

Kubernetes 调度器详解

张晋涛

API7.ai 云原生技术专家



个人介绍

- Apache APISIX PMC
- Kubernetes Ingress NGINX maintainer
- Microsoft MVP
- 『K8S 生态周报』维护者
- GitHub: tao12345666333
- zhangjintao@apache.org



目录

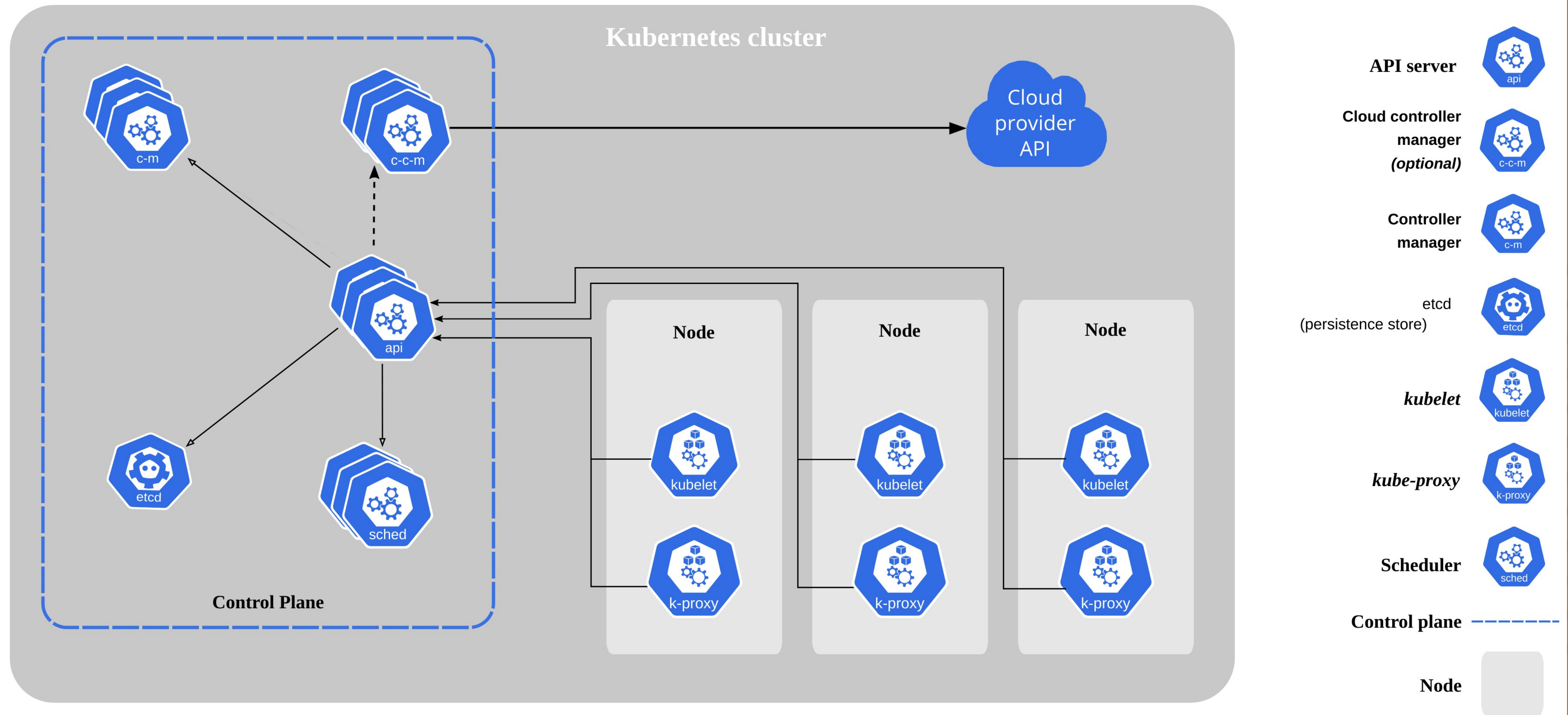
- Kubernetes 调度器的发展历史
- Kubernetes 如何进行调度
- 如何自定义 Kubernetes 调度器

