

What is Docker? And Why?

Docker is a **container** technology: A tool for creating and managing containers

Container

A standardized unit of software

A package of code **and** dependencies to run that code (e.g. NodeJS code + the NodeJS runtime)



The same container always yields the **exact same application and execution behavior!** No matter where or by whom it might be executed.

Support for Containers **is built into** modern operating systems!

Docker simplifies the creation and management of such containers

Let's Take A Step Back

Dishes

Food

A Picnic Basket



Let's Take A Step Back



It's portable

It contains food and dishes

You can share it and use it everywhere

No special environment or tools are required

Why Containers?

Why would we want independent, standardized “application packages”?



Different Development & Production Environments

We want to build and test in exactly (!) the same environment as we later run our app in



Different Development Environments Within a Team / Company

Every team member should have the exactly (!) same environment when working on the same project



Clashing Tools / Versions Between Different Projects

When switching between projects, tools used in project A should not clash with tools used in project B

The Problems

Environment: The runtimes, languages, frameworks you need for development

Development Environment



Production Environment

often not the same

Development Environment
for Employee A



Development Environment
for Employee B

often not the same

Tools & Libraries required
for Project A



Tools & Libraries required
for Project B

often not the same

We Want Reliability & Reproducible Environments



We want to have the **exact same environment for development and production** → This ensures that it works exactly as tested



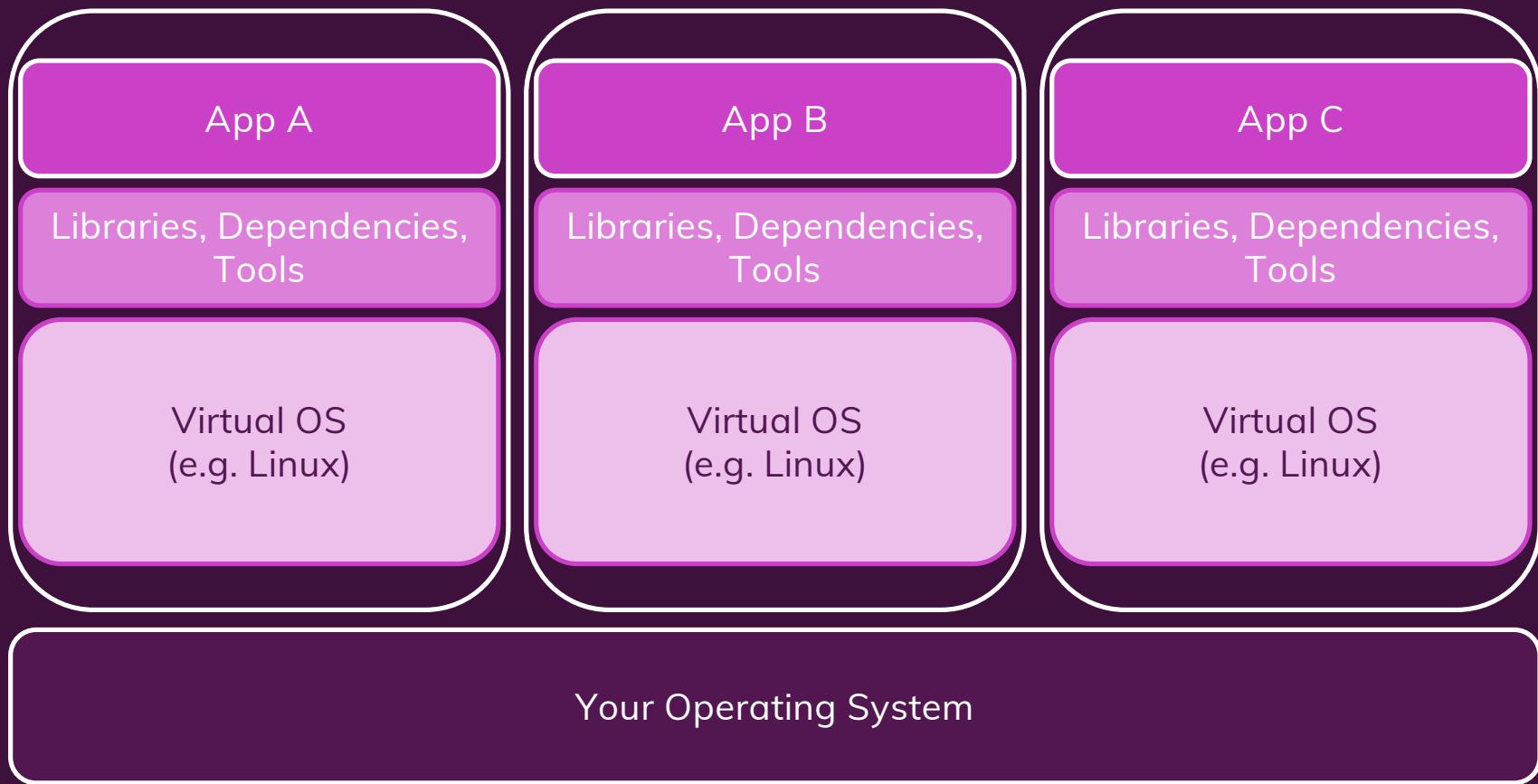
It should be easy to **share a common development environment/ setup** with (new) employees and colleagues



