FULL STACK DEV

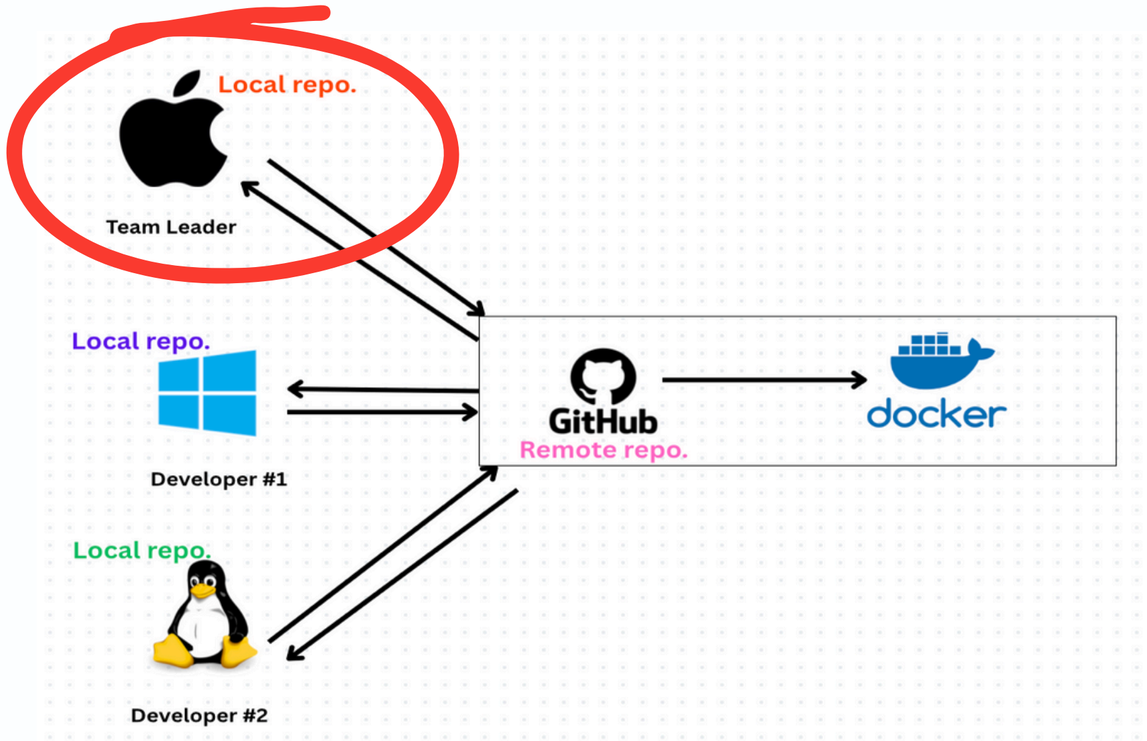
Team Leader - Build the Docker Image and Run Locally

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

TEAM LEADER



The Team Leader :

- Initialises the project
- Put on GitHub
- Put on DockerHub

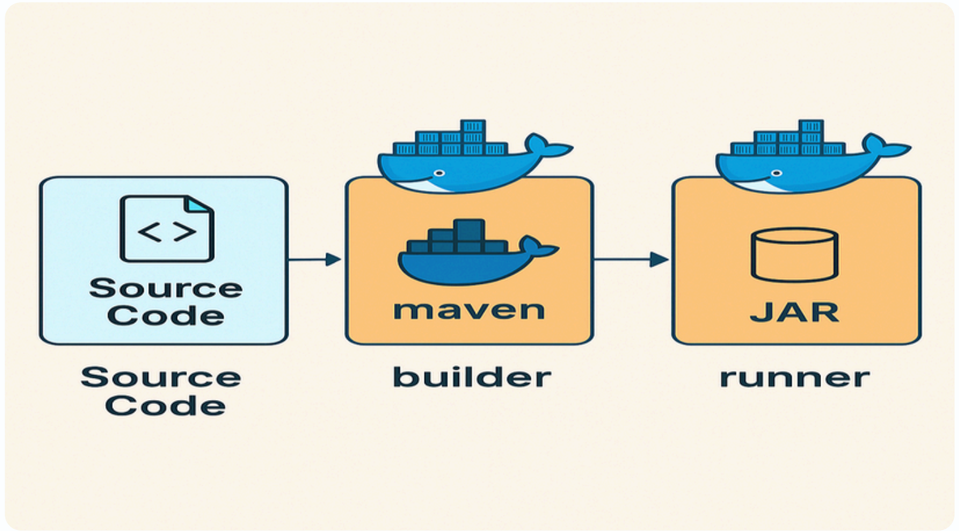
FIRST THING FIRST : DOCKER ?

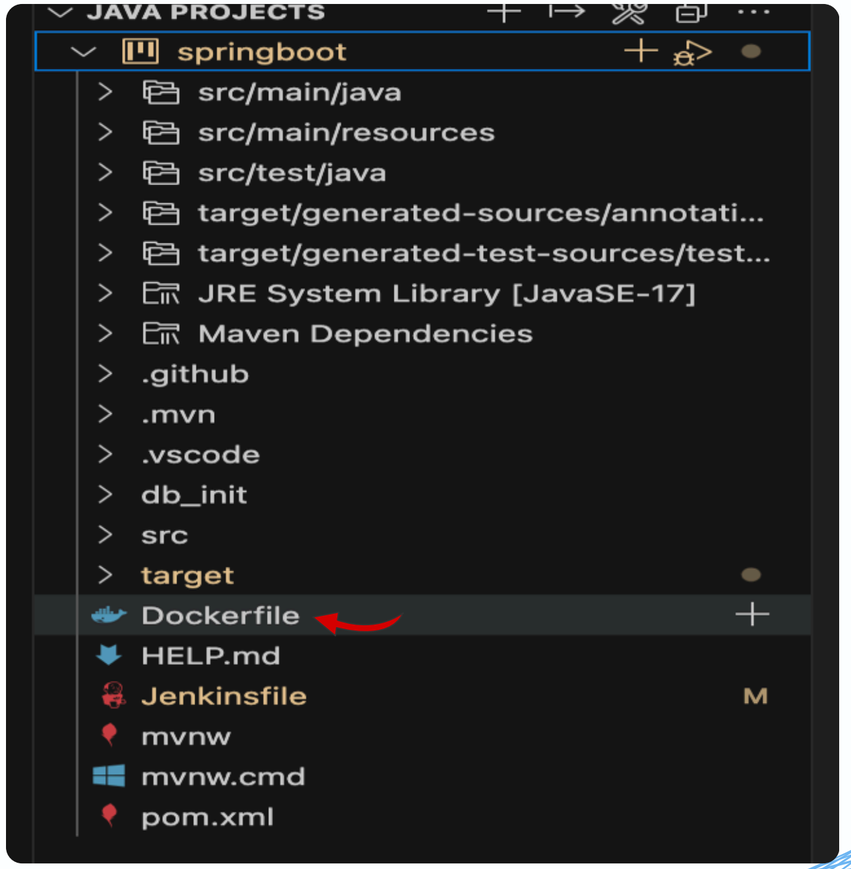
- A Dockerfile contains instructions to build a Docker image.
  - defining a self-contained, reproducible environment for our Spring Boot application.

- There are two ways we can write the Dockerfile:
  - **Single stage** : You write a single sequence of instructions
    - the image tends to be bigger so it is less efficient.

  - **Multi stage** : You define separate stages, typically one for building your app and another for the final runtime environment.
    - More complex but powerful.

DOCKER - HOW IT WORKS ?





