KUBERNETES

- Kubernetes we called k8s. It is having high demand because of it's high availability. And it contains many regions
- Why we're using K8S instead of Kubernetes, because of laziness. Ancient days Scientists don't like to pronounce big words. So, they reduced to K8S. Not only Kubernetes they reduced lot of technical terms

Relation between Docker & Kubernetes

- If you see the docker logo, ship contains containers. That means docker is used to create the containers
- For that created containers handle/managed by K8S. i.e. K8S controls the containers

That's why docker & k8s logos are very meaningful

History

- K8S was developed by GOOGLE using GO language
- In Google there is a specific tool called Borg, It is not open source. where they say that K8S is one of the part of Borg
- Borg is not a open source. So, the people at Google built a
- Google donated K8S to CNCF (Cloud Native Computing Foundation) in 2014
- K8S first version was released in 2015

K8S

- It is an open-source container orchestration platform
  - Orchestration is used in Docker Swarm also
    - i.e. K8S is alternative of Docker Swarm
- It is used to automate many of the manual processes like deploying the application, managing the containers and scaling containerized applications

(or)

- K8s is a popular open source platform for container orchestration. It enables developers to easily build containerized applications and services, as well as scale, schedule and monitor those containers

Container Orchestration :

It automates the deployment, management, scaling and networking of containers. Enterprises that need to deploy and manage hundreds (or) thousands of linux containers and hosts can benefit from container orchestration

WHY K8S ?

- Containers are a good and easy way to bundle and run your applications.
- In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime
- In docker we used docker swarm for this, but any how docker has some drawbacks

(or)

- K8S supports numerous container. Kubernetes services provide load balancing and simplify container management on multiple hosts.

So, that's the reason we moved to K8S

HOW K8S will help you with Docker Container ?

1. Scalability
   - Single docker container cannot serve millions of requests but with the help of k8s we can scale that particular docker container to serve millions of requests
2. Automate Docker Deployment
   - You can automate the docker container deployment inside your k8s cluster. So, you don't need to manually take are of the docker containers and inside our K8S cluster
3. Auto - healing
   - If a docker container is unhealthy then its responsibility of our k8s cluster to do the restart or do the fresh deployment of a docker container inside a K8S Cluster
   - So, K8S will delete the unhealthy container and can auto replace with healthy container
4. Rollout & Rollback
   - If there is something wrong with our docker container then K8S will help you to do the roll back and roll out of a new containers inside our K8S Cluster
   - So, K8S monitors the unhealthy docker container and restart unresponsive

