ASSIGNMENT -3: KUBERNETES

TASKS:

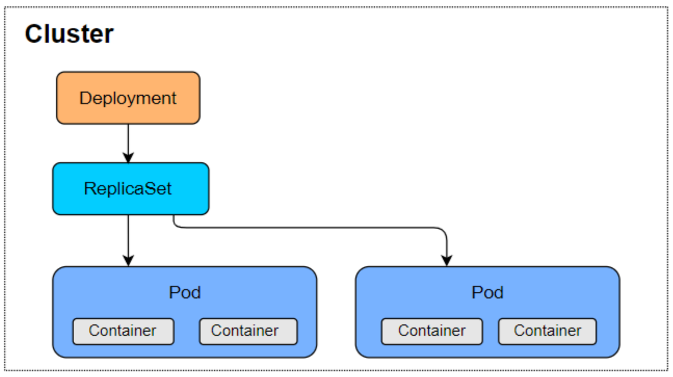
1. **Installations- from the installation document shared.**
2. **Creating Pods and Deployments.**
3. **Debugging pods.**
4. **Applying Configuration Files.**
5. **Self-Healing Feature.**
6. **Connecting services to Deployments.**
7. **Exposing a service externally.**
8. **Deletion and cleanup.**
9. **Exposing an External IP address to access an application in cluster.**

Introduction to Kubernetes:

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage:

- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again
- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies
- Automatically leveraging additional resources made available when a new host is added to the cluster
- Easily performing canary deployments and rollbacks.



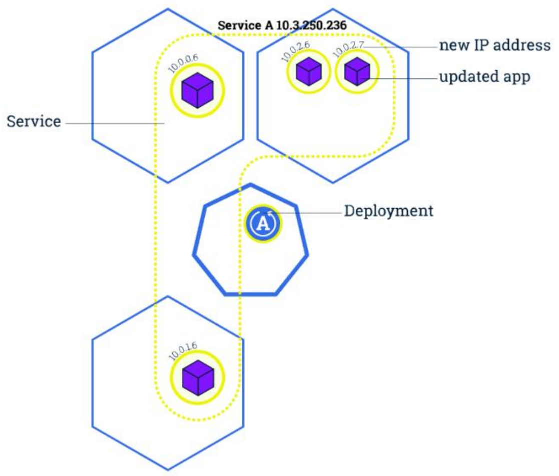


Fig: Kubernetes pods, deployments, services.

Important Kubernetes Terminologies:

1. Kubernetes Namespace
   a. Kubernetes Documentation: Namespaces
2. Pod
   a. Pods – Kubernetes Documentation
   b. What is a Pod.
3. Replica Set
   a. ReplicaSet
4. Deployment
   a. Deployments
5. Service
   a. Kubernetes Service
6. LoadBalancer Service
   a. Kubernetes LoadBalancer Service
7. NodePort Service
   a. Kubernetes NodePort Service
8. Ingress Controller (Not needed for this lab, but must know)
   a. Ingress
9. Horizontal Pod Autoscaler (Not needed for this lab, but must know)
   Kubernetes Horizontal Pod Autoscaler

Quick Points:

1. Make sure to use SRN wherever applicable properly, do not use as pes1ug21csxxx.

2. Ensure Docker is installed, up and running before using minikube.
3. Pods may take some time (a few minutes in a bad network) initially to start up, this is normal.
4. All resources mentioned in all the tasks must be created only in the **default Kubernetes namespace**, modifying other namespaces such as kube-system may cause Kubernetes to stop working.

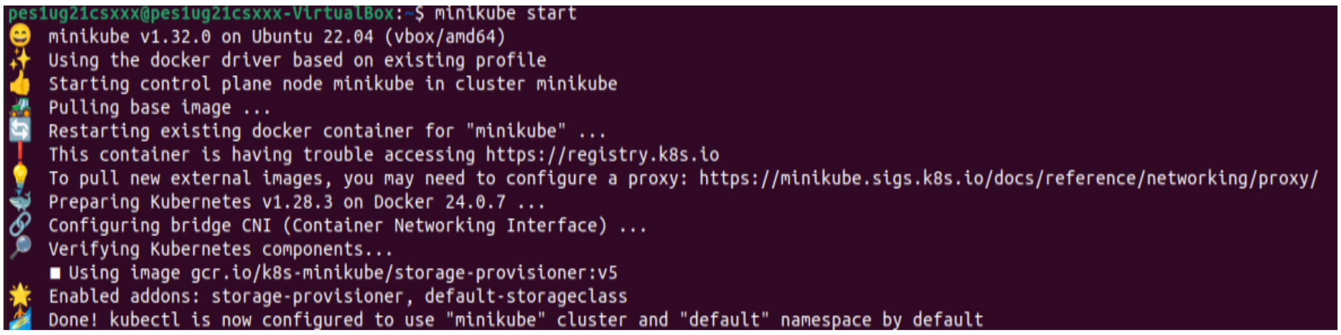
Deliverables:

Task-1: After installation of both kubectl and minikube.

Commands:

1. *sudo service docker start*
2. *minikube start*

✓ **SCREENSHOT 1a: Minikube running successfully.**



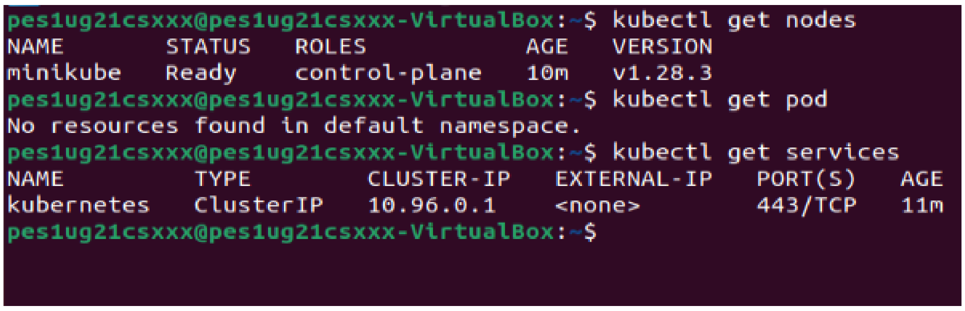
```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ minikube start
😄  minikube v1.32.0 on Ubuntu 22.04 (vbox/amd64)
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
❗  This container is having trouble accessing https://registry.k8s.io
💡  To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Task-2: Creating pods and deployments, editing them and observing Rollback.

✓ **SCREENSHOT 2a: Get nodes, pods, services.**

Commands:

1. *kubectl get nodes*
2. *kubectl get pod.*
3. *kubectl get services*

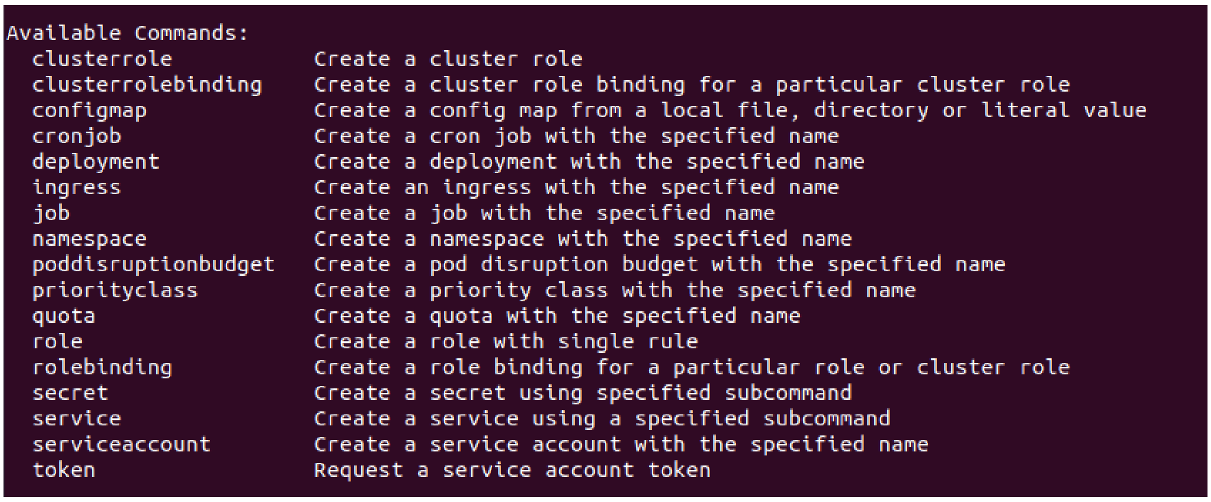


```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   10m   v1.28.3
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
No resources found in default namespace.
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   11m
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

Currently, only one default service is running. We will explore how to create a service later on.

To see what all we can create using kubectl:

Command- kubectl create -h



```
Available Commands:
  clusterrole           Create a cluster role
  clusterrolebinding    Create a cluster role binding for a particular cluster role
  configmap             Create a config map from a local file, directory or literal value
  cronjob               Create a cron job with the specified name
  deployment            Create a deployment with the specified name
  ingress               Create an ingress with the specified name
  job                   Create a job with the specified name
  namespace             Create a namespace with the specified name
  poddisruptionbudget   Create a pod disruption budget with the specified name
  priorityclass         Create a priority class with the specified name
  quota                 Create a quota with the specified name
  role                  Create a role with single rule
  rolebinding           Create a role binding for a particular role or cluster role
  secret                Create a secret using specified subcommand
  service               Create a service using a specified subcommand
  serviceaccount        Create a service account with the specified name
  token                 Request a service account token
```

Notice that there is no pod on this list. This is because in practice, we do not work with pods directly. There exists an abstraction layer over pods called deployments. We generally create a deployment which will create the pods underneath. The command to create a deployment is:

kubectl create deployment <deployment_name> --image=<image>

Note that pods need to be based on an image. Deployments hold all the blueprint information to create a pod. The command given above is the minimalistic information we have to provide to create any deployment (name and image).

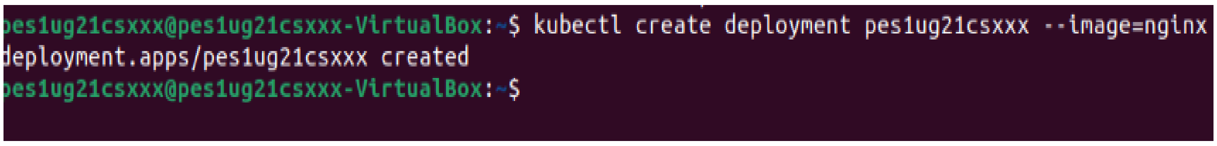
✓ **SCREENSHOT 2b: Deployment Created (with SRN)**

Command:

1.kubectl create deployment pes1ug21csxxx --image=nginx

➔ This command downloads the latest nginx image from DockerHub

***SRN has to be in lowercase.**

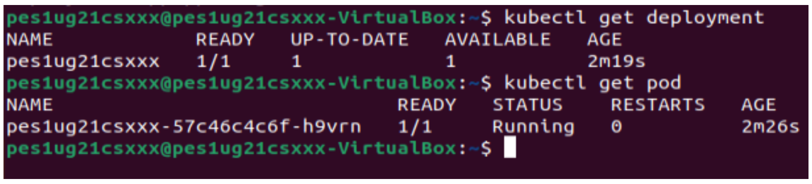


```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl create deployment pes1ug21csxxx --image=nginx
deployment.apps/pes1ug21csxxx created
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 2c: Get deployment and pod.**

Commands:

1. ***kubectl get deployment***
2. ***kubectl get pod***



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
pes1ug21csxxx   1/1     1            1           2m19s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                             READY   STATUS    RESTARTS   AGE
pes1ug21csxxx-57c46c4c6f-h9vrn   1/1     Running   0          2m26s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

The output might differ based on the state of creation/readiness.

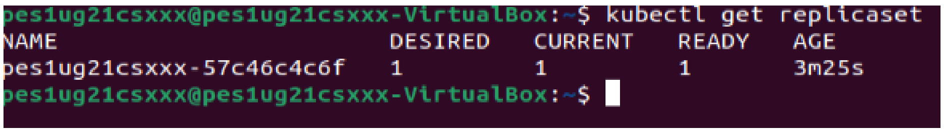
Notice that the pod name begins with the deployment name followed by two hashed strings. Pod name is also dependent on replicaset which is discussed below.

Replicaset:

Between deployments and pods, there exists another layer that is managed by Kubernetes deployment called replicaset.

Run the following command to see the available replicasets:

```
kubectl get replicaset
```



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get replicaset
NAME                      DESIRED   CURRENT   READY   AGE
pes1ug21csxxx-57c46c4c6f   1         1         1       3m25s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ 
```

