123

78. Multiple Schedulers

I feel quite Tough 109

⇒In this lecture, we look at deploying "multiple schedulers" in a K8S cluster

⇒We already know, how default scheduler works in K8S, it has an algorithm that distributes the pods across nodes evenly as well as takes into considerations various conditions we specify through "taints & tolerations" & "node affinity" etc.,

But what if none of these satisfies your needs

Say you've specific apps that require its components to be placed on nodes after performing some additional checks?

So you decide to have your own scheduling algorithm to place pods on nodes, so that you can add your own custom conditions & checks in it. K8S is highly extensible, we can write our own K8S scheduler program, package it & deploy it as the "default scheduler" or as an "additional scheduler" in the K8S cluster, that way all of the other apps can go through default scheduler

⇒ However, some specific apps that you may choose can use your own custom scheduler

⇒So, a K8S cluster can have "multiple schedulers" at a time

⇒When creating a "pod" or "deployment", we can instruct K8S to have the pod scheduled by a "specific scheduler"

Let see how to do this.

There are different Schedulers & have different names, so that we can identify them as separate schedulers, the "default scheduler" is named as "default scheduler" & this name is configured in a "kube-scheduler configuration file" that look like this

Scheduler-config.yaml

```
apiVersion: KubeScheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: default-scheduler
```

Now, the default scheduler doesn't really need one because if you don't specify a name, it sets the name to a "default scheduler"

⇒ And for the other schedulers, we could create a separate configuration file & set the "scheduler name" like this.

myscheduler-config.yaml

```
apiVersion: KubeScheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: my-scheduler
```

Deploy Additional Scheduler

→ Earlier we saw how to deploy the "kube-scheduler" cluster runs on systemd to run scheduler

1) we download the kube-scheduler binary & run it as a service with a set of options

wget https://storage.googleapis.com/kubernetes-release/release/v1.12.0/bin/linux/amd64/kube-scheduler

→ Now to deploy the additional scheduler, you may use the same "kube-scheduler" binary (or) use one that you might have built for yourself, which is what you would do if you needed the scheduler to work differently

Turn 2 pages

→ In this case, we'll use the same binary to deploy the "additional scheduler". This time, we point the configuration to custom configuration file that we created, so each scheduler uses a separate configuration file with each file having its own scheduler name

kube-scheduler.service

```
ExecStart = /usr/local/bin/kube-scheduler \
  --config = /etc/kubernetes/config/kube-scheduler.yaml
```

my-scheduler-2.service

```
ExecStart = /usr/local/bin/kube-scheduler \
  --config= /etc/kubernetes/config/my-scheduler-2-config.yaml
```

→ And note these are other options to be passed in, such as the kubeconfig file to authenticate into kube API but we're skipping this now to make it simple.

This is not how we deploy a custom scheduler because with "kubeadm" deployment all the control plane components run as a pod or a deployment within the K8S cluster. We won't follow this systemd binary service approach method 99% of time.

Deploy Scheduler as a pod

Deploy additional Scheduler as pod

⇒ We create a "pod-definition file" & specify the "kube config property" which is the path to the "scheduler config file" that has the authentication information to connect to K8S API server.

⇒ we then pass in our custom "kube-scheduler" configuration file as a config option to the scheduler.

Note: kube config = path to scheduler config file [It has the authentication to connect to apiserver
config = path to our custom scheduler configuration file

We've the scheduler name specified in the file, so that's how the name gets picked up by the scheduler