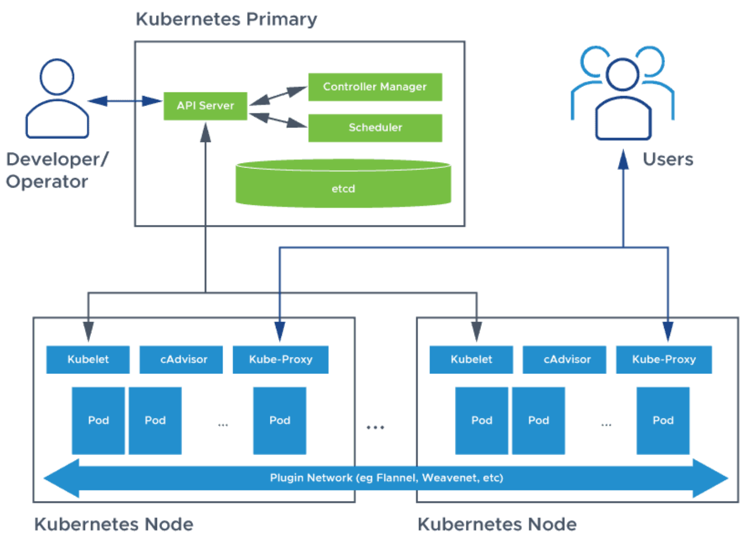
The differences between Docker & K8S

1. Setup
    - docker setup is very easy
    - k8s setup is complex
2. Auto Scaling
    - In docker - no autoscaling
    - In k8s - Autoscaling
3. Community
    - docker having community
    - k8s is having greater community for users like documentation, customer support 24/7 and resources
4. GUI
    - In docker there is no GUI (Portainer is not official gui)
    - In k8s, we are having GUI
5. Single host
    - the docker platform basically relies on one single host
    - In k8s, we are having cluster. Through cluster we can host our containers in multiple nodes
6. Auto healing
    - In docker - no Autohealing, i.e. containers not able to heal automatically
    - In k8s - Autohealing
7. Enterprise level
    - Docker is an simple platform. By default, docker doesn't support enterprise level application support
    - load balancer, firewall, Auto scale, Auto heal, API gateways these are enterprise level standards

So, docker has above drawbacks. And Docker never used in production because it's not a enterprise level solution because it doesn't have the above properties

So, for above issues and problems we're using K8S

K8S MASTER-NODE ARCHITECTURE



Here operator send the request to create a POD. That request was sent through the command. This architecture explains the flow of how to create a Pod

In real time we are having multiple master and multiple worker nodes are present

MASTER/CONTROL PLANE Components

- Operator
    - The person who deals with K8S
- API Server
    - It is the heart of the K8S, every request is received by the API Server
    - It exposes the K8S to outside world, which basically takes all the requests from external world
    - It provides an API that server as the front end of a K8S control plane
    - It will take the request from the user/operator and that will stored into the etcd cluster
- ETCD
    - It is basically a key-value store and the entire k8s cluster information is stored as objects (or) key value pairs
    - without etcd, we don't have the cluster related information
    - If you want to restore a cluster means, etcd is the component which we are using
    - It is a database to store the pending tasks in Key-Value Format (YAML)
- Scheduler
    - Scheduler will check the pending tasks in etcd cluster/database
    - If there is any pending tasks found, scheduler will schedule the job to k8s nodes that means (slaves)
- Controller Manager
    - It is manager for your inbuilt Kubernetes controllers
    - If you want to create the pod in K8S i.e. with the help of controller manager pods will create inside the k8s nodes
    - It helps to execute the scheduler tasks
- Cloud controller Manager
    - It is a open source utility
    - if you are running on your k8s in on-premise. This component is not required

  - If you are running your k8s on cloud platform like EKS. So, here there is a request to create a storage (or) load balancer
  - Instead of doing manually, we are write logic in CCM. So, it will take care everything

NODE/Data Plane Components

- kubelet
  - Agent ensures that pod is running (or) not checked
    - kubelet is responsible for creation of the pods and running the pods/application
  - i.e. In one particular node Kubelet will maintain the pod data whether it is running (or) not
  - It passes about the health status of the k8s node to scheduler
- kube-proxy
  - It is basically providing networking, ip address, and default load balancing
  - It is providing the cluster IPaddress to the pod
  - It uses IP tables on your linux machine
  - It is used to maintain a network connection between servers
  - i.e. worker & manager nodes
- Pod
  - It is nothing but a group of one (or) more containers at one place
- Container runtime
  - It is which actually runs the container
  - It is a virtual machine which does not have any OS
  - It is used to run the applications in worker nodes/severs
  - For that sake, we're using containers
- CAdvisor
  - It is used for monitoring the containers
  - It is not coming into pods architecture. It is separate component

CLUSTER

- By default, K8S is a cluster
- It is nothing but group of servers/nodes
- It will have both manager & worker nodes/servers
- Manager node is used to assigned tasks to worker node
- Worker node will perform the task

Here, we are having 2 types of clusters

1. Self Managed K8S Cluster

   This clusters are created by our own and we have to manage the clusters

Three types of clusters

- Mini Kube
  - It is single node server (or) cluster/ i.e. master & slave
  - It is a tool used to setup single node clusters on K8S
  - It contains API Servers, ETDC database and container runtime
  - It helps you to containerized applications
  - It is used for development, testing and experimentation purposes on local
  - Here, master and workers run on same machine
  - It is a platform independent
  - By default, it will create node only
  - installing minikube is simple compared to other tools
- Kubeadm
  - It is multinode cluster. Here master & multiple slaves are present
  - It is create only server like ec2 machines.
- KOPS
  - It is Kubernetes Operations
  - If we want to work with k8s, definitely we need a cluster
  - It is creating the servers and along with the entire infrastructure like VPC, buckets ,etc.,
  - So, overall if you want create an application related whole infrastructure, we're using KOPS

Note :

We're not going to use minikube in real time because it contains only one server. It can't handle the 1000's of requests. So, that's the reason we're not using minikube.

