Kubernetes

in 30 mins

The open-source container orchestration plataform

Agenda

01 History

02 Let's get to know k8s

03 Demo

04 End?

Hello! I am Mauro Bernal

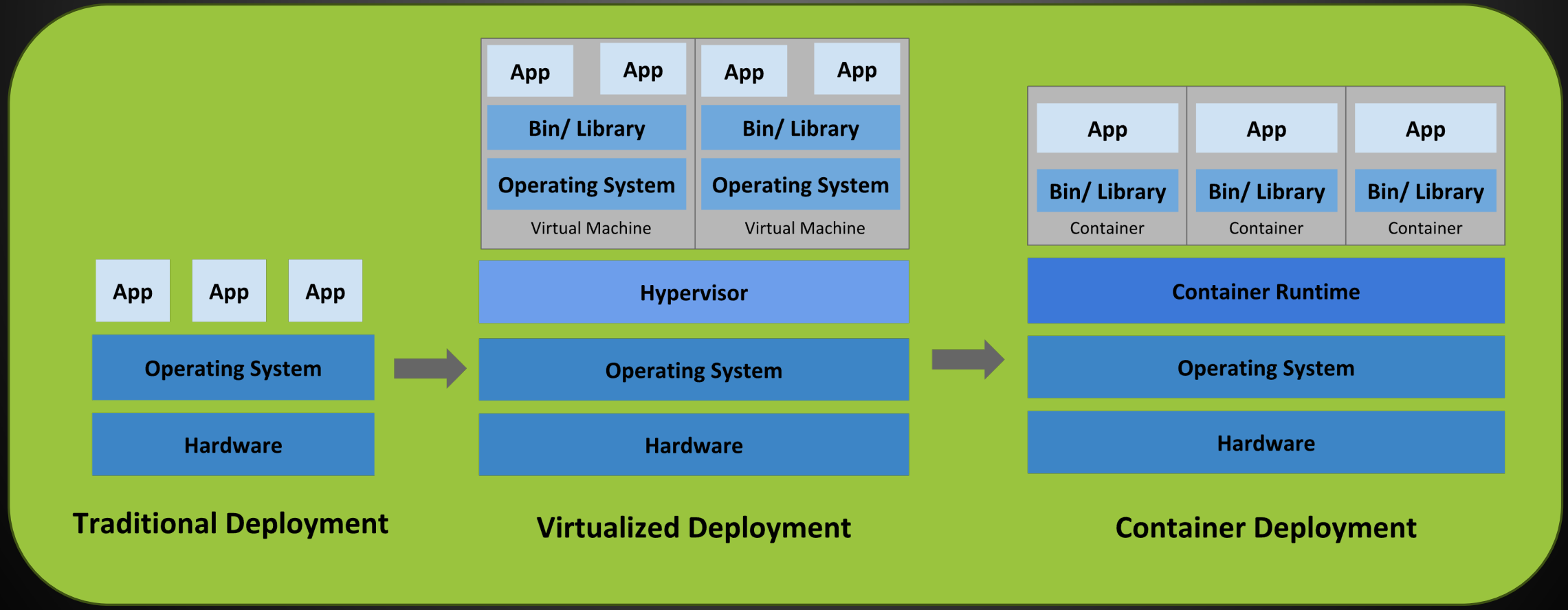
.NET Software developer

SysAdmin

SQL Server DBA

DevOps

1 - History


App
App
App
Operating System
Hardware
Traditional Deployment
App
App
App
App
Bin/ Library
Bin/ Library
Operating System
Operating System
Virtual Machine
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment
App
App
App
Bin/ Library
Bin/ Library
Bin/ Library
Container
Container
Container
Container Runtime
Operating System
Hardware
Container Deployment

