**CKAD**

**Kubernetes Architecture**

**Node** – A node is machine (physical or virtual) on which Kubernetes is installed.

**Cluster** – A set of nodes.

**Master** – Master is a node with Kubernetes installed on it. It monitors the nodes in cluster and orchestrate the container in the nodes.

**API Server** - It act as front end for the cluster. The user or devices talk using the api server to interact with the cluster.

**Etcd** – It’s a distributed Key-Value store to store all the data to manage the cluster.

**Scheduler** – Scheduler assigns the container to the nodes. It decides where the new container will be assigned.

**Controller** – Brain behind the orchestration. They are responsible for noticing when nodes, container or endpoint goes down. They decide to bring up new container.

**Container Runtime** – Underlying software to run the container.

**Kubelet** – It’s the agent that runs on each node in the cluster. It makes sure that the containers are running as expected.

**Pod**

If you are not given a pod definition file, you may

**Extract the definition to a file using the below command:**

kubectl get pod <pod-name> -o yaml > pod-definition.yaml

kubectl get pod web-app -o yaml > pod.yaml

**Create pod using the imperative commands**

kubectl run redis --image=redis123 --dry-run=client -o yaml > pod.yaml

kubectl run custom-nginx --image=nginx --port=8080

kubectl run redis --image=redis:alpine -l tier=db

Set Image

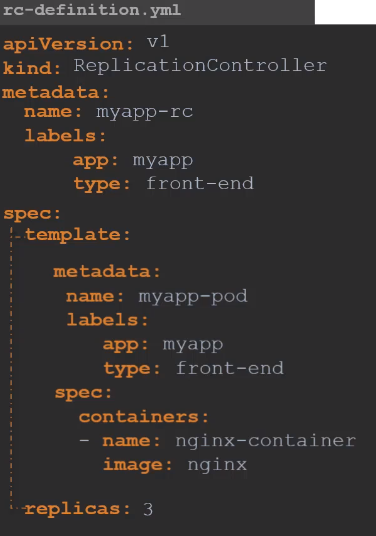
kubectl set image pod/nginx nginx=nginx:1.7.1

To edit

Kubectl edit pod <pod-name>

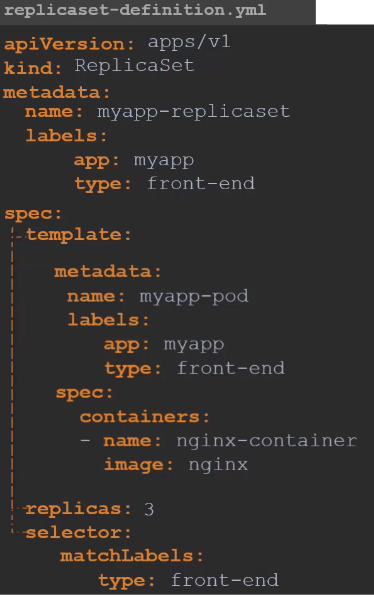
**ReplicationController**

Replication Controller helps us to run the multiple instances of pod in the cluster to provide the high availability. It makes sure that specified number of pods are running all time.



**ReplicaSet**

Replication Controller is older technology that is being replaced by Replica Set.



Imperative commands for replica set

kubectl get replicaset

kubectl replace -f replicaset-def.yml

To **scale up** the replica set and also update replica set definition file

kubectl scale --replicas=6 -f replicaset-def.yml

To **scale up** the replica set

kubectl scale --replicas=6 replicaset myapp-replicaset

**Deployment**

Deployment provide us the capabilities to upgrade the underlying instances seamlessly using rolling updates, undo, pause and resume changes as required. It also creates the replica set internally.

kubectl create deployment httpd-frontend --image=httpd:2.4-alpine

kubectl **scale** deployment --replicas = 3 httpd-frontend

kubectl **autoscale** deploy nginx --min=5 --max=10 --cpu-percent=80

**Rolling Update and Rollback in Deployment**

**Rollout and versioning**

Kubectl rollout status deployment myapp-deployment

Kubectl rollout history deployment myapp-deployment

kubectl rollout history deployment nginx --revision=1

**Deployment Strategy**

* RollingUpdate (default)
* Recreate

**Update Deployment / Upgrade**

Kubectl apply -f deployment-definition.yml

Kubectl set image deployment myapp-deployment nginx=nginx:1.9.1

kubectl set image deployment nginx nginx=nginx:1.17 --record

**Rollback**

Kubectl rollout undo deployment myapp-deployment

kubectl rollout undo deploy nginx --to-revision=2

**Pause the Deployment**

kubectl rollout pause deploy nginx

**Service**

Types of services

* NodePort
* ClusterIP (default)
* LoadBalancer

Kubectl create -f service-definition.yaml

Kubectl get services

Kubectl expose deployment simple-webapp-deployment –name=webapp-service –target-port=8080 –type=NodePort –port=8080 –dry-run=client -o yaml > service-definition.yaml