So, we're using Kubeadm & KOPS

2. K8S Managed Clusters

   This clusters are already defined by K8S, we're having default clusters in cloud

  - AWS EKS
  - Azure EKS
  - GCP EKS
  - IBM IKE
- Using the above clusters we can do our work

KUBECTL

- It is the CLI for K8S. i.e. command line tool.

- Using this kubectl, we can directly interact with k8s clusters
- We can create, manage, pods, services, deployments and other resources through kubectl
- We can also monitoring, troubleshooting, scaling and updating the pods
- To perform these tasks it communicates with K8S API Server
- Without kubectl, we can't perform any command in K8S

Note: Install the kubectl in cluster, when you want to work with k8s in CLI

POD

Pod is the definition of how to run a container.

- In docker, if you want to run a container in CLI, you have to provide the arguments inside the command, like given below
  - docker run -itd --name cont -p 8080:80 -v /volume --networks abc imagename
- But in K8S, we are passing the whole specifications in the pod.yml file

The question is : why can't we use the containers in k8s same as docker?

- Previously, we discussed like k8s is an enterprise level platform. It want to build declarative capabilities (or) standardization. In docker we don't have

In K8S, everything we will written in the form of YAML files

- Pod is nothing but one (or) group of containers
- It is the smallest unit of deployment in K8S
- Pods are ephemeral
- When we create a pod, containers inside pods can share the same network, namespace, and can share the same storage volumes
  - If we place group of containers in single pod, the advantage is k8s will allowed you shared networking, storage.
  - So, this way what happens is inside a single pod, containers will talk to each other using local host Eg: localhost:3000
- A pod is basically a wrapper
  - Let's take there is a pod and inside pod we are having container.
  - So, basically what k8s does is it allocates a cluster IPaddress to the pod, for access inside the applications inside the containers through pod cluster IPaddress
- Mostly we can use single container inside a pod. If we have any requirement, we can create multiple containers inside a same pod
- While creating pod, we must specify image along with any necessary configuration and resource limits. i.e. we have to mention the container specifications
- Pod's are created in Slaves not master
- K8S cannot communicate with containers, they can only communicate with pods

CREATING THE POD

1. IMPERATIVE
   - The imperative way uses kubectl command to create pods
   - This method is useful for quickly creating and modifying the pods

   Syntax : kubectl run pod-name --image=imagename

   Command : kubectl run pod-1 --image=nginx

     - kubectl → command line tool
     - run → action
     - pod-1 → name of the pod
     - nginx → name of image

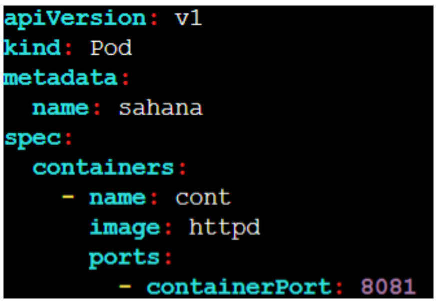
- Check the list of pods present/not in server
  - kubectl get pod (or) pods (or) po
- See the list of pods with full description
  - kubectl get pods -o wide
- Create a Pod
  - kubectl run podname --image=imagename
- Delete the pod
  - kubectl delete pod podname
  - kubectl delete pod --all
- If you want to debug kubectl, what internally happening when you perform kubectl command ?
  - kubectl get pod -v=7
    - Here, we are using verbose statement.
    - When we run the above command, it will gives you the information about API calls in kubectl
    - if we increase the verbose level means, we will get more data in JSON format

2. DECLARATIVE

   The Declarative way we need to create a manifest file in YAML extension

- This file contains the desired state of a pod
- It take cares of creating the pod, updating (or) deleting the resources/pods
- This manifest file need to follow the YAML indentation
- YAML file consists of Key-Value pair

