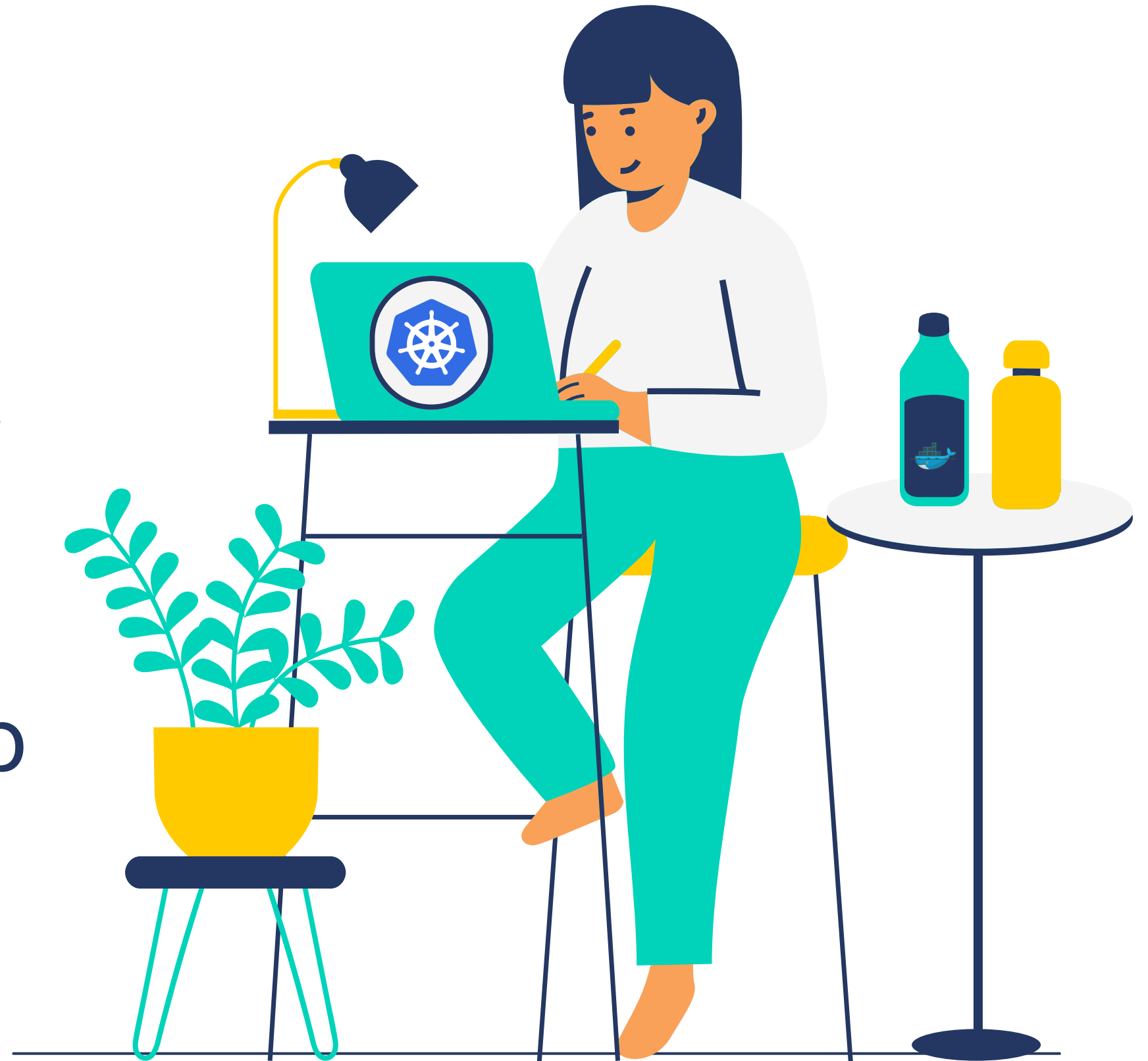


Containers & Kubernetes

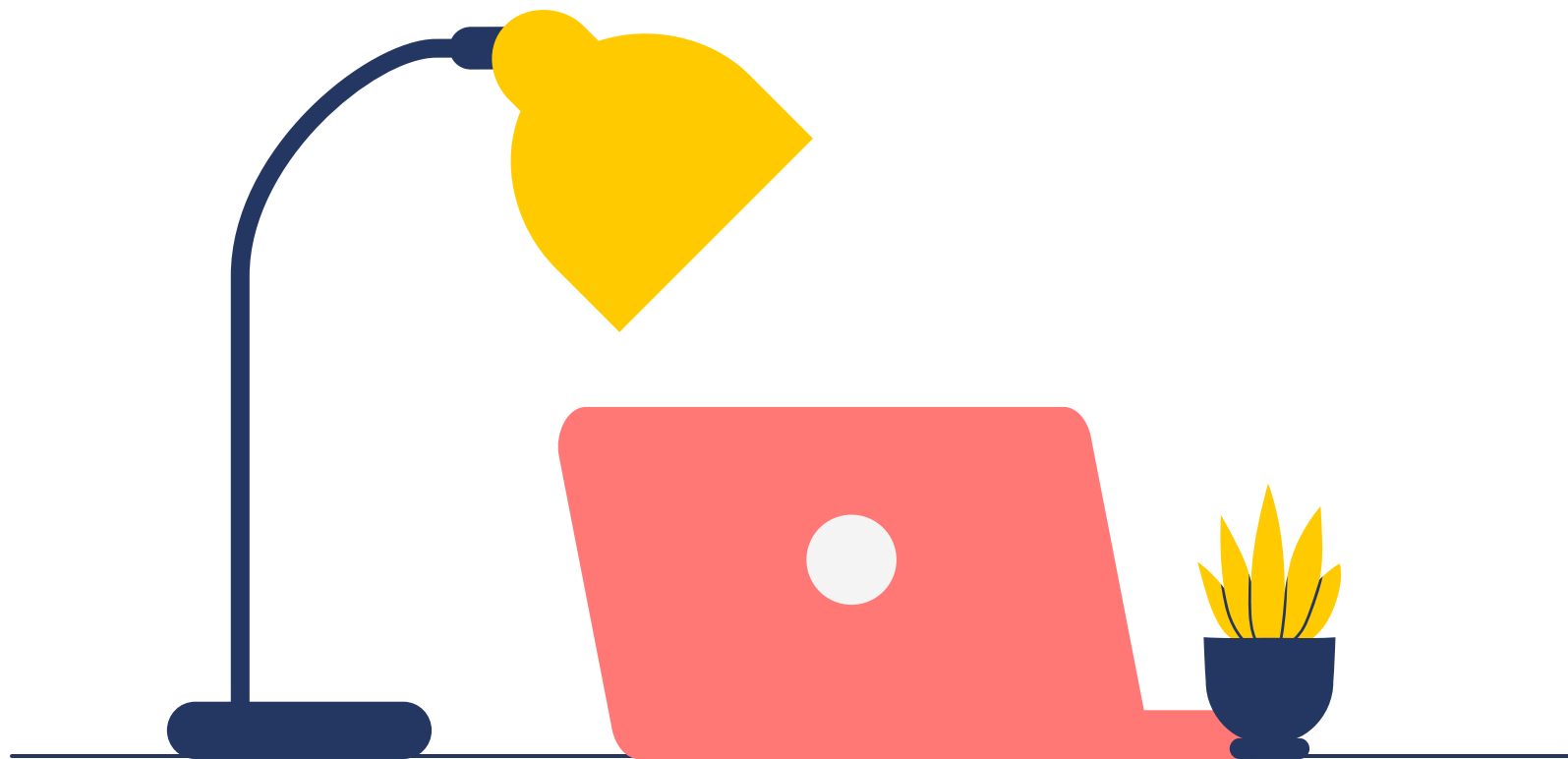
A Beginner's Workshop

Rigerta Demiri & Katharina Sick
#theNewITGirls | November 17, 2023



About today

Let's do a step-by-step journey into the world of containers and Kubernetes.



Containers & Docker

What are containers? Why are they cool?
How to create & use them?



Kubernetes

What is Kubernetes? Why do so many companies need it? How to get started?



Troubleshooting and Questions

Feel free to ask any time! :)



Hello, I'm Rigerta 🙋



Data Engineer @GitLab



Vienna Data Engineering Meetup



linkedin.com/in/rigerta/



ÖSTERREICHISCHES
ROTES KREUZ

Aus Liebe zum Men!



VIENNA DATA ENGINEERING
MEETUP

So, what is Docker?

An open-source tool used to turn your **application** into a **container** that can easily be deployed in any other system.





First things first, what is an Application?

Any piece of **Software** that performs a specific function either for an end user or for another application.

An **Application** is written in a specific **Programming language** and typically has one or more **Dependencies**.



Cool, so what is a Container?

A self-contained, runnable software application or service.

And how do we do this with Docker?

