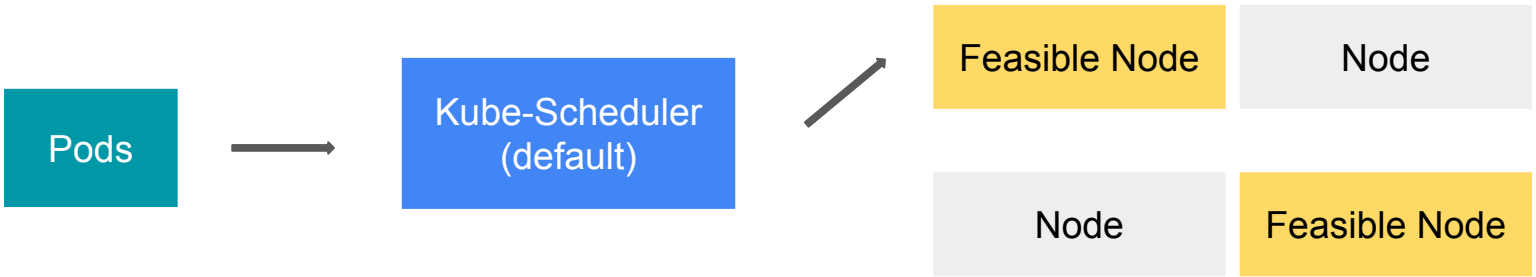
Scheduling, Preemption and Eviction

Topic Study

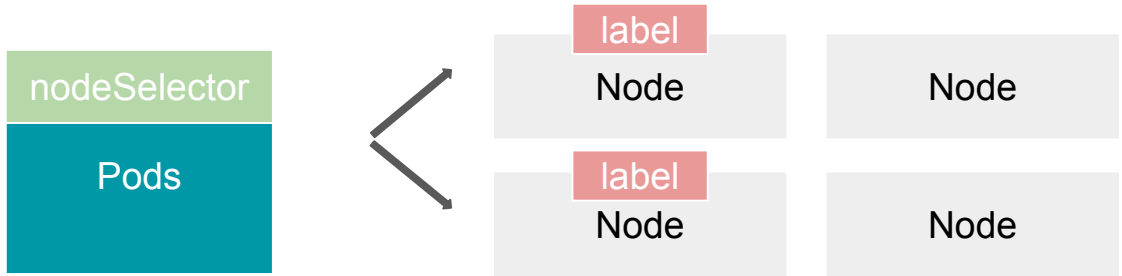
Scheduler

- 功能:觀看新產生但尚未被 assigned 給 Node 的 Pods, 並為 Pod 找到最佳的 Node 給 Pod
- Kube-Scheduler
  - 為 Kubernetes 的 default scheduler
  - Pods 中的 Container 有不同的需求, 其選取一個 Optimal 的 Node 給 Pods 跑在上面
  - Feasible Nodes: 滿足 Scheduling Requirements 的 Node, 如果都沒有合適的 Node, 則 Pod 會保持 Unscheduled 的狀態



Assign Pods to Nodes

- User 可以限制 Pod 跑在特定幾個 Nodes 之上
- 在 Nodes 上面加 Labels, 讓 Pods 可以針對有被 Label 到的 Node 做 Scheduling
- 讓 Pods 跑在安全且具有監控管理的 Labeled Nodes 上面



Node selection constraint - nodeSelector

- 加在 Pod specification上面, 使得 Pod 可以指定想要的 Node Labels
- 以下圖為例, 此 Pod 需要找到一個具有 disk=ssd 的 Node 才可以運行