→ vi myprod.yml



```
apiVersion: v1
kind: Pod
metadata:
  name: sahana
spec:
  containers:
    - name: cont
      image: httpd
      ports:
        - containerPort: 8081
```

- apiVersion → it is used to communicate with the master node
- metadata → It will gives the pod related information

Here, we can create (or) apply command to execute the manifest file

Practice

- vim prod.yml

DEMO

Installing the Kubectl and Minikube and Pod

Requirements :

- ubuntu - AMI
- We need 2 CPU and more RAM - t2.medium
- No key-pair
  - if you want terminal in your local. Use key-pair
- 20GB volume
- Regular Security group
  - Rule -1 → All traffic, anywhere
  - Rule -2 → SSH, anywhere

Step -1: First Update the Server

- sudo apt update -y

Step -2: Install Kubectl

- sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
- sudo curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
- sudo echo "$(cat kubectl.sha256) kubectl" | sha256sum --check
- sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
- sudo kubectl version --client
- If you want to see your o/p in YAML format use this command
  - sudo kubectl version --client --output=yaml

Step -3: Install the cluster - Minikube

- sudo curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
- sudo mv minikube-linux-amd64 /usr/local/bin/minikube
- sudo chmod +x /usr/local/bin/minikube
- sudo minikube version

- After installing minikube, you have to start the minikube
  - minikube start → it is asking driver
  - For that, we are installing docker. It is virtual machine manager

Step -4: Install Docker

- sudo apt install curl wget apt-transport-https -y
- sudo curl -fsSL https://get.docker.com -o get-docker.sh
- sudo sh get-docker.sh → execute the docker file
- docker version

Step -5: Start the minikube

- sudo minikube start --driver=docker --force

- [ ] See the MINIKUBE ip address → minikube ip

Create the pod → FYI, read Pod documentation

Getting the output from the pod

- first we login into the cluster, then we can see the pod output
  - minikube ssh
  - curl podIP
    - So, you can get the output in the code format

POD-DRAWBACKS

- From imperative and declarative methods we can be able to create a pod.

  But what if we delete the pods?

- Once you delete the pod, we cannot able to access the pod, So it will create a lot of difficulty in Real time
  - i.e. pod deleted means, container stopped. So application can't access
- i.e. in pods "Self Healing" is not available

  So, to overcome this, we use some K8S components called RC, RS, Deployments, Daemon Sets, Services, etc..,

K8S DEPLOYMENTS

So, you can deploy your application into k8s as pods, then why do you need deployment ?

In deployments, we are having auto scaling and auto healing.

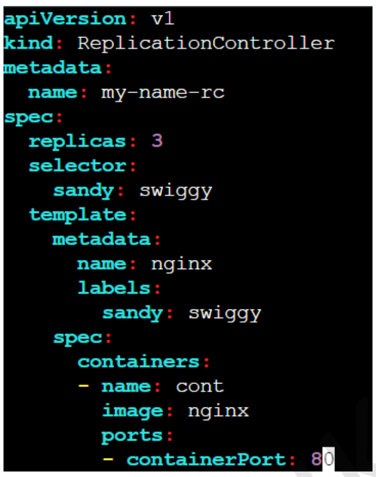
- i.e. if you want to do zero downtime deployments and you need auto healing & Scaling, then you should never deploy your application as pods in k8s. instead you should deploy in deployments
  - but end of the day it will deploy as a pod
- Overall, what k8s is suggesting - Do not create pod directly. But create the pod using the deployment resource

So in deployment, first we will create Replica Set in Deployment. i.e. k8s controller then the replica set roll out our pods

REPLICATION CONTROLLER (RC)

- Replication controller can run specific numbers of pods as per our requirements
- To Create the pods, we're using RC
- It is the responsible for managing the pod life cycle
- It will ensure that always pods are up and running
- If there are too many pods, RC will terminate the extra pods
  - If there are 2 pods are less means RC will create automatically 2 new pods
- This RC will have have self-healing capability
  - If a pod is failed, terminated (or) deleted. Then new pod will get created automatically
- RC use "labels" to identify the pods that they are managing
- We need to specify the desired number of replicas in YAML file
  - vim rc.yml

