Kubernetes Pods

- Pods have 3 types of containers
    - init containers
    - containers
    - ephemeral containers (debugging)
- init containers
    - are executed in a sequential order
    - are expected to be completed
- containers:
    - here we run the application or application component
    - if we write mutliple containers they start in parallel.
    - first container in spec is referred as main car and rest of containers are referred as side cars.
    - All containers in a Pod share same network namespace i.e. they can communicate with each other on localhost.
    - When we define containers its a good idea and best practice always to also incude atleast max memory and cpu in the manifest.
    - Always specify port information
- For verifying the basic manifest we can use playgrounds
    - k8s playground
    - killercoda Refer Here

Labels

- Refer Here for official docs
- Labels can be applied to k8s objects and later can be queried
- Querying k8s objects based on labels is referred as selctors in k8s
- Selectors
    - equality based:
        - we have two options `equals` and `not equals`
    - set based:
        - This gives more comprehensive way to select, we have multiple operators such as
            - in
            - not in
            - exists

Resources Requests and limits

- Using this we can specify the k8s resources limits Refer Here
- Requests: refer to lower boundary
- limits: refer to upper boundary

Activity 1: lets create a pod spec for nop commerce

- Refer Here for the changeset with pod spec

```
controlplane $ pwd
/root/manifests
controlplane $ kubectl apply -f activity1.yaml
pod/activity1 created
controlplane $ kubectl get po
NAME        READY   STATUS              RESTARTS   AGE
activity1   0/1     ContainerCreating   0          9s
controlplane $ kubectl get po -w
NAME        READY   STATUS              RESTARTS   AGE
activity1   0/1     ContainerCreating   0          12s
```

- Apply the manifest

- describe the pods

```
controlplane $ kubectl describe pod activity1
Name:             activity1
Namespace:        default
Priority:         0
Service Account:  default
Node:             node01/172.30.2.2
Start Time:       Thu, 13 Jun 2024 02:30:26 +0000
Labels:           app=nop
                  env=dev
                  purpose=understanding
Annotations:      cni.projectcalico.org/containerID: 4cc6d1725684b0faced9f37983dd3b558e7b3f40a27d7f1869ded752f8c9d696
                  cni.projectcalico.org/podIP: 192.168.1.4/32
                  cni.projectcalico.org/podIPs: 192.168.1.4/32
Status:           Running
IP:               192.168.1.4
IPs:
  IP:  192.168.1.4
Containers:
  nop:
    Container ID:   containerd://24b78a3db501cedfdbff3b5e2e3ec0002c29c098470e4d5fd3480df09155d842
    Image:          shaikkhajaibrahim/nopcommercejune2024:070624
    Image ID:       docker.io/shaikkhajaibrahim/nopcommercejune2024@sha256:deb0549b0368340eb26db30c8e584cafb3ffca52c15eba74
e4f3ffd07aa07939
    Port:           5000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 13 Jun 2024 02:31:23 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     500m
```

- Each pod gets an ip address

```
controlplane $ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP           NODE     NOMINATED NODE   READINESS GATES
activity1   1/1     Running   0          4m42s   192.168.1.4  node01   <none>           <none>
controlplane $
```

- When we get any k8s object we can ask to show labels

```
controlplane $ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP           NODE     NOMINATED NODE   READINESS GATES
activity1   1/1     Running   0          4m42s   192.168.1.4  node01   <none>           <none>
controlplane $ kubectl get pods --show-labels
NAME        READY   STATUS    RESTARTS   AGE     LABELS
activity1   1/1     Running   0          5m58s   app=nop,env=dev,purpose=understanding
controlplane $
```