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodehelloworld.example.com
  labels:
    app: helloworld
spec:
  containers:
  - name: k8s-demo
    image: 105552010/k8s-demo
    ports:
    - name: nodejs-port
      containerPort: 3000
  nodeSelector:
    disk: ssd
```

Node selection constraint - Affinity

- 相較於 nodeSelector 透過 Labels 來限制選取的 Nodes, Affinity 與 anti-affinity 可以使用的操作會更多
- NodeAffinity

Property	特性	關於
requiredDuringSchedulingIgnoredDuringExecution	硬規定	如果沒有滿足此條件，則沒有辦法 Schedule Pod
preferredDuringSchedulingIgnoredDuringExecution	軟規定	Scheduler 會盡量找到滿足此 Rule 的 Node，但如果沒有可用符合此條件的 Node，仍會 Schedule Pod

Node selection constraint - nodeAffinity

- Node 必須要有以下 key-value pair
  - key: kubernetes.io/os
  - value: linux
- weight：設置於 preferredDuringSchedulingIgnoredDuringExecution，1~100 的數值，從滿足 Preferred 的條件裡面 Sum 這些 weight，最高分的 Node 有較高的優先權

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/os
            operator: In
            values:
            - linux
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
```

Node selection constraint - Inter-pod affinity

- 限制 Pods 可以跑在哪些 Nodes 中，此規則基於在此 Nodes 中的 Pods 上的 Labels
- Pod 必須跑在一個 X 中，在這個 X 中的 Pods 皆符合某 Rule Y
  - X：可以為 Node、Rack、Cloud Provider Zone 或 Region
  - Y：為 Kubernetes 嘗試滿足的規則

```
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: topology.kubernetes.io/zone
    podAntiAffinity:
```

Node selection constraint - Taints and Tolerations

- Node affinity vs Taint
  - node affinity: 設計如何讓某個pod被分配到某個worker node
  - taint: 設計如何pod**不要**被分配到某個worker node
- Taint attribute
  - key
  - Value
  - Effect: 共有三種
    - NoSchedule: **不會**把該 pod 分派到該 node 上，但不影響正在運作中的 pod
    - PreferNoSchedule: **儘量不會**把該 pod 分派到該 node 上(最後要是沒辦法的時候還是會破功)
    - NoExecute: 已經存在該 node 上的 pod 趕走，也不會把該 pod 分派到該 node 上
- Pod toleration: 允許(但不要求)Pod調度到某個worker node帶有與之匹配的taint

Node selection constraint - Toleration 特殊案例

- 表示可以接受"帶有 key=value & effect=NodeSchedule”的taint

```
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoSchedule"
```

- 僅有設定operator為Exists，卻沒有設定key，那表示會tolerate所有的taint

```
tolerations:
- operator: "Exists"
```

- 僅有設定key，沒有設定effect，表示只帶有相同key的taint都會被tolerate

```
tolerations:
- key: "key"
  operator: "Exists"
```

Pod Overhead

- Why we need it?
  - To run an application smmothly within the container, we need to **provide required resources** to the container in terms of CPU and memory.
- Example of defining Pod Overhead

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
spec:
  runtimeClassName: demo-rc
  containers:
  - name: nginx
    image: nginx:alpine
    resources:
      limits:
        cpu: 250m
        memory: 100Mi
```

Pod Priority

- 每個 Pods 被設置優先權
- 當有 Pod 無法被 Schedule 時，Scheduler 會搶奪低優先權的 Pods
- 有些 Pods 可能是惡意的 User 創造出來的，把該 Pod 賦予較高的優先權，管理者可以使用 ResourceQuota 來避免 Users 建立高優先權的 Pods
- 操作方式：先新增一個或多個 PriorityClasses，建立 Pods 時，設置 priorityClassName

PriorityClass

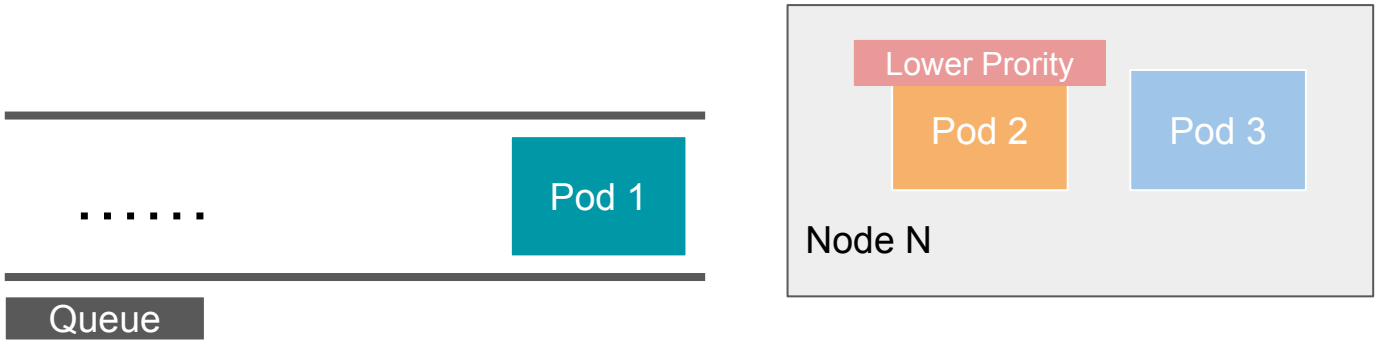
- 為一個 non-namespaced 的 Object，將 priorityClassName map 到一個 value(Priority)
- Value 越高，Priority 越高
- 有 2 個 Optional 的欄位可以設定
  - globalDefault：Boolean，設置為 True 時，沒有設定 priorityClassName 的 Pods 會使用這個 Value，設置成 False 時，沒有設置 priorityClassName 的 Pods 會變成 0
  - description：String，告訴 Users 甚麼時候要使用 Priority Class

