A Crash Course in Kubernetes

BYTEBYTEGO
OCT 26, 2023 · PAID

In today's world of complex, web-scale application backends made up of many microservices and components running across clusters of servers and containers, managing and coordinating all these pieces is incredibly challenging.

That's where Kubernetes comes in. Kubernetes (also known as "k8s") is an open-source container orchestration platform that automates deployment, scaling, and management of containerized applications.

With Kubernetes, you don't have to worry about manually placing containers or restarting failed ones. You simply describe your desired application architecture and Kubernetes makes it happen and keeps it running.

In this two-part series, we'll dive deep into Kubernetes and cover:

- Key concepts like pods, controllers, and services
- The components that make up a Kubernetes cluster
- When and why Kubernetes is useful for your applications
- Tradeoffs to consider before adopting Kubernetes

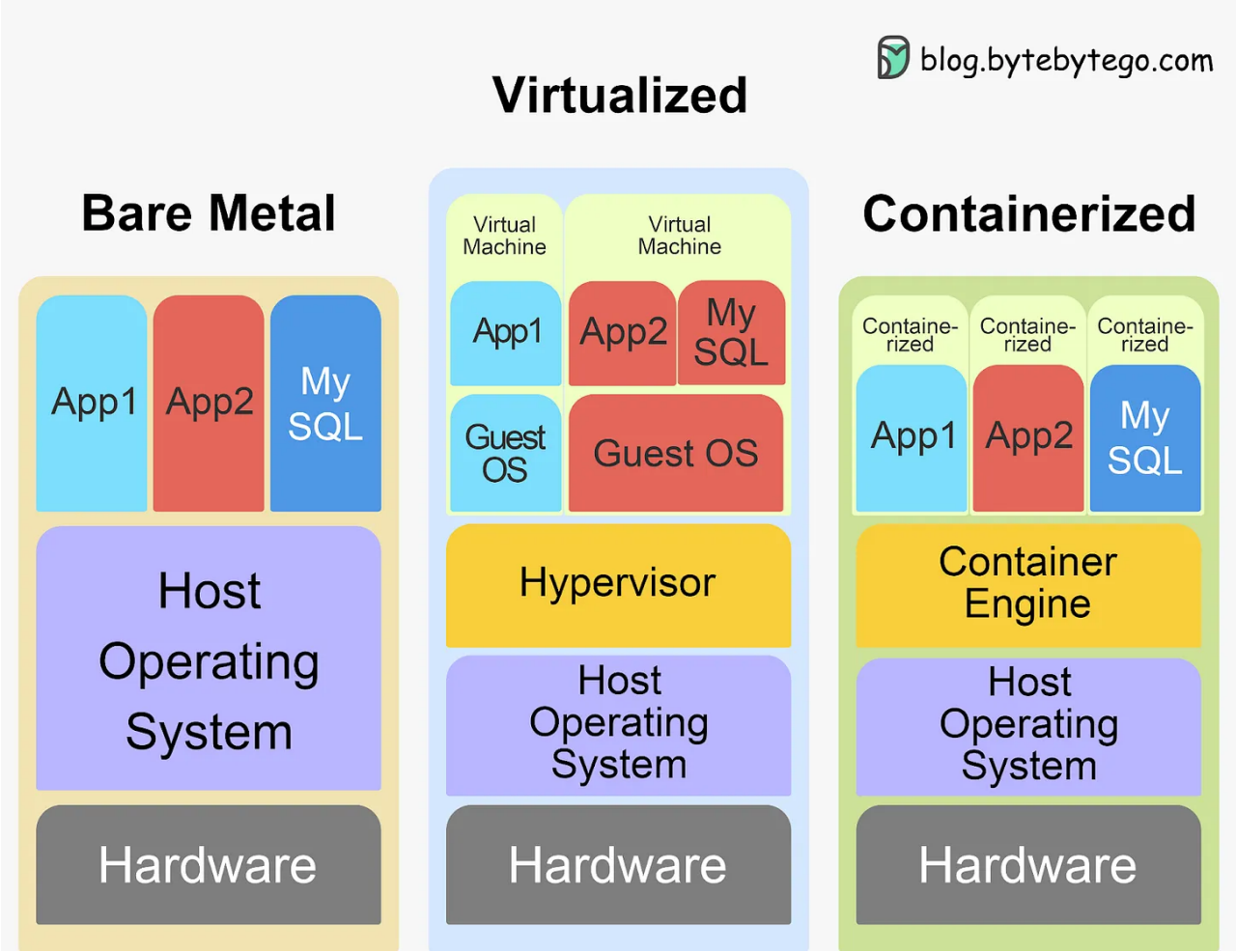
We'll demystify Kubernetes and equip you with everything you need to determine if and when Kubernetes could be the right solution for your applications. You'll walk away with a clear understanding of what Kubernetes is, how it works, and how to put it into practice.

