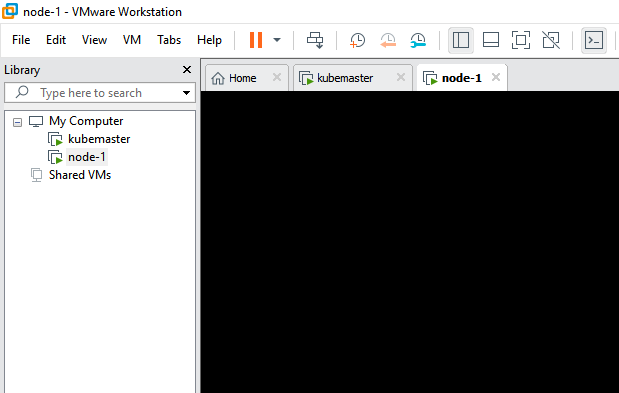
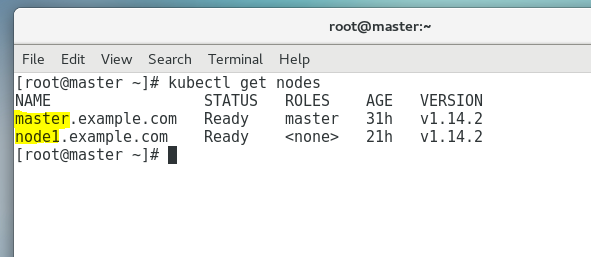
We have 2 node Kubernetes cluster: Node means either a Physical Computer or VM.

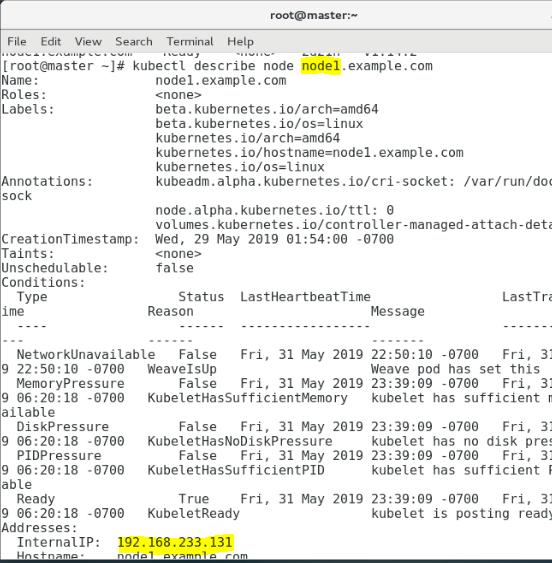
In our case, it is VM. One VM is considered as Master Node and other VM as Worker Node:







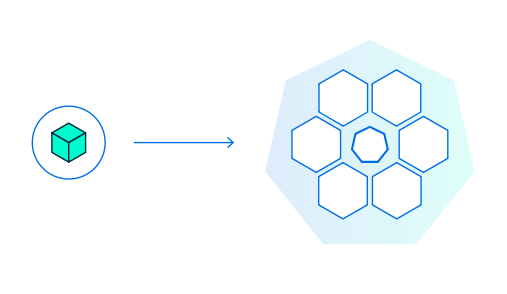
This IP is the master Node’s IP.



This IP is the Worker Node’s IP.

Now let’s deploy a Docker image:

Deploying an app from crow-eye view means this:



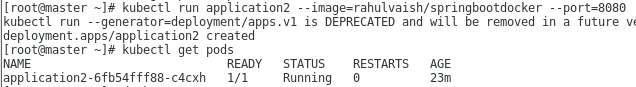
When we deploy a containerized application on Kubernetes, Master node’s component **scheduler** checks resource availability in the cluster i.e, which node (Master or which Worker Node) will process the deployment is decided.

Master Node will just manage things in the entire cluster, but is self not used for application deployment purpose (In normal scenario, but it is not that the deployment can’t/won’t happen on master).

Once the node is selected , a pod is created and containerized app is started as container (running instances of the containerized app is called container).

FMR: You can also consider Node as RAM, POD as Heap memory and Container as Object.

