

Pod

A Pod represents a running process on your cluster.

A Kubernetes pod is a group of containers that are deployed together on the same host.

A pod is a collection of one or more containers..

pod is modelled as a group of Docker containers with shared namespaces and shared volumes.

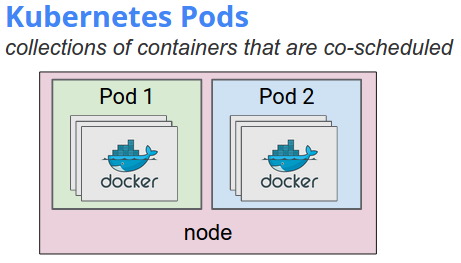
A Pod encapsulates an **application container** (or, in some cases, multiple containers), **storage resources**, **a unique network IP**, and options that how the container(s) should run.

Pod represents a unit of deployment: *a single instance of an application in Kubernetes*, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

[Docker](https://www.docker.com) is the most common container runtime used in a Kubernetes Pod, but Pods support other container runtimes as well.

**Shared Network**

All containers share the same network namespace & port space. Communication over localhost is encouraged. Each container can also communicate with any other pod or service within the cluster.



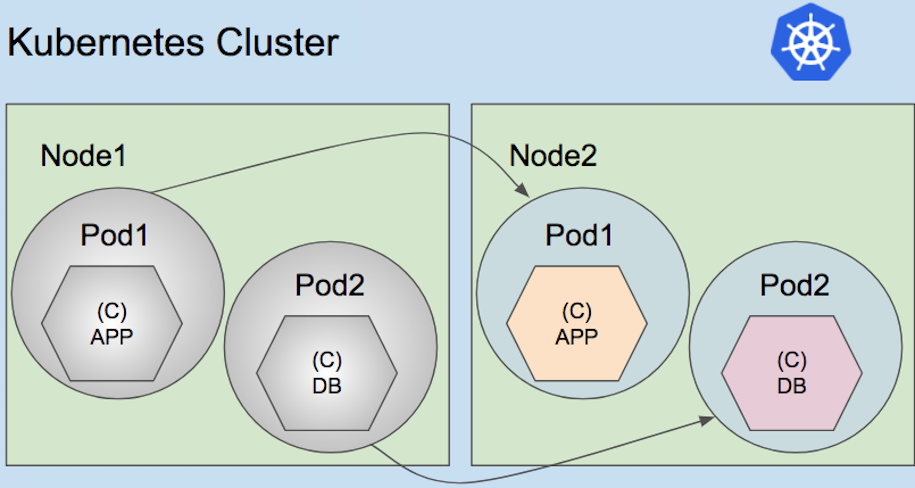
## Uses of pods

Pods can be used to host vertically integrated application stacks (e.g. LAMP), but their primary motivation is to support co-located, co-managed helper programs, such as:

* content management systems, file and data loaders, local cache managers, etc.
* log and checkpoint backup, compression, rotation, snapshotting, etc.
* data change watchers, log tailers, logging and monitoring adapters, event publishers, etc.
* proxies, bridges, and adapters
* controllers, managers, configurators, and updaters

Pods in a Kubernetes cluster can be used in two main ways:

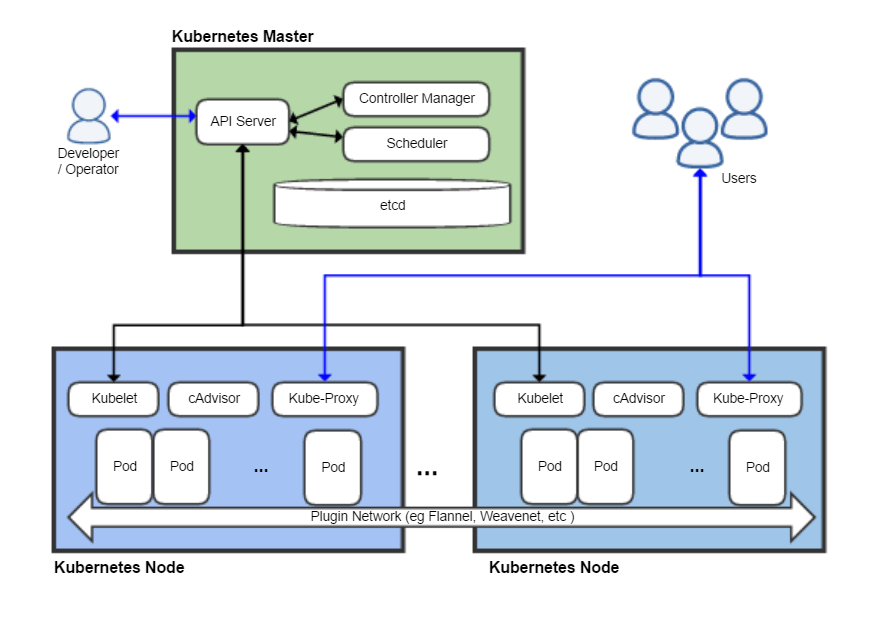
* **Pods that run a single container**. The “one-container-per-Pod” model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container, and Kubernetes manages the Pods rather than the containers directly.
* **Pods that run multiple containers that need to work together**. A Pod might encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources.



