**Section 01**

**1)**

a) Note - Included in the folder separately.

b)

Docker Hub - <https://hub.docker.com/repository/docker/gamaka7/webapp/general>

( docker pull gamaka7/webapp:pythonapp )

c)

Unit Tests:   
  
The preferred test framework for unit testing in Flask applications is pytest. pytest is widely adopted in the Python community and provides a simple and expressive syntax for writing tests. Use pytest's features like fixtures to set up the test environment, manage dependencies, and provide reusable test data. Leverage pytest's powerful assertion library to make assertions about the behavior and output of your code. Use pytest's test discovery feature to automatically discover and run unit tests in your project directory. Consider using tools like coverage.py to measure test coverage and identify areas of your codebase that need more testing.

Integration Tests:

For integration testing in Flask applications, pytest is also a preferred choice. It offers flexibility and a rich ecosystem of plugins that can simplify writing and managing integration tests. Utilize Flask-specific testing libraries like Flask-Testing or Flask-WebTest to assist in writing integration tests and mocking the Flask environment. Use libraries like requests or the built-in Flask test client to simulate HTTP requests and test the behavior of your Flask application from end to end. Set up a separate test database or use fixtures to ensure that integration tests run in an isolated environment, separate from the production database. Test the interaction between different components of your Flask application, such as routes, views, models, and database operations.

**2)**

This kind of shell script is very valuable since it automates the deployment process and saves the consuming time of the developers by reducing the human interaction.   
( Please find the deploy.sh shell script which contains in the folder.)

Instructions:

Ensure that the script is executable by running “chmod +x deploy.sh”.

Execute by using “./deploy.sh” command.

a)

Determine the schedule at which you want the deployment to occur. You can use the following format for scheduling in a cron job: \* \* \* \* \* command

Each asterisk represents a different time element: minute (0-59), hour (0-23), day of the month (1-31), month (1-12), and day of the week (0-7, where both 0 and 7 represent Sunday).

Open the crontab file by running the command crontab -e.   
This will open the file in the default text editor.

Add a new line in the crontab file to specify the schedule and command for the deployment. For example, if you want to schedule the deployment to occur every day at 2:00 AM, you can add the following line: 0 2 \* \* \* /path/to/deploy.sh

Save the crontab file and exit the text editor. The cron job is now set up to execute your deployment shell script based on the specified schedule. The script will be automatically triggered at the designated time, initiating the deployment process.