

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JNANA SANGAMA”, BELAGAVI – 590018, KARNATAKA



Seminar Report on

## “Kubernetes”

*Submitted in the partial fulfillment of the requirements for the award of the Degree of  
Bachelor of Engineering in Information Science and Engineering*

Submitted by

**Jyotsna B**

**1VA18IS010**

Under the support and guidance of

**Dr. Vrinda Shetty**

**HOD, ISE Dept,**

**SVIT**



**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**SAI VIDYA INSTITUTE OF TECHNOLOGY**

Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by  
AICTE, New Delhi)

Accredited by NBA, New Delhi (CSE, ISE, ECE, MECH, CIVIL)

**RAJANUKUNTE, BENGALURU-560064**

Tel: 080-2846 8196, Fax: 2846 8193 / 98, Web: [saividya.ac.in](http://saividya.ac.in)

**2021-22**

# SAI VIDYA INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belgaum, Approved by AICTE, New Delhi and Govt. of Karnataka)

Accredited by NBA, New Delhi (CSE, ISE, ECE, MECH, CIVIL)

Rajanukunte, Bengaluru-560064

## DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING



### Certificate

Certified that the Technical seminar (18CSS84) entitled “**Kubernetes**” was presented by **JYOTSNA B (1VA18IS010)**, a bonafide student of Sai Vidya Institute of Technology, Bangalore, in partial fulfillment for the award of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University, Belgaum during the year 2021-2022. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The seminar report has been approved as it satisfies the academic requirements in respect of seminar work prescribed for the said degree.

.....

**Abhijith H V**

Assistant Professor

Dept of ISE

.....

**Dr. Vrinda Shetty**

Professor & Head

Dept of ISE

.....

**Dr. H S Ramesh Babu**

Principal

# ACKNOWLEDGEMENT

The completion of Technical Seminar brings with a sense of satisfaction, but it is never completed without thanking the persons who are all responsible for its successful completion. First and foremost I wish to express my deep sincere feelings of gratitude to my Institution, **Sai Vidya Institute of Technology**, for providing me an opportunity to do my education.

I would like to thank the **Management** and **Prof. M R Holla**, Director, Sai Vidya Institute of Technology for providing the facilities.

I extend my deep sense of sincere gratitude to **Dr. H S Ramesh Babu**, Principal, Sai Vidya Institute of Technology, Bengaluru, for having permitted to carry out the Technical Seminar on “**KUBERNETES**” successfully.

I am thankful to **Prof. A M Padma Reddy**, Additional Director, Professor and Dean (Student affairs), Department of Computer Science and Engineering, Sai Vidya Institute of Technology, for his constant support and motivation.

I express my heartfelt sincere gratitude to **Dr Vrinda Shetty**, HOD and Professor, Department of Information Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for her valuable suggestions and support.

I express my heartfelt sincere gratitude to **Abhijith H V**, Guide, Assistant Professor, Department of Information Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for his valuable suggestions and support.

Finally, I would like to thank all the Teaching, Technical faculty and supporting staff members of Department of Information Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for their support.

Jyotsna B

1VA18IS010

# ABSTRACT

Earlier, applications were run on physical servers. This caused resource allocation issues as there was no way to define resource boundaries for applications in a physical server. If we run the applications on separate servers to solve this problem, resources would be underutilized, and it would be expensive for an organization. Virtualization solved this problem as it allows you to run multiple virtual machines on a single CPU server. Here, information on a single application cannot be accessed by another, hence providing security. It has provided better utilization of resources. Each Virtual Machine is a full machine running all the components and its operating system on a virtualized hardware. Containers are very similar to VMs but have the properties to share the operating system among other applications. They are a form of operating system virtualization. They are a good way to bundle and run your applications. They are packages of software and have all necessary components or elements to run in any environment and anywhere, i.e., from a private data center to the public cloud or on a person's laptop. Hence, they are considered as lightweight. Similar to a virtual machine, a container has its own filesystem, share of CPU, memory, process space and so on.

Kubernetes is a system that manages these containers or containerized applications. It can create and scale these containers automatically and manage storage among all the containers. A bunch of containers can be built, and Kubernetes can be used to manage those containers. It has a file which has the description of all the containers and how they work together. So, Kubernetes can keep track of your container applications that are deployed into the cloud. It provides you with a framework to run distributed systems resiliently. Kubernetes has a large, fast-growing ecosystem. Services, support and tools of Kubernetes are widely used.

The main strengths of Kubernetes are its generality and its modularity. Any kind of application that needs to be deployed can fit within Kubernetes. Understanding Kubernetes is important to make the development, management and deployment of the application easier and reliable.

# Table of Contents

---

<b>ACKNOWLEDGEMENT .....</b>	<b>I</b>
<b>ABSTRACT .....</b>	<b>II</b>
<b>TABLE OF CONTENTS.....</b>	<b>III</b>
<b>LIST OF FIGURES .....</b>	<b>V</b>

## **CHAPTER 1**

<b>INTRODUCTION.....</b>	<b>1</b>
1.1 KUBERNETES.....	2
1.2 CONTAINERS.....	2-3
1.3 CONTAINER ORCHESTRATION TOOL.....	3

## **CHAPTER 2**

<b>RELATED WORK .....</b>	<b>4</b>
2.1 KUBERNETES AND AMAZON EC2.....	4
2.2 AZURE KUBERNETES SERVICE.....	4
2.3 APACHE SPARK WITH KUBERNETES.....	5
2.4 BLOOMSBURG.....	5-6
2.5 GOOGLE KUBERNETES ENGINE.....	6
2.6 POKEMON .....	6-7
2.7 NEW YORK TIMES .....	7

## **CHAPTER 3**

<b>KUBERNETES.....</b>	<b>8</b>
3.1 ARCHITECTURE .....	8-13
3.2 ESSENTIAL CONCEPTS .....	13-15

## **CHAPTER 4**

<b>IMPLEMENTATION.....</b>	<b>16-21</b>
4.1 INSTALLATION AND SETUP.....	16-17
4.2 RUNNING CLUSTER.....	17-19
4.3 DEPLOYING APP.....	19-20
4.4 TESTING WITH CONTROL PLANE .....	20

## **CHAPTER 5**

<b>APPLICATIONS.....</b>	<b>21</b>
5.1 DEPLOYING A SIMPLE APP.....	21
5.2 MICROSERVICES ARCHITECTURE .....	21
5.3 LIFT AND SHIFT.....	22
5.4 CLOUD-NATIVE NETWORK FUNCTIONS .....	22
5.1 MACHINE LEARNING.....	23
5.5 CI/CD .....	23

## **CHAPTER 6**

<b>USECASES .....</b>	<b>24</b>
6.1 REDUCING TIMEFRAMES .....	24
6.2 OPTIMISING IT COSTS.....	24
6.3 INCREASED SCALABILITY AND AVAILABILITY .....	25
6.4 FLEXIBILITY IN MULTI-CLOUD ENVIRONMENTS .....	25
6.5 CLOUD MIGRATION PATHS.....	25-26

