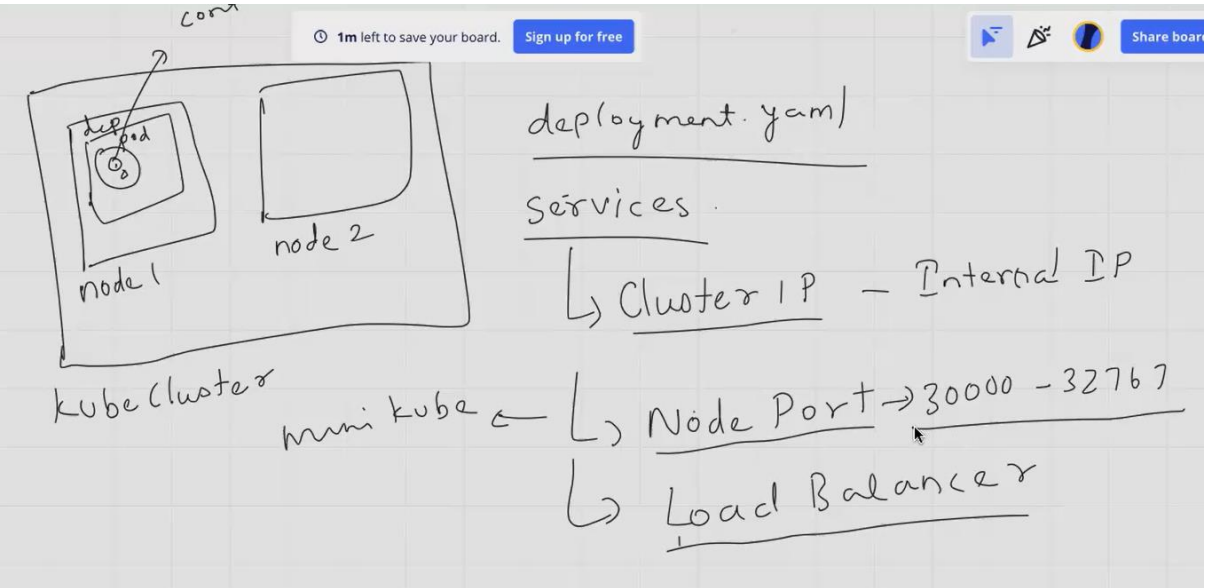
Day 2 - Kubernetes - Kube Objects & Namespace

Task: Setup minikube at your local and explore creating namespaces for this class

- So, inside your Kubernetes cluster, you have worker nodes.
- You have Worker Node1 and worker Node2. by default, 2 nodes get created. If you want, you can increase the size of your worker, node, or but by default 2 nodes will be created to form a cluster.
- You need minimum of 2 worker nodes. So, inside your cluster when you use your deployment file. So yesterday we wrote deployment dot yaml file, right? So this is the deployment manifest file. Just written in Yaml format.
- So we use this deployment file to deploy your application inside your cluster.
- So inside your worker node, we have this deployment.
- inside the deployment. You have your pods so you can have multiple pods inside your deployment. You can have N number of Pods. within your pod You have the containers, so your containers will be running inside your pod.
- and your application will be running inside your container. So to access your application. If you want to access your application through the Internet, you cannot directly access it. You have to create an object or resource for these services.
- So you, with the help of services you can, you will be able to access the application that is running inside your Kubernetes cluster. So you have different types of services.
- So you have different types of services like cluster IP. So by default, all your pods, all the resources that is getting created with a cluster IP only.
- So cluster IP is like, which provides you an internal IP [internal IP or internal communication] So you can use this cluster.
- If by default all your containers, will you can communicate with your containers. You can communicate with your pods within the cluster. So for the internal communication we have cluster IP.
- By default, this cluster IP will be attached to your resources that we create inside the cluster. Okay, but with the cluster IP, we can only reach the resources within the cluster. You can access it only within the cluster.

