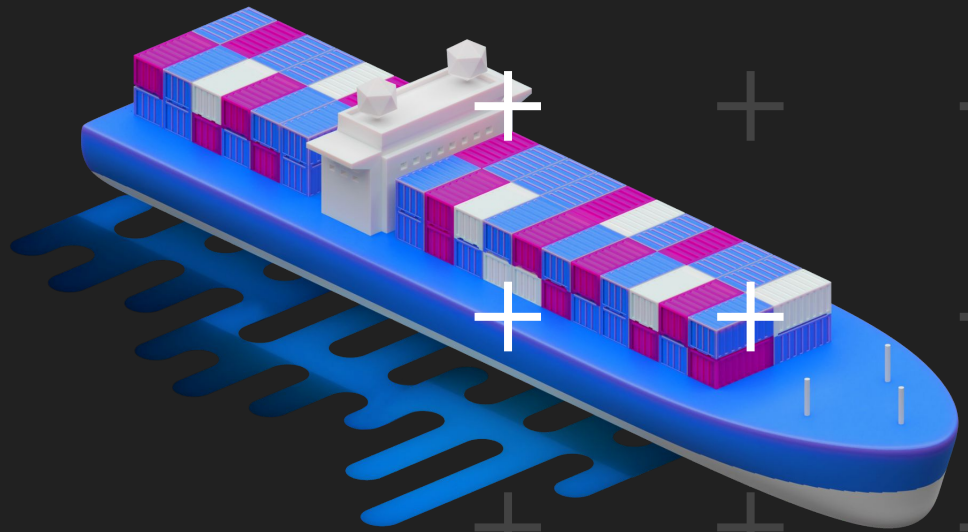
Intro to Kubernetes

for developers



April 8, 2024

ZHealth Documentation

Background

- Docker: A Project-Based Approach to Learning by Jason Cannon
- The Kubernetes Book: 2023 Edition by Nigel Poulton
- LFS158x: Intro to Kubernetes
- Documentation at kubernetes.io

What is Kubernetes?

- A way to automate the management of distributed services at scale running on unreliable hardware.
- Manual administration requires understanding the current state in order to make changes.
- This breaks down at scale: nobody can keep track of all the machines, services, settings, dependencies, versions, and cloud infrastructure as the number of services increases in orders of magnitude.

What does it do?

- You tell the **orchestrator** your **desired end state**.
- The orchestrator **applies changes** based on the **current state** that it's able to keep track of because it's an automated and distributed service itself.
- If something goes wrong, it's easier to **isolate** the problem down to a **transactional interaction of a few components**, so that it can be reproduced and fixed.
- Without this structure, problems are **band-aided** and Production team is endlessly **fighting fires**.

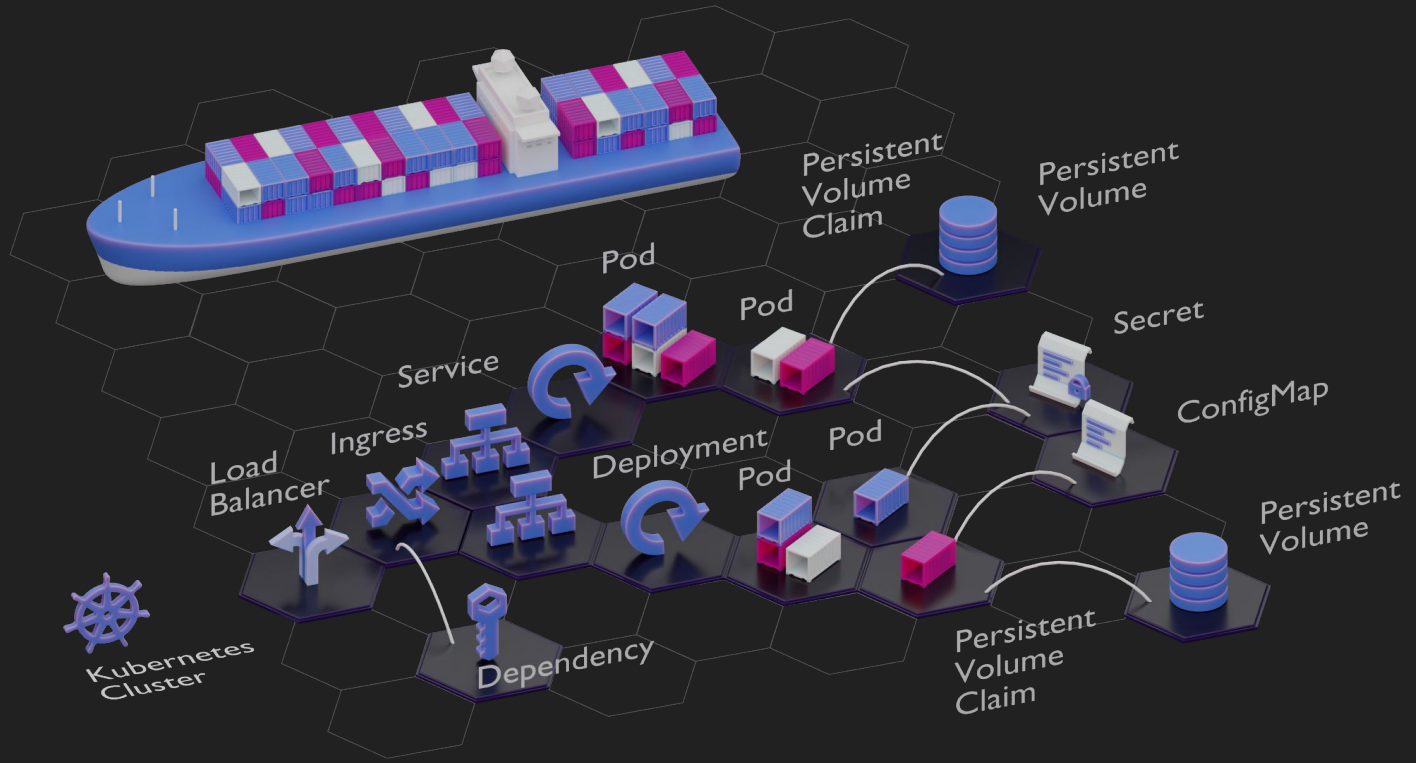
How is it used?

