Kubernetes - Introduction

Section 1 : Introduction

Course overview

- **Through this course you can understand the basic concept of Kubernetes**

- **Followed by some demos on how to setup Kubernetes test environment.**
- **Also we will be doing some hands-on coding exercise which can help you to understand Kubernetes better.**
- **Over the course I can also share some tips & tricks which can help you to create your own Kubernetes code.**

Kubernetes Overview

- **Kubernetes is also know as ("K8s" or "Kube") was built by google based on their experience running containers in real time.**
- **What does it mean K8s ? [K8s is derived by replacing the eight letters of ubernete]**
- **Also, its an open source product & its arguably one of the best and most popular container orchestration technology out their.**
- **To understand Kubernetes we must first understand 2 things [Container + Orchestration]**

Lets start with Container :

- **We are familiar with container already which we were dealing with last couple of days.**
- **Lets have small recap though.**
- **Right now the popular technology out there is Docker which help us to simplify hosting applications in container model.**
- **In Nutshell, the real use case of Docker is to install & run software without worrying about setup or dependencies.**

Container Orchestration

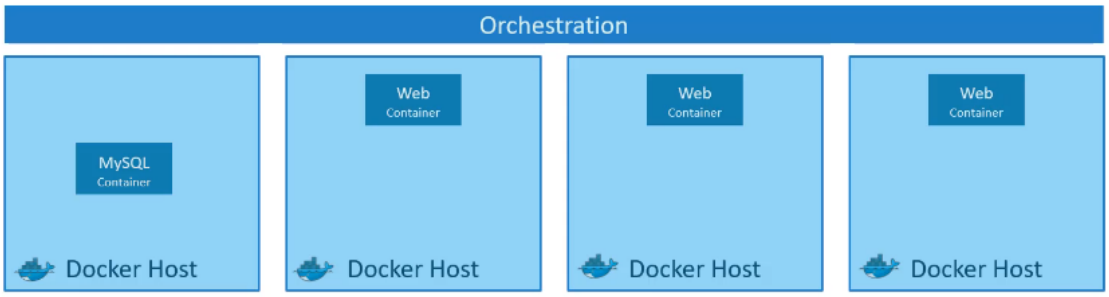
- **We know how our application is packaged into a docker container but what's next ?**
- **how do you run in at production ?**
- **What if your application relies on other containers such as databases, messaging services or other backend services?**

- **How you are going to scale up and scale down your application according to the incoming work load?**

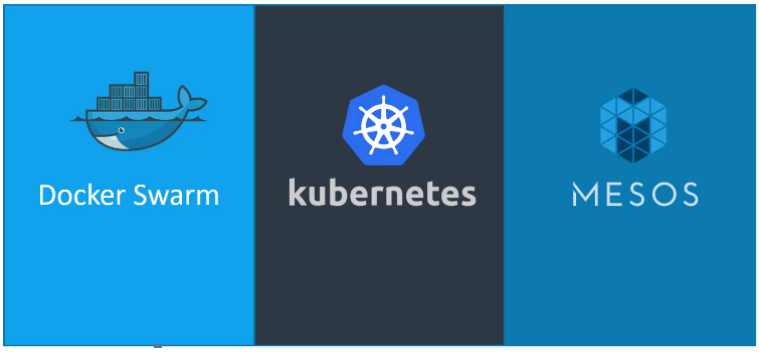
To enable all this functionalities you need a platform which has these capabilities.

The platform need to orchestrate the connectivity between containers.

Also, It should be able to automatically scale up and scale down based on load.



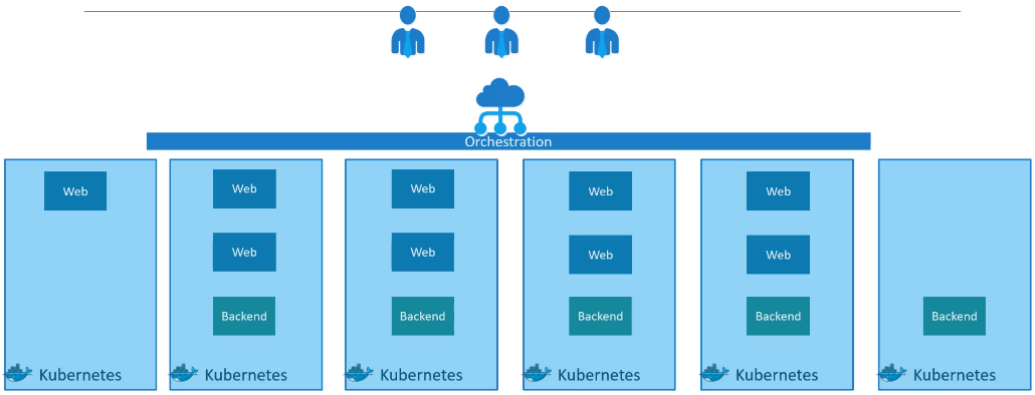
- **This whole process of automatically deploy and managing containers is known as container orchestration.**
- **There are many tools available in this segment.**
- **Docker swarm is native docker feature which will help for container orchestation.**
- **Kuberenetes is from Google.**
- **Mesos from Apache.**
- **Every tool has its own advantage and disadvantage, Kubernetes is standing out among them since because it is widely accepted.**
- **Kubernetes is bit difficult to setup and get started but provides lot of options to customize deployments and supports deployments of complex architecture.**



- **Kubernetes is also now supported on all public cloud platforms like AWS, GCP & AZURE.**