- Outside of the cluster, you cannot access it.so we use a service called note port or if you're running a Kubernetes cluster like how we do it in your cloud environment. We use a service called load balancer.
- So in case if you're running it in on premise, you can either go with the note port. Since we are using aws Cloud Provider, we make use of load balancer service.
- So this load balancer, and note port will help you to access the application which is running inside the cluster through the Internet. So via Browser, you'll be able to access the application that is running inside your cluster.
- So typically, we use only load balancer. We don't go with a note port for a local on-premise architecture. You're in your local. If you're setting up Kubernetes cluster, so we will also see how to set up Kubernetes cluster in the local machine using mini kube. In that case you can go with Node port.
- **we have different services.**
    1. **cluster IP -internal IP**
    2. **Node port** [minikube]. port no between 30,000 to 32,767. you can only configure the port number in this range. Outside of the range you cannot configure. That is the reason we use only node port for local configuration. If you're maintaining a Kubernetes cluster within in the local. then you can go for the in your local machine. Then you can go for note port, because in our laptops and PC. We don't have a load balancer.
    3. **Load balancer**



Duplicate the same machine and create a dir.

```
ubuntu@ip-172-31-38-243:~$ mkdir test
ubuntu@ip-172-31-38-243:~$ cd test/
ubuntu@ip-172-31-38-243:~/test$ ls
ubuntu@ip-172-31-38-243:~/test$ clea
```

you can get the Api version for every resource; you have a different Api version for deployment. You have one Api version. If you're not aware of it, initial stages, we will not know which resource is using which Api version. Okay? So, for that, you use a command called this kubeCtl

```
ubuntu@ip-172-31-38-243:~/test$ kubectl api-resources | grep deployment
```

Deployment file [deploy.yaml]

```
ubuntu@ip-172-31-38-243:~/test$ vi deploy.yaml
ubuntu@ip-172-31-38-243:~/test$
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dep
  labels:
    app: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: nginx-con
          image: nginx
          ports:
            - containerPort: 80
```

my nginx container will be created inside my deployment.

Code explanation:

This is a Kubernetes YAML file defining a Deployment object. Let's break down the key components:

1. **API Version and Kind:**

apiVersion: apps/v1: Specifies the API version being used, which is apps/v1 for Deployments.

kind: Deployment: Declares the type of object being defined, which is a Deployment.

2. Metadata:

name: nginx-dep: Sets the name of the Deployment to nginx-dep.

labels: Assigns labels to the Deployment for identification and organization.

app: web: Labels the Deployment as belonging to the web application.

3. Spec:

replicas: 3: Specifies that three replicas of the application should be deployed.

selector: Defines how the Deployment selects which Pods to manage.

match Labels: Selects Pods with the label app: web.

template: Defines the template for creating Pods.

metadata: Metadata for the Pods created from this template.

labels: Assigns labels to the Pods.

app: web: Labels the Pods as belonging to the web application.

spec: Specifies the details of the Pods.

containers: Defines the container(s) to run in the Pods.

- name: nginx-con: Sets the name of the container to nginx-con.

image: nginx: Specifies the Docker image to use for the container, which is the official Nginx image.

ports: Exposes port 80 on the container.

- containerPort: 80: Exposes port 80 within the container.

Overall, this YAML file creates a Deployment named nginx-dep that deploys three replicas of an Nginx web server. The Pods created by this Deployment will be labeled with app: web and will run the Nginx image, exposing port 80

```
ubuntu@ip-172-31-38-243:~$ ls
app1  test
ubuntu@ip-172-31-38-243:~$ cd test/
ubuntu@ip-172-31-38-243:~/test$ ls
deploy.yaml
ubuntu@ip-172-31-38-243:~/test$ kubectl apply -f deploy.yaml
deployment.apps/nginx-dep created
ubuntu@ip-172-31-38-243:~/test$ kubectl get all
NAME                             READY   STATUS              RESTARTS   AGE
pod/nginx-dep-5c979f95d4-fjbhw   0/1     ContainerCreating   0          8s
pod/nginx-dep-5c979f95d4-h8mhl   0/1     ContainerCreating   0          8s
pod/nginx-dep-5c979f95d4-p9tw7   0/1     ContainerCreating   0          8s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   12m

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-dep   0/3     3            0           8s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-dep-5c979f95d4   3         3         0       8s
ubuntu@ip-172-31-38-243:~/test$
```

now we have created the deployment in order to access the deployment through the browser. So now I want to access my nginx application through the Internet. So, what do I need? I need a service resource or a service object.

