

Assignment 2b - KUBERNETES

Tasks:

- 1 Installation
- 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 a cluster

Section 1: Installation

kubectl and minikube installation:

For Linux and MacOS users:

<https://kubernetes.io/docs/tasks/tools/>

Advised to logout/restart your system after installing kubectl before you begin installing minikube

<https://minikube.sigs.k8s.io/docs/start/> - Steps 1 and 2

Make sure docker is up and running. (Run: `docker run hello-world`)

For Windows/Docker Desktop users:

Make sure docker is up and running. (Run: `docker run hello-world`)

Open Docker Desktop → Settings → Kubernetes → Enable Kubernetes → Apply and Restart

Run `kubectl version` to ensure that kubectl is working properly.

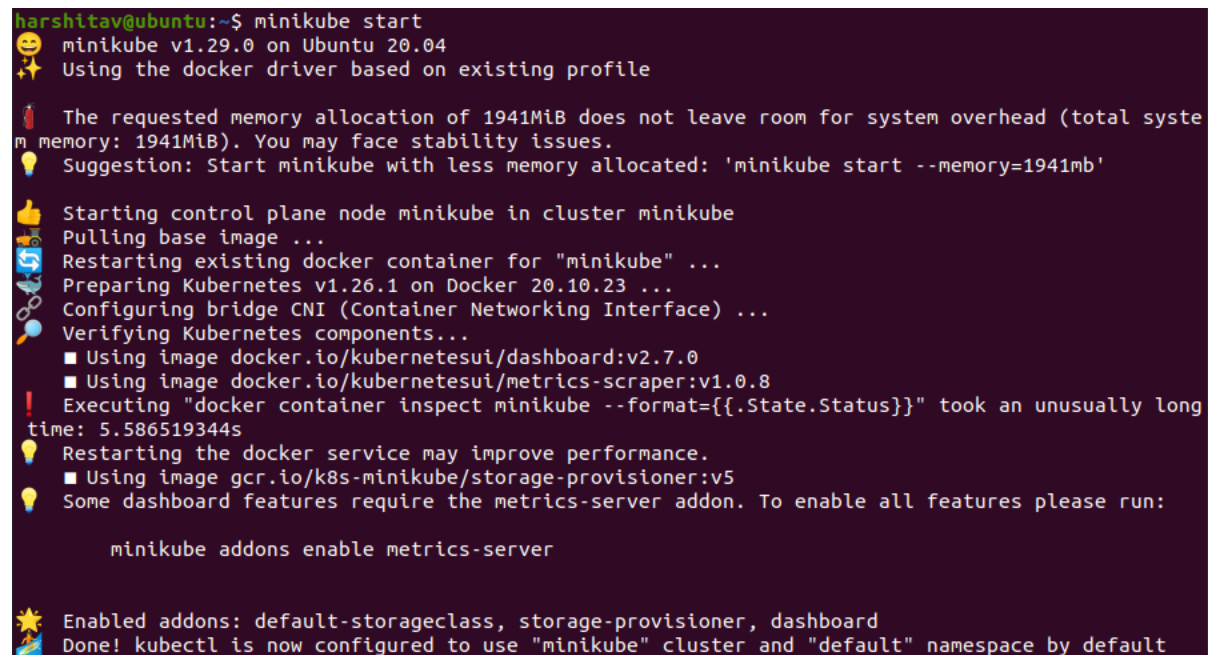
Open command prompt as administrator and run `winget install minikube`.

Close the command prompt and re-open it and run `minikube start`

Use the administrator mode to perform all the exercises to avoid permission issues.

