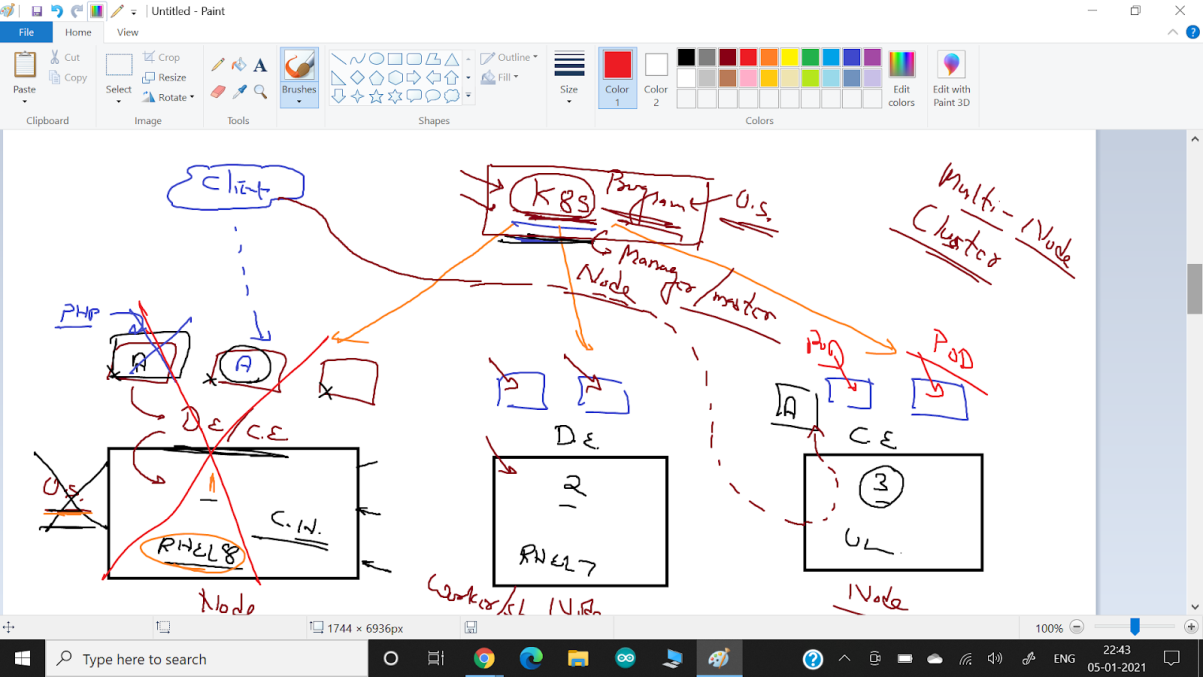
**Session 01 – Kubernetes//pod**

* One of the time-consuming works in industry is to keep on monitoring our OS.
  + They don’t want downtime in the system.
  + If somehow OS crash or terminate this guy will launch one new container.
  + Docker will launch container within a second, but it will be only noticed, when someone notice it and re-launch container.
    - They will manually launch docker run command.
  + In real world we have hundreds of OS is going on.
  + If our critical application will be down, company may face a loss of reputation.
* Here we don’t have any drawback in the containerization technology, problem is in the management.
  + More OS harder the management.
  + Human being is also slower many times.
* So, instead of human being we can use a program, we can create one OS which will keep on monitoring container.
  + Responsibility of this program is to monitor all OS, if any OS goes down they will launch exact new copy of a OS.
  + They will also manage session//cookies.
* So, here we need one program for managing container.
* One of the famous program/tool/software is **Kubernetes**.
  + Kubernetes will also provide you their own load balancer.
  + It will provide you auto scaling.
  + Also used for orchestration.
* Here we can do one more thing.
* Instead of launching multiple containera on top of same base OS, we can also launch multiple containera on top of multiple base OS, here OS is also known as Node.
  + If one complete Node fails, kubernetes will create same container on top of other running nodes.



* Here main node who will manage container & all other things, is known as master node.
  + Other nodes, on which container managed by master node, is known as worker node or slave node.
* So, this thing is known as Master-Slave architecture.
  + One master and multiple slave cluster is known as multi-node cluster.
* Here kubernetes wants to manage a container.
  + For this they will wrap this container on a box.
  + This box is known as **POD**.
  + Kubernetes will check pod, if container down, pod will also be down.
  + So if pod is down it will launch one more pod.
    - Kubernetes will manage pod.
  + We can also have multiple containerwithin one pod.
* Kubernetes cannot launch container, but they can help you.
* They will go to container engine and ask them to launch a container.
  + They will launch a container a Kubernetes will wrap this with pod.
* We can also use terminology called agent.
  + Kubernetes is agent oriented program.

**Session 02 – minikube//kubectl//demo**

* Kubernetes is a management tool to manage container and can also provides you complete infrastructure.
  + Eg, resources, security and lots of other things.
* Launching docker manually
  + Docker run -it --name myos vimal13/apache-webserver-php
    - We have to keep on monitoring this OS using docker ps command.
* There is one program available known as minikube, who will configure complete kubernetes server (Master/worker node).
  + Minikubeonly support virtualbox for windows.
  + <https://minikube.sigs.k8s.io/docs/start/>
* Now you can use minikube to download minikube command and minikube will createsingle node cluster for you.
  + Minikube is just anautomation tool to install Kubernetes.
    - Go to that folder and open cmd& write below command.



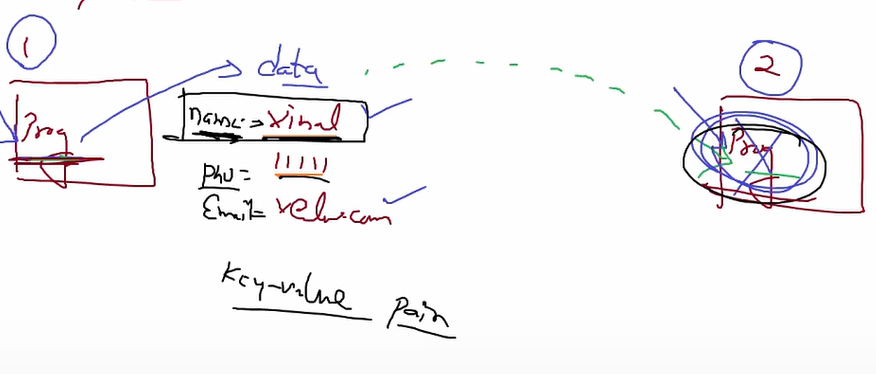
* + After downloading they will install vm in your vmbox, then they will start the vm and configure complete single node architecture for you.
    - You can use minikube status command to check it properly installed or not.
* If you want to use Kubernetes as a client you have to use tool/command called **kubectl**.
  + <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
  + Download client command – kubectl in same folder where minikube installed.
* Kubectl.exe get pods
  + To check how much pod running.
  + Kubectl.exe run --name=myweb1 --image=vimal13/apache-webserver-php
    - It will launch a pod for you.
  + Kubectl.exe delete pod myweb1
    - Now if you see pod is deleted, but it is not automatically started again.
    - In kubernetes we have deployment controller program that can re-launch container if somehow it is terminated or stopped.
  + Kubectl.exe create deployment myweb1 --image=vimal13/apache-webserver-php
    - It will launch one pod for you, with supervision.
    - If you delete this pod, one more pod created.
  + Kubectl.exe describe pods
  + Minikube dashboard
    - It will launch webUI for you.
* If you want to connect your container from outside world, you have to expose (PAT) it.
  + Kubectl.exe expose deployments myweb1 --port=80 --type=NodePort
    - Now from windows you can connect to this deployments.
  + Minikube service myweb1 --url
    - It will provide you URL for connecting to application (pod).
  + Kubectl.exe scale deployments myweb2 --replicas=4
    - It will launch 4 replica for you.
* For login inside minikube VM, Default login name=docker, password=tc-user
  + By default minikube have IP 100 (192.168.43.100).

**Session 03 – yaml//kubectl//kubelet//keywords**

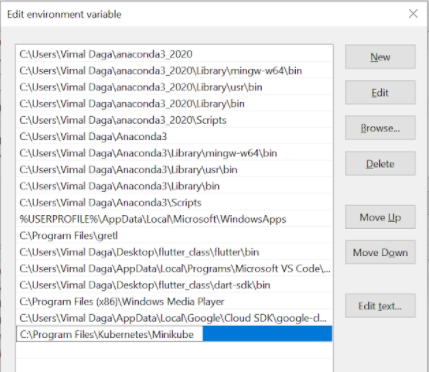
* If you want to launch a App, we need a environment.
  + For making environment we can use image.
* Users connect/interact to a program called API server/Kube API running in a kubernetes.
* We can use minikube command to manage a VM, setup kubernetes master/server program.
  + Minikube.exe status
  + Minikube.exe start
* Using Kubectl you can request a kubernetes to do something.
  + There are 2 way to send a request to API server.

1. CLI
2. code
   * using a code we can c
   * For writing a code we have to use **yaml language**.

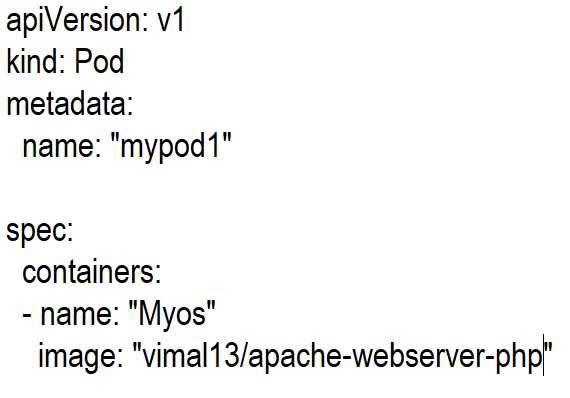
* We have many things in kubernetes where we don’t have command available.
  + Here we must have to use code (yaml language).
  + Kubectl delete all --all
    - It will delete all the things(pods, deployment, services) in kubernetes.
* For connecting to container enginekubernetes use program called **kubelet**.
  + Kubelet connect to container engine and ask him to launch container.
    - DefaultC.E. is docker engine.
  + Kubectl.exe describe pods myweb
  + Kubectl.exe describe <resource><resource\_name>
* Yaml language
  + What is yaml language?
  + For transferring data between multiple programs, or you can say between OS, we need certain way so we can transfer (and understand) data between multiple OS.



* + So, here program 2 receiving data from program 1.
    - When name variable comes, it will retrieve information of it.
  + Sometimes we have multiple values inside one variable.
    - For this you have to give them same indent started with (-).
  + There are three types for key-value pair format.
    - JSON
    - XML
    - YAML
  + Kubernetes is one type of declarative language.
  + For writing block of code we have to use indentation like python language supports.
  + Data (strings) you provide write inside double quotes.
* For giving resource\_name we have to use keyword kind.
  + You have to provide apiversion, that will tell program, which resources, services, which type of code we are going to use.
    - Eg, pod belongs to apiVersion 1.
  + For giving metadata (data related to resource), we have to use keyword metadata.
  + Spec keyword, to say container is one of my specifications.
* We can set PATH so; we can run this command from any folder.
  + Update PATH inside user variable.
* Minikube stop.



* Kubectl create –f pod.yml
  + To run yml file.
  + –f for file name.

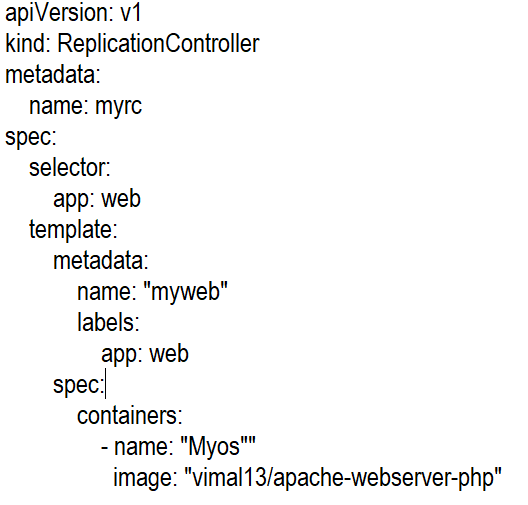


**Session 04 – RC//template**

* By default nature of container engine is, one container has connectivity with other containers (On same n/w only).
  + Container engine put all containers inside a private n/w.
  + They can’t be connected with outside systems.
    - But we can use worker node (minikube\_vm) to connect to this container.
  + If you want to get connectivity from outside, you have to use PATting (One type of NAT).
    - Docker run -it --name myos-p 1234:80 httpd
  + Because of routing table issue in container, no one can connect with container from outside world.
    - But here worker node (minikube\_vm) has outside connectivity.
    - So, worker node can route you to the container.
  + Now you can check your website from windows also using IP of minikube\_vm&exposed port.
* If your container goes down you have to manually launch a container, which will take some time.
  + Container is super fast, but human takes time for this.
* We need some program which will make our container **UpTime** until we ask him to down.
* **Replication Controller**&**Replica** Set can do these things for you.
  + You can use kubectl get rc to check we have any replication controller or not.
* In dynamic world IP is keep on changing, so we cannot use replication controller to monitor container using IP.
  + So instead of giving IP we can give **tag** to it.
  + Now we can ask rc to monitor tag, if tag is down we can ask rc to launch a new container.



* + - Kubectlcreate -f new.yml
* Kubectl get pods -L app
  + It will show you all pods assigned with label app.
* Some program Keep on searching something, also known as keep on selecting.
  + Now we can tell rc if label is down, use this **template** for launching new container.



* + - Kubectl create -f rc.yml
  + Now if your pod goes down, it will re launch your container with same label.
    - Kubectl describe rcmyrc
  + All resources of k8s give you events, using this we can troubleshoot if something happened.
    - Kubectl expose rcmyrc --port=80 --type=NodePort
  + Now you don’t need to worryabout IP, we can access same data everytime.
* POD is used for providing/attaching some extra information to the container which is used by kubernetes.
  + Here we have assigned labels to the pod, we can’t assign anything extra to containers directly.
* When you expose something in kubernetes, client connect to Pod using one program, this program will work as a load balancer.
  + In kubernetes load balancer is known as **service**.
  + So, because of load balancer/service we can launch multiple Pods,create replicas, and attach to same LB.
* Suppose our desire is 3, somehow any Pod goes down then they seamlessly launch one more Pod, so desire status satisfied.



