Docker Compose Configuration for Teamium Project

Tomasz Olejarczuk

Fontys

21-06-2024

Contents

Docker Compose Configuration for Teamium Project

1. Summary

This document provides a comprehensive overview of the `docker-compose.yml` file used to orchestrate the deployment of the Teamium project's containerized environment. It explains the roles of each service, their configurations, networking setup, and the use of shared volumes to facilitate communication and data exchange between containers. Notably, this configuration incorporates the `bitnami/os-shell` container to streamline the management of file permissions within shared volumes, ensuring seamless operation of the system.

2. Introduction

2.1. Project Overview

The Teamium project consists of multiple components, each running within a Docker container:

- **MATLAB Algorithm:** Executes the core MATLAB algorithm for data processing and analysis.
- **API Wrapper:** Provides a RESTful interface for interacting with the MATLAB server and managing data.
- **RabbitMQ:** A message broker that enables asynchronous communication between the API wrapper and MATLAB server.
- **PostgreSQL:** A relational database for storing logs, requests, and responses.
- **MinIO:** An S3-compatible object storage solution for storing heatmap images.
- **Redis:** An in-memory data store used for caching.

Docker Compose simplifies the management of this multi-container environment by defining the configuration and relationships between services in a single YAML file.

2.2. Objectives

The main objectives of this Docker Compose configuration are:

1. **Simplified Deployment:** Allow for easy setup and startup of the entire system with a single command (`docker compose up`).
2. **Container Orchestration:** Define dependencies and relationships between services to ensure proper startup order and communication.

3. **Shared Volumes:** Enable data sharing and persistence between containers using Docker volumes.
4. **Networking:** Establish a network for inter-container communication.
5. **Permission Management:** Utilize the `bitnami/os-shell` container to automate the setting of file permissions within shared volumes, preventing potential access issues.

3. Methodology

3.1. Services

The `docker-compose.yml` file defines the following services:

- **`prepare-data`:**
  - **Image:** `bitnami/os-shell`
  - **User:** `root` (to ensure permissions can be modified)
  - **Command:** Executes a Bash script that changes the ownership of shared folders (`/minio_data` and `/data`) to UID/GID 1001. This aligns with the user in the MATLAB image and ensures write permissions.
  - **Volumes:** Mounts the `miniovol` and `datavol` volumes.
  - **Networks:** Joins the `matlab-network` bridge network.
  - **Depends On:** None (runs before other services to set up permissions)
- **`matlab-algorithm`:**
  - **Image:** The MATLAB algorithm image from the GitHub Container Registry.
  - **Container Name:** Set dynamically based on an environment variable.
  - **Restart:** `always` to automatically restart the container if it fails.
  - **Ports:** Exposes port 9910 for communication with the API wrapper.
  - **Volumes:** Mounts the `datavol` volume for data sharing with other containers.
  - **Networks:** Joins the `matlab-network` bridge network.
- **`matlab-wrapper`:** (Similar configuration to `matlab-algorithm`)
- **`rabbitmq`:**
  - **Image:** The official RabbitMQ Docker image with the management plugin.
  - **Container Name:** `rabbitmq`
  - **Restart:** `always`
  - **Ports:** Exposes ports 5672 (AMQP) and 15672 (management UI).
  - **Environment:** Sets the RabbitMQ username and password from environment variables.

  - **Networks:** Joins the `matlab-network` bridge network.

- **`postgres`:** (Similar configuration to `rabbitmq`, but uses the PostgreSQL image and exposes port 5432)
- **`minio`:** (Similar configuration to `rabbitmq`, but uses the MinIO image and exposes ports 9000 and 9001)
- **`redis`:** (Similar configuration to `rabbitmq`, but uses the Redis image and exposes port 6379)

3.2. Volumes

The `docker-compose.yml` file defines the following volumes:

- **`miniovol`:** Persistent storage for MinIO data.
- **`postgresvol`:** Persistent storage for PostgreSQL data.
- **`redisvol`:** Persistent storage for Redis data.
- **`datavol`:** Shared volume for data exchange between the MATLAB container, API wrapper, and `prepare-data`. This is where the MATLAB algorithm stores its results and where the API wrapper accesses them.

3.3. Networks

The `matlab-network` bridge network is created to enable communication between all the containers in the environment.

4. Results & Insights

4.1. Key Outcomes

- **Simplified Deployment:** The entire system can be started with a single command, streamlining the setup process for both development and production environments.
- **Reliable Operation:** The use of `bitnami/os-shell` ensures that file permissions are set correctly within shared volumes, preventing runtime errors due to permission issues.
- **Scalability:** The configuration allows for easy scaling of individual services as needed, particularly when deployed on a platform like Kubernetes.

4.2. Further Considerations

- **Security:** Additional security measures, such as network isolation and encryption, may be necessary depending on the sensitivity of the data being handled and the deployment environment.
- **Resource Management:** For production environments, consider defining resource limits (CPU, memory) for each container to prevent resource exhaustion and ensure optimal performance.

5. Conclusion

The Docker Compose configuration for the Teamium project effectively orchestrates the deployment of multiple containers, providing a streamlined, scalable, and reliable environment for running the MATLAB-based application. The integration of `bitnami/os-shell` further enhances the setup by automating file permission management within shared volumes.