DevOps Lab1

Arek Uchymiak

February 2023

Github: https://github.com/UP941374/DevOps-Lab-1
Docker: https://hub.docker.com/repositories/auser93
Docker pull: docker pull auser93/student_service

Introduction

This section describes the stages of the typical DevOps life cycle, their main objectives as well a short description of how Lab1 exercises relate to all of these stages. Due to the nature of the assignment, some stages were less significant (e.g. planning or operate).

Planning - establishing a clear understanding of the project's goals and requirements and to create a plan for achieving them. In case of Lab1, the requirements were clear at the start (following the tutorial) of the project therefore the importance of this stage was significantly diminished.

Code (development/integration) - producing high-quality code that meets the project's requirements. The integration part ensures that newly produced code (changes) can be combined with the existing code without errors.

Build - compiling the source code into an executable application, running unit tests, and packaging the application components. Its primary objective is to create a stable and consistent build of the application that can be deployed to the production environment.

Test - testing the code for functionality, performance, and security. The testing stage helps is to identify and fix any issues before the code is released. The testing in Lab1 was automated thanks to GitHub actions, the code is tested on every push, making sure all committed changes pass the automated tests. Failed test could indicate a bug (or bugs) in code, which requires fixing.

Release - ensuring that the application is delivered to end-users with high quality, efficiency, and reliability, while minimizing the risk of errors and downtime. Any major changes made to the code require a new release (provided it builds and passes tests) which is then deployed.

Deploy - installing or deploying the software to the target environment, such as a production server or cloud-based platform (DockerHub in Lab1). The deployment stage involves activities such as configuring the target environment, installing the software artifacts, and verifying that the deployment was successful. In this stage, the software is made available for use by end-users.

Operate - During this stage, the operations team takes over from the development team and is responsible for ensuring that the application is running smoothly, meeting the needs of its users, and delivering business value. In Lab1, the operate stage was limited to making sure the API functions correctly without errors.

OpenAPI Exercises

Define Objects

Working with OpenAPI is easy thanks to a good documentation. Defining objects can be achieved by using YAML or JSON. I used YAML which means the curly braces can be dropped however the correct indentation of code is required which in my opinion makes the code look very clean. The actual object definition is as follows:

- **components:** container for various reusable definitions, in this assignment, there are two, Student (code provided) and GradeRecord
- **type:** type of objects, in this case, object
- **required:** specifies which parameters are required when a call is made, otherwise validation fails and error is returned
- **properties:** here we can define the actual properties, for each property a name is required, type, example is optional but its a good practise to add it. The property gradeRecord uses a reference ($ref), which points to another defined property. Min and Max definitions specify the valid range for the grade parameter.

Add Delete method

The definition of the delete method ensures that the API responds to the delete call correctly, i.e. removes a record from a database. The path is specified for all methods at the beginning and the parameter (student_id) is passed in the URL. The delete method has the following descriptions:

- **summary:** short description of the method
- **description:** similar to summary however this key is reserved for more detailed descriptions
- **operation_id:** a string (should be unique) which identifies the operation
- **parameters:** parameters of the method, there are several, some are required some are optional, it's good practice to always add as much as possible which makes it easier to use and maintain the API
- **parameters:name:** name (must be the same as the key in path (student_id))
- **parameters:in:** refers to the path
- **parameters:description:** description
- **parameters:required:** boolean value, for path parameters such as this one, required must be set to true

- **parameters:schema:type:** specifies the data type
- **parameters:schema:format:** (optional) specifies data type in more detail
- **responses:** describes the responses which are sent back to the client, openAPI specifies that it should contain at least one response (usually the successful response). For delete method we can specify 3 responses, one for successful delete (200), one for bad request e.g. invalid ID (400) and one for a request with a key that is not found in the database (404).

MongoDB Integration

Alpine does not contain mongoDB package by default so it needs to be installed first. This can be done by adding mongoDB dependency the requirements.txt file. Then, the dockerfile needs be edited to make sure that mongoDB is setup (data directory (RUN mkdir -p /data/db) and port (EXPOSE 27017) and the service is started (CMD ["mongod"]). The next steps are to modify the python code directly. First, the connection to the mongoDB needs to be established, this can be done in the __main__.py file. Once connection is established, the database can be queried which can be done by modifying the corresponding functions in the service/student_service.py file.

Question

This section describes some of the advantages and disadvantages of using the Alpine Linux distribution in Docker.

Advantages:

Small size - Alpine Linux is known for its small size, which means that Docker images built on Alpine will also be smaller in size compared to other distributions. This can be beneficial in terms of reducing the time it takes to transfer images over the network and can also save disk space.
Improved security - Alpine Linux is a security-oriented distribution, and it includes security features such as exploit mitigation and a minimal set of packages installed by default. This can reduce the attack surface and the number of potential vulnerabilities in a Docker image.
Improved performance - Due to its small size and minimalistic design, Alpine Linux can also provide improved performance compared to other distributions. This can result in faster boot times and improved overall performance in a Docker container.

Disadvantages:

Reduced compatibility - Due to its minimalistic design, Alpine Linux may not include all the packages and libraries required for a specific application. This can result in compatibility issues and may require additional work to make the application run on Alpine.
Steep learning curve - Alpine Linux is a different distribution from the more commonly used distributions like Debian or Ubuntu. This means that there is a steep learning curve for those who are used to working with other distributions.
Reduced community support - While Alpine Linux has a strong community of developers, it is still not as widely used as other distributions. This can result in reduced community support and fewer available resources compared to other distributions.