Zero to Docker and Kubernetes

A journey into the cloud-native world of containers

$whoami

- Yashvardhan Kukreja (Yash)
- Software Engineer @ Red Hat
- Masters @ University of Waterloo
- Working on Openshift, Kubernetes and cloud-native stuff
- Free time? - Open source Dev, Running, Building stuff

“But why the heck should I even listen about containers?”

- Backs the “cloud computing” world which everyone’s relying upon lately.
- Gives a lens into how these bigshot startups ship their products beyond “just coding”
- Witness Computer Science in its flesh - Super fun!
- Competitive advantage
  - When was the last time you heard someone learning about Containers v/s a web dev project.
- Pays well ;)

The ride we're up for today!

- We'll start from the early days of physical hosts.
- Travel across the virtualized realms of VMs, namespaces and containers.
- Finally, we'll land on the island of Kubernetes and take containers to the next level.

Evolution of IT Infrastructure

Physical hosts - Good ol’ days!

- Heavy dependence on physical hosts
  - You want to do something? Plug in a computer!
  - No websites, just good ol’ wires and flashy buttons.
- Quite inefficient and costly though
  - You’re forced to use a 16GBs RAM’d machine even if you want just 10GBs.
- Hard to scale
  - What if you realise that 10GBs is not enough and you want 20GBs of RAM?

VMs to the rescue!

- Virtual Machines == Operating System like a software (well, kinda)
- Imagine running MacOS on Windows!
- Have a computer with 32GBs of RAM but want to occupy only 8GBs?
    - Just create a VM occupying 8GBs of RAM

VMs even do more things

- Companies like Doordash want to use a software which was made for only Linux
  - Happens more often than not
- But say, they have some Windows server lying around
- They can install a Linux Virtual Machine on those Windows Servers
- Run that Linux-specific code on the Linux Virtual Machine
  - The code thinks it’s running on a legit VM

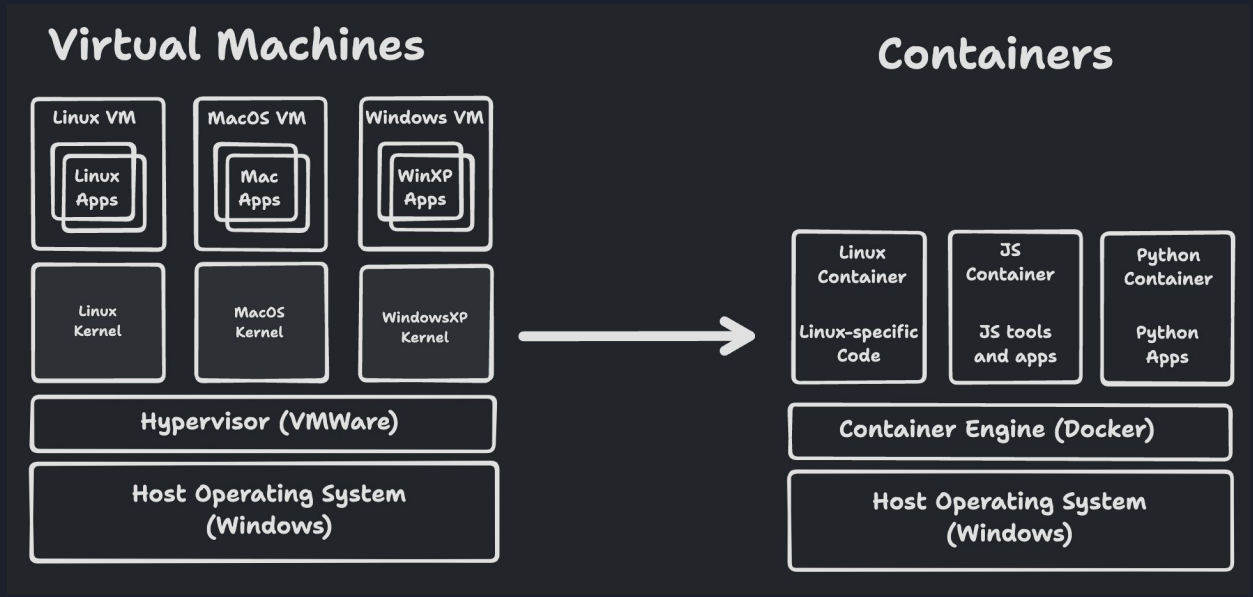
But well, nothing’s perfect

VMs are

- Slow to boot
- Heavy with their own UI and internal softwares
- Literally deploys its kernel over the hypervisor

