Deploy a MongoDB cluster on Kubernets using Percona Operator

Corrado Pandiani
Senior Architect

Milan, May 26th 2025

Corrado Pandiani
Senior Architect

This is me

- Open Source enthusiast
- MySQL and MongoDB expert
- Worked in the past as DBA, Web Developer, Project Manager, CRM and BI developer and instructor
- Spent 22 years in the football industry for a worldwide popular team
- Perconian since early 2018

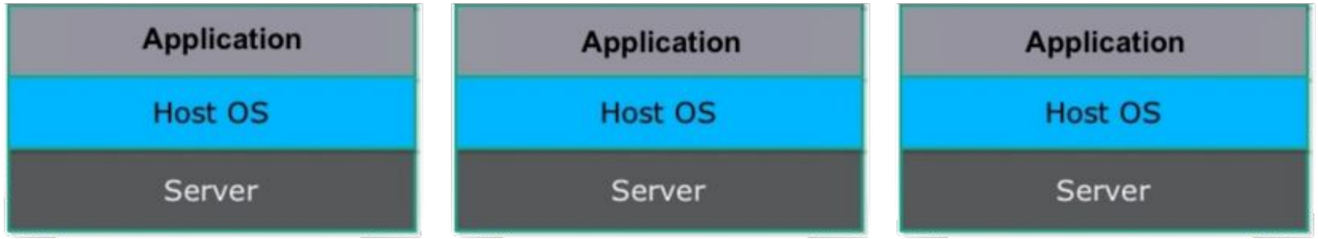
Agenda

1. Containers
2. Kubernetes
3. Percona Operator for MongoDB
4. Deploy a MongoDB cluster

Containers

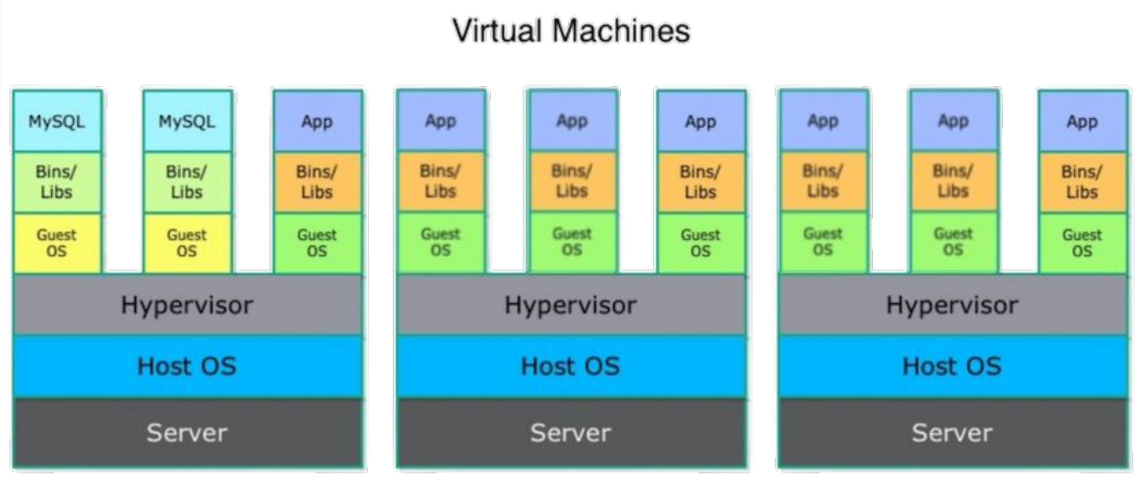
In the beginning

- There were physical servers (i.e.: "bare metal")
    - Operating Systems
    - Applications
- Scaling == Add more hardware
- Inefficient resource usage



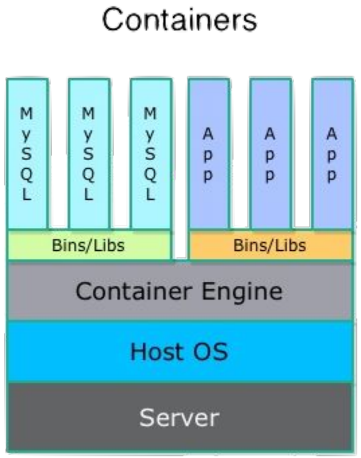
Then Virtual Machines

- Simulate physical machine
- Provide local file system
- Accessible over network
- Full/independent OS ("guest OS")
- Virtualized device drivers
- Resource and memory management
- Requires a hypervisor