```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
```

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

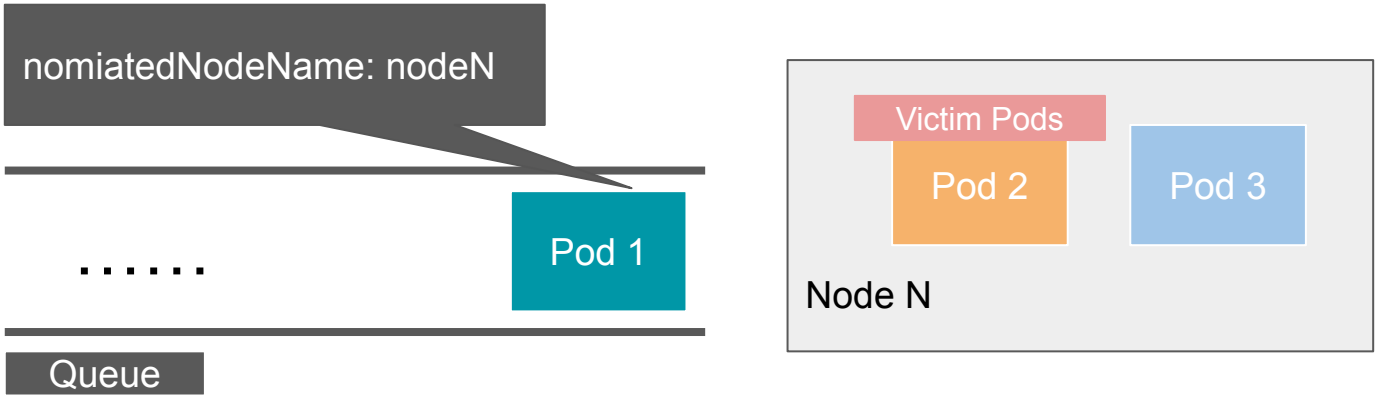
Preemption

- Pods 建立時, 在 Queue 之中等待被 scheduled, 如果沒有合適的 node 則做 Preemption, 尋找比該 pod 低優先權的 pods 運行的 node, 找到該 nodes, 丟棄較低優先權的 pod, 則優先權較高的 Pod 可以執行在 node 上面
- 當優先權較高的 Pod 1, 搶了 Pod 2, 則 Pod 1 狀態中的 nominatedNodeName 欄位, 會設成 Node N, 這個欄位可以讓 Scheduler 追蹤為 P 保留的資源且讓 user 知道 Cluster 中的 Preemption 狀況



Preemption

- nominatedNodeName = Node N 不代表 Pod 1 一定要跑在 Node N 上面, 因為必須等待 其他 Victim Pods (Pod 2) 在 Node N 上面先做結束, 但是如果過程中有其他 Node 是可以利用的, 則 Pod 1 就會配給它, 所以 nominatedNodeName 和 nodeName 並不總是相同的



Preemption

- Graceful termination of victims：會給 Victim 一個緩衝時間, 讓他們完成原本的工作並結束掉, 如果在其間還是無法完成, 則會被 killed. 從 Pod 1 被 scheduled 到 Node N 上面到真正可以執行, 中間會有一個 Gap 的時間, 可以透過降低 Low priority Pod(Pod 2) 的 graceful termination period, 來降低 gap 時間的長度
- Inter-Pod affinity on lower priority Pods：如果 Pending 中的 Pod 與其他的低優先權 Pods 有 inter-pod affinity, 則從 Node N 上面移除 Pods 會導致無法滿足 Inter-Pod affinity
  - 改進方式：建議只在更高優先權或相等優先權的 Pods 上面設置 inter-Pod affinity
- Cross Node Preemption：Pod 1 被指定給 node N, Pod 2 在與 node N 同一個 Zone 的 Node 上面執行, Pod 1 與 Pod 2 有 zone-wide anti-affinity, Pod 2 應該也要被 Preempted 掉, 但是因為沒有 Cross Node Preemption, Pod 1 會被當成 unschedulable on Node N

Node-Pressure Eviction

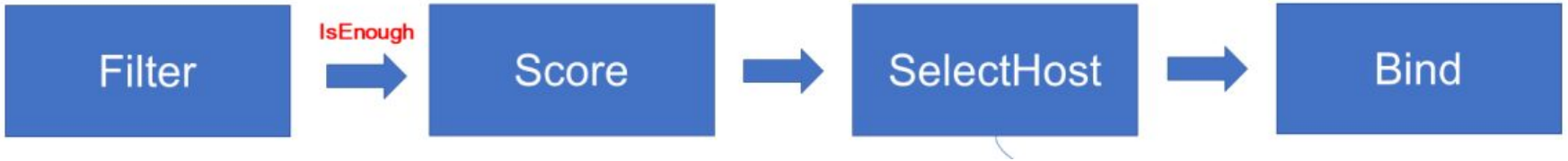
- kubelet 主動結束 Pod, 來回收 node 上面的 resources
- kubelet 會 monitor resources: Cluster 裡面的 Nodes 中的 CPU, Memory, Disk space, filesystem inodes, 當其中一個資源達到特定的消耗等級, kubelet 會主動地 fail 一個或多個 pods on the Node 來回收資源, 避免 Starvation, kubelet 會將該要被丟棄的 pod 的 PodPhase 設成 Failed, 導致 Pod terminates

Node selection in kube-scheduler

kube-scheduler 為 Pod 選擇一個 Node 會經過兩個步驟分別是 **Filtering、Scoring**

K8s v1.2.3 前可以透過以下指令設定特定的 Predicates for filtering、Priorities for scoring。

