1. Use label selectors to schedule Pods

Assign Pods using nodeSelector

- Labels the master as VIP hardware with status=vip, labels the worker1 as other hardwares with status=other and verify the settings
- Verify there are no deployments running, outside of the kube-system namespace. If there are, delete them. Then get a count of how many containers are running on both the master and secondary nodes.
- Create a pod name **vip** using vip.yaml to spawn four busybox containers which sleep the whole time. Include the nodeSelector entry to select the vip hardware.
- Deploy the new pod. Verify the containers have been created on the master node.

show

```
kubectl label node master status=vip
kubectl label node worker1 status=other
kubectl get nodes --show-labels

kubectl get deployments --all-namespaces
# On the master node
sudo docker ps | wc -l
# On the worker1 node
sudo docker ps | wc -l

kubectl run vip --generator=run-pod/v1 --image=busybox --dry-run -o yaml > vip.yaml
vi vip.yaml

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: vip
  name: vip
spec:
  containers:
  - image: busybox
    name: vip1
    command:
    - sleep
    - "1000000"
  - image: busybox
    name: vip2
    command:
    - sleep
    - "1000000"
  - image: busybox
    name: vip3
    command:
    - sleep
    - "1000000"
  - image: busybox
    name: vip4
    command:
    - sleep
    - "1000000"
  nodeSelector:
    status: vip

kubectl create -f vip.yaml
# On the master node
sudo docker ps | wc -l
# On the worker1 node
sudo docker ps | wc -l
```

Using Taints to control pod deployment

There are three taints, NoSchedule, PreferNoSchedule and NoExecute. The taints having to do with schedules will be used to determine newly deployed containers, but will not affect running containers. The use of NoExecute will cause running containers to move.

- Get a count of how many containers are running on both the master and secondary nodes.
- Create a deployment with name: taint-deployment which will deploy eight nginx containers using a YAML file (taint.yaml), determine where the containers are running.
- Delete the deployment and taint the secondary node (key 'lkd' and value 'value' and effect 'PreferNoSchedule'.), verify it has the taint then create the deployment again, locate where the containers are running.
- Delete the deployment, remove the taint, verify it has been removed, use the NoSchedule taint, then create the deployment again, locate where the containers are running.
- Delete the deployment, remove the taint, verify it has been removed, use the NoExecute taint, then create the deployment again, locate where the containers are running.
- Delete the deployment.

show

```
# On both master and worker nodes.
sudo docker ps | wc -l
```

Using kubectl with dry-run option to generate taint.yaml file, edit the yaml file and change the replicas to 8.

```
kubectl create deployment taint-deployment --image=nginx --dry-run -o yaml > taint.yaml
vi taint.yaml
kubectl create -f taint.yaml
# On both master and worker nodes.
sudo docker ps | wc -l

kubectl delete deployment taint-deployment
kubectl taint node worker1 lkd=value:PreferNoSchedule
kubectl describe node | grep Taint
kubectl create -f taint.yaml
# On both master and worker nodes.
sudo docker ps | wc -l

kubectl delete deployment taint-deployment
kubectl taint node worker1 lkd=value:NoSchedule
kubectl describe node | grep Taint
kubectl create -f taint.yaml
# On both master and worker nodes.
sudo docker ps | wc -l

kubectl delete deployment taint-deployment
kubectl taint node worker1 lkd=value:NoExecute
kubectl describe node | grep Taint
kubectl create -f taint.yaml
# On both master and worker nodes.
sudo docker ps | wc -l

kubectl delete deployment taint-deployment
```

We have deployed a number of PODs. They are labelled with 'tier', 'env' and 'bu'.

- How many PODs exist in the 'dev' environment?
- How many PODs are in the 'finance' business unit ('bu')?
- How many PODs and other objects including ReplicaSets, etc. are in the 'prod' environment?
- Identify the POD which is 'prod', part of 'finance' BU and is a 'frontend' tier?

show

```
kubectl get pod --selector=env=dev

kubectl get pod --selector=bu=finance

kubectl get all --selector=env=prod

kubectl get pod --selector=env=prod,bu=finance,tier=frontend
```

2. Understand the role of DaemonSets

Here.

3. Understand how resource limits can affect Pod scheduling

Create the deployment hog with the image vish/stree

show

```
kubectl create deployment hog --image vish/stress
```

Save the deployment to yaml file

- Add the memory limits of 4Gi and memory requests of 2500Mi

show

```
kubectl get deployment hog -o yaml --export > hog.yaml
vi hog.yaml

        imagePullPolicy: Always
        name: stress
        resources:
          limits:
            memory: "4Gi"
          requests:
            memory: "2500Mi"


kubectl replace -f hog.yaml
```

Edit the hog configuration file

- Limits of 1 cpu, 4Gi memory with requests of 0.5 cpu, 500Mi memory
- Add arguments for stress to consume CPU and memory (-cpus 2 -mem-total 900Mi -mem-alloc-size 100Mi -mem-alloc-sleep 1s).

show

```
kubectl get deployment hog -o yaml --export > hog.yaml
vi hog.yaml

        imagePullPolicy: Always
        name: stress
        resources:
          limits:
            cpu: "1"
            memory: "4Gi"
          requests:
            cpu: "0.5"
            memory: "2500Mi"
        args:
        - -cpus
        - "2"
        - -mem-total
        - "900Mi"
        - -mem-alloc-size
        - "100Mi"
        - -mem-alloc-sleep
        - "1s"

kubectl delete deployment hog
kubectl create -f hog.yaml
```

When a pod tries to exceed the resource limits

- In case of CPU, K8s throttles the CPU so that it does not go beyond the specified limit.
- In case of memory, a container can use more memory resources than its limit. If the pod tries to consume more memory than its limit constantly, the Pod will be terminated .

