**K8**

**What is pod?**

**Kubernetes could not deploy the container directly to the nodes. In Kubernetes the containers are encapsulated inside the pod and the pods are deployed to the nodes. So, Pod is the basic atomic unit of scheduling for Kubernetes. So, by adding or deleting the pods we can scale up and scale down the app. If the load increase further, we can add new node and start deploying pods onto the new node.**

**How to deploy pod?**

**To deploy pod, we need to write pod manifest file in json or yaml format. The pod manifest file contains the container images which we want to deploy. The manifest file submits to the API server on the master node. After that API server and the scheduler components on master nodes decides and deploys these pods onto appropriate worker nodes.**

**What is multi-container pod?**

**Mostly there will be one container inside one pod. But sometimes we can see there will be multiple containers inside a pod.**

**Sometimes you will come across a scenario where you have a helper container that might be doing some kind of supporting task. For our main web app, such as processing a user entered data or a processing a file uploaded by user etc. In that case you need both helper and app container inside one pod. So, in this case if the app container dies then the helper container also dies as they are part of same container.**

**Multi container inside the pod communicate with each other directly by referencing using a local host because they share the same network namespace. Also easily share the storage space as well.**

**So all these containers within the same pod share, same common IP address, volumes, network namespace , and IPC.**

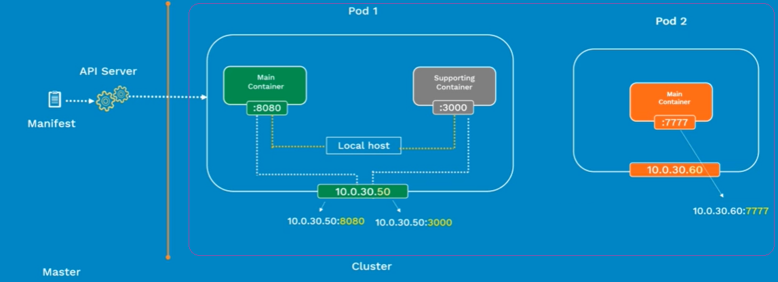
**How pod communicate with another pod?**

**When we deploy the pod onto the worker node inside the Kubernetes cluster, it will get its own IP address. That is known as pod IP address. And by using the pod ip one pod communicate with another pod.**

**How container inside pod communicate with the outside world?**

**All these containers inside a pod operate within network namespace. which means all these containers inside the pod will have the same pod IP address with container port. Using pod ip and container port the container inside the pod communicates outside world.**

**If there is different container inside single pod, then there will be different container ports for each container. In that case the containers inside the single pod can communicate directly with each other's port on localhost ( that is localhost interface ) within the pod.**

****

**How one pod communicates with another pod / Interpol communication?**

**In Kubernetes one pod communicate with another pod using pod network. When we install pod network plugin ( flannel ) and configure Cidr IP range then we will get unique ip address for pod. That's routable within that cluster. This means every pod can directly talk with every other pod and there is no need to mess with any port mapping or anything.**

**Explain the life cycle of pod?**

**Write pod manifest file 🡪 Submit manifest file to API server in k8 master 🡪 pod get scheduled onto a worker node in k8 cluster 🡪 once scheduled pod go to pending state 🡪 In pending state container image mention in pod manifest file get downloaded and start running the container 🡪 once container is up and running pod go to running stage else go to failed stage 🡪 Once the pod achieve its goal get shut down and go to succeeded stage.**

**Note -: if pod dies you can replace it with a new pod, but you cannot bring that pod back.**

**What is Api version?**

**API version defines the version number of Kubernetes object belongs to which stable release of Kubernetes API.**

****

**Write a pod manifest file / pod config file ?**

**vi nginx-pod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nginx-pod**

**labels:**

**app: nginx**

**tier: dev**

**spec:**

**containers:**

* **name: nginx-container**

**image: nginx**

**How to create\deploy a pod ?**

**Kubectl create -f pod\_Yaml\_file\_name**

**Ex🡪 Kubectl create -f nginx-prod.yaml**

**How to list all the pod in k8 cluster? How to list all pod status in k8 cluster?**

**Kubectl get pod**

**How to list specific pod status in k8 cluster?**

**Kubectl get pod pod\_name**

**Ex 🡪 Kubectl get pod nginx-pod**

**How to get the pod ip address and node in k8 cluster?**

**Kubectl get pod -o wide**

**How to get the pod config detail in yaml format ?**

**Kubectl get pod pod\_name -o yaml**

**Ex🡪 Kubectl get pod nginx-pod -o yaml**

**What are the details you will get when execute Kubectl describe?**

**Syntax 🡪 Kubectl describe pod pod-name**

**The Kubectl describe give list of all events from the time pod is assigned to the node till the current status of the pod. As well as it gives label of the pod , node and ip address of the pod on which it is running.**

**How to check the pod is up and running?**

**Go to the master node and ping the pod ip ( which you can get from Kubectl get command ).**

**How to get inside the pod?**

**Kubectl exec -it pod-name -- /bin/sh**

**Ex 🡪 Kubectl exec -it nginx-pod -- /bin/sh**

**How to delete the pod?**

**Kubectl delete pod pod-name**

**Ex 🡪 Kubectl delete pod nginx-pod**

**How to expose pod using NodePort service?**

**Kubectl expose pod pod\_name --type=NodePort --port=Port\_no**

**Ex 🡪 Kubectl expose pod nginx-pod --type=NodePort --port=80**

**How to display the service an node port ?**

**Kubectl describe svc pod\_name**

**Ex 🡪 Kubectl describe svc nginx-pod**

**How to delete service for a pod?**

**Kubectl delete svc pod-name**

**Ex 🡪 Kubectl delete svc nginx-pod**

**How to make the containerize app portable in k8?**

**Using ConfigMap, we can created containerize app portable in k8. With the help of config maps, we can containerize the image and share it with other teammate or open source community.**

**What is ConfigMap in k8?**

**ConfigMap is a Kubernetes object which allows you to separate your configuration from your pods and components. ConfigMap is an API object which contain non-sensitive data as key-value pairs and help to configure k8 pods.**

**So, if you reference a config map inside the podspec that doesn't exist, then pod won't start.**

**How to create ConfigMap? / how do we apply this configuration to the container?**

**We can apply the configuration in three ways.**

1. **Configuration files**
2. **Command line arguments**
3. **Environment variables**

**How to create a ConfigMap?**

**Create ConfigMap using file contain key value pair**

**Kubectl create configmap map\_name --from-file=directory path/file name**

**Create ConfigMap using literal**

**Kubectl create configmap map\_name --from-literal=literal\_vairble=value**

**How to create a configmap from a directory?**

**mkdir -p configure\_pod/configmaps/k8kubectl**

**vi game.properties vi ui.properties**

**enemies=aliens color.good=purple**

**lives=3 color.bad=yellow**

**Now create the configmap from directory**

**Kubectl create configmap my-game-cfg --from-file= configure\_pod/configmaps/k8kubectl/**

**How to display the configmaps in k8?**

**Kubectl get configmaps -o wide**

**How to verify data present in the configmap?**

**Kubectl get configmaps configmap\_name -o yaml**

**Ex🡪 Kubectl get configmaps my-game-cfg -o yaml**

**How to create a configmap from multiple files?**

**Kubectl create configmap map\_name --from-file=1st-file-name --from-file=2nd-file-name**

**How to create a configmap from single file?**

**cat redis-conf**

**maxmemory 2mb**

**maxmemory-policy allkeys-lru**

**Now create the configmap**

**Kubectl create configmap ex-redis-cfg --from-file=redis-conf**

**How to use configmap in pod?**

**We can use configmap in pod in two ways**

1. **Using volume**
2. **Using environment variable**

**How to use configmap in pod using volume?**

**Vi redis-cofigMap-pod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: redis**

**spec:**

**containers:**

**- name: redis**

**Image: Kubernetes/redis:v1**

**vloumeMounts:**

**- mountPath: /redis-master under this path configMap path present**

**name: config**

**volumes:**

**- name: config**

**configMap:**

**name: ex-redis-cfg name of the configmap created using a file**

**items:**

**- key: redis-config Key is nothing but file name created when volume created**

**path: redis.conf path is the location when data will present**

**Now create the pod**

**Kubectl create -f redis-cofigMap-pod.yaml**

**How to check the configmap file in a pod?**

**Kubectl exec pod-name cat volume-path**

**Ex 🡪 Kubectl exec redis cat /redis-master/redis.conf**

**To verify the redis config has applied successfully we can use below command as well**

**Kubectl exec redis redis-CLI**

**How to create a configmap using literal values / using key value pair variable?**

**Kubectl create configmap config-map-name --from-literal=literal-key=value**

**Ex🡪 Kubectl create configmap special-conf --from-literal=special.how=very**

**To use this configmap inside the pod we need to use the key ( in ex special.how is the key )**

**How to use configmap inside pod using environment variable?**

**Under spec section we need to use env section and under env we have to refer the configmap key which we want to use.**

**Vi busyboxTest.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: test-pod**

**spec:**

**containers:**

**- name: test-container**

**Image: k8.gcr.io/busybox**

**Command: [ “/bin/sh”, “-c” , “env” ]**

**env:**

**- name: SPECIAL\_USE\_KEY**

**valueFrom:**

**configMapKeyRef:**

**name: special-conf**

**key: special.how**

**restartPolicy: Never**

**Now create the pod Kubectl create -f busyboxTest.yaml**

**To test / validate the environment variable we can use Kubectl log command ( and we will get the value assigned to the variable name that is SPECIAL\_USE\_KEY=very ).**