## **CHAPTER 7**

<b>ADVANTAGES AND DISADVANTAGES OF KUBERNETES.....</b>	<b>27</b>
7.1 ADVANTAGES.....	27
7.2 DISADVANTAGES .....	28

## **CHAPTER 8**

<b>CONCLUSION .....</b>	<b>29</b>
<b>REFERENCES .....</b>	<b>30</b>

## List of Figures

---

Figure 1.1	Kubernetes Cluster	2
Figure 1.2	Containerized Application	3
Figure 3.1	Architectural design of Kubernetes cluster	9
Figure 3.2	Control Plane Components	11
Figure 3.3	Architectural diagram of Node	12
Figure 4.1	Install Minikube CLI	16
Figure 4.2	Starting Minikube	17
Figure 4.3	Change in status	18
Figure 4.4	Replica of Application Instance	18
Figure 4.5	Containers running on the node	19
Figure 4.6	Make application Accessible	19
Figure 4.7	Creating proxy	20
Figure 4.8	Run new pod	20

## Chapter 1

# INTRODUCTION

Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers. As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

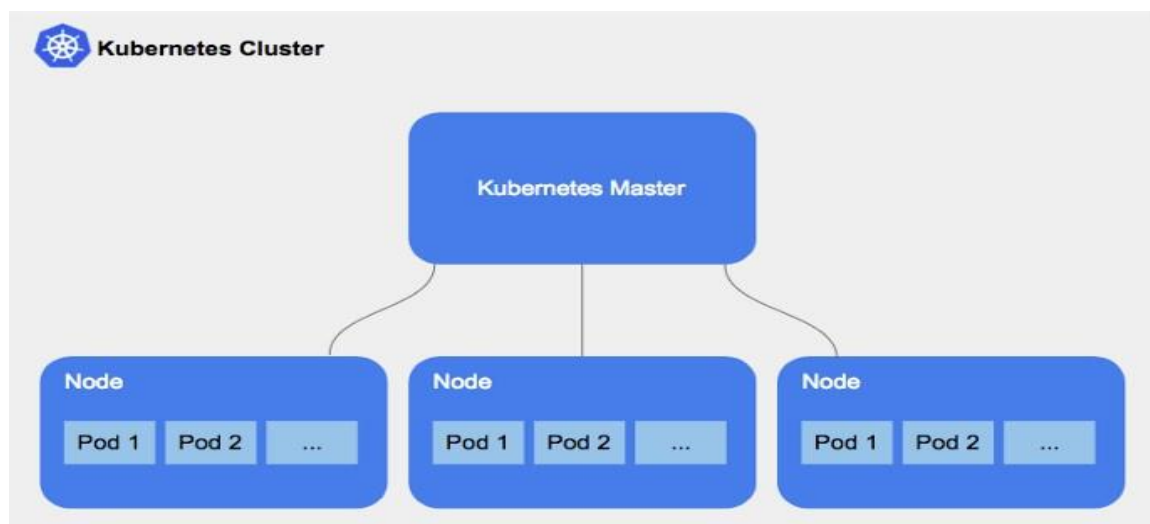
With virtualization you can present a set of physical resources as a cluster of disposable virtual machines. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware. Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions. Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime.

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system



## 1.1 Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. It was based on their Internal Platform called Borg. Borg was the predecessor to Kubernetes and the lessons learned from developing Borg over the years became the primary influence behind much of Kubernetes technology. Kubernetes cluster means multiple nodes are connected together to form a Kubernetes cluster.

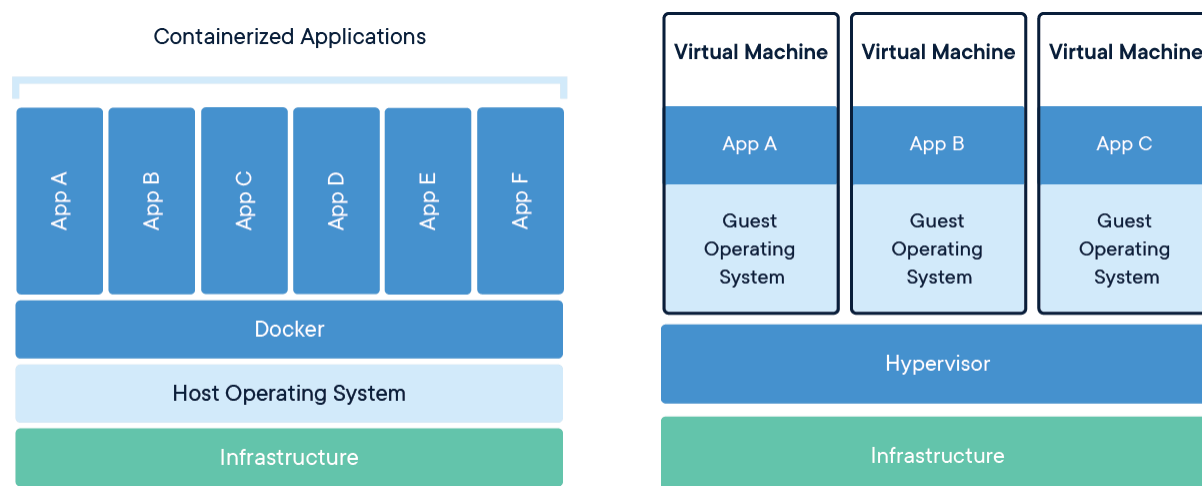


**Figure 1.1:** Kubernetes Cluster

## 1.2 Containers

A Container can be considered as an Isolated Process. There is no concrete primitive as a Container in Linux, but Containers can be considered as a group of isolated configurations applied to a Linux process. Docker is one of the popular tools, that makes it Unlike Virtual Machines, All Containers on the same host share the same Kernel, thus not having the overhead of start-up times, and performance which are common with Virtual Machines. Docker is a tool designed to make it easier to create, deploy, and run applications by using

containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package.



**Figure 1.2:** Containerized Application

### 1.3 Container Orchestration Tool

Container platforms like Docker are very popular these days to package applications based on a microservices architecture. Containers can be made highly scalable, which can be created on-demand. While this is good for a few containers but imagine you have hundreds of them. It becomes extremely difficult to manage the container lifecycle and its management when numbers increase dynamically with demand.

Container orchestration solves the problem by automating the scheduling, deployment, scalability, load balancing, availability, and networking of containers. Container orchestration is the automation and management of the lifecycle of containers and services

The main features of container orchestration tools are reliability, High Availability, Disaster Recovery and Scalability.