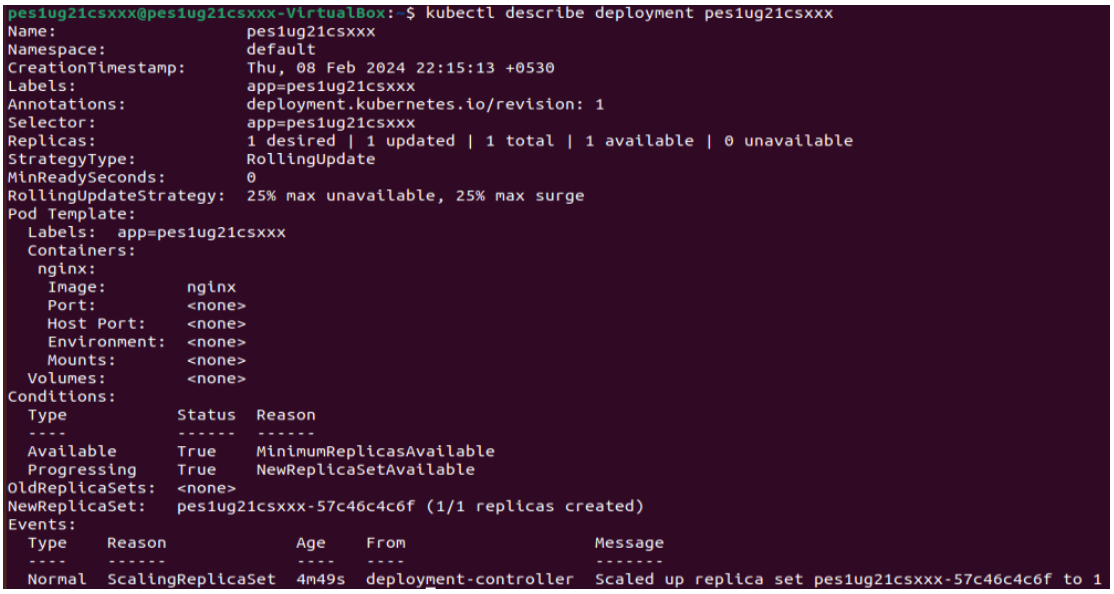
This shows the state of our replicas for the deployment we just created. Since we did not configure additional replicas while creating the deployment, the default value, 1, is considered.

Note that the name of the replicaset is the name of the deployment followed by a random hash. Then, check the name of your pod. It is the name of the replicaset appended by another random hash. We will deal with how to increase the number of replicas in the upcoming tasks.

Thus, we now understand that a deployment manages a replicaset which in turn manages pods. A pod is an abstraction of a container image. Everything below the deployment is managed by Kubernetes. For instance, to change the image used by a pod, we edit it at the deployment level.

To see further details about the deployment, run

```
kubectl describe deployment pes1ug21csxxx
```



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl describe deployment pes1ug21csxxx
Name:                   pes1ug21csxxx
Namespace:              default
CreationTimestamp:      Thu, 08 Feb 2024 22:15:13 +0530
Labels:                 app=pes1ug21csxxx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=pes1ug21csxxx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=pes1ug21csxxx
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   pes1ug21csxxx-57c46c4c6f (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  4m49s  deployment-controller  Scaled up replica set pes1ug21csxxx-57c46c4c6f to 1
```

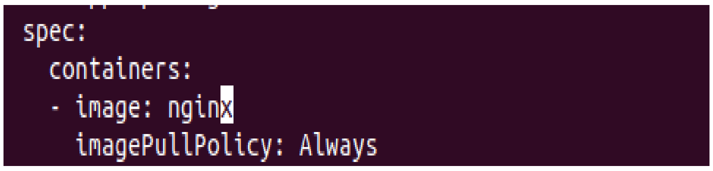
It contains details such as the name of the deployment(which is the name given by us), namespace – where this deployment was created, label which is assigned to the deployment, selector – used to connect pods to deployments. This is because in Kubernetes pods and deployments are separate objects and we have to know how to assign pods to particular deployments. For a pod assigned to this deployment, we will find the same label. The replicas field describes the quantity of pods needed by this deployment and number of pods in ready state. The Pod template shows the details of the pod. Observe that label name of the deployment and pod are the same.

Commands:

1. kubectl edit deployment pes1ug21csxxx

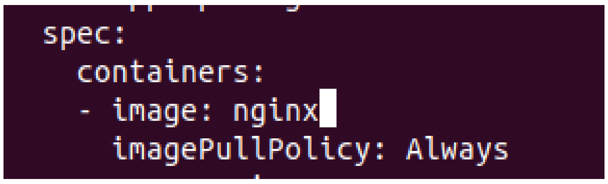
This uses a vim editor by default on linux machines so follow these commands (You can choose to open it in a editor of your choice too):

● Once the configuration file opens, scroll to the line with the text image:nginx using the keyboard arrow keys.



```
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
```

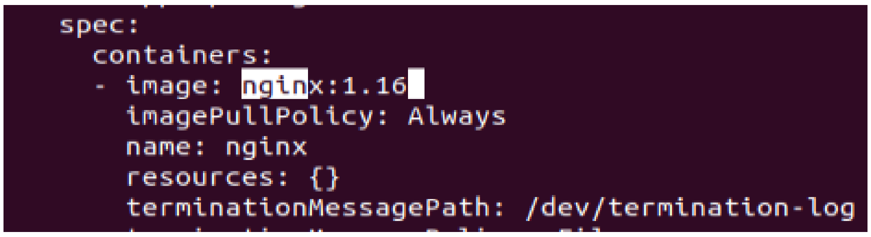
● Once you place your cursor on the 'x' of nginx, type 'a'. This will place the cursor at the end of the word nginx.



```
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
```

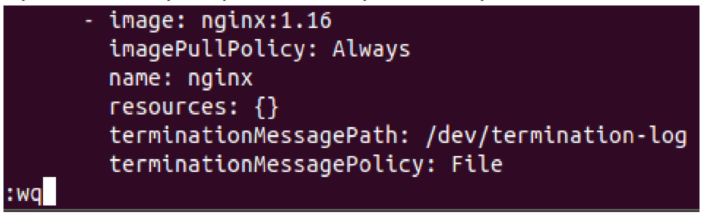
✓ **SCREENSHOT 2d: Editing '-image:nginx'**

- **Now, type ':1.16'**



```
spec:
  containers:
  - image: nginx:1.16
    imagePullPolicy: Always
    name: nginx
    resources: {}
    terminationMessagePath: /dev/termination-log
```

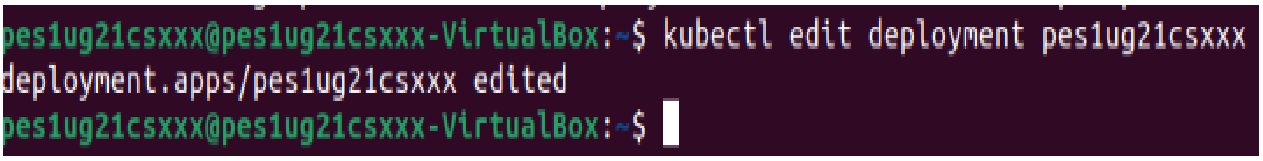
● Then hold the Esc key followed by :wq followed by Enter key. This ensures that your edit is saved.