// write the code



```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-name-rc
spec:
  replicas: 3
  selector:
    sandy: swiggy
  template:
    metadata:
      name: nginx
      labels:
        sandy: swiggy
    spec:
      containers:
      - name: cont
        image: nginx
        ports:
        - containerPort: 80
```

Selector → when the label is matches with selector then only replicas will create

Template → It contains metadata

Labels → Give the same data which is present inside the selector

Image → present, we are taking basic images. But in real-time we took docker hub images

replicas → Here, we're giving the number of pods which you're required

- Execute the file
  - kubectl apply -f rc.yml
- Check 'RC' created/not ?
  - kubectl get rc
- Check the pods
  - kubectl get pods
- See the RC full information
  - kubectl describe rc/rc-name

If you want to increase/decrease scaling. So, normally we can do the changes inside the replicas in code. But, everytime we can't change the code.

We can scale the replicas through CLI

- Scale the pods/rc
  - kubectl scale rc rc-name --replicas=3
  - kubectl get pod
    - you can get the pods
- Delete the pods in RC
  - kubectl delete rc rc-name
- Delete only RC not pods
  - kubectl delete rc rc-name --cascade=orphan
  - kubectl get rc
    - no resources
  - kubectl get pods
    - pods are present
- now, you can delete the pods
  - kubectl delete pods -all
    - pods will deleted, because RC deleted.

We can write multiple pods inside rc.yml. i.e. copy from metadata to end of the line. and change the names in after selector and metadata names. So, it works like that

- If you want to get inside the YAML
  - kubectl get kindOfFile -o filename
  - kubectl get rc (or) rs (or) deploy rc.yml

DRAWBACKS in RC

- RC is used only equality based selector
  - Eg: env=prod,
    - app=swiggy (or) sandy=swiggy
  - We can write like above, but not like below
  - Eg: env=prod, swiggy
- Here, we can assign only one value
- We are not using these RC in recent times, because RC is replaced by RS(Replica Set)

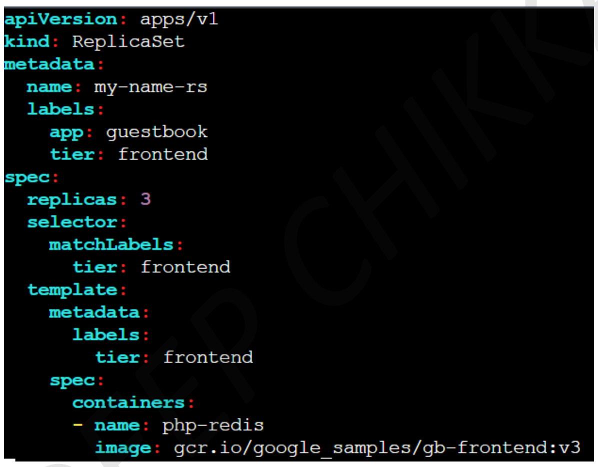
REPLICA SET

- It is nothing but group of identical pods.
- If one pod crashes automatically, replica sets give one more pods immediately
- It uses label to identify the pods

Difference between RC and RS is - selector and it offers more advanced functionality

- The key difference is that the RC only supports Equality-based selectors. Whereas the RS supports Set-based selectors
- It monitoring the no. of replicas of a pod running in a cluster and creating/deleting new pods
- It also provides better control over the scaling of the pod
- A RS identifies new pods to acquire by using its selector
  - i.e. using the selector, it identifies the new pod
- We can provide multiple values to same key
  - Eg: env=dev, prod

vim rs.yml



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-name-rs
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

- matchLabel → Label we want to match with the pods
- template → This is the template of the pods

Here, first we mentioned the label. So, we can use the labels for multiple times through matchLabels

So, overall working functionality is difference in RC & RS. We will get the same output for both RC & RS

In real time, we are not creating the pods directly. Using K8S components like RC, RS, etc..,

K8S SERVICES

Why k8s ?

