Cloud computing, containerization, and container orchestration are the most important trends in DevOps. Whether you're a DevOps, software developer, or Cloud manager, it's good to know Docker and Kubernetes basics. Both technologies help you collaborate with others, deploy your projects, and increase your value to employers.

In this article we'll cover essential Kubernetes concepts.

Kubernetes makes a lot more sense if you first know Docker concepts. Checkout my docker docs

Kubernetes what?

Kubernetes is an open-source platform for managing containerized apps in production. Kubernetes is referred to as K8s for short.

Why is K8s so in demand? Kubernetes makes it easier for you to automatically scale your app, reduce downtime, and increase security. No more writing scripts to check, restart, and change the

number of Docker containers. Instead, you tell K8s your desired number of containers and it does the work for you. K8s can even automatically scale your containers based on resources used.

Kubernetes is all about abstracting away complexity. It provides clear points to interface with your app's development environment.

K8s doesn't make a lot of sense for a basic static website that gets a handful of visitors per day. Its use case is for larger apps that might have to scale up and down quickly.

For large apps, you can use K8s to get the most out of your compute and storage resources. When paired with cloud providers, K8s can save you money. 💰 No matter where you run K8s, it should help you save time and reduce DevOps headaches.

Docker has a competing product named Docker Swarm that orchestrates containers. However, it doesn't have the features and market share that K8s does. While you might think that Docker wouldn't play nicely with K8s when it has its own offering, the two play extremely well together. I strongly suggest you use K8s to orchestrate your containers.

Keeping the many K8s abstractions straight can be tricky. I'll explain how the key parts fit together so you can wrap your head around this powerful platform. Let's explore key K8s concepts and how they relate to each other.

First we'll look at six layers of abstraction and the parts that make them up. Then we'll look at seven other key K8s API objects.

