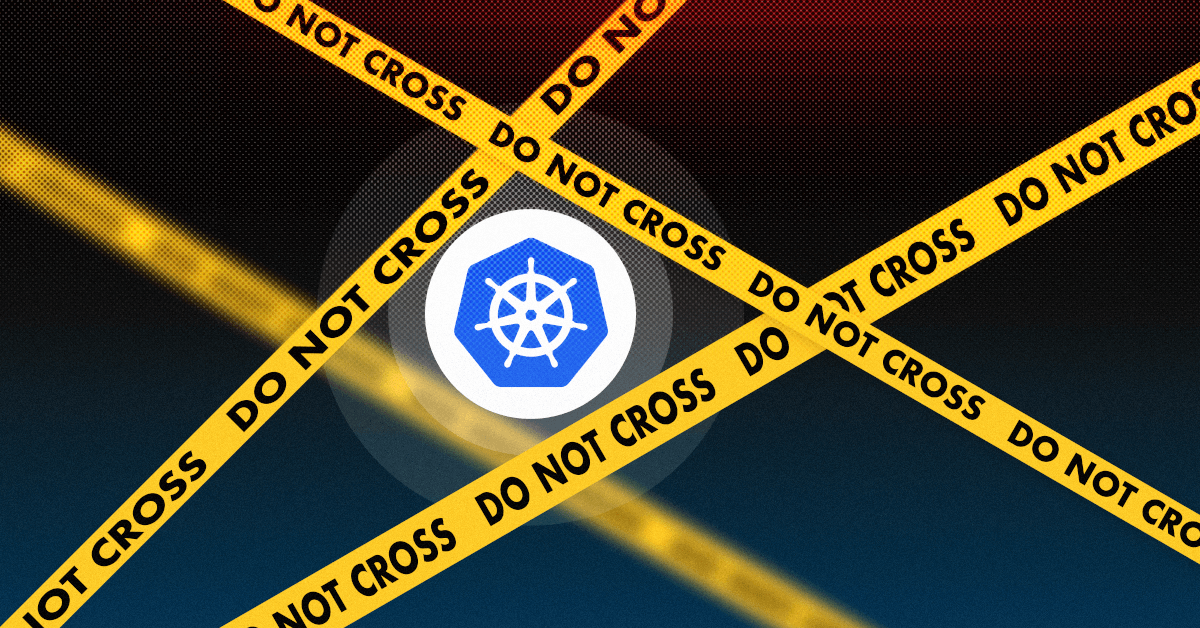
**Kubernetes Cordon: How It Works and When to Use It**

Kubernetes gives you the option to administer and maintain nodes manually. Using kubectl, you can easily create a node object and modify nodes – for example, set a label or mark it unschedulable. **This is where Kubernetes cordon comes in**. What is the Kubernetes cordon command, and how do you use it? Read this article to find out.



**What is Kubernetes cordon?**

Kubernetes cordon is an operation that marks or taints a node in your existing node pool as unschedulable. By using it on a node, you can be sure that no new pods will be scheduled for this node. The command prevents the Kubernetes scheduler from placing new pods onto that node, but it doesn’t affect existing pods on that node.

To mark a node unschedulable, all it takes is running this command:

kubectl cordon $NODENAME

Note that you can run pods that are part of a DaemonSet on an unschedulable Node. That’s because DaemonSets usually provide node-local services that should be running on the node, even if it’s marked as unschedulable and drained of workloads.

**When do you use the Kubernetes cordon?**

Nodes in a Kubernetes cluster may require maintenance – be it replacing hardware components in a self-hosted installation, updating the node’s kernel, or resizing its compute resources from a cloud provider.

Kubernetes cordon and drain prepare your application for node downtime by letting workloads running on a target node get rescheduled to other ones. You can then safely shut the node down and remove it from your cluster, confident that this doesn’t impact service availability in any way.

**How does Kubernetes cordon work – step-by-step guide**

**Step 1: Cordoning a node**

By cordoning a node, you mark it as unavailable to the scheduler. This makes the node not eligible to host any new pods that will be added to your cluster.

To place a cordon around a named node, use this command:

$ kubectl cordon node-12

The existing pods on this node won’t be affected by this and will remain accessible.

