Exploring the Components That Constitute the Cluster

In this lesson, we will explore the constituents of the cluster we created in the previous lesson.

WE'LL COVER THE FOLLOWING

Exploring the Constituents of the Cluster

When kops created the VMs (EC2 instances), the first thing it did was to execute `nodeup`. It, in turn, installed a few packages. It made sure that Docker, Kubelet, and Protokube are up and running.

Docker

Docker runs containers. It would be hard for me to imagine that you don't know what Docker does, so we'll skip to the next in line.

Kubelet

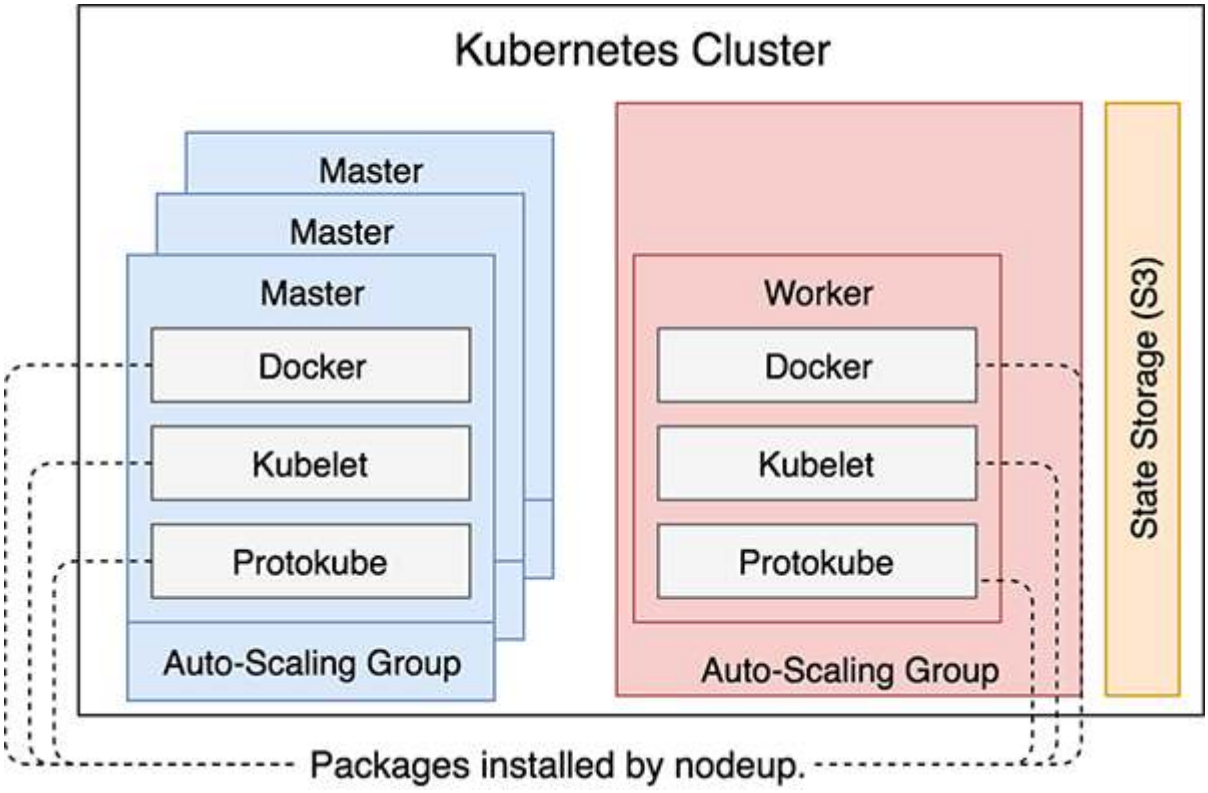
Kubelet is Kubernetes' node agent. It runs on every node of a cluster, and its primary purpose is to run Pods. Or, to be more precise, it ensures that the containers described in PodSpecs are running as long as they are healthy. It primarily gets the information about the Pods it should run through Kubernetes' API server. As an alternative, it can get the info through files, HTTP endpoints, and HTTP servers.