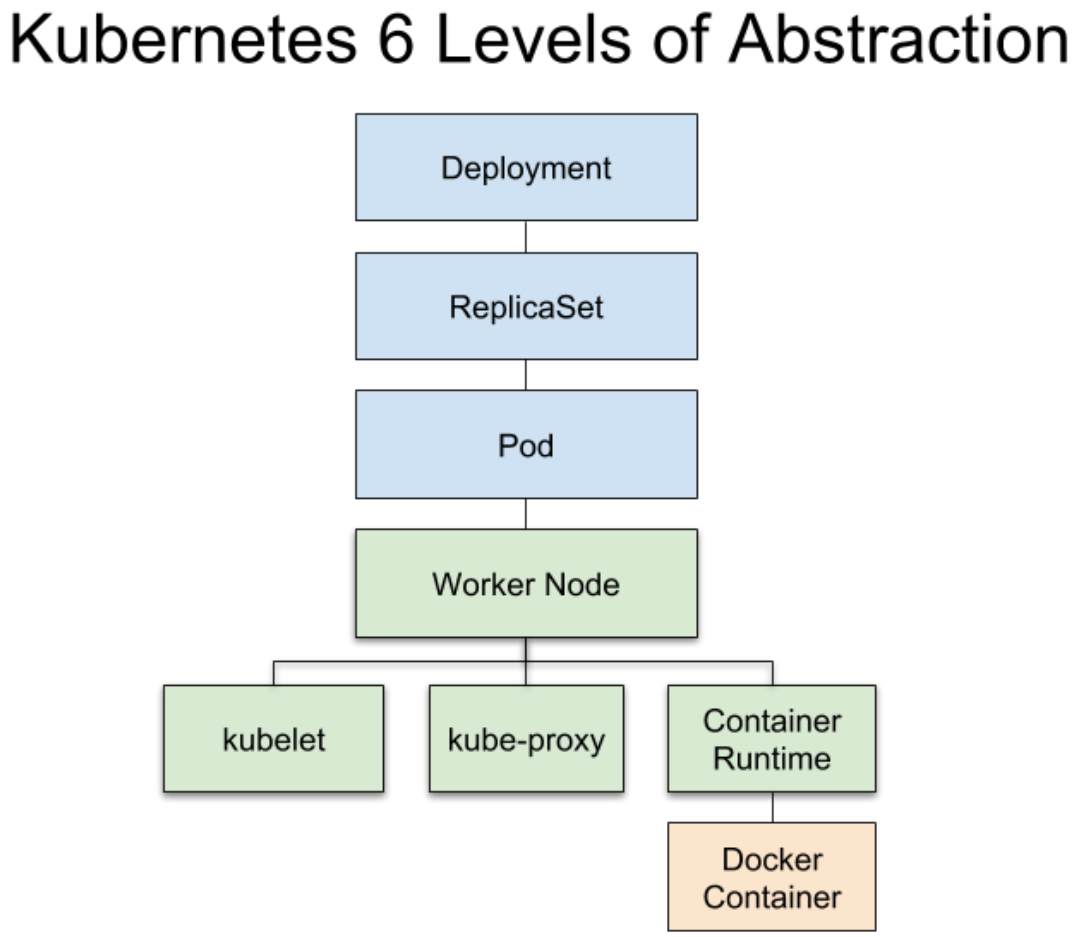
The Six Layers of K8s

Let's assume you have an app that is running continuously and doesn't need to store state.

Here are the six layers of K8s concepts, starting with the highest-level abstraction.

1. Deployment
2. ReplicaSet
3. Pod
4. Node Cluster
5. Node Processes
6. Docker Container

Deployments create and manage ReplicaSets, which create and manage Pods, which run on Nodes, which have a container runtime, which run the app code you put in your Docker image.



The levels shaded blue are higher-level K8s abstractions. The green levels represent Nodes and Node subprocess that you should be aware of, but may not touch.

Note that your K8s instances will often have multiple Pods that can run on a single Node.

The Docker container contains your application code.

Deployment

If you want to make a stateless app that will run continuously, such as an HTTP server, you want a Deployment. Deployments allow you to

update a running app without downtime. Deployments also specify a strategy to restart Pods when they die.

You can create a Deployment from the command line or a configuration file. Please follow me

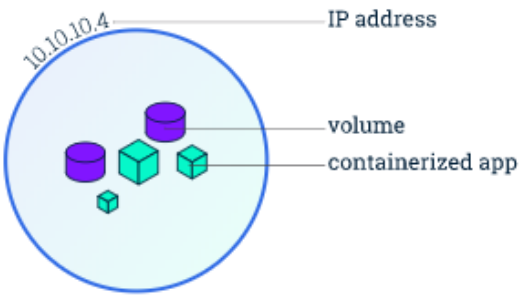
ReplicaSet

The Deployment creates a ReplicaSet that will ensure your app has the desired number of Pods. ReplicaSets will create and scale Pods based on the triggers you specify in your Deployment.

Replication Controllers perform the same function as ReplicaSets, but Replication Controllers are old school. ReplicaSets are the smart way to manage replicated Pods in 2019.

Pod

The *Pod* is the basic building block of Kubernetes. A Pod contains a group of one or more containers. Generally, each Pod has one container.



Pod 4

Pods handle Volumes, Secrets, and configuration for containers.

Pods are ephemeral. They are intended to be restarted automatically when they die.

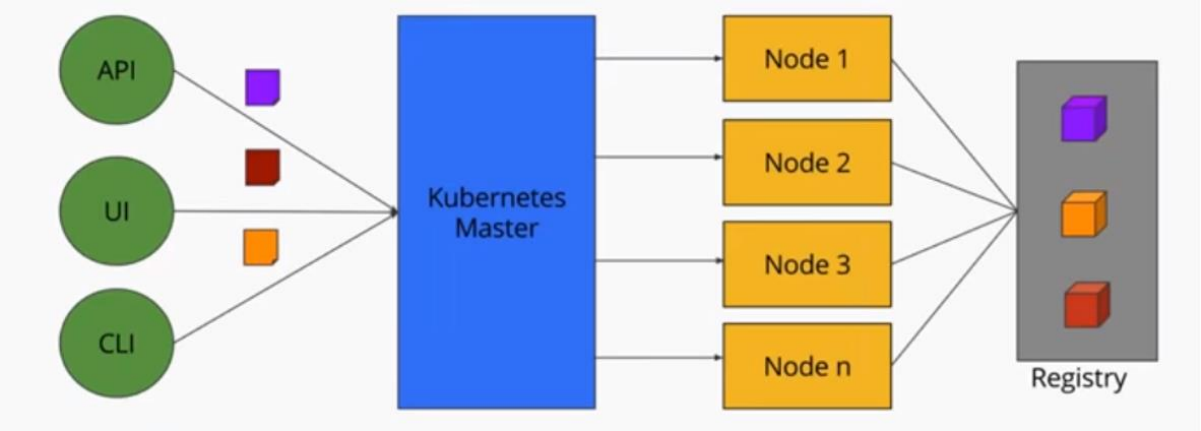
Pods are replicated when the app is scaled horizontally by the ReplicationSet. Each Pod will run the same container code.

