Node Selector, Node Affinity Taints & Tolerations, Pod Affinity/Anti-Affinity

Kubernetes scheduler ensures that the right node is selected by checking the node's capacity for CPU and RAM and comparing it to the Pod's resource requests. The scheduler makes sure that, for each of these resource types, the sum of all resource requests by the Pods' containers is less than the capacity of the node. This mechanism ensures that Pods end up on nodes with spare resources.

Kubernetes scheduler's default behaviour works well for most cases -- for example, it ensures that pods are only placed on nodes that have sufficient free resources, it ties to spread pods from the same set (ReplicaSet, StatefulSet, etc.) across nodes, it tries to balance out the resource utilization of nodes, etc.

However, there are some scenarios when you want your Pods to end up on specific nodes. For example:

- You want your Pod(s) to end up on a machine with the SSD attached to it.
- You want to co-locate Pods on a particular machine(s) from the same availability zone.
- You want to co-locate a Pod from one Service with a Pod from another service on the same node because these Services strongly depend on each other. For example, you may want to place a web server on the same node as the in-memory cache store like Memcached.

Node Selector

nodeSelector — This is a simple Pod scheduling feature that allows scheduling a Pod onto a node whose labels match the nodeSelector labels specified by the user.

The basic idea behind the nodeSelector is to allow a Pod to be scheduled only on those nodes that have label(s) identical to the label(s) defined in the nodeSelector . nodeSelector labels are key-value pairs that can be specified inside the PodSpec.

Check existing labels:

```
kubectl get nodes --show-labels
```

Add new labels to node:

Next, select a node to which you want to add a label. Use below command to add label to node.

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

ex:

```
kubectl label nodes ip-172.10.43.76 name=WorkerOne
```

In order to assign a Pod to the node with the label we just added, you need to specify a nodeSelector field in the PodSpec. You can have a manifest that looks something like this:

```
spec:
  nodeSelector:
    name: Workerone
```

Kubernetes also offers advanced scheduling features: node affinity ,taints and tolerations.

Node Affinity

As we've mentioned earlier, nodeSelector is the simplest Pod scheduling constraint in Kubernetes. The affinity greatly expands the nodeSelector functionality introducing the following improvements:

1. Affinity language is more expressive (more logical operators to control how Pods are scheduled).
2. Users can now "soft" scheduling rules. If the "soft" rule is not met, the scheduler can still schedule a Pod onto a specific node.

- Node affinity is a way to set rules based on which the scheduler can select the nodes for scheduling workload. Node affinity can be thought of as opposite of taints. Taints repel a certain set of nodes whereas node affinity attract a certain set of nodes.
- NodeAffinity is a generalization of *nodeSelector*. In *nodeSelector*, we specifically mention which node the pod should go to, using node affinity we specify certain rules to select nodes on which pod can be scheduled.
- These rules are defined by labeling the nodes and having pod spec specify the selectors to match those labels. There are 2 types of affinity rules. **Preferred rules and Required rules.**
- In *Preferred rule*, a pod will be assigned on a non-matching node if and only if no other node in the cluster matches the specified labels. ***preferredDuringSchedulingIgnoredDuringExecution*** is a preferred rule affinity.
- In Required rules, if there are no matching nodes, then the pod won't be scheduled. There are a couple of require rule affinities namely requiredDuringSchedulingIgnoredDuringExecution and requiredDuringSchedulingRequiredDuringExecution.

- In ***requiredDuringSchedulingIgnoredDuringExecution*** affinity, a pod will be scheduled only if the node labels specified in the pod spec matches with the labels on the node. However, once the pod is scheduled, labels are ignored meaning even if the node labels change, the pod will continue to run on that node.

- In **requiredDuringSchedulingRequiredDuringExecution** affinity, a pod will be scheduled only if the node labels specified in the pod spec matches with the labels on the node and **if the labels on the node change in future, the pod will be evicted**. This effect is similar to NoExecute taint with one significant difference.
  When NoExecute taint is applied on a node, every pod not having a toleration will be evicted, whereas, removing/changing a label will remove only the pods that do specify a different label.