- Note we can create k8s pods with imperative commands

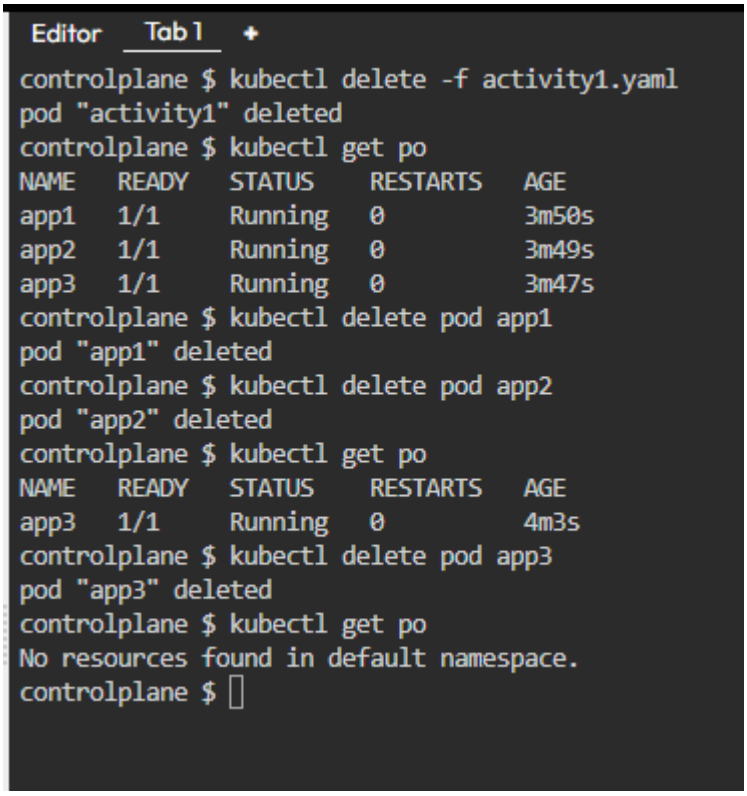
```
controlplane $ kubectl run --image nginx app1
pod/app1 created
controlplane $ kubectl run --image nginx app2
pod/app2 created
controlplane $ kubectl run --image nginx app3
pod/app3 created
controlplane $ kubectl get po
NAME        READY   STATUS              RESTARTS   AGE
activity1   1/1     Running             0          8m29s
app1        1/1     Running             0          10s
app2        0/1     ContainerCreating   0          9s
app3        0/1     ContainerCreating   0          7s
controlplane $
```

- We can select pods by labels

```
controlplane $ kubectl get pods --show-labels
NAME        READY   STATUS    RESTARTS   AGE    LABELS
activity1   1/1     Running   0          9m8s   app=nop,env=dev,purpose=understanding
app1        1/1     Running   0          49s    run=app1
app2        1/1     Running   0          48s    run=app2
app3        1/1     Running   0          46s    run=app3
controlplane $ kubectl get pods -l "env=dev"
NAME        READY   STATUS    RESTARTS   AGE
activity1   1/1     Running   0          9m43s
controlplane $ kubectl get pods -l "env"
NAME        READY   STATUS    RESTARTS   AGE
activity1   1/1     Running   0          9m53s
controlplane $ kubectl get pods -l "run"
NAME   READY   STATUS    RESTARTS   AGE
app1   1/1     Running   0          99s
app2   1/1     Running   0          98s
app3   1/1     Running   0          96s
controlplane $
```

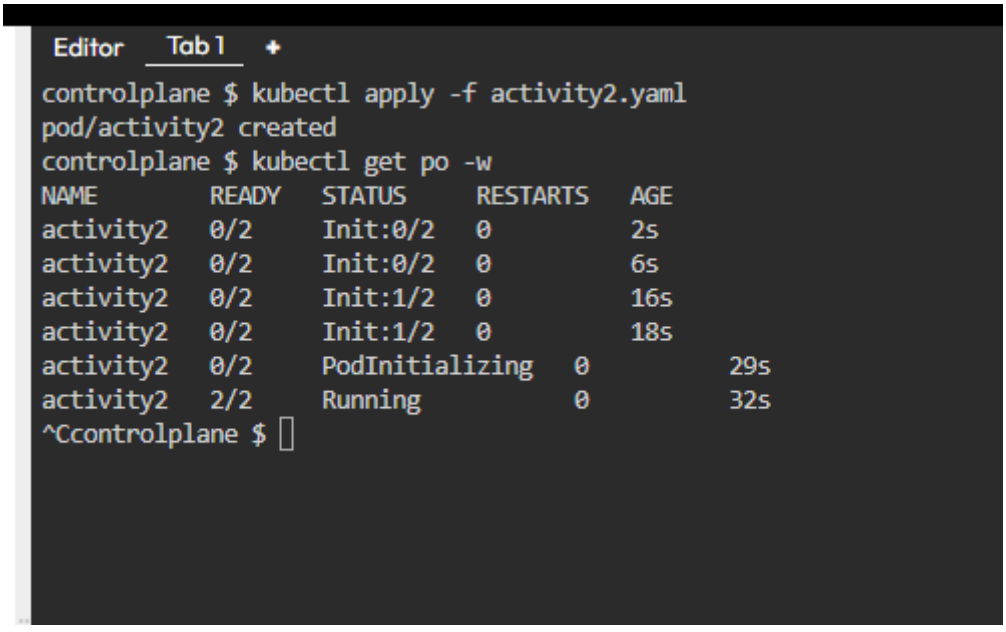
- Delete the pods
    - declartive `kubectl delete -f <manifest>`

- command `kubectl delete pod <name>`



Activity 2: Demonstrate init containers

- Order of creation is
  - init containers one by one
  - containers all at one shot



Refer Here

- in docker we can get into container and execute commands

```
docker container exec
docker container exec -it
```

- k8s also allows you to execute commands in container

```
kubectl exec podname -- command
kubectl exec -it podname -- command
```

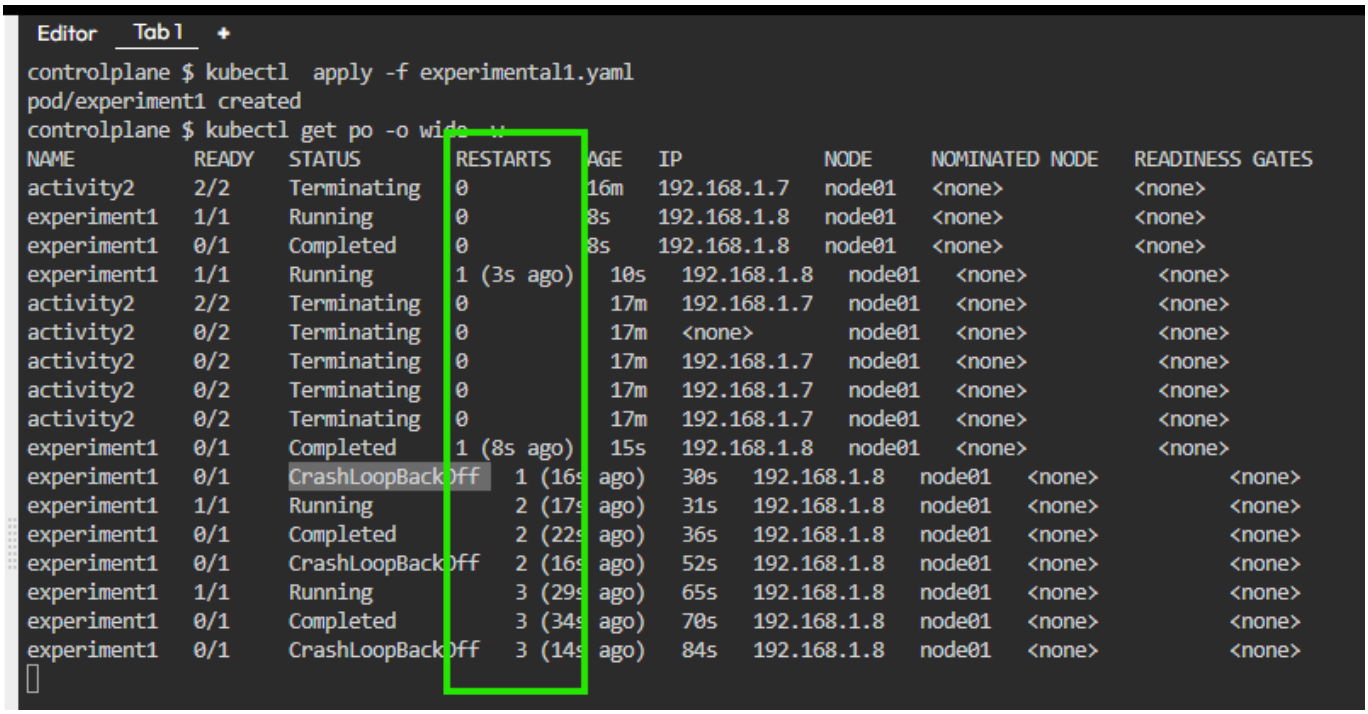
```
Editor   Tab 1   +
controlplane $ kubectl exec activity2 -- pwd
Defaulted container "maincar" out of: maincar, sidecar1, precondition1 (init), precondition2 (init)
/
controlplane $ kubectl exec activity2 -c sidecar1 -- uname
Linux
controlplane $ kubectl exec activity2 -c maincar -- uname
Linux
controlplane $ kubectl exec activity2 -c maincar -- uname -r
5.4.0-131-generic
controlplane $ kubectl exec activity2 -c sidecar1 -- uname -r
5.4.0-131-generic
controlplane $
```

```
controlplane $ kubectl exec activity2 -c sidecar1 -it -- /bin/sh
/ # nslookup kubernetes
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find kubernetes.cluster.local: NXDOMAIN

Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1

** server can't find kubernetes.svc.cluster.local: NXDOMAIN

** server can't find kubernetes.cluster.local: NXDOMAIN


** server can't find kubernetes.svc.cluster.local: NXDOMAIN

/ # cat /etc/resolv.conf
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.96.0.10
options ndots:5
/ #
```

- Exercise 3: create a pod manifest for mysql where we need to pass environmental variables
- Exercise 4: create a pod manifest for postgres where we need to pass environmental variables
- Desired state of k8s pod is to ensure container is in running,
- lets write a k8s spec where containers go into exited state after some time

```
---
apiVersion: v1
kind: Pod
metadata:
  name: experiment1
```

```
    labels:
      purpose: experiment
  spec:
    containers:
      - name: suicidal
        image: alpine
        args:
          - sleep
          - 5s
```



- kuberenetes will restart the container and display the error as CrashloopBackoff.