SINGLE STAGE

```
# Use an official OpenJDK runtime as a parent image
# This Dockerfile sets up a Java application environment,
# builds the application using Maven, and runs it.
# Single stage build for simplicity (but larger images).
# Author : Rajeev Khoodeeram

FROM openjdk:17-jdk-slim

WORKDIR /app
COPY . .
RUN ./mvnw clean package -DskipTests && cp target/*.jar app.jar

EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

- *Here everything — building and running — happens in the same image, making it larger and less efficient, since all build tools (like Maven) remain in the final image.*

MULTI-STAGE

- What happens during docker build:

- **Stage 1 (builder)**
  - Maven and other heavy build dependencies are installed.
  - The project is compiled, and a .jar file is generated.

- **Stage 2 (runner)**
  - Starts from a much smaller image (openjdk:17-jre-slim).
  - Only copies the compiled JAR from the builder stage (COPY --from=builder).
  - No build tools like Maven are included.

STAGE 1: BUILD THE APPLICATION JAR

- FROM maven:3.8.7-openjdk-17-slim AS builder
- WORKDIR /app

- # Copy the Maven project file first to leverage Docker cache
- COPY pom.xml .

- # Download dependencies to cache them
- RUN mvn dependency:go-offline

- # Copy the rest of the source code
- COPY src ./src

- # Package the application into a JAR file
- RUN mvn clean package -DskipTests

STEP 1: THE BUILD STAGE (BUILDER)

- The first stage is responsible for compiling our Java code and creating the final executable .jar file. It's like a temporary workshop where we get all the tools we need to create our product.
  - **FROM**: a base image that has all the necessary tools: a Java Development Kit (JDK) and Maven.
  - **WORKDIR**: set the working directory inside the container, where all subsequent commands will run.
  - **COPY**: We copy the pom.xml first. This is a crucial optimization.
  - **RUN**: We download the project dependencies using mvn dependency:go-offline. This caches them.
  - **COPY**: We copy the rest of the source code.
  - **RUN**: We run the final Maven command to package the application into a .jar file.

STAGE 2: CREATE A MINIMAL RUNTIME IMAGE

- FROM openjdk:17-jre-slim AS runner

- WORKDIR /app

- # Copy the JAR file from the builder stage
- COPY --from=builder /app/target/*.jar app.jar

- # Expose the port the Spring Boot application runs on
- EXPOSE 8080

- # The command to run the application
- CMD ["java", "-jar", "app.jar"]

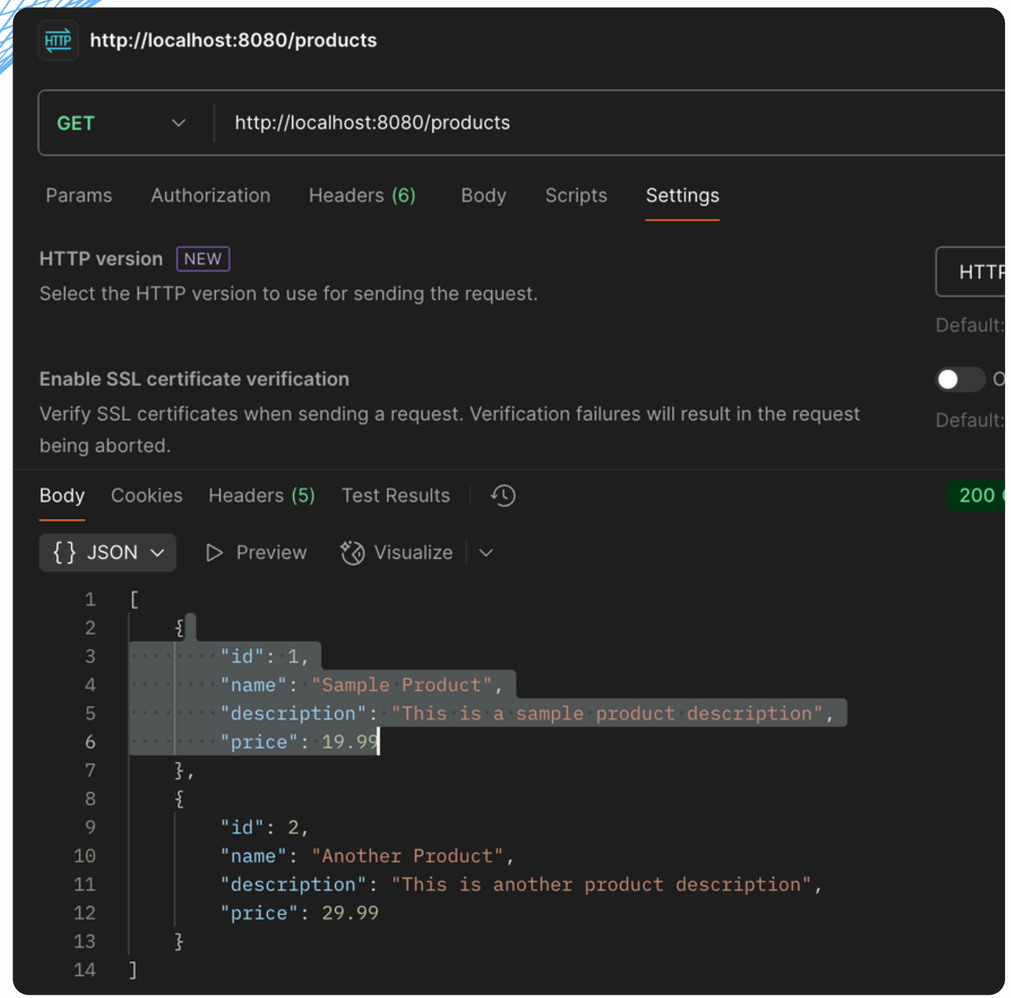
STEP 2: THE RUNTIME STAGE (RUNNER)

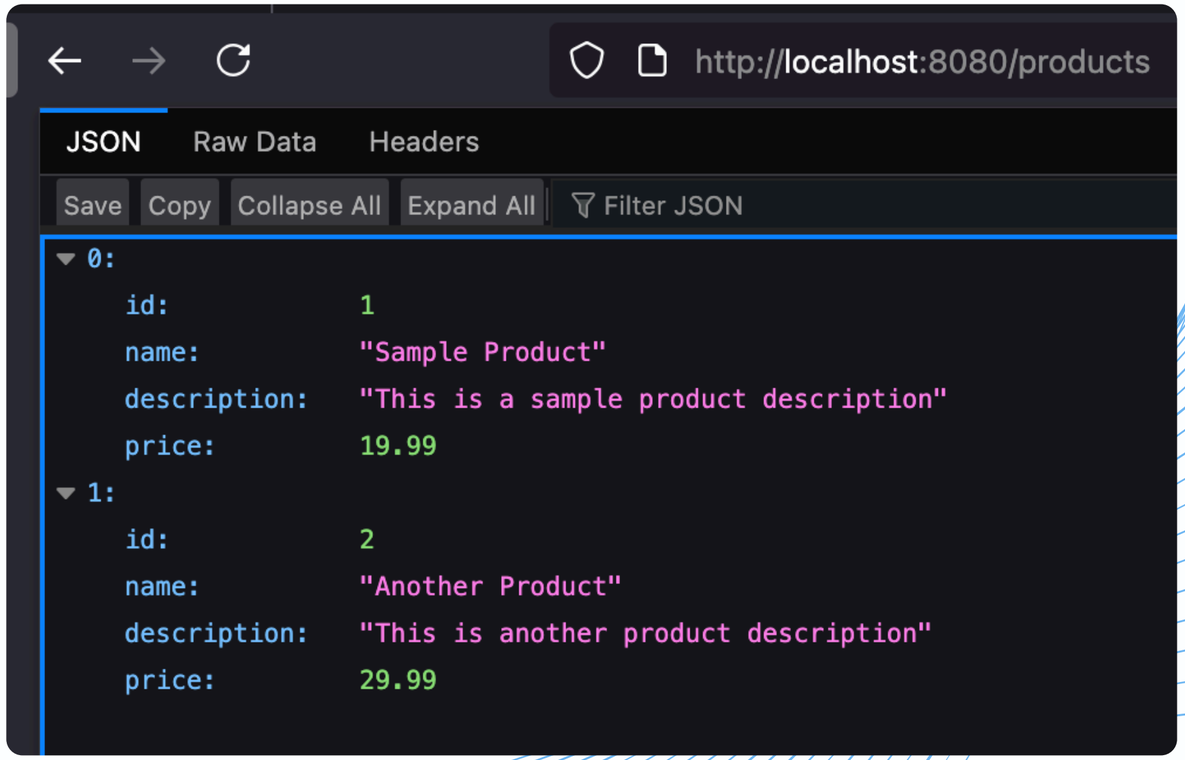
- This is the final, production-ready stage. Its goal is to be as small and secure as possible.
- FROM: A base image that only contains the JRE, not the full JDK.
- WORKDIR: We set the working directory for our final application.
- COPY: We copy the .jar file that was built in the builder stage.
- EXPOSE: We inform Docker that our application listens on port 8080.
- CMD: We specify the command that will be executed when the container starts.

SUMMARY : TEAM LEADER

- Create the Spring boot project
- Run the app without docker
- Initialize the local repo.
- Write the Dockerfile
- Build using maven
- Build and test the app with docker
- Create an online repo. (Show on Github)
- Use git to commit the Spring boot project to the main branch on the remote repo.

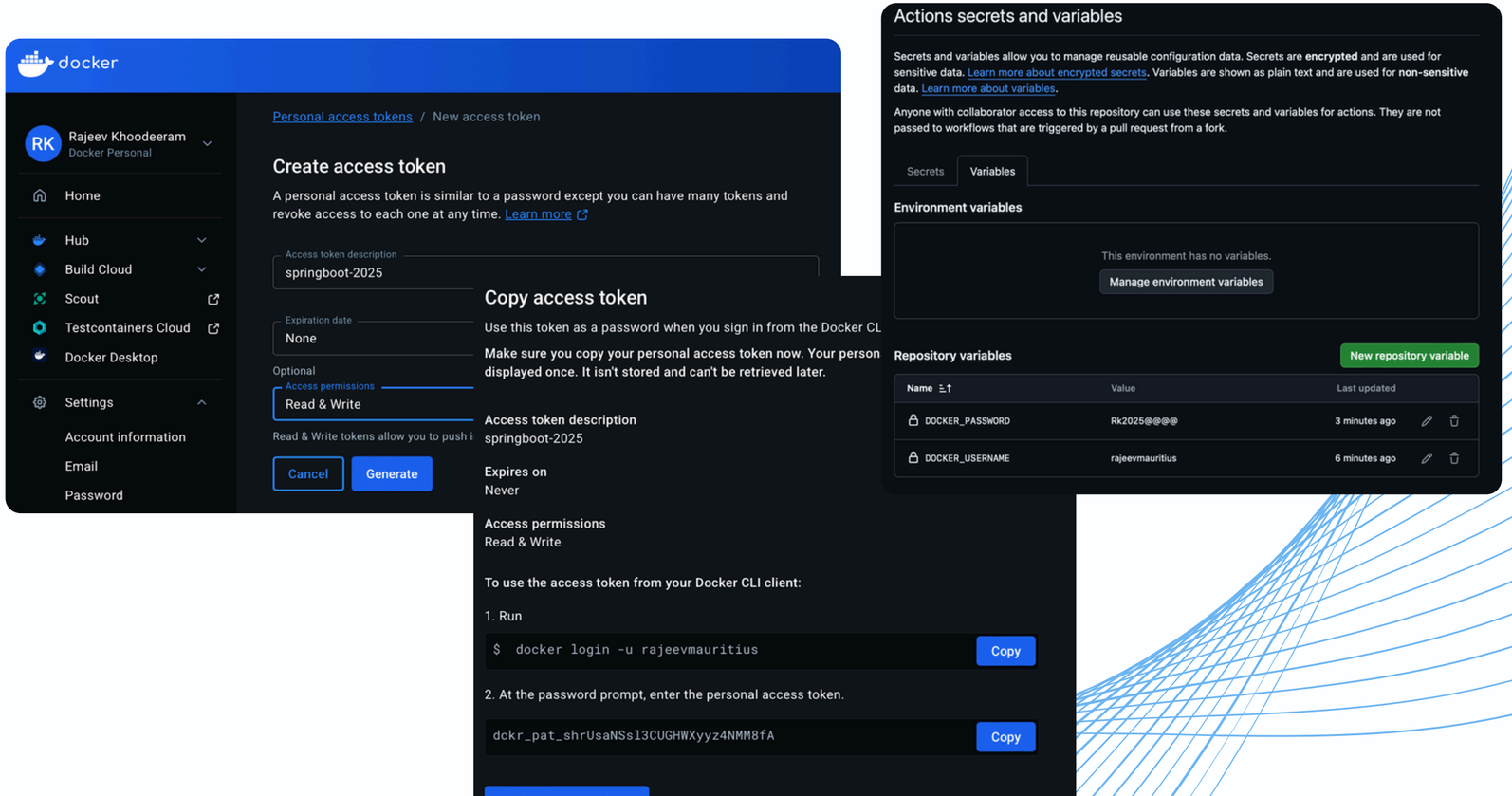
TESTING


http://localhost:8080/products
GET
http://localhost:8080/products
Params
Authorization
Headers (6)
Body
Scripts
Settings
HTTP version
NEW
Select the HTTP version to use for sending the request.
Enable SSL certificate verification
Verify SSL certificates when sending a request. Verification failures will result in the request being aborted.
Body
Cookies
Headers (5)
Test Results
200
JSON
Preview
Visualize
[
{
"id": 1,
"name": "Sample Product",
"description": "This is a sample product description",
"price": 19.99
},
{
"id": 2,
"name": "Another Product",
"description": "This is another product description",
"price": 29.99
}
]


http://localhost:8080/products
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
id: 1
name: "Sample Product"
description: "This is a sample product description"
price: 19.99
1:
id: 2
name: "Another Product"
description: "This is another product description"
price: 29.99

HOW TO PUSH TO DOCKER

- Make sure, you have created the docker repo.
- Build the container :
  - **sudo docker build -t rajeevmauritius/sbdockergit:v1 .**
- Commit to repo.
  - **docker push rajeevmauritius/sbdockergit:v1**
- If needs to login, then
  - **docker login**
- View the container
  - **docker ps -a**



FULL STACK DEV

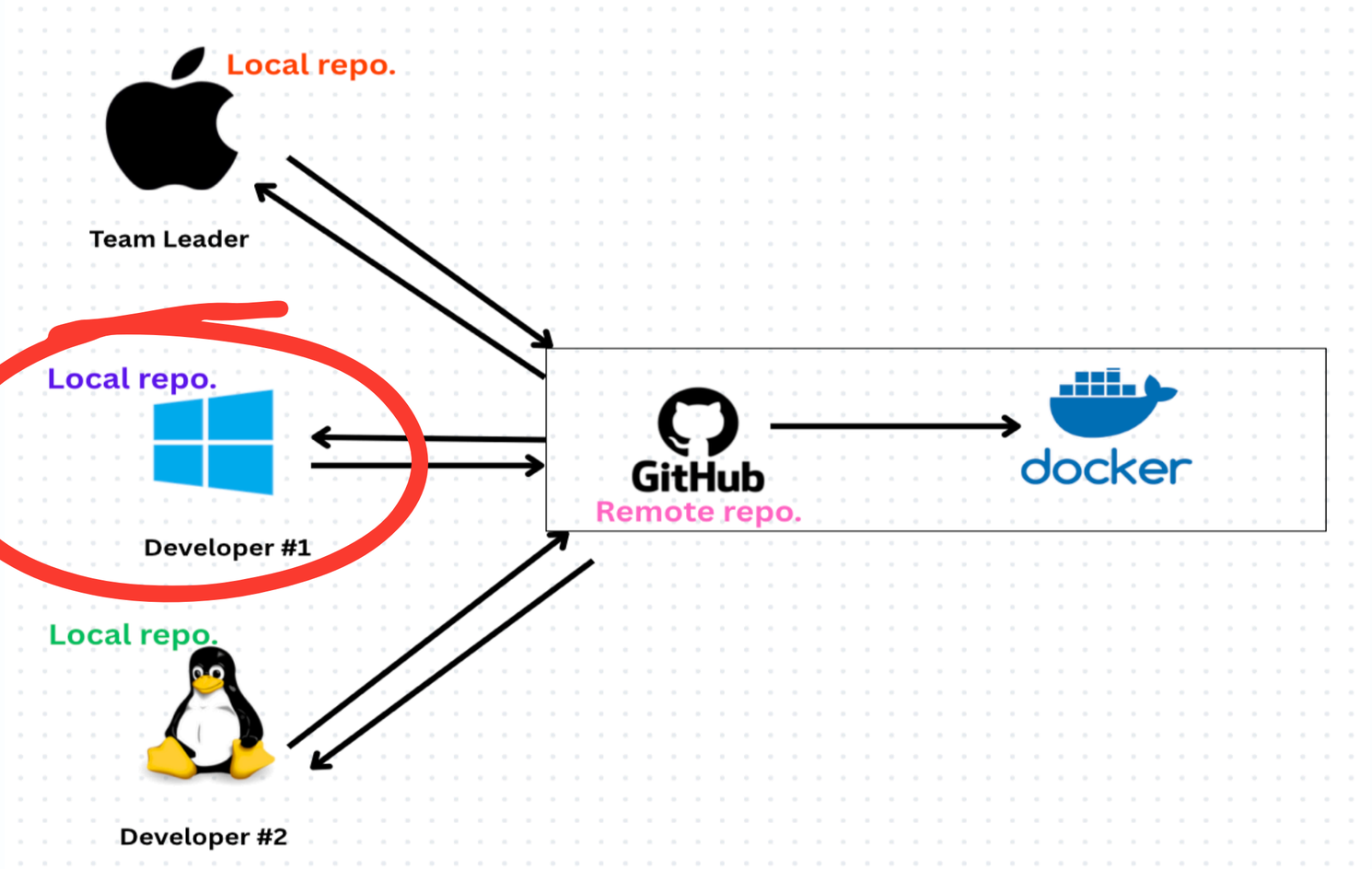
Developer #1 - Working on Windows and uses Git

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

DEVELOPER#1


Local repo.
Team Leader
Local repo.
Developer #1
Local repo.
Developer #2
GitHub
Remote repo.
docker

DEVELOPER#1 - ENVIRONMENT

- Install all the required tools
  - jdk 21/24
  - VS Code
  - git

- Make sure Dev#1 has a git account
  - Will be asked to verify his credential and commit to the git repository

GET A COPY OF MAIN PROJECT

- Create a folder where you want to put the project
  - cd in that folder

- git clone --branch feature/initial-project --single-branch https://github.com/rajeev-khoodeeram/JavaFullStack.git
  - (where initial-project should be MAIN in reality)

- Do a clean maven if any problem with Java (we are using JDK 24 here)
  - .\mvnw clean install

- Run the spring boot project and checks if the /products

MODIFY PRODUCTCONTROLLER

- Dev#1 works in this own branch /feature and commits
  - git checkout -b feature/product-controller-updated-getProductbyId

- Changes the ProductController
  - add a method to retrieve a product by id (getProduct)

- Now the only change is this ProductController file