kubectl expose pod redis --port=6379 --name redis-service

kubectl expose pod redis --port=6379 --name redis-service --dry-run=client -o yaml

kubectl expose pod nginx --port=80 --name nginx-service --type=NodePort --dry-run=client -o yaml

kubectl create service clusterip redis --tcp=6379:6379 --dry-run=client -o yaml

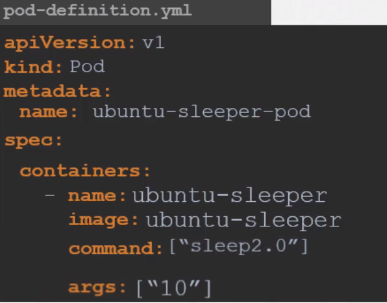
kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml

kubectl run httpd --image=httpd:alpine --port=80 --expose

**Namespace**

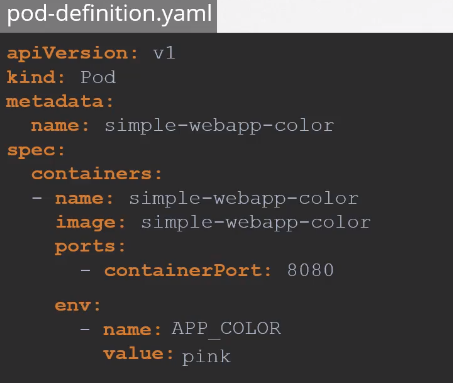
kubectl create namespace dev-ns

**Commands and Arguments**



Kubectl run ubuntu-sleeper-pod --image=ubuntu-sleeper --command sleep2.0 10

**Environment Variable**



Kubectl run simple-webapp-color –image=simple-webapp-color –port=8080 –env=”APP\_COLOR=pink”

**Config Map**

**Create config map**

kubectl create configmap <config-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>

kubectl create configmap <config-name> --from-file=<path-to-file>

kubectl create cm configmap3 --from-env-file=config.env

**Update ConfigMap in pod**

envFrom:

- configMapRef:

name: app-config

**Single Env**

env:

- name: APP\_COLOR

valueFrom:

configMapKeyRef:

name: app-config

key: APP\_COLOR

**Volume**

volumes:

- name: app-config-volume

configMap:

name: app-config

**Secret**

**Create Secret**

kubectl create secret generic <secret-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>

kubectl create secret generic <secret-name> --from-file=<path-to-file>

**Encode and decode secret**

echo -n 'mysql' | base64

echo -n 'mysql' | base64 --decode

**Update Secret in pod**

envFrom:

- secretRef:

name: app-secret

**Single Env**

env:

- name: DB\_Password

valueFrom:

secretKeyRef:

name: app-secret

key: DB\_Password

**Volume**

volumes:

- name: app-secret-volume

secret:

name: app-secret

**Security Context**

kubectl exec ubuntu-sleeper – whoami

kubectl exec -it ubuntu-sleeper -- date -s '19 APR 2012 11:14:00'

securityContext:

runAsUser: 1000

capabilities:

add: ["SYS\_TIME"]

**Service Account**

Kubectl create serviceaccount dashboard-sa

**Update sa in pod**

Spec:

Containers:

- name: my-k8-dashboard

Image: my-dashboard

serviceAccount: dashboard-sa

**Resource Requirements**

Containers:

- name: my-k8-dashboard

Image: my-dashboard

resources:

limits:

memory: 20Mi

requests:

memory: 5Mi

**Taints and Tolerations**

Taint and Toleration is configured only to tell node to accept only the certain pods with toleration. It doesn’t guarantee that a particular pod will run on that node. That is something needs to be achieved using NodeSelector and NodeAffinity