Kubernetes 调度器的发展历史

前置知识

- Pod 是 Kubernetes 中最小的调度单元
- 一个 Pod 中可以包含一个或多个 container
- Pod 运行在 Node 上
- Pod 中包含的 container 运行需要消耗资源
 - CPU
 - 内存
 - 存储
 - 扩展资源
- Pod 存在优先级的区别
- Pod 可以被单独创建也可以由其他资源控制器创建

Kubernetes 概览

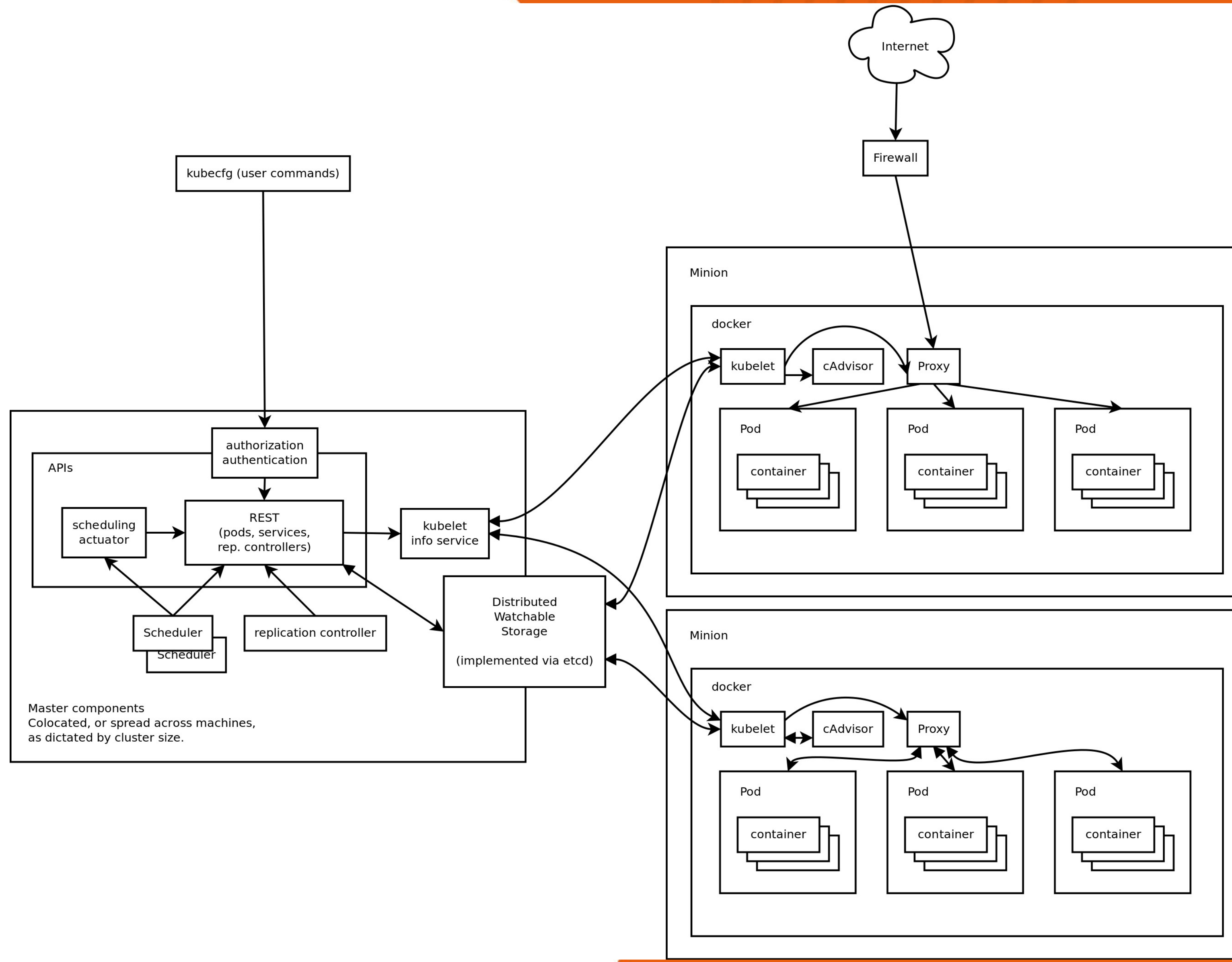


什么是调度

- 将新创建的 Pod 与 Node 进行关联
 - Node 可用性
 - 资源容量
 - 亲和性 / 反亲和性
 - 拓扑感知
 - taint
 - ...