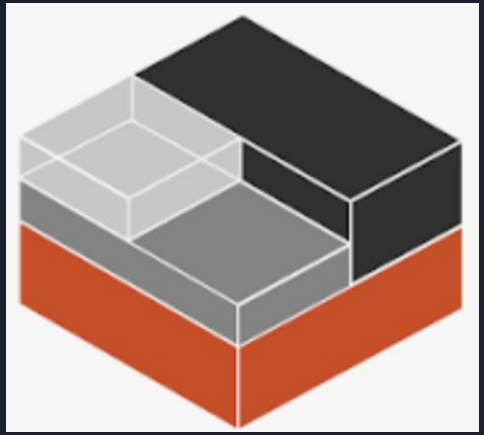
Containers to the rescue

- Lightweight, executable units of software
- Super quick to start and stop
- Unlike VMs, containers share the host OS kernel
- Have their own filesystem, CPU, memory, process space, and more, giving it a VM-ish feel.



History of containers (docker wasn't the OG)

- Process namespaces (cgroups) by Google in 2006.
- LXC containers in 2008.
- Docker ecosystem in 2013 <3
- Even podman came with its own powers!



docker

podman

Containers aren’t “just” lightweight, they are….

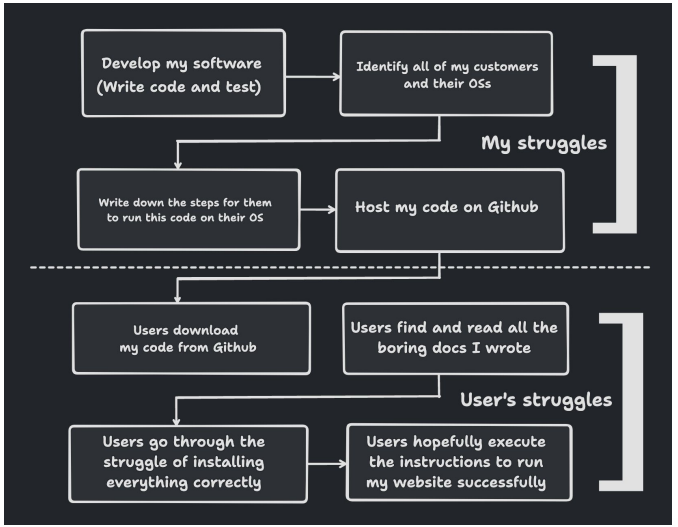
- Extremely easy to build (as images)
- Pre-baked with everything required to just run stuff
- The end-user just has to download and “run it”!
    - Docker engine will take care of the rest

Time to get our hands dirty

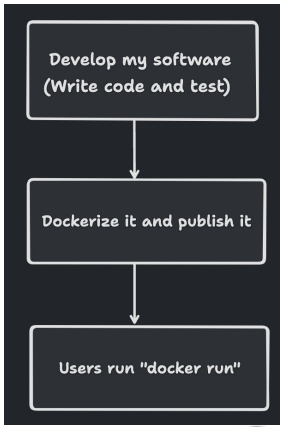
For Windows/MacOS - https://www.docker.com/products/docker-desktop/

For Linux - https://docs.docker.com/engine/install/ubuntu/


Without
Docker


Develop my software (Write code and test)
Identify all of my customers and their OSs
My struggles
Write down the steps for them to run this code on their OS
Host my code on Github
Users download my code from Github
Users find and read all the boring docs I wrote
User's struggles
Users go through the struggle of installing everything correctly
Users hopefully execute the instructions to run my website successfully


With
Docker


Develop my software (Write code and test)
Dockerize it and publish it
Users run "docker run"

But is Docker really enough?

- What if the container crashes?
- What if you want your containers to automatically increase/decrease as per the incoming traffic?
- What if you want your containers to be scheduled in a certain manner?
    - Say, you have a pool of computers and you want your containers to automatically run on just the right computer.

Enter the belle of the ball.... Kubernetes!

- Container Orchestration System.
- Manages your containers in a ton of ways.
- Automates the deployment, scheduling and scaling of your containers.
- Takes your containers to the next level.
- Saves your System Administrators from pulling apart their hair.

Long story short!

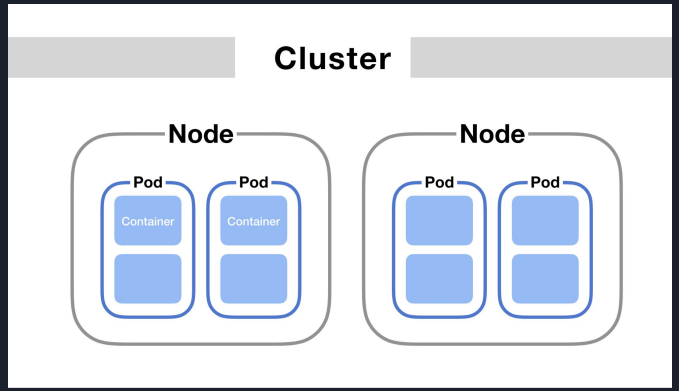
Long story short!