## Chapter 2

# RELATED WORK

### 2.1 Kubernetes and amazon EC2

Kubernetes manage clusters for Amazon EC2, compute instances as well as run containers on these instances using processes of deployment, maintenance, and scaling. This enables one to run any type of containerized application using the same on-premise and in-cloud toolset. Amazon Elastic Kubernetes Services (EKS) is a certified conformant that manages the Kubernetes control plane. AWS has collaborated with Kubernetes' community and making contributions to its code-base which helps users take advantage of AWS's features and services. This collaboration allows users to fully manage their Kubernetes deployment and utilize a powerful, community-back integration and get automatic provisions.

### 2.2 Azure Kubernetes Service

AKS is a completely managed service the enables Kubernetes in Azure, without having to manage Kubernetes clusters separately. Azure is able to manage all the complexities of Kubernetes, while one concentrates on the containers. Some basic features include Pay only for the nodes, easier cluster upgrades as well as Kubernetes RBAC and Azure Active Directory integration. A great example of AKS's real-life usage is Logicworks, which is a Microsoft Azure Gold Partner, that works with companies to help them migrate their applications to Azure. With the help of AWS, they were able to achieve portability across on-prem and public clouds. It also accelerated containerized application development, and unified development and operational teams on a single platform. AKS is a powerful service for running containers in the cloud, as companies only pay for the VMs and resources consumed. Solutions like AKS Quickstart is able to launch a test cluster within an hour.

## 2.3 Apache Spark with Kubernetes

Apache Spark is an open-source distributed computing framework. Spark helps manage a large number of data sets with the help of a cluster of machines. But it does not manage the machine, this is where Kubernetes helps. Spark creates its own driver which runs within the Kubernetes Pod. The driver then creates executors that also run within Kubernetes pods, connects them, and then executes the application code. Once the application is completed, the executioner pods are terminated and cleared, but the driver pods persist logs and remain in the “completed” state in the API. Kubernetes takes care of scheduling the driver and executor pod. Fabric8 is used to communicate to Kubernetes API. Kubernetes enables containerization which is helpful in traditional software engineering and big data as well as Spark. The containers make the application more portable, simplifies the packaging of dependencies, and build a reliable and secure workflow. In the recent version of Spark, Spark on Kubernetes is marked as Generally Available and production-ready in the official docs.

## 2.4 Bloomberg

Bloomberg, the financial data analytics company, began utilizing Kubernetes in 2015 when this tool was still very new in the market. Bloomberg processes a large number of data sets every day, working with 14,000 different applications.

Now, they wanted to decrease their processing time and free up operational tasks so that the developers could concentrate on tasks more beneficial to the company. Bloomberg went through a plethora of platforms namely Cloud Foundry, Mesosphere Marathon, and a variety of Docker offerings.

According to Andrey Rybka, head of the computing infrastructure in the office of the CTO at Bloomberg, “Kubernetes had a good foundation and it was clear they were confronting the right problems. You could see a vision and roadmap as to how it would evolve, that was aligned with what we were thinking.”

Kubernetes has enabled them to work on a PaaS layer which gives the developers the right level of abstraction to work with. The major objective for Bloomberg was to efficiently utilize the existing hardware they had already invested in. With Kubernetes, they were not just able to do that, but they were able to allocate data by themselves along with networking and storage without issuing tickets.

## **2.5 Google Kubernetes Engine**

Google Cloud is where Kubernetes was first originated in 2014. Google developed the tool to containerize their 15 years' worth of workloads. When they announced that it was an open-source tool, it paved the way for the community to make their contributions to the tool.

Kubernetes was never built with the agenda to replace Borg, as it would've taken a tremendous amount of effort for the engineers to migrate. Hence developers learned from their past experiences and developed an open-sourced version for various other companies to depend on.

Google Kubernetes Engine (GKE) is a secured and managed service with a 4-way auto-scaling and multi-cluster support. Companies like Pizza-Hut U.S, heavily depend on Google solutions including GKE. They used the tools to transform their E-Commerce infrastructure and increase the response time for orders.

## **2.6 Pokémon Go**

Pokémon GO was developed and published by Niantic Inc. The game had more than 500 million downloads and more than 20 million active users. The developers did not expect such an exponential increase in users and surpass the expectations.

They soon realized that the servers could not handle the amount of traffic they received. Then the application was run on GKE, which had the ability to orchestrate their container cluster at a manageable scale.

This enabled the team to focus on deploying live changes for their users. GKE helped the company to not only serve, by allowing Niantic to improve their user's experience but also add newer features to the application.

## **2.7 New York Times**

The New York Times is one of the biggest publications in the world, with over 150 million monthly unique readers. The NYT like any other publication and magazine entered into the digital era to cater to its audiences, who now consume content on their phones instead of the grey paper.

One of the craziest problems that it faced was that of Fake News. They wanted to avoid sending out incorrect and unverified news to their readers, for which they wanted to check and cross-check each piece of information aggressively.

Now, the amount of data processed in a newsroom like this would require a strict back-up. They began their journey with LAMP Stack but were broadly dissatisfied and soon moved on to a React-based front end using Apollo.

A few years ago, the company decided to move out its data centers and less critical applications that shall be managed on VMs. That's when Kubernetes offered its solution GKE.

There was a significant increase in the spread, and deployment, and productivity. The Legacy deployments took less than 45 minutes and are now pushed in just a few minutes.

NYT has now moved on from a ticket-based system to requesting resources that deploy weekly schedules. This has enabled developers to push updates independently.

## CHAPTER 3

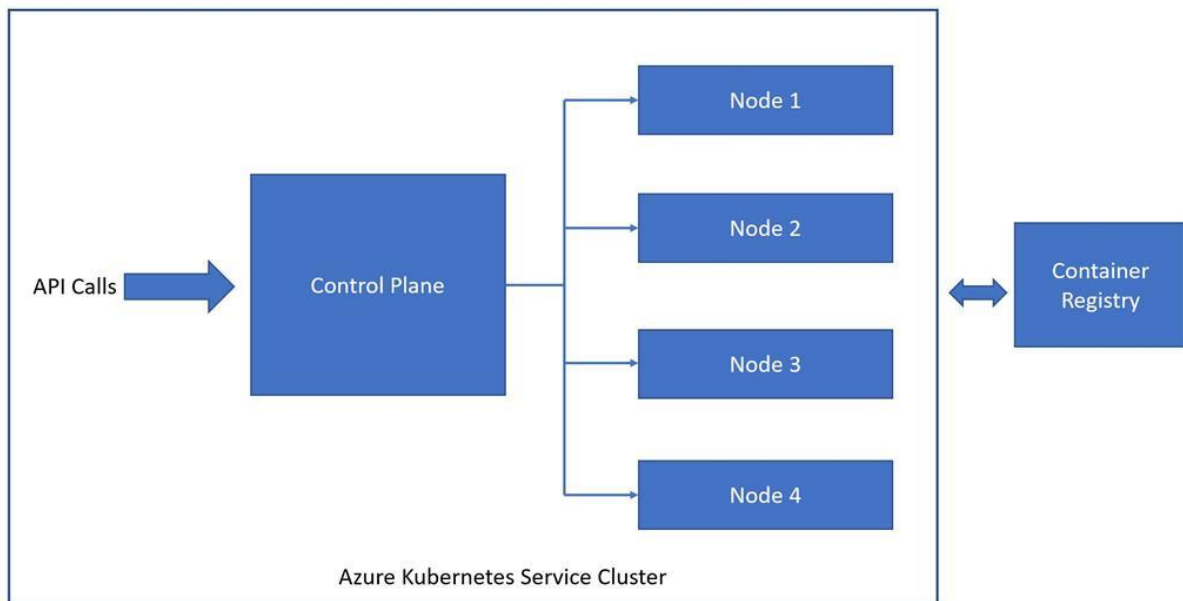
# Kubernetes Cluster

### 3.1 Architecture

Kubernetes architecture is straightforward and intuitive. The loose coupling between control plane and node allows for nearly infinite flexibility and the ability for an application to scale out virtually instantaneously to meet changing needs, to migrate users to new builds, and to support migration from on-premises to cloud-based nodes or between multiple clouds to take advantage of desired features of each cloud provider.

