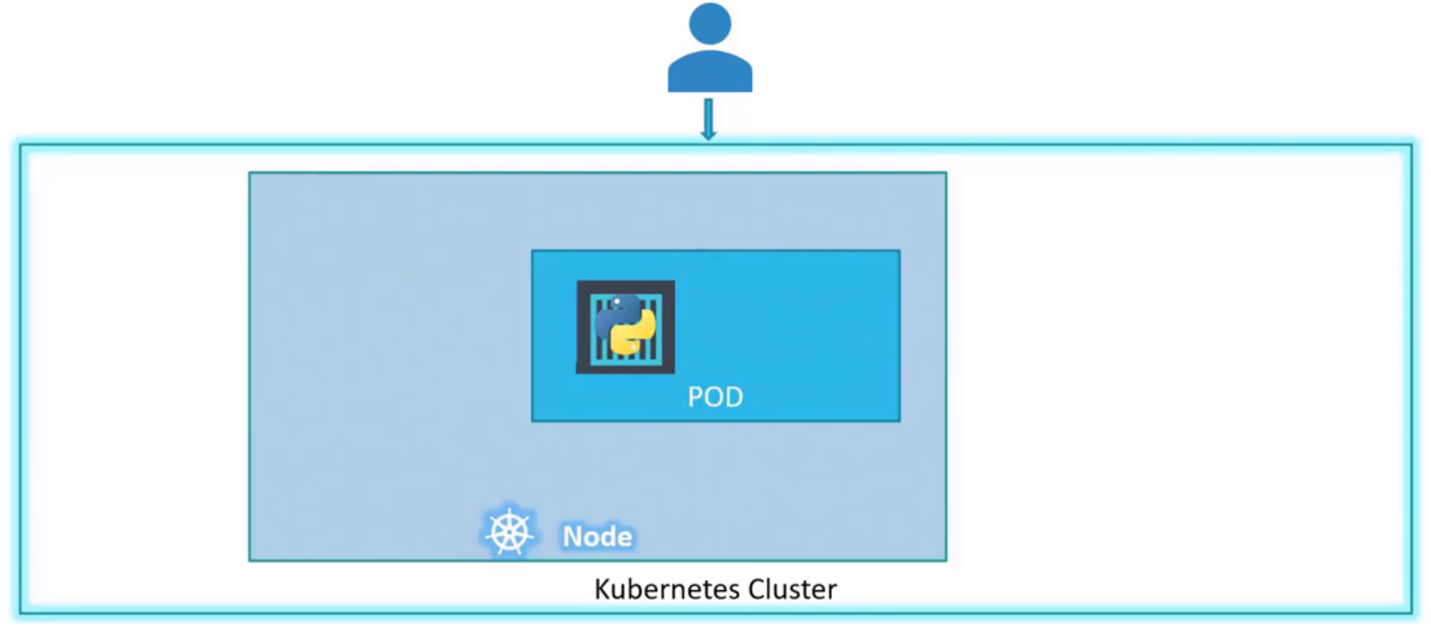
At this point, we assume that the application is already developed and built into Docker images and it is available on a Docker repository like Docker Hub, so Kubernetes can pull it down. We also assume that the Kubernetes Cluster has already been set up and is working. This could be a single node setup or a multi-node set up. All the services need to be in a running state.

With Kubernetes, our goal is to deploy our application in the form of containers on a set of machines that are configured as worker nodes in a cluster. However, Kubernetes does not deploy containers directly on the worker nodes. The containers are encapsulated into a Kubernetes object known as pods. A pod is a single instance of an application. A pod is a smallest object that you can create in Kubernetes.