Want to check which nodes are cordoned at the moment? Use the get nodes command:

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

node-12 Ready,SchedulingDisabled control-plane,master 22m v1.23.3

All the cordoned nodes will appear with the status SchedulingDisabled.

**Step 2: Draining a node**

To empty the node from the remaining pods, you need to drain it. This process evicts the pods so they can be rescheduled to other nodes in your cluster. You can gracefully terminate pods before forcing them to be removed from the node you’re planning to drain.

To start the drain procedure, run kubectl drain, specifying the name of the node you’re draining:

$ kubectl drain node-12

node/node-12 already cordoned

evicting pod kube-system/storage-provisioner

evicting pod default/nginx-7c658794b9-zszdd

evicting pod kube-system/coredns-64897985d-dp6lx

pod/storage-provisioner evicted

pod/nginx-7c658794b9-zszdd evicted

pod/coredns-64897985d-dp6lx evicted

node/node-12 evicted

Note: In practice, draining without ignoring daemonsets hardly happens, as illustrated [here](https://youtu.be/IHrFubX5WtU?t=210).

How does the drain procedure work? First, we have Kubernetes cordoning – it cordons the node if you haven’t done that manually. Next, it evicts running pods (even if there is no space to reschedule them). Once the node is empty, you can shut it down or destroy it. The cordon assures that no new workloads can be scheduled on this node once the drain is completed.

**What happens if your pods have long grace periods?**

Kubernetes node draining might take a while if your pods happen to have long grace periods. This might become a problem if you’re looking to take a node offline quickly.

You can always use the –grace-period flag to override your pod termination grace periods and force an instant eviction:

$ kubectl drain node-12 --grace-period 0

Use this one carefully. Some workloads might not respond well when you force them to stop without having the chance to clean up.

Another command worth knowing is one for draining node even if there are no pods managed by a ReplicationController, Job, or DaemonSet on it:

$ kubectl deain node-12 --force

**How do you undrain/uncordon a node in Kubernetes?**

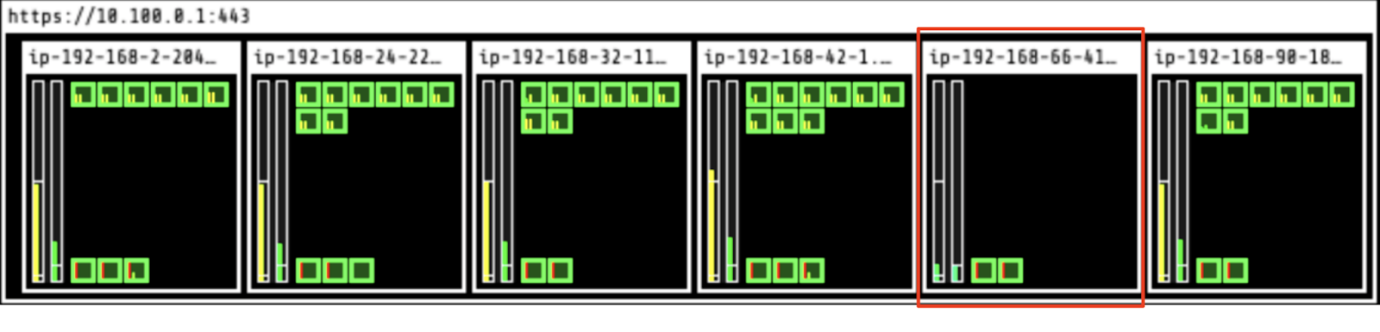
It’s possible to undrain or uncordon a node in Kubernetesn by marking this node as schedulable. Use the following command:

kubectl uncordon NODE

**Using Kubernetes cordon for cluster optimization**

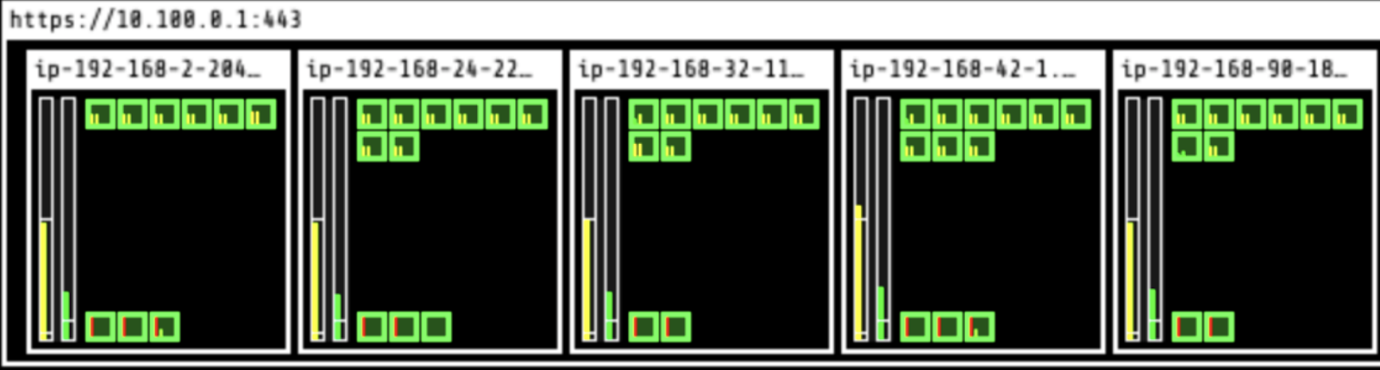
Suppose you analyze your cluster with a third-party optimization solution and discover a great potential for cost savings. But to get there, you need to change the compute resources your cluster is running on. This is where Kubernetes taint comes in handy.

Our customers can maximize their cost savings immediately by using Evictor – a mechanism that **looks for inefficiencies and compacts your pods into fewer nodes**, creating empty nodes that will be removed by the Node deletion policy.

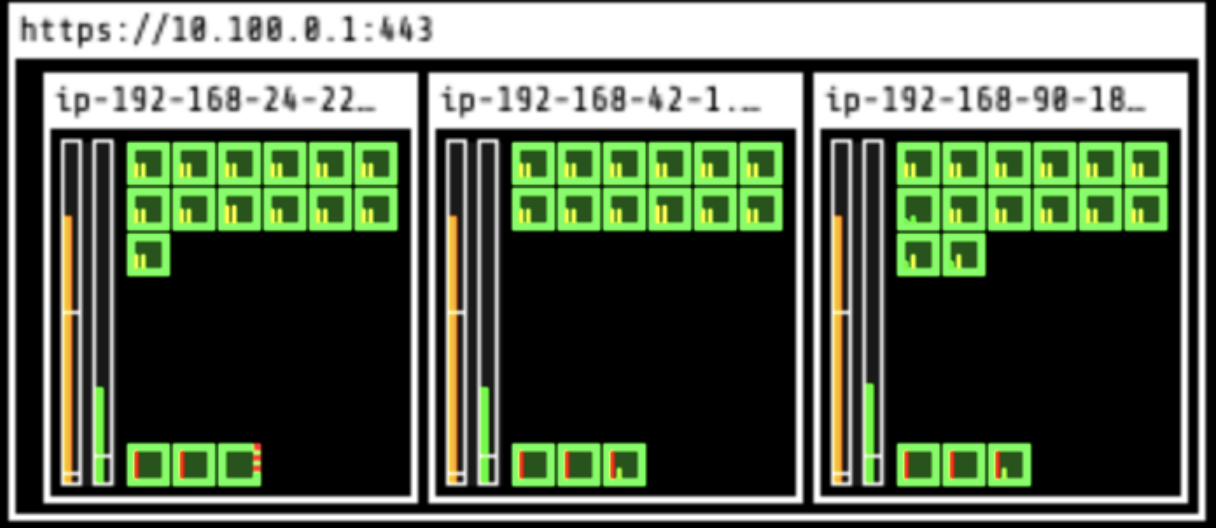


1. One node (in red below) is identified as a candidate for eviction.
2. Evictor automatically moves the pods to other nodes – this i called “bin-packing.”
3. Once the node becomes empty, it’s deleted from the cluster.
4. Go back to step 1.

One node is deleted:



And at the end of the process, 3 nodes remain:



In 10 minutes, Evictor deleted 3 nodes and left 3 nodes running to optimize the setup and cut costs.

**Automate Kubernetes node cordoning and draining**

Changing the compute resources your nodes run on manually quickly becomes time-consuming if you’re dealing with many nodes. You can let CAST AI do the job and do the node cordoning and draining for you as part of its automated optimization mechanisms.