A quick look at Kubernetes

Hairizuan
Sparkline
20171028

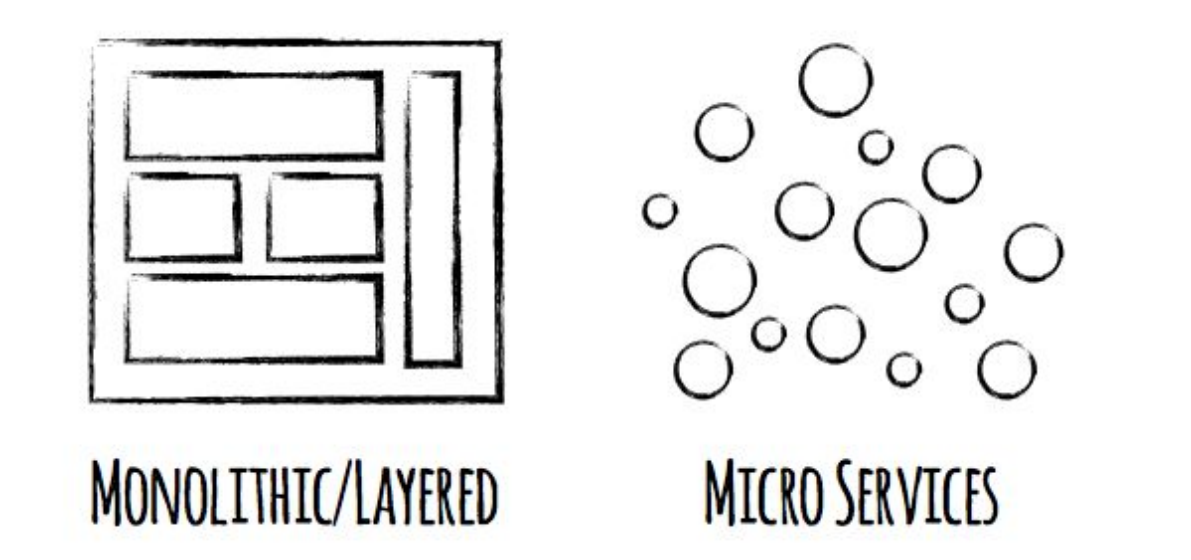
What is Kubernetes?

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

Why use Kubernetes?

Monoliths vs Microservices

- A change in the way how developers develop applications for an organization.

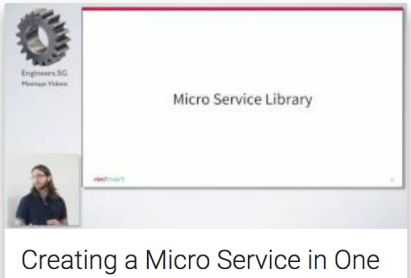


Monoliths vs Microservices

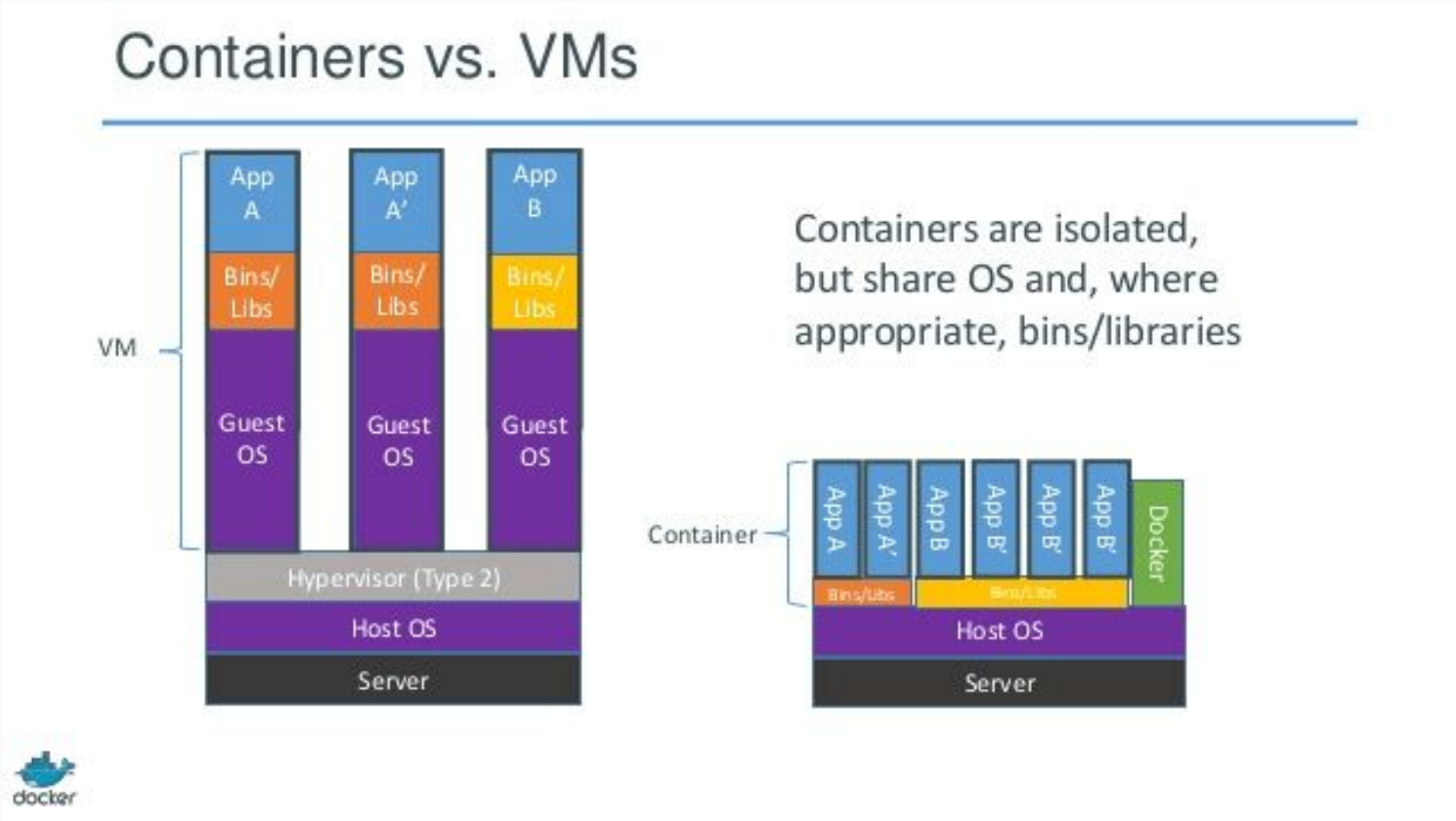
- Independent Deployments
  - Services can be scaled independently
  - Services can be deployed independantly
  - Less likely to cause catastrophic failures if a service goes down (Less risk)
- Strong Module Boundaries
  - Highly likely to reduce or prevent coupling between services
  - Harder to pull functionality from different module
- Allow polyglot environments to exist
  - Use the best language for each use case

