What is Kubernetes Architecture?

Kubernetes is an architecture that offers a loosely coupled mechanism for service discovery across a cluster. A [Kubernetes cluster](https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html) has one or more control planes, and one or more compute nodes. Overall, the control plane is responsible for managing the overall cluster, exposing the application program interface (API), and for scheduling the initiation and shutdown of compute nodes based on a desired configuration. Each of the compute nodes runs a container runtime like Docker along with an agent, kubelet, which communicates with the control plane. Each node can be bare metal servers, or on-premises or cloud-based [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine.html) (VMs).

What are Kubernetes architecture components?

The main components of a Kubernetes cluster include:

**Nodes:** Nodes are VMs or physical servers that host containerized applications. Each node in a cluster can run one or more application instance. There can be as few as one node, however, a typical Kubernetes cluster will have several nodes (and deployments with hundreds or more nodes are not uncommon).

**Image Registry**: Container images are kept in the registry and transferred to nodes by the control plane for execution in container pods.

**Pods**: Pods are where containerized applications run. They can include one or more containers and are the smallest unit of deployment for applications in a Kubernetes cluster.

What is Kubernetes Control Plane architecture?

A Kubernetes control plane is the control plane for a Kubernetes cluster. Its components include:

* **kube-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 interfaces (UI) pass to the kube-apiserver, and all control planes 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 (which stands for the recursive “YAML Ain’t Markup Language”).
* **kube-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.
* **kube-controller-manager**: Although a Kubernetes cluster has several controller functions, they are all compiled into a single binary known as kube-controller-manager.

Controller functions included in this process include:

* **Replication controller**: Ensures the correct number of pods is in existence for each replicated pod running in the cluster
* **Node controller**: Monitors the health of each node and notifies the cluster when nodes come online or become unresponsive
* **Endpoints controller**: Connects Pods and Services to populate the Endpoints object
* **Service Account and Token controllers**: Allocates API access tokens and default accounts to new namespaces in the cluster
* **cloud-controller-manager**: If the cluster is partly or entirely cloud-based, the cloud controller manager links the cluster to the cloud provider’s API. Only those controls specific to the cloud provider will run. The cloud controller manager does not exist on clusters that are entirely on-premises. More than one cloud controller manager can be running in a cluster for [fault tolerance](https://www.vmware.com/products/vsphere/fault-tolerance.html) or to improve overall cloud performance.

Elements of the cloud controller manager include:

* **Node controller**: Determines status of a cloud-based node that has stopped responding, i.e., if it has been deleted
* **Route controller**: Establishes routes in the cloud provider infrastructure
* **Service controller**: Manages cloud provider’s load balancers

What is Kubernetes node architecture?

Nodes are the machines, either VMs or physical servers, where Kubernetes place Pods to execute. Node components include:

**kubelet**: Every node has an agent called kubelet. It ensures that the container described in PodSpecs are up and running properly.

**kube-proxy**: A network proxy on each node that maintains network nodes which allows for the communication from Pods to network sessions, whether inside or outside the cluster, using operating system (OS) packet filtering if available.

**container runtime**: Software responsible for running the containerized applications. Although Docker is the most popular, Kubernetes supports any runtime that adheres to the Kubernetes CRI (Container Runtime Interface).

What are other Kubernetes infrastructure components?

**Pods**: By encapsulating one (or more) application containers, pods are the most basic execution unit of a Kubernetes application. Each Pod contains the code and storage resources required for execution and has its own IP address. Pods include configuration options as well. Typically, a Pod contains a single container or few containers that are coupled into an application or business function and that share a set of resources and data.

**Deployments**: A method of deploying containerized application Pods. A desired state described in a Deployment will cause controllers to change the actual state of the cluster to achieve that state in an orderly manner. Learn more about Kubernetes Deployments.

**ReplicaSet**: Ensures that a specified number of identical Pods are running at any given point in time.

**Cluster DNS**: serves DNS records needed to operate Kubernetes services.

**Container Resource Monitoring**: Captures and records container metrics in a central database.

What are Kubernetes architecture best practices and design principles?

Gartner’s Container Best Practices suggest a platform strategy that considers security, governance, monitoring, storage, networking, container lifecycle management and orchestration like Kubernetes.

