Rest of the content is already explained in the Readme.md file. Here, we are doing the Github Action setup as we are providing the docker username and password that is required in yml ile:

Graphical user interface, text, application, email

Description automatically generated

line no, 42, we need docker username and password. That setup we will configure here.

Steps:

1. Go to your repo and go under security.

Graphical user interface, text, application, email

Description automatically generated

1. Under security tab, click on action under “secrets and variables.”

Graphical user interface, text, application, email

Description automatically generated

1. Click on “new repository secret.”

Graphical user interface, text, application, email

Description automatically generated

1. Provide the same variable name defined in yml file:

Graphical user interface, text, application, email

Description automatically generated

1. Add both username and password for docker hub:

Graphical user interface, text, application, email

Description automatically generated

Once this is done, we need to create **“.github/workflows/maven.yml**” file.

**Working maven.yml file:**

name: project cicd flow

on:

push:

branches: [ feature/kubernetes-exercise-1 ]

pull\_request:

branches: [ feature/kubernetes-exercise-1 ]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up JDK 17

uses: actions/setup-java@v1

with:

java-version: '17'

distribution: 'adopt'

cache: maven

- name: Build with Maven

run: mvn clean install

- name: Test cases execution

run: mvn clean test

#if condition, if the branch is equal to main then create the docker image for it and push to docker hub

#bookmarker-api-jib:latest image name defined in docker push command below will pick from image

# name defined in pom. file in JIB plugin. If you want to user dockerFile then you should define

# file path but here we are using JIB so it's picking from pom.

- if: ${{ github.ref == 'refs/heads/feature/kubernetes-exercise-1' }}

name: Build and Publish Docker Image

run: |

echo "This job runs in custom branch"

./mvnw clean package jib:dockerBuild -DskipTests

docker login -u ${{ secrets.DOCKER\_USERNAME }} -p ${{ secrets.DOCKER\_PASSWORD }}

docker push ${{ secrets.DOCKER\_USERNAME }}/bookmarker-api-jib:latest

Make sure spring boot project folders like this:

Graphical user interface, website

Description automatically generated

Earlier, I had folder like “Documents/spring-boot-projects/branch name” and due to that my Github action was not working.

The image name we defined online

no 37, picked from pom.xml file jib plugin:

Text, letter

Description automatically generated

Timeline

Description automatically generated

Once that is done, you are good to go….

Graphical user interface, text, application, email

Description automatically generated

16. Now, we will add post method for bookmark. For bookmark method, we are not giving all details. We are passing only 2 arguments so that's why I have created the "BookmarkRequest" class for taking the input and add @Valid @NotEmpty annotation.

17. Now, if there is something error in your payload then it will through big error, that’s good

but we need an accurate error due to which our post method is failed. So, we need to add one annotation.

**to fix it:**

<problem-spring-web.version>0.27.0</problem-spring-web.version>

**<dependency>**

**<groupId>org.zalando</groupId>**

**<artifactId>problem-spring-web-starter</artifactId>**

**<version>${problem-spring-web.version}</version>**

**</dependency>**

Before applying this annotation, the error will be like this:

Graphical user interface, text, application

Description automatically generated

**After adding this dependency:**

**Graphical user interface, text, application, email

Description automatically generated**

**Docker-Compose.yml:**

Until now, we create the common docker-compose file for all the services like for spring boot app along with databases that we are using. If you want to change anything in your spring boot application, then you down all the services that are running in docker-compose however we just need only application to restart. So, this is not a right approach. Instead of it, we can create 2 docker-compose files in which under one file we mention information regarding the spring boot app and for other services we will create a different docker-compose file.

So, we created 2 files:

1. docker-compose.yml: it contains all the services like DB.
2. docker-compose-app.yml: It contains only spring boot app.