I have to create a manifest file or a **configuration file for creating my service Object/file.**

```
ubuntu@ip-172-31-38-243:~/test$ vi service.yaml
ubuntu@ip-172-31-38-243:~/test$ kubectl api-resources | grep deployment
deployments                     deploy       apps                           true         Deployment
```

```
ubuntu@ip-172-31-38-243:~/test$ kubectl explain deployment
KIND:     Deployment
VERSION:  apps/v1

DESCRIPTION:
     Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this representation of an
     object. Servers should convert recognized schemas to the latest internal
     value, and may reject unrecognized values. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
```

```
ubuntu@ip-172-31-38-243:~/test$ kubectl explain services
KIND:     Service
VERSION:  v1
```

if you want to get the Api version of you don't know what is the Api version of service? We are going to create the manifest file. So, we don't know what is the Api version of service. Then you can give Ctl Explain service. It will give you the Api version

service file/object to expose my data via browser:

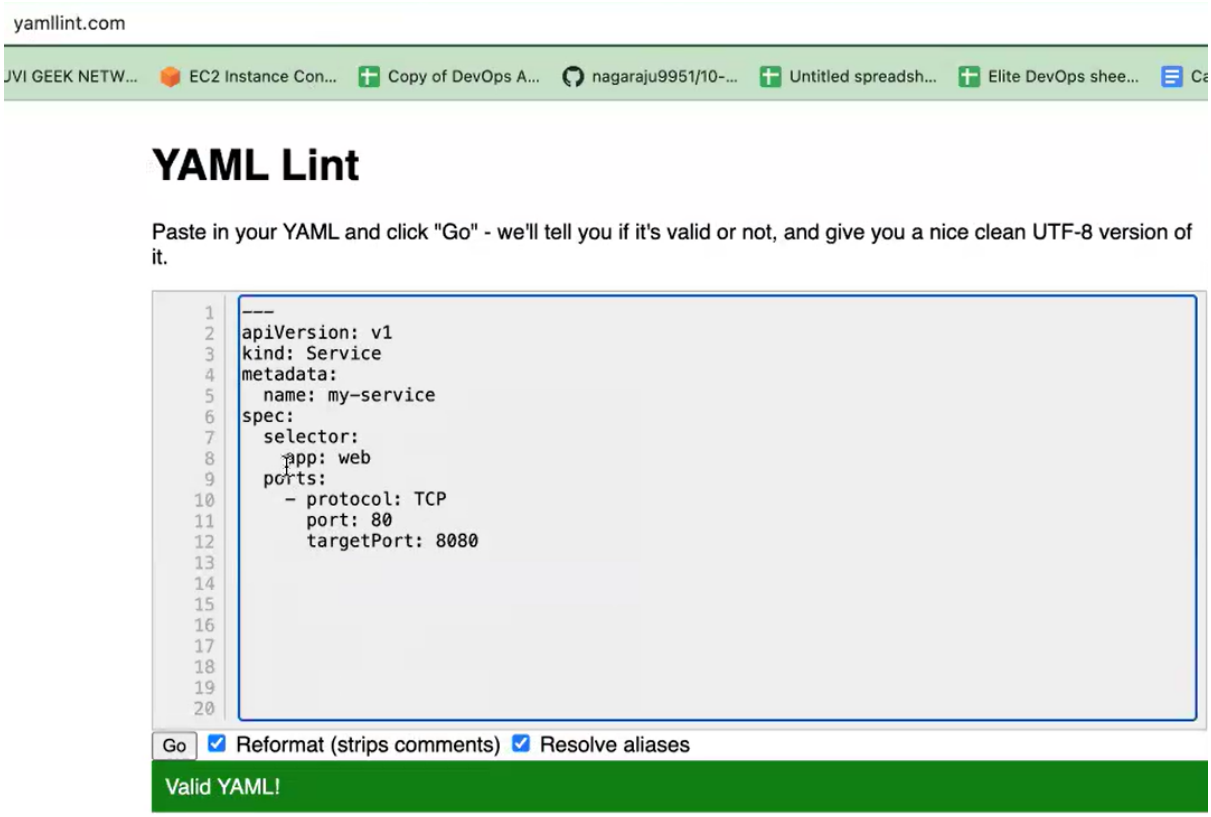
service.yaml:

```
ubuntu@ip-172-31-38-243:~/test$ vi service.yaml
```

```yaml
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
  selector:
    app: web
  ports:
   - protocol: TCP
     port: 80
     targetPort: 80
  type: LoadBalancer
```