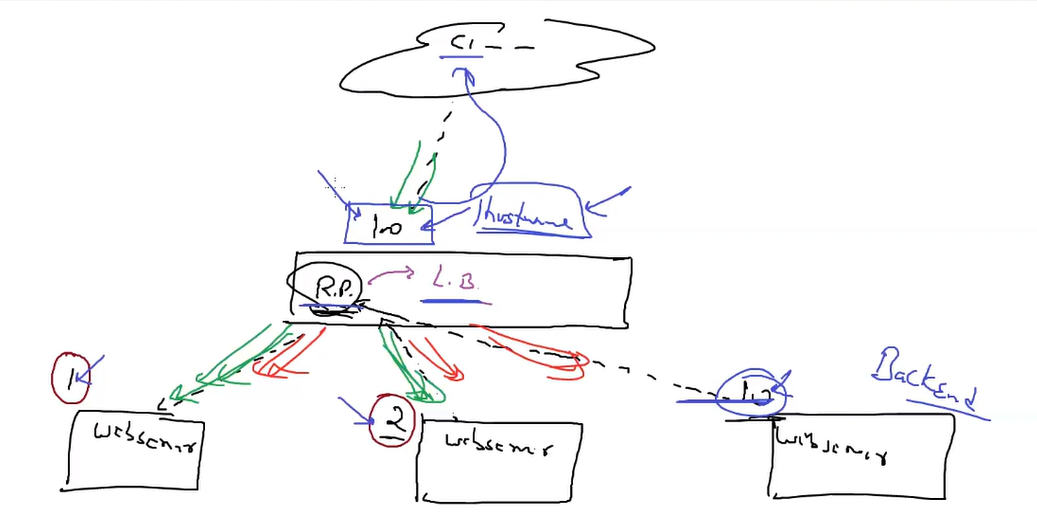
* + - Kubectl apply -f rc.yml
  + We already running with this same rc, so we have to use keyword apply.
    - Kubectl get rcmyrc
  + Now you can check some other container also launched.
  + You can check this in browser.

**Session 05 –Service//RS**

* **Service** resource is used for **Load Balancing**.
  + Three type of service available in Kubernetes.

1. ClusterIP
2. NodePort
3. LoadBalancer (External)

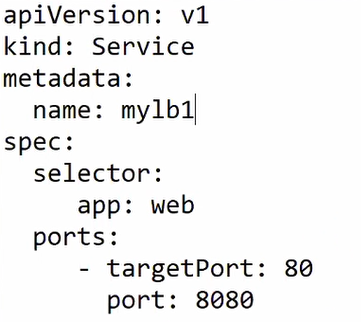
* What is Load Balancing?
* Suppose we have launched a web server Pod using Kubernetes.
* Now client come to the website, port 80 using internet.
* But every server has hardcoded limit.
  + In one point in time, only 100 clients can connect.
  + So, if 100 clients connected and now if some new clients come then these new clients get response server is busy or get timeout error.
  + But company never wants to lose his clients and his reputation.
  + For solving this we can launch more webserver.
    - For launching this OS faster, we can use ansible.
    - But how client can know we have to go to this different system, different IP.
    - In company also we are terminating some servers, when there is less traffic, so we can save some cost.
    - So, client come to this IP it is not running and when it again launches IP may be changed.
  + For solving this we launch one more OS (IP 100).
    - This OS don’t have real website.
    - Any client come to this OS, they will allow as I am webserver and behind the seen on your behalf, they send request to webserver (webserver think that is the client), and webserver give reply to client (proxy server) and this proxy server reply to real client.
      * According to client IP 100 is server.
      * According to server IP 100 is client.
      * This thing is known as reverse proxy.
    - Here by this method, they are also giving you security.
      * In this client never hit to server they only
    - In webserver firewall only allow one IP of proxy server.
      * We isolated our servers.
  + Somebody behalf, somebody running this type of terms known as proxy.
  + Here all are going through IP address.
    - Here we can do one thing. We can guide proxy server to only send 100 clients to one server, then don’t send to same server, instead of this send to the different server (launch one more webserver) .
  + Reverse proxy also gives you one more thing, load balancer.
    - First client connects to first server second client to second server.
      * This thing also known as round robbing.
    - So, any one server doesn’t get more loads.



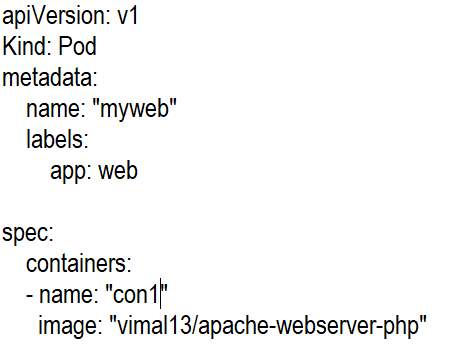
* + What you achieve

1. Reverse proxy
2. More security
3. Load balancer
4. And lots of other things.
   * Here reverse proxy is known as frontend.

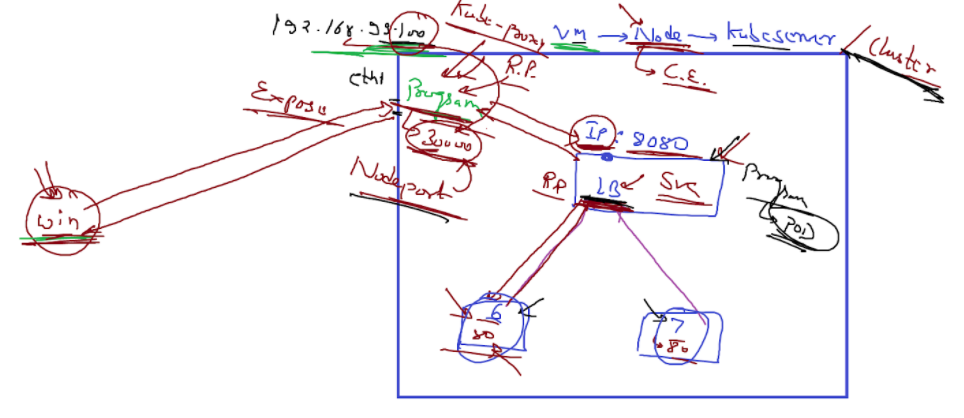
* Service will help you to implementreverse proxy with load balancer.
  + We don’t require to register backend server manually, Kubernetes will service do this thing behind the scene.
  + Container port == Target port
  + Port == service port == load balancer port
* Here you can launch Pod with Labels, so instead of looking for IP load balancer will always look for labels.
  + Launch a service using yml file.



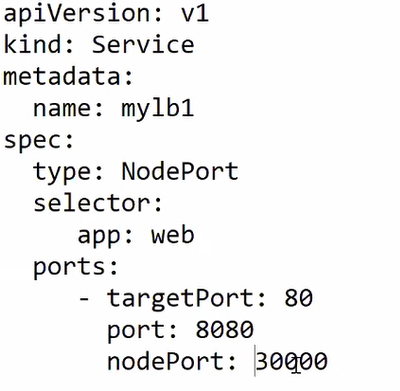
* + Kubectl create -f svc.yml
    - It will only look for selector app=web.
    - In our case we don’t have any pod with this label, so we don’t have any endpoint, you can check this using kubectl describe svc mylb1.
  + Launch a Pod using yml file.



* + Kubectl create -f pod.yml
  + Kubectl get svc
  + They will auto assign IP address/endpoint/front end IP.
    - We can also assign this manually.
    - If you launch one more pod with this label, it will automatically be added to the load balancer.
  + By default, this service don’t have connectivity from outside world.
    - We launched from minikube so other vms& windows also come inside outside world.
    - By default, service will provide you service called clusterIP.
* **ClusterIP service** will give you load balancing inside a cluster.
  + Within a cluster you can connect to outside world.
  + Outside world OS cannot connect to container within the cluster.
    - This service is lost of used in micro services.
    - Used in databases also.
* If you want outside connectivity to your load balancer, for this you have to use **NodePort** service.
  + Here we are going to take help from windows(Node) port.
  + Here any client come to windows at particular port use PAT & connect to load balancer.



* + **Kubeproxy** program will give you this facility of reverse proxy.
  + Launch this service using yaml.



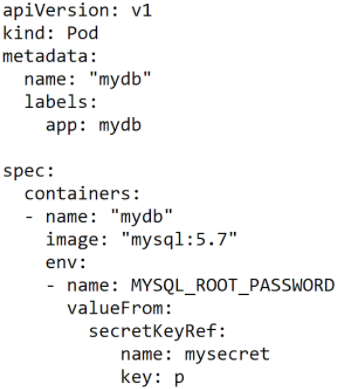
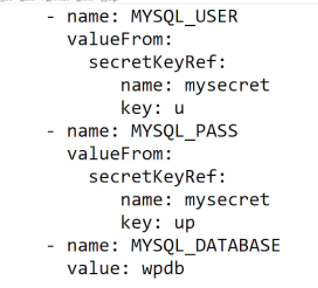
* + - Kubectl apply -f svc.yml
    - kubectl get svc
* **External load balancer**
  + Now k8s is famous and many clouds have their own managed LB service, which will manage entire k8s cluster for you.
  + If you want you can Use ELB you have to use External Load balancer.

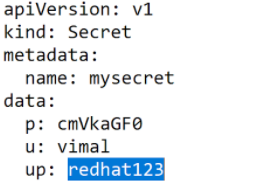
**Session 06 – wp-mysql//multi\_tier\_architecture**

* Creating multitier architecture for Wordpress-mysql.
  + Kubectl run mydb --image=mysql5.7
    - But this command will give you error, you can use describe.
  + Kubectl logs mydb
    - Here they are looking for username & password.
    - These keywords are also known as entrypoint.
  + Kubectl exec –it myos1 -- bash
  + Kubectl run mydb --image=mysql5.7 --env=MYSQL\_ROOT\_PASSWORD=redhat --env=MYSQL\_DATABASE=wpdb --env=MYSQL\_USER=akshit --env=MYSQL\_PASSWORD=redhat
  + Kubectl run mywp --image=wordpress:5.1.1-php7.3-apache
  + By default, in k8s all pods are launched in isolated world.
    - We need isolation for database, but we have to expose wordpress pod.
  + Kubectl expose pod mywp --type=NodePort --port=80
  + Configure using wordpress GUI.

**Session 07 –Secret**

* Make your database isolated.
* Safeguard our password.
* In mysql we have to use environmental variable.
  + Using describing pod one can we your password in clear text.
* We can do one thing.
  + Write yaml file but don’t give password inside yaml file.
  + Now we can store password inside one box and give reference of this box in yaml file.
  + In Kubernetes this concept is known as **Secret**.
  + In ansible we have same concept with name **vault**.
  + Here we are not locking or encrypting the data.
    - Secret is also one type of resource.
  + For passing a environment variable inside yaml file use **env** keyword.
    - Kubectl apply –f.yml
  + Creating yaml file for secret.
    - Here you have to password in **encode** format.
      * Here we use Base64 format.
    - In k8s we can have lots of different things where we can give reference.
    - Here we want it for secret so we have to provide keyword called **secretKeyRef**.

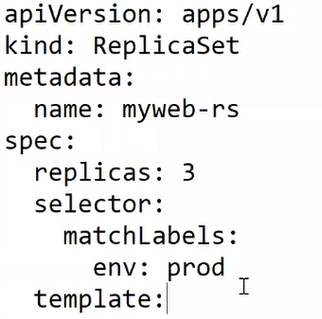
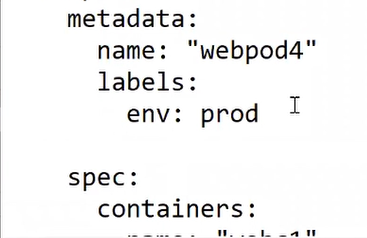
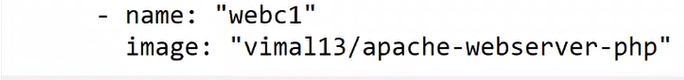
* + - Secret file  
      
    - Kubectl apply –f secret.yml
  + Kube-API only understood yaml file.
  + When you run command manually your CLI will convert it to the yaml file and then send to the API server.
    - You can check this with command
    - Kubectl get pods myweb -o yaml
  + Kubectl get secrets mysecret -o yaml

**Session 08 – deployment**

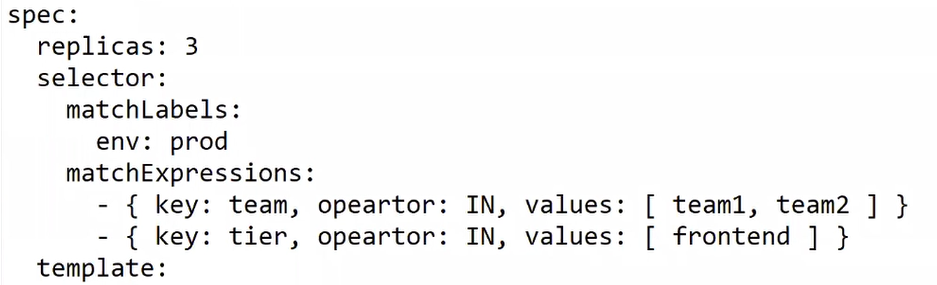
* Replica Set is a upgraded version of replication controller.
  + In dynamic world we need some more facility which is available in the replica set.
  + Launching new Pods == Scale Out
* Suppose we have running a website in production environment with LB and all the facilities.
  + Now our developer has created a new version of App and we have to update this also in real production running app.
  + For this we can use version.
    - Eg, app: v1, app: v2
  + Now for this new image we have to launch new pod.
* Pods are running with replication controller, if our desire is 3 pods, then replication controller will not give you create you new pods.
* We can terminate old pods and create new pods, its take around 5-6 seconds only but our client face **downtime** for 5-6 seconds, saying server is down.
  + It will harm your reputation.
  + Loss our clients.
* For solving this we have lots of **strategies**.
  + Recreate
    - Same as we discuss above
    - Downtime for 5-6 seconds.
  + Rolling update//upgrade (ramped)
    - We will terminate one pod having 0 live client.
    - New traffic sends to new app, after 0 traffic to old pods, they also terminated.
    - Replication set will create a new replica to fulfil desire replica=3.
    - For k8s perspective we have almost **100% UpTime**.
  + This strategy available in many places. But k8s gives you controller.
    - You just have to tell k8s use this strategy.
  + **Deployment** will do this thing for you.
  + In deployment we have also option for rollback.
  + Kubectl create deployment myd --image=vimal13/apache-webserver-php
    - Kubectl get rs
    - Kubectl desc myd
      * You can see by default deployment using strategy called rolling update.
    - Kubectl scale deployment myd --replicas=3
      * It will configure/change rs for you.
      * You will get 3 replicas using rolling update strategy.
  + Let’s check this using example.
    - Kubectl create deployment myd --image=vimal13/apache-webserver-php:lwv1
    - Kubectl expose deployment myd --port=80 --type=NodePort
  + Update the code and create version 2.
    - Kubectl set image deployment apache-webserver-php=vimal13/ apache-webserver-php:lwv2
  + Minikube ssh
    - It will directly give you access to minikube VM from windows.
  + Now keep on connecting using curl 192.168.99.100:30520.
    - You can see without downtime your website updated.
  + You can check with this with multiple ways.
    - Kubetl describe deployment myd
      * In rolling update they have said that, 25% max surge, terminate 25% and create 25%.
      * You can see this describe.
    - Kubectl rollout status deployment myd
    - Kubectl rollout history deployment myd
  + Undo changes.
    - Kubectl rollout undo deployment myd
      * It will rollback to the last version.