Now, spring boot app is dependent on the service that is present in another docker file. So, here is the way to run multiple docker files together:

**docker-compose -f docker-compose.yml -f docker-compose-app.yml up -d**

**logs:**

**docker-compose -f docker-compose.yml -f docker-compose-app.yml logs -f**

Still, these commands are not easy to remember so we can create a shell script to run these commands:

Now, will create a shell script named as “run.sh”

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

In this file, we will write the different methods to execute our spring boot app.

Steps:

1. create “run.sh” file
2. execute “chmod +x run.sh”
3. run “./run.sh” file

when you execute this file, it will by default call the “start” method written inside this file as we mentioned the default action to the “start” method.

A picture containing logo

Description automatically generated

It will run both the docker-compose files and our app will up and work.

But if you want to restart the existing running application then you can do it by providing the method name as:

For stop the containers: **“./run.sh stop”**

For starting only infra: “**./run.sh start\_infra”**

Meantime, when we run our application using “run.sh” file, it will run in docker. So, if we want to see tables are getting created or not or if you want to see other things you need to login into the docker container. It is time consuming so to avoid this, we can run spring boot app in local profile.

Steps:

1. Run the “**./run.sh start\_infra”** // it will run the docker postgres.
2. Create the “application-local.properties” file.

Graphical user interface, text, application

Description automatically generated

Added the host as localhost because docker postgres DB is running on local docker and assigned the local port to 15432 as we assigned it docker-compose file.

Run the spring boot in local profile: **-Dspring.profiles.active=local**

Now, it’s time to create frontend page for this spring boot app. So, for this we are using “next js”.

Steps:

1. Download <https://nodejs.org/en>
2. After installing node, run this command under our spring boot app folder:

“**npx create-next-app bookmarker-ui-nextjs --ts**”

**Create-next-app** is syntax to create next app.

**bookmarker-ui-nextjs** is our next app name.

**--ts** means we are using typescript over the JavaScript.

Graphical user interface, text, application

Description automatically generated

1. Now, go inside this folder “**nextjs-demo**” and run the “**npm run dev**”
2. Our application will start on localhost:3000 port.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Inside the nextjs folder, also install the bootstrap server using command:

**npm i** [**bootstrap@5.3.0-alpha3**](mailto:bootstrap@5.3.0-alpha3)

1. Bootstrap server, we use for responsive websites. So, we have added it into nextjs project. To load this dependency, inside the project, import it inside the **“\_app.tsx”**

**Graphical user interface, text, application

Description automatically generated**

1. Import the bootstrap server in this file:

Graphical user interface, text, application, chat or text message, email

Description automatically generated

1. Once, bootstrap is imported. Now we can use it in global project:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Check the button, bootstrap server is working as expected.
2. Meantime, not doing the UI work as we need to learn about next, react.

**Kubernetes:**

As of now, we were using docker images to run the spring boot application on local machine. For development purpose, it works fine, whenever we have issue, we can restart the docker file and fix that, but it is not possible in production side. We need 0 sec downtime for our application and here Kubernetes comes into the picture. For higher availability and scalability, we can scale up/down the application any time and if there is any need to fix the issue we can run the multiple instances of same application.

For creating the cluster, we will create the shell scripts for it:

**Kind-config.yml:**

Graphical user interface, text, application

Description automatically generated

**create-cluster.sh:**

Graphical user interface, text, application, email

Description automatically generated

**delete-cluster.sh:**

**Graphical user interface, text, application, email

Description automatically generated**

Once you create the cluster, download “lens” software for Kubernetes UI:

Download link: <https://k8slens.dev/>

Credentials:

Email: mani\*\*\*\*\*

Pass: Si\*\*\*\*1

Graphical user interface, text, application

Description automatically generated

Commands for Kubernetes:

1. To check does pods exist:

**kubectl get pods**

1. To create pods:

**kubectl run bookmarker-api --image=maninder40407/bookmarker-api-lib --port=8080**

**bookmarker-api:** pod name

**maninder40407/bookmarker-api-lib:** image name