my-custom-scheduler.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-custom-scheduler
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-scheduler
    - --address=127.0.0.1
    - --kubeconfig=/etc/kubernetes/scheduler.conf
    - --config=/etc/kubernetes/my-scheduler-config.yaml
    image: k8s.gcr.io/kube-scheduler-amd64:v1.11.3
    name: kube-scheduler
```

"kubeconfig" property is a "configuration file" that tells scheduler how to connect to K8S API server

"config" property refers to scheduler's configuration file

→ * my-scheduler-config.yaml

apiVersion: kubeschedular.config.k8s.io/v1

kind: KubeSchedularConfiguration

profiles:

- schedularName: my-scheduler

leaderElection:

leaderElect: true

resourceNamespace: kube-system

resourceName: lock-object-my-scheduler

⇒ Now, another important option to look here is the "leaderElect" option, & this goes into the "kube-schedular" configuration.

⇒ The "leaderElect" option is used, when you have multiple copies of the schedular running on different master nodes as an High-availability setup, where you have multiple master nodes with the k8s schedular process running on both of them.

⇒ If multiple copies of the same schedular are running on different nodes, only one can be active at a time & thats where the "leaderElect" option helps in choosing a leader who will lead the scheduling activities

Remember!-

If you have multiple masters, you can pass in this additional parameter to set a log object name & this is to differentiate the new custom schedular from the default election process

Deploying additional scheduler as a deployment

View Schedulers:-

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

now you can see the new custom scheduler running.

```
kube-scheduler-master      1/1   Running   0   1h
my-custom-scheduler        1/1   Running   0   9s
```

1:3)

Use Custom Scheduler

Once, we have deployed that custom scheduler, the next step is to configure a pod or a deployment to use this "new scheduler"

How do we use our custom scheduler

In the pod-definition file, add a new field called "scheduler name" & specify the name of the new scheduler", This way when the pod is created, the right scheduler gets picked up & the scheduling process works

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  schedulerName: my-custom-scheduler
```

Now create the pod

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

⇒ If the scheduler was not configured correctly, then the pod will continue to remain in pending state, if everything is good, then the pod will be in running state

⇒ If the pod is in pending state, then we can take the log under the pod describe command

⇒ kubectl describe command & you'll mostly notice that the scheduler isn't configured correctly.

View Events :-

How do you know which scheduler picked up & we've multiple schedulers, we can view this in the events using the

```
kubectl get events -o wide
```

⇒ This will list all the events in the current namespace look for "scheduled" events & you can see the source of the events is the custom scheduler that we created & thats the name that we give to the custom scheduler & the message - successfully assigned default/nginx to node01

LAST SEEN	COUNT	NAME	KIND	TYPE	REASON	SOURCE	MESSAGE
9s	1	nginx.15	Pod	Normal	Scheduled	my-custom-scheduler	Successfully assigned default/nginx to node01

View Scheduler Logs

In case if you run into issues, for that, view the logs using the kubectl logs command $

provide the scheduler name either the "pod name" or "deployment name" & then "right namespace"

```
kubectl logs my-custom-scheduler --namespace=kube-system
```

optional: Create the Config file

/etc/kubernetes/config/kubeschedulerconfiguration.yaml doesn't exists; create

```
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
clientConnection:
    kubeconfig: /etc/kubernetes/scheduler.conf
leaderElection:
    leaderElect: false
SchedulerName: my-scheduler
```

How to access & modify the kube-scheduler.service, if your cluster uses systemd to run the scheduler instead of static pods [which is rare in kubeadm clusters but can occur in custom setup or learning environment.

Step by step : Modifying kube-scheduler.service
This applies only when your cluster is running kube-scheduler as systemd service, not as a static pod in /etc/kubernetes/manifest

step1: Locate the kube-scheduler service
check if its managed by systemd
systemctl status kube-scheduler
you will see output like
• kube-scheduler.service - kubernetes scheduler
Loaded: loaded (/etc/systemd/system/kube-scheduler.service; enabled)
Active: active (running)

step2:- Edit the systemd unit
sudo systemctl edit kube-scheduler
This opens an override file. you'll usually see in:

[Service]
ExecStart=
ExecStart=/usr/local/bin/kube-scheduler --config=/etc/kubernetes/config/kubescheduler.yaml

The first ExecStart= line clears the default so the next one overrides it.
[Make sure the config file path & binary match your environment]

step3:- Reload and Restart
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl restart kube-scheduler

confirm its running:-
systemctl status kube-scheduler