**Session 09 – RS//Deployment-yml**

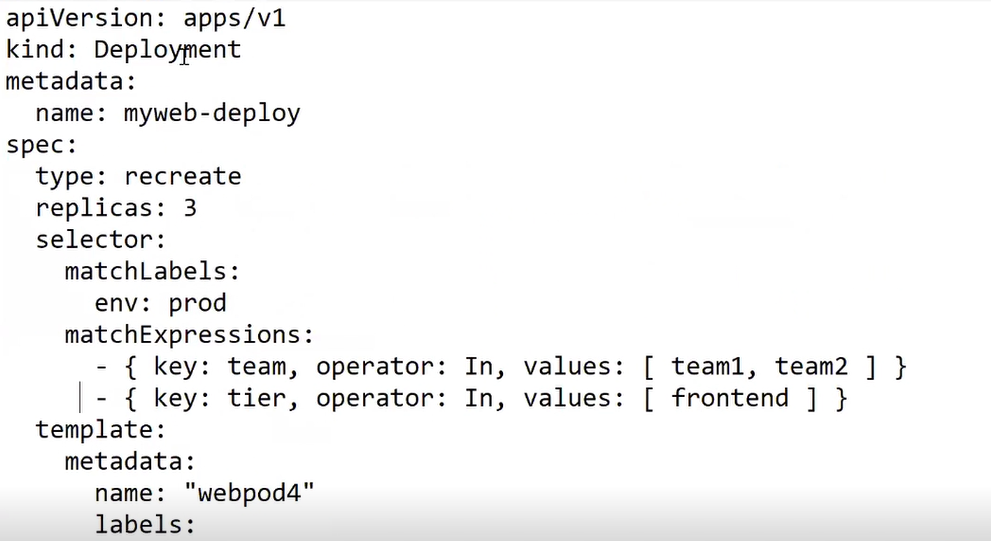
* Deployment behind the seen use Replica Set.
* If you want to do something deep in deployment you have to learn about replica set.
  + Replica set is same as replication controller.
    - But they have one better feature in pod management.
  + In k8s initially selectors are based on **equality**.
    - But it will harder to apply in complex environment.
    - So, after some time new **set-based** selector comes.
  + While you create a pod, try to give as much pod as you can give.
    - Launch 2 pods, (25)
    - Kubectl get pods --show-labels
  + Kubectl get pods -l “env=prod” --show-labels
  + Kubectl get pods -l “team!=team2” --show-labels
  + Replication controller will also using this thing behind the seen.
    - But here we can only use equal or not equal.
  + Kubectl get pods --show-labels -l “region in (IN,US)”
    - This will give you both India & US.
    - Same like Or operation.
  + Kubectl get pods --show-labels -l “region in (IN,US),team notin (team1)”
    - Now it will search in india & US region but not from team 1.
    - This is set based selection.
    - This is used by replica set behind the seen.
      * They also support equality-based selection.

* + Set based selector

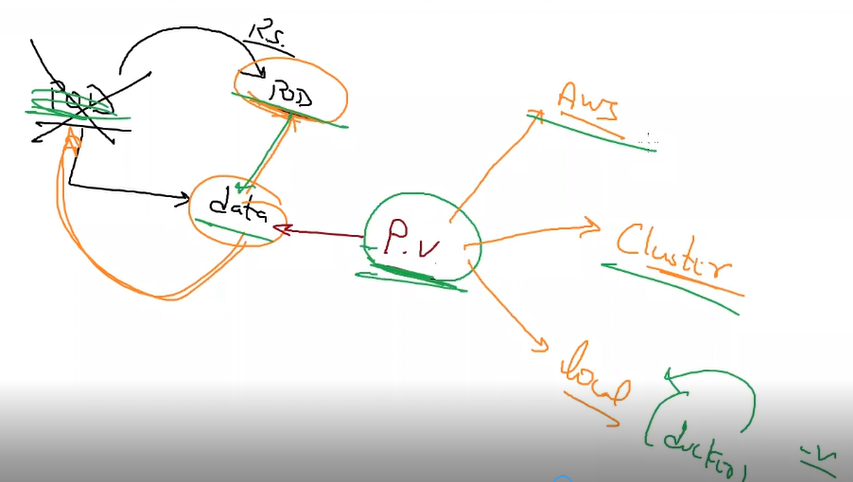


* + If you want to search with equality based, you have to use keyword called matchLabels.
  + If you want to search with set based, you have to use keyword called matchExpression.
    - You can also use both together.
  + Kubectl create deployment myd
    - Kubectl get deploy
    - Kubectl get rs --show-labels
    - Kubectl get pods
    - Kubectl deploy rs myd
      * It will delete Replica set, which also delete pods.
      * But rs managed by deployment.
      * So rs again created so, pods are also created again.
    - Kubectl get pods
  + For deployment yaml file, replica set is a template.
    - For replica set pod is a template.



**Session 10 – PVC//PV//SC**

* When you launch container manually using docker or any other container Engine, here storage we get is temporary (Ephemeral).
* You can with launching container.
  + Docker run -dit --name webos -p123:80 httpd:latest
  + Docker exec -it webos -- bash
    - * Here you can see your HD of your OS, containing all data.
      * Data you store inside is removed when you relaunch container.
      * Only data which comes from image are persistent.
    - Change in index.html file.
    - Remove container and launch again.
    - Now you cannot see this data in new container.
      * Docker rm -f webos
      * Docker run -dit --name webos -p123:80 httpd:latest
    - This storage comes from docker.
    - Storage is provided by docker is permanent (After reboot you can get your data) but temporary (in new container you will not get this data) in nature.
      * This is called **ephemeral storge**.
    - For storing data permanent we need one tool.
  + Here Kubernetes is useful, it will provide **persistent storage**.
  + When you launch pod, their storage is also ephemeral.
  + But Kubernetes have one plugin, using which we can provide persistent storage.
    - If you talk about apache webserver we are concerned about only /var/www/html folder.
    - We also have lots of other folders but when you relaunch container other folders again comes from image but our data comes from client (from dynamic pages) we loss it.
  + In docker world you can do one thing, you can link(Mount) base OS folder with this folder.
    - Docker run -dit --name myos -p 123:80 -v /data1:/var/www/html httpd:latest
    - But it will provide you local storage.
    - Most of time we need storage not from local but from storage unit.
      * Eg, AWS, GCP clouds, Ceph, GlusterFS
    - Kubernetes will have facility to connect with this type of storage providers using plugins.
  + For this we have to use resource called PV.

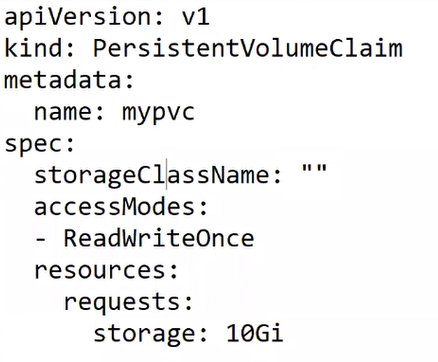


* First developer has created an app, he asks this much space, this is known as **Persistent Volume Claim (PVC).**
* Now IT team has a program/plugin **PV** goes to storage unit ask storage and give (Bound) it to the PVC.
  + Developer creates a Claim and IT team get storage from storage unit.
    - Every PV bound to 1 PVC.
      * 100 PVC = 100 PV
  + This is a manual process, so latest Kubernetes version come up with some extra facility.
    - As demand come PV is automatically created and bounded with pod.
    - For this we need concept called **Storage Class**.
      * Storage class comes with lots of provisioner.
      * Provisioner is also like a plugin.
      * Here PV is created by storage class dynamically, we don’t have to create PV manually.
    - Kubectl get SC
      * Here one sc is created by minikube.

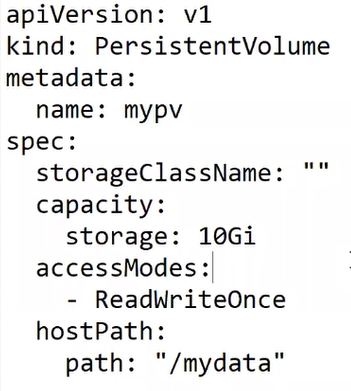
**Session 11 – Practical PVC//PV**

* Practical Of PVC-PV-SC
  + In storage access we have 2 types of access.

1. RO
2. RW
   1. RWO – Read write once
   2. RWX – Read write Many
   * PVC



* Here you have to provide access type, space & name of your PVC.
  + Kubectl apply -f pvc.yml
    - It will give you error and status in pending, because we don’t have PV to claim.
  + We can get storage from our worker Node.
    - Worker node is also known as Host, so this concept known as HostPath.
    - You have create a folder in worker node and use below yml file.
  + PV



* + - As soon as PV created, we already created PVC, PVC use this PV & get storge from it.
    - Here we cannot mention we have created PV for which PVC.



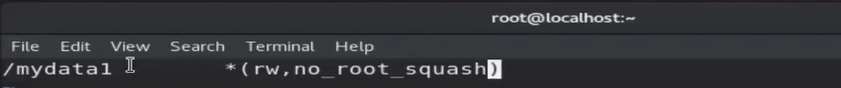
* + Kubectl get pv
  + Kubectl get pvc
  + By default, reclaim policy is retain.
    - Kubectl delete pvc mypvc
      * It will not delete until our Pod is using it.
    - Kubectl get pv
    - By default, you delete your PVC then your data is available in your main storage.
    - Here PVC is not deleted, so you cannot use it in another purpose.
  + Recycle reclaim policy has one drawback.
    - Once you delete your data will be removed.
    - Once your PVC deleted then PV will be available for other usage.
  + Delete reclaim policy
    - Suppose you have created PV using EBS storage.
    - Now once your PVC deleted, it will delete EBS volume also.
      * Check deference between this 3.
      * Recycle PV available for further use, Delete policy delete pv also?
* You have already created PV now for changing PV policy you have to use edit command.
  + Kubectl edit pv mypv
    - It is for changing ongoing running code.
    - Now change policy to recycle policy.
    - As soon as you save this file it will auto work.

**Session 12 – practical of SC//NAS storage**

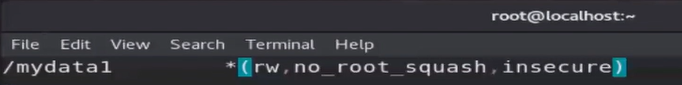
* In last practical users (dev) will claim PVC, our k8s admin/IT guys are keep on checking (Or they have notification program, when dev asks it will mail message admin), admin create PV manually and give it to developer.
  + We have one more approach.
  + We have one auto dynamic program checking for PVC request, they detect how much size of storage they need, dynamically it will get this storage from underlying storage unit, create dynamic PV and bound with PVC request.
  + This program known as Storage Class.
    - Now developer don’t have to wait for admin team.
    - SC provides you **Self Service**.
* By default, one SC is already created by minikube.
  + It will give you PV from your local storage (Worker Node).



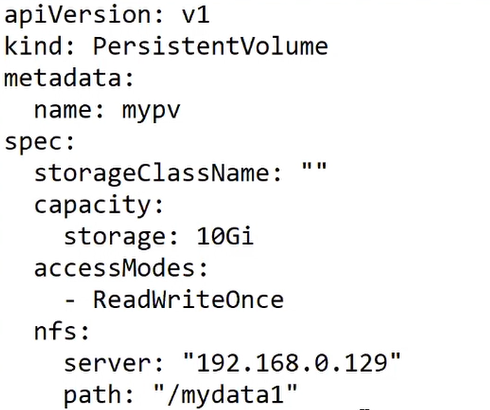
* If you don’t write storage Class complete line in the yaml file then it will use SC dynamically from default storage class.
  + For manuall PVC we have to write this line but don’t specify anything.
    - Eg, storageClassName: “”
  + This is the difference between static and dynamic PV.
* HostPath: it will get storage from worker node with specific path.
* Hostpath is not much useful when you have multi cluster node.
* PV using NAS storage
  + Here k8s get storage from NAS server, so it is known as NAS client.
  + Here we are getting storage from redhat.
  + Configure it as a NFS server



* + Systemctl start nfs-server
    - Start service
    - Root\_squash remove root power.
    - No\_root\_squash give root power.
  + Exportfs -v
    - Check storage is stored or not.
  + NFS client use some random port to connect with server.
    - But NFS only allow secure port (below 1024)
    - So, we have change to allow insecure ports also.



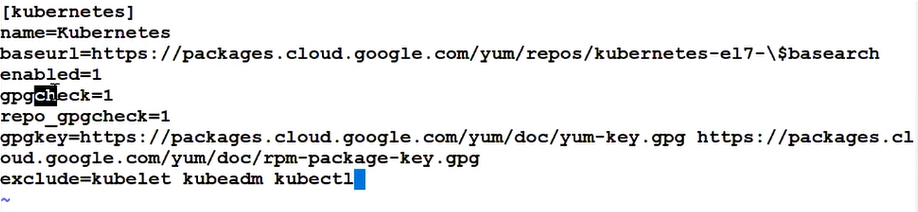
* + Create static PVC using yml file.



