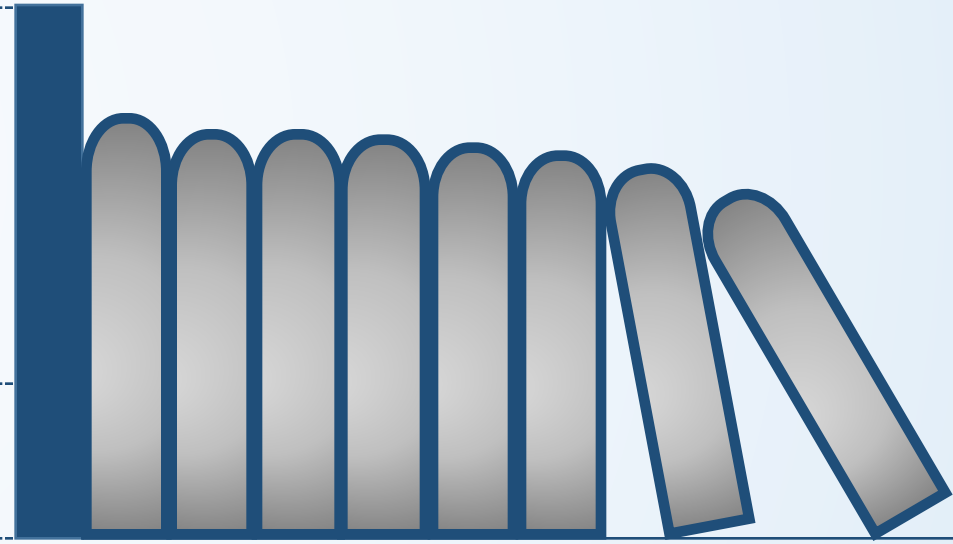
Kubernetes调度

主讲人：宋小金

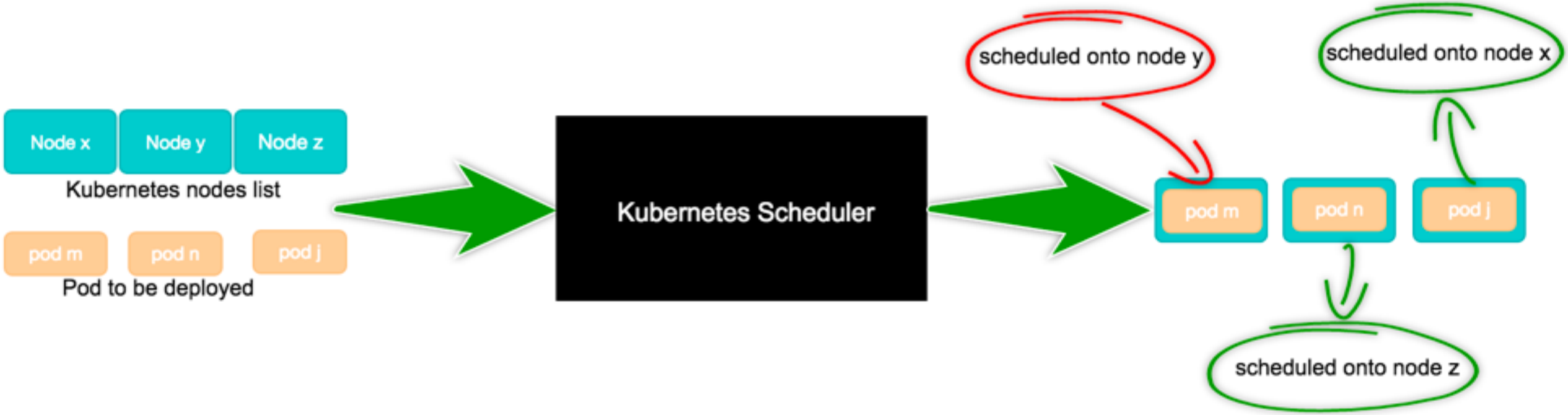


目录

预期收获

- *学习Kubernetes的调度策略*
- **学习调度失败问题定位**

Pod调度



这个过程看起来似乎比较简单，但实际生产环境的调度过程中，有很多问题需要考虑：

