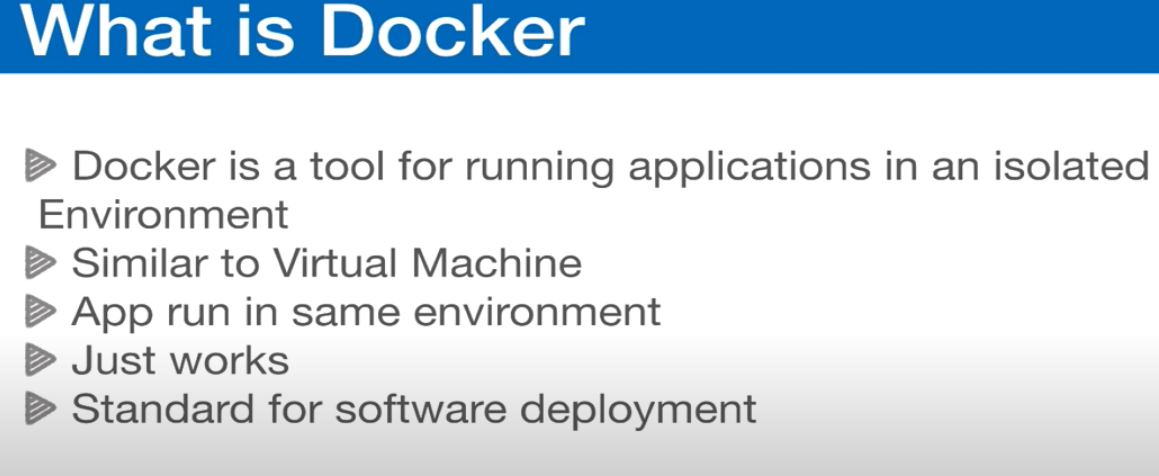
**Introduction to Docker**





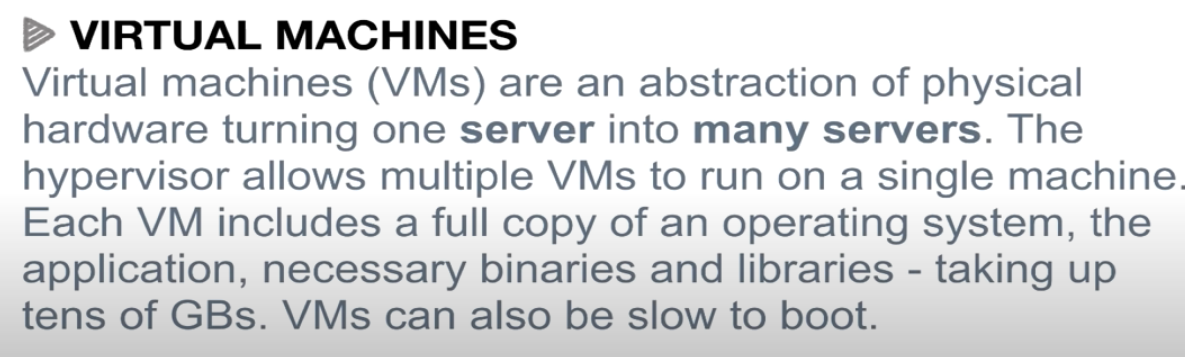
**------------------------------------------------------------------------------------------------**

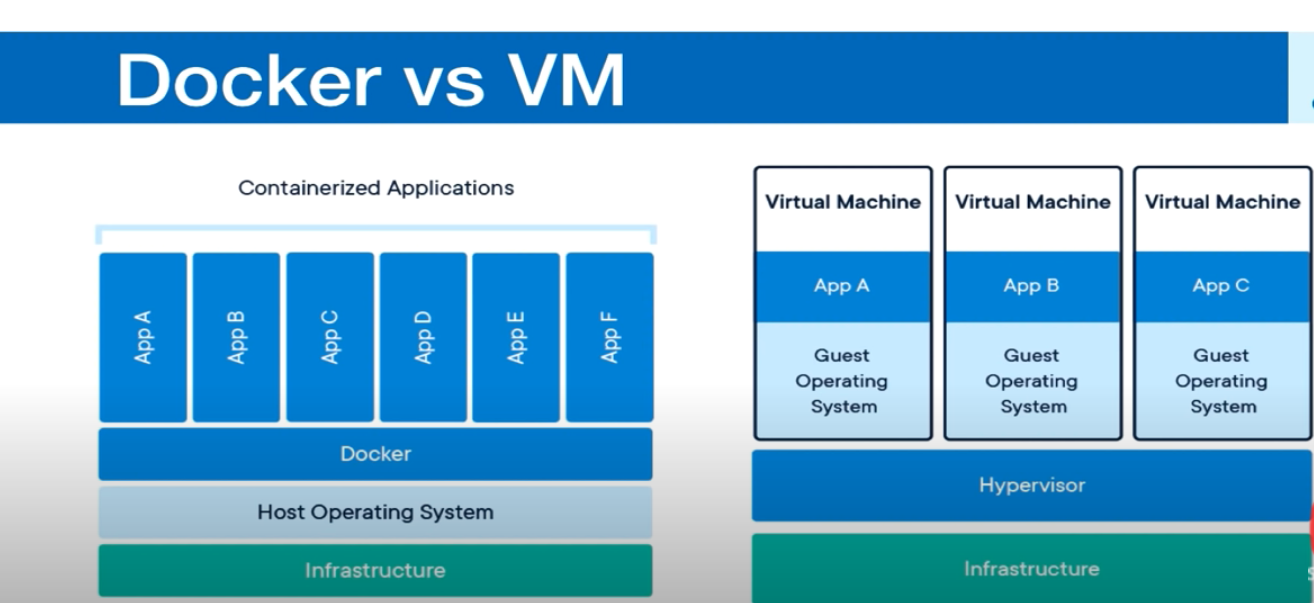
Docker is an operating system container management tool that allows you to easily manage and deploy applications by making it easy to package them within operating system containers. Docker’s **portability and lightweight** also make it easy to dynamically manage workloads, scaling up or tearing down applications, in near real time.One of the main benefits of using Docker, and container technology, is the portability of applications. It’s possible to spin up an application on-prem or in a public cloud environment in a matter of minutes.

