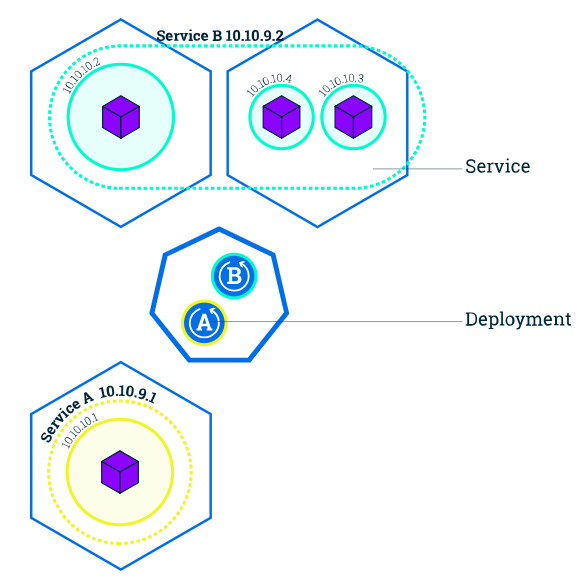
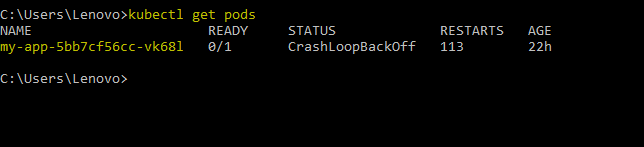
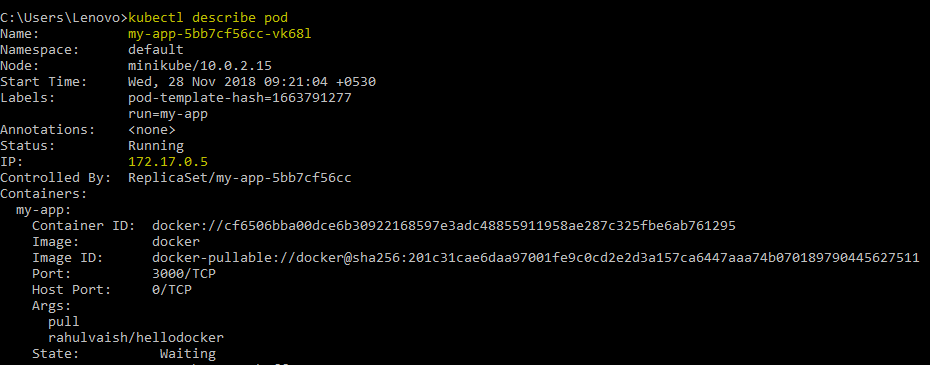
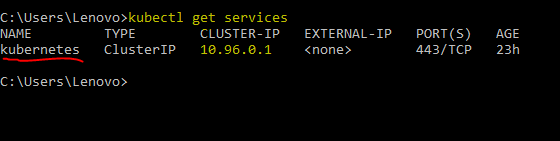
In the previous tutorial we have seen that **Pods have their own unique IP across the cluster, those IP’s are not exposed outside Kubernetes**. Taking this into account that over time *Pods may be terminated*, *deleted* or *replaced* by other Pods, we need a way to let other Pods and applications automatically discover each other. Kubernetes addresses this by **grouping Pods in Services**. A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

This abstraction will allow us to expose Pods to traffic originating from outside the cluster. **Services have their own unique cluster-private IP address and expose a port to receive traffic.**









We can see a listed service called kubernetes that is created by default when minikube starts the cluster.

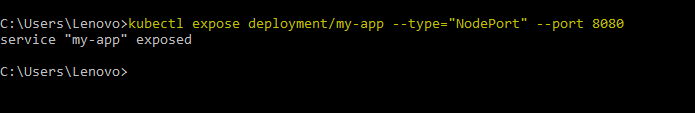
So, basically the above pod is virtually inside this service called kubernetes.

