From program to Kubernetes

Reproducible experiments: from dockerization to deployment on Kubernetes

Jey Puget Gil

Reproducibility

Why containerize our experiments in the first place?

Reproducibility: a cornerstone of science

Why we should care

1. Sharing **code & data** lets others verify results and build upon them
2. Increasingly **required** by funders, journals and conferences
3. Many results cannot be reproduced: code and/or data simply not provided

References: [Mic26]

Terminology

Repeat	**Replicate**	**Reproduce**	**Reuse**
Same experiment	Same experiment	Same experiment	New ideas/ experiment
Same setup	Same setup	~~Same setup~~	Same / **New** setup
Same lab	~~Same lab~~	~~Same lab~~	**New** lab

References: [Coh+17]

$$\text{Repeat} \rightarrow \underbrace{\textbf{Replicate} \rightarrow \textbf{Reproduce}}_{\approx \text{ Reproducible Research}} \rightarrow \text{Reuse}$$

The challenge in our experimental practices

> "It works on my machine" – the most expensive sentence in computational research.

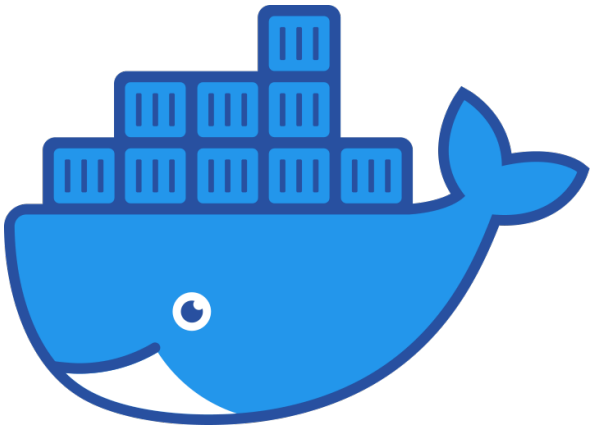
Where reproducibility breaks down

1. The experiment relies on an environment that lives **only** on the author's laptop
2. OS, system libraries, language runtimes and package **versions** drift over time
3. **Undocumented** dependencies and manual, one-off setup steps
4. Painful to re-run months later, on a server, or by a colleague

References: [Mic26]

Why containers are a good answer

Docker makes the environment reproducible



1. The **image** bundles OS libraries, dependencies and code together
2. It runs **identically** on a laptop, CI, a server or a cluster
3. The **Dockerfile** is an explicit, versioned recipe of the setup
4. Images are **immutable** and pinned to an exact version (tags / digests)
5. Pinned images can be **archived, shared and cited** → referenceable runs

References: [Lau22], [Mic26]

Containerization

Image and container

Containerization

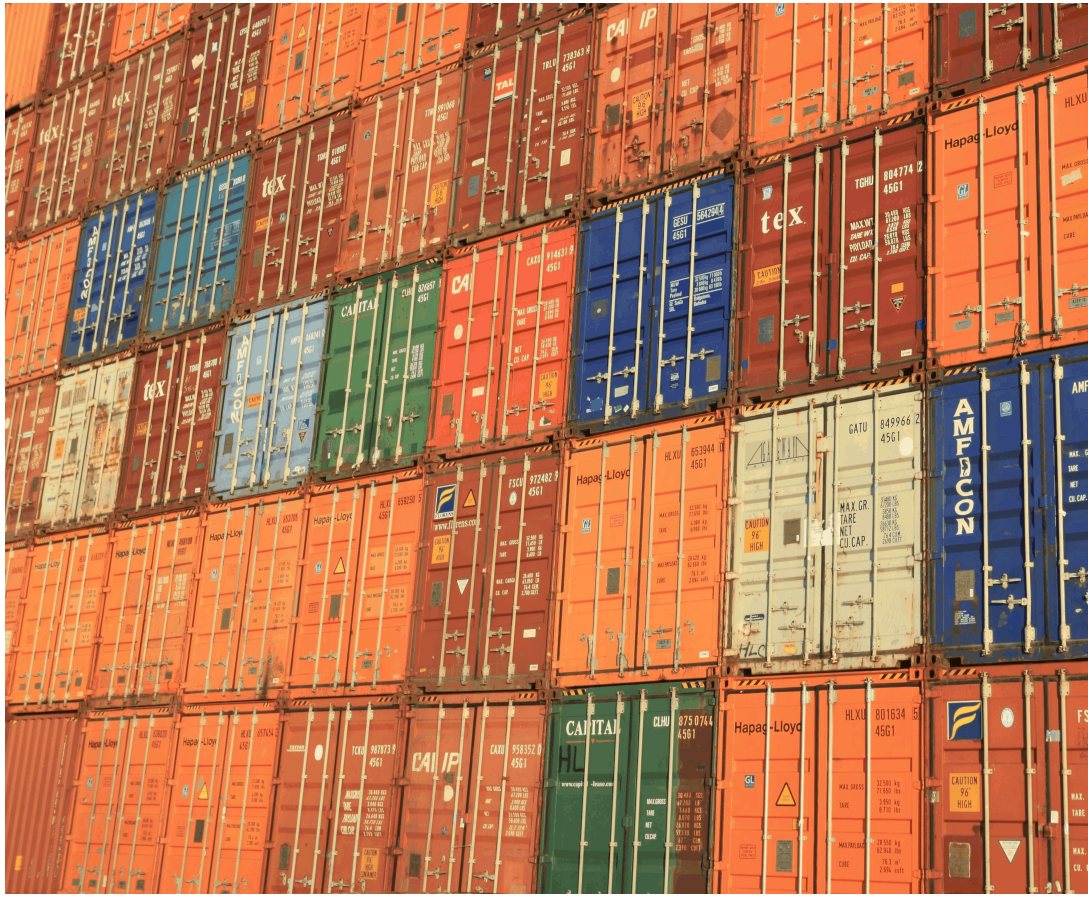
What is an image?

