There are three different types of affinity there is Node affinity Pod affinity and Pod anti-affinity

Three Types of Affinity

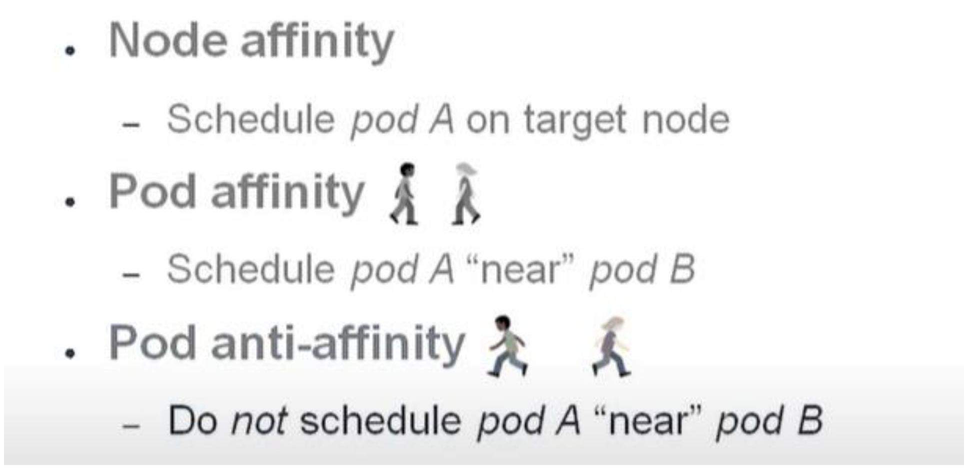
- Node affinity
- Pod affinity
- **Pod anti-affinity**

Node affinity dictates that the pod should only be scheduled on two nodes with specific labels

This is basically what the node selector does for example only schedule the cache pod on nodes with the label memory equals SSD

Next we have Pod affinity it dictates that the pod should only be scheduled on nodes where other specific pods are already running for example only schedule the cache pod on nodes where the web server pod has already been scheduled this way we can reduce the network latency between the two pods

Finally there's pod anti affinity this is how you require that pod a should not be scheduled on the same node as podB for instances you probably don't want to put too heavily

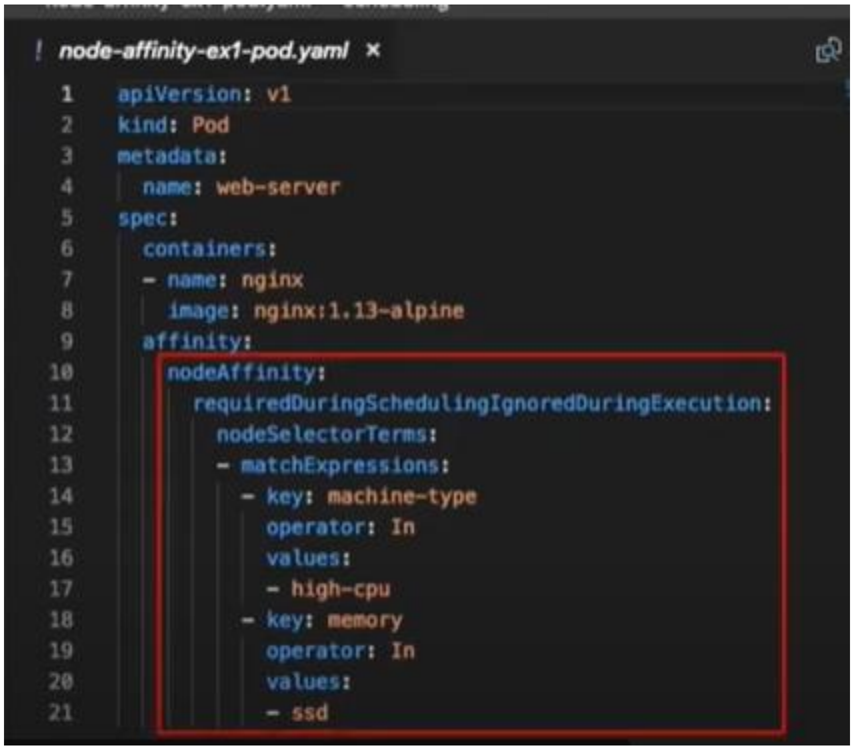


- Node affinity
  - Schedule *pod A* on target node
- Pod affinity
  - Schedule *pod A* "near" *pod B*
- Pod anti-affinity
  - Do *not* schedule *pod A* "near" *pod B*

