Towards partial fulfilment for Undergraduate Degree Level Programme
Bachelor of Technology in Computer Science and Engineering

Seminar Evaluation Report on:

Kubernetes

Prepared by:

Name of the Student:	Nehal Jhajharia
Admission No.:	U20CS093
Class:	B.TECH. III (Computer Science and Engineering) Semester - V
Year:	2022-23
Name of the Supervisor:	Dr. S J Patel

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY,
SURAT - 395007 (GUJARAT, INDIA)

Student Declaration

This is to certify that the work described in this seminar report has been actually carried out and implemented by me i.e.

Admission No.	Name of the Student
U20CS093	Nehal Jhajharia

Neither the source code there in, nor the content of the seminar report have been copied or downloaded from any other source. I understand that my result grades would be revoked if later it is found to be so.

Signature of the Student

Certificate

This is to certify that the seminar report entitled **“Kubernetes”** is prepared and presented by,

Admission No.	Name of the Student
U20CS093	Nehal Jhajharia

Third Year of Computer Science and Engineering and his/her work is satisfactory.

SIGNATURE:

SUPERVISOR JURY HEAD OF THE DEPARTMENT

Abstract

The era of monolithic applications, i.e. applications bundled together with all their parts like frontend, backend, database queries, etc, and then shipped, have gone due to scaling.

The major issue with these applications was that, if some part has some issue, the whole service goes down and therefore the whole application has to be started on another server.

This was a bottleneck for scaling the applications. And comes into picture, microservices.

These microservices can be run on different servers and provide modularity and reliability.

Issues with one microservice will not cause the whole application to go down. Neither the whole application has to be run on a new server. Rather, the microservice with issues can be restarted in another server and the application can go on.

Modern software is increasingly run as fleets of containers, or simply put, microservices. A complete application may comprise many containers, all needing to work together in specific ways. Kubernetes is software that turns a collection of physical or virtual hosts (servers) into a platform that hosts containerized workloads, providing them with compute, storage, and network resources and automatically manages large numbers of containerized applications — keeping them healthy and available by adapting to changes and challenges.

Kubernetes is not a traditional, all-inclusive PaaS (Platform as a Service) system. Since, Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS offerings, such as deployment, scaling, load balancing, and lets users integrate their logging, monitoring, and alerting solutions. However, Kubernetes is not monolithic, and these default solutions are optional and pluggable. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

Index

List of Figures

Chapter 1: Introduction

1.1 What is Kubernetes?

Kubernetes is an open source container orchestration framework which was originally developed by Google. So, on the foundation it manages containers, be it Docker containers or from some other technology which basically means that Kubernetes helps you manage applications that are made up of hundreds or maybe thousands of containers and it helps you manage them in different environments like physical machines virtual machines or cloud environments or even hybrid deployment environments.

1.2 What problems does Kubernetes solve?

Or tasks of a container orchestration tool.
So, to go through this chronologically the rise of microservices caused increased usage of container technologies because the containers actually offer the perfect host for small independent applications like microservices and the rise of containers and the microservice technology actually resulted in applications they're now comprised of hundreds or sometimes maybe even thousands of containers managing those loads of containers across multiple environments using scripts and self-made tools that can be really complex and sometimes even impossible. This specific scenario actually caused the need for having container orchestration technologies.

1.3 Kubernetes advantageous features

What these orchestration tools like Kubernetes do is actually guarantee following features:

- **High availability**, meaning that the application has no downtime so it's always accessible by the users.
- **Scalability**, meaning you can scale your applications fast when you have more load on it and more users are trying to access it and the same way you can easily scale it down when the load goes down so it makes your application more flexible to adjust to the increasing or decreasing load.
- **Disaster recovery**, which basically means that if an infrastructure has some problems like data is lost or the servers explode or something bad happens with the service centre, the infrastructure has to have some kind of mechanism to back up the data and to restore it to the latest state so that application doesn't actually lose any data and the containerized application can run from the latest state after the recovery.

1.4 Kubernetes Cluster

The kubernetes cluster is made up with at least one master node and then connected to it, you have a couple of worker nodes where each node has a Kubelet process running on it.

