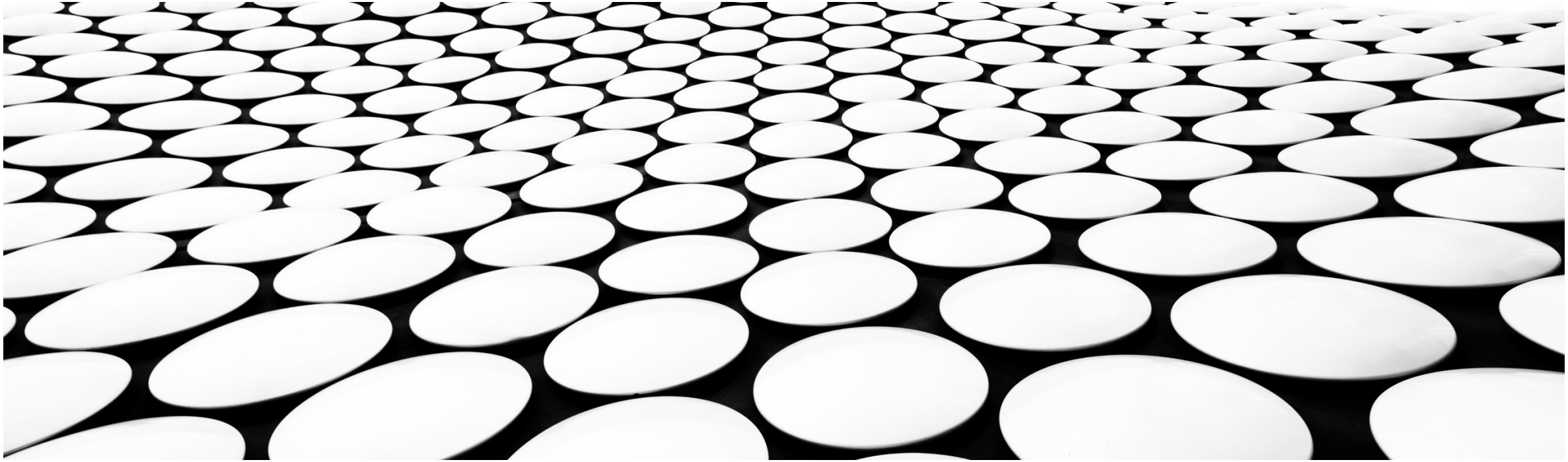
DOCKER – AN INTRODUCTION

SHAHBAZ CHAUDHARY



DOCKER KEEPS LAPTOP TO PROD ENVIRONMENTS IDENTICAL

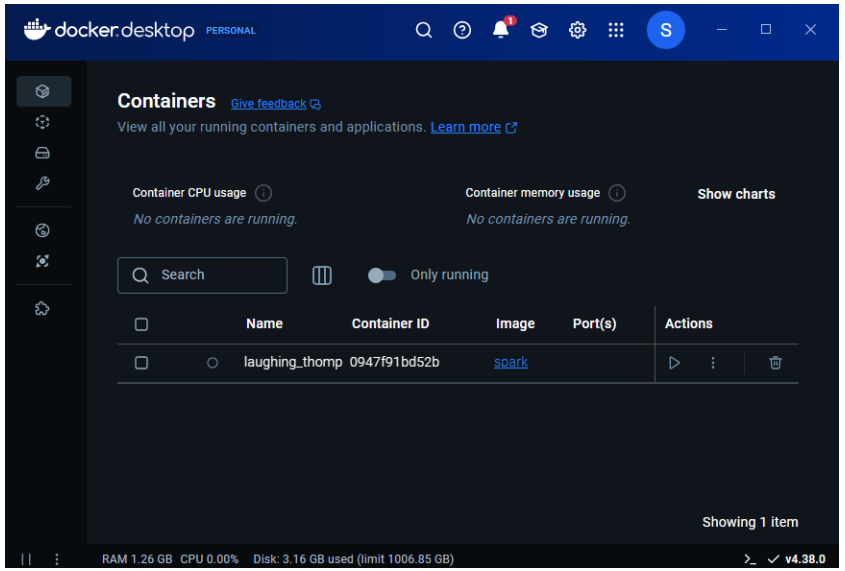
- Bad environments can cause production crashes, which are very difficult to catch earlier
- Python has `conda, virtualenv or uv`, but `docker` is a more general solution to freezing environments
- Virtual machines simulate the whole machine (including hardware), docker simulates running a process on a linux machine – hence MUCH lighter

😵 SORRY, IT WORKED ON MY LAPTOP
😡 THEN WE'LL SHIP YOUR LAPTOP TO THE CLIENT

SINCE WE HAVE A COUPLE OF MINUTES
PLEASE DOWNLOAD, INSTALL AND TEST SPARK (THE BIG DATA ENV)

- Open Docker Desktop (and keep an eye on it) and the terminal app
- Run `docker images` and `docker ps –all` to make sure nothing is running
- Run `docker pull spark` and `docker run -it spark /opt/spark/bin/spark-shell` to run spark
  - actually, you can skip `docker pull`
- Run `docker images` and `docker ps –all` to see what you downloaded and ran
- How does this compare to:

  `https://phoenixnap.com/kb/install-spark-on-ubuntu`



TAKE A STEP BACK

IMAGES ARE THE PACKAGE; CONTAINERS ARE WHAT IS RUNNING

Docker provides an interface for two main objects: images and containers

Images are the files, executables, configurations that we build, upload, download or deploy

search | pull | *list* | build | remove

Containers are the actual running instances of those images, the programs, the servers and the processes

run | exec | ps

IMAGES

Search Docker for an image	docker search <image name>
Download an image to local machine	docker pull <image name>
List downloaded images	docker images
Remove downloaded image	docker rmi <img>

CONTAINERS

Run an image	`docker run --name <container_name> <image_name>`
Run, but map hosts	`docker run -p <host_port>:<container_port> <image_name>`
	`docker run –d –restart always|unless-stopped|.. …`
Start or stop a container	`docker start|stop <container name|id>`
Which containers are running (or have run)	`docker ps --all`
“ssh” into the container	`docker exec -it <container_name> sh`

LECTURES/DOCKERIZE_PYTHON_APP/100_MINIMAL_27

BUILD YOUR OWN DOCKER IMAGE (FOR DEPLOYMENT)

app.py

```
print "Hi from many years ago :)"
```

Dockerfile

```
# Use a small base image .... dangerously old!
FROM python:2.7-slim

# Set the working directory
WORKDIR /app

# Copy the application code
COPY app.py .

# Run the application
CMD ["python", "app.py"]
```

Commands to run

```
docker build -t hello_app .
docker run hello_app
```

DOCKER CREATES “LAYERS” AND CACHES THEM

DOCKERFILE

Base image, starting point

“Current” directory, no need to ‘cd’

Copy code from local machine to docker

Run this command when docker “runs”

```
# Use a small base image .... dangerously old!
FROM python:2.7-slim

# Set the working directory
WORKDIR /app

# Copy the application code
COPY app.py .

# Run the application
CMD ["python", "app.py"]
```

LECTURES/120_DOCKERIZE_PYTHON_APP/110_MINIMAL_SERVER

BUILD YOUR OWN DOCKER IMAGE (FOR DEPLOYMENT)

app.py

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message":
   "Hello, FastAPI in Docker!"}
```

Dockerfile

```dockerfile
# Use the official Python image as a base
FROM python:3.11-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code
COPY . .

