Lesson 3: Container Orchestration with Kubernetes

VMs v/s Dockers
Docker File
Docker Image
Docker Registry
Kubernetes
Understanding Kubernetes Resources
Useful kubectl Commands
Edge Case: Failing Control for K8s

VMs v/s Dockers

▼ Introduction to Dockers and containers

https://www.youtube.com/watch?v=JSLpG_spOBM

Difference between VMs and Dockers

Property	VM	Docker
Lightweight	👎	👍
Expensive	👍	👎
Portable	👎	👍
Fast	👎	👍
Hypervisor needed	👍	👎
Reuse resources	👎	👍
Apps run in isolation	👍	👍

Docker File

- A bunch of instructions (to package the application code and dependencies) that helps create a Docker Image
- Each operation is a layer in the file. If any change is made, only that layer is rebuilt

```
#format: INSTRUCTION arguments
FROM python:3.8
LABEL maintainer="your-name"
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

Docker Image

- A read-only template that is used to spin-up a runnable instance of an application

```
# build an image
# OPTIONS - optional;  define extra configuration
# PATH - required;  sets the location of the Dockefile and  any referenced files
docker build [OPTIONS] PATH

# Where OPTIONS can be:
-t, --tag - set the name and tag of the image
-f, --file - set the name of the Dockerfile
--build-arg - set build-time variables

# Find all valid options for this command
docker build --help
```

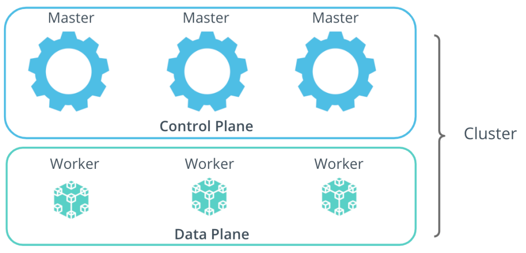
Docker Registry

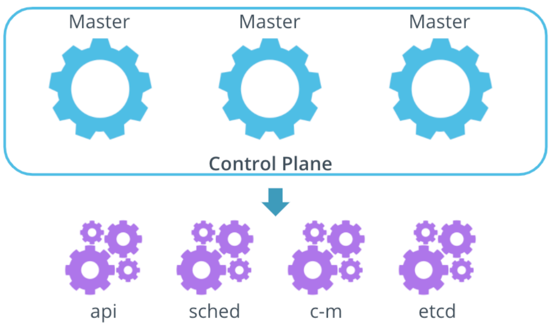
- A public registry where people can access your builds and images. (Similar to the NPM registry?)
- It's recommended to tag your images (otherwise an auto generated ID will be allocated to it)

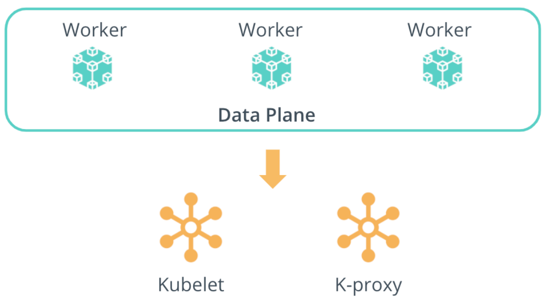
Kubernetes

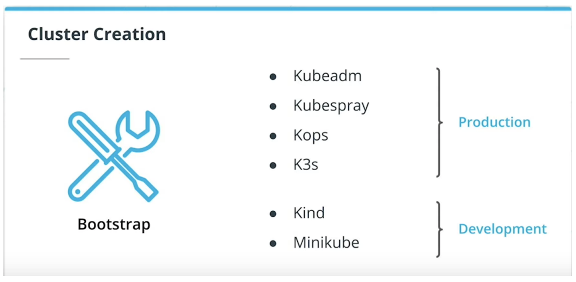
https://youtu.be/cC46cg5FFAM

- Open-source
- Portability
- Platform agnostic
- Scalability (HPA – Horizontal Portable Auto-scaler)
- Elasticity
- Handles failures
- Cluster-level DNS (Domain Name System) which makes it easier to identify and modify the clusters
- Distributes traffic
- CRD (Custom Resource Definitions) for extensibility
- Operational costs
- Cluster autoscaler





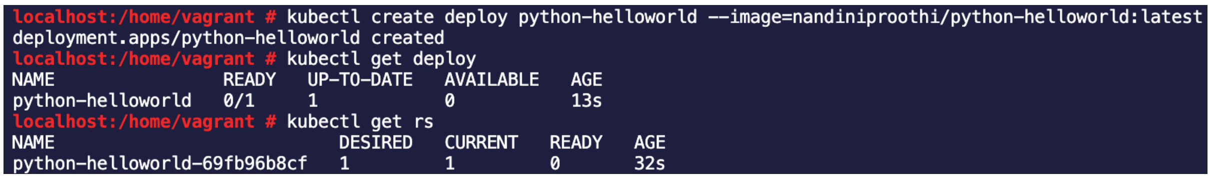




Understanding Kubernetes Resources

https://whimsical.com/kubernetes-resources-5nAUygYA97TfJ71aUKs6Tg

- To login as the root user in Vagrant `sudo su`
- Get more information about nodes `kubectl get nodes -o wide`



```
localhost:/home/vagrant # kubectl create deploy python-helloworld --image=nandiniproothi/python-helloworld:latest
deployment.apps/python-helloworld created
localhost:/home/vagrant # kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
python-helloworld   0/1     1            0           13s
localhost:/home/vagrant # kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
python-helloworld-69fb96b8cf   1         1         0       32s
```

`kubectl get po` gives error `CreateContainerError`

`kubectl describe pod <podname>` gives error `Error: failed to create containerd container: get apparmor_parser version: exec: "apparmor_parser": executable file not found in $PATH`

I fixed this issue by installing the missing package `zypper install -t pattern apparmor`

→ this uses SUSE's CLI `zypper` which can install, update, remove, and manage repositories `apparmor_parser`

→ loads `apparmor` (a linux security module) profiles into the linux kernel

https://youtu.be/UIVtDHaCSZ8

Useful kubectl Commands

Create Resources

To create resources, use the following command:

```
kubectl create RESOURCE NAME [FLAGS]
```

Describe Resources

To describe resources, use the following command:

```
kubectl describe RESOURCE NAME
```

Get Resources

To get resources, use the following command, where `-o yaml` instructs that the result should be YAML formated.

```
kubectl get RESOURCE NAME [-o yaml]
```

Edit Resources

To edit resources, use the following command, where `-o yaml` instructs that the edit should be YAML formated.

```
kubectl edit RESOURCE NAME [-o yaml]
```

Label Resources

To label resources, use the following command:

```
kubectl label RESOURCE NAME [PARAMS]
```

Port-forward to Resources

To access resources through port-forward, use the following command:

```
kubectl port-forward RESOURCE/NAME [PARAMS]
```

Logs from Resources

To access logs from a resource, use the following command:

```
kubectl logs RESOURCE/NAME [FLAGS]
```

Delete Resources

To delete resources, use the following command:

```
kubectl delete RESOURCE NAME
```

Kubernetes Manifests

Kubernetes is widely known for its support for imperative and declarative management techniques. The **imperative** approach enables the management of resources using kubectl commands directly on the live cluster. For example, all the commands you have practiced so far (e.g. kubectl create deploy [...]) are using the imperative approach. This technique is best suited for development stages only, as it presents a low entry-level bar to interact with the cluster.

On the other side, the **declarative** approach uses manifests stored locally to create and manage Kubertenest objects. This approach is recommended for production releases, as we can version control the state of the deployed resources. However, this technique presents a high learning curve, as an in-depth understanding of the YAML manifest structure is required. Additionally, using YAML manifests unlocks the possibility of configuring more advanced options, such as volume mounts, readiness and liveness probes, etc.

Edge Case: Failing Control for K8s

- ReplicaSets - to ensure that the desired amount of replicas is up and running at all times
- Liveness probes - to check if the pod is running, and restart it if it is in an errored state
- Readiness probes - ensure that traffic is routed to that pods that are ready to handle requests
- Services - to provide one entry point to all the available pods of an application