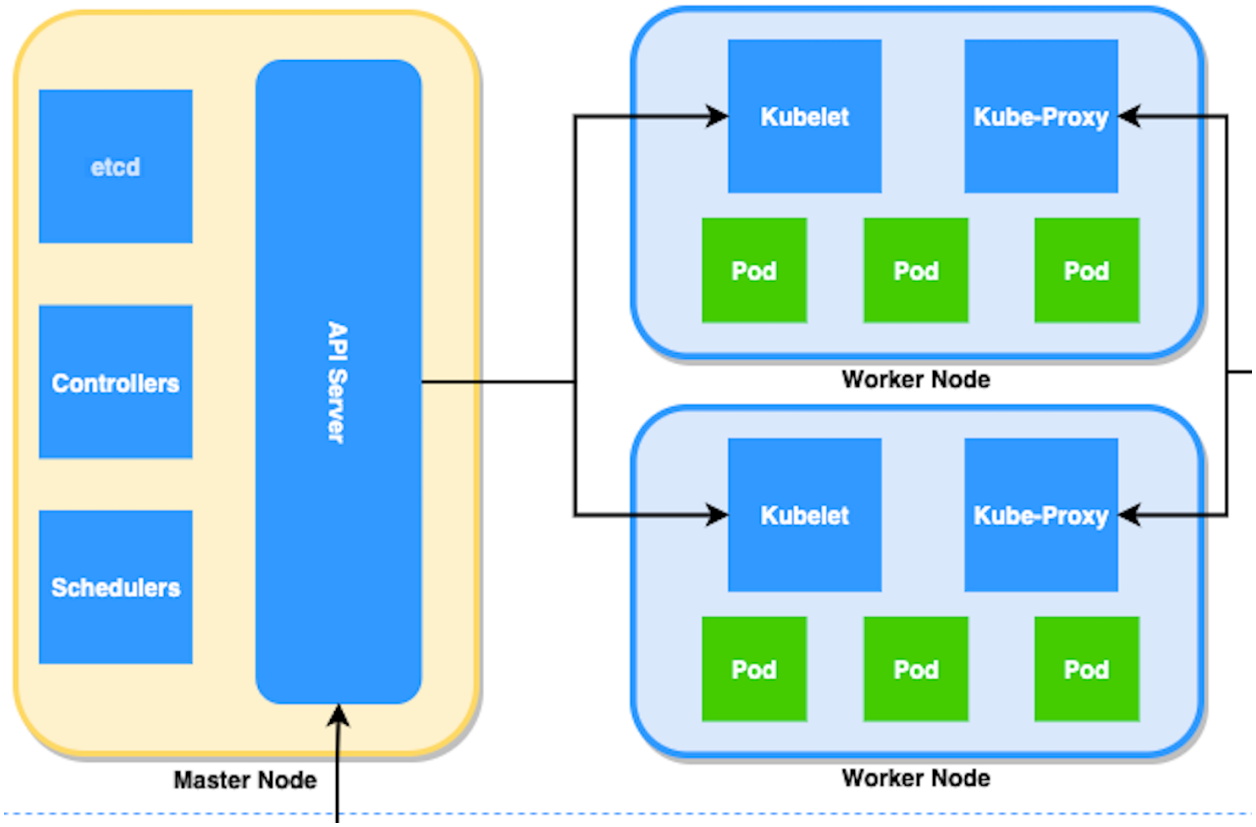

Fig 1: Kubernetes Cluster

Each worker node has containers of different applications deployed on it. So, depending on how the workload is distributed, you would have a different number of Docker containers running on worker nodes and worker nodes are where the actual work is happening, simply put, here is where your applications are running.

Chapter 2: Kubernetes Components

2.1 Kubelet

Kubelet is actually a Kubernetes process that makes it possible for the cluster to communicate with each other and actually execute some tasks on those nodes like running application processes.

