# **Dockerfile compose file**

**Example docker-compose.yml**

version: '3.8'

services:

web:

build: .

ports:

- "4000:80"

environment:

- FLASK\_ENV=development

- REDIS\_HOST=redis

depends\_on:

- redis

redis:

image: "redis:alpine"

### **Step-by-Step Explanation**

1. **version: '3.8'**
   * **Explanation**: Specifies the version of the Docker Compose file format. Version 3.8 is compatible with the latest Docker engine features.
   * **Real-time Scenario**: Specifying the version ensures compatibility and access to the latest features and improvements in Docker Compose.
2. **services:**
   * **Explanation**: Defines the services that make up your application. Each service represents a container.
   * **Real-time Scenario**: Breaking down your application into multiple services (like web and redis) allows you to manage and scale each component independently.
3. **web:**
   * **Explanation**: Defines the web service, which will run your Flask application.
   * **build: .**
     + **Explanation**: Tells Docker Compose to build the Dockerfile in the current directory for this service.
     + **Real-time Scenario**: This ensures the web service uses the custom image built from your Dockerfile, including all your application code and dependencies.
   * **ports:**
     + **Explanation**: Maps port 4000 on the host to port 80 in the container.
     + **Real-time Scenario**: This makes your Flask application accessible at http://localhost:4000.
   * **environment:**
     + **Explanation**: Sets environment variables for the container. Here, FLASK\_ENV is set to development and REDIS\_HOST is set to redis.
     + **Real-time Scenario**: Environment variables allow you to configure your application dynamically. Setting FLASK\_ENV to development enables debug mode, while REDIS\_HOST tells the application where to find the Redis service.
   * **depends\_on:**
     + **Explanation**: Specifies that the web service depends on the redis service. Docker Compose will start the redis service before the web service.
     + **Real-time Scenario**: This ensures that the Redis service is up and running before your Flask application tries to connect to it.
4. **redis:**
   * **Explanation**: Defines the Redis service.
   * **image: "redis**

**"**

* + - **Explanation**: Uses the official Redis image with the Alpine Linux distribution, which is small and efficient.
    - **Real-time Scenario**: Using an official image like redis:alpine ensures you get a stable and well-maintained Redis server, which is crucial for caching and data storage in your application.

### **Real-time Scenario: Using Docker Compose**

1. **Create a docker-compose.yml file**:
   * Save the provided docker-compose.yml content into a file named docker-compose.yml in the root directory of your project.
2. **Start the Application**:

docker-compose up

*  This command starts all the services defined in the docker-compose.yml file. The web service builds the Dockerfile, and the redis service pulls the Redis image from Docker Hub.
* You can access your Flask app at http://localhost:4000.

 Stop **the Application**:

docker-compose down

* + his command stops and removes the containers, networks, and volumes created by docker-compose up.

By using Docker Compose, you can manage and orchestrate multiple services that your application relies on, making it easier to set up, develop, and deploy your application in a consistent environment.