```
~$ kube-scheduler --policy-config-file <filename>
~$ kube-scheduler --policy-configmap <ConfigMap>
```



Node selection in kube-scheduler: Filtering

- Filtering 階段會透過一系列 Filtering **過濾出適合調度 Pod 的 Node**。
- 以 **PodFitsResource** filter 為例，PodFitsResources 會篩選出 Node 是否有足夠的資源調度這個 Pod（包括：CPU / Memory / GPU … 等）。
  - 具體**使用情境**：將負責 ML 訓練的 Pod 調度到有 GPU 的 Node。
- K8s 有一系列的預設 filter 包括：PodFitsPorts、PodFitsResources、NoDiskConflict、PodFitsHost … 等等。
- 經過 Filtering 階段後會篩選出一批 Node，供 Socring 進一步處理。如果這個階段篩選後沒有適合的 Node 代表現在沒辦法為這個 Pod 做調度。

Node selection in kube-scheduler: Scoring

- Scoring 階段會根據 Filtering 階段篩出來的 Node 進一步針對各個 **Node 去做調度的評分、加權**。
- 以 **LeastRequestedPriority** 為例，會根據 Node 資源使用情況，將 Pod 調度到負載較輕的 Node。也就是說資源較為空閒的 Node 會有較高的 score。
  - 參考公式：`cpu((capacity - sum(requested)) * 10 / capacity) + memory((capacity - sum(requested)) * 10 / capacity) / 2`
- 最後 kube-scheduler 會根據 Scoring 算出的最終分數將 Pod 調度到分數最高的 Node。
- 如果有一個以上的 Node 分數一致的話，kube-scheduler 會用 random 的方式選擇要調度的 Node。

API-initiated Eviction

- API-initiated eviction 藉由使用Eviction API 創建Eviction object觸發pod termination。
  - 直接呼叫Eviction API要求eviction
  - 使用 API server的client（如 kubectl drain 命令)
- API-initiated evictions必須遵照PodDisruptionBudgets和terminationGracePeriodSeconds配置。

API-initiated Eviction

- Calling the Eviction API
  - 方法一:
    - POST the attempted operation
    - 使用 Kubernetes language client 存取Kubernetes API，並創建一個 Eviction object.
  - 方法二:
    - 通過使用 curl 或 wget訪問API來嘗試eviction operation.