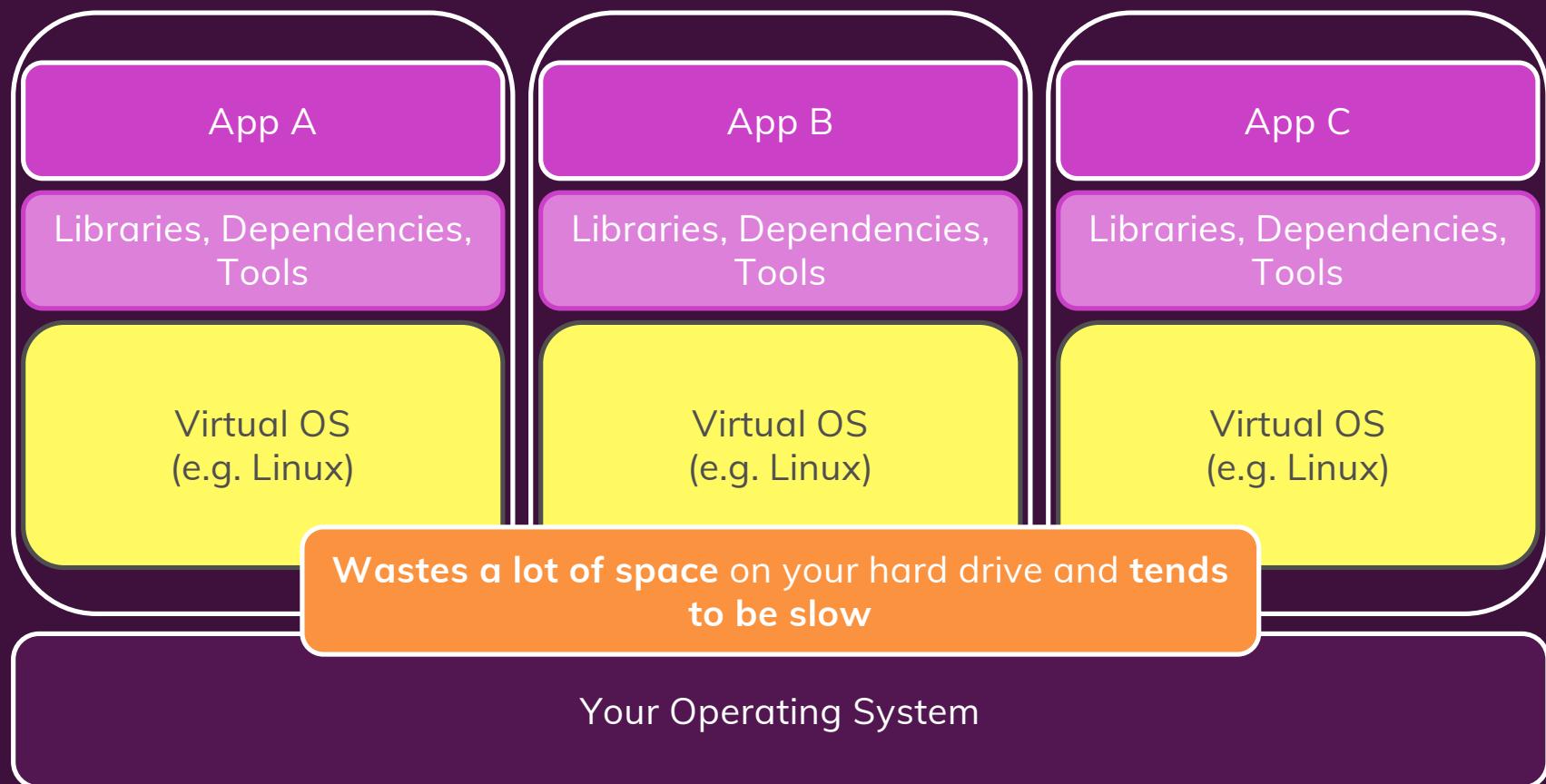
We **don't want to uninstall and re-install** local dependencies and runtimes all the time