```
@GetMapping("/products/{id}")
public ResponseEntity<Product> getProduct(@PathVariable int id) {
  Product product;
  product = productService.getProductById(id);

  if (product.getName().length() == 0) {
    return ResponseEntity.badRequest().build(); // we are assuming
  }
  return new ResponseEntity<>(product, HttpStatus.OK);
}
```

AUTHSERVICE

- In VS Code; open terminal to execute the following :
  - git add .

- # will be required to use git (applied only when using for the first time)
  - git config --global user.email "khoodeeram.rajeev@gmail.com"
  - git config --global user.name "Developer #1"

- git commit -m "getProductById added by Developer #1"
- git push https://github.com/rajeev-khoodeeram/JavaFullStack.git feature/product- controller-updated-getProductbyId
  - if developer 1 is already connected to github...and Team leader has allowed him as collaborator

- Now on github, Developer #1 will create a pull request and add a message

TYPICAL PR → MERGE FLOW

- **Dev #1**
  - Creates a feature branch
  - Commits code
  - Opens a Pull Request (PR) into main (or develop in Git Flow)
- **Team Leader**
  - Reviews the code (comments, requests changes if needed)
  - Approves the PR once it meets quality standards
- **Merging**
  - Option A (common in small/medium teams):
    - The team leader clicks “Merge” after approval
  - Option B (common in larger or more autonomous teams):
    - The PR author merges it themselves after approval (if branch protection rules allow it).

FULL STACK DEV

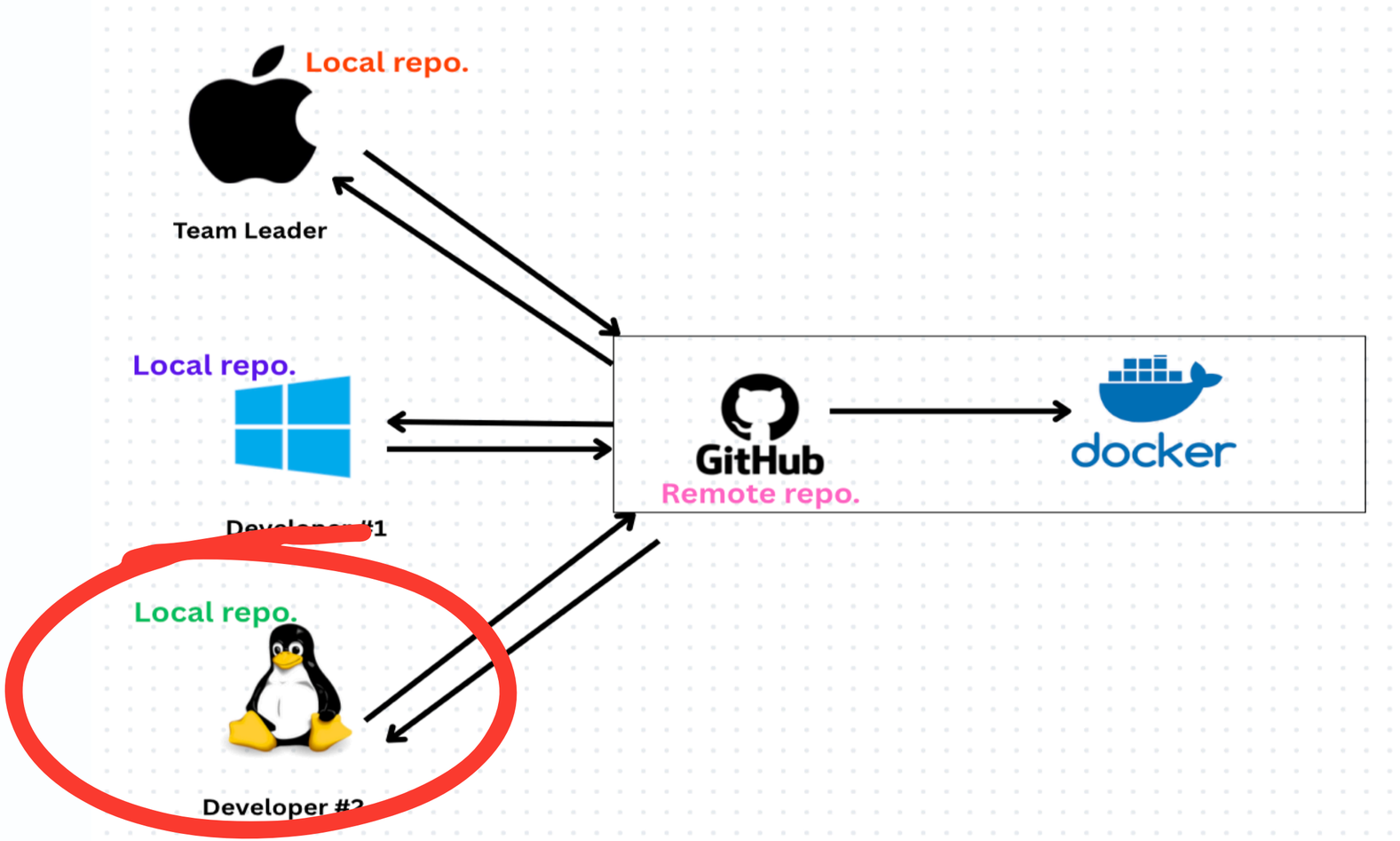
Dev#2 : Working on Ubuntu (docker)

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

DEVELOPER #2


Local repo.
Team Leader
Local repo.
GitHub
Remote repo.
docker
Local repo.
Developer #2

DEV#2 : ENVIRONMENT SETUP (1)

- **Step 1: Install Git and Docker (if not already installed)**
  - sudo apt update
  - sudo apt install git docker.io -y

- **Then start and enable Docker:**
  - sudo systemctl start docker
  - sudo systemctl enable docker

- **Give current user permission to run Docker without sudo:**
  - sudo usermod -aG docker $USER
  - newgrp docker

DEV#2 : PROJECT SETUP

- **Step 2: Clone the GitHub Repository**

- Create a folder in which the project will be stored
- cd inside this folder
  - git init .
  - git clone --branch feature/initial-project --single-branch https://github.com/rajeev-khoodeeram/JavaFullStack.git

- Open the folder in VS Code

- Add all required extensions (ex Java extension pack, etc)

GET THE PROJECT !

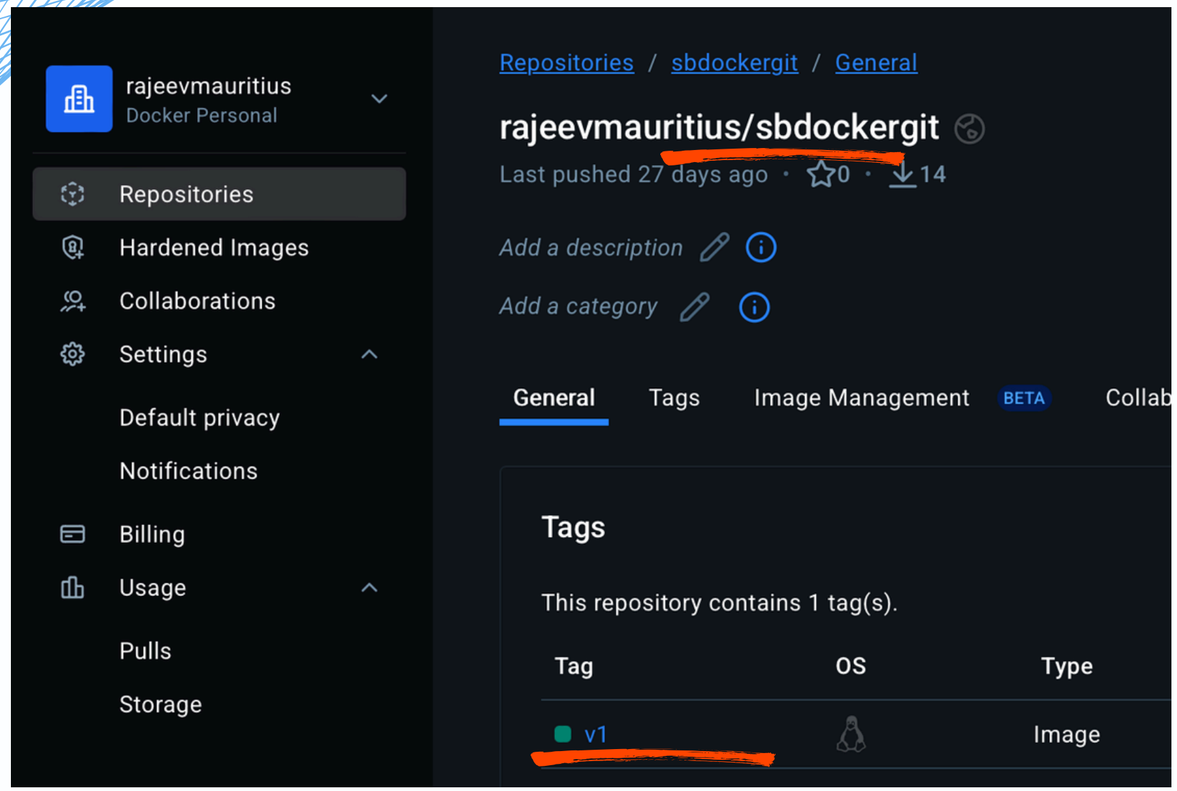
- You can check which user you are :
  - **sudo docker whoami**
  - **sudo docker info**

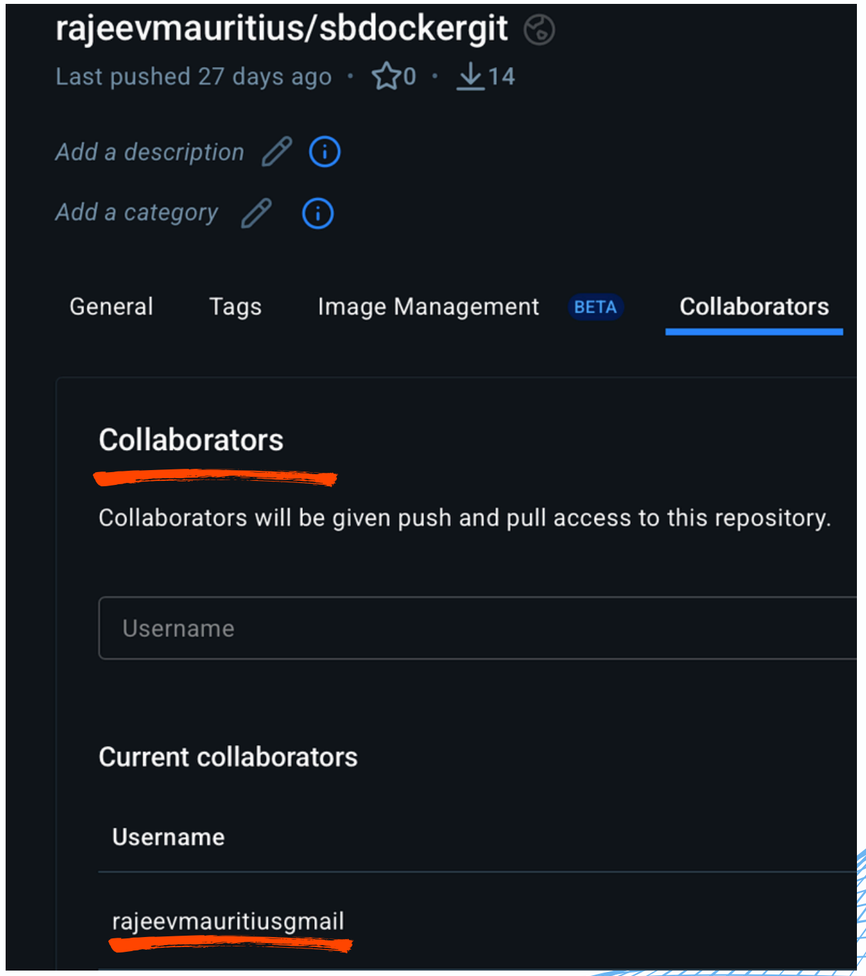
- Ubuntu user (Developer 2) is user : rajeevmauritiusgmail

- If ask for login :
  - **docker login (or logout)**
  - Use browser to login to your account

- **docker pull rajeevmauritius/sbdockergit:v1**

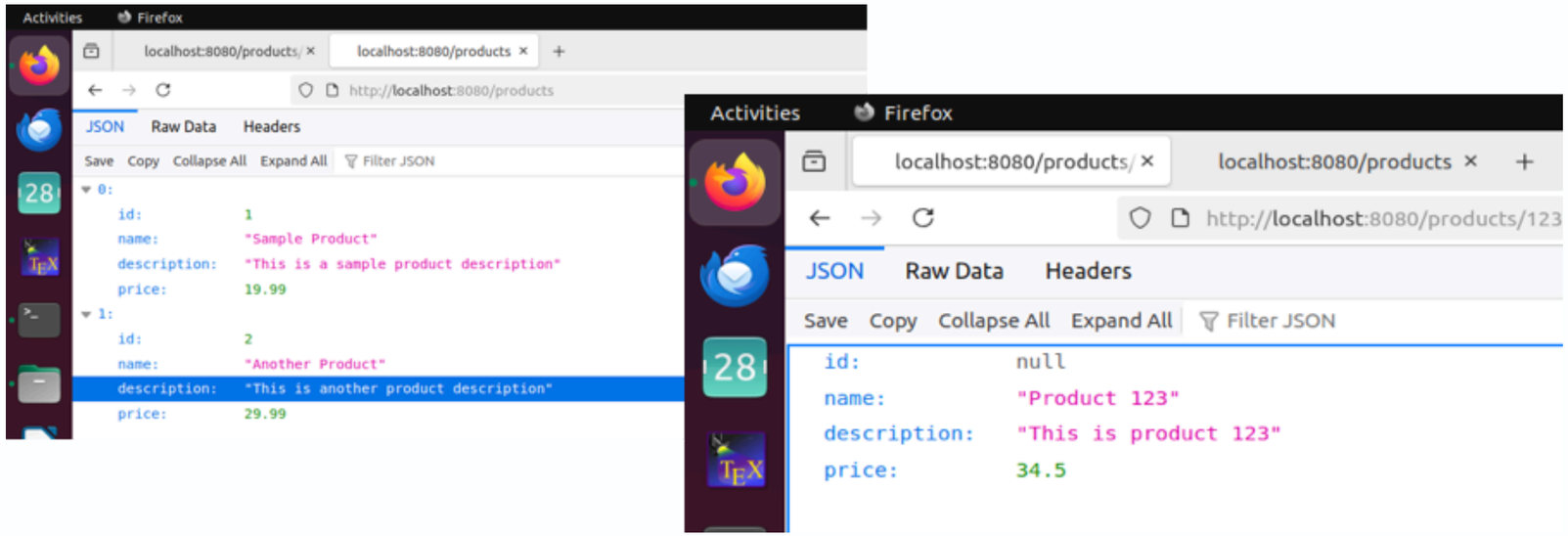
DOCKERHUB REPO.





RUN AND TEST THE APP

- sudo docker info (checks if docker is working)
- sudo docker build -t rajeevmauritius/sbdockergit:v1 .
- docker run -p 8080:8080 rajeevmauritius/sbdockergit:v1



SOME DOCKER COMMANDS...

- To see all running containers:
  - **docker ps**

- To see all containers (including stopped ones)
  - **docker ps -a**

- If the container is still running, stop it first:
  - **docker stop <container_name_or_id>**

- Once stopped, remove it using:
  - **docker rm <container_name_or_id>**

- Force
  - **docker rm -f <container_name_or_id>**

IF USING SNAP...

- **Containers**
  - /var/snap/docker/common/var-lib-docker/containers/
- **Images**
  - /var/snap/docker/common/var-lib-docker/image/
- **Volumes**
  - /var/snap/docker/common/var-lib-docker/volumes/

- sudo snap run docker ps -aq # list all container IDs
- sudo snap run docker rm -f $(sudo snap run docker ps -aq)
- sudo snap run docker images -q # list all image IDs
- sudo snap run docker rmi -f $(sudo snap run docker images -q)
- sudo snap run docker volume prune -f
- sudo snap run docker network prune -f
- sudo snap stop | start docker

FULL STACK DEV

Write a Docker Compose File for Multi-Container Setup (App + mySQL)

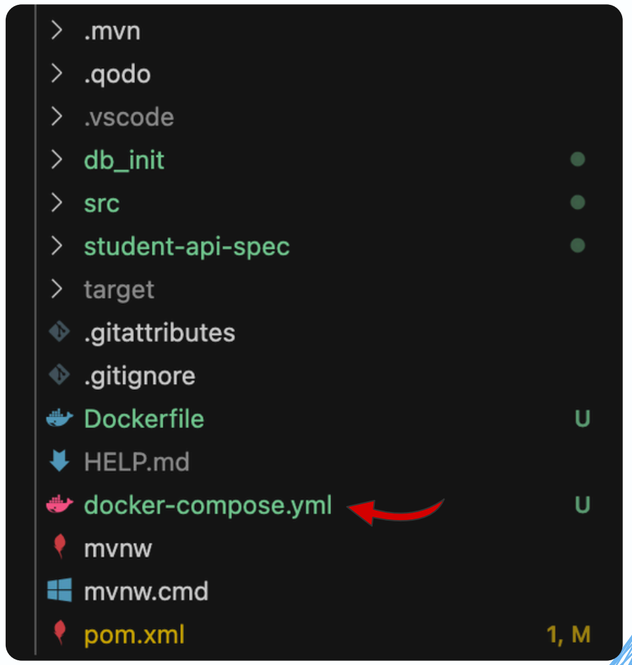
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

FOLDERS/FILES

- Docker Compose helps manage multiple containers.
  - We’ll create a YAML file ***(docker-compose.yml)*** to start the app and database containers together with networking.



- This file defines two services:
  - **db** :
    - for the MySQL database
  - **app** :
    - Spring Boot app .

- It configures them to communicate with each other on an isolated network and ensures the database's data is persistent.

CONTAINERS