Used databases on the same machine therefore you can use pod anti affinity to make sure that these DB pods repel each other and get scheduled on to separate nodes

Node Affinity example of a pod config file which specifies a node affinity requirement its a bit more verbose than the

But two are more or less equivalent this is a node affinity rule node affinity comes in two



```yaml
! node-affinity-ex1-pod.yaml ×
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
  - name: nginx
    image: nginx:1.13-alpine
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: machine-type
            operator: In
            values:
            - high-cpu
          - key: memory
            operator: In
            values:
            - ssd
```

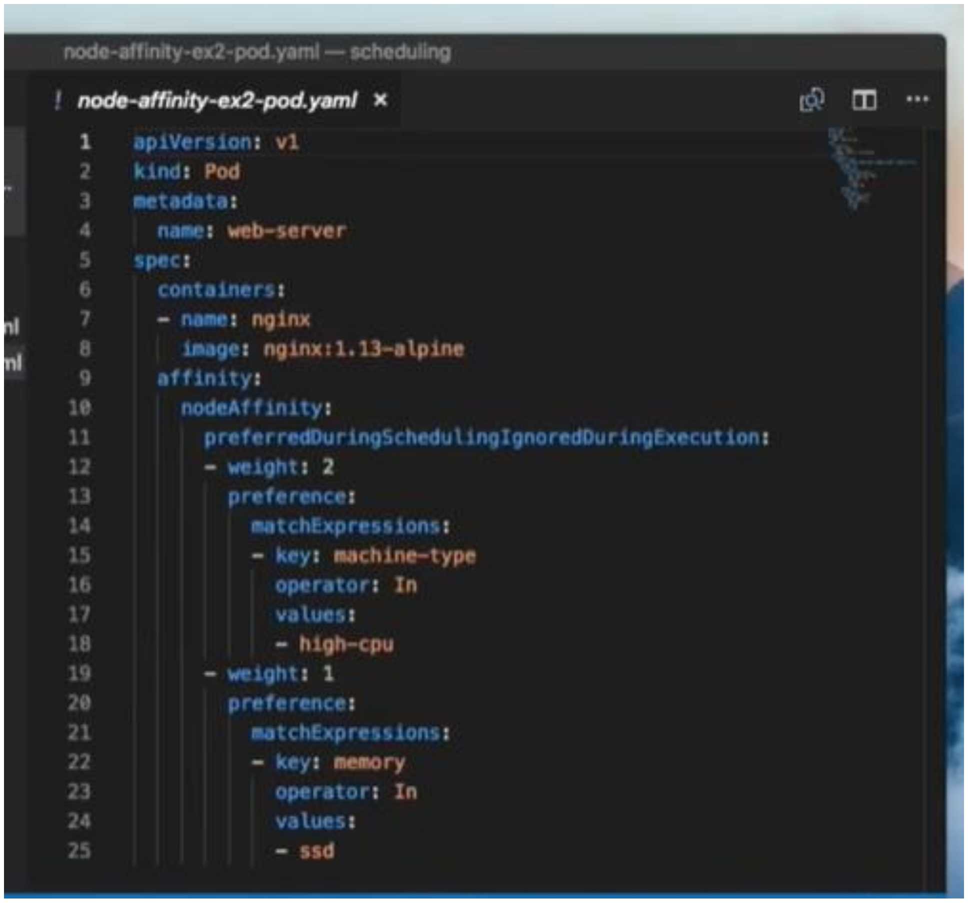
Variants required during scheduling ignored during execution and preferred during scheduling ignored during execution

