CKA Curriculum Part 1 - Scheduling

**

Use labelselectors to schedule Pods.**

Labels are key/value pairs that are applied to objects in Kubernetes, such as pods, nodes, services and much more. After we've defined and applied labels, we can leverage labels to help us identify and distinguish components, including (but not limited to):

- Application Environment
  - Environment=Prod, Environment=Test, Environment=Dev
- Tier
  - Tier=Web, Tier=App, Tier=DB
- Distinguish nodes, or a collection of nodes based on location
  - Location=GB, Location=GER
- Distinguish nodes based on features
  - CPU=AMD, CPU=Intel

The example below labels a pod as belonging in the Dev environment, App Tier

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
  labels: {
    "Environment" : "Dev",
    "Tier" : "Web"
  }
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
```

Validate with `kubectl get pods --show-labels`

```
NAME            READY   STATUS    RESTARTS   AGE   LABELS
busybox-sleep   1/1     Running   0          25s   Environment=Dev,Tier=Web
```

Labels can also be used to schedule specific pods on specific nodes based on labels.

As a example, a node can be labelled with having a AMD cpu with the following:

```
kubectl label node k8s-worker-03 CPU=AMD
```

Validate with `kubectl get nodes --show-labels`

```
k8s-worker-03   Ready    <none>   11d   v1.14.1   CPU=AMD
```

Specify in a pod manifest to run on a node with an appropriate label

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
  labels: {
    "Environment" : "Dev",
    "Tier" : "Web"
  }
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
```

```
  nodeSelector:
    CPU: AMD
```

To validate, we can check kubectl get pods -o wide. Note how all three pods are running on the same node, which is what we've labelled with CPU=AMD.

```
busybox-amd     1/1     Running   0          8s      10.244.1.56   k8s-worker-03
```

Further validation can be obtained by executing a describe pod [podname] and identifying the nodeselector value:

```
Node-Selectors:  CPU=AMD
```

**Understand the role of DaemonSets. **

A DaemonSet is a way of ensuring that a particular pod runs on all (or a subset) of nodes. A very common use case is a CNI plugin. This should not be confused with a Deployment, or (in my opinion) be considered a mechanism to scale an application. We define a Daemonset like so:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: nginx
spec:
 selector:
   matchLabels:
     name: nginx-ds
 template:
   metadata:
     labels:
       name: nginx-ds
   spec:
     containers:
     - name: nginx-ds
       image: nginx
```

This basic example will create a Daemonset based on the nginx image. Once executed we can check by executing the following:

```
kubectl get ds

NAME    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
nginx   2         2         2       2            2           <none>          33s
```

These are also listed as individual pods:

```
kubectl get pods -o wide | grep nginx

nginx-97hp4     1/1     Running   0          97s   10.244.1.57   k8s-worker-03
nginx-hmtzp     1/1     Running   0          97s   10.244.2.43   k8s-worker-04
```

The Daemonset has created two pods based on the manifest - this is because, in this environment, we have three nodes that facilitate workloads.

**Understand how resource limits can affect Pod scheduling. **

At a namespace level, we can define resource limits. This enables a restriction in resources, especially helpful in multi-tenancy environments and provides a mechanism to prevent pods from consuming more resources than necessary, which may have a detrimental effect on the environment as a whole.

We can define the following:

Default memory / CPU **requests & limits **for a namespace

Minimum and Maximum memory / CPU **constraints **for a namespace

Memory/CPU **Quotas **for a namespace

Default Requests and Limits:

If a container is created in a namespace with a default request/limit value and doesn't explicitly define these in the manifest, it inherits these values from the namespace

Note, if you define a container with a memory/CPU limit, but not a request, Kubernetes will define the limit the same as the request.

Minimum / Maximum Constraints:

If a pod does not meet the range in which the constraints are valued at, it will not be scheduled.

Quotas:

Control the *total* amount of CPU/memory that can be consumed in the *namespace* as a whole.

Example: Attempt to schedule a pod that request more memory than defined in the namespace

Create a namespace: kubectl create namespace tenant-mem-limited \

Create a YAML manifest to limit resources:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: tenant-max-mem
spec:
  limits:
  - max:
      memory: 250Mi
    type: Container
```

Apply this to the aforementioned namespace: `kubectl apply -f maxmem.yaml --namespace tenant-mem-limited`

To create a pod with a memory request that exceeds the limit:

```
apiVersion: v1
kind: Pod
metadata:
  name: too-much-memory
  namespace: tenant-mem-limited
spec:
  containers:
  - name: too-much-mem
    image: nginx
    resources:
      requests:
        memory: "300Mi"
```

Executing the above will yield the following result:

```
The Pod "too-much-memory" is invalid: spec.containers[0].resources.requests: Invalid value: "300Mi": must be less than or equal to memory limit
```

As we have defined the pod limit of the namespace to 250MiB, a request for 300MiB will fail.

