Chapter 6

An Overview of Kubernetes Architecture

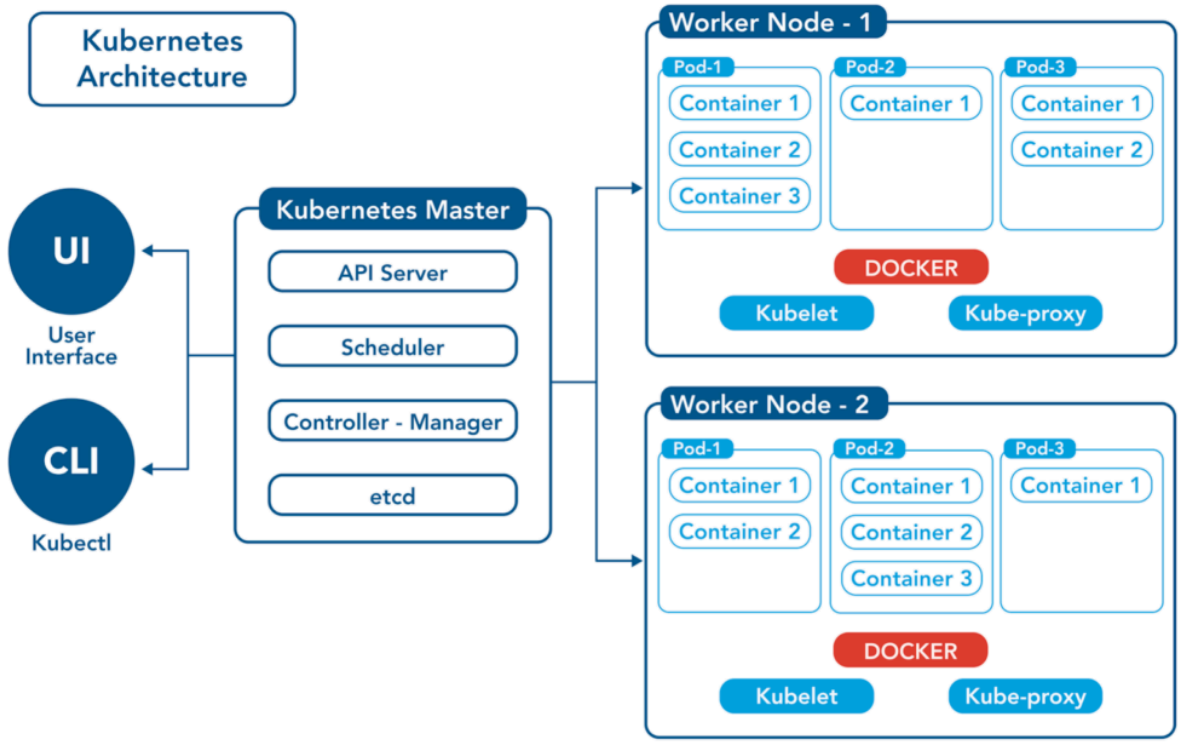
Kubernetes (K8s) is an orchestration platform for containerized applications. It enables enterprises to automate the management and configuration of container workloads at scale. As containers have become the cornerstone of modern application delivery, K8s has become the most popular platform to orchestrate ther

To help you gain a firm understanding of
will explain K8s architecture and functio
most out of this content, you should be fa
containerization, microservices, immutable architecture, and infrastructure-as-code. If you're not comfortable with those concep

yet, we recommend starting with the first articles of our executive guide to Kubernetes.

Kubernetes Architecture

The diagram below illustrates the many components that make up the K8s architecture. A description of each of the core components follows.



The Kubernetes Architecture. Source

Containers: A container is a self-enclosed software instance with all the required software packages and libraries to run an isolated "micro-application." If you are new to containers, read our article dedicated to explaining the purpose of a container.

Pods: A pod groups one or more containers and is a core building block of the K8s architecture. For example, K8s can automatically replace a pod when it goes down, add more CPU and memory to it when needed, or even replicate it to scale out. Pods are assigned IP addresses. A set of pods together form a scalable "workload" that a "controller" manages. These workloads connect to "services" that represent the entire set of pods. A service load balances web traffic across the pods even as some pods are scaled or destroyed. It's worth noting that