Kubernetes is an architecture that offers a loosely coupled mechanism for service discovery across a cluster. A Kubernetes cluster has one or more control planes, and one or more compute nodes. Overall, the control plane is responsible for managing the overall cluster, exposing the application program interface (API), and for scheduling the initiation and shutdown of compute nodes based on a desired configuration. Each of the compute nodes runs a container runtime like Docker along with an agent, kubelet, which communicates with the control plane. Each node can be bare metal servers, or on-premises or cloud-based virtual machines (VMs). When you deploy Kubernetes, you get a cluster. A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node. The worker node(s) host the pods that are the components of the application. The Control Plane manages the worker nodes and the pods in the cluster. In production environments, the Control Plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

- Control plane
- Node
- Addons



**Fig 3.1:** Architectural design of Kubernetes cluster

### **Control Plane:**

The Control Plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied). Control Plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all Control Plane components on the same machine, and do not run user containers on this machine. See Building High-Availability Clusters for an example multi-master VM setup.

### **Node:**

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

### **Addons:**

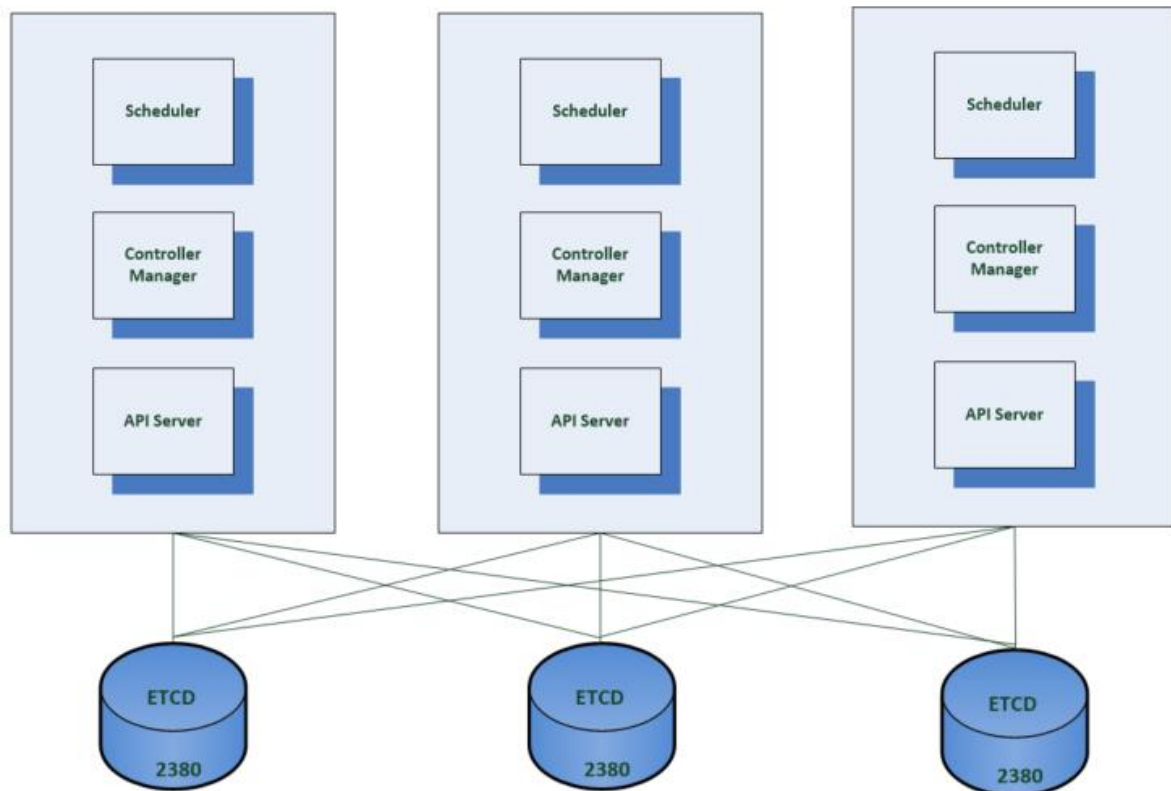
Addons use Kubernetes resources (Daemon Set, deployment etc.) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace



### 3.1.1 Control Plane Components

The Control Plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied). Control Plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all Control Plane components on the same machine, and do not run user containers on this machine. See Building High-Availability Clusters for an example multi-master VM setup. A Kubernetes control plane is the control plane for a Kubernetes cluster. Its components include:

- **kube-apiserver.** As its name suggests the API server exposes the Kubernetes API, which is communications central. External communications via command line interface (CLI) or other user interfaces (UI) pass to the kube-apiserver, and all control planes to node communications also goes through the API server.
- **etcd:** The key value store where all data relating to the cluster is stored. etcd is highly available and consistent since all access to etcd is through the API server. Information in etcd is generally formatted in human-readable YAML (which stands for the recursive “YAML Ain’t Markup Language”).
- **kube-scheduler:** When a new Pod is created, this component assigns it to a node for execution based on resource requirements, policies, and ‘affinity’ specifications regarding geolocation and interference with other workloads.
- **kube-controller-manager:** Although a Kubernetes cluster has several controller functions, they are all compiled into a single binary known as kube-controller-manager.