Screenshot 1a:

```
minikube start
```



```
harshitav@ubuntu:~$ minikube start
🐹 minikube v1.29.0 on Ubuntu 20.04
🌟 Using the docker driver based on existing profile

💡 The requested memory allocation of 1941MiB does not leave room for system overhead (total system memory: 1941MiB). You may face stability issues.
💡 Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1941mb'

👉 Starting control plane node minikube in cluster minikube
🔄 Pulling base image ...
🔄 Restarting existing docker container for "minikube" ...
🔄 Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
🔄 Configuring bridge CNI (Container Networking Interface) ...
🔍 Verifying Kubernetes components...
   ■ Using image docker.io/kubernetesui/dashboard:v2.7.0
   ■ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
! Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 5.586519344s
💡 Restarting the docker service may improve performance.
   ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
💡 Some dashboard features require the metrics-server add-on. To enable all features please run:

    minikube addons enable metrics-server

🌟 Enabled addons: default-storageclass, storage-provisioner, dashboard
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

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

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)
 - a. [Kubernetes Horizontal Pod Autoscaler](#)

Quick Points:

1. Ensure Docker is installed, up and running before using minikube.
2. Pods may take some time (a few minutes in a bad network) initially to start up, this is normal.
3. 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.

Assignment 2 deliverables:

1. Section 1: Installation
 - Screenshot 1a - Minikube running successfully
2. Section 2: Creating pods and deployments, Editing them and observing Rollback:-
 - Screenshot 2a - get nodes, pod and services command.
 - Screenshot 2b- Deployment created.
 - Screenshot 2c- get deployment and pod command .
 - Screenshot 2d- editing '-image:nginx.'
 - Screenshot 2e- showing edited deployment.
 - Screenshot 2f- deployment is rolled back.
 - Screenshot 2g- showing original nginx image.
3. Section 3: Debugging Pods:-
 - Screenshot 3a - Kubectl logs displayed.
 - Screenshot 3b- Kubectl 'describe pod ' command.
 - Screenshot 3c - Create mongo deployment.
 - Screenshot 3d - Delete both requirements.
4. Section 4: Applying configuration files:-
 - Screenshot 4a - Kubectl apply command on yaml file.
 - Screenshot 4b- Kubectl get on yaml file
5. Section 5: Delete a pod to observe the self-healing feature.
 - Screenshot 5a - Deleted pod:-
6. Section 6 : Connecting Services to Deployments
 - Screenshot 6a- Kubectl apply and get command.
 - Screenshot 6b-kubectl get pod -o wide command
7. Section 7: Port Forwarding:-
 - Screenshot 7a -Kubectl port-forward command

- Screenshot 7b- Display welcome to nginx on web page
- Section 8: Deleting service/deployment and Cleanup
 - Screenshot 8a - Delete nginx deployments
 - Screenshot 8b - stop minikube
 - Section 9: Expose an external IP address to access an Application in a cluster
 - Screenshot 9a- the command which exposes specifies the type of service (NodePort)
 - 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)

Section 2: Creating pods and deployments, Editing them and observing Rollback

Firstly, view the nodes, pods and services present currently.

```
kubectl get nodes
```

```
kubectl get pod
```

```
kubectl get services
```

Screenshot 2a:

```
harshitav@ubuntu:~$ kubectl get nodes
NAME      STATUS   ROLES    AGE   VERSION
minikube   Ready    control-plane  19m   v1.26.1
harshitav@ubuntu:~$ kubectl get pod
No resources found in default namespace.
harshitav@ubuntu:~$ kubectl get services
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes  ClusterIP  10.96.0.1     <none>         443/TCP    19m
```

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:

```
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).

Create a deployment with the name of the deployment being your SRN and the image nginx

```
kubectl create deployment peslug20csxxx --image=nginx
```

*SRN has to be in lowercase.

Screenshot 2b:

```
harshitav@ubuntu:~$ kubectl create deployment peslug20csxxx --image=nginx
deployment.apps/peslug20csxxx created
```

This command downloads the latest nginx image from DockerHub

Now run

```
kubectl get deployment and kubectl get pod
```

Screenshot 2c:

```
harshitav@ubuntu:~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
peslug20csxxx 1/1     1            1           2m36s
harshitav@ubuntu:~$ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
peslug20csxxx-7f4c796c86-7x72h 1/1     Running   0           2m41s
```