We write a **Dockerfile**, which is “a recipe” defining how to build a **Docker Image** and when we run the image so we can reach the application, we are in a **Container**



Key Concepts



- **Application**
 - Any piece of software you write, to fulfil a goal for end users or other applications
- **Dependencies**
 - All software libraries an application needs to be able to run successfully
- **Dockerfile**
 - A text file containing a set of instructions
- **Docker image**
 - A read-only blueprint that includes container-creation instructions
 - An **executable** application artifact
- **Docker container**
 - A running instance of a Docker **image** that gets created when the **\$ docker run** command is implemented
 - Multiple containers can run from the same Docker image
- **Docker registry**
 - A storage and distribution system for container images (can be **private** or **public**)
 - Docker Hub (hub.docker.com) is the **official public** Docker registry

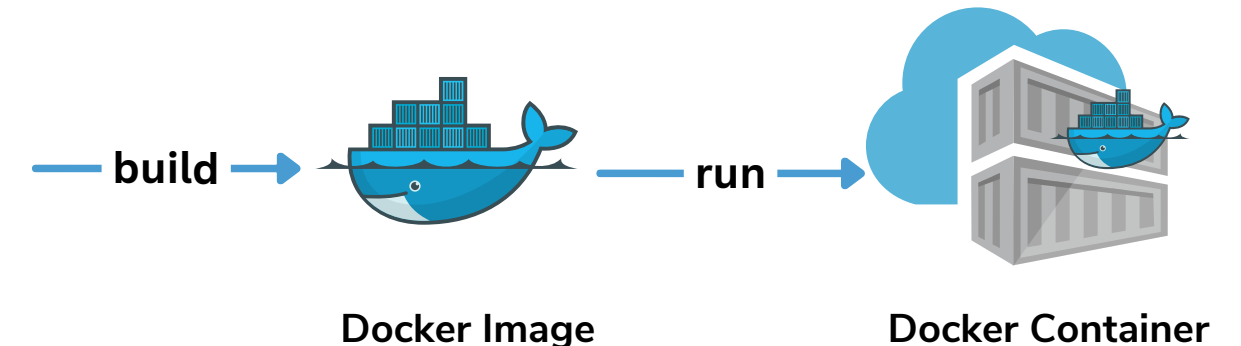
```
# Base Image: Stage 1
FROM ubuntu:16.04 as stage1

RUN apt-get update
RUN apt-get -y install make curl
RUN curl http://xyz.com/abc.tar.gz -O
RUN tar xzf abc.tar.gz && cd abc
RUN make DESTDIR=/tmp install

# Stage 2
FROM alpine:3.10

COPY --from=stage1 /tmp /abc
ENTRYPOINT ["/abc/app"]
```

