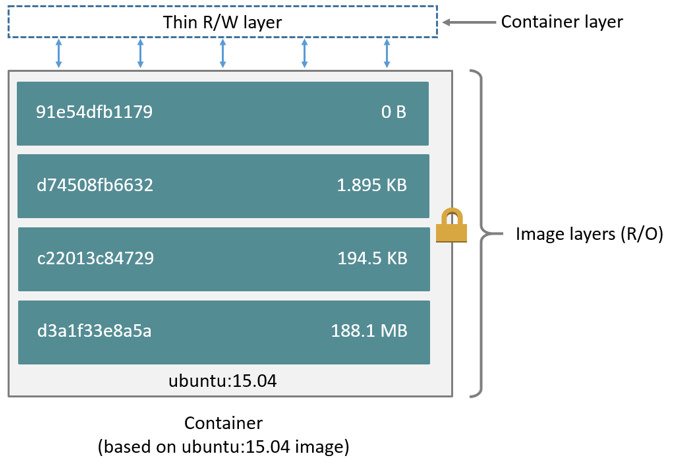
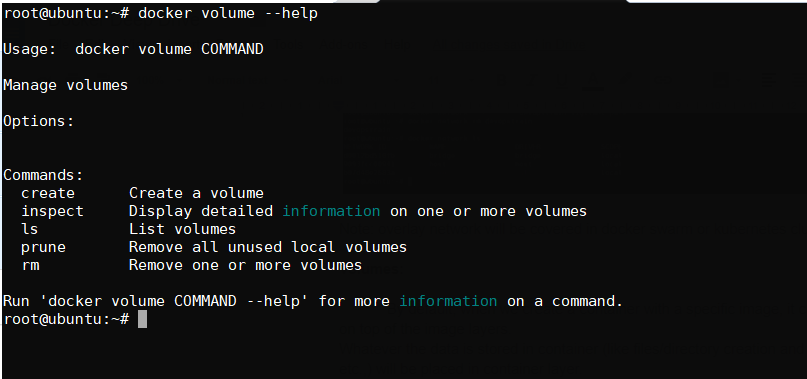
**Docker Volumes**

**Volumes:**

By default, when we create a container with a specific image, it creates a thick layer on top of the image layers.

Whatever the data is stored in a container (like files/directory creation and installing packages etc.,) will be placed in container layer.





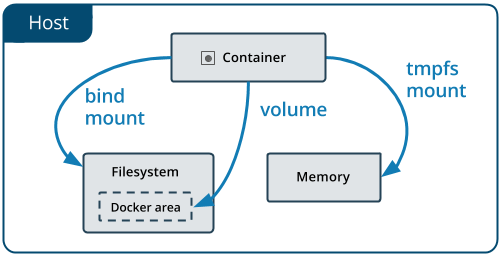
To write the data in to the container, we need to have storage driver configured to manage the file system. The storage driver provides a union file system, using the Linux kernel.

There are two ways to store the data in the host machine.

**Persistent storage and Ephemeral storage**

In persistent storage, data will remain even after the container was removed.

In ephemeral storage, data will be lost once the container is terminated



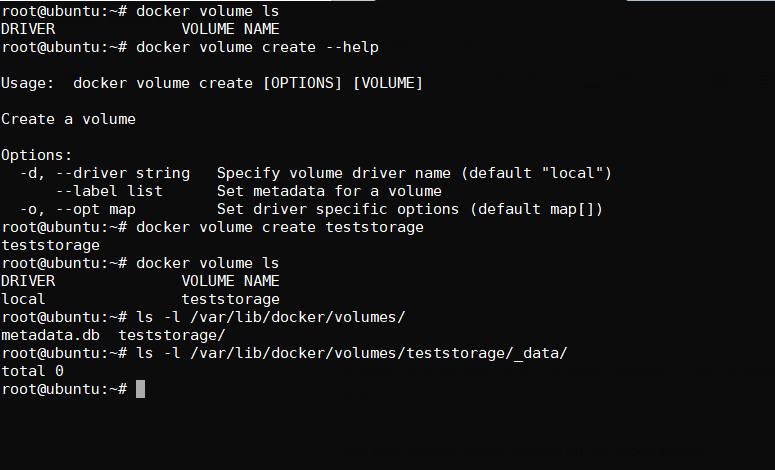
* **Volumes** are stored in a part of the host filesystem which is managed by Docker(/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