```
{
  "apiVersion": "policy/v1",
  "kind": "Eviction",
  "metadata": {
    "name": "quux",
    "namespace": "default"
  }
}
```

```
curl -v -H 'Content-type: application/json' https://your-cluster-api-endpoint.example/api/v1/namespaces/default/pods/quux/eviction -d @eviction.json
```

API-initiated Eviction

- How API-initiated eviction works
  - API server 的 admission checks 以及 responds:
    - 200 OK: eviction 被允許、Eviction subresource 被建置以及 Pod 被刪除
    - 429 Too Many Requests:eviction"目前"不被允許由於已配置的 PodDisruptionBudget.
    - 500 Internal Server Error:eviction 不被允許由於 misconfiguration，例如:多個 PodDisruptionBudgets 參考到相同的 pod.

Resource Bin Packing for Extended Resources

- kube-scheduler可以配置為使用RequestedToCapacityRatioResourceAllocation優先級函數啟用資源的 bin package以及擴展資源。
- Kubernetes 允許用戶指定資源以及每個資源的權重，以根據請求與capacity的比率對node進行評分.
- 這允許用戶通過使用適當的參數來打包擴展資源，並提高large clusters中稀缺資源的利用率.

Resource Bin Packing for Extended Resources

- Tuning the Priority Function
  - (左圖)如果利用率為 0%，則得分為 0，而利用率為 100% 時得分為 10。
  - (右圖) weight 參數是可選的，如果未指定，則設置為 1。此外，權重不能設置為負值。

