Content

- **3.1 Base Software Stack**
  - Foundational Software Stack
  - Environment Modules System
  - Compiling and Running C Programs
- **3.2 Workload Management with SLURM**
  - The Essentials of SLURM
  - Job Directives
  - Computing Resource Allocation
- **3.3 Getting Started with Docker Containers**
  - Docker basics
  - Launching a Jupyter Notebook server inside a Docker
- **3.4 Containerization with Singularity**
  - Containers in Supercomputing
  - Building Containers

Getting Started with Docker Container

Dockers Basics

- **Docker = leading container platform**
  - `https://www.docker.com`
- **Packages app + dependencies → standardized unit**
- **Runs the same across Linux, Windows, macOS**
- **Containers = isolation from environment differences**
- **Benefits: reproducibility, portability, …**

Docker Image

- Lightweight, standalone, executable package
- Includes:
  - Code
  - Runtime
  - Libraries
  - System tools & settings
- Ensures identical behavior across environments

Docker Hub

- **Official repository for container images**
  - `https://hub.docker.com`
- **Public & private images supported**
- **Developers can share or reuse images**
- **Example:** `jorditorresbcn/dl`

Push Image to Docker Hub

1. Create account on Docker Hub
2. Log in:

```
docker login
```

3. Build image locally:

```
docker build -t <user>/<image>:<tag> .
```

4. Push to Docker Hub:

```
docker push <user>/<image>:<tag>
```

5. Others can pull it:

```
docker pull <user>/<image>:<tag>
```

Install Docker

- **Windows:**
  - Docker Desktop (includes CLI, Compose, Kubernetes)
- **Mac:**
  - Docker Desktop (≥ macOS 10.14)
- **Linux:**
  - Install Docker Engine via package manager (e.g., Ubuntu, CentOS, Fedora)
- **Verify installation:**

```
docker --version
docker run hello-world
```

Download Docker Image

- **Command:**

```
docker pull jorditorresbcn/dl
```

- **Verify:**

```
docker images
```

- **Info: name, tag, size**
- **Benefit: reuse pre-built images instead of building from scratch**

Run & Stop Containers

- **Run basic:**

```
docker run <image>
```

- **Run interactive:**

```
docker run -it jorditorresbcn/dl
```

- **List running:**

```
docker ps
```

- **Stop / kill:**

```
docker stop <id_or_name>
docker kill <id_or_name>
```

- **Restart / remove:**

```
docker start <id_or_name>
docker rm <id_or_name>
```

Tasks

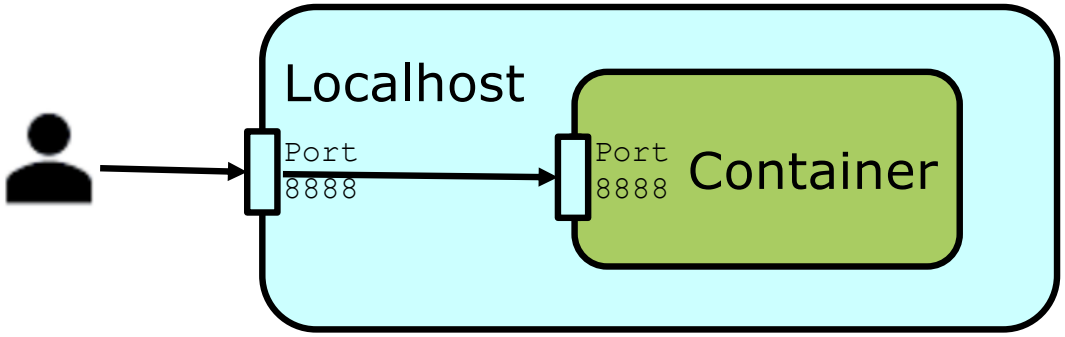
- **Task 3.4 – Install Docker**
  - Install on your platform, verify with docker --version
  - Test with docker run hello-world
- **Task 3.5 – Download Image**
  - Pull jorditorresbcn/dl
  - Verify with docker images
- **Task 3.6 – Run Image**
  - Run interactive container: docker run -it jorditorresbcn/dl
  - Inside: check OS (cat /etc/os-release), list Python libs (pip list)
- **Task 3.7 – Stop Container**
  - Exit with exit or stop with docker stop <id>

Launching a Jupyter Notebook Server Inside a Docker Container

Port Mapping in Docker

```
docker run -it -p 8888:8888 --name test
jorditorresbcn/dl:latest
```

- **-p 8888:8888 → maps container port → host port**
- **Access service inside container (e.g., Jupyter) from browser:** `http://localhost:8888`
- **Ports = endpoints for services, allow multiple apps without conflict**



Jupyter Notebook Basics

- **Open-source web application for creating and sharing documents**
- **Combines in a single file:**
  - Live executable code (Python), Narrative text (Markdown), Equations (LaTeX), Visualizations and interactive widgets, ..
- **Notebook structure:**
  - Organized in cells → run independently
  - Immediate feedback → great for teaching & experimentation
- **File format:**