Protokube

Unlike Docker and Kubelet, **Protokube** is specific to kops. Its primary responsibilities are to discover master disks, to mount them, and to create manifests. Some of those manifests are used by Kubelet to create system-level Pods and to make sure that they are always running.

Besides starting the containers defined through Pods in the manifests (created by Protokube), Kubelet also tries to contact the API server which, eventually, is also started by it. Once the connection is established, Kubelet registers the node where it is running.

All three packages are running on all the nodes, no matter whether they are masters or workers.



The servers that form the Kubernetes cluster

System-level Components

Let's take a look at the system-level Pods currently running in our cluster.

```
kubectl --namespace kube-system get pods
```

The **output** is as follows.

```
NAME                                          READY STATUS  RESTARTS AGE
dns-controller-...                            1/1   Running 0        5m
etcd-server-events-ip-172-20-120-133...       1/1   Running 0        5m
etcd-server-events-ip-172-20-34-249...        1/1   Running 1        4m
etcd-server-events-ip-172-20-65-28...         1/1   Running 0        4m
etcd-server-ip-172-20-120-133...              1/1   Running 0        4m
etcd-server-ip-172-20-34-249...               1/1   Running 1        3m
etcd-server-ip-172-20-65-28...                1/1   Running 0        4m
kube-apiserver-ip-172-20-120-133...           1/1   Running 0        4m
kube-apiserver-ip-172-20-34-249...            1/1   Running 3        3m
kube-apiserver-ip-172-20-65-28...             1/1   Running 1        4m
kube-controller-manager-ip-172-20-120-133... 1/1   Running 0        4m
kube-controller-manager-ip-172-20-34-249...   1/1   Running 0        4m
kube-controller-manager-ip-172-20-65-28...    1/1   Running 0        4m
kube-dns-7f56f9f8c7-...                       3/3   Running 0        5m
kube-dns-7f56f9f8c7-...                       3/3   Running 0        2m
kube-dns-autoscaler-f4c47db64-...             1/1   Running 0        5m
kube-proxy-ip-172-20-120-133...               1/1   Running 0        4m
kube-proxy-ip-172-20-34-249...                1/1   Running 0        4m
kube-proxy-ip-172-20-65-28...                 1/1   Running 0        4m
kube-proxy-ip-172-20-95-101...                1/1   Running 0        3m
kube-scheduler-ip-172-20-120-133...           1/1   Running 0        4m
kube-scheduler-ip-172-20-34-249...            1/1   Running 0        4m
kube-scheduler-ip-172-20-65-28...             1/1   Running 0        4m
```

As you can see, quite a few core components are running.

We can divide core (or system-level) components into two groups.

- Master components
- Node components

Master Components

Master components run only on masters. In our case, they are `kube-apiserver`, `kube-controller-manager`, `kube-scheduler`, `etcd`, and `dns-controller`.

Kubernetes API Server (`kube-apiserver`): Kubernetes API Server accepts requests to create, update, or remove Kubernetes resources. It listens on ports `8080` and `443`. The former is insecure and is only reachable from the same server. Through it, the other components can register themselves without requiring a token. The later port (`443`) is used for all external communications with the API Server. That communication can be user-facing like, for example, when we send a `kubectl` command. Kubelet also uses `443` port to reach the API server and register itself as a node.

No matter who initiates communication with the API Server, its purpose is to validate and configure an API object. Among others, those can be Pods, Services, ReplicaSets, and others. Its usage is not limited to user-facing interactions. All the components in the cluster interact with the API Server for the operations that require a cluster-wide shared state.

The shared state of the cluster is stored in **etcd**. It is a key/value store where all cluster data is kept, and it is highly available through consistent data replication. It is split into two Pods, where `etcd-server` holds the state of the cluster and `etcd-server-events` stores the events.

Kops creates an **EBS volume** for each `etcd` instance. It serves as its storage.

Kubernetes Controller Manager (`kube-controller-manager`): Kubernetes Controller Manager is in charge of running controllers. You already saw a few controllers in action like ReplicaSets and Deployments. Apart from object controllers like those, `kube-controller-manager` is also in charge of Node Controllers responsible for monitoring servers and responding when one becomes unavailable.