2.2 Control Plane or Masternode

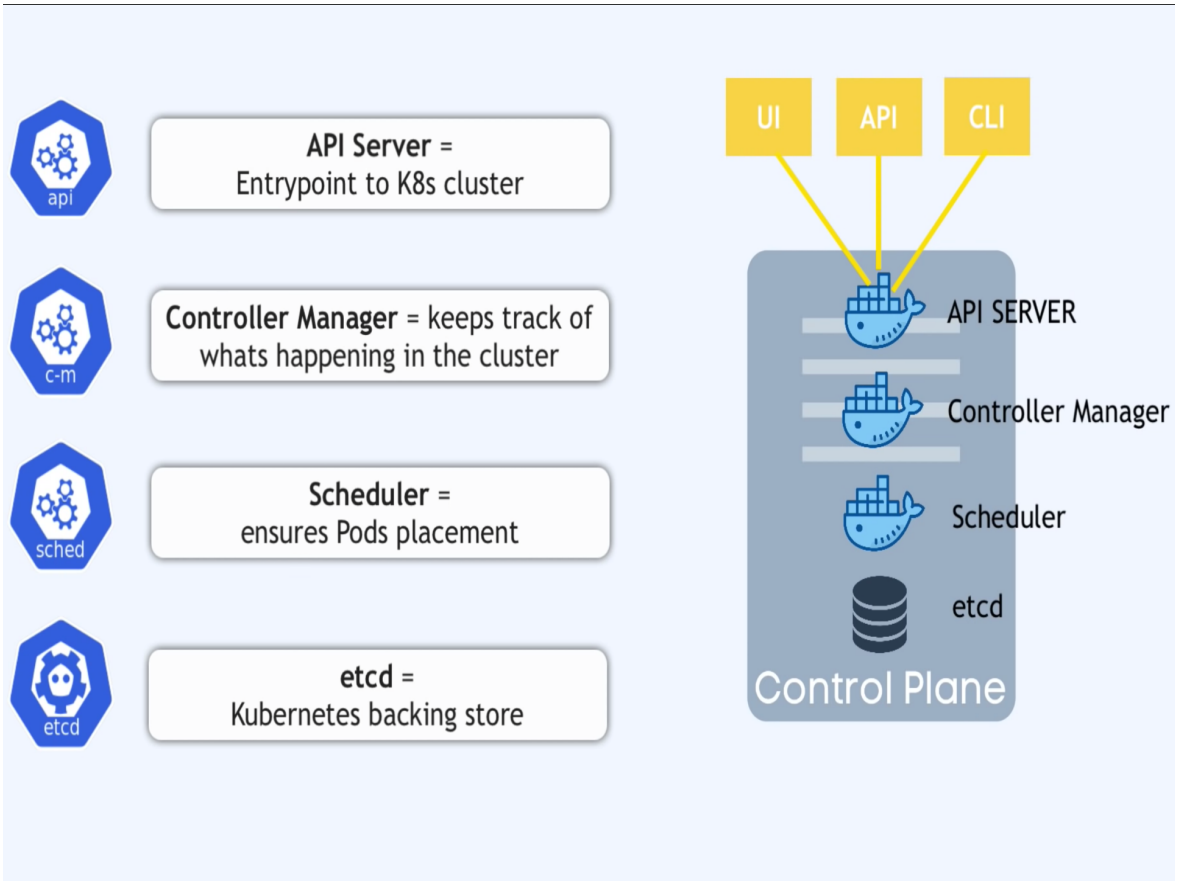

Fig 2: Control Plane

Masternode actually runs several Kubernetes processes that are absolutely necessary to run and manage the cluster properly:

- **Api server** is actually the entry point to the Kubernetes cluster.
- This is the process through which the different kubernetes clients will talk to like UI, if you're using Kubernetes dashboard, an api, if you're using some scripts/automating technologies and a command line tool.
- **Controller manager** basically keeps an overview of what's happening in the cluster whether something needs to be repaired or maybe if a container died and it needs to be restarted etc.
- **Scheduler** which is basically responsible for scheduling containers on different nodes based on the workload and the available server resources on each node. It's an intelligent process that decides on which worker node the next container should be scheduled based upon the available resources on those worker nodes and the load that container needs.
- **etcd** key value storage which basically holds at any time the current state of the Kubernetes cluster. It has all the configuration data inside and all the status data of each node and each container inside of that node and the backup and restore that we mentioned previously is actually made from these etcd snapshots.
- **Virtual network** which enables those nodes worker nodes masternodes talk to each other. It spans all the nodes that are part of the cluster and in simple words virtual network actually turns all the nodes inside of a cluster into one powerful machine that has the sum of all the resources of individual nodes.

