**Introduction**

Library Book Management System is a web-based application designed to facilitate the management and discovery of books in the library environment. This design document describes the design process, material, design, database model, authentication process, error handling, and other functions.

**Project Overview**

The library management system provides many functions such as user registration, searching and searching, transferring books, reading materials, managing books and notes. The system is designed to provide a user-friendly interface for both users and librarians/administrators.

**Scope**

This technical design document covers the following aspects of the Libro Book Management System:

* **Architecture**: Clean Architecture, promoting modularity and testability.
* **Design Patterns**: Service Pattern, Repository Pattern, and Unit of Work Pattern.
* **Error Handling**: Middleware-based logging for capturing and tracking important events, requests, and errors.
* **Authentication**: Form based authentication for secure user authentication and authorization.
* **Database Design**: Utilizing a suitable database structure for storing book information, user profiles, and other relevant data.
* **Additional Features**: Reading Lists, Book Reviews and Ratings, Notifications, and Book Recommendations.

**Clean Architecture**

Libraries maintain books in accordance with clean design standards. This architecture supports change, control, and testing by separating concerns and making the code base flexible and adaptable to future changes. It includes domains, applications, infrastructure, and presentation layers, each with their own roles and dependencies.

**Service Pattern**

System management libraries use service models to organize business logic and functionality into reusable and interoperable components. This model promotes flexibility and separation of concerns, allows for a clean organization, increases security and provides a variety of services to achieve higher levels of implementation.

**Repository Pattern**

The repository model is used in library management systems to provide an abstraction layer between the application and the data persistence layer. This model encourages clear separation of concerns and standards. It provides a consistent and standardized approach to accessing and managing data, developing regulatory frameworks, measurement metrics and benchmarking.

**Error Handling and Logging**

Library management library includes middleware-based decision making to capture and track critical events, requests, and errors throughout the application process. This approach allows centralized access, making it easier to debug, monitor and analyse usage behaviour, performance and security. Agent software logging provides critical information for troubleshooting, auditing, and compliance purposes.

**Fluent Validation Package**

Fluent Validation package is used in the library manual administration process to simplify and improve the validation process. It provides simple and clear instructions for validating code by separating the validation logic from the written or view model. This separation of concerns promotes a cleaner and more manageable approach. Fluent Validation improves user experience by providing error messages for invalid validation.

**Getting Started**

To run the Library Book Management System locally, follow the instructions below:

**Prerequisites**

Web library management system based on ASP.NET MVC.

Relationship Object Model is Entity framework

Environment Requirements: Visual Studio 2017+, MVC, SQL server

**Steps**

Clone the repository: <https://github.com/vigneshkumar-957/LibraryManagement_1.git>

Navigate to the project directory:

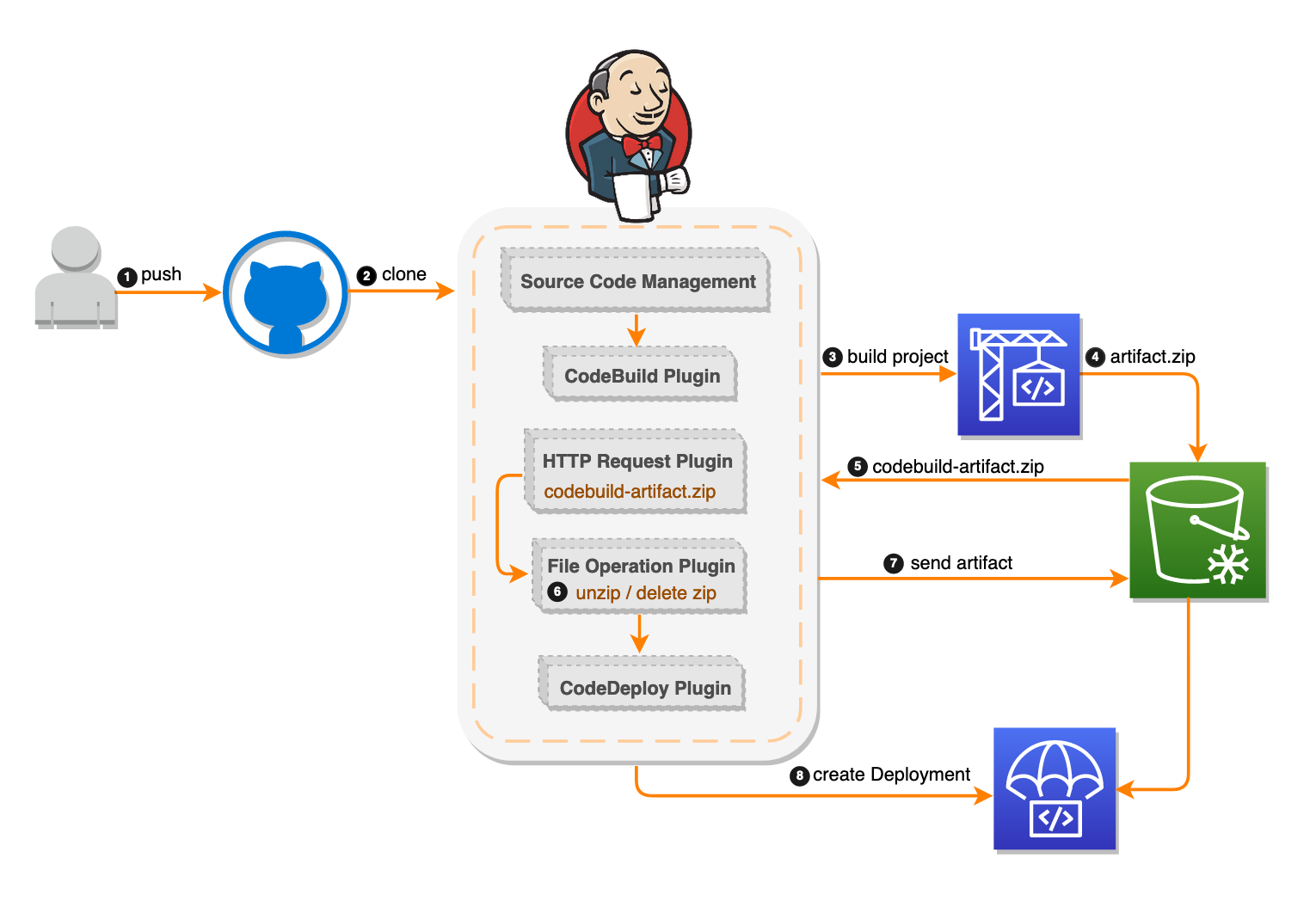
* Visual studio Open LMS solution
* Open the PM console
* Type cd LMS.DataAccess to enter the project folder
* Please enter **update-database –verbose** after configuring the sql server connection string
* Run IIS press

When the web page refreshes without the error, it can be run.

**CI –CD Implementation**

### Pre-requisites

1. Jenkins Server with Windows Slave attached.
2. AWS Account

**Jenkin file**

This is the sequential script for the pipeline to execute the stages one by one. Here we used 4 steps - Checkout, Build, Release, Deploy.

**Stage 1 -** Checkout: In this stage, we provide the URL and Git repository, branch as main and git credentials id. I pull the source code to workspace. Default workspace is - C:\ProgramData\Jenkins.jenkins\workspace

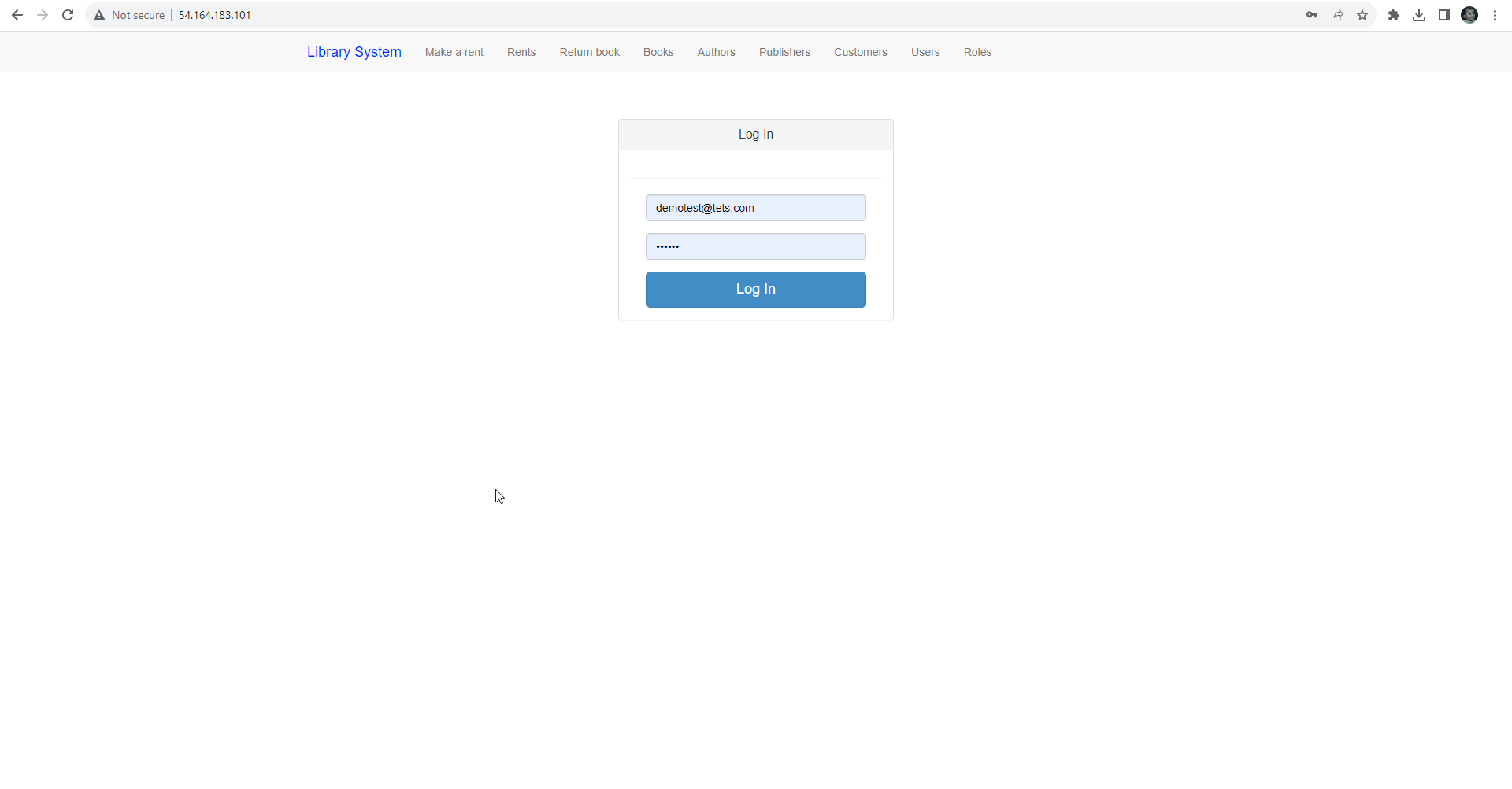
**Stage 2 -** Build: In this stage, I have build the specific project. Here LMS.API. Keep in mind you have to direct the location where sln file exists.

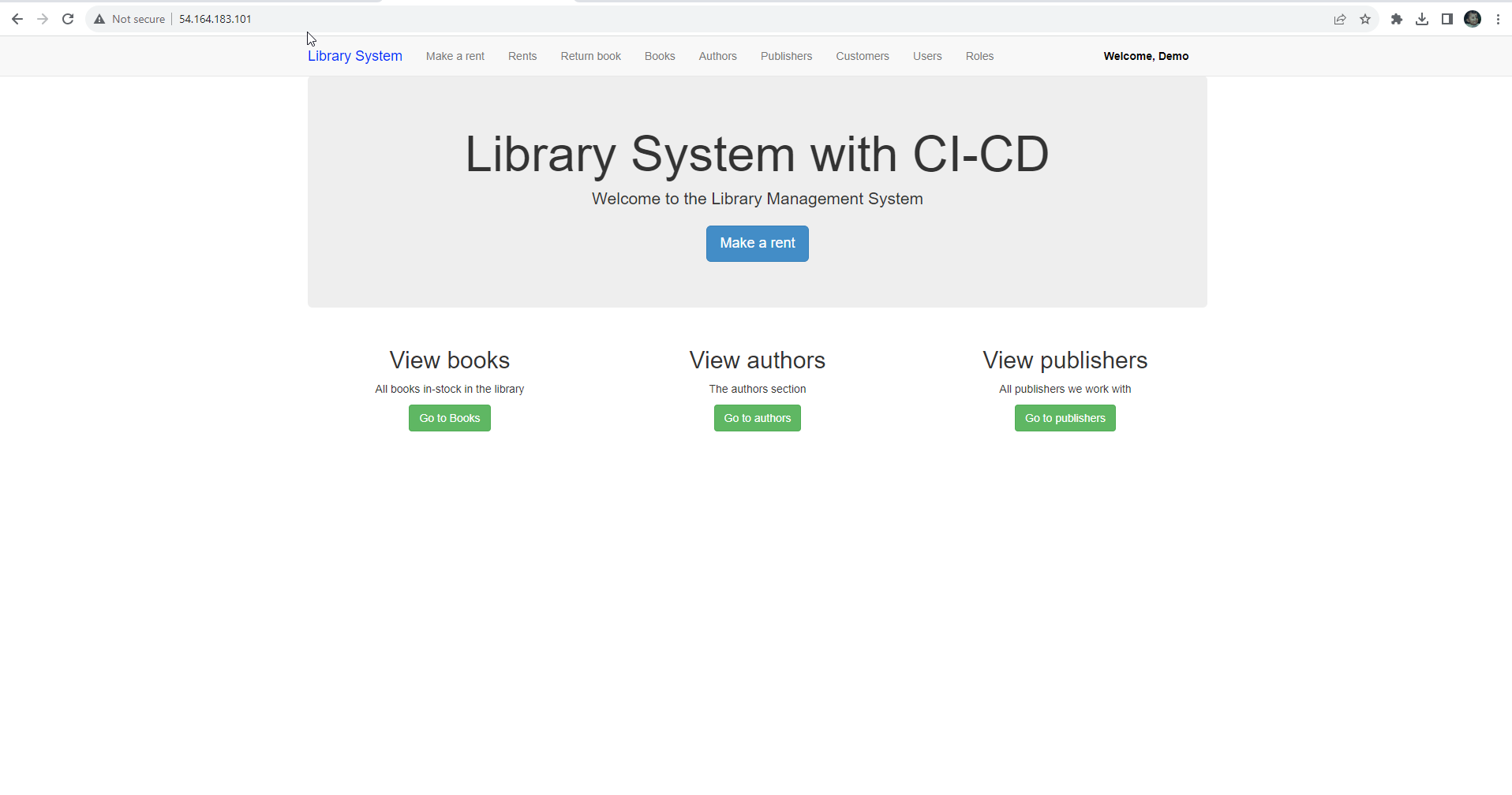
**Stage 3 -** Release: In this stage, A release package LMS.zip is created in the mentioned location which is provided on JenkinsProfile.pubxml. You may check each steps output using console after executing pipeline.

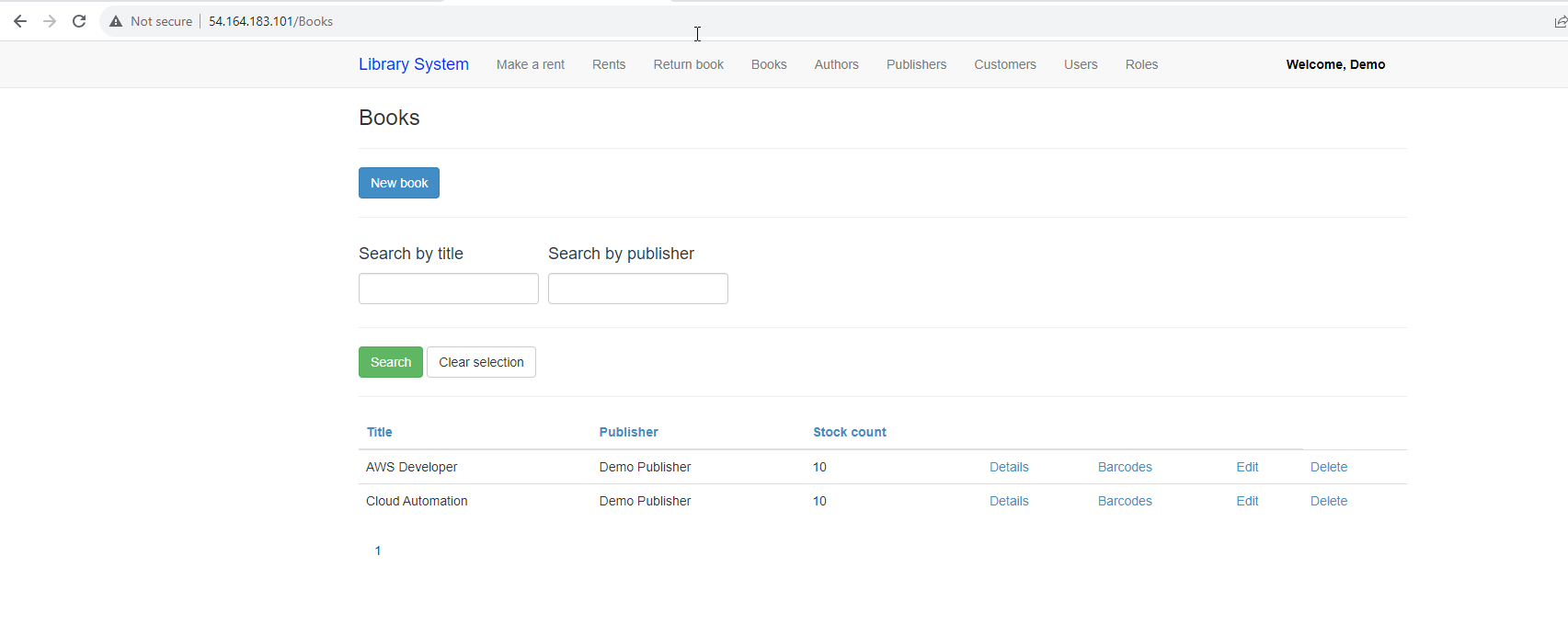
**Stage 4 -** Deploy: In this stage, first stop IIS. Then deploy package to IIS and then Start IIS again. So, I have used three bat commands here.

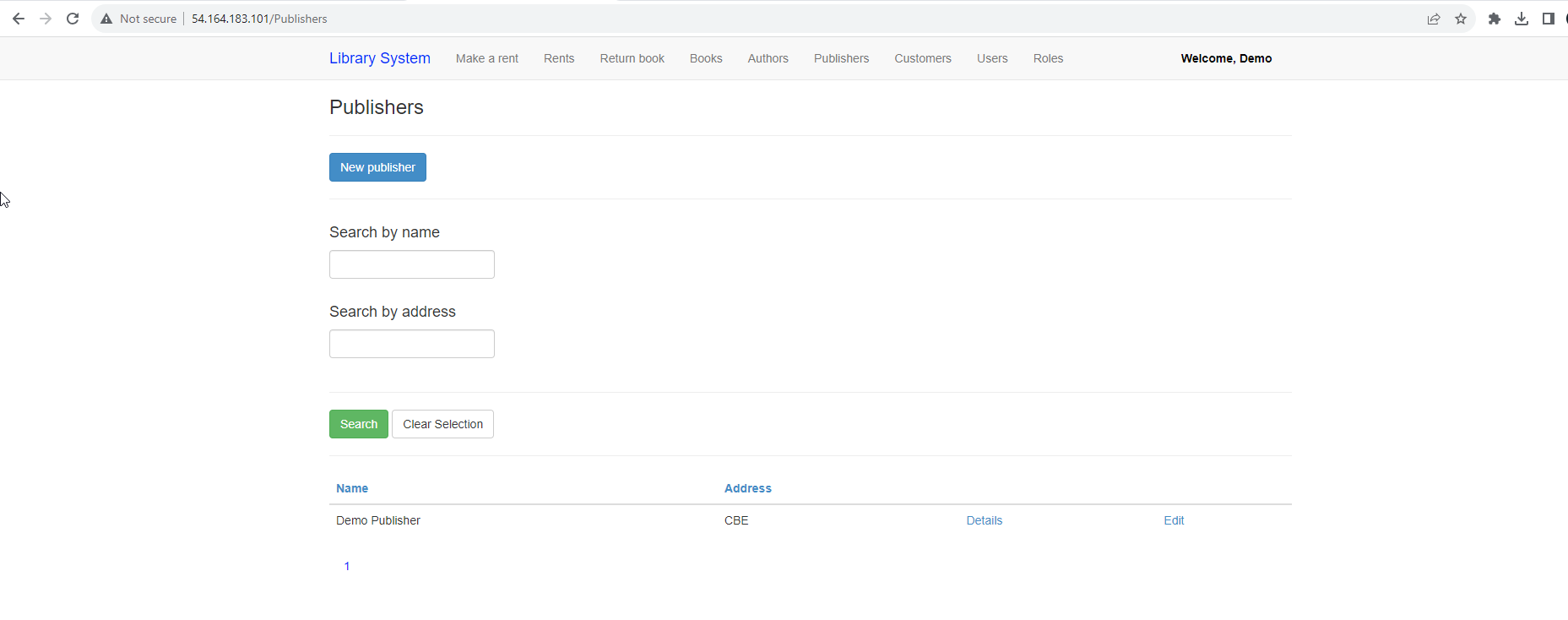
**Screenshots:**

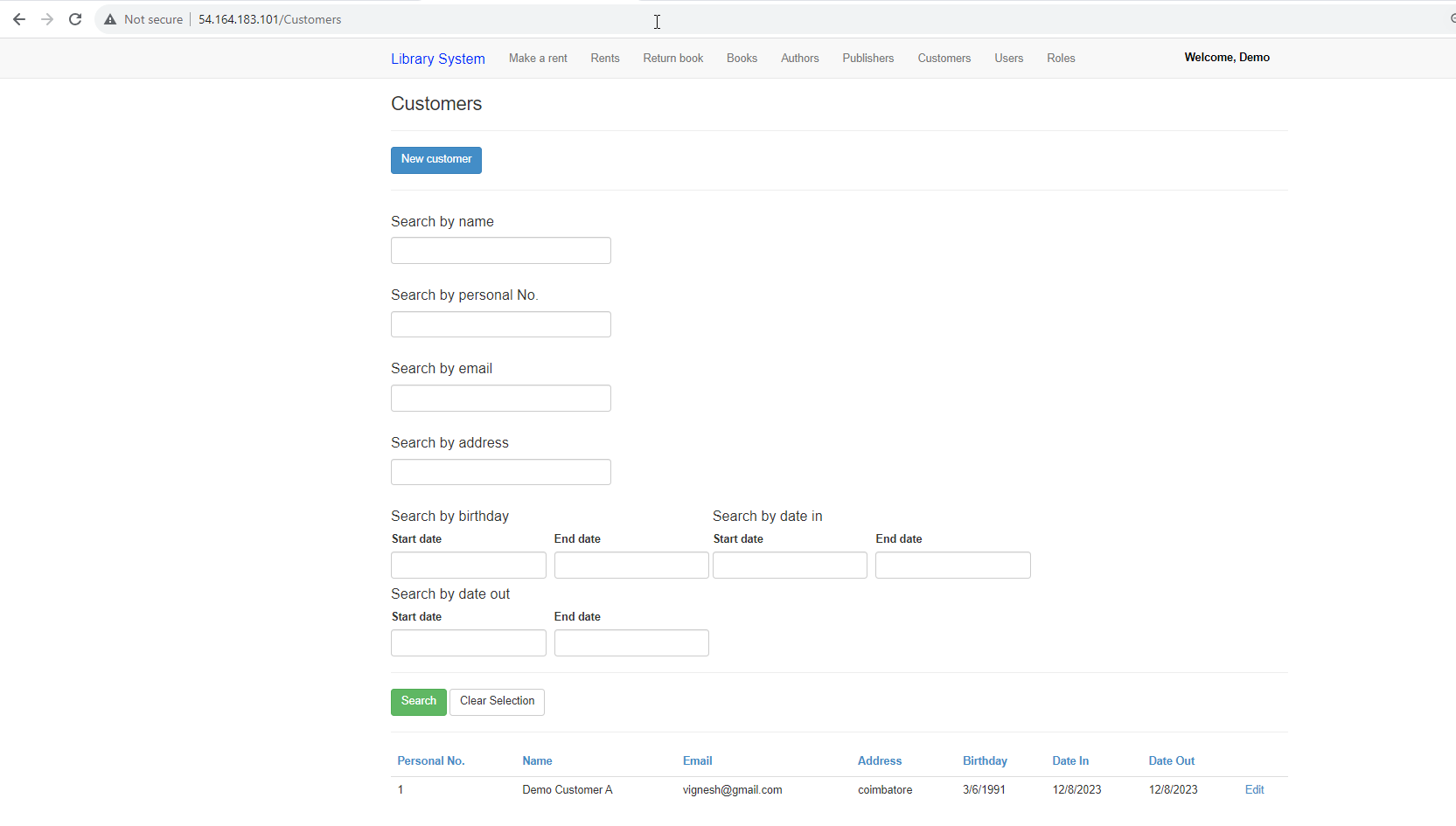
**Url:** [**http://54.164.183.101/**](http://54.164.183.101/)