Whether you're a developer, ops engineer, or technology leader, you'll find invaluable insights in this deep dive into Kubernetes. Let's get started!

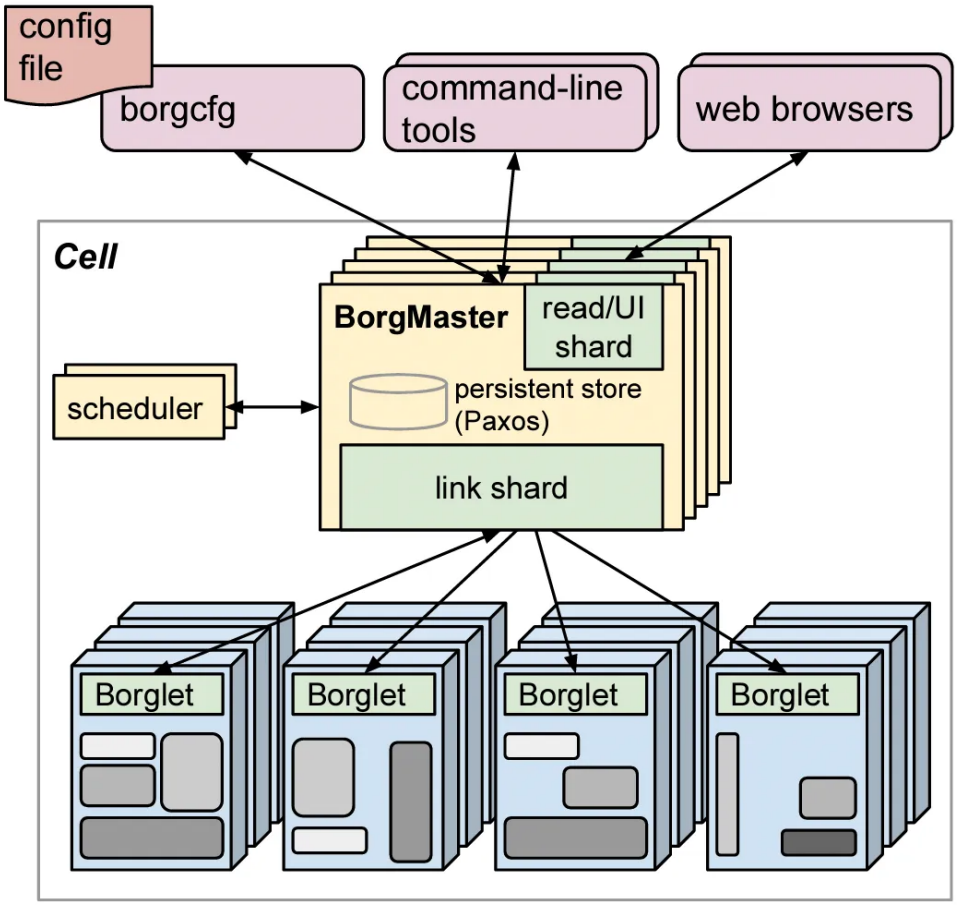
Brief History

Kubernetes can be traced back to Google's internal container orchestration system, Borg, which managed the deployment of thousands of applications within Google. Containers

are a method of packaging and isolating applications into standardized units that can be easily moved between environments. Unlike traditional virtual machines (VMs) which virtualize an entire operating system, containers only virtualize the application layer, making them more lightweight, portable and efficient.



In 2014, Google open-sourced a container orchestration system based on its learnings from Borg. This is Kubernetes. Kubernetes provides automated deployment, scaling and management of containerized applications. By leveraging containers rather than VMs, Kubernetes provides benefits like increased resource efficiency, faster deployment of applications, and portability across on-prem and cloud environments.

