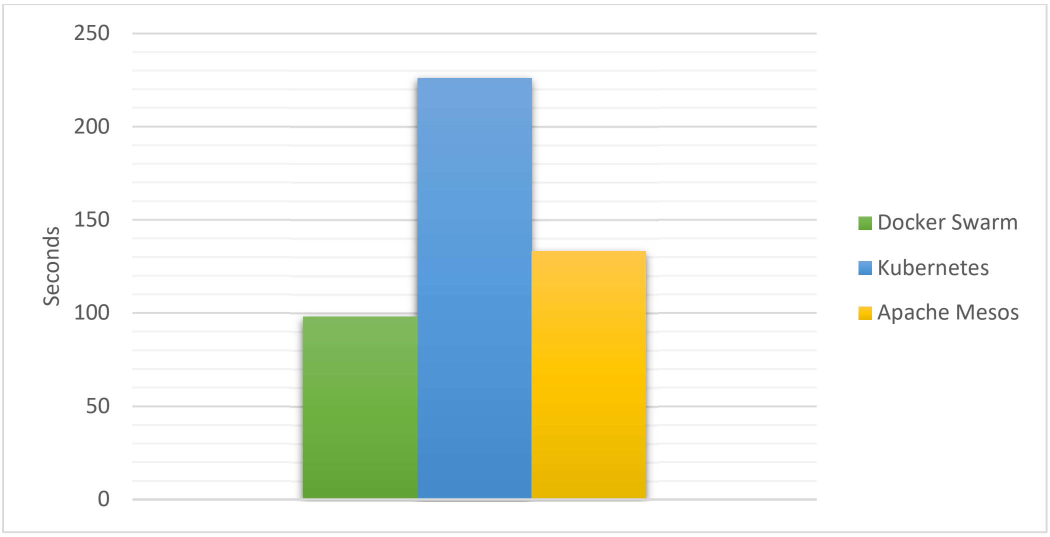

Figure 16 Provisioning time of cluster using various orchestration tools.

In practical terms, Kubernetes is the solution with the most management and deployment options for production-level services. Instead, Docker Swarm and Apache Mesos demonstrated faster provisioning time due to their lower complexity.

Breakdown of time taken for each component in orchestration tools

Component	Time Taken
Master node	76 seconds
Worker node(s)	40 seconds per node
Node components (Pod, kubelet)	46 seconds
Control plane components (Service)	57 seconds
Ingress	13 seconds
Total	**232 seconds**

Table 1 Time taken for each component in K8s

Table 1 provides insights into the time durations associated with various Kubernetes components, which can be utilised to better understand the performance or setup timeframes of a Kubernetes environment.