**Containers vs Virtual Machines**



**A container does not require the full OS but runs on the underlying operating system , and each container is an isolated process.**





Terms “Containers” and “Virtual Machines” are often used interchangeably, however, this is often a misunderstanding. But, both are just different methods to provide Operating System Virtualization.

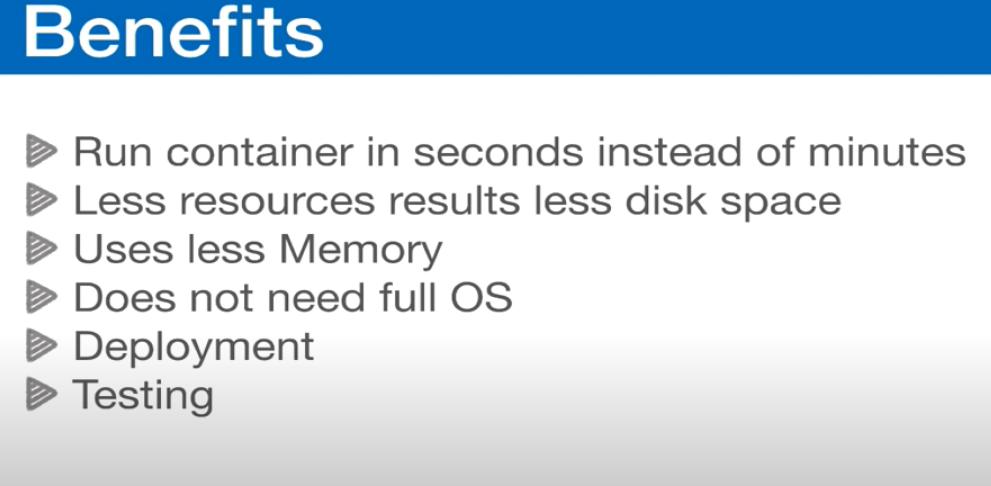
**Standard virtual machines** generally include a full Operating System, OS Packages and if required, few applications. This is made possible by a Hypervisor which provides hardware virtualization to the virtual machine. This allows for a single server to run many standalone operating systems as virtual guests.

**Containers** are similar to virtual machines except that Containers are not full operating systems. Containers generally only include the necessary OS Packages and Applications. They do not generally contain a full operating system or hardware virtualization, that’s why these are “lightweight”.

**Virtual Machines** are a way to take a physical server and provide a fully functional operating environment that shares those physical resources with other virtual machines.

Whereas, a **Container** is generally used to isolate a running process within a single host to ensure that the isolated processes cannot interact with other processes within that same system. Containers sandbox processes from each other. For now, you can think of a container as a lightweight equivalent of a virtual machine.

**Docker** enables creating and working with Containers as easy as possible.



Docker is a company that provides a container platform. Containers are a way to pack and isolate a piece of software with everything that it needs to run. I mean “isolate” in the sense that containers can assign separate resources from the host where it’s running. You might be thinking this[sounds pretty similar to VMs](https://www.docker.com/what-container), but the difference is that containers are more lightweight: they don’t need another OS to make software run. Containers let you be more [agile](https://stackify.com/agile-methodology/) and build secure and portable apps, which lets you save some costs in infrastructure when done well.

Let’s say John decides to start his containers journey. He learns that Docker containers work with base images as their foundation to run an app. A base image and all its dependencies are described in a file called “Dockerfile.”  A Dockerfile is where you define something like a recipe that you usually have in docs (or in your mind) for anyone who wants to run your app. He starts with the .NET Core app, and the Dockerfile looks like this. Take a look:

FROM microsoft/aspnetcore-build:2.0 AS build-env

WORKDIR /app

# Copy csproj and restore as distinct layers

COPY \*.csproj ./

RUN dotnet restore

# Copy everything else and build

COPY . ./

RUN dotnet publish -c Release -o out

# Build runtime image

FROM microsoft/aspnetcore:2.0

WORKDIR /app

COPY --from=build-env /app/out .