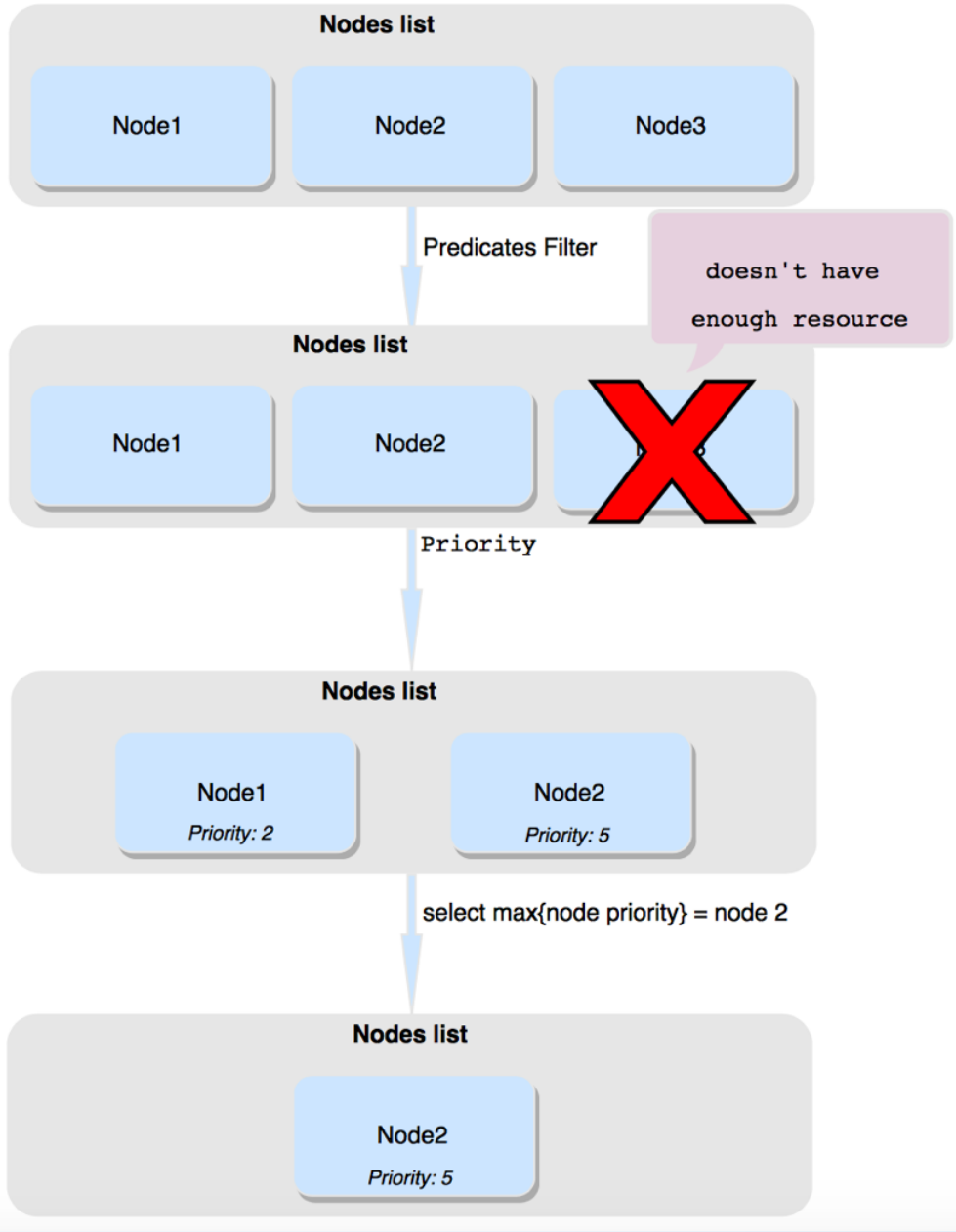
- 首先，如何保证全部计算节点调度的公平性？如何保证每个节点都能被分配资源？
- 其次，计算资源如何能够被高效利用？集群所有计算资源如何才能被最大化的使用？
- 再次，如何保证Pod调度的性能和效率？如何能够快速的对大批量的Pod完成调度到较优的计算节点之上？
- 最后，用户最了解自己的业务，用户是否可以根据实际需求定制自己的调度逻辑和策略？

