What are Kubernetes Clusters vs. Nodes vs. Pods vs. Containers vs. Containerized Applications?

A Kubernetes environment consists of several components, hardware and software, which all work together to manage the deployment and execution of containerized applications. Here are the key components and how they fit into the picture.

vSphere with Tanzu 101 - An Introduction for vSphere Admir

DOWNLOAD NOW

Choose the Right VMware Solution to Run Your Kubernetes W

LEARN MORE

What are containerized applications?

Containerized applications are bundled with their required libraries, binaries, and configuration files into a container. Not every application is a perfect candidate for containerization. Many developers adhere to the Twelve-Factor App guidelines for cloud-native applications:

- I. Codebase: One codebase tracked in revision control, many deploys
- II. Dependencies: Explicitly declare and isolate dependencies
- III. Config: Store config in the environment
- IV. Backing services: Treat backing services as attached resources
- V. Build, release, run: Strictly separate build and run stages
- VI. Processes: Execute the app as one or more stateless processes
- VII. Port binding: Export services via port binding
- VIII. Concurrency: Scale out via the process model
- IX. Disposability: Maximize robustness with fast startup and graceful shutdown
- X. Dev/prod parity: Keep development, staging, and production as similar as possible
- XI. Logs: Treat logs as event streams
- XII. Admin processes: Run admin/management tasks as one-off processes

Many popular languages and applications have been containerized and are in open source repositories, however it may be more efficient to build an application container with only the libraries and binaries required to run the application, rather than importing everything available. Creating containers can be programmatic, enabling continuous integration and deployment (CI/CD) pipelines to be created for efficiency. Containerized applications are in the developers' domain.

What are Kubernetes containers?

Containers are standardized, self-contained execution enclosures for applications. Typically, a container will include a single application, often composed of microservices, along with the binaries and libraries needed to execute properly. By limiting containers to a single process, diagnosis of problem is easier, as is updating applications. Unlike VMs, containers do NOT contain the underlying operating system, and thus considered lightweight as compared to VMs. Kubernetes containers are in the developers' domain.

What are Kubernetes pods?

Pods are the smallest execution unit in a Kubernetes cluster. In Kubernetes, containers do not run directly on cluster nodes; instead one or more containers are encased in a pod. All applications in a pod share the same resources and local network, easing communications between applications in a pod. Pods utilize an agent on each node called a kubelet to communicate with the Kubernetes API and the rest of the cluster. Although developers need API access, management of pods is transitioning to the domain of DevOps.

As the load on a pod increases, Kubernetes can automatically replicate the pod to achieve desired scalability. Thus it is important to design a pod to be lean as possible. Pods should contain a single main process along with any help or 'side-car' containers necessary for their execution.

What is the difference between containers vs. pods?

Containers encompass the code required to execute a specific process or function. Before Kubernetes, organizations would run containers directly on a physical or virtual server, but without the scalability and flexibility offered by a Kubernetes cluster.

Pods offer another level of abstraction for containers. One or more application can be wrapped into a pod (think peas in a pod), and the pod is the smallest unit of execution in a Kubernetes cluster. For example, pods can contain initialization containers that prepare the environment for the containerized application code and then terminate before the application container begins execution. Pods are the smallest unit of replication in a cluster, so all containers in a pod will scale up or down together.

Pods include persistent storage volumes as well as containers, if access to persistent storage is necessary for the application.

Just as the pod is the smallest execution unit in Kubernetes, the node is the smallest unit of compute hardware in a Kubernetes cluster. Nodes can be physical on-premises servers, or VMs that reside either on-premises or at a cloud provider.

Like containers, nodes provide a layer of abstraction. If operations teams think of a node as simply a resource with processing power and memory, each node becomes interchangeable with the next. Working together, nodes form the Kubernetes cluster, which automates distributing workloads as demands change. If a node fails, it is automatically removed from the cluster and other nodes take over. Every node runs an agent called kubelet, which communicates with the cluster control plane.

Nodes are the domain of DevOps and IT.

What is the difference between Kubernetes pods vs. nodes?

Pods are an abstraction of executable code, nodes are abstractions of computer hardware, so the comparison is a bit apples-and-oranges.

Pods are simply the smallest unit of execution in Kubernetes, consisting of one or more containers, each with one or more application and its binaries.

Nodes are the physical servers or VMs that comprise a Kubernetes Cluster. Nodes are interchangeable and typically not addressed individually by users or IT, other than when maintenance is required.

What is a Kubernetes Control Plane?

- The Kubernetes control plane is the controller for a Kubernetes cluster. Although most clusters will have a single control plane, there can be multiple for resiliency. For example, in a large cloud deployment that spans availability zones, there may be a control plane running in each availability zone. These are the components of the Kubernetes control plane:
- **apiserver**: As its name suggests the API server exposes the Kubernetes API, which is communications central. External communications via command line interface (CLI) or other user interface (UI) pass to the kube-apiserver, and all control plane to node communications also goes through the API server.
- **etcd**: The key value store where all data relating to the cluster is stored. etcd is highly available and consistent since all access to etcd is through the API server. Information in etcd is generally formatted in human-readable YAML (YAML Ain't Markup Language).
- **scheduler**: When a new Pod is created, this component assigns it to a node for execution based on resource requirements, policies, and 'affinity' specifications regarding geolocation and interference with other workloads.
- **controller-manager**: Although a Kubernetes cluster has several controller functions, they are all compiled into a single binary.

What is a Kubernetes Cluster?

A Kubernetes cluster is comprised of nodes, which can be either VMs or physical servers. When you use Kubernetes, you are always managing a cluster. There must be at least one instance of the Kubernetes control plane running on a node, and at least one node for pods to execute on. Typically, the cluster will have multiple nodes to handle the scaling of applications as workloads change, whether due to time of day, seasonality, or other reason.

If nodes are added or subtracted from the cluster, the cluster will automatically redistribute the workload as necessary.

What is the difference between Kubernetes Nodes vs. Clusters?

A node is the smallest element of a cluster. A cluster is comprised nodes. The cluster is a collective that shares overall execution of pods, reflected in the original name for the Google Kubernetes cluster project: Borg.

What are Kubernetes volumes?

Since containers were originally designed to be ephemeral and stateless, there was little need to address storage persistence. However, as more applications requiring reading and writing from persistent storage are containerized, the need to have access to persistent storage volumes has emerged.

To achieve this, Kubernetes has Persistent Volumes. Unique in that they are external to the cluster, Persistent Volumes can be mounted to the cluster without the need to associate them with a particular node, container, or pod.

Persistent Volumes can be either local or cloud-based, and are the domain of DevOps and IT.

How do the components of Kubernetes work together?

Simply put, applications are created or migrated to containers, which are then used to create pods that run on a Kubernetes cluster.

Once pods are created, Kubernetes assigns them to one or more nodes in the cluster, and ensures the correct number of replica pods are running. Kubernetes scans the cluster to ensure each set of pods is running as specified.

Recommended for You

- Kubernetes
- Kubernetes Namespace
- Kubernetes Networking
- Kubernetes Deployment

Related Solutions and Products

vSphere

Server virtualization software

vSphere with Tanzu

Run Kubernetes workloads using your existing IT infrastructure

Skip to main content

VMware Tanzu Kubernetes Grid

Streamline operations across multi-cloud infrastructure.

Also of Interest

- What are Kubernetes Pods?
- What are Kubernetes Services?
- What is a Kubernetes Namespace?