ENTRYPOINT ["dotnet", "hello.dll"]

As you can see, it’s as if you were programming. The only difference is that you’re just defining all dependencies and declaring how to build and run the app.

John needs to put that file in the root of the source code and run the following command:

docker build -t dotnetapp .

This command will create an image with the compiled code and all of its dependencies to run. He’ll only do the “build’ once because the idea is to make the app portable to run anywhere. So when he wants to run the app, only Docker needs to be installed. He just needs to run the following command:

docker run -d -p 80:80 dotnetapp

This command will start running the app on port 80 of the host. It doesn’t matter where he runs this command. As long as port 80 isn’t in use, the app will work.

John is now ready to ship the app anywhere because he’s packed it in a Docker container.

So why is this better? Well, John doesn’t have to worry about forgetting what he installed on his local computer or on any other server. When the team grows, a new developer will rapidly start coding. When John’s company hires an operations guy, the new hire will know what exactly what’s included in the container. And if they want to do an upgrade of the framework or some dependency, they’ll do it without worrying about affecting what’s currently working.

Use Docker to pack and ship your app without worrying too much about whether the app will work somewhere else after you’ve tested it locally. If it works on your machine, it will work on others’ machines.

https://www.youtube.com/watch?v=bhBSlnQcq2k

Kubernates:

https://www.youtube.com/watch?v=bhBSlnQcq2k

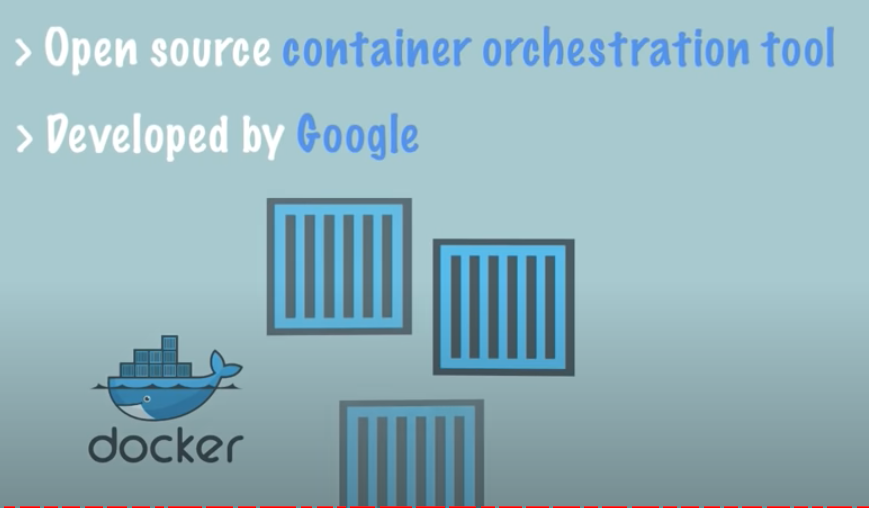
Using docker we can start building up complex applications which can use tons of containers (100 to 1000).These containers can talk to each other and interact with each other in some way.

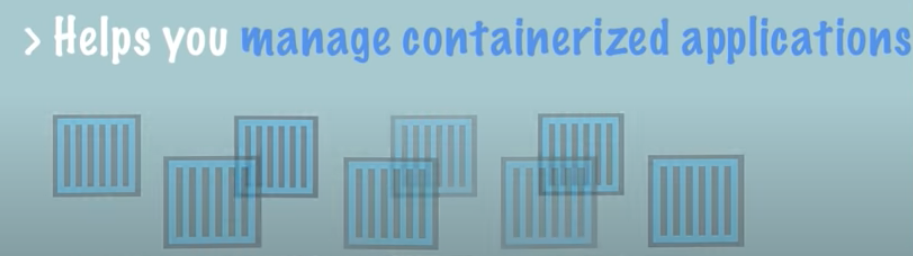
Also these containers have to be deployed in Servers (which can be distributed across in a distributed way. Now we can imagine how much efforts have to be made to image and manage these container with a very little automation way.

Whenever the containers crash or have problem in interacting with each other you have to manually interfere and fix these problems across different servers in a distributed system.