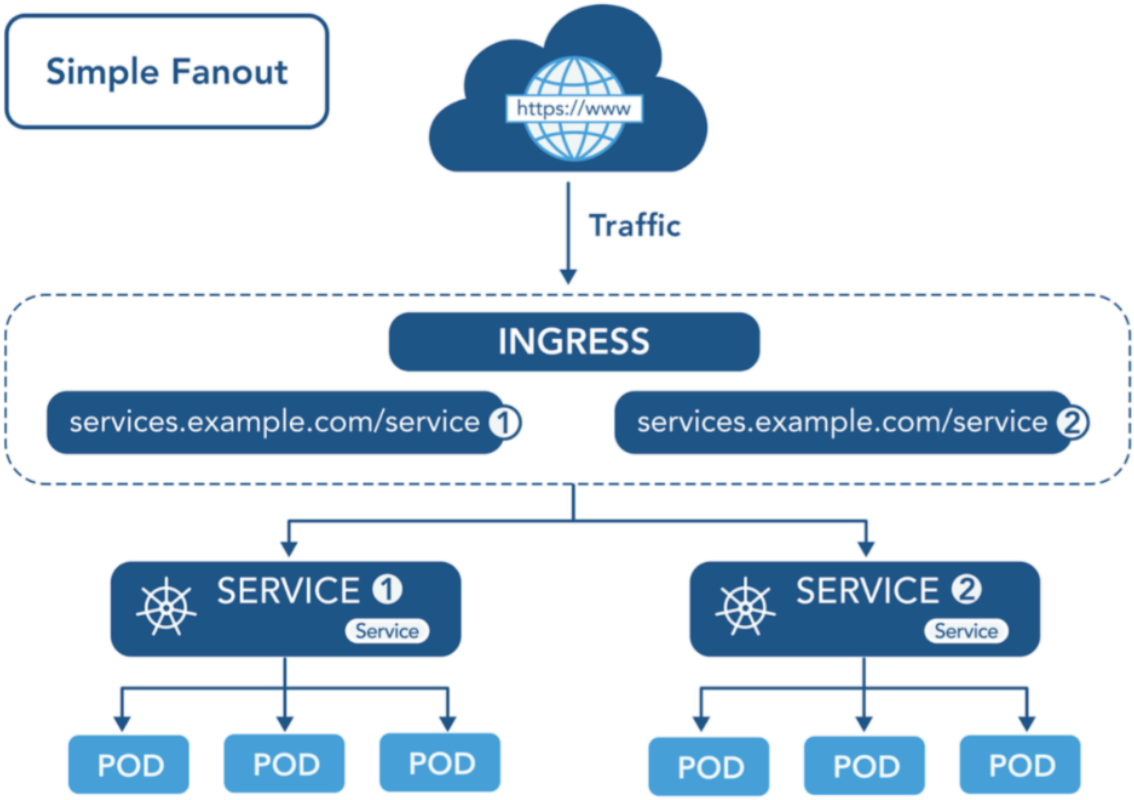
storage volumes are also attached to pods and exposed to containers within pods.

Controller: A controller is a control loop that assesses the difference between the desired state and the actual state of the K8s cluster and communicates with the API server to create, update, and delete the resources that it manages. The label selectors that are part of the controller define the set of pods controlled by the controller. K8s has multiple types of controllers: Replication Controller (replicates pods for scaling), DeamonSet Controller (ensures each node gets one copy of a designated pod), Job Controller (runs software at a specific time), CronJob Controller (schedules jobs that run periodically), StatefulSet Controller (controls stateful applications), and Deployment Controller (controls stateless applications).

Node: A node is a physical or virtual server on which pods can be scheduled. Every node has a container runtime, a kubelet pod, and a kube-proxy pod (more on those three items to come). Node groups (also known as autoscaling groups or node pool groups) can be scaled manually and automatically.

Volumes: K8s storage volumes provide persistent data storage available throughout the lifetime of the pod. Containers within a pod can share a storage volume. A node can also directly access a storage volume.

Services: A Kubernetes service is a set of pods that work together. Here's an example of a K8s service.



Kubernetes load balances traffic across the services. Source.

A “label selector” (in a service configuration file) defines a set of pods that compose a single service. The service feature assigns an IP address and DNS name to the service and load balances traffic in a round-robin fashion to the addresses that match the label selector. This approach effectively allows “decoupling” (or abstracting) of the frontend from the backend.

Kube-proxy: The kube-proxy component runs on each node and maintains network services on worker nodes. It also maintains network rules, allows network communication between services and pods, and is responsible for routing network traffic.

Kubelet: A kublet runs on each node and communicates information about the state and health of containers to Kubernetes.

Container Runtime: The container runtime is the software that runs the containers. Popular container runtime examples include containerd, Docker, and CRI-O.

