Get Started Docker

Introduction

Docker is an open-source platform that enables the isolation and deployment of applications in containers. Containers are lightweight, portable, and self-contained units that include everything an application needs to run.

In this tutorial, you will learn how to:

- Install Docker,
- Create and manage containers,
- Build and store Docker images,
- Orchestrate containers using Docker Compose,
- Set up Docker networks,
- Use pre-built Docker containers from Docker Hub.

---

1. Installing Docker

1.1 For Ubuntu/Debian-based Systems:

1. Update your package list:

   ```
   sudo apt-get update
   ```

2. Install the required packages:

   ```
   sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
   ```

3. Add Docker's official GPG key:

   ```
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
   ```

4. Set up the Docker repository:

   ```
   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
   ```

5. Install Docker:

   ```
   sudo apt-get update
   sudo apt-get install docker-ce
   ```

6. Verify the installation:

   ```
   docker --version
   ```

1.2 For Windows:

Docker Desktop is the recommended method for running Docker on Windows. Docker Desktop includes everything you need to build and run Docker containers.

1.2.1 System Requirements:

- **Windows 10 Pro, Enterprise, or Education** (Build 19041 or later) or **Windows 11**.
- Hyper-V and Containers Windows features must be enabled.
- WSL 2 (Windows Subsystem for Linux) should be installed for better performance and compatibility.

1.2.2 Installation Steps:

1. **Download Docker Desktop:** Go to the Docker Desktop download page and download the installer.
2. **Run the installer:** Double-click the downloaded installer file and follow the on-screen instructions.
3. **Enable WSL 2:** During installation, ensure that WSL 2 is selected as the default backend for Docker.
4. **Complete installation:** After the installation finishes, Docker Desktop will start automatically. You may need to log out and log back in to complete the installation.
5. **Verify the installation:** Open PowerShell or the command prompt and run:

   ```
   docker --version
   ```

1.2.3 Post-Installation:

- **Enable Hyper-V and Containers:** To enable Hyper-V and Containers, open PowerShell as Administrator and run:

```
dism.exe /online /enable-feature /featurename:Microsoft-Hyper-V-All /all /norestart
dism.exe /online /enable-feature /featurename:Containers-DisposableClientVM /all /norestart
```

- **Configure WSL 2:** If you don't have WSL 2 installed, follow the official WSL 2 installation guide to set it up.

---

2. Basic Docker Commands

2.1 Running Your First Docker Container

To run a simple "Hello World" container:

For **Linux/macOS**:

```
docker run hello-world
```

For **Windows (PowerShell)**:

```
docker run hello-world
```

This command pulls the `hello-world` Docker image from Docker Hub (if not already present), starts a container, and runs a simple program that outputs "Hello from Docker."

2.2 Checking Container Status

To view running containers:

```
docker ps
```

To view all containers (including stopped ones):

```
docker ps -a
```

2.3 Stopping and Removing Containers

To stop a container:

```
docker stop <CONTAINER_ID>
```

To remove a container:

```
docker rm <CONTAINER_ID>
```

---

3. Building a Docker Image

Docker images are templates used to create containers. You can build your own images using a `Dockerfile`.

3.1 A Simple Dockerfile

Create a `Dockerfile` as shown before:

```
# Choose a base image
FROM python:3.8-slim

# Set the working directory
WORKDIR /app

# Copy all files to the working directory
COPY . /app

# Install dependencies
RUN pip install -r requirements.txt

# Define the command to run the application
CMD ["python", "app.py"]
```

3.2 Building a Docker Image

You can build the image using the following command:

```
docker build -t my-python-app .
```

Explanation of Parameters:

- **`build`** : This tells Docker to create (build) an image based on the instructions in the `Dockerfile` .
- **`-t my-python-app`** : The `-t` flag tags the image with a name (`my-python-app`). This is useful when you need to reference this image later (e.g., when running a container from it). Tagging helps you manage multiple images and their versions.
- **`.` (dot)**: The dot represents the current directory, indicating to Docker where to look for the `Dockerfile` .

