Multiple Schedulers

Learn how multiple Kubernetes schedulers work together.

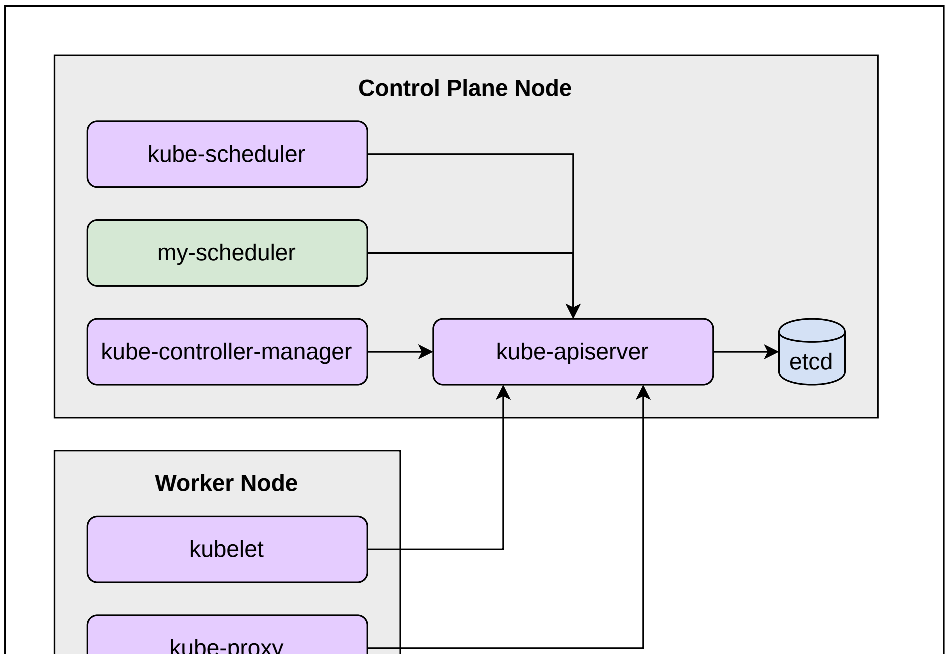
We'll cover the following

Multiple schedulers

In Kubernetes, a scheduler works similar to an operator or controller. It watches all the Pods and nodes, finds the nodes of best fit for newly created Pods, and binds node info to the Pod `spec.nodeName`. As a result, to meet our special scheduling requirements, one straightforward way to extend Kubernetes scheduling is by running a separate customized scheduler along with the default scheduler. We can run multiple such schedulers simultaneously.

How multiple schedulers work together

Each scheduler should have a different name to indicate itself. We instruct Kubernetes which scheduler to use for a Pod based on the `spec.schedulerName` in the Pod specification. Schedulers having the matching name with Pod `spec.schedulerName` are responsible for scheduling work, which is exactly like the default scheduler.



Multiple schedulers in Kubernetes

The default Kubernetes scheduler name is `default-scheduler`, which is unchangeable for now. The empty `spec.schedulerName` value in a Pod specification will default to `default-scheduler` on creation. It's worth noting that the default scheduler is irreplaceable, because we do need to schedule Pods that don't specify schedulers with the `spec.schedulerName`. We must make sure the default scheduler works with the other schedulers, unless we apply another scheduler name to be the default using a mutating webhook.

```
// PodSpec is a description of a Pod.
type PodSpec struct {
    ...

    // Specifies the hostname of the Pod
    // If not specified, the Pod's hostname will be set to a system-defined value.
    // +optional
    Hostname string `json:"hostname,omitempty" protobuf:"bytes,16,opt,name=hostname"`
    // If specified, the fully qualified Pod hostname will be "<hostname>.<subdomain>.<pod namespace>.sv
    // IIf not specified, the Pod will not have a domain name at all.
    // +optional
    Subdomain string `json:"subdomain,omitempty" protobuf:"bytes,17,opt,name=subdomain"`
    // If specified, the Pod's scheduling constraints.
    // +optional
    Affinity *Affinity `json:"affinity,omitempty" protobuf:"bytes,18,opt,name=affinity"`
    // If specified, the Pod will be dispatched by specified scheduler.
    // If not specified, the Pod will be dispatched by default scheduler.
    // +optional
    SchedulerName string `json:"schedulerName,omitempty" protobuf:"bytes,19,opt,name=schedulerName"`

    ...
}
```

The SchedulerName in PodSpec

How to configure multiple schedulers

Now, let's learn how to configure multiple schedulers working together.

The flag `--config` in the `kube-scheduler` is the entry where we configure the default scheduler. This configuration accepts a YAML file and uses the `KubeSchedulerConfiguration` to serialize the configurations, as shown below:

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

The sample KubeSchedulerConfiguration file

In this definition, we can define multiple profiles with discrepant names for the `schedulerName`, where our implementations of schedulers can use the `KubeSchedulerConfiguration` for configurations as well. Each scheduler uses this name to determine if it's responsible for scheduling a specific Pod. Let's take a look at the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-demo-my-scheduler
  labels:
    name: multi-scheduler-example
spec:
  schedulerName: my-scheduler
  containers:
  - name: pod-for-my-scheduler
    image: nginx:1.23
    restartPolicy: OnFailure
    ports:
    - containerPort: 80
```

Pod uses my-scheduler as schedulerName

In this case, we declare a Pod called `pod-demo-my-scheduler` and specify the value of the `spec.schedulerName` to the `my-scheduler`, instead of `kube-scheduler`. If we leave this field empty, it will default to `kube-scheduler`. Upon creating this Pod on the `kube-apiserver`, the default scheduler will directly ignore scheduling this Pod, because it has a scheduler name that doesn't match. The scheduler with the name `my-scheduler` takes charge of this Pod's scheduling process.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-demo
  labels:
    purpose: demo
spec:
  schedulerName: default-scheduler
  containers:
  - name: pod-demo
    image: nginx:1.23
    restartPolicy: OnFailure
    ports:
    - containerPort: 80
```

Pod uses default scheduling

If a Pod sets an unavailable value on the `spec.schedulerName`, it will remain in the `Pending` state until a scheduler having the same name with the `spec.schedulerName` is available and finishes the scheduling process.

Disadvantages

The coexistence of multiple schedulers in a cluster might bring in some troubles.

We may have problems when Pods are scheduled onto the same node by multiple schedulers. Every running scheduler has a separate global view on the whole cluster, including nodes and resource utilizations. When multiple Pods get scheduled at the same time, the schedulers may assign the same node for these Pods. As a result, that node runs out of resources. What's worse, the node may be marked as not ready, and containers running on that node may be evicted.

Additionally, maintaining a custom scheduler is not trivial, not to mention the high performance and low latency. It requires developers to comprehensively understand the whole scheduler design, architecture, implementations, and multiple stages/processes during a scheduling cycle.

Conclusion

A custom scheduler, especially a high performance scheduler, has set a high bar for developers, and brings in maintenance costs and compatibility problems. We need a better way to extend our scheduling.

← Back
A Deep Dive into Kubernetes Scheduling

☑
Next →
Scheduler Extender