Here are some best practices for architecting Kubernetes clusters:

* Ensure you have updated to the **latest Kubernetes version** (1.18 as of this writing).
* **Invest up-front in training** for developer and operations teams.
* **Establish governance** enterprise-wide. Ensure tools and vendors are aligned and integrated with Kubernetes orchestration.
* **Enhance security by integrating image-scanning processes** as part of your CI/CD process, scanning during build and run phases. Open-source code pulled from a Github repository should always be considered suspect.
* **Adopt role-based access control (RBAC) across the cluster.** Least privilege, zero-trust models should be the standard.
* Further secure containers by **using only non-root users** and **making the file system read-only.**
* **Avoid use of default value**, since simple declaratives are less error-prone and demonstrate intent more clearly.
* **Be careful when using basic Docker Hub images**, which can contain malware or be bloated with unnecessary code. Start with lean, clean code and build packages up from there. Small images build faster, are smaller on disk, and image pulls are faster as well.
* **Keep containers simple**. One process per container will let the orchestrator report if that one process is healthy or not.
* **When in doubt, crash**. Kubernetes will restart a failed container, so do not restart on failure.
* **Be verbose**. Descriptive labels help current developers and will be invaluable to developers to follow in their footsteps.
* **Don’t get too granular with microservices**. Not every function within a logical code component need be its own microservice.
* **Automate, where it makes sense**. Automating CI/CD pipeline lets you avoid manual Kubernetes deployments entirely.
* **Use livenessProbe and readinessProbe** to help manage Pod lifecycles, or pods may end up being terminated while initializing or begin receiving user requests before they are ready.

**Kubernetes architecture** is straightforward and intuitive. The loose coupling between control plane and node allows for nearly infinite flexibility and the ability for an application to scale out virtually instantaneously to meet changing needs, to migrate users to new builds, and to support migration from on-premises to cloud-based nodes or between multiple clouds to take advantage of desired features of each cloud provider.

# What is Kubernetes infrastructure?

**Kubernetes infrastructure** is the combination of resources (including servers, physical or [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine.html), cloud platforms and more) that supports a Kubernetes environment.

[Kubernetes](https://www.vmware.com/topics/glossary/content/kubernetes.html" \t "_self) is a popular open source platform for container orchestration, which is the practice of automating many of the operational tasks required of a container’s lifecycle, from deployment to retirement. ]

Under the hood, Kubernetes’ infrastructure and architecture revolves around the concept of a cluster, which is a set of machines called “nodes” in Kubernetes terminology. When you deploy containerized workloads with Kubernetes, you deploy them onto the cluster. Nodes represent the machines, which may be physical or virtual machines, responsible for running your containerized workloads. Every [Kubernetes cluster](https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html) has a master node and at least one such worker node, though typically a cluster will have several or more worker nodes.

Another important Kubernetes concept is the “pod.” Per the official documentation, a pod is the smallest deployable unit in Kubernetes, and they run on the nodes of your cluster. To think of it a different way, the pods represent the various components of your application. A pod often runs a single container, though it can run multiple containers in certain circumstances.

Finally, another fundamental element of Kubernetes cluster architecture is the control plane. This includes the API server and four other components that effectively manage your nodes (or machines) according to your desired state.

## What are Kubernetes infrastructure requirements?

The infrastructure requirements necessary to run Kubernetes depend on what specifically you’re using it for. You can run a version of Kubernetes in a virtual machine on a laptop, for example, for development or testing purposes. You can similarly set up an environment quickly on a hosted service. Many production use cases, however, will require more substantial resources to support the applications you’re running. Large-scale, highly distributed systems might run across multiple cloud and on-premises servers to achieve performance goals or ensure high availability. These scenarios would require a greater investment in appropriate infrastructure resources.

## What is infrastructure as code and Kubernetes?

[Infrastructure as code](https://www.vmware.com/topics/glossary/content/infrastructure-as-code.html) is the practice of provisioning and operating modern infrastructure using a programming language, similar to how software applications are developed and maintained.

Sometimes referred to as [configuration management](https://www.vmware.com/topics/glossary/content/configuration-management.html) tools, infrastructure as code tools introduce more automation to traditional IT operations and allow [DevOps](https://www.vmware.com/topics/glossary/content/devops.html) teams, in particular, to focus more on optimizing areas like development time, deployment frequency, system performance, reliability, resiliency and security. They are able to do this because infrastructure as code allows them to spend less time on repetitive infrastructure management tasks.

