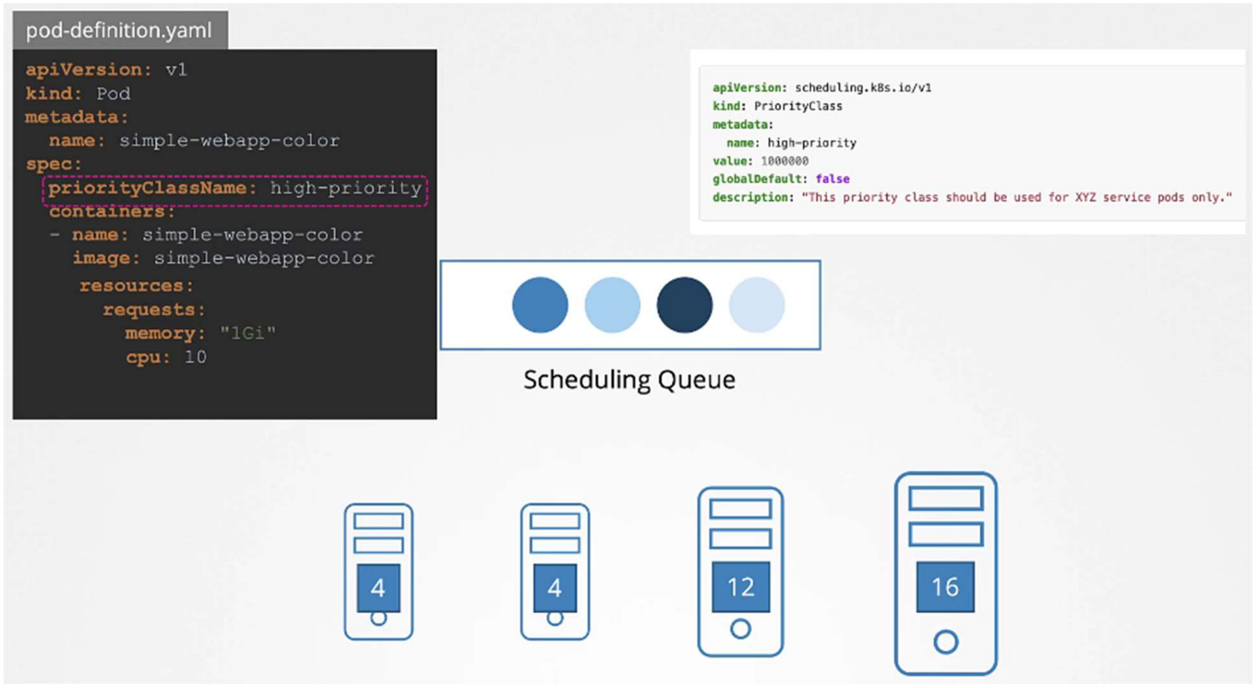
Multiple Scheduler

NLet us now look at what scheduler profiles are. So let's first recap how the Kubernetes scheduler works using this simple example of scheduling a pod to one of these four nodes that you can see here that are part of the Kubernetes cluster.

So here, we have our pod definition file and there's our pod. It is waiting to be scheduled on one of these four nodes. Now, it has a resource requirement of 10 CPU so it's only gonna be schedule on a node that has 10 CPU remaining. And you can see the available CPU on all of these nodes that are listed here. Now, it is not alone, there are some other pods that are waiting to be scheduled as well.

So the first thing that happens is that when these pods are created, the pods end up in a scheduling queue. So this is where the pods wait to be scheduled. So at this stage, pods are sorted based on the priority defined on the pods. So in this case, our pod has a high priority set. So, to set a priority, you must first create a priority class that looks like below and you should set it a name and set it a priority value. In this case, it's set to 1 million so that's really high priority.