Kubernetes from a high-level

- A bunch of nodes (computers) working together
- Kubernetes runs on them.
- Takes incoming requests about which containers to run and how to run them.
- Schedules, Deploys and Manages them over these computers accordingly.

Let’s talk about Pods!

- The smallest deployable unit in Kubernetes.
- Can contain one or more containers (e.g., Docker containers).
- Containers in the same pod can talk to each other via “localhost” and even share the same storage
- These containers in a pod can contain and run your website.



A Pod will contain the container running our website's code!

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    app: hello-world
spec:
  containers:
    - name: hello-world
      image: yashvardhankukreja/hello-world
      ports:
        - containerPort: 3000
```

Just a random label to identify our pod later

our website container which will run in this pod

because our website will run at port 3000

But we don’t want our website to run alone

We want it to be safe and well-scaled

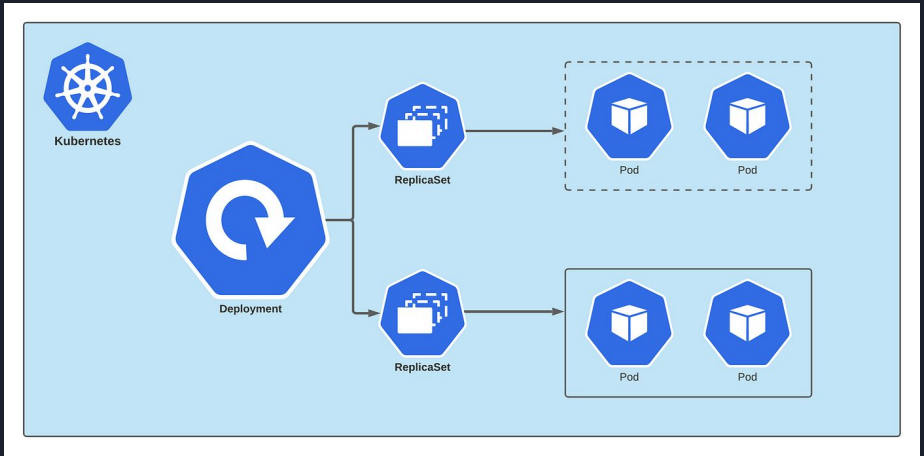
We want to run it at a higher scale

We want to guarantee that it heals itself if anything goes wrong

A single pod isn’t enough!

Time for “Deployments”

- A high-level abstraction which manages the pods.
- It makes sure to “always” keep a certain N number of pods alive.
- When you roll out a new version, it smartly upgrades the pods one-by-one ensuring that not at any time, all the pods would be down.



How a Deployment would look in our case

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello-world
          image: yashvardhankukreja/hello-world
          ports:
            - containerPort: 3000
```

we want 3 copies of our website no matter what happens!

our website pods are identified and selected by these labels (not names)

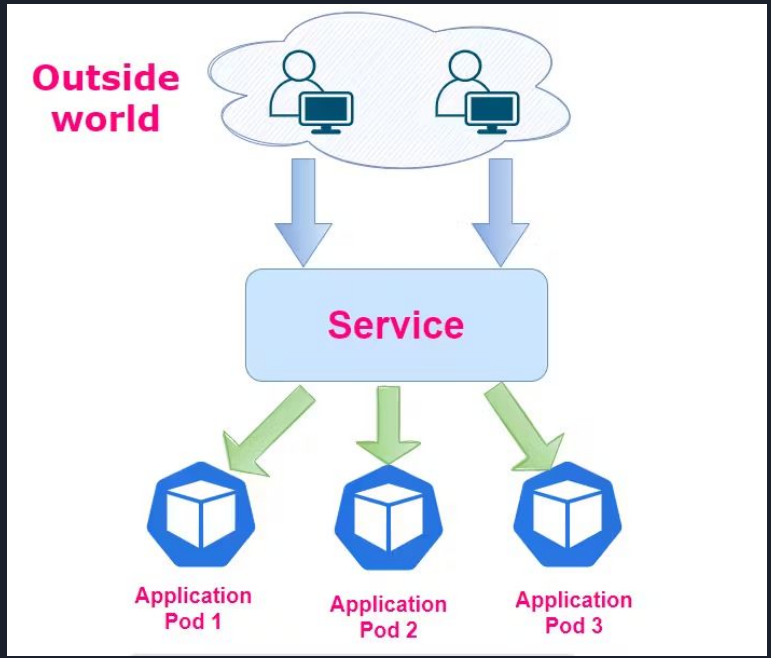
This is how each pod would look
(Exactly the pod template we applied before)

But now what? Nobody still uses our website!