Target port no means container port no. service runs via load balancer

To check for any error, check in this page



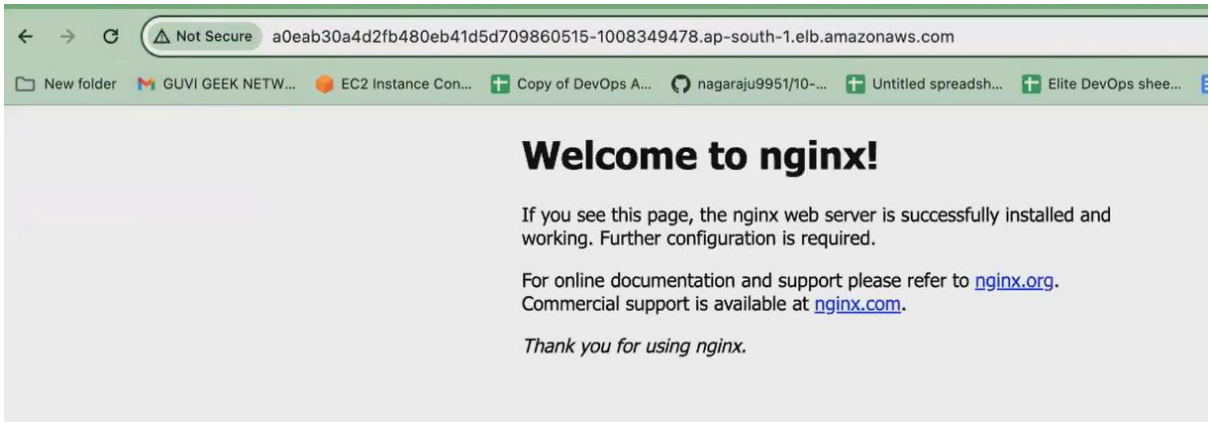

```
ubuntu@ip-172-31-38-243:~/test$ kubectl apply -f service.yaml
service/my-service created
```

```
ubuntu@ip-172-31-38-243:~/test$ kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
kubernetes   ClusterIP      10.100.0.1      <none>                                                                     443/TCP        24m
my-service   LoadBalancer   10.100.210.61   a0eab30a4d2fb480eb41d5d709860515-1008349478.ap-south-1.elb.amazonaws.com   80:31260/TCP   17s
ubuntu@ip-172-31-38-243:~/test$
```

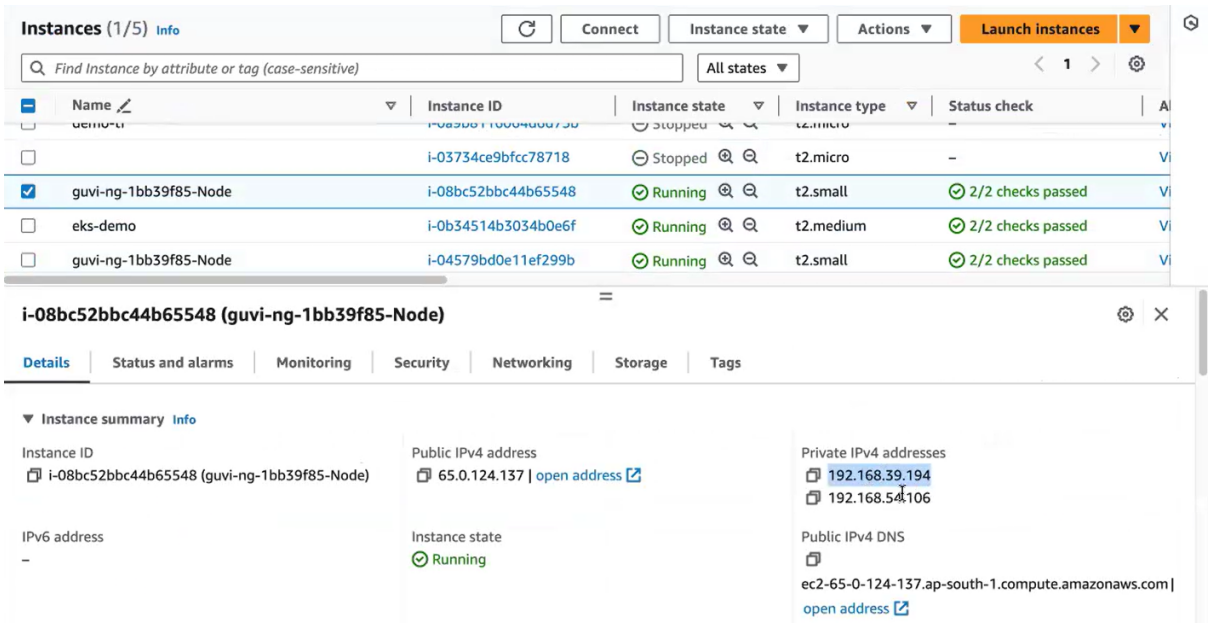
```
ubuntu@ip-172-31-38-243:~/test$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-dep-5c979f95d4-fjbhw      1/1     Running   0          12m
pod/nginx-dep-5c979f95d4-h8mhl      1/1     Running   0          12m
pod/nginx-dep-5c979f95d4-p9tw7      1/1     Running   0          12m

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
service/kubernetes   ClusterIP      10.100.0.1      <none>                                                                     443/TCP        25m
service/my-service   LoadBalancer   10.100.210.61   a0eab30a4d2fb480eb41d5d709860515-1008349478.ap-south-1.elb.amazonaws.com   80:31260/TCP   38s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-dep    3/3     3            3           12m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-dep-5c979f95d4    3         3         3       12m
ubuntu@ip-172-31-38-243:~/test$
```