# Expose the FastAPI default port
EXPOSE 8000

# Command to run the application
CMD ["fastapi", "run", "app.py", "--host", "0.0.0.0", "--port", "8000"]
```

requirements.txt

```
fastapi[standard]
```

Commands to run

```
docker build -t minimal_server .
docker run -d -p 8000:8000 minimal_server
curl http://localhost:8000
```

USE “DOCKER RUN” AND ITS MANY VARIATIONS

DOCKER *EXEC* TO CONNECT TO A RUNNING CONTAINER

RUN A CONTAINER, EXIT WHEN THE COMMAND EXITS

```
docker run hello_app
```

START A CONTAINER, CONNECT TO IT INTERACTIVELY

Connect a fake ”console” to the container

```
docker run -it hello_app bash
```

Run container *interactively*, accept keyboard input, etc.

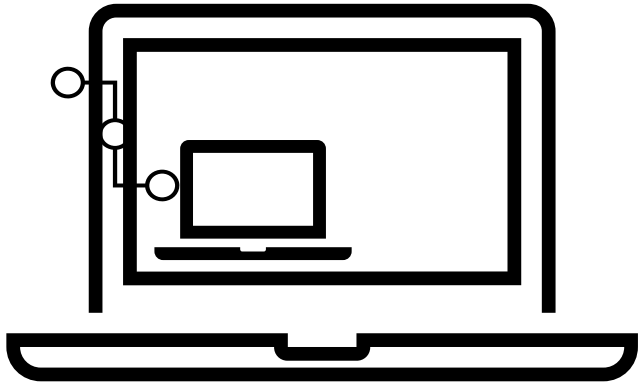
Run this command inside the container.

As if you *ssh* into a machine

If the container is already running

```
docker exec -it hello_app bash
```

START A CONTAINER, RUN A SERVICE INSIDE IT



Map port inside container to a port on the host machine

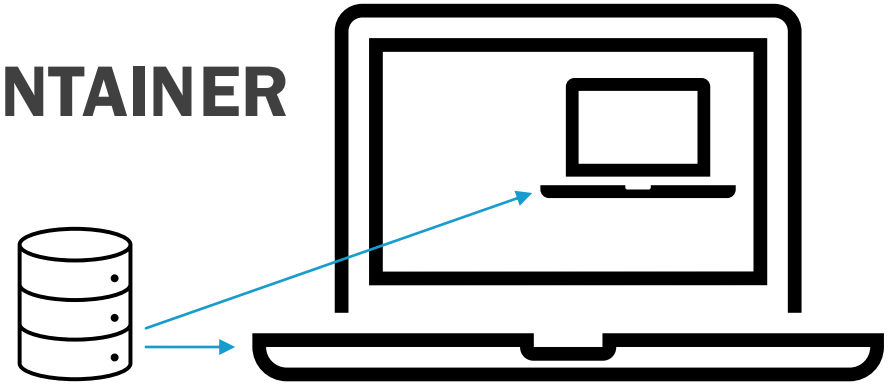
```
docker run -d -p 8000:8000 minimal_server
```

Run container in “detached” mode, so you don’t have to keep the terminal window open

Run this command inside the container.

As if you *ssh* into a machine

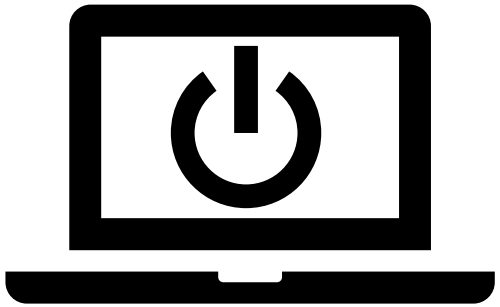
SHARE A FOLDER BETWEEN HOST AND CONTAINER



```
docker run -d -v /host/path:/container/path hello_app
```

The volume flag shares a folder
between the host and the container

START DOCKER, EVEN ON REBOOT



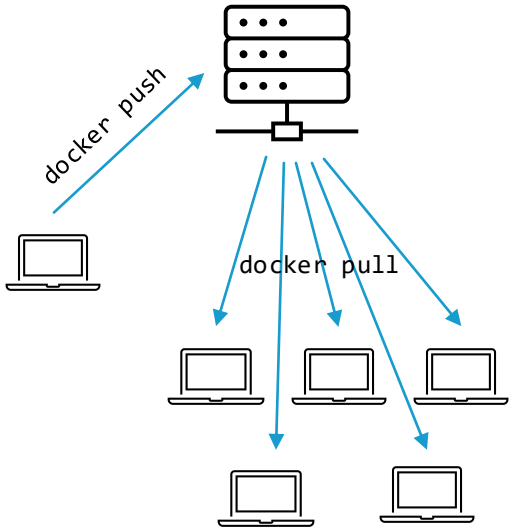
```
docker run -d -restart always hello_app
```

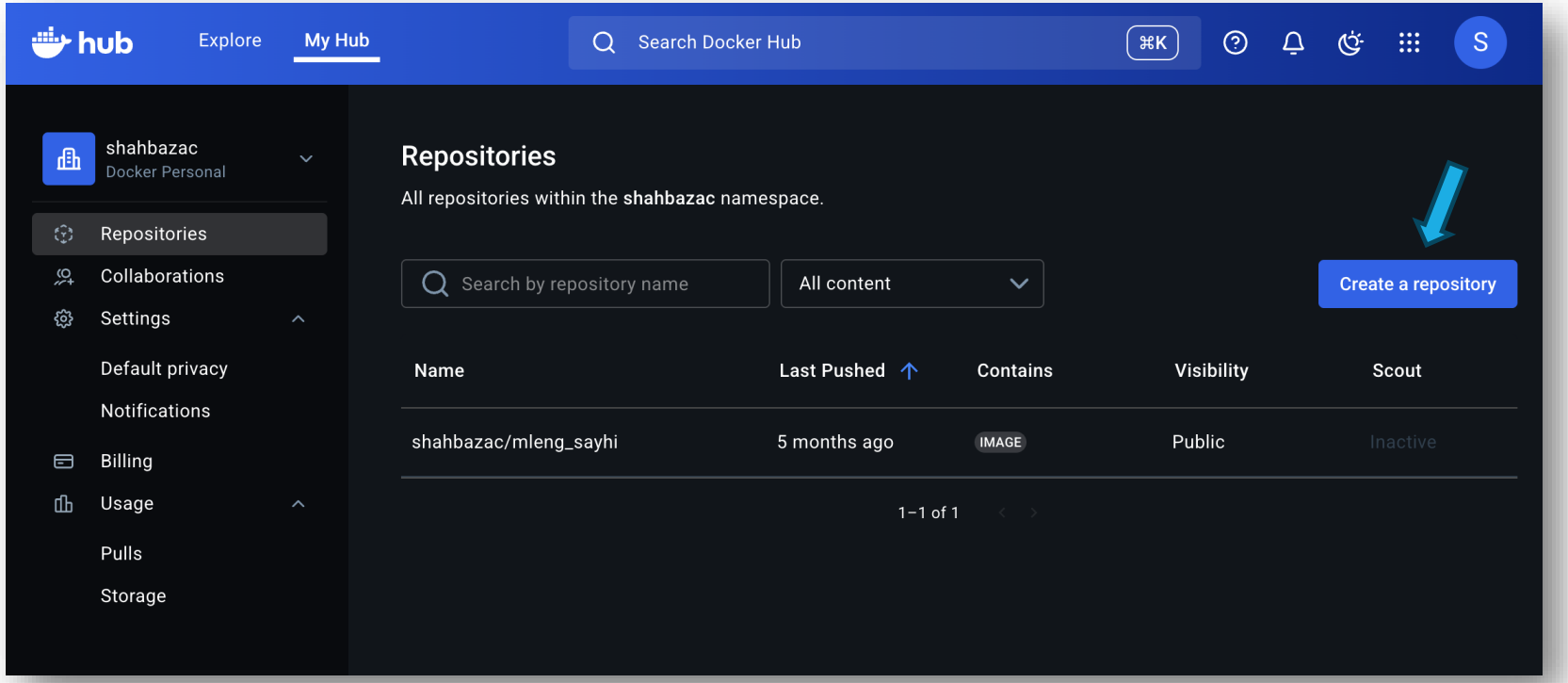
Even if the computer shuts down, when it starts back up, bring up docker container

“PUSH” YOUR IMAGE TO DOCKER HUB

AND “PULL” AS WELL

HUB.DOCKER.COM IS ONE OF MANY PUBLIC DOCKER REPOS





FIRST, CREATE A REPOSITORY

Notice the user name

hub | Explore | My Hub | Search Docker Hub ⌘K | S

shahbazac
Docker Personal

- Repositories
- Collaborations
- Settings
  - Default privacy
  - Notifications
- Billing
- Usage
  - Pulls
  - Storage

Repositories / Create

Using 0 of 1 private repositories. Get more

Create repository

Repository Name *
hello_app

Short description
Test docker image

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. Get more

Public
Appears in Docker Hub search results

Private
Only visible to you

Cancel | Create

Pushing images

You can push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

TAG YOUR IMAGE

