QA-DevOps Final Project Oct 31st 2022

Introduction:

Overall Concepts and Technologies:

- Project Management
- Python and Flask Fundamentals
- Python and Flask Testing
- Git
- Linux
- Flask Web Development
- Databases
- Continuous Integration and Deployment (CI/CD) With Jenkins
- Cloud Fundamentals with Azure
- Containerisation using Docker Containers and Docker Swarm

Overview

Main objectives

- To create a **web application** that integrates with a **database** and demonstrates **CRUD** functionality.
- To utilise **containers** to host and deploy your application.
- To create a **continuous integration (CI)/continuous deployment (CD) pipeline** that will automatically **test**, **build** and **deploy** your application.

@Kwin-22's DevOps-Project

application development | View 3 | New view

	Title	Assignees	Status
1	complete list-app update functionality	Kwin-22	Done
2	complete delete functionality	Kwin-22	Done
3	set up Jenkins	Kwin-22	Done
4	Set up docker	Kwin-22	Done
5	Create docker container	Kwin-22	Done
6	create pipeline	Kwin-22	Done
7	unit testing	Kwin-22	In Progress
8	webhook for jenkins	Kwin-22	In Progress

You can use Control + Space to add an item

Project Management and Version Control

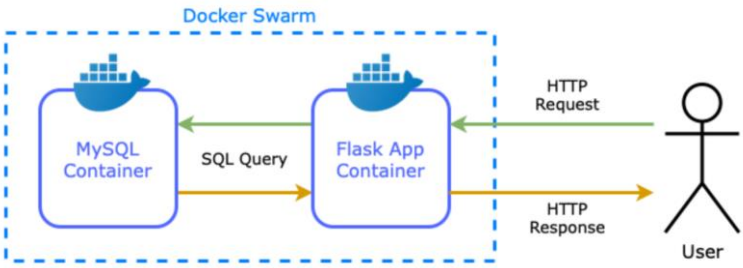
I used **Git** and **GitHub** to keep track of Version Control and maintain the repository of the source code. I had considered Jira to keep track of the Software development but was outside the scope of the project. I used GitHub's new Project functionality to keep a note of the jobs and tasks using an Agile Kanban approach.

Application

Technologies:

- Python
- Pytest
- Flask (with Jinja templates)
- Jenkins
- Docker/Docker Compose
- Docker Swarm
- MySQL

- The frontend aspect of the app used Flask with HTML templates (using Jinja Framework) to serve the web pages that allows the user to perform CRUD functionality with the database.
- The backend aspect of the application uses SQLAlchemy to model and integrate with the database. The database used was MySQL and was in a separate container.

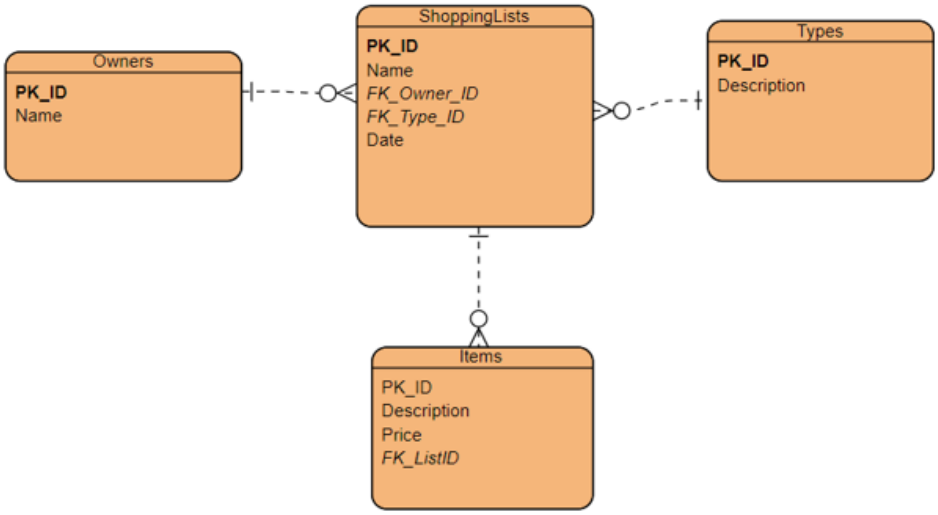


ER-DIAGRAM

Shopping List Application

This application will allow users(owners) to create shopping lists. There may be different types of shopping lists such as (Groceries, Wishlist, Wedding, Birthday etc.) Each list would hold a number of items. Below is the anticipated ER diagram that represents this design.

ER Diagram



Testing

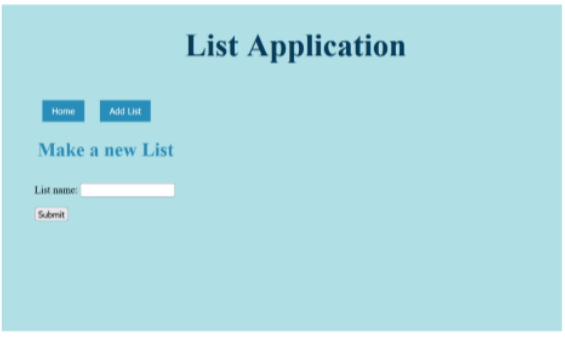
Unit tests was done manually using whitebox approach. However, automation of this is in progress with the use of 'pytest' and due for delivery in the next release.

