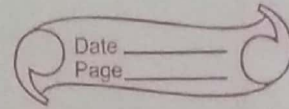


systemctl restart docker ← check docker Services



- Minikube:

Minikube is one of the easiest, most flexible and popular methods to run all-in-one or a multi-node local Kubernetes cluster directly on our local workstations.

- Requirements to run Minikube on our Local Workstation:

- Kubectl is a binary used to access and manage any Kubernetes cluster.
- Type-2 Hypervisor or Container Runtime.

- Start Kubernetes cluster:

minikube start:

Continue as root, use --force.

It will automatically install Kubernetes.

- Check the status:

minikube status

Kubectl ← Manage the Cluster.

- Stop Minikube:

minikube stop

- Delete minikube:

minikube delete

Dashboard:

minikube dashboard

2/2

Getting just the dashboard URL
minikube dashboard --url.

• Basic Kubectl Commands:

kubectl version.

To get namespaces:

kubectl get namespaces.

Details for Specific Namespace:

kubectl get all --namespace default.

To check status of nodes:

kubectl get nodes.

kubectl get nodes --namespace kube-system

To check status of pods:

kubectl get pods.

To check status of services

kubectl get services.

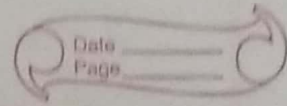
To check status of deployments:

kubectl get deployments.

To check status of Replicaset:

kubectl get replicaset.

kubectl get all ← check clusters.



Deployment : manage.



Services / Replicaset : manage



Pod



Container.

To check logs :

kubectl logs <pod-name>

• Details of node :

kubectl describe node <node-name>

kubectl describe node <node-name> --namespace <namespace-name>.

Details of Deployment :

kubectl describe deployment <deployment-name>

Details of services :

kubectl ~~services~~ describe services <services-name>

Details of replicaset :

kubectl describe replicaset <replicaset-name>

Details of pod :

kubectl describe pod <pod-name>

- To interact with kubernetes:

```
# kubectl cluster-info
```

```
# kubectl cluster-info dump
```

```
# kubectl cluster-info dump | jq
```

- # kubectl create namespace dai ← namespace_name

```
# kubectl get namespace
```

- ```
kubectl delete namespace <namespace_name>
```

- ## Using kubectl to create a Deployment

a) Deploy an app:

We need to provide the deployment name and app image location (include the full repository url for image hosted outside Docker hub).

```
kubectl create deployment <deploy_name> --
image = httpd:latest
```



-- interact with shell of

```
kubectl get deployments
kubectl get deployments -o wide
```

### NOTE :

Pods that are running inside Kubernetes are running on a private, isolated network.

```
kubectl proxy
```

```
curl http://localhost:8001/version.
```

### ⑥ Exploring Your App:

```
kubectl describe pods
```

```
kubectl logs <pod_name>
```

```
kubectl logs <deployment_name> -c <container_name>
```

```
kubectl exec <pod_name> --env
```

```
kubectl exec <pod_name> --bash
```

```
kubectl exec -it lab1_56ds94bcb4-6w6b2 --bash
```

```
kubectl exec -it <deployment_name> -c
<container_name> --bash
```

### Create Service:

```
kubectl get svc /service
```

```
kubectl describe svc
```



- Accessing Apps:

Minikube type of service in kubernetes:

1. NodePort :

Is The way to get external traffic directly to your service.

## 2. Load balancer:

Load balancer service is the way to expose a service to the internet, each service gets its own IP Address.

```
kubectl get services
```

```
kubectl expose deployment /<deploy-name>
--type = "NodePort" --port 80
```

```
Kubectl describe services / <deploy_name>
```

```
kecbctl port-forward <pod> 80:80
```

# minikube service lab2 -n dai  
~~depla~~ service-name.      namespace-name.

Make ~~deployment~~ <sup>namespace</sup> → CDAC

Make 2 deployments → 1. DBDA → `httpd:latest`  
2. PITISS → `httpd:latest`

Expose service publically.



## ① Create replica

# kubectl create deployment lab3 --replicas=5  
--image=nginx

### Scale :

If the deployment named httpd's <sup>replicasets is 3</sup> current size is 2, Scale httpd to 5.

# kubectl scale --current-replicas=3 --replicas=5  
deployment/lab3

## • Performing a Rolling Update:

Note : To update the image of the application to version 2, use the set image command, followed by the deployment name and the new image version :

# kubectl set image deployments/<deploy\_name>  
<image-name>=repo\_name/image:v2

# kubectl get pods

# kubectl describe services/<deploy-name>

confirm the update by running

# kubectl rollout status

• To roll back the deployment to your last working version :

# kubectl rollout undo deployments/<deploy-name>



## • Rolling updates:

### Rollout:

- # kubectl set image deployments/test1 httpd=httpd:2.4
- # kubectl get deployments -o wide
- # kubectl rollout status deployments/test1

deployment name

### Rollback:

- # kubectl rollout undo deployments/test1
- # kubectl rollout history deployments/test1
- # kubectl rollout undo deployments/test1 --to-revision=2

- Create a busybox pod that echoes 'hello.world' and then exits, but have the pod deleted automatically when it's completed.

```
kubectl run <image-name> --image=busybox
 φ -- echo φ "hello world"
```

↑  
Use bash → to open terminal

```
kubectl logs busybox1
```