**Kubectl log test-pod | grep SPECIAL**

**How to validate the configmaps in k8?**

**Kubectl get configmaps**

**How to delete the configmap?**

**Kubectl delete configmaps configmap-name1 configmap-name2 …….**

**What is secrets in k8?**

**Secrets is a k8 object that contain sensitive data which includes passwords, token or SSH keys.**

**Using secrets, we can reduce the risk of accidental exposure of confidential information while deploying the pod.**

**What is the size of secrets file and where is stored in k8?**

**Size of secrets file is 1MB and it is stored in ETCD. Generally secret is stored in tmpfs so that will restrict access to the application in the node. Another important thing is , as it is stored in tmpfs an application running another container can add access.**

**How will you create secrets in k8?**

**There are two ways to create secretes in k8.**

1. **Using manifest file**
2. **Using Kubectl command line**

**How to create secrets using command line ?**

**Kubectl create secret secret\_type name\_of\_secrete --from-file secrete\_data**

**Or**

**Kubectl create secret secret\_type name\_of\_secrete --from-literal secrete\_data**

**Generic path of the directory**

**File path of the file with file name**

**Directory key value pair**

**Literal value**

**Docker-registry**

**tls**

**How to use multiple file to create secrets in k8?**

**echo –n ‘admin’ > ./username.txt**

**echo –n ‘suman@#345’ > ./password.txt**

**Now create secrets**

**Kubectl create secrets db\_user\_passwd --from-file ./username.txt --from-file ./password.txt**

**How to display the content of a secret?**

**Kubectl describe secret name-of-secret**

**Kubectl describe secret db-user-passwd**

**How to create secret manually using manifest file?**

**First create the encrypted data**

**echo –n ‘admin’ | base64**

**o/p 🡪 YWRtaW4**

**echo –n ‘suman@#345’ | base64**

**o/p 🡪 MWETY345DAV45AK**

**Now create the manifest file**

**Vi mySecrete.yaml**

**apiVersion: v1**

**kind: Secret**

**metadata:**

**name: mysecret**

**type: Opaque**

**data:**

**username: YWRtaW4 Here username is the key and encrypted data is value**

**password: MWETY345DAV48AK password is key and encrypted data is value**

**Now create secret using Kubectl**

**Kubectl create -f mySecrete.yaml**

**How to decode secret data?**

**We need to get the data in yaml format using Kubectl and then decode it using base64**

**Kubectl get secrets mysecret -o yaml it will show the result in yaml format**

**echo ‘YWRtaW4’ | base64 --decode**

**echo ‘MWETY345DAV48AK’ | base64 --decode**

**How to use secrete inside pod in k8?**

**We can use secrete in two ways inside the pod.**

**Using volume**

**Using environment variable**

**How to use secret inside pod using volume?**

**Once the secret is created we can use it in the volume section of the pod.**

**Vi mySecretPod.yaml**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: mysecPod**

**spec:**

**containers:**

**- name: mysecPod**

**Image: redis**

**volumeMounts:**

**- name: foo**

**mountPath: “/etc/foo”**

**readOnly: true**

**volumes:**

**- name: foo**

**Secret:**

**secretName: mysecret**

**now create the pod 🡪 Kubectl create -f mySecretPod.yaml**

**Now check the secret mounted in the pod**

**Kubectl exec mysecPod ls /etc/foo**

**o/p 🡪 password   
 username**

**To check the data inside the secret file mounted in pod**

**Kubectl exec mysecPod cat /etc/foo/username**

**Kubectl exec mysecPod cat /etc/foo/password**

**How to use secret inside pod using environment variable?**

**First create secrets using command line ( from-file / from-literal) or by manifest file then use it in the pod**

**Vi mySecretEnvPod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: secretEnvPod**

**spec:**

**containers:**

**- name: mycontainer**

**Image: redis**

**Env:**

**- name: SECRETUSERNAME**

**valueFrom:**

**secretKeyRef:**

**name: mysecret**

**key: username**

**- name: SECRETPASSWORD**

**valueFrom:**

**secretKeyRef:**

**name: mysecret**

**key: password**

**restartPolicy: Never**

**now create the pod -------- Kubectl create -f mySecretEnvPod.yaml**

**now check the variable passed to the pod manifest file**

**Kubectl exec secretEnvPod env | grep SECRET**

**o/p 🡪 SECRETUSERNAME = admin**

**SECRETPASSWORD = suman@#345**

**How to delete secret in k8?**

**Kubectl delete secret secret\_name1 secret\_name2**

**What is replication control in k8? What is replica set in k8?**

**Replication controller ensures that a specified number of pod replicas are running always.**

**If we define there should be five pods running at any time using replication controller in the manifest file, then replication controller will make sure these five parts are running always. The replication controller will terminate the extra pods if more pods are running than what is defined same time replication controller will start more pods if less pods are running than what is defined.**

**Replication controller is often abbreviated as RC or RCS.**

**Replica set does the same as replication controller. ( so read the above definition replacing replication controller with replica set )**

**How replication control / replica set manages number of pod active?**

**Using label and selectors replication controller are aware of the pods that it needs to manage.**

**What is Label in k8?**

**Labels are nothing but a tag that was given to Pod.**

**What is the advantage of replication controller / replica set?**

**High availability 🡪 It ensures that a specified number of pod replicas are running always ( if port or node get inactive )**

**Load balancing 🡪 When number of user increase, we can change the replica count to server the users.**

**Create a manifest file to manage pods, when Node fails will the replication controller will recreate new pods on a healthy node inside the k8 cluster?**

**Vi nginxRc.yaml**

**apiVersion: v1**

**kind: ReplicationController**

**metadata:**

**name: nginx-rc**

**spec:**

**replicas: 3**

**selector:**

**app: nginx-app same label under selector for RC**

**template:**

**metadata:**

**name: nginx-pod**

**labels:**

**app: nginx-app**

**spec:**

**containers:**

**- name: nginx-container pod defination**

**Image: nginx**

**Ports:**

**- containerPort: 80**

**Now deploy the replication controller**

**Kubectl create -f nginxRc.yaml**

**When we execute Kubectl get command we will get three nginx pods created**

**Kubectl get pods -o wide 🡪 if any node dies then the new pod will run on another node.**

**If there are more pods running in k8 cluster then how we will get these pods?**

**Using label in kubectl get command we can find the pods**

**Kubectl get pods -l app=nginx-app**

**How to print the complete details of replication controller?**

**Kubectl describe rc replicatonContoller\_name**

**Ex🡪 Kubectl describe rc nginx-rc**

**How to manage traffic as per the user access to the application ?**

**When traffic is more/less then we can scale up/down the replicas for the pod by specifying the number of replicas need to run to manage the load.**

**Kubectl scale rc replicatonContoller\_name --replicas=no**

**Ex 🡪 kubectl scale rc nginx-rc --replicas=5**

**How to delete the objects created by replication controller manifest file?**

**Kubectl delete -f ReplicationController\_filename**

**Ex 🡪 kubectl delete -f nginxRc.yaml**

**What is the difference between replication controller and replica set?**

**replication controller is old, it is replaced by replica set.**

**replication controller supports equality-based selectors whereas replica set supports Set based selectors.**

**What is the difference between equality-based selectors and Set-based selectors?**

|  |  |
| --- | --- |
| **Equality-based selector** | **Set-based selector** |
| **Equality-based selector uses = and != operators** | **Set-based selector uses in , notin and exists operators** |
| **Example environment = production  tier != frontend** | **Example environment in (production , qa)  tier notin (frontend, backend)** |
| **For equality-based selector kubectl command will be as below. Kubectl get pod -l environment = production** | **For set-based selector kubectl command will be as below. Kubectl get pod -l environment in (production)** |
| **In replication controller we have to mention the label in from of key value pair under selector section  selector:  environment: production  tier: frontend** | **In replication controller we have to mention the label  under matchExpressions in selector section  selector:  matchExpressions:  - { key: environment, operator: in , values: [pod, qa]}  - {key: tier, operator: Notin, values: [frontend, backend]}** |
| **As ReplicationController is the older one we can use selector:  environment: production  tier: frontend** | **But in case of replica set we need to use matchLabels  selector:  matchLabels:  environment: production  tier: frontend** |
| **Equality-based selector is simple and easy to use but not powerful.** | **Set-based selector is complex but more powerful than equality-based selectors.** |

**Note 🡪 The old resources such as ReplicationController and services we no need to use matchLabels. But for the new resources such as ReplicaSets, Deployment, job and DemonSet we need to use matchLabels.**

**When will you use matchLabels and when will you use matchExpressions?**

**when there is a key and for that key we have only one value we should use matchLabels. When there is a key and for that key we have set of values we should use matchExpressions.**

**How to create a replica set manifest file?**

**Vi myReplicaSet.yaml**

**ApiVersion: app/v1**

**Kind: ReplicaSet**

**Metadata:**

**Name: nginx\_rs**

**Spec:**

**Replicas: 3**

**Selector:**

**matchLabels:**

**app: nginx-app**

**matchExpressions:**

**- {key: tier, operator: In, values: [frontenc] }**

**Template:**

**Metadata:**

**Name: nginxProd**

**Labels:**

**app: nginx-app**

**tier: frontend**

**Spec:**

**Containers:**

**- name: nginx-container**

**Image: nginx**

**Ports:**

**- containerPort: 80**

**Now let’s go and create a replica set**

**Kubectl create -f myReplicaSet.yaml**

**Get the pods created by replica set**

**Kubectl get pods**

**Get the pods which are created an having the label frontend**

**Kubectl get pods -l tier=frontend**

**To get the selector detail for the replica set**

**Kubectl get rs replica\_set\_name -o wide**

**Ex 🡪 kubectl get rs nginx\_rs -o wide**

**To get complete detail of the replica set**

**Kubectl describe rs replica\_set\_name**

**Ex 🡪 kubectl describe rs nginx\_rs**

**How to pods are rescheduled by replica set when the underlying node dies?**

**If the node will be down at that time the pod will get assigned to the active node to maintain the replica count mention in the manifest file.**