```
[→ docker image list
REPOSITORY        TAG         IMAGE ID       CREATED       SIZE
minimal_server    latest      370d24a95939   2 days ago    324MB
hello_app         latest      48acd56fa5bc   2 days ago    205MB
rayproject/ray    latest      e9950dadc62e   5 weeks ago   3.06GB
python            3.11-slim   139020233cc4   7 weeks ago   221MB
spark             latest      23553639f445   8 weeks ago   2.1GB
```

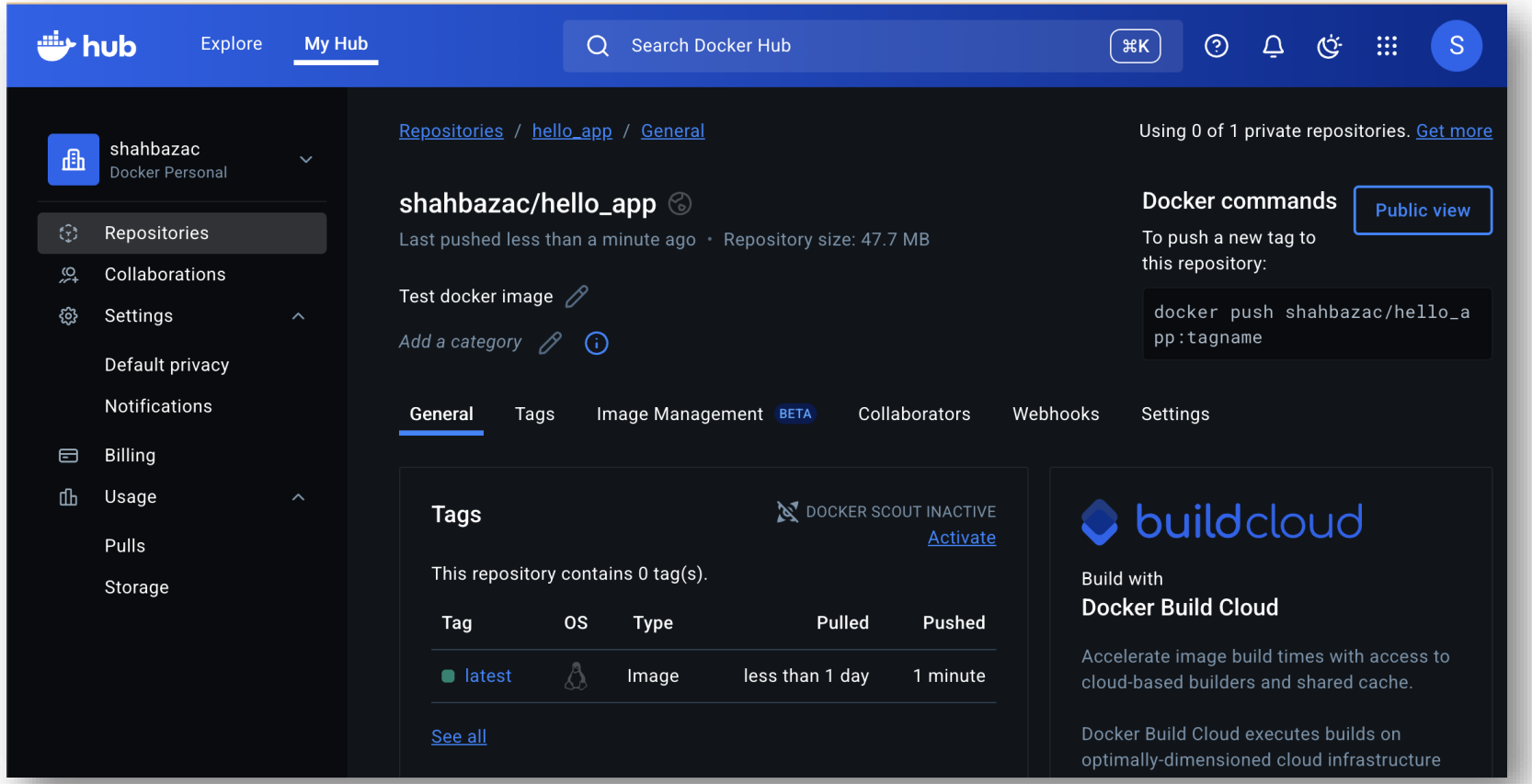
```
[→ docker tag hello_app shahbazac/hello_app:latest
```

Notice the user name

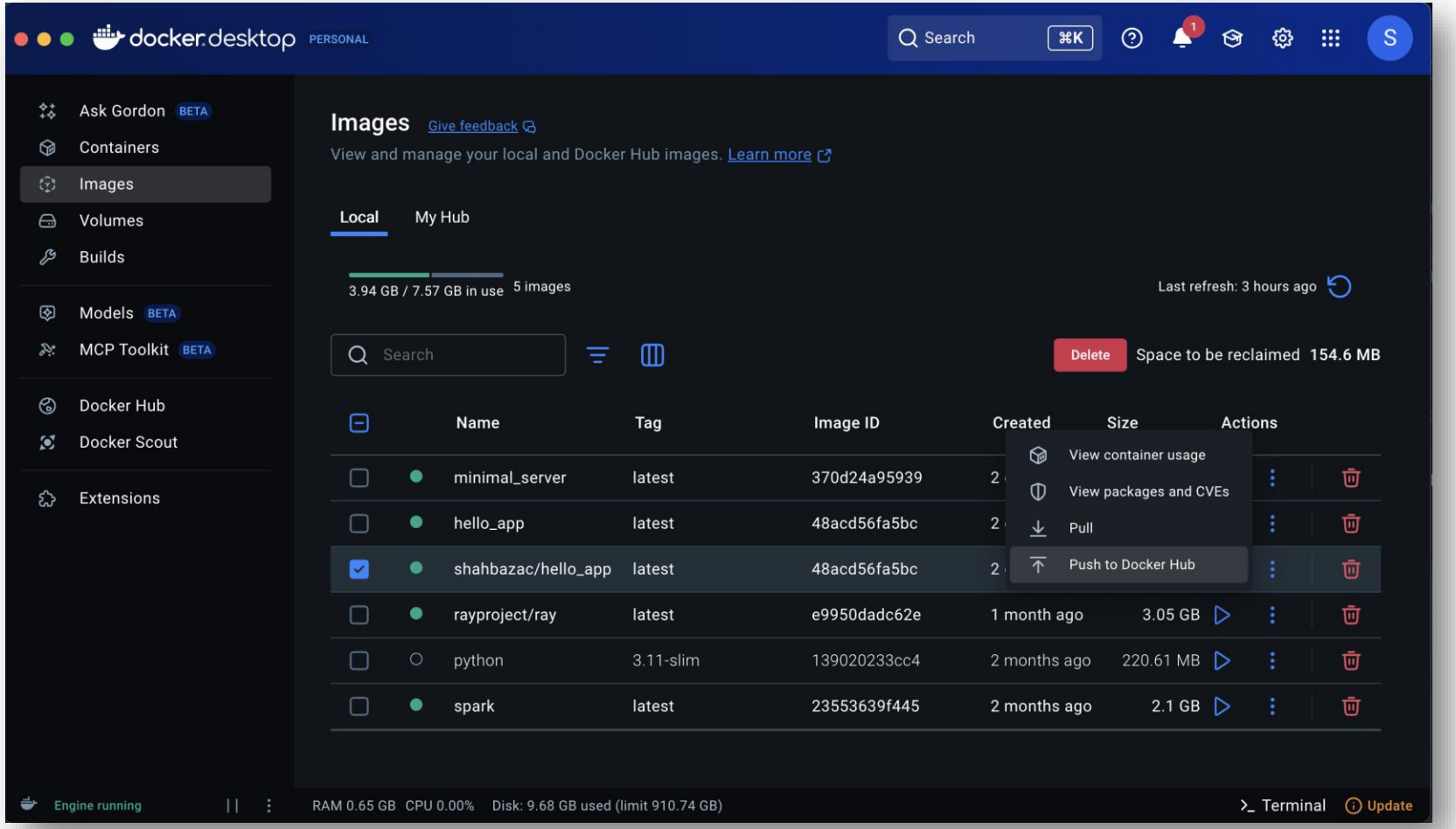
```
[→ docker image list
REPOSITORY            TAG         IMAGE ID       CREATED       SIZE
minimal_server        latest      370d24a95939   2 days ago    324MB
hello_app             latest      48acd56fa5bc   2 days ago    205MB
shahbazac/hello_app   latest      48acd56fa5bc   2 days ago    205MB
rayproject/ray        latest      e9950dadc62e   5 weeks ago   3.06GB
python                3.11-slim   139020233cc4   7 weeks ago   221MB
spark                 latest      23553639f445   8 weeks ago   2.1GB
```