Advantages of Kubernetes

- **There are various advantages in using Kubernetes.**
- **Your application is highly available as hardware failure do no bring your application down because we have multiple instances of your application running on different nodes.**
- **The user traffic is load balanced across multiple containers. [When demand increases deploy more instances of the application seamlessly in matter of seconds]**
- **When we run out of hardware resources can scale up or down the underlying nodes without disturbing running application.**



- **To do all such thing we need flexible and easy to manage tool and that going to be Kubernetes.**
- **In Nutshell, Kubernetes is an container orchestration tool [which helps to orchestrate 100s & 1000s of container deployment and managing all those**

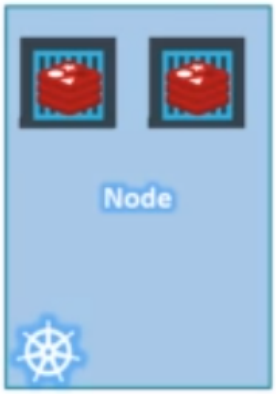
in an easy way..]

Kubernetes Architecture

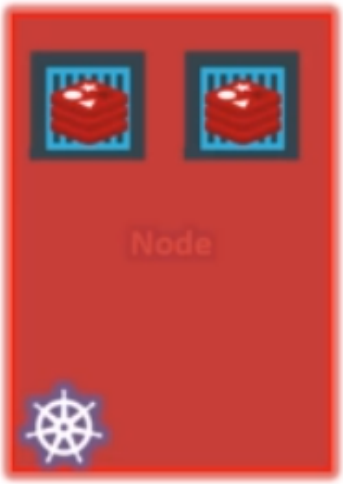
- **Before we get into setting up the Kubernetes it is important to understand some of the basic concepts.**
- **Lets first understand what is nodes.**

Nodes

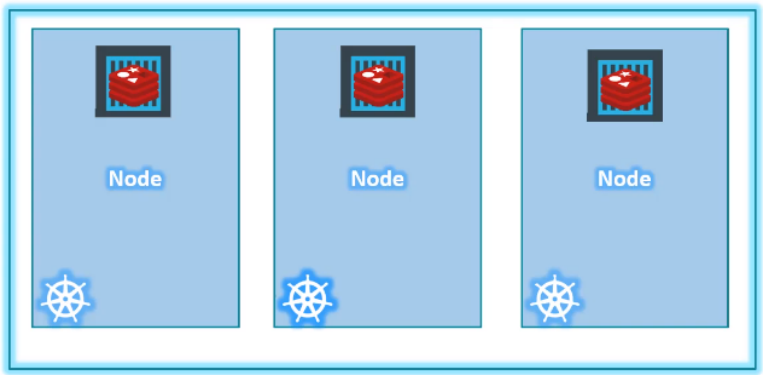
- **A Node is a machine [Physical or Virtual] whatever it could be where Kubernetes is installed.**
- **Nodes is also known as worker machines where containers is launched by Kubernetes.**
- **Nodes were also known as Minions in past.**



- **But the real question here would be what if the node goes down? Eventually your application may go down isn't ?**



- To avoid such scenario we are going to have more then one nodes which is called cluster.
- A cluster is set of nodes grouped together. [This way even if one node fails your application might be still accessible from other nodes]

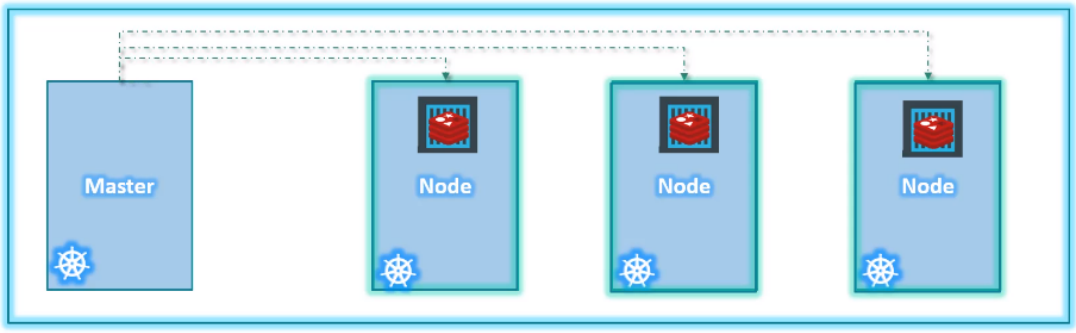


- Moreover having multiple nodes help to share the load.

Master

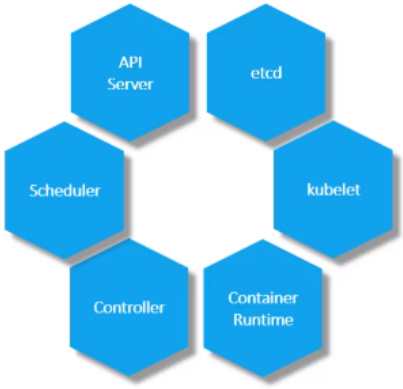
- Now we have cluster but who is responsible to manage this cluster ?
- Where is the nodes information's stored?
- How are the nodes monitored ?
- When a node fails how the workloads are migrated to other node ?
- That's where the "Master" comes in.

- **Master is an other node with Kubernetes installed in it and it is configured as Master.**
- **Master is responsible to monitor the worker nodes and manage the container orchestration.**



Components

- **When you are installing Kubernetes it actually installs following components.**



- **API SERVER ： Acts as frontend for Kubernetes. [Users, Command line interfaces all talk to API SERVER to interact with Kubernetes cluster]**
- **ETCD ： ETCD is distributed reliable Key-value store used by Kubernetes to store all data used to manage cluster. [To put it simple this component is going to store all the information's about Master & Node in distributed manner] When I say distributed "information's are not going to be centrally store instead its going to be stored in each**

machines locally". ETCD is also responsible for implementing logs within the cluster to ensure that there are no conflicts between Masters.