Open the port no:80. Copy the external ip and paste it in browser

```
ubuntu@ip-172-31-38-243:~/test$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-dep-5c979f95d4-fjbhw      1/1     Running   0          12m
pod/nginx-dep-5c979f95d4-h8mhl      1/1     Running   0          12m
pod/nginx-dep-5c979f95d4-p9tw7      1/1     Running   0          12m

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP
service/kubernetes   ClusterIP      10.100.0.1      <none>
service/my-service   LoadBalancer   10.100.210.61   a0eab30a4d2fb480eb41d5d709860515-1008349478.ap-south-1.elb.amazonaws.com
```



2 worker node created in same cluster, but running diff ip and machine



NAMESPACE:

- So, suppose assume that you work on a project. You have multiple people working on the same Kubernetes cluster.
- what happened? You and your team member unknowingly you create a deployment with the same name.
- What will happen? You have a conflict there.
- what will happen. You are deleting instead of deleting your deployment, you delete your teammates Deployment. So, what will happen in order to avoid all those things, to maintain the isolation of your resources.

- We go with namespaces so you can create namespaces to isolate the resource creation inside your Kubernetes cluster.
- So, I will tell you how to create the namespace
- so, we can create namespace your any resources, even with the single line command, or you can write a yaml file. So now what we have done. We have written a Yaml file for creating deployment for creating services. You have created yaml file. You can also create the deployment with a single line command also.
- And same way, you can create all the resources, not only deployment, you can create your port, you can create your service object with a single line. We call it as kube ctl cli command.
- Without creating the manifest file. You can go with the kubectl cli Command for emergency purpose.
- You can go with it, but it is always good to follow or create your resources via manifest file only then you will have a proper documentation.

1. Deployment file

2. Service file [used to access the application via browser. That time load balancer created]

✓ Deployment file:

- Create a new dire [test2].
- Mydepolyment [deployment file name]
- Httpd [image name]

```
ubuntu@ip-172-31-38-243:~$ cd test2
ubuntu@ip-172-31-38-243:~/test2$ ls
ubuntu@ip-172-31-38-243:~/test2$ kubectl create deployment mydeployment --image=httpd --port=80
deployment.apps/mydeployment created
ubuntu@ip-172-31-38-243:~/test2$
```

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/mydeployment-f65dd7df4-6rk6f     1/1     Running   0          22s
pod/nginx-dep-5c979f95d4-fjbhw       1/1     Running   0          39m
pod/nginx-dep-5c979f95d4-h8mhl       1/1     Running   0          39m
pod/nginx-dep-5c979f95d4-p9tw7       1/1     Running   0          39m

