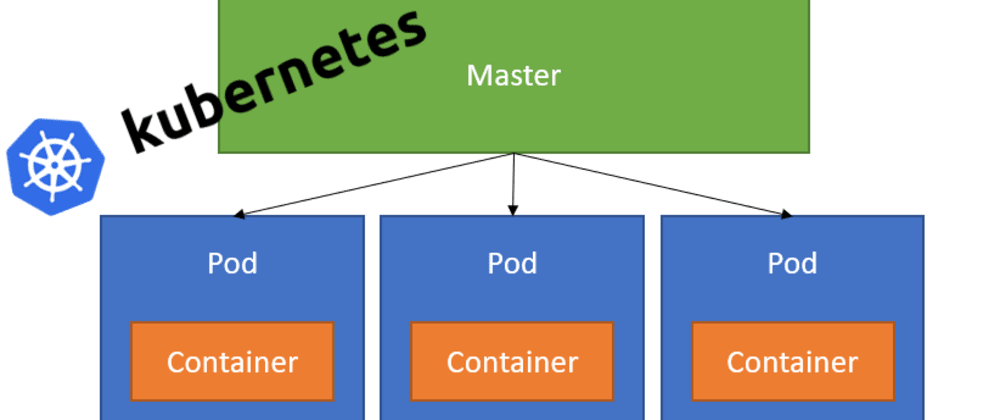
**What are Pods in Kubernetes?**



* What are Pods?
* What’s the difference between single container and multi-container Pods?
* How can we create Pods in Kubernetes?
* How can we deploy and interact with our Pods using kubectl?
* How can we ensure that our Pods in Kubernetes are healthy?

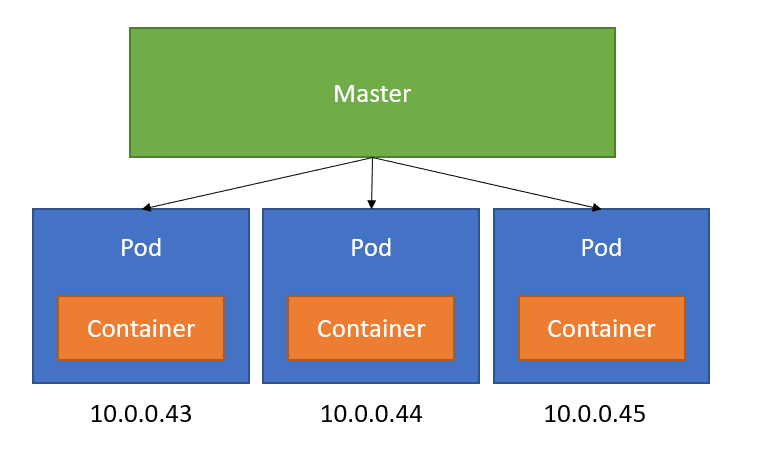
**What are Pods?**

Pods are Kubernetes Objects that are the basic unit for running our containers inside our Kubernetes cluster. In fact, Pods are the smallest object of the Kubernetes Model.

Kubernetes uses pods to run an instance of our application and a single pod represents a single instance of that application. We can scale out our application horizontally by adding more Pod replicas.

Pods can contain a single or multiple containers as a group, that share the same resources within that Pod (storage, network resources, namespaces). Pods typically have a 1–1 mapping with containers, but in more advanced situations, we may run multiple containers in a Pod.

Pods are epheremeral resources, meaning that Pods can be terminated at any point and then restarted on another node within our Kubernetes cluster. They live and die, but Pods will never come back to life.



Pod containers will share the name network namespace and interface. Container processes need to bind to different ports within a Pod and ports can be reused by containers in separate containers. Pods do not span nodes within our Kubernetes cluster.

**What’s the difference between single container and multi-container Pods?**

Like I mentioned before, Pods can contain contain either a single or multiple containers.

Running a **single container in a Pod** is a common use case. Here, the Pod acts as a wrapper around the single container and Kubernetes manages the Pods rather than the containers directly.

We can also run **multiple containers in a Pod**. Here, the Pod wraps around an application that’s composed of multiple containers and share resources.

If we need to run multiple containers within a single Pod, it’s recommended that we only do this in cases where the containers are tightly coupled.

**How can we create Pods in Kubernetes?**

We can define Pods in Kubernetes using YAML files. Using these YAML files, we can create objects that interact with the Kubernetes API (Pods, Namespace, Deployments etc.). Under the hood, kubectl converts the information that we have defined in our YAML file to JSON, that makes the request to the Kubernetes API.