# Pod Lifecycle

Here are the possible values for phase:

| **Value** | **Description** |
| --- | --- |
| Pending | The Pod has been accepted by the Kubernetes system, but one or more of the Container images has not been created. This includes time before being scheduled as well as time spent downloading images over the network, which could take a while. |
| Running | The Pod has been bound to a node, and all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting. |
| Succeeded | All Containers in the Pod have terminated in success, and will not be restarted. |
| Failed | All Containers in the Pod have terminated, and at least one Container has terminated in failure. That is, the Container either exited with non-zero status or was terminated by the system. |
| Unknown | For some reason the state of the Pod could not be obtained, typically due to an error in communicating with the host of the Pod |



#### ****WORKING WITH PODS****

* The smallest unit of scheduling in the kubernetes world is **POD**.
* POD’s can contain **one or more containers**.
* We deploy a POD in a cluster using the **manifest** file(yaml or json) or kubectl run command
* We provide the manifest file to the **API server**. It will schedule to deploy in a **node**. Based on the manifest file, the POD will have one (or) more containers.
* irrespective how many containers inside a POD, the POD will get only a **single IP**. It’s like one IP to one POD relationship.
* Every container inside a POD shares single **Network Namespace.**
* All the containers inside a POD shares same **CGROUP Limits, Volumes, Network and IPC namespaces**.

#### ****INTER-POD COMMUNICATION****

* All the POD addresses are fully routable on the POD Network.
* Every POD gets own IP and it’s routable to that Network.
* POD’s can communicate each other without any port mapping.

#### ****INTRA-POD COMMUNICATION****

* Inside the POD, the multiple containers will communicate through a shared local network interface.
* The containers will be available outside the POD by exposing them through ports.
* Two containers cannot use the **same port**.
* Pods will either run as fully functional (or) its down completely.

Pod creation

1.using manifest file (yaml or JSON) using kubectl

2.using kubectl run

(example

kubectl run <podname> --image= imagename --generator=run-pod/v1

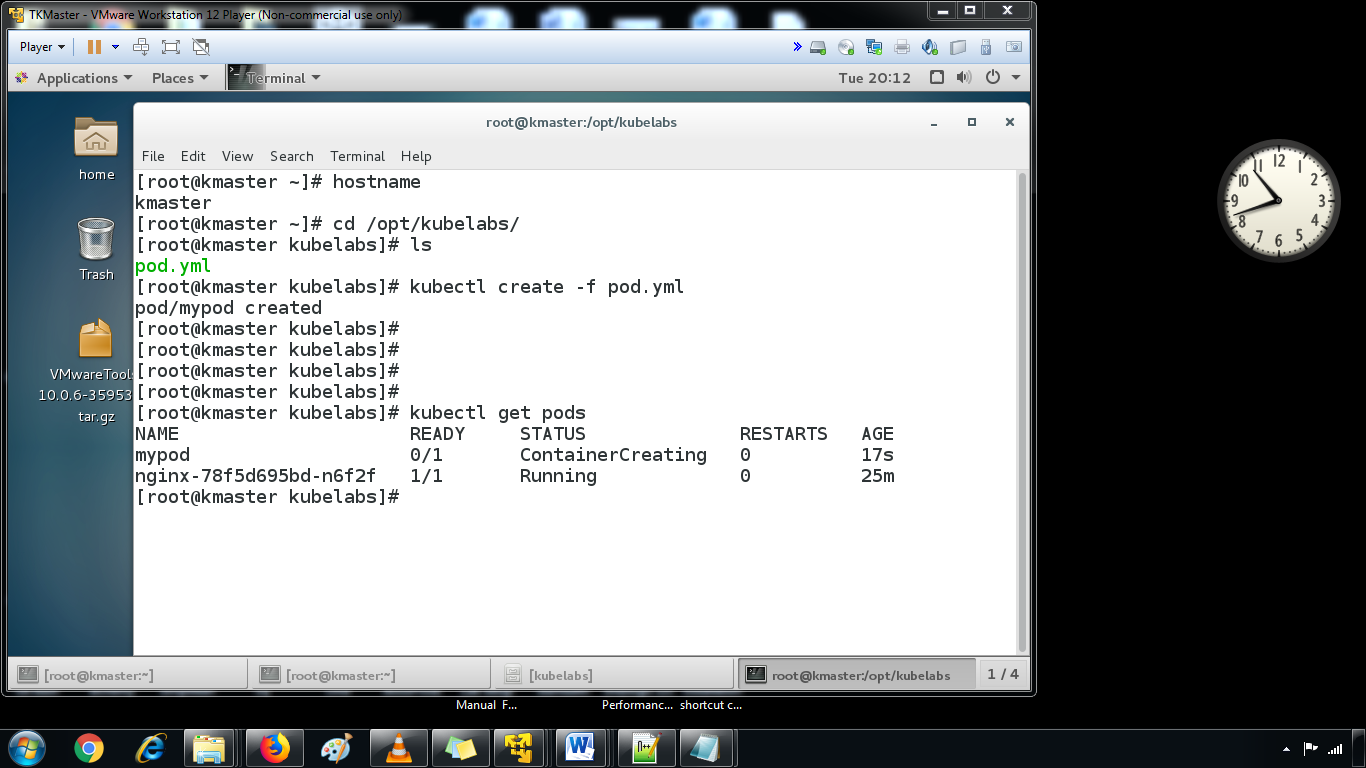
kubectl run cgipod1 --image=nginx --port=80 --labels="app=web,env=dev" --generator=run-pod/v1