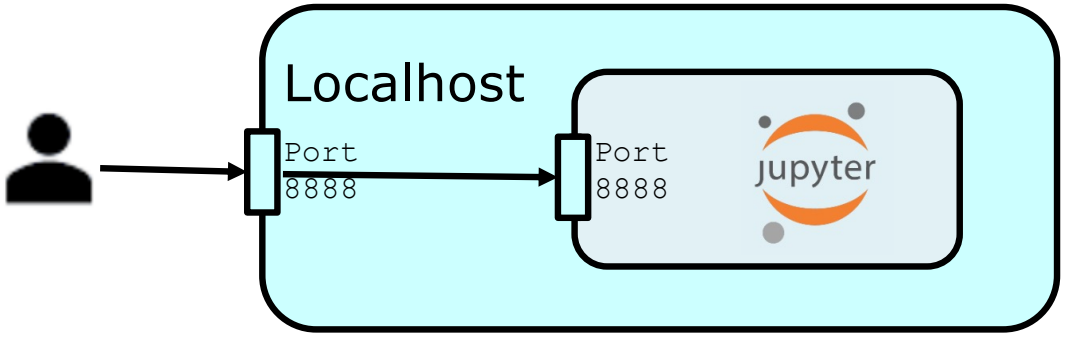
  `.ipynb` (JSON with code, text, outputs, metadata)
- **Where we use them in this course:**
  - Docker
  - Google Colab
  - MareNostrum 5

Starting Jupyter in the Container

- **Command:**

```
jupyter notebook --ip=0.0.0.0 --allow-root
```

  - --ip=0.0.0.0 → accept connections from any IP (not just localhost)
  - Defaults to port 8888 (or next available if in use)
  - --allow-root → required in container environments



Accessing Jupyter Notebook

- **Start server with:**

```
jupyter notebook --ip=0.0.0.0 --port=8888 --no-browser
```

- **Terminal shows URL with token → copy/paste in host browser**
- **Default access:** `http://127.0.0.1:8888`

  (password: `dl`)
- **Now you can create and run notebooks from your browser**

Tasks

- **Task 3.8 – Run Docker with Port Mapping → run container with -p 8888:8888 and test access**
- **Task 3.9 – Start Jupyter Server → run jupyter notebook inside container**
- **Task 3.10 – Create Test Notebook → confirm setup with print("Docker and Jupyter are working!")**

Containers in Supercomputing

Containers in Supercomputing

- In HPC, reproducibility and portability are critical.
- Containers package the complete software stack (OS, libraries, dependencies, tools).
- This ensures consistent execution across different clusters, avoiding local configuration issues.
- Increasingly important for complex scientific workflows and ML applications.

Docker vs Singularity in HPC

- **Docker:**
  - Industry standard for cloud & microservices.
  - Requires root privileges → **unsafe in shared systems**.
  - Not designed for MPI tightly coupled computations.
- **Singularity:**
  - Created by the HPC community.
  - **Runs without root access**, integrates with SLURM.
  - Provides direct access to GPUs, filesystems, and interconnects.
  - De facto standard for HPC at BSC (SingularityCE).

Running Singularity on MareNostrum 5

- **Load module:** `module load SINGULARITY/3.11.5`
- **Container format:** `.sif` **(Singularity Image Format)**
- **Basic execution with GPU passthrough:**
  `singularity exec --nv image.sif python train.py`
- **BSC helper tool:** `bsc_singularity`
  - **Automatically adds GPU support and filesystem bindings**
  - **Provides easy commands:**

```
bsc_singularity ls
bsc_singularity exec <container>
bsc_singularity shell <container>
```

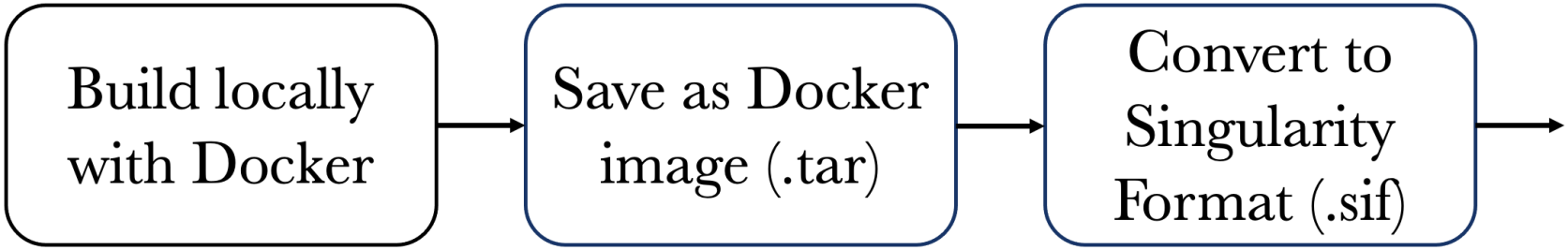
Building Containers

Overview

- **Containers can be built from different sources:**
  - Singularity Library
  - Docker Hub
  - Definition file
  - NVIDIA NGC Docker images
- **The common workflow:**
  - Build locally (requires root + internet).
  - Transfer to HPC system (e.g., MareNostrum 5).

Preferred hybrid approach

- **develop with Docker, deploy with Singularity:**



  - Ensures flexibility in development + compatibility in HPC.
  - For more detail visit section “Building Containers”.

Pb: Containers

- **Tasks included:**

  Task 3.4 – Install Docker in Your Platform

  Task 3.5 – Download Docker Image

  Task 3.6 – Run Docker Image

  Task 3.7 – Stop a Docker Container

  Task 3.8 – Run Docker with Port Mapping

  Task 3.9 – Start the Jupyter Notebook Server

  Task 3.10 – Create and Run a Test Notebook

- **Deliverable:**
  - Upload a single PDF (per group) to the intranet racó@FIB containing one slide per task. Each slide should report results (if applicable) or briefly explain how the task was completed.
  - In class (evaluation day), one student (chosen at random) will give a short *"elevator pitch"-style* presentation — clear, concise, and straight to the point.

These slides are based on the book *Supercomputing for Artificial Intelligence* (Torres, 2025).
more info: https://torres.ai/hpc4aibook/

PDF slides are freely available for students.

Teachers using this book may request the PPTX version for classroom use.