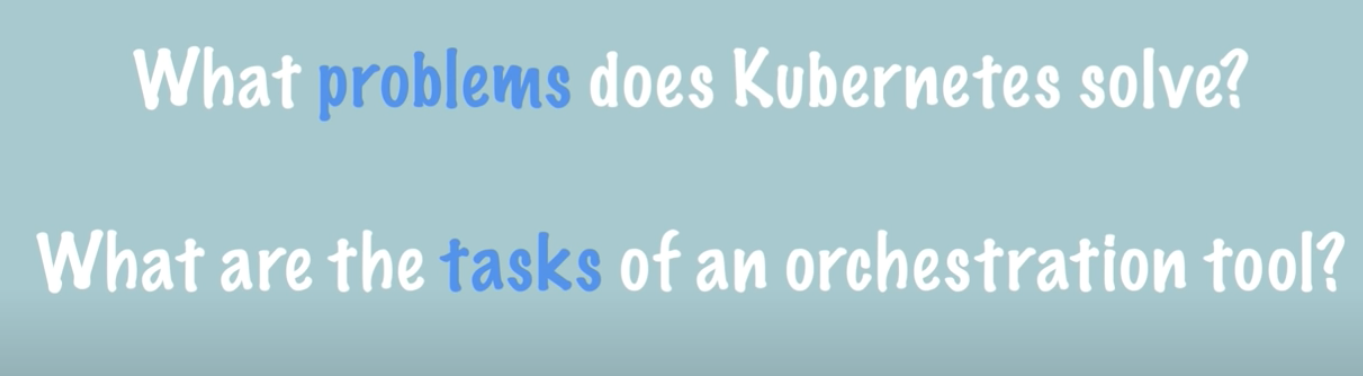
To fix up these problems and get the application in secure and recover the application state.

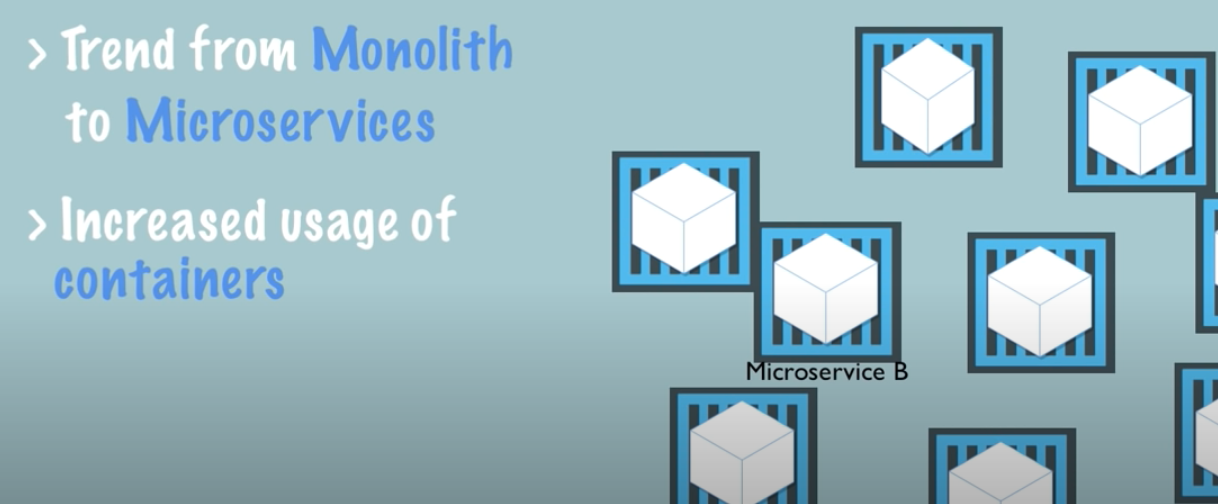
So it would be better we have a automated tool ie the container orchestration tool which is Kubernates. Kubernates is the most popular orchestration tool.