```
shape:
 - utilization: 0
   score: 0
 - utilization: 100
   score: 10
```

```
resources:
  - name: intel.com/foo
    weight: 5
  - name: cpu
    weight: 3
  - name: memory
    weight: 1
```

Scheduling Framework

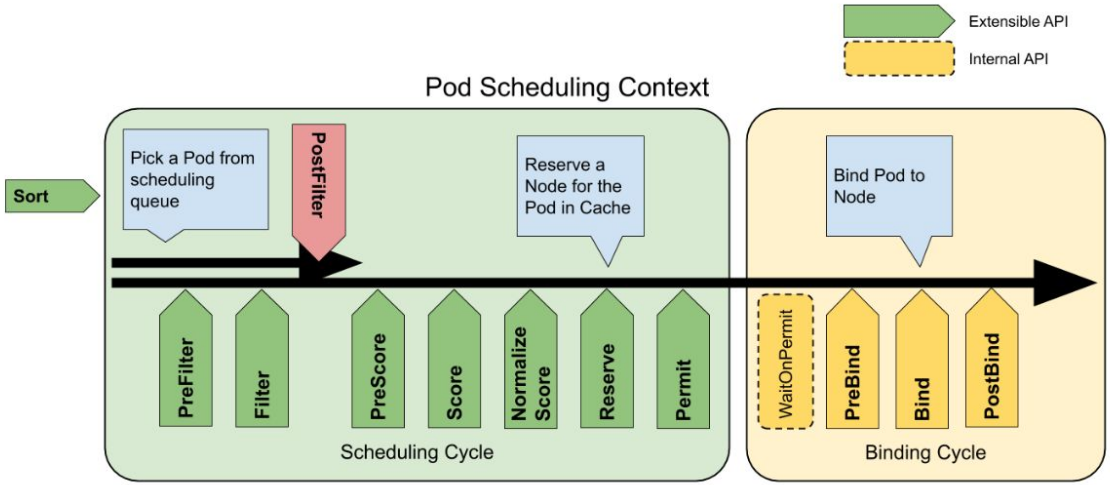
- 由於Kube scheduler基本上不會考慮一個 job 內各個 pod 的相關性，所以這幾年來也有不少 客製化的方法，而 k8s 在 v1.19 版本中提出了一個新的 Scheduling Framework
- 目標:
  - 使scheduler 更有擴充空間
  - 使 scheduler core 更簡單
  - 在 framework 中提出「extension points」

Scheduling Framework

- 此 framework 引進「plugin」的概念。提供開發人員在scheduler 流程的每個步驟 — Queueing、Filtering、Scoring、Binding，都可以加入 plugin，直接客製化 Kube scheduler的功能
- 會定義一些 extension points 讓用戶可以在 Plugin 上依照情境搭配多個 extension points使用

Scheduling Framework - Scheduling Context

- 每一次試圖去 schedule 一個 pod，分為兩個部分，scheduling cycle 以及 binding cycle
- 每個部分又分成好幾個 task，每一個 task 由一到多個 plugins 組合而成。
  - scheduling cycle: 選擇一個 node 給 pod， 串行運行
  - binding cycles: 將調度的結果通知 Cluster，可平行運行



Extension points - Scheduling Cycle

- QueueSort
  - 對在scheduling queue中的pod做排序，利用「Less(Pod1, Pod2)」來實現，一次只有一個可以運作
- PreFilter
  - 在 scheduling 前先對 cluster 或 pod 進行檢查，一個 scheduling 週期只會做一次，如果檢查失敗會中斷 scheduling
- Filter
  - 過濾不能運行Pod的節點
- PostFilter
  - 在 filter 後，若沒有找到任何的 worker node，會呼叫這個 plugin，去決定如何繼續

Extension points - Scheduling Cycle

- PreScore
  - 為了增加效能，此 plugin 可以先算出當下 scheduling 的運行狀況，並緩存在記憶體內，增加scoring 運算速度
- Score
  - 對應每個 worker node 算出在 [MinNodeScore, MaxNodeScore] 之間的分數，若不在此範圍內，則會abort
- NormalizeScore (optional)
  - 在結算 node 排名之前修改 node 分數，若不在[MinNodeScore, MaxNodeScore]內，會總和目前的 node score，依照configured plugin weights 來重新計算 score

Extension points - Scheduling Cycle

- Reserve/Unreserve：當 scheduler 替一個 pod 選定了一個 worker node 後，就會進行 binding。但binding 可能是一個需要時間的 operation，有兩個可能原因
  - Binding phase 還可能要等worker node 把資源配置出來
  - 該 pod 需要等另外一個 pod 被 schedule 好後一起dispatch
- 由於上述兩個原因，才會出現 Reserve/Unreserve
  - Reserve：用於在 pod 綁定一個 node 後，保留 worker node，等被其他scheduler 考慮，避免race condition
  - Unreserve: 留給Binding 不成功的狀況下，把目標worker node釋放出來

Extension points - Scheduling Cycle

- Permit：為 scheduling cycle 的最後一個步驟，可做 approve、deny、wait
  - approve : 送去 binding
  - deny : 會回到 scheduling queue，會在 Reserve plugin 中 trigger「Unreserve method」
  - wait (with a timeout): 此 pod 會保持在 permit phase 直到 plugin approve 若發生 timeout，wait 會變成 deny

Extension points - Binding Cycle

- Prebind
  - 有一些 pod 被dispatch 到 worker node 前要做的所有動作
  - ex: 規定 network volume 並 mount 至 target node
- Bind
  - Bind 插件會被呼叫，使用者可以決定這個worker node assign 給 pod 之後，是否需要呼叫 post-bind 插件
- PostBind
  - 在 binding 成功後要做某一些 cleanup

Scheduler Configuration / Profile

- K8s 可以透過 scheduling Profile 去設定不同的 Stage 的行為（K8s 將他抽象為 Extension points）
- 針對每一個 Extension points 可以 disable/enable 特定的 plugin。下面這個範例就是將 **MyCustomPluginA、MyCustomPluginB** 加入 default scheduler 的 **Score extension point** 並且把 PodTopologySpread disable 掉