**How to scale up when there is an increase in load and scale down when there is no load using replica set?**

**If traffic increase/decrease for end users, then we can increase/decrease the application instances based on the load.**

**Kubectl scale rs replica\_set\_name --replicas=<number> 🡪 it will run the specified no of RS**

**How to delete the replica set?**

**Kubectl delete -f replica\_set\_manifest\_filename 🡪 it will delete pod and replica set**

**Ex 🡪 kubectl delte -f myReplicaSet.yaml**

**What is deployment in k8?**

**Generally, we deploy microservices applications through containers. related containers (more than one container) are grouped together inside a pod. And replica set create multiple replicas of pods for high availability and load balancing.**

**But if we want to change older version number inside the spec file to newer version or want to increase or decrease number of pod replicas. K8 provides an object know as deployment, by using single deployment manifest file we can achieve all these.**

**Deployment manifest file contains the pod definition number of pod replicas and contain preferred upgrade strategy you want.**

**What is the advantage in deployment in k8?**

**If you don’t mention replicas in the deployment, manifest file deployment will create replica set with a count of one. So, advantage of having replica set with count one is if something happens to this pod and it dies, it will try to recreate it again on the same node or a different healthy node.**

**Using deployment controller, we can upgrade the application.**

**If anything goes wrong with the current upgrade, then we can rollback it using deployment.**

**Based on the load you can update replica field and deployment spec file accordingly in the manifest file to scale up and scale down the application instances.**

**Suppose you are performing some deployment and want to test some thing then You can pause the deployment process in progress as and when needed.**

**How many types of deployment you can do using deployment control? How many types of upgrades in deployment?**

**There is two type of deployments.**

**Restart 🡪 In restart we will shutdown the running version and once running version is turned off we will deploy the new version. But to perform restart deployment there will be a downtime.**

**Rolling update 🡪 Rolling Update deployment will slowly roll out a version of an app by replacing instances one after the other until all the instances are successfully rolled out.**

**canary deployment 🡪 Canary deployment consists of gradually shifting production traffic from older version to new version. Let’s imagine there is 10 instance of an app is running. You want to upgrade the app from version-A to version-B. By using canary deployment you can upgrade 2 instance of app version-A to version-B test the result. Once everything looks good you can upgrade remaining 8 instances of app version-A to version-B.**

**blue green upgrade strategy 🡪 In blue green upgrade strategy the new version ( version-B) , which is green, is deployed alongside the old version ( version-A) , which is blue with exactly same amount of instances. Once the testing of new version ( version-B) is success then the traffic is switched from old version to new version. Advantage of this is instant rollout and rollback.**

**Rolling update is a default update strategy in Kubernetes.**

**Create a deployment manifest file?**

**Vi myNginxDeploy.yaml**

**apiVersion: app/v1**

**kind: deployment**

**metadata:**

**name: nginxDeploy**

**labels:**

**app: nginx\_app**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: nginx\_app**

**template:**

**metadata:**

**labels:**

**app: nginx\_app**

**spec:**

**containers:**

**- name: nginx-container**

**Image: nginx1.7.9**

**Ports:**

**- containerPort: 80**

**Now create the deployment**

**Kubectl create -f myNginxDeploy.yaml**

**Display the deployment created**

**To check the deployment, we need to filter the label from kubectl get command.**

**Kubectl get deploy -l app: nginx\_app**

**Now check the replica set created by the deployment**

**To check the replica set created by the deployment we can filter the label from kubectl get rs command.**

**Kubectl get rs -l app: nginx\_app**

**Now check the pods created by deployment**

**Kubectl get pod -l app: nginx\_app**

**Now display the complete detail of deployment**

**Kubectl display deploy nginxDeploy 🡪 Imp thing we will see strategyType is rolling update**

**How to update/upgrade current version of application to a new version in deployment?**

**We can upgrade current version to new version in two different ways.**

**Kubectl set command**

**Kubectl edit command**

**How to upgrade using kubectl set command?**

**Kubectl set image deploy name\_of\_deployment name\_of\_container=new-version --record**

**Ex 🡪 kubectl set image deploy nginxDeploy nginx-container= nginx1.9.1 --record**

**How to upgrade using kubectl edit command?**

**When you use kubectl Edit deploy command Configuration of the deployment will open in VI editor. Then you can change the replica count, image version and other configuration and close it as soon as you close it. Configuration will get automatically updated.**

**Kubectl edit deploy name\_of\_deployment**

**Ex 🡪 kubectl edit deploy nginxDeploy**

**How to check the status of the deployment?**

**Kubectl rollout status name\_of\_deployment**

**Ex 🡪 kubectl rollout status nginxDeploy**

**How to check the deployments in k8?**

**Kubectl get deploy**

**To display specific deployments**

**Kubectl get deploy name\_of\_deployment**

**Ex 🡪 kubectl get deploy nginxDeploy**

**How you can roll back the deployment?**

**Lets say we want to upgrade nginx1.7.1 to nginx1.9.1 version. But accidentally we deployed nginx1.91 version using below command.**

**Kubectl set image deploy nginxDeploy nginx-container= nginx1.91 --record**

**But you will get the output “ image updated ” which is misleading.**

**So to verify the deployment status we need to use below command.**

**Kubectl rollout status nginxDeploy**

**The output will be “ it is waiting for something “**

**So to verify what command you executed for the upgrade we need to use below command.**

**Kubectl rollout history nginxDeploy**

**From the output we will get the mistake done from the user and we will roll back the upgrade**

**Kubectl rollout undo nginxDeploy**

**Now check the rollout status again and we will see nginx rollout successfully with its previous stable version.**

**Kubectl rollout status nginxDeploy**

**How to scale up and scale down the pod instances in deployment?**

**If sudden increase in traffic from end users accessing your website. We need to scale up app instances using the below command.**

**Kubectl scale deployment name\_of\_deployment --replicas=no**

**Ex🡪 kubectl scale deployment nginxDeploy --replicas=5 ----- we will get 5 instances running**

**How to check the no of instances are running for the deployment?**

**Kubectl get deploy name\_of\_deployment**

**Ex 🡪 kubectl get deploy nginxDeploy**

**How to scale down the app instances in deployment?**

**Once the load and the traffic from outside world is back to normal, we can bring down the replica count using same kubectl scale deployment command.**

**Kubectl scale deployment name\_of\_deployment --replicas=no**

**Ex 🡪 kubectl scale deployment nginxDeploy --replicas=2**

**Now you can check the no of instance running for the deployment**

**Kubectl get deploy nginxDeploy --------- you can see two instance will be running**

**How to delete the deployment and the objects created by the deployment?**

**Kubectl delete -f deployment\_manifest\_filename ------ it will delete pods,**

**replica set and deployments.**

**Ex🡪 kubectl delete -f myNginxDeploy.yaml**

**Why DemonSet in k8?**

**When we use replica set, we can scale up and scale down the no of pods. But the scheduler takes the control and schedule the pod inside the Kubernetes cluster. So multiple pods can be schedule in one node due to replica set.**

**Think we have a monitoring app and we need to schedule it in all the nodes. But by the replica set it is not possible. That’s why we need DemonSet.**

**What is DemonSet in k8?**

**DemonSet basically ensures that all or required nodes inside the cluster runs a copy of a pod.**

**How DemonSet is useful?**

**When node is added to the cluster DemonSet add the pod to that node.**

**As node is removed from the cluster those pods are deleted and goes to garbage collected.**

**Delete the DemonSet from master will delete the pod from the nodes.**

**What is the use of the DemonSet?**

**We can use DemonSet to monitor the nodes that is known as collectd**

**DemonSet can be used for collecting logs that is known as fluentd**

**Using DemonSet we can create storage daemons known as ceph**

**How to write a DemonSet manifest file?**

**Vi myFluentdDemonSet.yaml**

**apiVersion: apps/v1**

**kind: DemonSet**

**metadata:**

**name: fluentd-ds**

**spec:**

**template:**

**metadata:**

**labels:**

**name: fluentd**

**spec:**

**containers:**

**- name: fluentd**

**Image: gcr.io/google-containers/fluentd-elasticsearch:1.20**

**Selector:**

**matchLabels:**

**- name: fluentd**

**Let’s deploy the DemonSet controller**

**Kubectl create -f myFluentdDemonSet.yaml**

**To check the specific DemonSet running on the cluster.**

**Kubectl get ds name\_of\_demonSet**

**Ex 🡪 kubectl get ds fluentd-ds**

**To confirm each node has a DemonSet pod running**

**Kubectl get pod -o wide**

**To check completed detail of the DemonSet**

**Kubectl describe ds name\_of\_demonSet**

**Ex 🡪 kubectl describe ds fluentd-ds**

**How to delete the DemonSet?**

**Kubectl delete ds name\_of\_demonSet**

**Ex 🡪 kubectl delete ds fluentd-ds**

**Once we delete the DemonSet, it will delete the pod which is create by the DemonSet.**

**For any reason if pod dies and if DemonSet is active then the DemonSet will make sure the pod is created again.**

**What is job in k8?**

**Job is a controller which supervise pods are carrying out certain tasks.**

**What is the use of jobs? How many types of jobs in k8?**

**There are two types of job in k8.**

**One task run to completion is known as run till completion job/job**

**Run a task in particular time is known as cron job**

**What is run till completion job?**

**Run till completion job run a task till the end until it gets the exist status 0. Once the exit status is success then the pod went shutdown and not consume any resource. But the pod will not be deleted we need to delete it manually.**

**What is cron job?**

**The job run at a particular time and perform the assigned task.**

**What is the used of job?**

**Depending on job manifest file k8 api create one or more pods**

**Job ensures the pods get terminated successfully. That means the container inside the pod should return an exist code zero.**