**Types of Taint**

* NoSchedule
* PreferNoSchedule
* NoExecute

**Taints – Node**

Kubectl taint nodes node-name key=value:taint-effect

kubectl taint nodes node01 spray=mortein:NoSchedule

**Tolerations – Pod**

tolerations:

- key: "spray"

operator: "Equal"

value: "mortein"

effect: "NoSchedule"

**Remove Taint**

kubectl taint nodes master node-role.kubernetes.io/master:NoSchedule-

**Node Selectors**

spec:

containers:

- name: my-k8-dashboard

Image: my-dashboard

nodeSelector:

size: Large

**Label the Node**

Kubectl label nodes <node-name> <key>=<value>

**Label the Pod**

Kubectl label pod <pod-name> <key>=<value>

**Remove Label**

kubectl label pod nginx1 nginx2 nginx3 app-

**Annotate Pod**

kubectl annotate pod nginx1 nginx2 nginx3 description='my description'

**Get Annotation**

kubectl describe po nginx1 | grep -i 'annotations'

**Remove Annotation**

kubectl annotate pod nginx1 nginx2 nginx3 description-

**Node Affinity**

spec:

containers:

- name: my-k8-dashboard

Image: my-dashboard

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution /

preferredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

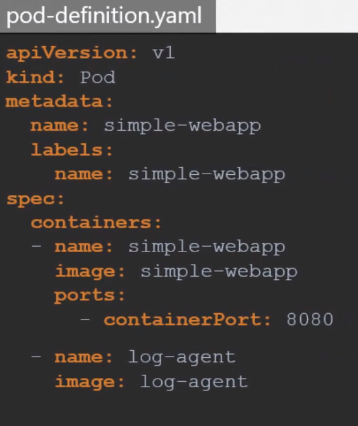
- key: size

Operator: In

Values:

- Large

**Multi Container Pod**



**Readiness Probes**

**For HTTP**

readinessProbe:

httpGet:

path: /api/ready

port: 8080

initialDelaySeconds: 10

periodSeconds: 5

failureThreshold: 8

**For TCP**

readinessProbe:

tcpSocket:

port: 3306

**For Exec Command**

readinessProbe:

exec:

command:

- cat

- /app/is\_ready

**Liveness Probe**

**For HTTP**

livenessProbe:

httpGet:

path: /api/ready

port: 8080

initialDelaySeconds: 10

periodSeconds: 5

failureThreshold: 8

**For TCP**

livenessProbe:

tcpSocket:

port: 3306

**For Exec Command**

livenessProbe:

exec:

command:

- cat

- /app/is\_ready

**Logging**

Kubectl logs -f even-simulator-pod

Kubectl logs -f even-simulator-pod event-simulator

**If pod crashed and restarted, get logs about the previous instance**

kubectl logs nginx -p

**Monitoring**

Kubernetes doesn’t come with full featured monitoring solution. However, there are multiple open source monitoring solution available

* Metrics Server
* Prometheus
* Elastic Stack
* DataDog
* dynatrace

**MetricServer**

Steps to install Metric server

* + Git Clone <https://github.com/kubernetes-incubator/metrics-serve>
  + From inside the cloned directory run Kubectl create -f .