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
service/kubernetes   ClusterIP      10.100.0.1      <none>                                                                     443/TCP        51m
service/my-service   LoadBalancer   10.100.210.61   a0eab30a4d2fb480eb41d5d709860515-1008349478.ap-south-1.elb.amazonaws.com   80:31260/TCP   27m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mydeployment         1/1     1            1           22s
deployment.apps/nginx-dep            3/3     3            3           39m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/mydeployment-f65dd7df4     1         1         1       22s
replicaset.apps/nginx-dep-5c979f95d4       3         3         3       39m
ubuntu@ip-172-31-38-243:~/test2$
```

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/mydeployment-f65dd7df4-6rk6f     1/1     Running   0          22s
```

- So you have the name of your deployment.
- so the centre one is your replica set Id [f65dd]
- the last 5-digit 5 character is your pod Id. [6rk]

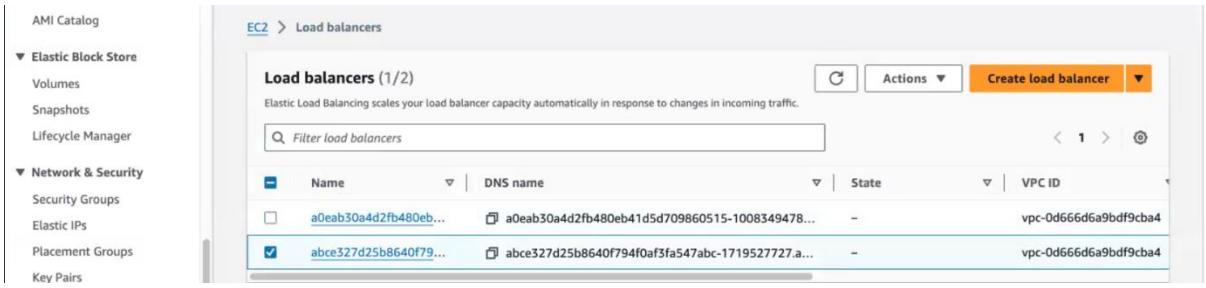
✓ service file:

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl expose deployment mydeployment --type=LoadBalancer --name=apacheservice --port=80 --target-port=80
service/apacheservice exposed
```

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get svc
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
apacheservice  LoadBalancer   10.100.132.10   abce327d25b8640f794f0af3fa547abc-1719527727.ap-south-1.elb.amazonaws.com   80:30941/TCP   8s
kubernetes     ClusterIP      10.100.0.1      <none>                                                                     443/TCP        54m
my-service     LoadBalancer   10.100.210.61   a0eab30a4d2fb480eb41d5d709860515-1008349478.ap-south-1.elb.amazonaws.com   80:31260/TCP   29m
```

Copy the ip [abce327d] paste it in browser. Apache server running

Load balancer created:



So, all your resources got created inside the default namespace.

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get ns
NAME              STATUS   AGE
default           Active   57m
kube-node-lease   Active   57m
kube-public       Active   57m
kube-system       Active   57m
```

How to create custom name space:

Kubernetes, everything is created as a resource, everything is treated as objects only. So, your deployment is an object. Your service is an object. And now we are going to create a namespace that is also treated as an object.

Name space file also created via kubectl cli command or manifest file

1. **Kubectl cli command:**

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl create namespace my-ns
namespace/my-ns created
ubuntu@ip-172-31-38-243:~/test2$ kubectl get ns
NAME              STATUS   AGE
default           Active   61m
kube-node-lease   Active   61m
kube-public       Active   61m
kube-system       Active   61m
my-ns             Active   10s
ubuntu@ip-172-31-38-243:~/test2$
```

2. **Manifest file:**

```
ubuntu@ip-172-31-38-243:~/test2$ vi namespace.yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-ns2
```

Execute the namespace.yaml file:

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl create -f namespace.yaml
namespace/my-ns2 created
ubuntu@ip-172-31-38-243:~/test2$ kubectl get ns
NAME              STATUS   AGE
default           Active   63m
kube-node-lease   Active   63m
kube-public       Active   63m
kube-system       Active   63m
my-ns             Active   2m19s
my-ns2            Active   4s
```

will create a small deployment or pod object inside my namespace.