Because Kubernetes similarly automates a lot of the operational effort required to run containerized applications, some people think of it as infrastructure as code, even though it is actually considered a [container orchestration](https://www.vmware.com/topics/glossary/content/container-orchestration.html) tool. Like infrastructure as code, Kubernetes allows you to declare a desired state for your cluster and automate many of the tasks required to achieve or return to that state. Some organizations and teams use an infrastructure as code (or configuration management) tool in conjunction with Kubernetes for needs such as managing their cloud infrastructure.

## How to secure Kubernetes

Kubernetes security is one facet of an overall approach to [container security](https://www.vmware.com/topics/glossary/content/container-security.html). The aim of this latter field is ultimately to secure all aspects of the software development lifecycle (including your CI/CD pipeline, if applicable).

Kubernetes itself includes many important security features, though they typically need to be configured (and optimized over time) for your environment and threat model in order to minimize risks.

A good example of this is Role-Based Access Control, or RBAC. This gives administrators granular control over what users are able to access and do within the environment, which can reduce the risk of privilege escalation attacks and other threats. Older versions of Kubernetes, however, did not turn on RBAC by default. Even if you’re running the latest version, you’ll want to ensure your settings are properly set for each user, and only grant access or privileges that are absolutely necessary. It’s also a good idea to review these on a regular basis.

The Cloud Native Computing Foundation, the organization that houses the Kubernetes open source project, published a [blog post](https://www.cncf.io/blog/2019/01/14/9-kubernetes-security-best-practices-everyone-must-follow) on nine different best practices for [Kubernetes security](https://www.vmware.com/topics/glossary/content/kubernetes-security.html)—including properly setting up RBAC and making sure you’re using the most recent version. (This is a general best practice with any software platform.) They also advise making use of the [Kubernetes namespaces](https://www.vmware.com/topics/glossary/content/kubernetes-namespace.html) feature for increased isolation, as well as hardening the nodes running on your cluster by turning off access to sensitive ports and limiting access to the Kubernetes API.

Security and infrastructure operations experts also usually advise making sure you’ve got proper tools for monitoring and logging in place when running containerized applications in production environments. This is to help ensure you retain the necessary visibility and auditability of your environments as they become increasingly complex.

## Best ways to manage Kubernetes

There are multiple options for implementing and managing Kubernetes. The best choice depends on your use cases, available skills and resources and other factors. Getting up and running with the open source platform can require a lot of internal expertise. In general, Kubernetes and cloud-native development are often best for teams that have already adopted DevOps and other modern software practices and technologies.

Organizations that lack the necessary internal skills to effectively manage Kubernetes on their own may want to consider one of the commercial or hosted options that are based on the underlying open source platform. There are also many options for complementary tools (such as infrastructure as code and security tools) and third-party support.

What is a Kubernetes cluster?

A **Kubernetes cluster** is a set of nodes that run containerized applications. Containerizing applications packages an app with its dependences and some necessary services. They are more lightweight and flexible than virtual machines. In this way, Kubernetes clusters allow for applications to be more easily developed, moved and managed.

Kubernetes clusters allow containers to run across multiple machines and environments: virtual, physical, cloud-based, and on-premises. Kubernetes containers are not restricted to a specific operating system, unlike virtual machines. Instead, they are able to share operating systems and run anywhere.

Kubernetes clusters are comprised of one master node and a number of worker nodes. These nodes can either be physical computers or virtual machines, depending on the cluster.

The **master node** controls the state of the cluster; for example, which applications are running and their corresponding container images. The master node is the origin for all task assignments. It coordinates processes such as:

* **Scheduling** and scaling applications
* **Maintaining** a cluster’s state
* **Implementing** updates

The **worker nodes** are the components that run these applications. Worker nodes perform tasks assigned by the master node. They can either be [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine.html) or physical computers, all operating as part of one system.

There must be a minimum of one master node and one worker node for a Kubernetes cluster to be operational. For production and staging, the cluster is distributed across multiple worker nodes. For testing, the components can all run on the same physical or virtual node.

A **namespace** is a way for a Kubernetes user to organize many different clusters within just one physical cluster. Namespaces enable users to divide cluster resources within the physical cluster among different teams via resource quotas. For this reason, they are ideal in situations involving complex projects or multiple teams.

What is Kubernetes Networking?