Whenever a pod created, IP address will change due to lack of self-healing capability. For that we are creating the service using labels and selectors. Through this, we can access the application

- k8s services is a method of exposing pods in our cluster
- Each pod gets its own IP address. But, we need to access from IP of the node
  - i.e. we can't access the application through pod IP. Because pod is present in slave server.
  - So, we're using slave server IP address to access the application because pods are ephemeral
- If you want to access pod/application from inside of the cluster. We're using cluster-IP. But this is not acceptable for clients/users
- If the service is of type NodePort (or) LoadBalancer. It can also be accessed from outside of the cluster
- It enables the pods to be decoupled from the network tropology, which makes its easier to manage and scale applications

ADVANTAGES OF SERVICES

1. Load balancing
2. Service discovery
   - Through labels and selectors, it will track the IP address for pods
3. Exposing applications to world

COMPONENTS OF SERVICES

A Service is defined using a k8s manifest file that describes its properties and specifications. Some of the key properties of a service include:

- Selector
  - It is used to select the label, which is assigned to the pod. Based on this we can add replicas
- Port
  - The port number on which the service will listen for incoming traffic
  - This is host port
- TargetPort
  - The port number on which the pods are listening for traffic
  - Here, container port is target port

TYPES OF SERVICES

In k8s we're having different services present.

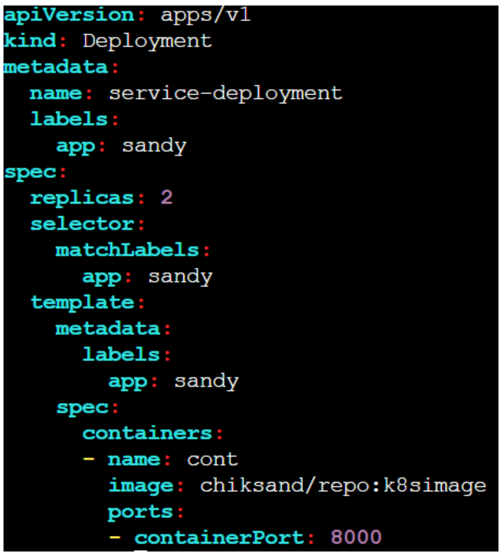
1. Cluster-IP
   - A clusterIP provides a stable IP address and DNS name for pods within a cluster
   - This type of service is only accessible within the cluster and is not exposed externally
2. NodePort
   - This service provides a way to expose a service on a static port on each node in the cluster
   - This type of service is accessible both within the cluster and externally, using the node IP address and the node port
   - when you're trying to access VPC, etc.., use this
3. LoadBalancer
   - This service provides a way to expose a service externally, using a cloud provider's load balancer
   - This type of service is typically used when an application needs to handle high traffic loads and requires automatic scaling and load balancing capabilities
   - When you are using Amazon.com, will use this
4. ExternalName
   - This service provides a way to give a DNS name to service that maps to an external service (or) endpoint
   - This type of service is typically used when an application needs to access an external service, such as database (or) API, using a stable DNS name

See the list of services → kubectl get svc

PRACTICAL - DEEPDIVE

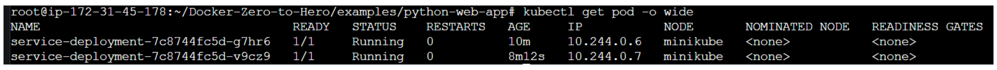
First, do Minikube setup

- Install git → apt install git -y
- Getting the repo in github → git clone Docker-zero-to-hero
- Go to this folder → cd /root/Docker-Zero-to-Hero/examples/python-web-app
- Here, In this folder Dockerfile is present. We need to build the docker file
- And Send that image in docker registry
  - For that, first perform → docker login → provide credentials
  - Build the Dockerfile → docker build -t chiksand/repo:k8simage
  - docker push chiksand/repo:k8simage
- When you perform → kubectl get all
  - Default, we are having clusterIP Service
- Now, we have to create one deployment file
  - vi deploy.yml
    - Here, you have to add the docker image, inside the containers