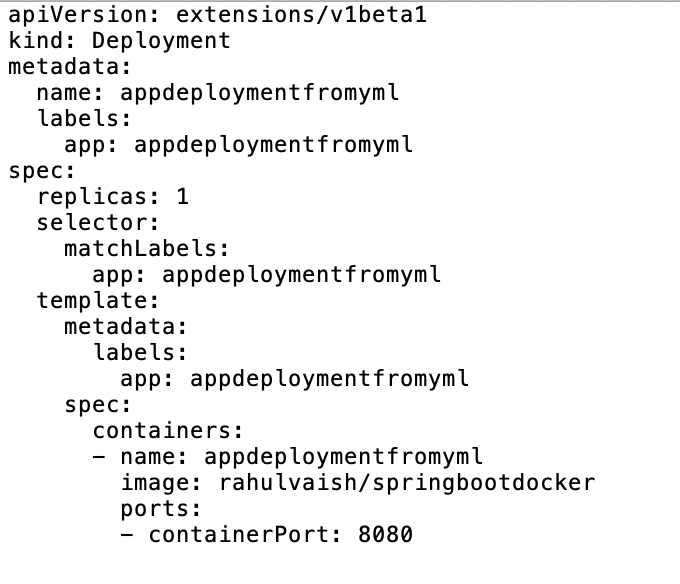
Let’s see how to deploy a containerized app:



Another way to deploy containerized application is through YML:

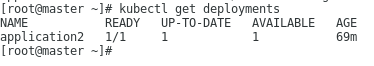
**STEP#1**: Create the YML File by executing vi **appdeploymentfromyml.yml**

**STEP#2:** Prepare the yml File:



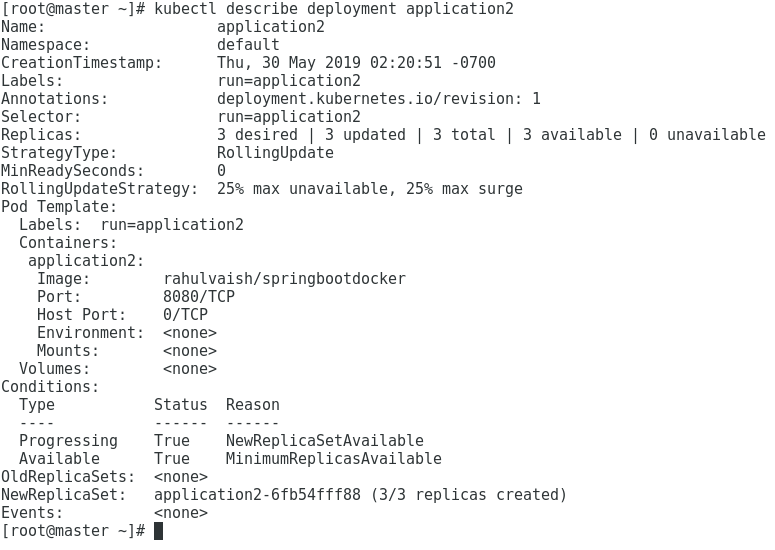
**STEP#3:** If we execute: **kubectl create -f applicationdeploymentfromyml.yml** the deployment happens.

In this tutorial, we are choosing simple execution (not YML way), for deployment.



We can also list our deployments (above). This lists all the deployments happened within the ‘**default’** namespace.

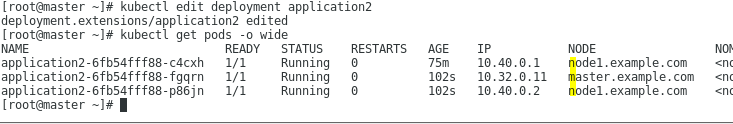
To investigate further about our deployment:



To determine, in which node this deployment happened.

