Scheduling

Contents

NodeName

- the scheduler ignores the pod and the kubelet running on the named node tries to run the pod.
- Limitations:
    - If the named node does not exist, the pod will not be run, and in some cases may be automatically deleted.
    - If the named node does not have the resources to accommodate the pod, the pod will fail and its reason will indicate why, e.g. OutOfmemory or OutOfcpu.
    - Node names in cloud environments are not always predictable or stable.

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

Node Selector

- For the pod to be eligible to run on a node, the node must have each of the indicated key-value pairs as labels (it can have additional labels as well). The most common usage is one key-value pair.

```
 apiVersion: v1
kind: Pod
metadata:
 name: nginx
 labels:
   env: test
spec:
 containers:
 - name: nginx
   image: nginx
   imagePullPolicy: IfNotPresent
 nodeSelector:
   disktype: ssd
```

Node Restriction

- Node labels can be updated by kubelet
- Compromised node and kubelet process can update label values and affect scheduling..
- The **NodeRestriction admission plugin prevents kubelets from setting or modifying labels with a node-restriction.kubernetes.io/ prefix**. To make use of that label prefix for node isolation:
- Example:

example.com.node-restriction.kubernetes.io/fips=true or
example.com.node-restriction.kubernetes.io/pci-dss=true

Affinity

Provides below benefits over nodeSelector

1. the language is **more expressive (not just “AND of exact match”)**
2. you can indicate that the rule is **“soft”/“preference” rather than a hard requirement**, so if the scheduler can’t satisfy it, the pod will still be scheduled

3. you can **constrain against labels on other pods running on the node (or other topological domain), rather than against labels on the node itself**, which allows rules about which pods can and cannot be co-located

4. Node affinity is specified as field `nodeAffinity` of field `affinity` in the PodSpec.

Types:

- **requiredDuringSchedulingIgnoredDuringExecution**
  - **Hard requrement**
  - **rules that *must* be met for a pod to be scheduled onto a node (just like nodeSelector but using a more expressive syntax),**
- **preferredDuringSchedulingIgnoredDuringExecution**
  - **Soft requirement**
  - **scheduler will try to enforce but will not guarantee.**

1. **The "IgnoredDuringExecution" part of the names means that, similar to how nodeSelector works, if labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod will still continue to run on the node**

- If you specify both nodeSelector and nodeAffinity, *both* must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple nodeSelectorTerms associated with nodeAffinity types, then the pod can be scheduled onto a node **if one of** the nodeSelectorTerms is satisfied.
- If you specify multiple matchExpressions associated with nodeSelectorTerms, then the pod can be scheduled onto a node **only if all** matchExpressions can be satisfied.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
```

```
      preference:
        matchExpressions:
        - key: another-node-label-key
          operator: In
          values:
          - another-node-label-value
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
```

Anti-affinity

- allow you to constrain which nodes your pod is eligible to be scheduled ***based on labels on pods*** *that are already running on the node* **rather than based on labels on nodes**.
- The rules are of the form “this pod should (or, in the case of anti-affinity, should not) run in an X if that X is already running one or more pods that meet rule Y”. Y is expressed as a LabelSelector with an optional associated list of namespaces; unlike nodes,
- Inter-pod affinity is specified as field `podAffinity` of field `affinity` in the PodSpec
- And inter-pod anti-affinity is specified as field `podAntiAffinity` of field `affinity`
- `Types`
    - `requiredDuringSchedulingIgnoredDuringExecution`
    - `preferredDuringSchedulingIgnoredDuringExecution`

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
```

```
          values:
          - S1
      topologyKey: failure-domain.beta.kubernetes.io/zone
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S2
        topologyKey: failure-domain.beta.kubernetes.io/zone
containers:
- name: with-pod-affinity
  image: k8s.gcr.io/pause:2.0
```

- The pod affinity rule says that the pod can be scheduled onto a node only if that node is in the same zone as at least one already-running pod that has a label with key “security” and value “S1”. (More precisely, the pod is eligible to run on node N if node N has a label with key `failure-domain.beta.kubernetes.io/zone` and some value V such that there is at least one node in the cluster with key `failure-domain.beta.kubernetes.io/zone` and value V that is running a pod that has a label with key “security” and value “S1”.)
- The pod anti-affinity rule says that the pod prefers not to be scheduled onto a node if that node is already running a pod with label having key “security” and value “S2”. (If the `topologyKey` were `failure-domain.beta.kubernetes.io/zone` then it would mean that the pod cannot be scheduled onto a node if that node is in the same zone as a pod with label having key “security” and value “S2”.)
- Go through the example for co- locating web and cache.. and ensuring web/redis replicas scheduled on different nodes..

https://kubernetes.io/docs/concepts/configuration/assign-pod-node/

Taints and tolerations

- One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.
- Add taint
    - `kubectl taint nodes node1 key=value:NoSchedule`

- Remove taint
    - `kubectl taint nodes node1 key:NoSchedule-`
- Matching tolerations on POD

A toleration “matches” a taint if the keys are the same and the effects are the same, and:

- the `operator` is `Exists` (in which case no `value` should be specified), or
- the `operator` is `Equal` and the `value`s are equal
- `Operator` defaults to `Equal` if not specified.