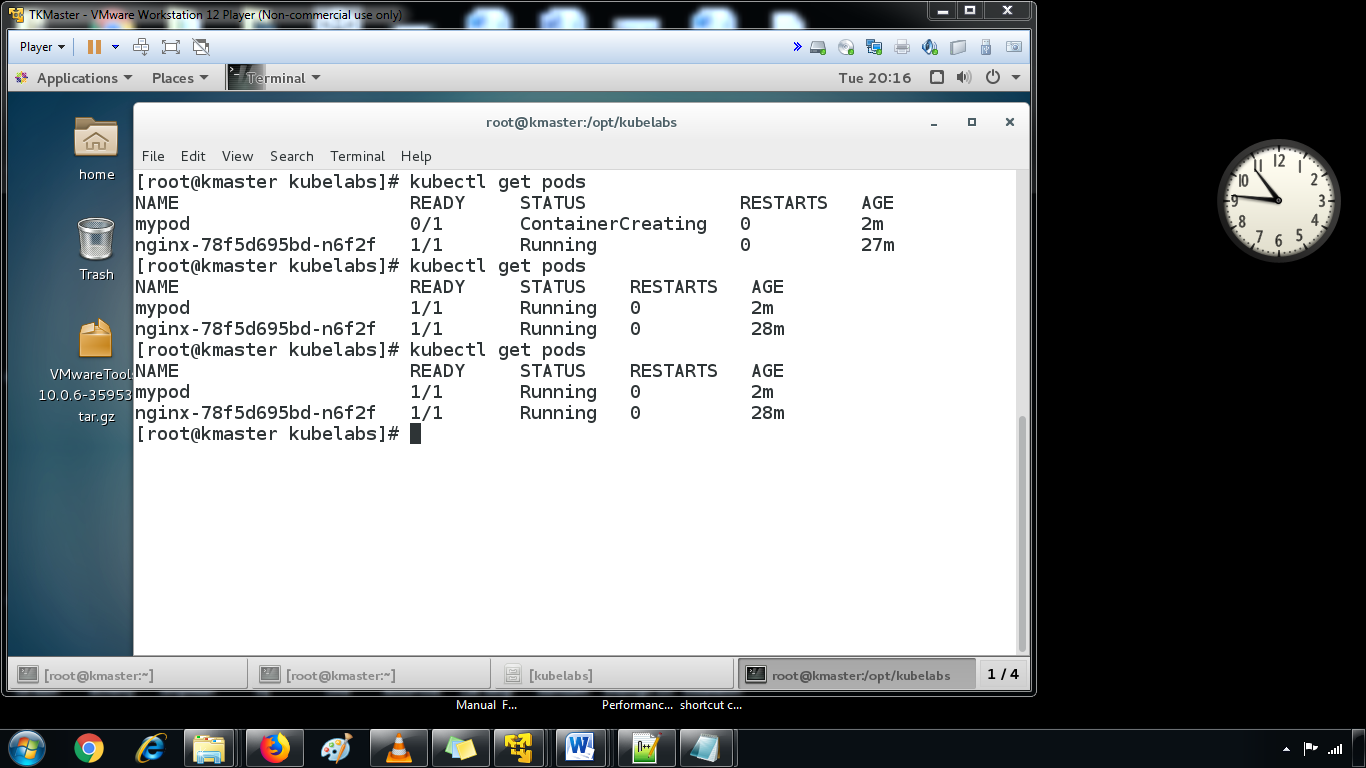
)