Dockerfile



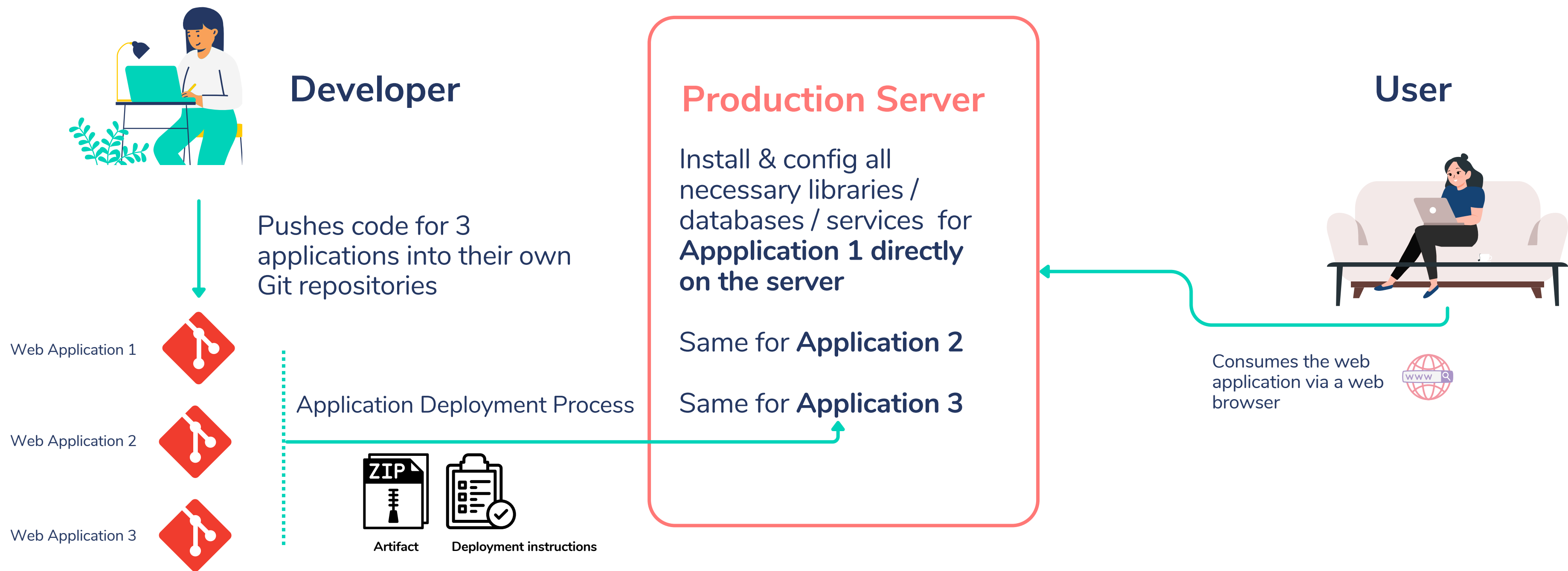


OK, let's take a step back

Why do we even **need** Docker?

How were applications deployed before it?