```yaml
services:
  ▷ Run Service
  db:
    image: mysql:8.0
    container_name: mysql_db2
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: "Rk2025.;"
      MYSQL_DATABASE: test
      MYSQL_USER: rajeev
      MYSQL_PASSWORD: "Rk2025.;"
    ports:
      - "3307:3306"
    volumes:
      - db_data:/var/lib/mysql
      - ./db_init:/docker-entrypoint-initdb.d
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5
```

```yaml
  app:
    build: .
    image: rajeevmauritius/springboot-mysql-app:latest
    container_name: my_app2
    restart: unless-stopped
    ports:
      - "8081:8080"
    environment:
      #SPRING_PROFILES_ACTIVE: local
      DB_HOST: db
      DB_USER: root
      DB_PASSWORD: "Rk2025.;"
      DB_NAME: test
    depends_on:
      db:
        condition: service_healthy

volumes:
  db_data:
```

SERVICES : DB

- This is the core of the file, where you define each container.

- **db**: #db stands for **database**
- **image**: mysql:8.0: Uses the official MySQL image from Docker Hub.
- **environment**: Sets environment variables that the MySQL image uses to configure the database
- **ports**: Maps the container's internal MySQL port (3306) to the host machine's port (3306)
- **volumes**: Mounts a named volume (db_data) to the MySQL data directory inside the container.
- **- db_data**:/var/lib/mysql. # where your database is stored
- **- ./db_init**:/docker-entrypoint-initdb.d # contains the .sql file for your entire database.

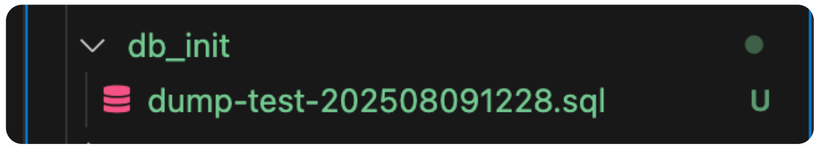
SERVICES : APP

- **app**: # app stands for your spring boot **application**
- **build**: .: Tells Docker to build an image for this service from the Dockerfile.
- **ports**: Maps your application's internal port (8080) to the host machine's port (8080).
- **environment**: Sets environment variables for your application to use.
- **depends_on**: - db: A simple dependency rule that ensures the db service is started and healthy before the app service is started.
- **volumes**: Defines the named volume db_data so Docker can manage it.

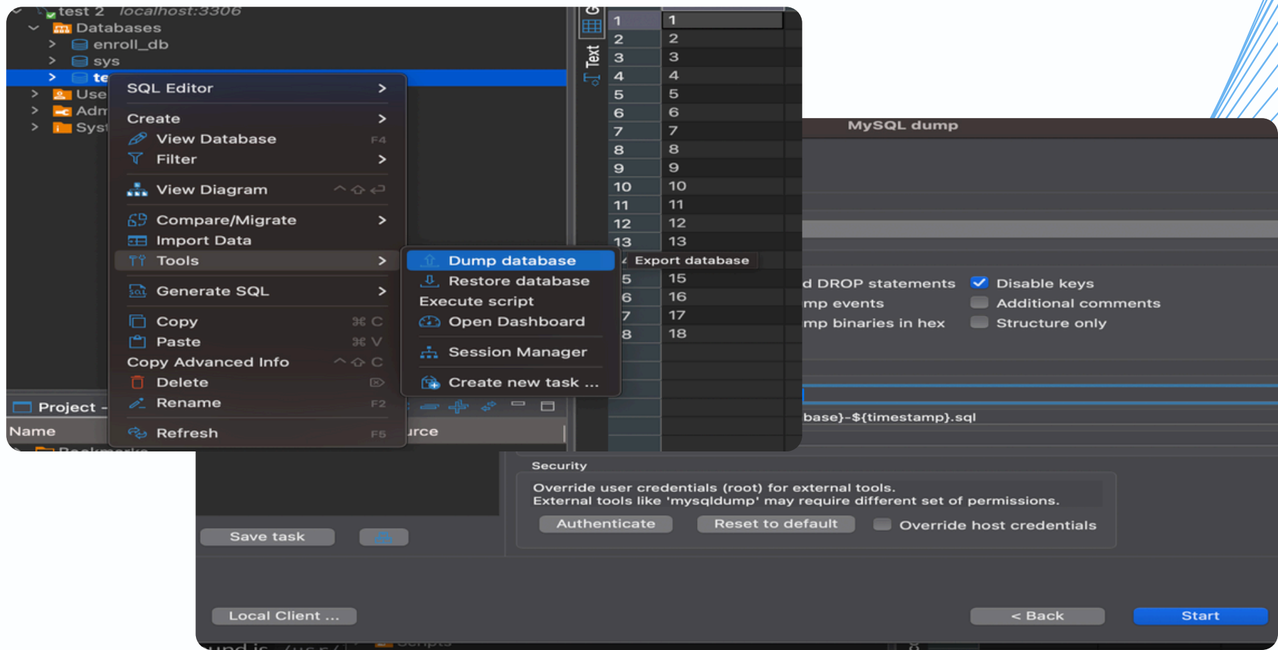
HOW IT WORKS ?

- You export your database along with your container

- When Developer #2 pulls the Docker image and runs it , Docker :
  - runs mysql
  - executes the .sql file inside db_init
  - runs the tomcat server
  - runs the Spring Boot



- Reads application.properties
- Reads the docker-compose.yml



HOW TO RUN THE APP?

- **Use only one application.properties file**

- spring.datasource.url=jdbc:mysql://db:3306/my_app_db
- Change the host here either as localhost:3306 (host) or db:3306 (container)
  - Then rebuild your application using mvn then run docker commands

- **Use two application.properties file**
- In this case, you create two properties file :
  - application-local.properties (used on your computer)
    - spring.datasource.url=jdbc:mysql://localhost:3306/my_app_db
  - application.properties (used inside the docker container)
    - spring.datasource.url=jdbc:mysql://db:3306/my_app_db
- Docker Compose automatically creates a network where containers can reach each other using their service names.

HOW THE DOCKER NETWORK HANDLES COMMUNICATION

- Docker Compose creates a private network for all the services defined in your docker-compose.yml.

- Within this network, each service is automatically given a hostname that is the same as its service name (**db vs app**)

- Internal DNS: Docker provides a simple DNS service for this network. When your Spring Boot app tries to connect to db:3306, the network's DNS resolves db to the correct internal IP address of the MySQL container.

- No Port Mapping Needed: Because the communication is happening internally on this private network

RUNNING THE DOCKER APP

- docker run -p 8080:8080 your-dockerhub-username/your-app- name:tag.

- docker-compose up - -build
- If you want to restart then :
  - docker-compose down

```
 => => writing image sha256:417d5c40816a2ebd0b82723194b6238930f99902e317f88a215d5
 => => naming to docker.io/library/section3-app
 => [app] resolving provenance for metadata file
[+] Running 3/3
 ✔ app                  Built
 ✔ Container mysql_db2  Created
 ✔ Container my_app2    Recreated
Attaching to my_app2, mysql_db2
```

```
CONTAINER ID   IMAGE     COMM
(base) rajeev@Rajeev-Khoodeer
CONTAINER ID   IMAGE
                  PORTS
55239504d7e4   section3-app
) 4 minutes ago
8289b8f512fa   mysql:8.0
4 minutes ago
c1f83b6c2937   43ee22767648
8 hours ago
7828e64e3852   43ee22767648