A given volume can be mounted into multiple containers simultaneously. When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using **docker volume prune.**

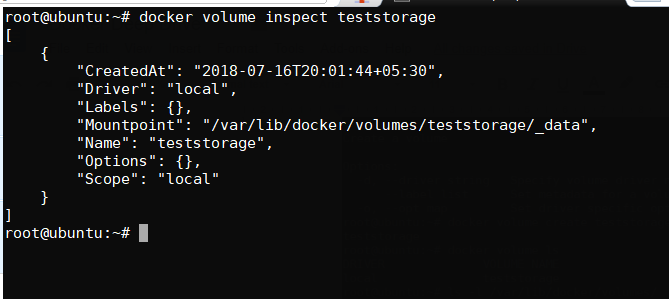
* **Bind mounts** may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
* **tmpfs mounts** are stored in the host system’s memory (RAM) only, and are never written to the host system’s filesystem.

Bind mounts and volumes can both mounted into containers using the -v or --volume flag, but the syntax for each is slightly different. For tmpfs mounts, you can use the --tmpfs flag. However, in Docker 17.06 and higher, we recommend using the --mount flag for both containers and services, for bind mounts, volumes, or tmpfs mounts, as the syntax is more clear.

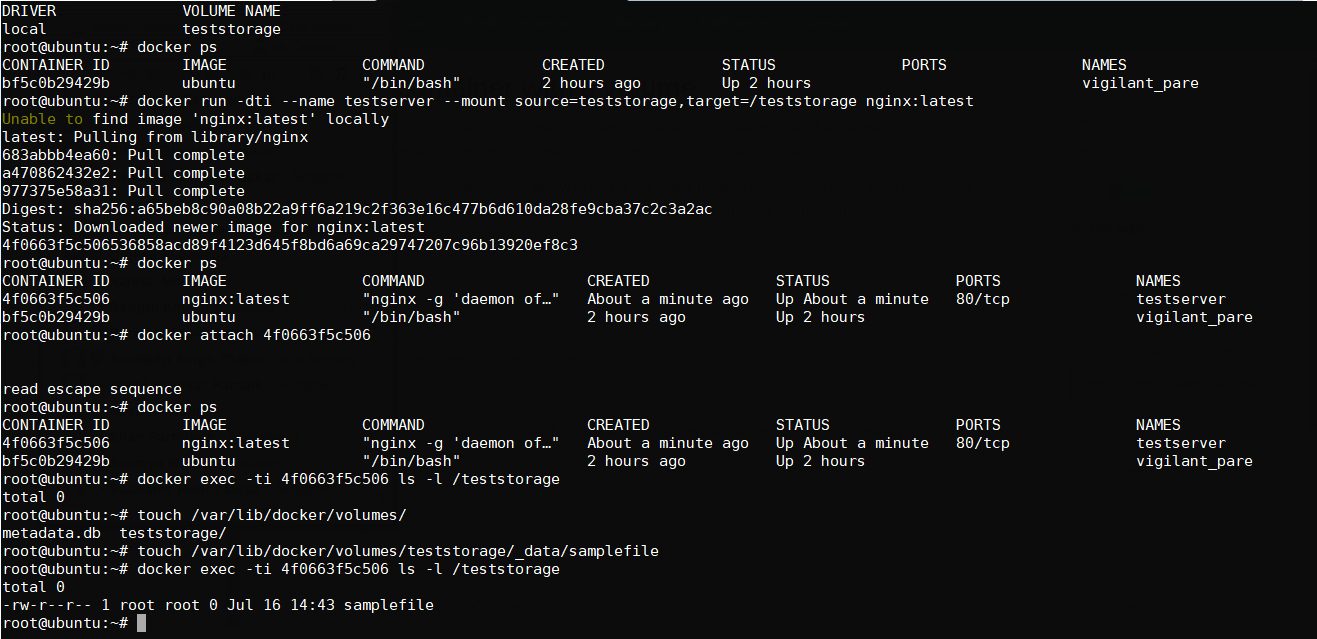
Let’s start with the volume creation on the docker engine:



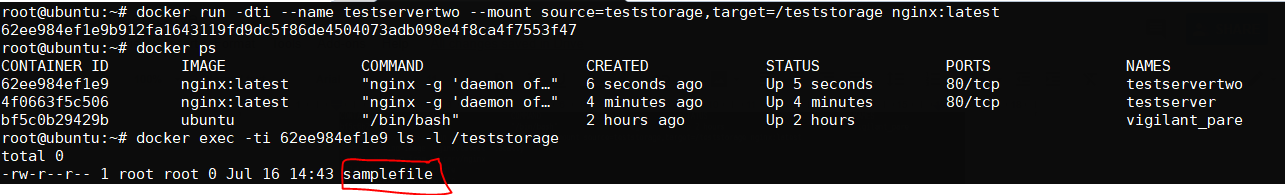
In order to view the volume details:



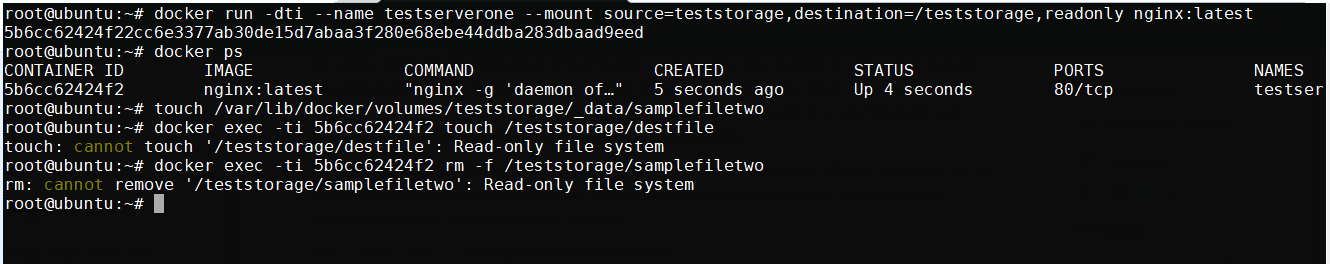
Now let us attach the volume to the container as mentioned below:



From the below example, we can see that single volume “teststorage” to second container too:



Sample example for volumes with read-only permissions:



**Dockerfile Complete:**

A Dockerfile is a text file which has some series to instructions to build own image with customized settings.

Let us look into dockerfile syntax and variable that need to passed

Key variables in Dockerfile:

FROM

MAINTAINER(optional)

RUN

CMD

ENV

ADD  
COPY

WORKDIR

ENTRYPOINT

LABEL

VOLUME

USER

EXPOSE

Example:

FROM ubuntu

MAINTAINER sudhams reddy duba ([dubareddy.383@gmail.com](mailto:dubareddy.383@gmail.com))

RUN apt-get update

RUN apt-get install nginx -y

COPY index.html /usr/share/nginx/html/

ENTRYPOINT [“/usr/sbin/nginx”, “-g”, “daemon off;”]

EXPOSE 80

The ENTRYPOINT specifies a command that will always be executed when the container starts. The CMD specifies arguments that will be fed to the ENTRYPOINT.

Iptables:

iptables -t nat -A DOCKER -p tcp --dport ${YOURPORT} -j DNAT --to-destination ${CONTAINERIP}:${YOURPORT}

iptables -t nat -A POSTROUTING -j MASQUERADE -p tcp --source ${CONTAINERIP} --destination ${CONTAINERIP} --dport ${YOURPORT}

iptables -A DOCKER -j ACCEPT -p tcp --destination ${CONTAINERIP} --dport ${YOURPORT}

**Kubernetes  
  
Kubernetes Objects:**

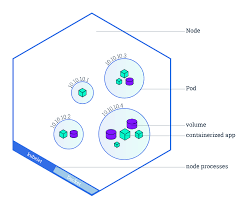
Kubernetes contains a number of abstractions that represent the state of your system: deployed containerized applications and workloads, their associated network and disk resources, and other information about what your cluster is doing.