```
      - image: nginx:1.16
        imagePullPolicy: Always
        name: nginx
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
:wq
```

● Incase you wish to exit without saving the changes: Esc key followed by typing 'q!' followed Enter key.
● 'x' is the key used for Backspace. Make sure to use the Esc key before using the 'x' key.

✓ **SCREENSHOT 2e: Showing edited deployment.**

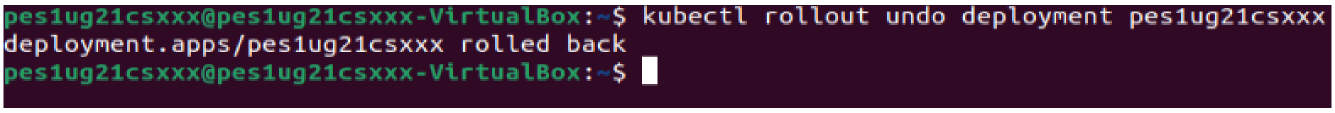


```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl edit deployment pes1ug21csxxx
deployment.apps/pes1ug21csxxx edited
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 2f: Deployment rolled back.**

Command:

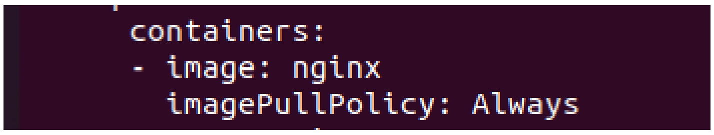
1. kubectl rollout undo deployment pes1ug21csxxx



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl rollout undo deployment pes1ug21csxxx
deployment.apps/pes1ug21csxxx rolled back
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 2g: Changes after rolling back to original.**

Observe the version has been reverted from 1.16 to the latest version



```
      containers:
      - image: nginx
        imagePullPolicy: Always
```

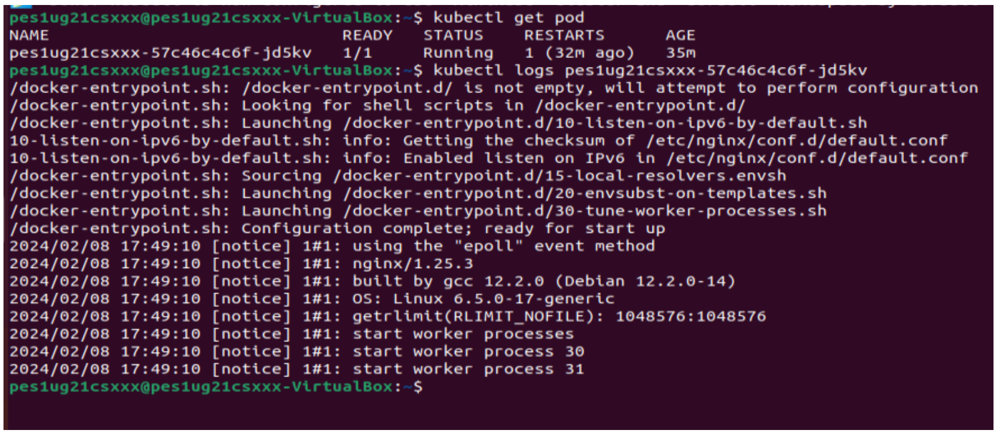
Task-3: Debugging Pods.

A common way to debug is to look at logs. Copy paste your pod name after running the below command and run command 3a by replacing the pod name with yours. Please note that the pod name would have changed after editing the configuration file.

✓ **SCREENSHOT 3a: Kubectl logs displayed**

Command:

1. *kubectl get pod*
2. *kubectl logs<pod_name>*



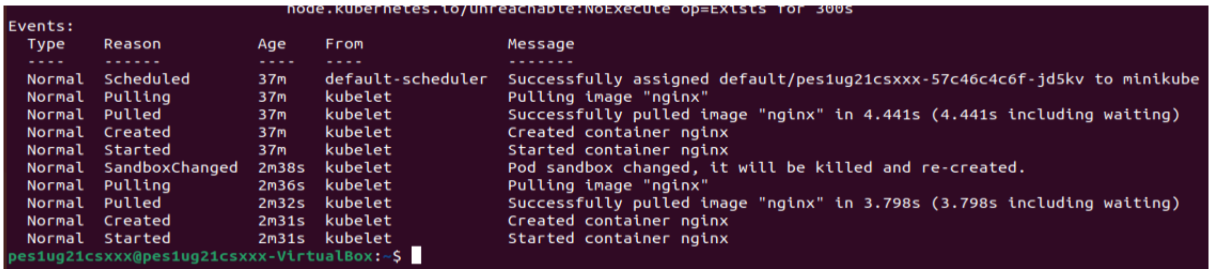
```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                             READY   STATUS    RESTARTS      AGE
pes1ug21csxxx-57c46c4c6f-jd5kv   1/1     Running   1 (32m ago)   35m
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl logs pes1ug21csxxx-57c46c4c6f-jd5kv
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/02/08 17:49:10 [notice] 1#1: using the "epoll" event method
2024/02/08 17:49:10 [notice] 1#1: nginx/1.25.3
2024/02/08 17:49:10 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/02/08 17:49:10 [notice] 1#1: OS: Linux 6.5.0-17-generic
2024/02/08 17:49:10 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/02/08 17:49:10 [notice] 1#1: start worker processes
2024/02/08 17:49:10 [notice] 1#1: start worker process 30
2024/02/08 17:49:10 [notice] 1#1: start worker process 31
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 3b: Kubectl 'describe pod' command – Screenshot of "events" section.**

Command:

1. kubectl describe pod <pod_name>

- Scroll down to the events section.



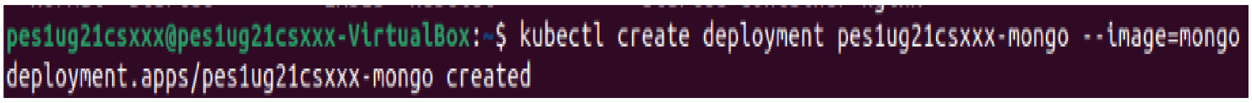
```
                                    node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age    From               Message
  ----    ------          ----   ----               -------
  Normal  Scheduled       37m    default-scheduler  Successfully assigned default/pes1ug21csxxx-57c46c4c6f-jd5kv to minikube
  Normal  Pulling         37m    kubelet            Pulling image "nginx"
  Normal  Pulled          37m    kubelet            Successfully pulled image "nginx" in 4.441s (4.441s including waiting)
  Normal  Created         37m    kubelet            Created container nginx
  Normal  Started         37m    kubelet            Started container nginx
  Normal  SandboxChanged  2m38s  kubelet            Pod sandbox changed, it will be killed and re-created.
  Normal  Pulling         2m36s  kubelet            Pulling image "nginx"
  Normal  Pulled          2m32s  kubelet            Successfully pulled image "nginx" in 3.798s (3.798s including waiting)
  Normal  Created         2m31s  kubelet            Created container nginx
  Normal  Started         2m31s  kubelet            Started container nginx
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 3c: Creating mongo deployment**

Create a deployment with a mongo image. Wait till the container is created and the pod is in Running State.