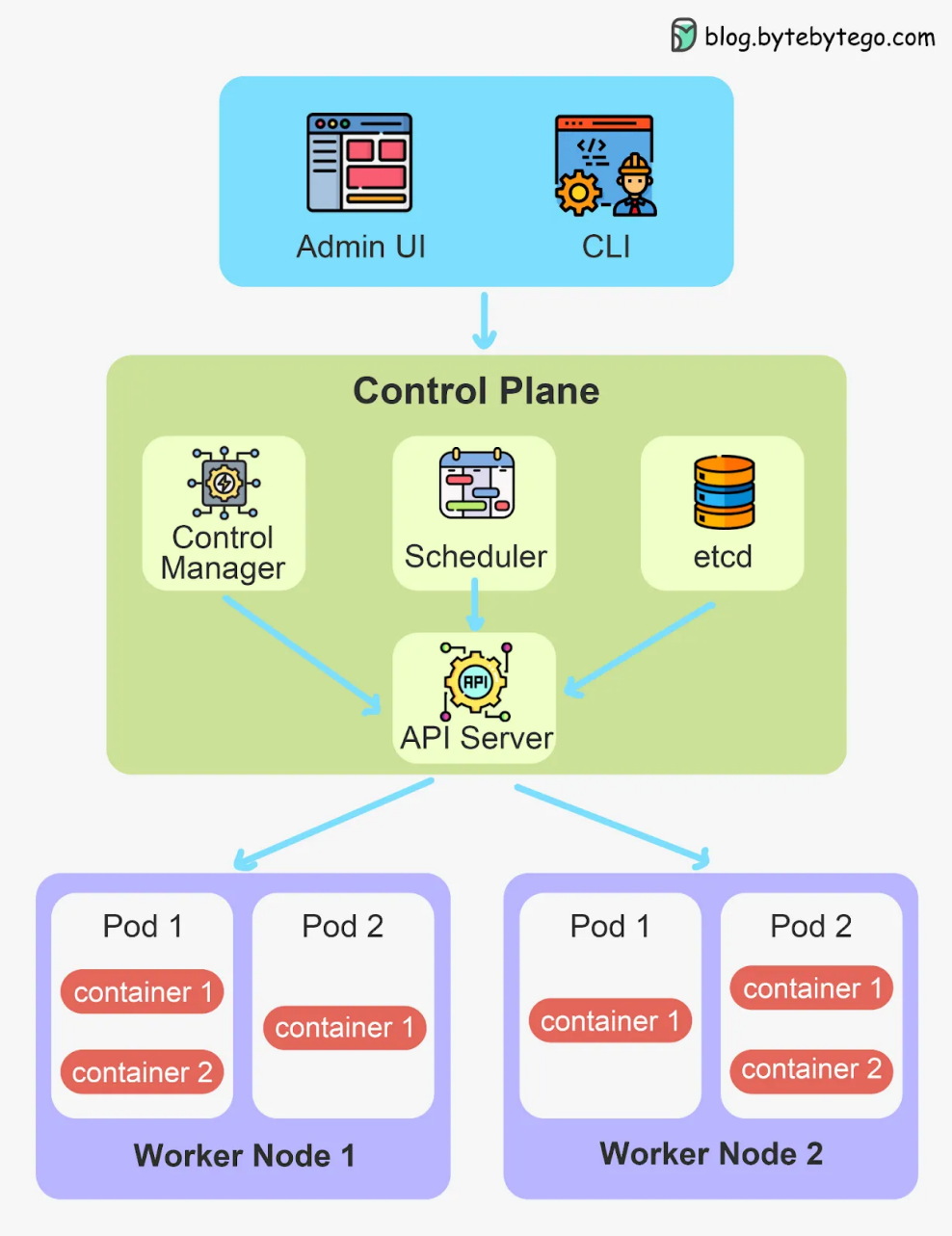


Source: Large-scale cluster management at Google with Borg

Why is it also called k8s? This is a somewhat nerdy way of abbreviating long words. The number 8 in k8s refers to the 8 letters between the first letter “k” and the last letter “s” in the word Kubernetes.

Kubernetes Architecture and Key Components

At its core, Kubernetes follows a client-server architecture. There are two core pieces in a Kubernetes cluster - control plane and worker nodes.



The control plane is responsible for managing the state of the cluster. In production environments, the control plane usually runs on multiple nodes that span across several data center zones.

In other words, the control plane manages worker nodes and the containers running on them.

The containerized applications run in a Pod. Pods are the smallest deployable units in Kubernetes. A pod hosts one or more containers and provides shared storage and networking for those containers. Pods are created and managed by the Kubernetes control plane. They are the basic building blocks of Kubernetes applications.