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
```

```
harshitav@ubuntu:~$ kubectl get replicaset
NAME          DESIRED   CURRENT   READY   AGE
peslug20csxxx-7f4c796c86 1          1         1       7m37s
```

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 peslug20csxxx
```

```
harshitav@ubuntu:~$ kubectl describe deployment peslug20csxxx
Name:                peslug20csxxx
Namespace:            default
CreationTimestamp:    Mon, 06 Feb 2023 05:00:57 -0800
Labels:               app=peslug20csxxx
Annotations:          deployment.kubernetes.io/revision: 1
Selector:              app=peslug20csxxx
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=peslug20csxxx
  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:   peslug20csxxx-7f4c796c86 (1/1 replicas created)
Events:
  Type     Reason          Age    From          Message
  ----     -
  Normal   ScalingReplicaSet   2m35s  deployment-controller  Scaled up replica set peslug20csxxx-7f4c796c86 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.

Run:

```
kubectl edit deployment peslug20csxxx
```

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
```

- Now type ‘:1.16’

Screenshot 2d:

```
spec:
  containers:
  - image: nginx:1.16
    imagePullPolicy: Always
```

- Then type 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:

```
harshitav@ubuntu:~$ kubectl edit deployment peslug20csxxx
deployment.apps/peslug20csxxx edited
```

Screenshot 2f:

```
kubectl rollout undo deployment/peslug20csxxx
```

```
harshitav@ubuntu:~$ kubectl rollout undo deployment peslug20csxxx
deployment.apps/peslug20csxxx rolled back
```

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

Screenshot 2g:

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

Section 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.

```
kubectl get pod
```

Screenshot 3a:

```
kubectl logs <pod_name>
```

```
harshitav@ubuntu:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
peslug20csxxx-57fbc45849-p6z9v     1/1     Running   0           11m
harshitav@ubuntu:~$ kubectl logs peslug20csxxx-57fbc45849-p6z9v
/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: 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
2023/02/03 11:16:43 [notice] 1#1: using the "epoll" event method
2023/02/03 11:16:43 [notice] 1#1: nginx/1.22.1
2023/02/03 11:16:43 [notice] 1#1: built by gcc 10.2.1 20211010 (Debian 10.2.1-6)
2023/02/03 11:16:43 [notice] 1#1: OS: Linux 5.15.0-58-generic
2023/02/03 11:16:43 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/02/03 11:16:43 [notice] 1#1: start worker processes
2023/02/03 11:16:43 [notice] 1#1: start worker process 30
2023/02/03 11:16:43 [notice] 1#1: start worker process 31
```

You can see all the state changes for a pod to debug it in the following manner (Under the “Events” section – scroll down to view it):

```
kubectl describe pod <pod_name>
```

Screenshot 3b: (Screenshot of the “Events” section)

```
Events:
  Type     Reason      Age    From          Message
  ----     -
Normal    Scheduled   13m    default-scheduler    Successfully assigned default/peslug20csxxx-57fbc45849-p6z9v to minkube
Normal    Pulling     13m    kubelet          Pulling image "nginx:1.22"
Normal    Pulled      13m    kubelet          Successfully pulled image "nginx:1.22" in 15.39246965s (15.392478048s including waiting)
Normal    Created     13m    kubelet          Created container nginx
Normal    Started     13m    kubelet          Started container nginx
```

Similarly, the terminal of the application is also used for debugging. The following steps show how to use a mongo terminal.

Create a deployment with a mongo image.

```
kubectl create deployment peslug20csxxx-mongo --image=mongo
```

```
harshitav@ubuntu:~$ kubectl create deployment peslug20csxxx-mongo --image=mongo
deployment.apps/peslug20csxxx-mongo created
```

```
kubectl get pod
```

Wait till the container is created and the pod is in Running State. Once it is, run the following command to get the mongo terminal:

Screenshot 3c:

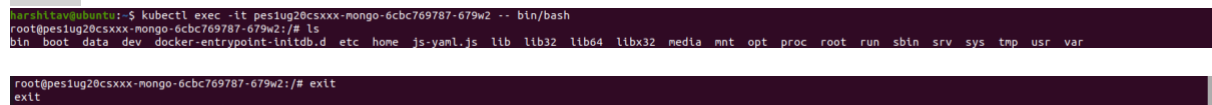
```
kubectl exec -it <pod_name> -- bin/bash
```