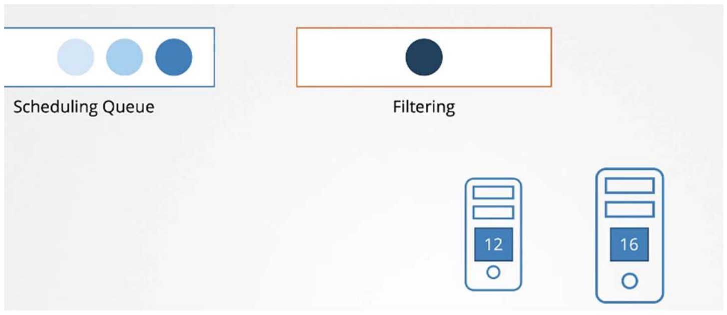


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
spec:
  priorityClassName: high-priority
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
     resources:
       requests:
         memory: "1Gi"
         cpu: 10
```

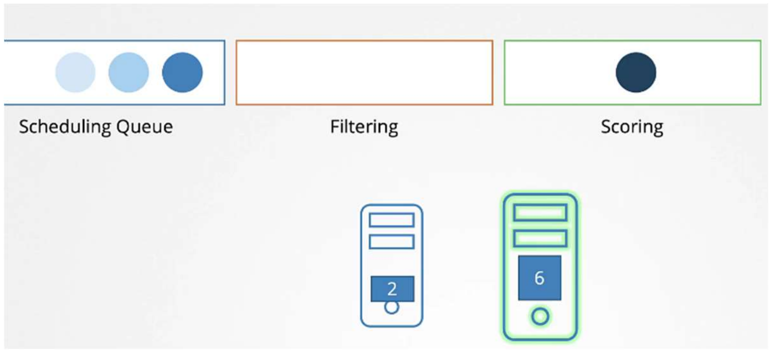
```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

So, this is how pods with higher priority gets to the beginning of the queue to be scheduled first. And so that sorting happens in this scheduling phase.

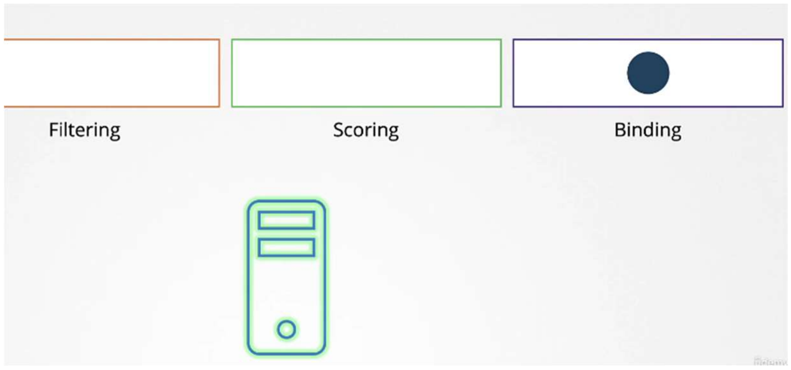
Then, our pod enters the filter phase. This is where nodes that cannot run the pod are filtered out. So, in our case, the first two nodes do not have sufficient resources so do not have 10 CPU remaining so they are filtered out.



The next phase is the scoring phase. So this is where nodes are scored with different weights. From the two remaining nodes, the scheduler associates a score to each node based on the free space that it will have after reserving the CPU required for that pod. So, in this case, the first one has two left and the second node will have six left. So, the second node gets a higher score. And so, that's the node that gets picked up.



And finally, in the binding phase,this is where the pod is finally bound to a node with the highest score.

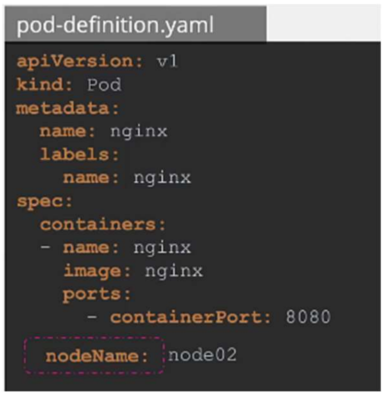


Now, all of these operations are achieved with certain plugins. For example, while in the scheduling queue, it's the **prioritySort** plugin that sorts the pods in an order based on the priority configured on the pods. This is how pods with priority class gets higher priority over the other pods when scheduling.

Filtering phase

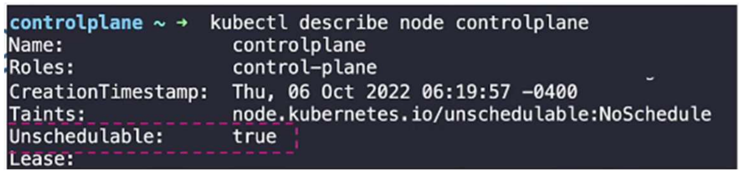
In the filtering stage, it's the **NodeResourcesFit** plugin that identifies the notes that has sufficient resources required by the pods and filters out the nodes that doesn't.