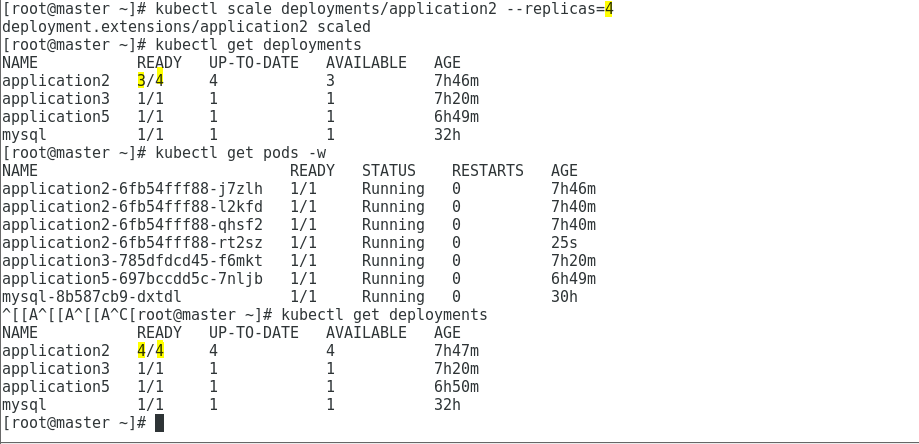


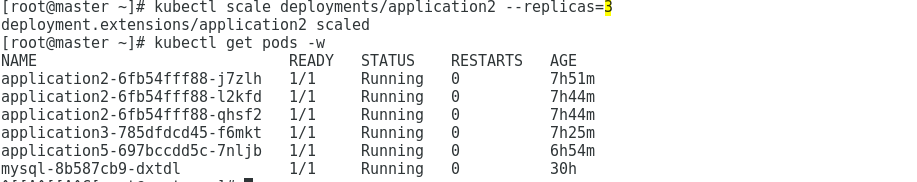
In case of minikube we just had one node; therefore, deployment happened only on the same node. But here we have setup 1 master and 1 worker node, therefore the scenario is different. As we know and saw (above) that by default one deployment made one pod . We can also create replicas of pods*-* ***We need to edit the deployment, increase/decrease the number of replicas (under spec).*** Here, we have increased the number of replicas to 3. Once done, we can see the number of pods increased:



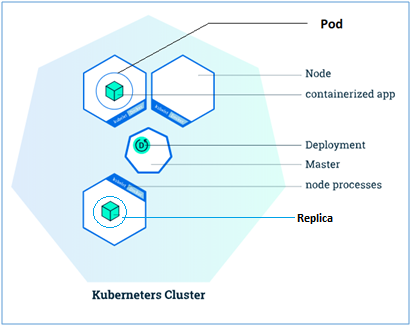
**NOTICE: Out of 3, 1 pod is spinning on master. This behavior is ‘by default’ is unpredictable!**

Another way to scale the application is:

****

****

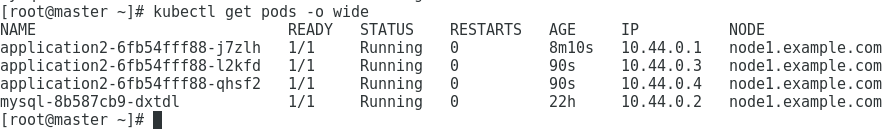
The replication process above can be *roughly* represented diagrammatically as:



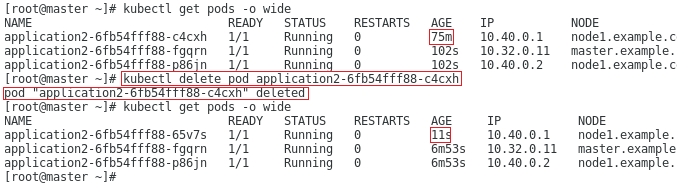
P.S. The replicated pods can be establish anywhere in the cluster. The decision is taken by the scheduler (component of Master Node)

**Information**: In every node (including Master) – ***kubelet*** runs. It’s a daemon service that is used to establish communication between all the nodes. Plus, in every node, Docker runs as pods contain containers which will require Docker as platform.

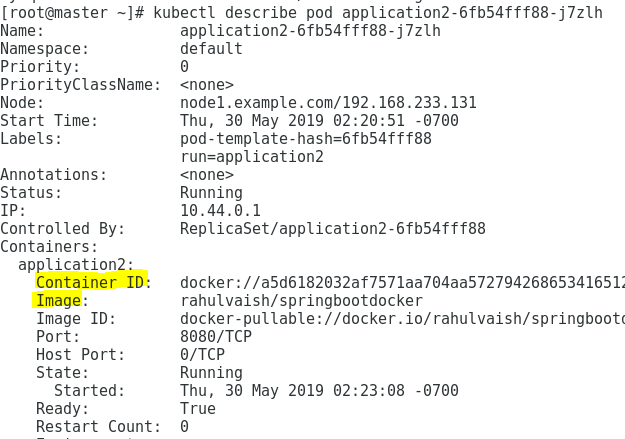
**I also re-tried creating everything from scratch: This time everything got deployed on worker node. This concludes that the deployment happens on random basis (controlled by scheduler).**

****

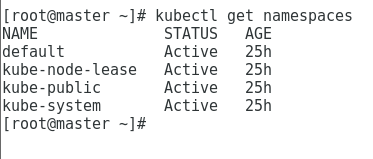
**NOTE: Also, if you delete any pod, it gets recreated automatically – The new one starts forming and old one starts terminating in parallel.**

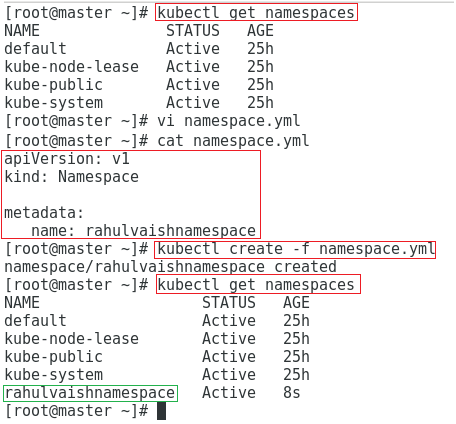
****

On describing a pod we can also see the container id within.