Solution: Virtual Machines / Virtual Operating Systems

Share within multiple applications



Solution: Virtual Machines / Virtual Operating Systems



Virtual Machines / Virtual OS: Summary

Pro

Separated environments

Environment-specific configurations are possible

Environment configurations can be shared and reproduced reliably

Con

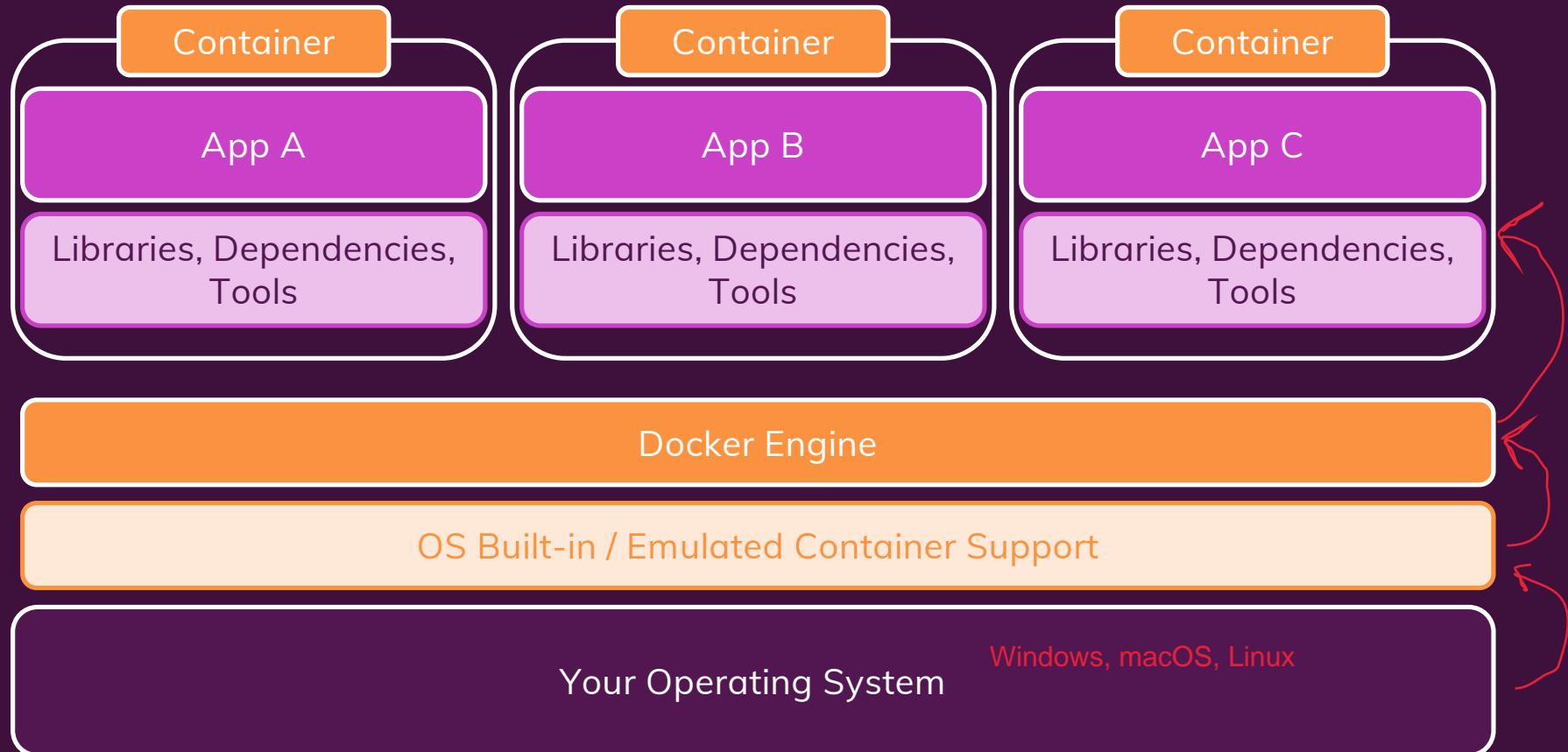
Redundant duplication, waste of space

Performance can be slow, boot times can be long

Reproducing on another computer/server is possible but may still be tricky

configure it in the same way everytime in production machine

Docker Helps You Build & Manage “Containers”



Containers vs Virtual Machines

Docker Containers

Low impact on OS, very fast,
minimal disk space usage

Sharing, re-building and
distribution is easy

Encapsulate apps/ environments
instead of “whole machines”