Pod调度过程

调度过程分为2个阶段：

- 第一阶段：预选过程，过滤节点，调度器用一组规则过滤掉不符合要求的主机。比如Pod指定了所需要的资源量，那么可用资源比Pod需要的资源量少的主机会被过滤掉。
- 第二阶段：优选过程，节点优先级打分，对第一步筛选出的符合要求的主机进行打分，在主机打分阶段，调度器会考虑一些整体优化策略，比如把容一个Replication Controller的副本分布到不同的主机上，使用最低负载的主机等。

代码位置（1.10）：
https://github.com/kubernetes/kubernetes/tree/release-1.10/pkg/scheduler/algorithm



Pod调度过程

优选（Priorities）

经过预选策略（Predicates）对节点过滤，获取节点列表，再对符合需求节点列表进行打分，最终选择Pod调度到一个分值最高节点

最终主机的得分用以下公式计算得出：
finalScoreNode = (weight1 * priorityFunc1) + (weight2 * priorityFunc2) + … + (weightn * priorityFuncn)

	node1	node2	.....	nodeN
PriorityFunc1	S(1, 1)	S(1, 2)		S(1, N)
PriorityFunc2	S(2, 1)	S(2, 2)		S(2, N)
.....				
PriorityFuncM	S(M, 1)	S(M, 2)		S(M, N)
Result	Score1	Score2	.....	ScoreN

Pod will be scheduled onto the node with the highest score

Node定义

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
status:
  addresses:
  - address: 10.162.197.135
    type: InternalIP
  allocatable:
    cpu: "8"
    memory: 16309412Ki
    pods: "110"
  capacity:
    cpu: "8"
    memory: 16411812Ki
    pods: "110"
  conditions: {...}
  daemonEndpoints:
    kubeletEndpoint:
      Port: 10250
  images: {...}
  nodeInfo: {...}
```

```
kubectl get node <node-name> -o yaml
```

一个node内可分配资源总量

Pod定义

执行 kubectl explain pod.spec 查看 pod.spec

资源分配

采用的调度器

普通调度策略

高级调度策略

```
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      requests:
        memory: "10Gi"
        cpu: "500m"
      limits:
        memory: "10Gi"
        cpu: "500m"
  schedulerName: default-scheduler
  nodeName: node-n1
  restartPolicy: Always
  nodeSelector: {...}
  affinity: {...}
  tolerations: {...}
status: {}
```

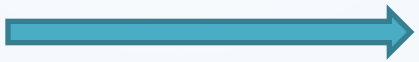
K8S 调度器的资源分配机制

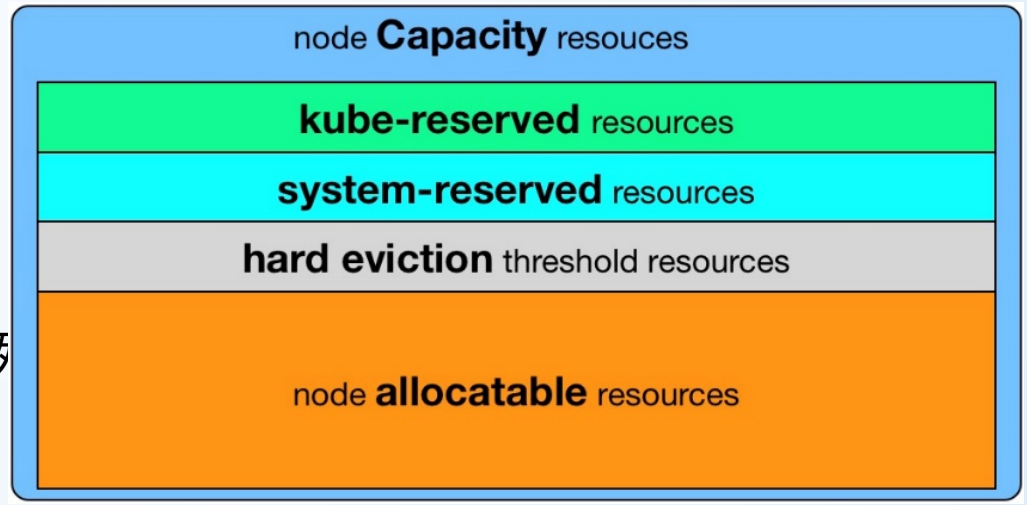
- 基于Pod中容器request资源“总和”调度
  - resoureces.limits影响pod的运行资源上限，**不影响调度**
  - initContainer取**最大值**，container取**累加值**，最后**两者中取大**
    即 Max(Max(initContainers.requests), Sum(containers.requests))
  - 未指定request资源时（QoS Guaranteed除外），按**资源需求为0**进行调度

- 基于**资源声明量**的调度，即：request字段值，而非实际占用
  - 不依赖监控，系统不会过于敏感
  - 能否调度成功：pod.request < node.allocatable - node.requested

- Kubernetes node 资源的盒子模型

- 资源分配相关算法
  - GeneralPredicates（主要是PodFitsResources）
  - LeastRequestedPriority
  - BalancedResourceAllocation，平衡cpu/mem的消耗比例





K8S 调度器的资源分配机制

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  initContainers:
  - name: ic1
    resources:
      requests:
        cpu: "1"
        memory: "1G"
  - name: ic2
    resources:
      requests:
        cpu: "1"
        memory: "3G"
  containers:
  - name: container1
    resources:
      requests:
        cpu: "500m"
        memory: "1G"
  - name: container2
    resources:
      requests:
        cpu: "500m"
        memory: "1G"
```