har
d53e4851241d   43ee22767648
```

```
mysql> update student set name="dev saheb" where id=2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> update students set name="dev saheb" where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> s
(base) rajeev@Rajeev-Khoodeeram ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
(base) rajeev@Rajeev-Khoodeeram ~ % docker exec -it 8289b8f512fa mysql -uroot -p
```

FULL STACK DEV

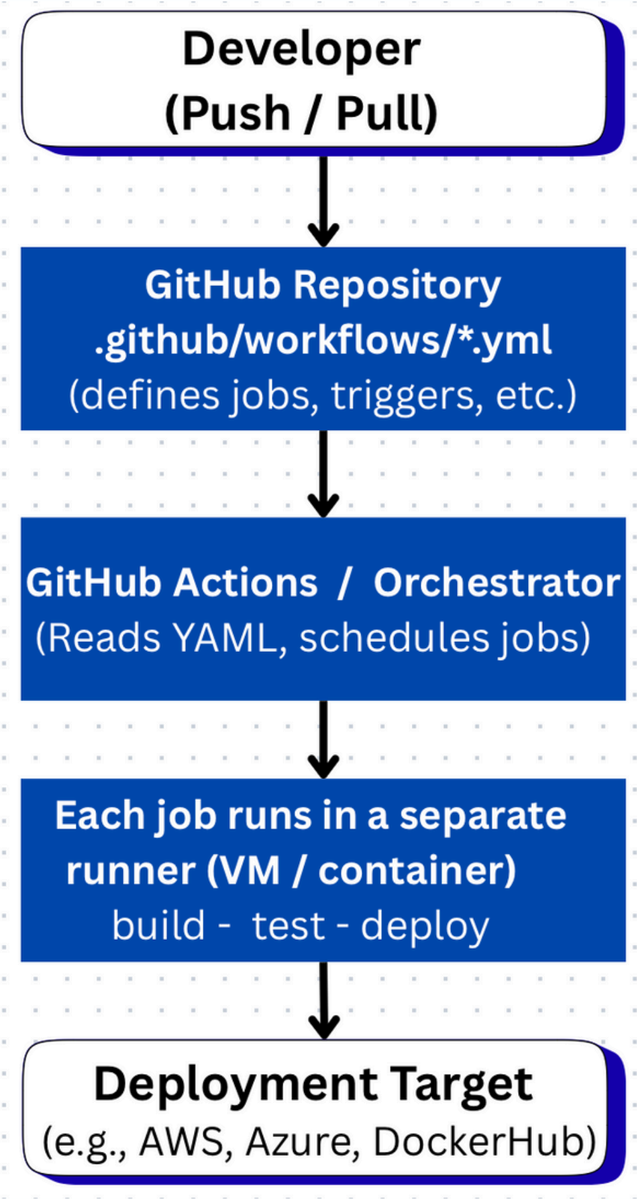
Set Up a basic CI/CD pipeline for automated deployment with Github Actions

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

STEPS IN A CI-CD



STEP 1: PREPARE YOUR GITHUB REPO

- Make sure your repo has:
- Your Spring Boot source code
- mvnw Maven wrapper (recommended)
- You must run this command :
  - (base) rajeev@Rajeev-Khoodeeram springboot % **mvn -N io.takari:maven:wrapper**

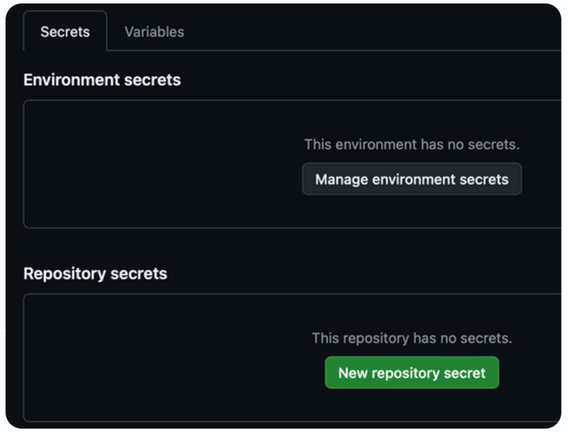
- What it does:
  - It creates a Maven Wrapper for your project.

- A Dockerfile at the root

GITHUB

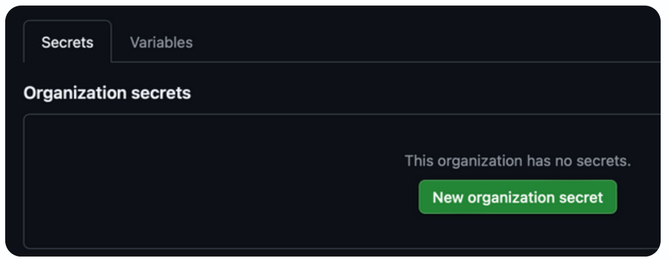
- In GitHub Actions (on github !!), secrets can be stored at different levels :

- **Repository**
  - Stored per repository.
  - Available only to workflows running in that repository.



- **Environment**
  - Define deployment targets (e.g., staging, production).

- **Organisation**
  - Can be shared with multiple repositories inside that organization.



STEP 2: CREATE GITHUB SECRETS

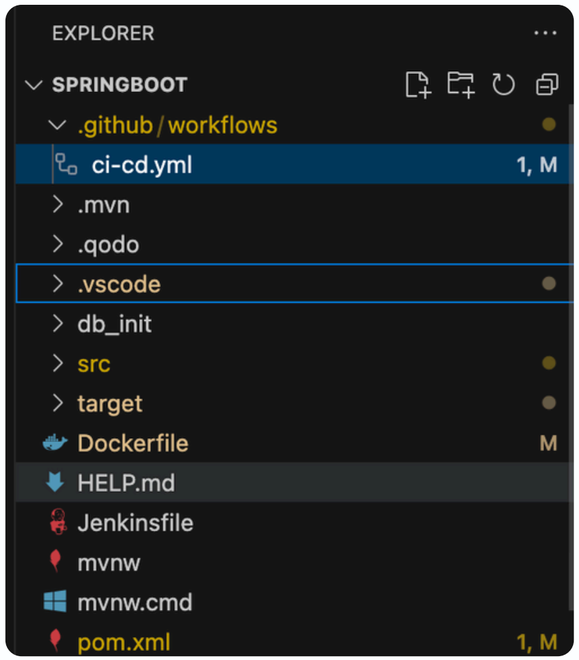
- Click the ⚙️ “Settings” tab near the top of the **repository**.
- On the left sidebar, scroll down and click:
  - Settings → Secrets and variables → Actions

- You’ll see two tabs:
  - Secrets
  - Variables

Name	Description
DOCKER_USERNAME	Your Docker Hub username
DOCKER_PASSWORD	Your Docker Hub access token

- Under Secrets, click the “New repository secret” button.
- Add your secret
  - Name: (all uppercase, no spaces)
  - e.g. DOCKER_USERNAME, DOCKER_PASSWORD,etc.
  - Value: your actual credential (e.g. your token, password, or key).

STEP 3: CREATE GITHUB ACTIONS WORKFLOW



- Before that, steps to complete :
- Install the extension GitHub Actions in VS code

- Create your docker hub Personal access token (PAK)

- Create the folder .github/workflows in the root folder of our app
  - Create a .github/workflows/ci-cd.yml file in your repo:

.GITHUB/WORKFLOWS/CI-CD.YML (1)

ci-cd.yml 1, M

.github > workflows > ci-cd.yml > { } jobs > { } build-and-push > runs-on

GitHub Workflow - YAML GitHub Workflow (github-workflow.json)

```yaml
name: Docker Git Spring Boot CI/CD

on:
  push:
    branches:
      - main
      - develop
      - 'feature/*'

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repo
        uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Build with Maven
        run: ./mvnw clean package --no-transfer-progress
```

.GITHUB/WORKFLOWS/CI-CD.YML (2)

```yaml
      - name: Check DOCKER_USERNAME secret
        run: |
          USERNAME="${{ secrets.DOCKER_USERNAME }}"
          echo "Docker username starts with: ${USERNAME:0:3} wow"

      # Must check if this works
      - name: Get latest tag
        id: vars
        run: |
          TAG=$(git describe --tags --abbrev=0)
          echo "TAG=$TAG" >> $GITHUB_ENV

      - name: Build Docker image
        run: docker build -t rajeevmauritius/dockergitspring-app:${{ github.sha }}

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: rajeevmauritius
          password: dckr_pat_raA-hslxZhJRxzODIacnKSrgmPs

      - name: Push Docker image
        run: docker push rajeevmauritius/dockergitspring-app:${{ github.sha }}
```

STEPS

- Explanation:
- **Trigger**: On push to main or develop, and on Pull Reqs
- **Checkout**: Gets your code
- **Setup JDK**: Installs Java 17 for Maven
- **Build**: Compiles your Spring Boot app with Maven
- **Build Docker image**: Creates a tagged Docker image using the commit SHA
- **Docker Login & Push**: Authenticates and pushes image to Docker Hub
- **Deploy**: Placeholder for your deployment step

STEP 4: TEST IT

- Push the .github/workflows/ci-cd.yml file to your repo’s branch
- Go to Actions tab in GitHub to watch the workflow run
- If successful, check Docker Hub for your pushed image
- Important : For the pipeline to work, several aspects must be taken into consideration :
  - All tests must pass or succeed !!
  - The JDK must be 17 or 21 (***as 24 is not yet supported***)

FULL STACK DEV

Set Up a basic CI/CD pipeline for automated deployment with Jenkins

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

REFRESH YOUR MEMORY

- **Let’s outline a CI/CD pipeline that:**
- Checks out your Java Spring Boot app from GitHub
- Builds it with Maven
- Runs unit tests
- Builds a Docker image
- Pushes that Docker image to Docker Hub (or your container registry)
- Optionally deploys the image to a server or Kubernetes cluster

STEP 1: PREREQUISITES

- Jenkins installed and running
- Jenkins user with Docker installed or Docker daemon accessible (for building images)
- Jenkins credentials for Docker Hub (username/password or token) saved in Jenkins Credentials Manager
- A GitHub repo with your Spring Boot app and Dockerfile

- On **MAC**
  - Install the latest LTS version: **brew install jenkins-lts**
  - Start the Jenkins service: **brew services start jenkins-lts**
  - Restart the Jenkins service: **brew services restart jenkins-lts**
  - Update the Jenkins version: **brew upgrade jenkins-lts**

STEP 2: DOCKERFILE

Jenkinsfile M | Dockerfile 2, M

Dockerfile > ...

```
1  # Stage 1 - Build
2  FROM maven:3.9.6-eclipse-temurin-17  AS builder
3  WORKDIR /app
4  COPY . .
5  RUN mvn clean package -DskipTests
6  # Skip tests for faster build
7
8  # Stage 2 - Runtime
9  FROM openjdk:17-jdk-slim AS runner
10 WORKDIR /app
11 COPY --from=builder /app/target/*.jar app.jar
12 EXPOSE 8080
13 CMD ["java", "-jar", "app.jar"]
14
```

INSTALLING JENKINS ON LINUX

- sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

- echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

- sudo apt-get update

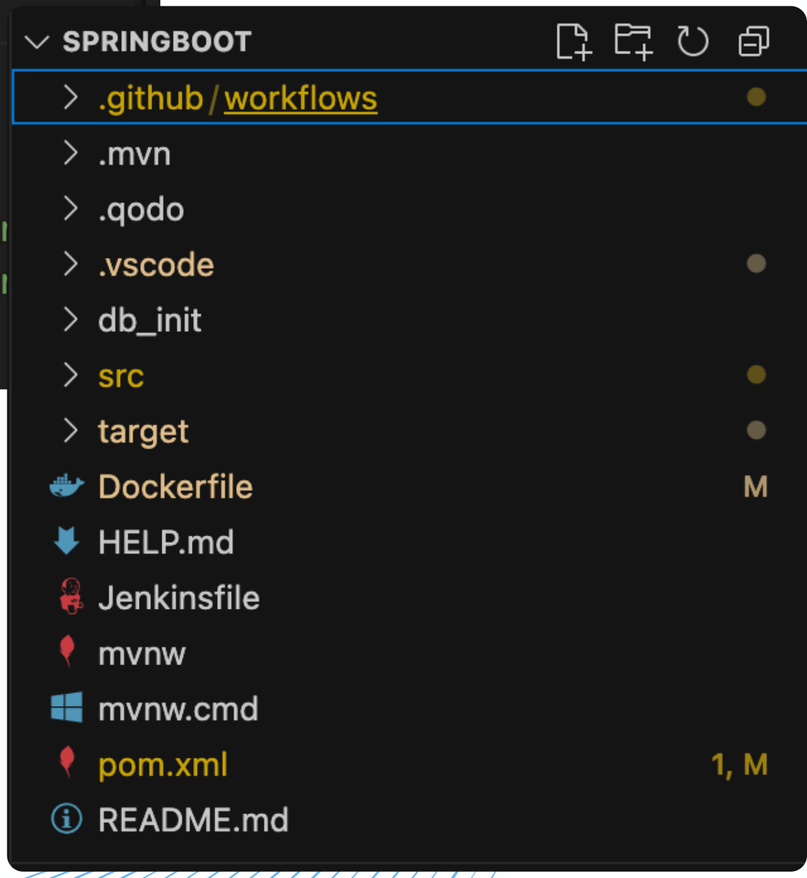
- sudo apt-get install jenkins

STEP 3 : JENKINS FILE (1)

Jenkinsfile

