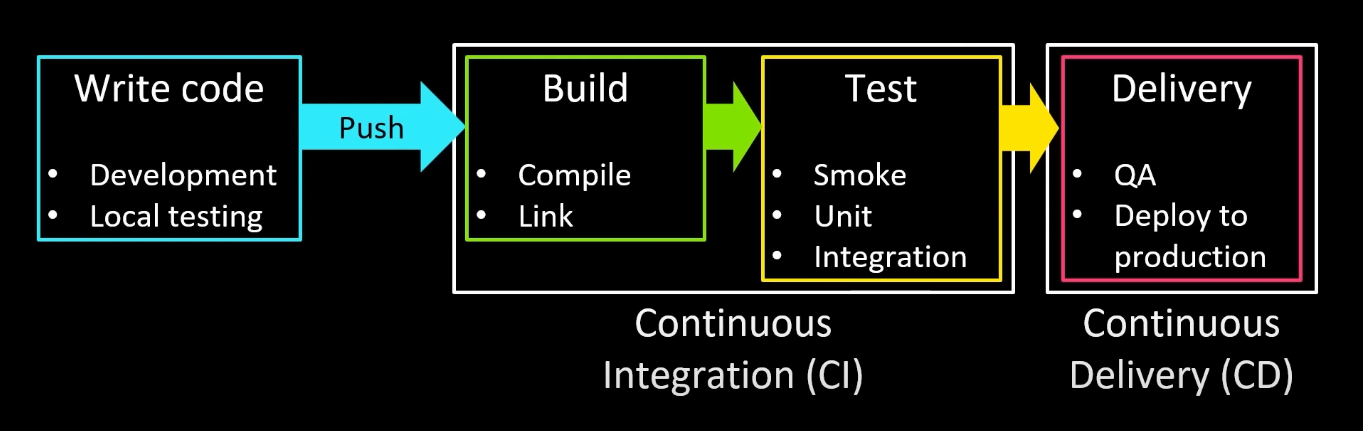
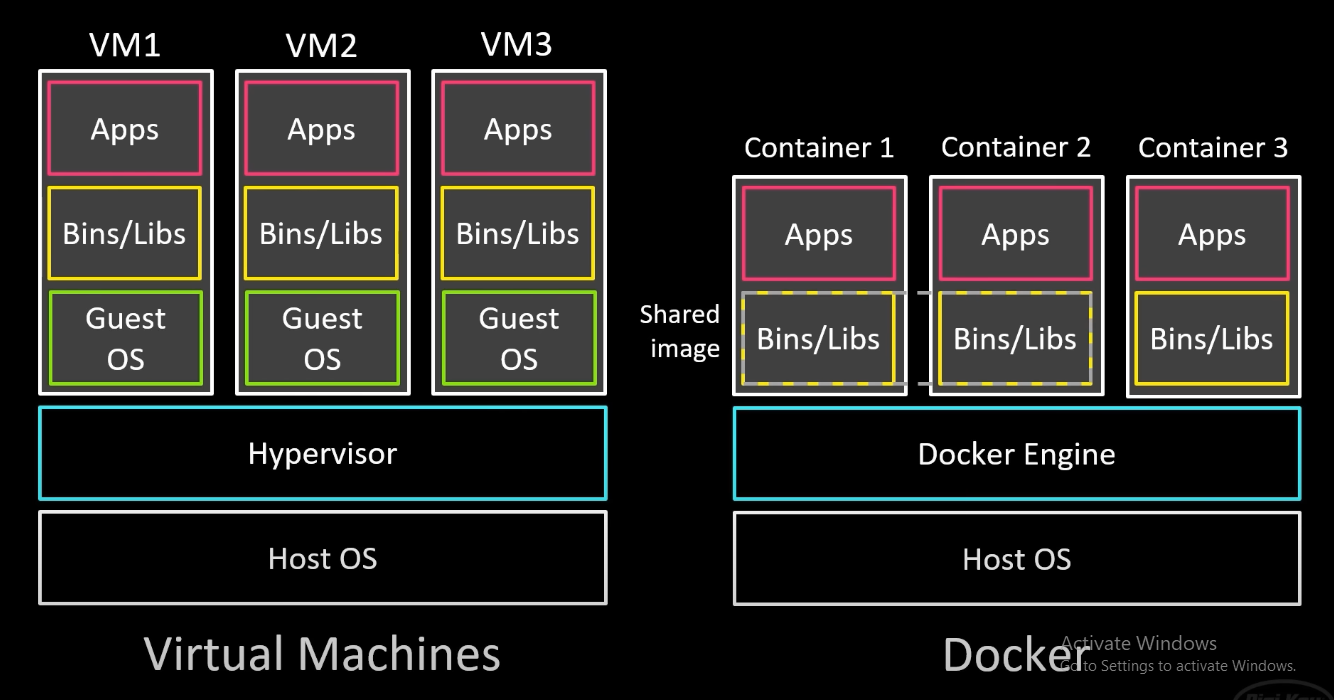
**Docker for Embedded**

1. **CI/CD –** continuous Integration/ continuous deployment

* Process of automating the testing and deployment of a software project.
* CICD pipeline****

1. **Docker Intro:**

* Docker is a tool that lets you package your code and all its dependencies into a single container so it runs the same everywhere, on any computer.
* Docker is a platform that runs applications in virtualized instances known as “containers.” Containers use a shared set of binaries and libraries as well as low-level drivers from the host operating system (OS) to run applications. As a result, applications can be made portable and scalable, regardless of the host OS.

