KUBERNETES ARCHITECTURE

Kubernetes Architecture

Control Plane Node

The Kubernetes cp runs various server and manager processes for the cluster. As the software has matured, new components have been created to handle dedicated needs, such as the **cloud-controller-manager**; it handles tasks once handled by the **kube-controller-manager** to interact with other tools, such as Rancher or DigitalOcean for third-party cluster management and reporting.

There are several add-ons which have become essential to a typical production cluster, such as DNS services. Others are third-party solutions where Kubernetes has not yet developed a local component, such as cluster-level logging and resource monitoring.

As a concept, the various pods responsible for ensuring the current state of the cluster matches the desired state are called the *control plane*.

When building a cluster using kubeadm, the kubelet process is managed by systemd. Once running, it will start every pod found in **`/etc/kubernetes/manifests/`**.

Click on each box to learn about the components of the control plane node.

Components of the Control Plane Node

kube-apiserver Close

The **kube-apiserver** is central to the operation of the Kubernetes cluster. All calls, both internal and external traffic, are handled via this agent. All actions are accepted and validated by this agent, and it is the only connection to the **etcd** database. It validates and configures data for API objects, and services REST operations. As a result, it acts as a cp process for the entire cluster, and acts as a frontend of the cluster's shared state.

Starting as a beta feature in v1.18, the Konnectivity service provides the ability to separate user-initiated traffic from server-initiated traffic. Until these features are developed, most network plugins commingle the traffic, which has performance, capacity, and security ramifications.

kube-scheduler

Close

The **kube-scheduler** uses an algorithm to determine which node will host a Pod of containers. The scheduler will try to view available resources (such as volumes) to bind, and then try and retry to deploy the Pod based on availability and success. There are several ways you can affect the algorithm, or a custom scheduler could be used instead. You can also bind a Pod to a particular node, though the Pod may remain in a pending state due to other settings. One of the first settings referenced is if the Pod can be deployed within the current quota restrictions. If so, then the taints and tolerations, and labels of the Pods are used along with the metadata of the nodes to determine the proper placement.

The details of the scheduler can be found on GitHub.

etcd Database

Close

The state of the cluster, networking, and other persistent information is kept in an **etcd** database, or, more accurately, a b+tree key-value store. Rather than finding and changing an entry, values are always appended to the end. Previous copies of the data are then marked for future removal by a compaction process. It works with **curl** and other HTTP libraries, and provides reliable watch queries.

Simultaneous requests to update a value all travel via the **kube-apiserver**, which then passes along the request to **etcd** in a series. The first request would update the database. The second request would no longer have the same version number, in which case the **kube-apiserver** would reply with an error 409 to the requester. There is no logic past that response on the server side, meaning the client needs to expect this and act upon the denial to update.

There is a *Leader* database along with possible *followers*, or non-voting Learners who are in the process of joining the cluster. They communicate with each other on an ongoing basis to determine which will be the Leader, and determine another in the event of failure. While very fast and potentially durable, there have been some hiccups with new tools, such as **kubeadm**, and features like whole cluster upgrades.

While most Kubernetes objects are designed to be decoupled, a transient microservice which can be terminated without much concern **etcd** is the exception. As it is, the persistent state of the entire cluster must be protected and secured. Before upgrades or maintenance, you should plan on backing up **etcd**. The **`etcdctl`** command allows for **`snapshot save`** and **`snapshot restore`**.

Other Agents

Close

The **kube-controller-manager** is a core control loop daemon which interacts with the **kube-apiserver** to determine the state of the cluster. If the state does not match, the manager will contact the necessary controller to match the desired state. There are several operators in use, such as endpoints, namespace, and replication. The full list has expanded as Kubernetes has matured.

Remaining in beta since v1.11, the **cloud-controller-manager** (**ccm**) interacts with agents outside of the cloud. It handles tasks once handled by **kube-controller-manager**. This allows faster changes without altering the core Kubernetes control process. Each kubelet must use the **`--cloud-provider-external`** settings passed to the binary. You can also develop your own ccm, which can be deployed as a daemonset as an in-tree deployment or as a free-standing out-of-tree installation. The **cloud-controller-manager** is an optional agent which takes a few steps to enable. You can learn more about the cloud-controller-manager online.

Depending on which network plugin has been chosen, there may be various pods to control network traffic. To handle DNS queries, Kubernetes service discovery, and other functions, the **CoreDNS** server has replaced **kube-dns**. Using chains of plugins, one of many provided or custom written, the server is easily extensible.