The names of these two fields are ridiculosly long but they spell out exactly what they do the required during scheduling pods means that the pod can only be scheduled on nodes that meet the requirements laid out below at scheduling time the ignored during execution pod means that the pod will not be unscheduled from a node which no longer meets the requirements the preferred during scheduling ignored during execution field is very similar the prefer during scheduling pod specifies that the pod prefers to be scheduled on a node which meets the requirements below

How ever the pod may be scheduled to a node which does not meet one or more of the requirements the ignored during execution pod is the same as what we saw before the node selector terms field contains a list of match expressions elements in this case there’s only one but if there were multiple their results would be logically or together the match expressions list is one of your standard label selectors that we learned about way back at the beginning of this course each of its elements are logically all together which means that this pod will only be scheduled on two nodes which have the labels machine-type equals high CPU and memory equals SSD

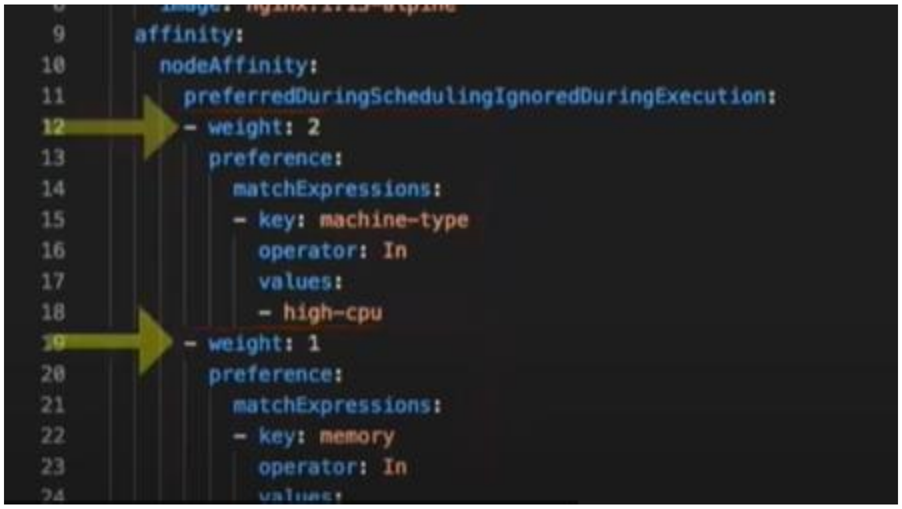
=======

Node affinity example-2


node-affinity-ex2-pod.yaml — scheduling
! node-affinity-ex2-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
  - name: nginx
    image: nginx:1.13-alpine
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 2
        preference:
          matchExpressions:
          - key: machine-type
            operator: In
            values:
            - high-cpu
      - weight: 1
        preference:
          matchExpressions:
          - key: memory
            operator: In
            values:
            - ssd
```

The preferred during scheduling field expresses a list of node affinity preferences each element in the array has a weight