```
pipeline {
    agent any
    // add this tools section in case Jenkins cannot find maven (you will get messa
    // could not find mvn in the console output)
    tools {
        maven 'Rajeev-maven'  // name you gave in Global Tool Config
    }

    environment {
        PATH = "/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin"
        DOCKER_HUB_CREDENTIALS = credentials('docker-hub-credentials-id')
        IMAGE_NAME = 'rajeevmauritius/myapp' //name of image that will appear
        IMAGE_TAG = "jenkins-${env.BUILD_NUMBER}". //this number will be incr
    }
```



JENKINS FILE (2)

```
16      stages {
17          stage('Checkout') {
18              steps {
19                  //  normally you will branch into main
20                  //  but since I was using a git repository that was  a branch
21                  //  I replaced main with the branch name
22                  // works - git branch: 'feature/initial-project', url: 'https://git
23                  git branch: 'main', url: 'https://github.com/rajeev-khoodeeram/java
24              }
25          }
26
27          stage('Build') {
28              steps {
29                  sh 'mvn clean package -DskipTests'  // Build without tests here
30              }
31          }
```

JENKINS FILE (3)

```
33          stage('Test') {
34              steps {
35                  sh 'mvn test'                          // Run tests separately
36                  junit 'target/surefire-reports/*.xml'  // Publish test results
37              }
38          }
39
40          stage('Build Docker Image') {
41              steps {
42                  script {
43                      sh 'docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .'
44                  }
45              }
46          }
```

```
67      post {
68          always {
69              cleanWs()  // Clean workspace after build
70          }
71          success {
72              echo 'CI/CD Pipeline succeeded!'
73          }
74          failure {
75              echo 'CI/CD Pipeline failed!'
76          }
77      }
78  }
```

STEP 4: ADD DOCKER HUB CREDENTIALS IN JENKINS

- Go to Jenkins dashboard → Manage Jenkins → Manage Credentials → (Global)

- Normally it is http://localhost:9090
- But if it does not work, then use http://127.0.0.1:9090

- ps aux | grep jenkins