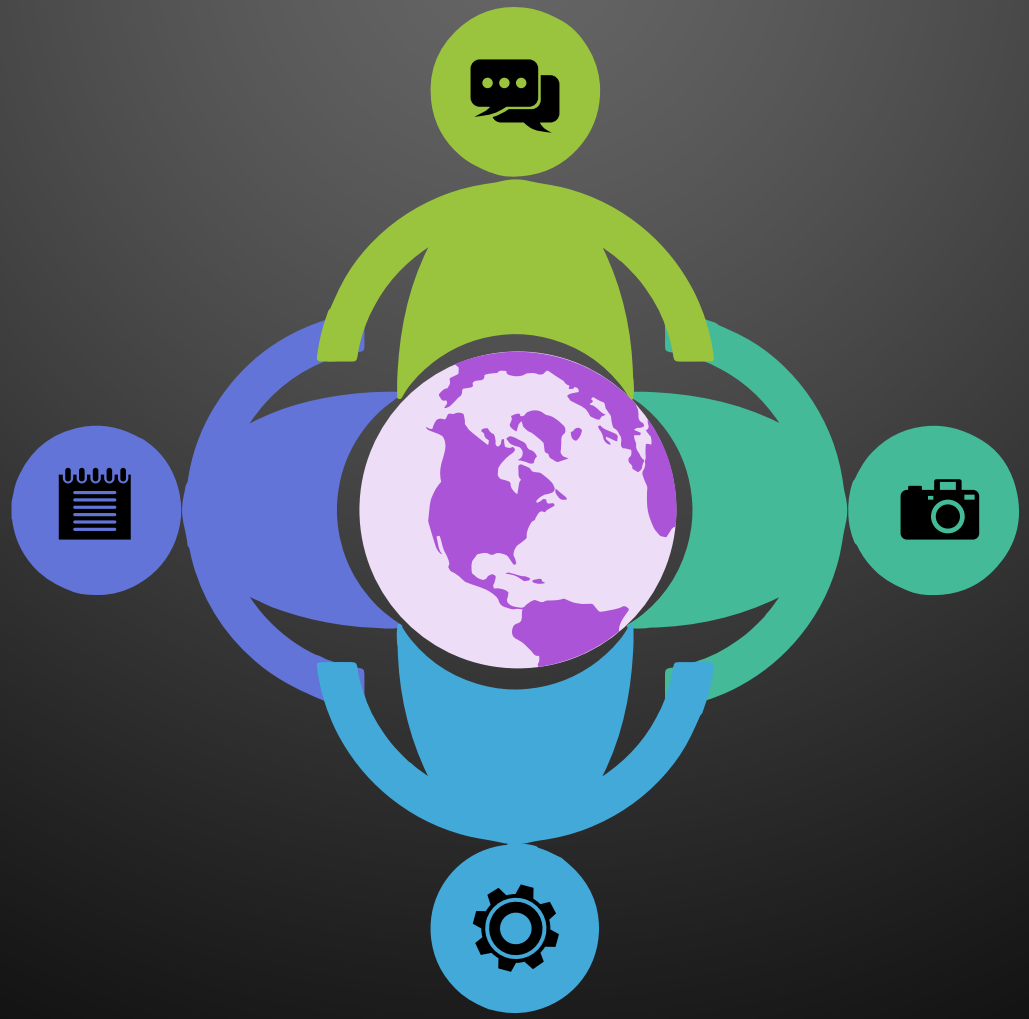
Traditional IT infrastructure

Bare Metal Servers

Physical servers for unique user. Its is rent (o buy) from provider that is not shared with any other tenants.

Apps in servers

organizations ran their apps solely on physical servers



Not control for resources

However, there was no way to maintain system resource boundaries for those apps. For instance, whenever a physical server ran multiple applications, one application might eat up all of the processing power, memory, storage space or other resources on that server.

High costs

To prevent this from happening, businesses would run each application on a different physical server. But running apps on multiple servers creates underutilized resources and problems with an inability to scale. What's more, having a large number of physical machines takes up space and is a costly endeavor.

Virtualization

From 1960

the process that forms the foundation for cloud computing. While virtualization technology can be traced back to the late 1960s, it wasn't widely adopted until the early 2000s.

VMs

While VMs create more efficient usage of hardware resources to run apps than physical servers, they still take up a large amount of system resources. This is especially the case when numerous VMs are run on the same physical server, each with its own guest operating system.

Hypervisor

Relies on software known as a hypervisor. A hypervisor is a lightweight form of software that enables multiple virtual machines (VMs) to run on a single physical server's central processing unit (CPU). Each virtual machine has a guest operating system (OS), a virtual copy of the hardware that the OS requires to run and an application and its associated libraries and dependencies.