```
[21:01:04] (base) ~
[→ docker push shahbazac/hello_app:latest
The push refers to repository [docker.io/shahbazac/hello_app]
281ab480dfcb: Pushed
776e16a27dc9: Pushed
3db6e1b8fb28: Pushed
421e7d14367e: Pushed
2f3301b95e67: Pushed
fbf3a209535f: Pushed
3d48095d71a3: Pushed
latest: digest: sha256:48acd56fa5bc3ef37bf0f67cd9d28d21f97e1d26bc238af4130996033467d820 size: 856
```

YOUR IMAGE IS NOW AVAILABLE FOR OTHERS TO “PULL”

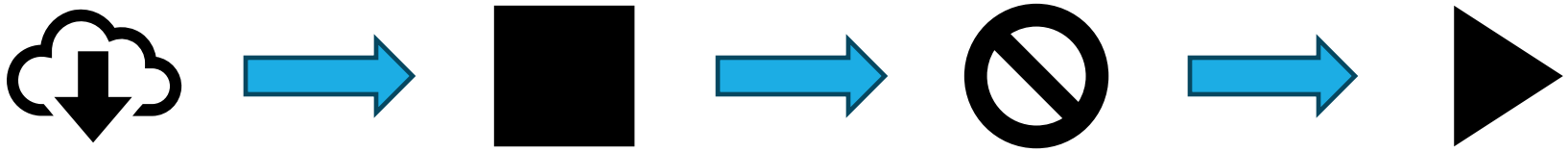


YOU CAN ALSO USE DOCKER DESKTOP TO PUSH



“DEPLOY” DOCKER TO A REMOTE SERVER

“DEPLOY” DOCKER CONTAINER TO A REMOTE MACHINE VIA SSH



```
docker pull image_name:latest
docker stop image_name
docker rm image_name
docker run -d -restart always -name image_name
```

Either *ssh* into the remote machine and run these commands or execute them as a remote script, executed locally:

```
ssh user@remote_host << ‘EOF’
docker pull image_name:latest
docker stop image_name
docker rm image_name
docker run -d -restart always -name image_name
EOF
```

BUILD

~~DEPLOY~~ DOCKER ~~TO A REMOTE MACHINE VIA GITHUB ACTIONS~~

See this for an up-to-date template

```yaml
name: Build and Push Docker Image
# "best practice", from https://docs.docker.com/guides/gha/
on:
push:
branches:
- prod

jobs:
build:
runs-on: ubuntu-latest
steps:
- name: Get Git commit timestamps
run: echo "TIMESTAMP=$(date +%Y%m%d%H%M%S)" >> $GITHUB_ENV

- name: Checkout
uses: actions/checkout@v4

- name: Extract Docker image metadata
id: meta
uses: docker/metadata-action@v5
with:
images: ${{ vars.DOCKER_USERNAME }}/mleng_sayhi
```

```yaml
- name: Log in to Docker Hub
uses: docker/login-action@v3
with:
username: ${{ vars.DOCKER_USERNAME }}
password: ${{ vars.DOCKER_PASSWORD }}