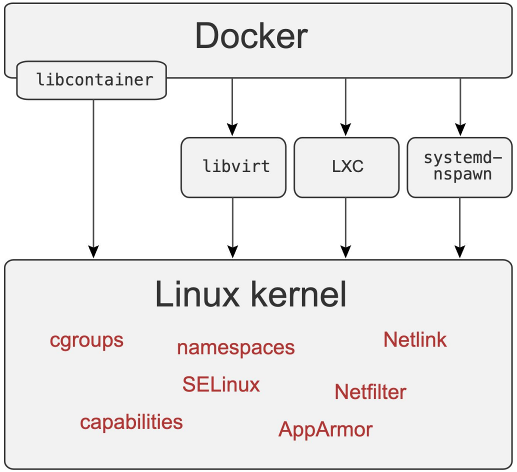
Containers (LXC)

- OS-level virtualization method
- Allows running multiple, isolated systems, using a single kernel ("host")
- Kernel provides *cgroup* (control group) functionality
  - CPU/memory/disk/network
  - No need for complete OS install
  - No device emulation/hypervisor
- "Lightweight" virtual machine
- Local file system
- Accessible over network
- Container itself is isolated process
- "Bare Metal" performance



Containers (Docker)

- Open Source PAAS project
    - Initially built to extend LXC
    - LXC was eventually replaced by the libcontainer library (Go)



- Can package applications and dependencies as "images"
- "git-like" capabilities for tracking versions of each container
- Build new container using others as base
- Ecosystem for sharing pre-build containers
- The "easy button"

Containers pros and cons

- Pros
  - No independent OS overhead
  - "jailed" environment
  - All-in-one deployment
- Cons
  - Hard to manage multiple containers
  - Disk Persistence is complicated
  - Not fully isolated workload

Kubernetes

Containers are simple enough for single use

- That's easy for 5-10 containers on a single host
  - …but what about 50-100 containers…
  - …on 20 different hosts!