3.3 Running a Container from Your Image

To run a container from the built image, use:

```
docker run -d -p 5000:5000 my-python-app
```

Explanation of Parameters:

- **`run`** : This tells Docker to start a container.
- **`-d`** : This runs the container in detached mode, meaning it runs in the background, and you get back control of your terminal immediately.
- **`-p 5000:5000`** : This maps the container's internal port 5000 to the host machine's port 5000. The first `5000` refers to the host port, while the second `5000` refers to the port inside the container. This allows external traffic (from your browser, for instance) to access the containerized application by accessing `localhost:5000` .
    - The ports are written twice to define the mapping between the host system's port and the container's port. This is necessary when you want external systems to interact with the containerized service.
- **`my-python-app`** : This is the name of the image from which the container will be created.

---

4. Docker Compose

Docker Compose allows you to orchestrate multiple containers, meaning you can start several containers with a single command. Compose is particularly useful when applications consist of multiple services that need to interact with each other, like a web service and a database.

4.1 Creating a `docker-compose.yml` File

```
version: '3'
services:
  web:
    image: my-python-app
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Explanation:

- `version: '3'` : Defines the version of the Docker Compose file format.
- `services` : This section defines the services (containers) that make up your application. In this case, there are two services: `web` and `redis` .
  - `web` : Refers to the Python web application defined by the `my-python-app` image. The `ports` section maps the container's port 5000 to the host's port 5000, similar to the standalone `docker run` command.
  - `redis` : Refers to the Redis database, using the pre-built `redis:alpine` image from Docker Hub.

4.2 Do You Need a Docker Network with Docker Compose?

No separate Docker network configuration is required when using Docker Compose. Docker Compose automatically creates a custom network for all the services defined in the `docker-compose.yml` file. This means the services (`web` and `redis`) are connected by default, allowing them to communicate with each other.

In this network, **container names are used as DNS names**, which means you can refer to one container from another by its service name (e.g., the `web` container can access the `redis` container by simply referring to it as `redis`).

4.3 Docker Compose Commands

To start the containers:

```
docker-compose up
```

To stop the containers:

```
docker-compose down
```

---

5. Docker Networks

Docker networks allow containers to communicate with each other, whether on the same host or across multiple hosts.

5.1 Types of Docker Networks

Docker supports different types of networks:

- **Bridge Network (default)**: Containers on this network can communicate using IP addresses or container names.
- **Host Network**: Removes network isolation between the container and the Docker host.
- **Overlay Network**: Enables communication between Docker containers across different Docker hosts.
- **Custom Network**: You can create your own networks with specific configurations.

5.2 Creating a Custom Network

```
docker network create my-custom-network
```

You can then attach containers to this network using the `--network` flag:

```
docker run -d --network my-custom-network --name container1 my-python-app
docker run -d --network my-custom-network --name container2 redis:alpine
```

Containers on the same custom network can communicate using their container names (DNS names). For example, `container1` can communicate with `container2` by using the name `container2` .

---

6. Using Pre-built Docker Containers from Docker Hub

Docker Hub is a public repository of pre-built Docker containers. You can search for existing containers, download them, and run them on your system.

6.1 Searching for Docker Containers

```
docker search mysql
```

6.2 Pulling a Docker Container from Docker Hub

```
docker pull mysql
```

6.3 Running a Pre-built Container

```
docker run -d -e MYSQL_ROOT_PASSWORD=my-secret-pw mysql
```

This will start a MySQL container with the root password set to `my-secret-pw`.

Conclusion

In this tutorial, you learned how to:

- Install Docker on Linux and Windows,
- Build and run containers,
- Set up and use Docker Compose,
- Configure Docker networks,
- Leverage pre-built Docker containers from Docker Hub.

Docker Compose automatically creates a network for your containers, enabling seamless communication between services. Containers in the same network can use service names as DNS names, making interaction between them straightforward. Additionally, the `docker build` and `docker run` commands allow you to customize container behavior, including port mappings and background operation with detached mode.