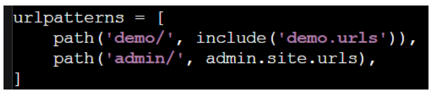
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-deployment
  labels:
    app: sandy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sandy
  template:
    metadata:
      labels:
        app: sandy
    spec:
      containers:
      - name: cont
        image: chiksand/repo:k8simage
        ports:
        - containerPort: 8000
```

- kubectl create -f deploy.yml
- kubectl get all
    - you will get services, pods, deployments
    - If you delete the pod means automatically you will get another pod because of replicas, but IP address will change.
    - For that, we need Service discovery mechanism. If k8s services identifying the pods using the IP address, you will face a traffic loss because the IP address has changed
    - That's the reason we use concept called labels & Selectors
    - Using the above concept, k8s identifying the pods.
    - So, everytime new pod will create, the label will remain the same
- kubectl get pod -o wide



```
root@ip-172-31-45-178:~/Docker-Zero-to-Hero/examples/python-web-app# kubectl get pod -o wide
NAME                                  READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
service-deployment-7c8744fc5d-g7hr6   1/1     Running   0          10m     10.244.0.6   minikube   <none>           <none>
service-deployment-7c8744fc5d-v9cz9   1/1     Running   0          8m12s   10.244.0.7   minikube   <none>           <none>
```

- minikube ssh
- curl -L http://10.244.0.6:8000/demo
    - The application that I have written it requires a redirect
    - We add /demo, FYI, go to the below path
        - cd Docker-Zero-to-Hero/examples/python-web-app/devops/devops

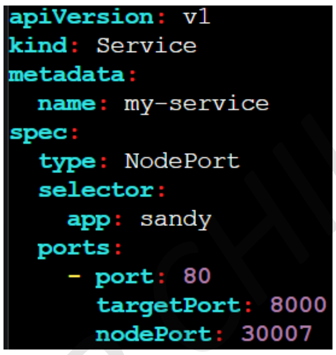


```
urlpatterns = [
    path('demo/', include('demo.urls')),
    path('admin/', admin.site.urls),
]
```

    - We can get the output
- Now, perform the "curl" command in outside minikube cluster, it won't work
    - By default, A pod will have the cluster network attached to it. It is using cluster IP
    - So, above only is useful, when a person is inside the cluster he can only access. But, outside people can't access the pod
- To solve the above problem we are using NODEPORT and LOADBALANCER

NODEPORT

Now, create one file → vi service.yml



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: sandy
  ports:
    - port: 80
      targetPort: 8000
      nodePort: 30007
```

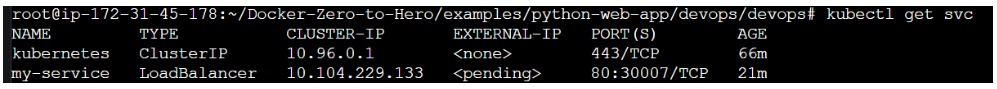
- Here, in the TargetPort you have to mention the Dockerfile port number and
- In NodePort you can mention in range 30,000-32,767 port number
- Inside selector, keep label name same as deployment to the pods
    - Because, service will be directly looking at the pods, using selectors
    - So, now go inside the deploy.yml file, Always pick the label from the template section.
    - If you change label means, you can't access application inside cluster
- Create the service file → kubectl create -f service.yml
- See the list of service → kubectl get svc
- See the minikube IP address → minikube ip
- perform the command in outside the cluster
    - curl -L http://192.168.49.2:30007/demo
    - Now, you can access the application inside and outside the cluster

Now, We can't access the application in browsers, For that we're using Load balancer

So, perform this command → kubectl edit svc serviceName → kubectl edit svc my-service

- Here, you can change the type NodePort to LoadBalancer. But it will not work in minikube cluster
- It will work if you're on any cloud provider

Now, if you perform → kubectl get svc



```
root@ip-172-31-45-178:~/Docker-Zero-to-Hero/examples/python-web-app/devops/devops# kubectl get svc
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        66m
my-service   LoadBalancer   10.104.229.133   <pending>     80:30007/TCP   21m
```

- The External IP address will not be allocated it is in pending state. Because we're using minikube
- If it was AWS (or) Azure (or) GCP. You will get the IP address
- IP address will generated by Cloud Controller Manager of k8s
- Then after, you can share external IP address to the clients/anyone. So, they can access the application

So, this is the concepts of how to expose your applications

KUBESHARK

KubeShark explains how the traffic is flowing within the cluster.