Now, some other plugin examples that come into this particular stage are the **NodeName** plugin that checks if a pod has a node name mentioned in the pods spec and filters out all the nodes that does not match this name.



```
pod-definition.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
      - containerPort: 8080
  nodeName: node02
```

Another example is the **NodeUnschedulable** plugin that filters out nodes that has the unschedulable flag set to true. So, this is when you're on the drain, the cordon command on a node, which we will discuss later. But all the nodes that has the unschedulable flags set are true, this particular plugin that makes sure that no pods are set on those nodes.



```
controlplane ~ ➜  kubectl describe node controlplane
Name:               controlplane
Roles:              control-plane
CreationTimestamp:  Thu, 06 Oct 2022 06:19:57 -0400
Taints:             node.kubernetes.io/unschedulable:NoSchedule
Unschedulable:      true
Lease:
```

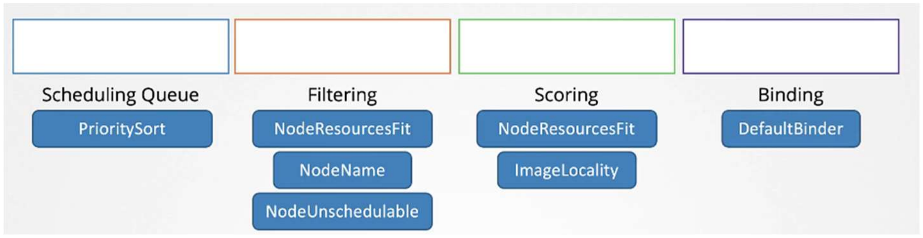
Scoring Phase

Now, in the scoring phase, again, the **NodeResourcesFit** plugin associates a score to each node based on the resource available on it after the pod is allocated to it. So, as you can see, a single plugin can be associated in multiple different phases.

Another example of a plugin in this stage would be the **ImageLocality** plugin that associates a high score to the nodes that already has the container image used by the pods among the different nodes. Now, note that at this phase, the plugins do not really reject the pod placement on a particular node. For example, in case of the image locality node, it ensures that pods are placed on a node that already has the image but if there are no nodes available, it will anyway place the pod on a node that does not even have the image. So, it's just a scoring that happens at this stage.

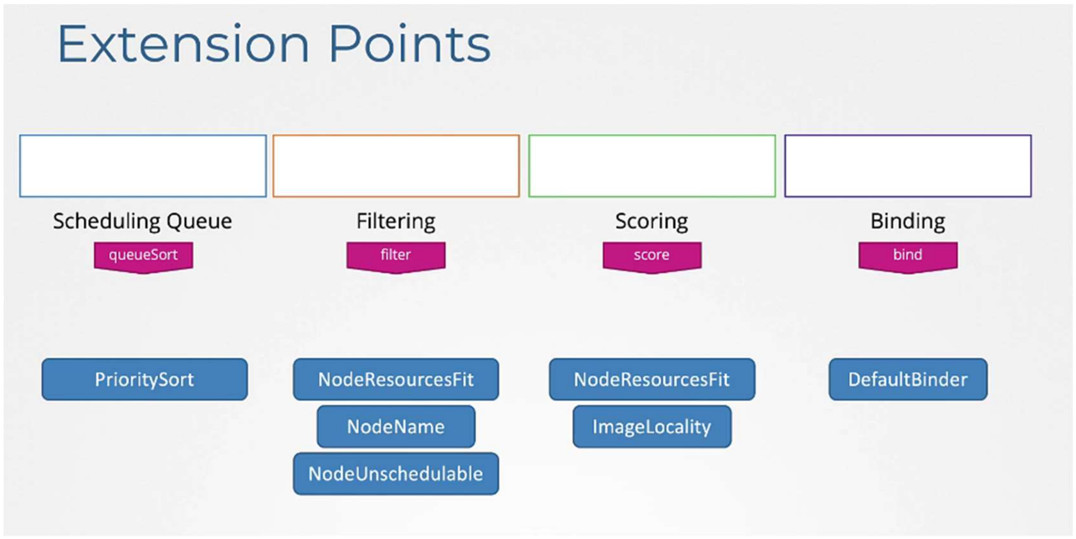
Binding Phase

And finally, in the binding phase, you have the **DefaultBinder** plugin that provides the binding mechanism.