- Kubernetes builds on the concept of a **container**.
- It lets you deploy a **cluster** of **containerized services** by using a **manifest**.
- The **manifest** is a **declaration** of which **containers** you want to be running along with their **dependencies**.
- Kubernetes has a **package manager** called **Helm** that can be used to assemble a manifest from standard parts.

How does it work?

- Kubernetes reads the given **manifest** and **continuously applies changes** to the underlying **infrastructure**, **services**, and **configuration**.
- Kubernetes talks to the underlying infrastructure via **plugins**, such as the **container runtime**, **storage**, **software-defined networking**, and **cloud services** like load balancers and firewalls.
- **Manifest** changes are applied via **dashboard** or **CLI**.
- Manifests can be **source controlled** & applied by **CI**.

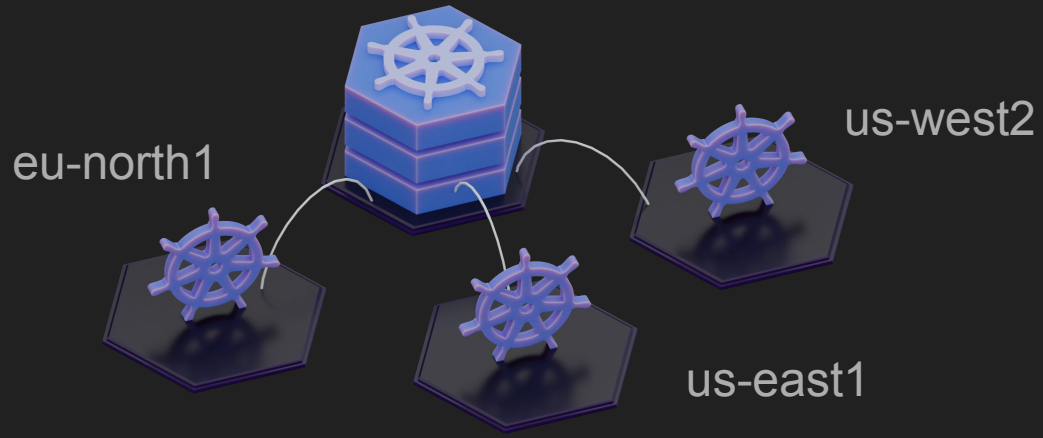
What's in a cluster?



A cluster of clusters?

+

- Kubernetes Federation lets you manage clusters in multiple availability zones to enable failover.



Kubernetes History

- Kubernetes was created by **Google** to support **Search** and **Ads** products responsible for most of its revenue.
- Both products required running a **large number of jobs** on **cheap hardware** with a **free operating system**, all Google could afford when they were starting out.
- Kubernetes was **released** to **steer developers** toward **Google** and poach customers from Amazon and Microsoft.
- Amazon and Microsoft promptly released **plugins** that only work with Kubernetes on **their platforms**.

Why use Kubernetes?

+

- Standardizes cloud provider features.
- Simplifies deploying a new version of a service.
- Unhealthy services are automatically restarted.
- Unhealthy machines are taken offline and services are re-scheduled on different machines automatically.
- Simplifies dynamically assembling infrastructure and services taking all dependencies into account.
- Canary and Blue/Green deployments are supported out of the box.

Cons

- The complexity of maintaining large distributed systems have simply been shifted to another frequency.
- Instead of struggling with network administration, now you struggle with writing manifests and configuring plugins so that orchestrator can apply changes.
- Complicated enough that you can't muddle your way through it for real use in production.
- Another package ecosystem to learn.

Kubernetes in Practice

- **Security** is complicated to setup **properly**, with lots of conflicting **constraints** to keep in mind.
- High-availability/multi-zone deployment requires **Kubernetes Federation**.
- **Software-defined networking** was built on top of Linux, understanding limitations requires sysadmin experience.
- You have to memorize what each feature can handle.
- **Plugin system** for cloud-provider-specific services ensures learning "never stops".

Growing pains

- Some features like load balancing and software-defined networking were meant to be a core part of the system, but after the public release it became apparent that they didn't handle many real-world scenarios.
- In the end Kubernetes became "a little of everything for everybody", with muddy "design by committee".
- Some features superseded by plugins but left in for backward compatibility.

Should we use it?

+

- **Maintaining services at a scale**: we are not yet affected by issues of scale.
- **Reducing cloud hosting costs**: we won't realize cost savings unless we switch to a different cloud.
- **Running on our own hardware**: running on managed cloud is cheaper without a production team.
- **Our deployments are manageable**, can’t afford canary & blue/green yet due to cloud infrastructure costs.