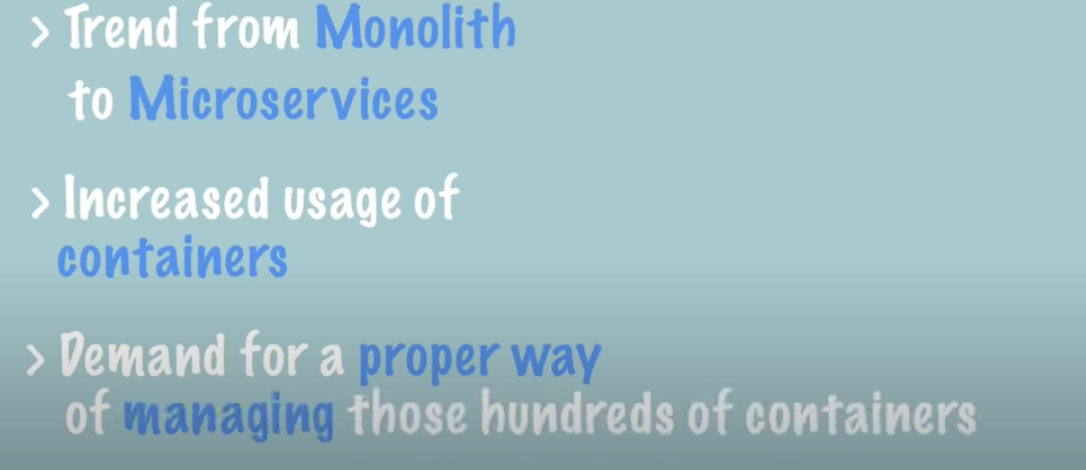












There is a need for container orchestration technologies

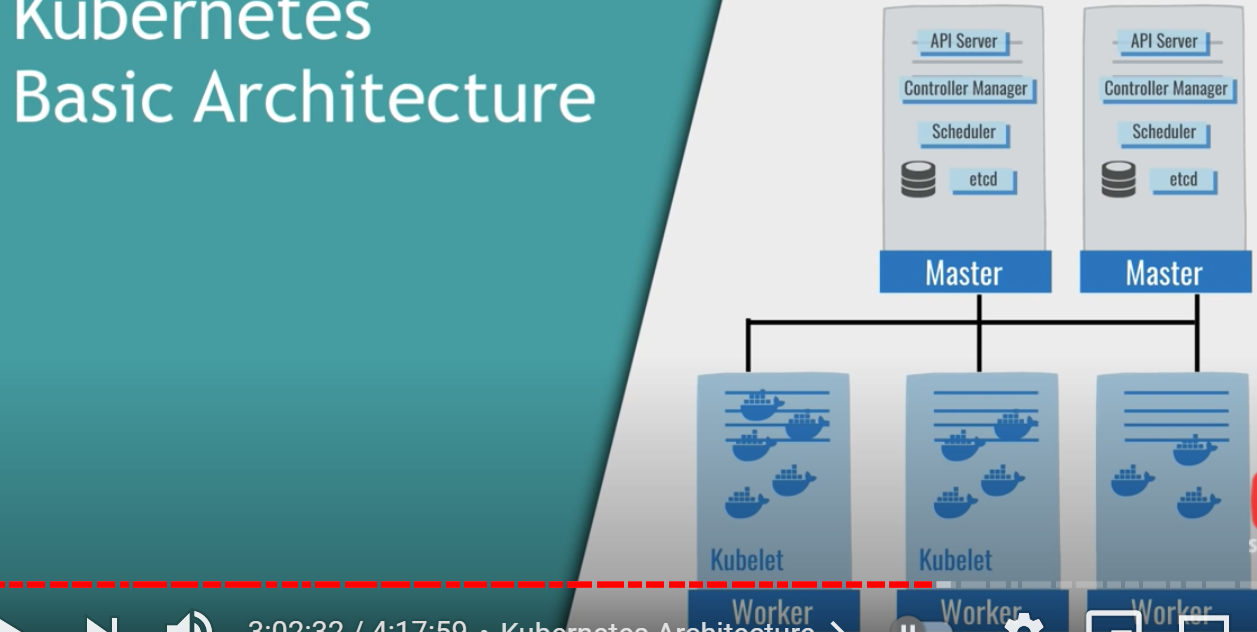


The application is available through out there is no time the application is down

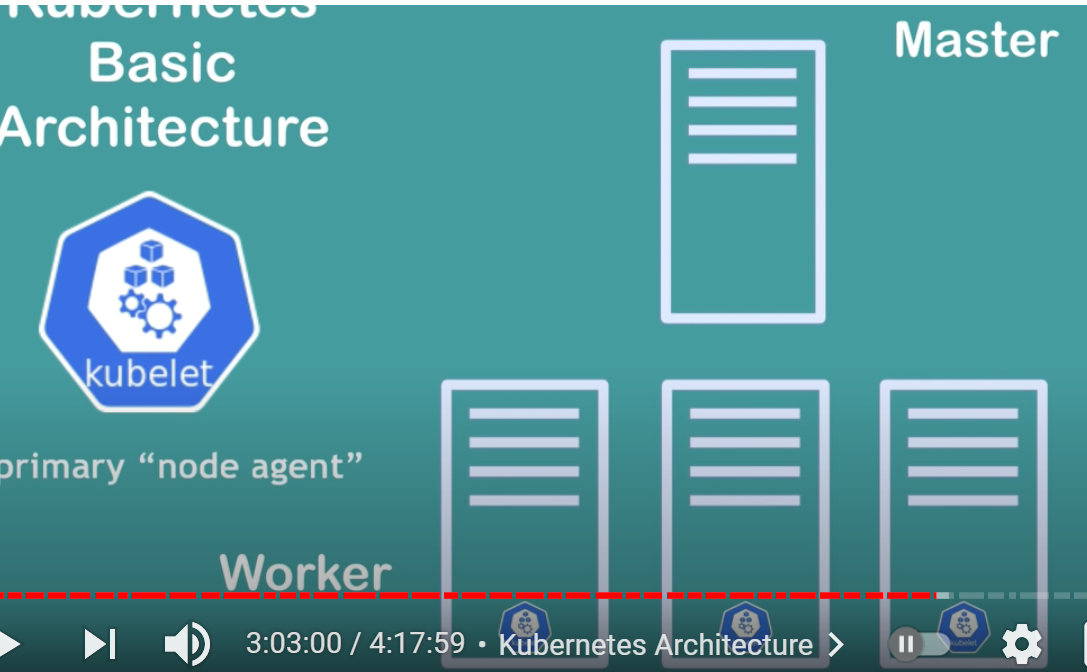
It is deployed with a condition that no user needs to wait for any response from the server

It has a fast responsive features for the users and can be accessed with unlimited users.

The Back up and recovery from any disaster due to server failure or any other factors are garunteed.

