**K8s SchedulinG**

## **Manual Scheduling**

Text

Description automatically generated with low confidence

Every POD has a field called NodeName that by default is not set. You don’t typically specify this field when you create the manifest file, Kubernetes adds it automatically.

The scheduler goes through all the pods and looks for those that do not have this property set. Those are the candidates for scheduling.

It then identifies the right node for the POD by running the scheduling algorithm. Once identified it schedule the POD on the node by setting the node Name property to the name of the node by creating creating bindng object.

So if there is no scheduler to monitor and schedule nodes what happens ?? The pods continue to be in a Pending State.

**In that case you can manually assign the pods to node yourself. Well without a scheduler the easiest way to schedule a POD is to simply set the node name field to the name of the node in your pod specification file while creating the POD.**

**The pod then get assigned to the specified node. You can only specify the node name at creation time**

What if the pod is already created and you want to assign the pod to a node ? Kubernetes wont allow you to modify the node name propertir of a pod.

**So another way to assign a node to an existing pod is to create a binding object and send a post request to the pod binding API.**

POD BINDING

Graphical user interface, application

Description automatically generated

In the binding object you specify a target node with the name of the node. Then send a post request to the pods binding API with the data set to the binding object in a JSON format.

So you must convert the YAML file into its equivalent JSON format.

## **Labels && Selectors**

Labels >> Grouping

Selectors >> Filtering

In Kubernetes we have created lot of dif types of objects in Kubernetes. PODS, SERVICES, REPLICA SETS, DEPLOYMENTs etc. For Kubernetes all of these are dif objects , over time you may end up with having hundreds of these objects in your cluster.

Then you will need a way to filter and view dif objects by dif categories such as to group objecst by their type or view objects by application or by their functionality whatwever it may be. You can group and select objects using labels and selctors.

For each objects attach labels as per your needs, like app function etc. Then while selecting specify a condiftionto filter to filter specific objects.

**Label Example Config File Under Meta data**

 Here's the configuration file for a Pod that has two labels environment: production and app: nginx

Text

Description automatically generated

**Selector**

$ kubectl get pods –-selector app=nginx

Kubernetes objetcs use labels and selectors internally to connect dif objecst together.

A picture containing table

Description automatically generated

## **Annotations**

You can use Kubernetes annotations to attach arbitrary non-identifying metadata to objects. Clients such as tools and libraries can retrieve this metadata.

Metadata like email, version ID , contact info etc…

Text

Description automatically generated

## **Taints && Tolerations**

Taint for Node  
Tolerations for POD  
Chart, treemap chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**NoSchedule** which means the PODS will not schedule on this node.

**PreferNoSchedule** Will try to avoid placing a pod on this node but not guarantee

**NoExecute** means that new pod will not be scheduled on the node && existing pods if any will be evicted if they do not tolerate the taint. These parts may have been scheduled on Node before the taint was applied to the node .

**Tolerations**

**Its added to the POD**

Graphical user interface, text

Description automatically generated

To add the Toleration to the POD , Pull the pod definition file in the spec section of the pod def file had a section called TOLERATIONS move the same value used to while creating the taint under this section.

**Master Node :: By Default it is tainted**  
  
Graphical user interface, website

Description automatically generated

## **Node Selectors**

**STEP1 : Label the Node**Graphical user interface, text, application

Description automatically generated

STEP 2 :: NODE SELECTOR IN POD MANIFEST FILE

Graphical user interface, application

Description automatically generated

$ kubectl create -f pod-definition.yml

**Node Selector Limitations**

Graphical user interface, application

Description automatically generated

## **Node Affinity**

The primary purpose of Node Affinity is to ensure that the pods are hosted on particular nodes.

You cannot provide Large or Small , Not operations in Node selector.  
  
Node Affinity features provides us with advanced capability to limit the pod placement on specific nodes with great power and complexity.

Text

Description automatically generated

**Node Affinity Types**  
  
Graphical user interface, text, application, email

Description automatically generated

**Node Affinity Types**

Table

Description automatically generated

Type of node affinity defines the behavior of the scheduler with respect to node affinity and the stages in the lifecycle of the pod.

There are two states in the lifcycle of a pod when considering node affinity during Scheduling && During Execution.

**During Scheduling** Is the the State where a pod does not exist and is created for the first time. We have no doubt that 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 for forgot to label the node as large. That is where the **type of node affinity** used comes into play.

If you select **Required Type** will mandate that the pod be placed on a node with given affinity Rules. If it cannot find one the pod will not be scheduled.This type will be used in cases the placement of pod is crucial.

If you select **Prefered Type Where a amtchng node is not found, The scheduler will simply ignore node affinity rules and place the pod on any available node.**

During Execution :: The second part of the property or the other state is during the execution . During the 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 Admin removed the label we said earlier called size equals large from the node. Now what would happen to the pods that are running on the Node ?? Above type of Affinity set to ignore during execution. So pods will continue to run and any changes in node affinity will not impact them once they are scheduled.

**New Type**

The new types expected in the future only have a difference in the suting 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 affity rules.

Graphical user interface, application

Description automatically generated

## **Resourse Requirements && Limits**

Whenever a POD is placed on a node, it consumes resources available to that node. Scheduler decides which node a pod goes to. The scheduler takes into consideration the amount of resources required by a pod and those available on the nodes.

Pending state : If no resources avaiavle on any of the node.

By Default Kubernetes assumes that a pod or container within a pod requires 0.5 CPU && 256MB of mem. This is know as the resource request for a container, the minimum amount of CPU or memory requested.

If you know that your aplciation will need more than this, you can modify these values by specifying them in the POD or in your deployment manifest file.

**resources**:

**requests**:

**memory**: "64Mi"

**cpu**: "250m"

**limits**:

**memory**: "128Mi"

**cpu**: "500m"

* 1. CPU can be expressed as 100m where m stands for milli

1. count of CPU is equivalent to 1 AWS vCPU, 1GCP core, 1 Azure Core, 1 Hyperthread

**Limit**  
By Default Containers in the POD sets the limit of 1 vcpu && 512Mi.

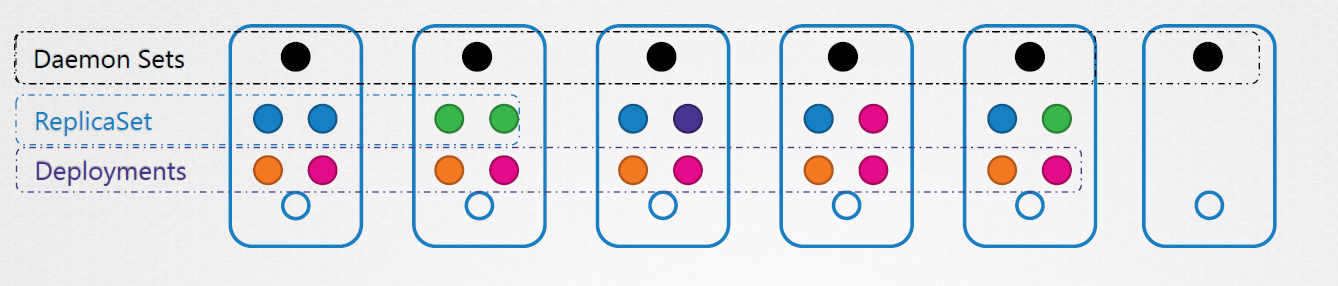
What happens when a pod tries to exceed resources beyond its specified limit ?? In case of CPU kubernetes throttles the CPU so that it does not go beyond the specified limit. A container cannot use mor CPU resourses than its limit.

Howvere this is not the case with the memory , A container can use more memory resources than its limit. So if a pod tries to consume more memory than its limit constantly that pod will be terminated.

When a pod is created the containers are assigned a default CPU request of .5 and memory of 256Mi". For the POD to pick up those defaults you must have first set those as default values for request and limit by creating a LimitRange in that namespace

1. apiVersion: v1
2. kind: LimitRange
3. metadata:
4. name: mem-limit-range
5. spec:
6. limits:
7. - default:
8. memory: 512Mi
9. defaultRequest:
10. memory: 256Mi
11. type: Container

## **DaemonSets**



DaemonSets are like replica sets as it helps you deploy multiple instances of POD but it runs one copy of pod on each node in your cluster. Whenever a new node is added to the cluster, a replica of the pod is automcaitcally added to that node. And when a node is removed , the pod is automtcailly removed.

The dameonset ensures that onecopy of the pod is always present in allnodes in the cluster.

**USECASES**

Say you would like to deploy a monitoring agent or log collector on each of your nodes in the cluster so you can monitor your cluster better.

KubeProxy component can be deployed as Daemon Sets.

Another usecase is Networkng soultions like Weave-Net requires Agent to be deployed on each node in the cluster.

