Crash Course on Docker

(with Visual Studio Code Integration)

1. Introduction

Docker is an open-source platform that allows developers to automate the deployment of applications inside lightweight, portable containers. A **container** bundles an application with all its dependencies—libraries, configurations, and system tools—ensuring that it runs consistently across environments.

With Docker, you can:

- Package your application and dependencies into a single portable unit.
- Run identical environments on development, testing, and production systems.
- Simplify setup and eliminate the "it works on my machine" problem.

Docker has become a cornerstone of modern DevOps, cloud computing, and software development workflows.

2. Core Concepts

2.1 Containers

A **container** is an isolated runtime environment for your application. It includes everything needed to run—code, libraries, configuration files—but shares the host system's kernel, making it lightweight and fast.

2.2 Images

An **image** is a blueprint for a container. It contains the instructions needed to create a running container instance. Images are typically built from a `Dockerfile`.

2.3 Dockerfile

A **Dockerfile** is a text file containing the commands needed to assemble a Docker image. It defines what software to install, what files to copy, and what commands to run when the container starts.

Example `Dockerfile`:

```
# Base image
FROM python:3.11

# Set working directory
WORKDIR /app

# Copy application files
COPY . /app
```

```
# Install dependencies
RUN pip install -r requirements.txt

# Run the application
CMD ["python", "app.py"]
```

2.4 Docker Daemon and CLI

- **Docker Daemon (`dockerd`)**: The background service that manages containers.
- **Docker CLI (`docker`)**: The command-line tool for communicating with the Docker Daemon.

2.5 Docker Hub

A public registry of Docker images. Developers can **pull** existing images (e.g., Ubuntu, MySQL, Python) or **push** custom images for others to use.

---

3. Installing Docker

3.1 On Windows

1. Download Docker Desktop from https://www.docker.com/products/docker-desktop.
2. Enable **WSL 2** (Windows Subsystem for Linux).
3. Run the installer and restart your computer.
4. Launch Docker Desktop and verify installation:

```
docker --version
```

3.2 On Linux

Install via your package manager:

```
sudo apt update
sudo apt install docker.io
sudo systemctl enable --now docker
```

3.3 On macOS

Install **Docker Desktop for Mac** and verify with:

```
docker --version
```

---

4. Basic Docker Commands

Command	Description
`docker run hello-world`	Run a test container
`docker ps`	List running containers
`docker ps -a`	List all containers (including stopped ones)
`docker images`	List available images
`docker pull ubuntu`	Download an image from Docker Hub
`docker build -t myapp .`	Build an image from the current directory
`docker run -it myapp`	Run a container interactively
`docker stop <container_id>`	Stop a running container
`docker rm <container_id>`	Remove a container
`docker rmi <image_id>`	Remove an image
`docker exec -it <container_id> /bin/bash`	Open a shell inside a container

Example:

```
docker run -it ubuntu bash
```

Starts a new Ubuntu container and opens a terminal inside it.

---

5. Docker Workflow Overview

1. **Write your application** in your preferred language.
2. **Create a Dockerfile** describing the environment.
3. **Build the image** using `docker build`.
4. **Run containers** from the image using `docker run`.
5. **Share images** via Docker Hub or private registries.

This workflow allows developers to run consistent environments across different systems with minimal setup.

---

6. Docker Compose

For applications that rely on multiple containers (e.g., a web app + database), **Docker Compose** simplifies orchestration.

Example `docker-compose.yml`:

```
version: '3'
services:
  web:
```

```
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
```

To run:

```
docker-compose up
```

This starts both the web application and database containers simultaneously, with automatic networking.

---

7. Managing Data with Volumes

Containers are ephemeral—data disappears when they are removed. To persist data, use **volumes**.

Example:

```
docker run -d -v /my/local/folder:/data myimage
```

This mounts a local folder into the container, allowing data persistence and easy file sharing.

---

8. Networking in Docker

Docker automatically isolates containers in networks, but you can connect them:

- **Bridge network (default)**: Containers can communicate using internal IPs.
- **Host network**: Container shares the host's networking stack.
- **Custom networks**: Created via `docker network create mynetwork`.