Pods live on Worker Nodes.

Cluster Level

Cluster

A K8s Cluster consists of a C*luster Master* and *Worker Nodes.*



Below is a representation of a Cluster. This diagram emphasizes how multiple Pods can run on a Worker Node and multiple Worker Nodes are governed by a Master.

Worker Node

A *Worker Node* is also referred to as a *Node* for short. A Node is an abstraction for a machine — either a physical machine or a virtual machine. Think of a Node as a computer server.

One or more Pods run on a single Worker Node.

A Pod is never split between two Nodes — a Pod's contents are always located and scheduled together on the same Node.

Cluster Master

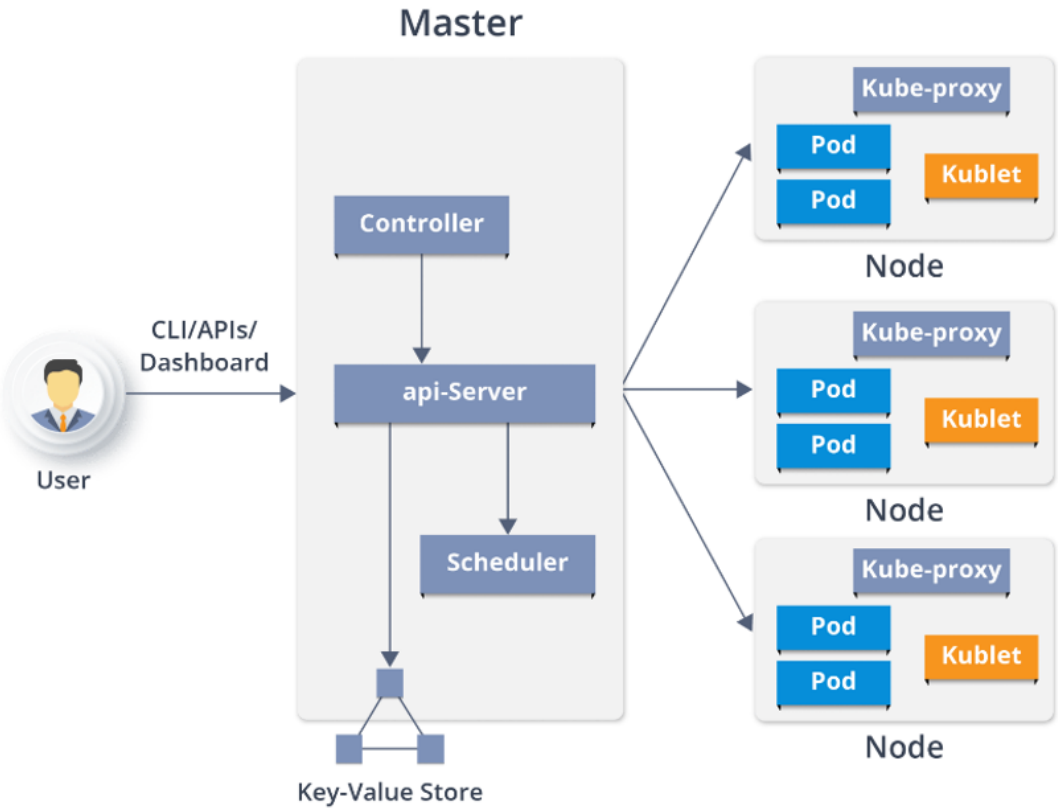
The *Cluster Master* has a seriously ridiculous number of aliases. It's also referred to as the *Master Node, Kubernetes Master, Cluster Control Plane, Control Plane,* and just M*aster.* Whatever you call it, it directs the Worker Nodes. Masters make scheduling decisions, respond to events, implement changes, and monitor the Cluster.