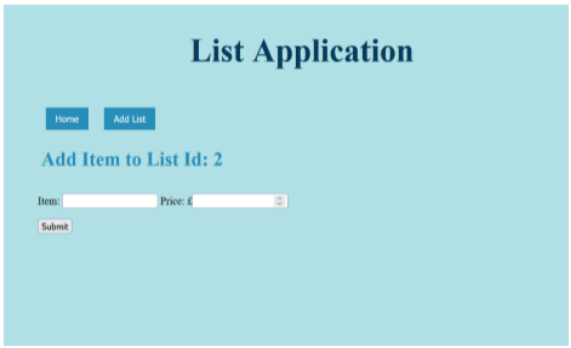
Application

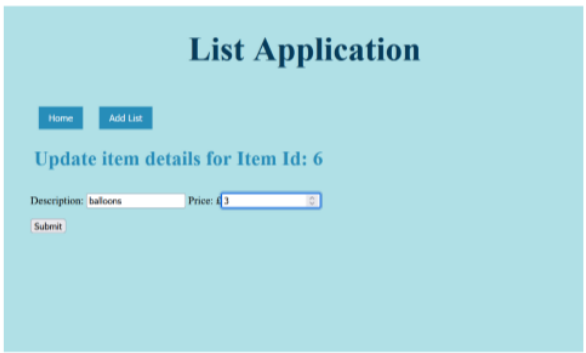
The Shopping Application was designed to allow the production and management of multiple lists that have none or many items. (Version 1 only has this functionality. Version 2 will introduce the Owner and Type entities and relationships shown in the ER-diagram).

It uses Python Flask to create HTML templates with the help of Jinja Framework for the front end and MySQL at the backend via SQLAlchemy. SQL Alchemy's ORM uses the models from the flask app to map to the database. The overview of the structure is as below:

Below, you can see some of the screens that are used to perform the CRUD operations to create a list, and items for the list, to update these, to view them (on the ALL LISTS (index) page), or to finally delete them.







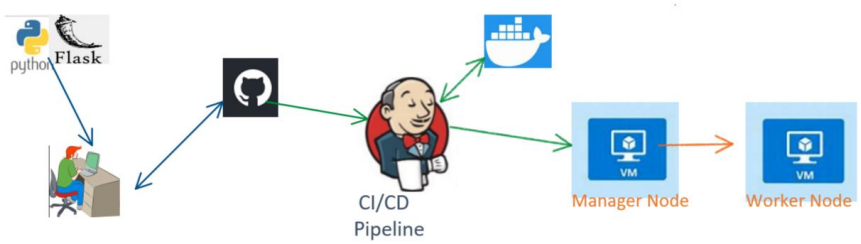
Deployment

Technologies:

- Docker Swarm

The application is deployed to a Docker Swarm hosted in the cloud. It consists of one manager node and one worker node. The manager deploys the application onto both virtual machines.

- Jenkins – for CI/CD Pipeline
-



PIPELINE

The Jenkins CI/CD pipeline uses my Github Repository to get the latest changes. It runs the steps on the Jenkinsfile to perform the deployment to the virtual machines automatically.

It accesses DockerHub to push and pull down the latest images of the Flask-app, the SQL database, and Nginx.

Three Docker containers are created and used to hold these images separately. Nginx is used as a reverse proxy to redirect the traffic from the default flask port to the browsers default port 80.

A Docker Compose .yaml file is used to create the Swarm and deploy the application across 2 virtual machines (master and worker nodes).

A Webhook was created used in Jenkins to automate this process every time a change was made in the application.

RISK ASSESSMENT

I have started to look at the risks that may affect this application and pipeline setup. Below is the starting point and further risks will be added to the working document as they become apparent.

Risk Assessment

Description	Likelihood	Impact	Responsibility	Response
Bug in Flask App	3	3	Developer	Email Developer with Bug details
VM down	4	4	DevOp/Azure	Start new VM in meantime while waiting for restoration
SQL Injection Breach	2	4	Developer/Cyber tea	Follow Security Protocol retore from backup and increase security
Dockerhub down	2	3	Dockerhub	Revoke partial updates
Jenkins unavailable	2	3	Jenkins	Run updates manually

Likelihood		Impact	
Code	Description	Code	Desciption
1	Not Likely	1	Minimal
2	Moderate	2	Low
3	Likely	3	Moderate
4	Very likely	4	High

FUTURE IMPROVEMENTS

Flask Application:

The following are considerations for further development.

1. Improve the linking of pages programmatically through use of buttons rather than user URL input.
2. Display **List Name** as well as **Id** in all pages that require it.
3. Add items to a list by clicking on the list instead of referring to primary key.
4. Improve **Exception Handling**.
5. Move Styling to **CSS.**
6. **Add UNIT TESTING (high priority)**
7. Add functionality of other entities in ER diagram (for Version 2) – to include a relationship between lists and owners and a choice of list type in a drop down.
8. Add Authorisation/Authentication.

Pipeline:

1. Add tests to ensure deployment works correctly.
2. Add automatic alerts.
3. **Add a webhook** to make deployment easier (once all functioning as expected) **(High Priority)**

Swarm:

1. Add further VMs to further split the container distribution and manage workload.
2. Do testing with Load Balancing.