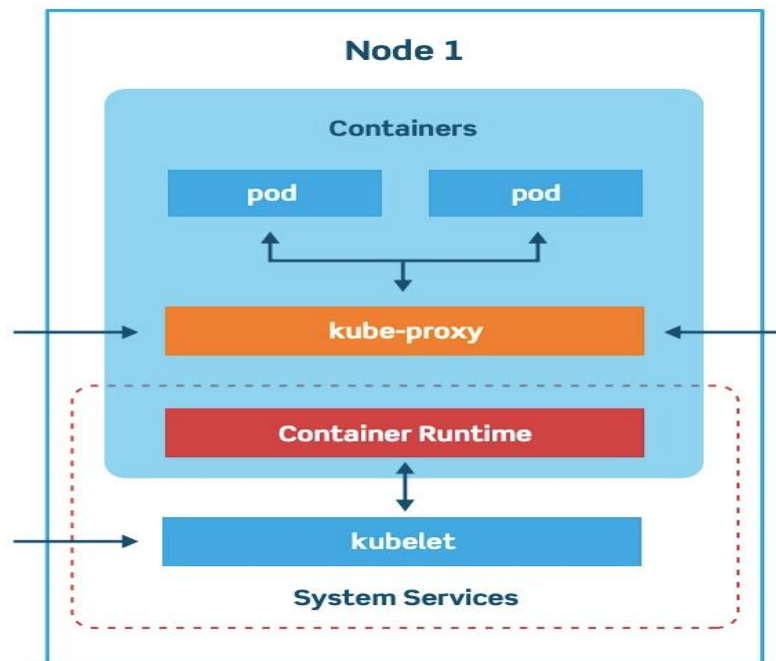
**Fig 3.2:** Control Plane Components

### 3.1.2 Kubernetes Node Architecture

Nodes are the machines, either VMs or physical servers, where Kubernetes place Pods to execute. Node processes must include:

- **kubelet:** Every node has an agent called kubelet. It ensures that the container described in PodSpecs are up and running properly. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- **kube-proxy:** A network proxy on each node that maintains network nodes which allows for the communication from Pods to network sessions, whether inside or outside the cluster, using operating system (OS) packet filtering if available.

- **container runtime:** Software responsible for running the containerized applications. Although Docker is the most popular, Kubernetes supports any runtime that adheres to the Kubernetes CRI (Container Runtime Interface).



**Fig 3.3:** Architectural diagram of Node

### 3.1.3 Addons

Addons use Kubernetes resources (Daemon Set, Deployment , etc.) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

- **DNS:** While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

- **Web UI (Dashboard):** Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
- **Container Resource Monitoring:** Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- **Cluster-level Logging:** A cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface

## 3.2 Essential Concepts

Flow diagram is a collective term for a diagram representing a flow or set of dynamic relationships in a system. It represents the picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.

### 3.2.1 Workloads

- **Node** - A node may be a virtual or physical machine, depending on the cluster. Each node contains the services necessary to run Pods, managed by the control plane.
- **Cluster** - A cluster is the foundation. All your containerized applications run on top of a cluster.
- **Pod** - the basic unit of work. Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. Pods are almost never created on their own, instead pod controllers do all the real work.
- **Namespace** - Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Namespaces are intended for use in environments with many users spread across multiple teams, or projects.

### 3.2.2 Pod Controllers

- Deployment - Most common way to get your app on Kubernetes. Creates a ReplicaSet for each Deployment spec.
- ReplicaSet - Creates a stable set of pods. You will almost never create this directly.
- DaemonSet - A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created. Common for system process like CNI, Monitor agents, proxies, etc.
- StatefulSet - StatefulSet is the workload API object used to manage stateful applications with persistent storage. Pod names are persistent and are retained when rescheduled (app-0, app-1). Pods have DNS names, unlike deployments. Storage stays associated with replacement pods. Volumes persist when pods are deleted. Pods are deployed/updated individually and in order.
- HorizontalPodAutoscaler - The Horizontal Pod Autoscaler automatically scales the number of pods in a replication controller, deployment, replica set or stateful set based on standard metrics like CPU utilization or custom metrics like request latency.
- PodDisruptionBudget - the ability to limit the number of Pods that are disrupted during upgrades and other planned events, allowing for higher availability while permitting the cluster administrator to manage the clusters nodes.
- Job - A Job creates one or more Pods and expects them to successfully terminate.
- CronJob - create Jobs on a schedule.

### 3.2.3 Configuration

- ConfigMaps - an API object used to store non-confidential data in key-value pairs.
- Secrets - let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.

### 3.2.4 Networking

- Ingress - An API object that manages external access to the services in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting.
- Service - an abstract way to expose an application running on a set of Pods as a network service.
- Network Policy - a specification of how groups of pods are allowed to communicate with each other and other network endpoints.

### 3.2.5 Security-RBAC

- ServiceAccount - provides an identity for processes that run in a Pod.
- Role - a Role is a set of permissions within a particular namespace; when you create a Role, you have to specify the namespace it belongs in.
- ClusterRole - ClusterRole is exactly the same as a Role except it is a non-namespaced resource. The resources have different names (Role and ClusterRole) because a Kubernetes object always has to be either namespaced or not namespaced; it can't be both.
- RoleBinding - grants the permissions defined in a Role to some group of Users or ServiceAccounts.

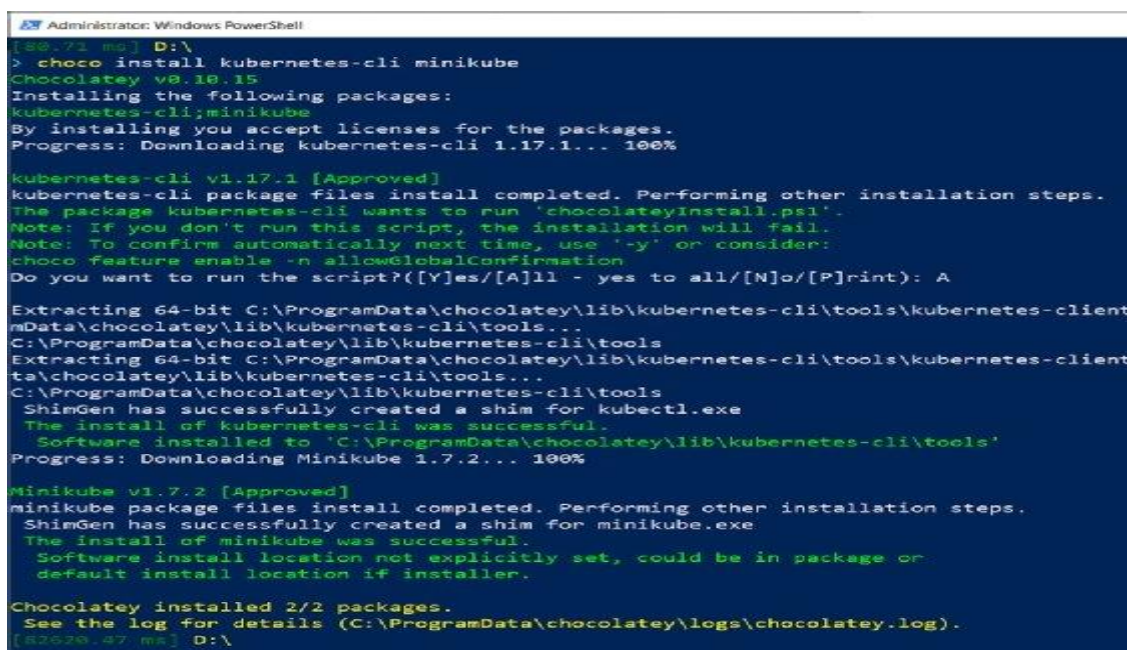
## Chapter 4

# IMPLEMENTATION

### 4.1 Installation and setup

To install and set up Kubernetes on Windows, load kubectl and install minikube. The Chocolatey package manager helps in this process. A command-line tool, kubectl runs commands against Kubernetes clusters, while minikube is a tool that enables us to run a single-node cluster in a VM on a machine.