Kubernetes (sometimes referred to as K8s) is an open-source platform that is used to manage and automate the deployment, scheduling, monitoring, maintenance, and operation of application containers across a cluster of machines.

Developed by Google, networking with Kubernetes allows administrators to move workloads across private, public, and [hybrid cloud](https://www.vmware.com/topics/glossary/content/hybrid-cloud.html) infrastructures. Developers use Kubernetes to package software applications with their required infrastructure and deploy new versions quickly.

Kubernetes networking allows [Kubernetes components](https://www.vmware.com/topics/glossary/content/components-kubernetes.html) to communicate with each other and with other applications. The Kubernetes platform is different from other networking platforms because it is based on a flat network structure that eliminates the need to map host ports to container ports. The Kubernetes platform provides a way to run distributed systems, sharing machines between applications without dynamically allocating ports.

What is a Kubernetes Pod Network?

A Kubernetes Pod network connects several interrelated components:

* **Pods:** [Kubernetes Pods](https://www.vmware.com/topics/glossary/content/kubernetes-pods.html) are inspired by pods found in nature (pea pods or whale pods). The Pods are groups of containers that share networking and storage resources from the same node. They are created with an API server and placed by a controller. Each Pod is assigned an IP address, and all the containers in the Pod share the same storage, IP address, and port space (network namespace).
* **Containers**: A Kubernetes container is like a [virtual machine](https://www.vmware.com/topics/glossary/content/virtual-machine.html) that shares its Operating System (OS) among several applications. It has its own filesystem, CPU, memory, and process space. Containers are always created in Pods, and multiple containers can be created in one Pod. The containers in a Pod all move together, are scheduled together, and are terminated together. The Pod can move across clouds. Containers can connect through container networking to other containers, a host, and external networks.
* **Nodes**: Pods always run inside a worker node. Nodes are machines that run containerized applications. Kubernetes groups these nodes in clusters.
* **Master node**: Each cluster of nodes will have at least one master node that manages the worker nodes. The master can communicate with each node in a cluster or it can communicate directly to any individual Pod.

How does Networking work in Kubernetes?

The different components in the Kubernetes platform (Pods, containers, nodes, applications) use different networking methods to communicate. There is container-to-container communication, Pod-to-Pod communication, Pod-to-service communication, and external-to-service communication.

Pod-to-Pod communication is the foundation of Kubernetes. Pods communicate with each other following network policies set by the network plugin, communicating with other Pods without explicitly creating links between them or mapping container ports to host ports. Because Pods share the same network namespace and have their own IP addresses, they can find and communicate with all other Pods on all nodes using localhost, without using network address translation (NAT).

One of the challenges of Kubernetes networking is addressing how internal (east-west) traffic and external (north-south) traffic interact, because the internal network is isolated from the external network. However, traffic that flows between nodes can also flow to and from nodes and an external physical or virtual machine. There are a few different ways of getting external traffic into a Kubernetes cluster:

* **LoadBalancer**: LoadBalancer is the standard way to connect a service externally to the internet. In this scenario, a network load balancer forwards all external traffic to a service. Each service gets its own IP address.
* **ClusterIP**: ClusterIP is the default Kubernetes service for internal communications. However, external traffic can access the default Kubernetes ClusterIP service through a proxy. This can be useful for debugging services or displaying internal dashboards.
* **NodePort**: NodePort opens ports on the nodes or virtual machines, and traffic is forwarded from the ports to the service. It is most often used for services that don’t always have to be available, such as demo applications.
* **Ingress**: Ingress acts as a router or controller to route traffic to services via a load balancer. It is useful if you want to use the same IP address to expose multiple services.

One more important aspect of Kubernetes networking is the [Container Networking](https://www.vmware.com/solutions/container-networking.html) Interface, or CNI. The CNI connects Pods across nodes, acting as an interface between a network namespace and a network plug-in or a network provider and a Kubernetes network. There are many different CNI providers and plug-ins to choose from with different sets of features and functionality. CNI plug-ins have the ability to dynamically configure a network and resources as Pods are provisioned and destroyed. They provision and manage IP addresses as containers are created and deleted. Kubernetes is the default networking provider for Kubernetes, but CNI plug-ins such as Flannel, Calico, Canal, and Weave Net offer additional features.

How to implement Kubernetes Networking?