Command:

1. ***kubectl create deployment pes1ug21csxx-mongo --image=mongo***



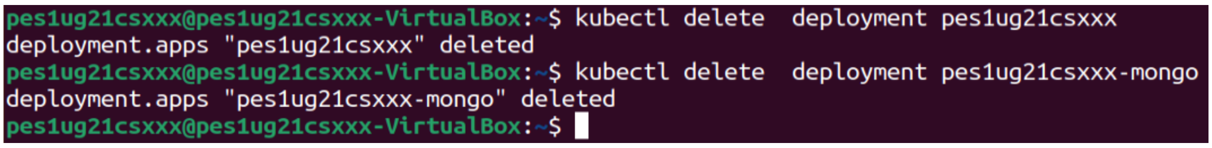
```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl create deployment pes1ug21csxxx-mongo --image=mongo
deployment.apps/pes1ug21csxxx-mongo created
```

✓ **SCREENSHOT 3d: Deleting both requirements.**

Command:

1.kubectl delete deployment <deployment_name>

Delete both your deployments. Now run kubectl get pod and observe the output.



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl delete  deployment pes1ug21csxxx
deployment.apps "pes1ug21csxxx" deleted
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl delete  deployment pes1ug21csxxx-mongo
deployment.apps "pes1ug21csxxx-mongo" deleted
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

Task-4: Applying configuration files.

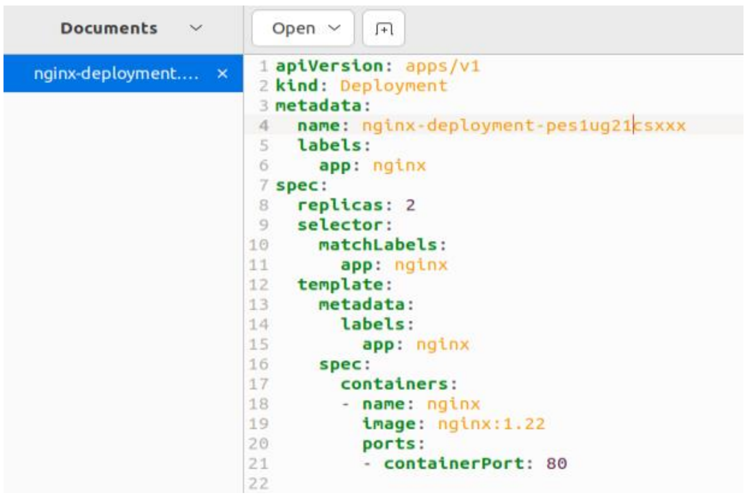
Since deployments can have various configurations or flags that need to be set, it is difficult to type them all in a terminal interface. Thus, use configuration files. These are .yaml files which can be used to create pods, deployments and services. Each configuration file has 3 components:

1) Metadata
2) Specification (under "spec:")
3) Status – Automatically generated and edited by Kubernetes

This status compares the desired and actual states of the components and fixes it in case of a difference between the two. This is part of the self-healing feature of Kubernetes.

Configuration file:

The **file 'nginx-deployment.yaml'** provided by your professor will be used to create a deployment. **Make sure to change 'pes1ug21csxxx' to your SRN in the "name" field nginx-deployment-pes1ug21csxxx.**



```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment-pes1ug21csxxx
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16     spec:
17       containers:
18       - name: nginx
19         image: nginx:1.22
20         ports:
21         - containerPort: 80
22
```

Under the specification part of the deployment, template is present. Notice that template has its own metadata and specification. The reason for this is the configuration under template applies to a pod. Pods have their own configuration inside the deployment's configuration file.

✓ **SCREENSHOT 4a: Kubectl apply command on yaml file.**

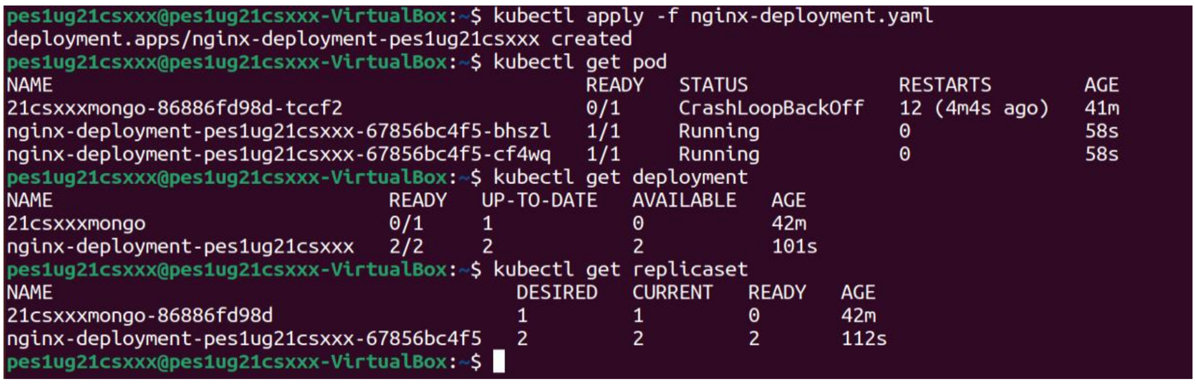
Command:

1. kubectl apply -f <filename>

2.kubectl get pod

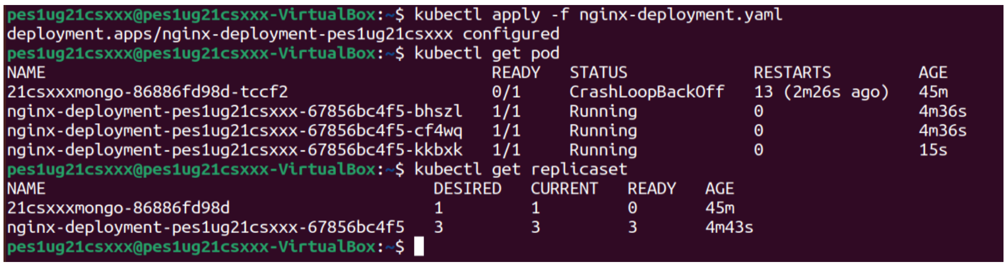
3.kubectl get deployment

4.kubectl get replicaset



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment-pes1ug21csxxx created
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                                              READY   STATUS             RESTARTS       AGE
21csxxxmongo-86886fd98d-tccf2                     0/1     CrashLoopBackOff   12 (4m4s ago)  41m
nginx-deployment-pes1ug21csxxx-67856bc4f5-bhszl   1/1     Running            0              58s
nginx-deployment-pes1ug21csxxx-67856bc4f5-cf4wq   1/1     Running            0              58s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get deployment
NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
21csxxxmongo                     0/1     1            0           42m
nginx-deployment-pes1ug21csxxx   2/2     2            2           101s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get replicaset
NAME                                        DESIRED   CURRENT   READY   AGE
21csxxxmongo-86886fd98d                     1         1         0       42m
nginx-deployment-pes1ug21csxxx-67856bc4f5   2         2         2       112s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

Now change the replicas to 3 in the file and run the command again. (Line 8 in the .yaml file). Notice that it says "configured" and not "created" this time. Check the pods and replicaset again.



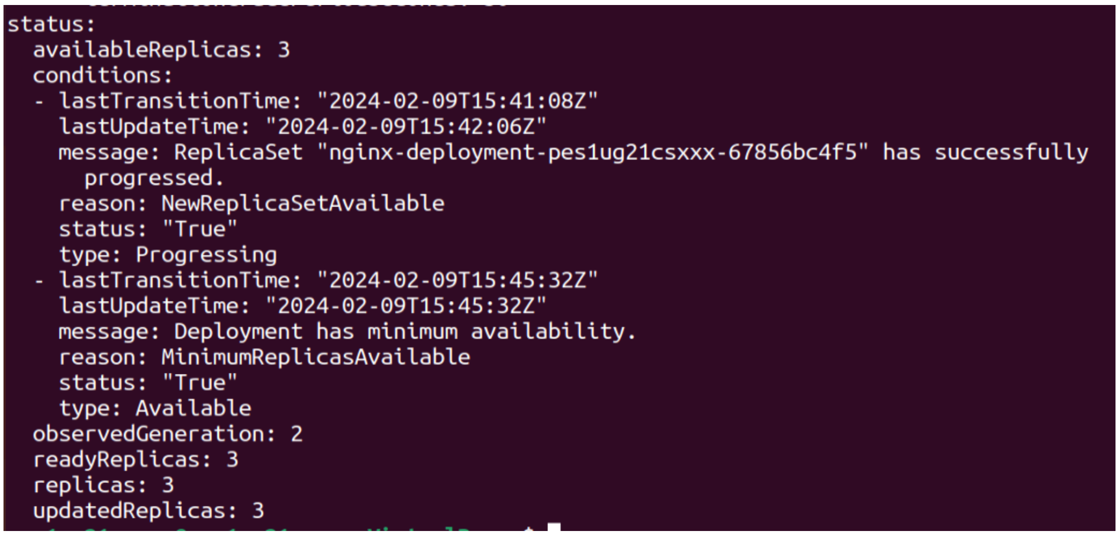
```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment-pes1ug21csxxx configured
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                                          READY   STATUS             RESTARTS        AGE
21csxxxmongo-86886fd98d-tccf2                 0/1     CrashLoopBackOff   13 (2m26s ago)  45m
nginx-deployment-pes1ug21csxxx-67856bc4f5-bhszl   1/1     Running            0               4m36s
nginx-deployment-pes1ug21csxxx-67856bc4f5-cf4wq   1/1     Running            0               4m36s
nginx-deployment-pes1ug21csxxx-67856bc4f5-kkbxk   1/1     Running            0               15s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get replicaset
NAME                                        DESIRED   CURRENT   READY   AGE
21csxxxmongo-86886fd98d                     1         1         0       45m
nginx-deployment-pes1ug21csxxx-67856bc4f5   3         3         3       4m43s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

There are 3 parts to any configuration file as discussed above. The third part is Status. It is added to the configuration file automatically by Kubernetes and it can be viewed.

✓ **SCREENSHOT 4b: Kubectl get on yaml file.**

Command:

1. kubectl get deployment nginx-deployment-pes1ug21csxxx -o yaml



```
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: "2024-02-09T15:41:08Z"
    lastUpdateTime: "2024-02-09T15:42:06Z"
    message: ReplicaSet "nginx-deployment-pes1ug21csxxx-67856bc4f5" has successfully
      progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: "2024-02-09T15:45:32Z"
    lastUpdateTime: "2024-02-09T15:45:32Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 2
  readyReplicas: 3
  replicas: 3
  updatedReplicas: 3
```

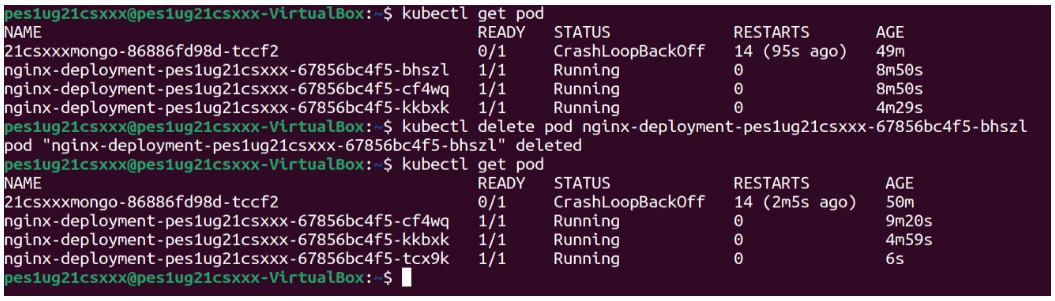
Task-5: Delete a pod to observe the self-healing feature.

✓ **SCREENSHOT 5a: Delete pod.**

Command:

1.kubectl delete pod <pod_name>

2.kubectl get pod



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                                          READY   STATUS             RESTARTS      AGE
21csxxxmongo-86886fd98d-tccf2                 0/1     CrashLoopBackOff   14 (95s ago)  49m
nginx-deployment-pes1ug21csxxx-67856bc4f5-bhszl   1/1     Running            0             8m50s
nginx-deployment-pes1ug21csxxx-67856bc4f5-cf4wq   1/1     Running            0             8m50s
nginx-deployment-pes1ug21csxxx-67856bc4f5-kkbxk   1/1     Running            0             4m29s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl delete pod nginx-deployment-pes1ug21csxxx-67856bc4f5-bhszl
pod "nginx-deployment-pes1ug21csxxx-67856bc4f5-bhszl" deleted
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod
NAME                                          READY   STATUS             RESTARTS       AGE
21csxxxmongo-86886fd98d-tccf2                 0/1     CrashLoopBackOff   14 (2m5s ago)  50m
nginx-deployment-pes1ug21csxxx-67856bc4f5-cf4wq   1/1     Running            0              9m20s
nginx-deployment-pes1ug21csxxx-67856bc4f5-kkbxk   1/1     Running            0              4m59s
nginx-deployment-pes1ug21csxxx-67856bc4f5-tcx9k   1/1     Running            0              6s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

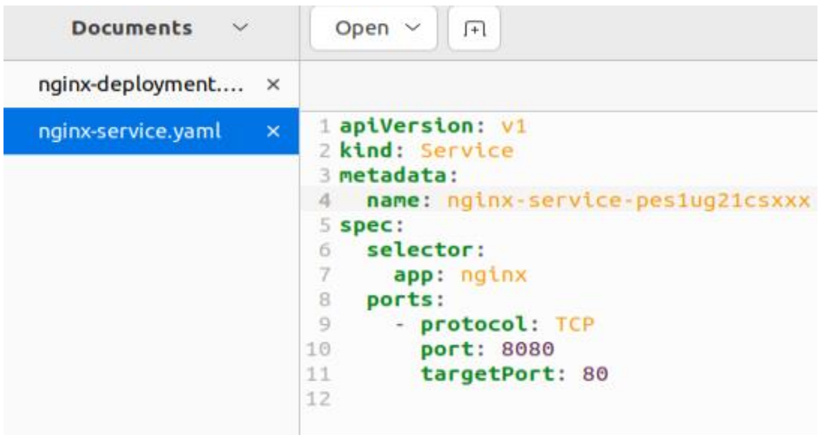
Notice that the pod has been replaced. This is part of the self-healing feature of Kubernetes.

Task-6: Connecting Services to Deployments.

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector.

The way a connection is established is using labels and selectors. The metadata portion of the configuration file consists of labels and the specification part consists of selectors.

The file 'nginx-service.yaml' provided by your professor will be used to create a deployment. **Make sure to change 'pes1ug21csxxx' to your SRN in the "name" field nginx-service-pes1ug21csxxx.**