**Once job completed the pod will not deleted automatically, so it is easy to view logs, check for errors, warnings, or other diagnostic output.**

**Using job control logic created by the developer the long running job resume its run if pod get deleted due to node filter or some other issue when the issue gets fixed.**

**Job can run multiple pods in parallel. For example, Let's say you are processing a lot of messages flowing into rabbitmq, to process that you need multiple pods running parallel.**

**Write a job manifest file?**

**Vi myCountDownJob.yaml**

**apiVersion: batch/v1**

**kind: job**

**metadata:**

**name: countdown**

**spec:**

**template:**

**metadata:**

**name: countdown**

**spec:**

**containers:**

**- name: counter**

**Image: centos:7**

**Command:**

**- “bin/bash”**

**- “-c”**

**- “for I in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done”**

**restartPolicy: Never**

**How to create the job from manifest file?**

**Kubectl create -f myCountDownJob.yaml**

**How to check the jobs in k8?**

**Kubectl get jobs**

**How to check the job output?**

**As pod will be deleted after the job completed we can use below command to get the job output**

**Kubectl get pod ---- give the pod name and use the pod name to get the output of the job**

**Kubectl logs pod\_name**

**Ex 🡪 kubectl logs countdown**

**How to get the complete description of the job?**

**Kubectl describe jobs jobname -------- it will show start, end time and events**

**Ex 🡪 kubectl describe jobs countdown**

**How to delete the job?**

**Kubectl delete jobs job\_name**

**Ex 🡪 kubectl delete jobs countdown ------ delete job delete the job object and pod created by it.**

**How to create cronjob manifest file in k8?**

**Vi myCronJob.yaml**

**apiVersion: batch/v1**

**kind: CronJob**

**metadata:**

**name: hello minute (0 - 59)**

**spec: hour (0 - 23)**

**schedule: "\* \* \* \* \*" day of the month (1 - 31)**

**jobTemplate: month (1 - 12)**

**spec: day of the week (0 - 6) (Sun to Satur)**

**template:**

**spec:**

**containers:**

**- name: hello**

**image: busybox:1.28**

**imagePullPolicy: IfNotPresent**

**command:**

**- /bin/sh**

**- -c**

**- date; echo Hello Kubernetes cluster**

**restartPolicy: OnFailure**

**Why is service required?**

**Let’s say there is some pod which interact between them to achieve the goal. But we know due to different reasons pod dies. When the pod recreate ( using replicaSet , deployment ) it get new IP address.**

**So, problem here is how one pod keep track of other pod (For example a frontend pod how communicates to backend pod if backend prod is recreated).**

**App is made up of multiple microservices. Each of the microservice is deployed into their individual pod need to be connected and communicated together.**

**How do these apps are exposed to end users who are trying to access the apps from Internet?**

**To solve these three problems, we need services.**

**What is service in k8?**

**Service is a standardized way of grouping of pods that are running on the cluster.**

**Let’s think we have two pods; one is frontend pod and other is backend pod/ database pod. The end users will access the frontend pod not the backend pod that means we need to expose frontend pod for the users. But frontend pod needs to communicate to backend pod. So, in this case we need two services. One will be frontend service which will be responsible to establish communication between user and frontend pod. Other will be backend service which is interact between frontend and backend pod.**

**What are the features of services?**

**Services provide some of the important features across the cluster, such as load balancing, service discovery between apps and features to support zero downtime app deployments.**

**How do these services discover their respective pods and connect to it?**

**service spec file Use same label present in pod Labels and selectors.**

**what are the different types of services in Kubernetes?**

**cluster IP service**

**node port service**

**Load balancer service**

**what is cluster IP service?**

**The cluster IP service is reachable within the cluster. For example, you want to connect front end pods to the back-end pods, in such cases, your cluster IP comes into the picture.**

**What is node port service?**

**The pods that you need to expose to the outside (frontend pods ) world then you have to use node port.**

**What is load balancer service?**

**Think when front end app instances are distributed across a couple of nodes inside your Kubernetes cluster. Then it is difficult to identify node which should be used to access the app. Also, we cannot overload one specific node.**

**So to maintain proper load balance across the node we need load balance service.**

**Why node port service?**

**Think we have a frontend web app pod which is deployed in a node. But when user from outside try to access the frontend application they have two challenges.**

**First one is when pod dies, it will be recreated by controllers with another IP address.**

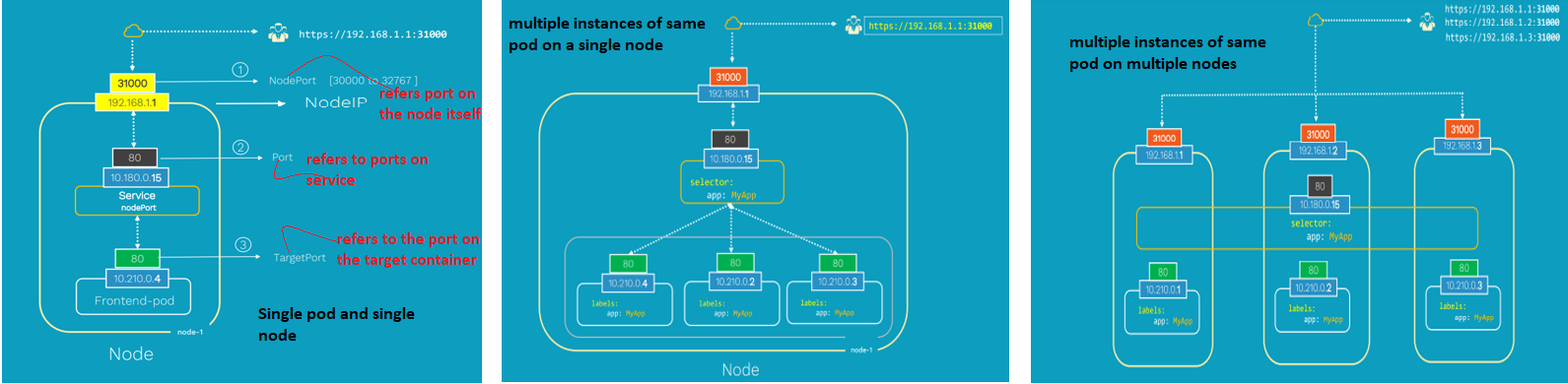
**Second one is, there is no connectivity between users and Pod.**

**To solve these two problems, we need node port service.**

**What is node port service?**

**To expose this application to outside world on the internet, we need to create a node port service.**

**When the Nodeport service is deployed, the application is exposed on node IP along with node port which is mentioned in the manifest file. Node port range comes between 30,000 to 32 767. If you don't mention nodeport specifically in the manifest file, then Kubernetes will assign unused node port within that range dynamically.**

****

**whether it be a single pod and single node or multiple pods on a single node or multiple pods on multiple nodes, the service is created exactly the same without any other additional steps during the service creation.**

**What is the dis-advantage / downside of nodeport service?**

**You have only one service per port.**

**you can only use your ports between 30,000 and 32 767.**

**if your node or VM IP address changes, you need to deal with that.**

**So, it is not really recommended to use this method in production to directly expose your app.**

**How to create manifest file for nodeport service?**

**To create nodeport service we have three basic need they are node port, port and target port.**

**Nodeport is a port on the node itself. The port refers to the ports on service. And target port refers to the port on the target container where actual web app is running.**

**Let's assume we have deployed this Nginx web app using deployment object. Now we need to expose this Nginx app to the outside world, we need a service of type node port.**

**Step 1 🡪 create a deployment manifest file nginx-deploy.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**labels:**

**app: nginx-app**

**spec:**

**replicas: 1**

**selector:**

**matchLabels:**

**app: nginx-app**

**template:**

**metadata:**

**labels:**

**app: nginx-app**

**spec:**

**containers:**

**- name: nginx-container**

**Image: nginx:1.7.0**

**Ports:**

**- containerPort: 80**

**Step 2 🡪 create service manifest file nginx-svc-nodeport.yaml**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: my-service**

**labels:**

**app: nginx-app**

**spc:**

**selector:**

**app: nginx-app**

**type: NodePort**

**ports:**

**- nodePort: 31000 ( node port )**

**Port: 80 ( service port )**

**targetPort: 80 ( the port no mention is deployment manifest file )**

**now create the deployment**

**kubectl create -f nginx-deploy.yaml**

**now create the service**

**kubectl create -f nginx-svc-nodeport.yaml**

**how to print all the services inside the Kubernetes cluster?**

**Kubectl get service -l app=nginx-app 🡪 output will show the service IP and the port**

**Lets, check the ip and port where pod is running**

**Kubectl get pod -o wide**

**Note 🡪 it is confirmed that there is one port is created as part of deployment and it is running successfully**

**How to get complete details about the service in k8?**

**Kubectl describe svc my-service**

**How to validate the service which you configure is working fine?**

**If you will get the response from**

**Step 1. the pod ip with the service port it will be confirm page is accessible.**

**Kubectl get pod -o wide give the pod ip and kubectl describe svc service\_name give the service port**

**Curl <http://pod-IP:servicePort>**

**Step 2. By accessing service IP and service port**

**Kubectl get svc -l app=nginx-app it will give the service IP and port**

**Curl** [**http://service-IP:servicePort**](http://service-IP:servicePort)

**Step 3. By accessing the node ip and nodeport in the browser**

**Kubectl get pod -o wide give the node where the pod is running. Now to get that node IP in google cloud we can use gcloud compute instances list**

**Now on the browser use external-node-ip:nodePort if the page is opening fine**

**Then you can say your services is running fine.**

**How to delete service in k8?**

**Kubectl delete svc service-name ----- service has reference to pod label so pod will be automatically deleted if you delete service**