Network administrators can implement the Kubernetes networking model in a wide variety of different ways. Gartner recognized the following Kubernetes networking examples as “Visionary” in its 2019 Magic Quadrant for [Data Center Networking](https://www.vmware.com/topics/glossary/content/data-center-networking.html) report:

* **Big Switch Networks’ Big Cloud Fabric**: A [cloud-native networking](https://www.vmware.com/cloud-solutions/app-modernization/cloud-native-apps.html) architecture that runs Kubernetes in private cloud environments.
* **Cumulus NetQ**: A telemetry-based fabric validation system that provides visibility into [container deployments](https://www.vmware.com/topics/glossary/content/container-deployment.html).
* **Dell EMC**: A cloud platform that supports both traditional applications and cloud-native environments with integrated support for Kubernetes and containers.
* **VMware NSX-T**: [Network virtualization](https://www.vmware.com/topics/glossary/content/network-virtualization.html) and security platform that virtualizes networks for multi-cloud or multi-hypervisor environments, including native container networking.

What makes up a Kubernetes cluster?

A Kubernetes cluster contains six main components:

1. **API server:** Exposes a REST interface to all Kubernetes resources. Serves as the front end of the Kubernetes control plane.
2. **Scheduler:** Places containers according to resource requirements and metrics. Makes note of Pods with no assigned node, and selects nodes for them to run on.
3. **Controller manager:** Runs controller processes and reconciles the cluster’s actual state with its desired specifications. Manages controllers such as node controllers, endpoints controllers and replication controllers.
4. **Kubelet:** Ensures that containers are running in a Pod by interacting with the Docker engine , the default program for creating and managing containers. Takes a set of provided PodSpecs and ensures that their corresponding containers are fully operational.
5. **Kube-proxy:** Manages network connectivity and maintains network rules across nodes. Implements the Kubernetes Service concept across every node in a given cluster.
6. **Etcd:** Stores all cluster data. Consistent and highly available Kubernetes backing store.

These six components can each run on Linux or as Docker containers. The master node runs the API server, scheduler and controller manager, and the worker nodes run the kubelet and kube-proxy.

How do you work with a Kubernetes cluster?

To work with a Kubernetes cluster, you must first determine its desired state. The desired state of a Kubernetes cluster defines many operational elements, including:

* **Applications** and workloads that should be running
* **Images** that these applications will need to use
* **Resources** that should be provided for these apps
* **Quantity** of needed replicas

To define a desired state, JSON or YAML files (called manifests) are used to specify the application type and the number of replicas needed to run the system.

Developers use the Kubernetes API to define a cluster’s desired state. This developer interaction uses the command line interface (kubectl) or leverages the API to directly interact with the cluster to manually set the desired state. The master node will then communicate the desired state to the worker nodes via the API.

Kubernetes automatically manages clusters to align with their desired state through the Kubernetes control plane. Responsibilities of a Kubernetes control plane include scheduling cluster activity and registering and responding to cluster events.

The Kubernetes control plane runs continuous control loops to ensure that the cluster’s actual state matches its desired state. For example, if you deploy an application to run with five replicas, and one of them crashes, the Kubernetes control plane will register this crash and deploy an additional replica so that the desired state of five replicas is maintained.

Automation occurs via the Pod Lifecycle Event Generator, or PLEG. These automatic tasks can include:

* **Starting** and restarting containers
* **Adjusting** the number of replicas for an application
* **Validating** container images
* **Launching** and managing containers
* **Implementing** updates and rollbacks

How to create a Kubernetes cluster?

You can create and deploy a Kubernetes cluster on either a physical or a virtual machine. It is recommended for new users to start creating a Kubernetes cluster by using Minikube. Minikube is an open-source tool that is compatible with Linux, Mac and Windows operating systems. Minikube can be used to create and deploy a simple, streamlined cluster that contains only one worker node.

In addition, you can use Kubernetes patterns to automate the management of your cluster’s scale. Kubernetes patterns facilitate the reuse of cloud-based architectures for container-based applications. While Kubernetes does provide a number of useful APIs, it does not supply guidelines for how to successfully incorporate these tools into an operating system. Kubernetes patterns provide a consistent means of accessing and reusing existing [Kubernetes architectures](https://www.vmware.com/topics/glossary/content/kubernetes-architecture.html). Instead of creating these structures yourself, you can tap into a reusable network of Kubernetes cluster blueprints.