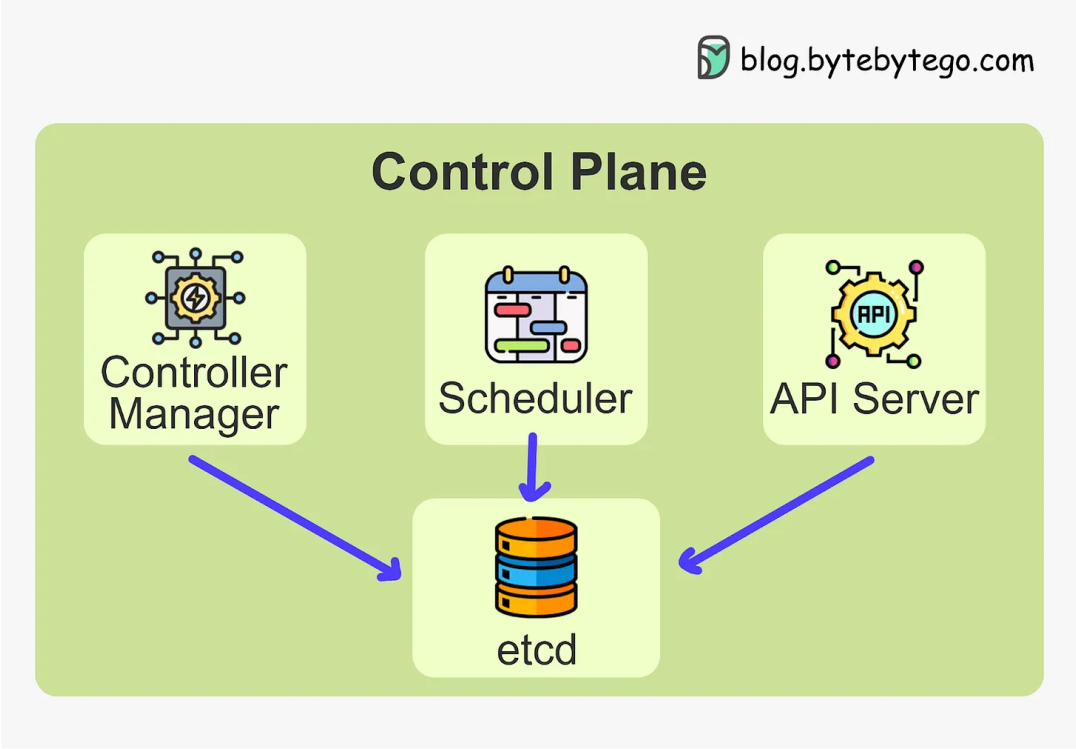
Let’s dive deeper into the main pieces.

Kubernetes Control Plane

The control plane is the brain of Kubernetes. It consists of various components that, together, make global decisions about the cluster. The control plane components run on multiple servers across availability zones to provide high availability.

The key components are:

- Kubernetes API Server
- Etcd
- Kubernetes Scheduler
- Kubernetes Controller Manager



Kubernetes API Server

The API server is the primary interface between the control plane and the rest of the cluster. It exposes a RESTful API that allows clients to interact with the control plane and submit requests to manage the cluster. It provides the interaction layer for other components to communicate with the cluster.

Etcd

Etcd is a lightweight distributed key-value store that persistently stores the cluster configuration. It ensures components have a consistent view of the cluster state.

It is used by the API server and other components of the control plane to store and retrieve information about the cluster.

Kubernetes Scheduler

The scheduler is responsible for scheduling pods onto the worker nodes in the cluster. It uses information about the resources required by the pods and the available resources on the worker nodes to make placement decisions. It selects the optimal worker node to run each pod based on resource requirements and other policies.

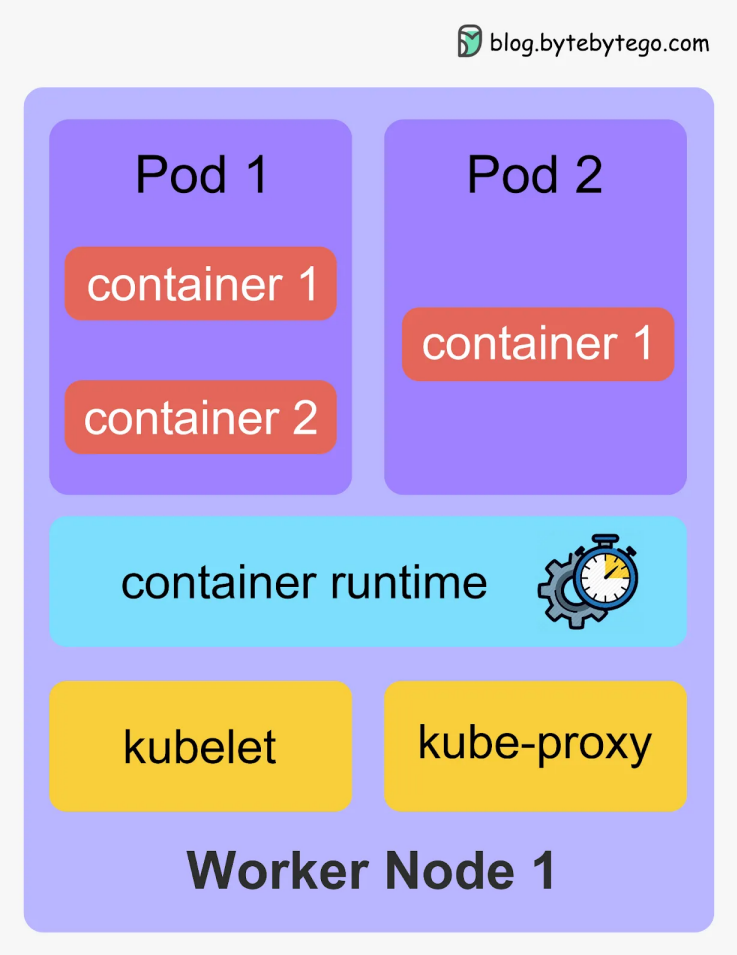
Kubernetes Controller Manager

The controller manager is responsible for running core Kubernetes controllers that manage the state of the cluster.

