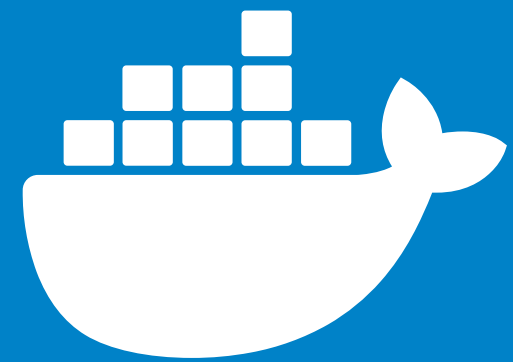


## Day 7 Agenda

# Containerization with Docker

- Recap
- Quiz for the previous Day
- Dockerfile for ML pipeline
- Packaging training/inference
- Run model container locally
- Workshop
- Quiz



# What is Docker ?

# What is Docker?

- Docker is a platform that uses containerization to run applications in isolated environments.
- It helps package code, dependencies, and runs into a single unit.
- Designed to be lightweight, portable, and consistent across environments.

# Docker Platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container.

- Containerization & Isolation
- Simultaneous Execution
- Development to Deployment Workflow
- Platform Agnostic Deployment

# What can I use Docker for?

- Fast, consistent delivery of your applications
- Responsive deployment and scaling
- Running more workloads on the same hardware

# Docker



## The Docker daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

## The Docker client

- The Docker client (docker) is the main tool used by most users to interact with Docker.
- When you run commands like docker run, the client sends them to the Docker daemon (dockerd) for execution.
- These commands are processed through the Docker API.

# Docker



## Docker Desktop

Docker Desktop is a user-friendly application available for Mac, Windows, and Linux that simplifies the process of building and sharing containerized applications and microservices.

## Docker Registries

A Docker registry is a storage system for Docker images. By default, Docker pulls images from Docker Hub, a public registry accessible to all users. You can also configure and use your own private registry.

- `docker pull` or `docker run` fetches images from the registry.
- `docker push` uploads your custom image to the configured registry.

# Containers versus virtual machines

- Without getting too deep, a VM is an entire operating system with its own kernel, hardware drivers, programs, and applications. Spinning up a VM only to isolate a single application is a lot of overhead.
- A container is simply an isolated process with all of the files it needs to run. If you run multiple containers, they all share the same kernel, allowing you to run more applications on less infrastructure.