Containers

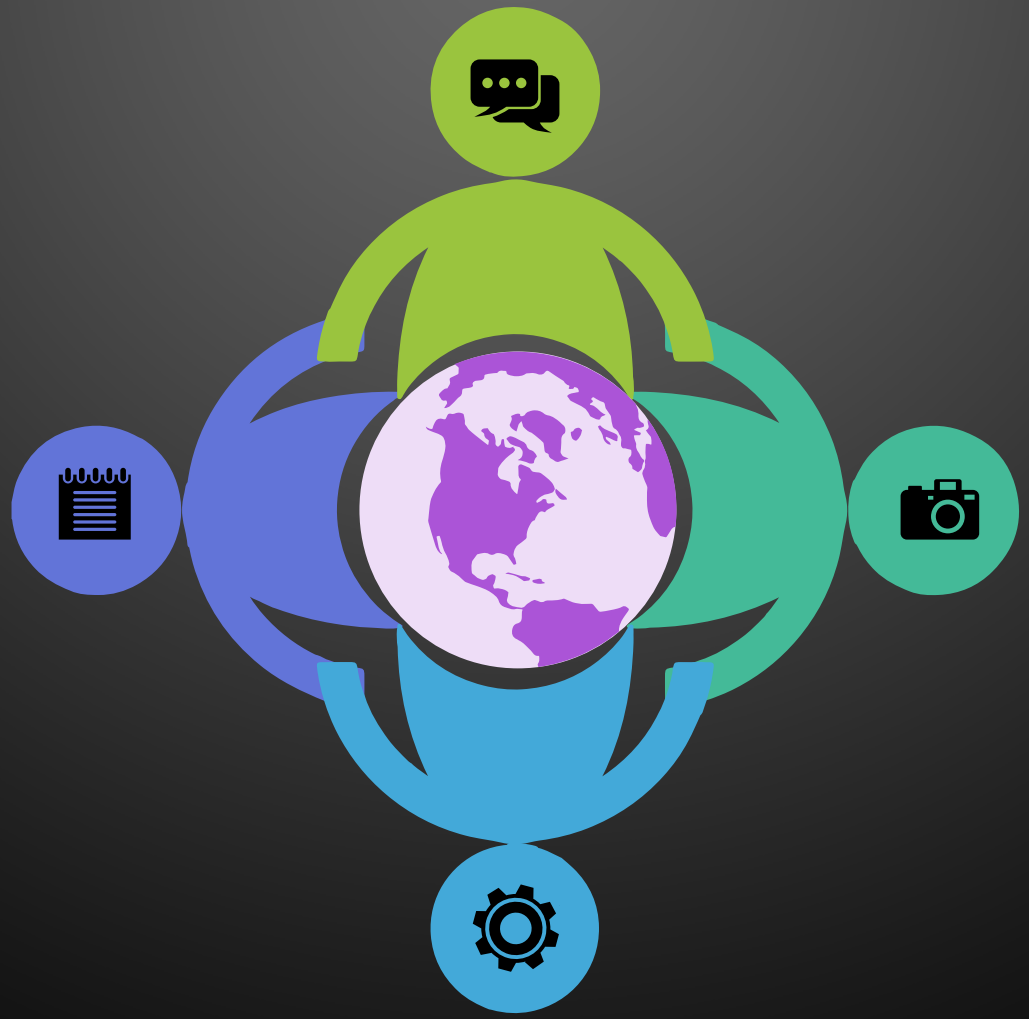
chroot() was added to the Version 7 Unix in 1979 and used for filesystem isolation.

CHROOT

1979 with the development of chroot (link resides outside ibm.com), part of the Unix version 7 operating system. Chroot introduced the concept of process isolation by restricting an application's file access to a specific directory (the root) and its children (or subprocesses).

Containers

Containers are executable units of software that package application code along with its libraries and dependencies. They allow code to run in any computing environment, whether it be desktop, traditional IT or cloud infrastructure.



Moderns Containers

Are defined as units of software where application code is packaged with all its libraries and dependencies. This allows applications to run quickly in any environment—whether on- or off-premises—from a desktop, private data center or public cloud.

Lightweight

Rather than virtualizing the underlying hardware like VMs, containers virtualize the operating system (usually as Linux or Windows). The lack of the guest OS is what makes containers lightweight, as well as faster and more portable than VMs.