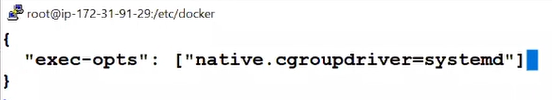
* + Kubectl create -f pv.yml
  + Launch same pod file using PV.
  + Scale your replicas to 3.
    - All the pods get data from same NFS server.
    - Change data in one of the pod it will reflect in the all pods.
  + RWO vs RWX
    - RWO work in all pod in same Node.
      * Don’t work in another nodes.
    - RWX work in all pod in same or different Node.
      * Can write data from multiple pods from multiple Node.

**Session 13 – Multi Node on aws**

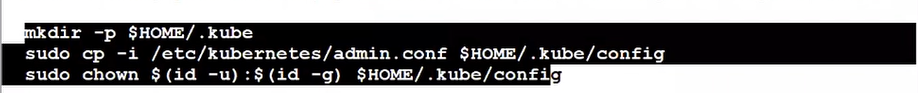
* Container engine is also known as container runtime.
  + We can use docker CRIO or rocket as a container runtime but as a user we have to only concerned about kubectl commands.
  + K8s will create interface on top of this container engines (abstract this is a box), this interface known as CRI interface (Container runtime Interface).
  + Kubectl interact with CRI and CRI will launch container.
  + Kubelet will help you to implement this CRI concept.
* Single node cluster will lead towards SPOF.
* Instead of single worker Node, use multiple worker node.
* If any node fails, we can launch new node and replication controller will launch new pods there.
* **Kube-controller manager (K.C.M.)** is used here.
  + This program will keep on checking health of all the worker node.
  + As soon as one of the Node health goes down, controller node will tell replica set to launch container in new Node.
  + For checking health, we have to create one program in all the worker node.
    - Controller node will keep on check this program it is live or not.
    - This program is a **kubelet**.
  + Client connect to API server, API server contact to controller manager and ask to schedule 3 pods.
* For creating multimode cluster first time worker node has to register himself as a worker node.
  + This thing is known as **JOIN**.
* **Complete Setup**
  + When client come with requesting launching pod with specific image to API server, **Kube schedular** will plan where to launch.
  + Schedular will only decide after this controller manager will connect to replica set and he will launch in IP 1.
  + When controller manager goes to IP 1, it will connect with kubelet.
  + **Kubelet** connect to **C.E.** using CRI.
  + C.E. will launch container.
    - Kubelet is known as **agent program**.
  + Master has some programs, api server, kube schedular, controller manager, etcd.
    - This all program will also launched inside separate containers inside master node.
* **Configure master-slave architecture**
  + Launch 3 instances in cloud.
    - Allow all traffic.
    - Mark one has master, two are worker node.
  + Connect to master node using putty.
    - Install docker
      * Yum install docker
      * Systemctl enable docker
  + Kubeadm is a official software & command using this we can launch multi node cluster.
    - Configure yum for kubeadm
      * <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
      * Use this link with below steps.



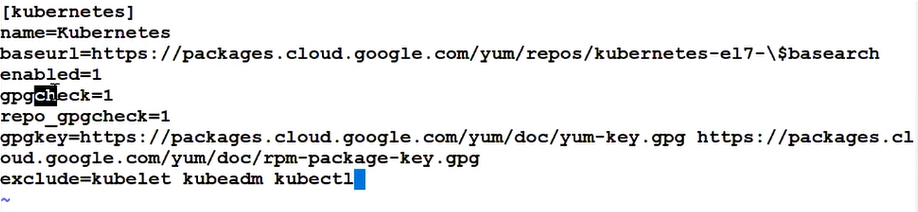
* + - * Save with Kubernetes.repo name.
      * Check with yum repolist
    - Yum install kubeadm
      * For some reason this yum repository has excluded kubeadm software, you can see in last line of pic.
      * So, we have to use this command.
      * Yum install kubeadm --disableexcludes=Kubernetes
  + Install kubelet in all nodes, master also.
    - It is already installed.
      * Rpm -q kubelet
      * Systemctl enable kubelet --now
      * Systemctl status kubelet
        + It will take some time.
  + Download all images for launching pods (controller & all)
    - We can download one by one manually or use kubeadm for this.
    - Kubeadm config images pull
  + Kubeadm init
    - This command we always run-in master.
    - We initializing cluster.
    - But it will give you error.
      * In k8s master we also have to give IP to different pods.
      * What range we have to give is decided by master.
        + While configuring master you have to decide you network range.
        + For this we have to use program called **CNI**.
  + Kubeadm init --pod-network-cidr=10.240.0.0/16
    - This will give network range to pod.
  + Now you can see one warning comes
  + In back of k8s we are using docker and docker using one driver called cgroupfs, But Kubernetes don’t support this driver.
    - They support systemd driver.
    - So, we have to change driver of docker.
    - For changing docker internal we have to change configure file of docker.
      * Cd /etc/docker
      * Vi daemon.json



* + - * Restart docker services.
        + Systemctl restart docker
  + Kubeadm init --pod-network-cidr=10.240.0.0/16
    - Now you can see new error.
    - Yum install iproute-tc
  + Now you can see errors because we have given 1 Gb ram & 1 CPU & minimum requirement is 2Gb with 2 CPU.
    - Kubeadm init --pod-network-cidr=10.240.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
    - Now you can see all warning are resolved.
      * If still not work use vim /etc/sysctl.d/k8s.conf
      * Explained in worker node setup.
  + Lets check our system.
  + Kubectl get pods
    - But you can see it is not working properly.
    - Because for connecting we require IP address, port, username, password.
    - For this copy this thing in your master.



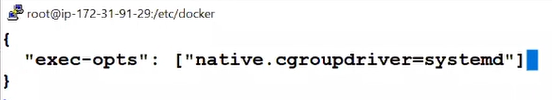
* + - Now your system has been configured as a user also.
  + Kubectl get pods
  + Systemctl status kubelet.
    - This time this program is running.
  + Kubectl get nodes
    - It will tell you, you have only one node which is working as a master node.
    - But status is not ready.
* Worker node commands
  + Yum install docker -y
  + Systemctl enable docker --now
    - Now we have to install agent.
  + Yum install kubelet
    - Configure yum for this.
    - Kubernetes.repo



* + - Yum install kubelet kubeadm kubectl --disableexcludes=Kubernetes -y
      * We require kubeadm for JOIN command.
  + Systemctl enable kubelet --now
    - Systemctl status kubelet
    - Wait for this.
  + Kubeadm join
    - This will fail.
    - For authenticate master has to give token to worker nodes.
    - Kubeadm token list
    - Kubeadm token create --print-join-command
      * It will give you token URL.
      * Paste it into the worker node.



* + Change docker driver.
    - Cd /etc/docker
    - Vi daemon.json



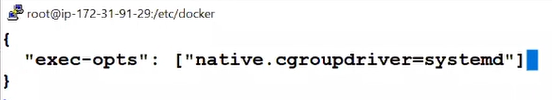
* + - Systemctl restart docker
  + Now again use join, you get error for tc.
    - Yum install iproute-tc
  + Vim /etc/sysctl.d/k8s.conf

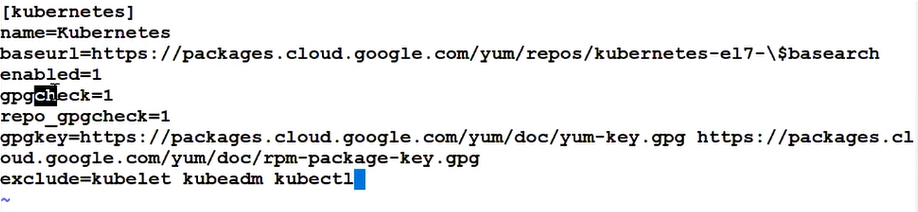


* + Now Join command will work.

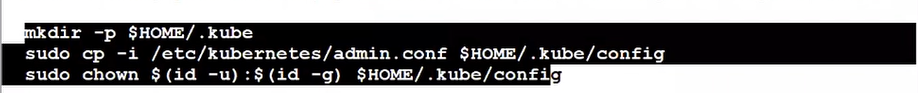
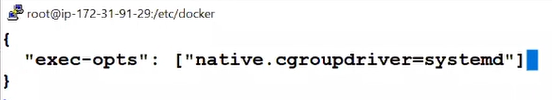


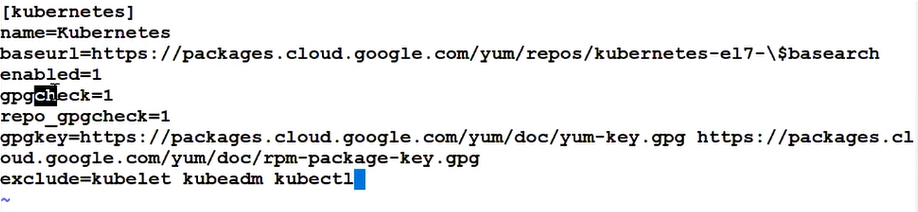
* + - When you write join request from worker node, behind the seen they download some docker images and run inside containers.
* Now you can check with kubectl get nodes.
  + You can see one more node, worker node there.
* While configuring master they require to add some CNI.
  + kubectl apply  -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>
  + kubectl get nodes
    - now you can see your complete cluster is ready.
    - You require any one CNI.
* Short steps for installing Kubernetes cluster.
  + **Master**

1. Install docker
   1. Yum install docker
   2. Systemctl start docker
   3. Cd /etc/docker
   4. Vi daemon.json
      1. 
   5. Systemctl restart docker
2. Configure yum for kubeadm
   1. Kubernetes.repo



Yum install kubeadm

1. Systemctl enable kubelet --now
2. Kubeadm config images pull
3. Yum install iproute-tc
4. Kubeadm init --pod-network-cidr=10.240.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
5. kubectl apply  -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>
6. 
   * **Worker**
7. Install docker
   1. Yum install docker
   2. Systemctl start docker
   3. Cd /etc/docker
   4. Vi daemon.json
      1. 
   5. Systemctl restart docker
8. Configure yum for kubelet
   1. Kubernetes.repo



1. Systemctl enable kubelet --now
2. Yum install iproute-tc
3. Vim /etc/sysctl.d/k8s.conf



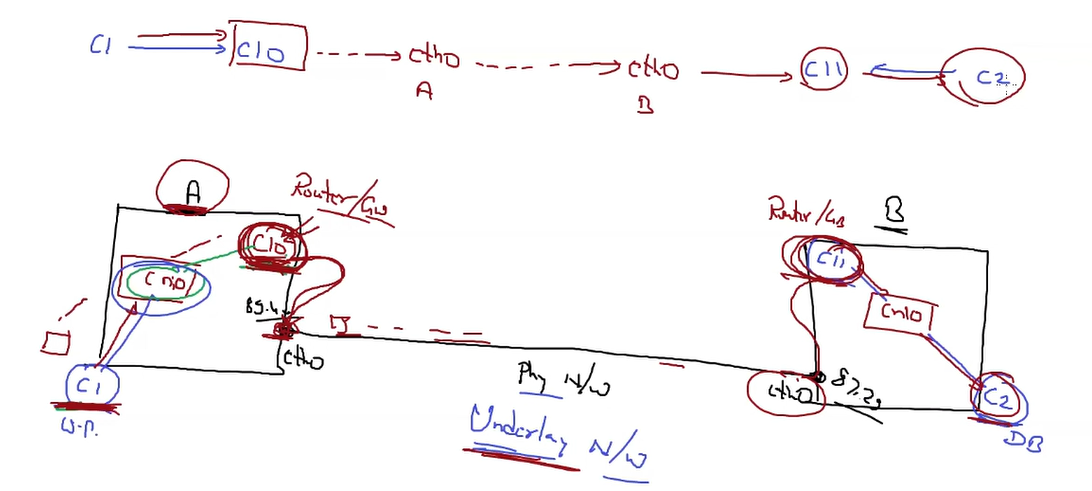
1. Kubeadm token create --print-join-command (master)
   1. Copy this command and run.

kubeadm join 172.31.40.201:6443 --token 6iqkhl.f143xnv9iubk4zyg --discovery-token-ca-cert-hash sha256:6463cbfb6f25e953db0cb7c535ad8e3f167764d3022a6edb58514f39c5062049



**Session 14 – Overlay//flannel**

* Launch 2 worker node & one master node.
  + You can use ASG & inside metadata write this thing, so new instance launches automatically join cluster.
* Create one single multi node cluster in company and give access to all teams working there.
  + So, we don’t have to create separate cluster and waste our hardware resources per team.
  + But here security issue comes up.
  + So here we can use **namespace**.
    - Separate isolated room for separate team.
    - Here it is known as tenant/user.
    - You can specify how much resources you want to give to particular team.
    - Known as multi tenancy.
  + We require namespace to do anything inside Kubernetes cluster.
  + By default, k8s create one namespace called default namespace.
* Kubernetes own setup also have their own namespace for internals (**kube-system**).
  + Kubectl create namespace ak
  + Kubectl create deployment myd –image=httpd --namespace ak
  + Kubectl create deployment myd –image=httpd -n ak
  + Kubectl delete namespace ak
* Kubectl get pods -n kube-system
* Here you can see some pods are not running.
* So, our cluster is not completely working.
* In networking we have different network cards.
  + Their names are something like eth0, eth1.
  + Docker provides you virtual network card.
  + So known as veth0.
* Inside OS if you want to create switch we have to use some software.
* Here we have to use **SDN** (concept//technology), for creating software define network.
  + We can create switch also.
  + Software world this switch known as bridge.
  + One of the famous software is bridge-utils.
    - Brctl show
      * Available inside aws by default.
      * This will show you how many switch you have.
  + When you join to master, behind the seen k8s automatically create switch with name cni0 with help of docker.
  + When you launch pod one virtual card interface is attached, you can check with ifconfig -a
    - Brctl show
      * It will also show you one interface with cni0.
      * Multiple pods connected to same switch have connectivity with each other.
      * But pods from different node can’t connect to each other.
    - For this we require physical network between two nodes.
      * This network is known as **overlay network**.
      * Now we launch one container in all worker node.
      * When any pod of that node wants to go to outside world, it will go to this special container.
        + This container work as router for them.
      * Now this special container get packet from this, it will send packet to the real network card (Underlay n/w).