# Docker Objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

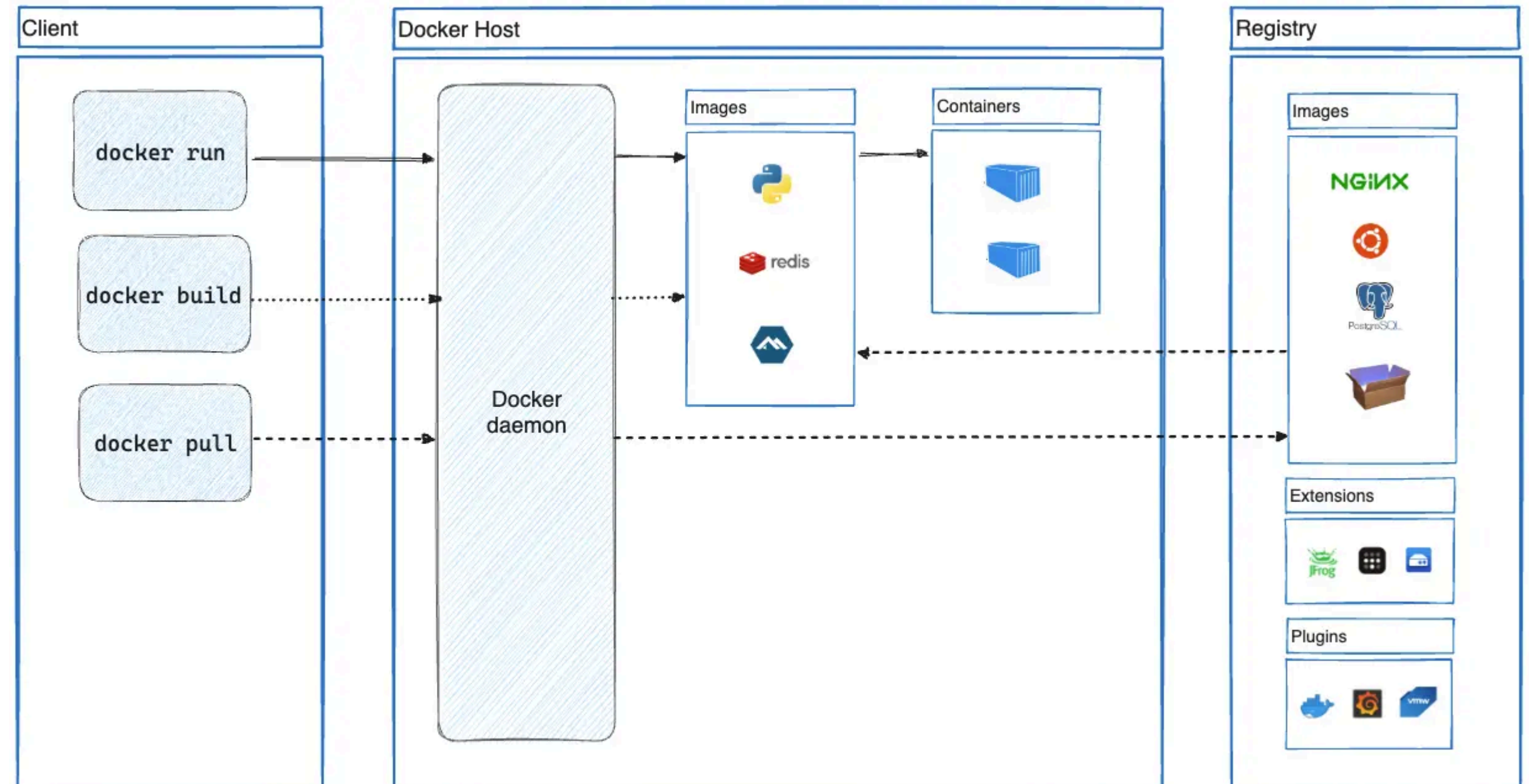
**Image:** A snapshot of a container with app + dependencies.

**Container:** A running instance of an image.

**Dockerfile:** Instructions to build a Docker image.

**Docker Hub:** Central registry for Docker images.

# Docker architecture



**Why Docker?**



# Why Docker?

- Solves the classic issue: "it works on my machine" problem.
- Ensures consistency from development to production.
- Simplifies deployment and scaling of applications.