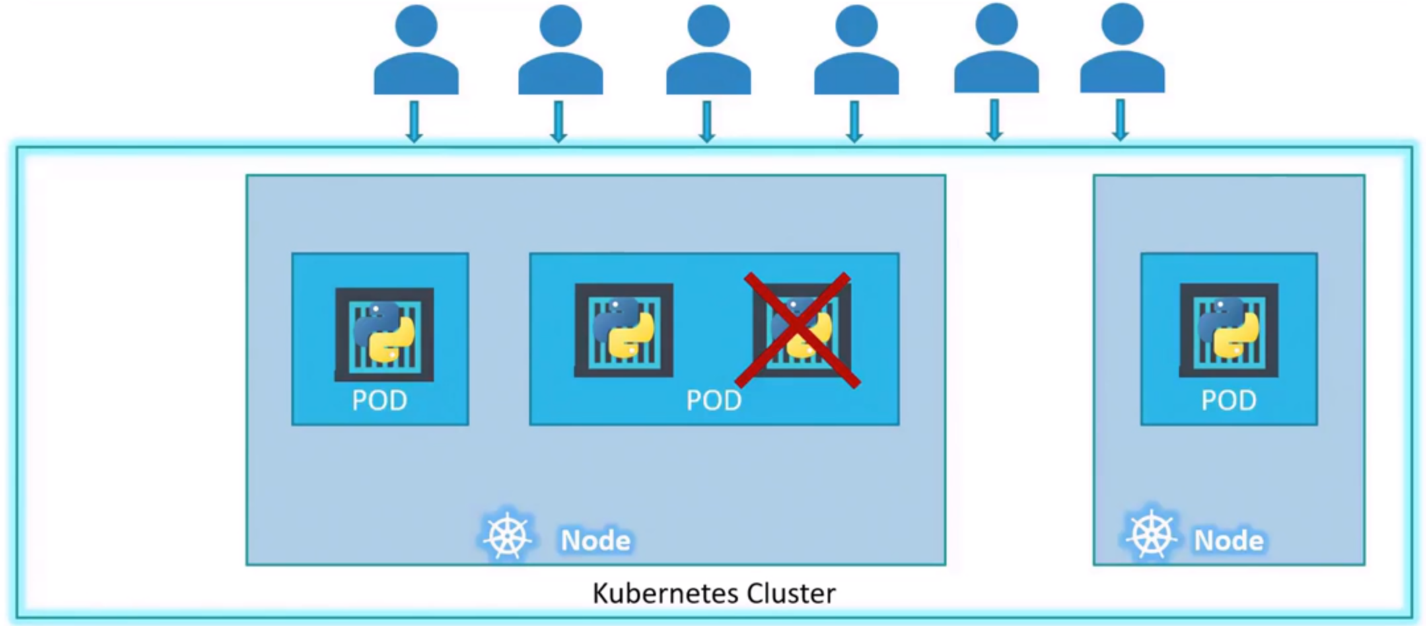
Below is the simple example where you have a single node Kubernetes Cluster with a single instance of your application running in a single Docker container encapsulated in a Pod. What if the number of users accessing your application increase and you need to scale your application?



You need to add additional instances of your web application to share the load. Where would you spin up additional instances? Do we bring up new container instances within the same pod?

No, we create new pod altogether with a new instance of the same application. What if the user base further increases and your current node has no sufficient capacity?

In that case you can always deploy additional pods on a new node in the cluster. You will have a new node added to the cluster to expand the cluster’s physical capacity. As per the image below, it can be seen that Pods usually have a 1 to 1 relationship with containers running your application. To scale up, you create new Pods and to scale down you delete existing pods. You do not add additional containers to an existing pod to scale your application.

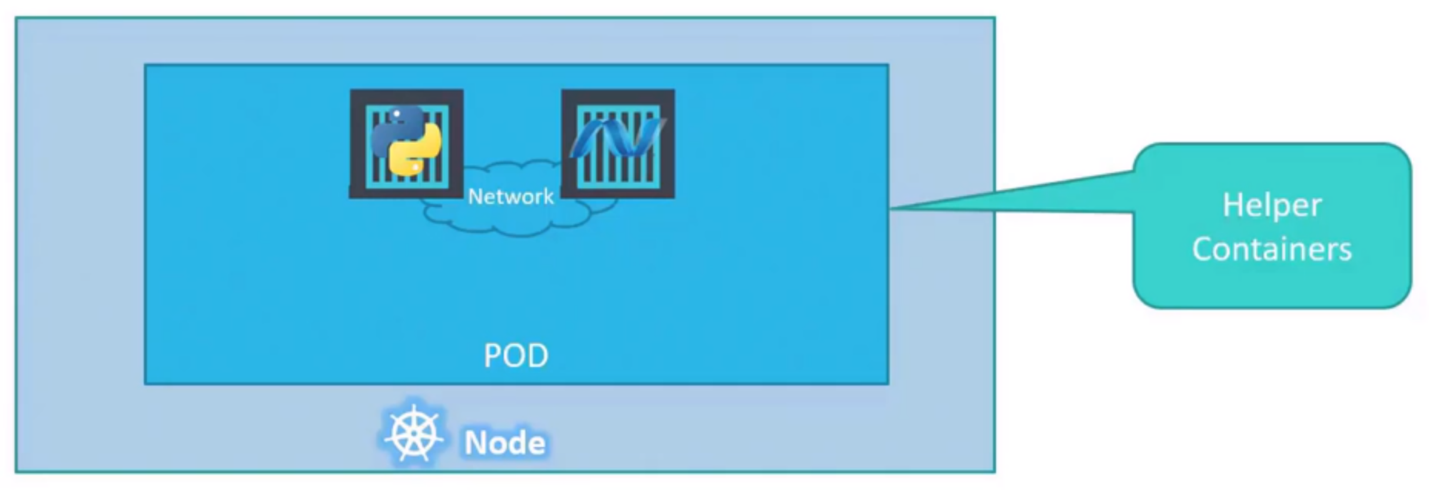


**Multi-Container Pods**

As we said that Pods usually have a 1 to 1 relationship with the Containers, but we are restricted to having a single container in a single pod?

No, a single Pod can have multiple containers except for the fact that they’re usually not multiple containers of the same kind. If our intention is to scale our application, then we would need to create additional pods. But sometimes you might have a scenario where you have a helper container that might be doing some kind of supporting task for our web application, such as processing a user entered data, processing a file uploaded by the user. And you want these helper containers to live alongside your application container.