```
apiVersion: kubescheduler.config.k8s.io/v1beta2
kind: KubeSchedulerConfiguration
profiles:
  - plugins:
      score:
        disabled:
        - name: PodTopologySpread
        enabled:
        - name: MyCustomPluginA
          weight: 2
        - name: MyCustomPluginB
          weight: 1
```

Scheduler Configuration / Profile

- 此外，k8s 也有提供在一個 plugin 中可以應用多個 extension points
- 若 MyPlugin 實現了 preScore、score、preFilter、filter extension points，若想要 MyPlugin 能夠應用在所有的extension point 上，則 profile config 會長成以下形式，在 plugins下方加上**multiPoint**，代表把 **MyPlugin** 加入 **multipoint-scheduler** 的所有 extension point 中

```
apiVersion: kubescheduler.config.k8s.io/v1beta3
kind: KubeSchedulerConfiguration
profiles:
  - schedulerName: multipoint-scheduler
    plugins:
      multiPoint:
        enabled:
        - name: MyPlugin
```

Scheduler Performance Tuning

- 如果 K8s 有數千個 Node 的話，這些 Node 又剛好可以通過 Filtering 成為 feasible node，這樣 kube-scheduler 每次都需要針對數千個 Node 做 Scoring，對大規模的 K8s 來說會是一種負擔（overhead）。
- 可以透過 **percentageOfNodesToScore** 去調整要被調度的 Node 數量的閾值（threshold）
  - 換句話說可以針對 **latency**（Pod 調度的速度）、**accuracy**（資源分配的準確性）去做 trade-off
- percentageOfNodesToScore 是一個 0 - 100 的值，代表要被 feasible node 的百分比，K8s 會自動調整這個值，當 K8s Cluster 少於 100 個 Node 這個值預設是 50，超過 5000 個 Node 則是 10%，K8s 自動調整的 lower bound 是 5%。

Scheduler Performance Tuning (cont.)

下面是一個 percentageOfNodesToScore 調整為 50% 的範例

