# Docker Setup Guide for Contributors to Swarms

Welcome to the `swarms` project Docker setup guide. This document will help you establish a Docker-based environment for contributing to `swarms`. Docker provides a consistent and isolated environment, ensuring that all contributors can work in the same settings, reducing the "it works on my machine" syndrome.

### Purpose

The purpose of this guide is to:

- Ensure contributors can quickly set up their development environment.
- Provide a consistent testing and deployment workflow.
- Introduce Docker basics and best practices.

### Scope

This guide covers:

- Installing Docker
- Cloning the `swarms` repository
- Building a Docker image
- Running the `swarms` application in a Docker container
- Running tests using Docker
- Pushing changes and working with Docker Hub

## Docker Installation

### Windows

1. Download Docker Desktop for Windows from the official website.
2. Install Docker Desktop, ensuring that the "Use Windows containers instead of Linux containers" option is unchecked.
3. Start Docker Desktop and wait for the Docker engine to start.

### macOS

1. Download Docker Desktop for macOS from the official website.
2. Follow the installation instructions, drag-and-drop Docker into the Applications folder.
3. Start Docker Desktop from the Applications folder.

### Linux (Ubuntu)

1. Update your package index: `sudo apt-get update`.
2. Install packages to allow apt to use a repository over HTTPS.
3. Add Dockers official GPG key.
4. Set up the stable repository.
5. Install the latest version of Docker Engine and containerd.

```bash
```

sudo apt-get install docker-ce docker-ce-cli containerd.io

6. Verify that Docker Engine is installed correctly by running the hello-world image.

```bash
sudo docker run hello-world
```

### Post-installation Steps for Linux

- Manage Docker as a non-root user.
- Configure Docker to start on boot.

## Cloning the Repository

```bash
git clone https://github.com/your-username/swarms.git
cd swarms
```

## Docker Basics

### Dockerfile Overview

- Explain the structure and commands of a Dockerfile used in the `swarms` project.

### Building the Image

```bash
docker build -t swarms-dev .
```

### Running a Container

```bash
docker run -it --rm swarms-dev
```

## Development Workflow with Docker

### Running the Application

- Commands to run the `swarms` application within Docker.

### Making Changes

- How to make changes to the code and reflect those changes within the Docker container.

### Running Tests

- Instructions on running tests using `pytest` within the Docker environment.

## Docker Compose for Local Development

- Introduce Docker Compose and its role in simplifying multi-container setups.
- Create a `docker-compose.yml` file for the `swarms` project.

## Dockerfile

Creating a Dockerfile for deploying the `swarms` framework to the cloud involves setting up the necessary environment to run your Python application, ensuring all dependencies are installed, and configuring the container to execute the desired tasks. Here's an example Dockerfile that sets up such an environment:

```Dockerfile
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Set the working directory in the container
WORKDIR /usr/src/swarm_cloud

# Install system dependencies
```

```
RUN apt-get update \
    && apt-get -y install gcc \
    && apt-get clean


# Install Python dependencies

# COPY requirements.txt and pyproject.toml if you're using poetry for dependency management

COPY requirements.txt .

RUN pip install --upgrade pip

RUN pip install --no-cache-dir -r requirements.txt


# Install the 'swarms' package, assuming it's available on PyPI

ENV SWARM_API_KEY=your_swarm_api_key_here

ENV OPENAI_API_KEY=your_openai_key

RUN pip install swarms


# Copy the rest of the application

COPY . .


# Add entrypoint script if needed

# COPY ./entrypoint.sh .

# RUN chmod +x /usr/src/swarm_cloud/entrypoint.sh


# Expose port if your application has a web interface

# EXPOSE 5000


# Define environment variable for the swarm to work
# Add Docker CMD or ENTRYPOINT script to run the application
# CMD python your_swarm_startup_script.py
# Or use the entrypoint script if you have one
# ENTRYPOINT ["/usr/src/swarm_cloud/entrypoint.sh"]

# If you're using `CMD` to execute a Python script, make sure it's executable
# RUN chmod +x your_swarm_startup_script.py
```

To build and run this Docker image:

1. Replace `requirements.txt` with your actual requirements file or `pyproject.toml` and `poetry.lock` if you're using Poetry.
2. Replace `your_swarm_startup_script.py` with the script that starts your application.
3. If your application requires an API key or other sensitive data, make sure to set these securely, perhaps using environment variables or secrets management solutions provided by your cloud provider.
4. If you have an entrypoint script, uncomment the `COPY` and `RUN` lines for `entrypoint.sh`.
5. If your application has a web interface, uncomment the `EXPOSE` line and set it to the correct port.

Now, build your Docker image:

```sh
docker build -t swarm-cloud .
```

And run it:

```sh
docker run -d --name my-swarm-app swarm-cloud
```

For deploying to the cloud, you'll need to push your Docker image to a container registry (like Docker Hub or a private registry), then pull it from your cloud environment to run it. Cloud providers often have services specifically for this purpose (like AWS ECS, GCP GKE, or Azure AKS). The deployment process will involve:

- Pushing the image to a registry.
- Configuring cloud services to run your image.
- Setting up networking, storage, and other cloud resources.
- Monitoring, logging, and potentially scaling your containers.

Remember to secure sensitive data, use tagged releases for your images, and follow best practices for operating in the cloud.