1. Image = files + metadata
2. These files form the root filesystem of our container.
3. Set of layers (each layer is a set of filess)

What is a container?

1. Allows you to run an application in an isolated environment
2. Is never modified
3. Will always be identical

References: [AJ 17], [Lau22]



Differences between containers and images

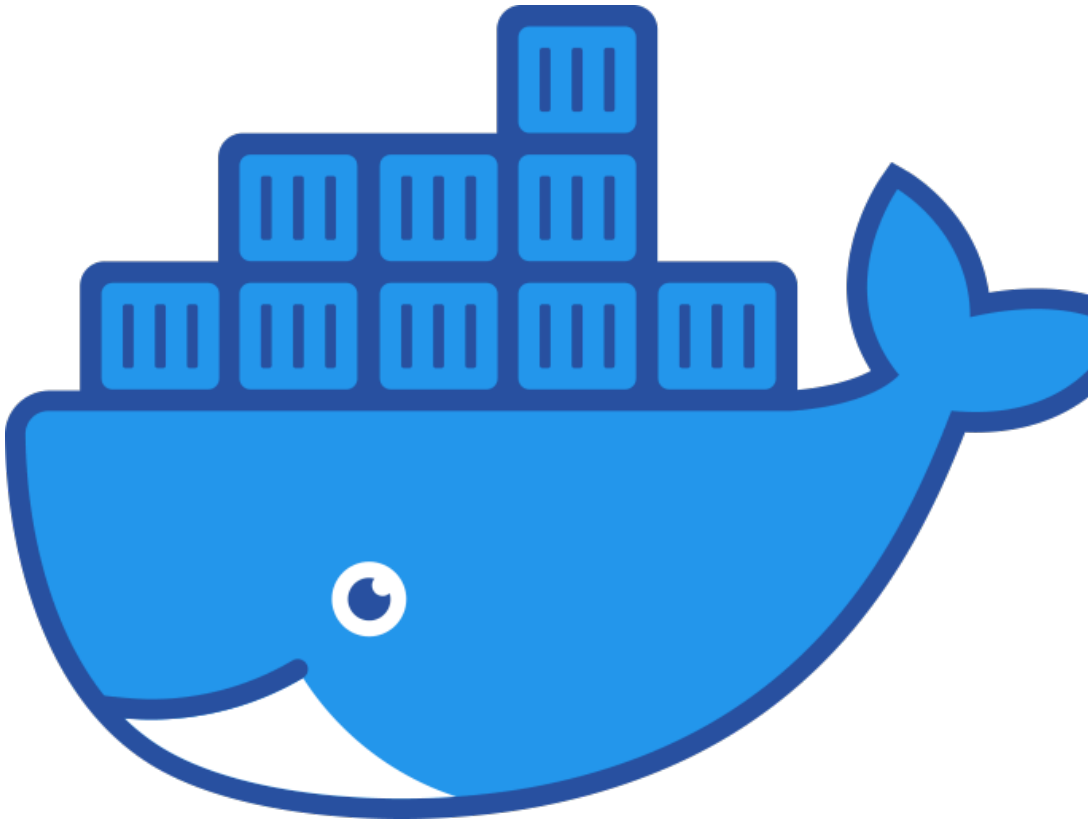
Takeaways

1. An image is a read-only filesystem.
2. A container is an encapsulated set of processes running in a read-write copy of that filesystem.