VMs in a Microservice architecture

- 1 service 1 server?
- VMs are quite heavy - Takes some time to set up
- Use Puppet or Ansible to compose such VMs into the cloud architecture
- https://engineers.sg/video/creating-a-micro-service-in-one-step-redmart-x-devfest-asia--1270



VMs vs Containers



Using Docker to aid Microservices

- Marketing Tagline: Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

- Encapsulation of services into a package. The aim is to ensure reproducible builds; builds can be moved across the different environments and ecosystems. Builds continue to work and operate similarly. Less of "Works on my machine" scenario.

- Note: Not a necessary piece for microservices. It just turns out that Docker containers are very light weight and can serve the needs of the required application with quick start time.

Demo Time!
Build a docker web application

Commands:

- **List all docker containers**
  docker ps -a
- **Run a nginx web server**
  docker run -p 8080:80 nginx
- **Build a dockerfile and play around with it**
  docker build -t example_python .
  docker run -td example_python
  docker exec -it <image_name> /bin/bash

Demo Time!
Build a docker web application

Commands

- **Build a web application example**
  docker build -t webapp .
  docker run -p 8080:8080 --name webapp

Why not just stop at Docker?

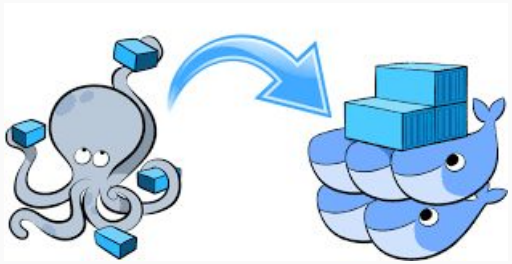
- Issues with scheduling of workloads
  The usual problem of deploying such workloads becomes a problem of tracking such instances

- Some of the usual problems
  - Tracking and maintaining state of applications - whether the application is healthy or not
  - Effort needed to scale systems up and down
  - Upgrading of applications - Need to prioritize

- There is a need for orchestration of applications on a platform

Orchestration Tools

- We would need container orchestration tools in order to deploy such services with minimal efforts. There are various efforts done by various companies to get them running but here are a few:
  - Docker Compose + Docker Swarm
  - Kubernetes (Use managed Kubernetes via GKE)
  - Mesos
  - AWS ECS (Elastic Container Service) and ACS (Azure Container Service)



Let’s start with Kubernetes

We will be deploying the following:

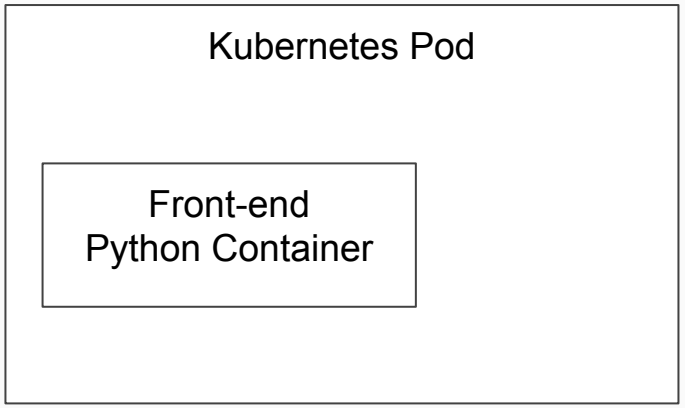
- Python Web Application
- Go Application