Both the Worker Nodes and the Master have subprocess components.

Node Processes

Master Components

The Master components are the *API server (*aka *kube-apiserver)*, *etcd*, S*cheduler* (aka *kube-scheduler), kube-controller-manager, and cloud-controller manager*. **I added the controller-managers for completeness Apr. 10, 2019**



API Server — Exposes the K8s API. It's the frontend for Kubernetes control.

etcd — Distributed key-value store for Cluster state data.

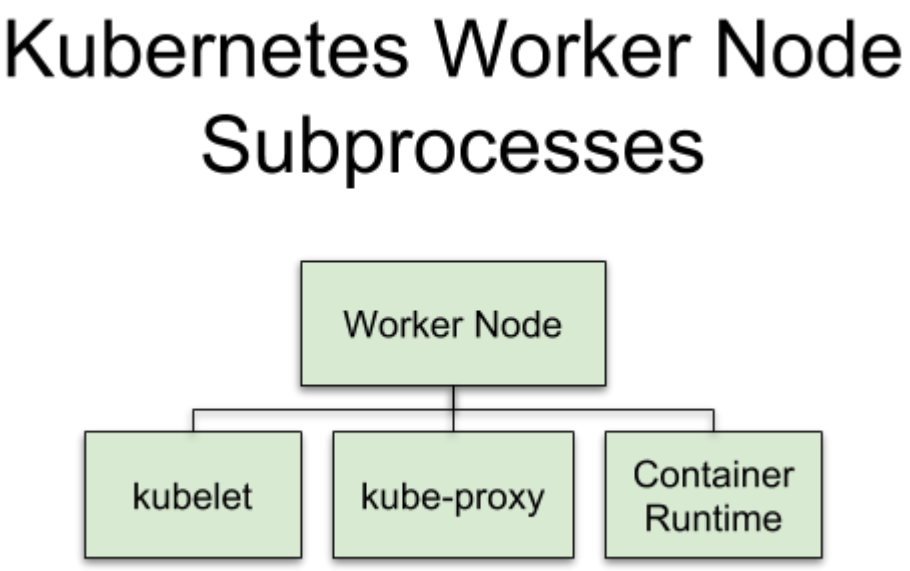
Scheduler — Schedule the pod on the Nodes.

kube-controller-manager —Process that runs controllers to handle Cluster background tasks.

cloud-controller-manager — Runs controllers that interact with cloud providers.

Worker Node Components

The Worker Node's components are the *kubelet*, *kube-proxy*, and *Container Runtime*.

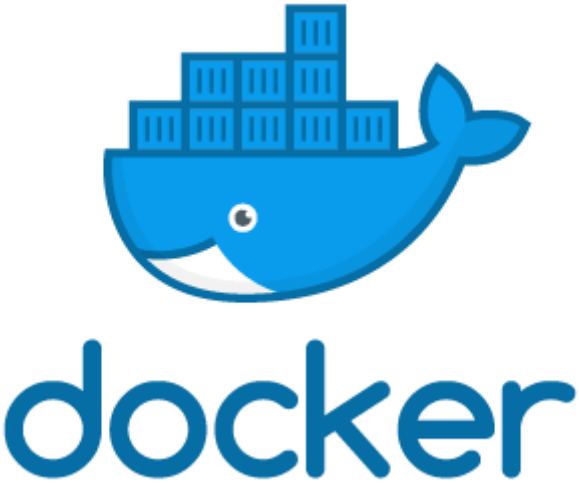


kubelet — Responsible for everything on the Worker Node. It communicates with the Master's API server. Think *brain* for Worker Node.
kube-proxy — Routes connections to the correct Pods. Also performs load balancing across Pods for a Service.

Container Runtime — Downloads images and runs containers. For example, Docker is a Container Runtime. Think *Docker*.

Docker Container Level

Your app needs to be in a container of some sort if you want to run it with K8s. Docker is by far the most common container platform. We'll assume you're using it.