****

**Understanding Namespaces:**

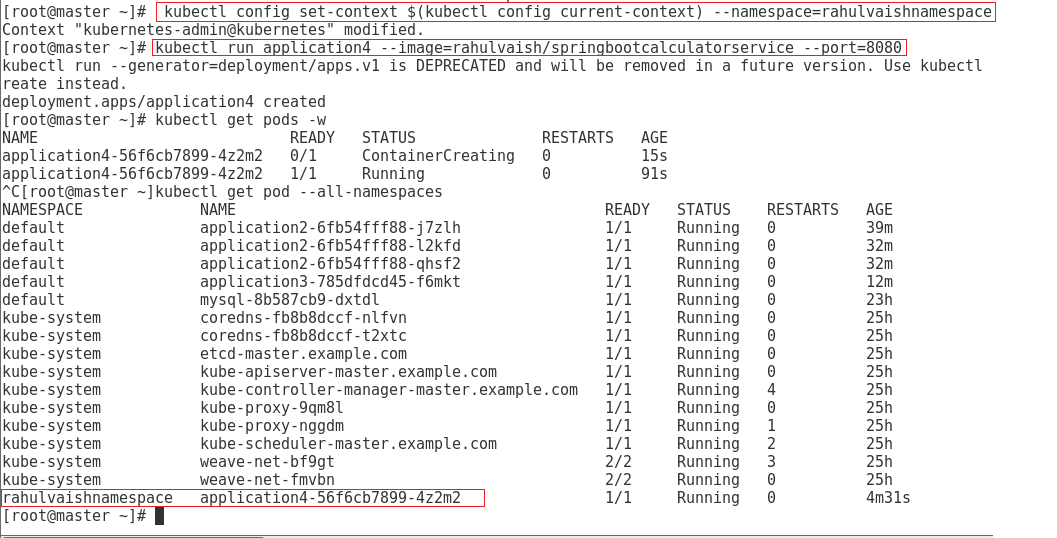
****

****

**To determine which pod is running on which namespace:**

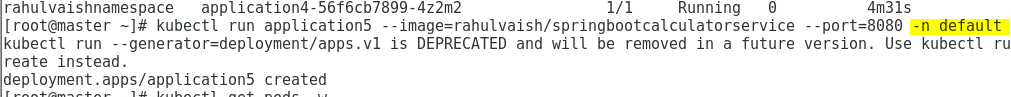
****

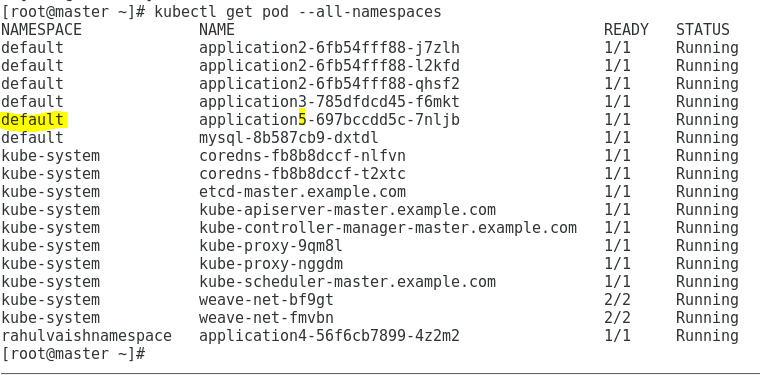
To set namespace:



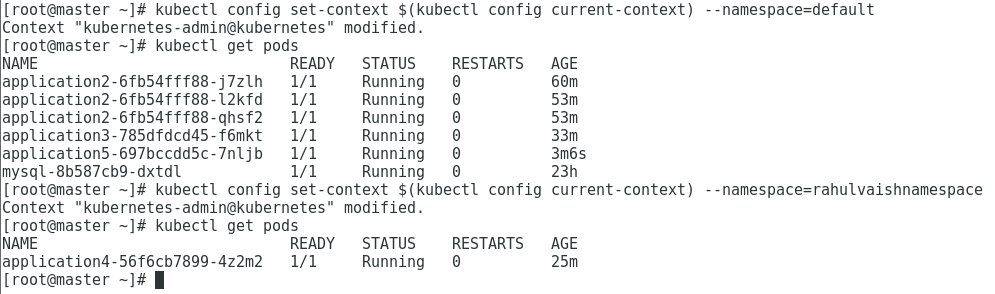
We can see that after setting the namespace to rahulvaishnamespace, the new deployment dropped into rahulvaishnamespace.

To make deployment on specific namespace we can do likewise:





To switch between namespaces:



To check the current Namespace, we can go to .kube folder and view the config file:

