Project 3: Flask-Based Student Registration Web App with Jenkins Deployment

Intern: Shyam Chamapkbhai Chotaliya **Company:** Fortune Cloud Technologies **Date:** [14-07-2025]

Project Objective & Description

The objective of this project was to develop, deploy, and automate a full-stack student registration web application. The project involved creating a Python Flask backend to handle form submissions, a MySQL database for persistent data storage, a custom HTML/CSS frontend for user interaction, and a Jenkins CI/CD pipeline to automate the deployment process on an AWS EC2 instance.

The final application allows users to:

1. Register a new student via a web form.
2. View a list of all registered students displayed in a clean, tabular format.

Tools & Technologies Used

- **AWS EC2:** Hosted the entire stack, including the Flask application, Jenkins server, and MySQL database on an Ubuntu 22.04 instance.(t2.small)
- **Python Flask:** A micro web framework used to build the application's backend logic, define routes, and connect to the database.
- **MySQL Server:** A relational database used to store student records (name, email, mobile, course).
- **Jenkins:** An automation server configured to pull the latest application code from GitHub and automatically run the deployment script.
- **Git & GitHub:** Version control system used to manage the source code and host the repository.
- **HTML5 & CSS3:** Used to create a custom, responsive user interface for the registration form and the student list page.
- **Pymysql:** A Python library used to connect the Flask application to the MySQL database.

Step-by-Step Setup and Deployment

1. **EC2 Instance Configuration:**
    - An AWS EC2 `t2.small` instance was launched with an **Ubuntu 22.04 LTS** AMI.

- A security group was configured to allow inbound traffic on ports **22 (SSH)**, **8080 (Jenkins)**, and **5000 (Flask App)**.
- Required software was installed via the command line: `python3-pip`, `git`, `mysql-server`, and `jenkins`.

2. **MySQL Database and Table Setup:**

   - MySQL Server was installed and started on the EC2 instance.
   - A dedicated database `studentdb` and a user `flaskuser` were created with specific privileges to enhance security.
   - A `students` table was created with columns: `id`, `name`, `email`, `mobile`, and `course`.

3. **Flask Application Structure:**

   - The application code was structured with separate directories for templates and static files, following Flask best practices.
   - **`app.py`:** Contains the core backend logic, including routes for `/register` (to handle both GET and POST requests) and `/students` (to display the data).
   - **`templates/`:** Contains the `register.html` and `students.html` files.
   - **`static/css/`:** Contains the `style.css` file for all custom styling.

4. **Jenkins Job for Automated Deployment:**

   - Jenkins was installed and configured on the EC2 instance.
   - A new **Freestyle project** was created.
   - The job was configured to pull source code from the project's GitHub repository.
   - An **"Execute shell"** build step was added to:
     1. Find and stop any old running instance of the Flask application.
     2. Start the new application in the background using `nohup flask run --host=0.0.0.0 &`.

Explanation of Major Components

- **Flask Backend (`app.py`):**

  - **Routing:** Uses decorators like `@app.route('/register')` to map URLs to specific Python functions.
  - **Request Handling:** Differentiates between `GET` (displaying a page) and `POST` (submitting form data) requests.
  - **Templating:** Uses `render_template()` to dynamically generate HTML pages from files in the `templates` folder.

- **MySQL Connection:**

  - The `pymysql` library is used to connect the Python application to the MySQL database.
  - A dedicated function `get_db_connection()` handles the creation of a connection, which is used to execute SQL queries for inserting and retrieving data.

- **HTML/CSS Frontend:**
    - The project follows Flask's standard structure: HTML files are in the `templates` directory and CSS files are in `static/css`.
    - `register.html` provides the user-facing form.
    - `students.html` uses a Jinja2 for-loop (`{% for student in students %}`) to dynamically build an HTML table from the data passed by Flask.
- **Jenkins Integration:**
    - Jenkins automates the deployment workflow. When a build is triggered, Jenkins pulls the latest source code from GitHub, installs any dependencies, and runs a shell script to restart the Flask application, ensuring the live application is always up-to-date with the `main` branch.

GitHub Repository Integration

The complete source code for this project is hosted on GitHub. The repository follows a standard structure for a Flask application.

- **Repository Link:** https://github.com/Shyamchotaliya/stud-reg-flask-app (https://github.com/Shyamchotaliya/stud-reg-flask-app)
- **Repository Structure:**
    - `app.py`: Main application file.
    - `templates/`: Contains all HTML files.
    - `static/`: Contains CSS file.
    - `requirements.txt`: Lists all Python dependencies.
    - `README.md`: Project documentation.
    - `.gitignore`: Specifies files to be ignored by Git (e.g., `__pycache__`,`__venv__`).

Testing and Verification

The application was tested thoroughly to ensure all components work together correctly:

1. **Form Submission:** Navigated to the `/register` page, filled in the student details, and submitted the form. The application correctly redirected to the `/students` page, showing the newly added student.
2. **Database Validation:** Connected directly to the MySQL database via the command line and ran a `SELECT * FROM students;` query to confirm that the submitted data was successfully stored.
3. **Jenkins Trigger:** Made a small change to the HTML code in the GitHub repository, then manually triggered the "Build Now" button in Jenkins. The successful build log and the updated content on the live website confirmed that the CI/CD pipeline was working as expected.

Screenshots Summary

The submission includes a `screenshots/` folder containing visual evidence of:

- The Jenkins job configuration and a successful console output.
- The live web application: the registration form and the student list page.
- The MySQL command-line interface showing the table data.
- The project's GitHub repository page.

Key Takeaways and Learnings

- **Full-Stack Integration:** This project provided a holistic view of how different technologies (backend, frontend, database, CI/CD) connect to form a complete web application.
- **Importance of Automation:** Manually deploying applications is tedious and error-prone. Using Jenkins to automate the pull-and-deploy process saves time and ensures consistency.
- **DevOps Principles:** The project is a practical implementation of CI (Continuous Integration) principles, where changes from a central repository are automatically built and deployed.
- **Security Best Practices:** Understood the importance of configuring EC2 security groups correctly and creating dedicated database users with limited permissions.

> ***Note:*** *This document covers only Project 3 out of the 6 assigned internship projects. The remaining projects are in progress and will be submitted separately.*