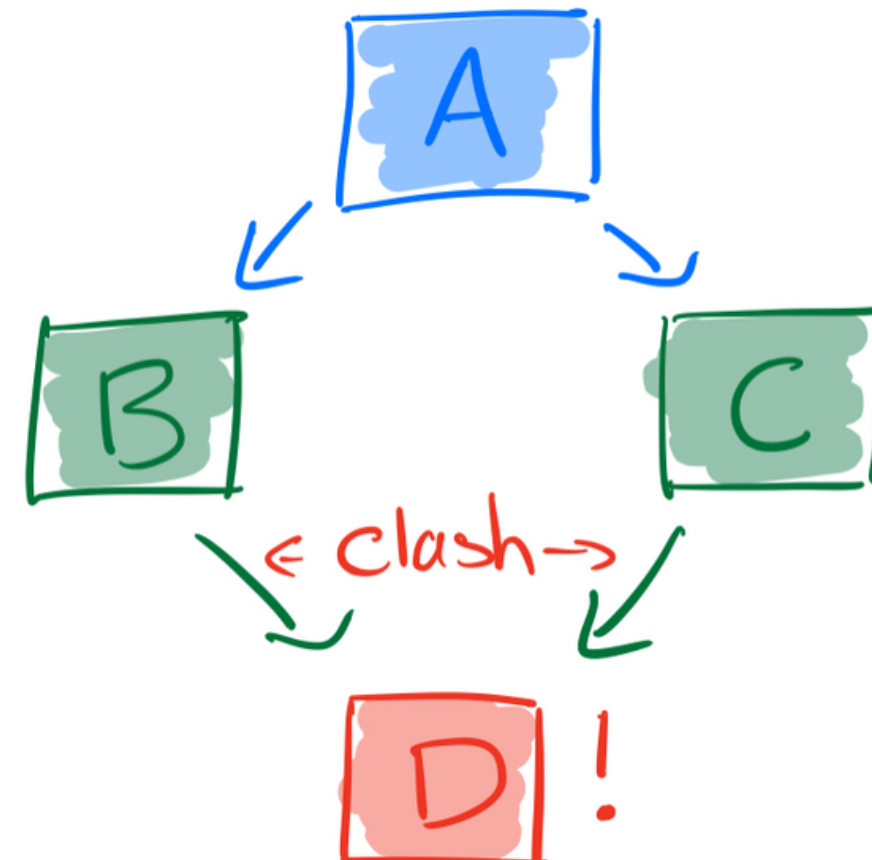
Traditional Application Deployment



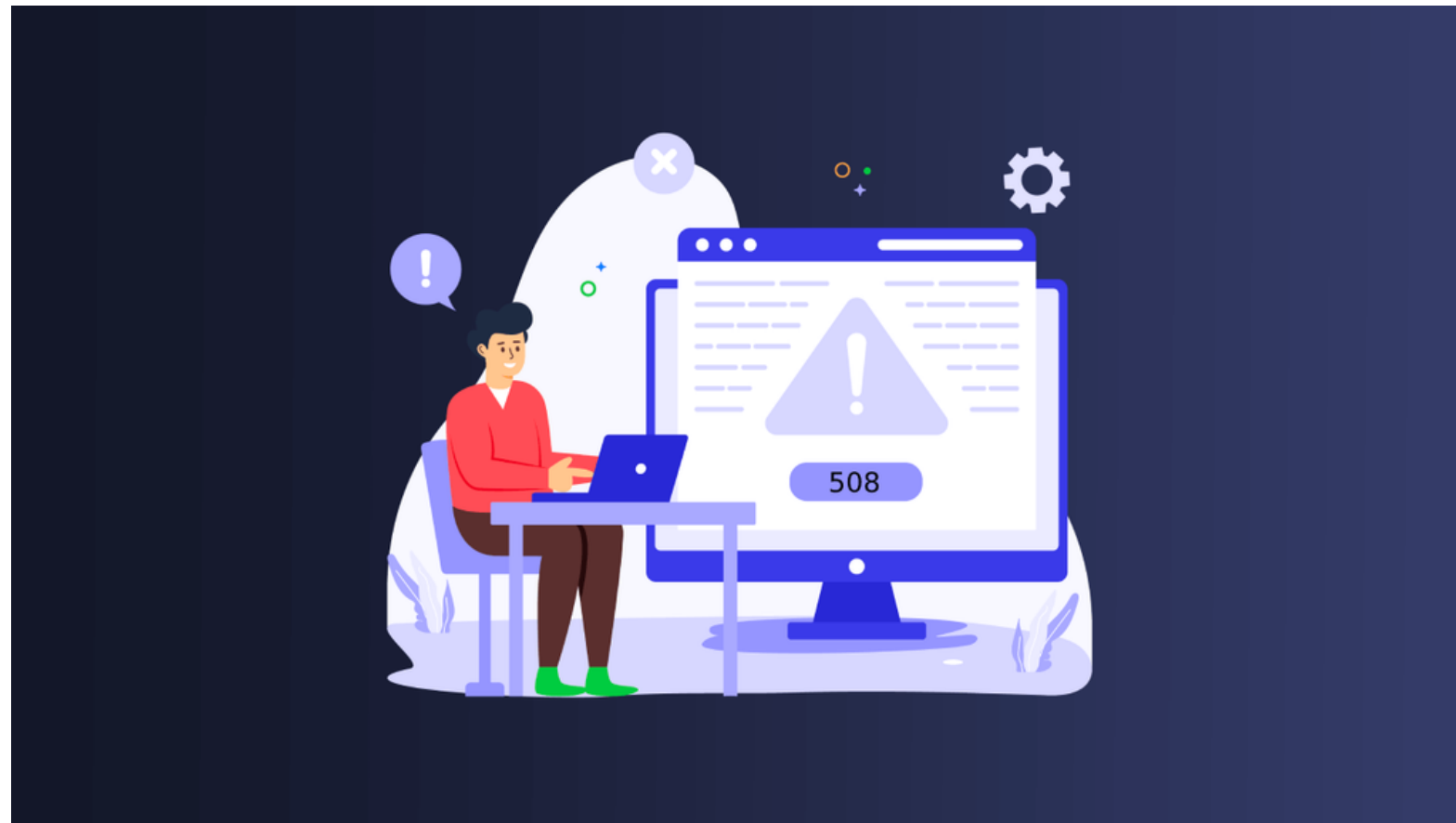
What is the problem with that?

The **same** production server is usually used for **multiple** applications, for efficiency, **but**:

- Error prone process
- Different applications might need different
 - **software & software versions**
 - **libraries & library versions**
- This causes:
 - dependency issues
 - a painful **deployment process**



What is the problem with that?

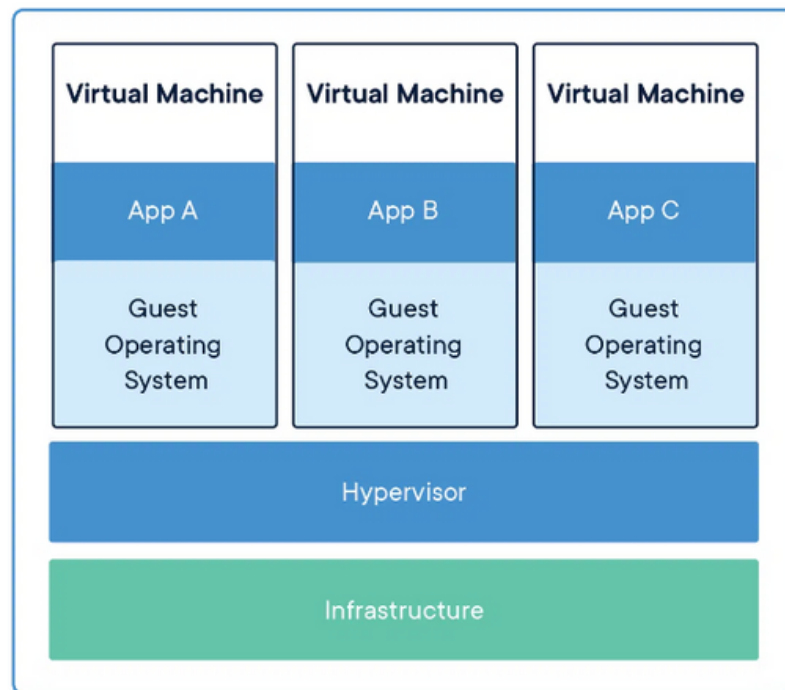


Memory and processing power of the production server are **limited**



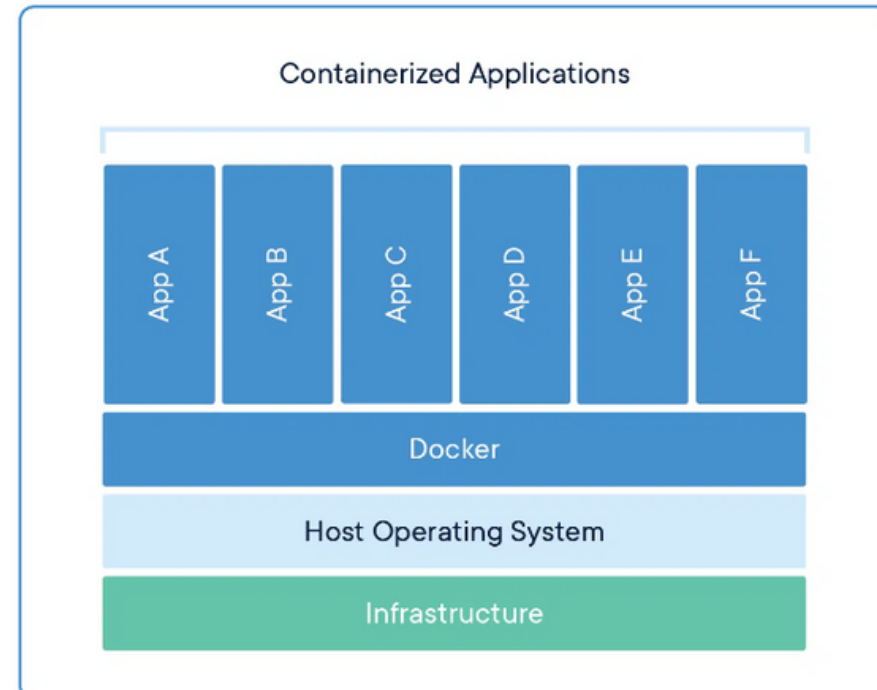
This is where Kubernetes & Docker play very well together and you will hear all about it in Part 2 of this workshop by Katharina