**Ex 🡪 kubectl delete sve my-service**

**Why load balancer service in k8?**

**Let’s say we have multiple instances of same pod in multiple nodes. Now we have some question comes to our mind.**

**if you had to provide access to the end users, which node IP will you provide?**

**are these end users comfortable using IP and port numbers to access this app?**

**How is the traffic balanced equally among all nodes inside the cluster?**

**So, to solve these queries we need load balancer service.**

**what is a load balancer service?**

**Load Balancer service is a standard solution to expose your app onto the Internet.**

**So, when you create load balancer service, it creates public IP which will forward all the traffic to your service and you can send Http, TCP, UDP and all different kinds of traffic.**

**For cloud like GCD, AWS, Azure,  you need to think twice before you create load balancers because they are not cheap they are very expensive.**

**Note 🡪 we can use Ingress instead of load balancer service for cost effective ness. Also, Ingress is more secure than the services.**

**How to create load balancer service manifest file?**

**Step 1. Create the deployment file nginx-deployment.yaml**

**apiVersion: apps/v1**

**kind: deployment**

**metadata:**

**name: nginx-deploy**

**labels:**

**app: nginx-app**

**spec:**

**replicas: 1**

**selector:**

**matchLabels:**

**app: nginx-app**

**template:**

**metadata:**

**labels:**

**app: nginx-app**

**spec:**

**containers:**

**- name: nginx-container**

**Image: nginx:1.7.0**

**Ports:**

**- containerPort: 80**

**Step 2 🡪 create load balancer service manifest file nginx-service-lb.yaml**

**apiVersion: v1**

**kind: service**

**metadata:**

**name: my-service**

**labels:**

**app: nginx-app**

**spec:**

**selector:**

**app: nginx-app**

**type: LoadBalancer**

**ports:**

**- nodePort: 31000**

**Port: 80**

**targetPort: 80**

**Lets create the deployment --- kubectl create -f nginx-deploy.yaml**

**Now create the service ----- kubectl create -f nginx-service-lb.yaml**

**Display the load balancer service ----- kubectl get service -l app=nginx-app ( you can see type is load balancer, port service IP/cluster IP and External IP ( which you use in browser http://extenal IP**

**) )**

**How to get the complete detail of load balancer service?**

**Kubectl describe service service-name**

**Ex 🡪 kubectl describe service my-service**

**How to delete load balancer service?**

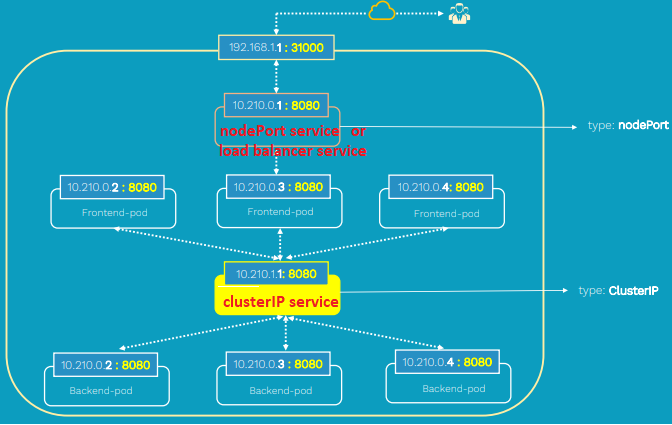
**Kubectl delete service service-name**

**Ex 🡪 kubectl delete service my-service ------ it will delete load balancer service as well as the pods as it refer to the same labels**

**how can we restrict access of backend database to only within Kubernetes cluster? / which service will restrict access only within k8 cluster?**

**Using cluster IP service, we can restrict access only within k8 cluster.**

**Let's assume that you want to deploy one full-fledged application which consists of frontend app and the backend database. As data is sensitive, we cannot provide direct access of database to the outside world, only we can access the data in front end wherever it is required. In that case we can use cluster IP service.**

****

**Explain an end to end deployment and service flow for an application where end user can access it from outside world?**

**Let’s, deploy a Guestbook application where it has a front-end web app and back-end as Redis database which contains the data and cannot expose to outside world directly. So, we need to create few components inside k8 cluster so that the user can access the data through the app on internet.**

**Deploy Redis DB master**

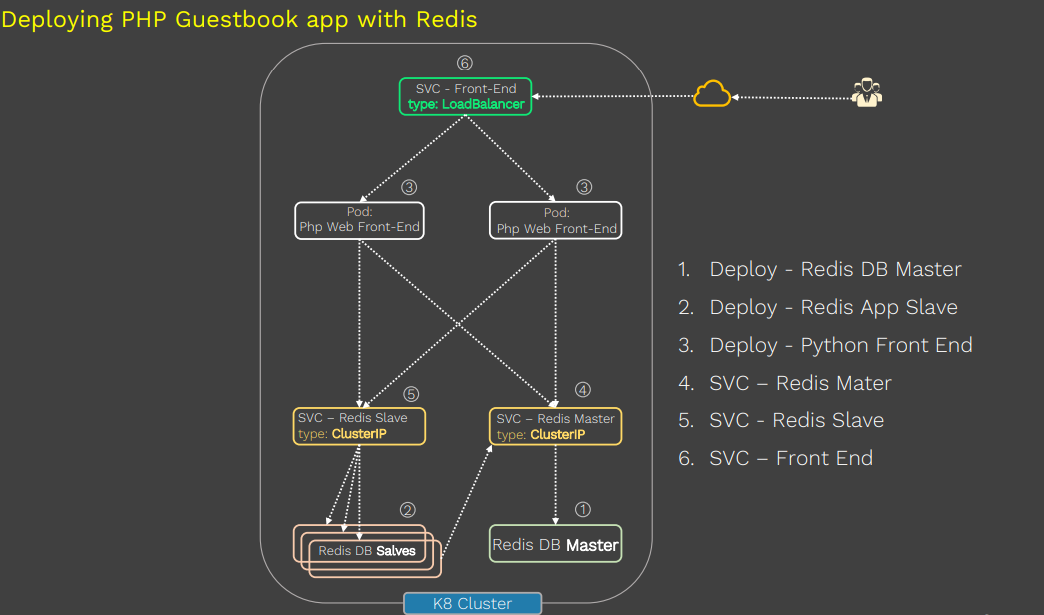
**Deploy Redis Slave**

**Deploy python front end ( your application)**

**Here we are creating six pods which include one DB master, three DB slaves and two front-end app pod instances. To connect these components, we need services.**

**cluster IP  service to connect front end web app to the back end database**

**Load balancer service to expose front end to outside world.**

****

**Now create Redis master deployment manifest file redis-master-deployment.yaml**

**apiVersion: apps/v1**

**kind: deployment**

**metadata:**

**name: redis-master**

**labels:**

**app: redis**

**spec:**

**replicas: 1**

**selector:**

**matchLabels:**

**app: redis**

**role: master**

**tier: backend**

**template:**

**metadata:**

**labels:**

**app: redis**

**role: master**

**tier: backend**

**spec:**

**containers:**

**- name: master**

**Image: k8s.gcr.io/redis:e2e**

**Resources:**

**Requests:**

**Cpu: 100m**

**Memory: 100Mi**

**Ports:**

**- containerPort: 6379**

**Execute kubectl command to create master pod**

**Kubectl create -f redis-master-deployment.yaml**

**Now create Redis slave manifest file redis-slave-deployment.yaml**

**apiVersion: apps/v1**

**kind: deployment**

**metadata:**

**name: redis-slave**

**labels:**

**app: redis**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: redis**

**role: slave**

**tier: backend**

**template:**

**metadata:**

**labels:**

**app: redis**

**role: slave**

**tier: backend**

**spec:**

**containers:**

**- name: slave**

**Image: gcr.io/google\_samples/gb-redisslave:v1**

**Resources:**

**Requests:**

**Cpu: 100m**

**Memory: 100Mi**

**Ports:**

**- containerPort: 6379**

**Execute kubectl command to create slave pod**

**Kubectl create -f redis-slave-deployment.yaml ------ it will create 3 replicas**

**Now we have master Redis DB and slave is crated.**

**Now Let’s create the frontend deployment manifest file guestbook-frontend.yaml**

**apiVersion: apps/v1**

**kind: deployment**

**metadata:**

**name: frontend**

**labels:**

**app: guestbook**

**spec:**

**replicas: 2**

**selector:**

**matchLabels:**

**app: guestbook**

**tier: frontend**

**template:**

**metadata:**

**labels:**

**app: guestbook**

**tier: frontend**

**spec:**

**containers:**

**- name: php-guestbook-page**

**Image: gcr.io/google-samples/gb-frontend:v4**

**Resources:**

**Requests:**

**Cpu: 100m**

**Memory: 100mi**

**Ports:**

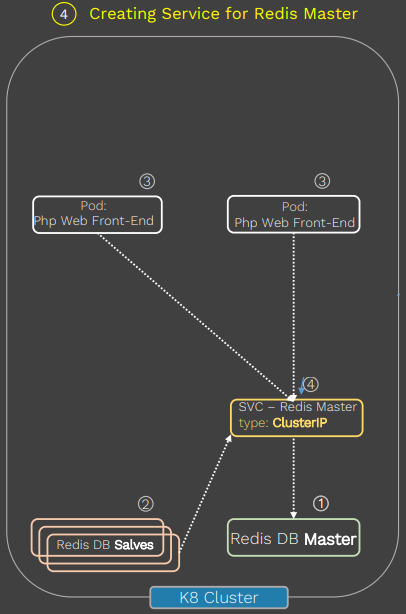
**- containerPort: 80**

**Execute kubectl command to create frontend pods**

**Kubectl create -f guestbook-frontend.yaml**

**Now we need to create the services like redis DB master, redis DB slave and frontend. The redis DB master, redis DB slave service will connect the frontend pods to the backend database pods using cluster IP service. Because it connects and operate inside k8 cluster. Also, to expose the frontend app to the outside world on the Internet, then we need to create the service of type load balancer service.**