```yaml
affinity:
  nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 2
      preference:
        matchExpressions:
        - key: machine-type
          operator: In
          values:
          - high-cpu
    - weight: 1
      preference:
        matchExpressions:
        - key: memory
          operator: In
          values:
```

Weight value between one and a hundred k8s looks at each node in the cluster and compares it to each of the labeled requirements in the list the weight values get added together for each label expression which matches k8s then tries to schedule the pod on to the node with the highest total weight of course the pod may be scheduled onto a node with none of these labels because the node affinity is preferred not required so here's simple example of how this works the pod would prefer to be scheduled on a node which has both label values pairs but if it can't have that it would prefer to be scheduled on a node with a machine-type equals high CPU labels versus the node with a memory equals SSD label because the former weighs more than the latter

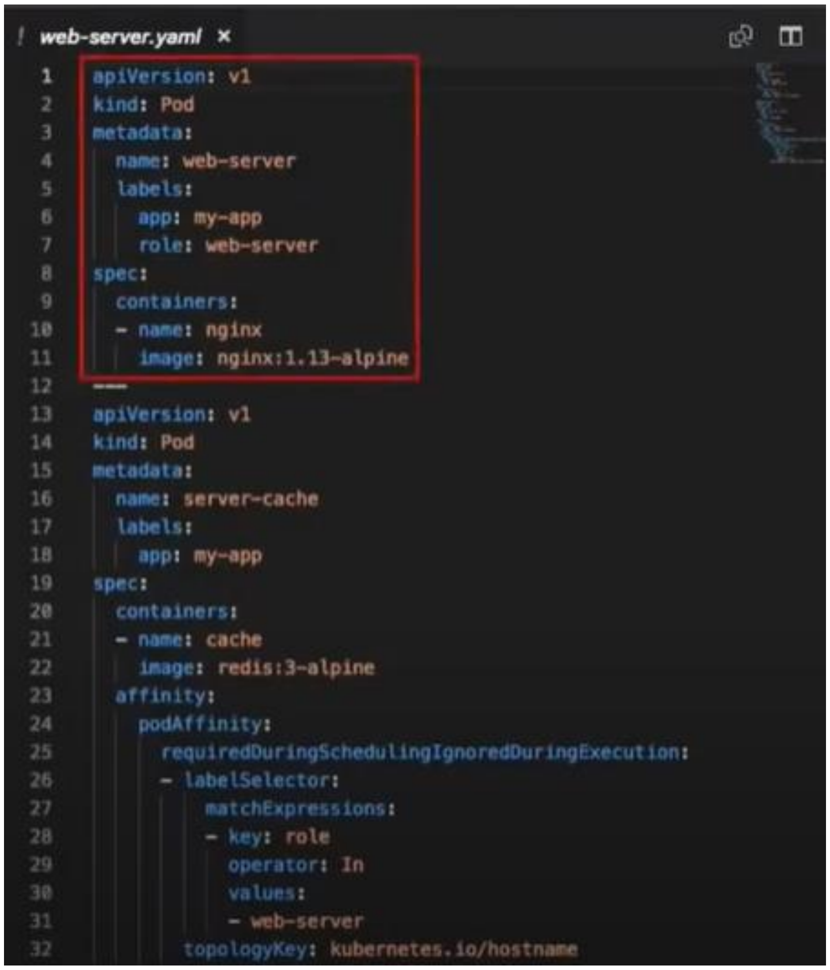
=====#####Pod Affinity #######===========

The intent behind the pod affinity feature is to give you the ability to co-locate certain pods

Always scheduled the server cache on to the same node as the webserver or always schedule the web server in the same cloud provider availability zone as the database node affinity and pod affinity are similar in many ways but they also differ in a few important ways

Pod Affinity Example1:-

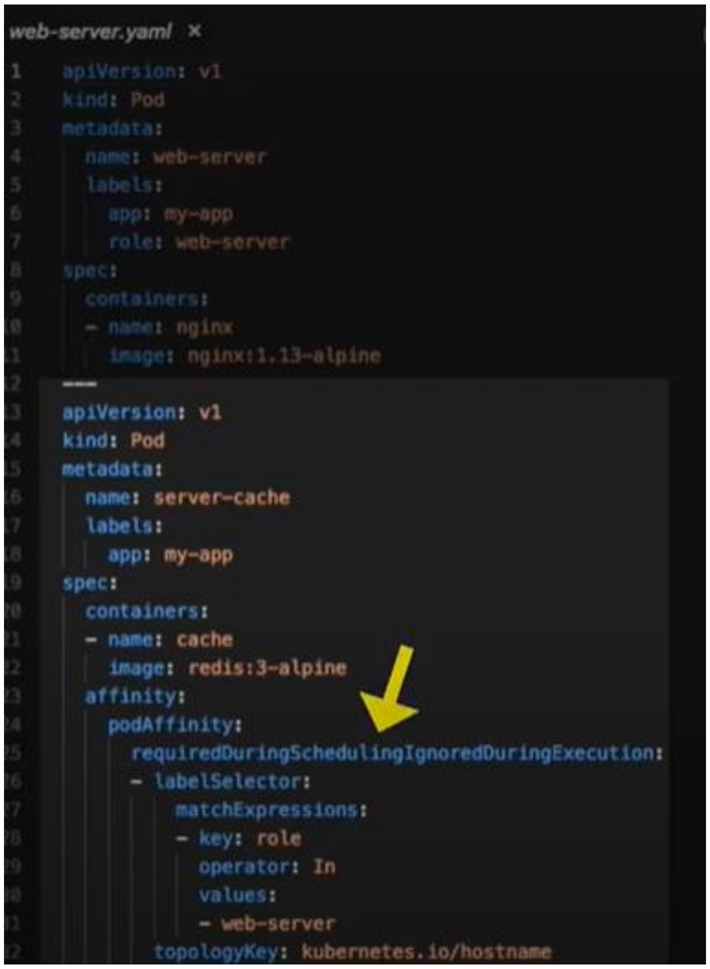
Here we have a config file the webserver pod and a cash pod node that server pod gets



web-server.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-server
  labels:
    app: my-app
    role: web-server
spec:
  containers:
  - name: nginx
    image: nginx:1.13-alpine
---
apiVersion: v1
kind: Pod
metadata:
  name: server-cache
  labels:
    app: my-app
spec:
  containers:
  - name: cache
    image: redis:3-alpine
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: role
            operator: In
            values:
            - web-server
        topologyKey: kubernetes.io/hostname
```