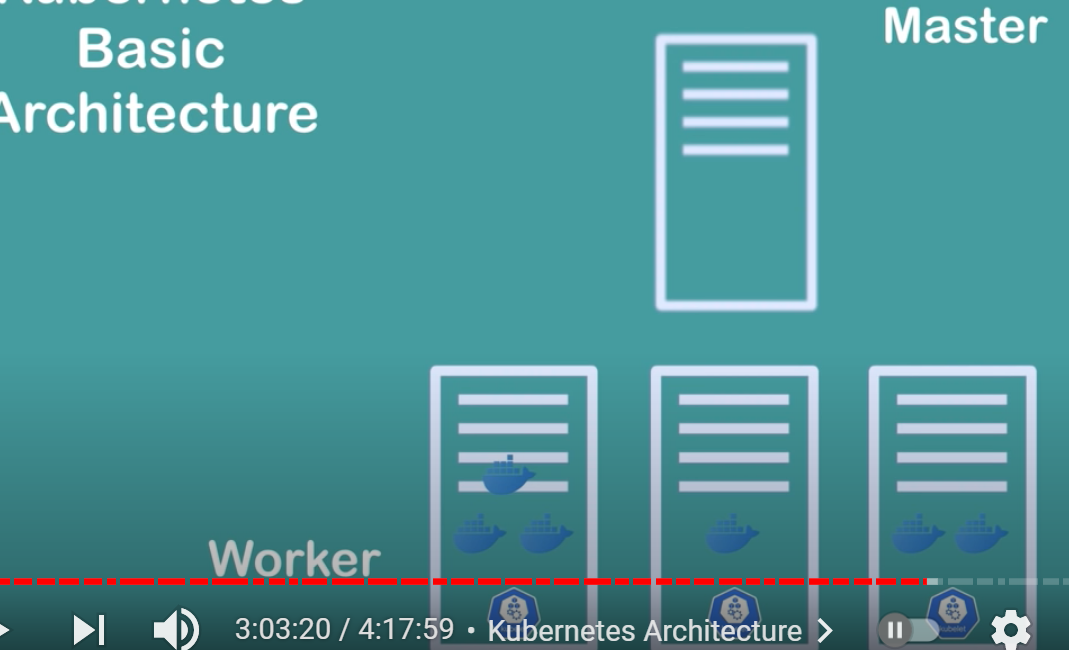


Master Nodes and Worker Nodes. Master nodes manages a hundereds of worker nodes.

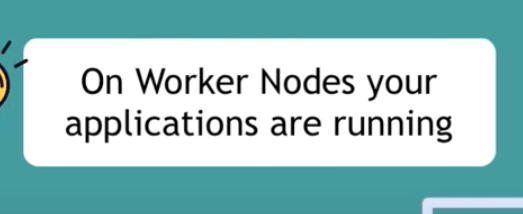


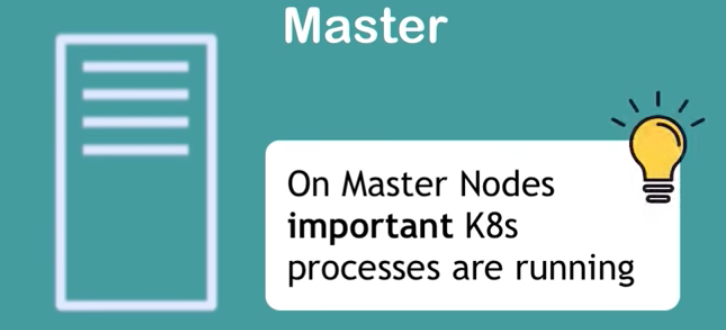
Kubelet is process which running on each node.

Kuelet is process by which these containers in the cluster they can talk to each other, communicated with each other and actually execute certain tasks also.