Enter Docker



VM Virtualization

OS Kernel & the application layer of the OS

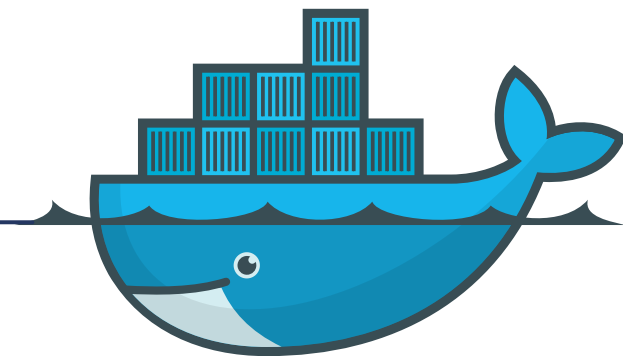


Docker Virtualization

Only the application layer of the OS

Benefits

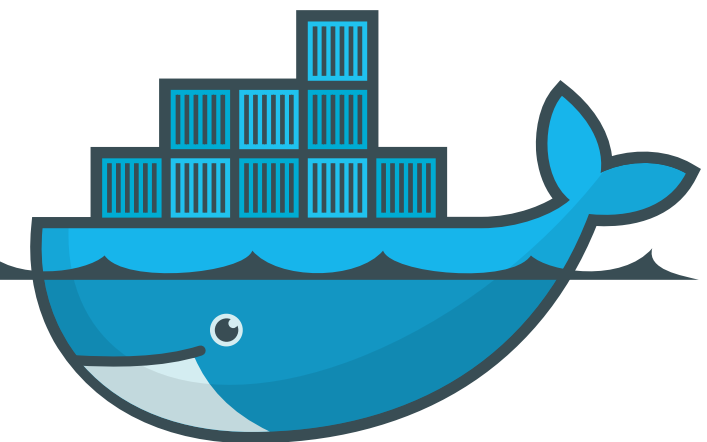
- **Isolation**
 - No conflicting with the host system
- **Packaged application and all its dependencies** (code, libraries, system binaries etc)
 - No more “it works on my machine” issues
- **Reliable deployment**
 - A **self-contained** image is used to deploy, making the deployment independent of the OS or other details of the host system
 - Docker runtime is **the only** necessary installation



Enter Docker

More benefits

- **Better resource consumption management**
 - Configurable memory & cpu a container can use ([docs on resource constraints](#))
- **Efficiency**
 - A side effect of the lightweight, efficient isolation model of containers
 - **Many** Docker containers can run on a single production server

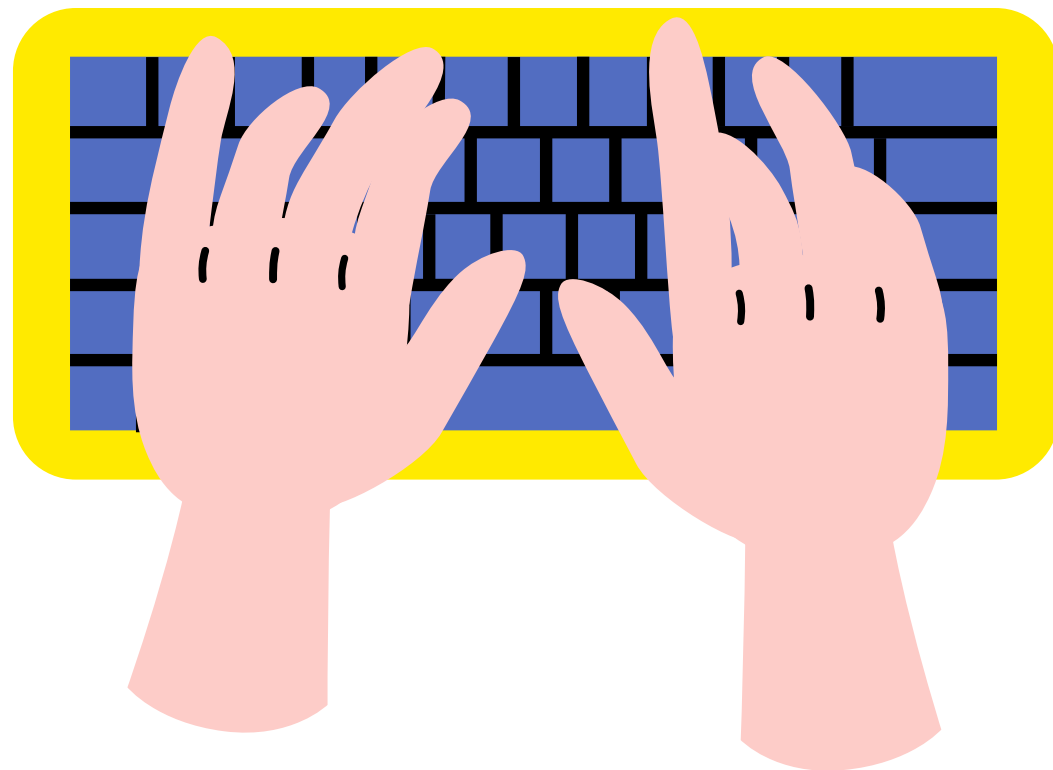
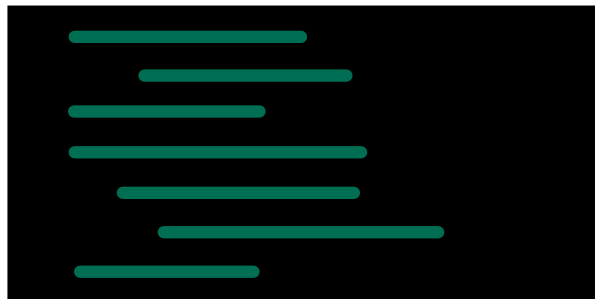