Following are the basic Kubernetes Object types:

1. Pod
2. Service
3. Volume
4. Deployment
5. Replication Controller
6. Replication Set
7. Namespace
8. ConfigMap
9. PersistantVolume
10. PersistantVolume Claim
11. Secrets
12. Jobs
13. Statefulset
14. DaemonSet
15. RBAC
16. CronJobs

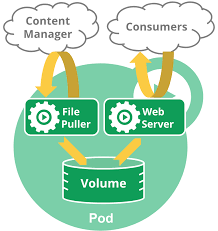
**POD:**

A POD (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers.



Pods in kubernetes can be allocated with two different ways:

1. Pods that run a single container
2. Pods that run with multiple containers to work together



Sample POD with basic needed specification:

Below yaml show the basic pod creation with nginx image:

===

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

===

From the above yaml, we have called keys which represent as follows

apiVersion → Object api version to create pod which a specific API call

Kind → represent the Object type which you are trying to create

metadata → allocating names and labels to add some additional names

under spec we will be writing the pod config details like image which you want to use, port that need to be exposed from container running in the pod and we can define other keys which are mentioned below as max.

---

apiVersion: v1

kind: Pod

metadata:

name: nginx

namespace: default # If you want to launch POD on specific namespace

labels:

env: test

spec:

containers:

- name: "local-nginx-server"

imagePullPolicy: IfNotPresent

image: nginx:1.7.9

ports:

- containerPort: 80

hostIP: 192.168.2.10

hostPort: 8080

protocol: TCP

resources:

limits:

memory: "100Mi"

ephemeral-storage: "4Gi"

cpu: "1"

requests:

memory: "95Mi" #E, P, T, G, M, K, Ei, Pi, Ti, Gi, Mi, Ki

ephemeral-storage: "2Gi"

cpu: "0.5"

# command: ["apt", "update"] # Or we can pass as arguments args: ["aux", "|", "grep", "nginx"]

lifecycle:

postStart:

exec:

command: ["/bin/sh", "-c", "echo Hello from the postStart handler > /usr/share/message"]

preStop:

exec:

command: ["/usr/sbin/nginx","-s","quit"]

