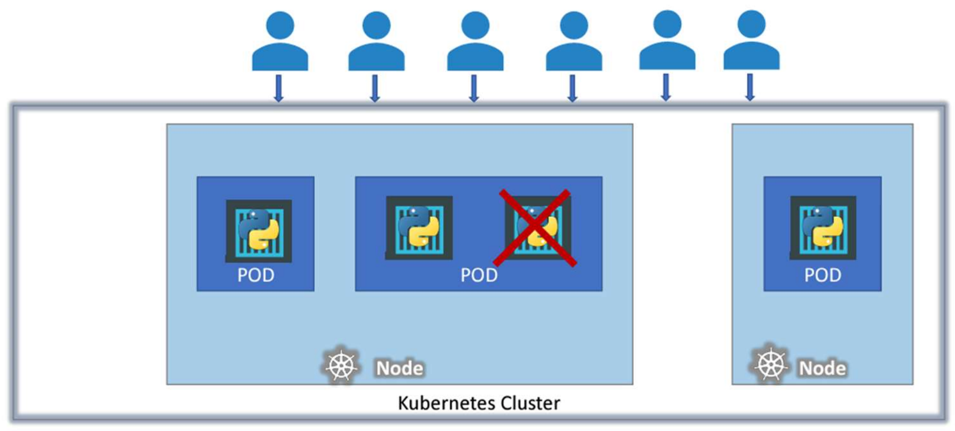
POD

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 the smallest object that you can create in Kubernetes.

Here we see the simplest of simplest cases 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 increases and you need to scale your application? You need to add additional instances of your web application to share the load. Now, where would you spin up additional instances? Do we bring up new container instance within the same pod? No, we create a new pod altogether with a new instance of the same application. As you can see, we now have two instances of our web application running on two separate pods on the same Kubernetes system or node.

What if the user base further increases and your current node has no sufficient capacity? Well, then 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.

POD

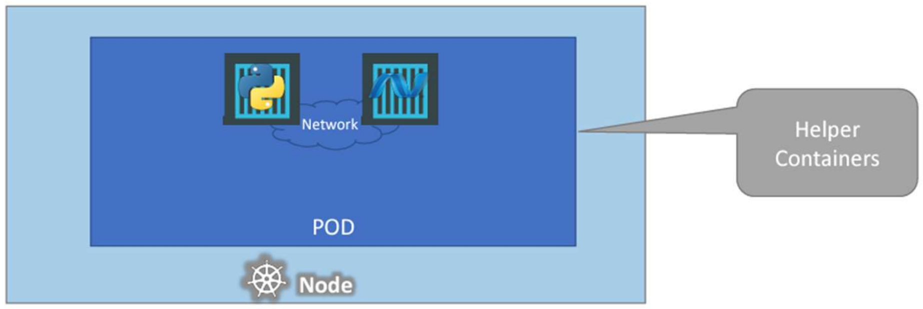


So what I'm trying to illustrate in this slide is that pods usually have a one-to-one relationship with containers running your application. To scale up, you create new pods, and to scale down, you delete existing pods. You do not add additional containers to an existing pod to scale your application.

We just said that pods usually have a one-to-one relationship with the containers, but are we 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.

As we discussed in the previous slide, if our intention was 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 and their data, processing a file uploaded by the user, etc., and you want these helper containers to live alongside your application container. In that case, you can have both of these containers 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're part of the same pod. The two containers can also communicate with each other directly by referring to each other as localhost since they share the same network space, plus they can easily share the same storage space as well.

Multi-Container PODs



Let's us now look at how to deploy pods. Earlier we learned about the kubectl run command. What this command really does is it deploys a Docker container by creating a pod, so it first creates a pod automatically and deploys an instance of the NGINX Docker image, but where does it get the application image from? For that, you need to specify the image name using the dash dash image parameter. The application image, in this case, the NGINX image, is downloaded from the Docker Hub repository. Docker Hub, as we discussed, is a public repository where latest Docker images of various applications are stored. You could configure Kubernetes to pull the image from the public Docker Hub or a private repository within the organization. Now that we have a pod created,

How do we see the list of pods available? The kubectl get pods command helps us see the list of pods in our cluster. In this case, we see the pod is in a container creating state and soon changes to a running state when it is actually running. Also, remember that we haven't really talked about the concepts on how a user can access the NGINX web server, and so in the current state, we haven't made the web server accessible to external users. You can, however, access it internally from the node. For now, we will just see how to deploy a pod, and in a later lecture, once we learn about networking and services, we will get to know how to make this service accessible to end users.

kubectl

```
• kubectl run nginx --image nginx
```

```
kubectl get pods
NAME     READY    STATUS              RESTARTS    AGE
nginx    0/1      ContainerCreating   0           6s
```

```
NAME     READY    STATUS     RESTARTS    AGE
nginx    1/1      Running    0           34s
```