2.3 Load of worker node and master node

One thing to be noted here is worker nodes as they actually have the most load as they are running the applications. They usually are much bigger and have more resources because they will be running hundreds of containers inside of them whereas the master node will be running just a handful of master processes. So it doesn't need that many resources.
However, as you can imagine, masternode is much more important than the individual worker nodes because if for example you lose a masternode access, you will not be able to access the cluster anymore and that means that you absolutely have to have a backup of your masternode at any time. So in production environments, usually you would have at least two masters inside of your Kubernetes cluster.

2.4 Worker Node and Pod

Worker node or in Kubernetes terms, a node which is a simple server, a physical or virtual machine and the basic component or the smallest unit of Kubernetes is a pod which is basically an abstraction over a container, like Docker containers or container images. Basically, what pod does is it creates a running environment or a layer on top of the container and the reason is that Kubernetes wants to abstract away the container runtime or container technologies so that you can replace them if you want to and also because you don't have to directly work with Docker or other container technology you use in Kubernetes, simply put, you only interact with the Kubernetes layer.
An important concept here is that a pod is usually meant to run one application container inside of it. You can run multiple containers inside one pod but usually it's only the case if you have one main application container and the helper container or some side service that has to run inside of that pod.

2.5 Virtual Network

Let's see how they communicate with each other in the Kubernetes world. Kubernetes offers out of the box, a virtual network which means that each pod gets its own ip address, not the container, the pod gets the ip address and each pod can communicate with each other using that ip address which is an internal ip address. It's not the public one so our application container can communicate with the database using the ip address. However pod components in Kubernetes, also an important concept, are *ephemeral* which means that they can die very easily and when that happens for example if we lose a database container because the container crashed because the application crashed inside or because the nodes the server that we are running them on ran out resources, the pod will die and a new one will get created in its place and when that happens, it will get assigned a new ip address which obviously is inconvenient if you are communicating with the database using the ip address because now you have to adjust it every time the pod restarts and because of that another component of Kubernetes called **service** is used.

2.6 Service & Ingress

Service is basically a static ip address or permanent ip address that can be attached to each pod so our app will have its own service and database pod will have its own service and the good thing here is that the life cycles of service and the pod are not connected. So even if the pod dies, the service and its ip address will stay.
We would like our application to be accessible through a browser and for that, we would have to create an **external service** which is a service that opens the communication from external sources but obviously we wouldn't want our database to be open to the public requests and for that we would create something called an **internal service** which is a service that we specify when creating one.
However, if you notice the url of the external service is not very practical, so basically what we have is an http protocol with a node ip address of the node not the service and the port number of the service which is good for test purposes if we want to test something very fast but not for the end product. So usually, we would want our url to look human readable if we want to talk to our application with a secure protocol and a domain name and for that there is another component of Kubernetes called **ingress.** So instead of service, the request goes first to ingress and it does the forwarding then to the service.