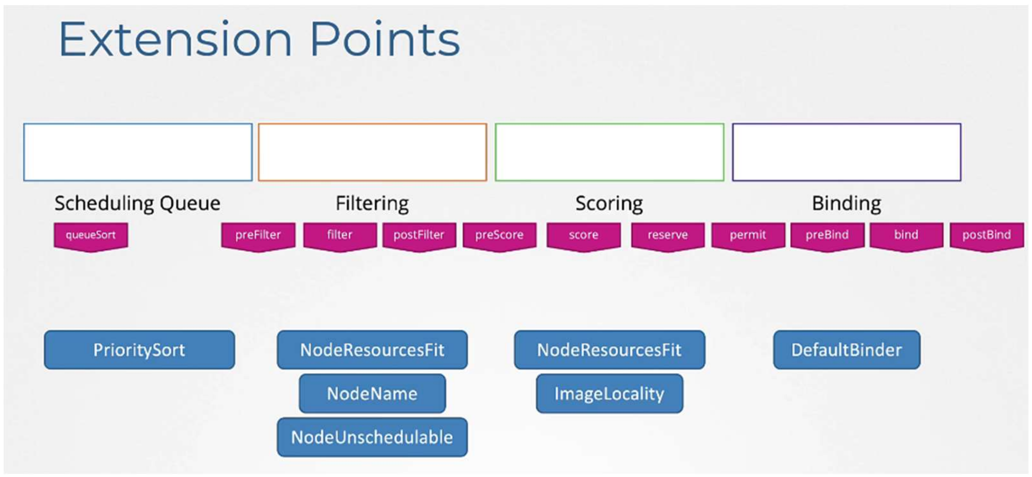


Now, the highly extensible nature of Kubernetes makes it possible for us to customize what plugins go where and for us to write our own plugin and plug them in here. And that is achieved with the help of what is called as extension points.

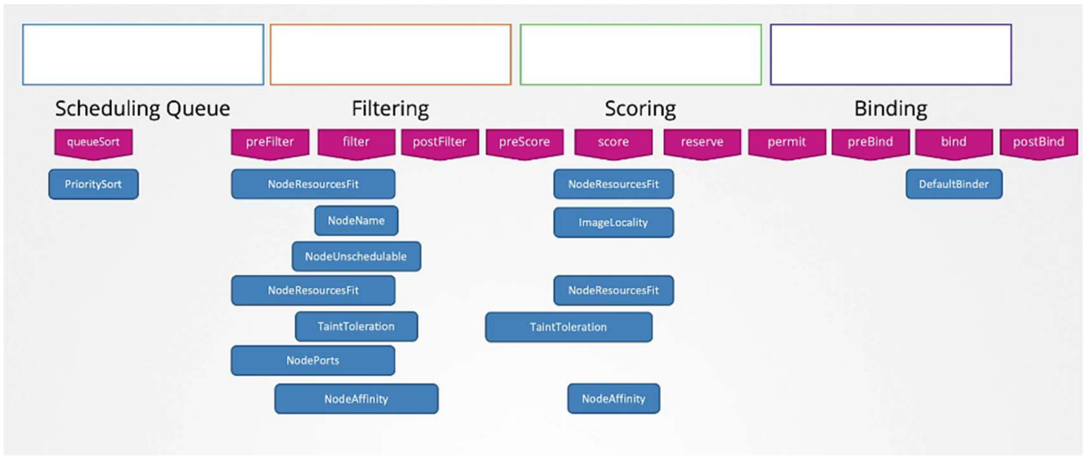
So, at each stage, there is an extension point to which a plugin can be plugged to. In the scheduling queue, we have a queue sort extension to which the priority sort plugin is plugged to. And then we have the filter extension, the score and the bind extension to which each of these plugins that we just talked about are plugged to



As a matter of fact, there's more. So, there are extensions before entering the filter phase called the **pre-filter extension** and after the filter phase called **post filter**. And then there are **pre-score** before the **score** extension point and **reserve** after the extension point, the score extension point. And then there's **permit** and **pre-bind** before the **bind** and **post bind** after the binding phase. So, there are so many options available. Basically you can get a custom code of your own to run anywhere in these points by just creating a plugin and plugging it into the respective kind of point that you want to plug it to.