Demo Time



- 💡 Step by step Docker introduction
- 💡 Docker Registry
- 💡 Use existing Docker Images
- 💡 Create your own Docker Image

First steps



- **Docker running locally**

- Instructions to download and install Docker
- I run/prefer **Rancher Desktop**
 - An open-source application that provides all the essentials to work with containers and Kubernetes on the desktop



- **An IDE or text editor to use for editing files.**

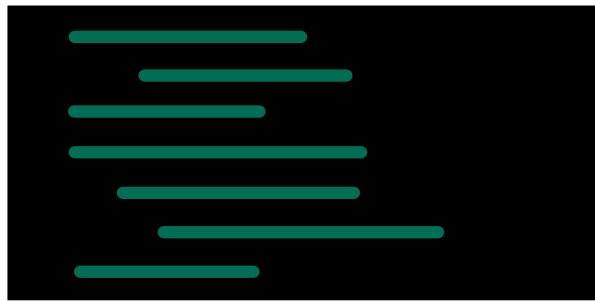
- I prefer VSCode - [Download link](#)



- **Free Docker Account**

- You can sign-up for a free Docker account (<https://hub.docker.com>) and access free unlimited public repositories
- **Not mandatory**, images are available to unauthenticated users too

Demo Part 1



- **Where to find Docker Images?**

- Docker Registries
 - Public (Docker Hub)
 - Private (AWS ECR, Google Container Registry, etc)



- **How to create a Container?**



- **How to access a containerised application**

- Port Binding



- **How to stop Containers**



- **How to remove Docker Images and Containers**

Demo Part 2

Creating your own Docker Image



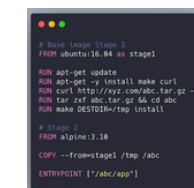
- Creating a Streamlit Application



- Running it locally with `streamlit run`



- Building a Docker Image for this Application



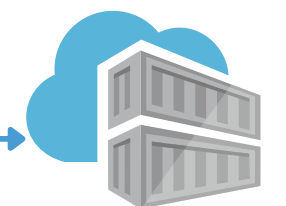
Dockerfile

— build —>



Docker Image

— run —>



Docker Container

Main Takeaways



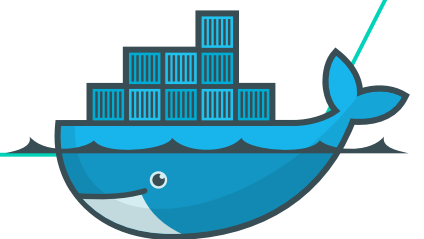
Docker is an open source virtualization technology



With Docker you can package your application into a single runnable artifact - Docker Image



Docker Images will run everywhere, independent of the platform you deploy it to



Where to go from here?

Explore the official documentation

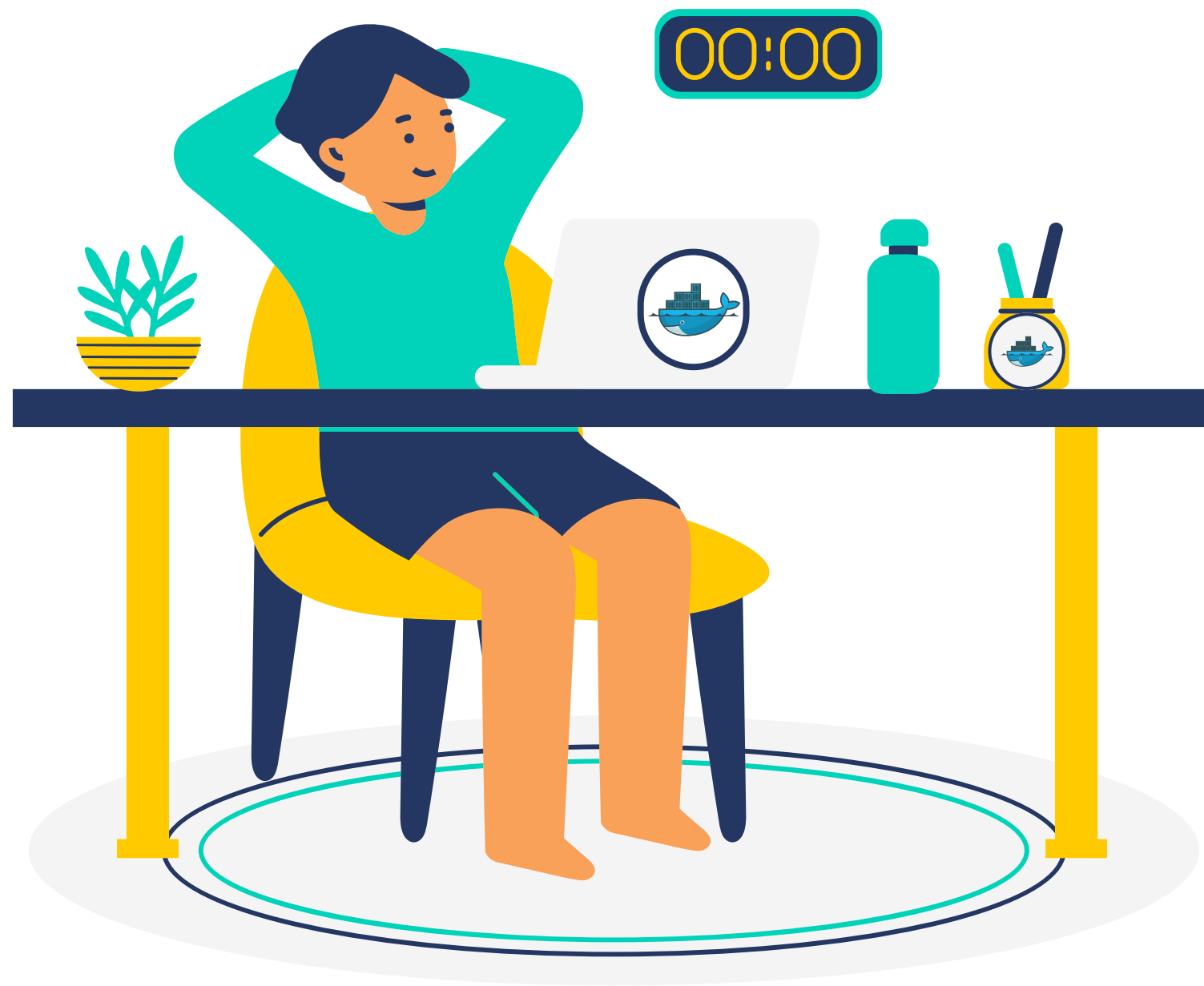
- <https://docs.docker.com/get-started/>
- <https://www.docker.com/resources/what-container/>
- <https://docker-curriculum.com/#docker-compose>
- <https://rancherdesktop.io/>
- <https://docs.streamlit.io/>