kubectl top node

kubectl top pod

**Labels and Selectors**

Kubectl get pods --show-labels

Kubectl get pods -l env=dev

Kubectl get pods --selector app=App1

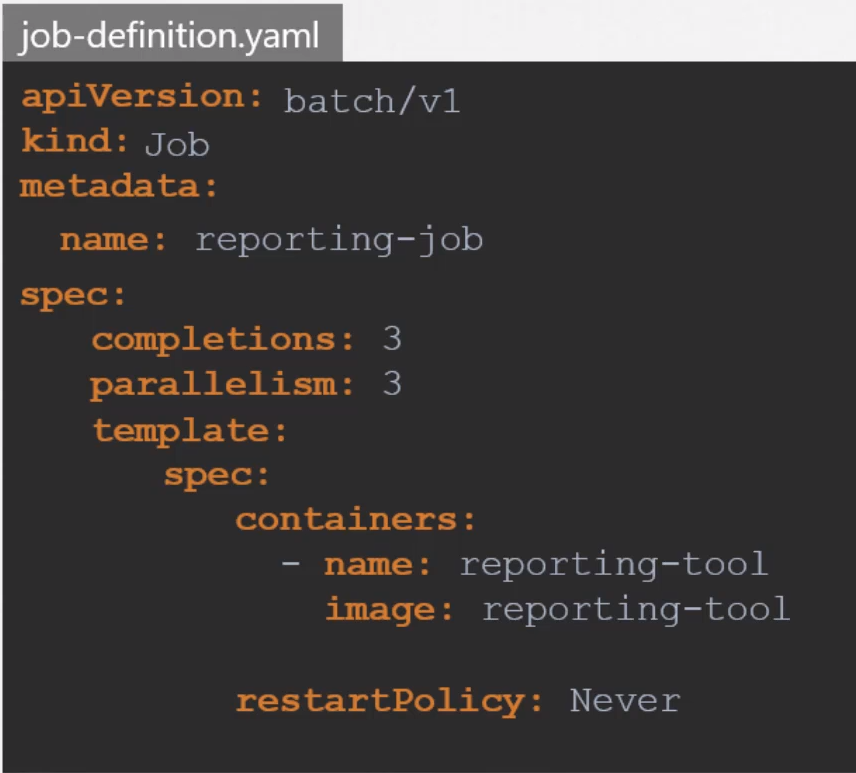
**Annotations**

To record details for information purpose

Annotations:

buildVersion: 1.34

**Jobs**



Kubectl create -f job-definition.yaml

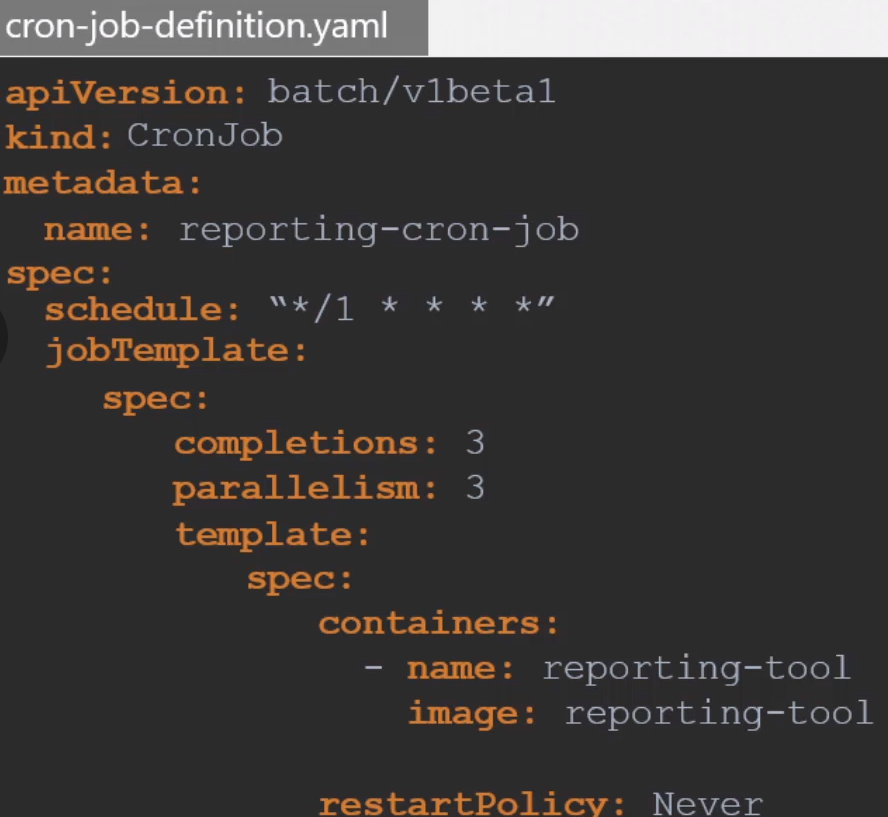
kubectl create job throw-dice-job --image=kodekloud/throw-dice

Kubectl get jobs

Kubectl get pods

Kubectl logs math-add-job

**CronJob**



Kubectl create -f cron-job-definition.yaml

kubectl create cronjob throw-dice-cron-job --image=kodekloud/throw-dice --schedule="30 21 \* \* \*"