最简单的 Scheduler 实现

- 随机 (random)
- 轮训 (roundrobin)



随机 (random)

- 随机数
- 取余

```
41 // Schedule schedules a pod on a random machine which matches its requirement.
40 func (s *RandomFitScheduler) Schedule(pod api.Pod, minionLister MinionLister) (string, error) {
39     machines, err := minionLister.List()
38     if err != nil {
37         return "", err
36     }
35     machineToPods := map[string][]api.Pod{}
34     // TODO: perform more targeted query...
33     pods, err := s.podLister.ListPods(labels.Everything())
32     if err != nil {
31         return "", err
30     }
29     for _, scheduledPod := range pods {
28         host := scheduledPod.CurrentState.Host
27         machineToPods[host] = append(machineToPods[host], scheduledPod)
26     }
25     var machineOptions []string
24     for _, machine := range machines {
23         podFits := true
22         for _, scheduledPod := range machineToPods[machine] {
21             for _, container := range pod.DesiredState.Manifest.Containers {
20                 for _, port := range container.Ports {
19                     if port.HostPort == 0 {
18                         continue
17                     }
16                     if s.containsPort(scheduledPod, port) {
15                         podFits = false
14                     }
13                 }
12             }
11         }
10         if podFits {
9             machineOptions = append(machineOptions, machine)
8         }
7     }
6     if len(machineOptions) == 0 {
5         return "", fmt.Errorf("failed to find fit for %#v", pod)
4     }
3     s.randomLock.Lock()
2     defer s.randomLock.Unlock()
1     return machineOptions[s.random.Int()%len(machineOptions)], nil
04 }
```