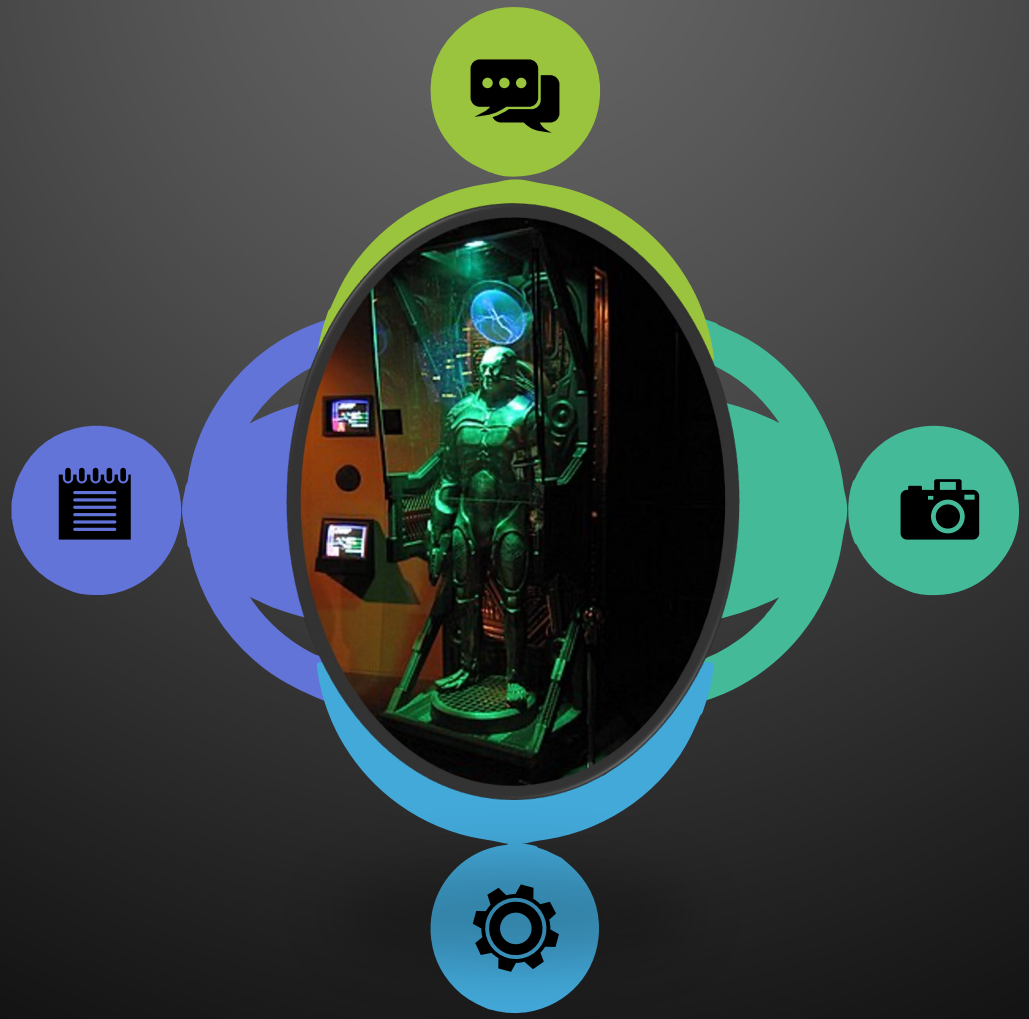
Borg

2003

Google needed a way to get the best performance out of its virtual server to support its growing infrastructure and deliver its public cloud platform. This led to the creation of Borg, the first unified container management system.

Borg's large-scale

Designed to run alongside Google's search engine, Borg was used to build Google's internet services, including Gmail, Google Docs, Google Search, Google Maps and YouTube.



Large scale workloads

allowed run hundreds of thousands of jobs, from many different applications, across many machines. This enabled Google to accomplish high resource utilization, fault tolerance and scalability for its large-scale workloads. Borg is still used at Google today as the company's primary internal container management system.

Omega

In 2013, Google introduced Omega, its second-generation container management system. Omega took the Borg ecosystem further, providing a flexible, scalable scheduling solution for large-scale computer clusters.