Kubectl get cronjob

**Ingress**

**Deploy**

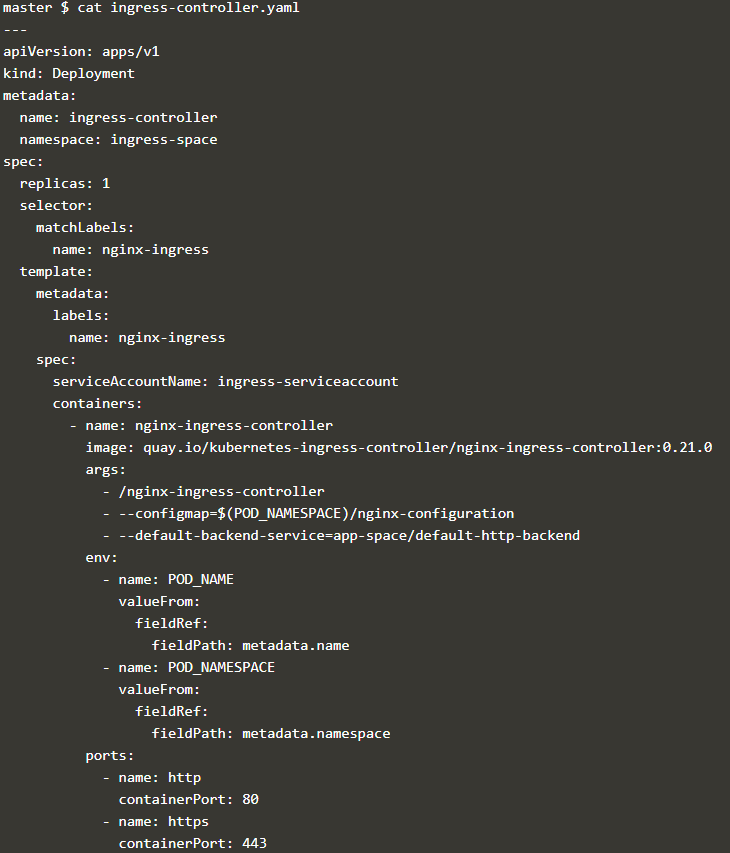
**IngressController** – Nginx, HAProxy, Traefik, Contour, Istio, GCP HTTP(s) Load Balancer (GCE)

In order to deploy the Ingress Controller, we would need a Deployment (eg: nginx), Service, ConfigMap and ServiceAccount

kubectl create namespace ingress-space

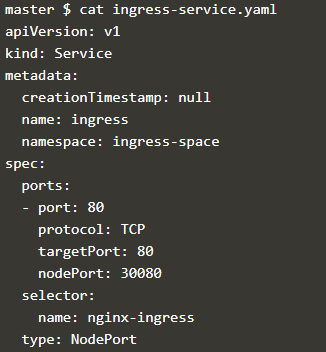
kubectl create configmap nginx-configuration -n ingress-space

kubectl create sa ingress-serviceaccount -n ingress-space



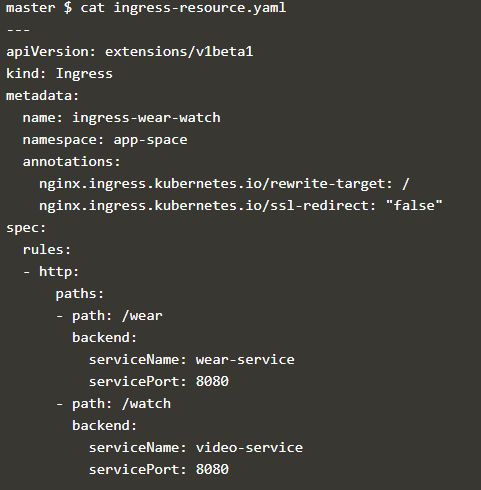
kubectl create -f ingress-controller.yaml

kubectl expose deployment -n ingress-space ingress-controller --type=NodePort --port=80 --dry-run=client -o yaml > ingress-service.yaml