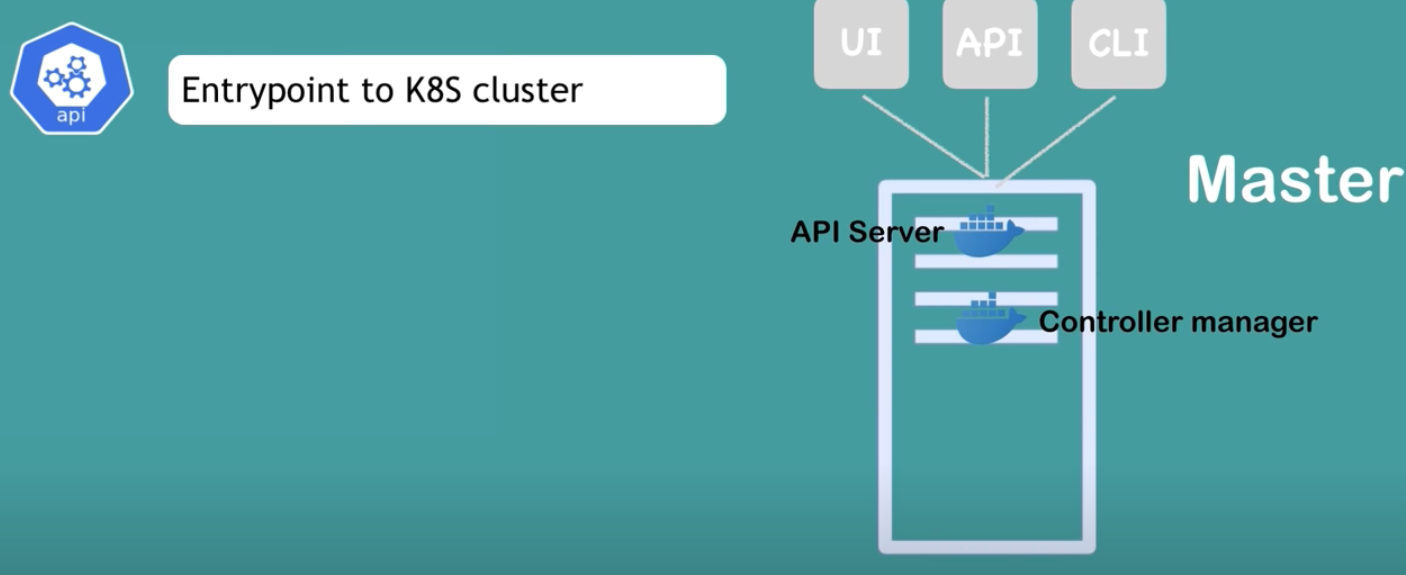
Different docker containers in working nodes.



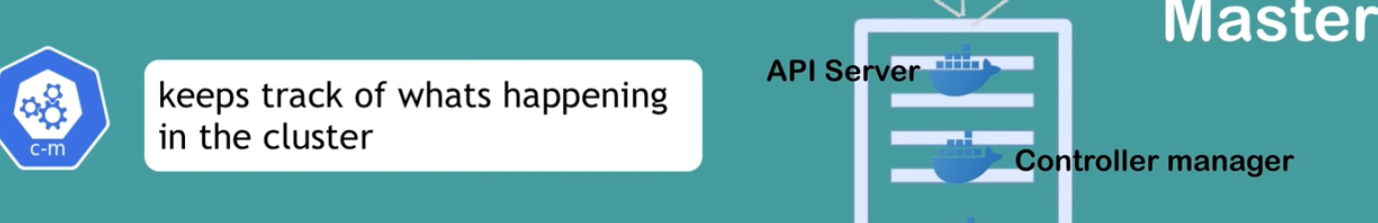




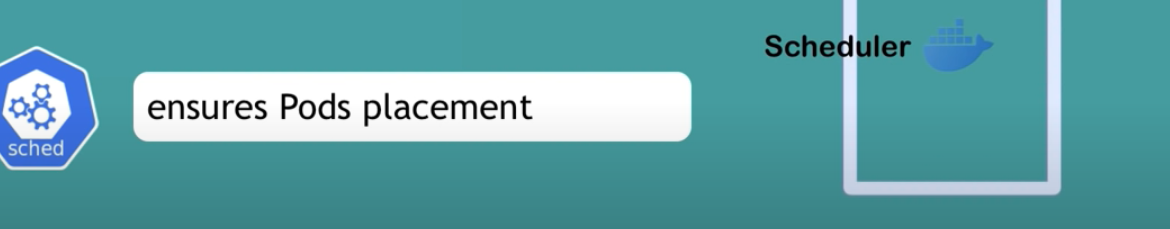
It is a container by which application developed in different servers can interact. Like you web servers, script servers ect. Can interact.