NORMAL +0 ~0 -0 ↵ v0.2 pkg/scheduler/randomfit.go

分阶段调度

- 预选（Predicate）
 - 是否满足基本条件
- 优选（Priorities）
 - 是否为最佳选择

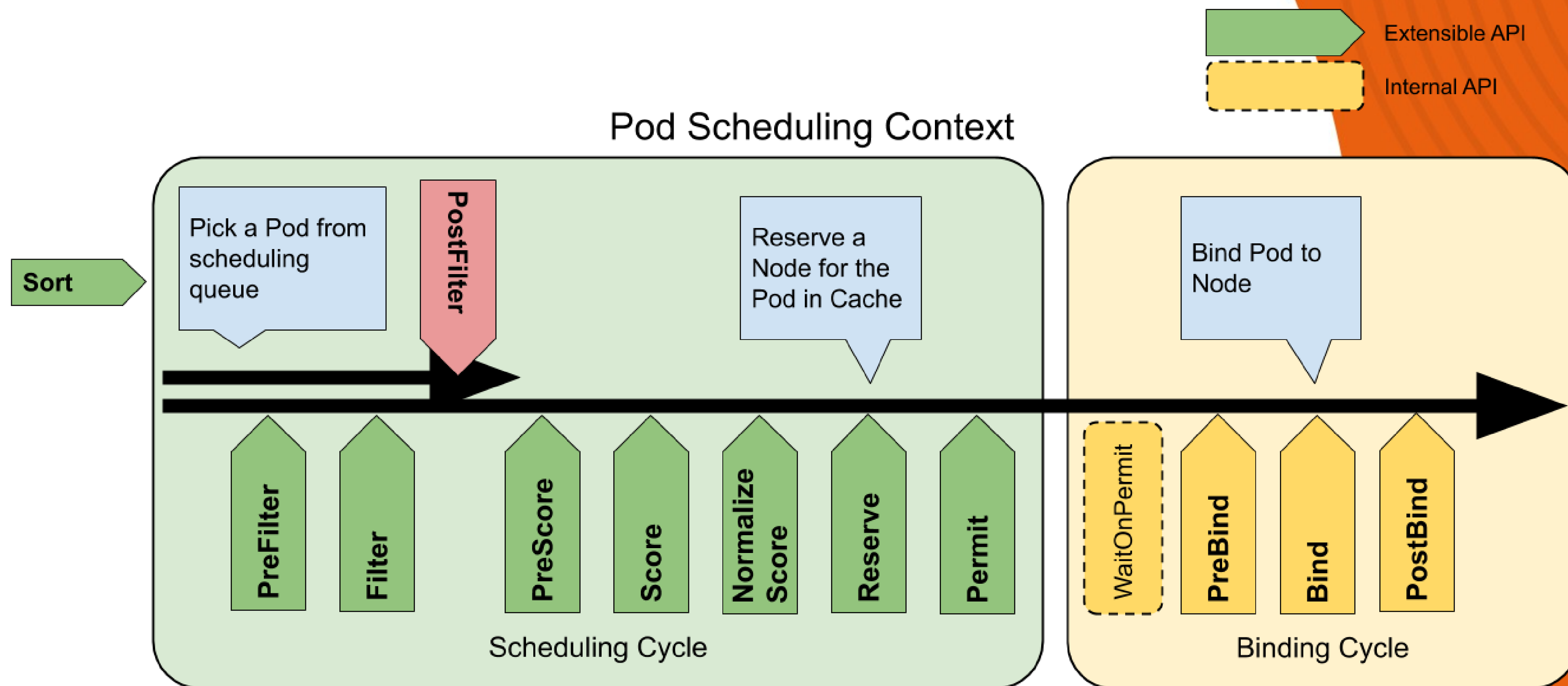
可扩展的 Scheduler （extender）

- 使用 webhook 方式进行扩展
 - 扩展点有限
 - 需要 JSON 编解码（效率低）
 - 无法及时终止（交互）
 - 不能使用 Scheduler 的缓存

```
{
  "predicates": [
    {
      "name": "HostName"
    },
    {
      "name": "MatchNodeSelector"
    },
    {
      "name": "PodFitsResources"
    }
  ],
  "priorities": [
    {
      "name": "LeastRequestedPriority",
      "weight": 1
    }
  ],
  "extenders": [
    {
      "urlPrefix": "http://127.0.0.1:12345/api/scheduler",
      "filterVerb": "filter",
      "enableHttps": false
    }
  ]
}
```