- **SCHEDULER : SCHEDULER is responsible for distributing work (containers) across multiple nodes. The main role of this component is to look for newly created container and assign it to node in an automated way.**
- **CONTROLLER : CONTROLLER is the brain behind orchestration, they are responsible to notice and respond when nodes or containers goes down. Controllers are also responsible to make decisions to bring new containers in such situations.**
- **CONTAINER RUNTIME : This component is the piece of software used to run containers. In our case it happens to be Docker.**
- **KUBELET : KUBELET is an agent which runs in each node inside cluster. Agents are responsible to ensure containers are running in nodes as expected.**

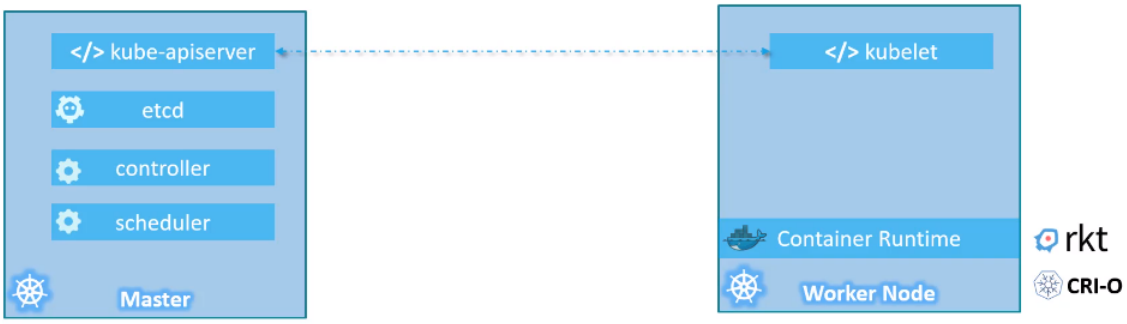
Master vs Worker node

So far we just saw what is master & worker node and set of components which are used to run the show without any hiccups.

- **But how are these components distributed across multiple server?**
- **In other how are we going to classify which machine is Master and which one is Worker ?**
- **The worker node is the place where container is going to reside. For example Docker container.**
- **To run docker container - we may need component "container runtime" configured in machine.**
- **We are going to use docker in this segment though we have lot more components available like [Rocket & CRI-O]**
- **Master server is going to have [Kube-apiserver] configured in it. This component is actually making it as Master.**
- **Similarly worker node is going to have [KUBELET agent] that is responsible to make interact with Master [Which also transfers health**

information about worker node and carry out actions requested by Master]

- Other components like ETCD, Scheduler & Controller is going to residing on Master.



KUBECTL

- KUBECTL (KUBE control) is an command line utility used to deploy and manage containers in Kubernetes cluster.
- Also it is useful to get the cluster/node/master/container information's.
- In Nutshell, we are going to use several KUBECTL commands to create and manage containers in Kubernetes cluster.

Section 2 : Setup Kubernetes

Various options available to build Kubernetes cluster

There are many ways to to setup Kubernetes.

We can set it up in our laptop or virtual machines using solutions like

- MiniKube
- MicroK8s
- Kubeadm

There are also hosted solutions available to setup Kubernetes environment.

- GCP

- **AWS**
- **Azure**

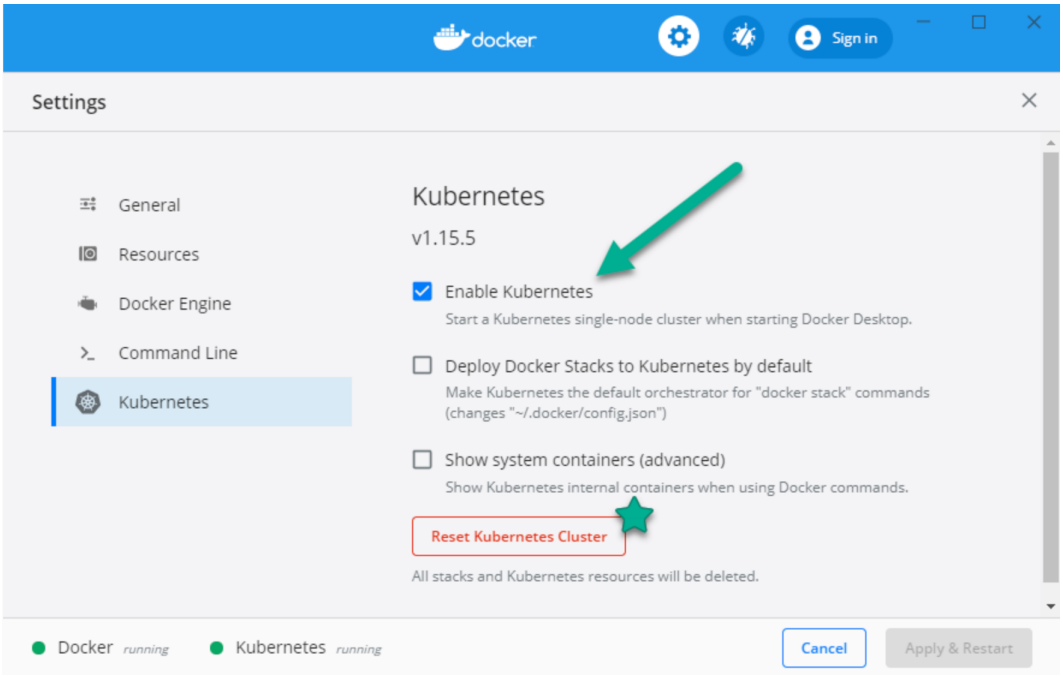
These are few among the many of solutions to host Kubernetes cluster.

