Agenda:

1. Create a simple pod
2. Create a pod in different namespaces
3. Creating a Pod in a Different Namespace using yaml configuration
4. Create a Pod running a container
5. Change the default namespace
6. Pod Running a Container That Exposes a Port
7. Pod Running a Container with Resource Requirements
8. Create a Pod with Resource Requests That Can’t Be Met by Any of the Nodes
9. Pod with Multiple Containers
10. Pod Lifecycle

What is a POD

- A pod is the basic building block of Kubernetes
- Basic unit of deployment
- A pod can have any number of containers running in it
- A pod is basically a wrapper around containers running on a node
- Containers in a pod have shared volumes, Linux namespaces, and cgroups. Each pod has a unique IP address and the port space is shared by all the containers in that pod. This means that different containers inside a pod can communicate with each other using their corresponding ports on localhost.

Creating a first pod with single container

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-1
spec:
  containers:
  - name: container-1
    image: nginx
```

Create a pod

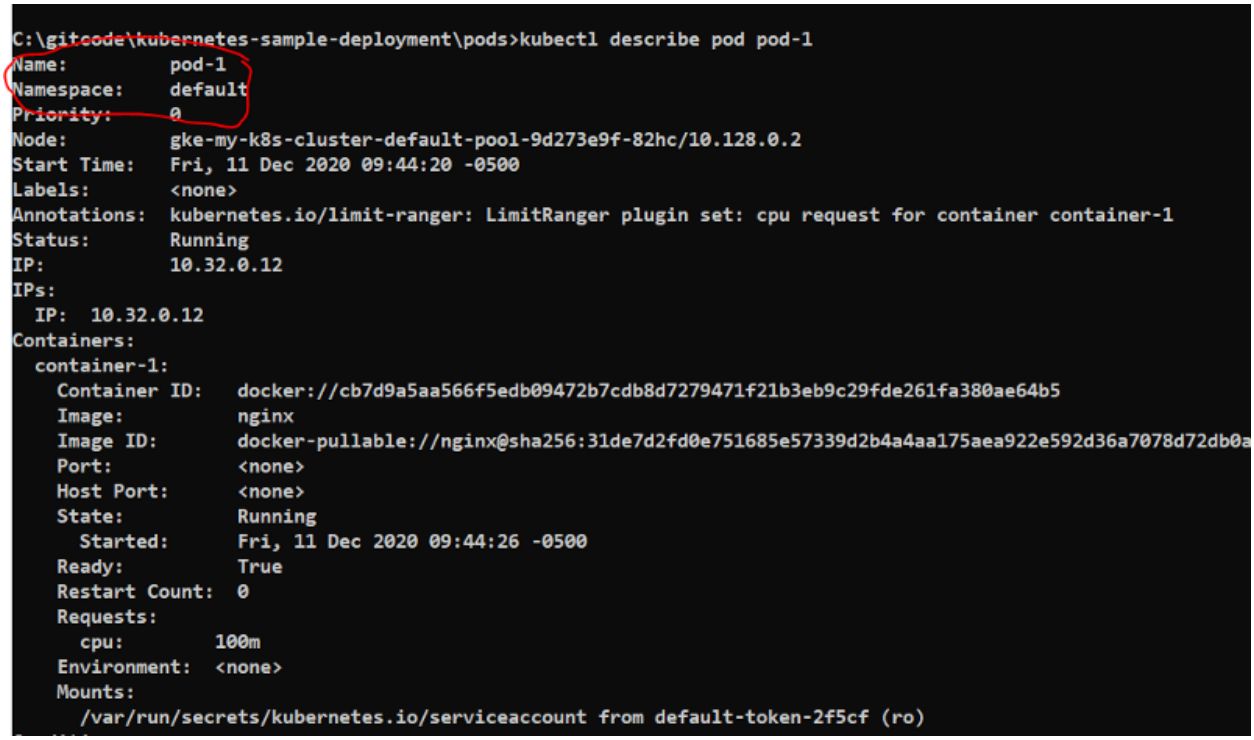
```
kubectl create -f pod-singlecontainer.yaml
kubectl get pods
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
pod-1   1/1     Running   0          105s

C:\gitcode\kubernetes-sample-deployment\pods>
```

Describe the pod

```
kubectl describe pod pod-1
```



Name: Pod name is unique in a particular namespace

Namespace: Kubernetes supports namespaces to create multiple virtual clusters within the same physical cluster.

Why we use namespace ?

When we have only one cluster and different teams are using that. In that case it would be great if every team create their resources in their own namespaces.

We want to separate the environment like dev, stage in different namespaces

Create a pod in different namespaces

First you can check how many namespaces are already present

```
kubectl get namespaces
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl get namespaces
NAME              STATUS   AGE
default           Active   46h
kube-node-lease   Active   46h
kube-public       Active   46h
kube-system       Active   46h
services          Active   22h

C:\gitcode\kubernetes-sample-deployment\pods>
```

Here, I am using the same yaml file to create a pod if I don't specify the namespace the command will fail with the error "**pod is already exist**"

```
kubectl --namespace=kube-public create -f pod-singlecontainer.yaml
```

Verify the pod

Now to check pods you need to define the namespace else by default `kubectl get pods` command will show only pods which are running in the default namespace.

```
kubectl --namespace kube-public get pods
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl --namespace kube-public get pods
NAME    READY   STATUS    RESTARTS   AGE
pod-1   1/1     Running   0          55s

C:\gitcode\kubernetes-sample-deployment\pods>
```

Pod in a Different Namespace by using yaml configuration

In the yaml file we will define which namespace to use while creating a pod

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-1
  namespace: kube-public
spec:
  containers:
  - name: container-1
    image: nginx
```

Create and verify a Pod

```
kubectl create -f pod-namespacesinglecontainer.yaml
kubectl --namespace kube-public get pods
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl --namespace kube-public get pods
NAME                  READY   STATUS    RESTARTS   AGE
pod-1                 1/1     Running   0          3m33s
pod-1-with-namespace  1/1     Running   0          7s
```

Change the default namespace

So we saw that we need to explicitly define the namespace to show all the pods which are running in that namespace.

This is not a convenient way if we are doing this for each and every command

When we know that we are working only on one namespace than we can set that using the below command

```
kubectl config set-context --current --namespace kube-public
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl config set-context --current --namespace kube-public
Context "gke_kubernetes-cluster-298116_us-central1-a_my-k8s-cluster" modified.
```

Check the pods

```
kubectl get pods
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
pod-1                 1/1     Running   0          8m1s
pod-1-with-namespace  1/1     Running   0          4m35s

C:\gitcode\kubernetes-sample-deployment\pods>
```

Pod Running a Container

In this yaml file we are providing the configuration for a container as shown below:

```
kind: Pod
apiVersion: v1
metadata:
  name: command-pod
spec:
  containers:
  - name: container-with-command

image: ubuntu
    command:
    - /bin/bash
    - -ec
    - while :; do echo '.'; sleep 5; done
```

Create a pod

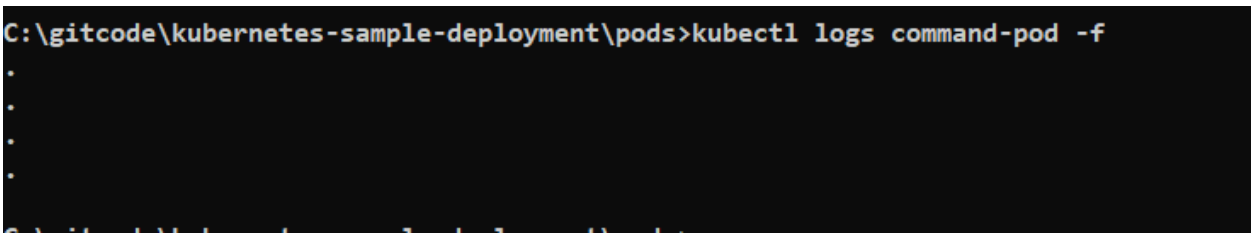
```
kubectl create -f pod-singlecontainer.yaml
```

Check the logs

Now we will check the logs of our container using the pod name

The **-f** *flag is to follow the logs on the container. That is, the log keeps updating in real-time*

```
kubectl logs command-pod -f
```



Pod Running a Container That Exposes a Port

In this yaml file we are using a `nginx` image and exposing it on the port `80`

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-exposed-port
spec:
  containers:
  - name: container-exposed-port
    image: nginx
    ports:
      - containerPort: 80
```

Create the pod

```
kubectl create -f pod-expose-pod.yaml
```

This pod should create a container and expose it on port `80`

Now we will use the `port-forward` to expose this port to the localhost or you can define the another port also using the second command

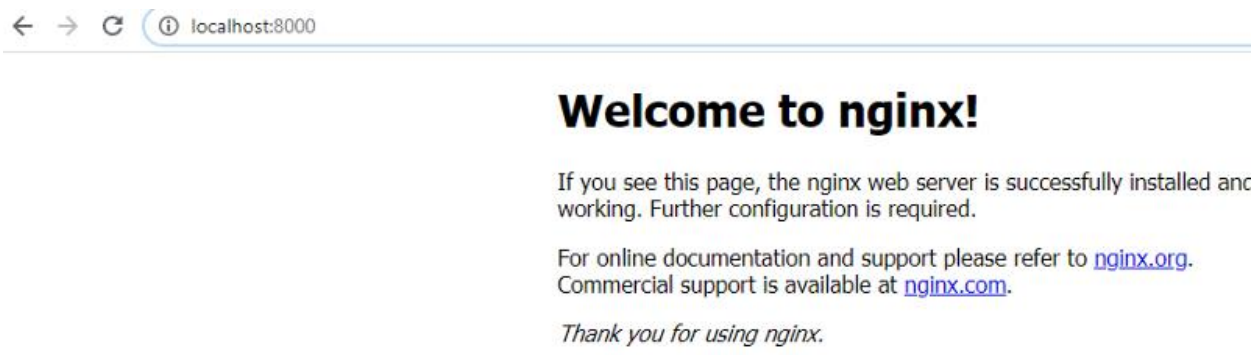
```
kubectl port-forward pod-exposed-port 80

kubectl port-forward pod-exposed-port 8000:80
```

```
C:\gitcode\kubernetes-sample-deployment\pods>kubectl port-forward pod-exposed-port 80
Forwarding from 127.0.0.1:80 -> 80
Forwarding from [::1]:80 -> 80
Handling connection for 80
```

Now you can access the url

```
http://localhost
http://localhost:8000
```



Pod Running a Container with Resource Requirements

In this YAML file, we define the

Memory requirement for our container

Minimum Memory—`64MB`

Maximum Memory- `128MB`

If the container tries to allocate more than 128 MB of memory, it will be killed with a status of **OOMKilled**.

CPU Requirement for our container

Minimum CPU— `0.5`

Maximum CPU- `1`

The minimum CPU requirement for CPU is 0.5 (which can also be understood as 500 milli-CPUs and can be written as **500m** instead of **0.5**) and the container will only be allowed to use a maximum of 1 CPU unit.

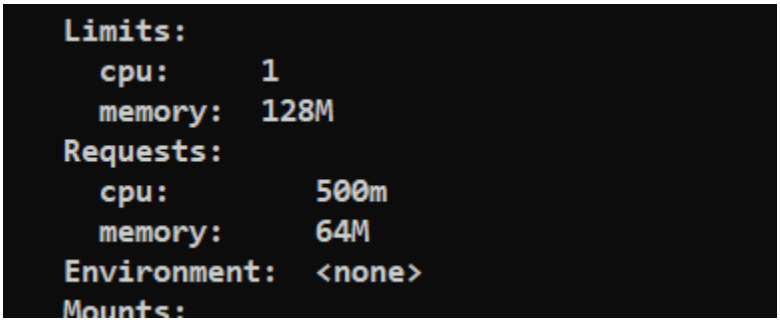
```
kind: Pod
apiVersion: v1
metadata:
  name: pod-resources
spec:
  containers:
  - name: container-resource-requirements
    image: nginx
    resources:
      limits:
        memory: "128M"
        cpu: "1"
      requests:
        memory: "64M"
        cpu: "0.5"
```

Create the pod

```
kubectl create -f pod-with-resources.yaml
```

Describe the pod

```
kubectl describe pod-resources
```



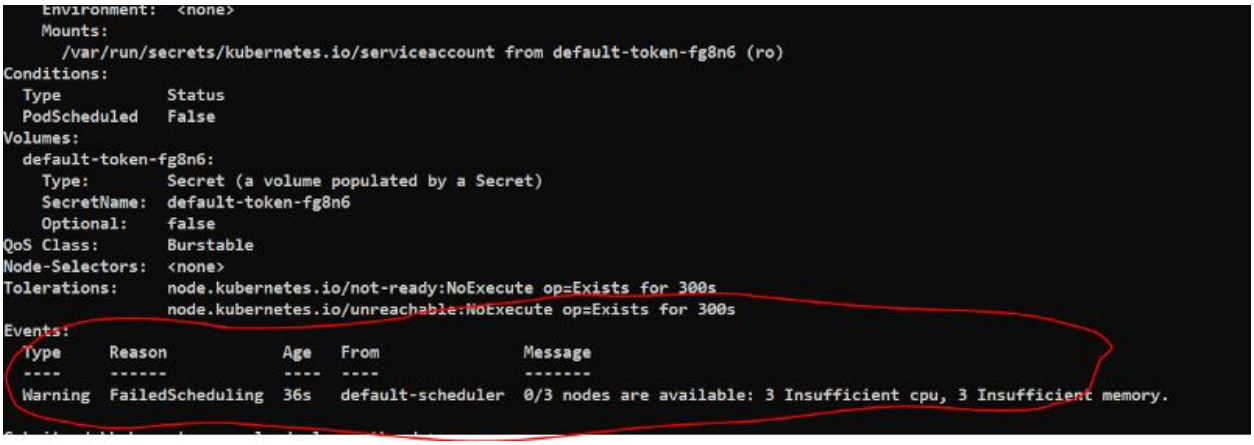
Pod with Resource Requests That Can’t Be Met by Any of the Nodes

In this yaml file we are using resources which are not available in our cluster nodes.

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-huge-resources
spec:
  containers:
  - name: container-resource-requirements
    image: nginx
    resources:
      limits:
        memory: "128G"
        cpu: "1000"
      requests:
        memory: "64G"
        cpu: "500"
```

Create a pod

```
kubectl create -f pod-with-huge-resources.yaml
```



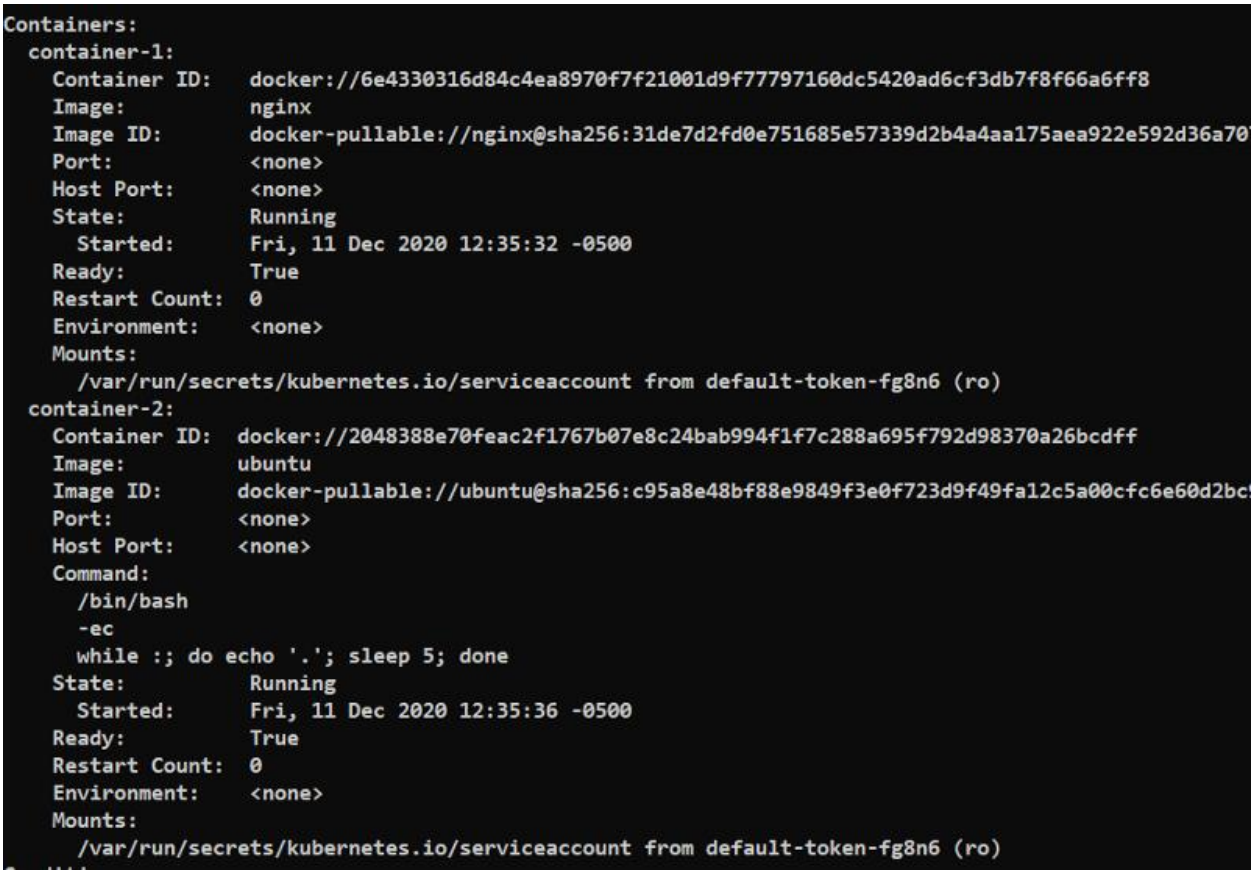
Pod with Multiple Containers Running inside It

In this yaml file you can see that we are creating two container inside a pod

```
kind: Pod
apiVersion: v1
metadata:
  name: multi-container
spec:
  containers:
  - name: container-1
    image: nginx
  - name: container-2
    image: ubuntu
    command:
    - /bin/bash
    - -ec
    - while :; do echo '.'; sleep 5; done
```

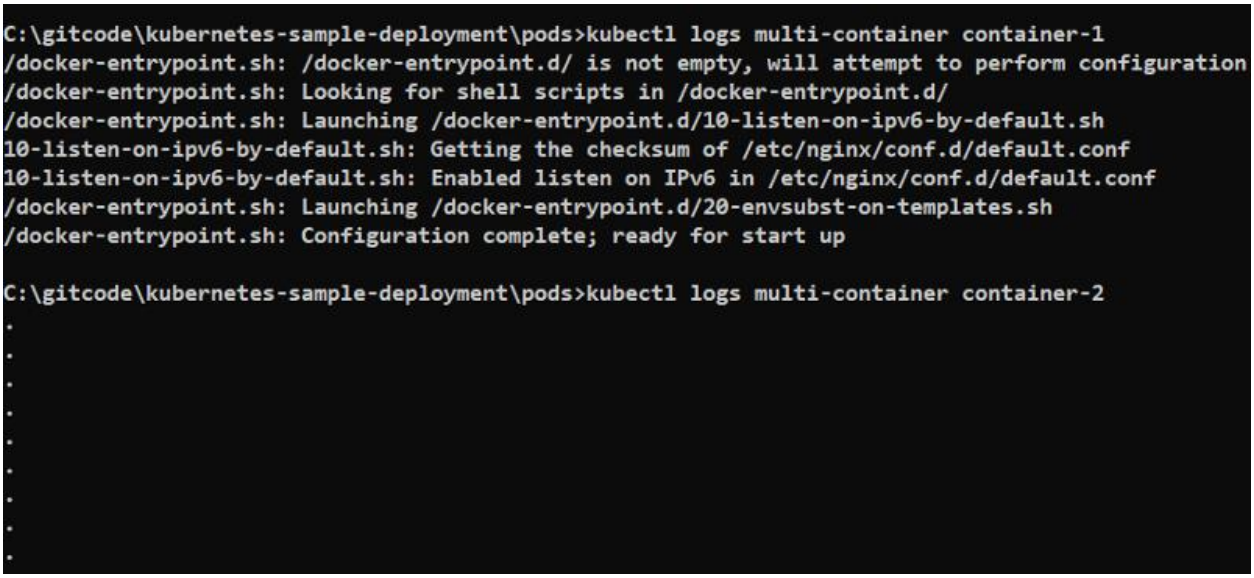
Create a Pod

```
kubectl create -f pod-multi-container.yaml
kubectl describe pod multi-container
```



We can specify the container name to get the logs for a particular container running in a pod, as shown here:

```
kubectl logs multi-container container-2
kubectl logs multi-container container-1
```



Pod Lifecycle

Pods has different States as described below:

- **Pending**: This means that the pod has been submitted to the cluster, but the controller hasn't created all its containers yet. It may be downloading images or waiting for the pod to be scheduled on one of the cluster nodes.
- **Running**: This state means that the pod has been assigned to one of the cluster nodes and at least one of the containers is either running or is in the process of starting up.
- **Succeeded**: This state means that the pod has run, and all of its containers have been terminated with success.
- **Failed**: This state means the pod has run and at least one of the containers has terminated with a non-zero exit code, that is, it has failed to execute its commands.
- **Unknown**: This means that the state of the pod could not be found. This may be because of the inability of the controller to connect with the node that the pod was assigned to.