```
(base) rajeev@Rajeev-Khoodeeram ~ % ps aux | grep jenkins

rajeev           77698   0.0  0.0 410063264    192 s002  R+   11:02pm   0:00.00
grep jenkins
rajeev           77310   0.0  0.4 419548112  70176   ??  S    11:01pm   0:07.23
/opt/homebrew/opt/openjdk@21/bin/java -Dmail.smtp.starttls.enable=true -jar /opt
/homebrew/opt/jenkins-lts/libexec/jenkins.war --httpListenAddress=127.0.0.1 --ht
tpPort=8080
```

CONFIGURING JENKINS

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/Users/rajeev/.jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

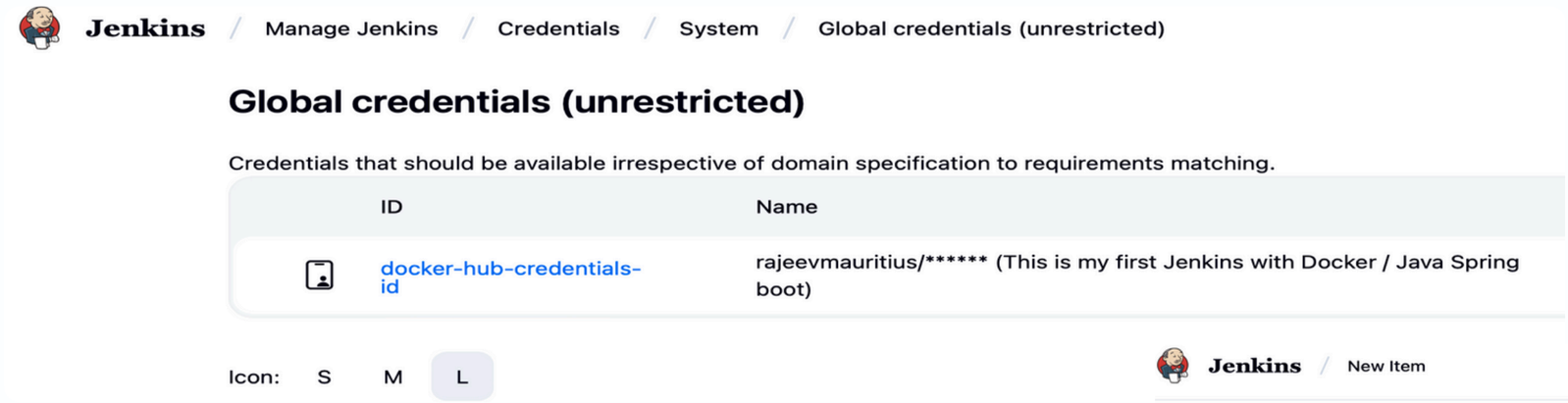
Install plugins the Jenkins community finds most useful.

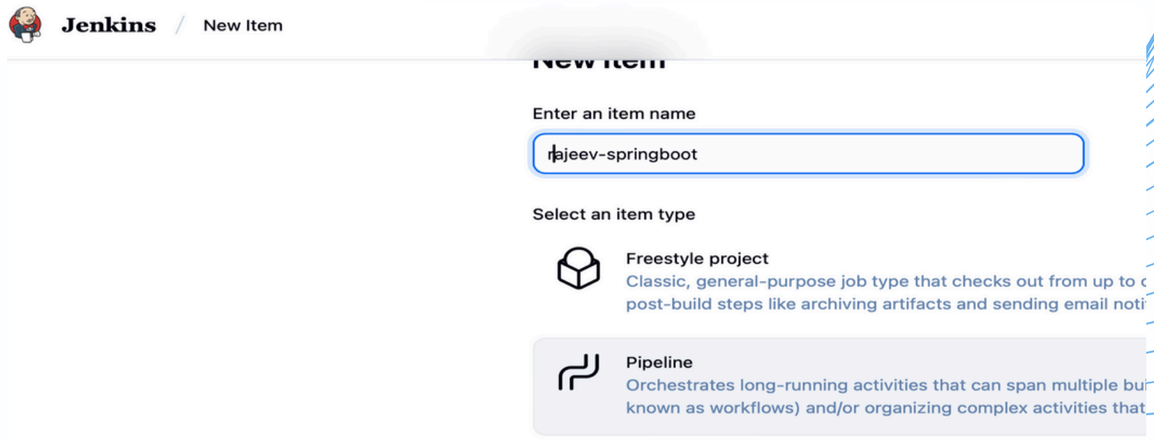
Select plugins to install

Select and install plugins most suitable for your needs.

PROCEED WITH THE INSTALLATION

- Add Username with password credential for your Docker Hub account
- Give it ID: docker-hub-credentials-id (or change the Jenkinsfile accordingly)

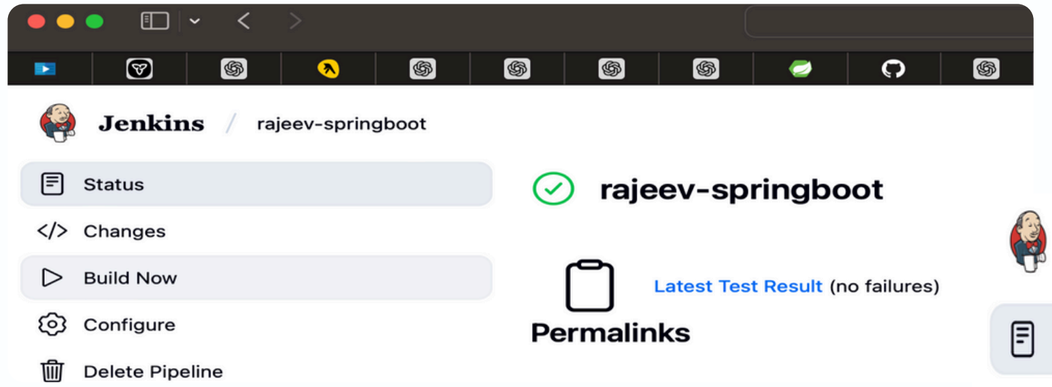


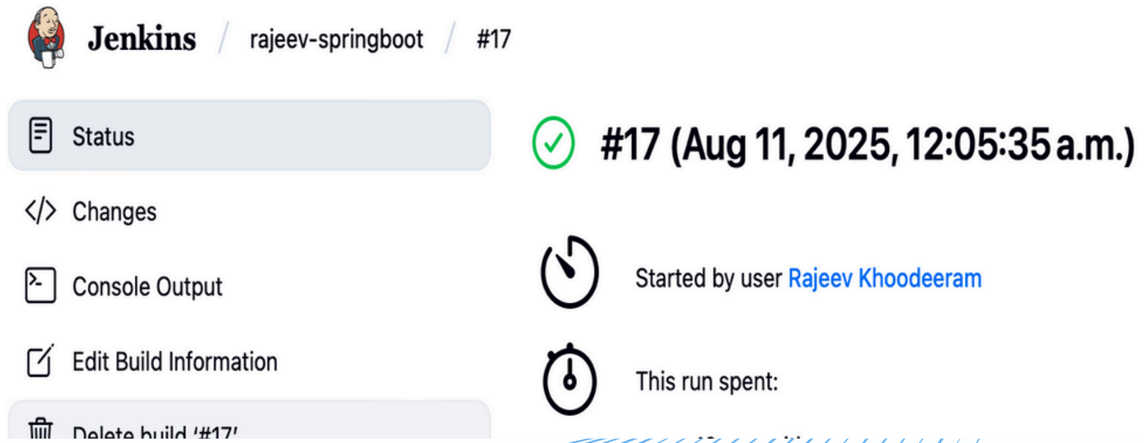


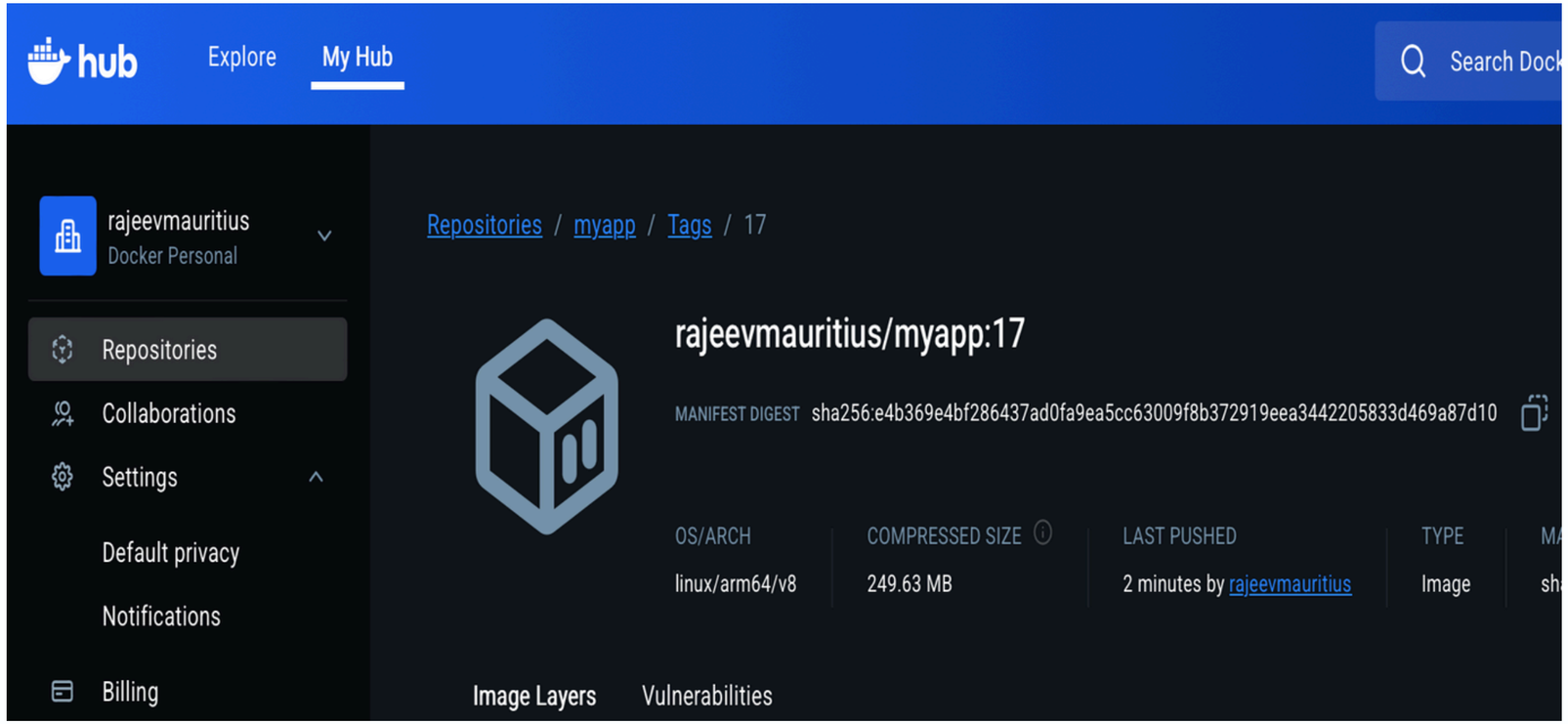
RUN YOUR PIPELINE

- Create a new Pipeline job in Jenkins and point it to your Jenkinsfile (either in repo or directly in the pipeline editor)

- Run the job and watch it build your app, run tests, build and push the Docker image







DON'T FORGET TO DISABLE GITHUB ACTIONS IN CASE YOU SWITCH TO JENKINS