choco install Kubernetes-cli minikube



```
Administrator: Windows PowerShell
[10:21:51 AM] D:\
> choco install kubernetes-cli minikube
Chocolatey v0.10.15
Installing the following packages:
kubernetes-cli;minikube
By installing you accept licenses for the packages.
Progress: Downloading kubernetes-cli 1.17.1... 100%

kubernetes-cli v1.17.1 [Approved]
kubernetes-cli package files install completed. Performing other installation steps.
The package kubernetes-cli wants to run 'chocolateyinstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): A

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client
mData\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client
ta\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
ShimGen has successfully created a shim for kubectl.exe
The install of kubernetes-cli was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'
Progress: Downloading Minikube 1.7.2... 100%

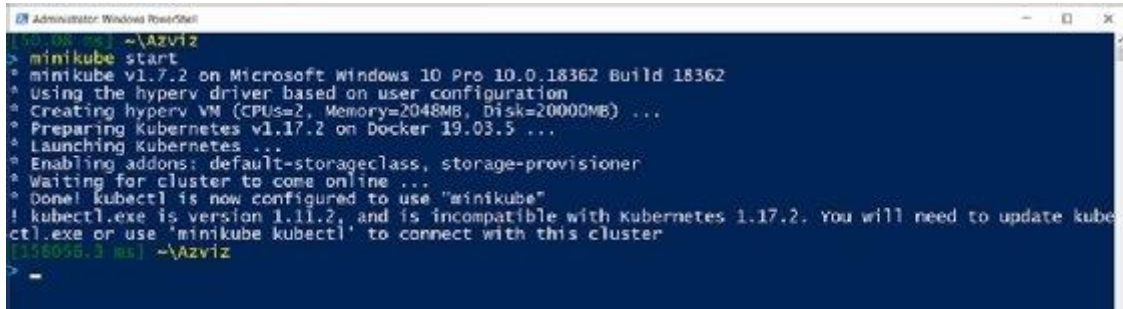
Minikube v1.7.2 [Approved]
minikube package files install completed. Performing other installation steps.
ShimGen has successfully created a shim for minikube.exe
The install of minikube was successful.
Software install location not explicitly set, could be in package or
default install location if installer.

Chocolatey installed 2/2 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
[10:22:07 AM] D:\
```

**Fig 4.1:** Install Minikube CLI

Next, spin up a worker machine -- or node -- in Kubernetes. It can be a physical or virtual machine. To do this, use the following command to start minikube: minikube start

This will return an output as shown below:



```
Administrator: Windows PowerShell
C:\Azviz> minikube start
minikube v1.7.2 on Microsoft Windows 10 Pro 10.0.18362 Build 18362
* Using the hyperv driver based on user configuration
* Creating hyperv VM (CPUs=2, Memory=2048MB, Disk=20000MB) ...
* Preparing Kubernetes v1.17.2 on Docker 19.03.5 ...
* Launching Kubernetes ...
* Enabling addons: default-storageclass, storage-provisioner
* Waiting for cluster to come online ...
* Done! kubectl is now configured to use "minikube"
! kubectl.exe is version 1.11.2, and is incompatible with kubernetes 1.17.2. You will need to update kubectl.exe or use 'minikube kubectl' to connect with this cluster
C:\Azviz>
```

**Fig 4.2:** Starting Minikube

This completes the setup.

## 4.2 Running Cluster

To start containers, use the Kubernetes command below to create a deployment. Provide a name for the deployment and the container image to deploy. Kubernetes will automatically pick Docker as the default container runtime. Here we use an image that will run the Nginx web server:

```
kubectl.exe create deployment my-nginx --image nginx
```

When a deployment is created, Kubernetes builds pods to host application instances.

Enter `get pods` just after running the previous command to catch the *ContainerCreating* status as pods are deployed: `kubectl.exe get pods`

This will complete in a few seconds and the container status should change to *Running*:



```

[22 ms] C:\
> kubectl.exe create deployment helloworld-nginx --image nginx
deployment.apps/helloworld-nginx created
[90.03 ms] C:\
> kubectl.exe get pods
NAME                                READY   STATUS             RESTARTS   AGE
helloworld-nginx-67bb76cf46-8mtq2  0/1     ContainerCreating   0          4s
[81.21 ms] C:\
> kubectl.exe get pods
NAME                                READY   STATUS    RESTARTS   AGE
helloworld-nginx-67bb76cf46-8mtq2  1/1     Running   0          34s
[85.03 ms] C:\
> -

```

**Fig 4.3:** Change in status

In this Kubernetes implementation tutorial, we run only one container or Nginx server, but sometimes it's necessary to accommodate increased workload and traffic. In that case, scale up the number of application instances. This can be achieved using `kubectl scale deployment` with a `--replicas` parameter:

```
kubectl.exe scale deployment helloworld-nginx --replicas 4
```

Check the deployment. You will observe that four replicas of the application instance have been deployed:

```

[26 ms] C:\
> kubectl.exe scale deployment helloworld-nginx --replicas 4
deployment.apps/helloworld-nginx scaled
[90.07 ms] C:\
> kubectl.exe get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
helloworld-nginx    4/4     4            4           6m12s
[94.08 ms] C:\
> kubectl.exe describe deployment helloworld-nginx
Name:                helloworld-nginx
Namespace:           default
CreationTimestamp:   Sun, 16 Feb 2020 02:18:36 +0530
Labels:              app=helloworld-nginx
Annotations:         deployment.kubernetes.io/revision: 1
Selector:             app=helloworld-nginx
Replicas:            4 desired | 4 updated | 4 total | 4 available | 0 unavailable
RollingUpdate:       RollingUpdate
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=helloworld-nginx
  Containers:
    nginx:
      Image:          nginx
      Port:           <none>
      Host Port:      <none>
      Environment:    <none>
      Mounts:         <none>
      Volumes:        <none>

```

**Fig 4.4:** Replica of Application Instance

Now, check the Kubernetes pods; there should be four containers running on the node:

```
[28.3 ms] C:\
> kubectl.exe get pods
NAME                                READY   STATUS    RESTARTS   AGE
helloworld-nginx-67bb76cf46-7qq5n  1/1     Running   0           2m19s
helloworld-nginx-67bb76cf46-8mtq2  1/1     Running   0           7m32s
helloworld-nginx-67bb76cf46-d9265  1/1     Running   0           116s
helloworld-nginx-67bb76cf46-mks6j  1/1     Running   0           116s
[84.1 ms] C:\
>
```