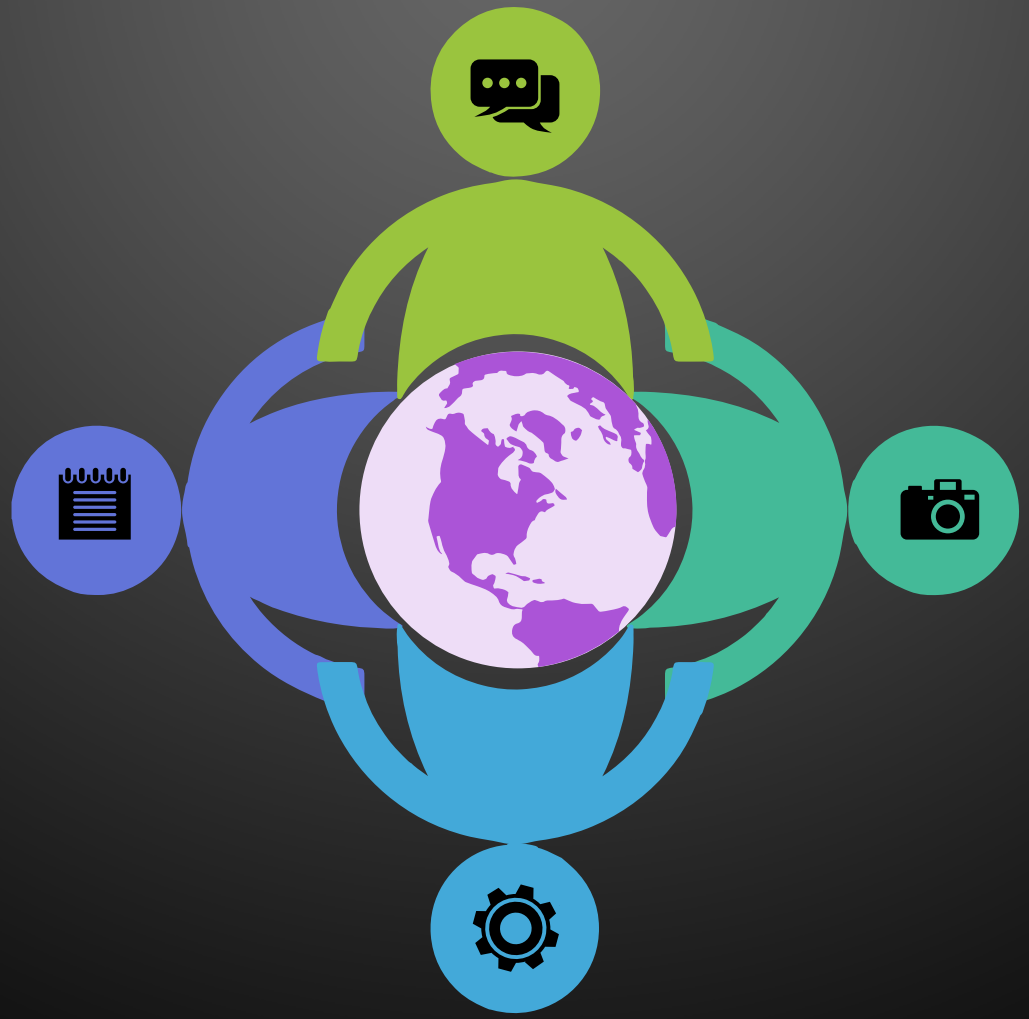
Docker

2013

Developed by dotCloud, a Platform-as-a-Service (PaaS) technology company, was released in 2013 as an open-source software tool that allowed online software developers to build, deploy and manage containerized applications.

Inspiration

By popularizing a lightweight container runtime and providing a simple way to package, distribute and deploy applications onto a machine, Docker provided the seeds or inspiration for the founders of Kubernetes



Single node

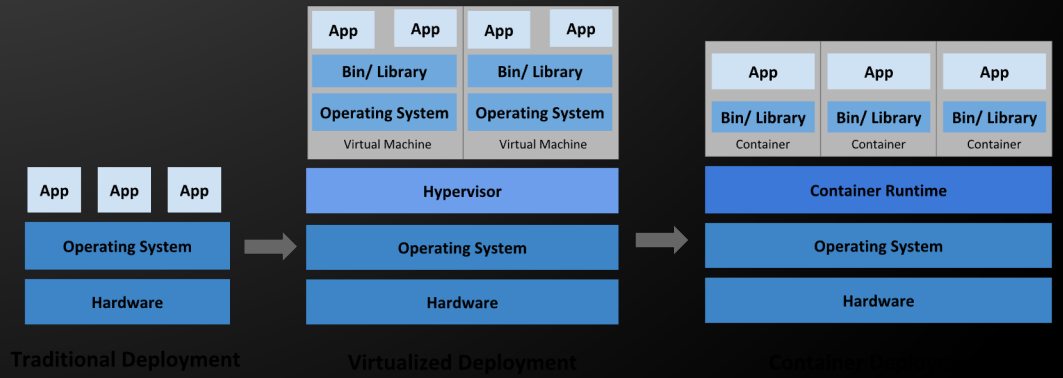
While Docker had changed the game for cloud-native infrastructure, it had limitations because it was built to run on a single node, which made automation impossible

container orchestrator

For instance, as apps were built for thousands of separate containers, managing them across various environments became a difficult task where each individual development had to be manually packaged

Resume

Traditional deployment era: Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.
Virtualized deployment era: As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.
Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.
Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.
Container deployment era: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.
Containers have become popular because they provide extra benefits, such as:
•Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
•Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
•Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
•Observability: not only surfaces OS-level information and metrics, but also application health and other signals.
•Environmental consistency across development, testing, and production: runs the same on a laptop as it does in the cloud.
•Cloud and OS distribution portability: runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
•Application-centric management: raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
•Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
•Resource isolation: predictable application performance.
•Resource utilization: high efficiency and density.



Timeline

2003

- Borgs: the first unified container management system

2013

- Omega its second-generation container management system

2013

- Docker Developed by dotCloud, a Platform-as-a-Service (PaaS) technology company

2014

- Kubernetes was created by Google engineers in 2014

2016

- became the Cloud Native Computing Foundation’s first hosted project

2 – Let's get to know k8s

Logo

Name of k8s

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s"

Many of the developers of Kubernetes had worked to develop Borg and wanted to build a container orchestrator that incorporated everything they had learned through the design and development of the Borg and Omega systems to produce a less complex open-source tool with a user-friendly interface (UI). As an ode to Borg, they named it Project Seven of Nine after a Star Trek: Voyager character who is a former Borg drone. While the original project name didn't stick, it was memorialized by the seven points on the Kubernetes logo