****

****

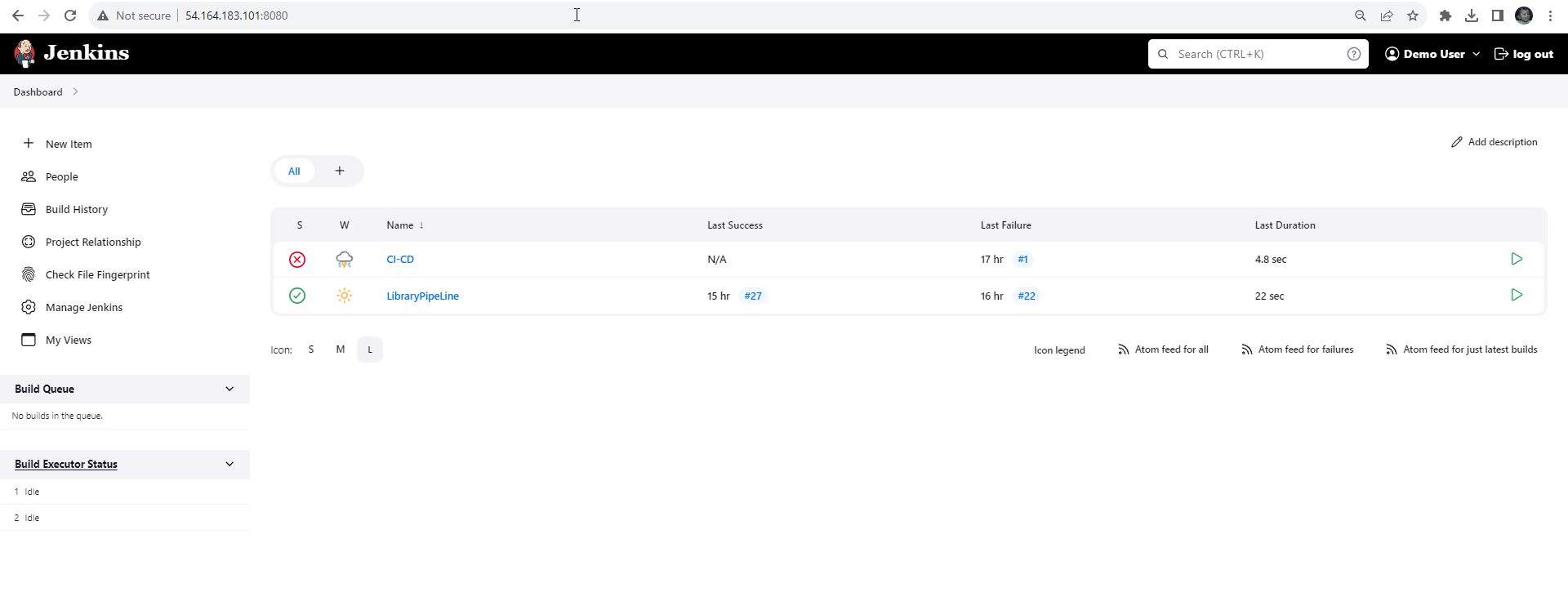


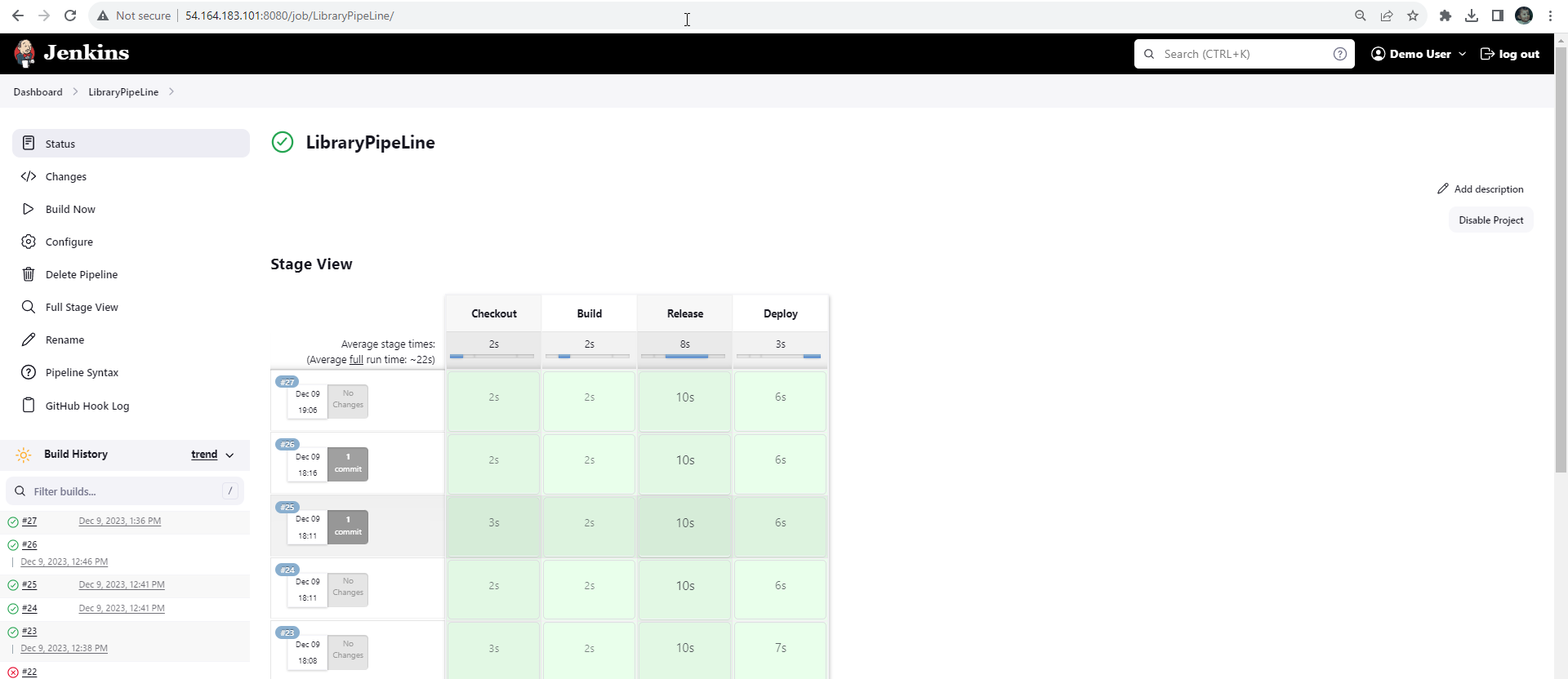
****

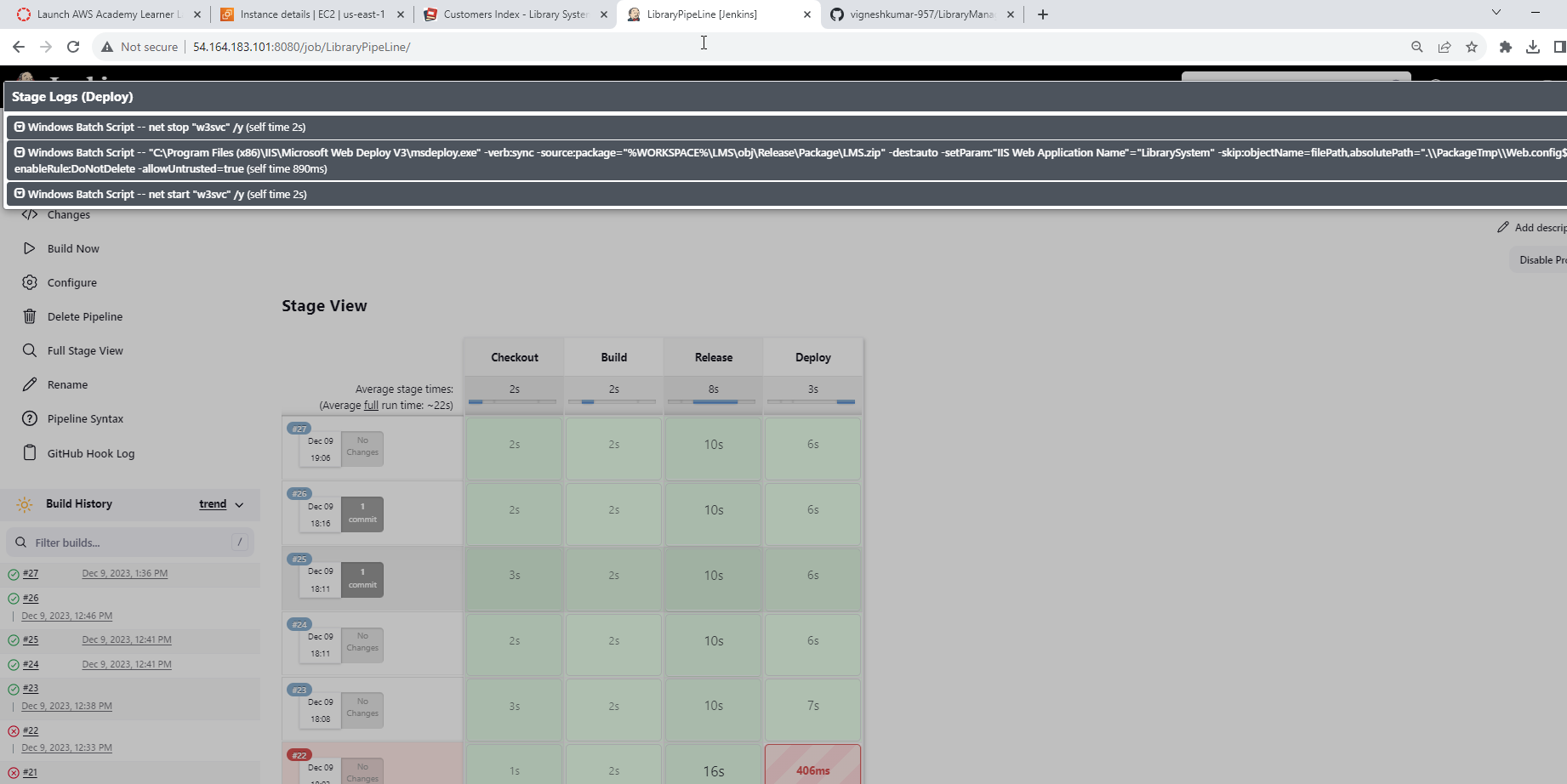
****

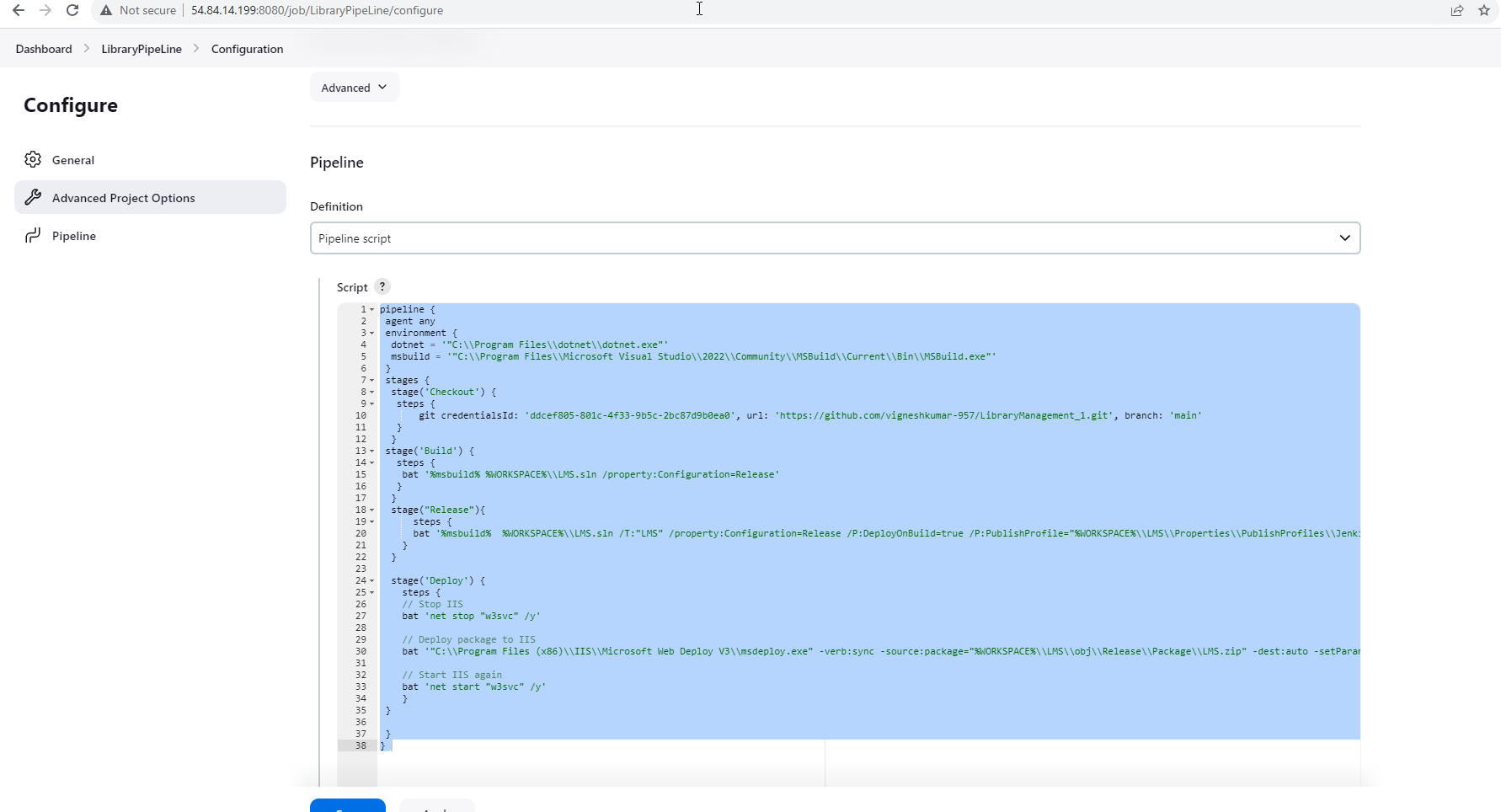
**CI-CD Scrrenshots**

**Url:** [**http://54.84.14.199:8080/**](http://54.84.14.199:8080/)



****

****

****

**Conclusion**

I have hosted the LMS application on 80 port on IIS and CICD Jenkins on 8080 port with the following URLs. This URL may get changes if aws EC2 turned off.

<http://54.164.183.101/>

<http://54.164.183.101:8080/>

**Reference**

## Pipeline : <https://www.jenkins.io/solutions/pipeline/>

**AWS** : <https://aws.amazon.com/blogs/devops/setting-up-a-ci-cd-pipeline-by-integrating-jenkins-with-aws-codebuild-and-aws-codedeploy/>