We want it to be accessible to the world

We expose stuff through “Services”

- It is a logical grouping of pods, which we want to expose over a URL.
- A Service can be of multiple types:
  - ClusterIP - Private and not accessible to the outside world
  - NodePort - Public and accessible to the outside world directly via the Node it is on.
  - LoadBalancer - A new load balancer is created in real life which forwards traffic to the pods.
  - ExternalName - Maps and redirects the service to another URL (like CNAME).



Our website's "Service"

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-world-service
spec:
  selector:
    app: hello-world
  type: NodePort
  ports:
    - port: 3000
      nodePort: 30000
```

This service should point to all the (3) pods with this label

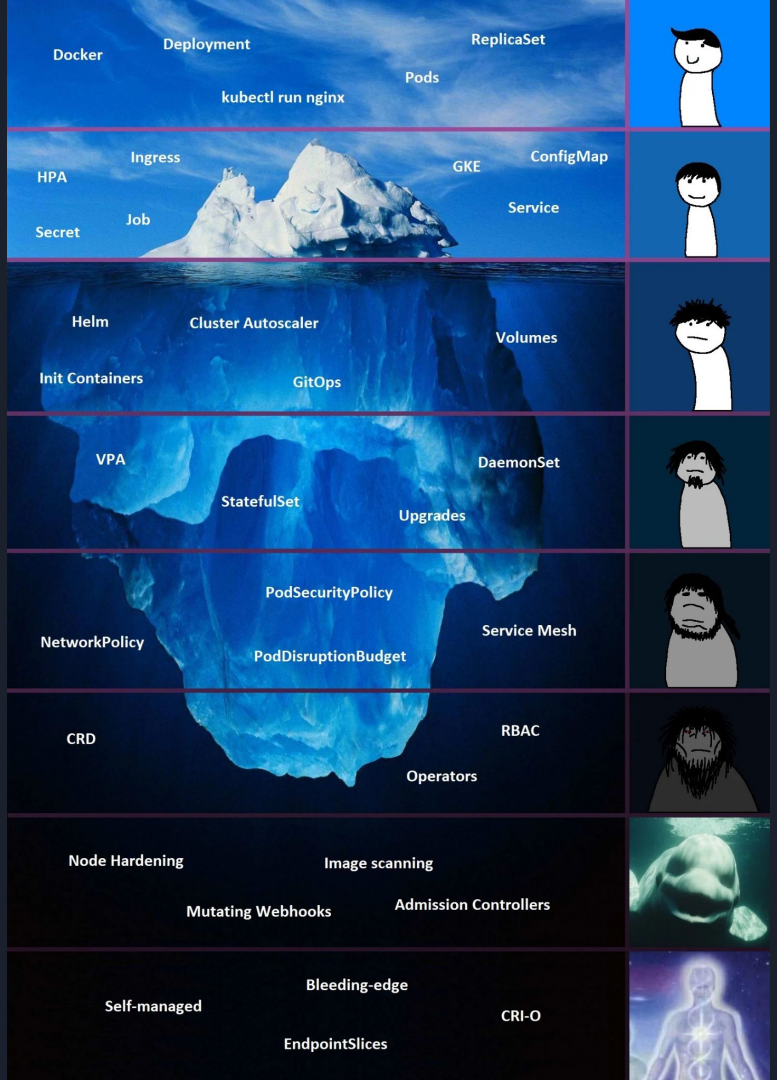
Expose this service publicly on the node

Incoming traffic should be directed to our website (container's port 3000)

Expose this service at the port 30000 of this node

Is that it though?

No, lol!


Docker
Deployment
ReplicaSet
Pods
kubectl run nginx
HPA
Ingress
GKE
ConfigMap
Service
Secret
Job
Helm
Cluster Autoscaler
Volumes
Init Containers
GitOps
VPA
DaemonSet
StatefulSet
Upgrades
PodSecurityPolicy
NetworkPolicy
Service Mesh
PodDisruptionBudget
CRD
RBAC
Operators
Node Hardening
Image scanning
Mutating Webhooks
Admission Controllers
Bleeding-edge
Self-managed
CRI-O
EndpointSlices

Concluding notes

- We dove into the world of containers
- We saw the history of computers and how it led to where we are today.
- We went through Docker and understood shipping applications via it.
- We didn’t stop though, we took our containers to the next level via Kubernetes.

Helpful Resources

[Under construction, you are welcome to access this later on :)]

All of this content (slides, programs) on

https://github.com/yashvardhan-kukreja/zero-to-docker-and-k8s-event

Feel free to connect with me on...

- Twitter @yashkukreja98
- GitHub @yashvardhan-kukreja
- LinkedIn @yashvardhan-kukreja
- My blog @yash-kukreja-98.medium.com

Thanks for your time folks!

Feel free to raise any questions