# Docker Advantages



**Portability:** Runs the same on any system with Docker.

**Isolation:** Each container is sandboxed.

**Scalability:** Easily spin up multiple instances.

**Efficiency:** Less overhead than traditional VMs.

**Speed:** Faster boot-up and deployment times.

# Docker Setup

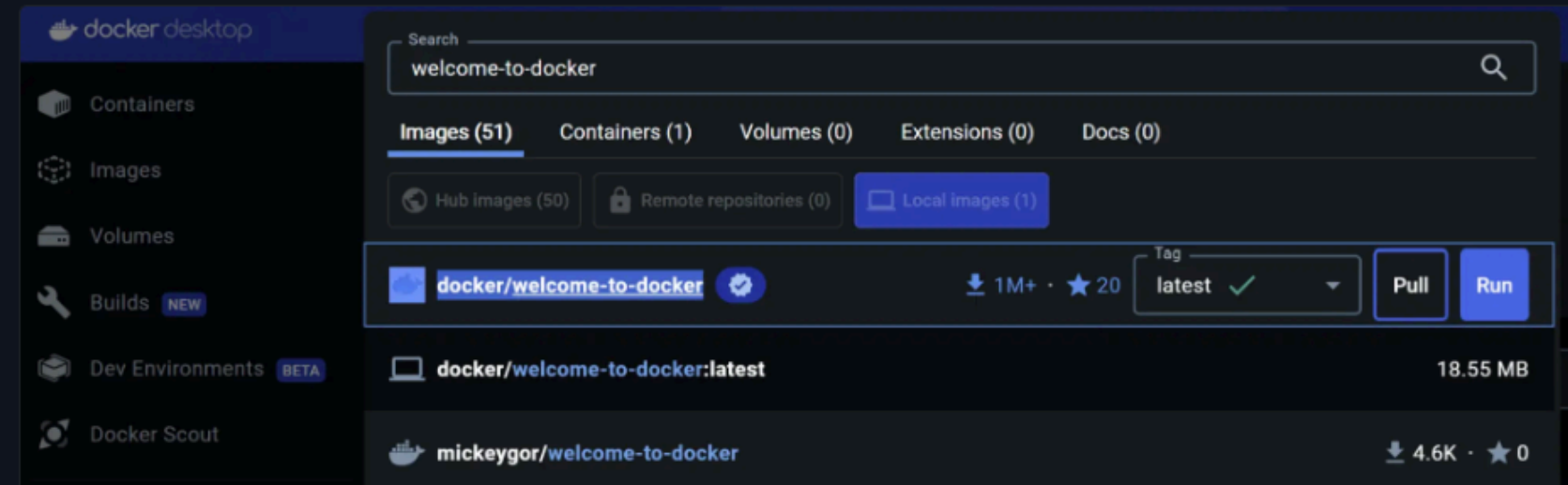
- Step 1: Download Docker Desktop (Windows/Mac) or install via package manager (Linux).
  - Go to [official docker desktop](#) To download the Docker
- Step 2: Run the installer and follow prompts.
- Step 3: Open terminal (CMD/PowerShell) and run:
  - `$ docker --version`
- Step 4: Test Docker With : `docker run hello-world`

# Try It Out

Using the GUI    Using the CLI

Use the following instructions to run a container.

1. Open Docker Desktop and select the **Search** field on the top navigation bar.
2. Specify `welcome-to-docker` in the search input and then select the **Pull** button.



3. Once the image is successfully pulled, select the **Run** button.
4. Expand the **Optional settings**.
5. In the **Container name**, specify `welcome-to-docker`.
6. In the **Host port**, specify `8080`.

# Try It Out

## View your container

You can view all of your containers by going to the **Containers** view of the Docker Desktop Dashboard.







Containers [Give feedback](#)

Container CPU usage ⓘ  
0.00% / 800% (8 CPUs available)

Container memory usage ⓘ  
6.88MB / 7.48GB

Show charts

Only show running containers

<input type="checkbox"/>	Name ↑	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	<div> <a href="#">keen_lamarr</a> 038fc64d186f</div>	<a href="#">docker/welcome-</a>	Running	0%	<a href="#">8080:80</a>  	2 minutes ago	<div>  </div>



# Try It Out



Using the GUI

Using the CLI

Follow the instructions to run a container using the CLI:

1. Open your CLI terminal and start a container by using the `docker run` command:

```
$ docker run -d -p 8080:80 docker/welcome-to-docker
```

The output from this command is the full container ID.

Congratulations! You just fired up your first container! 🎉

### View your running containers

You can verify if the container is up and running by using the `docker ps` command:

```
docker ps
```

You will see output like the following:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
a1f7a4bb3a27	docker/welcome-to-docker	"/docker-entrypoint..."	11 seconds ago	Up 11 seconds

# Docker Registry

- A Docker Registry is a storage and distribution system for Docker images.
- It allows users to push, pull, and manage container images.
- Two types:
  - Public registry (e.g., Docker Hub)
  - Private registry (self-hosted or third-party)



snapshot of the docker image



Docker Registry

# Introduction to Docker Hub



- Docker Hub is the default public registry for Docker.
- Hosted by Docker Inc.
- Provides:
  - Image storage and sharing
  - Official and verified images

# How Docker Hub Works

- Logs in to Docker Hub: `docker login`
- Tags and pushes image:
  - `docker tag my-app username/my-app:latest`
  - `docker push username/my-app:latest`
- Other users pull image:
  - `docker pull username/my-app:latest`



# Introduction to Dockerfile

# **What is Dockerfile?**

**A Dockerfile is a script containing instructions to build a Docker image.**

**Defines the environment and how the application runs.**

# Dockerfile Instructions



## FROM

- Sets the base image for the Docker image
- Must be the first instruction
  - `FROM python3:10-slim`