Chapter 3: Deploying an application

3.1 Our application

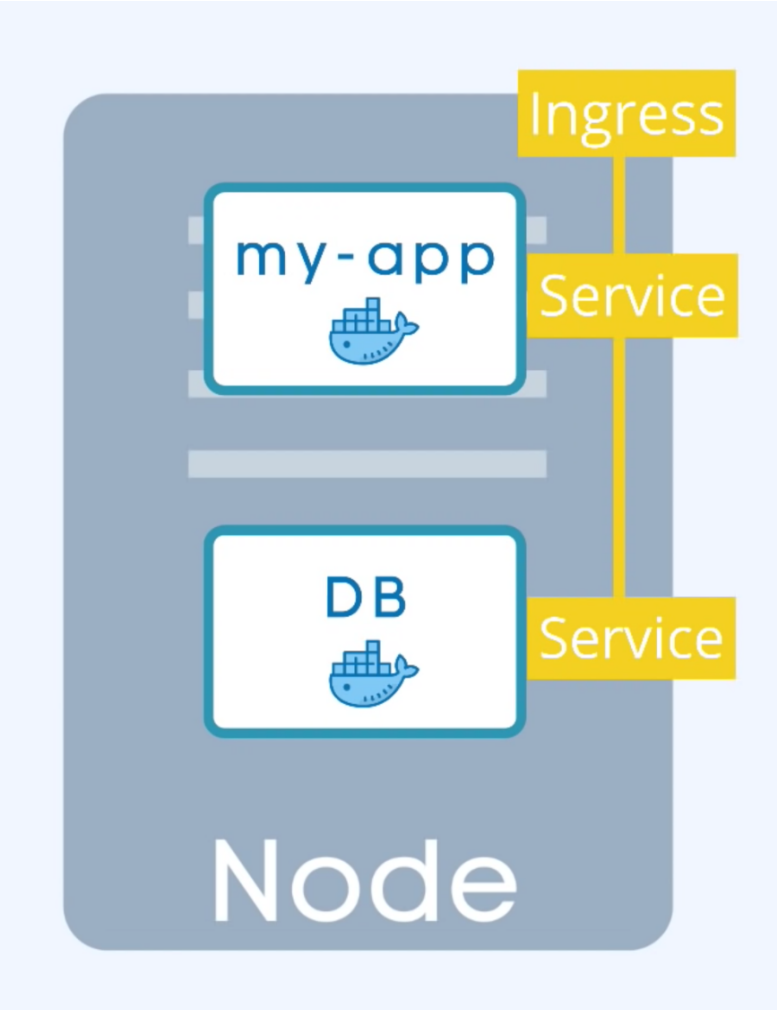

Fig 3: Application structure

Application pod which is our own application and that will maybe use a database pod with its own container.

3.2 Config Map

So, our application will have a database endpoint let's say called *mongodb service* that it uses to communicate with the database but whether you configure usually this database url or endpoint, you would do it in application properties file or as some kind of external environmental variable. But usually it's inside of the built image of the application. So for example if the endpoint of the service or service name in this case changed to *mongodb*, you would have to adjust that url in the application so usually you'd have to rebuild the application with a new version and you have to push it to the repository and now you'll have to pull that new image in your pod and restart the whole thing which is a bit tedious for a small change like database url. Kubernetes has a component called **config map**. It's basically your external configuration to your application.

3.3 Secret

Config map would usually contain configuration data like urls of a database or some other services that you use and in Kubernetes you just connect it to the pod so that pod actually gets the data that config map contains and now if you change the name of the service the endpoint of the service you just adjust the config map and that's it. You don't have to build a new image and have to go through this whole cycle now.
Part of the external configuration can also be database username and password, which may also change in the application deployment process but putting a password or other credentials in a config map in a plain text format would be insecure even though it's an external configuration. For this purpose, Kubernetes has another component called **secret** which is just like config map but the difference is that it's used to store secret data credentials for example and it's stored not in a plain text format but in base 64 in encoded format.

3.4 Volume

Let's see another very important concept generally which is data storage and how it works in kubernetes.
We have this database pod that our application uses and it has some data or it generates some data with this setup. If the database container or the pod gets restarted the data would be gone and that's problematic and inconvenient because you want your database data or log data to be persisted reliably long term and the way you can do it in kubernetes is using another component of kubernetes called **volumes** and how it works is that it basically attaches a physical storage on a hard drive to your pod and that storage could be either on a local machine meaning on the same server node where the pod is running or it could be on a remote storage meaning outside of the kubernetes cluster. It could be a cloud storage or it could be your own premise storage which is not part of the kubernetes cluster.

Fig 4: Cluster with storage

So you just have an external reference on it so now when the database pod or container gets restarted, all the data will be there persisted.
it's important to understand the distinction between the kubernetes cluster and all of its components and the storage regardless of whether it's a local or remote storage think of a storage as an external hard drive plugged into the kubernetes cluster because the point is, kubernetes cluster explicitly doesn't manage any data persistence which means that you as a kubernetes user or an administrator are responsible for backing up the data, replicating and managing it and making sure that it's kept on a proper hardware.

