The access modes are:

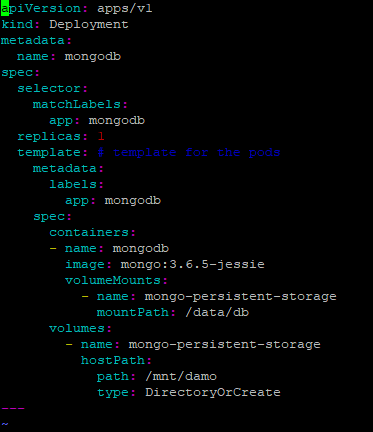
* ReadWriteOnce – the volume can be mounted as read-write by a single node(mostly used)
* ReadOnlyMany – the volume can be mounted read-only by many nodes
* ReadWriteMany – the volume can be mounted as read-write by many nodes

**EXAMPLE 1**:

Type of volume : here we are going to see hostpath volume(local host)---this is used for development environment

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/#volume-v1-core>

| Value | Behavior |
| --- | --- |
|  | Empty string (default) is for backward compatibility, which means that no checks will be performed before mounting the hostPath volume. |
| **DirectoryOrCreate** | If nothing exists at the given path, an empty directory will be created there as needed with permission set to 0755, having the same group and ownership with Kubelet. |
| **Directory** | A directory must exist at the given path |
| **FileOrCreate** | If nothing exists at the given path, an empty file will be created there as needed with permission set to 0644, having the same group and ownership with Kubelet. |
| **File** | A file must exist at the given path |
| **Socket** | A UNIX socket must exist at the given path |
| **CharDevice** | A character device must exist at the given path |
| **BlockDevice** | A block device must exist at the given path |



apiVersion: apps/v1

kind: Deployment

metadata:

name: mongodb

spec:

selector:

matchLabels:

app: mongodb

replicas: 1

template: # template for the pods

metadata:

labels:

app: mongodb

spec:

containers:

- name: mongodb

image: mongo:3.6.5-jessie

volumeMounts:

- name: mongo-persistent-storage

mountPath: /data/db

volumes:

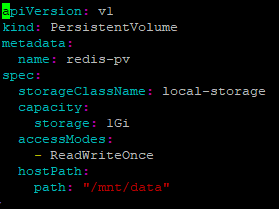
- name: mongo-persistent-storage

hostPath:

path: /mnt/damo

type: DirectoryOrCreate

**EXAMPLE 2:**



apiVersion: v1

kind: PersistentVolume

metadata:

name: redis-pv

spec:

storageClassName: local-storage

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

root@ip-172-31-41-82:~# kubectl get pv

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

mongodb-pv 1Gi RWO Retain Available local-storage 5s

**persistent volume claim** :

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: redisdb-pvc

spec:

storageClassName: local-storage

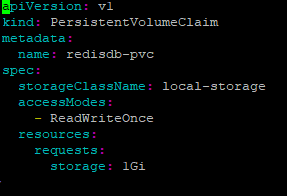
accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi



Create one pod

apiVersion: v1

kind: Pod

metadata:

name: redispod

spec:

containers:

- image: redis

name: redisdb

volumeMounts:

- name: redis-data

mountPath: /data

ports:

- containerPort: 6379

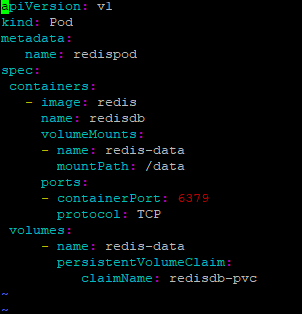
protocol: TCP

volumes:

- name: redis-data

persistentVolumeClaim:

claimName: redisdb-pvc



root@ip-172-31-41-82:~# kubectl exec -it redispod redis-cli

127.0.0.1:6379> SET server:name "redis server"

OK

127.0.0.1:6379> GET server:name

"redis server"