-it stands for Interactive terminal

Run

```
ls
```

```
exit
```



```
harshitav@ubuntu:~$ kubectl exec -it peslug20csxxx-mongo-6cbc769787-679w2 -- bin/bash
root@peslug20csxxx-mongo-6cbc769787-679w2:/# ls
bin boot data dev docker-entrypoint-initdb.d etc home js-yaml.js lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var

root@peslug20csxxx-mongo-6cbc769787-679w2:/# exit
exit
```

```
kubectl delete deployment <deployment_name>
```

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

Screenshot 3d:



```
harshitav@ubuntu:~$ kubectl delete deployment peslug20csxxx
deployment.apps "peslug20csxxx" deleted
harshitav@ubuntu:~$ kubectl delete deployment peslug20csxxx-mongo
deployment.apps "peslug20csxxx-mongo" deleted
```

Section 4: Applying configuration files and Scaling

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 ‘peslug20csxxx’ to your SRN in the “name” field nginx-deployment-peslug20csxxx


```

1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment-peslug20csxxx
5   labels:
6     app: nginx
7 spec:
8   replicas: 3
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 -f <filename>
```

Note: For Windows, the filename should be within double quotes

```

harshitav@ubuntu:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment-peslug20csxxx created

```

Check the pods, deployment and replicaset.

```

harshitav@ubuntu:~$ kubectl get deployment
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment-peslug20csxxx      2/2     2            2           101s
harshitav@ubuntu:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-peslug20csxxx-95585b474-gpgmz  1/1     Running   0          107s
nginx-deployment-peslug20csxxx-95585b474-m6h4v  1/1     Running   0          106s
harshitav@ubuntu:~$ kubectl get replicaset
NAME                                DESIRED   CURRENT   READY   AGE
nginx-deployment-peslug20csxxx-95585b474  2         2         2       119s

```

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.

```

harshitav@ubuntu:~$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment-peslug20csxxx configured
harshitav@ubuntu:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-peslug20csxxx-95585b474-gpgmz  1/1     Running   0          3m7s
nginx-deployment-peslug20csxxx-95585b474-m6h4v  1/1     Running   0          3m6s
nginx-deployment-peslug20csxxx-95585b474-rrvdp  0/1     ContainerCreating   0          3s
harshitav@ubuntu:~$ kubectl get replicaset
NAME                                DESIRED   CURRENT   READY   AGE
nginx-deployment-peslug20csxxx-95585b474  3         3         2       3m38s

```

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 by:

Screenshot 4b:

```
kubectl get deployment nginx-deployment-peslug20csxxx -o yaml
```

```

harshitav@ubuntu:~$ kubectl get deployment nginx-deployment -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  kubectl.kubernetes.io/last-applied-configuration: |
    {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx-deployment","namespace":"default"},"spec":{"replicas":3,"selector":{"matchLabels":{"app":"nginx"},"template":{"metadata":{"labels":{"app":"nginx"},"spec":{"containers":[{"image":"nginx:1.16","name":"nginx","ports":[{"containerPort":8080}]}}}}}
  creationTimestamp: "2023-02-03T11:46:21Z"
  generation: 2
  labels:
    app: nginx
  name: nginx-deployment
  namespace: default
  resourceVersion: "5866"
  uid: 1df5ec51-d6bc-4587-a774-f12cbd6bed02
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:1.16
        name: nginx
        ports:
        - containerPort: 8080
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: "2023-02-03T11:46:21Z"
    lastUpdateTime: "2023-02-03T11:46:46Z"
    message: ReplicaSet "nginx-deployment-95585b474" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: "2023-02-03T11:49:49Z"
    lastUpdateTime: "2023-02-03T11:49:49Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 2
  readyReplicas: 3
  replicas: 3
  updatedReplicas: 3

```

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

Screenshot 5a: (Should contain the list of pods before deleting, deletion command and list of pods after deletion)

```
kubectl delete pod <pod_name>
```

```

harshitav@ubuntu:~$ kubectl delete pod nginx-deployment-pes1ug20csxxx-95585b474-gpgnz
pod "nginx-deployment-pes1ug20csxxx-95585b474-gpgnz" deleted