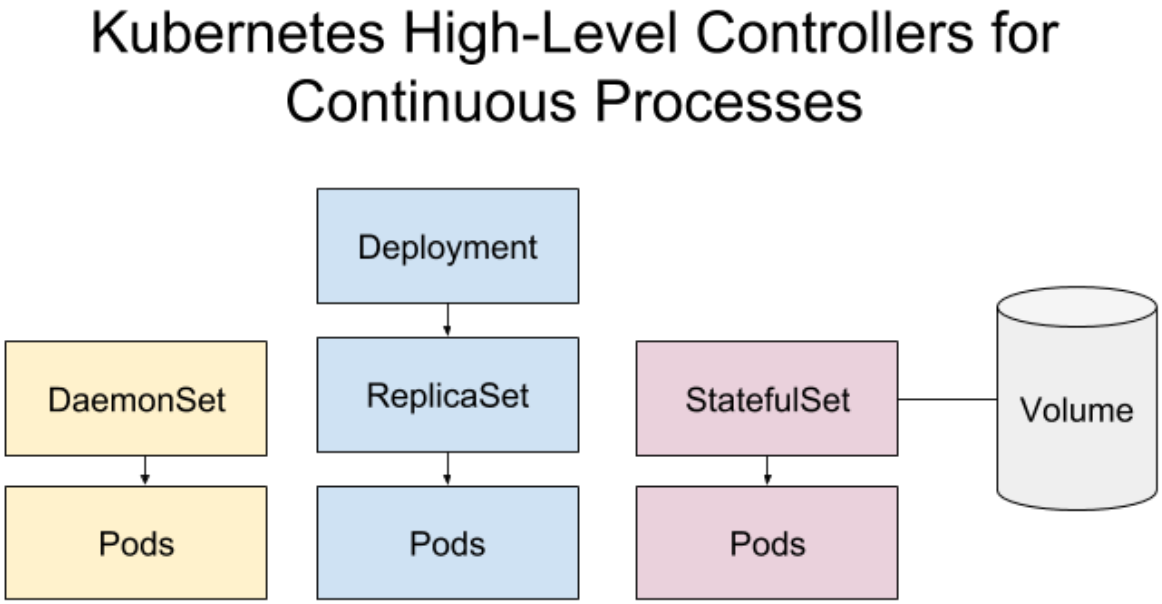
You'll specify which Docker image your Pods should use when you create your Deployment. The Container Runtime will download the image and create the containers.

K8s doesn't create containers directly. It creates Pods that hold containers inside them. The containers in a Pod share any configured resources, such as Volume storage.

There are five high-level K8s API resources that manage and run Pods: *Deployments, StatefulSets, DaemonSets, Jobs,* and *CronJobs*. These objects are responsible for managing and running the Pods that create and run your containers. Let's look at these controllers that create and manage continuous processes

ReplicaSets, StatefulSets, and DaemonSets

As you've seen, a ReplicaSet creates and manages Pods. If a Pod shuts down because a Node fails, a ReplicaSet can automatically replace the Pod on another Node. You should generally create a ReplicaSet through a Deployment rather than creating it directly, because it's easier to update your app with a Deployment.



A Volume can be attached to a ReplicaSet, but it's required for a StatefulSet.

Sometimes your app will need to keep information about its state. You can think of state as the current status of your user's interaction with your app. So in a video game it's all the unique aspects of the user's character at a point in time.

What do you do when your app has state you need to keep track of? Use a StatefulSet.

StatefulSet

Like a ReplicaSet, a *StatefulSet* manages deployment and scaling of a group of Pods based on a container spec. Unlike a Deployment, a StatefulSet's Pods are not interchangeable. Each Pod has a unique, persistent identifier that the controller maintains over any rescheduling. StatefulSets for good for persistent, stateful backends like databases.

The state information for the Pod is held in a Volume associated with the StatefulSet. We'll discuss Volumes in a bit.

DaemonSet

DaemonSets are for continuous process. They run one Pod per Node. Each new Node added to the cluster automatically gets a Pod started by the DaemonSet. DaemonSets are useful for ongoing background tasks such as monitoring and log collection.

