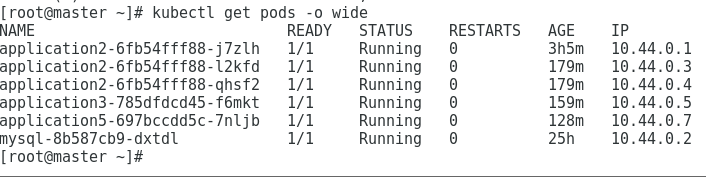
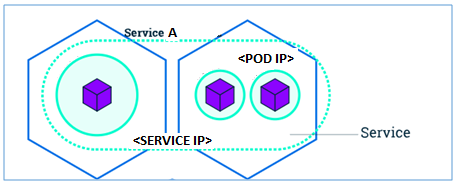
We have seen that **Pods have their own unique IP across the cluster, those IP’s are not exposed outside Kubernetes**.

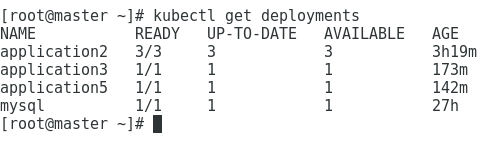


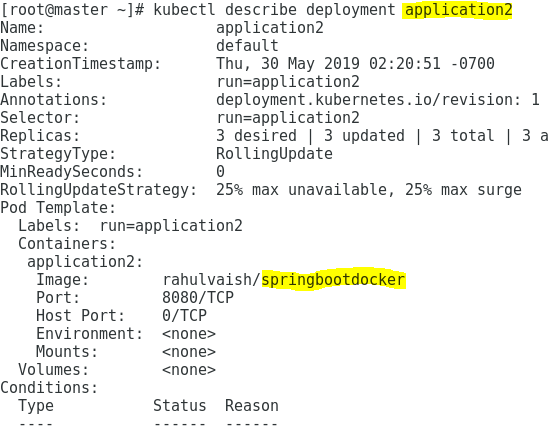
Taking this into account that over time *Pods may be terminated*, *deleted* or *replaced* by other Pods, we need a way to let other Pods and applications automatically discover each other. Kubernetes addresses this by **grouping Pods in Services**. A Kubernetes Service is an abstraction layer which defines 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. In other words, exposing a deployment makes a service!**

**Example:**

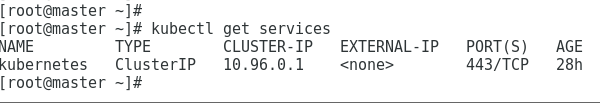




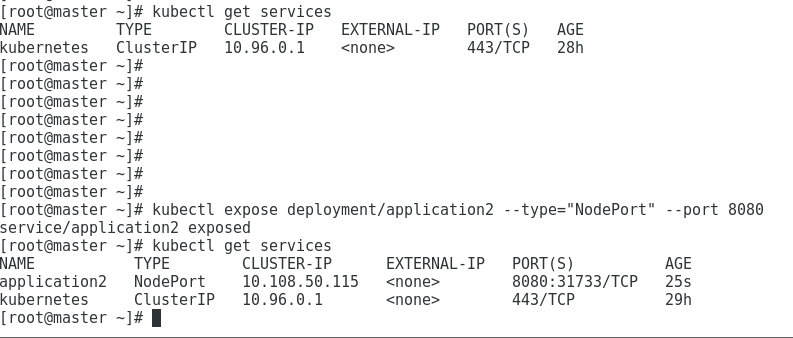


So we can see that deployment **application2** is **springbootdocker** image. Now we need to expose this deployment as service.

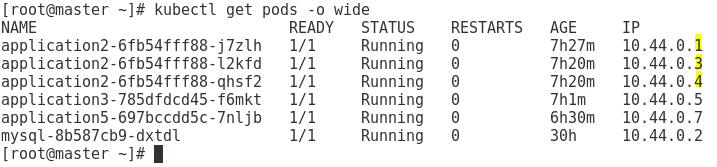
There is a default service running on the cluster, named as **kubernetes**.



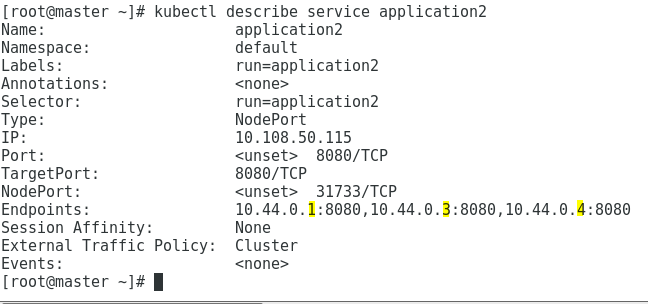
Now, we have exposed our deployment as service:



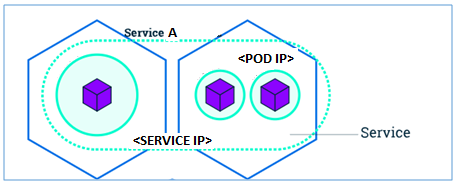
For application2 deployment we have 3 pods running:



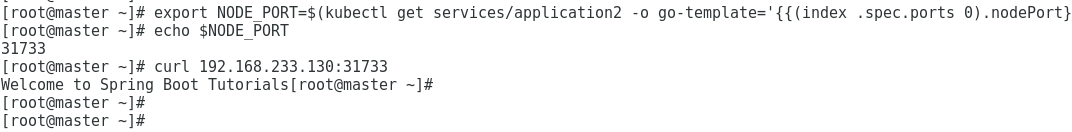
Now when we expose the deployment, it becomes a service. The service has all the pods registered. We can confirm the same by the below command:

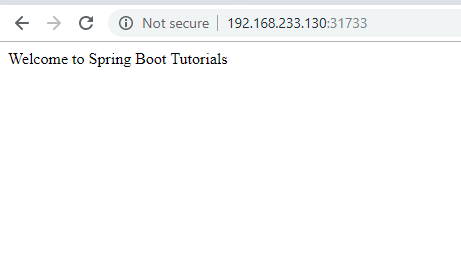


**Example: The below example is justified!**



Now, let’s try to export the service outside the cluster:



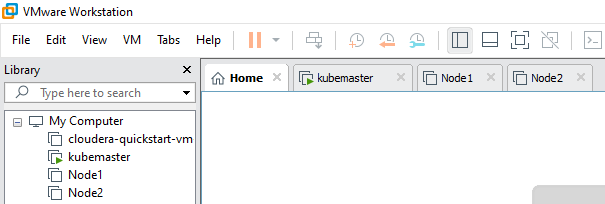


**Important Note:** To access the application we used the <IP of the Node>:<Node Port>. To determine our pod is running on which Node, we can use our familiar command: **kubectl get pods –o wide**

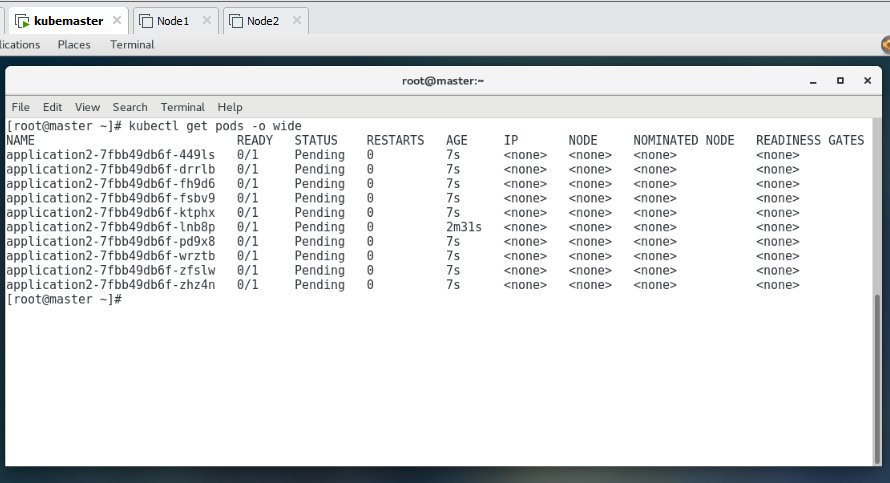
**But, what if the Node goes down?**

**Yes, you are correct; you won’t be able to access the application and this is ‘super dangerous’!!**

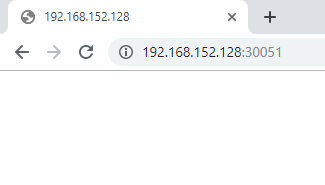
**To experiment:**  I deployed, exposed and scaled (as 10 pods) a simple SpringBoot Application (on master). Node1 & Node2 are stopped.



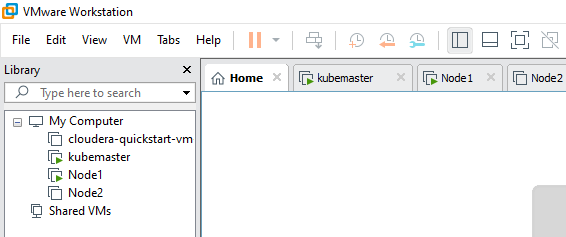
The 10 replicas are in pending state. Master is waiting for slaves to join.