```
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
algorithmSource:
    provider: DefaultProvider
…
percentageOfNodesToScore: 50
```

Conclusion

結論

- Scheduler 是為了可以讓 Pods 對應到適合的 Nodes 上面執行
- Preemption 使得較高優先權的 Pods 可以搶奪較低優先權的 Pods, 並且執行在 Nodes 上面
- Eviction 主動地結束掉跑在資源缺乏的 Nodes 上面的 Pods
- K8S 使用以上三個概念, 讓任務執行更有效率, 也能依據任務優先權高低, 讓重要任務優先執行, 達到 K8S 最佳化

Insights

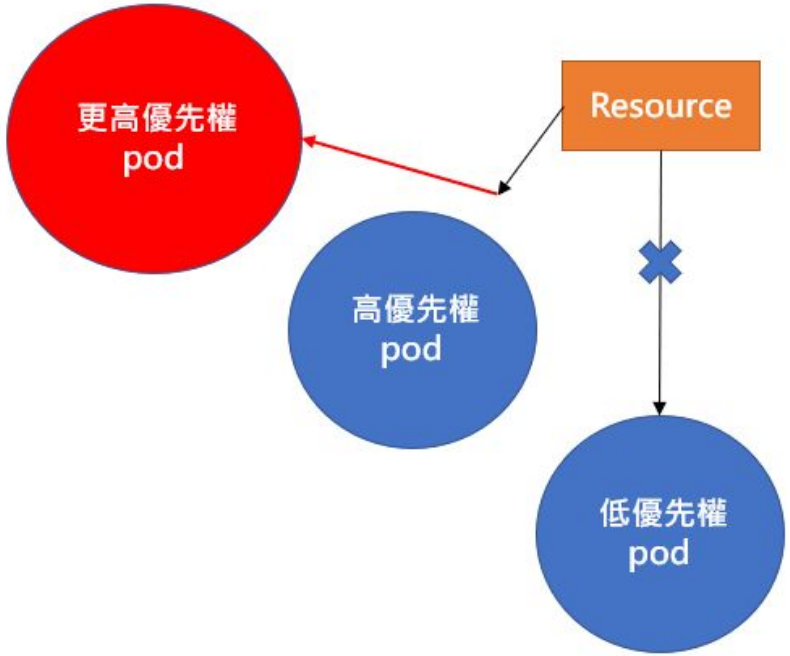
Issue 1

- Pods are preempted unnecessarily
  - preempt 是發生在資源不夠時，分配資源給高優先權的 pod 使用。
  - 可透過人為設置 priorityClassName 改變 pod 的 priority。
- 解決方法
  - 人為設置的 priority 皆設定較低優先權。
  - 優先權留空，優先權將默認為 0。

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

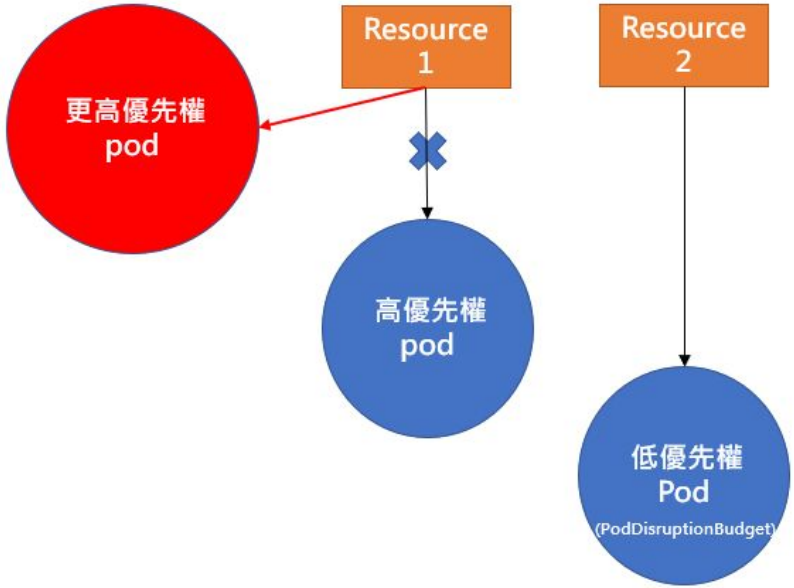
Issue 2

- Pods are preempted, but the preemptor is not scheduled
  - 有可能發生高優先權搶佔資源後，資源卻不是高優先權pod使用。
- 解決方法
  - 此情況發生在高優先權 pod 搶到資源後，有更高優先權的 pod 插隊搶資源，那就把資源再讓給更高優先權資源。



Issue 3

- Higher priority Pods are preempted before lower priority pods
  - 有可能較高優先權 pod 被迫放棄資源，給其他pod使用。
- 解決方法
  - 此情況發生在不能搶佔低優先權具有PodDisruptionBudget pod 的資源，所以只能選比較高優先權的 pod 逼迫它放棄資源。
  - 但比較高優先權 pod 的優先權仍然要低於搶佔 pod。



Contribution

id	Name	Content
310551129	許蔓萍	Topic Study
310551171	游朝荃	Topic Study
310551098	林和俊	Topic Study
310554029	陳淞榆	Topic Study
310554031	葉詠富	Insights
309551005	王麒銘	Topic Study