Stats

the Kubernetes community have added

74,680+
Contributors

314,188+
Code commits

263,906+
Pull requests

3.4 million+
Contributions

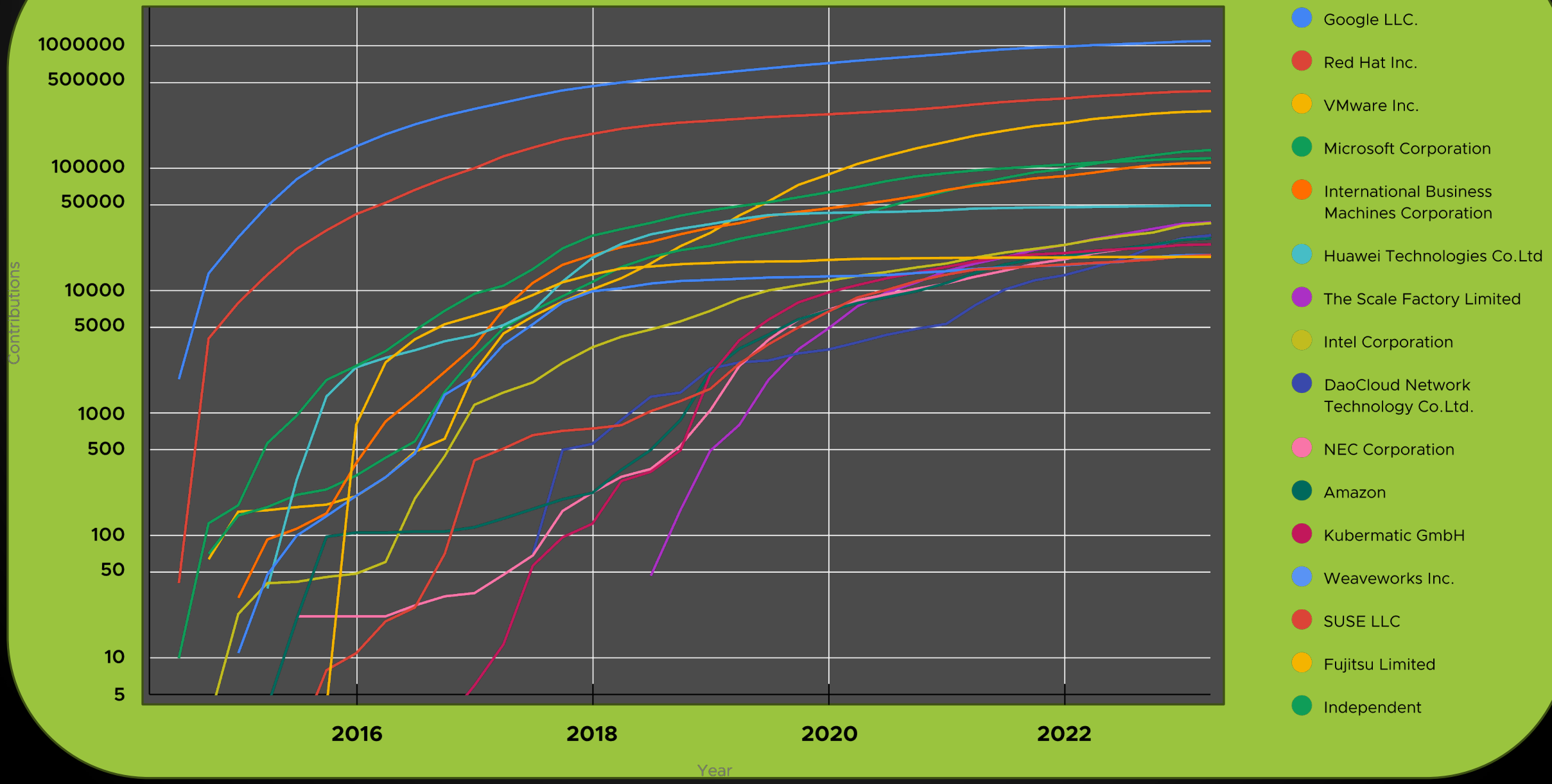
7,812+
Contributing companies

Is the second largest open source project in the world after Linux and is the primary container orchestration tool for 71% of Fortune 100 companies.

Joe Beda made the **first commit** to Kubernetes on June 6, 2014. Kubernetes joined CNCF on March 10, 2016, and on March 6, 2018, was the first CNCF project to graduate.