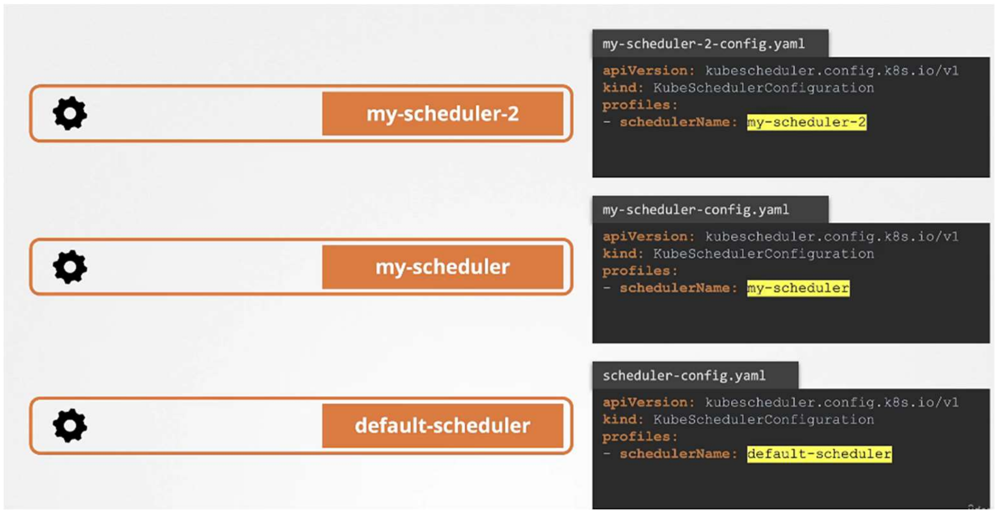
And here is a little bit more details on some additional plugins that come by default that are associated with the different extension points. As you can see, some of the plugins span across multiple extension points. Some of them are just within a specific extension point.



So, that's what scheduling plugins and extension points are. So the highly extensible nature of Kubernetes allows us to customize the way that these plugins are called and write our own scheduling plugin if needed.

So having learned that, let's look at how we can change the default behavior of how these plugins are called and how we can get our own plugins in there if it's really needed.

So, taking a step back, earlier we talked about deploying three separate schedulers each with a separate scheduler binary. So we have the the default scheduler and then the my scheduler and then the my scheduler 2. Now, all of these are three separate scheduler binaries that are run with a separate scheduler config file associated with each of them.



my-scheduler-2-config.yaml
```yaml
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
- schedulerName: my-scheduler-2
```

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

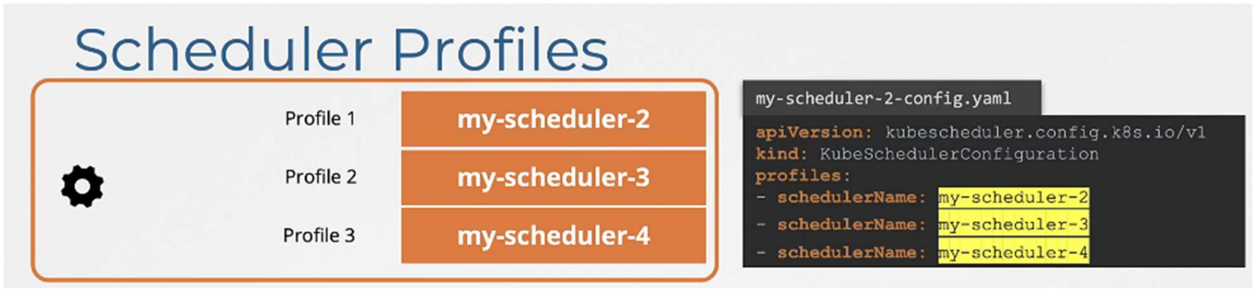
scheduler-config.yaml
```yaml
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
- schedulerName: default-scheduler
```

Now, that's one way to deploy multiple schedulers. Now the problem here is that since these are separate processes there is an additional effort required to maintain these separate processes and also, more importantly, since they are separate processes, they may run into race conditions while making scheduling decisions. For example, one scheduler may schedule a workload on a node without knowing that there's another scheduler scheduling a workload on that same node at the same time.

So, with the 1.18 release of Kubernetes, a feature to support multiple profiles in a single scheduler was introduced.