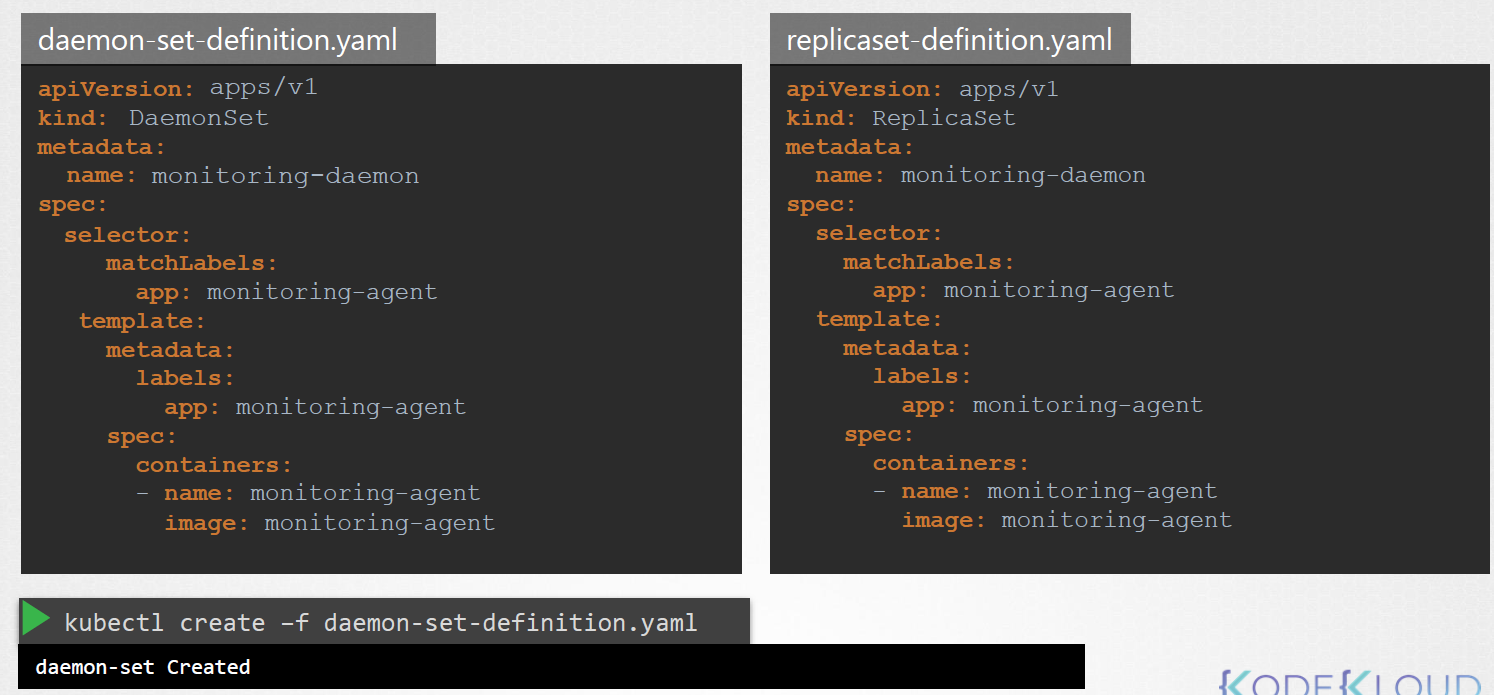
Diagram

Description automatically generated

A picture containing diagram

Description automatically generated

**DaemonSet Creation**



Text, website

Description automatically generated

**How does it work ??**

A picture containing text

Description automatically generated

We could set the node name property on the pod to bypass the scheduler and get the pod placed on a node directly.

## **Static Pods**

Kubelet relies on Kube-API server for instructions on what PODs to load on its side. Which is nased on the decision made by the Kube-scheduler which was stored in the ETCD datastore. What if there was no kube-api server and scheduler and no controllers and no ETCD cluster ??? What if no master ??

Kubelet can able manage a node independently. The one thing that the kubelet knows to do is create PODs. But we don’t have an API server to provide the pod details.  
  
You can configure the kubelet to read the pod definition files from a directory on the server designated to store information about pods. Place the POD manifest files in the directory. The kubelet periodically check this directory for files , reads these files and create the pods on the host.Not only does it create the pod it can ensure that the pod statys alive. If you make a change to any of the files within the directory, the kubelet recreates the pod for those changes take effect. If you remove a file from this dir then pod gets deleted. These are called Static pods.

Note: You can only create the pods, not other objects .

Designated directory could be any directory in the HOST. Location of that directory passed to the kubelet as an option while running the service.

Once pod creates use $ docker ps command instead of kubectl as there is no API server.

Kubelet can communicate with API at the same time can able to create the static pod as well. Will the API server aware abt the static pod ?? Yes it is .   
  
$ kubectl get pods will list the static pod as well. But you can only view not edit .

**Static POD use case**

Since static pods are not dependent on the Kubernetes control plane, you can use the stsic pods to deploy the control plane components itself as pod on a node. Start by installing kubelet on all the master nodes. Then create pod def file that uses docker images of the various control plane component such as….

## **Multiple Scheduler**