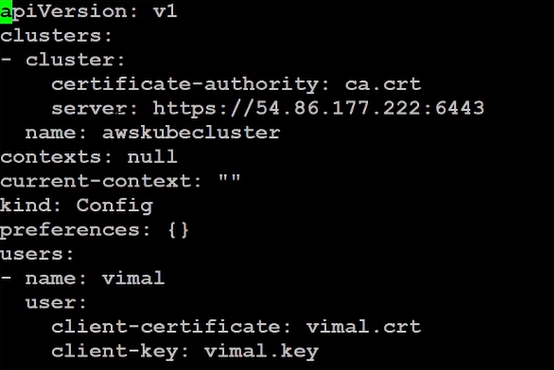
* + - Here c10 & c11 are this special container.
  + C1 use c10 then use underlay network reach to second node, use c11 reach to c2.
  + C10 & c11 are work as router & DHCP both.
  + Here c10 connecting to c11 using underlay network, but its look like they have real direct connectivity from c10 to c11.
    - When 2 containers establish their own network, look—like they have direct connectivity is known as **Overlay network setup**.
    - For this setup we have to use **tunnelling** concept.
    - You can feel like instead of two separate LAN we have one single big LAN.
  + In master node at end, we have use GitHub URL for flannel app, this will launch this container c10 & c11.
  + Kubectl get pods -n kube-system -o wide
    - You can check we have 3 flannel, 2 for worker node, 1 for master.
  + Here master give range of IP for pods, flannel will manage this IP.
  + While launching master we have given IP range in master.
  + But in flannel config file, it is written that they will only manage this IP range.
  + So, we have to update flannel default conf file.
* In k8s we have one concept.
  + Suppose our App working on worker node B.
  + But if you connect with Node A with same port number, Node A go to B get data from pod and provides to our clients.
  + This concept is also known as overlay.
  + In k8s this is known as **Kube-proxy** program.
    - But this is not working in our setup.
    - We have to change conf file of flannel.
  + Vim /var/run/flannel/subnet.env
    - You can see default flannel network.
  + For change configuration file we can use k8s.
    - Kubectl get configmap -n kube-system
      * You can see kube-flannel file.
    - Kubectl edit configmap kube-flannel-cfg -n kube-system
      * Change network range.
      * Save and restart service.
      * Instead of restart service, restart pods.
    - Kubectl get pod -l app=flannel -n kube-system
    - Kubectl delete pod -l app=flannel -n kube-system
      * It will restart pod.
    - Vim /var/run/flannel/subnet.env
      * Now you can see conf file properly working.

**Session 15 – client-password**

* In industry we have multi master node cluster which gives you high availability.
  + Our current cluster has one master, so we have SPOF with respect to master.
  + Kubectl get pods --server 54.52.152.254
    - It will not work.
    - Client connect to master API server.
    - Port number of API server is 6443.
    - Kubectl get pods --server 54.52.152.254:6443
      * Now it will give you error.
      * You have to give username password.
    - Cd /etc/Kubernetes
    - vim admin.conf
      * here you can find your credentials.
      * Kubernetes-admin is your username.
      * Client-key-data is your password (private key).
      * Download complete file and save as some name (admin.yml).
        + This file is known as kube-config file.
    - Kubectl get pods --kubeconfig admin.yml
      * If you use file don’t use IP address.
      * This file already contains your IP address.
      * Here we are using AWS so it has private IP of master.
      * So change it with public IP.
      * In latest k8s version only limited IP can allow to connect to server.
      * For this we have to use IAM, discuss in next class.
  + when you use local cluster, admin.yml (kube-config) automatically create in your windows.
    - So, you can easily connect to cluster.
    - You can create multiple switch logically from one switch.
      * For this we have to use VLAN concept.
    - In flannel we have given one big range of network.
      * And we have multiple subnet having smaller range known as LAN.
      * Flannel hide underlay network known as IPMASQ=true
      * Cat /var/run/flannel/subnet.env
    - Flannel maintain all his details in database called **etcd**.
      * Secret database is also stored inside etcd.
      * All management details of k8s is stored in etcd.
  + For networking in k8s we have plugin called flannel, calico.
    - But here we have to manage, change something.
    - One more plugin aws support is AWS VPC.
      * Now you don’t have to change something manually.

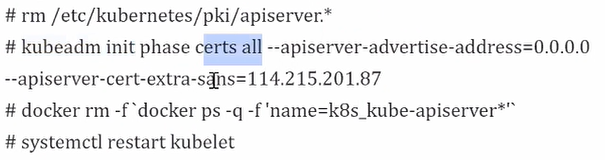
**Session 16 – security//IAM**

* You can create multiple namespace with specific permissions.
* How much resource you can use is decided by Kubernetes master.
  + In terms of create pod, list all pod, delete deployment, etc.
  + What verb (action) you can perform on resources in decided by cluster master team, admin team.
  + For this we have to provide identity and access management.
  + This is known as **role**.
* Sometimes this is also known as **RBAC** (roll-based access control) (Concept name).
  + Kubectl cluster-info
* In k8s we cannot create username & password.
* Here instead of key based authentication we have **certificate-based** authentication.
* When you connect to cluster, Kubernetes will tell your OS—windows—program kubectl to authenticate himself.
  + Here we pass our certy to k8s, k8s will check and if correct stamp (signed) by master and send back to your OS.
    - In certificate you put your details and public keys.
      * We don’t share private key because of security.
      * Known as **CSR** (certificate singing request).
    - After stamp by master this certificate is known as **CRT** (Chinese reminder theorem).
    - Cd /etc/kubernetes/pki
    - Ls
      * It will give you ca.crt & ca.key file.
* If you want to create private key & CSR one of the famous commands is openssl.
* Openssl genrsa -out vimal.key 1024
  + Generate private key using rsa algorithm.
  + Size of key is 1024.
* Openssl req -new -key vimal.key -out vimal.csr
  + They will ask you some details.
  + This will also create public key from private key.
  + Copy this file in master pki folder.
* Openssl x509 -req -in vimal.csr -CA ca.crt -CAKey ca.key -CAcreateserial -out vimal.crt
  + First time we have to use CAcreateserial command.
  + Copy vimal.crt file inside your kubectl client.
* If you use linux for this process you have to install kubectl there also.
* Kubectl config view
  + Kubeconfig is configuration file for client program.
  + Kubectl config --kubeconfig vimal.kubeconfig set-cluster awskubecluster --server <https://54.52.63.112:6443> --certificate-authority=ca.crt
    - Awskubecluster is cluster name.
  + kubectl config view --kubeconfig vimal.kubeconfig
  + kubectl config view --kubeconfig vimal.kubeconfig set-credentials vimal --client-certificate vimal.crt --client-key vimal.key
  + kubectl get pods --kubeconfig vimal.kubeconfig
  + vim vimal.kubeconfig

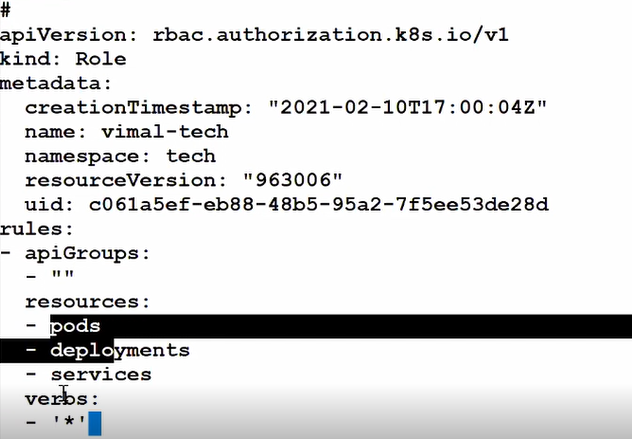


session 17 –

* In real world kubectl client running on multiple clusters.
* So kubeconfig file has multiple cluster info.
* In kubectl get pods --kubeconfig vimal.kubeconfig, we cannot know which cluster we want to access.
* Kubectl get-clusters
  + Here we may have multiple files so we have to specify config file name.
  + Kubectl config get-clusters --kubeconfig vimal.kubeconfig
  + Kubectl config set-context vimal@awskubecluster --user=vimal --cluster awskubecluster --kubeconfig vimal.kubeconfig
    - vimal@awskubecluster here vimal is variable name mapped with awskubecluster.
  + Kubectl config get-context awskubecluster --user=vimal --kubeconfig vimal.kubeconfig
    - Using context we can map cluster with credentials.
    - You can check with vim vimal.kubeconfig
    - We can specify which is default context.
  + Kubectl config use-context vimal@awskubecluster --kubeconfig vimal.kubeconfig
  + Run inside master
    - Openssl -in apiserver.crt -text
    - Here it is mentioned certificate is only work for this 2 IP’s.
    - For solving this we have to create new key and certificate so we can connect to our cluster.



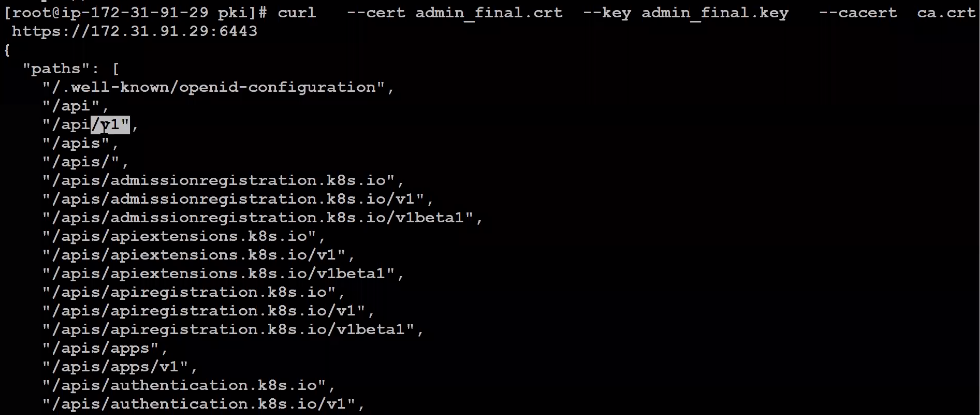
* + - kubectl get pods --kubeconfig vimal.kubeconfig (client)
    - kubectl create ns tech
    - kubectl create role vimal-tech --resource=pods --verb=get, list -n tech
      * here verb == action
      * vimal-tech is a name
        + it will help to see in future vimal user with tech room.
        + Vimal as manager vimal-manager.
    - kubectl get roles -n tech
    - Here we just created a role but we have to bind it with user.
    - Kubectl create rolebinding vimal-tech-rb --role vimal-tech --user vimal -n tech
    - Kubectl get rolebinfing -n tech
    - Kubectl describe role vimal-tech-rb -n tech
  + Client
    - kubectl get pods --kubeconfig vimal.kubeconfig --namepsace tech
    - kubectl edit role vimal-tech -n tech



* + ‘\*’ will for all power to specific resources.

Session 18 –

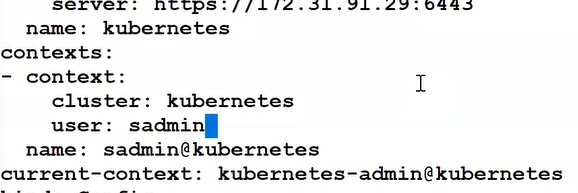
* Kubectl is calling a API.
* K8s has given lots of **rest-API’s**.
* If you try to connect app server from own computer (master node) you have to use
  + Curl <http://172.52.63.95:6443>
    - But it will say you have to connect with secure connection.
  + Curl <https://172.63.95.23:6443>
    - Now it will show you, you don’t have ssl certy.
  + Because we are working in master we have ca certy in /etc/Kubernetes/pki folder.
  + Curl --cacert ca.crt <https://172.63.95.23:6443>
    - But it will not allow because you are anonymous (don’t have password).
    - But there are some API which don’t require username-password.
  + Curl --cacert ca.crt <https://172.63.95.23:6443/version>
    - You can connect it.
  + Curl --cacert ca.crt <https://172.63.95.23:6443/api/vi/namespaces/default/pods>
    - It will require password.
    - Kubectl behind the seen use this type of URL.
  + Here we require certificate cacertificate all things in separate file
    - Copy certificate in admin.crt file from /root/.kube/config.
    - Copy key in admin.key file
  + Curl --cert admin.crt --key admin.key --cacert ca.crt <https://172.63.95.23:6443/api/vi/namespaces/default/pods>
  + But you have change format of certificate.
    - Cat admin.crt | base64 -d > admin\_final.crt
    - Cat admin.key | base64 -d > admin\_final.key
  + Curl --cert admin.crt --key admin.key --cacert ca.crt [https://172.63.95.23:6443/](https://172.63.95.23:6443/%20/) 
    - It will list you all the API’s available in k8s world.
    - You have API’s for logs also you can integrate it with splunk or some other tools.
  + Kubernetes create a user with least privilege.
    - When new user created they don’t have any power.



* + - Here we have multiple folder (API) which are used when you run code.
    - This folders are known as **API groups**.

**Session 19 – cluster role**

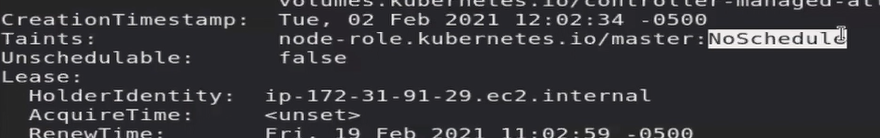
* Suppose you are creating wordpress mysql cluster using Kubernetes.
* We have 2 separate namespaces for this.
* There are certain resources which we don’t put in namespaces, we put it inside cluster, this are known as **cluster resources**.
  + They come under scope of cluster.
  + PV & SC are comes under cluster scope.
    - They are common.
    - Whenever any namespace require it will given to that namespace.
  + We are creating a role and bind with one of the namespace request (Eg, PV request) it is known as **cluster role binding**.
* Master will create a kubeconfig file send to storage admin.
  + Master will create a cluster role.
  + Master will bind this cluster role with storage admin.
* Cd /etc/Kubernetes/pki
  + Openssl genrsa -out sadmin.key 1024
  + Openssl req -new -key sadmin.key -out sadmin.csr -subj “CN=sadmin/O=storage”
    - This is request certificate we have to sign it.
  + Openssl x509 -req -in sadmin.csr -CA ca.crt -Cakey ca.key -CAcreateserial -out sadmin.out -days 365
    - Sadmin.crt & sadmin.key we have pass to our storage admin.
    - We require config file.
    - We already have it for kubectl so we can use it directly.
  + Cp /root/.kube/config sadmin.kubeconfig



* + - Set current context to sadmin@kubernetes
    - Remove client -certificate & client-key data.
    - We have copied admin file so this remove credentials we have to remove.
  + Kubeconfig need data in base64 encode format.
    - Cat sadmin.crt | base64 -w0
      * Here base64 will encode it line by line.
      * But we don’t need new line after each line so we have to use w0.
    - Now add this all inside sadmin.kubeconfig file.
* Now we have to transfer this to our storage admin.
* Now you can use your local windows.
  + Kubectl get pods --kubeconfig sadmin.kubeconfig (client)
  + But right now this user don’t have any power, so he cant do any thing.
  + Here we don’t want ti give power in any namespace, we need to give power of storage, PV and all.
* Kubectl api-resources --kubeconfig sadmin.kubeconfig
  + It will show you resources belongs to namespace or not.
  + Kubectl create clusterrole mysadminrole --verb=get,list,watch,create --resources=pv
  + Kubectl create clusterrolebinding myclusterroleforstorage --clusterrole=mysadminrole --user=sadmin
* Master has role called cluster-admin.
  + If you bind this with any user then he has complete access to cluster.
  + He will also has all power of admin.