- Default Port number is 8899
- In real time, Using the KubeShark we can see the load balancer → URL is localhost:8899
- Kubeshark offers real-time, cluster-wide, identity-aware, protocol-level visibility into API traffic, empowering its users to see in their own eyes what's happening in all (hidden) corners of their K8s clusters.
- The API traffic analyzer for Kubernetes providing real-time K8s protocol-level visibility, capturing and monitoring all traffic and payloads going in, out and across containers, pods, nodes and clusters. Inspired by Wireshark, purposely built for Kubernetes. Documentation.

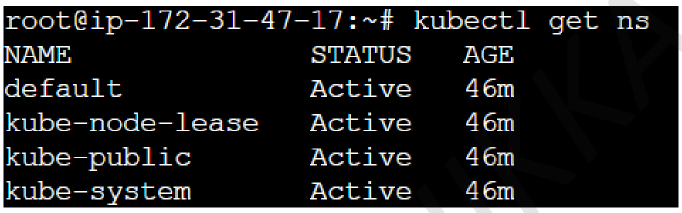
  If you want more knowledge about Load balancer, Browse "KUBESHARK" in google and study

NAMESPACES

This namespaces concept is used for isolation purpose

- namespaces are used to group the components like pods, services and deployments
- This can be useful for separating environments such as development, staging and production (or) for separating different teams (or) applications
- In real-time all the frontend pods are mapped to one namespace and backend pods are mapped to another namespace
- It represents the cluster inside the cluster
- You can have multiple namespaces within one k8s cluster and they are logically isolated from one to another
- Namespaces provide a logical separation of cluster resources between multiple users, teams, project and even customers/clients
- Within the same namespace, pod to pod communication
- Namespaces are only hidden from each other but they are not fully isolated from each other
- One service in a namespace can talk to another service in another namespace. if the target and sources are used with the full name which includes service/object name followed by namespace
- The name of resources within one namespace must be unique
- When you delete namespace all the resources will get deleted

Check the default namespaces → kubectl get ns



```
root@ip-172-31-47-17:~# kubectl get ns
NAME              STATUS   AGE
default           Active   46m
kube-node-lease   Active   46m
kube-public       Active   46m
kube-system       Active   46m
```

DEFAULT

- When we create resources like pod, service, deployments all will gets stored in default namespace

KUBE-PUBLIC

- If your namespace will be available means use kube-public namespace

KUBE-SYSTEM

- The namespace for objects created by the k8s system
- K8s created automatic resources will be stored in kube-system

KUBE-NODE-LEASE

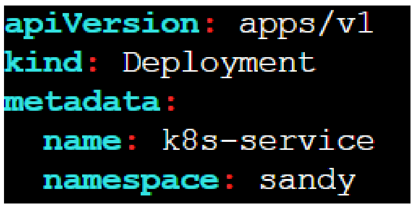
- It is used for the lease objects associated with each node that improves the performance of the node heartbeats as the cluster scales
- i.e. we are taking the lease objects in k8s and we have to deploy the application

COMMANDS

- Create custom namespace
  - kubectl create ns sandy
- Create the pod in namespace
  - kubectl run pod-1 -n sandy --image=nginx
- Now, check the list of pods
  - kubectl get po
  - You won't find the pod-1 here
- Now, we can see the pod in particular namespace
  - kubectl get pod -n sandy (or) kubectl get pod --namespace=sandy
  - Here, you will get the list of pods. i.e. pod-1 will present here
- See the all objects in a particular namespace
  - kubectl get all --namespace=sandy
- Delete the namespace
  - kubectl delete ns sandy
- See the list of namespaces
  - kubectl get ns

Declaring the Namespace in Manifest file

Under metadata, we have to give namespace like this



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-service
  namespace: sandy
```

After doing the changes,

- kubectl create -f deploy.yml
- kubectl get po -n sandy