

KubeAPI Server

- KubeAPI server is a process running on master nodes that manages the cluster. The `kubelet` services running on each node communicate with the `kube-apiserver` to share information about each node in the cluster.
- When we interact with the cluster using `kubectl` tool, it sends API requests to KubeAPI server which then authenticates the request, checks if the user is authorized to perform the required action and then saves the state in `etcd`.
- It is the only component of the cluster that directly interacts with `etcd`. All the other components of the cluster (eg. `scheduler` or `kubelet`) use API server to interact with the cluster.
- When setting up the cluster from scratch, download the `kube-api` binary and run it as a service.
- If the cluster is set up using **KubeAdmin**, the kube-api server is automatically deployed as a static pod in the `kube-system` namespace on the master node. The config is present at `/etc/kubernetes/manifests/kube-apiserver.yaml`

ETCD

- Distributed key-value store that stores the cluster state. ETCD is distributed datastore, which means it replicates the data across all its instances. That way, if one of them crashes, the data is still available on the rest of them.
- When setting up the cluster from scratch, the `etcd` binary needs to be downloaded and run as a service on the master node. By default it listens on port `2379`.
- When setting up the cluster using **KubeAdmin**, `etcd` is automatically deployed as a static pod on the master node.
- `etcdctl` utility can be used to interact with `etcd` to get and put items in it.
- In a cluster with multiple master nodes (for high availability), the `etcd` service will be running on each master node and will be communicating with each other on port `2380`. This can be configured in the `etcd` config.

`etcdctl`

Installation

Follow the release notes for any `etcd` release to download and unzip the `etcd` and `etcdctl` binary from <https://github.com/etcd-io/etcd/releases/>. At the end, move the extracted binaries to `/usr/local/bin` or `/usr/bin`.

Commands


- Backup a cluster's ETCD

```
ETCDCTL_API=3 etcdctl \ --cacert=/etc/kubernetes/pki/etcd/ca.crt \ --cert  
=/etc/kubernetes/pki/etcd/server.crt \ --key=/etc/kubernetes/pki/etcd/ser  
ver.key \ snapshot save <backup-filename>
```

- Restore a cluster from an ETCD backup

```
ETCDCTL_API=3 etcdctl snapshot restore --data-dir="/var/lib/etcd-snapshots" <backup-filename>
```

Kube Controller Manager

- A collection of all the  **Controllers** in K8s packaged as a single process. Controllers continuously monitor K8s objects in the `etcd` and make the necessary changes to them in the cluster.
- When setting up the cluster from scratch, download the `kube-controller-manager` binary and run it as a service.
- If the cluster is set up using **KubeAdmin**, the `kube-controller-manager` is automatically deployed as a static pod in the `kube-system` namespace on the master node. The config is present at `/etc/kubernetes/manifests/kube-controller-manager.yaml`

Kube Scheduler

- Kube Scheduler decides on which node the pod should be scheduled. It doesn't create the pod on the required node. That is the job of `kubelet` service. It only decides which node the pod will go to.
- It looks for pods that don't have `nodeName` property set (usually every pod) and uses its scheduling algorithm to set the `nodeName` property. Once the `nodeName` property is set, the `kubelet` service running on that node requests the container runtime to create the pod.

```
apiVersion: v1 kind: Pod metadata: labels: name: frontend spec: container
s: - name: httpd image: httpd:2.4-alpine nodeName: node02
```

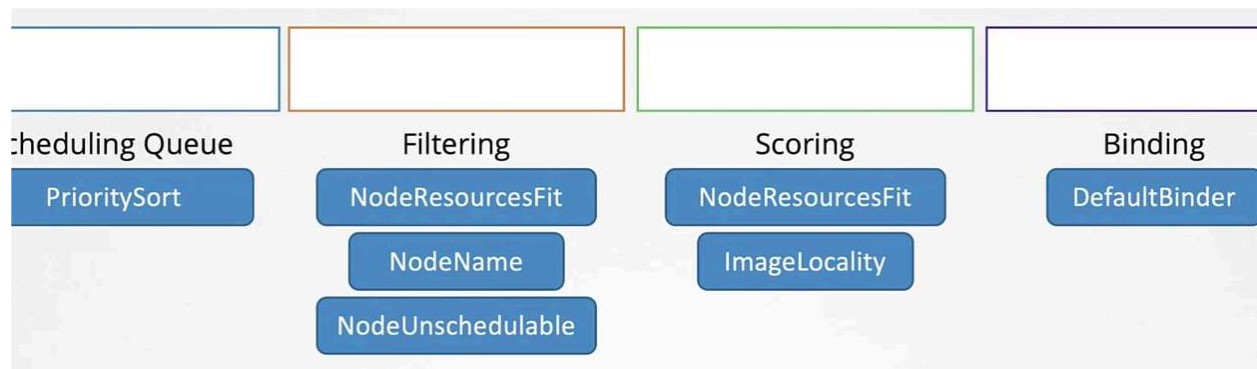
- If the cluster doesn't have a scheduler, and the `nodeName` property is not set manually, the pod will remain in pending state.
- When setting up the cluster from scratch, download the `kube-scheduler` binary and run it as a service.
- If the cluster is set up using **KubeAdmin**, the `kube-scheduler` is automatically deployed as a static pod in the `kube-system` namespace on the master node. The config is present at `/etc/kubernetes/manifests/kube-scheduler.yaml`

Manual Scheduling

If the cluster doesn't have `kube-scheduler`, we can manually schedule a pod to a node by setting the `nodeName` property. This can only be done when the pod is created. To schedule a pod on another node, recreate the pod.

```
apiVersion: v1 kind: Pod metadata: labels: name: frontend spec: containers: -
name: httpd image: httpd:2.4-alpine nodeName: node02
```


Scheduling Plugins



The scheduler has 4 phases, each having a set of plugins that operate at that phase.

- **Scheduling Queue:** pod are queued based on their priority
- **Filtering:** nodes that don't satisfy the pod requirements are discarded
- **Scoring:** filtered nodes are scored based on various factors like the amount of resources present after scheduling the pod and whether or not the container image is already present on that node
- **Binding:** the node with the highest score is selected

Kubelet

- It's a process running on the worker nodes that registers the worker nodes to the cluster.
- When the scheduler schedules a pod on a given node (populates its `nodeName` property), the `kubelet` service running on that node requests the container runtime engine (eg. Docker) to pull the image and run the container.
- It periodically monitors the status of the node and the pods on them and reports them to the `kube-api` server.
- It can independently manage pods on the node without relying on other K8s components. Kubelet can be configured to look for K8s manifest files in a directory. It can then automatically create, update and manage pods on the node based on the manifests files present in the directory. These pods are called  Static Pods.
- The `kubelet` service must be run manually on the worker nodes. It is not setup automatically when we use **KubeAdmin** to setup the cluster.

Kube Proxy

- A process, running on all the nodes in a cluster, that implements network address translation of packets going to a service, so that they are sent to the backend pods. It does not implement pod networking, that's the job of CNI plugin.
- Running on every node, it listens for new service creation, at which it creates IP table rules on that node to route traffic going to that service to the backend pods running on the node. If the cluster spans multiple nodes, the CNI plugin should be working to ensure pod connectivity across the cluster.
- When setting up the cluster from scratch, download the `kube-proxy` binary and run it as a service on every node.
- If the cluster is set up using **KubeAdmin**, the `kube-proxy` is automatically deployed as a daemonset (one pod on every node) in the `kube-system` namespace.
- Excellent explanation of how KubeProxy works - [Demystifying kube-proxy | Mayank Shah](#)