InitContainers：

逐个运行并退出，之后才拉起containers

资源需求取单个容器的最大值

Containers：

同时运行，资源需求为所有容器累加

最终结果：cpu 1，memory 3G

普通调度策略

nodeSelector【将来会被废弃】：将 Pod 调度到特定的 Node 上

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    pod-template-hash: "4173307778"
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources: {}
  nodeSelector:
    disktype: ssd
    node-flavor: s3.large.2
```

- 语法格式：map[string]string
- 作用：
  - 匹配node.labels
  - 排除不包含nodeSelector中指定label的所有node
  - 匹配机制 —— 完全匹配

最终结果：cpu 1，memory 3G

高级调度策略

nodeAffinity：nodeSelector 升级版，涵盖其全部功能

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-flavor
            operator: In
            values:
            - s3.large.2
            - s3.large.3
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
            - key: node-flavor
              operator: In
              values:
              - s3.large.2
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
```

- 与nodeSelector关键差异
  - 引入运算符：In，NotIn（labelselector语法）
  - 支持枚举label可能的取值，如 zone in [az1, az2, az3...]
  - 支持硬性过滤和软性评分
  - 硬性过滤规则支持指定 多条件之间的逻辑或运算
  - 软性评分规则支持 设置条件权重值

硬性过滤：

排除不具备指定label的node

软性评分：

不具备指定label的node打低分，降低node被选中的几率

高级调度策略

podAffinity：让某些 Pod 分布在同一组 Node 上

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: kubernetes.io/zone
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```

- 与nodeAffinity的关键差异
  - 定义在PodSpec中，亲和与反亲和规则具有对称性
  - labelSelector的匹配对象为Pod
  - 对node分组，依据label-key = topologyKey，每个label-value取值为一组
  - 硬性过滤规则，条件间只有逻辑与运算

硬性过滤：

排除不具备指定pod的node组

软性评分：

不具备指定pod的node组打低分，降低该组node被选中的几率

高级调度策略

podAntiAffinity：避免某些 Pod 分布在同一组 Node 上

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: kubernetes.io/zone
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```

- 与podAffinity的差异
  - 匹配过程相同
  - 最终处理调度结果时取反
- 即
  - podAffinity中可调度节点，在podAntiAffinity中为不可调度
  - podAffinity中高分节点，在podAntiAffinity中为低分

最终结果：cpu 1，memory 3G

高级调度策略

Taints：避免 Pod 调度到特定 Node 上

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
  taints:
  - effect: NoSchedule
    key: accelerator
    timeAdded: null
    value: gpu
status: {...}
```

- 带effect的特殊label，对Pod有排斥性
  - 硬性排斥 NoSchedule
  - 软性排斥 PreferNoSchedule
- 系统创建的taint附带时间戳
  - effect为NoExecute
  - 便于触发对Pod的超时驱逐
- 典型用法：预留特殊节点做特殊用途

给node添加taint

[redacted]

删除taint

[redacted]

高级调度策略

Tolerations：允许 Pod 调度到有特定 taints 的 Node 上

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - name: my-pod
    image: nginx
  tolerations:
  - key: accelerator
    operator: Equal
    value: gpu
    effect: NoSchedule
```