Some examples include the replication controller, which ensures that the desired number of replicas of a pod are running, and the deployment controller, which manages the rolling update and rollback of deployments. More on these controllers later.

Kubernetes Worker Nodes

The worker nodes are the ones that actually run containerized applications. The core components of the worker nodes include kubelet, Kubernetes Container Runtime, and kube-proxy.



kubelet

The kubelet is a daemon that runs on each worker node. It is the primary node agent that communicates with the control plane. It receives instructions from the control plane about which pods to run on the node, and ensures that the desired state of the pods is maintained. It manages containers on the node via a container runtime.

Container Runtime

The container runtime runs the containers on the worker nodes. It is responsible for pulling the container images from a registry, starting and stopping the containers, and managing the containers' resources. Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

kube-proxy

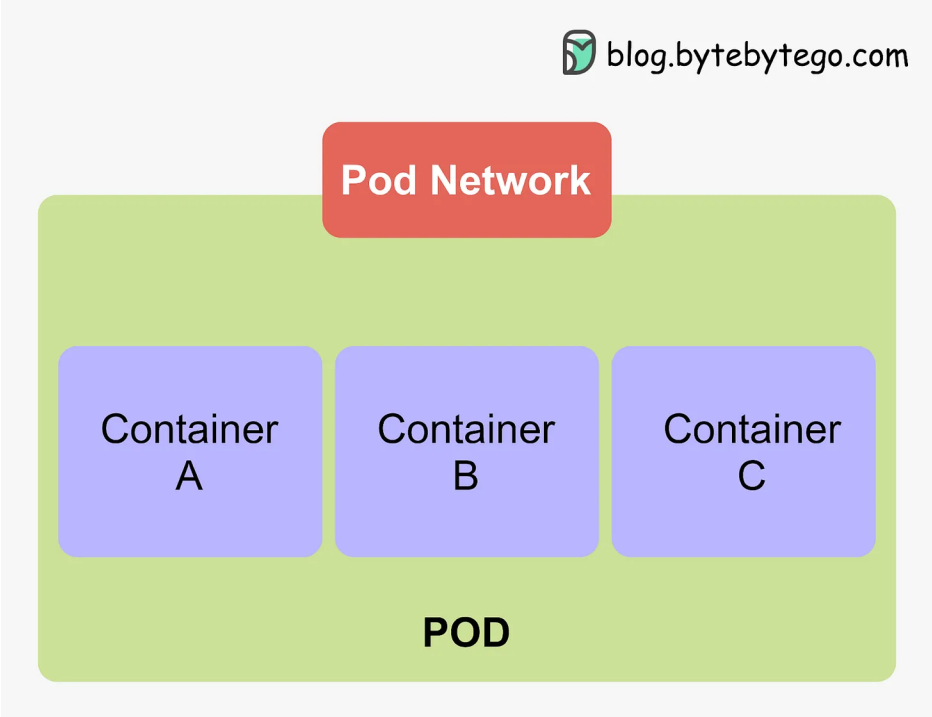
The kube-proxy is a network proxy that runs on each worker node. It is responsible for routing traffic to the correct pods. It also provides load balancing for the pods and ensures that traffic is distributed evenly across the pods.

Pods - The Atomic Unit of Kubernetes

Kubernetes deploys and manages applications in units called pods. Pods are the smallest deployable units that encapsulate one or more tightly coupled containers that share resources.

Pods offer three main benefits:

- Shared networking - The containers within a pod share the same IP address and port space. It allows seamless communication between containers.
- Shared storage - The containers within a pod can access shared volumes for storage. This allows containers to share data easily.
- Co-scheduling and co-location - Kubernetes always schedules and runs pods together on the same node. This allows the containers within pods to be co-located.



Pods represent a higher level abstraction over containers. It makes Kubernetes easier to use. Application modules can be coupled and co-located within a pod. This improves performance through locality. Pods can include helper containers that augment the main app container, such as sidecar containers for logging, metrics, proxies, data loading, and service meshes. These helpers don't have to run in the app container.

Pods allow for modular application composition with portable, encapsulated environments. They are the atomic unit that the Kubernetes control plane manages and manipulates.