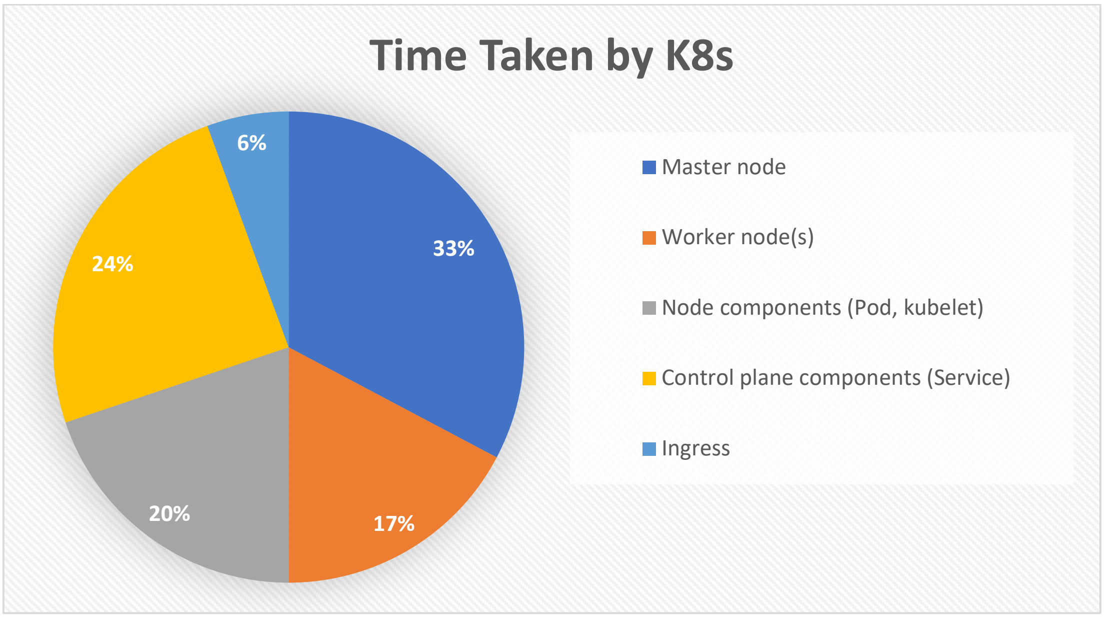

Figure 17 Time Distribution of K8s in a pie chart

Based on my findings, there is a significant variation in the provisioning durations of clusters across different container orchestration tools. Docker Swarm demonstrated the highest level of time efficiency, which can be attributed to its comparatively simpler architecture. In contrast, Kubernetes required the most time to provision due to its complicated setup. The total provisioning time for Kubernetes was 232 seconds, of which 76 seconds were required for the establishment of the master node and 40 seconds were required for the addition of each worker node. An additional level of clarity is provided regarding the complexities of Kubernetes' provisioning process through the detailed schedules of individual components, such as node components and control plane components, as illustrated in Table 1 and illustrated in Figure 17.

Component	Time Taken
Master node	24 seconds
Worker node(s)	11 seconds per node
Node components	41 seconds
Service	46 seconds
Ingress	11 seconds
Total	**133 seconds**

Table 2 Time taken for each component in Apache Mesos

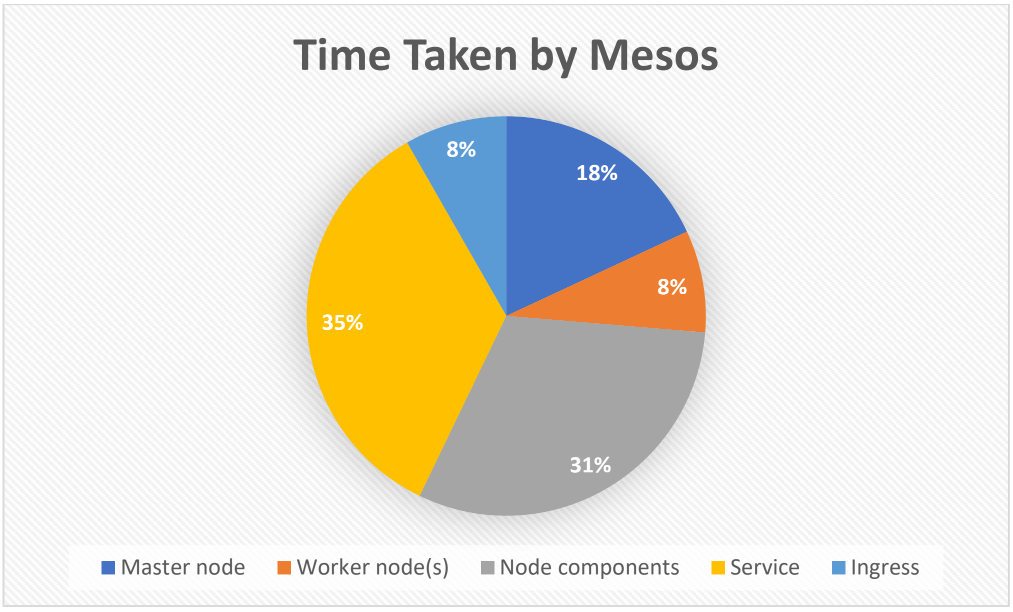

Figure 18 Time Distribution of Apache Mesos in a pie chart

From table 2, one can gather insights into the durations associated with various components of Mesos. Such data can be crucial for estimating the performance or initialization periods of a Mesos environment.

On the other hand, Apache Mesos exhibited a provisioning time of 133 seconds in its entirety, positioning it between Docker Swarm and Kubernetes with respect to complexity and time efficiency. This is supported by the data presented in Table 2 and Figure 18. The results of this study emphasize Kubernetes as the preferred platform in terms of flexibility in deployment and administration, although it compromises provisioning speed. In contrast, Docker Swarm and Mesos provide expedited setup times, which is essential in environments that prioritize rapid scaling or deployment.

4.2 COMPARISON OF THE TOOLS

A comparison is made out of a few attributes as follows:

- Architecture that these tools follow.
- Scaling capabilities of each tool
- Service Discovery capability of each tool
- Networking type followed by each tool.
- Load balancer whether integrated or not
- Availability, whether it supports high availability or not.
- Monitoring capabilities of each tool

- Security offered by each tool.
- Learning curve for come coming DevOps engineer.