```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-pes1ug21csxxx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
```

Notice the ports in the configuration file. Service needs to know to which pod it should forward the request to and these ports provide that information.

Port 80 is the default port. It's what gets used when no port is specified. 8080 is Tomcat's default port so as not to interfere with any other web server that may be running.

Create a service using the yaml file given and display the services.

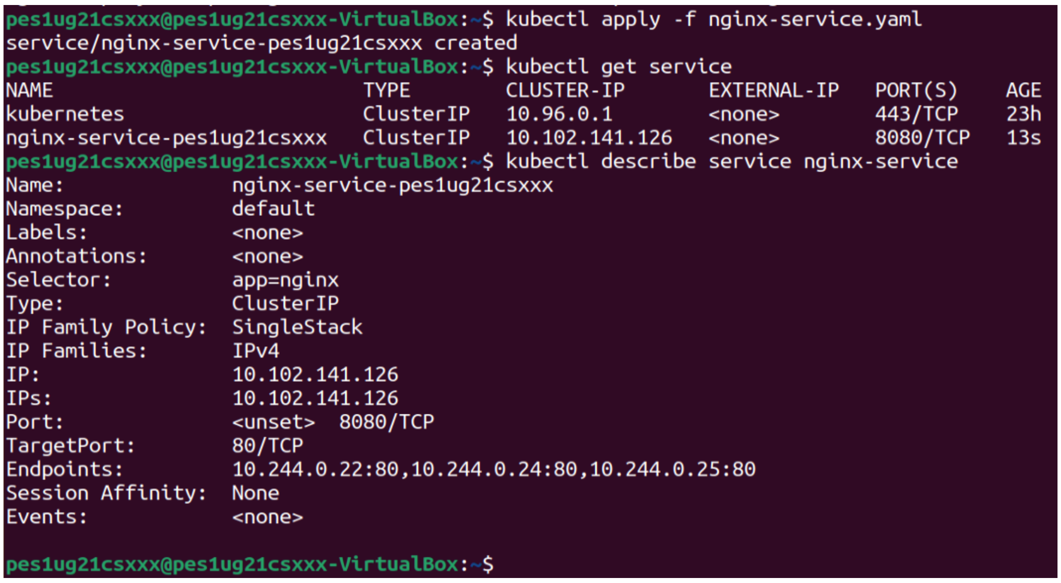
✓ **SCREENSHOT 6a: Kubectl apply and get command.**

Command:

1. kubectl apply -f <filename>

2.kubectl get service

3.kubectl describe service nginx-service



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl apply -f nginx-service.yaml
service/nginx-service-pes1ug21csxxx created
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get service
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP    23h
nginx-service-pes1ug21csxxx   ClusterIP   10.102.141.126   <none>        8080/TCP   13s
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl describe service nginx-service
Name:              nginx-service-pes1ug21csxxx
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=nginx
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.102.141.126
IPs:               10.102.141.126
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         10.244.0.22:80,10.244.0.24:80,10.244.0.25:80
Session Affinity:  None
Events:            <none>

pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

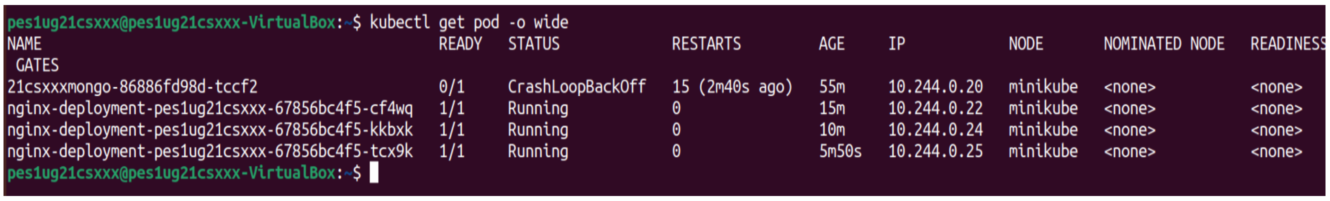
This shows the end points of the service.

✓ **SCREENSHOT 6b: kubectl get pod -o wide command.**

To see which pod it forwards the requests to, we can look at individual pods' information using the

command:

1.kubectl get pod -o wide



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl get pod -o wide
NAME                                           READY   STATUS             RESTARTS        AGE    IP            NODE       NOMINATED NODE   READINESS
 GATES
21csxxxmongo-86886fd98d-tccf2                  0/1     CrashLoopBackOff   15 (2m40s ago)  55m    10.244.0.20   minikube   <none>           <none>
nginx-deployment-pes1ug21csxxx-67856bc4f5-cf4wq   1/1  Running            0               15m    10.244.0.22   minikube   <none>           <none>
nginx-deployment-pes1ug21csxxx-67856bc4f5-kkbxk   1/1  Running            0               10m    10.244.0.24   minikube   <none>           <none>
nginx-deployment-pes1ug21csxxx-67856bc4f5-tcx9k   1/1  Running            0               5m50s  10.244.0.25   minikube   <none>           <none>
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

Task-7: Port Forwarding.

Make sure all pods of the deployment are up and running by running

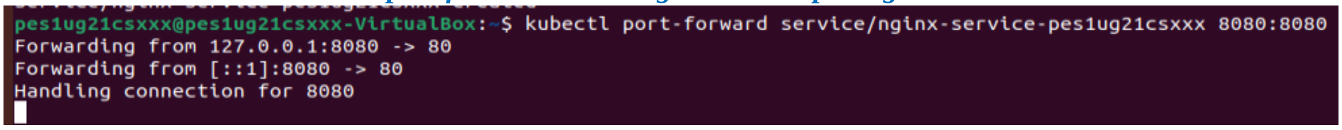
kubectl get pod

Expose the service.

✓ **SCREENSHOT 7a: Kubectl port-forward command .**

Command:

1. ***kubectl port-forward service/nginx-service-pes1ug21csxxx 8080:8080***

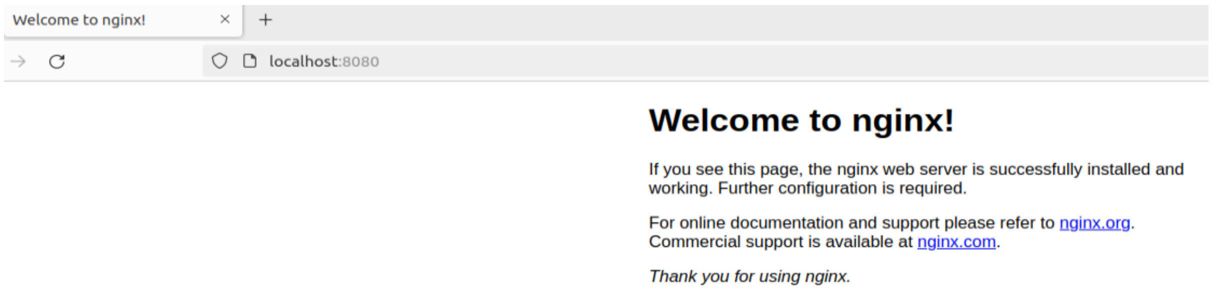