**Configure**

Ingress Resources

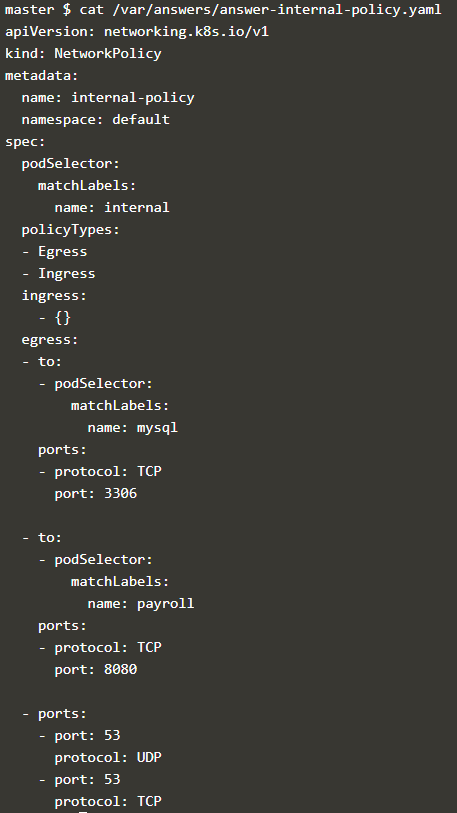






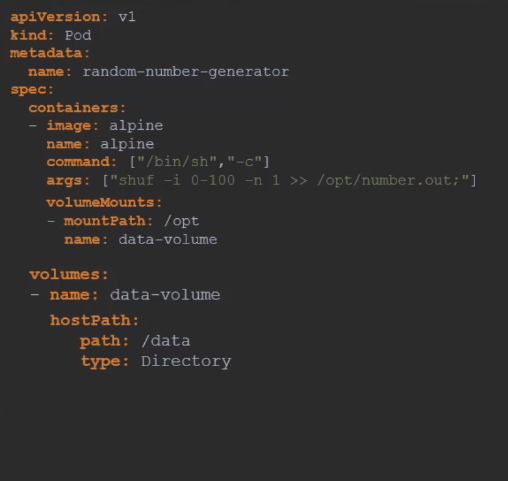


**Network Policy**



**Volume**

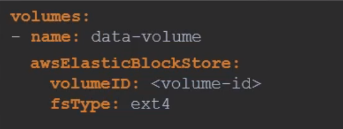
**Volumes and Mounts**



/data Directory works fine with single node architecture but in case of multi node architecture this is not recommended.

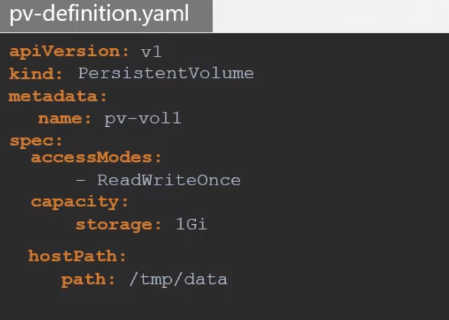
Kubernetes supports different **storage solution**

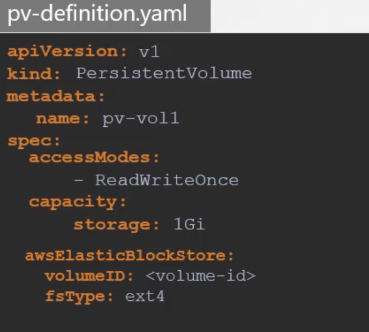
* + NFS
  + GlusterFS
  + Flocker
  + Ceph
  + Scaleio
  + AWS
  + Google Persistent Disk
  + AzureDisk



**Persistent Volume**

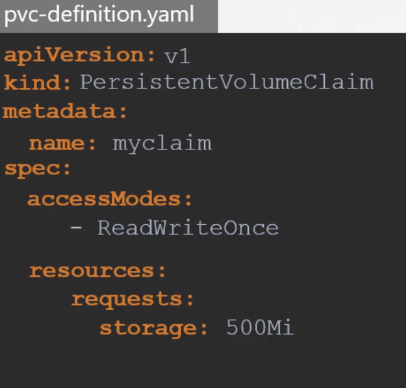
In the previous section we saw that the volume is defined within the pod definition. So in a scenario where thousands of pods are deployed in case of any modification or change user will have to manually update all the pod definition. The better solution would be managing the volume in a centralized way using the persistent volume.



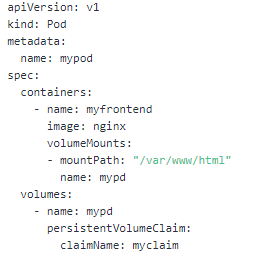


**Persistent Volume Claim**

Administrator creates the PV and user create the Persistent Volume Claim. Kubernetes binds the PV and PVC based on the request and properties. Every PVC is bound to a single PV.