- name: Set up Docker Buildx
uses: docker/setup-buildx-action@v3
- name: Build and push Docker image
uses: docker/build-push-action@v6
with:
push: ${{ github.event_name != 'pull_request' }}
tags: |
${{ vars.DOCKER_USERNAME }}/mleng_sayhi:${{ env.TIMESTAMP }}
${{ vars.DOCKER_USERNAME }}/mleng_sayhi:latest
annotations: ${{ steps.meta.outputs.annotations }}
provenance: true
sbom: true
```

Find these yamls on the actual GitHub project at:
https://github.com/falconair/mleng_sayhi/tree/main/.github/workflows

BUILD

~~DEPLOY~~ DOCKER ~~TO A REMOTE MACHINE VIA GITHUB ACTIONS~~

See this for an up-to-date template

```yaml
name: Build and Push Docker Image
# "best practice", from https://docs.docker.com/guides/gha/
on:
push:
branches:
- prod
```

When code is pushed to the prod branch, execute the following steps

```yaml
jobs:
build:
runs-on: ubuntu-latest
steps:
```

...in an Ubuntu virtual machine

```yaml
- name: Get Git commit timestamps
run: echo "TIMESTAMP=$(date +%Y%m%d%H%M%S)" >> $GITHUB_ENV
```

Get the current date and time

```yaml
- name: Checkout
uses: actions/checkout@v4
```

Check out the github repository

```yaml
- name: Extract Docker image metadata
id: meta
uses: docker/metadata-action@v5
with:
images: ${{ vars.DOCKER_USERNAME }}/mleng_sayhi
```

"GitHub Action to extract metadata (tags, labels) from Git reference and GitHub events for Docker" (??) from Docker

```yaml
- name: Log in to Docker Hub
uses: docker/login-action@v3
with:
username: ${{ vars.DOCKER_USERNAME }}
password: ${{ vars.DOCKER_PASSWORD }}
```

Log in to docker hub

```yaml
- name: Set up Docker Buildx
uses: docker/setup-buildx-action@v3
- name: Build and push Docker image
uses: docker/build-push-action@v6
with:
push: ${{ github.event_name != 'pull_request' }}
tags: |
${{ vars.DOCKER_USERNAME }}/mleng_sayhi:${{ env.TIMESTAMP }}
${{ vars.DOCKER_USERNAME }}/mleng_sayhi:latest
annotations: ${{ steps.meta.outputs.annotations }}
provenance: true
sbom: true
```

Build docker image and push to docker hub

DEPLOY DOCKER TO A REMOTE MACHINE VIA GITHUB ACTIONS

```yaml
name: Deploy Docker Image to Server
on:
workflow_run:
# This GitHub Action depends on another, which package up docker and pushes it to a docker repo
workflows: ["Build and Push Docker Image"]
types:
- completed