**Let’s create the manifest file redis-db-master.yaml where the service type is clusterIP**

** apiVersion: v1**

**kind: service**

**metadata:**

**name: redis-master-db-svc**

**labels:**

**app: redis**

**role: master**

**tier: backend**

**spec:**

**selector:**

**app: redis**

**role: master**

**tier: backend**

**type: ClusterIP**

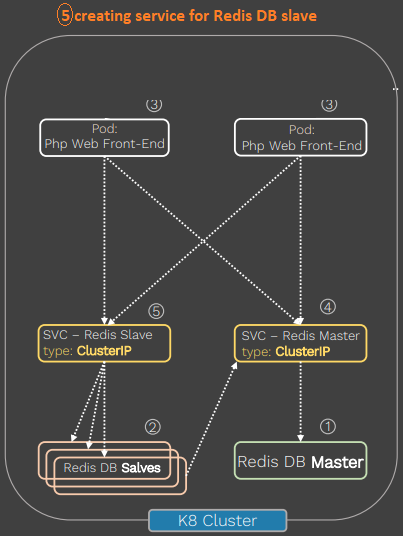
**ports:**

**- port: 6379**

**targetPort: 6379**

**kubectl create -f redis-db-master.yaml will create the service for redis DB master pod**

**Let’s create the manifest file redis-db-slave.yaml where the service type is clusterIP**

** apiVersion: v1**

**kind: service**

**metadata:**

**name: redis-slave-db-svc**

**labels:**

**app: redis**

**role: slave**

**tier: backend**

**spec:**

**selector:**

**app: redis**

**role: slave**

**tier: backend**

**type: ClusterIP**

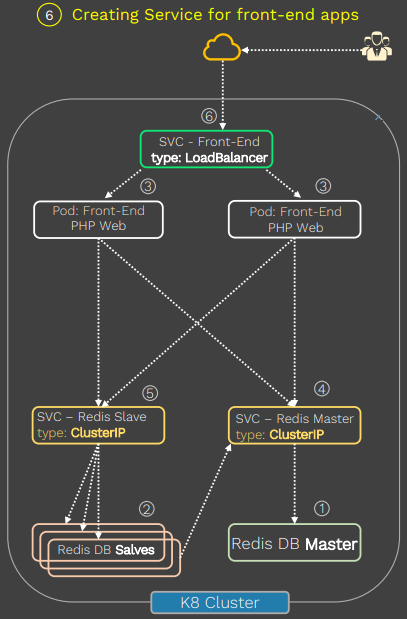
**ports:**

**- port: 6379**

**kubectl create -f redis-db-slave.yaml will create the service for redis DB slave pod**

**To expose the frontend service we need to create a service of loadBalancer type**

**Let’s create a Load balancer service manifest file redis-frontend-lb-svc.yaml**

** apiVersion: v1**

**kind: service**

**metadata:**

**name: fontend-svc**

**labels:**

**app: guestbook**

**tier: frontend**

**spec:**

**selector:**

**app: guestbook**

**tier: frontend**

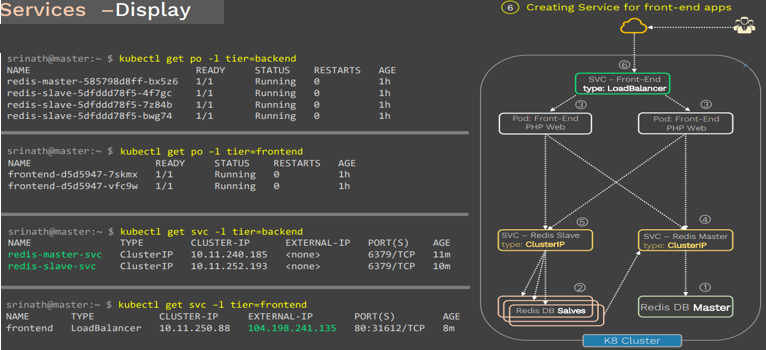
**type: LoadBalancer**

**ports:**

**- port: 80**

**Now execute kubectl create -f redis-frontend-lb-svc.yaml to create the frontend service.**

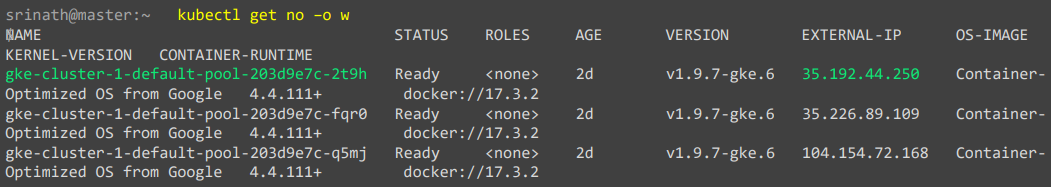
**Let’s display the pods and services by using kubectl get pod -l and kubectl get svc -l respectively.**

****

**To access the app from outside by** [**http://service-eternal-IP**](http://service-eternal-IP) **( <http://104.198.214.135> )**

**Or we can use** [**http://node-ip:expose-port**](http://node-ip:expose-port) **( <http://35.192.44.250:31612> )**

**So to get the node IP we can use kubectl get nodes -o wide**

****

**Note 🡪 if you are not running your setup on any cloud platform, then you can run the service type as nodeport instade of LoadBalancer.**

**Why you need storage volume? / How can data persist beyond pod lifecycle and how do these containers within same pod share data among them? / How do you handle application data running inside pods?**

**Using storage volume, the data can persist beyond prod lifecycle and containers within same pod can share data among them. Also, when we have any application which deals with data known as stateful apps and to handle application data running inside pod using Storage volume.**

**What is the advantage of volume in k8 over Docker volume?**

**In k8 the volume is attached to the pod whereas in Docker the volume is attached to container. when the containers get restarted in Docker, data inside the volume will be erased whereas, in pod data is preserved across the container restarts.**

**Once or more containers can be present in a pod, as volume attach to the pod all the container inside the pod can access the volume.**

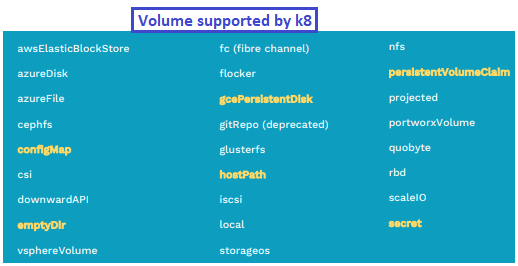
**what are the different categories of volume in k8?**

**There is two types of volume in k8.**

**Ephemeral 🡪 the volume has same life time as pod , It will be created when pod started and erased when pod dies.**

**Durable 🡪 These volumes get created when the pod is created, but data is persisted inside the volume even after the pod dies. So we can use this volume in another new pod to make use of the data which is existed already and create new data on top of that volume.**

**What are the volume type supported by k8?**

****

**Let’s discuss few of them.**

**What is emptyDir volume?**

**When Kubernetes creates the emptyDir volume on a worker node where pod is scheduled then containers inside that pod can write and read the data from this volume. Data inside that stays as long as the pod is alive. if the pod dies or removed from a worker node where it is scheduled inside the Kubernetes cluster, then the emptyDir volume is deleted and the data inside the volume is completely wiped off forever.**

**The primary use of this emptyDir volume is for temporary space and to share the data between the multiple containers in same pod.**

**What is hostPath volume?**

**In case of hostPath volume one or more directory on worker node used as the volume inside the pod. So the hostPath volume will remain even after the pod is terminated.**

**when the pods are scheduled on multiple worker nodes inside the Kubernetes cluster, then Kubernetes will create the Hostpath volume on each of these worker node and having their own exclusive hostpath volume, which may not be in sync. And every pod will deal with the storage very differently. So until and unless you have a very specific requirement, don't use the hostpath.**

**If the hostPath volume is created on NFS then it is perfectly valid instead of to a specific node.**

**what is GCE persistent disk volume?**

**The contents inside the GCE persistent disk are preserved and the volume is just merely unmounted when the pod is removed or dies for any reason.**

**GCE persistent disk can only be mounted by a single pod in read write mode. Not the simultaneous writes are allowed.**

**This restriction is similar to other cloud storage disk such as Elastic Block Store and Azure disk. The restrictions are**

**You must create a persistent disk using gcloud or by GCE API or UI before you use it**

**The node on which pod are running that must be GCE VM**

**Those VMs need to be in the same GCE project and zone as persistent disk**

**Write an emptyDir volume manifest file test-emptyDir.yaml?**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: test-ed**

**spec:**

**containers:**

**- image: k8s.gcr.io/test-webserver**

**name: test-container**

**volumeMounts:**

**- name: cache-volume**

**mountPath: /cache**

**volume:**

**- name: cache-volume**

**emptyDir: {}**

**create the pod using kubectl create -f test-emptyDir.yaml**

**to get the pods kubectl get pod -o wide**

**To check the mount path we can use kubectl exec test-ed df /cache**

**To get completed detail of the pod by using kubectl describe pod test-ed**

**When we delete the pod the volume will be deleted kubectl delete pod test-ed**

**Write an hostPath volume manifest file test-hostPath.yaml?**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: test-redis-hp**

**spec:**

**containers:**

**- name: redis-container**

**image: redis**

**volumeMounts:**

**- name: test-hostpath-volume**

**mountPath: /test-mnt ------------- mount point inside pod**

**volume:**

**- name: test-hostpath-volume**

**hostPath:**

**path: /test-vol -------------- path in node**

**create the pod using kubectl create -f test-hostPath.yaml**

**to get the pods kubectl get pod -o wide**

**To check the mount path we can use kubectl exec test-redis-hp df /test-mnt**

**Test 1 🡪**

**To test the directory/path in the node mounted to path present in pod, Let’s create a file in the node path ( /test-vol) and when we will check the file will be available in pod mount path.**

