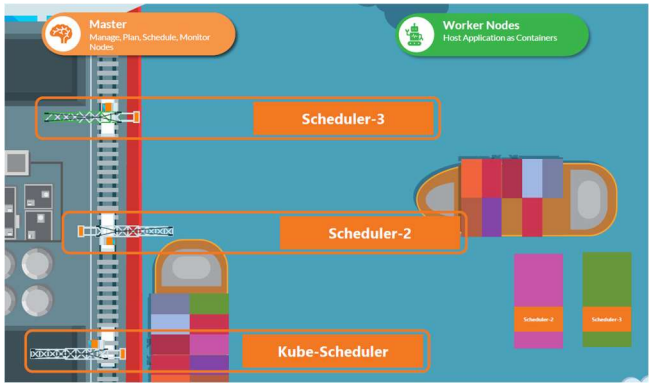
Multiple Scheduler

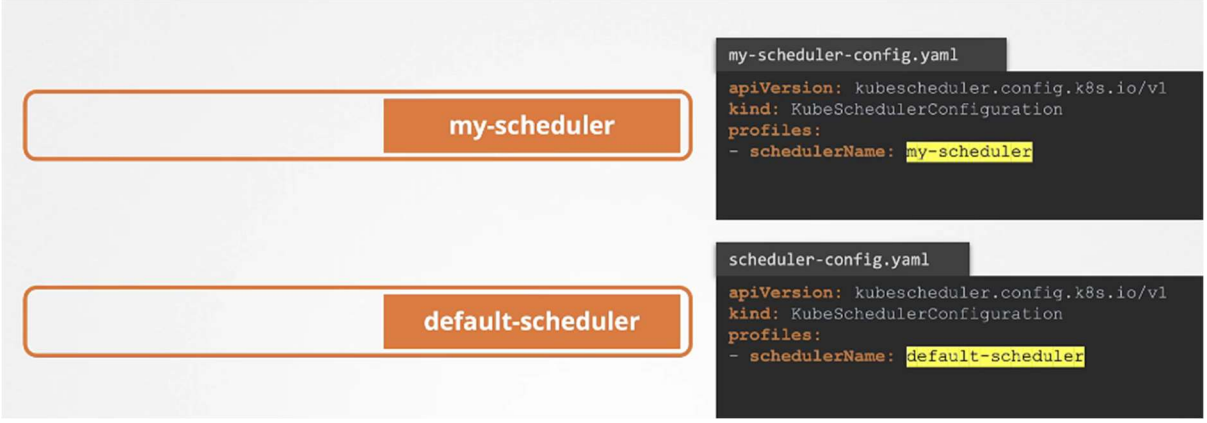
Now, we have seen how the default scheduler works in Kubernetes in the previous lectures. It has an algorithm that distributes pods across nodes evenly, as well as takes into consideration various conditions we specify through taints and tolerations and node affinity, et cetera.

But what if none of these satisfies your needs? Say you have a specific application that requires 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 and checks in it.

Kubernetes is highly extensible. You can write your own Kubernetes scheduler program, package it, and deploy it as the default scheduler or as an additional scheduler in the Kubernetes cluster. That way, all of the other applications can go through the default scheduler. However, some specific applications that you may choose can use your own custom scheduler. So your Kubernetes cluster can have multiple schedulers at a time. When creating a pod or a deployment, you can instruct Kubernetes to have the pod scheduled by a specific scheduler. So let's see how that's done.



