CONTAINERS

Introduction to Kubernetes architecture

If you know only the basics of Kubernetes, you know it's an open source container orchestration platform designed for running distributed applications and services at scale. But you might not understand its components and how they interact.

Let's take a brief look at the design principles that underpin Kubernetes, then explore how the different components of Kubernetes work together.

Kubernetes design principles

The design of a Kubernetes cluster is based on 3 principles, as explained in the Kubernetes implementation details.

A Kubernetes cluster should be:

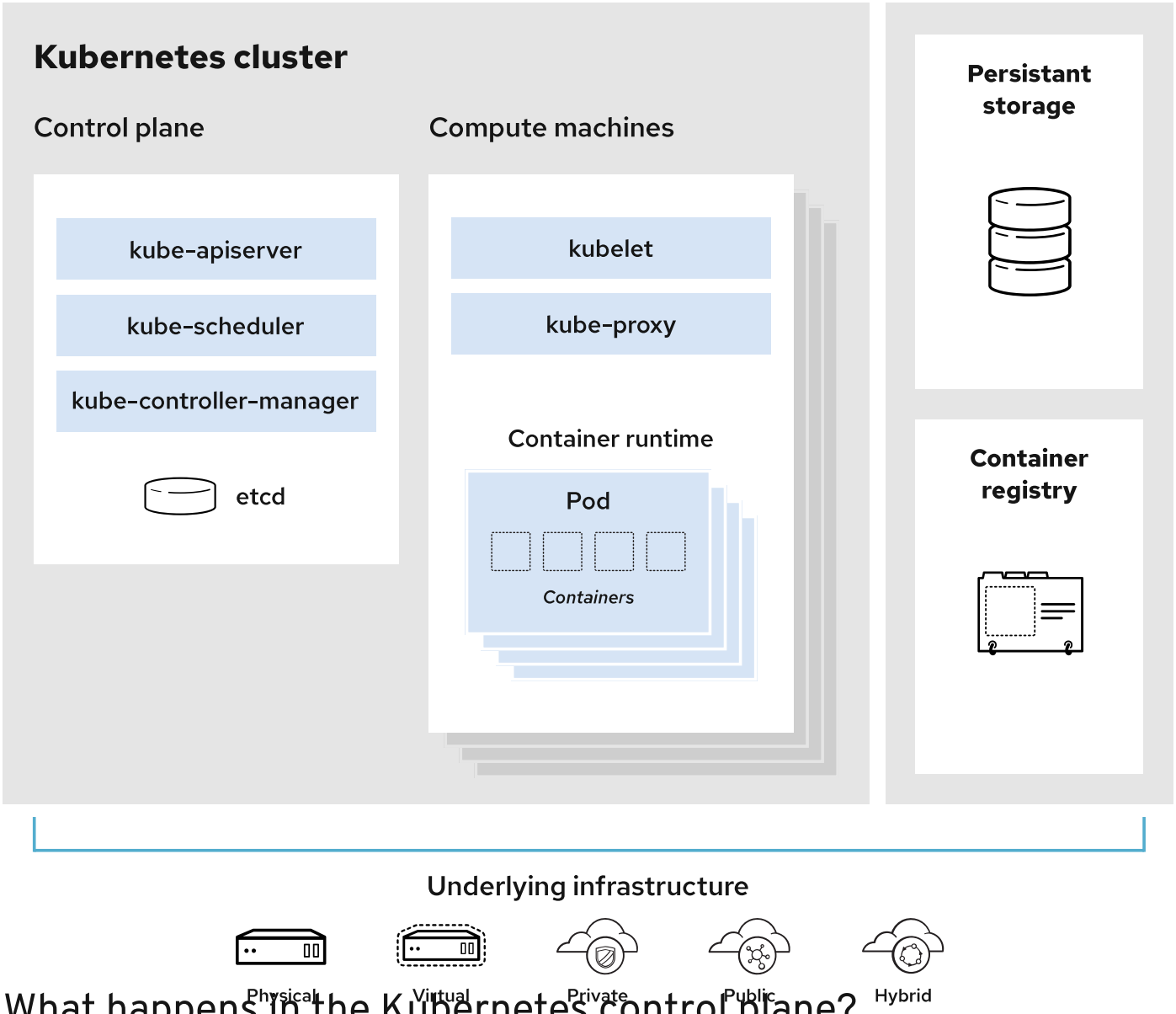
- **Secure.** It should follow the latest security best-practices.
- **Easy to use.** It should be operable using a few simple commands.
- **Extendable.** It shouldn't favor one provider and should be customizable from a configuration file.