## WORKDIR

- Sets working directory inside container.
- Applies to RUN, CMD, COPY, ADD, etc.
  - `WORKDIR /app`

## COPY

- Copies files/folders from local machine to image
- Syntax: `COPY <source> <destination>`
  - `COPY . /app`

## RUN

- Execute commands during image build.
- Creates a new image layer
  - `RUN apt-get install jq -y`

# Dockerfile Instructions



## ENV

- Sets environment variables inside container
- Accessible during build and runtime
- Syntax: ENV <key>=<value>
  - ENV NODE\_ENV=production

## EXPOSE

- Declares the port the container listens on
- Does not actually publish the port
  - EXPOSE 3000

## CMD

- Specifies the default command when a container runs
- Can be overridden at runtime
  - CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "5005"]

## OTHER INSTRUCTIONS

- ARG: Build-time variables
- ENTRYPOINT: Main command to run
- USER: Run as specific user





# Dockerfile for ML pipeline

# ML Pipeline Dockerfile

```
1 FROM python:3.10-slim
2
3 WORKDIR /app
4
5 COPY linear_regression_model.pkl app.py requirements.txt .
6
7 RUN pip install -r requirements.txt
8
9 EXPOSE 5005
10
11 CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "5005"]
```

# ML Pipeline Dockerfile

```
FROM python:3.11-slim
WORKDIR /app

RUN apt-get update && \
    apt-get install -y curl && \
    curl -LsSf https://astral.sh/uv/install.sh | bash && \
    cp /root/.local/bin/uv /usr/local/bin/uv && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

RUN uv --version

COPY pyproject.toml ./
COPY ice_cream.py ./
COPY ice_cream.csv ./
COPY app.py ./
COPY linear_regression_model.pkl ./

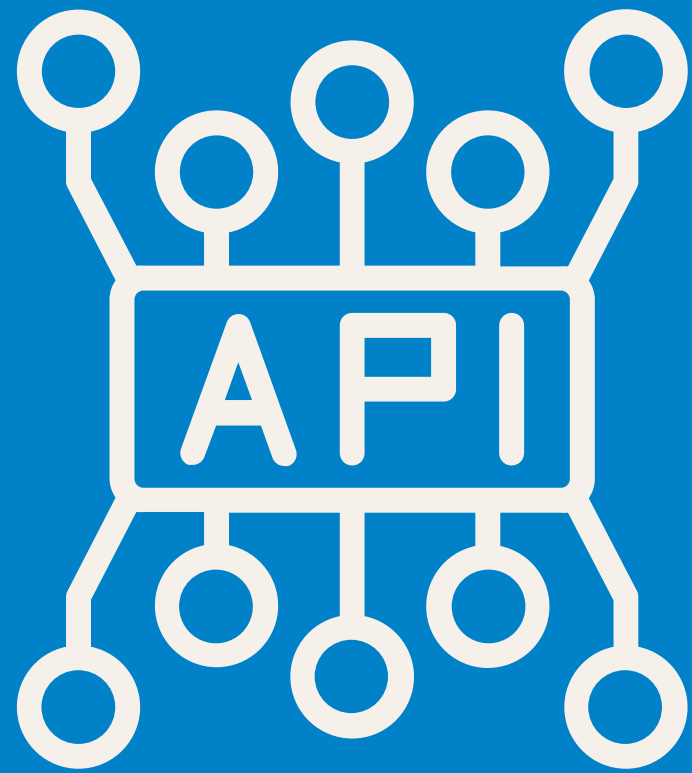
RUN uv pip install --system .

EXPOSE 8000

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

## Docker Image Creation

- To build the docker image we use command like
  - `docker build -f dockerfile -t ice_cream_sale .`



# Model Serving - Docker container

## What is Model Serving?

- Exposing a trained ML model to external applications.
- Typically done through a REST API.
- Enables real-time predictions.

## Docker Image Container

- To run the docker container we use command like
  - `docker run -it -p 5005:5005 ice_cream_sale`