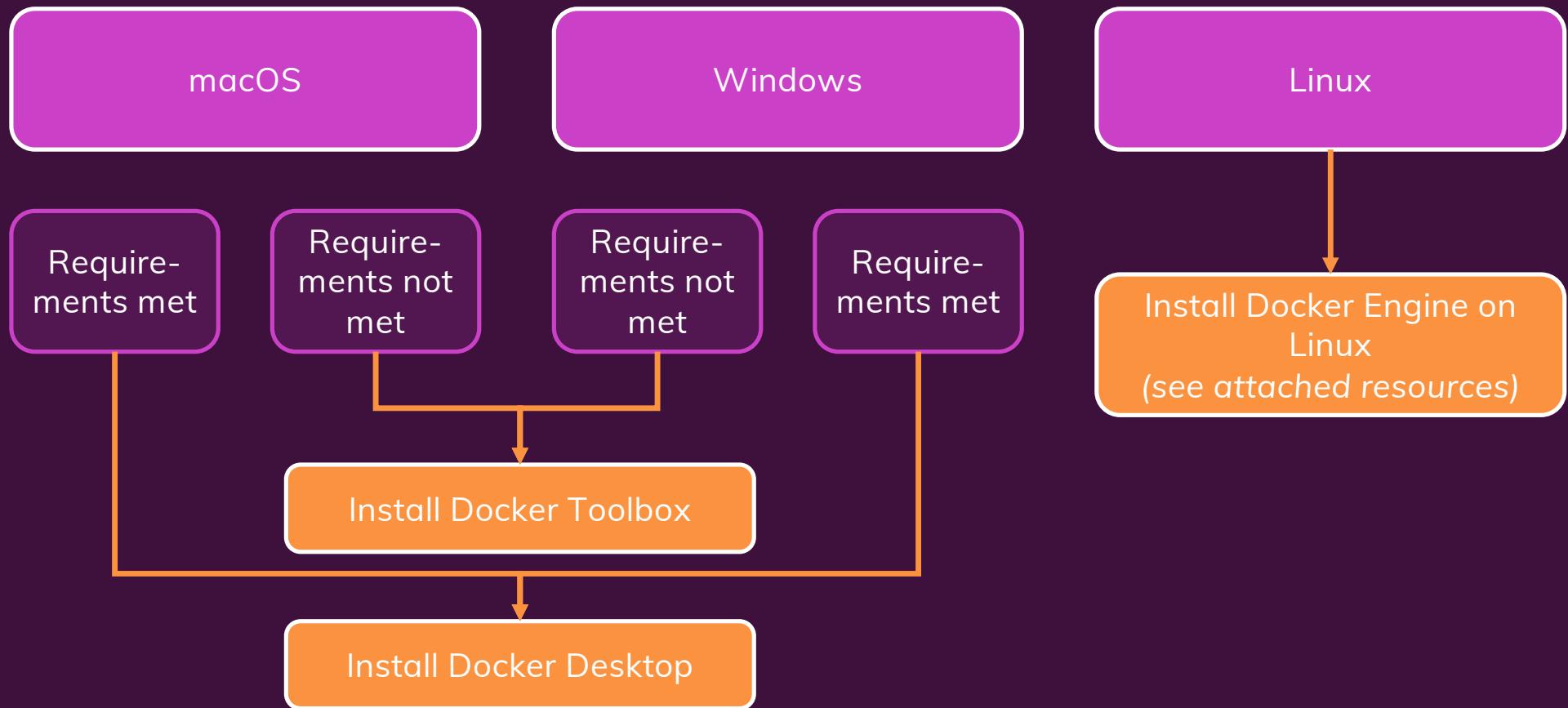
Virtual Machines

Bigger impact on OS, slower,
higher disk space usage