**Fig 4.5:** Containers running on the node

### 4.3 Deploy your app

Now there is an application running in multiple containers with their own IP addresses. Next, expose them outside the cluster so that the application is accessible:

Kubectl.exe expose deployment helloworld-nginx --port=80 --type=NodePort

Verify this via the kubectl get services command. This will return a service type of NodePort to expose port 80 on each node of the Kubernetes cluster. This service is an abstraction layer that basically load balances and groups more than one pod in a cluster that shares an IP address.

```
Administration: Windows PowerShell
[20.04 ms] C:\
> kubectl.exe expose deployment helloworld-nginx --port=80 --type=NodePort
service/helloworld-nginx exposed
[96 ms] C:\
> kubectl get services
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
helloworld-nginx                    NodePort    10.106.255.64 <none>         80:30550/TCP     2s
kubernetes                          ClusterIP   10.96.0.1     <none>         443/TCP          2d1h
[84.04 ms] C:\
>
```

**Fig 4.6:** Make application Accessible

To open this application in a web browser, create a proxy to connect the local port to the cluster port, which we exposed using the *NodePort* service in the previous step:

kubectl.exe port-forward svc/helloworld-nginx 80:80

This will look as follows

```

Administrator: Windows PowerShell
[20.44 ms] C:\
> kubect1 get services
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
helloworld-nginx    NodePort      10.106.255.64 <none>         80:30550/TCP     3m30s
kubernetes           ClusterIP     10.96.0.1     <none>         443/TCP          2d1h
[135.22 ms] C:\
> kubect1.exe port-forward svc/helloworld-nginx 80:80
Forwarding from 127.0.0.1:80 -> 80
Forwarding from [::1]:80 -> 80

```

Fig 4.7: Creating proxy

#### 4.4 Testing with control plane

Lastly, test that the Kubernetes control plane, or master server, is able to maintain the desired state of the pods that run on the node server. To check this, use the following command to forcibly delete one of the pods that runs the application instance:

kubect1.exe delete pod helloworld-nginx-67bb76cf46-mks6j

and Kubernetes will immediately run a new instance

```

Administrator: Windows PowerShell
[25.03 ms] C:\
> kubect1.exe get pods
NAME                                READY   STATUS    RESTARTS   AGE
helloworld-nginx-67bb76cf46-7qq5n   1/1     Running   0          16m
helloworld-nginx-67bb76cf46-8mtq2   1/1     Running   0          22m
helloworld-nginx-67bb76cf46-d9265   1/1     Running   0          16m
helloworld-nginx-67bb76cf46-mks6j   1/1     Running   0          16m
[91 ms] C:\
> kubect1.exe delete pod helloworld-nginx-67bb76cf46-mks6j
pod "helloworld-nginx-67bb76cf46-mks6j" deleted
[12939.05 ms] C:\
> kubect1.exe get pods
NAME                                READY   STATUS    RESTARTS   AGE
helloworld-nginx-67bb76cf46-7qq5n   1/1     Running   0          17m
helloworld-nginx-67bb76cf46-8mtq2   1/1     Running   0          22m
helloworld-nginx-67bb76cf46-d9265   1/1     Running   0          16m
helloworld-nginx-67bb76cf46-prv5m   1/1     Running   0          14s
[83.06 ms] C:\

```

Fig 4.2: Run new pod

## CHAPTER 5

# APPLICATIONS

The main reasons for choosing this container orchestration tool were cost reduction, increased automation, and more frequent deployments.

### 5.1 Deploying a simple app

Kubernetes is complex and creating a Kubernetes cluster to run one simple app would mean doing unnecessary work. There is still one more practical and advanced scenario where we can use Kubernetes to deploy apps. Imagine we work in a creative agency that is developing a marketing webpage for a client in the pharmaceutical industry. Each medicine advertised on the main page requires a separate webpage where a leaflet of information about the medicine, its ingredients, dosage, possible adverse effects etc. Each medicine would also have a dedicated app. In this scenario, we would be well advised to call on the power of Kubernetes. Thanks to the better resource allocation it affords, it will be cheaper to run one dedicated K8s cluster than many separate servers for each website. What's more, it will be much easier to manage such a cluster than to employ separate hosts.

### 5.2 Microservices architecture

A use case where you want to deploy a more complicated app with many components that will communicate with one another is a classic scenario for Kubernetes. Consider an example of an Internet bookstore. In such a store, we have different functionalities: manage users, order books, manage order lists, etc. There can be many such functionalities and each of them is a separate app. This is a practical realization which are aptly called microservices. All these apps must communicate with each other. To enable such communication and coordination, code must be written to conform with the programming language of each component.

Kubernetes handles for developers such tasks as detecting problems with communication between the intra-app components, managing the behaviour of components in the event of a

failure or managing the authentication processes between components. What's more, as more or less resources are needed for a particular component, Kubernetes automatically scales them up or down. This is a clear advantage of the microservice architecture: scalability. You can scale a single component rather than the whole app.

### **5.3 Lift and shift**

This scenario occurs frequently today, as software is migrated from on-prem infrastructure to cloud solutions. Let's imagine the following situation. We have an application deployed on physical servers in a classical datacentre. For practical or economic reasons, it has been decided to move it to the cloud: either to a Virtual Machine or to big pods in Kubernetes. Of course, moving it to big pods in K8s isn't a cloud native approach, but it can be treated as an intermediary phase. First, such a big app working outside the cloud is moved to the same big app in Kubernetes. It is then split into smaller components to become a regular cloud native-app. Such methodology is called "lift and shift" and is a good use case where Kubernetes can be used effectively.

### **5.4 Cloud-native Network Functions**

A few years ago, big telco companies had a problem. Their network services were based on hardware such as firewalls or load balancers provided by specialized hardware companies. Of course, this left them dependent on the hardware providers, and gave them little in the way of flexibility. If new functionality was needed, operators had to upgrade existing hardware. When a device firmware update was not possible, additional hardware had to be purchased. To address this disadvantage, the telcos opted to have all these network services as software and use Virtual Machines and OpenStack for network function virtualization (NFV).

A step further is to use containers rather than VMs for the same purpose. This approach is called Cloud-native Network Functions (CNF). Our R&D team has prepared a demo of CNFs



deployment in the **Kubernetes** environment along with an in-depth discussion of related network and operational aspects.

## 5.5 Machine learning

Machine learning (ML) techniques are now widely used to solve real-life problems. Successes have come in multiple fields--self-driving cars, image recognition, machine translation, speech recognition, game playing (Go or poker). Yet, the process of building an effective AI model and using it in production is complicated and time-consuming. Building an app that can reliably recognize whether an image presents a cat or a dog is a case in point. First of all, a large dataset of images tagged “cat” or “dog” must be uploaded. Then, an untrained machine learning model is trained to classify the data in mathematical terms; trained, that is, to recognize the images that are neither in the training nor in the test dataset. After the model is trained, it is implemented in an app that will be made available to the public. Enterprises can harness the power of Kubernetes, as all the calculations necessary to train the ML model are performed inside the **K8s cluster**. The data scientist or ML engineer will only need to clean the data and write the code. The rest will be handled by a toolkit based on Kubernetes. Such toolkits are already available on the market: Kubeflow by Google and CodiLime spin-off Neptune both come to mind. The increasing demand for AI-powered solutions will surely further promote the **adoption of Kubernetes**.