Next, let's examine several higher level abstractions that make managing pods easier. They are:

- Kubernetes ReplicaSets and Deployments
- Kubernetes Controllers
- Kubernetes Services

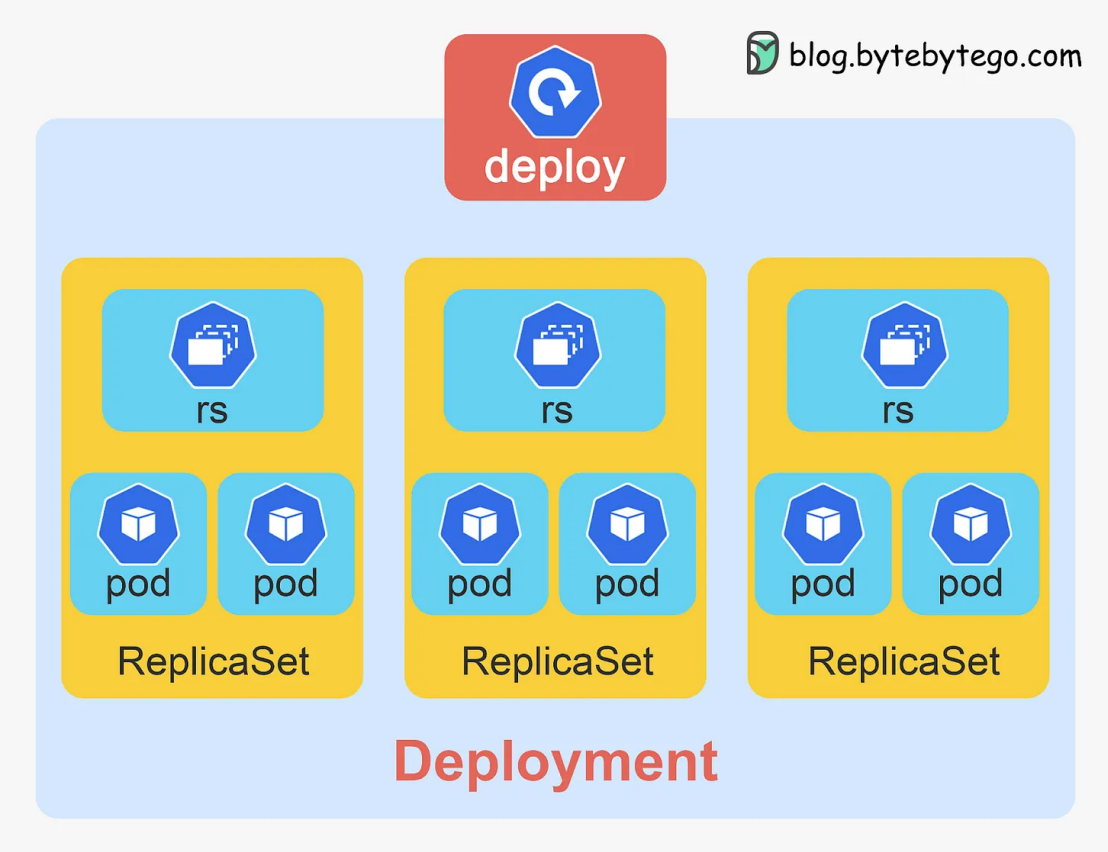
ReplicaSets and Deployments

Kubernetes applications often require multiple instances of pods to meet traffic demands and provide high availability.

ReplicaSets enable easily running identical pods at scale. They ensure the specified number of pod replicas are running, even as nodes fail or pods crash. This prevents disruption when pods fail and allows scaling to meet traffic loads.

Deployments build on ReplicaSets to provide rolling updates for pods. Deployments allow declaring application version changes in a version-controlled way. The deployment controller (explained below) smoothly transitions pod replicas between versions via controlled rollout and rollback.

Deployments enable continuous delivery workflows for Kubernetes applications. They allow teams to ship changes to production frequently and reliably via declarative rollouts. Deployments make it easy to scale pods while providing automated health checks and graceful rollbacks if issues arise.



ReplicaSets and Deployments are critical Kubernetes abstractions for running pods at scale and shipping changes with confidence.