jobs:
deploy:
runs-on: ubuntu-latest
# If the docker build went ok and the image was pushed to the docker repo, then deploy it to the server
if: ${{ github.event.workflow_run.conclusion == 'success' }}
steps:
- name: Deploy to Linux server
uses: appleboy/ssh-action@v1.0.3
with:
host: ${{ secrets.SERVER_HOST }}
username: ${{ secrets.SERVER_USER }}
key: ${{ secrets.SERVER_SSH_KEY }}
script: |
docker pull ${{ vars.DOCKER_USERNAME }}/mleng_sayhi:latest
docker stop ${{ vars.DOCKER_USERNAME }}/mleng_sayhi || true
docker rm ${{ vars.DOCKER_USERNAME }}/mleng_sayhi || true
docker run -d --name ${{ vars.DOCKER_USERNAME }}/mleng_sayhi
                -v /opt/assignment_outputs:/app/data ${{ vars.DOCKER_USERNAME }}/mleng_sayhi:latest
```

ssh into the remote server,
update the docker image,
stop the previous one,
remove the previous one
and run the latest

DEPLOY DOCKER TO A REMOTE MACHINE VIA GITHUB ACTIONS

USING AND ABUSING DOCKER

- Docker is based on a set of linux technologies which allow the operating system to isolate processes and make them believe they are the only ones running.
  - Compare this with normal linux accounts where you can see that there are other user on `/home` and other processes running via `ps -ef`
  - **When we run Docker on Windows or Mac, linux is being virtualized first, then docker is being run inside it!**

- Docker environment **should be set up entirely via Dockerfile**. Do NOT execute commands on the shell to update the environment. That defeats the purpose of docker. I have seen this in online tutorials
  - But you can enter the shell, as if you were ssh into a remote machine: `docker exec -it <container_name> sh`

ON TO KUBERNETES

RUN DOCKER ACROSS A CLUSTER

SEPARATE APP'S BUSINESS LOGIC FROM SCALING CONCERNS

- Provide a /status route which shows status and version
- Provide a /score route which returns a label: optimistic/pessimistic/neutral
- /score should accept a list of news headlines
- As client volume increases, app must scale to handle higher traffic
- If an empty list is provided as an argument to /score, an empty list must be returned as the output
- As volume decreases, fewer or cheaper resources should be used
- App must keep logs for 30 days
- Operations team must be provided with resource monitors
- If the app crashes, it must be restarted

SEPARATE APP'S BUSINESS LOGIC FROM SCALING CONCERNS

- Provide a /status route which shows status and version
- Provide a /score route which returns a label: optimistic/pessimistic/neutral
- /score should accept a list of news headlines
- As client volume increases, app must scale to handle higher traffic
- If an empty list is provided as an argument to /score, an empty list must be returned as the output
- As volume decreases, fewer or cheaper resources should be used
- App must keep logs for 30 days
- Operations team must be provided with resource monitors
- If the app crashes, it must be restarted

Business logic

- Provide a /status route which shows status and version
- Provide a /score route which returns a label: optimistic/pessimistic/neutral
- /score should accept a list of news headlines
- As client volume increases, app must scale to handle higher traffic
- If an empty list is provided as an argument to /score, an empty list must be returned as the output
- As volume decreases, fewer or cheaper resources should be used
- App must keep logs for 30 days
- Operations team must be provided with resource monitors
- If the app crashes, it must be restarted

Cluster logic

- Provide a /status route which shows status and version
- Provide a /score route which returns a label: optimistic/pessimistic/neutral
- /score should accept a list of news headlines
- As client volume increases, app must scale to handle higher traffic
- If an empty list is provided as an argument to /score, an empty list must be returned as the output
- As volume decreases, fewer or cheaper resources should be used
- App must keep logs for 30 days
- Operations team must be provided with resource monitors
- If the app crashes, it must be restarted

Data retention logic…

THINK BEYOND INDIVIDUAL MACHINES

- Acquire machines (physical or virtual), name them, create user accounts, …
- Install operating systems on them (manually or using terraform)
- Install necessary software dependencies (python, fastpi, etc.) and keep it up to date (manually or via ansible/chef/puppet)
- Set up networking so they can all communicate with each other
- Run ML code on machines with GPUs
- Designate one machine as the load balancer

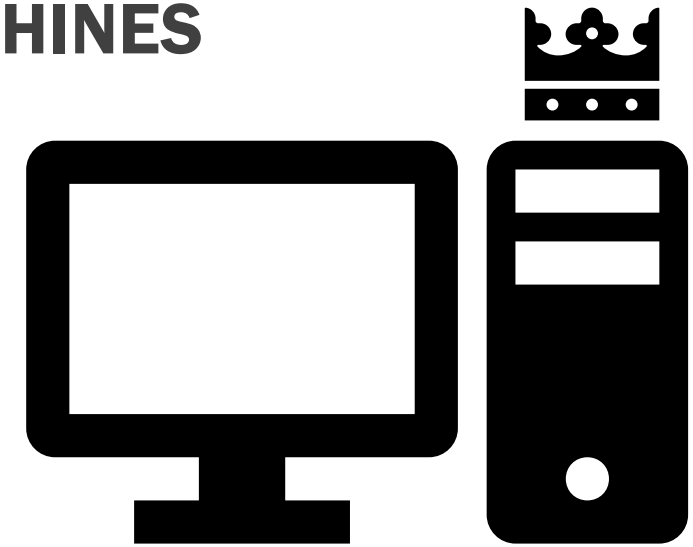
…

THE CLUSTER IS THE MACHINE

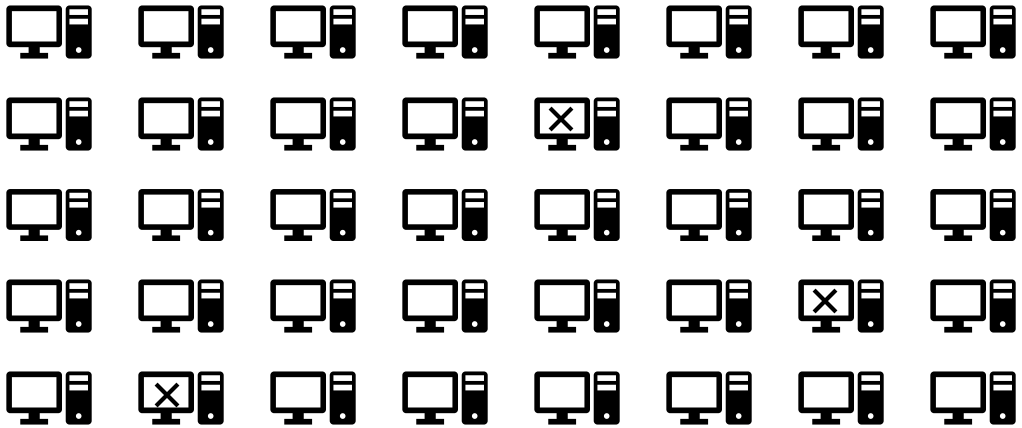
- Set up a cluster (simulated, on-prem or on the cloud)
- Package your code in a docker image
- Deploy that image to a Kubernetes cluster, while adding constraints (must have a GPU, min X amount of ram and Y mount of disk)
- Now manage the whole cluster

...

HISTORICAL CONTEXT: BEOWULF, BORG AND LOTS OF SMALL MACHINES



Until the early 2000s, “serious” companies bought powerful, “million dollar” machines. These machines had very expensive support contracts.



However, there was a sub-culture of using lots of commodity machines. Individual machines were expected to break and cluster software accounted for it.

In the 90s, ”Beowulf” cluster software allowed even hobbyists to access large amount of compute.
Google popularized this technique in the industry. They created “Borg” which became Kubernetes.

“INSTALL” KUBERNETES (K8S) ON YOUR LAPTOP

1 2 3 4 5 6 7 8

Docker desktop will “simulate” a cluster on your laptop. The API to manage this cluster will be very similar to actual clusters on the cloud.

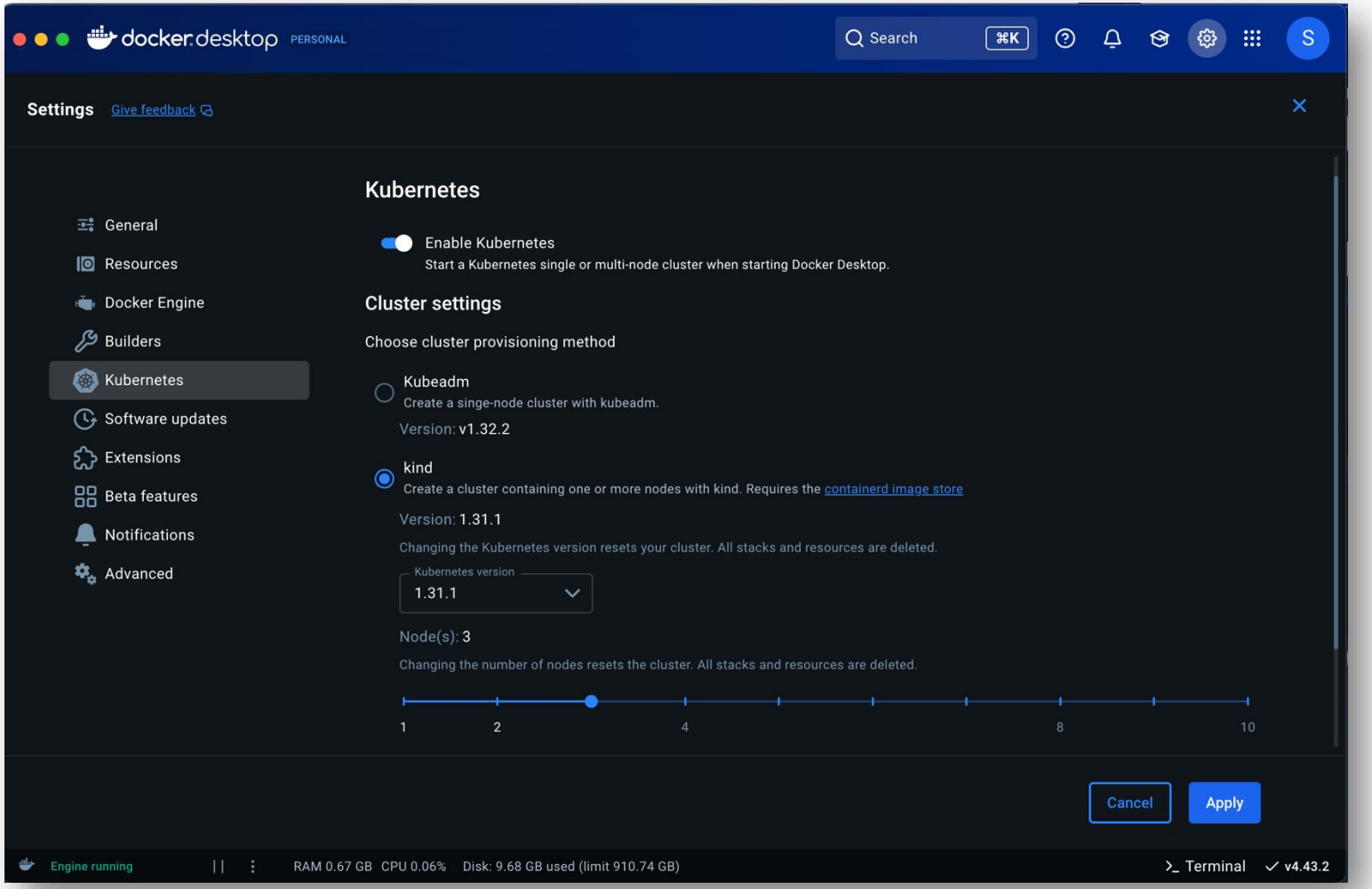
You will use the `kubectl` command to work with your cluster on your laptop (and on the cloud)

Amazon Elastic Kubernetes Service (EKS): https://aws.amazon.com/eks/

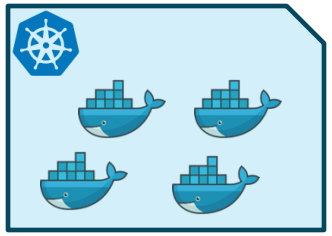
Google Kubernetes Engine (GKE): https://cloud.google.com/kubernetes-engine

Azure Kubernetes Service (AKE): https://azure.microsoft.com/en-us/products/kubernetes-service

Digital Ocean Kubernetes (DOKS): https://www.digitalocean.com/products/kubernetes



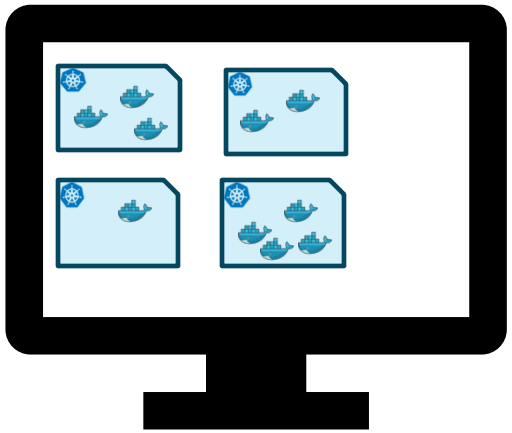
“PODS” ARE THE SMALLEST OBJECT IN KUBERNETES



A Kubernetes POD can contain any number of docker containers

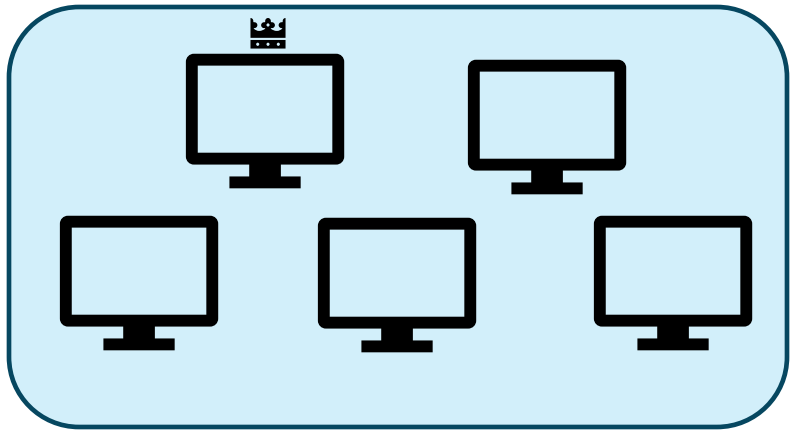
A pod gets its own IP and containers can access each other via “localhost”

This means that two containers can’t listen on the same port



A node can run any number of pods. Each pod will have its own IP.

Pods an also run services, which get their own IP which makes it easy to access them



A collection of nodes is a Kubernetes cluster.

Earlier, we simulated a cluster on our laptop. This cluster contains three nodes. One is a “control-plane”, a manager of sorts. The other two nodes are workers

LET'S DEPLOY "MINIMAL SERVER"

(LECTURES/120_DOCKERIZE_PYTHON_APP/110_MINIMAL_SERVER)

LECTURES/120_DOCKERIZE_PYTHON_APP/110_MINIMAL_SERVER

BUILD YOUR OWN DOCKER IMAGE (FOR DEPLOYMENT)

app.py

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message":
   "Hello, FastAPI in Docker!"}
```