In that case, you can have both of these containers as part of the same pod so that when a new application container is created, the helper is also created and when it dies, the helper also dies since they are part of the same pod. The two containers can communicate with each other directly by referring to each other as localhost since they share the same network space. They can also share the same storage as well.



**Let’s understand Pods better with the below example**

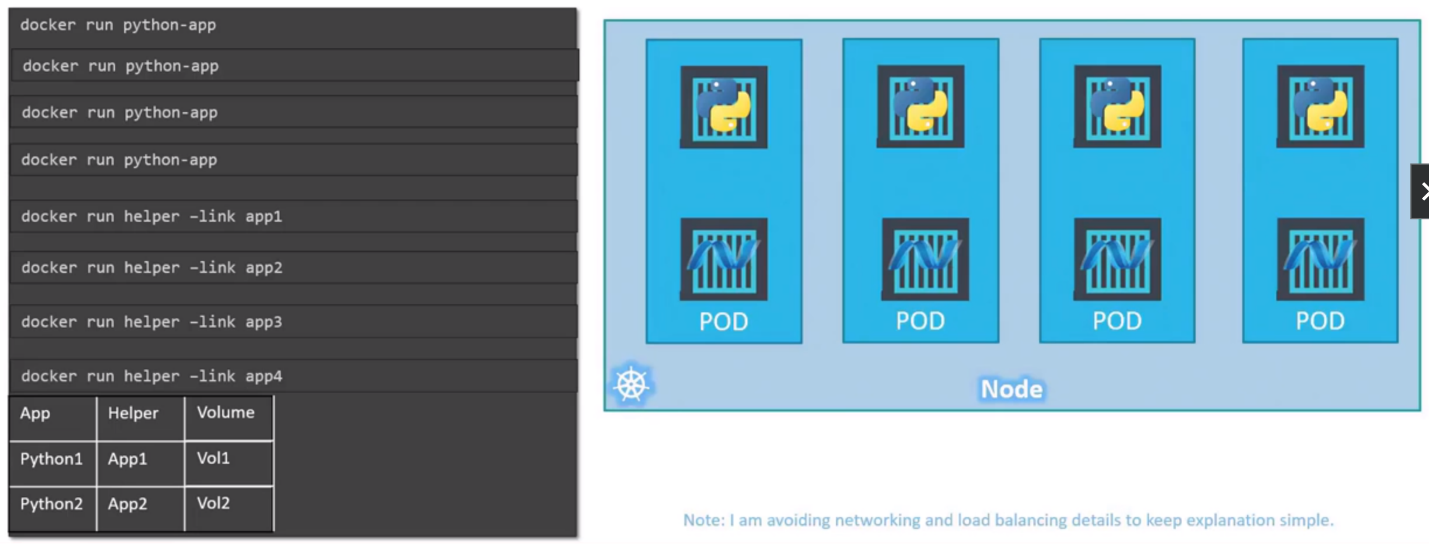
Let’s assume we were developing a process or a script to deploy our application on a Docker Host. Then we would first simply deploy our application using a simple Docker run python-app command, and the application runs fine and our users are able to access it. When the load increases, we deploy more instances of our application by running the docker run commands more times.

Sometimes in the future our application is further developed, undergoes architectural changes and grows and gets complex. We now have a new helper container that helps our web application by processing or fetching data from elsewhere. These helper containers maintain a 1 to 1 relationship with our application container and thus needs to communicate with the application containers directly and access data from those containers.

For this, we need to maintain a map of what app and helper containers are connected to each other. We need to establish network connectivity between these containers ourselves using links and custom networks. We would need to create sharable volumes and share it among the containers. We need to maintain the map of that as well. And most importantly, we would need to monitor the state of the application container and when it dies, manually kill the helper container as well as it’s no longer required.

When a new container is deployed, we would need to deploy the new helper container as well. With pods, Kubernetes does all of this for us automatically. We just need to define what containers a pod consists of and the containers in a pod by default will have access to the same storage, the same network namespace and same fate as they will be created together and destroyed together.

Even if our application didn’t happen to be so complex and we could live with a single container, Kubernetes still requires you to create pods, but this is good in the long run as your application is now equipped for architectural changes and scale in the future. However, also note that multi-containers pod are a rare use case and we’re going to stick to single container per pod.



**How to deploy Pods**

Kubectl command deploys a container by creating a pod. It first creates a Pod automatically and deploys an instance of the Nginx docker image.

**Kubectl run nginx**

But where does it get the application image from? For that you need to specify the image name using the image parameter.

**Kubectl run nginx --image nginx**

The nginx image is downloaded from the Docker Hub repository. Docker Hub is a public repository where latest Docker images of various applications are stored. You can configure Kubernetes to pull the image from the public Docker Hub or a private repository within the organization.

**How to see the list of Pods available?**

The Kubectl (Kube Control) gets pod command helps us see the list of pods in our cluster.

**Kubectl get pods**