Node Selectors

In this lecture, we will talk about node selectors in Kubernetes. Let us start with a simple example. You have a three node cluster of which two are smaller nodes with lower hardware resources, and one of them is a larger node configured with higher resources.

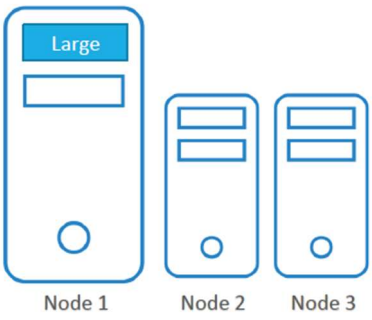
You have different kinds of workloads running in your cluster. You would like to dedicate the data processing workloads that require higher horsepower to the larger node as that is the only node that will not run out of resources in case the job demands extra resources. However, in the current default setup, any pods can go to any nodes. So, pod C in this case, may very well end up on nodes two or three, which is not desired.

To solve this, we can set a limitation on the pods so that they only run on particular nodes. There are two ways to do this. The first is using node selectors, which is the simple and easier method.

For this, we look at the pod definition file we created earlier. This file has a simple definition to create a pod with a data processing image. To limit this pod to run on the larger node, we add a new property called node selector to the spec section and specify the size as large.

pod-definition.yml

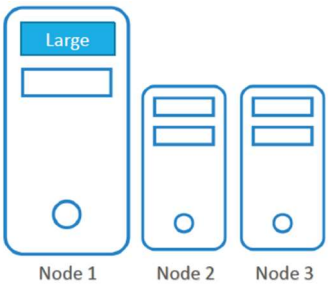
```
apiVersion:
kind: Pod
metadata:
 name: myapp-pod
spec:
  containers:
  - name: data-processor
    image: data-processor

  nodeSelector:
    size: Large
```



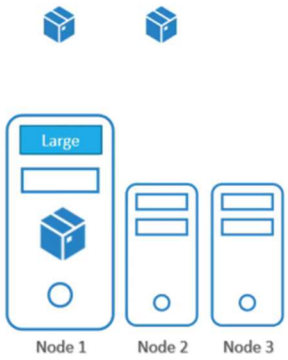
But wait a minute, where did the size large come from and how does Kubernetes know which is the large node? The key value pair of size and large are in fact labels assigned to the nodes. The scheduler uses these labels to match and identify the right node to place the pods on. Labels and selectors are a topic we have seen many times throughout this Kubernetes course such as with services, replica sets, and deployments. To use labels in a node selector like this, you must have first labeled your nodes prior to creating this pod.

So, let us go back and see how we can label the nodes. To label a node, use the command kubectl label nodes followed by the name of the node and the label in a key-value pair format. In this case, it would be kubectl label nodes node one followed by the label in a key-value format such as size=large.

```
kubectl label nodes <node-name> <label-key>=<label-value>
kubectl label nodes node-1 size=Large
```



Now that we have labeled the node, we can get back to creating the pod. This time with the node selector set to a size of large. When the pod is now created, it is placed on node one as desired.



pod-definition.yml

```
apiVersion:
kind: Pod
metadata:
 name: myapp-pod
spec:
  containers:
  - name: data-processor
    image: data-processor

  nodeSelector:
    size: Large
```

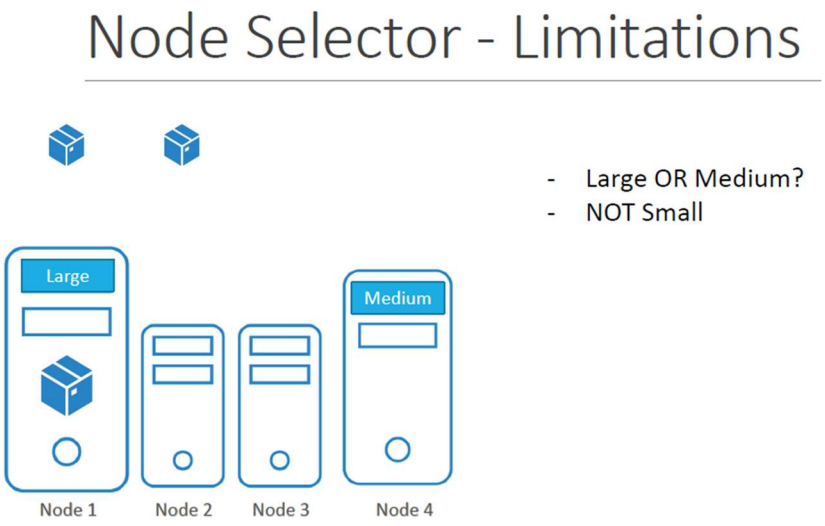
```
kubectl create -f pod-definition.yml
```

Node selectors served our purpose, but it has limitations. We used a single label and selector to achieve our goal here. But what if our requirement is much more complex? For example, we would like to say something like place the pod on a large or medium node, or something like place the pod on any nodes that are not small. You cannot achieve this using node selectors. For this, node affinity and anti-affinity features were introduced, and we will look at that next.

Node Affinity

Now we will talk about the node affinity feature in Kubernetes. The primary purpose of the node affinity feature is to ensure that pods are hosted on particular nodes, in this case to ensure the large data processing pod ends up on node one.

In the previous lecture, we did this easily using node selectors. We discussed that you cannot provide advanced expressions like "or" or "not" with node selectors. The node affinity feature provides us with advanced capabilities to limit pod placement on specific nodes.