Now quit

Next delete pod

Change the pod name and apply yaml file

Again login to new pod kubectl exec -it redispod2 redis-cli

127.0.0.1:6379> GET server:name

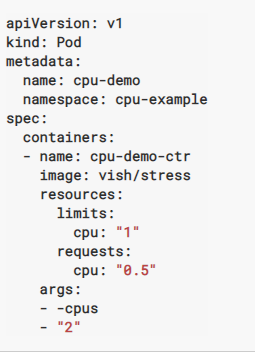
"redis server"

**REQUEST AND LIMIT**:

1 Mi = 1024ki 1ki = 1024 bytes

1 M =1000k 1k=

pods/resource/cpu-request-limit.yaml



The unit suffix **m** stands for “thousandth of a core,” so this resources object specifies that the container process needs 50/1000 of a core (5%) and is allowed to use at most 100/1000 of a core (10%). Likewise **2000m** would be two full cores, which can also be specified as **2** or **2.0**.

If actual memory exceeds ,container will be killed

But if CPU exceeds ,container wont be killed it will be throttling

apiVersion: v1

kind: Pod

metadata:

name: cpu-demo

namespace: cpu-example

spec:

containers:

- name: cpu-demo-ctr

image: vish/stress

resources:

limits:

cpu: "1"

requests:

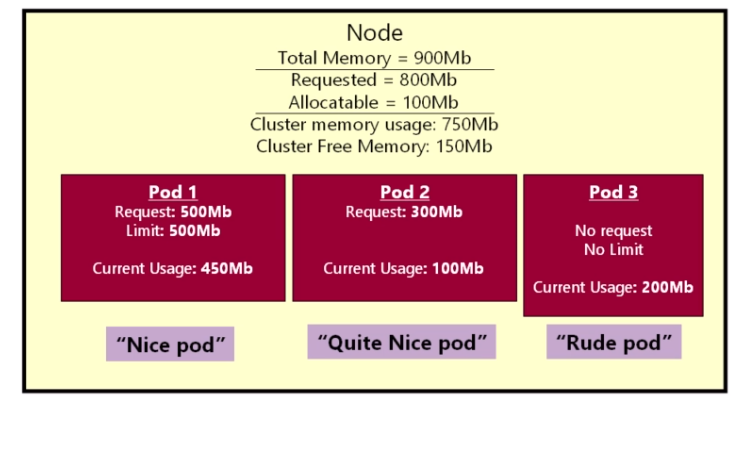
cpu: "0.5"

args:

- -cpus

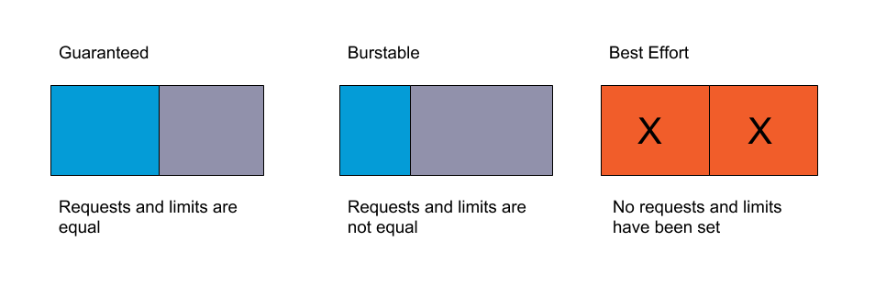
- "2"

**QOS**:

POD 1- QOS:guaranteed

Pod 2 – only request no limit ----QOS: burstable

Pod 3 –QOS- Best Effort



Pods are evicted only if they are using more resources than the user-defined request. This means guaranteed pods will never be evicted, burstable pods will only be evicted if they are using more of the starved resource than allowed, and best effort pods can be preempted at any time.

**POD PRIORITIES**:

U can give priority to pod.   
Pod which will have high priority will be considered first and while eviction pod which has low priority value will be evicted. But its not used in production as in microservices architecture all the microservice application has equal weightage