无视排斥

```yaml
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node-n1
  name: node-n1
spec:
  externalID: node-n1
  taints:
  - effect: NoSchedule
    key: accelerator
    timeAdded: null
    value: gpu
status: {...}
```

- 完全匹配
  - 例：<key>=<value>:<effect>
- 匹配任意taint value
  - Operator为Exists，value为空
  - 例： <key>:<effect>
- 匹配任意 taint effect
  - effect为空
  - 例： <key>=<value>

注：<key>=<value>:<effect>为 kubectl describe pod中的写法

不经过调度器调度Pod

nodeName：将Pod手动调度到特定的 Node 上

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
    resources:
      requests:
        memory: "10Gi"
        cpu: "500m"
      limits:
        memory: "10Gi"
        cpu: "500m"
  schedulerName: default-scheduler
  nodeName: node-n1
  restartPolicy: Always
```

- 适用场景：
  - 使用简单，调度器不工作时，临时救急

最终结果：cpu 1，memory 3G

不经过调度器调度Pod

DaemonSet：

- 每个node上部署一个相同的pod
- 通常用来部署集群中的agent，例如filebeat

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
spec:
  selector:
    matchLabels:
      name: my-daemonset
  template:
    metadata:
      labels:
        name: my-daemonset
    spec:
      containers:
      - name: container
        image: k8s.gcr.io/pause:2.0
```

等价于

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
spec:
  replicas: <# of nodes>
  selector:
    matchLabels:
      podlabel: daemonset
  teplate:
    metadata:
      labels:
        podlabel: daemonset
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: podlabel
                operator: In
                values:
                - daemonset
            topologyKey: kubernetes.io/hostname
      containers:
      - name: container
        image: k8s.gcr.io/pause:2.0
```

调度结果和失败原因分析

- 查看调度结果： kubectl get po pod_name –o wide
- 查看调度失败原因: kubectl describe po pod_name
- 调度失败错误列表（1.10版本）：https://github.com/kubernetes/kubernetes/blob/release-1.10/pkg/scheduler/algorithm/predicates/error.go

```
// ErrPodNotFitsHostPorts is used for PodFitsHostPorts predicate error.
ErrPodNotFitsHostPorts = newPredicateFailureError("PodFitsHostPorts", "node(s) didn't have free ports for the requested
// ErrNodeLabelPresenceViolated is used for CheckNodeLabelPresence predicate error.
ErrNodeLabelPresenceViolated = newPredicateFailureError("CheckNodeLabelPresence", "node(s) didn't have the requested la
// ErrServiceAffinityViolated is used for CheckServiceAffinity predicate error.
ErrServiceAffinityViolated = newPredicateFailureError("CheckServiceAffinity", "node(s) didn't match service affinity")
// ErrMaxVolumeCountExceeded is used for MaxVolumeCount predicate error.
ErrMaxVolumeCountExceeded = newPredicateFailureError("MaxVolumeCount", "node(s) exceed max volume count")
// ErrNodeUnderMemoryPressure is used for NodeUnderMemoryPressure predicate error.
ErrNodeUnderMemoryPressure = newPredicateFailureError("NodeUnderMemoryPressure", "node(s) had memory pressure")
// ErrNodeUnderDiskPressure is used for NodeUnderDiskPressure predicate error.
ErrNodeUnderDiskPressure = newPredicateFailureError("NodeUnderDiskPressure", "node(s) had disk pressure")
// ErrNodeOutOfDisk is used for NodeOutOfDisk predicate error.
ErrNodeOutOfDisk = newPredicateFailureError("NodeOutOfDisk", "node(s) were out of disk space")
// ErrNodeNotReady is used for NodeNotReady predicate error.
ErrNodeNotReady = newPredicateFailureError("NodeNotReady", "node(s) were not ready")
```

调度失败原因分析

