# A Comprehensive Guide to Docker: The Modern Containerization Tool

**Introduction to Docker**

Docker is a powerful, open-source platform designed to simplify the process of building, running, testing, and deploying applications using containerization. It provides an efficient way to develop distributed applications by packaging software into standardized units known as containers.  
  
- Written in: Go Language  
- Type: Open-source  
- Developed by: Solomon Hykes and Sebastian Pahl  
- Initial Release: March 2013  
  
Docker allows applications to be isolated in containers, enabling more efficient resource utilization and consistency across different environments. While Docker can be installed on any operating system, the Docker Engine natively runs on Linux distributions. This container-based virtualization is often referred to as OS-level virtualization.

**Virtual Machines vs Containers**

Virtual Machines (VMs):  
A single physical server can be logically divided into multiple virtual machines using a hypervisor. Each VM includes a full OS and can run applications independently.  
  
Example: Microservices for Roboshop Project using VMs:  
- Cart Service -> VM  
- User Service -> VM  
- Catalogue Service -> VM  
  
Containers:  
Unlike VMs, containers share the host OS kernel and isolate application processes. Multiple containers can run inside a single VM, making better use of system resources.  
  
Example: Roboshop Project using Containers:  
- One VM hosts multiple containers:  
 - Cart -> Container  
 - User -> Container  
 - Catalogue -> Container

**Installing Docker on Amazon Linux (EC2)**

Step 1: Launch an EC2 Instance  
Go to AWS Console and create a new EC2 instance.  
  
Step 2: Update Package Cache  
$ sudo yum update -y  
  
Step 3: Install Docker  
For Amazon Linux 2:  
$ sudo amazon-linux-extras install docker  
  
For Amazon Linux 2023:  
$ sudo yum install -y docker  
  
Step 4: Start Docker Service  
$ sudo service docker start  
  
Step 5: Add User to Docker Group  
$ sudo usermod -a -G docker ec2-user  
  
Step 6: Reconnect to Your Instance  
Log out and log back in to apply group changes.  
  
Step 7: Verify Docker Installation  
$ docker ps

**Understanding Docker Components**

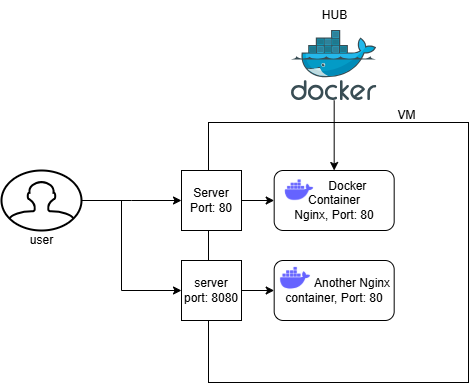
Image:  
An image is a lightweight, standalone package that includes everything needed to run a piece of software: code, runtime, libraries, and environment variables.  
  
Container:  
A container is a runnable instance of an image.  
  
Docker Hub:  
A cloud-based registry where Docker users can store and share container images.

**Common Docker Commands**

docker images - List all images  
docker pull <image>:<version> - Download image from Docker Hub  
docker create <image> - Create container from image  
docker ps -a - List all containers (including stopped ones)  
docker ps - List running containers  
docker start <container> - Start an existing container  
docker stop <container> - Stop a running container  
docker run <image> - Pull, create, and start a container  
docker rm $(docker ps -a -q) - Remove all containers  
docker rmi $(docker images -q) - Remove all images  
docker rmi <image-id> - Remove specific image

**Port Forwarding in Docker**

Each container, like a server, has its own port range (0-65535). To expose a container's service to the host machine:  
  
docker run -p <host-port>:<container-port> <image>  
  
Foreground Mode:  
docker run -p 8080:80 nginx  
  
Background Mode:  
docker run -d -p 8080:80 nginx



**Useful Docker Commands**

Access container shell:  
docker exec -it <container-id> bash  
  
Set custom container name:  
docker run -d --name mynginx -p 8080:80 nginx  
  
View container logs:  
docker logs <container-id>

**Interview Questions on Docker**

1. How can you do port forwarding in Docker?  
 - Foreground: docker run -p <host-port>:<container-port> <image>  
 - Background: docker run -d -p <host-port>:<container-port> <image>  
  
2. How do you access a running container?  
 - docker exec -it <container-id> bash  
  
3. How to list all Docker containers including stopped ones?  
 - docker ps -a  
  
4. How do you remove all Docker containers?  
 - docker rm $(docker ps -a -q)  
  
5. What is the difference between Docker image and container?  
 - Image is a static package; container is the running instance of that image.