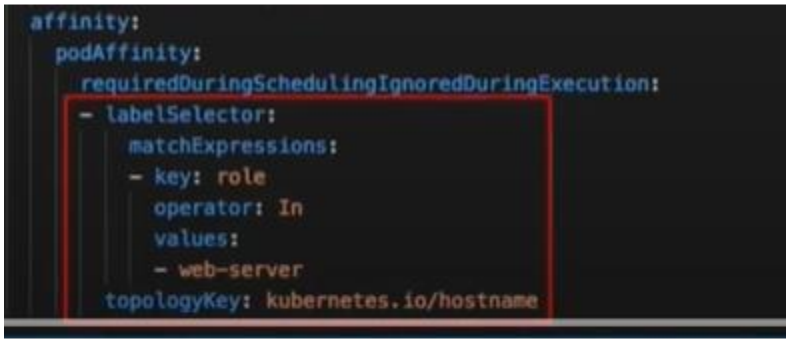
Scheduled first since it comes up first in the config file the cash pod is where all the action is it defines a pod affinity scheduling requirement notice that it has the required during scheduling ignored during the execution field



web-server.yaml ×

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-server
  labels:
    app: my-app
    role: web-server
spec:
  containers:
  - name: nginx
    image: nginx:1.13-alpine
---
apiVersion: v1
kind: Pod
metadata:
  name: server-cache
  labels:
    app: my-app
spec:
  containers:
  - name: cache
    image: redis:3-alpine
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: role
            operator: In
            values:
            - web-server
        topologyKey: kubernetes.io/hostname
```

This is the same type of hardware requirement that we learn about with node affinity as a consequence the cash pod will only be scheduled on to a node which meets the following requirements the