Metrics	Docker Swarm	Kubernetes	Apache Mesos
Architecture	Single-host or multi-host cluster	Multi-node cluster with a master and worker nodes	Distributed system with a master and agent nodes
Scaling	Limited scaling capabilities	Excellent scaling capabilities	Excellent scaling capabilities
Service Discovery	Built-in service discovery	Built-in service discovery	Requires external tools
Networking	Overlay networking	Flexible networking options	Supports different networking plugins
Load balancer	Integrated	Integrated	External
Availability	Supports high availability	Built-in high-availability features	Supports high availability through failover
Monitoring	Basic monitoring capabilities	Rich monitoring and log features	Basics monitoring features
Security	Basic security features	Advance security features	Basic security features
Learning curve	Relatively easy to use	Steeper learning curve	Steeper learning curve

Table 3 comparison in between orchestration tools

4.3 STRENGTHS AND WEAKNESSES OF THE ORCHESTRATION TOOLS

Strengths and weaknesses of the three orchestration tools: docker swarm, Kubernetes and Apache Mesos is mentioned below as in a tabular form.

Strengths of Docker Swarm	Weaknesses of Docker Swarm
Simplicity and ease of use	Limited feature set
Integration with Docker	Less extensive ecosystem and community support
High availability	Limited scalability
Load balancing and scaling	Limited flexibility and customization
Self-healing capabilities	Learning curve for complex deployments

Table 4 Dockers Swarm strengths and weaknesses

Table 4 shows the pros and cons of Docker Swarm, one of the three orchestration tools that were tested along with Kubernetes and Apache Mesos. It indicates that Docker Swarm's best features are that it is simple and easy to use, that it works well with Docker, that it is highly available, that it can balance load and grow, and that it can fix itself. The downsides of Docker Swarm, on the other hand, are shown in Table 4. These include a limited set of features, a community and ecosystem that aren't as strong as those of its competitors, limited scalability, limited flexibility and customization options, and a harder learning curve when it comes to managing complex deployments. While this comparison shows that Docker Swarm works effectively in certain circumstances, it also shows that it cannot cope with more difficult situations.

Strengths of Kubernetes	**Weaknesses of Kubernetes**
Scalability and High Availability	Complexity and Learning Curve
Rich Feature Set	Resource Intensive
Large and Active Community	Complexity of Networking and Service Discovery
Flexibility and Customization	Continuous Maintenance and Upgrades
Portability and Cloud-Native	Compatibility and Interoperability

Table 5 Kubernetes strengths and weaknesses

Table 5 shows a few Kubernetes's strengths, such as its high availability and excellent scalability, as well as its large and active community that supports it. It also talks about Kubernetes' portability and cloud-native nature, as well as its ability to be flexible and customised. All of these are big benefits for a variety of deployment environments. Along the same lines, Table 5 also shows the problems that Kubernetes has. Some of these are how hard the system is to understand and how long it takes to get good at it. This balanced overview from the table 5 is crucial for understanding the trade-offs involved when choosing Kubernetes as an orchestration solution.

Strengths of Apache Mesos	**Weaknesses of Apache Mesos**
Scalability and Fault Tolerance	Complexity and Learning Curve
Resource Efficiency	Lack of Native Features
Multi-Framework Support	Limited Ecosystem and Community
Flexibility and Extensibility	Administrative Overhead
Mature and Stable	Documentation and Learning Resources

Table 6 Apache Mesos strengths and weaknesses

Table 6 shows both positive and negative aspects regarding Apache Mesos. Mesos's strengths are its ability to grow and adapt to problems, its ability to make good use of resources, its ability to work with many frameworks, its adaptability, and its reputation for being a mature and stable platform. The downsides are that it's hard to learn and understand, doesn't have many built-in features, has a small community and ecosystem, has a lot of administrative work to do, and doesn't have a lot of documentation and learning materials. This brief overview helps you compare Mesos to its alternatives for managing resources in distributed systems.

4.4 RECOMMENDATIONS

If the primary requirement is orchestrating containerized applications for mission-critical operations with scalability, flexibility, and community support, Kubernetes is the top choice among the three. Kubernetes has become the de-facto standard in the industry for container orchestration due to its robust features, vast community, and backing from major cloud providers.

Docker Swarm is another popular container orchestration platform, but it is not as widely used as Kubernetes. Docker Swarm is a good choice for smaller organizations, it does not offer the same level of features or capabilities as Kubernetes.