**Conclusion**

Docker revolutionizes how modern applications are built and deployed. Its lightweight and consistent containerized approach simplifies development workflows and enhances resource efficiency. Whether you're building microservices or deploying production-ready applications, Docker is a must-have tool in every DevOps engineer's toolkit.

Creating Custom Docker Images Using Dockerfile: A DevOps Guide

## Introduction

Docker has revolutionized the way modern applications are built and deployed. At its core, Docker uses images—lightweight, standalone, and executable packages that include everything needed to run a piece of software.  
  
In this blog, we'll explore how to create custom Docker images using a Dockerfile, understand key instructions like FROM, RUN, and CMD, and walk through practical commands used in real-world DevOps environments.

## What is a Dockerfile?

A Dockerfile is a declarative script that contains a series of instructions for Docker to build a custom image. Instead of configuring your container manually every time, a Dockerfile automates the process and ensures consistency.  
  
✅ Dockerfile = Declarative way of building Docker images  
  
With a properly written Dockerfile, you can:  
- Build consistent environments  
- Automate provisioning using CI/CD tools

### Common Dockerfile Instructions

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| FROM | Specifies the base image (e.g., Ubuntu, Alpine, Nginx) |
| RUN | Executes commands at image build time to install/configure packages |
| CMD | Specifies the command to run at container startup |

### Interview Insight: RUN vs CMD

|  |  |
| --- | --- |
| **RUN** | **CMD** |
| Executes during image build | Executes when container starts |
| Used for installing packages, setting environment | Used to define the default app/process |
| Appears in image history | Doesn’t change image but affects container behavior |

## Creating a Docker Image: Step-by-Step

Let’s say you want to build a custom Nginx image using your own Dockerfile.

1. Create a Dockerfile

FROM nginx:latest  
RUN apt-get update && apt-get install -y curl  
CMD ["nginx", "-g", "daemon off;"]

The CMD must keep the container running; otherwise, Docker will shut it down once the main process exits.

2. Build the Docker Image

docker build -t mynginx:1.0 .

3. Run the Docker Container

docker run -d --name mynginx -p 8080:80 mynginx:1.0

Access your app at http://<your-EC2-public-IP>:8080

4. Check Container Logs

docker logs <container-id or container-name>

## Build Images from Git-Tracked Code (VS Code → GitHub → Docker)

Step 1: Push Project Code from VS Code to GitHub

git init  
git add .  
git commit -m "initial commit"  
git remote add origin https://github.com/yourusername/your-repo.git  
git branch -M main  
git push -u origin main

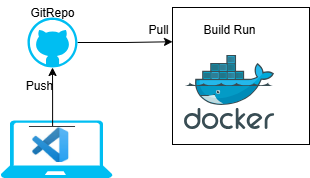
Step 2: SSH into Your EC2 (or Docker Host) and Clone the Repo

git clone https://github.com/yourusername/your-repo.git  
cd your-repo

Step 3: Build Docker Image from Cloned Code

docker build -t custom-app:1.0 .

Step 4: Run Container from Your Custom Image



docker run -d --name custom-app -p 8080:80 custom-app:1.0

## Note: systemctl Doesn’t Work Inside Containers

systemctl and other init system services are not designed to work within containers. Containers are meant to run a single foreground process, unlike full VMs that run background services managed by a system manager.  
  
If you need multiple services inside a container, consider:  
- Using supervisord  
- Or better: Break your services into multiple containers (microservices pattern)

## Conclusion

Creating custom Docker images using a Dockerfile is an essential DevOps skill. By learning how to push source code from VS Code to GitHub, pull it into your server, and build Docker images from it, you ensure a smooth and scalable development pipeline. Understanding the differences between RUN and CMD, and using them correctly, is key to building reliable containers for your apps.