With great power comes great complexity, so the simple node selector specification will now look like this with node affinity, although both do exactly the same thing: Place the pod on the large node.

pod-definition.yml

```
apiVersion:
kind: Pod

metadata:
 name: myapp-pod
spec:

  containers:
  - name: data-processor
    image: data-processor

  nodeSelector:
    size: Large
```

pod-definition.yml

```
apiVersion:
kind:

metadata:
 name: myapp-pod
spec:

 containers:
  - name: data-processor
    image: data-processor

 affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: size
          operator: In
          values:
          - Large
```

Let us look at it a bit closer. Under spec, you have affinity, and then node affinity under that. And then you have a property that looks like a sentence called "**required during scheduling, ignored during execution**." No description needed for that. And then you have the node selector terms, that is NRA, and that is where you will specify the key and value pairs. The key-value pairs are in the form key, operator, and value, where the operator is "in." The "in" operator ensures that the pod will be placed on a node whose label "size" has any value in the list of values specified here. In this case, it is just one called "large."

If you think your pod could be placed on a large or a medium node, you could simply add the value to the list of values like below.

You could use the NotIn operator to say something like, size not in small, where node affinity will match the nodes with a size not set to small.

pod-definition.yml

```
apiVersion:
kind:

metadata:
 name: myapp-pod
spec:

 containers:
  - name: data-processor
    image: data-processor

 affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: size
          operator: In
          values:
          - Large
          - Medium
```

pod-definition.yml

```
apiVersion:
kind:

metadata:
 name: myapp-pod
spec:

 containers:
  - name: data-processor
    image: data-processor

 affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: size
          operator: NotIn
          values:
          - Small
```

We know that we have only set the label "size" to large and medium nodes. The smaller nodes don't even have the label set, so we don't really have to check the value of the label. As long as we are sure we don't set a label "size" to the smaller node, using the "exists" operator will give us the same result. The "exists" operator will simply check if the label "size" exists on the nodes, and you don't need the "value" section for that as it does not compare the values.

pod-definition.yml

```
apiVersion:
kind:
metadata:
 name: myapp-pod
spec:
 containers:
  - name: data-processor
    image: data-processor
 affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: size
          operator: Exists
```

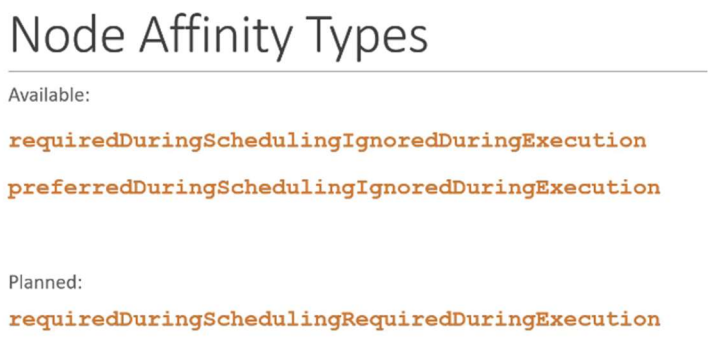
There are a number of other operators as well. Check the documentation for specific details.

Now, we understand all of this and we're comfortable with creating a pod with specific affinity rules. When the pods are created, these rules are considered and the pods are placed onto the right nodes.

But what if node affinity could not match a node with a given expression? In this case, what if there are no nodes with the label called "size"? Say we had the labels and the pods are scheduled. What if someone changes the label on the node at a future point in time? Will the pod continue to stay on the node?

All of this is answered by the lengthy sentence-like property under node affinity, which happens to be the type of node affinity.

The type of node affinity defines the behavior of the scheduler with respect to node affinity and the stages in the life cycle of the pod. There are currently two types of node affinity available: "required during scheduling, ignored during execution" and "preferred during scheduling, ignored during execution." There are additional types of node affinity planned as of this recording, such as "required during scheduling, required during execution."



We will now break this down to understand further.

We will start by looking at the two available affinity types. There are two states in the lifecycle of a pod when considering node affinity: **"during scheduling"** and **"during execution."**

"During scheduling" is the state where a pod does not exist and is created for the first time. We have no doubt that when a pod is first created, the affinity rules specified are considered to place the pods on the right nodes.

Now, what if the nodes with matching labels are not available? For example, we forgot to label the node as large. That is where the type of node affinity used comes into play.

If you select the "required" type, which is the first one, the scheduler will mandate that the pod be placed on a node with a given affinity rules. If it cannot find one, the pod will not be scheduled. This type will be used in cases where the placement of the pod is crucial. If a matching node does not exist, the pod will not be scheduled.

But let's say the pod placement is less important than running the workload itself. In that case, you could set it to "preferred," and in cases where a matching node is not found, the scheduler will simply ignore node affinity rules and place the pod on any available node. This is a way of telling the scheduler, "Hey, try your best to place the pod on a matching node. But if you really cannot find one, just place it anywhere."

The second part of the property or the other state is "during execution."

"During execution" is the state where a pod has been running, and a change is made in the environment that affects node affinity, such as a change in the label of a node.

For example, say an administrator removed the label we set earlier called "size=large" from the node. Now what would happen to the pods that are running on the node?

As you can see, the two types of node affinity available today have this value set to "ignored," which means pods will continue to run, and any changes in node affinity will not impact them once they are scheduled.

Available:

requiredDuringSchedulingIgnoredDuringExecution

preferredDuringSchedulingIgnoredDuringExecution

	DuringScheduling	DuringExecution
Type 1	Required	Ignored
Type 2	Preferred	Ignored

The new types expected in the future only have a difference in the during execution phase. A new option called required during execution is introduced which will evict any pods that are running on nodes that do not meet affinity rules. In the earlier example, a pod running on the large node will be evicted or terminated if the label large is removed from the node.

Planned:

requiredDuringSchedulingRequiredDuringExecution

	DuringScheduling	DuringExecution
Type 1	Required	Ignored
Type 2	Preferred	Ignored
Type 3	Required	Required