```

```
kubectl get pod
```

```

harshitav@ubuntu:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-pes1ug20csxxx-95585b474-2757g  1/1     Running   0           92s
nginx-deployment-pes1ug20csxxx-95585b474-n6h4v  1/1     Running   0          6m13s
nginx-deployment-pes1ug20csxxx-95585b474-rrvdp  1/1     Running   0          3m18s

```

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

Section 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 ‘pes1ug20csxxx’ to your SRN in the “name” field nginx-service-pes1ug20csxxx**

```

1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nginx-service-pes1ug20csxxx
5 spec:
6   selector:
7     app: nginx
8   ports:
9     - protocol: TCP
10       port: 8080
11       targetPort: 80
12

```

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.

```
kubectl apply -f <filename>
```

```
kubectl get service
```

Screenshot 6a:

```
harshitav@ubuntu:~$ kubectl apply -f nginx-service.yaml
service/nginx-service created
harshitav@ubuntu:~$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	114m
nginx-service	ClusterIP	10.110.155.16	<none>	80/TCP	13s

```
kubectl describe service nginx-service
```

```
harshitav@ubuntu:~$ kubectl describe service nginx-service
```

```
Name:
Namespace:
Labels:
Annotations:
Selector:
Type:
IP Family Policy:
IP Families:
IP:
IPs:
Port:
TargetPort:
Endpoints:
Session Affinity:
Events:
```

```
nginx-service
default
<none>
<none>
app=nginx
ClusterIP
SingleStack
IPv4
10.110.155.16
10.110.155.16
<unset> 80/TCP
8080/TCP
10.244.0.15:8080,10.244.0.16:8080,10.244.0.17:8080
None
<none>
```

This shows the end points of the service. To see which pod it forwards the requests to, we can look at individual pods' information using the command:

```
kubectl get pod -o wide
```

Screenshot 6b:

```
harshitav@ubuntu:~$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
nginx-deployment-95585b474-4xx6x	1/1	Running	0	9m32s	10.244.0.17	minikube	<none>	<none>
nginx-deployment-95585b474-64wk8	1/1	Running	0	12m	10.244.0.15	minikube	<none>	<none>
nginx-deployment-95585b474-dmpfc	1/1	Running	0	12m	10.244.0.16	minikube	<none>	<none>

Section 7: Port Forwarding:

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

```
kubectl get pod
```

Expose the service using the command:

Screenshot 7a:

```
kubectl port-forward service/nginx-service-peslug20csxxx 8080:8080
```

```
harshitav@ubuntu:~$ kubectl port-forward service/nginx-service-peslug20csxxx 8080:8080
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::]:8080 -> 80
Handling connection for 8080
```

This basically says that if any HTTP request is received on port 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:



Section 8: Deleting service/deployment and Cleanup

Screenshot 8a:

```
kubectl delete deployment nginx-deployment-peslug20csxx
```

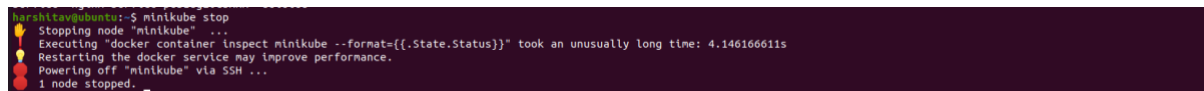
```
kubectl delete service nginx-service-peslug20csxxx
```



Run this command after Section 9!!

Screenshot 8b:

```
minikube stop
```



This ensures all resources are stopped and not seen when running the Kubernetes commands.

Section 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
- 4) Access the service on your browser using minikube ip address and the Nodeport

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.

Deliverables of Section 9:

- **Screenshot 9a:** Screenshot of the command which exposes specifies the type of service (NodePort)

- **Screenshot 9b:** Screenshot of `kubectl get service` command which displays the node port
- **Screenshot 9c:** Screenshot of minikube IP address
- **Screenshot 9d:** Screenshot of 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)