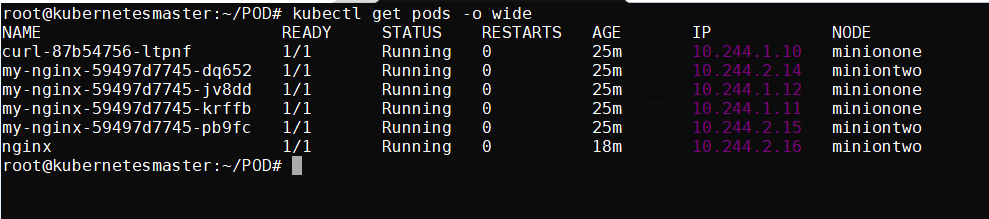
restartPolicy: Always

nodeSelector: #This key will search for specific node with the label mentioned

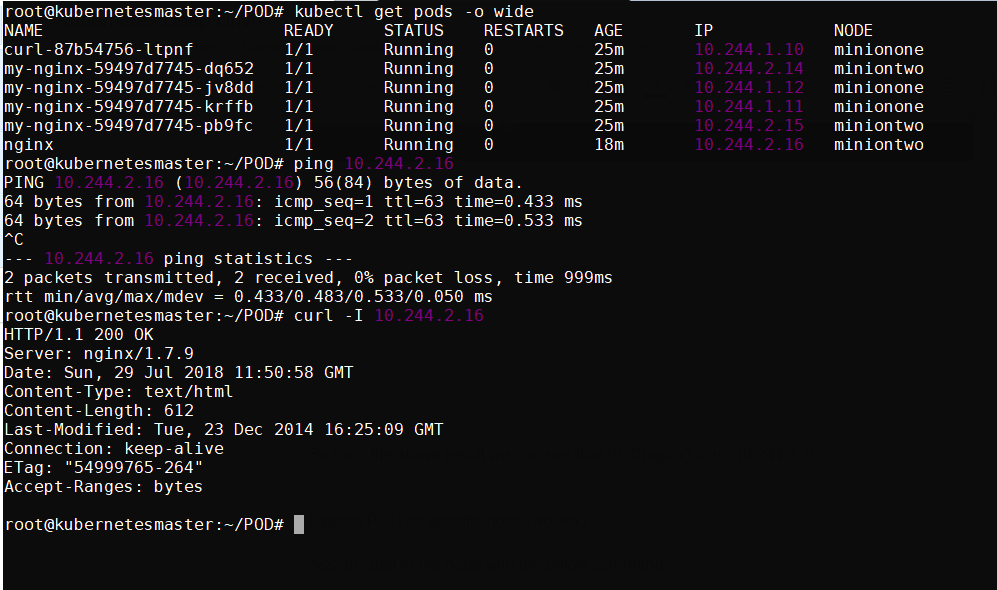
disktype: ssd

++

How to view pod Ip address with simple output



So from the above result we can see that POD(nginx) ip is “10.244.2.16”



Launch POD on specific node (worker)

Assign label to the node with the below command:

# kubectl label nodes <your-node-name> disktype=ssd

To verify the node assigned with the Label

# kubectl get nodes –show-labels

++

apiVersion: v1  
kind: Pod  
metadata:  
 name: nginx  
 labels:  
 env: test  
spec:  
 containers:  
 - name: nginx  
 image: nginx  
 imagePullPolicy: IfNotPresent  
 nodeSelector:  
 disktype: ssd

++

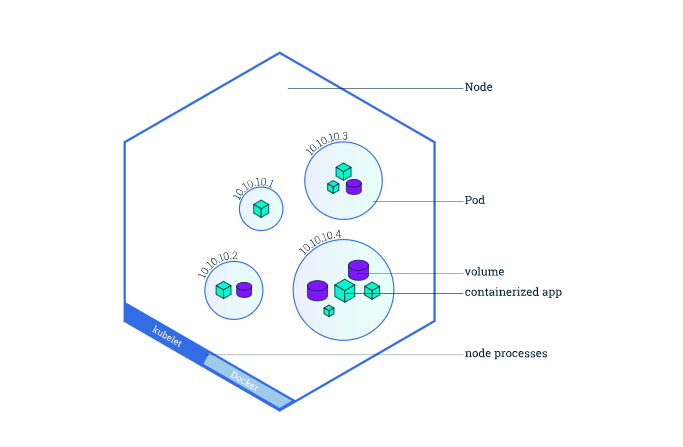
DNS entries:

++

**dnsConfig:  
 nameservers:  
 - 1.2.3.4  
 searches:  
 - ns1.svc.cluster.local  
 - my.dns.search.suffix**

++

**Resource allocation(CPU, Memory and Storage) for POD and Containers:**



These examples shows how to set resource limit and request for the container creation in the POD.

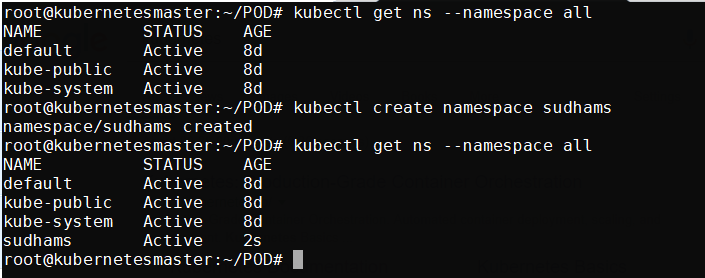
Containers which are launched in specific namespace has quota limits.

Note: if you want to launch the resource pods in a specific namespace.

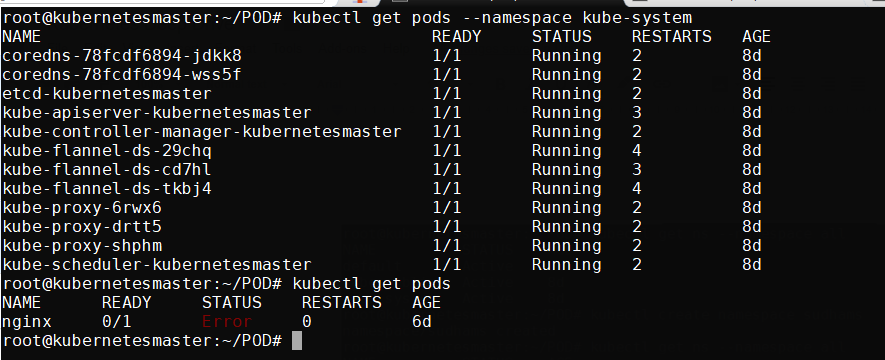
Kubernetes starts with three initial namespaces:

* **default** The default namespace for objects with no other namespace
* **kube-system** The namespace for objects created by the Kubernetes system
* **kube-public** The namespace is created automatically and readable by all users (including those not authenticated). This namespace is mostly reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster. The public aspect of this namespace is only a convention, not a requirement.