Example:

```
docker network create app-network
docker run -d --network app-network --name web nginx
docker run -d --network app-network --name db postgres
```

Now `web` and `db` can communicate via their container names.

---

9. The Docker Extension in Visual Studio Code

The **Docker Extension for VS Code** integrates Docker directly into your development workflow. It provides a graphical interface to manage containers, images, volumes, and Docker Compose projects.

9.1 Installation

1. Open VS Code.
2. Go to the **Extensions** panel (`Ctrl+Shift+X`).
3. Search for **Docker**.
4. Install the official Microsoft extension.

9.2 Features

Feature	Description
Explorer View	View running containers, images, volumes, and networks from a sidebar.
Interactive Management	Start, stop, and remove containers directly from VS Code.
Dockerfile Intellisense	Syntax highlighting, autocompletion, and linting for Dockerfiles.
Compose Support	Run and debug multi-container applications from `docker-compose.yml`.
Logs and Terminals	View container logs and open terminal sessions without leaving VS Code.
Context Menu Actions	Right-click containers or images for quick commands like “Inspect” or “Attach Shell.”
Integration with Remote Development	Works with VS Code Remote Containers for developing inside a containerized environment.

9.3 Example Workflow in VS Code

1. Open your project folder.
2. Create a `Dockerfile` and/or `docker-compose.yml`.
3. The Docker extension automatically detects them.
4. Use the Docker sidebar to:
    - Build your image (`Right-click Dockerfile → Build Image`).
    - Run a container (`Right-click Image → Run`).
    - Attach a terminal to a container.
5. To debug Python or Node.js inside a container:
    - Click “Add Docker Files to Workspace” from the Command Palette (`Ctrl+Shift+P`).
    - Choose the runtime (Python, Node.js, etc.).
    - VS Code generates configuration files for containerized debugging.

This integration allows full container lifecycle management without using the command line.

---

10. Developing Inside Containers (VS Code Remote Containers)

The **Dev Containers** feature (previously Remote - Containers) lets you open your entire project inside a containerized environment. This ensures everyone on your team develops in the same setup.

Steps

1. Install the **Dev Containers** extension in VS Code.
2. Create a `.devcontainer/devcontainer.json` file:

```
{
  "name": "Python Dev Environment",
  "image": "python:3.11",
  "extensions": ["ms-python.python"],
  "settings": { "terminal.integrated.shell.linux": "/bin/bash" }
}
```

3. Run “Reopen in Container” from the Command Palette.

VS Code will build and open your workspace inside the container, with full IntelliSense, debugging, and terminal access.

---

11. Best Practices

1. **Use official base images** when possible for security and stability.
2. **Keep Dockerfiles small** by combining commands and cleaning up temporary files.
3. **Use** `.dockerignore` to exclude unnecessary files from builds.
4. **Tag images properly** (`myapp:1.0`, not just `latest`).
5. **Regularly prune** unused images and containers:

```
docker system prune
```

6. **Avoid running as root** inside containers for security.
7. **Use multi-stage builds** to keep final images lightweight.

---

12. Common Issues

Issue	Cause	Solution

Issue	Cause	Solution
Container exits immediately	No long-running process	Use `-it` or specify a `CMD`
Port not accessible	Port not mapped	Use `-p host:container`
Build slow	Cache invalidated too early	Order Dockerfile commands properly
Permission errors	Host file system restrictions	Adjust mounted volume permissions
Out of disk space	Unused images or volumes	Run `docker system prune -a`

13. Summary

Docker revolutionizes software deployment by providing a consistent, isolated, and reproducible environment for applications. Its key advantages include:

- Rapid, consistent setup.
- Simplified deployment and scaling.
- Easy collaboration between developers.
- Compatibility with cloud and CI/CD pipelines.

The **Docker Extension in Visual Studio Code** enhances productivity by integrating Docker management directly into your IDE. With it, you can build, run, debug, and inspect containers without leaving your development environment.

By mastering Docker and its VS Code integration, you gain control over every stage of the development lifecycle—from local testing to production deployment—using the same tools, images, and configurations everywhere.