Guide Overview

Prerequisites

Before you start, ensure you have the following installed on your machine:

- **Docker**: Docker Engine is required to build and run containers.
- **Docker Compose**: Docker Compose is used to manage multi-container Docker applications through a YAML file.

Recommendation:
Install docker desktop to fulfill the above prerequisites.

1. Building the Image

To build the Docker image for the ossqa service, navigate to the directory containing your docker-compose.yml file and run:

```
docker-compose build
```
This command tells Docker Compose to build the image using the specified Dockerfile and context.

2. Running the Container

After the image is built, you can run the container with the following command:
```
docker-compose up
```

This command starts the ossqa service. By default, Docker Compose runs in the foreground, displaying the service's log output to the console.

To run the services in the background, add the -d (detached) option:
```
docker-compose up -d
```

3. Viewing Logs and Managing the Service

To view the logs of the running service, use:
```
docker-compose logs
```

You can follow the logs in real-time by adding the -f option:
```
docker-compose logs -f
```

To stop the service, use:
```
docker-compose down
```

This command stops and removes the containers, networks, and volumes associated with the service.

4. Understanding the Compose File

The provided Docker Compose file defines a service named ossqa. Here's a breakdown of the key components:

- **version**: This specifies the version of the Docker Compose file format.
- **services**: This section defines the services (containers) that make up the application.
- **ossqa**: The name of the service.
- **image**: Specifies the image to use for the service. If the image does not exist locally, Docker will attempt to pull it from the configured registry.
- **build**: Defines configuration options that are applied at build time.
- **context**: The build context path to the directory containing the Dockerfile and any other build files.
- **dockerfile**: The path to the Dockerfile within the build context.

5. Understanding the Dockerfile

Our Dockerfile is designed to create a Docker container for a Python application. The Dockerfile uses a base image of Python 3.12 and includes configurations for environment variables, dependency installations, working directory setup, user permissions, and an entry point command. Below is a detailed breakdown of each instruction in the Dockerfile:

Base Image

- **FROM python:3.12:** Specifies the base image for this container, using Python version 3.12. This forms the foundation of the container and includes the Python runtime environment.

Environment Variables

- **ENV PYTHONDONTWRITEBYTECODE=1:** Prevents Python from writing .pyc files, which are compiled bytecode files. This is useful in a container environment to avoid unnecessary files that are not needed for execution.
- **ENV PYTHONUNBUFFERED=1:** Disables Python's buffering, forcing it to flush its output directly to the terminal/console. This makes the logs from the container appear in real time, aiding in debugging and logging.

Dependency Installation

- **COPY requirements.txt .:** Copies the requirements.txt file, which lists the Python package dependencies, from the host machine into the root of the container's filesystem.
- **RUN python -m pip install -r requirements.txt:** Installs the Python dependencies specified in requirements.txt using pip, Python's package installer.

Working Directory and Application Code

- **WORKDIR /app:** Sets the working directory inside the container to /app. All subsequent instructions will be executed in this directory.
- **COPY . /app:** Copies the entire context (i.e., the source code and other relevant files from the project directory) into the /app directory inside the container.

User Configuration

- **RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app:** Creates a non-root user named appuser with a specific UID (5678) for security purposes. It also changes the ownership of the /app directory to appuser, ensuring that the application can run under this user for added security.

Entry Point

- **CMD ["python", "scr/main.py"]:** Specifies the default command to run when the container starts. This command executes the Python application's main script, main.py, located in the src directory.