# Build, Run, Containerization & CI/CD with Github Actions:

### Build and run from local Dev IDE:

1. Unzip the *library-system.zip* Maven project

2. Open your favorite IDE (Spring Tool Suite, IntelliJ, Eclipse etc)

3. Import Project to your IDE.

4. Build and run the project (*\*set VM arguments: -Dspring.profiles.active=dev)*.

5. Access API documentation using this url:

<http://localhost:8080/library-system/swagger-ui/index.html>

### Build and run on local Docker Desktop (Windows):

1. Start Docker Desktop

2. Maven build:

> mvn clean install

3. To build docker image:

> docker build -f Dockerfile-dev -t library-system .

4. To create and run a new container from the created docker image on local Docker Desktop:

> docker run -p 8080:8080 library-system

5. To list the running containers:

> docker ps

6. To SSH into the running container:

> docker exec -it <CONTAINER\_ID\_FROM\_STEP\_5> sh

7. Access API documentation using this url:

<http://localhost:8080/library-system/swagger-ui/index.html>

8. To stop the running container:

> docker stop <CONTAINER\_ID\_FROM\_STEP\_4>

### Access to H2 Database console:

1. URL: http://localhost:8080/library-system/h2-console

2. JDBC URL: jdbc:h2:mem:aeonbank

2. Username: sa

3. Password: password

### Docker Hub Account (for docker image repo):

1. Create access token

Docker Hub Account -> Personal access token -> create access token: <https://app.docker.com/settings/personal-access-tokens>

* Name: github-access-token
* Expiration Date: 30 days
* Access permission: Read-Only
* Copy the generated access token and set to your Github Repository Secrets for DOCKER\_PASSWORD.

1. Create your Docker Hub repositories (https://hub.docker.com/repositories):

* Create a repository -> repo name: staging-library-system
* Create a repository -> repo name: production-library-system

### Github Account (configuration for code repo and ci/cd):

1. Create a Github Account and create a repository named ***library-system***.
2. Login to Github, click Settings -> Secrets and variables -> Actions

A black text on a white background

AI-generated content may be incorrect.

1. Create all required secrets for sensitive data:

A screenshot of a computer

AI-generated content may be incorrect.

Github Account -> Settings -> Secrets and variables -> Actions:

https://github.com/<username>/library-system/settings/secrets/actions

**Environment secrets:**

1. DB\_USERNAME: sa

2. DB\_PASSWORD: password

**Repository secrets:**

1. DOCKER\_USERNAME

2. DOCKER\_PASSWORD = docker hub access token (generate one if not yet generated before)

3. DOCKER\_USERNAME (your Docker hub username)

4. DOCKER\_REPO\_STAGING (value similar as Docker Hub repo name created above e.g. staging-library-system)

5. AWS\_ACCESS\_KEY\_ID (a temporary access key id to be used)

6. AWS\_SECRET\_ACCESS\_KEY (a temporary secret access key to be used)

7. ECR\_REPO (AWS ECR repository without tag)

8. ECR\_REPO\_STAGING (AWS ECR repository with tag)

9. EKS\_CLUSTER\_STAGING (AWS EKS Cluster name for staging namespace)

### Push Code to Github repository

1. Open command prompt and change directory to library-system project directory.
2. Run:
   1. git init
   2. git add –all
   3. git commit -m "added initial codebase"
   4. git branch -M main
   5. git remote add origin https://github.com/<YOURUSERNAME>/library-system.git
   6. git push -u origin main
3. Create staging branch for staging deployment.

### Github Actions

After code push to Github, Github action workflow files will be triggered and run the jobs/steps based on branch (staging and main) to:

*\*main = production*

CI/CD Workflow files under library-system/.github/workflows:

1. **Docker Hub CI/CD & Local Deploy** (ci-cd-docker-hub.yml)
   1. After job completed successfully, you can docker pull the image from your Docker Hub repository and run locally

> docker pull chuasoonee/staging-library-system:latest

> docker run -d -p 8080:8080 chuasoonee/staging-library-system:latest

* 1. Access <http://localhost:8080/library-system/swagger-ui/index.html>

1. **AWS ECR & EKS CI/CD** (ci-cd-aws-ecr-eks.yml)
   1. Must have AWS EKS cluster created
   2. Create namespace staging and prod
   3. End to end build, test, push docker image to AWS ECR, deploy to AWS EKS.

> kubectl get svc

> kubectl get po

Verify the pod is running