Watch YouTube videos

- [Techworld with Nana](#)

Experiment with side projects

- Build a simple application in any language of your choice and containerise it
- Play around and experiment with Docker
- Have fun!



Thank you!

Happy to take any
questions.

Kubernetes

What is Kubernetes? Why do so many companies need it? How to get started?



What is Kubernetes?

Container Orchestration System

- Takes care of all the containers you want to run

Automates Deployment, Scaling and Management

- Deploys your applications based on your description (declarative configuration)
- Scale on demand
- Self-healing capabilities

Open Source

- Open-sourced by Google in 2014
- Part of the Cloud Native ecosystem

Kubernetes = Helmsman/Pilot = k8s



Why is it so useful?

It helps to organize and manage a huge amount of containers

It can scale your applications in busy times

It makes communication between various components easier

It allows you to update your applications without downtime

It has a very big and supportive community



When to use it?

If you have a huge amount of applications and services running

- Basically, it makes the most sense in enterprise environments

It can be an overkill in many situations and there are good alternatives for smaller projects

- Hosting your own server
- Serverless functions (Google Cloud Run, AWS Lambda,...)



Core Components



Core Components

Node

- A worker machine (physical or virtual)
- A Kubernetes cluster runs on one or multiple nodes



Core Components

Node

- A worker machine (physical or virtual)
- A Kubernetes cluster runs on one or multiple nodes

Pod

- Runs one or multiple containers



Core Components

Node

- A worker machine (physical or virtual)
- A Kubernetes cluster runs on one or multiple nodes

Pod

- Runs one or multiple containers

Deployment

- Manages a set of pods and ensures they are up and running



Core Components

Node

- A worker machine (physical or virtual)
- A Kubernetes cluster runs on one or multiple nodes

Pod

- Runs one or multiple containers

Deployment

- Manages a set of pods and ensures they are up and running

Service

- Exposes pods to other pods and services in the cluster



Core Components

Node

- A worker machine (physical or virtual)
- A Kubernetes cluster runs on one or multiple nodes

Pod

- Runs one or multiple containers

Deployment

- Manages a set of pods and ensures they are up and running

Service

- Exposes pods to other pods and services in the cluster

Namespace

- Isolates resources in a cluster



Let's go!

Install kind

- We'll use kind to locally run a Kubernetes cluster
- <https://kind.sigs.k8s.io/docs/user/quick-start/#installation>
- Verify if kind is working:
 - **kind version**

Install kubectl

- We'll use kubectl to interact with the Kubernetes cluster
- <https://kubernetes.io/docs/tasks/tools/#kubectl>
- Verify if kubectl is working:
 - **kubectl version --client**

Let's really go!

Create a local Kubernetes cluster

- `kind create cluster --name kubernetes-playground`

Execute first commands in the cluster

- `kubectl cluster-info`
- `kubectl explain pod`

Take a look at kubeconfig

- `cat ~/.kube/config`

Explore the newly created cluster

- `kubectl get namespaces`
- `kubectl get nodes -o wide`
- `kubectl get pods`

Let's deploy something !

Create a file called deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-kubernetes
spec:
  template:
    metadata:
      labels:
        app: hello-kubernetes
    spec:
      containers:
        - image: paulbouwer/hello-kubernetes:1
          name: hello-kubernetes
          ports:
            - containerPort: 8080
              name: http
      selector:
        matchLabels:
          app: hello-kubernetes
```

Let's deploy something II

Create a namespace

- `kubectl create namespace hello-kubernetes`

Create a deployment

- `kubectl apply -n hello-kubernetes -f deployment.yaml`

Take a look at the created resources

- `kubectl -n hello-kubernetes get deployments`
- `kubectl -n hello-kubernetes get pods`
- `kubectl port-forward -n hello-kubernetes deployment/hello-kubernetes 8080:8080`

Let's set environment variables

Add an environment variable

- `spec.template.spec.containers[0].env:`
 - - `name: MESSAGE`
 - `value: 'Hello #theNewITGirls! :)'`

We can also read information from Kubernetes

- `spec.template.spec.containers[0].env:`
 - - `name: KUBERNETES_NAMESPACE`
 - `valueFrom:`
 - `fieldRef:`
 - `fieldPath: metadata.namespace`
 - - `name: KUBERNETES_POD_NAME`
 - `valueFrom:`
 - `fieldRef:`
 - `fieldPath: metadata.name`
 - - `name: KUBERNETES_NODE_NAME`
 - `valueFrom:`
 - `fieldRef:`
 - `fieldPath: spec.nodeName`

Let's use a configmap

Create a file called configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-kubernetes-config
data:
  message: "Hello from the config map! :)"
```