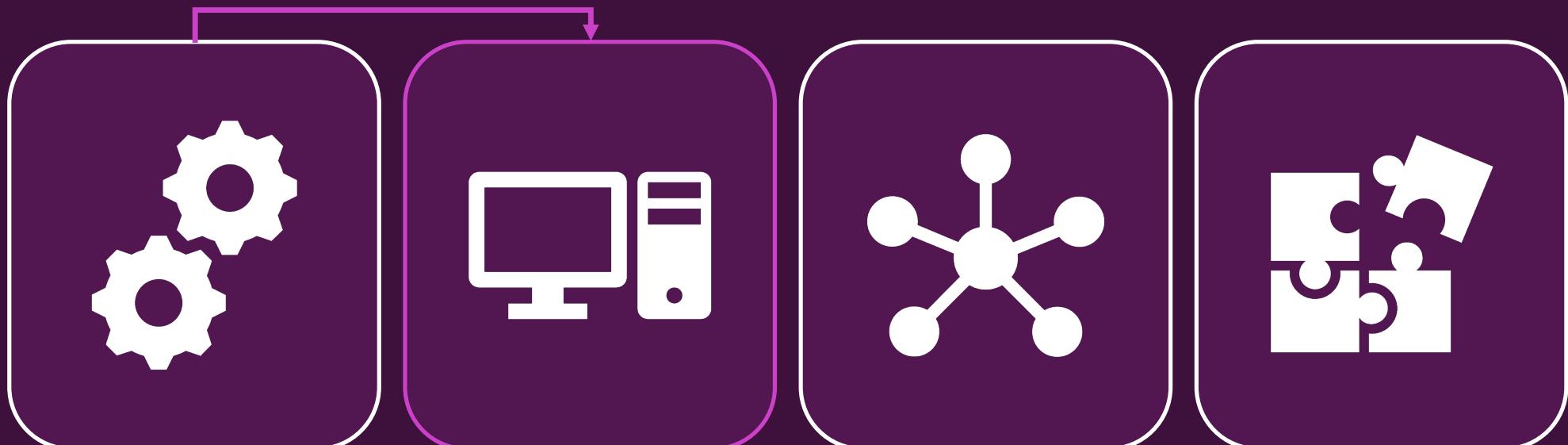
Sharing, re-building and
distribution can be challenging