How are we going to practice ?

We will be using the Kubernetes solution which is provided by Docker Desktop.

Steps to enable Kubernetes in Docker desktop

- **Install docker desktop [I hope which everyone already done]**
- **Enable Kubernetes**



- **That's it! Docker Desktop will download all the Kubernetes images in the background and get everything started up. When it's ready you'll see two green lights in the bottom of the settings screen saying Docker running and Kubernetes running.**
- **Also, the star in the screenshot shows the "Reset Kubernetes Cluster button", which is one of the reasons why Docker Desktop is the best of the local Kubernetes options.**
- **Click that and it will reset your cluster back to a fresh install of Kubernetes.**

Verify your Kubernetes cluster

- **If you've worked with Docker before, you're used to managing containers with the "docker & docker-compose cmd lines".**
- **Kubernetes uses a different tool called "kubectl" to manage apps - Docker Desktop installs kubectl for you too.**

Check the state of your Docker Desktop cluster

```
kubectl get nodes
```

- **You should see a single node in the output called docker-desktop.**

- **That's a full Kubernetes cluster, with a single node that runs the Kubernetes API and your own applications.**

Other Kubernetes components

- **The Kubernetes components are running in Docker containers, but Docker Desktop doesn't show them by default to keep things simple when you're running docker commands.**

```
docker ps
```

```
docker ps -a
```

- **Both commands are not going to show Kubernetes containers.**
- **But you can try below command to see the number of containers running in your machine.**

```
docker info
```

- **Also, you can check the docker images command to see the configured containers.**

```
docker images
```

Section 3 : Kubernetes Concepts

PODs

Before we get into PODs, lets try to assume the following have been setup already.

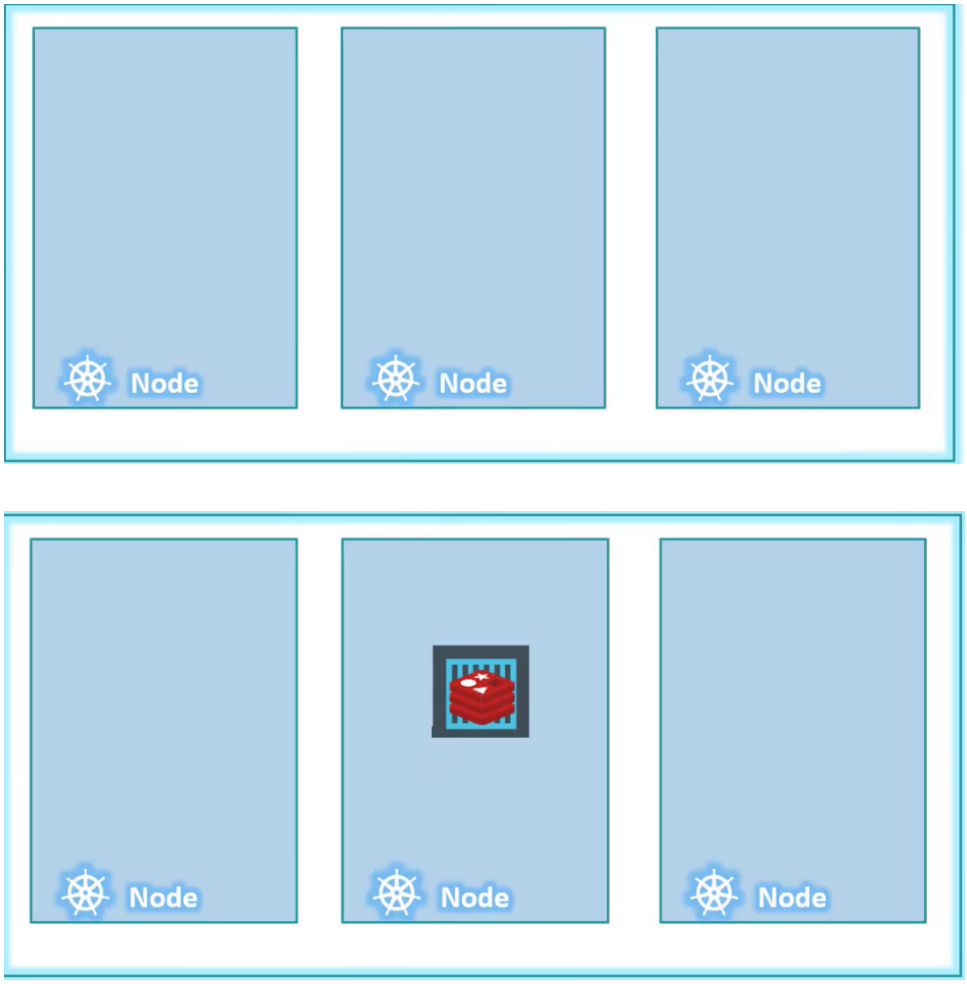
- **At this point, lets assume the application is already developed and built into docker images and it is available in the docker repostitory (Docker hub), Hence Kubernetes can pull it down.**