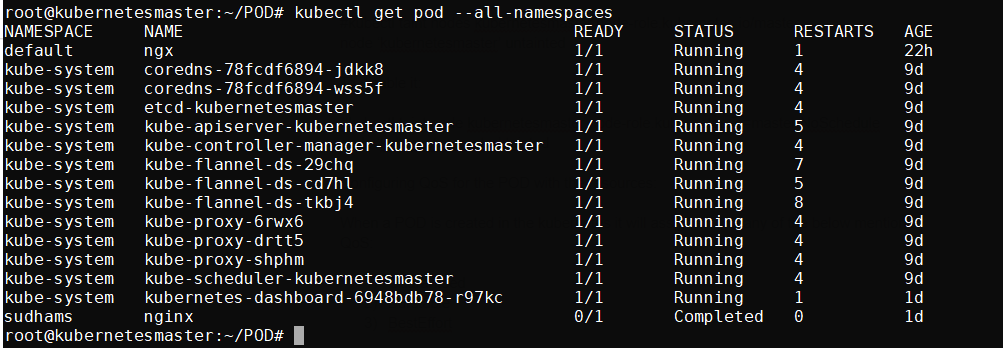
#kubectl create namespace <name>



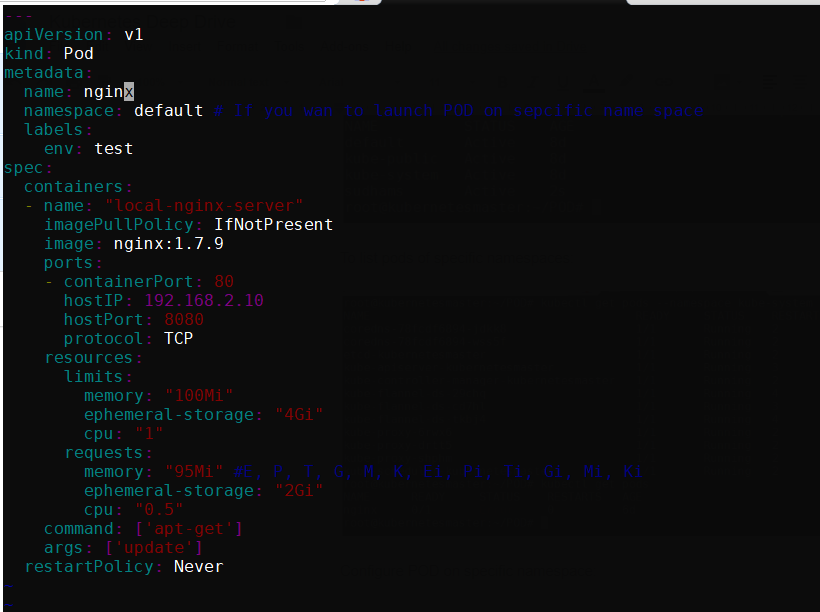
To list pods of specific namespaces:



To list pods of all the namespaces in kubernetes cluster:

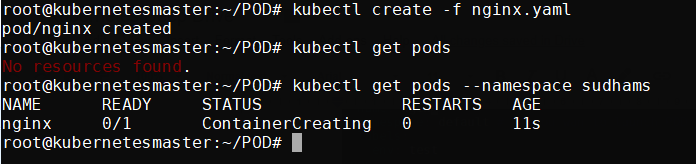


Configure POD on specific namespace:



Under metadata section, we have define a key ‘namespace’.

We have specific the limit of resource that we need for a container in the POD and also specified the resource request.



If you want to launch the pod in specific namespace from CLI, give option like --namespace while creating POD.

In order check the events and POD metadata info like node, events, IP address of the POD etc.,

#kubectl describe pod nginx --namespace srikanth