Using kubectl top pod, you can see that the hog pod consumes around 987m of CPU (~ 1vCPU = 1000m) even when the container tries to use 2 units of CPU with the command -cpus 2, and 929Mi of memory.

```
kubectl top pod
```

Edit the hog configuration file

- Limits of 1 cpu, 500Mi memory with the same args above (pod tries to consumes more memory than limit)

show

The pod will be terminated. It tries to restarts and will be terminated again and again.

4. Understand how to run multiple schedulers and how to configure Pods to use them

Deploy an additional scheduler to the cluster following the given specification

- Namespace: kube-system
- Name: my-scheduler

show

Use the manifest file used by kubeadm tool. Use a different port than the one used by the current one. Copy the kube-scheduler.yaml file from /etc/kubernetes/manifests/ folder.

```
sudo cp /etc/kubernetes/manifests/kube-scheduler.yaml my-custom-scheduler.yaml
```

Edit the my-custom-scheduler.yaml file, change leader-elect to false, then add --scheduler-name=my-scheduler in the command options:

```
metadata:
  labels:
    component: kube-scheduler
    tier: control-plane
  name: my-scheduler
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-scheduler
    - --bind-address=127.0.0.1
    - --kubeconfig=/etc/kubernetes/scheduler.conf
    - --leader-elect=false
    - --scheduler-name=my-scheduler
    image: k8s.gcr.io/kube-scheduler:v1.14.4
    name: my-scheduler
```

Create a POD with nginx image with the new custom scheduler.

show

Use kubectl run with dry-run option to generate the nginx.yaml file, then edit the file:

```
vi nginx.yaml
```

```
kubectl run nginx --generator=run-pod/v1 --dry-run -o yaml > nginx.yaml
```

Adding the schedulerName: option in the spec:

```
spec:
  containers:
  - image: nginx
    name: nginx
  schedulerName: my-scheduler
```

5. Manually schedule a pod without a scheduler

Manually schedule a pod with the nginx image to the master node without a scheduler

show

Adding the nodeName field with the node you want to schedule the pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx

spec:
  containers:
  - image: nginx
    name: nginx
  nodeName: master
```

A quick note on editing PODs and Deployments

Edit a POD

Remember, you CANNOT edit specifications of an existing POD other than the below.

- spec.containers[*].image
- spec.initContainers[*].image
- spec.activeDeadlineSeconds
- spec.tolerations

For example you cannot edit the environment variables, service accounts, resource limits of a running pod. But if you really want to, you have 2 options:

1. Run the kubectl edit pod command. This will open the pod specification in an editor (vi editor). Then edit the required properties. When you try to save it, you will be denied. This is because you are attempting to edit a field on the pod that is not editable. A copy of the file with your changes is saved in a temporary location as shown above. You can then delete the existing pod, then create a new pod with your changes using the temporary file by running the commands:

   ```
   kubectl delete pod webapp
   kubectl create -f /tmp/kubectl-edit-ccvrq.yaml
   ```

2. The second option is to extract the pod definition in YAML format to a file using the command, then make the changes to the exported file using an editor (vi editor). Then delete the existing pod and create a new pod with the edited file.

   ```
   kubectl get pod webapp -o yaml > my-new-pod.yaml
   vi my-new-pod.yaml
   kubectl delete pod webapp
   kubectl create -f my-new-pod.yaml
   ```

Edit Deployments

With Deployments you can easily edit any field/property of the POD template. Since the pod template is a child of the deployment specification, with every change the deployment will automatically delete and create a new pod with the new changes. So if you are asked to edit a property of a POD part of a deployment you may do that simply by running the command

```
kubectl edit deployment my-deployment
```

Find and kill the static pod in a node

show

Show the kubelet service.

```
sudo systemctl cat kubelet.service
sudo systemctl status kubelet.service
```

find the --config option

```
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"

Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
```

Look in to the file /var/lib/kubelet/config.yaml, looking for staticPodPath

```
staticPodPath: /etc/just-to-mess-with-you
```

- Run the command ps -aux | grep kubelet and identify the config file - --config=/var/lib/kubelet/config.yaml. Then check in the config file for staticPdPath.

- Identify which node the static pod is created on, ssh to the node and delete the pod definition file. If you don't know the IP of the node, run the kubectl get nodes -o wide command and identify the IP. Then SSH to the node using that IP. For static pod manifest path look at the file /var/lib/kubelet/config.yaml on node01

6. Display scheduler events

show

```
kubectl get events
kubectl -n kube-system logs my-custom-scheduler
```