Analogy (OOP)

1. Class = Image
2. Instance = Container

References: [AJ 17], [Lau22]



Building Images

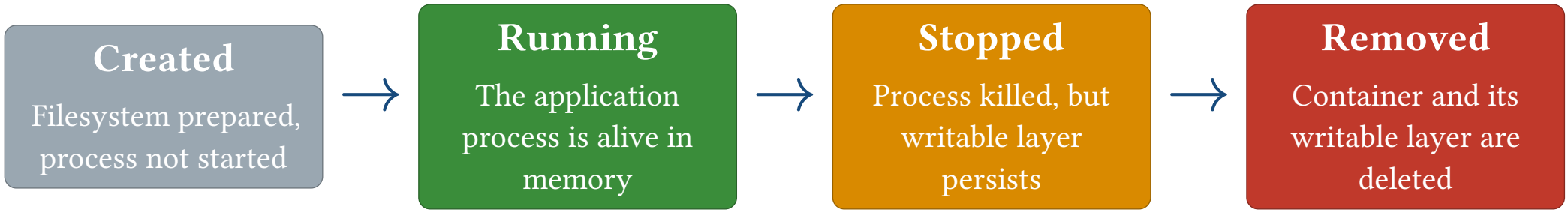
Creating images with Dockerfile

1. Dockerfile is a text file with instructions to build an image
2. Each instruction creates a new layer in the image
3. Example:

```
1 FROM python:3.9-slim
2 WORKDIR /app
3 COPY . .
4 RUN pip install -r requirements.txt
5 CMD ["python", "app.py"]
```

References: [AJ 17]

Container lifecycle



Note: The underlying Image remains untouched throughout.

Volumes

Why use volumes?

1. Containers are ephemeral; data inside them is lost when they stop.
2. Volumes provide persistent storage that can be shared between containers.
3. Data to persist beyond the lifecycle of a container.

Experienced data loss?

Types of volumes

1. **Named Volumes:** Stored in Docker's internal volume store
2. **Bind Mounts:** Mounted from the host filesystem
3. **Anonymous Volumes:** Created without a name and deleted when the container is removed
4. **tmpfs Volumes:** Stored in memory and never written to disk

References: [Rah24]

Orchestration

What is container orchestration?

Container orchestration

What is container orchestration?

1. Automated process of managing, scaling, and deploying containerized applications
2. Tools: Kubernetes, Docker Swarm, Apache Mesos

Welcome to Kubernetes (K8s)

1. K8s starts with an K followed by 8 letters and ends with s
2. It runs and manages containerized applications on a cluster

References: [AJ 19]

