Guide Overview

1. 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.
- **The OSSQA git repo:** This guide assumes that we are working within the OSSQA git repository.

Recommendation:
Install docker desktop to fulfil the ***Docker*** and ***Docker Compose*** prerequisites.

2. Setup

Locate your installation the OSSQA repository, navigate to the root of the project */<local-path>/OSSQA/* and create a file named "***.env***" there. Edit the */OSSQA/.env/* file and add your github authentication token: `GITHUB_AUTH_TOKEN=<your-token>` then save.

This will allow ***ssf scorecard*** to make requests to github.

3. 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 <image name>`
Example: `docker compose build ossqa`
This command tells Docker Compose to build the ossqa image using the specified Dockerfile and context provided in the docker-compose.yml file.

4. Running the Container

After the image is built, you can run the container with the following command:

`docker compose run --rm <image name>`
Example: `docker compose run --rm ossqa`
This command executes a one-off command for the ***ossqa service*** defined in the ***docker-compose.yml*** file and automatically removes the container after the command completes. The ***--rm*** flag tells docker to remove the container after command execution. This command is ideal for any operation that does not need to persist a container after its job is done.

NOTE: If a build for the service has not already been built the run command will automatically build the image.

5. Viewing Logs and Managing the Service

Since we are running the ***--rm*** option in the command from 2 is not directly applicable in the same way.

However, you can capture the output (logs) of a running container by using the following command when running the ossqa service:
`docker compose run --rm ossqa > ossqa_logs.txt`

This will capture the cmd output of the program and store it in a **new** text file called ***ossqa_logs.txt***.

6. Understanding the Compose File

The Docker Compose file contains services for the ossqa application and its prototype variant. Below is a detailed breakdown of its key components:

- **version: '3.4':** This line specifies the version of the Docker Compose file format being used. Version 3.4 supports a specific set of features and configurations compatible with Docker Engine 17.09.0+.
- **services:** This section outlines the services (containers). In this configuration, two services are defined: ossqa and ossqa-prototype.

Service: ossqa

- **image: ossqa:** Designates the Docker image for the service. Docker will pull this image from the default Docker registry if it's not found locally.
- **build:**
    - **context: .:** The build context is set to the current directory (.), indicating that Docker should look here for any files referenced during the build.
    - **dockerfile: ./Dockerfile:** Specifies the path to the Dockerfile used to build the image, relative to the build context.
    - **env_file: .env:** Points to a file named .env in the same directory as the Docker Compose file. This file contains environment variables that are passed into the container at runtime.

Service: ossqa-prototype

- **image: ossqa-prototype:** Specifies the Docker image for the prototype version of the service. Similar to the ossqa service, Docker will attempt to pull this image from the registry if it's not available locally.
- **build:**
    - **context .**: Sets the current directory as the build context, similar to the ossqa service.
    - **dockerfile: ./src/prototype/Dockerfile:** Indicates the Dockerfile path for the prototype service, which is located in the *./src/prototype directory*, relative to the build context.
    - **env_file: .env:** Like the ossqa service, this specifies the .env file from which environment variables are loaded into the prototype service's container.

Summary

This Docker Compose file defines two closely related services, ossqa and ossqa-prototype, each built from their respective Dockerfiles and sharing a common environment configuration file .env. The setup illustrates a flexible approach to managing multiple, related containers within a single Compose file.

7. Understanding the Dockerfile

Overview

Our Dockerfile creates a Docker image based on the official Python 3.11 runtime using an Alpine Linux base image. It sets up an application with its dependencies, prepares the environment for running a Python application, and specifies how the application should be executed. Below is a detailed breakdown of each instruction in the Dockerfile:

Breakdown of the Dockerfile

Base Image

- **FROM python:3.11-alpine:** Specifies the base image for this container, using Python version 3.11 and the Alpine Linux operating system known for its small size and security.

Working Directory

- **WORKDIR /app:** Sets */app* as the working directory inside the container. This directive ensures that all subsequent commands are run from this directory, serving as the default location for the application's source code and dependencies.

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.
- **ENV GITHUB_AUTH_TOKEN=$GITHUB_AUTH_TOKEN:** This line forwards a GitHub authentication token from the host environment to the container. This allows us to use ssf scorecards to analyse github repos.

Copying Files

- **COPY requirements.txt .:** Copies the *requirements.txt* file from the host into the container's current working directory. This file lists the project's Python package dependencies.
- **COPY /src /app:** Transfers the application's source code from the host's src directory to the /app directory inside the container, ensuring the application code is available for execution.

Executable Binary

- **COPY src/scorecard-build/scorecard /usr/local/bin/scorecard:** Copies a custom executable, scorecard, into the container. This binary is then available globally within the container's PATH.
  RUN chmod +x /usr/local/bin/scorecard: Sets the scorecard binary as executable. Allowing it to be executed as part of the container's operations or application processes.

Virtual Environment

- **RUN python -m venv /app/venv:** Initialises a Python virtual environment within the */app/venv* directory inside the container.
- **ENV PATH="/app/venv/bin:$PATH":** Adjusts the PATH environment variable to prioritise the virtual environment's binaries, ensuring that subsequent Python and pip commands use the versions and dependencies specific to the application.

Dependency Installation

- **RUN python -m pip install -r requirements.txt:** Instals the Python package dependencies listed in requirements.txt. This command utilises pip from the previously created virtual environment.

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.added security.

Entry Point

- **CMD ["python", "main.py"]:** Specifies the default command to run when the container starts. In this case, it executes the main.py script with Python, initiating the application. This command can be overridden with command-line arguments when starting the container.