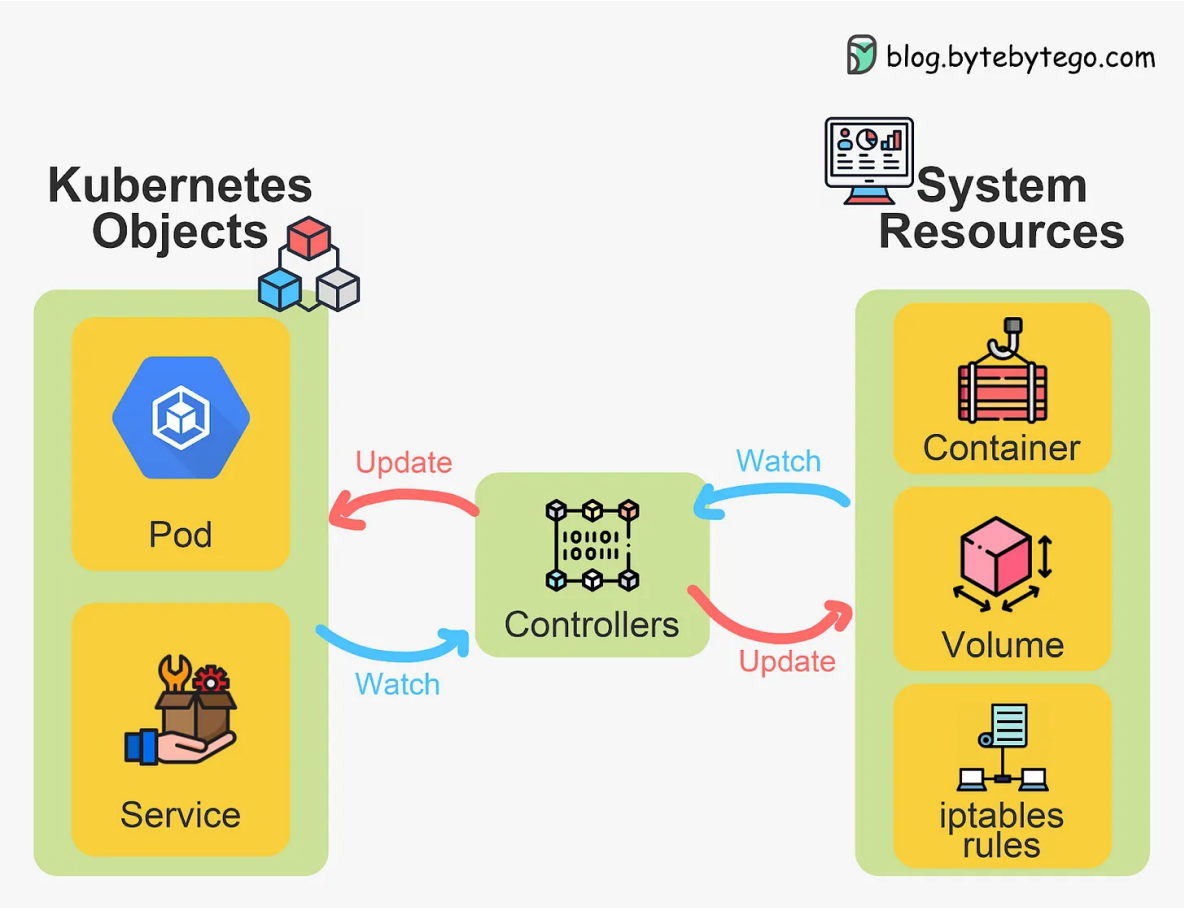
Kubernetes Controllers - The Control Logic

Kubernetes Controllers are control loops that watch the state of the cluster, make changes to move it towards the desired state, and continuously reconcile current state with desired state.

Some key Kubernetes controllers include:

- ReplicaSet Controller - Ensures the specified number of pod replicas are running at all times.
- Deployment Controller - Provides declarative updates to pods and ReplicaSets via rollouts and rollbacks.
- StatefulSet Controller - Manages stateful pods with persistent storage and unique network IDs.
- DaemonSet Controller - Runs a copy of a pod on selected or all nodes in a cluster.
- Job Controller - Manages pods that run to completion to finish batch jobs.
- CronJob Controller - Runs jobs on a periodic schedule.

Controllers simplify Kubernetes cluster operations. Rather than manually manage pods, ReplicaSets, rollouts etc., you simply declare the desired state via Kubernetes API objects. The controllers then continuously work to achieve that state.



For example, a deployment allows declaring a desired number of pod replicas. The deployment controller will automatically:

- Create new pods if replicas are less than desired
- Delete extra pods if replicas are more than desired
- Update pods to new versions during rollouts

Without controllers, users would have to manually handle all these tasks. Controllers turn desired state into reality.

Services and Networking

Kubernetes Services provide stable networking and load balancing for pods. Services enable loose coupling between pods by providing a single IP address and DNS name to access a changing group of pods.

This is useful because pods are ephemeral resources that are frequently created and destroyed. The pods running an application at one moment may be different from the pods running that application later.

Each pod gets its own IP address, so for a deployment with dynamic pods, the set of IP addresses is unpredictable. This presents a discovery problem - if the clients need to connect to our backend applications, how do they find and track the backend IP addresses as pods come and go?

Services solve this problem. A Service provides a stable endpoint to access a changing set of pods. The clients only need to know the service name, not the frequently changing pod IPs.

Services load balance requests across pod replicas seamlessly. This allows scaling backend pods without affecting the clients. Services can also route traffic progressively to new pods when rolling out updates.

Under the hood, Services leverage kube-proxy to implement virtual IPs and packet forwarding rules to handle load balancing.