**Session 20 – Taint/tolerance**

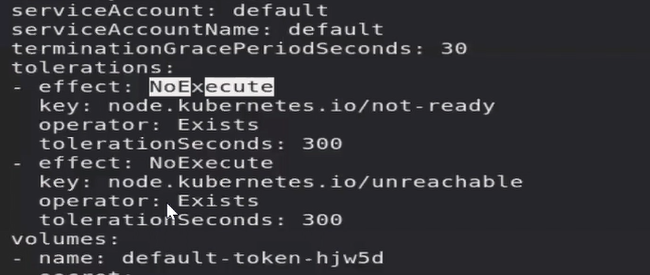
* We are using linux to connect with AWS multimode cluster.
* For this we have to pass sadmin.kubeconfig file to linux.
* But you know we have to always pass this kubeconfig file.
* So we can do one thing.
  + Create .kube directory and put sadmin file here.
  + Now you can check with kubectl get pods.
  + Kubectl describe nodes <master-node>



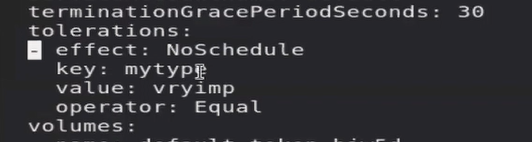
* + It showing NoSchedule.
    - Means you cannot launch pods in master node.
    - Master node is **tainted** that they are working with policy NoSchedule.
    - Kube schedular may decide to launch pod in master, but master will not allow this.
  + Kubectl explain pod.
    - It will give you details of keyword used in yaml file for pod.
    - You can use --recursive option to get complete list of keywords.
  + Kubectl run mypod --image=vimal13/apache-webserver-php --dry-run
    - Dry-run is a way to test something.
    - It will show you we have some syntax error or other errors.
    - Now you can create yaml file using this.
  + Kubectl run mypod --image=vimal13/apache-webserver-php **--dry-run** -o yaml > mypod.yml
* Sometimes because of some company policy or some other reason instead of schedular we (**node**) want to decide where to launch.
* We can add labels to node.
  + Now node decide allow this pod to launch in my node or not.
  + Kubectl label node <node\_name> region=US
  + Kubectl label node <node\_name> storage=HUGE
* In yaml you have to write **nodeSelector** key word in specification of pod.



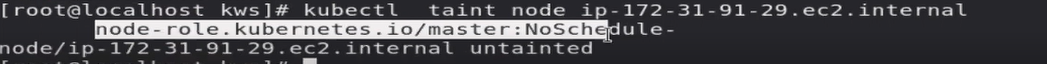
* + Suppose we have tag w1 node for this purpose.
  + But schedular has already launched 2 pods there for some other purpose.
  + They belongs to different team and namespace.
  + So dev cannot see this pods are already running.
  + For this we can tell our node w1.
    - Don’t allow anyone only allow these things.
    - This is known as taint.
    - This type is known as NoSchedule.
    - We have 2 mode types.
      * noExecute & noprefer
      * Taint is basically block all pods only allow those pods, for which they are meant.
  + Kubectl taint node <ip-172-…ec2.internal> mytype=vryimp:NoSchedule
    - Mytype=vryimp is a key value pair, we have to use it in tolerance.
  + Now by default all pods are blocked and cannot be launched in node w1.
  + If you want to launch pod in w1 then you have to make your pod tolerance.
    - By default all pods work with intolerance.
  + Kubectl edit pods myapp1



* + - Change effect to NoSchedule.
    - Change key to your key and give it value.
    - Change operator to equal.
    - Remove other things.



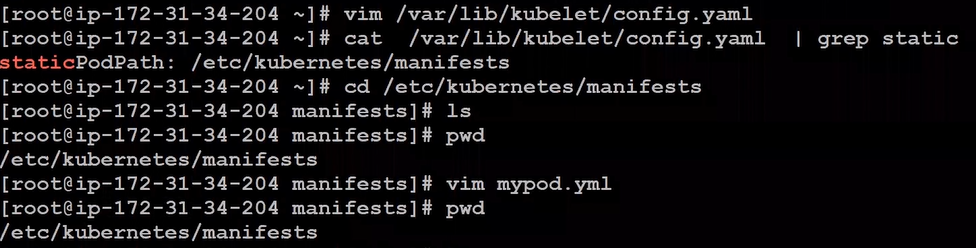
* + Now if you see your pod will schedule to particular node.
  + You can use – at the end of key to remove taint from any node.



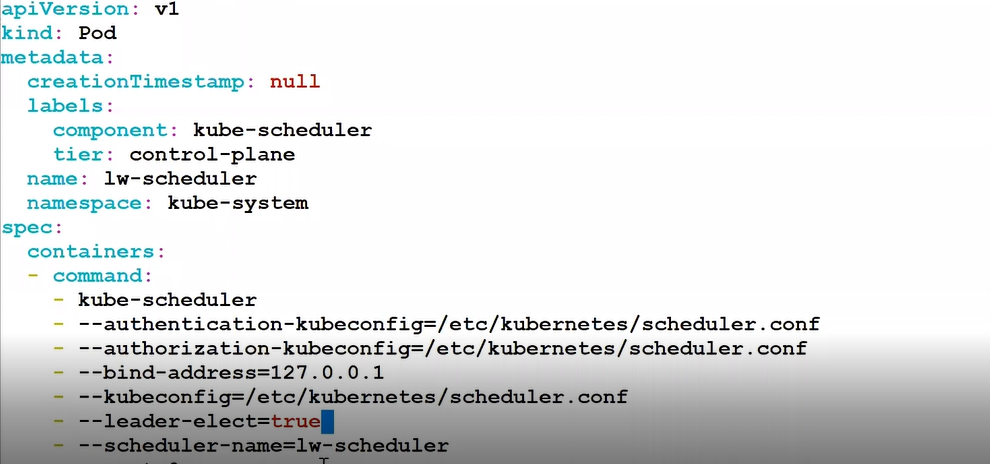
* Here we are using **NoSchedule** type of taint.
  + So we cannot launch new pod in restricted nodes.
  + But older pods are already running.
* **preferNoSchedule**
  + They will not prefer this node for schedule a pod.
  + But all other nodes has not enough resources, and only this node left, then it will launch pod in this node.
  + It is a last option.
* **NoExecute**
  + It will not allow you to create a new pod.
  + Older pods which are in running set is also down.
  + They are running with deployment so they will try to launch in other nodes.
* One of the best uses of NoExecute is maintenance.
  + It will remove all pods from one node and transfer to another node.

**Session 21 – Daemonset//static pod**

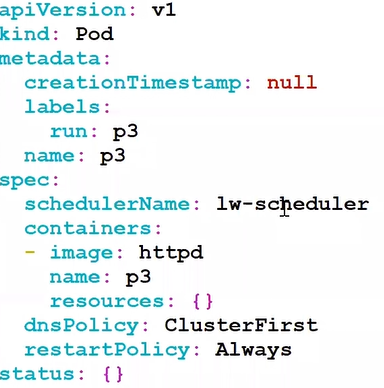
* We can directly use AMI of Kubernetes & then we have to run just one command.
* Kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem --node-name=master
  + It will setup node name is master.
  + Till now we are using hostname as a node name.
* <https://docs.google.com/document/d/1357-iK9iah7aM9RI1S1s-9ibpCP7jeG0tW2qsyym9dQ/edit?usp=sharing>
  + You can use this, it contains complete quick setup for launching cluster on AWS.
  + You can also launch pods without master.
  + We can directly use kubelet connect to kubelet and launch pods (not recommended).
* Kubelet has his own config file located /var/lib/kubelet/config.yaml
  + In config file you can see keyword **staticPodPath** keyword.
  + You can use this location to launch **static pods**.
    - When you create any file here for pod, as soon as you save your file it will launch a pod for you.



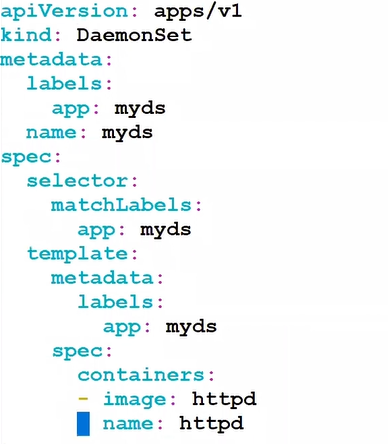
* + This pod is controlled by node itself so master cannot delete this.
  + All pods available in master as a system pods are static pods.
  + You can check this in above mentioned file.
    - If you remove schedular, there will be no one to decide where to launch pod so no pods can be launched.
    - If you want to change schedular open schedular file, change name & image.
      * You can also create a new schedular using a programming language.
  + Here we also have existing default schedular.
    - Second schedular we are using for **fail over**.
    - Here for this we have to add some keywords which is already available in file.
    - If you want both schedular to run then make both schedular leader.
    - You have to put both schedular **leader** true for working of them.



* + - Cp kube-schedular.yml /etc/Kubernetes/manifests/my-schedular.yml
    - Kubectl create -f my-schedular.yml
  + Create a yaml file using dry run.



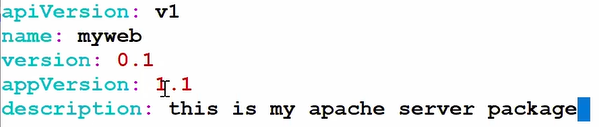
* + - You have to mention in this file which schedular you want to use, by default they use default schedular.
* Daemon set give you guaranty that if you launch 7 copies they will launch 7 copies in different-different nodes.
* Rs will run with daemon set.
  + For creating a daemonset you have to just make kind=daemonset in deployment file.



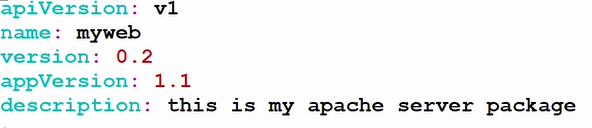
* + If you create this file then you can see they launch pods in all nodes.
  + Kubectl get ds

**Session 22 – helm//chart**

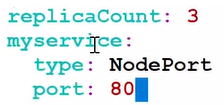
* Helm is k8s own package manager.
* In k8s our end goal is deploying a app.
  + But for this we have to run lots of yaml.
  + Secret svc deployment and many other.
* We can also create variables for running packages.
  + Suppose while launching you can decide you want to run this at port 80 or 81 or any other.
  + We may have multiple micro services and instead of running every command we can use package.
* Instead of this click on package and it will run all things.
  + Eg, for redhat we have package manager yum/dnf.
  + They will also solve dependencies for you.
* In k8s world package is known as **chart**.
  + We can use tool called helm to install chart.
* we can download chart from **public repo** or **hub**.
  + Or we can also manually create own chart known as **custom chart**.
  + Install helm in workstation.
  + You can use this link to install helm.
    - <https://helm.sh/docs/intro/install>
* Helm currently has 2 versions.
  + Version 2
  + Version 3
* Version 2 requires **tiller**.
  + But version 3 comes up with a new architecture and it is completely different from version 2.
  + It does not require tiller.
  + In version 2 if you want helm connect to k8s and run something for uh then you need tiller program in k8s server.
    - You have to launch pod there.
    - It will work as server and helm as a client.
  + In version 3 there is no server program exists.
* Helm will directly use helm.
  + For linux use Linnu amd64.
  + Wget <https://get.helm.sh/helm-v3.5.2-linux-and64.tar.gz>
  + Tar -xvzf helm-v3.5.2-linux-and64.tar.gz
  + Cp linux-amd64/helm /usr/bin/
  + Helm version
* Mkdir /ws
  + cd ws
  + mkdir myweb
    - now you have to install all things here.
    - When you want to give it to someone send him this complete folder.
  + Vim chart.yaml



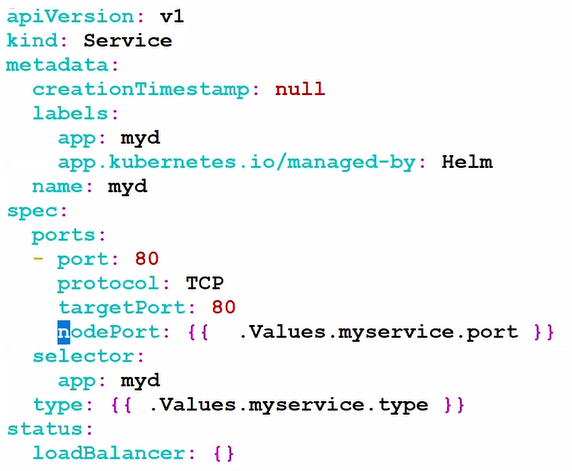
* + Helm install myweb myweb/
    - This will install software for you.
  + Helm list
    - Right now it is not doing anything.
  + Helm delete myweb
    - for helm all resources are known as template.
    - So we have to create template folder for templates.
  + Mkdir templates
    - Cd templates
    - Kubectl create deployment myd --image=vimal13/apache-webserver-php:v1 --dry-run -o yaml > deployment.yaml
  + Helm install myweb myweb/
    - Now if you see one deployment has been launch.
    - Kubectl expose deploy myd --port=80 --type=NodePort --dry-run -o yaml > service.yaml
  + Now we want to update our running chart.
  + So upgrade your chart file.