1. **Docker Installation and setup:**

* Enter <https://www.docker.com/> and click **Download Docker Desktop** ( If you have an AMD or Intel processor (x64), you should download the AMD64 version. If you have an ARM-based processor, you should download the ARM64 version)

1. **Creating a simple Docker project**

* Create a project folder wherever necessary “hello\_docker” and $cd hello\_docker
* Create a **Dockerfile** and open it in any Editor(vs code)

# Fetch ubuntu image

FROM ubuntu:22.04

# Install Python on image

RUN \

    apt-get update && \

    apt-get install -y python3 && \

    apt-get install -y build-essential

# Create a directory for our tests

RUN mkdir /tests

# Copy in our Python script

COPY test.py /tests/test.py

# Copy in our program under test

COPY main.c /tests/main.c

# Command that will be invoked when container starts

ENTRYPOINT ["python3", "tests/test.py"]

* After creating Dockerfile in the same directory, open cmd or terminal and run the command **> docker build -t my-image .**
* To listout all built images **> docker images** and you’ll be able find the created image in docker desktop -> images
* Create new Container using command **> docker create -i -t --entrypoint="/bin/bash" --name my-container my-image**
* *-i* means “interactive mode,” which keeps the container running to allow for interactions.
* *-t* means “tty” to add a pseudo terminal for command line interactions with the container.
* *--entrypoint* overrides the entrypoint in the Dockerfile. We do this for this example so you can see how to log in to a container without running the test.py script.
* *--name* gives a name to the container. If you don’t specify a name, Docker will randomly assign a name by default.
* To list all containers **> docker container ls -a**
* To start a container **>** **docker start -i my-container**
* To add another file test2.py **> docker cp test2.py my-container:/tests/test2.py**
* Now to start our container > docker **start** -i my-**container**
* To get out of the container(my-container) type **>exit**
* To remove container **> docker rm my-container**
* To remove image **> docker rmi my-image**
* To run a container from the image my\_image, and automatically clean it up after it stops **>docker run –rm my\_image**
* Reference
  + Docker images: https://hub.docker.com/
  + Dockerfile: <https://docs.docker.com/reference/dockerfile/>
  + Youtube: <https://youtube.com/playlist?list=PLEBQazB0HUyR00bgoctk-5uZoOrbmPASC&si=SDyVC7qhEzAGdHvi>

**5. Why Is the Dockerfile Inside the .devcontainer/ Folder?**

* This is by design, for VS Code Dev Containers.
* VS Code uses the .devcontainer/ folder as a **self-contained config directory** for development containers.
* When you open a folder in VS Code, it looks for .devcontainer/devcontainer.json.
* That .json file points to the Dockerfile (or image) and describes how to set up the container for your **development environment** (not deployment).
* **Key reason**: Keeping it in .devcontainer/ makes it clear the Dockerfile is for **development only**, not production.

**devcontainer.json** - It defines how VS Code should build and run your development container, including tools, settings, and extensions. This file tells VS Code:

| **Field** | **Purpose** |
| --- | --- |
| "name" | Name of the dev container |
| "build" | Path to the Dockerfile or Docker Compose config |
| "extensions" | Auto-install these VS Code extensions in the container |
| "settings" | VS Code-specific settings (like terminal shell) |
| "postCreateCommand" | Script/command to run after setting up container |
| "mounts" (optional) | Mount extra volumes into the container |
| "remoteUser" | User VS Code should use inside the container |

**Which Is Better Practice?**

| **Approach** | **When to Use** | **Notes** |
| --- | --- | --- |
| Dockerfile inside .devcontainer/ | **Best for VS Code Dev Containers** | Keeps dev environment config isolated and portable |
| Dockerfile in project root | **Better for production containers** (e.g., backend APIs, deployment) | Keeps deployment concerns separate from dev tools |
| Both | Use .devcontainer/Dockerfile for **dev**, and a separate one at root for **prod** | Common in serious projects (e.g., React + Node apps, microservices) |

6. **Docker Core Commands Cheatsheet**

| **Command** | **Purpose** |
| --- | --- |
| docker version | Show Docker version info |
| docker pull <image> | Download image from registry |
| docker images | List local images |
| docker build -t <name> . | Build image from Dockerfile |
| docker run <image> | Run container from image |
| docker run -d -p 80:80 <image> | Run in background, map ports |
| docker exec -it <container> bash | Open interactive shell in container |
| docker ps | List running containers |
| docker ps -a | List all containers |
| docker stop <container> | Stop container |
| docker rm <container> | Remove container |
| docker rmi <image> | Remove image |
| docker logs <container> | View logs from container |
| docker cp <src> <container>:<dst> | Copy file into container |
| docker volume ls | List volumes |
| docker system prune -a | Remove stopped/unused containers, images, volumes |
| docker login | Login to registry |
| docker logout | Logout from registry |

**Essential Dockerfile Instructions**

| **Command** | **Description** | **Example** |
| --- | --- | --- |
| FROM | Base image | FROM ubuntu:22.04 |
| WORKDIR | Set working directory | WORKDIR /app |
| COPY | Copy files/directories | COPY . /app |
| ADD | Like COPY, can extract and fetch URLs | ADD code.tar.gz /src/ |
| RUN | Execute command when building | RUN apt update |
| CMD | Default container command | CMD ["python", "main.py"] |
| ENTRYPOINT | Main executable for container | ENTRYPOINT ["npm"] |
| ENV | Set environment variables | ENV HOST=0.0.0.0 |
| EXPOSE | Document container port | EXPOSE 8080 |
| VOLUME | Declare mount point for data volume | VOLUME /data |
| ARG | Build-time variable | ARG VERSION=1.0 |
| USER | Set user | USER appuser |
| LABEL | Add image metadata | LABEL maintainer="dev@site.com" |
| ONBUILD | Trigger command when base for another image | ONBUILD COPY . /src |