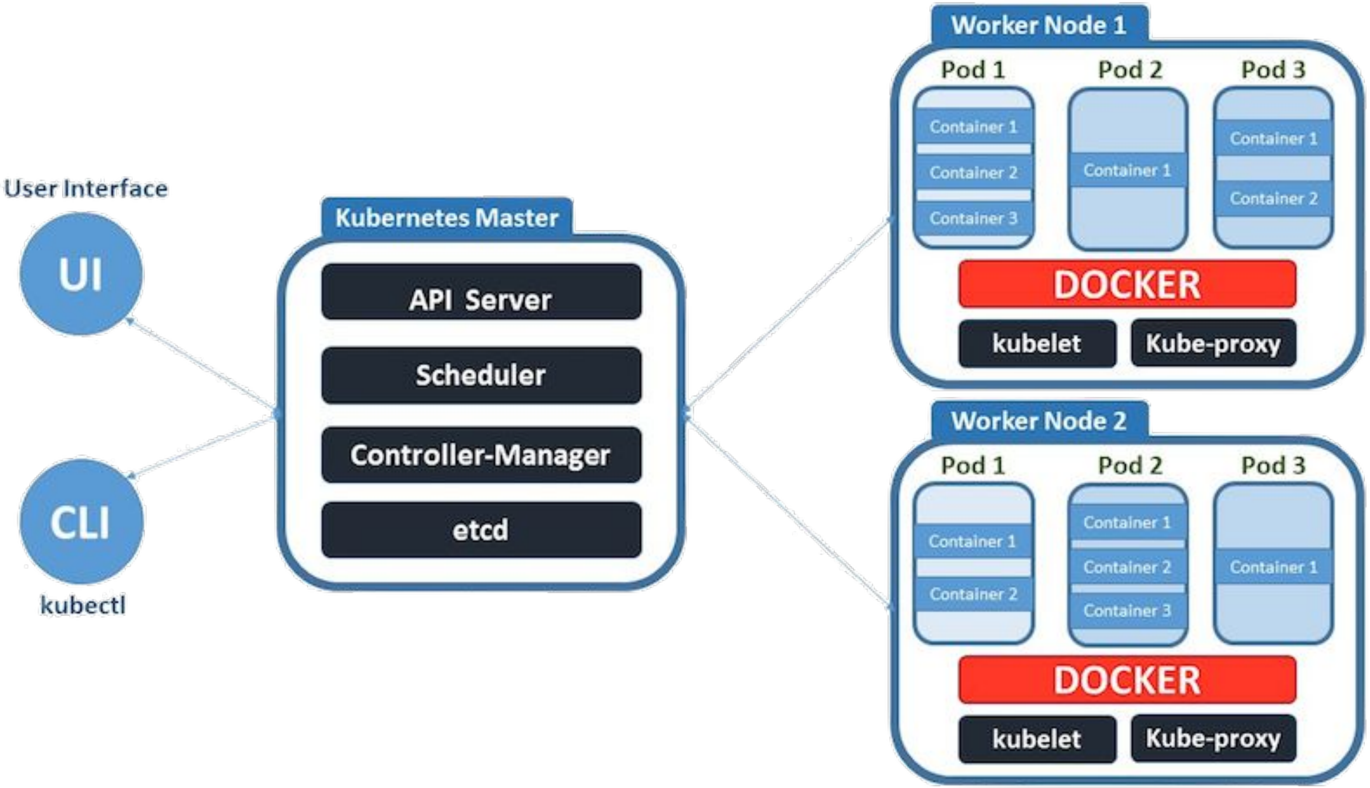
Containers Orchestration

- Now you have to run hundreds of containers
    - across, potentially, hundreds of hosts
- Health checks on the containers
- Launching X copies for a particular container
- Scaling the number of containers up and down depending on load
- Performing rolling updates across containers
- Services in container X discovering services in Y

What is Kubernetes?

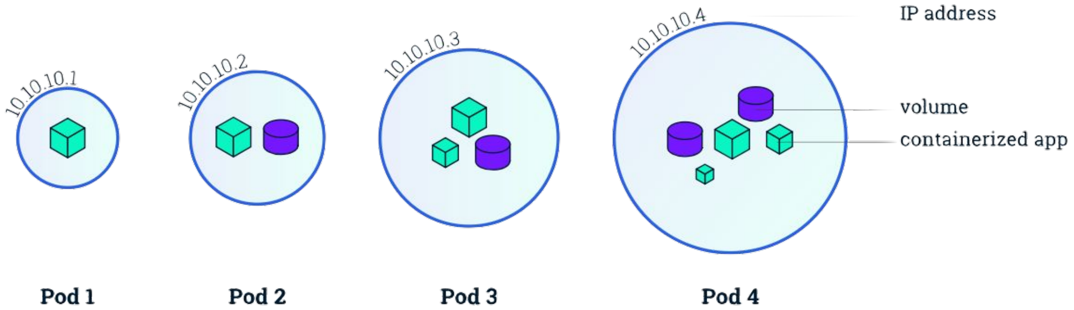
- Greek for "captain", or "navigator"
- Created by Google, 2014
  - Heavily influenced by Google's Borg system
- Written in Go
- 2015, Google partnered with the Linux Foundation to form the Cloud Native Computing Foundation (CNCF)
  - CNCF is the current maintainer
- A cluster, consisting of at least one control plane and multiple worker machines ("nodes")

Architecture



Peas in a Pod

- A unit of deployment
  - If single containers are deployed, then you can generally replace the word "pod" with "container" and accurately understand the concept
- A group of one or more containers, with shared storage/network, and a specification for how to run the containers
- A pod's containers are always co-located and co-scheduled, and run in a shared context
- A pod receives a unique IP to prevent port conflicts



Kubernetes Flavors

- Open Source
  - Rancher
  - Docker Kubernetes Service
- Cloud Managed
  - Amazon EKS
  - Google GKE
  - Azure AKS
- Enterprise
  - OpenShift (Red Hat)
  - VMWare Tanzu
  - Mirantis