K8s architecture

Reference Architecture

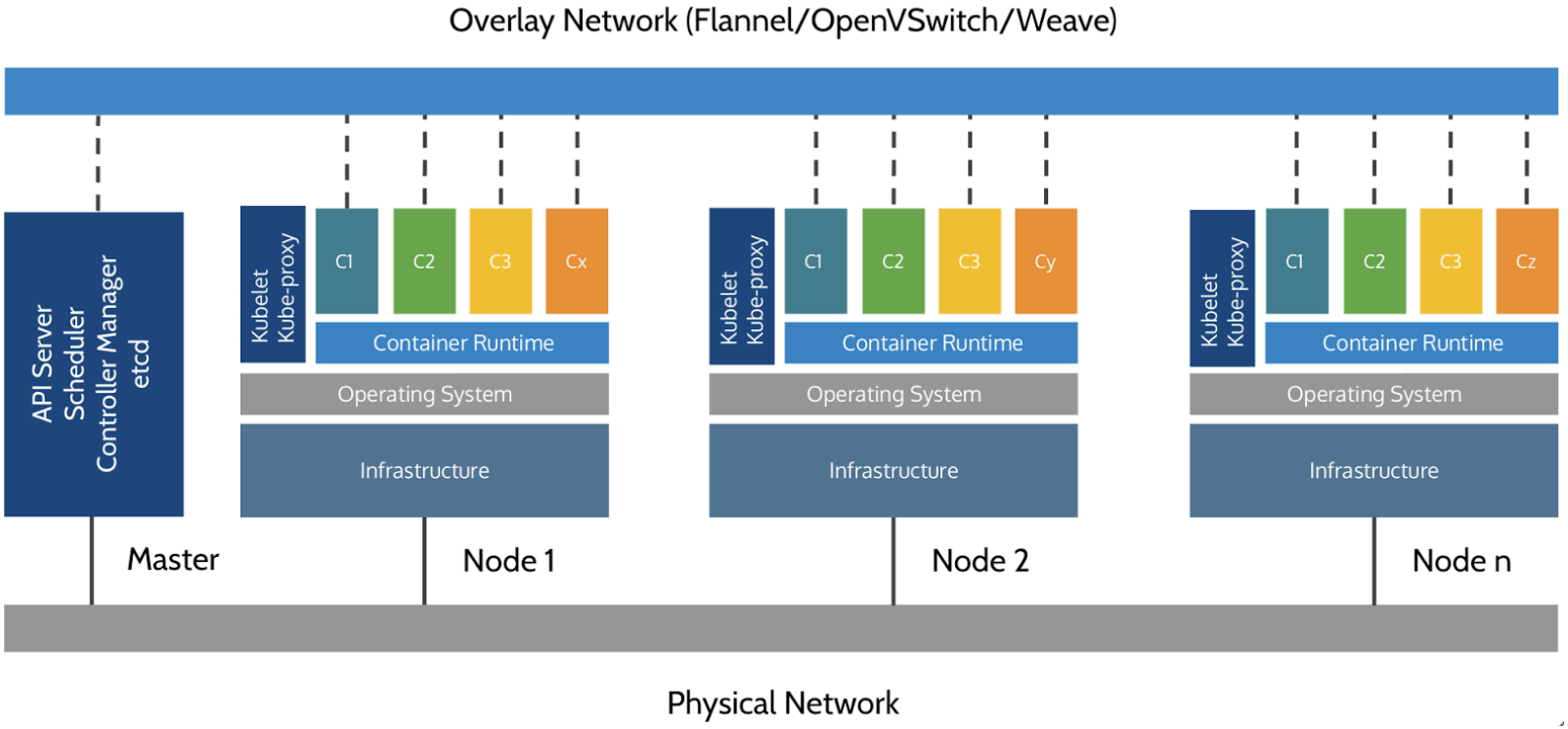


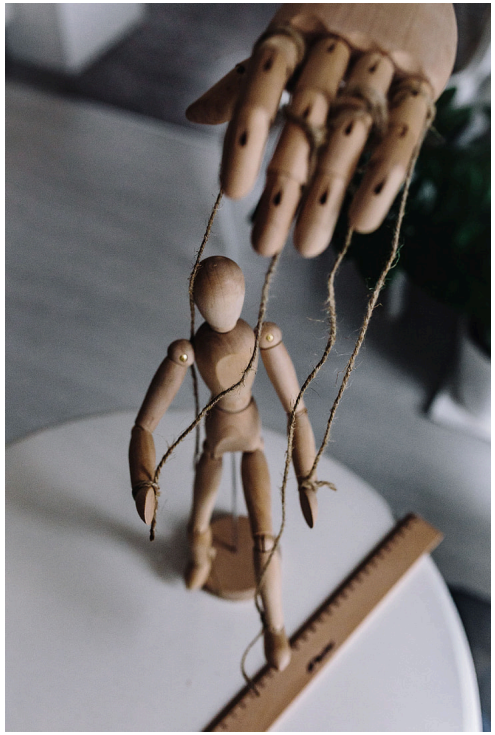
Figure 1: A Reference Architecture for Deploying WSO2 Middleware on K8s, [Gun17]

K8s use cases

What can we do with K8s?

1. Deploy and manage microservices
2. Scale applications up and down based on demand
3. Roll out new features without downtime (rolling updates)
4. Self-healing: automatically restart failed containers, replace and reschedule containers when nodes die

References: [Aut26]



Thank you!

Let’s discuss and then create a reproducible experiment together

Figure 2: QR code - https://github.com/JPugetGil/Argo_Hera_Workflow_tutorial

References

Mic26 Michel, F.: “Open Science, reproducible research, and the citation of articles, code and data alike”, https://doi.org/10.5281/zenodo.19557161

Coh+17 Cohen-Boulakia, S., Belhajjame, K., Collin, O., et al.: “Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities”Future Generation Computer Systems, 2017, **75**, pp. 284–298.

Lau22 Laurent Facq, R.T.: “Introduction et bonnes pratiques de la conteneurisation” (2022)

AJ 17 AJ Bowen, J.P.: “Introduction to Docker and Containers” (2017)

Rah24 Rahul: “Understanding Docker Volumes: A Comprehensive Guide” (2024)

AJ 19 AJ Bowen, J.P.: “Getting started with Kubernetes and container orchestration” (2019)

Gun17 Gunaratne, I.: “A Reference Architecture for Deploying WSO2 Middleware on Kubernetes” (2017)

Aut26 Authors, T.K.: “Kubernetes Documentation” (2026)