StatefulSets and DaemonSets are not controlled by a Deployment. Although they are at the same level of abstraction as a ReplicaSet, there is not a higher level of abstraction for them in the current API.

Now let's look at Jobs and CronJobs.

Service

A K8s *Service* creates a single access point for a group of Pods. A Service provides a consistent IP address and port to access underlying Pods. Both external users and internal Pods use Services to communicate with other Pods.

Services come in a variety of flavors. Networking with K8s a topic worthy of its own guide. Fortunately, there's a good one by me here.

Now let's look at storing data with Volumes and PersistentVolumes.

Volumes, PersistentVolumes, and PersistentVolume Claims

Volumes

A *Volume* is a directory that can hold data. A Volume is a component of a Pod, and is not independent of it. A Volume is created in the Pod specification. A Volume cannot be deleted on its own.

A Volume is made accessible to all the containers in a Pod. Each container that you want to access the Volume must mount it individually.

A K8s Volume outlives any individual containers, but when the enclosing Pod dies, the Volume dies, too. However, the files of some Volume types continue to exist in local or cloud storage, even after the Volume is gone.

Volume

A K8s Volume has more functionality than a Docker volume. A Volume can provide access to local disk storage, memory storage, or cloud storage. A Pod can use a combination of them simultaneously.

K8s Volume types include empty directories, Worker Node's filesystems, and cloud provider-specific storage. For example, awsEleasticBlockStore and gcePersistentDisk are provider-specific options for long-term storage. See more in the docs here.

PersistentVolumes and PersistentVolumeClaims

To help abstract away infrastructure specifics, K8s developed *PersistentVolumes* and *PersistentVolumeClaims*. Unfortunately the names are a bit misleading, because vanilla Volumes can have persistent storage, as well.

PersisententVolumes (PV) and PersisentVolumeClaims (PVC) add complexity compared to using Volumes alone. However, PVs are useful for managing storage resources for large projects.

With PVs, a K8s user still ends up using a Volume, but two steps are required first.

1. A PersistentVolume is provisioned by a Cluster Administrator (or it's provisioned dynamically).
2. An individual Cluster user who needs storage for a Pod creates a *PersistentVolumeClaim* manifest. It specifies how much and what type of storage they need. K8s then finds and reserves the storage needed.

The user then creates a Pod with a Volume that uses the PVC.

PersistentVolumes have lifecycles independent of any Pod. In fact, the Pod doesn't even know about the PV, just the PVC.

PVCs consume PV resources, analogously to how Pods consume Node resources.

Let's recap the K8s concepts we've seen. Here are the six levels of abstraction for a Deployment:

- **Deployment**: manages ReplicaSets. Use for persistent, stateless apps (e.g. HTTP servers).
- **ReplicaSet***:* creates and manages Pods.
- **Pod***:* basic unit of K8s.

- **Node Cluster**: Worker Nodes + Cluster Master.
  - **Worker Nodes**: machines for Pods.
  - **Cluster Master***:* directs worker nodes.

- **Node Processes**
  Master subcomponents:
  - **API server**: hub.
  - **etcd**: cluster info.
  - **scheduler**: matcher.
  - **kube-controller-manager***:* cluster controller.
  - **cloud-controller-manager:** cloud interface.
  Worker Node subcomponents:
  - **kubelet**: Worker Node brain.
  - **kube-proxy**: traffic cop.
  - **container-runtime**: Docker.

- **Docker Container:** where the app code lives.

Here are the 7 additional high-level K8s API objects to know:

- **StatefulSet**: Like a ReplicaSet for stateful processes. Think *state*.

- **DaemonSet:** One automatic Pod per Node. Think *monitor*.

- **Job**: Run a container to completion. Think *batch*.

- **CronJob:** Repeated Job. Think *time*.

- **Service**: Access point for Pods. Think *access point*.

- **Volume**: Holds data. Think *disk.*

- **PersistentVolume, PersistentVolumeClaim:** System for allocating storage. Think *storage claim*.

Wrapping your head around K8s requires understanding many abstract concepts. Also, follow me to make sure you don't miss my future articles on K8s.

Happy Learning!!!