- **Lets also assume Kubernetes cluster is already been setup and working.**

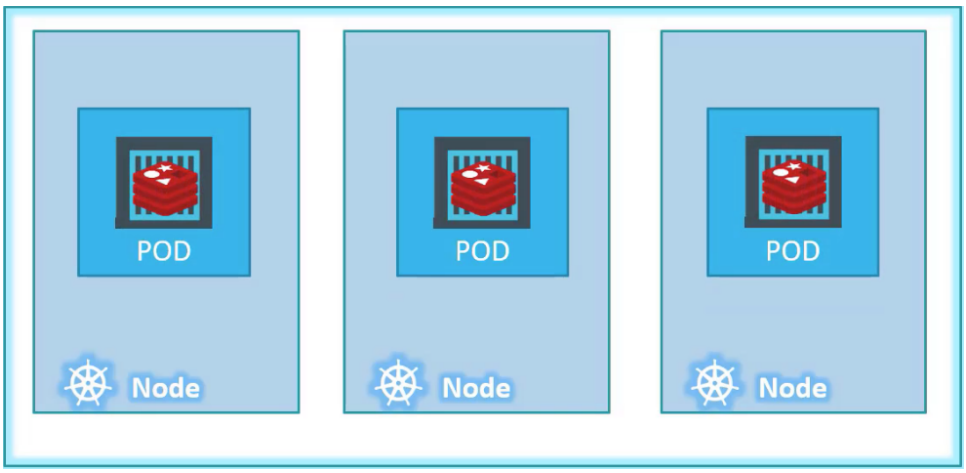
Docker Image

Kubernetes Cluster

- **As we discussed before with help of Kubernetes our ultimate aim is to deploy applications in form of containers on a set of machines. [Which are configured as worker nodes in a cluster]**



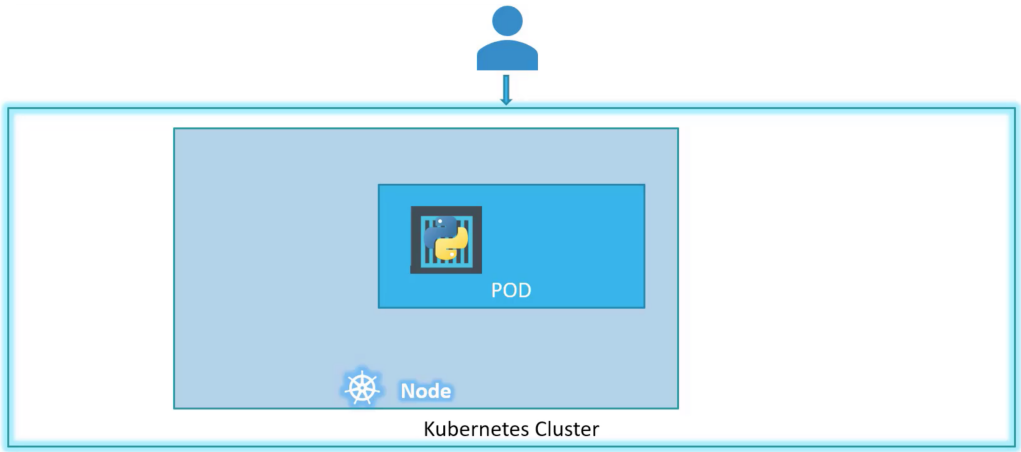
- **However Kubernetes is not going to deploy containers directly in worker nodes.**
- **Instead, containers are going to be encapsulated into the Kubernetes object called PODs.**



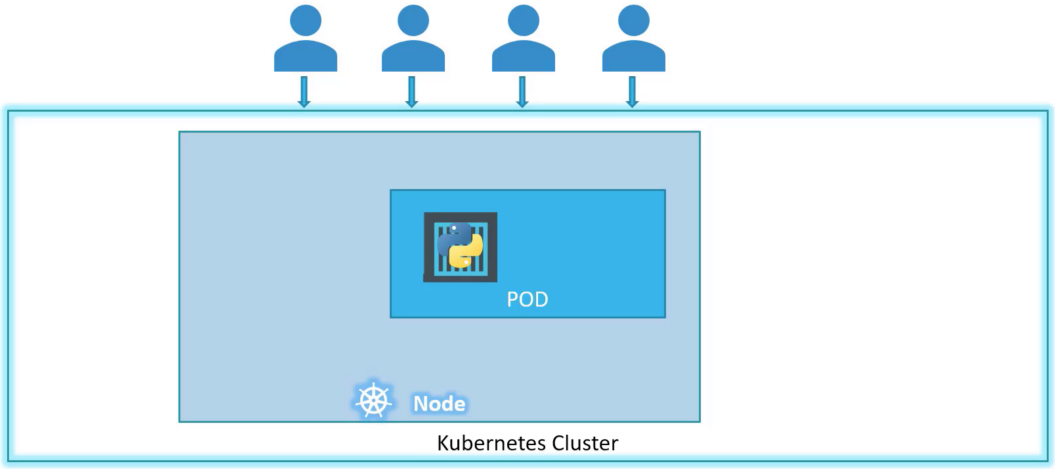
- **PODs is the smallest object which you can create in Kubernetes.**