3.5 Deployments

So now that everything is running perfectly and a user can access our application through a browser, what happens if our application pod dies, crashes or we have to restart the pod because we built a new container image.
Basically, we would have a downtime where a user can reach our application which is obviously a very bad thing if it happens in production and this is exactly the advantage of distributed systems and containers. So instead of relying on just one application part and one database part etc, we are replicating everything on multiple servers so we would have another

node where a replica or clone of our application would run which will also be connected to the service. So remember previously we said the service is like a persistent static ip address with a dns name so that you don't have to constantly adjust the end point when a pod dies but service is also a load balancer which means that the service will actually catch the request and forward it to whichever part is least busy.
In order to create the the second replica of the our application pod we wouldn't create a second part but instead we will define a blueprint for our application pod and specify how many replicas of that pod we would like to run and that component or that blueprint is called **deployment** which is another component of kubernetes.
In practice, we would not be working with pods or we would not be creating pods. We would be creating deployments because there we can specify the number of replicas and we can also scale up or scale down the number of replicas of pods that we need.
Pod is a layer of abstraction on top of containers and deployment is another abstraction on top of pods which makes it more convenient to interact with the pods, replicate them and do some other configuration.
Now if one of the replicas of our application pod would die, the service will forward the requests to another one and our application would still be accessible for the user.

3.6 Statefulsets

Now you're probably wondering what about the database pod because if the database part dies, our application also wouldn't be accessible. So we need a database replica as well. However, we can't replicate database using a deployment and the reason for that is because database has a state which is its data, meaning that if we have clones or replicas of the database they would all need to access the same shared data storage and there we would need some kind of mechanism that manages which parts are currently writing to that storage or which pods are reading from the storage in order to avoid data inconsistencies and that mechanism in addition to replicating feature is offered by another kubernetes component called **statefulset.** This component is meant specifically for applications like databases.

So now that we have two replicas of our application pod and two replicas of the database and they're both load balanced, our setup is more robust which means that now even if node one or the whole node server was actually rebooted or crashed and nothing could run on it we would still have a second node with application and database pods running on it and the application would still be accessible by the user until these two replicas get recreated avoiding any downtime.

Chapter 4: Kubernetes Configuration

4.1 Basic Configuration

Now that we have seen the basic concepts of kubernetes, how do we actually create those components like pods and services to configure the kubernetes cluster?
All the configuration in kubernetes cluster actually goes through a master node with the process called api server, which we mentioned briefly earlier, so kubernetes clients which could be a UI, a kubernetes dashboard for example, or an api which could be a script or a curl command or a command line tool like **kubectl.**
They all talk to the api server and they send their configuration requests to the api server which is the main entry point or the only entry point into the cluster and these requests have to be either in *yaml* format or *json* format.

4.2 Minikube

When you are setting up a production cluster it will look something like:

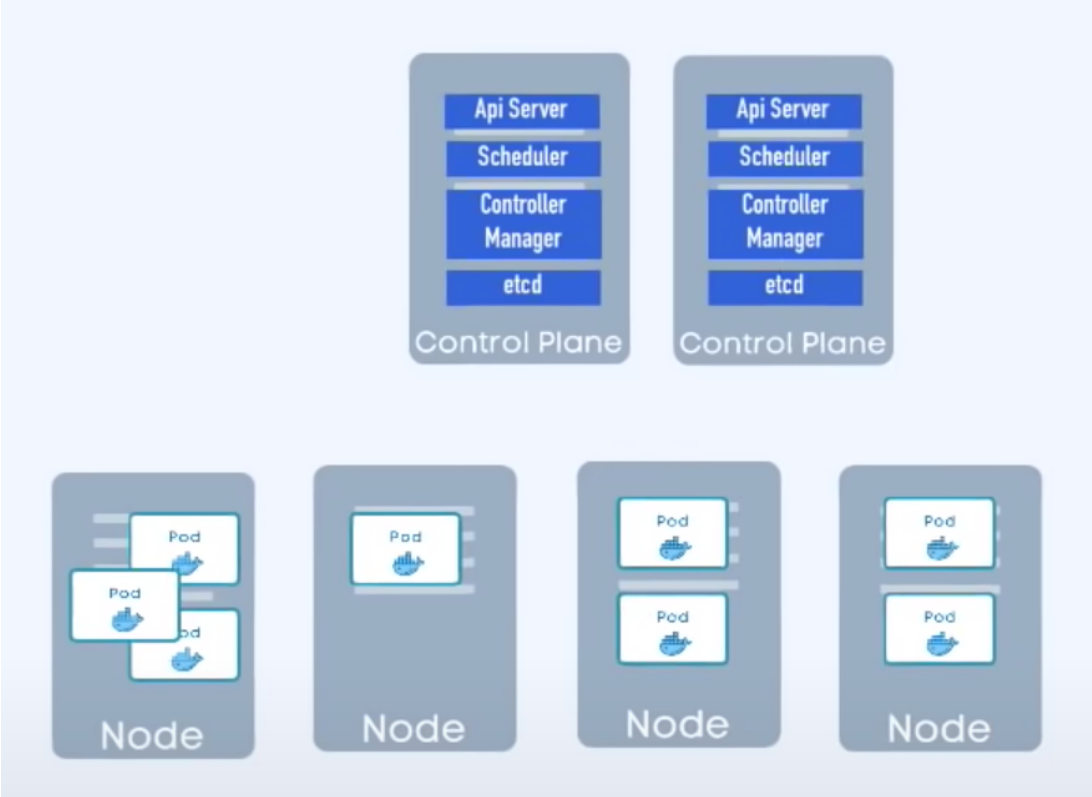

Fig 5: Production Cluster

You would have multiple masters, at least two, in a production setting and you would have multiple worker nodes and master nodes and the worker nodes have their own separate responsibility.
You would have actual separate virtual or physical machines that each represent a node. Now if you want to test something on your local environment or if you want to try something out very quickly, for example deploying new application or new components and you want to test it on your local machine, obviously setting up a cluster like this will be pretty difficult or maybe even impossible if you don't have enough resources like memory and cpu etc. And exactly for the use, there's this open source tool that is called a **minikube.**
Minikube is basically one node cluster where the master processes and the worker processes both run on one node and this node will have a docker container runtime pre-installed so you will be able to run the containers or the pods with containers on this node.

4.3 KubeCTL

Now that you have this virtual node on your local machine that represents minikube, you need some way to interact with that cluster so you need a way to create pods and other kubernetes components on the node and the way to do it is using **kubectl** which is a command line tool for kubernetes cluster.

Remember, we said that minikube runs both master and work processes. So one of the master processes called the api server is actually the main entry point into the kubernetes cluster. If you want to do anything in the kubernetes, like if you want to configure anything, create any component, you first have to talk to the api server and the way to talk to the api server is through different clients. You can have a UI, like a dashboard, you can talk to it using kubernetes api or a command line tool which is kubectl and kubectl is actually the most powerful of all the three clients because with kubectl, you can basically do anything in the kubernetes that you want.

Once the kubectl submits commands to the api server to create components, delete components etc, the work processes on minicube nodes will actually make it happen, i.e. they will be actually executing the commands to create the parts, to destroy the parts, to create services etc.

An important thing to note here is that kubectl isn't just for minikube cluster, if you have a cloud cluster or a hybrid cluster, kubectl is the tool to use to interact with any type of kubernetes cluster setup.

Acknowledgement

I would like to extend my sincere gratitude to everyone who gave me the chance to finish this report. I would especially like to express my gratitude to Dr. S. J. Patel Ma'am, the project manager for our senior project, whose contribution of energising suggestions and support allowed me to coordinate my project, particularly in writing this report. Additionally, I would like to express my gratitude for the important role played by Kunal Kushwaha, who taught me about the various facets of cloud networking and kubernetes.

References

- https://www.freecodecamp.org/news/the-kubernetes-handbook/#introduction-to-container-orchestration-and-kubernetes
- https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-is-kubernetes/
- https://kubernetes.io/docs/home/
- https://enterprisersproject.com/article/2017/10/how-explain-kubernetes-plain-english
- https://jsdw.me/posts/kubernetes/
- https://www.youtube.com/watch?v=KVBON1lA9N8&list=WL&index=2
- https://www.youtube.com/watch?v=IA90BTozdow&list=WL&index=2
- https://www.youtube.com/watch?v=17Bl31rlnRM