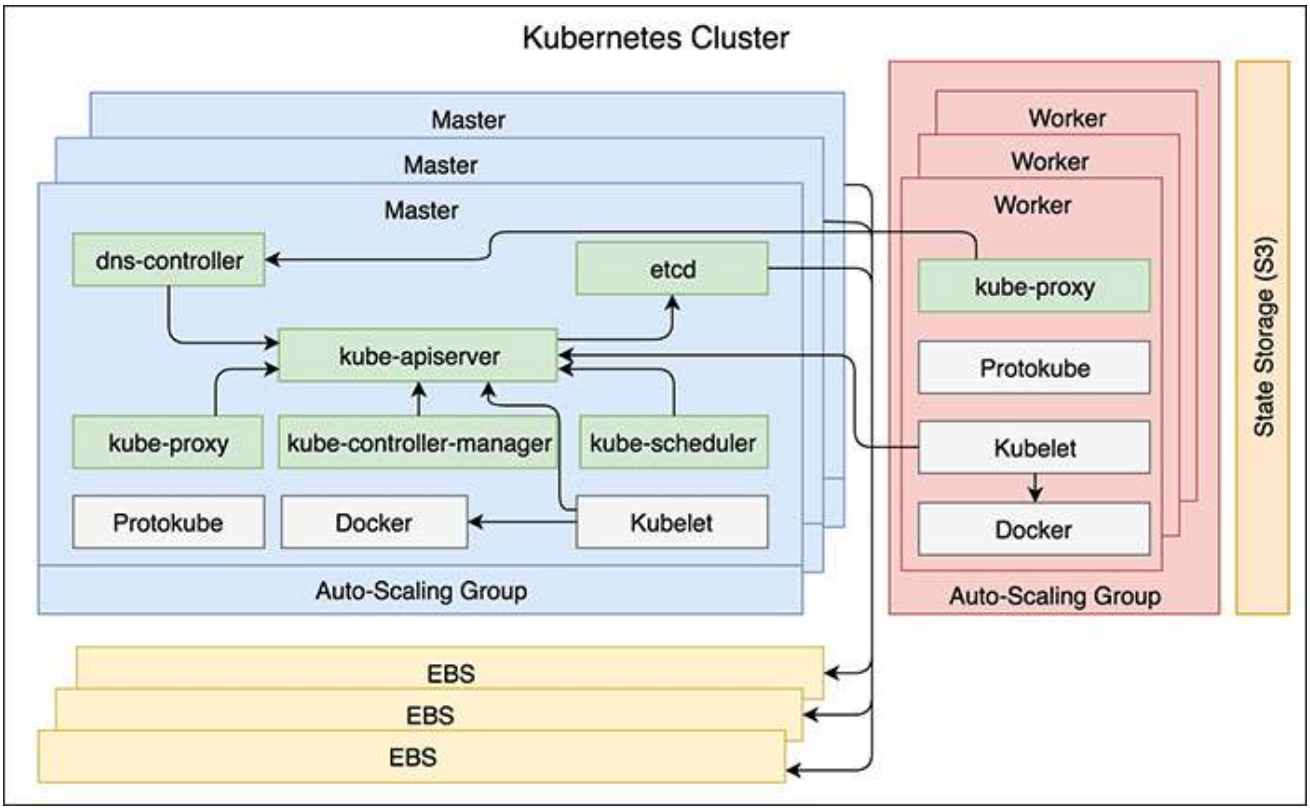
Kubernetes Scheduler (`kube-scheduler`): Kubernetes Scheduler watches the API Server for new Pods and assigns them to a node. From there on, those Pods are run by Kubelet on the allocated node.

DNS Controller (`dns-controller`): DNS Controller allows nodes and users to discover the API Server.

Node components

Node components run on all the nodes, both masters and workers. In addition to Protokube, Docker, and Kubelet, we got `kube-proxy`, as one more node component.

Kubernetes Proxy (`kube-proxy`): Kubernetes Proxy reflects Services defined through the API Server. It is in charge of TCP and UDP forwarding. It runs on all nodes of the cluster (both masters and workers).



The core components of the cluster

There's much more going on in our new cluster. For now, we explored only the major components.

Next, we'll try to update our cluster.