Lets try to understand this more in details

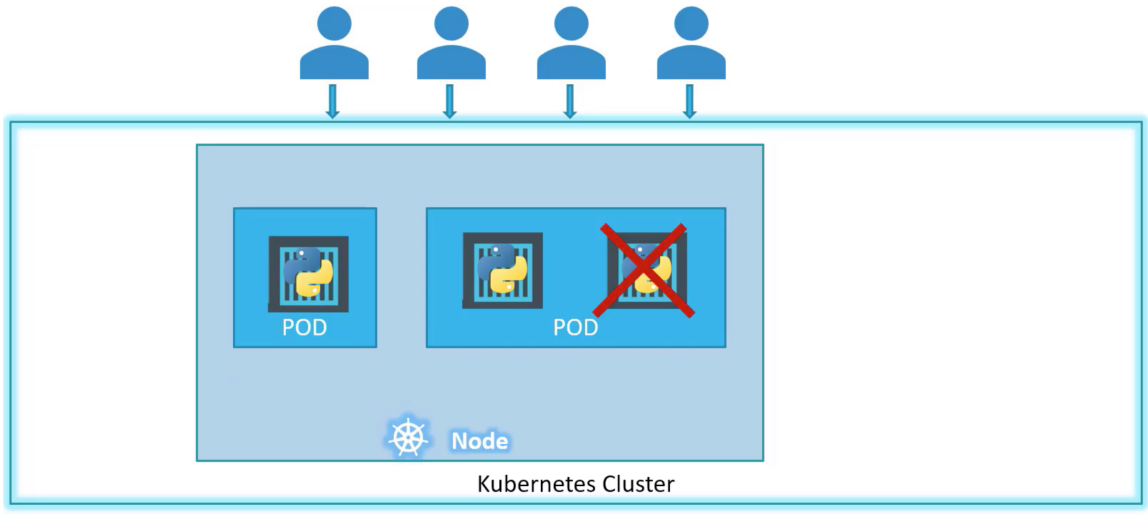
- **Here we see, we have single node Kubernetes cluster with single container contains your application which is running inside Pod.**



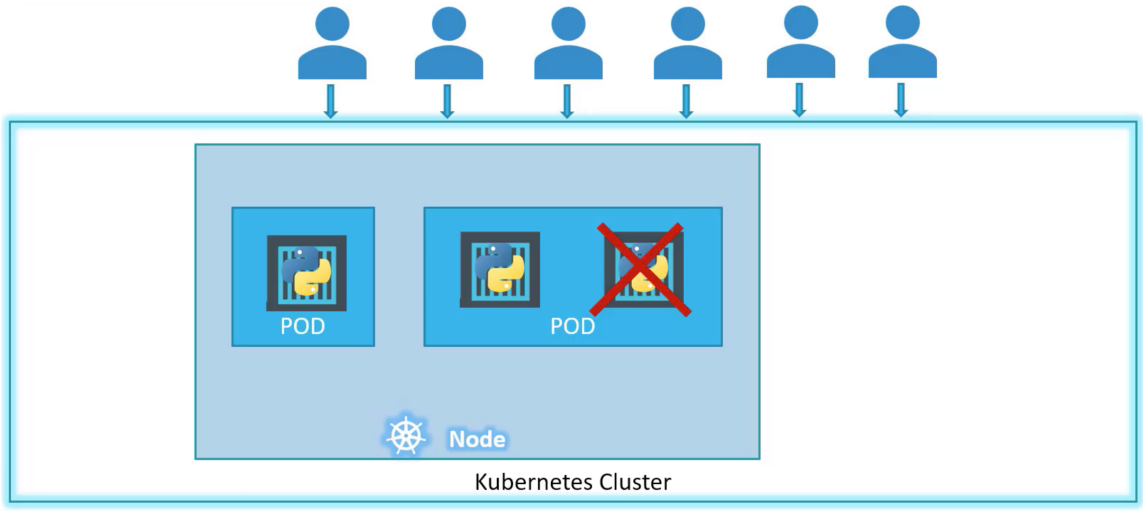
- **Right now the number users accessing your application is one, What if the number of user accessing your application is getting increased ?**



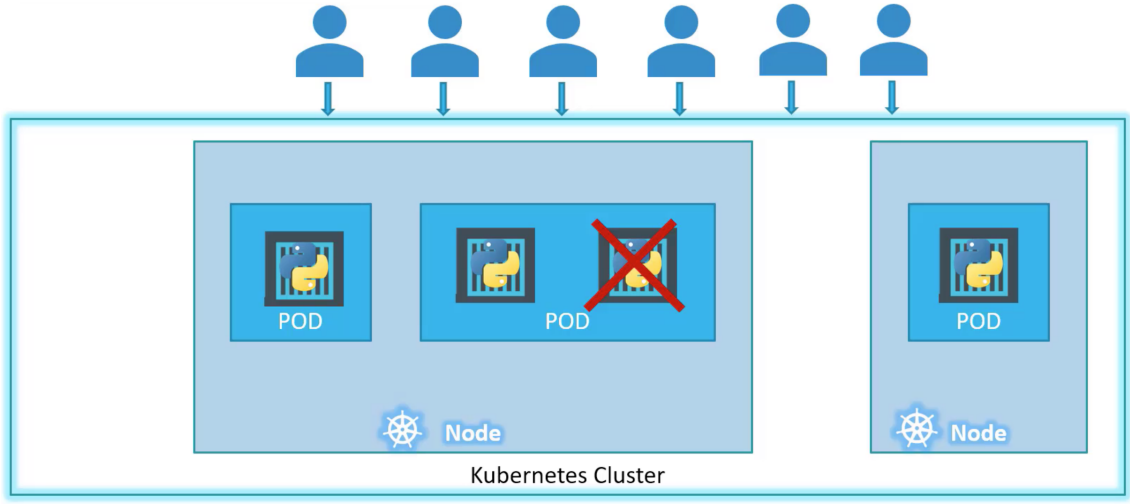
- **In this case you may need to scale your application to server the demand.**
- **You may need to create additional instances to share the workload.**
- **Now where would your spin up additional instances (containers).**
- **Do we bring new instances within the same PODs ?**
- **No we are going to create new pod altogether which is going to contain fresh container with same application.**
- **As you can see, we now have two instances of our web application running on two separate PODs on same Kubernetes cluster.**



- **But what if the user base further increases and your current node has sufficient capacity ?**



- **Well in that case you can create new set of nodes and add it to cluster.**
- **Further PODs will be deployed in on a new node, This will expand the clusters physical capacity.**