**[root@nodeXX ~ ]# cd /test-vol**

**[root@nodeXX test-vol ]# echo “in the host” > from-host.txt**

**Now check the pod mount path kubectl exec test-redis-hp ls /test-mnt -------- o/p from-host.txt**

**To get the content inside pod kubectl exec test-redis-hp cat /test-mnt/from-host.txt**

**Test 2 🡪**

**Let’s create a file in the pod**

**Kubectl exec test-redis-hp -it -- /bin/sh**

**Cd /test-mnt**

**Echo “inside from pod” >from-pod.txt**

**Exit**

**Let’s check in the node path**

**Cd /test-vol**

**ls --------------------------- from-pod.txt will be present**

**cat from-pod.txt ------- you will get “inside from pod”**

**From both the test ( test1 and test2) it is confirmed the data stored in the mount point in the pod is same with the data stored in the directory on the node where the pod is running.**

**After delete the pod using kubectl delete pod test-redis-hp when we will check the directory present in the node we will still get the files.**

**Cd /test-vol**

**ls ------------- we will see all the files**

**How to use GCE Persistent disk?**

**GCE Persistent disk is a persistent disk on Google Compute engine. So GCE persistent disk volume type mounts a persistent disk into your pod.**

**The contents inside the GCE persistent disk are preserved even after the pod is removed or dies for any unknown reason. GCE persistent disk can only be mounted by a single pod in a read write mode, not the simultaneous writers are allowed. But data can be read by multiple pods simultaneously.**

**What are the restrictions for GCE Persistent disk?**

**First you must create a GCE persistent disk before you can use it in Podspec.**

**Second Kubernetes worker node on which pods are running must be Google Cloud instances.**

**And finally, those VMs needs to be in the same GCP project and zone as a persistent disk.**

**Note 🡪 AWS Elastic Block Store and Azure disk are similar to this and has the same restrictions.**

**How to create GCE Persistent disk manifest file and mount it?**

**Step 1 🡪 create the persistent disk on Google Cloud platform.**

**gcloud compute disks create –size=10GB –zone=us-central1-a my-data-disk**

**step 2 🡪 create the pod manifest file gcePersistentDisk.yaml**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: gce-pd**

**spec:**

**containers:**

**- name: test-container**

**Image: nginx**

**volumeMounts:**

**- mountPath: /test-pd ----------------- created inside pod**

**Name: test-volume**

**Volumes:**

**- name: test-volume**

**gcePersistentDisk:**

**pdName: my-data-disk ------------ created in Gcloud node**

**fsType: ext4**

**Now create the pod using kubectl create -f gcePersistentDisk.yaml**

**To check the pod status kubectl get pod -o wide**

**To display complete detail of the pod kubectl describe pod gce-pd**

**How to validate the GCE persistent disk is configured correctly?**

**Test 1 🡪 let’s create a file inside the pod mount path**

**Kubectl exec gce-pd -it -- /bin/sh**

**df ----------- find /test-pd is present**

**echo “testing in pod” > /test-pod/test1.html**

**exit**

**now delete the pod using kubectl delete -f gcePersistentDisk.yaml**

**Again recreate the pod using kubectl create -f gcePersistentDisk.yaml**

**When we will check the pod status kubectl get pod -o wide pod will be created in different node.**

**Let’s check the newly created pod has the data or not.**

**Kubectl exec gce-pd df /t est-pd**

**Kubectl exec gce-pd ls /test-pd**

**Kubectl exec gce-pd cat /test-pd/ test1.html**

**or**

**Kubectl exec gce-pd -it -- /bin/sh**

**df ----------- find /test-pd is present**

**cat /test-pod/test1.html**

**What is persistent volumes PV?**

**Persistence volume is a piece of pre-provisioned storage inside k8 cluster created by administrator. The data inside these volumes can exist beyond the lifecycle of an individual pod.**

**Persistence volumes can be any storage types such as Block, NFS or distributed.**

**What is  persistent volume claim (PVC)?**

**A persistent volume claim is a storage of some capacity along with some access modes such as read, write or read only request by a user. After that, developer can use this volume inside the pod to mount the volumes and use it.**



**How PV and PVC works? / Life cycle of PV?**

**Once the storage/ persistence volume is provisioned, we bind the storage request to the persistence. So as soon as the persistence volume claim is created, a control loop on Kubernetes master watches for any new persistent volume claims and binds the matching PV if it's available in that pool. And finally, when a user is done with their volume, they can delete the persistent volume claim objects from the Kubernetes.**

**Provisioning ---- > binding ---- > using ------- > reclaiming**

**What if we don't find a match between our request storage and available storage in the pool?**

**We'll see the two scenarios here.**

**First, let's imagine that we have requested 100 gig storage, but the available storage chunks in that**

**storage pool is only 50. Then Kubernetes doesn't find any PVS that match to our request.**

**Then in that case, persistence volume claim request will wait at least 100 gig. Persistent volume is created and added to the pool.**

**In another scenario, let's imagine that you have requested 10 gig storage.**

**But the available storage pieces inside the pool is 12 gig.**

**In this case, persistence volume claim will bound to the 12 gig PV because the user will always get**

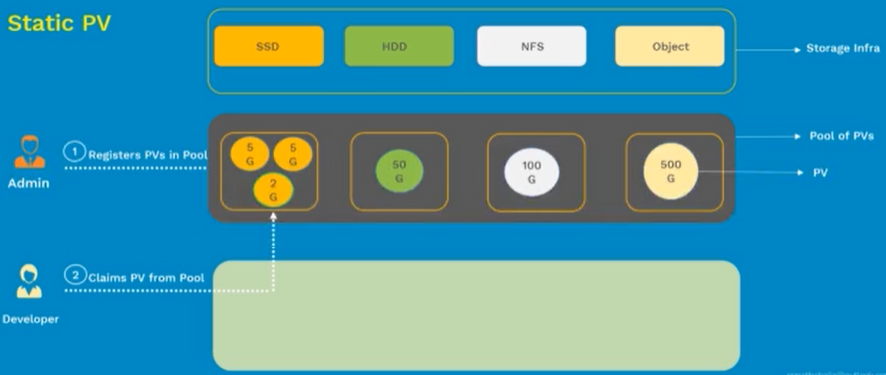
**at least what they asked for. But the volume may be in excess of what was requested, but the exit should not be too much. Point here is it's okay to be a bit excess, but not the double.**

**How can you provision storage volume in K8?**

**There is two ways to provision storage volume in k8. One is static volume provisioning and other is dynamic volume provisioning.**

**How to create static volume provisioning happens?**

**Let's imagine that you have a storage infrastructure with different types of storage, such as fast speed , SSDs, slow speed, hard disks, NFS and then the object storage. First thing in static pvc process is to create the persistence volumes of different capacity, which is done by the administrator. The storage chunks are called as persistence volumes. So, when, developer need to create the persistent volume claim or create PVC request to the storage from this pool. ( let’s say developer has requested for two gig of fast speed SSD storage ) Then this PVC claim gets bonded to the (two gig ) PV. Now developer can make use of bonded PVC inside the pod.**



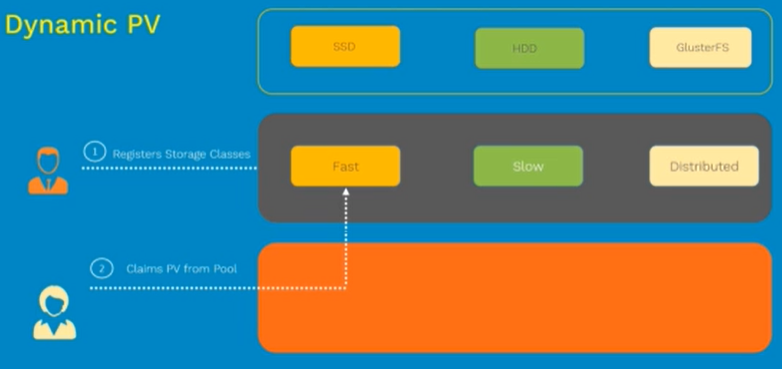
**But there is one problem with it.**

**What if the size of requested storage doesn't match with one of the storage chunks which are available inside the pool? Unfortunately, developer has to wait till admin to create the right size. What developer needs.**

**How to create dynamic volume provisioning happens?**

**Let's say you have a storage infrastructure with different types of storages such as fast speed, SSDs, slow speed, hard disks and distributed Gluster storage. In dynamic provisioning we don't create PVs manually. Instead, admin will create the storage classes. These storage classes are generally created depending upon the type of medium in the backend.**

**For example, as you can see, we have created storage class of fast for SSD drives, slow storage class for hard disks and distributed for gluster-fs.**

****

**So in this case developer doesn't have to worry about the correct storage size is available inside the storage pool or not. Developer needs to confirm whether the required storage class is there or not. Once that is confirmed, developer can create the PVC and which in turn creates the storage respective PV and gets bonded.**