```
ubuntu@ip-172-31-38-243:~/test2$ vi deployment.yaml
```

Instead of default namespace give our namespace name [ns]

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
  namespace: my-ns
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd
        ports:
        - containerPort: 80
```

```
ubuntu@ip-172-31-38-243:~/test2$ vi deployment.yaml
ubuntu@ip-172-31-38-243:~/test2$ kubectl create -f deployment.yaml
deployment.apps/apache-deployment created
```

- what is your deployment name Apache-deployment. It got created inside my custom namespace. So if you want to list it.
- if you give cubectl get all. you cannot find this deployment.
- When you give cubectl getall will only display the resources that is there inside your default. Namespace.

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/mydeployment-f65dd7df4-6rk6f     1/1     Running   0          13m
pod/nginx-dep-5c979f95d4-fjbhw       1/1     Running   0          53m
pod/nginx-dep-5c979f95d4-h8mhl       1/1     Running   0          53m
pod/nginx-dep-5c979f95d4-p9tw7       1/1     Running   0          53m

NAME                     TYPE           CLUSTER-IP       EXTERNAL-IP
service/apacheservice    LoadBalancer   10.100.132.10    abce327d25b8640f794f0af3fa547abc-1719527727.
service/kubernetes       ClusterIP      10.100.0.1       <none>
service/my-service       LoadBalancer   10.100.210.61    a0eab30a4d2fb480eb41d5d709860515-1008349478.

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mydeployment         1/1     1            1           13m
deployment.apps/nginx-dep            3/3     3            3           53m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/mydeployment-f65dd7df4     1         1         1       13m
replicaset.apps/nginx-dep-5c979f95d4       3         3         3       53m
ubuntu@ip-172-31-38-243:~/test2$
```

To list your custom namespaces:

```
ubuntu@ip-172-31-38-243:~/test2$ kubectl get all -n my-ns
NAME                                        READY   STATUS    RESTARTS   AGE
pod/apache-deployment-7749849b9c-2s245      1/1     Running   0          60s
pod/apache-deployment-7749849b9c-jvtdr      1/1     Running   0          60s
pod/apache-deployment-7749849b9c-pfdlm      1/1     Running   0          61s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/apache-deployment      3/3     3            3           61s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/apache-deployment-7749849b9c      3         3         3       61s
ubuntu@ip-172-31-38-243:~/test2$
```

- we use this namespace just for organizing or for the isolation purpose within. Your Kubernetes cluster mainly for the resource organization and for isolation purpose. You go with your namespace.
- your services, your deployments we go with this namespace. And one more thing is for the access control also.
- Okay, so you can also implement role-based access, which means what you can do. You can further customize this. You can further control this namespace. But what will happen? Certain users cannot access your namespaces.
- So, you can either grant a permission to a user or deny a permission to a user to access this namespace. Okay, so there we will be **using a concept called as Role-Based**

Access Control (RBAC). So, we have, like in your Jenkins. You use the concept called RBAC for managing your roles and users.

Commands:

- kubectl create deployment mydeployment --image=httpd --port=80
- kubectl expose deployment mydeployment --type=LoadBalancer --name=apacheservice --port=80 --target-port=80
- kubectl create namespace my-ns

documents link:

https://docs.google.com/document/d/10dyXzkVgeqWaaja6AKDaKNfs-tXwu9kgRGwWXm23yz8/edit#heading=h.nf327o3e3njg

https://docs.google.com/document/d/10dyXzkVgeqWaaja6AKDaKNfs-tXwu9kgRGwWXm23yz8/edit?usp=sharing

minikube official documentation:

https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download

Setup minikube at your local: [task]

- ✓ launch an instance with t2.medium [its is chargable.dont use more than 30-40 m].because minikube needs 2 cpu for install.so we go with t2.medium
- ✓ after launch instance, connect and update it. then install docker.
- ✓ Install kubectl
- ✓ Install minikube

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

- ✓ Minikube start [minikube start]
- ✓ Delete minikube [minikube delete –all]

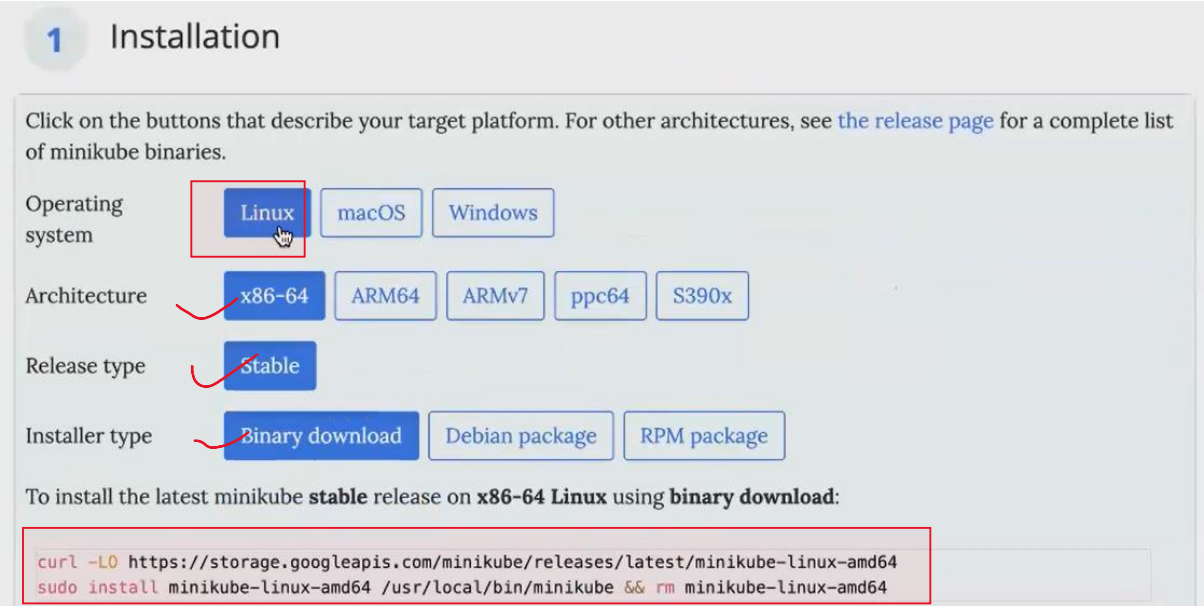
minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.

All you need is Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away: `minikube start`

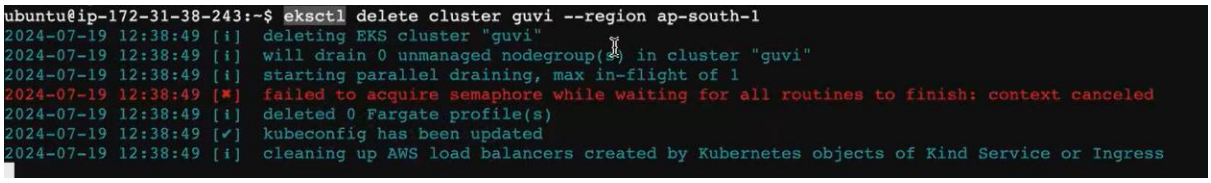
What you'll need

- 2 CPUs or more
- 2GB of free memory
- 20GB of free disk space
- Internet connection
- Container or virtual machine manager, such as: Docker, QEMU, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

- binary download If you're using ubuntu you go with Debian package. If you're using cento is on Amazon Linux, you can go with Rpm Base.
- **But binary download is standard for both the distributions**. Binary download You can use it for ubuntu or Amazon Linux or Centos Mission. Better. You go with the binary download itself. Just copy this command, put it in your Ec. 2. Execute it. Once that is done.
- you have to run the Mini cube. Start so Mini Cube will start your cluster.
- Copy and paste the below marked commands in your machine.



- Mini Cube start your Mini cube will be started.
- You can see the command control plane running kubectl running kubes running all those things.
- When you give kubeCtl get notes you can see the nodes running. you will not see multiple nodes running. because kubectl is a single node architecture. You can only see one node. minikube is single node architecture. Used only for development purpose not used for production purpose.
- After install write manifest file
- Don't terminate the instance without delete the cluster. Give command for deleting the cluster [in aws page deleting the cluster is difficult, because del load balancer and all]



Day 3 - Kubernetes - Pods & Services . doubt session

https://docs.google.com/document/d/1Is4h94KVFliaNxSuBZX9Spui98CYiFOIawjLfYa3wBU/edit

this doc shared by Naveen mentor