Kubernetes Project Journey Report | CNCF

CUMULATIVE GROWTH OF KUBERNETES CONTRIBUTIONS BY COMPANY Q2 2014 – Q2 2023

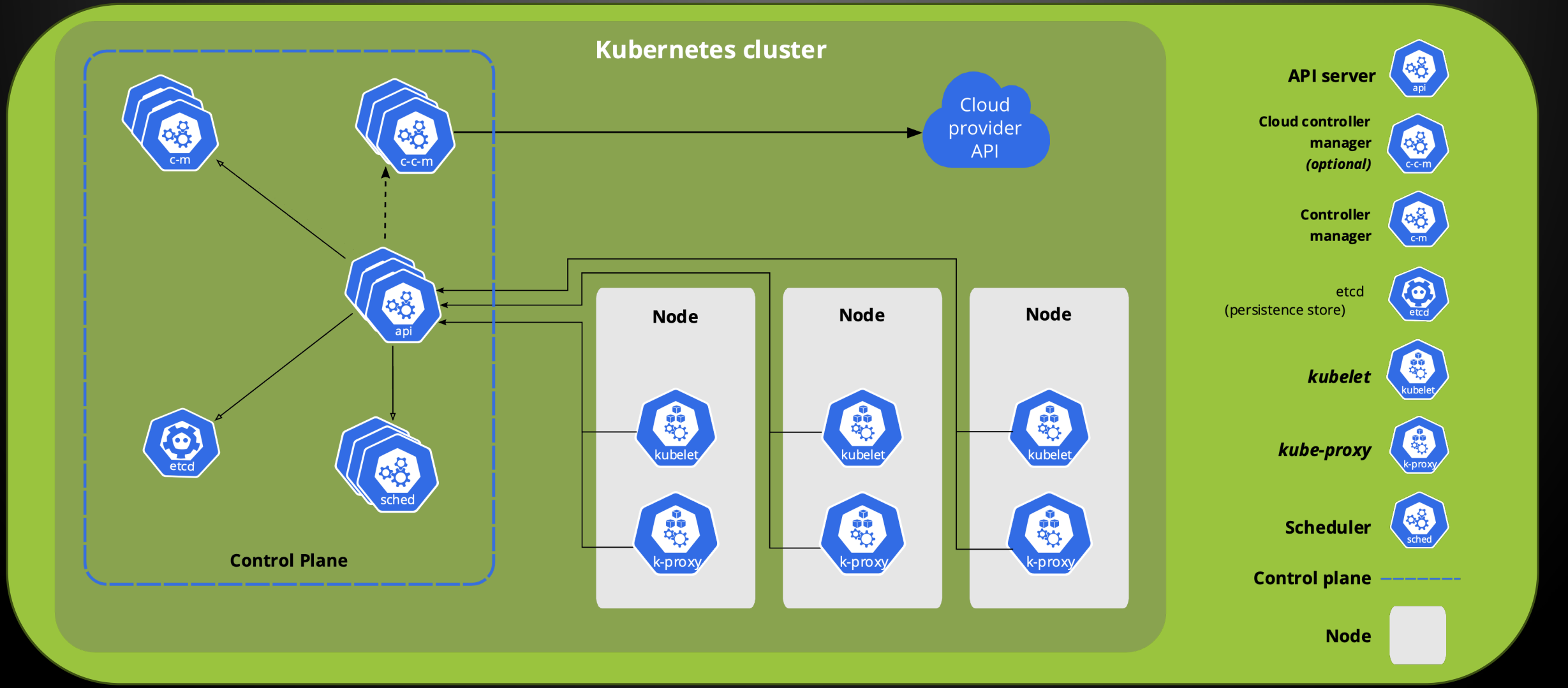


What not is

•Does not limit the types of applications supported. Kubernetes aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it should run great on Kubernetes.
•Does not deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements.
•Does not provide application-level services, such as middleware (for example, message buses), data-processing frameworks (for example, Spark), databases (for example, MySQL), caches, nor cluster storage systems (for example, Ceph) as built-in services. Such components can run on Kubernetes, and/or can be accessed by applications running on Kubernetes through portable mechanisms, such as the Open Service Broker.
•Does not dictate logging, monitoring, or alerting solutions. It provides some integrations as proof of concept, and mechanisms to collect and export metrics.
•Does not provide nor mandate a configuration language/system (for example, Jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
•Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.
•Additionally, Kubernetes is not a mere orchestration system. In fact, it eliminates the need for orchestration. The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible.

Components

cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.



Control Plane Components

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod

control plane setup that runs across multiple machines.

Kube-apiserver

- exposes the Kubernetes API
- scale horizontally

etcd

- highly-available key value store used as Kubernetes' backing store for all cluster data.

Kube-scheduler

- watches for newly created Pods with no assigned node, and selects a node for them to run on

Kube-controller-manager

- Node controller: Responsible for noticing and responding when nodes go down.
- Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
- EndpointSlice controller: Populates EndpointSlice objects (to provide a link between Services and Pods).
- ServiceAccount controller: Create default ServiceAccounts for new namespaces.

cloud-controller-manager

- only runs controllers that are specific to your cloud provider.
  Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route controller: For setting up routes in the underlying cloud infrastructure
- Service controller: For creating, updating and deleting cloud provider load balancers
-

Node Components

run on every node, maintaining running pods and providing the Kubernetes runtime environment.

control plane setup that runs across multiple machines.

kubelet

- Agent makes sure that containers are running in a Pod.

Kube-proxy

- kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
- kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster

Container-runtime

- A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.
- supports container runtimes such as containerd (Docker), CRI-O, and any other implementation of the Kubernetes CRI

Addons

Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

DNS

- Agent makes sure that containers are running in a Pod.

WebUI (Dashboard)

- kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
- kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster

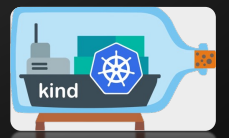
Container Resource Monitoring

- A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.
- supports container runtimes such as containerd (Docker), CRI-O, and any other implementation of the Kubernetes CRI