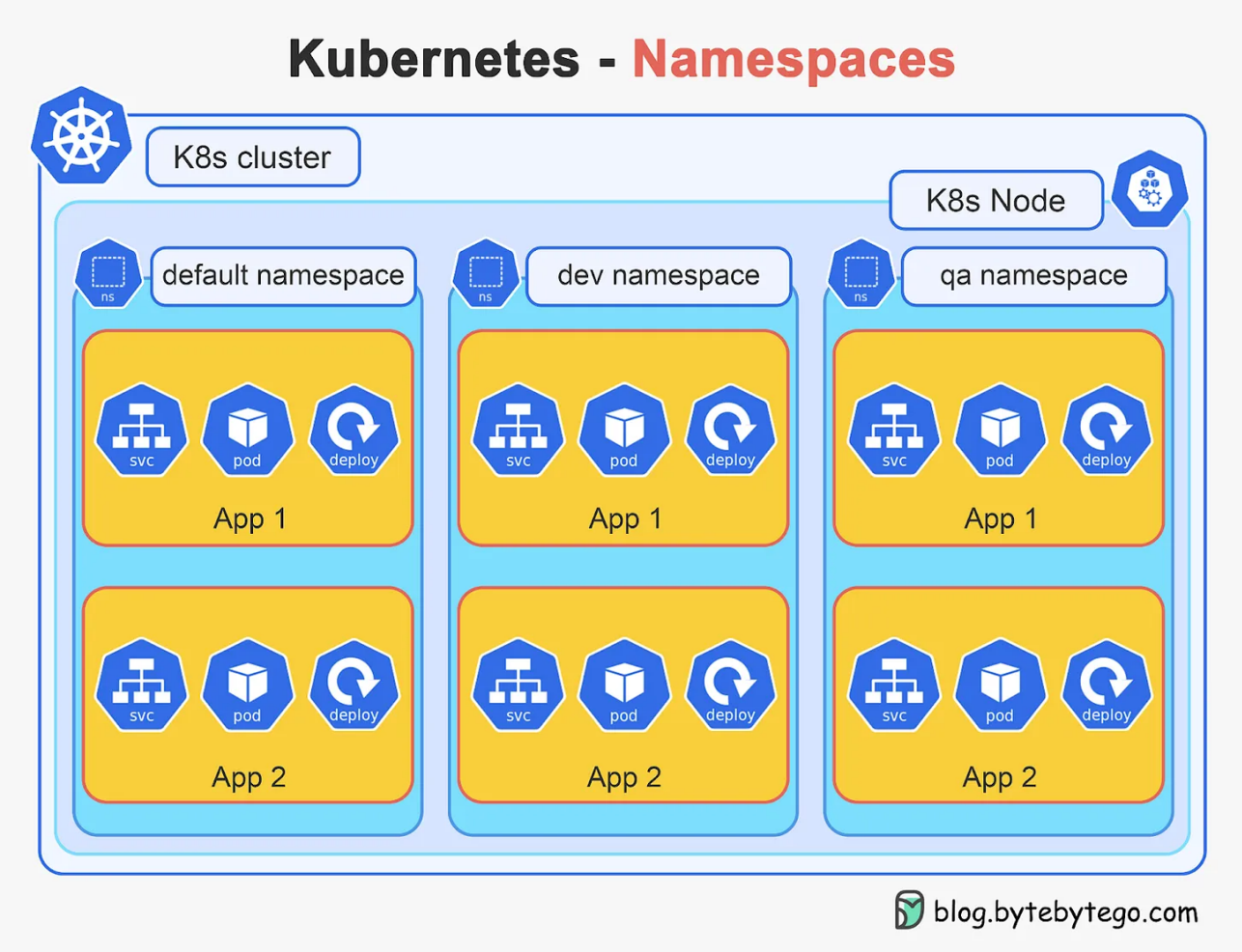
Several types of Services provide different means of external access:

- ClusterIP - Internal IP only accessible within the cluster.
- NodePort - Exposes the Service on a static port on each node.
- LoadBalancer - Creates a cloud load balancer to access the service externally.
- ExternalName - Exposes the service via an arbitrary DNS name.

Kubernetes Services offer discovery, routing, and networking for Pods to simplify application deployment.

Namespaces and Resource Quotas

Kubernetes supports multiple virtual clusters backed by the same physical cluster via Namespaces. Namespaces partition cluster resources between multiple users and applications.



Some key properties of namespaces:

- Provide isolation for teams/apps - resources in one namespace are hidden from other namespaces
- Allow delegated management - access and quotas can be assigned per namespace
- Prevent namespace collisions - two resources can have the same name in different namespaces

Resource quotas can be applied per namespace to limit resource consumption and prevent noisy neighbors. Quotas constrain the total resources that a namespace can consume.

Together, Namespaces and quotas provide safe, shared mult-itenancy on a Kubernetes cluster. Multiple teams and workloads can securely co-exist.

Conclusion

In this issue, we covered the fundamentals of Kubernetes including its history, architecture, main components like pods and controllers, and core features like services

for networking. We discussed how Kubernetes provides benefits like portability across environments and improved infrastructure efficiency.

Stay tuned for our next issue, where we'll explore more advanced Kubernetes capabilities, evaluate the tradeoffs of adoption, and provide recommendations on when Kubernetes is the right fit.

73 Likes

1 Comment

Write a comment...

1 more comment...