```yaml
affinity:
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: role
          operator: In
          values:
          - web-server
      topologyKey: kubernetes.io/hostname
```

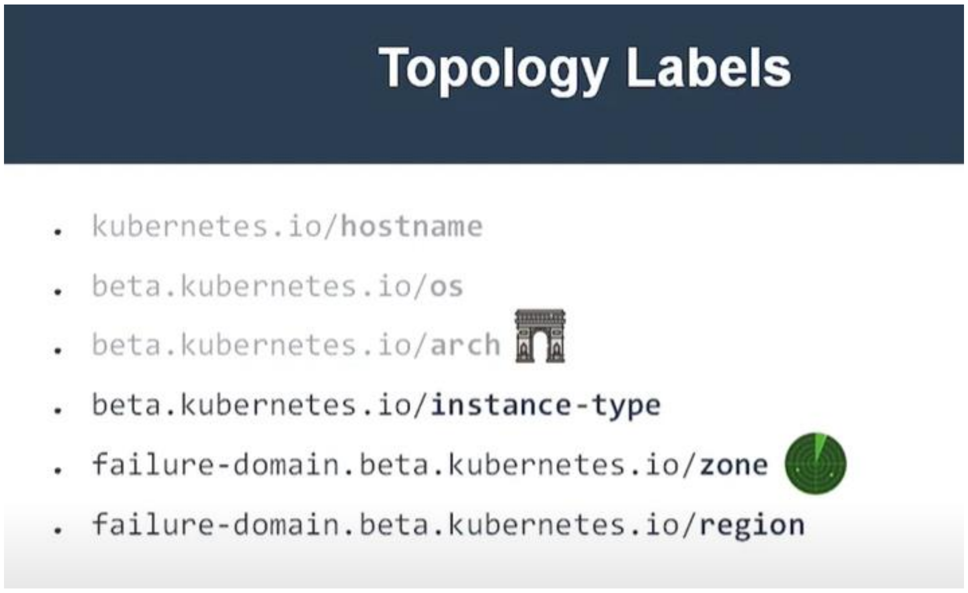
Required during scheduling ignore during execution array takes the union of the requirements dictated by its array elements in this case there's only one the label selector only matches pods which have the label role equals webserver like the one on the webserver pod

The idea of pod affinity is all about co-locating pods in the same place where place could be the same node the same cloud provider zone same region and

Well the topology key field is how you specify what constitutes co-location

Before i can give you a better explanation i need to take a few seconds to talk about some special node labels

Every node is automatically pre-populated with a subset of the following labels host OS arch for architecture instance type zone and region the



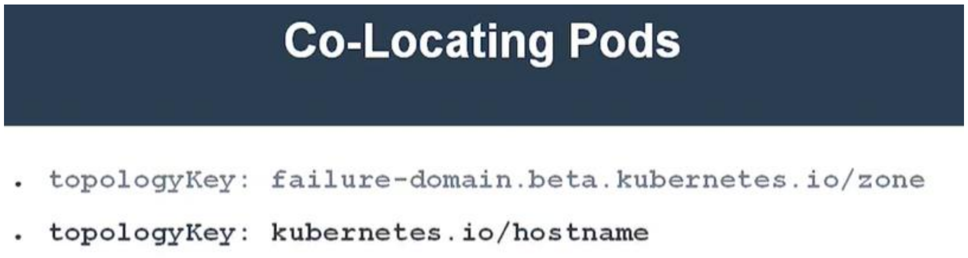
Topology Labels

- kubernetes.io/**hostname**
- beta.kubernetes.io/**os**
- beta.kubernetes.io/**arch**
- beta.kubernetes.io/**instance-type**
- failure-domain.beta.kubernetes.io/**zone**
- failure-domain.beta.kubernetes.io/**region**

The first three host name operating system and arch will probably be there on every node the last three instance type zone and region are more relevant to cloud providers

The latter three may or may not show up on a node the label keys are universal but the label values are not guaranteed to be same from one k8s provider to the next

Thats because keys like hostname zone and region are the values that you use for the topology key field for instance if you want to co –locate pods in the same cloud provider zone you put the value of the topology key field as failure domain beta k8s



Co-Locating Pods

- topologyKey: failure-domain.beta.kubernetes.io/zone
- **topologyKey: kubernetes.io/hostname**

If you want to co-locate pods on the same node you set the value of the topology key field as k8s

Hostname under the host topology key is just another label selector all you need to remembser is that A it’s required B it can not be emptu and C it defines what consitutes co-location

Pod affinity is the cache pod will only be scheduled on the same node as the webserver pod this is due to the combination of the pod lable selector and the topology key field