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

Special cases:

- An empty `key` with operator `Exists` matches all keys, values and effects which means this will tolerate everything.

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

- An empty `effect` matches all effects with key `key`.

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

Effect Types”

- NoSchedule
    - No pods will be scheduled on a node unless matching tolerations
- PreferNoSchedule
    - The system will *try* to avoid placing a pod that does not tolerate the taint on the node, but it is not required.
- NoExecute
    - pod will be evicted from the node (if it is already running on the node)

Example Use cases :

- Dedicated nodes - want to dedicate a set of nodes for exclusive use by a particular set of users
    - `kubectl taint nodes nodename dedicated=groupName:NoSchedule`
- **Nodes with Special Hardware**
    - In a cluster where a small subset of nodes have specialized hardware (for example GPUs), it is desirable to keep pods that don't need the specialized hardware off of those nodes, thus leaving room for later-arriving pods that do need the specialized hardware. This can be done by tainting the nodes that have the specialized hardware (e.g. `kubectl taint nodes nodename special=true:NoSchedule` or `kubectl taint nodes nodename special=true:PreferNoSchedule`

Taint based evictions

- Using NoExecute..

Taint nodes by condition

- `TaintNodesByCondition`
  - node lifecycle controller automatically creates taints corresponding to Node conditions

- `node.kubernetes.io/memory-pressure`
- `node.kubernetes.io/disk-pressure`
- `node.kubernetes.io/out-of-disk` (*only for critical pods*)
- `node.kubernetes.io/unschedulable` (1.10 or later)
- `node.kubernetes.io/network-unavailable` (*host network only*)

Resource requests and Limits

Each Container of a Pod can specify one or more of the following:

- `spec.containers[].resources.limits.cpu`
- `spec.containers[].resources.limits.memory`
- `spec.containers[].resources.requests.cpu`
- `spec.containers[].resources.requests.memory`

Although requests and limits can only be specified on individual Containers, it is convenient to talk about Pod resource requests and limits. A *Pod resource request/limit* for a particular resource type is the sum of the resource requests/limits of that type for each Container in the

- CPU is specified in units of cores. Fractional requests are allowed.
  - `spec.containers[].resources.requests.cpu`
  - `.1 or 100m -` “one hundred millicpu”, or “one hundred millicpu”
  - `Default –` 100m
  - `Minimum –` 1m
- memory is specified in units of bytes.
  - 128 MiB

If a Container exceeds its memory limit, it might be terminated. If it is restartable, the kubelet will restart it, as with any other type of runtime failure.

If a Container exceeds its memory request, it is likely that its Pod will be evicted whenever the node runs out of memory.

A Container might or might not be allowed to exceed its CPU limit for extended periods of time. However, it will not be killed for excessive CPU usage.

With docker

- `docker run --cpu-shares`
- `docker run --memory`

Request and Limits for local ephemeral storage

- `spec.containers[].resources.limits.ephemeral-storage`
- `spec.containers[].resources.requests.ephemeral-storage`

```
resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
```

Extended resources.
TODO…

Daemon Sets

- A *DaemonSet* ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them
- Example - fluentd, promethus

Scheduled by DaemonSet controller (disabled by default since 1.12)

Normally, the machine that a Pod runs on is selected by the Kubernetes scheduler. However, Pods created by the **DaemonSet controller have the machine already selected** (.spec.nodeName is specified when the Pod is created, so it is ignored by the scheduler). Therefore:

The unschedulable field of a node is not respected by the DaemonSet controller

Scheduled by default scheduler (enabled by default since 1.12)

ScheduleDaemonSetPods allows you to schedule DaemonSets using the default scheduler instead of the DaemonSet controller, by adding the NodeAffinity term to the DaemonSet pods, instead of the .spec.nodeName term. The default scheduler is then used to bind the pod to the target host. If node affinity of the DaemonSet pod already exists, it is replaced. The DaemonSet controller only performs these operations when creating or modifying DaemonSet pods, and no changes are made to the spec.template of the DaemonSet.

```
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchFields:
      - key: metadata.name
        operator: In
        values:
        - target-host-name
```

In addition, node.kubernetes.io/unschedulable:NoSchedule toleration is added automatically to DaemonSet Pods. The default scheduler ignores unschedulable Nodes when scheduling DaemonSet Pods

https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

Pod Priority

- Priority indicates the importance of a Pod relative to other Pods. If a Pod cannot be scheduled, the scheduler tries to preempt (evict) lower priority Pods to make scheduling of the pending Pod possible.
- Priority also **affects scheduling order** of Pods and **out-of-resource eviction** ordering on the Node.

Priority Class

- A PriorityClass is a non-namespaced object that defines a mapping from a priority class name to the integer value of the priority. The name is specified in the `name` field of the PriorityClass object's metadata. The value is specified in the required `value` field. The higher the value, the higher the priority.

- `globalDefault` and `description`. The `globalDefault` field indicates that the value of this PriorityClass should be used for Pods without a `priorityClassName`. Only one PriorityClass with `globalDefault` set to true can exist in the system.
- If there is no PriorityClass with `globalDefault` set, the priority of Pods with no `priorityClassName` is zero.

```
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
only."


apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent

  priorityClassName: high-priority
```