Kubernetes on your laptop

- Run a minimal installation with all K8s components in a single machine
- Some alternatives:
  - K3s
  - Kind (Kubernetes in Docker)
  - Microk8s
  - Minikube

Kubernetes Operators

- An Operator is a method of packaging, deploying and managing a Kubernetes application
- Analogous to a *systemd* service, but manages an application deployed on Kubernetes
- The Operator itself runs in a container inside a pod

Percona Operator for MongoDB

PERCONA

Kubernetes Operators

Kubernetes Operator for MongoDB

- Automates the creation, modification, or deletion of Percona Server for MongoDB (PSMDB) replica sets or sharded clusters.
- Based on best practices for the configuration of PSMDB, the Operator provides many benefits; but saving time, and having a standard environment are the most important
- Supported platforms*
  - Google Kubernetes Engine (GKE)
  - Amazon Elastic Container Service for Kubernetes (EKS)
  - OpenShift Container Platform
  - Azure Kubernetes Service (AKS)
  - Minikube

(*) Other Kubernetes platforms may also work but have not been tested

Minimal requirements

- A cluster running an officially supported platform contains at least 3 nodes and the following resources:
  - 2 GB of RAM
  - 2 CPU threads per Node for Pods provisioning
  - 60GB of available storage for Private Volumes provisioning
- Consider using 4 CPU / 6 GB of RAM if sharding is turned on (the default behavior)

Installation Options

- We recommend installing the Operator with the `kubectl` command line utility
- It is the universal way to interact with Kubernetes
- Alternatively, you can install it using **Helm**
  - Helm is the package manager for Kubernetes
  - A Helm chart is a package that contains all the necessary resources to deploy an application to a Kubernetes cluster
  - You can find Percona Helm charts in percona/percona-helm-charts repository in Github

PERCONA

Deploy a MongoDB cluster

PERCONA

Kubernetes Operators

Quick installation

- Create a namespace on Kubernetes and make it the default

```
$ kubectl create namespace <namespace name>
$ kubectl config set-context $(kubectl config current-context)
--namespace=<namespace name>
```

- Deploy the operator

```
$ kubectl apply --server-side -f
https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.18.0/d
eploy/bundle.yaml
```

- Clone the git repository

```
$ git clone -b v1.18.0 https://github.com/percona/percona-server-mongodb-operator
```

Quick installation

- Edit the Custom Resource file `deploy/cr.yaml`
- Deploy the MongoDB Cluster

```
$ kubectl apply -f deploy/cr.yaml
```

- Check when the cluster is **ready** status. It means it is deployed correctly

```
$ kubectl get psmdb
```

Connect to MongoDB cluster

- List the secret objects

```
$ kubectl get secrets -n <namespace>
```

- View the Secret contents to retrive the admin user credentials

```
$ kubectl get secret my-cluster-name-secrets -o yaml
```

- The actual login name and password on the output are base64-encoded. To bring it back to a human-readable form, run:

```
$ echo 'MONGODB_DATABASE_ADMIN_USER' | base64 --decode
$ echo 'MONGODB_DATABASE_ADMIN_PASSWORD' | base64 --decode
```

Connect to MongoDB cluster

- Run a container with a MongoDB client and connect its console output to your terminal.

```
$ kubectl run -i --rm --tty percona-client
--image=percona/percona-server-mongodb:7.0.14-8 --restart=Never -- bash -il
```

Connect to MongoDB cluster

- Now run **mongosh** tool inside the *percona-client* command shell using the admin user credentials you obtained from the Secret

  <u>If sharding is on</u>

  ```
  $ mongosh
  "mongodb://clusterAdmin:clusterAdminPassword@my-cluster-name-mongos.<namespacename>
  .svc.cluster.local/admin?ssl=false"
  ```

  <u>If sharding is off</u>

  ```
  $ mongosh
  "mongodb+srv://clusterAdmin:clusterAdminPassword@my-cluster-name-rs0.<namespacename
  >.svc.cluster.local/admin?replicaSet=rs0&ssl=false"
  ```

Thank You

corrado.pandiani@percona.com