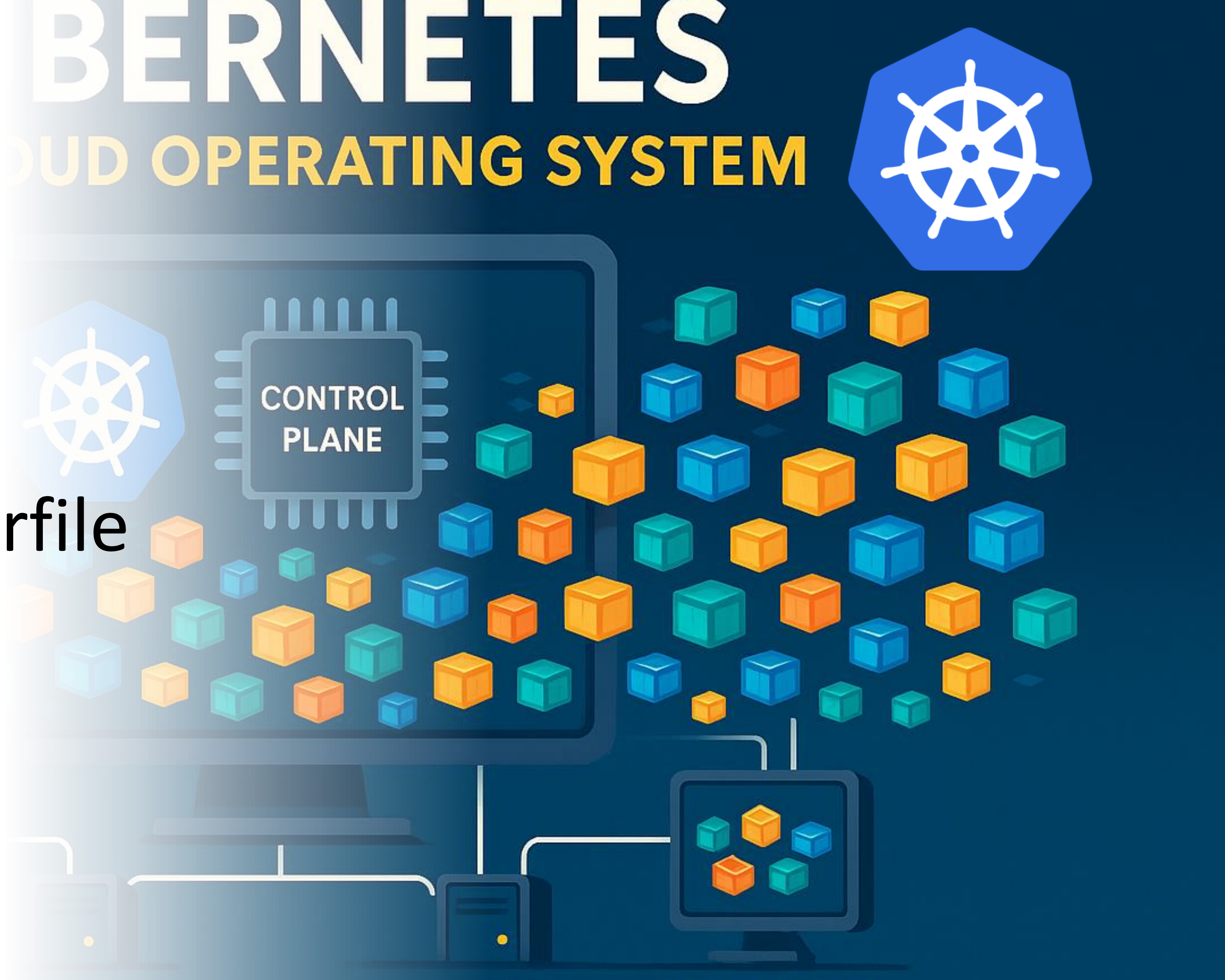
CS-UY 3913
Container
Operating Systems



Building Dockerfile

Instructor: Magdy Salem

Agenda

- Announcement
- Dockerfile Overview
- Basic Syntax
- Demo
- Lab

Announcement

- Lab/Homework
- GCP
- ChatGPT

Docker Features

- Layered image architecture
- Dockerfile & declarative builds
- Image registry integration
- Portable “build once, run anywhere”
- Networking modes (bridge, host, overlay)
- Data persistence with volumes

Docker Features

- Layered image architecture
- Dockerfile & declarative builds
- Image registry integration
- Portable “build once, run anywhere”
- Networking modes (bridge, host, overlay)
- Data persistence with volumes

Docker Command

- docker pull
- docker build
- docker run
- docker ps
- docker logs
- docker images

Dockerfile Overview

- **Dockerfile** is a simple, declarative text file that acts like a recipe for building a Docker image.
- It defines all the steps like setting a base image, installing dependencies, copying code, exposing ports, and specifying the startup process
- Making the build repeatable, version-controlled, and consistent across environments.

Why use Dockerfile

- **Reproducibility**
- **Version control**
- **Automation**
- **Portability**

Key concepts

- **Base Image:** You start with a minimal OS or language runtime (e.g., `FROM python:3.9-slim`) so you don't reinvent the wheel.
- **Build Instructions:** Lines like `RUN apt-get update && apt-get install -y git` or `COPY ./app /usr/src/app` layer in packages, code, and configuration.
- **Metadata & Defaults:** ENV and ARG set build-time or runtime variables; EXPOSE documents which ports your app will listen on; ENTRYPOINT/CMD define the default process.
- **Layered Image Model:** Each instruction creates a new layer; smart caching means unchanged steps are skipped on rebuild, speeding development iterations.

How it fits into workflow

- **Local development:** Build (`docker build`) and run (`docker run`) images to test changes quickly.
- **CI/CD pipelines:** Automate image builds, run tests inside containers, and push to a registry.
- **Production deployments:** Orchestrators like Kubernetes pull your immutable images, guaranteeing consistency across staging and prod environments.

Dockerfile Syntax

- FROM
- RUN
- CMD
- ENTRYPOINT
- COPY
- ADD
- WORKDIR

Dockerfile Syntax

`FROM`

- **Purpose:** Sets the base image for the Docker image.
- **Usage:** Always the first line in a Dockerfile (except comments).
- **Example:**

```
FROM ubuntu:22.04
```

Dockerfile Syntax

RUN

- **Purpose:** Executes commands inside the container during image build.
- **Usage:** Used to install packages, configure software, etc.
- **Example:**

```
RUN apt-get update && apt-get install -y nginx
```

Dockerfile Syntax

ENTRYPOINT

- **Purpose:** Defines the executable that will always run in the container.
- **Usage:** Provides more control over container startup behavior.
- **Example:**

```
ENTRYPOINT ["python", "app.py"]
```

Dockerfile Syntax

3. CMD

- **Purpose:** Sets the default command to run when the container starts.
- **Usage:** Only one CMD is allowed per Dockerfile (last one wins).
- **Example:**

```
CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile Syntax

COPY

- **Purpose:** Copies files/directories from local context into the image.
- **Usage**

```
COPY ./src /app
```

Dockerfile Syntax

ADD

- **Purpose:** Like COPY, but with extra features.
- **Usage:**

```
ADD https://example.com/file.zip /tmp/file.zip
```

Dockerfile Syntax

`WORKDIR`

- **Purpose:** Sets the working directory for following instructions.
- **Usage:**

```
WORKDIR /app
```

Dockerfile Syntax

```
FROM python:3.11-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Dockerfile Syntax

Read more about the concept at Docker Docs

https://docs.docker.com/build/concepts/dockerfile/

Demo

LAB

Lab Github link here