```
⚡ root@SZV1000112844 ~ kubectl describe po/my-pod-85546fffc4-kzxcl
Name:           my-pod-85546fffc4-kzxcl
Namespace:      default
Node:           <none>
Labels:         pod-template-hash=4110299970
                run=my-pod
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"ReplicaSet","namespace":"default","r
b50-c23d-11e8-8128-286ed488fc60",...
Status:         Pending
IP:
Created By:     ReplicaSet/my-pod-85546fffc4
Controlled By:  ReplicaSet/my-pod-85546fffc4
Containers:
  my-pod:
    Image:        nginx
    Port:         80/TCP
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-gv7vg (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  default-token-gv7vg:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-gv7vg
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  foo=bar
Tolerations:     <none>
Events:
  Type     Reason            Age               From               Message
  ----     ------            ----              ----               -------
  Warning  FailedScheduling  7s (x5 over 14s)  default-scheduler  No nodes are available that match all of the predicates: MatchNodeSelector (1).
```

自定义调度

```
#!/bin/bash

SERVER='localhost:8001'

while true;

do

    for PODNAME in $(kubectl --server $SERVER get pods -o json | jq '.items[] | select(.spec.schedulerNa
me == "my-scheduler") | select(.spec.nodeName == null) | .metadata.name' | tr -d '"')

;

    do

        NODES=($(kubectl --server $SERVER get nodes -o json | jq '.items[].metadata.name' | tr -d '"'))

        NUMNODES=${#NODES[@]}

        CHOSEN=${NODES[$[$RANDOM % $NUMNODES]]}

        curl --header "Content-Type:application/json" --request POST --data '{"apiVersion":"v1", "kind":
"Binding", "metadata": {"name": "'$PODNAME'"}, "target": {"apiVersion": "v1", "kind"

: "Node", "name": "'$CHOSEN'"}}' http://$SERVER/api/v1/namespaces/default/pods/$PODNAME/binding/

        echo "Assigned $PODNAME to $CHOSEN"

    done

    sleep 1

done
```

随机选择空闲节点

调用binding接口进行节点绑定

```
apiVersion: v1

kind: Pod

metadata:

  name: nginx

  labels:

    app: nginx

spec:

  schedulerName: my-scheduler

  containers:

  - name: nginx

    image: nginx:1.10
```

例子：
https://kubernetes.io/blog/2017/03/advanced-scheduling-in-kubernetes/

Pod优先级

Pod优先级（Priority）

Pod优先级（Priority）和抢占（Preemption）是Kubernetes 1.8版本引入的功能，在1.8版本默认是禁用的，1.11版本当前处于

与前面所讲的调度优选策略中的优先级（Priorities）不同，前文所讲的优先级指的是节点优先级，而pod priority指的是Pod

Pod优先级和抢占

为了定义Pod优先级，需要先定义PriorityClass对象，该对象没有Namespace限制，官网示例：

```yaml
apiVersion: scheduling.k8s.io/v1beta1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used
```

然后通过在Pod的spec.priorityClassName中指定已定义的PriorityClass名称即可：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Pod抢占

当节点没有足够的资源供调度器调度Pod、导致Pod处于pending时，抢占（preemption）逻辑会被触发。Preemption会尝试

Pod优先级（Priority）和抢占（preemption）具体介绍可参见：https://kubernetes.io/docs/con ... tion/。

多调度器及调度器配置

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: my-pod
  name: my-pod
  namespace: default
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: my-pod
    ports:
    - containerPort: 80
      protocol: TCP
  schedulerName: my-custom-scheduler
```

