A Deep Dive into Kubernetes Scheduling

Learn about the powerful capabilities of the Kubernetes scheduling component.

We'll cover the following

Kubernetes scheduling

The `kube-scheduler` is one of the three core components in the Kubernetes control plane, together with the `kube-apiserver` and `kube-controller-manager`. It can run with multiple replicas, but only the one that acquires the leader lock takes the charge of scheduling a Pod to a node that fits it best. Various scheduling strategies are supported, such as Pod topology spread, Pod Quality of Service (QoS), Pod priority, node taints, Pod tolerations, node anti-affinity, Pod affinity/anti-affinity, etc. When the Pod is bound to a target node, the `kubelet` running on that node will get notified and retrieve that Pod from the `kube-apiserver`. Then, the `kubelet` calls the container runtime (such as containerd) to create containers according to Pod specification.

In this lesson, we'll learn about the Kubernetes scheduling system in detail.

What does a node of best fit mean?

The `kube-scheduler` keeps watching the `kube-apiserver` for newly created Pods and finds a node that best fits each of them. However, how does the `kube-scheduler` define a node as being the best fit for a Pod?

Let's find out. Below is a code snippet that describes the core part for Pod scheduling (**lines 17–46**).

```
 1  // The schedulePod() function tries to schedule the given Pod to ones of the nodes in the node list.
 2  // If it succeeds, it will return the name of the node.
 3  // If it fails, it will return a FitError with reasons.
 4  func (sched *Scheduler) schedulePod(ctx context.Context, fwk framework.Framework, state *framework.Cycl
 5      trace := utiltrace.New("Scheduling", utiltrace.Field{Key: "namespace", Value: pod.Namespace}, utiltr
 6      defer trace.LogIfLong(100 * time.Millisecond)
 7
 8      if err := sched.Cache.UpdateSnapshot(sched.nodeInfoSnapshot); err != nil {
 9          return result, err
10      }
11      trace.Step("Snapshotting scheduler cache and node infos done")
```

```go
    if sched.nodeInfoSnapshot.NumNodes() == 0 {
        return result, ErrNoNodesAvailable
    }

    feasibleNodes, diagnosis, err := sched.findNodesThatFitPod(ctx, fwk, state, pod)
    if err != nil {
        return result, err
    }
    trace.Step("Computing predicates done")

    if len(feasibleNodes) == 0 {
        return result, &framework.FitError{
            Pod:         pod,
            NumAllNodes: sched.nodeInfoSnapshot.NumNodes(),
            Diagnosis:   diagnosis,
        }
    }

    // When only one node after predicate, just use it.
```

Core part of Pod scheduling

In the function `schedulePod`, there are two main functions, `findNodesThatFitPod` and `prioritizeNodes`. They clearly describe how the `kube-scheduler` finds a node that best fits the a Pod, in other words, through `filtering` and `prioritizing`. The **filtering cycle** is also called the **nodes predicting cycle**, where a set of scheduling filters, such as node selectors and node affinities, will be applied on all nodes to filter out qualified nodes. If more than 1 node is qualified to run the Pod, we'll prioritize all those qualified nodes. The node getting the highest score is the node best fit for the Pod.

A simple Pod with `nodeName`

A simple way to run a Pod on the desired node is setting the `nodeName` in `PodSpec` directly as follows:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
  nodeName: node-01
```

A simple Pod with nodeName

The Pod above named `nginx` will run directly on `node-01` without scheduling. However, this explicit way has side effects that may lead to unhealthy Pods. The `nodeName` may be invalid or changed. The desired node may run out of resources as well. As a result, it's not suggested to explicitly use the `nodeName` in production environments.

An example Pod with scheduling rules

If we want to run Pods on a specific set of nodes, we should use `nodeSelector` or `nodeAffinity` to specify the matching labels. The `nodeAffinity` rules could work together with those of `nodeSelector`, but are more powerful. The `nodeAffinity` rules can be set to match node labels and also be expressions that contains a group of selectors for matching.

There are four `nodeAffinity` rules we can use:

- `requiredDuringSchedulingIgnoredDuringExecution`
- `requiredDuringSchedulingRequiredDuringExecution`
- `preferredDuringSchedulingIgnoredDuringExecution`
- `preferredDuringSchedulingRequiredDuringExecution`

The rule names are quite straightforward, and they consist of two conditions (`required` and `preferred`) and two stages (`scheduling` and `execution`). The `nodeAffinity` rules that start with `required` set hard requirements that must be satisfied during scheduling, while `preferred` sets soft enforced requirements but aren't guaranteed. The **scheduling stage** means the rule will be enforced during the node assignment of the Pod. The **execution stage** refers to node labels changing after the Pod has been assigned.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: production
spec:
  tolerations:
  - key: "node.kubernetes.io/demo"
    operator: "Exists"
    effect: "NoSchedule"
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-west
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-west-1
            - us-west-2
  containers:
  - name: nginx
    image: nginx
```

Pod with tolerations and node affinities

We've set `nodeAffinity` rules in the `nginx` Pod above. Those affinity rules will make sure the `kube-scheduler` places the Pod with a node in the `us-west` region and has preferences for node zones `us-west-1` or `us-west-2`.

Together with the `nodeAffinity` rules, we set the tolerations so that the Pod can run on the node with the taint `node.kubernetes.io/demo=true:NoSchedule`. Taints can be used to indicate the node features, node issues, etc.

Conclusion

Most of the time, we don't have to check the logs and configuration parameters of the `kube-scheduler`. It has been quite stable and mature enough to meet our business needs.

← Back
Quiz on Admission Control

☑

Next →
Multiple Schedulers