Encapsulate “whole machines”
instead of just apps/ environments

Docker Setup



Docker Tools & Building Blocks



Docker Engine

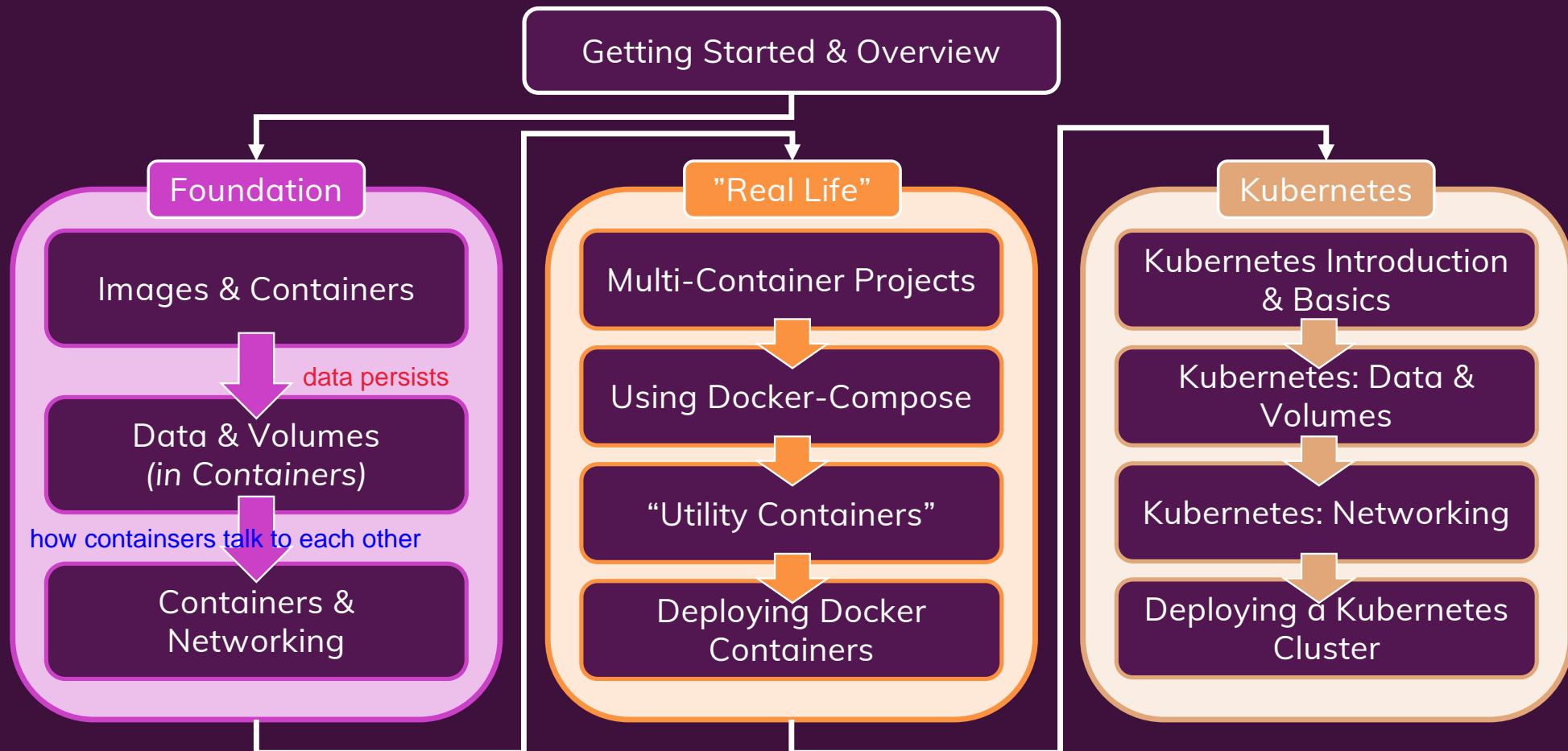
Docker Desktop (incl.
Daemon & CLI)

Docker Hub

Docker Compose

Kubernetes

Course Outline



Getting The Most Out Of This Course

Docker Image:

- A Docker image is a file used to execute code in a Docker container.
- Docker images act as a set of instructions to build a Docker container, like a template.
- Docker images also act as the starting point when using Docker.
- An image is comparable to a snapshot in virtual machine (VM) environments.

Watch the Videos



At your pace – use the video player controls

Code along



Pause videos, code along, code ahead

Repeat Concepts



Repeat videos or sections if unclear

Google, Stackoverflow



Search the web in case of errors

Ask & Answer in Q&A Section



Ask and help others in the Q&A board