KUBERNETES GLOSSARY FOR EXECUTIVES

As Kubernetes becomes a bigger topic in your organization, you'll need to discuss it with technical and non-technical audiences.

Here's a primer on key Kubernetes terminology for IT and business leaders. For a more comprehensive guide on services, networking, and load balancing, we recommend the Kubernetes documentation.

Cluster:

You can begin to understand this major piece literally: A cluster is a group or bunch of nodes that run your containerized applications. You manage the cluster and everything it includes – in other words, you manage your application(s) – with Kubernetes.

Node:

Nodes are comprised of physical or virtual machines on your cluster; these "worker" machines have everything necessary to run your application containers, including the container runtime and other critical services. (The Kubernetes Github repository has a good, detailed breakdown of the Kubernetes node.)

Pod:

This is essentially the smallest deployable unit of the Kubernetes ecosystem; more accurately, it's the smallest object. A pod specifically represents a group of one or more containers running together on your cluster.

Kubernetes API:

The Kubernetes API is the lifeblood of the system. You may have heard of Kubernetes described as a "declarative" tool – in other words, Kubernetes lets you say "this is how I want things to run," and then it does what's needed to make that happen in a highly automated way. The Kubernetes API helps make that a reality. The official Kubernetes site defines the Kubernetes API as "the application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster."

Kubernetes Control Plane:

This sits between a cluster and Kubernetes basically as a necessary intermediary; it makes sure everything behaves properly – like a chaperon at a container dance party. When people extol automation as one of the key benefits of Kubernetes and container orchestration, this is a key piece. Says the Kubernetes official site: "The Control Plane maintains a record of all of the Kubernetes Objects in the system, and runs continuous control loops to manage those objects' state." The control plane continuously checks and rechecks that everything matches your desired state. In general, the job of a controller in Kubernetes – there are multiple types – is to take actions needed to manage a specific type of resource. (We'll call out one key piece of the control plane next.)

Master:

The Kubernetes master maintains the desired state of your cluster; you will commonly see it referred to as the master node. Every cluster has a master node, as well as several "worker" nodes. The master includes three critical processes for managing the state of your cluster: kube-apiserver, kube-controller-manager and kube-scheduler. When you make changes, you're almost always making them to the master node, not to each individual node in a cluster.

kubectl:

Simply put, kubectl is a command line interface (CLI) for managing operations on your Kubernetes clusters. It does so by communicating with the Kubernetes API. (It's not a typo, either: The official Kubernetes style, as it were, is to lowercase the k in kubectl.) It follows a standard syntax for running commands: kubectl [command] [TYPE] [NAME] [flags].

You can find an in-depth explanation of kubectl here, as well as examples of common operations, but here's a basic example of an operation: "run." This command runs a particular container image on your cluster.

Minikube:

Minikube is a tool for running Kubernetes on a local machine such as a laptop. You'll likely hear more about it as more individuals begin doing hands-on tinkering to get up to speed with Kubernetes; it can also be used by developers for working on local machines. Per the official documentation, Minikube runs a single-node cluster inside a VM on your local machine.

Volume:

A volume is simply a directory of data; it lives within a pod and can be accessed by any container running in that pod. A volume is the abstraction that lets Kubernetes deal with the ephemeral nature of containers; when a container is retired, the volume (and its data) continues to exist within the pod, still accessible to other containers. It exists as long as its pod exists; once the latter "dies," so does the volume and its data.

Persistent Volume:

Speaking of ephemerality and data: Persistent volumes deal with the issue of storage that needs to exist outside of the lifetime of any particular container or application, whereas general volumes deal with compute. This becomes particularly important when you're discussing stateful applications like databases.

Want more? You may also need to boost your broader knowledge of the technologies driving Kubernetes adoption. Consider these recommended reads:

How to explain containers in plain English »

How to explain microservices in plain English »

How to explain Kubernetes in plain English »

How to explain Kubernetes Operators in plain English »

How to explain cloud-native apps in plain English »

How to explain APIs in plain English »