So now you can configure multiple profiles within a single scheduler in the configuration file by adding more entries to the list of profiles and for each profile, specify a separate scheduler name.

So this creates a separate profile for each scheduler which acts as a separate scheduler itself, except that now multiple schedulers are run in the same binary as opposed to creating separate binaries for each scheduler.

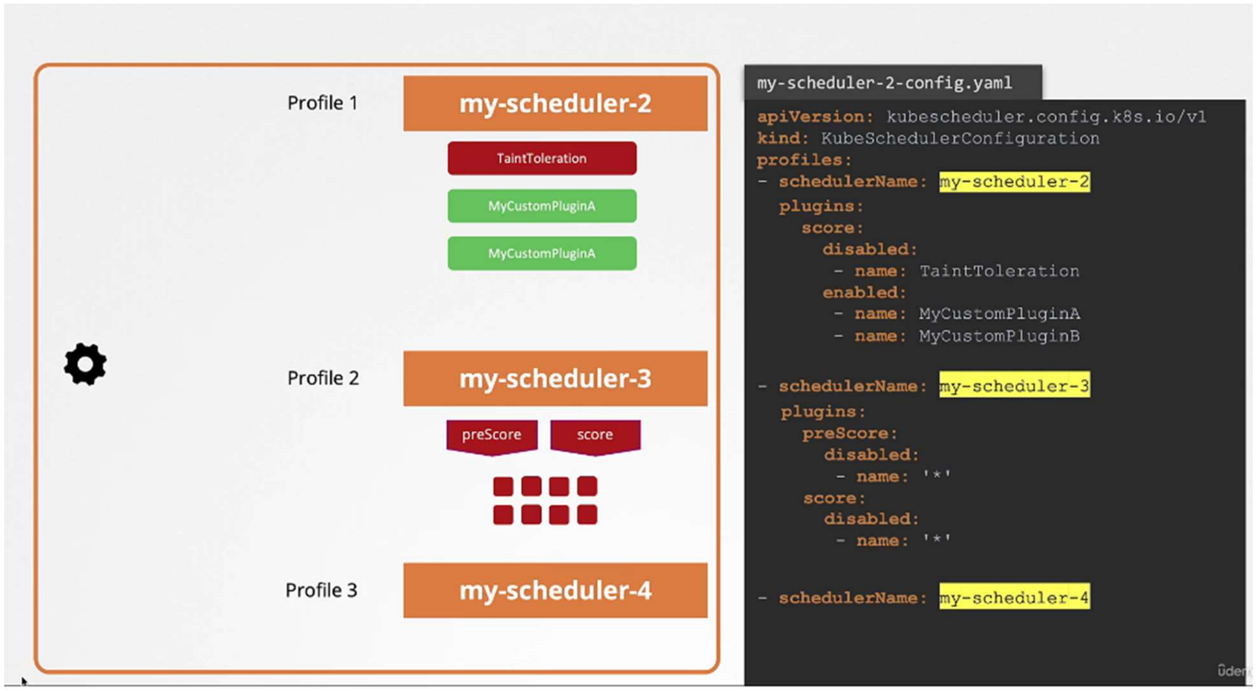


Scheduler Profiles

```yaml
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
- schedulerName: my-scheduler-2
- schedulerName: my-scheduler-3
- schedulerName: my-scheduler-4
```

Now, how do you configure these different scheduler profiles to work differently? Because right now, all of them just simply have different names. So they're gonna work just like the default scheduler. How do you configure them to work differently?

Under each scheduler profile, we can configure the plugins the way we want to. For example, for the my scheduler 2 profile, I'm going to disable certain plugins like detained and toleration plugin and enable my own custom plugins. For the my scheduler 3 profile, I'm going to disable all the pre score and score plugins. So this is how that's going to look under the plug-in section, specify the extension point and enable or disable the plug-ins by name or a pattern as as shown in this case.



```yaml
apiVersion: kubescheduler.config.k8s.io/v1
kind: KubeSchedulerConfiguration
profiles:
- schedulerName: my-scheduler-2
  plugins:
    score:
      disabled:
       - name: TaintToleration
      enabled:
       - name: MyCustomPluginA
       - name: MyCustomPluginB

- schedulerName: my-scheduler-3
  plugins:
    preScore:
      disabled:
       - name: '*'
    score:
      disabled:
       - name: '*'

- schedulerName: my-scheduler-4
```