* + Helm upgrade myweb myweb
  + Kubectl get svc
    - You can see one more service comes here.
  + Helm history myweb
  + Helm rollback myweb 1
* Suppose you know your final team wants to change these things.
  + For this we have to create a variable file.
  + Vim values.yaml



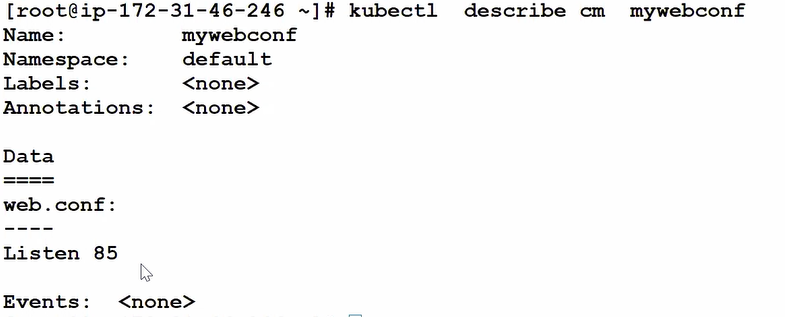
* + - Add variable to deployment file.



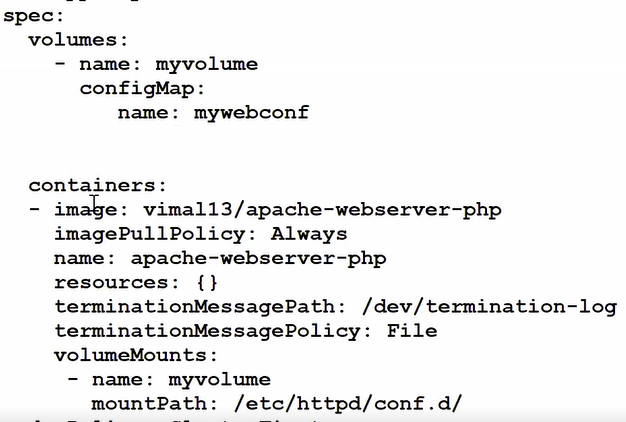
* + Helm show values myweb/
    - It will show you variable.
  + Helm install akws myweb/ --set myservice.port=30003
* It has hub called **artifact hub.**
  + Helm repo list.
    - Currently helm don’t know from where to download.
  + Helm repo add bitnami <https://charts.bitnami.com/bitnami>
  + Helm install my-release bitnami/wordpress
    - But PVC require storage class or PV, so it will show you in pending state.

**Session 23 – configMap//CoreDNS**

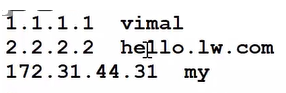
* Most of time we install httpd in top of linux.
  + Then we run service, but many times we use config file and change document root or port.
  + For this we have to login inside port.
  + But in terms of management this is not good to launch pod and then login inside pod.
    - If you are using rs, you have to manually login to every pod and change there.
    - Instead of this we can directly create image and use it for launching pod.
    - But many times requirement comes and we have to change it.
  + So we can do one thing.
    - We can store all config file in some **centralized volume** and **mount** with /etc/httpd/conf.d/
    - For this k8s has one resource for manage config files, this storage resource is known as **ConfigMap**.
* Create a file in your client computer.
  + Kubectl create configmap mywebconf --from-file=web.conf
    - Your centralized configmap storage has been created.
  + Kubectl get cm
  + Kubectl describe cm mywebconf



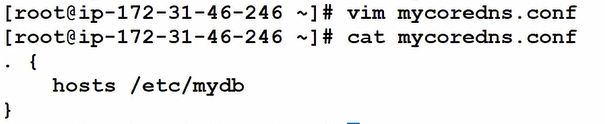
* + Now you can directly change config file using kubectl edit.
  + In configMap world, file name is known as key and data is known as value.
    - Now we have to link this with pod or deployment.
    - We have already launch deployment so we can directly edit it.



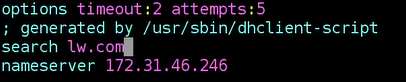
* + We have to specify configmap in spec and mount it inside pod.
* **CoreDNS**
  + Suppose you have 2 worker node A & B.
  + Now instead of IP A connect to B using IP it will connect using a name.
  + For getting IP of this name they will reach to master which is known as resolve or lookup.
  + For this purpose we have to make system A DNS client.
  + For creating DNS one of the famous DNS product is **BIND**.
  + In k8s we use **coreDNS product**.
    - It supports plugins & light weight software.
    - Wget <https://github.com/coredns/coredns/releases/download/v1.8.3/coredns_1.8.3_linux_amd64.tgz>
      * Download above link for linux.
    - Tar -xvzf coredns….
    - Create a database for DNS.
  + Vim /etc/mydb



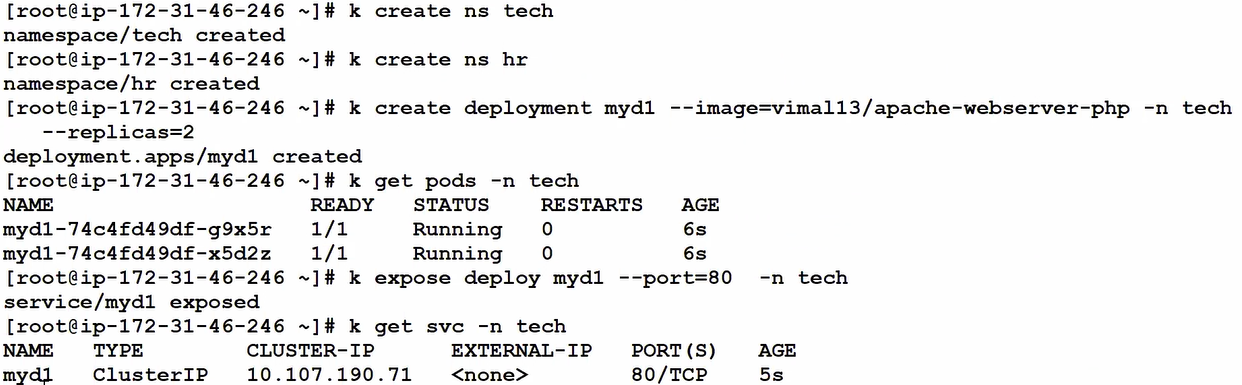
* + Coredns also has his config file.
  + We have to tell config file to read from this database.
    - Vim coredns.conf



* + ./coredns -conf mycoredns.conf
* We have to make other system as a DNS client.
  + Vim /etc/resolv.conf
  + Give IP of DNS server in this file.



* + Now you can ping with hostname.
  + Search will automatically add lw.com at end of hostname.
  + After restart or something IP may be changed, But DNS server has to update IP.
    - But for client perspective there is no any change you can see.
* So we don’t want to change IP and do all entry manually.
  + We want dynamically IP changed for all resources.
  + This thing is known as **service discovery**.
* For solving fail over we launch 2 coredns pods with deployment.



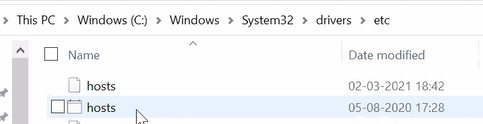
* + Launch same things in hr ns also.
* Here for connecting to pods we connect to load balancer.
  + But IP may be changed next time.
  + Here we use coreDNS.
    - Coredns keep on checking all services running in k8s, they use servicce name for for giving hostname and dynamically create a database.
    - Here coredns using **Kubernetes plugin**.
    - You can go inside any pod and check /etc/resolv.conf
    - This file automatically created.
  + You can check this with below command.
    - Curl <http://myd1.tech.svc.cluster.local>
      * Here they are using conversation like, svc name, ns, svc keyword, cluster.local as domain name.

**Session 25 – Ingress**

* In google we have multiple products running with same domain name google.com.
  + We have search engine mail service and many other services.
  + Multiple service individually launched with hundreds or thousands of products.
  + We have multiple teams working on a single App.
  + We have one load balancer which will send a traffic to related OS.
  + If you searched google.com/search then It will route you to the search engine.
  + If you search for google.com/mail then it will route you to mail service of google.
  + For this we require level 7 or application level load balancer.
  + They can read hostname/path?key-value pair.
    - They will route you according to these values.
    - Its also called **path-based routing**.
    - In k8s application layer load balancer is known as **Ingress**.
  + We are using here nginx as a ingress controller.
* Minikube has one addon for this.
  + Minikube addons enable ingress
  + Kubectl get pods -n kube-system
  + Now we have to create a rule.
  + Here we don’t have to worry about what underlying architecture you are using.
    - Eighter azure alb, HAproxy or Nginx.
    - As a admin you have to only create a rule, /search, /path where to route and all.
  + K8s has created one resource for implementing ingress.
  + Resource is also known as ingress.
  + Ingress resources is exactly equal to rules.
    - Kubectl get ingress.
* For this first of all we are going to make this scenario.
  + We have multiple search pods exposed using searchlb service, and mail pods using maillb service.
  + Create a dockerfile.



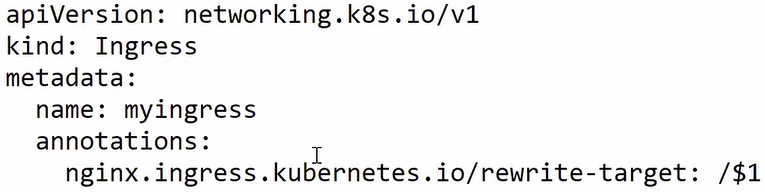
* + - Docker build -t vimal13searchapp:v1
  + Create same with mail app.
    - Docker push vimal13/mailapp:v1
    - Docker push vimal13searchapp:v1
  + Now launch pods using deployment.
    - Kubectl create deployment maild --image=vimal13/mailapp:v1
    - Kubectl scale deploy maild --replicas=2
    - Kubectl create deployment searchd --image=vimal13/searchapp:v1
    - Kubectl scale deploy maild --replicas=3
    - Kubectl expose deploy maild --port=80 --type=NodePort
    - Kubectl expose deploy seachd --port=80 --type=NodePort
  + Now we can separate app with port number.
    - Eg, 192.168.99.100:32433, 192.168.99.100:31423
* L-7 load balancer working with http header.
  + If you type http://IP:80/search in URL, IP will go to network layer and other part of URL goes to http header.
  + If you type <http://www.ak.com:80/search>
    - DNS will convert name into IP.
    - IP goes to L-3, but they create a key value pair for hostname, which goes inside L7.





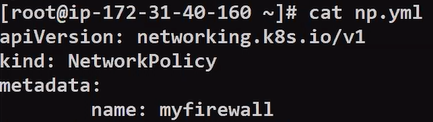
* + Go to hosts file and upload this at last.
  + Here we have to give hostname so instead of changing DNS we can change in local file.
* Create a yml file of it.
  + Here we want to check protocol, hostname & Path all things.
  + We also have to give pathType, it is for matching path.
    - Maximum time we use Prefix.
    - Here /ab & /ab/ both are same.
    - Kubectl create -f lwingress.yml
    - Kubectl get ingress

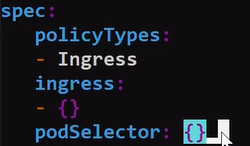

* + - P is capital here.
  + Now if you check with <http://www.ak.com/mail>, it is still not working.
  + Here we have given /mail, work of it to only check & route, but after routing k8s search /mail page, maild service.
    - So we have to rewrite rule and after check route path change /mail to /index.
    - 
  + Here $1 will search for any path given after / and replace it with /
  + What you write after / is stored inside $1
  + Kubectl apply -f lwingress.yml
* Now if you connect with URL you can check this is working fine.
* Sometimes we have multiple company server going on in same OS or in same IP.
  + Here also we must have to type hostname.
  + This concept is known as virtual hosting.

**Session 26 – weave//network policy**

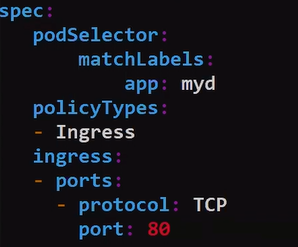
* We are changing overlay network.
* Instead of flannel we are going to use weave.
* We can only allow specific IP to connect to pod or LB.
  + And block all other source IP.
  + For this we are going to do firewall type of setup.
  + In k8s firewall is known as **network policy**.
  + By default, we don’t have any network policy.
* For this we are going to launch a new cluster.
  + We have used image of Kubernetes pre setup, we don’t setup flannel we just have launch cluster & init command.
  + In worker node we just use join command.
  + Now we setup network.
  + Kubectl apply -f [https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base 64 | tr -d ‘\n’)](https://cloud.weave.works/k8s/net?k8s-version=$(kubectl%20version%20|%20base%2064%20|%20tr%20-d%20‘\n’))
    - We can install **weave** using this.
    - Here we have one more extra feature which is not provided by flannel, is network policy.
    - Cd /etc/cni/net.d
    - Here Kubernetes put their conf file which he is using for networking.
      * Here you can see weave file
  + Cd /opt/cni/bin
    - All network related command, Kubernetes put here.
    - Kubernetes will use this command behind the seen.
* Kubectl create deployment myd --image=vimal13/apache-webserver-php --replicas=3
  + Kubectl expose deploy myd --port=80 --type=NodePort
  + Here you can easily connect to website.
  + Now we are going to configure network policy for it.
  + Kubectl get networkpolicy
  + Vim np.yml



* + - Explicit policy for network policy is deny all.
    - So if you don’t write anything in specification, it will block all.
    - If you want to allow all, you have to do **curly { }** braces.