Create cloud-native applications with Kubernetes patterns

What are the components of a Kubernetes cluster?

A working Kubernetes deployment is called a cluster. You can visualize a Kubernetes cluster as two parts: the control plane and the compute machines, or nodes. Each node is its own Linux®environment, and could be either a physical or virtual machine. Each node runs pods, which are made up of containers.

This diagram shows how the parts of a Kubernetes cluster relate to one another:



What happens in the Kubernetes control plane?

Control plane

Let's begin in the nerve center of our Kubernetes cluster: The control plane. Here we find the Kubernetes components that control the cluster, along with data about the cluster's state and configuration. These core Kubernetes components handle the important work of making sure your containers are running in sufficient numbers and with the necessary resources.

The control plane is in constant contact with your compute machines. You've configured your cluster to run a certain way. The control plane makes sure it does.

kube-apiserver

Need to interact with your Kubernetes cluster? Talk to the API. The Kubernetes API is the front end of the Kubernetes control plane, handling internal and external requests. The API server determines if a request is valid and, if it is, processes it. You can access the API through REST calls, through the kubectl command-line interface, or through other command-line tools such as kubeadm.

kube-scheduler

Is your cluster healthy? If new containers are needed, where will they fit? These are the concerns of the Kubernetes scheduler.

The scheduler considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster. Then it schedules the pod to an appropriate compute node.

kube-controller-manager

Controllers take care of actually running the cluster, and the Kubernetes controller-manager contains several controller functions in one. One controller consults the scheduler and makes sure the correct number of pods is running. If a pod goes down, another controller notices and responds. A controller connects services to pods, so requests go to the right endpoints. And there are controllers for creating accounts and API access tokens.

etcd

Configuration data and information about the state of the cluster lives in etcd, a key-value store database. Fault-tolerant and distributed, etcd is designed to be the ultimate source of truth about your cluster.

What happens in a Kubernetes node?

Nodes

A Kubernetes cluster needs at least one compute node, but will normally have many. Pods are scheduled and orchestrated to run on nodes. Need to scale up the capacity of your cluster? Add more nodes.

Pods

A pod is the smallest and simplest unit in the Kubernetes object model. It represents a single instance of an application. Each pod is made up of a container or a series of tightly coupled containers, along with options that govern how the containers are run. Pods can be connected to persistent storage in order to run stateful applications.

Container runtime engine

To run the containers, each compute node has a container runtime engine. Docker is one example, but Kubernetes supports other Open Container Initiative-compliant runtimes as well, such as rkt and CRI-O.

kubelet

Each compute node contains a kubelet, a tiny application that communicates with the control plane. The kublet makes sure containers are running in a pod. When the control plane needs something to happen in a node, the kubelet executes the action.

kube-proxy

Each compute node also contains kube-proxy, a network proxy for facilitating Kubernetes networking services. The kube-proxy handles network communications inside or outside of your cluster—relying either on your operating system's packet filtering layer, or forwarding the traffic itself.

What else does a Kubernetes cluster need?

Persistent storage

Beyond just managing the containers that run an application, Kubernetes can also manage the application data attached to a cluster. Kubernetes allows users to request storage resources without having to know the details of the underlying storage infrastructure. Persistent volumes are specific to a cluster, rather than a pod, and thus can outlive the life of a pod.

Container registry

The container images that Kubernetes relies on are stored in a container registry. This can be a registry you configure, or a third party registry.

Underlying infrastructure

Where you run Kubernetes is up to you. This can be bare metal servers, virtual machines, public cloud providers, private clouds, and hybrid cloud environments. One of Kubernetes's key advantages is it works on many different kinds of infrastructure.

Nobody said this would be easy

This simplified overview of Kubernetes architecture just scratches the surface. As you consider how these components communicate with each other—and with external resources and infrastructure—you can appreciate the challenges of configuring and securing a Kubernetes cluster.

Kubernetes offers the tools to orchestrate a large and complex containerized application, but it also leaves many decisions up to you. You choose the operating system, container runtime, continuous integration/continuous delivery (CI/CD) tooling, application services, storage, and most other components. There's also the work of managing roles, access control, multitenancy, and secure default settings. Additionally, you can choose to run Kubernetes on your own or work with a vendor who can provide a supported version.

This freedom of choice is part of the flexible nature of Kubernetes. While it can be complex to implement, Kubernetes gives you tremendous power to run containerized applications on your own terms, and to react to changes in your organization with agility.

Build cloud-native applications with Kubernetes

Watch this webinar series to get expert perspectives to help you establish the data platform on enterprise Kubernetes you need to build, run, deploy, and modernize applications.