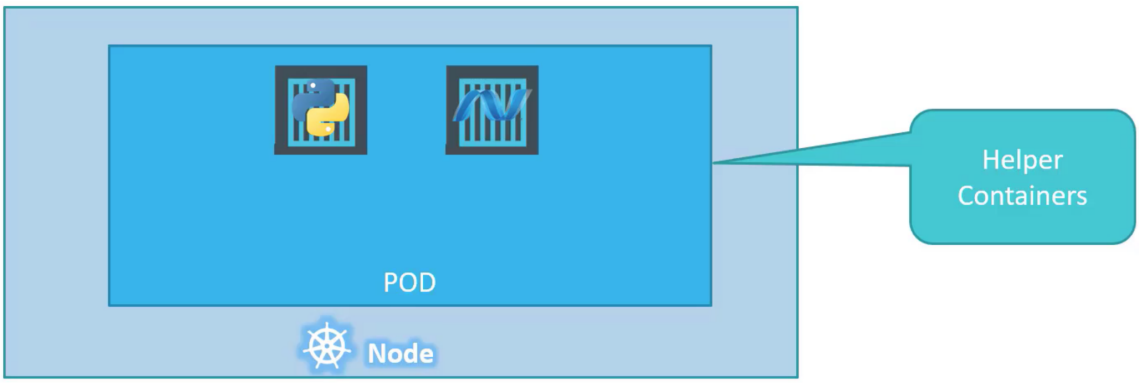
To summarise :

- **POD is an Kubernetes object which is going to help you to encapsulate and run the container.**
- **PODs usually have 1-1 relationship with containers which is running your application.**
- **To scale up you can create new POD and to scale down you can delete the existing POD.**
- **Also, you are not going to add any containers into existing POD to scale your application.**

Multi-container PODs

I just said that PODs usually maintain 1-1 relationship with containers but are we restricted to having single container in an single POD ? and the Answer would be No.

- **A single POD can have multiple containers, but the criteria to have many container in POD is every container should server different purpose.**
- **As we discussed earlier, if our intention is to scale up the existing container we should always go with new POD.**
- **Lets assume this scenario, we have an web application and it may require some helper container which is going to serve required data main container. [Basically supporting machines for main application container]**
- **In this case you may need your helper containers to live alongside with your main container.**

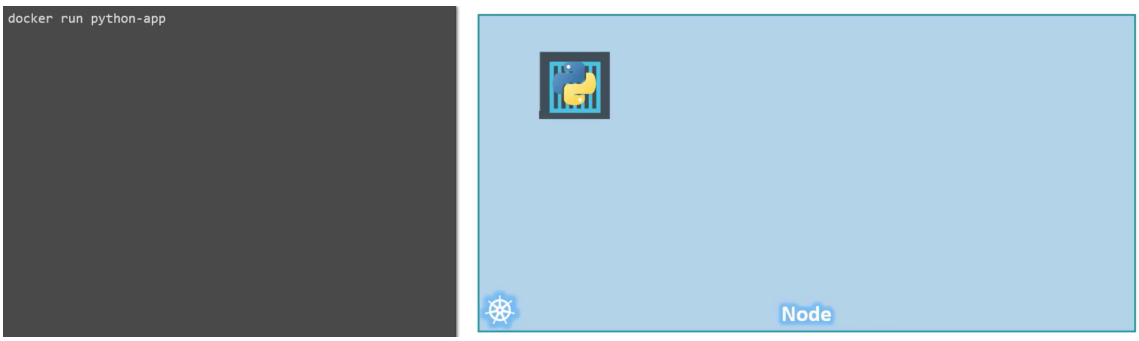


- **As an exceptional Kubernetes allows you to put this kind of containers into same POD.**
- **Now, when an application container is created the helper containers also get created and when it dies helper also dies since they are part of same POD.**
- **These containers can communicate each other directly by referring to each other as localhost since they share the same network space.**
- **Also they can easily share the same storage between each other.**

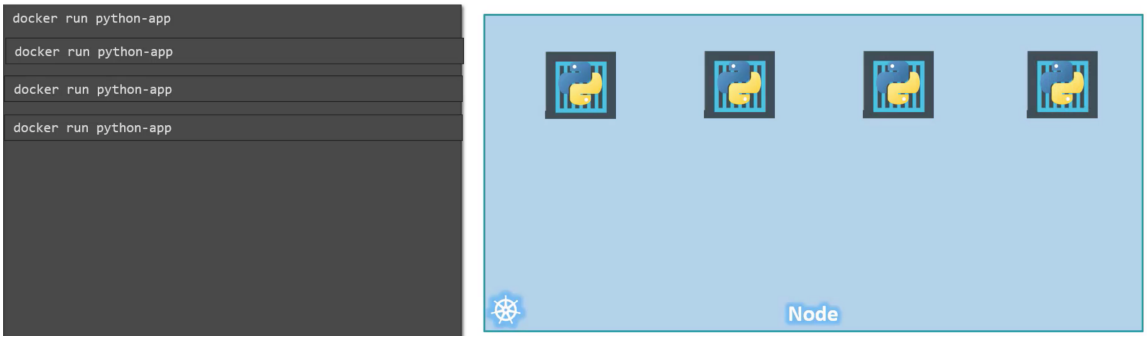
PODs summarize

To understand better about PODs - Lets try to understand the below scenario.