Dockerfile

```dockerfile
# Use the official Python image as a base
FROM python:3.11-slim

# Set the working directory
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code
COPY . .

# Expose the FastAPI default port
EXPOSE 8000

# Command to run the application
CMD ["fastapi", "run", "app.py", "--host", "0.0.0.0", "--port", "8000"]
```

requirements.txt

```
fastapi[standard]
```

Commands to run

```
docker build -t minimal_server .
docker run -d -p 8000:8000 minimal_server
curl http://localhost:8000
```

Do you still have "minimal_server"?

```
docker image list
```

If not, let's build it

```
docker build -t minimal_server .
```

WRITE A DEPLOYMENT FILE
A “DEPLOYMENT” WILL MANAGE THE LIFECYCLE OF YOUR POD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: fastapi-app
 labels:                          # Labels to identify the Deployment
   app: fastapi-app
spec:                             # Specification for the Deployment
 replicas: 2                      # Number of replicas to run
 selector:                        # Selector to identify the pods managed by this Deployment
   matchLabels:
     app: fastapi-app
 template:                        # Template for the pods created by this Deployment
   metadata:
     labels:
       app: fastapi-app
   spec:                          # Specification for the pod
     containers:                  # List of containers in the pod
     - name: fastapi
       image: minimal_server:latest    # Name of the Docker image to use
       imagePullPolicy: IfNotPresent   # Pull the image only if not present locally
       ports:
       - containerPort: 8000    # Port on which the FastAPI app will run
```

```
kubectl apply -f deployment.yaml
```

WRITE A SERVICE FILE
THIS WILL PROVIDE AN END-POINT FOR CLIENTS TO ACCESS

```yaml
apiVersion: v1
kind: Service
metadata:
 name: fastapi-service
spec:
 selector:
   app: fastapi-app
 ports:
   - protocol: TCP
     port: 8000 # Port exposed by the service
     targetPort: 8000 # Port on which the FastAPI app is running in the container
 type: LoadBalancer  # Other options: ClusterIP, NodePort, ExternalName
```

```
kubectl apply -f service.yaml
```

USE “KUBECTL” TO INTERACT WITH YOUR CLUSTER

```
→ kubectl get nodes
NAME                    STATUS   ROLES           AGE     VERSION
desktop-control-plane   Ready    control-plane   3d17h   v1.31.1
desktop-worker          Ready    <none>          3d17h   v1.31.1
desktop-worker2         Ready    <none>          3d17h   v1.31.1
```

```
→ kubectl get pods
NAME                          READY   STATUS    RESTARTS        AGE
fastapi-app-dcfb94cf6-2wzmk   1/1     Running   1 (2d15h ago)   3d14h
fastapi-app-dcfb94cf6-czbz4   1/1     Running   1 (2d15h ago)   3d14h
```

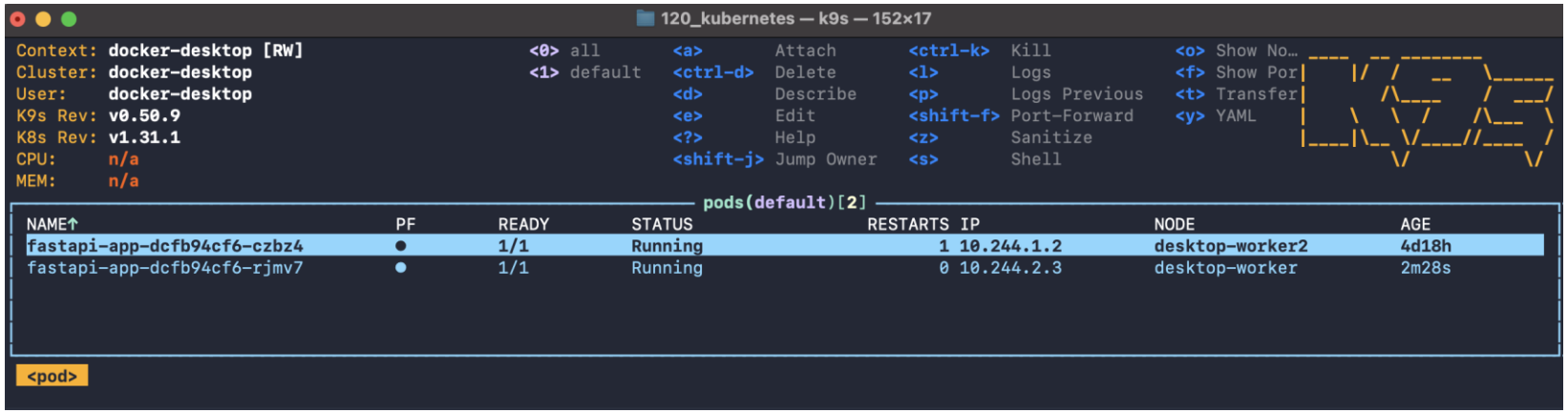
```
→ kubectl get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
fastapi-service   LoadBalancer   10.96.207.105   172.19.0.7    8000:30485/TCP   3d14h
kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP          3d17h
```

USEFUL TO INSTALL ADDITIONAL CLIENTS

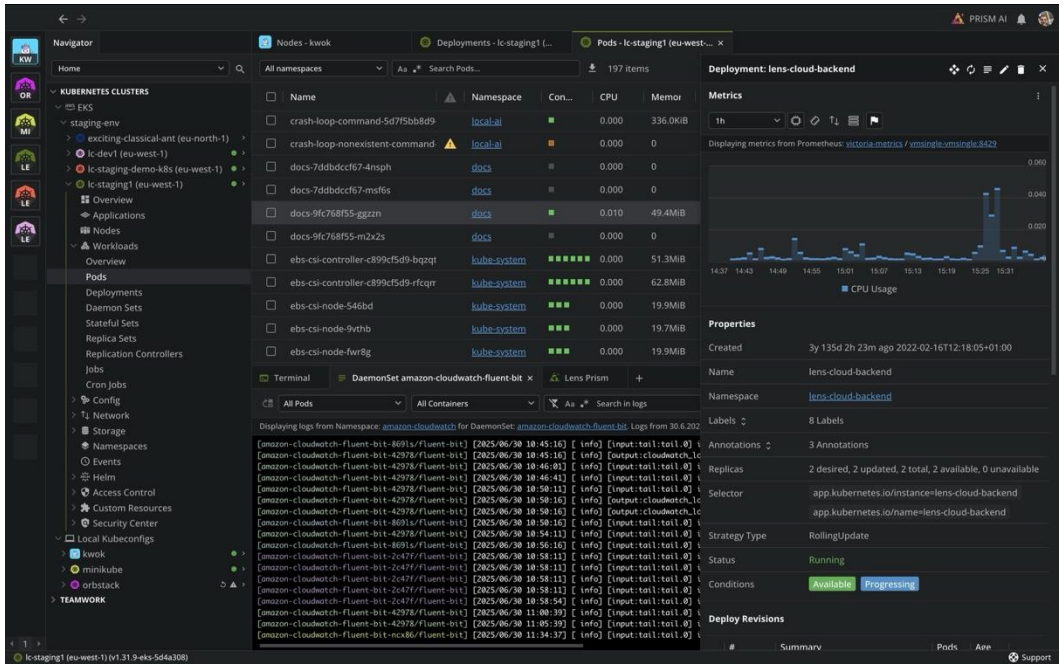
k9s: https://k9scli.io/

brew install derailed/k9s/k9s

choco install k9s



Lens: https://k8slens.dev/



WHAT CAN YOU DO WITH IT?

EASILY SCALE THE APP

```
kubectl scale deployment fastapi-app --replicas=5
```

2 pods

5 pods

```
→ kubectl get pods
NAME                          READY   STATUS    RESTARTS        AGE
fastapi-app-dcfb94cf6-czbz4   1/1     Running   1 (3d20h ago)   4d18h
fastapi-app-dcfb94cf6-rjmv7   1/1     Running   0               10m
[19:19:55] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
→ kubectl scale deployment fastapi-app --replicas=5

deployment.apps/fastapi-app scaled
[19:20:09] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
→ kubectl get pods
NAME                          READY   STATUS    RESTARTS        AGE
fastapi-app-dcfb94cf6-czbz4   1/1     Running   1 (3d20h ago)   4d18h
fastapi-app-dcfb94cf6-fdmmf   1/1     Running   0               6s
fastapi-app-dcfb94cf6-gv6dx   1/1     Running   0               6s
fastapi-app-dcfb94cf6-mh8sz   1/1     Running   0               6s
fastapi-app-dcfb94cf6-rjmv7   1/1     Running   0               11m
[19:20:19] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
```

WHAT CAN YOU DO WITH IT?

CLUSTER IS SELF HEALING

```
kubectl delete pod <pod name>
```

5 pods

5 pods

```
fastapi-app-dcfb94cf6-czbz4   1/1     Running   1 (3d20h ago)   4d18h
fastapi-app-dcfb94cf6-fdmmf   1/1     Running   0               6s
fastapi-app-dcfb94cf6-gv6dx   1/1     Running   0               6s
fastapi-app-dcfb94cf6-mh8sz   1/1     Running   0               6s
fastapi-app-dcfb94cf6-rjmv7   1/1     Running   0               11m
[19:20:19] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
→ kubectl delete pod fastapi-app-dcfb94cf6-czbz4
pod "fastapi-app-dcfb94cf6-czbz4" deleted
[19:24:05] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
→ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
fastapi-app-dcfb94cf6-fdmmf   1/1     Running   0          3m58s
fastapi-app-dcfb94cf6-gv6dx   1/1     Running   0          3m58s
fastapi-app-dcfb94cf6-mh8sz   1/1     Running   0          3m58s
fastapi-app-dcfb94cf6-p4mqm   1/1     Running   0          3s
fastapi-app-dcfb94cf6-rjmv7   1/1     Running   0          15m
[19:24:07] (base) [?master*] ~/Documents/GitHub/ProgrammingForAnalytics/lectures/120_dockerize_python_app/120_kubernetes
```

WHAT CAN YOU DO WITH IT?

ROLLING UPDATES (ZERO DOWNTIME)

```
kubectl set image ...
```

```
# Build new version
docker build -t shahbazac/minimal_server:v2 .
docker push shahbazac/minimal_server:v2

# Rolling update
kubectl set image deployment/fastapi-app fastapi=shahbazac/minimal_server:v2

# Watch the rolling update
kubectl rollout status deployment/fastapi-app
kubectl get pods -w
```