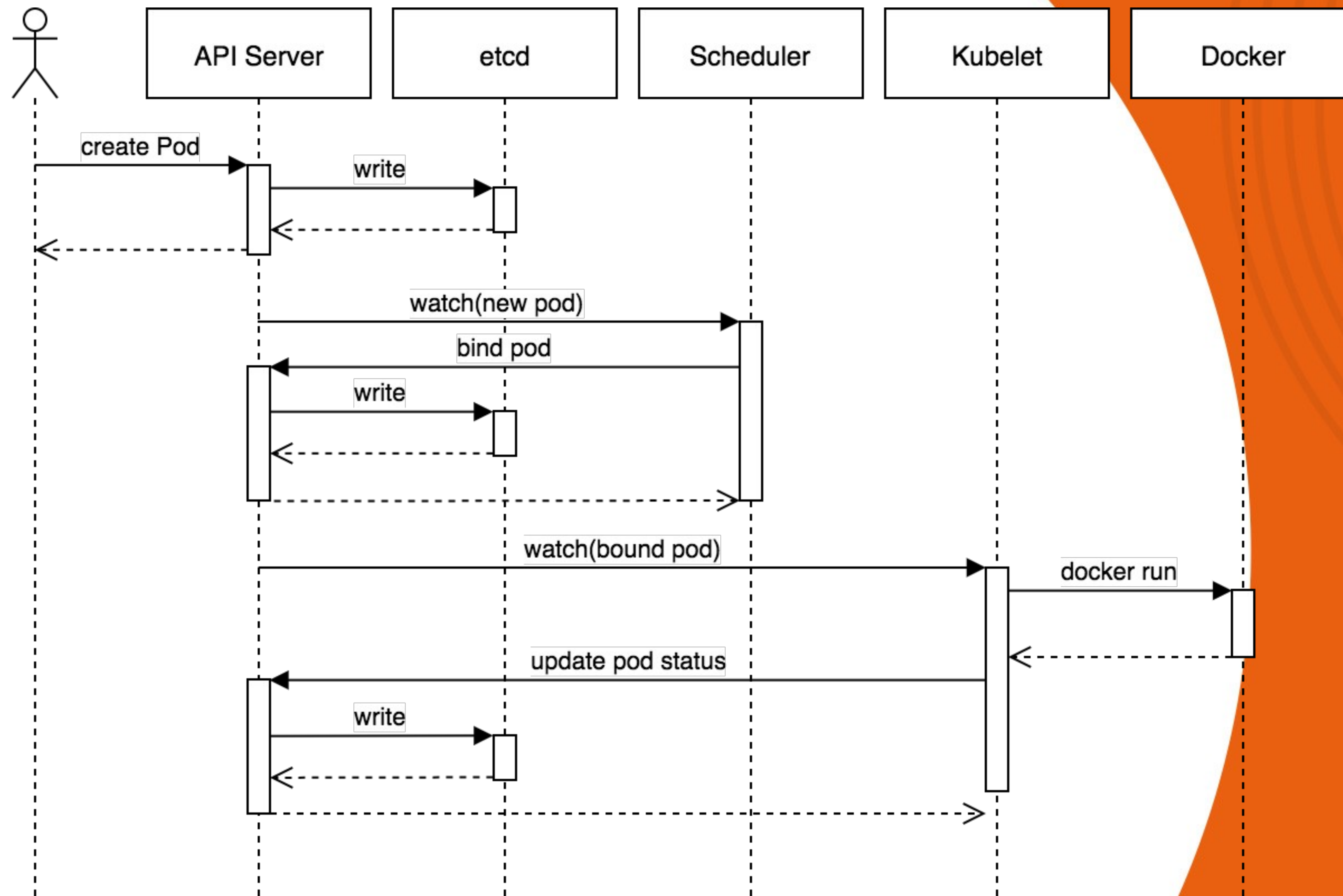
可扩展的 Scheduling Framework

- 扩展点丰富
- 在框架中提供扩展点



Kubernetes 如何进行调度

创建 Pod 的过程



schedulePod

```
311 func (sched *Scheduler) schedulePod(ctx context.Context, fwk framework.Framework, state *framework.CycleState, pod *v1.Pod) (result ScheduleResult, err error) {
1   if err := sched.Cache.UpdateSnapshot(sched.nodeInfoSnapshot); err != nil {
2       return result, err
3   }
4   if sched.nodeInfoSnapshot.NumNodes() == 0 {
5       return result, ErrNoNodesAvailable
6   }
7   feasibleNodes, diagnosis, err := sched.findNodesThatFitPod(ctx, fwk, state, pod)
8   if err != nil {
9       return result, err
10  }
11  if len(feasibleNodes) == 0 {
12      return result, &framework.FitError{
13          Pod:      pod,
14          NumAllNodes: sched.nodeInfoSnapshot.NumNodes(),
15          Diagnosis:  diagnosis,
16      }
17  }
18  // When only one node after predicate, just use it.
19  if len(feasibleNodes) == 1 {
20      return ScheduleResult{
21          SuggestedHost:  feasibleNodes[0].Name,
22          EvaluatedNodes: 1 + len(diagnosis.NodeToStatusMap),
23          FeasibleNodes:  1,
24      }, nil
25  }
26  priorityList, err := prioritizeNodes(ctx, sched.Extenders, fwk, state, pod, feasibleNodes)
27  if err != nil {
28      return result, err
29  }
30  host, err := selectHost(priorityList)
31  trace.Step("Prioritizing done")
32
33  return ScheduleResult{
34      SuggestedHost:  host,
35      EvaluatedNodes: len(feasibleNodes) + len(diagnosis.NodeToStatusMap),
36      FeasibleNodes:  len(feasibleNodes),
37  }, err
38 }
```

策略

- 存储
- Node 匹配度
- Pod 拓扑
- 资源水位
- Node 亲和 / 反亲和
- Pod 亲和 / 反亲和
- ...