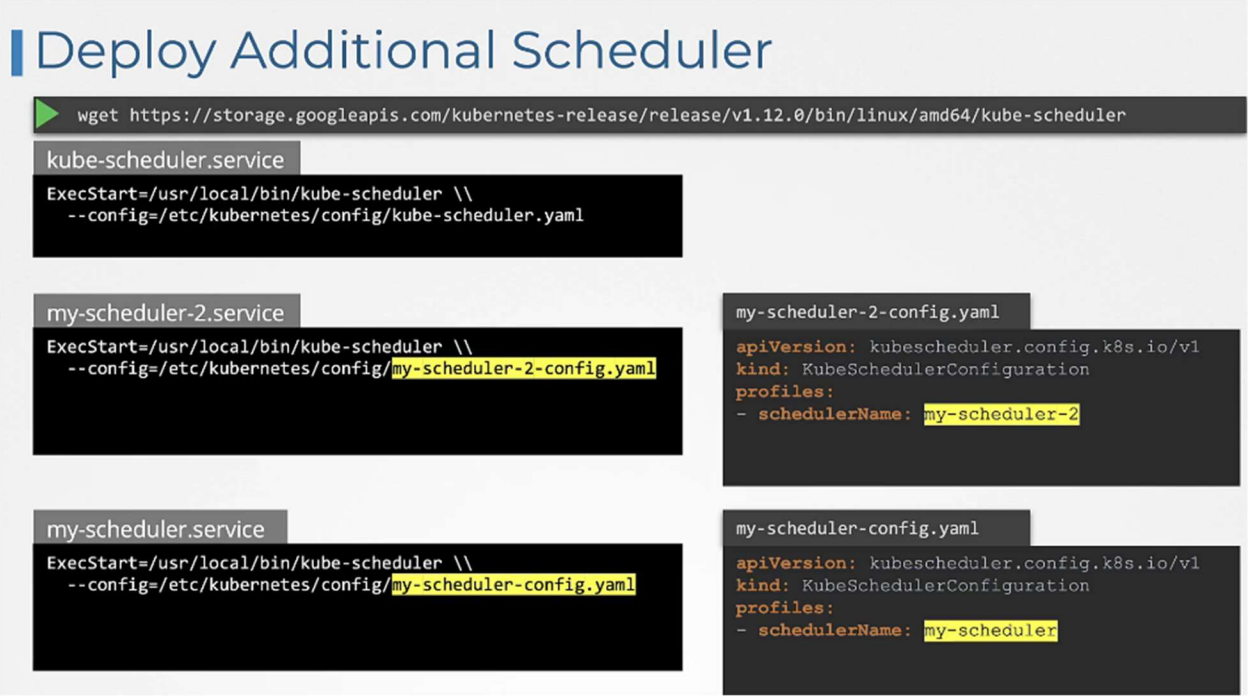
Now, when there are multiple schedulers, they must have different names so that we can identify them as separate schedulers. So the default scheduler is named default scheduler. And this name is configured in a kube-scheduler configuration file that looks like below. Now, the default scheduler doesn't really need one because if you don't specify a name, it sets the name to a default scheduler. But this is how it would look if you were to create one. And for the other schedulers, we could create a separate configuration file and set the scheduler name like below.



So let's start with the most simple way of deploying an additional scheduler. Now, we earlier saw how to deploy the Kubernetes kube-scheduler. We download the kube-scheduler binary and run it as a service with a set of options.

Now, to deploy an 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. In this case, we're going to use the same binary to deploy the additional scheduler. And this time, we point the configuration to the custom configuration file that we created.

Each scheduler uses a separate configuration file, with each file having its own scheduler name. Note that there are other options to be passed in, such as the kubeconfig file to authenticate into the Kubernetes API, but I'm just skipping that for now to keep it super simple.



This is not how you would deploy a custom scheduler 99% of the time today because with kubeadm deployment, all the control plane components run as a pod or a deployment within the Kubernetes cluster.

Let's explore another way to deploy the scheduler, specifically as a pod. We create a pod definition file and specify the kubeconfig property, which is the path to the scheduler config file containing the authentication information to connect to the Kubernetes API server. We also pass our custom kube-scheduler configuration file as a config option to the scheduler. It's important to have the scheduler name specified in the file so that it gets picked up by the scheduler.

Another crucial option to note is the leader-elect option, which goes into the kube-scheduler configuration. The leader-elect option is used when you have multiple copies of the scheduler running on different master nodes in a high-availability setup. This setup involves multiple master nodes, each with the Kubernetes scheduler process running. When multiple copies of the same scheduler are running on different nodes, only one can be active at a time. The leader-elect option helps in choosing a leader who

will lead the scheduling activities. We will discuss high-availability setups in another section. If you do have multiple masters, remember that you can pass in the additional parameter to set a log object name. This is to differentiate the new custom scheduler from the default election process.

Deploy Additional Scheduler as a Pod

my-custom-scheduler.yaml

```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
```

my-scheduler-config.yaml

```yaml
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
- schedulerName: my-scheduler
leaderElection:
  leaderElect: true
  resourceNamespace: kube-system
  resourceName: lock-object-my-scheduler
```

Now, let's take a look at how to deploy the additional scheduler as a deployment, and for this, I'm gonna go into the Kubernetes documentation pages and for the one for configuring multiple schedulers.