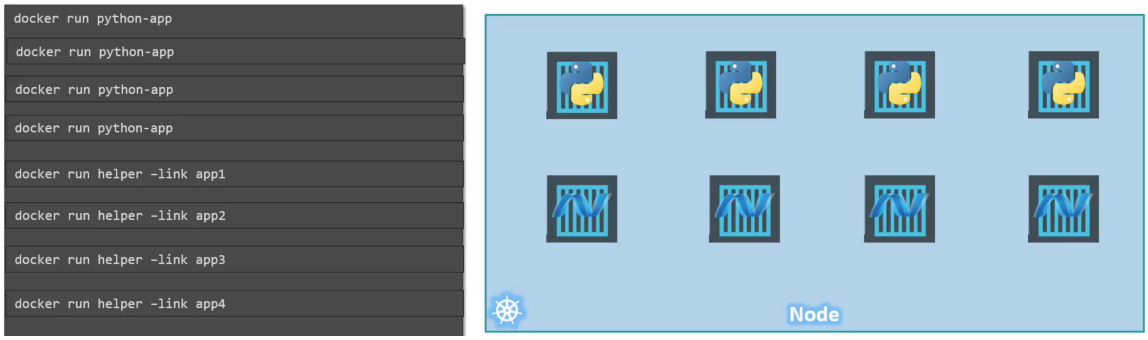
- Let's not think about Kubernetes now.
- Imagine you want to host your application in an container.
- You are going to simpy trigger docker run command and create container which will accessed by your customers.



- Eventually over the period of time when load increases you main create more number of containers more similar to that.



- In future, our application is further developed undergoes architectural changes. In this case number of machine grows and gets complex.

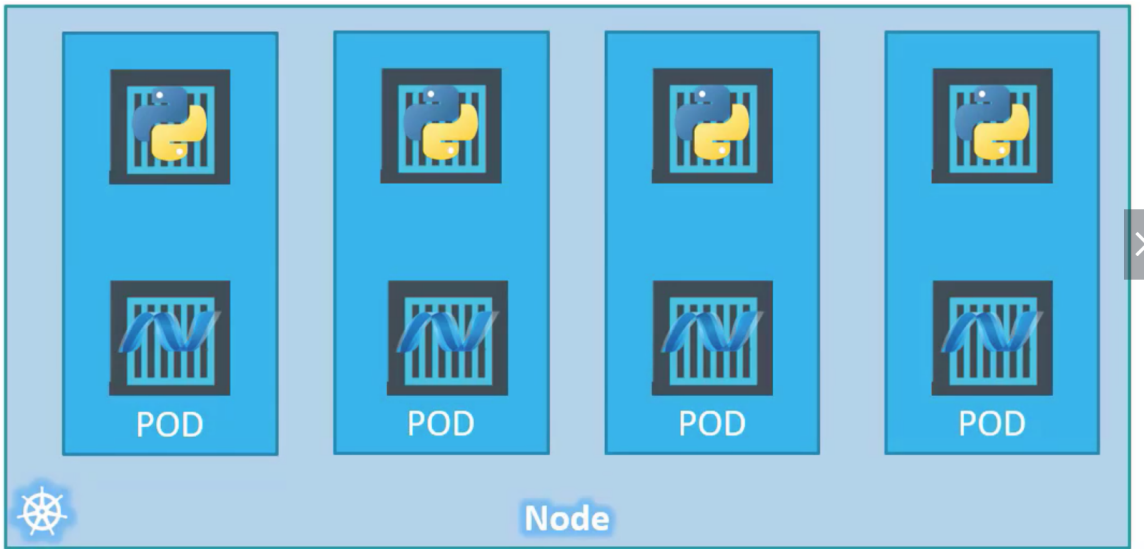


- In this case you may need to care of enabling communication between containers.

- Ensure machines are servers with proper shared storage.
- Also keep a track of which machine is created for which purpose.
- Also need to ensure removing/creating helper machine when relevant master machines is removed.
- Hence while your infrastructure grows it may become very difficult to manage these things.

Now with help of Kubernetes "PODs" we are going to eradicate all this complications because PODs will take care of

- Creating & removing containers.
- Maintaining communication between other machines in POD.
- Ensuring Network & Storage is properly shared among machines.



- If you think in longer run this way of implementation is good by foreseeing the upcoming architectural changes and scalability.
- Also note that multi-pod container is a rare use case. Our focus would be more on single container per PODS in upcoming sessions.

Section 4 : PODs Hand-on

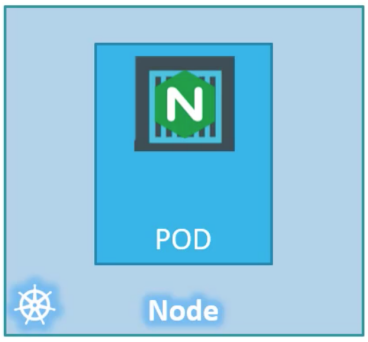
Lets now look at how to deploy PODs.

- **As you know to "talk with Kubernetes we are going to use command line utility called KUBECTL"**
- **We are going to trigger the command which is going to create POD.**
- **This POD is going to have "Docker container deployed inside to it"**

```
kubectl run nginx
```

- **First this command is going to create only POD.**
- **To deploy container into POD you may need to provide additional parameter "—image"**

```
kubectl run --generator=run-pod/v1 nginx --image=nginx
```



- **Application image is going to be downloaded from Docker Hub in this case.**
- **Now we have the PODs created with container, how do we see the list of PODs available.**

```
kubectl get pods
```

- **We can also get more detailed information about PODs with below command.**
- **This command provides events about your PODs and many more detailed informations.**

```
kubectl describe pod nginx
```

- **We can also use another command which give more information about PODs.**
- **This command can tell you where the POD is exactly located out of your worker nodes.**

```
kubectl get pods -o wide
```

- How to login into the container.

```
kubectl exec -it nginx -- /bin/bash
```

- Delete created PODs

```
kubectl delete pod nginx
```