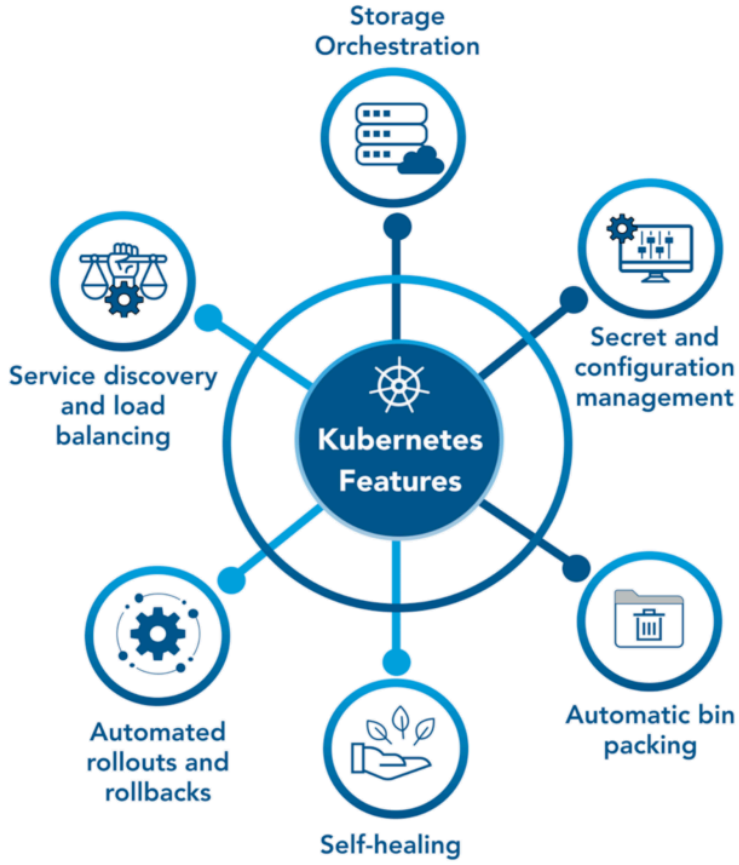
Controller Plane: The Kubernetes master is the main controlling unit of a cluster. It manages the entire K8s cluster, including workloads, and provides a communications interface for the cluster. The Kubernetes controller plane consists of multiple components. These components

allow Kubernetes to run highly available applications. The main components of the Kubernetes control plane are:

- **Etcd**: etcd stores the overall configuration data of the cluster (i.e., state and details of a pod), thereby representing the state of the cluster in Kubernetes master nodes. The API Server uses etcd data to monitor the cluster and enact changes to the cluster to resemble the desired state set.
- **API Server**: transmits data through HTTP using JSON. It provides both the internal and external interface to Kubernetes. It processes requests, validates them, and instructs the relevant service or controller to update the state object in etcd, and allows users to configure workloads across the cluster.
- **Scheduler**: assesses nodes to select where an unscheduled pod should be placed based on CPU and Memory requests, policies, labels affinities, data locality of the workload.
- **Controller Manager**: The controller manager is a single process that encompasses all of the controllers within Kubernetes. While logically, the controllers are separate processes, they are run as a single process in a DaemonSet to reduce complexity.

Kubernetes Functionality

Simply referring to Kubernetes as an orchestration tool may sound limiting when considering the six areas of functionality it provides. Its functionality ranges from load balancing the incoming web traffic to enabling a rollback when a new software release deployment goes wrong. Below we explain each of the six areas of functionality in more detail.



Kubernetes features grouped in six categories. Source.

Service Discovery and Load Balancing: Kubernetes assigns each set of pods a single DNS name and an IP address. If traffic is high, K8s automatically balances the load across a service which may include multiple pods.

Automated Rollouts and Rollbacks: K8s can roll out new pods and swap them with existing pods. It can also change configurations without impacting end-users. Additionally, Kubernetes has an automated rollback feature. This rollback functionality can revert changes if not completed successfully.

Secret and Configuration Management: Configuration and secrets information can be securely stored and updated without rebuilding images. Stack configuration does not require the unveiling of secrets, which limits the risk of data compromise.

Storage Orchestration: Different storage systems such as local storage, network storage, and public cloud providers can be mounted using K8s.

Automatic Bin Packing: Kubernetes algorithms seek to allocate containers based upon configured CPU and RAM requirements efficiently. This feature helps enterprises optimize resource utilization.

Self-Healing: Kubernetes performs health checks on pods and restarts containers that fail. If a pod fails, K8s does not allow connections to them until the health checks have passed.

Conclusion

Kubernetes is a complex system. However, it has proven to be the most robust, scalable, and performant platform to orchestrate highly available container-based applications, support decoupled and diverse stateless and stateful workloads, and provide automated rollouts and rollbacks. If this intricate architecture has left you wondering about the origins and the evolution of Kubernetes, you can read our article on this subject.