```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl port-forward service/nginx-service-pes1ug21csxxx 8080:8080
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
```

This basically says that if any HTTP request is received on post 8080, it has to be forwarded to this service which forwards it to its respective pod which will handle that request.

Access this service on your default browser on localhost port 8080.

✓ **SCREENSHOT 7b: Display welcome to nginx on web page.**



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

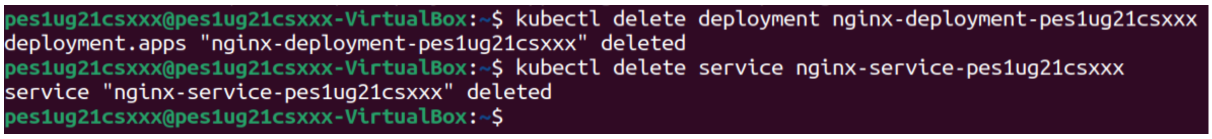
Task-8: Deleting service/deployment and Cleanup.

✓ **SCREENSHOT 8a: Delete nginx deployments.**

Commands:

1.kubectl delete deployment nginx-deployment-pes1ug21csxxx.

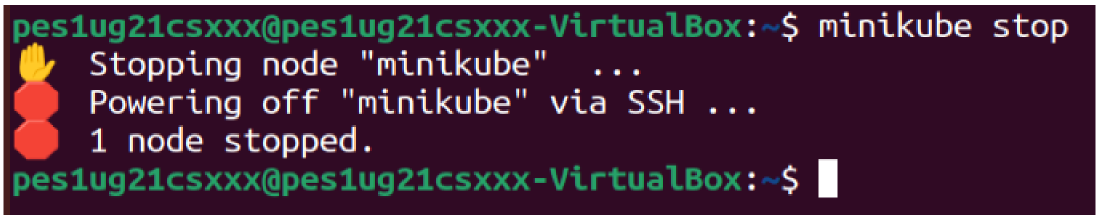
2.kubectl delete service nginx-service-pes1ug21csxxx.



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl delete deployment nginx-deployment-pes1ug21csxxx
deployment.apps "nginx-deployment-pes1ug21csxxx" deleted
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ kubectl delete service nginx-service-pes1ug21csxxx
service "nginx-service-pes1ug21csxxx" deleted
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$
```

✓ **SCREENSHOT 8b: Minikube stop – Do this after the 9th Task.**

minikube stop



```
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ minikube stop
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
🛑  1 node stopped.
pes1ug21csxxx@pes1ug21csxxx-VirtualBox:~$ █
```

Task-9: Expose an external IP address to access an Application in a cluster (To be done by the student).

Problem Statement:

1) Create a deployment named nginx-<srn> with image as nginx.
2) Expose the deployment which automatically creates the service to be exposed
3) The service should be of type NodePort or LoadBalancer
4) Access the service on your browser using minikube ip address and the Nodeport or LoadBalancer and the respective port

Note:

1) CLI alone is sufficient to achieve this. .yaml configuration files are not required.
2) Do not try to display the webpage by using port-forwarding as demonstrated in Section 7. Note: Read the difference between the two ways for better understanding.
3) Make sure the firewall is turned off. Open the browser in incognito mode in case the page doesn't load.
4) Windows users should access the service using the external IP address(in case of LoadBalancer service) or the IP Address used by minikube to tunnel(NOT minikube IP)(in case of NodePort service). This is because the direct IP of docker containers is not available on Windows.
5) VM/Linux users should access the service using external IP address(in case of LoadBalancer service) or minikube IP address(in case of NodePort service).

- ✓ **SCREENSHOT 9a: The command which exposes specifies the type of service (NodePort/LoadBalancer)**
- ✓ **SCREENSHOT 9b: kubectl get service command which displays the node port**
- ✓ **SCREENSHOT 9c: minikube IP address**
- ✓ **SCREENSHOT 9d: the webpage with the IP Address visible. (If the IP Address is not visible in the screenshot, you will lose significant portion of marks w.r.t. Section 9)**