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

Kubernetes comes with its own scheduler which acts as the default for all related actions. You can, however, define and implement your own to exist in parallel to the default scheduler, should you wish.

This quite an advanced topic. It would be surprising if the exam asked the tester to create a new scheduler, but the process is defined https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/

As an example, a second custom scheduler has been implemented in a kubernetes cluster called "custom-scheduler". In a pod manifest we can specify a scheduler to use, if different from the default under "spec".

kind: Pod
metadata:
name: busybox-sleep
spec:

** schedulerName: custom-scheduler**
containers: \

- name: busybox
  image: busybox

  args: \
    - sleep \
    - "1000000"

Execute a kubectl describe pod busybox-sleep to validate which scheduler is being used:

```
File  Edit  View  Bookmarks  Settings  Help
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-24xbn:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-24xbn
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                                          Message
  ----    ------     ----  ----                                          -------
  Normal  Scheduled  22m   default-scheduler                             Successfully assigned default/busybox-sleep to
```

Note : to get familiar with the syntax, use “default-scheduler” in the manifest. This is the name for the scheduler that comes with Kubernetes

Manually schedule a pod without a scheduler

Pods that are created without a scheduler are otherwise known as _“static pods”_. Static pods are managed directly by kubelet daemon on a specific node, without the API server observing it. It does not have an associated replication controller, and kubelet daemon itself watches it and restarts it when it crashes. There is no health check. Static pods are always bound to one kubelet daemon and always run on the same node with it.

There are two ways to create a static pod:

1. From a directory
2. From a URL

Either method requires a modification to how the Kubelet functions on a given node. Because we’re not using a scheduler, the kublet daemon takes on the responsibility of creating the pods.

From a Directory or web address

First, we create a directory on a specific node

```
mkdir /etc/staticpods
```

Next we place a pod manifest file into this directory. Below is an example from a previous exercise:

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
```

Next we run the kubelet service to reference this directory : --pod-manifest-path=/etc/staticpods

Note, to identify the file that needs modifying:

```
sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
  Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)

[Service]
ExecStart=/usr/bin/kubelet \
 --pod-manifest-path=/etc/staticpods
```

Note you can also specify a url with the format --manifest-url=https://virtualthoughts.co.uk/somepod.yaml should you wish to load a static pod from a URL.

Display scheduler events

Viewing and assessing logs generated by schedulers provide a myriad of functions, including but not limited to - troubleshooting, performance monitoring, and various others. There are several ways to get scheduler events:

```
kubectl get events | grep scheduler
```

Note this will only work if you're using the default scheduler (default-scheduler) or a customer scheduler with "scheduler" in the name. Replace accordingly or don't pipe anything to grep to get a system wide set of events

Shortly after performing an action, such as deploying a pod, you should see something in the logs resembling the following:

Successfully assigned default/busybox-sleep to 632efcbf-8027-4711-b6f4-e0aea7db9ed3

To acquire logs from the scheduler itself, one of two methods must be used, depending on how the scheduler is deployed.

For schedulers running as a pod

Locate the pod facilitating the scheduler:

```
kubectl get pods --namespace kube-system
```

Inspect the logs accordingly:

```
kubectl logs [Name of pod] --namespace kube-system
```

For schedulers running locally on a node

Log onto the master node and cat / copy / open the following file:

```
/var/log/kube-scheduler.log
```

Know how to configure the Kubernetes Scheduler

Kube-Scheduler has a number of flags which can influence its behavior: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/

How the kubernetes cluster is constructed will influence how you modify kube-scheduler. But as an example for kubeadm installations a manifest file for kube-scheduler is located in `/etc/kubernetes/manifests` on the master node(s) as `kube-scheduler.yaml`. Add/modify the flags as appropriate:

```
apiVersion: v1
kind: Pod
```

```
metadata:
 creationTimestamp: null
 labels:
   component: kube-scheduler
   tier: control-plane
 name: kube-scheduler
 namespace: kube-system
spec:
 containers:
 - command:
   - kube-scheduler
   - --bind-address=127.0.0.1
   - --kubeconfig=/etc/kubernetes/scheduler.conf
   - --leader-elect=true
```