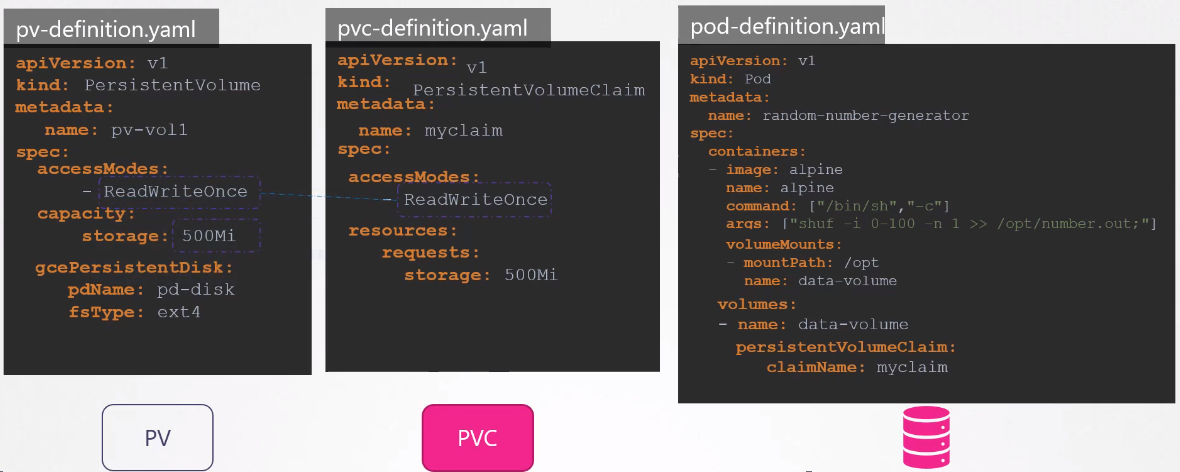
Use PVC in pod definition file



When PVC is deleted there are many ways to handle the PV using the **persistentVolumeReclaimPolicy**.

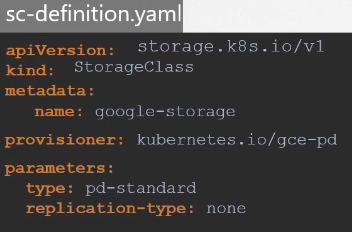
* Retain
* Delete
* Recycle

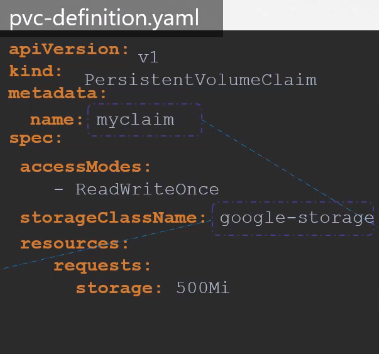
**Storage Class**



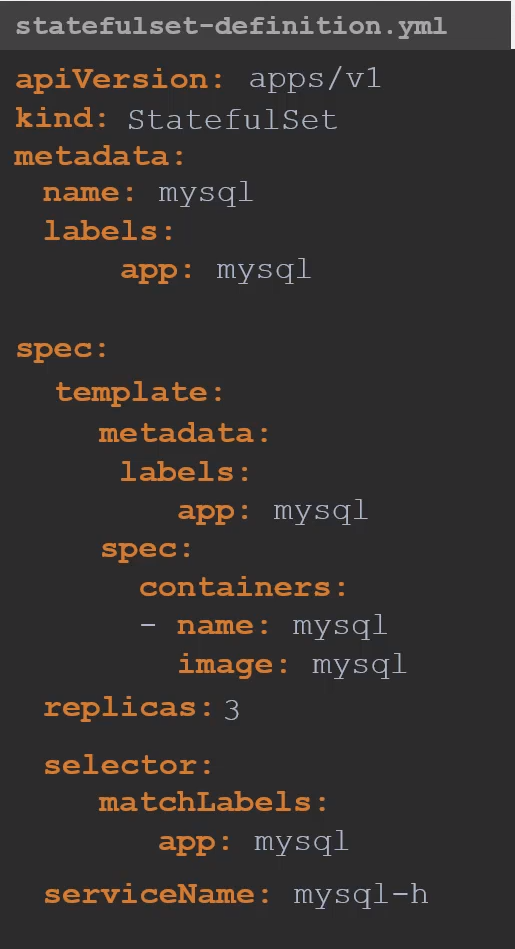
The problem here is before even PV is created you must have created the disk on Google Cloud. Every time an application needs storage you would need to manually create the disk and then create PV with the same name. This is called Static Provisioning.