Use Cases:

- While scheduling workload, when we need to schedule a certain set of pods on a certain set of nodes but do not want those nodes to reject everything else, using node affinity makes sense.

NodeAffinity works on label matching. Let's label node1 as,

```
kubectl label nodes <nodeId/Name> node=workerone
kubectl get nodes --show-labels
```

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: "node"
          operator: In
          values:
          - workerone
```

```
affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
        matchExpressions:
         - key: name
           operator: In
           values:
           - workerone
```

Pod Affinity & Anti Affinity

- Node affinity allows you to schedule a pod on a set of nodes based on labels present on the nodes. However, in certain scenarios, we might want to schedule certain pods together or we might want to make sure that certain pods are never scheduled together. This can be achieved by *PodAffinity* and/or *PodAntiAffinity* respectively.
- Inter-pod affinity and 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.
- Similar to node affinity, there are a couple of variants in pod affinity namely *requiredDuringSchedulingIgnoredDuringExecution* and *preferredDuringSchedulingIgnoredDuringExecution*

Use cases

- While scheduling workload, when we need to schedule a certain set of pods together, *PodAffinity* makes sense. Example, a web server and a cache.

- While scheduling workload, when we need to make sure that a certain set of pods are not scheduled together, *PodAntiAffinity* makes sense.

```
affinity:
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: app
          operator: In
          values:
          - nginx
      topologyKey: "kubernetes.io/hostname"
```

```
affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: app
          operator: In
          values:
          - nginx
      topologyKey: "kubernetes.io/hostname"
```

Taints

Node affinity is a property of pods that attracts them to a set of nodes. Taints are the opposite, they allow a node to repel a set of pods.

Taint is a property of node that allows you to repel a set of pods unless those pods explicitly tolerates the said taint.

Taint has three parts. A key, a value and an effect.

For example,

kubectl taint nodes <nodeName> <key>=<value>:<effect>

kubectl taint nodes <nodeName> node=HatesPods:NoSchedule

The above taint has key=node, value=HatesPods and effect as NoSchedule. These key value pairs are configurable. Any pod that doesn't have a matching toleration to this taint will not be scheduled on node1.

To remove the above taint, we can run the following command

kubectl taint nodes <nodeName> HatesPods:NoSchedule-
What are some of the Taint effects?

- NoSchedule - Doesn't schedule a pod without matching tolerations
- PreferNoSchedule - Prefers that the pod without matching toleration be not scheduled on the node. It is a softer version of NoSchedule effect.
- NoExecute - Evicts the pods that don't have matching tolerations.

A node can have multiple taints. For example, if any pod is to be scheduled on a node with multiple *NoExecute* effect taints, then that pod must tolerate all the taints. However, if the set of taints on a node is a combination of *NoExecute* and *PreferNoExecute* effects and the pod only tolerates *NoExecute* taints then kubernetes will prefer not to schedule the pod on that node, but will do it anyway if there's no alternative.

Tolerations

Nodes are tainted for a simple reason, to avoid running of workload. Toleration is simply a way to overcome a taint.

For example, In the above section, we have tainted thisnode.compute.infracloud.io

To schedule the pod on that node, we need a matching toleration. Below is the toleration that can be used to overcome the taint.

tolerations:
- key: "node"

```
  operator: "Equal"
  value: "HatesPods"
  effect: "NoSchedule"
```

What we are telling kubernetes here is that, on any node if you find that there's a taint with key *node1* and its value is *HatesPods* then that particular taint should not stop you from scheduling this pod on that node.

Toleration generally has four parts. A key, a value, an operator and an effect. Operator, if not specified, defaults to *Equal*

Use cases

- Taints can be used to group together a set of Nodes that only run a certain set of workload, like network pods or pods with special resource requirement.
- Taints can also be used to evict a large set of pods from a node using taint with *NoExecute* effect.