Cluster Level Logging

Network plugins

	Minikube	K3S	MicroK8S	KinD	KOS
CNCF Certificated	✓	✓	✓	✓	✓
Single Node Cluster	✓	✓	✓	✓	✓
Multi Node Cluster	✓	✓	✓	✓	✓
Architecture Support	x86, ARM64, ARMv7, ppc64, s390x	x86_64, armhf, arm64, aarch64, s390x	x86, ARM64, s390x, POWER9	amd64,arm64	x86-64, arm64, armv7
Min. CPU Req.	2+	1+	1+	?	1+
Min. Mem Req.	2GB	512MB	540MB	?	1GB
Container Runtimes	Docker, containerd, CRI-O	Docker, containerd	containerd, kata	CRI-O, Docker	containerd
Networking	Calico, Cilium, Flannel, ingress, DNS, Kindnet	Flannel, CoreDNS, Traefik, Canal, Klipper	Calico, Cilium, CoreDNS, Traefik, NGINX, Ambassador, Multus, MetalLB	kindnetd	kube-router, Calico
Operating Systems	Linux, Windows, macOS	Linux	Linux	Linux, macOS, Windows	Linux, Windows Server 2019

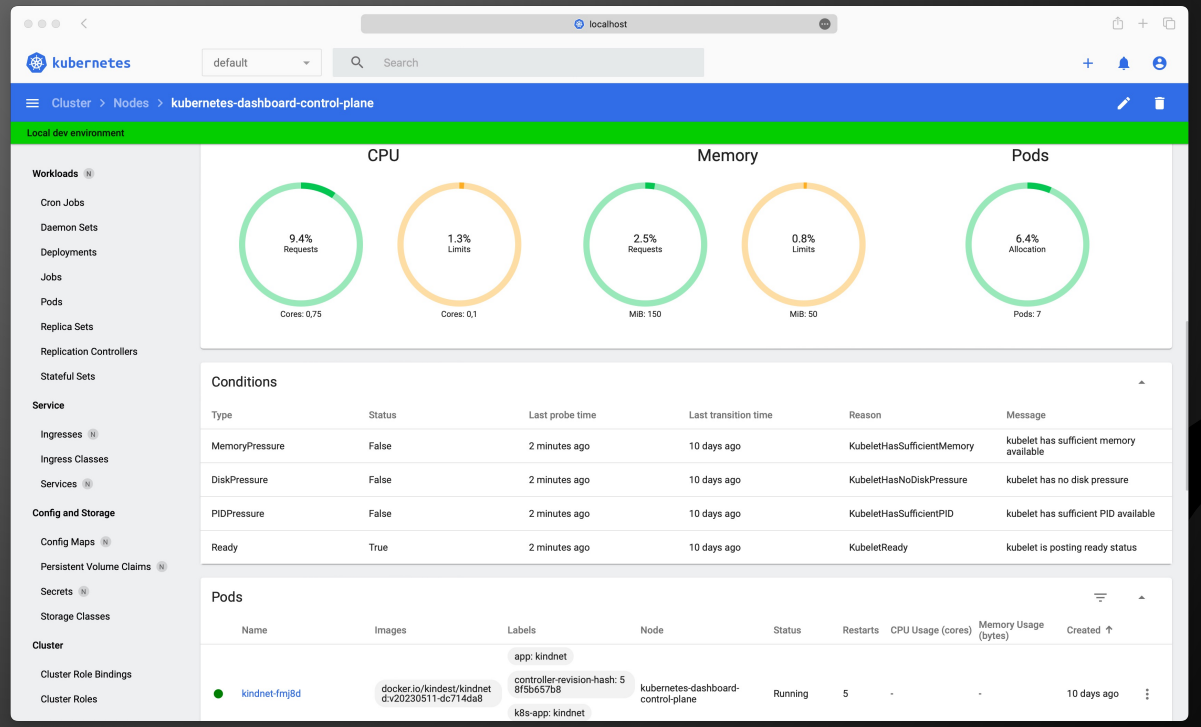


Manager

kubectl

LENS

K8Studio



[kubernetes/dashboard: General-purpose web UI for Kubernetes clusters (github.com)]

Objets in K8s

- persistent entities

manifests are YAML
(you could also use JSON format)

•What containerized applications are running (and on which nodes)
•The resources available to those applications
•The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

SPEC

STATUS

•apiVersion - Which version of the Kubernetes API you're using to create this object
•kind - What kind of object you want to create
•metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
•spec - What state you desire for the object

application/deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Pods

are the smallest deployable units of computing that you can create and manage

Is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

- **Pods that run a single container**. The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.
- **Pods that run multiple containers that need to work together**. A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit.

pods/simple-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Creatings pods

Deployment

DaemonSet

StatefulSet

Job

Usually you don't need to create Pods directly, even singleton Pods. Instead, create them using workload resources such as Deployment or Job. If your Pods need to track state, consider the StatefulSet resource.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # This is the pod template
    spec:
      containers:
      - name: hello
        image: busybox:1.28
        command: ['sh', '-c', 'echo
      restartPolicy: OnFailure
    # The pod template ends here
```

YOU CAN FIND ME

- ✨ https://youtube.com/maurobernal
- ✨ https://twitch.tv/maurobernal
- ✨ https://twitter.com/_maurobernal
- ✨ https://tiktok.com/@_maurobernal
- ✨ https://maurobernal.com.ar