#### Generators (Mandatory flag needs to define whenever we create any resources---pods,rc)

You can generate the following resources in kubectl run using --generator flag:

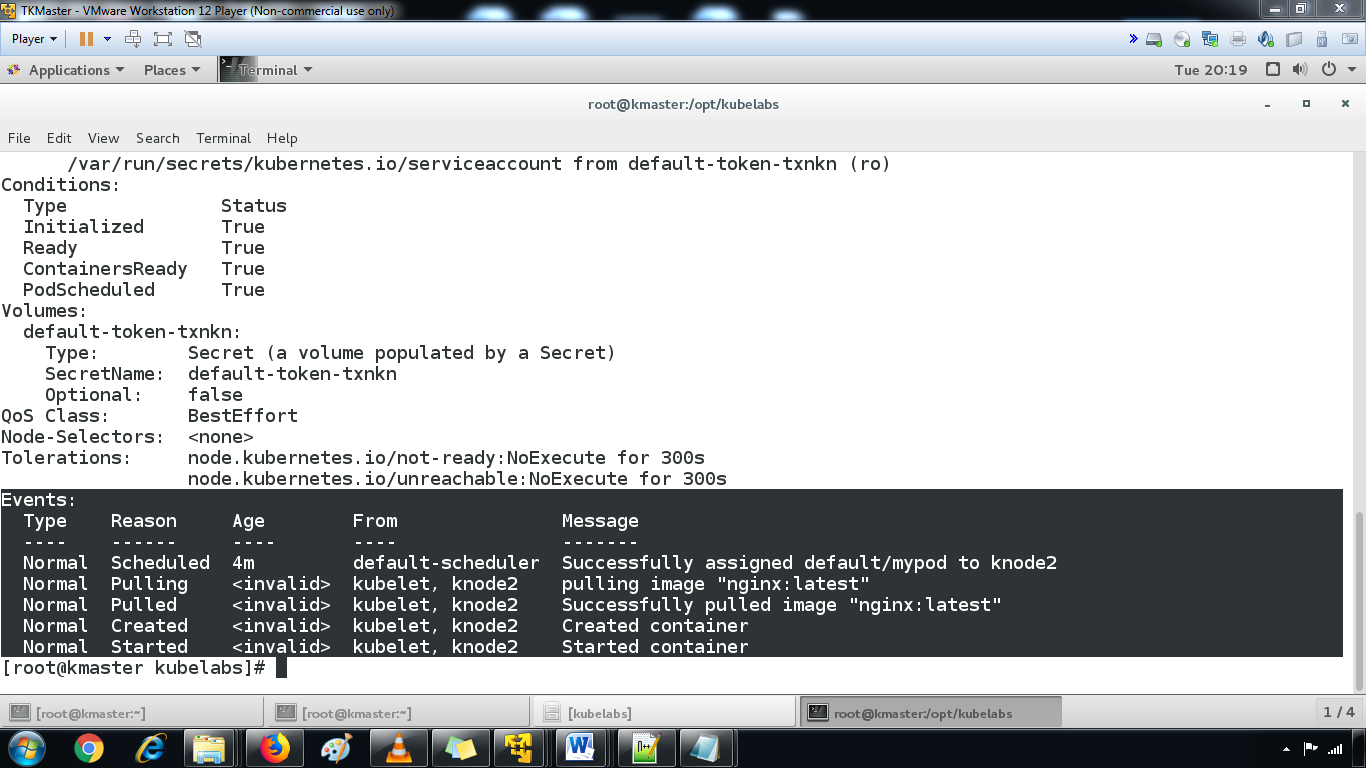
* Pod - use run-pod/v1.
* Replication controller - use run/v1.
* Deployment - use extensions/v1beta1 and for an endpoint - use deployment/v1beta1 (default).
* Deployment - use apps/v1beta1 and for an endpoint - use deployment/apps.v1beta1 (recommended).
* Job - use job/v1.
* CronJob - use batch/v1beta1and for an endpoint - use cronjob/v1beta1(default).
* CronJob - usebatch/v2alpha1 and for an endpoint - use cronjob/v2alpha1 (deprecated).