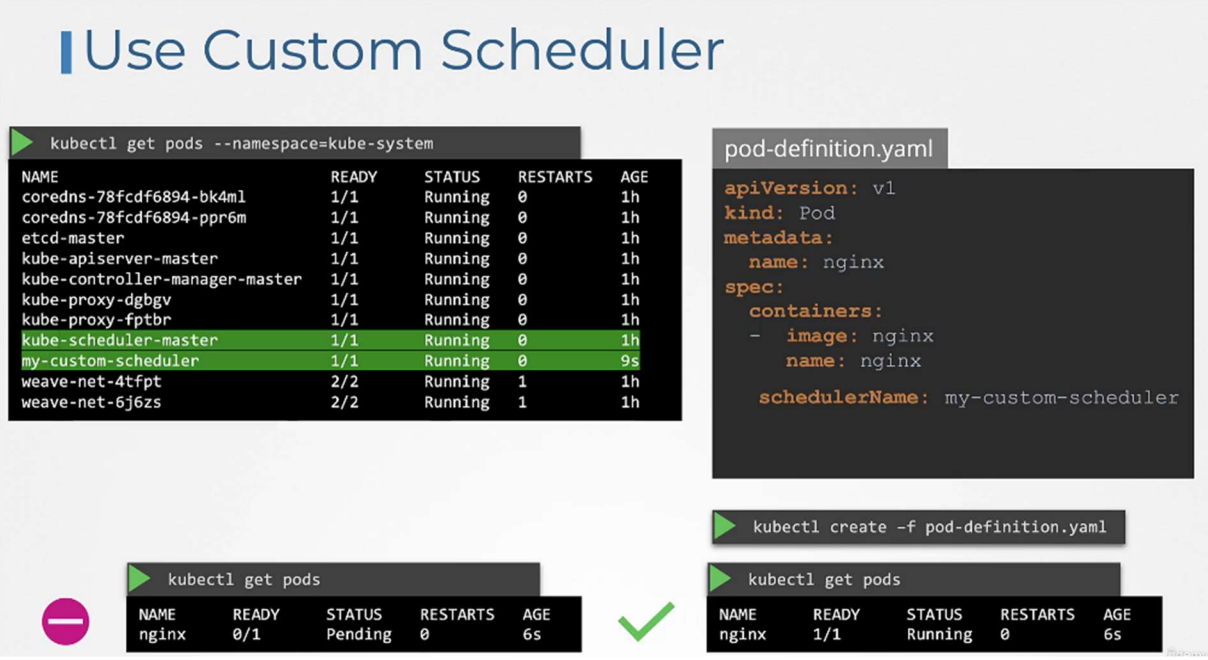
https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/

So just proceeding with our lecture. So when you run the get pods command in the kube-system namespace, you can then see the new custom scheduler running. So this is if you ran it as a pod, and if you run as a deployment, then you'll probably see a slightly different naming convention but you'll be able to see the pod there. Just make sure you're checking the right namespace.

Now, once we have deployed that custom scheduler, the next step is to configure a pod or a deployment to use this new scheduler. So how do you use our custom scheduler?

Here we have a pod definition file, and what we need to do is add a new field called schedulerName and specify the name of the new scheduler, and that's basically it. This way when the pod is created, the right scheduler gets picked up, and the scheduling process works.

Now, we can create the pod using the kubectl create command. If the scheduler was not configured correctly, then the pod will continue to remain in a pending state. If everything is good, then the pod will be in a running state. So if the pod is in a pending state, then you can look at the logs under the kubectl describe command, and you'll mostly notice that the scheduler isn't configured correctly.



How do you know which scheduler picked up scheduling a particular pod?

Now, we can view this in the events using the kubectl get events command with the -o wide option. And this will list all the events in the current namespace and look for the scheduled events and as you can see, the source of the event is the custom scheduler that we created. That's the name that we gave to the custom scheduler. And the message says that successfully assigned the image. So that indicates that it's working.

```
kubectl get events
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
8s          1       nginx.15    Pod   Normal  Pulling     kubelet, node01       pulling image "nginx"
2s          1       nginx.15    Pod   Normal  Pulled      kubelet, node01       Successfully pulled image "nginx"
2s          1       nginx.15    Pod   Normal  Created     kubelet, node01       Created container
2s          1       nginx.15    Pod   Normal  Started     kubelet, node01       Started container
```

You could also view the logs of the scheduler in case you run into issues. So for that, view the logs using the kubectl logs command and provide the scheduler name, either the pod name or the deployment name, and then the right namespace.

```
kubectl logs my-custom-scheduler --name-space=kube-system
I0204 09:42:25.819338       1 server.go:126] Version: v1.11.3
W0204 09:42:25.822720       1 authorization.go:47] Authorization is disabled
W0204 09:42:25.822745       1 authentication.go:55] Authentication is disabled
I0204 09:42:25.822801       1 insecure_serving.go:47] Serving healthz insecurely on 127.0.0.1:10251
I0204 09:45:14.725407       1 controller_utils.go:1025] Waiting for caches to sync for scheduler controller
I0204 09:45:14.825634       1 controller_utils.go:1032] Caches are synced for scheduler controller
I0204 09:45:14.825814       1 leaderelection.go:185] attempting to acquire leader lease  kube-system/my-custom-scheduler...
I0204 09:45:14.834953       1 leaderelection.go:194] successfully acquired lease kube-system/my-custom-scheduler
```