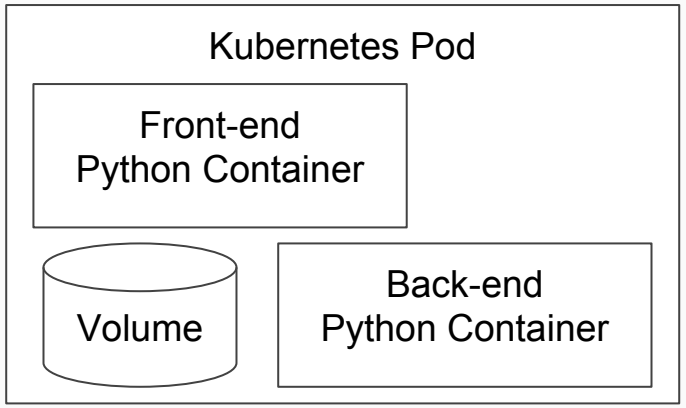
We will cover the following:

- Authentication to GKE (Managed Kubernetes)
- A hello world test run
- Deploying a flask web application
- Updating the flask web application
- Scaling the flask web application

Some Kubernetes Concepts

- Pods
  A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster. It is something that encapsulates a bunch of containers.





Some Kubernetes Concepts

- Deployments
  Declaratively control the replication of pods via replica sets

- Services
  Expose the endpoints and ports from containers within the pods to the outside world

Authentication to GKE

Commands:

- gcloud config list
- gcloud container clusters list
- gcloud container clusters get-credentials <cluster-name> --zone <zone-name>

Trying it out with nginx

Commands:

- kubectl run nginx --image=nginx:1.13.5 --port=80
- kubectl expose deployments nginx --target-port=80 --port=80 --name=nginx-service --type=LoadBalancer

Cleanup commands

- kubectl delete service nginx
- kubectl delete deployment nginx

Trying to deploy the python application

Commands:

- **Build the docker container**
  docker build -t gcr.io/<project_name>/awesome-web-app:v1 .
- **Push container to private repository**
  gcloud docker -- push gcr.io/<project_name>/awesome-web-app:v1
- **Create the deployment**
  kubectl run --image=gcr.io/<project_name>/awesome-web-app:v1 awesome-web-app
- **Create the service and expose it externally**
  kubectl expose deployments awesome-web-app --target-port=8080 --port=80 --name=awesome-web-app --type=LoadBalancer

Trying to deploy the go web api application

Commands:

- **Build the docker container**
  docker build -t gcr.io/<project_name>/go_api:v1 .
- **Push container to private repository**
  gcloud docker -- push gcr.io/<project_name>/go_api:v1
- **Create the deployment**
  kubectl run --image=gcr.io/<project_name>/go_api:v1 go-api
- **Create the service and expose it only within the cluster**
  kubectl expose deployments go-api --target-port=8080 --port=80 --name=go-api --type=ClusterIP

Trying to update the python application

Commands

- **Rebuild and push image**
  docker build -t gcr.io/<project_name>/awesome-web-app:v2 .
  gcloud docker -- push gcr.io/<project_name>/awesome-web-app:v2
- **Set image of deployment to new image**
  kubectl set image deployment/awesome-web-app
  awesome-web-app=gcr.io/<project_name>/awesome-web-app:v2 --record

Scaling the python application

Commands

- **Scale out awesome web app to 8 replicas**
  kubectl scale deployments awesome-web-app --replicas=8
- **Scale down awesome web app to 1 replica**
  kubectl scale deployments awesome-web-app --replicas=1

Rolling update of the python application

Commands

- **Deploy a new version of awesome web app**
  docker build -t gcr.io/<project_name>/awesome-web-app:v3 .
  gcloud docker -- push gcr.io/<project_name>/awesome-web-app:v3
- **Scale up awesome web app to 8 replicas**
  kubectl scale deployments awesome-web-app --replicas=8
- **Set the image of awesome web app to new version**
  kubectl set image deployment/awesome-web-app
  awesome-web-app=gcr.io/<project_name>/awesome-web-app:v3 --record

Rollback to previous version

Commands

- **View history of deployments**
  kubectl rollout history deployment/awesome-web-app
- **Undo deployment**
  kubectl rollout undo deployment/awesome-web-app

Demo End

Other things to look into

- Transport Layer
  - HTTPS requirement
  - Use RPC instead of http transport?
- Packaging multiple containers together
  - Application requires Database, Caching Layer
  - Docker has the compose project
  - Kubernetes has the helm project
- Ensuring reliability
  - Health Checks?
- Logging solution
  - What kind of logging solution to use? Use stackdriver? Datadog? Prometheus?
- Frontend layer
  - How to combine output from all the different services to a single frontend layer -> How to manage the team for that?

https://hairizuan.wordpress.com/

Hairizuan
Sparkline
hairizuanbinnoorazman@gmail.com