**How to create static volume provisioning in k8?**

**We need to create persistent volume 🡪 persistent volume claim 🡪 referencing /using the claim in pod.**

**How to create persistent volume (PV) config file in k8?**

**vi my-pv.yaml**

**apiVersion: v1**

**kind: PersistentVolume**

**metadata:**

**name: pv-gce**

**spec:**

**capacity:**

**storage: 15Gi**

**accessModes:**

**- ReadWriteOnce ------- storage mount to single node having read write mode**

**storageClassName: slow ------- slow for HD and first for SSD storage type**

**gcePersistentDisk: ------- HD is on google cloud means we are in google cloud**

**pdName: my-data-disk --- you need to create it in GCD before using it**

**fsType: ext4**

**To create the PV using the below kubectl command**

**Kubectl create -f my-pv.yaml**

**How to get the detail about the PV?**

**Kubectl get pv**

**How to get the complete detail of the PV?**

**Kubectl describe pv pv-name**

**Kubectl describe pv pv-gce**

**How to create PVC ( persistent volume claim ) wrt the PV?**

**vi my-pvc.yaml**

**apiVersion: v1**

**kind: PersistentVolumeClaim**

**metadata:**

**name: my-disk-claim**

**spec:**

**resources:**

**requests:**

**storage: 15Gi**

**accessModes:**

**- ReadWriteOnce this detail should match to the PV config file**

**storageClassName: slow**

**to create PVC using the below command**

**kubectl create -f my-pvc.yaml**

**How to get the PVC?**

**Kubectl get pvc ----------- in the o/p you will see the PV under Volume section.**

**Note 🡪 now if we execute kubectl get pv in the o/p you will see the PVC under Claim section.**

**How to use/referencing the pvc inside the pod? / how to referencing claim in pod?**

**Vi nginx-pv.ymal**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: pv-pod**

**spec:**

**containers:**

**- name: test-container**

**Image: nginx**

**volumeMounts:**

**- mountPath: /test-pd**

**name: test-volume**

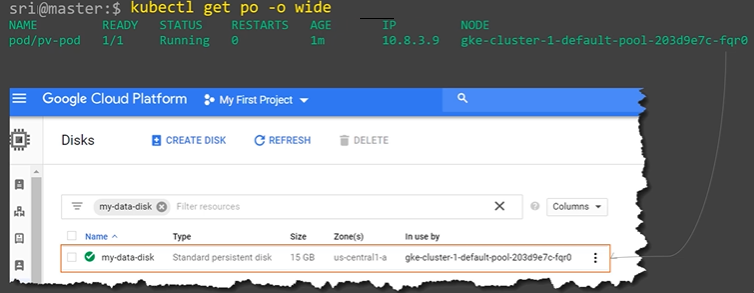
**volumes:**

**- name: test-volume**

**PersistentVolumeClaim:**

**claimName: my-disk-claim**

**now lets create the pod kubectl create -f nginx-pv.yaml**



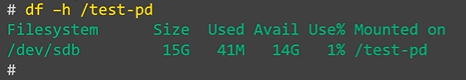
**For the above screen shot we can see the my-data-disk assign to the same node where the pod is created.**

**How to test the PV and PVC is configured properly using static volume provisioning?**

**Create a test file inside the disk where the disk is mounted, we will do that by going inside the pod.**

**Kubectl exec pv-pod -it -- /bin/sh**

**# df -h /test-pd ------------- check the disk mounted to the mount path mention in pod**

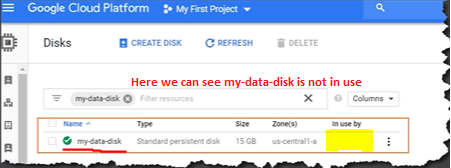


**Add some content in the sample file.**



**Delete the pod , Even if the pod is delete the data will be there in disk in cloud.**

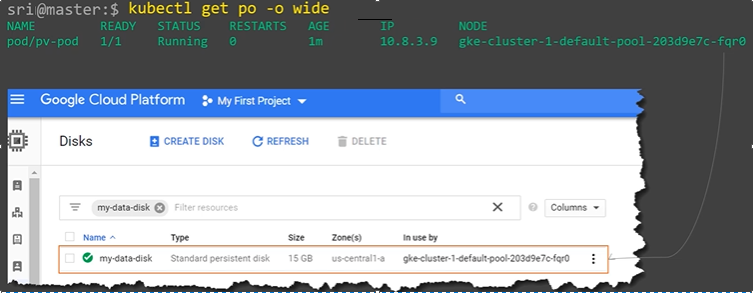
**Kubectl delete -f nginnx-pv.yaml**



**Recreate the pod with the same configuration**

**Kubectl create -f nginx-pv.yaml**

**Verify pod is create**



**Verify the data created in the second step is still available.**

**Kubectl exec pv-pod df /test-pd**



**Kubectl exec pv-pod ls /test-pd**



**Kubectl exec pv-pod cat /test-pd/test1.html**



**How to clean the PV?**

**Remove pod then pvc and then pv.**

**Kubectl delete -f nginx-pv.yaml**

**Kubectl delete -f my-pvc.yaml**

**Kubectl delete -f my-pv.yaml**

**How to create dynamic volume provisioning in k8?**

**We need to create storage class ----- persistent volume claim ---- use/referencing claim in pod**

**How to create a storage class config file?**

**Vi my-sc.yaml**

**apiVersion: storage.k8s.io/v1**

**kind: StorageClass**

**metadata:**

**name: fast ------ name of the storage class, provide it carefully as it can not change**

**provisioner: Kubernetes.io/gce-pd ----- Provisioner is what determines the volume plugin.**

**For AWS it will be AWS-EBS**

**parameters:**

**type: pd-ssd -----------  type of the disk and here we are using SSD**

**Let’s create the storage class kubectl create -f my-sc.yaml**

**How to get the storage class in k8?**

**Kubectl get storageclass**

**How to get the complete detail of the storage class in k8?**

**Kubectl describe storageclass fast**

**In the output, you will get the complete details about the provisioner parameters, Reclaim policy and list of all the events about the storage class.**

**How to create PVC ( persistent volume claim ) wrt the Storage class?**

**vi my-sc-pvc.yaml**

**apiVersion: v1**

**kind: PersistentVolumeClaim**

**metadata:**

**name: my-disk-claim-1**

**spec:**

**resources:**

**requests:**

**storage: 30Gi**

**accessModes:**

**- ReadWriteOnce**

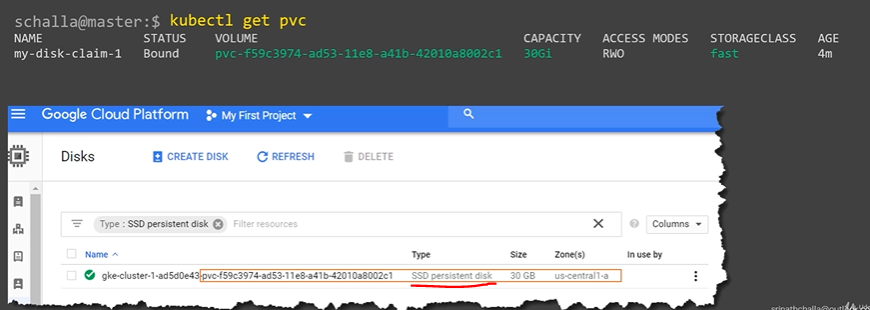
**storageClassName: fast ----------- this detail should match to the SC config file**

**to create PVC using the below command**

**kubectl create -f my-sc-pvc.yaml**

**How to get the PVC?**

**Kubectl get pvc ----------- in the o/p you will see the SC under Volume section.**



**Note 🡪 now if we execute kubectl get pv in the o/p you will see the PVC under Claim section.**

**How to use/referencing the pvc inside the pod? / how to referencing claim in pod?**

**Vi nginx-pv.ymal**

**apiVersion: v1**

**kind: pod**

**metadata:**

**name: pv-pod**

**spec:**

**containers:**

**- name: test-container**

**Image: nginx**

**volumeMounts:**

**- mountPath: /test-pd**

**name: test-volume**

**volumes:**

**- name: test-volume**

**PersistentVolumeClaim:**

**claimName: my-disk-claim-1 ------------- this should match to PVC manifest file**

**Now let’s create the pod kubectl create -f nginx-pv.yaml**



**From the above out put we can see the disc width same (30 gig) is marked as in-use by the same node where the pod is created.**

**How to test the PV and PVC is configured properly using dynamic volume provisioning?**

**Create a test file inside mount, we will do that by going inside the pod.**

**Kubectl exec pv-pod -it -- /bin/sh**

**# df -h /test-pd ------------- check the disk mounted to the mount path mention in pod**

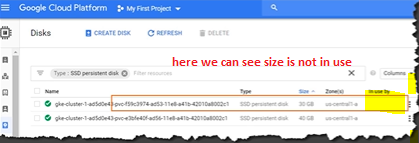
****

**Add some content in the sample file.**



**Delete the pod , Even if the pod is delete the data will be there in disk in cloud.**

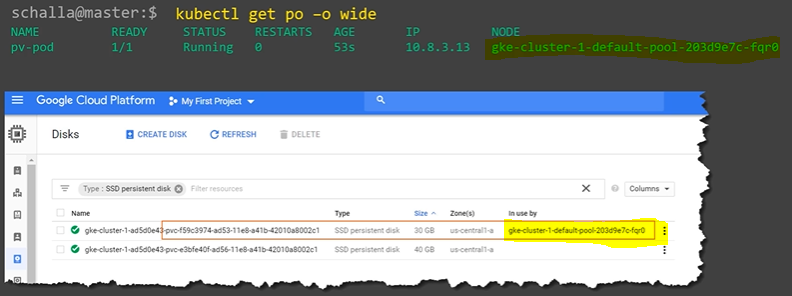
**Kubectl delete -f nginnx-pv.yaml**



**Recreate the pod with the same configuration**

**Kubectl create -f nginx-pv.yaml**

**Verify pod is create**



**Verify the data created in the second step is still available.**

**Kubectl exec pv-pod df /test-pd**



**Kubectl exec pv-pod ls /test-pd**



**Kubectl exec pv-pod cat /test-pd/test1.html**



**How to clean the SC?**

**Remove pod then pvc and then sc.**

**Kubectl delete -f nginx-pv.yaml**

**Kubectl delete -f my-pvc.yaml**

**Kubectl delete storageclass fast**