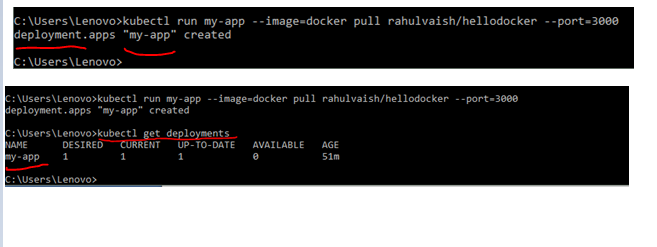
**Service IP = 10.96.0.1**

**Pod IP = 172.17.0.5 (unexposed)**

If you choose to expose the service outside the cluster, we have two options:

* **LoadBalancer** - provides a public IP address (what you would typically use when you run Kubernetes on GKE or AWS. minikube does not support the LoadBalancer option yet).
* **NodePort** - exposes the Service on the same port on each Node of the cluster using NAT (available on all Kubernetes clusters, and in Minikube)Let’s see how to do this-



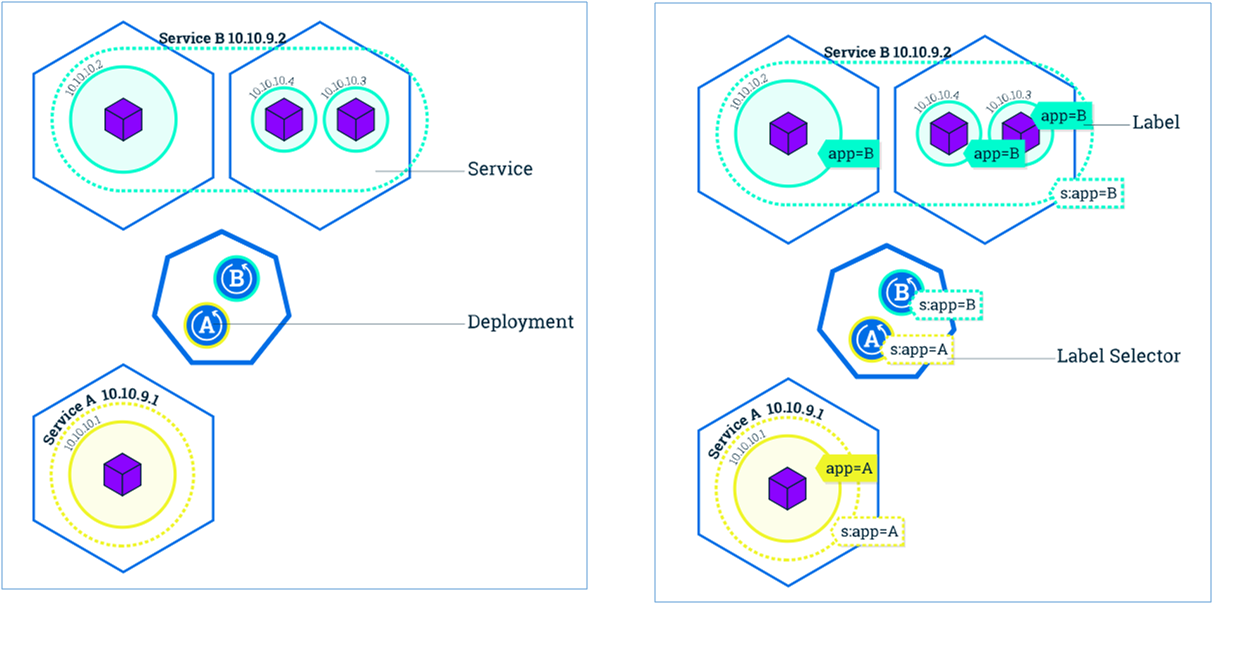
**NOTE: Hope you remember this deployment step from Chapter2-3. **

**Now the IMPORTANT POINT to note is that our containerized application name was hellodocker. When we deployed it on Kubernetes, we gave the name of its running instance as my-app. This my-app resides inside a POD which is my-app-5bb7cf56cc-vk68l. To expose service we will rhave to refer the name deployment/my-app**

A Service provides load balancing of traffic across the contained set of Pods. This is useful when a service is created to group all Pods from a specific Deployment (our application will make use of this in the next module, when we’ll have multiple instances running).

Services are also responsible for service-discovery within the cluster (covered in Module 6). This will for example allow a frontend service (like a webserver) to receive traffic from a backend service (like a database) without worrying about Pods.

Services match a set of Pods using Label Selectors, a grouping primitive that allows logical operation on Labels.

Labels can be attached to objects at the creation time or later and can be modified at any time. The kubectl run command sets some default Labels/Label Selectors on the new Pods/ Deployment. The link between Labels and Label Selectors defines the relationship between the Deployment and the Pods it creates.

Let’s expose now our application with the help of a Service, and apply some new Labels.