如何自定义 Kubernetes 调度器

自定义方式

- 修改 kube-scheduler 调度规则并重新编译
- 实现独立的 scheduler 组件，代替或和 kube-scheduler 共同使用
- 实现 scheduler extender
- Scheduling Framework

扩展配置

- kube-scheduler 可接受多个配置文件
- <https://github.com/kubernetes-sigs/scheduler-plugins/>

```
apiVersion: kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfiguration
leaderElection:
  leaderElect: false
clientConnection:
  kubeconfig: "REPLACE_ME_WITH_KUBE_CONFIG_PATH"
profiles:
- schedulerName: default-scheduler
  plugins:
    score:
      enabled:
        - name: PodState
```

初始化 plugin

- 选择扩展点
- 注册名称

```
package podstate

import (
    "context"
    "fmt"
    "math"

    "k8s.io/api/core/v1"
    "k8s.io/apimachinery/pkg/runtime"
    "k8s.io/kubernetes/pkg/scheduler/framework"
)

type PodState struct {
    handle framework.Handle
}

var _ = framework.ScorePlugin(&PodState{})

// Name is the name of the plugin used in the Registry and configurations.
const Name = "PodState"

func (ps *PodState) Name() string {
    return Name
}

// core logic

// New initializes a new plugin and returns it.
func New(_ runtime.Object, h framework.Handle) (framework.Plugin, error) {
    return &PodState{handle: h}, nil
}
```


核心逻辑

- 越多终止状态的 Node 份数越高
- 返回最终打分结果

```
// Score invoked at the score extension point.
func (ps *PodState) Score(ctx context.Context, state *framework.CycleState, pod *v1.Pod, nodeName string)
(int64, *framework.Status) {
    nodeInfo, err := ps.handle.SnapshotSharedLister().NodeInfos().Get(nodeName)
    if err != nil {
        return 0, framework.NewStatus(framework.Error, fmt.Sprintf("getting node %q from Snapshot: %v",
            nodeName, err))
    }

    // pe.score favors nodes with terminating pods instead of nominated pods
    // It calculates the sum of the node's terminating pods and nominated pods
    return ps.score(nodeInfo)
}

// ScoreExtensions of the Score plugin.
func (ps *PodState) ScoreExtensions() framework.ScoreExtensions {
    return ps
}

func (ps *PodState) score(nodeInfo *framework.NodeInfo) (int64, *framework.Status) {
    var terminatingPodNum, nominatedPodNum int64
    // get nominated Pods for node from nominatedPodMap
    nominatedPodNum = int64(len(ps.handle.NominatedPodsForNode(nodeInfo.Node().Name)))
    for _, p := range nodeInfo.Pods {
        // Pod is terminating if DeletionTimestamp has been set
        if p.Pod.DeletionTimestamp != nil {
            terminatingPodNum++
        }
    }
    return terminatingPodNum - nominatedPodNum, nil
}

func (ps *PodState) NormalizeScore(ctx context.Context, state *framework.CycleState, pod *v1.Pod, scores
framework.NodeScoreList) *framework.Status {
    // Find highest and lowest scores.
    var highest int64 = -math.MaxInt64
    var lowest int64 = math.MaxInt64
    for _, nodeScore := range scores {
        if nodeScore.Score > highest {
            highest = nodeScore.Score
        }
        if nodeScore.Score < lowest {
            lowest = nodeScore.Score
        }
    }

    // Transform the highest to lowest score range to fit the framework's min to max node score range.
    oldRange := highest - lowest
    newRange := framework.MaxNodeScore - framework.MinNodeScore
    for i, nodeScore := range scores {
        if oldRange == 0 {
            scores[i].Score = framework.MinNodeScore
        } else {
            scores[i].Score = ((nodeScore.Score - lowest) * newRange / oldRange) + framework.MinNodeScore
        }
    }

    return nil
}
```

总结

- 调度是 Kubernetes 的核心能力之一
- 良好的 / 适当的调度逻辑可辅助业务
- 降本增效
- 基于机器学习的调度也会有一些发展

THANKS

 极客时间 | 训练营