使用scheduleName指定调度器

- 适用场景：
  - 集群中存在多个调度器，分别处理不同类型的作业调度
- 使用限制：
  - 建议对node做资源池划分，避免调度结果写入冲突

多调度器及调度器配置

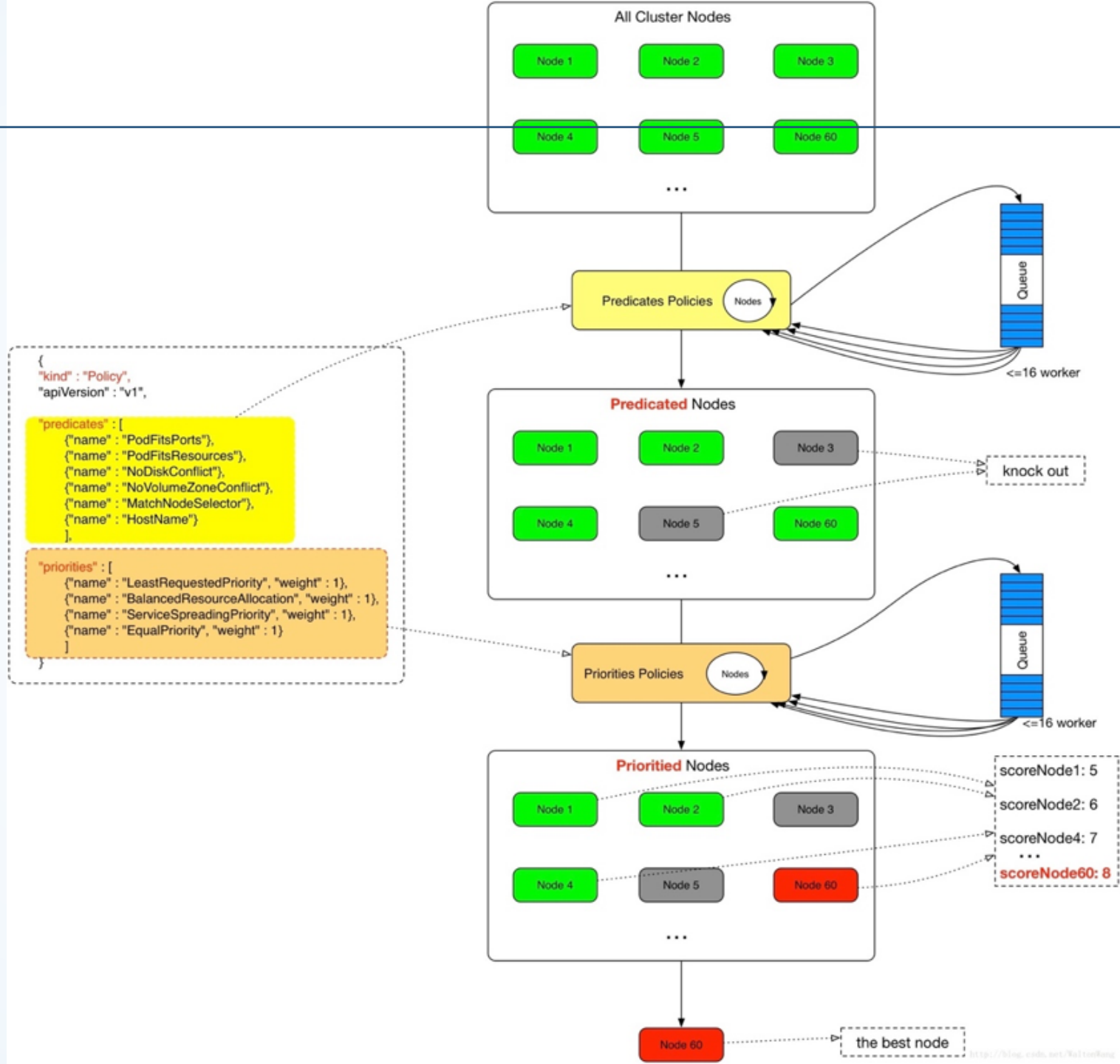
- --policy-config-file自定义调度器加载的算法，或者调整排序算法权重

```
{
  "kind" : "Policy",
  "apiVersion" : "v1",
  "predicates" : [
    {"name" : "PodFitsHostPorts"},
    {"name" : "PodFitsResources"},
    {"name" : "NoDiskConflict"},
    {"name" : "NoVolumeZoneConflict"},
    {"name" : "MatchNodeSelector"},
    {"name" : "HostName"}
  ],
  "priorities" : [
    {"name" : "LeastRequestedPriority", "weight" : 1},
    {"name" : "BalancedResourceAllocation", "weight" : 1},
    {"name" : "ServiceSpreadingPriority", "weight" : 1},
    {"name" : "EqualPriority", "weight" : 1}
  ],
  "hardPodAffinitySymmetricWeight" : 10,
  "alwaysCheckAllPredicates" : false
}
```

执行 kube-scheduler --help 查看更多调度器配置项

调度策略回顾

使用 workQueue 来并行运行检查，并发数最大是 16。对应源码示例：workqueue.Parallelize(16, len(nodes), checkNode)。



课程回顾

已学知识要点

学习Kubernetes的调度策略，使用自定义调度器，以及调度失败如何定位