* POD events information through **describe** command.

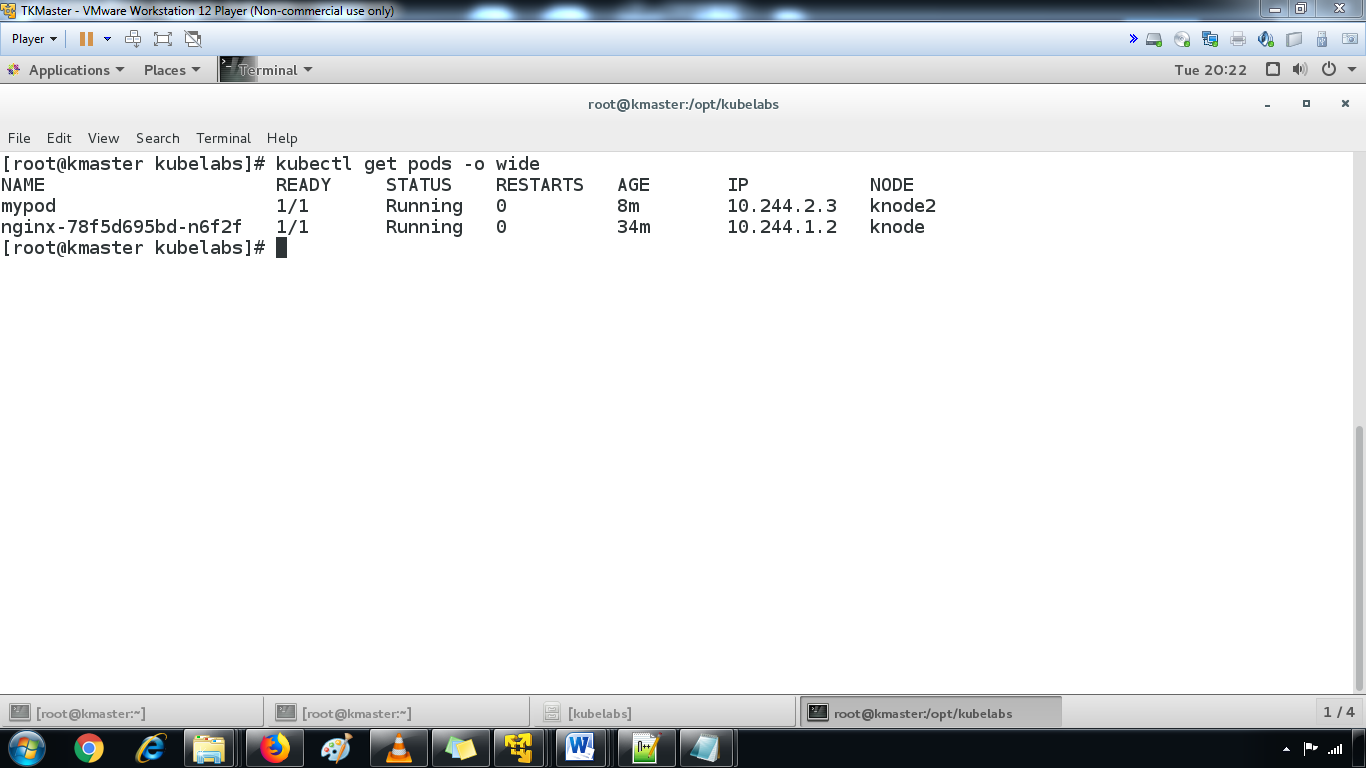
kubectl describe pods podname



kubectl describe pods

to get the pod ip

kubectl get pods -o wide



* To browse the container using the POD IP inside the cluster.

curl -v http://<Pod IP>

Here's an example pod:

apiVersion: v1

kind: Pod

metadata:

name: example-app

labels:

app: example-app

version: v1

role: backend

spec:

containers:

- name: java

image: companyname/java

ports:

- containerPort: 443

volumeMounts:

- mountPath: /volumes/logs

name: logs

- name: logger

image: companyname/logger:v1.2.3

ports:

- containerPort: 9999

volumeMounts:

- mountPath: /logs

name: logs

- name: monitoring

image: companyname/monitoring:v4.5.6

ports:

- containerPort: 1234