I’m a fan of YAML, it’s easy to understand what’s going on and thanks to extensions in tools like Visual Studio Code, they’re easy to create and manage.

I get that indentation can be a pain. I use the [YAML code extension](https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml) that Red Hat have developed in Visual Studio code to help me write my YAML files.

Let’s take a look at an example YAML definition for a Kubernetes Pod:

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx-2  
 labels:  
 name: nginx-2  
 env: production  
spec:  
 containers:  
 - name: nginx  
 image: nginx

Let’s break this down a bit. To create Kubernetes objects using YAML, we need to set values for the following fields.

**apiVersion** — This defines the Kubernetes API version that we want to use in this YAML file.

**kind** — This defines what kind of Kubernetes object we want to create.

**metadata** — This is data that helps us uniquely identify the object that we want to create. Here we can provide a name for our app, as well as apply labels to our object.

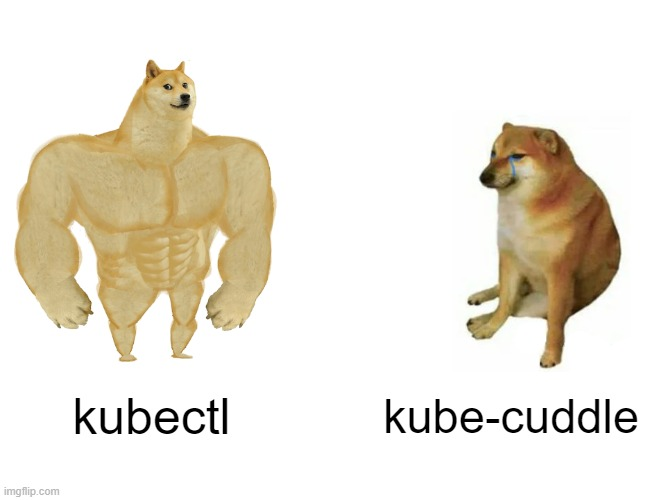
**spec** — This defines the state that we want or our object. The format that we use for spec. For our Pod file, we have provided information about the containers that we want to host on our Pod.

**How can we deploy and interact with our Pods using kubectl?**

There are a few way that we can use kubectl to deploy and interact with our Pods!

*Wait, what is kubectl?*

kubectl, (kube-control, or as some people call it, kube-cuddle) is the Kubernetes command-line tool. It allows us to run commands against Kubernetes clusters.



With kubectl, we can create a Pod using our YAML definition file like so:

kubectl apply -f mypod.yaml

We can list all of our Pods like so:

kubectl get pods

We can expose a container port externally using kubectl. Remember by default, Pods and Containers are only accessible within the Kubernetes Cluster. Using Kubectl, we can run the following command:

kubectl port-forward mypod 8080:80

We can also delete the pod. We can do this by deleting the pod directly like so:

kubectl delete pod mypod

This will cause the Pod to be destroyed and created. We can also delete the Deployment that manages the Pod (I’ll talk about Kubernetes Deployments in a future post) like so:

kubectl delete deployment mydeployment

**How can we ensure that our Pods in Kubernetes are healthy?**

Kubernetes relies on Probes to determine whether or not a Pod is healthy. Probes are diagnostic operations that are performed periodically by the Kubelet on containers.

There are three types of Probes:

**Liveness Probes** — These are used to determine if a Pod is healthy and running as expected. If the liveness probe fails, kubelet will kill the container and the container will then restart according to its defined policy (We can define these as either Always, OnFailure and Never)

* **Readiness Probes** — These are used to determine whether or not a pod should receive requests. If the probe fails, the endpoints controller removes the IP address of the Pod from the endpoints of all Services that match the Pod.
* **Startup Probe** — This is used to determine whether the container has started. All other probes are disabled if a startup probe is provided until it succeeds.

To perform diagnostic checks, Kubelet will call a Handler that has been implemented by the container. These types of handlers are:

* **ExecAction** — This executes an action inside the container
* **TCPSocketAction** — TCP checks are performed against the containers IP addreess on a specified port.
* **HTTPGetAction** — HTTP Get requests are performed against a Pods IP address on a specified port and path.

These probes can result in either a *Success*, *Failure* or *Unknown* result.

**Conclusion**

Hopefully this document has helped you understand the basics of Pods in Kubernetes!