Deployments in Kubernetes

For better understand Deployments first of all we need to know about pods.

* What is pod?

A single smallest unit in Kubernetes. A pod is single instance of an application. A pod is the smallest object that you can create in Kubernetes.

Before we head into understanding pods, we would like to assume that the following have been set up already.

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 setup. Doesn't matter.

All the services need to be in a running state. with Kubernetes, our ultimate aim 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.

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.

If you face difficulty to understand what is pod.

We could take another shot at understanding pods from a different angle. Let's, for a moment, keep Kubernetes out of our discussion and talk about simple Docker containers.

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 now have a new helper container that helps our web application by processing or fetching data from elsewhere. These helper containers maintain a one-to-one relationship with our application container and thus needs to communicate 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 would need to establish network connectivity between these containers ourselves using links and custom networks. We would need to create shareable volumes and share it among the containers. We would need to maintain a 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.

Similarly, 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 fit as in they will be created together and destroyed together.

Now we will learn about creating a pod using a YAML based configuration file.

Kubernetes uses YAML files as inputs for the creation of objects such as pods, replicas, deployments, services, et cetera. All of these follow similar structure.

A Kubernetes definition file always contains four top level fields; the

API version, kind, metadata, and spec.

apiVersion: v1

kind: Pod

metadata:

name: my-app

labels:

app: my-app

type: front-end

spec:

containers:

* Name: nginx

Image: nginx:latest

apiVersion: This is the version of the Kubernetes API we are using to create the object. Depending on what we are trying to create, we must use the right API version.

For now, since we are working on pods, we will set the API version as V1.

Few other possible values for this field are apps/V1beta, extensions/V1beta, etc.

Kind: The kind refers to the type of object we are trying to create, which in this case happens to be a pod, so we will set it as pod. Some other possible values here could be replica sets or deployment or service.

Metadata: The metadata is data above the object like its name, labels, etc.

this is in the form of a dictionary, so everything under metadata is intended to the right a little bit and so names and labels are children of metadata.

Spec: Spec is a dictionary so add a property under it called containers. Containers is a list or an array. The reason this property is a list is because the pods can have multiple containers within them.

For to run this pod on Kubernetes we use command

Kubectl apply -f pod.yaml



* Replica Sets
* What is replica and why we need it.

Replica is most important part of our deployment for example.

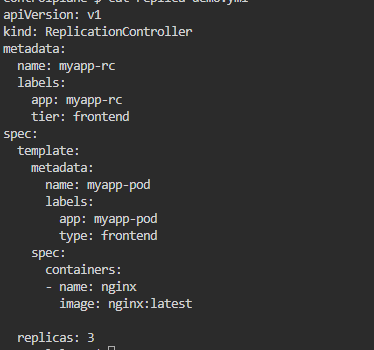
What if for some reason our application crashes and the pod fail?

Users will no longer be able to access our application. To prevent users from losing access to our application, we would like to have more than one instance or pod

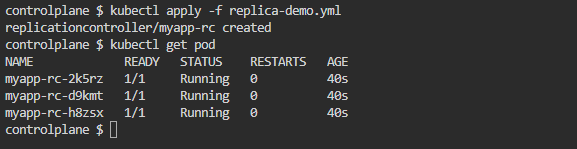
running at the same time. That’s way, if one fails, we still have our application running on the other one. The Replication Controller helps us run multiple instances of a single pod in the Kubernetes cluster, thus providing high availability. So, does that mean you can't use a Replication Controller if you plan to have a single pod?

No. Even if you have a single pod, the Replication Controller can help by automatically bringing up a new pod when the existing one fails. Thus, the Replication Controller ensures that the specified number of pods are running at all times even if it's just one or 100.

With the help of replica we can also share the load on our application.



This yaml file will create 3 desire pod for our application.



Now we are able to learn about Deployments.

we will discuss about Kubernetes deployment. For a minute, let us forget about pods, and Replica Sets, and other Kubernetes concepts, and talk about how you might want to deploy your application in a production environment.

Suppose there are 4 or more pods are running in environment of your application

And you want upgrade your application with the help of rolling update.

One by one you update your pod and deploy it and if the after upgrade your application not working properly then you can downgrade it one by one wise versa.

All of these capabilities are available with the Kubernetes deployments.