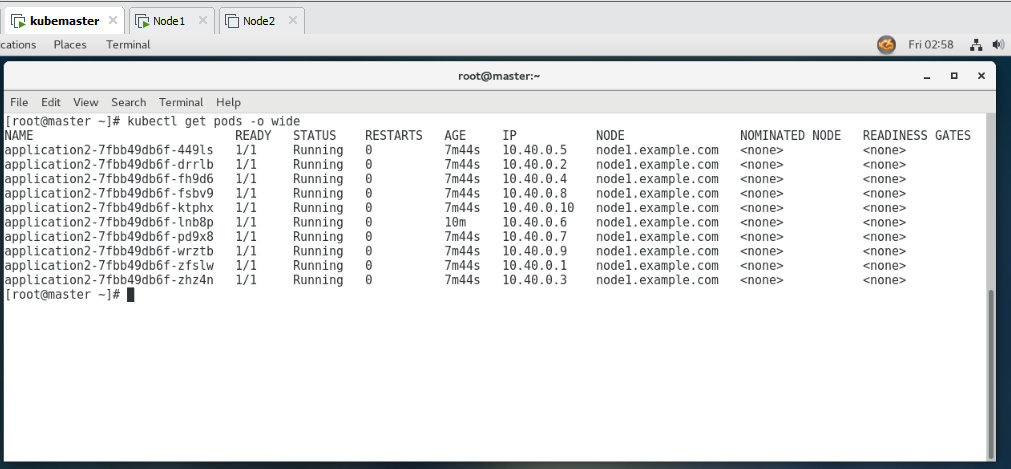
This means that: The service is not accessable from outside world. Here, we are using <IP of Master>:<Node Port>



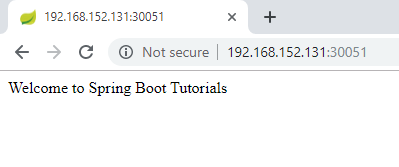
Now started node 1



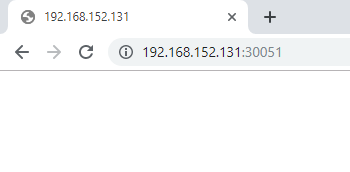
Now the pods will be created on Slave Node:

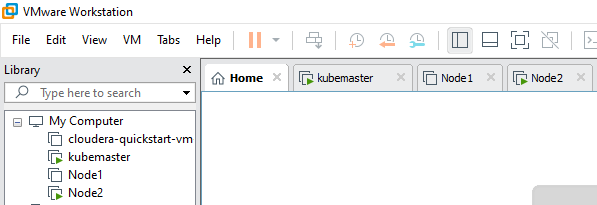


As expected, the pod replicas went on Node1 (above). Now, to access the application, I issued <IP of Node1>:<NodePort> . It was a success.

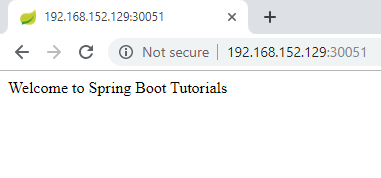


To test, I then stopped the Node1, and as thought <IP of Node1>:<Node Port> resulted failure.

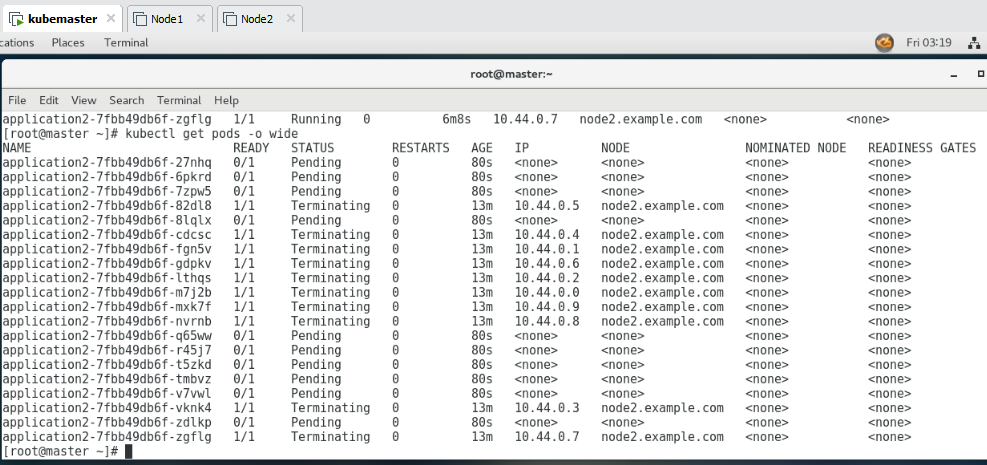




After some time, I started the Node2 (above), and as expected the replicas drifted on Node2. And, our application was accessible via <IP of Node2>:<NodePort>.



**The state of master on node closure**: As we have 10 pods (replicas) and on node closure, all 10 will be in terminating state while new 10 pods will be in pending state to get created – what they are waiting for is a slave node to get activated.



**NOTE:** When a both Nodes are Up & Running, you can access your application/Service using any <NodeIP>:<NodePort>. The reason lies behind coredns.

