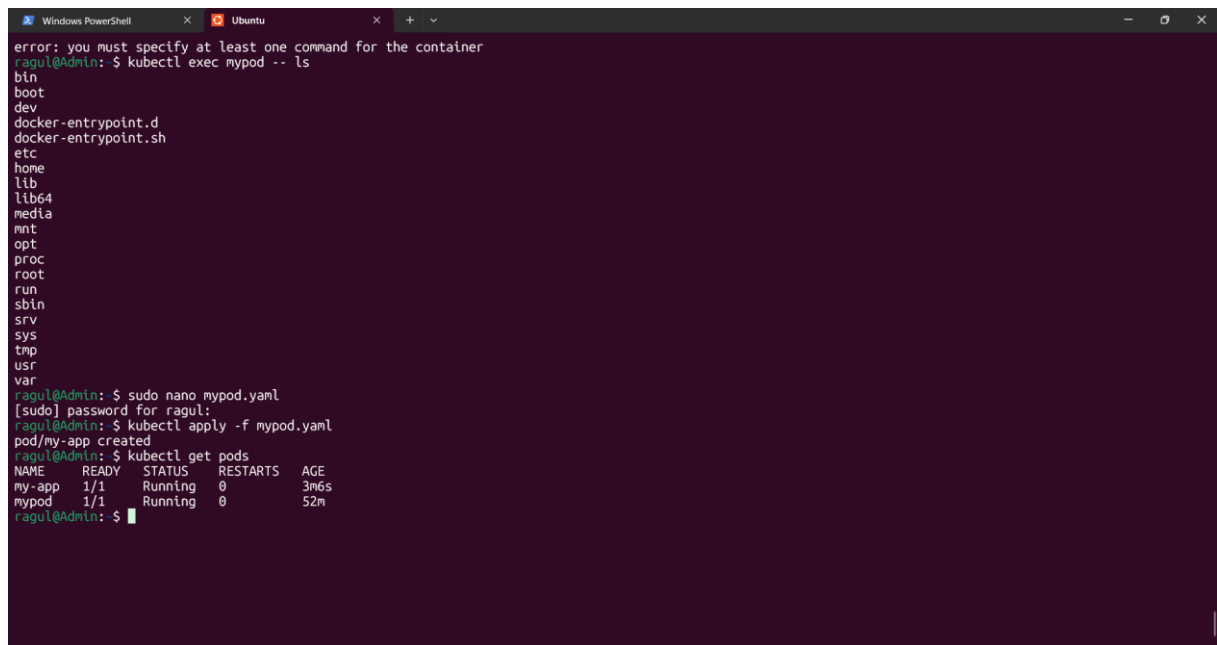


DevOps Day-4

Created pods and manipulated the files in pod.yml and created the running pods.

Created the pod.yml file:

A terminal window with a dark purple background. The window has two tabs: 'Windows PowerShell' and 'Ubuntu'. The terminal shows the following commands and output:

```
error: you must specify at least one command for the container
ragul@Admin: $ kubectl exec mypod -- ls
bin
boot
dev
docker-entrypoint.d
docker-entrypoint.sh
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
ragul@Admin: $ sudo nano mypod.yaml
[sudo] password for ragul:
ragul@Admin: $ kubectl apply -f mypod.yaml
pod/my-app created
ragul@Admin: $ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
my-app    1/1     Running   0           3m6s
mypod     1/1     Running   0           52m
ragul@Admin: $
```

apiVersion: v1 kind:

Pod metadata:

name: my-pod

labels:

app: my-web-app

type: backend

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

```
[sudo] password for ragul:
ragul@Admin: $ kubectl apply -f deploy.yml
error: the path "deploy.yml" does not exist
ragul@Admin: $ sudo nano deploy.yml
ragul@Admin: $ kubectl apply -f deploy.yml
deployment.apps/my-deploy created
service/my-service created
ragul@Admin: $ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
my-app        1/1     Running   1 (6m19s ago)  22h
my-deploy-df6c58bc8-mszc  0/1     ContainerCreating   0           4s
mypod         1/1     Running   1 (6m19s ago)  23h
ragul@Admin: $ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
my-app        1/1     Running   1 (6m24s ago)  22h
my-deploy-df6c58bc8-mszc  1/1     Running   0           9s
mypod         1/1     Running   1 (6m24s ago)  23h
ragul@Admin: $ minikube service my-service
Starting tunnel for service my-service.
NAME          NAME          TARGET PORT  URL
default       my-service    9000         http://192.168.49.2:30002
Opening service default/my-service in default browser...
http://127.0.0.1:36977
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
Stopping tunnel for service my-service.
ragul@Admin: $ curl http://192.168.49.2:30002/maven-web-app/
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
ragul@Admin: $
```

Kubectl commands:

\$ kubectl run <pod-name> --image=<image-name> --port=<container-port> \$

kubectl run my-pod --image=nginx --port=80

2. View all the pods

(In default namespace)

\$ kubectl get pods

(In All namespace)

\$ kubectl get pods -A

For a specific namespace

\$ kubectl get pods -n kube-system

For a specific type

\$ kubectl get pods <pod-name>

```
$ kubectl get pods <pod-name> -o wide
```

```
$ kubectl get pods <pod-name> -o yaml
```

```
$ kubectl get pods <pod-name> -o json
```

3. Describe a pod (View Pod details)

```
$ kubectl describe pod <pod-name>
```

```
$ kubectl describe pod my-pod
```

4. View Logs of a pod

```
$ kubectl logs <pod-name>
```

```
$ kubectl logs my-pod
```

Replica Set:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata: name:
```

```
my-deploy labels:
```

```
name: my-deploy
```

```
spec: replicas: 4
```

```
selector:
```

```
matchLabels:
```

```
  apptype: web-backend
```

```
strategy:
```

```
  type: RollingUpdate
```

```
template:
```

```
  metadata: labels:
```

```
    apptype: web-backend
spec:
containers:    -
name: my-app
image:
ports:
    - containerPort: 7070
```

```
kubectl create deployment webnginx2 --image=nginx:latest --replicas=1
```

Services (short name = svc):

Service is an abstraction that defines a logical set of pods and a policy to access them. Services enable network connectivity and load balancing to the pods that are part of the service, allowing other components within or outside the cluster to interact with the application.

Service Types: Kubernetes supports different types of services:

1. NodePort: Exposes the service on a static port on each selected node's IP. This type makes the service accessible from outside the cluster by the <NodeIP>:<NodePort> combination.
2. ClusterIP: Exposes the service on a cluster-internal IP. This type makes the service only reachable within the cluster.
3. LoadBalancer: Creates an external load balancer in cloud environments, which routes traffic to the service.

Kubernetes (K8s)

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

It provides a scalable and resilient framework for automating the deployment, scaling, and management of applications across clusters of servers.

A SMALL HISTORY OF K8S:

- ❑ In the early 2000s, Google started developing a system called Borg to manage their internal containerized applications.
- ❑ Borg enabled Google to run applications at scale, providing features such as automatic scaling, service discovery, and fault tolerance.
- ❑ In 2014, Google open-sourced a version of Borg called Kubernetes.
- ❑ Kubernetes was donated to the Cloud Native Computing Foundation (CNCF), a neutral home for open-source cloud-native projects, in July 2015.
- ❑ Kubernetes 1.8 added significant enhancements for storage, security, and networking. Key features included the stable release of the stateful sets API, expanded support for volume plugins, and improvements in security policies.

Control Plane /Master Node

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. Do not run user containers on this machine.

Node Components / Worker Nodes

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. Master Node: The master node is responsible for managing the cluster and coordinating the overall state of the system. It includes the following components:
 - a. API Server: The API server is the central control point for all interactions with the cluster. It exposes the Kubernetes API and handles requests from users and other components.
 - b. Scheduler: The scheduler is responsible for assigning workloads (pods) to individual worker nodes based on resource requirements, constraints, and other policies.
 - c. Controller Manager: The controller manager runs various controllers that monitor the cluster state and drive it towards the desired state. Examples include the replication controller, node controller, and service controller.
 - d. etcd: etcd is a distributed key-value store used by Kubernetes to store cluster state and configuration data.