Mesos is less focused on container orchestration than Kubernetes, and even this is not as widely used as k8s. But Mesos is a good choice for organizations that need to orchestrate a mix of containerized and non-containerized workloads, or that need more flexibility and control over their container orchestration platform.

However, the final decision should consider the existing infrastructure, team expertise, and specific requirements of any business organization. If the team is already well-versed in Docker and wants a simpler solution, Docker Swarm might suffice. If the organization has mixed workloads (both containerized and non-containerized) and wants a unified orchestrator, then Mesos could be the right choice.

CHAPTER 5
CONCLUSION

CONCLUSION

This study compares container orchestrators' capabilities, as well as the container orchestration structure. A thorough set of metrics has also been developed in order to compare their outcomes.

Kubernetes is one of the most flexible orchestrators on the market in terms of capability. This is why professionals choose it over various alternatives. However, due to the significant overhead created by its complicated architecture, its performance may suffer in some circumstances.

Both Kubernetes and Mesos aim to simplify application deployment and management in data centres or cloud containers. They both offer great support to businesses of various sizes. Kubernetes is an excellent option if you are starting from scratch. If you have legacy applications, Mesos is the superior choice. Finally, the ideal cluster management solution for your organization is determined by your existing and future requirements.

FUTURE WORK

The future scope of this study could involve several aspects:

Investigate methods to improve the performance of Kubernetes, particularly in scenarios where its complex architecture may cause overhead. Make another Performance metric test for host failover and container failover and generate the results. Conduct a detailed analysis of the security features and practices offered by different container orchestrators. Evaluate their effectiveness in protecting containerized applications and data and identify potential vulnerabilities.

REFERENCE

[1] *IEEE Standards Committee. (2021). IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment: IEEE Standard 2675-2021.https://ieeexplore.ieee.org/document/9415476*

[2] *M. Potdar, D. G. Narayan, S. Kengond, M. M. Mulla, "Performance Evaluation of Docker Container and Virtual Machine," Procedia Computer Science, vol. 171, pp. 1419-1428, 2020, doi: 10.1016/j.procs.2020.04. 152.*

[3] *C. Ruiz, E. Jeanvoire and L. Nussbaum, "Performance Evaluation of Container for HPC," in Parallel Processing Workshops, vol. 9523, pp. 813-824, 2015, doi: 10.1007/ 978-3-319-27308-2_65.*

[4] *N. Naik, "Performance Evaluation of Distributed Systems in Multiple Clouds using Docker Swarm," 2021 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 2021, pp. 1-6, doi: 10.1109/SysCon48628.2021.9447123.*

[5] *[5]L. Herrera-Izquierdo, and M. Grab, "A Performance Evaluation between Docker Container and Virtual Machines in Cloud Computing Architectures," MASKANA, CEDIA, 2017, vol. 7, pp. 127-133, 2017.*

[6] *Malviya, A., & Dwivedi, R. K. (2022, March). A comparative analysis of container orchestration tools in cloud computing. In 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 698-703). IEEE.*

[7] *Lokhande, S. R., & Kumar, S. A. (2022, October). Deployment Strategy using DevOps Methodology: Cloud Container based Orchestration Frameworks. In 2022 International Conference on Edge Computing and Applications (ICECAA) (pp. 113-117). IEEE.*

[8] *Kubernetes Components. (n.d.). Kubernetes. https://kubernetes.io/docs/concepts/overview/components/*

[9] *What Is DevOps? - DevOps Models Explained - Amazon Web Services (AWS), n.d.*

[10] *J. Ellingwood, " An Introduction to Kubernetes", Digit al Ocean, 14 October 2016. https://www.digitalocean.com/community/tutorials/an- introduct ionto- kubernetes.*

[11] *Baeldung. (n.d.). Mesos vs Kubernetes Comparison. Retrieved from Baeldung: https://www.baeldung.com/ops/mesos-kubernetes-comparison.*

[12] *Yao P an; Ian Chen, Richard Sinnott, 'A Performance Comparison of Cloud-Based Container Orchestration Tools' 2nd International Conference on Computing and Communications Technologies (ICCCT) Nov 2020*

[13] *I. M. A. Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6, doi: 10.1109/ICC.2019.8762053.*

[14] *Vincent Reniers, "The Prospects for Multi-Cloud Deployment of SaaS Applications with Container Orchestration Platforms", Middleware Doctoral Symposium'16,, article 5, 2 pages*

[15] *Wito Delnat et al., "K8-scalar: a workbench to compare autoscalers for container-orchestrated database clusters", in Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '18), pp. 33-39*