It would have been nice if the volume gets provisioned automatically when the application requires it. That is where Storage Class comes in. Now the user doesn’t need to create PV.





**Stateful Set**



Kubectl create -f statefulset-definition.yaml

Kubectl scale statefulset mysql –replicas=5

Kubectl scale statefulset mysql –replicas=

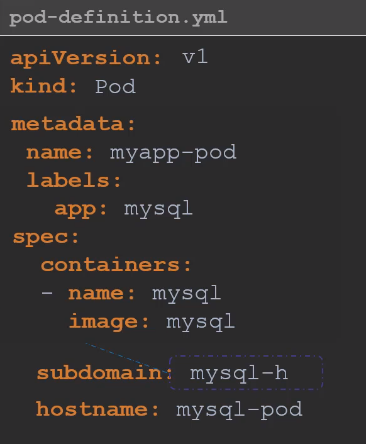
Kubectl delete statefulset mysql

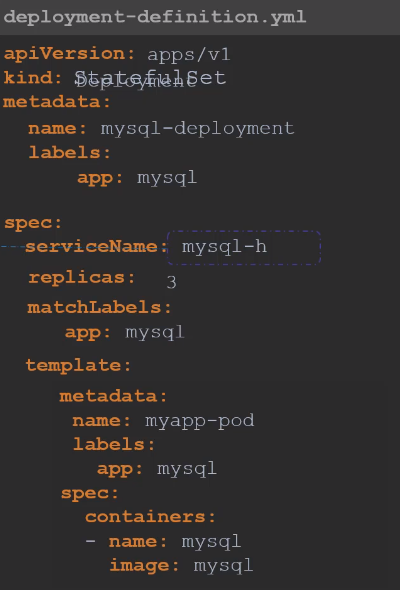
If you don’t want the pods to come up sequentially but want to keep other behavior of stateful set the use **podManagementPolicy: Parallel**

**Headless Service**



Pod definition will result into the same dns name in order to resolve that use Stateful Set





**Storage in StatefulSet**

**Miscellaneous Commands**

**Create alias for kubectl**

alias k=kubectl

complete -F \_\_start\_kubectl k

**To set the namespace context**

Kubectl config set-context –current –namespace=mynamespace

**Explain Command**

Kubectl explain po --recursive | grep -C10 -i liveness

kubectl explain cronjob.spec.jobTemplate --recursive

kubectl explain pods --recursive | grep envFrom

**Command to create pod with limits, requests and env**

kubectl run nginx -image=nginx --restart=Never --port=80 --namespace=myname --command --serviceaccount=mysa1 --env=HOSTNAME=local --labels=bu=finance,env=dev --requests='cpu=100m,memory=256Mi' --limits='cpu=200m,memory=512Mi' --dry-run -o yaml - /bin/sh -c 'echo hello world'

**Create Resource Quota**

kubectl create quota myrq --hard=cpu=1,memory=1G,pods=2 --dry-run=client -o yaml

**Wget from one pod to another**

kubectl run busybox --image=busybox --rm -it --restart=Never -- wget -O- 10.244.1.4:80

**Create a busybox pod that echoes 'hello world' and then exits**

kubectl run busybox --image=busybox -it --restart=Never -- echo "hello world"

**Do the same, but have the pod deleted automatically when it's completed**

kubectl run busybox --image=busybox -it --rm --restart=Never -- echo "hello world"

**Connect to the multi container pod**

kubectl exec -it busybox -c busybox2 -- ls

**# run connectivity with timeout (5 seconds)**

kubectl run curl --image=radial/busyboxplus -it --rm --restart=Never -- curl -m 5 my-service:8080

kubectl run wget --image=busybox -it --rm --restart=Never -- wget --timeout 5 -O- my-svc:8080

kubectl run netcat --image=busybox -it --rm --restart=Never -- nc -w 5 -zv my-service 8080

**Execute a shell on a pod**

kubectl exec -it webapp-color -- sh

**Linux Cut command**

cat /etc/passwd | cut -f 1 -d ':' > /etc/foo/passwd

**Linux copy command**

kubectl exec busybox -it -- cp /etc/passwd /etc/foo/passwd

**Linux List command**

kubectl exec busybox2 -- ls /etc/foo

**Copy '/etc/passwd' from the pod to your local folder**

kubectl cp busybox:/etc/passwd ./passwd

kubectl logs altapod | sudo tee ~/opt/answers/mypod.logs

sudo -i