Apply it

- `kubectl apply -n hello-kubernetes -f configmap.yaml.yaml`

Reference it

- `spec.template.spec.containers[0].env:`
 - `- name: MESSAGE`
 - `valueFrom:`
 - `configMapKeyRef:`
 - `name: hello-kubernetes-config`
 - `key: message`

Let's use a secret

Create a file called secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: hello-kubernetes-secrets
stringData:
  message: "I am a secret value!"
```

Apply it

- `kubectl apply -n hello-kubernetes -f secret.yaml`

Reference it

- `spec.template.spec.containers[0].env:`
 - - `name: MESSAGE`
 - `valueFrom:`
 - `secretKeyRef:`
 - `name: hello-kubernetes-secrets`
 - `key: message`

Let's scale it

Set replicas

- `spec.replicas: 3`

Take a look at the created resources

- `kubectl -n hello-kubernetes get deployments`
- `kubectl -n hello-kubernetes get pods`
- `kubectl port-forward -n hello-kubernetes deployment/hello-kubernetes 8080:8080`

Let's update without downtime

Change the update strategy

- `spec.strategy.type: RollingUpdate`

Take a look at the created resources

- `kubectl -n hello-kubernetes get deployments`
- `kubectl -n hello-kubernetes get pods`
- `kubectl port-forward -n hello-kubernetes deployment/hello-kubernetes 8080:8080`

Let's create a service

Create a file called service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-kubernetes
spec:
  selector:
    app: hello-kubernetes
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

Create a service

- `kubectl apply -n hello-kubernetes -f service.yaml`

Let's clean up

Delete the Kubernetes cluster

- `kind delete cluster --name kubernetes-playground`

Where to go from here?

Explore the official documentation

- <https://kubernetes.io/docs/home/>

Watch YouTube videos

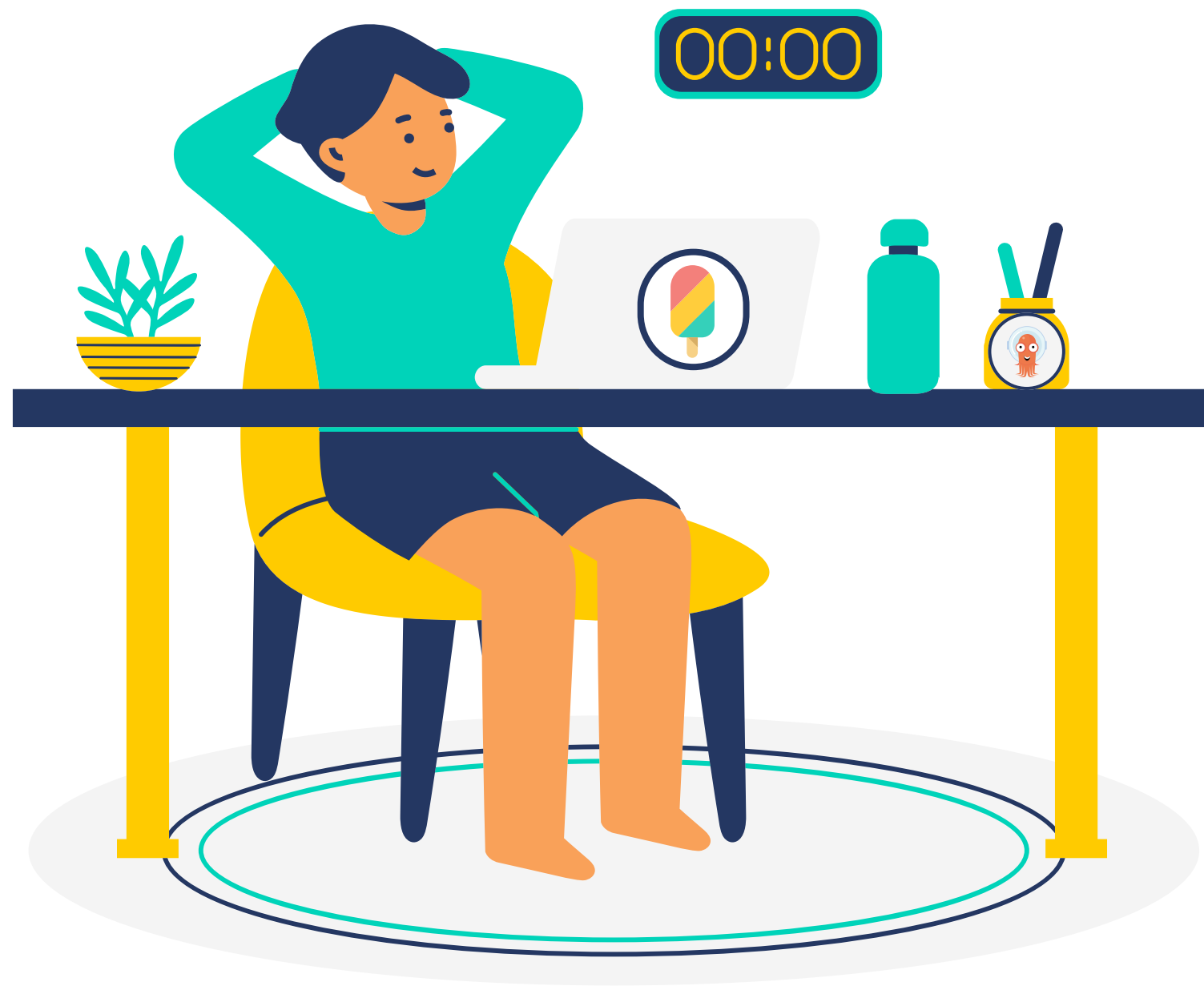
- [Techworld with Nana](#)
- [100 Days of Kubernetes](#)

Experiment with side projects

- Trying examples is a great way of learning
- Just play around and experiment with workloads
- Celebrate small successes

Join Kubernetes communities

- [Kubernetes Slack](#)
- [Cloud Native Linz](#)
- [Cloud Native Austria](#) (Graz, Innsbruck, Salzburg & Vienna)



Thank you! :)

Happy to take any
questions.