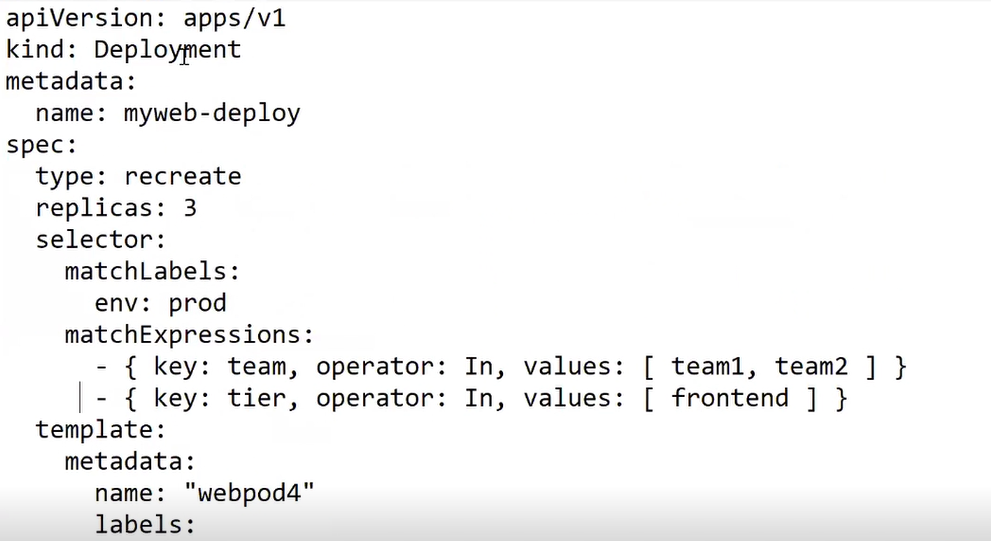
* + This way you can allow some specific ports.



* + - Kubectl apply -f np.yml

**Session 28 – Statefulsets//multi container pod//init container**

* Sometime we want that multiple programs run parallelly.
* If any one program will fail, we want to terminate complete setup.
* Eg, we have one webserver and one monitoring program (splunk elk something), we want that if any one fail then terminate complete setup and create a completely new environment.
* For this we can use **multi container pod**.
  + We put both container inside one pod, if any of the container fails then k8s will terminate entire pod and create a new pod.
  + Here main container is webserver and second one is **helper**.
  + Here c1 is webserver & c2 is monitoring.
* If c1 & c2 exchange data with each other and do all things locally, then this design is called **sidecar**.
* Instead of storing log local in c2 container, we want it will send log to the centralized program.
  + Here before sending data to centralized server they will some pre-processing.
  + This design is called **adaptor**.
* Same as adaptor but they will not do
  + It is known as **ambassador** design.
  + It is also known as **proxy**.
* For communicate to one another one approach is **shared volume**.
  + Both share a same volume.
  + Web server will store log in this shared volume and monitoring program will work on this.
  + Here k8s will give only one IP to any one container.
  + You can go to 1st container using this syntax.
    - Kubectl exec -it mc1 -c 1st --bash
* Use **IPC** for communicate.
  + Inter process communicate.
  + Also known as **shared memory** way.
  + Both containers will share the ram.
  + Kubectl apply -f multi\_ipc.yml
  + You can specify in yaml file that after completing task you want to container to be run or not.
    - We can apply this anywhere.
  + We have 3 type of restart policy for this.
    - Always
      * Restart container always.
    - Never
      * Once task complete it will terminate our pod.
    - Onfailue
      * Only restart pod if last time container fails without completing task.
* Use **network method** to communicate.
  + They can communicate with their IP address and port number.
  + Kubectl apply -f multi\_network.yml
* We want first container run and finish his task and after this second container will run.
  + So for this we have to run second container inside **init containers**.
  + Containers inside init container will run one by one.
* **Session 01 – kubernetes//pod**
  + Monitoring OS
  + More OS, hard to manage
  + Kubernetes
  + Pod
    - Auto scaling
    - Load balancer
  + Multiple containers in one mode
  + Master-slave architecture
  + Multi-node cluster
* **Session 02 –minikube//kubectl//demo**
  + Minikube installation
    - Only support vbox
    - <https://minikube.sigs.k8s.io/docs/start/>
  + Client command
    - Kubectl
    - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
  + Commands
    - Kubectl.exe get pods
    - Kubectl.exe run --name=myweb1 --image=vimal13/apache-webserver-php
    - Kubectl.exe delete pod myweb1
    - Kubectl.exe create deployment myweb1 --image=vimal13/apache-webserver-php
    - Kubectl.exe describe pods
    - Kubectl.exe expose deployments myweb1 --port=80 --type=NodePort
    - Minikube service myweb1 --url
    - Kubectl.exe scale deployments myweb2 --replicas=4
  + Minikube dashboard
  + Minikube vm
    - Login = docker
    - Password = tc-user
    - Default IP for minikube VM is 100 (192.168.43.100)
* **Session 03 – yaml//kubectl//kubelet//keywords**
  + Kube API server
  + Kubelet
  + Yaml language
  + commands
    - Kubectl delete all –all
    - Kubectl.exe describe pods myweb
    - Kubectl create –f pod.yml
    - Kubectl create –f pod.yml
  + Keywords
    - apiVersion
    - kind
    - metadata
    - spec
  + configure path for kubectl
* **session 04 – rc//template**
  + docker private n/w
    - patting
    - Docker run -it --name myos -p 1234:80 httpd
    - Expose port
  + Replication controller
  + Template
  + Tag-label
  + Load balancer
    - Service
  + Replication
  + Commands
    - Kubectl get pods -L app
    - Kubectl create -f new.yml
    - Kubectl apply -f rc.yml
    - Kubectl describe rcmyrc
    - Kubectl expose rcmyrc --port=80 --type=NodePort
* **Session 05 – service**
  + Load balancing
    - clusterIP
    - NodePort
    - External (LoadBalancer)
  + Kubeproxy
* **Session 06 – wp\_mysql//multi\_tier\_architecture**
  + Entrypoint
  + Wordpress with NodePort
  + Mysql with ClusterIP
  + commands
    - Kubectl run mydb --image=mysql5.7
    - Kubectl logs mydb
    - Kubectl run mydb --image=mysql5.7 --env=MYSQL\_ROOT\_PASSWORD=redhat --env=MYSQL\_DATABASE=wpdb --env=MYSQL\_USER=akshit --env=MYSQL\_PASSWORD=redhat
    - Kubectl run mywp --image=wordpress
    - Kubectl expose pod mywp --type=NodePort --port=80
* **Session 07 – secret** 
  + Secret resource
  + Reference inside env variable
  + Encode Base64
  + secretKeyRef
  + Kubectl get secrets mysecret -o yaml
* **Session 08 – deployment** 
  + Replicaset
  + Scale out—scale in
  + No downtime
  + 100% uptime
    - 25% max surge
  + Strategies
    - Recreate
    - Rolling update (ramped)
  + Rollout
  + Commands
    - Kubectl create deployment myd --image=vimal13/apache-webserver-php:lwv1
    - Kubectl get rs
    - Kubectl desc myd
    - Kubectl scale deployment myd --replicas=3
    - Kubectl expose deployment myd --port=80 --type=NodePort
    - Kubectl set image deployment apache-webserver-php=vimal13/ apache-webserver-php:lwv2
* **Session 09 – RS//Deployment-yml**
  + Replicaset
    - Updated of replication controller
  + Selector
    - Equality-based
      * matchLabels
    - Set-based
      * MatchExpression



* + Commands
    - Kubectl get pods -l “team!=team2” –show-labels
    - Kubectl get pods --show-labels -l “region in (IN,US)”
    - Kubectl get pods --show-labels -l “region in (IN,US),team notin (team1)”
    - Kubectl get deploy myd
    - Kubectl get rs --show-labels
* **Session 10 – PVC//PV//SC**
  + Docker storage
    - Permanent but temporary
    - Ephemeral
    - OS restart data is there
    - Container destroy data destroy
  + K8s for persistent storage
  + PV plugin
  + PVC request
  + PV bound to PVC request
    - Create manual PV
    - PV dynamically
      * Storage Class
  + Commands
    - Kubectl get pv
    - Kubectl get pvc
    - Kubectl get sc
* **Session 11 – practical PVC//PV**
  + Storage access
    - RO
    - RW
      * ReadWriteOnce
      * ReadWriteMany
  + hostPath
  + Reclaim policy
    - Retain
      * Default policy
      * PVC will not be deleted
        + Available in your storage
      * Cannot use this PVC in other purpose.
    - Recycle
      * PVC deleted
      * PV not deleted, can be use for other purpose
    - Delete
      * Used for cloud storage
      * Once PVC deleted, EBS volume also deleted
  + Commands
    - Kubectl delete pvc mypvc
    - Kubectl edit pvc mypv
      * To change in online file.
* **Session 12 – practical of SC//NAS storage**
  + Storage class
    - Self service
    - storageClassName: “”
  + hostPath
  + NAS storage for static PV
    - Client-server
  + commands
    - systemctl start nfs-server
    - exportfs -v
* **Session 13 – Multi node cluster on AWS**
  + Container runtime
  + CRI interface
  + API server
  + Kube-controller manager
  + Kube schedular
  + Kubelet Agent program
  + Etcd
  + Worker node connect to master send JOIN request
  + Configuring master client on aws
    - Yum install kubeadm --disableexcludes=Kubernetes
    - Systemctl enable kubelet --now
    - Kubeadm config images pull
    - Vi /etc/docker/daemon.json
    - Yum install iproute-tc
    - Kubeadm init --pod-network-cidr=10.240.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
    - Vim /etc/sysctl.d/k8s.conf
    - kubectl apply  -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>
    - kubeadm join 172.31.40.201:6443 --token 6iqkhl.f143xnv9iubk4zyg --discovery-token-ca-cert-hash sha256:6463cbfb6f25e953db0cb7c535ad8e3f167764d3022a6edb58514f39c5062049
* **Session 14 – overlay//flannel**
  + Multi tenancy
  + Kube-system
  + Software switch=bridge
  + Overlay network
  + Tunnelling
  + Kube-proxy
  + Commands
    - Brctl show
    - Vim /var/run/flannel/subnet.env
    - Kubectl edit configmap kube-flannel-cfg -n kube-system
    - Kubectl get pod -l app=flannel -n kube-system
* **Session 15 – client-passowrd** 
  + K8s allow limited IP to connect using kubectl
  + Etcd
    - Secret database
    - Flannel data
  + Commands
    - Kubectl get pods --kubeconfig admin.yml
* Session 16 – security//IAM//role
  + Verb
  + Role
  + Role-Based access control
  + Certificate-based authentication
    - CSR
    - CRT
  + Commands
    - Kubectl cluster-info
    - Openssl genrsa -out vimal.key 1024
    - Openssl req -new -key vimal.key -out vimal.csr
    - Openssl x509 -req -in vimal.csr -CA ca.crt -CAKey ca.key -CAcreateserial -out vimal.crt
    - Kubectl config --kubeconfig vimal.kubeconfig set-cluster awskubecluster --server <https://54.52.63.112:6443> --certificate-authority=ca.crt
    - kubectl config view --kubeconfig vimal.kubeconfig set-credentials vimal --client-certificate vimal.crt --client-key vimal.key
    - kubectl get pods --kubeconfig vimal.kubeconfig
* **Session 17 – context//role binding**
  + Bind role with role binding
  + Commands
    - Kubectl config get-clusters --kubeconfig vimal.kubeconfig
    - Kubectl config set-context vimal@awskubecluster --user=vimal --cluster awskubecluster --kubeconfig vimal.kubeconfig
    - Kubectl config use-context vimal@awskubecluster --kubeconfig vimal.kubeconfig
    - kubectl create role vimal-tech --resource=pods --verb=get, list -n tech
    - Kubectl create rolebinding vimal-tech-rb --role vimal-tech --user vimal -n tech
* **Session 18 – API group**
  + Rest-API’s
  + API groups
  + Commands
    - Curl --cert admin.crt --key admin.key --cacert ca.crt <https://172.63.95.23:6443/api/vi/namespaces/default/pods>
    - Curl --cert admin.crt --key admin.key --cacert ca.crt [https://172.63.95.23:6443/](https://172.63.95.23:6443/%20/)
* **Session 19 – cluster role//cluster role binding** 
  + Cluster resources
  + Cluster role binding
  + Commands
    - Openssl genrsa -out sadmin.key 1024
    - Openssl req -new -key sadmin.key -out sadmin.csr -subj “CN=sadmin/O=storage”
    - Openssl x509 -req -in sadmin.csr -CA ca.crt -Cakey ca.key -CAcreateserial -out sadmin.out -days 365
    - Cp /root/.kube/config sadmin.kubeconfig
    - Kubectl api-resources --kubeconfig sadmin.kubeconfig
    - Kubectl create clusterrole mysadminrole --verb=get,list,watch,create --resources=pv
    - Kubectl create clusterrolebinding myclusterroleforstorage --clusterrole=mysadminrole --user=sadmin
* **Session 20 – Taint//tolerance** 
  + Tainted
    - * Key-value
    - NoSchedule
    - NoExecute
    - **preferNoSchedule**
  + Dry-run
  + Commands
    - Kubectl describe nodes <master-node>
    - Kubectl taint node <ip-172-…ec2.internal> mytype=vryimp:NoSchedule
    - Kubectl explain pod
    - Kubectl label node <node\_name> storage=HUGE
* **Session 21 – Daemonset//static pod//Schedular**
  + Use AMI for launch kube cluster.
  + Static pods
  + staticPodPath
  + multiple schedular
  + daemonset – per node 1 Pod.
* **Session 22 – helm // chart**
  + Helm version 3
    - No tiller required.
  + Public repo
    - Artifact hub
    - helm chart
  + Commands
    - Wget <https://get.helm.sh/helm-v3.5.2-linux-and64.tar.gz>
    - Tar -xvzf helm-v3.5.2-linux-and64.tar.gz
    - Cp linux-amd64/helm /usr/bin/
    - Helm install myweb myweb/
    - Helm list
    - Kubectl expose deploy myd –port=80 --type=NodePort --dry-run -o yaml > service.yaml
    - Helm upgrade myweb myweb
* **Session 23 – configMap//coreDNS**
  + Centralized volume for config files.
  + DNS products
    - coreDNS
    - BIND
  + Service discovery
  + <http://myd1.tech.svc.cluster.local>
    - svc name, ns, svc keyword, cluster.local
  + Commands
    - Kubectl create configmap mywebconf --from-file=web.conf
    - Kubectl describe cm mywebconf
    - ./coredns -conf mycoredns.conf
* **Session 25 – Ingress** 
  + Ingress = ALB
  + Path based routing
  + Virtual hosting
    - Multiple company running in same OS.
    - Hostname defines where to route.
  + Commands
    - Minikube addons enable ingress
    - Kubectl get ingress.
* **Session 26 – weave//network policy**
  + Weave comes up with network policy facility.
  + Network policy == firewall // Security group
  + {} -- allow all