Docker Workshop

Jaime Canizales

City University of New York

jaime.canizales@hunter.cuny.edu

January 7, 2025

Overview

What is Docker?

Problem Statement

Docker is a platform for creating, running, and managing containers—lightweight, portable environments that package an application and its dependencies. Containers ensure the app runs consistently across different systems. Unlike virtual machines, containers share the host OS, making them faster and more efficient. Docker simplifies development, testing, and deployment, with tools like Docker Hub for sharing images and orchestration support for scaling.

Why use Docker?

- **Consistency:** Ensures that applications run the same way regardless of the host environment (development, testing, production).
- **Portability:** Docker containers can run on any system with Docker installed, including on-premises, cloud, or hybrid setups.
- **Isolation:** Containers encapsulate applications and their dependencies, avoiding conflicts between different environments or applications.
- **Efficiency:** Lightweight compared to virtual machines (VMs), as they share the host OS kernel and require fewer resources.
- **Scalability:** Makes it easy to scale applications horizontally by running multiple container instances.

Why use Docker?(Cont.)

- **Faster Development and Deployment:** Speeds up workflows with instant container creation and simplified testing across environments.
- **Microservices Architecture:** Ideal for breaking down applications into smaller, manageable, and independently deployable services.
- **Reproducibility:** Developers can create a predictable environment using Dockerfiles and Docker Compose, reducing "it works on my machine" issues.
- **DevOps Integration:** Fits seamlessly into CI/CD pipelines for automated testing, integration, and deployment.
- **Community and Ecosystem:** Extensive support with prebuilt images available on Docker Hub and integration with tools like Kubernetes for orchestration.

Installing Docker

- Docker install link

Docker Technical Terminology

- **Docker Client(CLI)**: The command-line interface used by users to interact with the Docker Daemon.
- **Docker Daemon**: The core background process that manages Docker objects like containers, images, networks, and volumes. It listens for API requests and communicates with the CLI and other components.
- **Docker Engine**: Refers to the system containing both Docker CLI and Docker Daemon
- **Docker Desktop**: Refers to the system containing both Docker Engine and a Docker graphical user interface(gui)
- **Docker Registry**: is a centralized storage and distribution system for Docker images.(version control for docker images) (we will use Dockerhub)

Docker Container

Docker Container

A Docker Container is a lightweight, portable runtime environment created from a Docker image. It runs an application and its dependencies in isolation, sharing the host system's kernel for efficiency.

Docker Image

Docker Image

A Docker Image is a lightweight, immutable blueprint for creating containers. It includes everything needed to run an application: code, dependencies, libraries, and settings. Images are built in layers, stored in registries, and used to launch containers.(show example of a dockerfile)

Dockerfile

Dockerfile

A Dockerfile is a text file that contains instructions to build a Docker image. Each line in the Dockerfile specifies a step in the build process.(Go over image)

Basic Docker Commands For Start Up

- docker build -t image-name . (creates an image named image-name from a dockerfile inside the cwd)
- docker run –name test-container -it –rm image-name (creates a containe named test-container from an image)
- docker ps -a (Shows container ids)
- docker exec -it container-id bash (opens another terminal in a current container)

Basic Docker Commands for Clean Up

- docker rm container-id (deletes a single container)
- docker container prune –force (deletes all unused containers)
- docker image prune –force (deletes all unused or dangling images)

Dev Containers

- Docker Dev Containers are containerized development environments created using Docker. They package tools, dependencies, and configurations, providing consistent, isolated setups for coding across machines. Often used with tools like VS Code for seamless integration.
- Basically you can use vscode and a json file to specify configuration and run containers without having to use the command line
- You can even include remote images in devcontainers.json by just adding one line. e.g. "image" : "matimoreyra/opencv:latest", (image with opencv for c++)

DockerHub

DockerHub

Docker Hub is a cloud-based repository for building, sharing, and managing Docker images. Instead of storing and building image locally, you can build them on the cloud instead! Or you can create contianers from pre-existing images on DockerHub! DockerHub is one of the many examples of a docker registry(a very popular one).

DockerHub Commands

- docker commit container-id repository-name:tag (to save changes on container to remote image)
- docker tag old-repository-name:tag dockerhub-username/new-repository-name:tag (add a tag to an image and/or give an image an alias)
- docker push dockerhub-username/repository-name:tag (update the remote copy of the image on the cloud)
- docker pull repository-name:tag (make a local copy of a remote image from the cloud)

Check out the Docker Examples provided in this repo

- authentication for node and psql image
- opencv image
- python stuff added to image
- Haskell image
- Extension name: latex workshop james yu, then check out file :./.vscode/settings.json to see how to include the docker image for latex