## 5.5 CI/CD

Kubernetes also brings considerable benefits to Continuous Integration/Continuous Deployment or Continuous Delivery methodology (you can read more about CI/CD in our blog post). This is a logical continuation of the use cases presented in points 1 and 2. In the cloud-native application development approach, choosing CI/CD pipeline tools wisely is necessary as once an app is deployed into operations, its work has to be constantly monitored. That is in addition to gathering users’ feedback, developing new features, and other business benefits. Whether it is for testing, frequent releases or deploying newer versions of an app, Kubernetes makes everything simpler and more manageable.

## CHAPTER 6

# USE CASES

### 6.1 REDUCING TIMEFRAMES

Kubernetes greatly simplifies the development, release, and deployment processes: for example, it enables container integration or facilitates the administration of access to storage resources from different providers.

Moreover, in scenarios where the architecture is based on microservices, the application is broken down into functional units that communicate with one another via APIs: the development team can thus be broken down into smaller groups, each specializing in a single feature. This organization allows IT teams to operate with greater focus and efficiency, accelerating release timeframes.

### 6.2 OPTIMIZING IT COSTS

Through dynamic and intelligent container administration, Kubernetes can help organizations save on their ecosystem management, ensuring scalability across multiple environments. Resource allocation is automatically modulated to the actual application needs, while low-level manual operations on the infrastructure are significantly reduced, thanks in part to native autoscaling (HPA, VPA) logics and integrations with major cloud vendors, capable of providing resources dynamically.

Thanks to automation, IT teams are no longer required to carry out large numbers of operational tasks related to system management and can therefore be used to carry out value-added tasks. Because applications run indiscriminately on any environment, businesses are free to decide which resources to rely on (on-premise, private cloud, or public cloud) for each specific workload, based on convenience.

### **6.3 INCREASED SCALABILITY AND AVAILABILITY**

Kubernetes is able to scale the applications and underlying infrastructure resources up or down, based on the contingent needs of the organization, facilitating the dynamic management of peaks. For example, as an event date nears, an e-ticketing system will experience a sudden increase in requests to purchase tickets.

Thanks to its native Autoscaling APIs, such as HPA and VPA, Kubernetes will be able to dynamically request new HW resources to be allocated to the infrastructure providing the service, in order to ensure the performance of the same. Once the emergency is over, Kubernetes will then scale down the resources that are no longer needed, avoiding waste.

### **6.4 FLEXIBILITY IN MULTI-CLOUD ENVIRONMENTS**

Containerization and Kubernetes – one of the biggest benefits offered by the solution – make it possible to realize the promises of the new hybrid and multi-cloud environments, guaranteeing the operation of applications in any public and private environment, without functional or performance losses. The risk of lock-in is thus also reduced (in other words the lack of interoperability of certain IT solutions, which force organizations to tie themselves to a single supplier, limiting freedom of choice).

### **6.5 CLOUD MIGRATION PATHS**

Finally, Kubernetes makes it possible to simplify and accelerate the migration of applications from an on-premises environment to public or private clouds, offered by any provider. Applications can be migrated to the cloud through the adoption of various methodologies:

- the simple transposition of the application, without any coding changes (Lift & Shift).



- the minimum changes necessary to allow the application to work on new environments (re platforming);
- the extensive rewriting of the application structure and functionality (refactoring).

A recommended approach is to re platform on on-premises systems (where it is easier), using the new containerized architectures and Kubernetes. The applications are therefore migrated to a cloud environment where an instance of Kubernetes is running. Here, the solution can then be optimized, with more extensive changes made to the code.

## CHAPTER 7

# ADVANTAGES AND DISADVANTAGES

### 7.1 Advantages

- Open source, backed by a very active foundation called cloud native computing foundation with a very active contributor base.
- A very flexible architecture, allowing users to plug and play components as they are needed without having to opt-in for the full package.
- Users can also write their own variants of the components based on their infrastructure, as most components are just interfaces.
- Cloud providers offer their own managed Kubernetes services, allowing users to directly take advantage of this technology without having to provision and maintain underlying machines.
- There is no vendor lock-in as the same Kubernetes cluster experience is present across various cloud providers, on-premises and even on edge now.
- Allows applications to be as containers, thus not only being light weight but also making the builds reproducible.
- An awesome oss ecosystem of projects that add features to Kubernetes like git ops, service meshes, etc. Thus, there are multiple solution available for most problems

## 7.2 Disadvantages

- It can be overkill for simple applications, as its relatively complex to manage and maintain.
- If not used correctly and with standards, it can even decrease developer productivity as maintaining the cluster could be hard.
- As it's a fast-moving project, Things keep changing every release and it's important to keep up with the changes and this requires additional developers to take this work.
- The transition to Kubernetes can be really hard, based on your existing infrastructure, as existing applications have to be first converted to microservices, and then containerized and then deployed into the cluster.
- Running Stateful Applications is comparatively harder and its definitely not easy right now

## CHAPTER 8

### CONCLUSION

Kubernetes has definitely re defined how Micro-services are built now. Most applications that we use daily like Spotify, twitter, etc. moved to Kubernetes as the underlying infrastructure platform and these applications are internet scale. So, Kubernetes can mostly support all types of use-cases. Though there is a debate on supporting stateful applications on Kubernetes, it's not easy and requires professional disks and underlying hardware to be highly available which is not the case in most on-premises Infrastructure. But there is a lot of work being added, to support stateful applications on Kubernetes. Kubernetes even after being late to the Distributed Systems ecosystem i.e 2014 but already is considered a winner as it's the most widely used orchestration platform. Even cloud providers like Microsoft, Google, etc are also not only offering Kubernetes as a service but also are using it as a platform to run services like Xbox, Gmail, etc. Massive Scale applications are being deployed on Kubernetes and are ran successfully, which is a huge testament to the project and its architecture.

## References

### Websites:

- [1] <https://kubernetes.io/>
- [2] <https://www.docker.com/resources/what-container>
- [3] <https://kubernetes.io/case-studies/newyorktimes/>
- [4] <https://codilime.com/blog/harnessing-the-power-of-kubernetes-7-use-cases/>
- [5] <https://blog.sparkfabrik.com/en/kubernetes-key-benefits-for-companies>
- [6] <https://www.vmware.com/topics/glossary/content/kubernetes-architecture.html>
- [7] <https://kubernetes.io/docs/home/#understand-kubernetes>
- [8] <https://kubernetes.io/blog/>