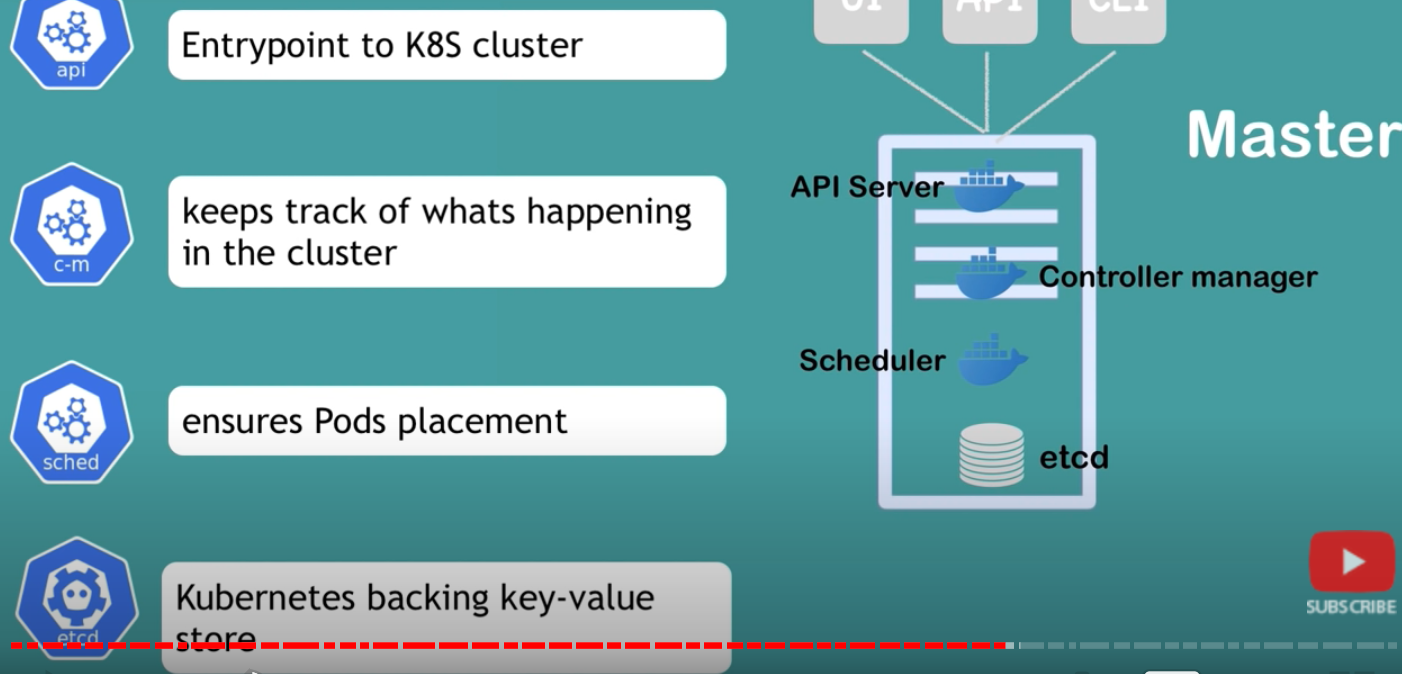
The servers in which your UI scripts runs, API for java prgrams run, also command line interfaces run all clustered servers can interact.



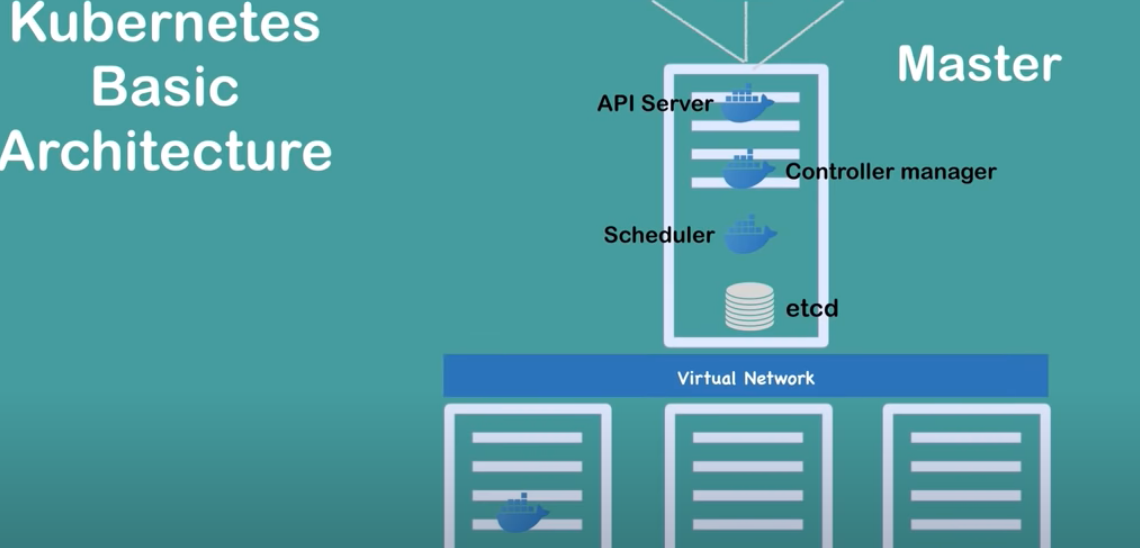
Is there any container failed , then restart it more managing the work space for the containers.



To schedule the containers in the clusters of work nodes. Which application container will work on which worknode. Which next worknode will be allocated or the incoming container. Based on the available worker nodes and the workload of the containers



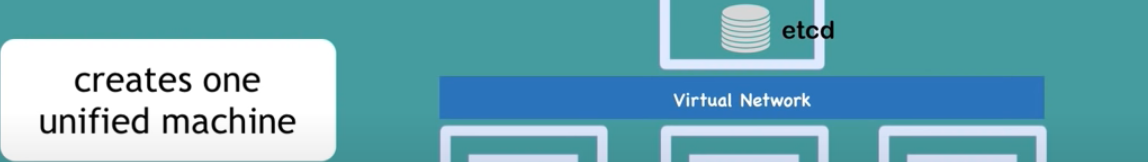
ETCD is sthe kubernetes backing key-value store to key track of the container and worknodes in the kubernates clusters.



Virtural network connects the workers with the master nodes.

Worker nodes are more high in configuration as applications run in it

At the same time master is very import as it spins all the worker nodes



Virtual network integrates all the containers as if it is all put together working in a singl unified powerful machine.