HOW TO EXPOSE YOUR LOCAL JENKINS

- By default, your Jenkins is only accessible on your private network (computer)
- To do remote, we will need to add a webhook to GitHub so that when it is triggered by a push, it will call the web hook.

- (base) rajeev@Rajeev-Khoodeeram ~ % **brew install ngrok**

- (base) rajeev@Rajeev-Khoodeeram ~ % **ngrok http 9090**
  - **ERROR**: authentication failed: Usage of ngrok requires a verified account and authtoken

- **You will need to configure ngrok !!**

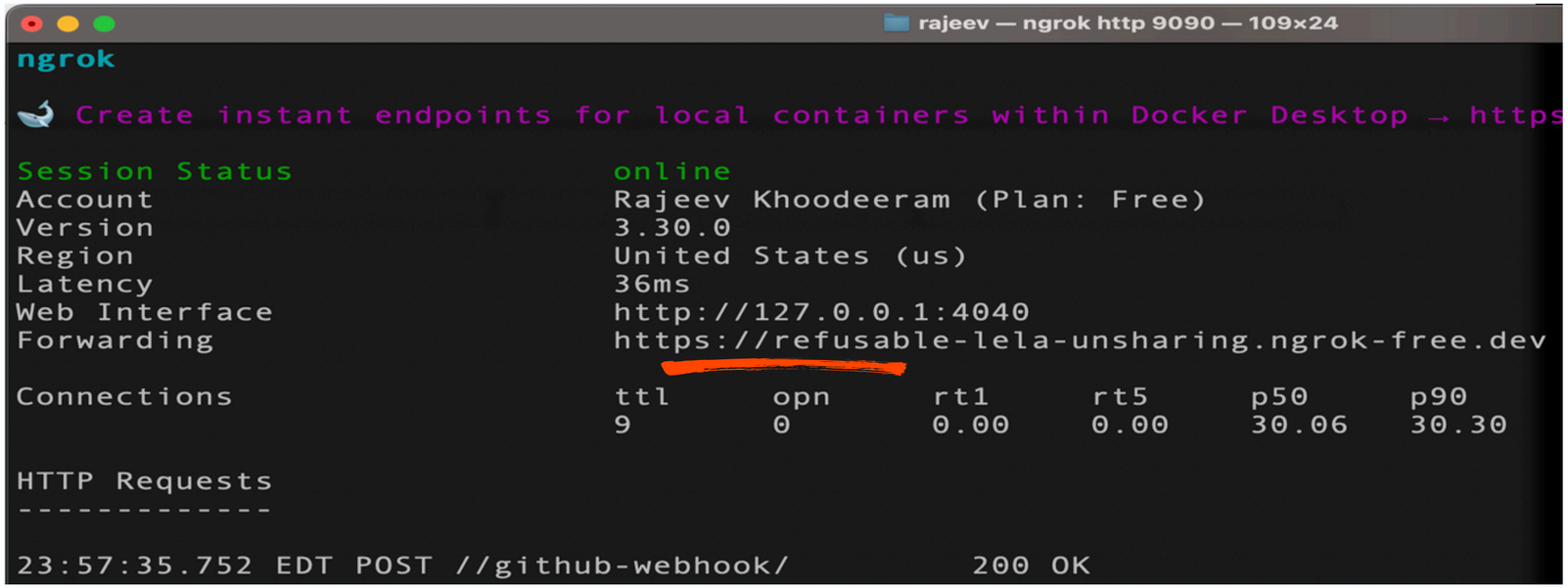
SET NGROK AUTHTOKEN

- **Log in to ngrok**
  - Go to → https://dashboard.ngrok.com

- **Copy your Authtoken**
  - After logging in, look in the left sidebar → Getting Started → Your Authtoken.

- (base) rajeev@Rajeev-Khoodeeram ~ % ngrok config add-authtoken
  - **Paste your token here**
  - Authtoken saved to configuration file: /Users/rajeev/Library/Application Support/ngrok/ngrok.yml

RUNNING NGROK



NOTE THE FORWARDING URL - WE WILL NEED THIS IN GITHUB

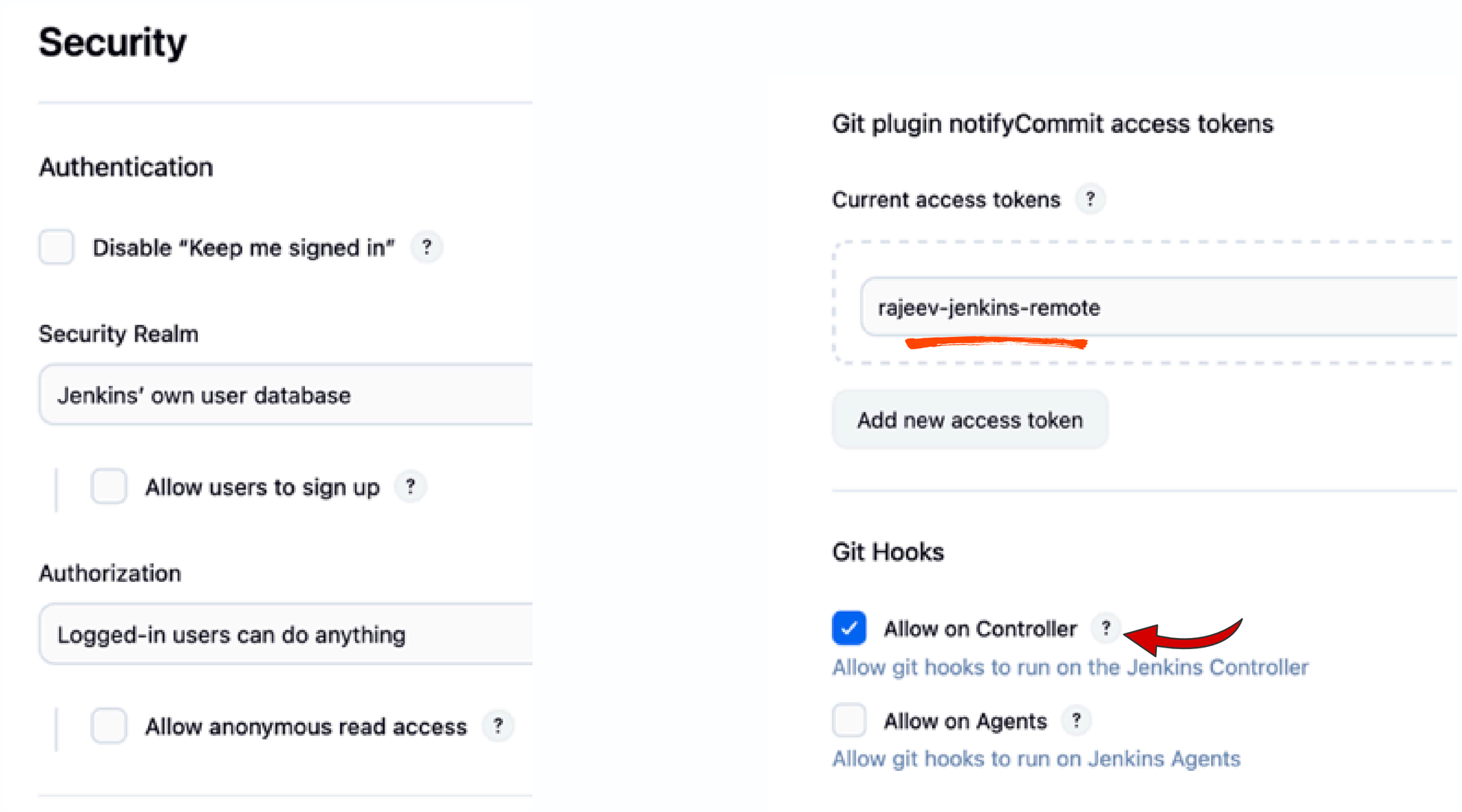
INSTALLING NGROK

- **Windows**
  - https://ngrok.com/download/windows

- **Linux**
  - curl -sSL https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
| sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
&& echo "deb https://ngrok-agent.s3.amazonaws.com bookworm main" \
| sudo tee /etc/apt/sources.list.d/ngrok.list \
&& sudo apt update \
&& sudo apt install ngrok

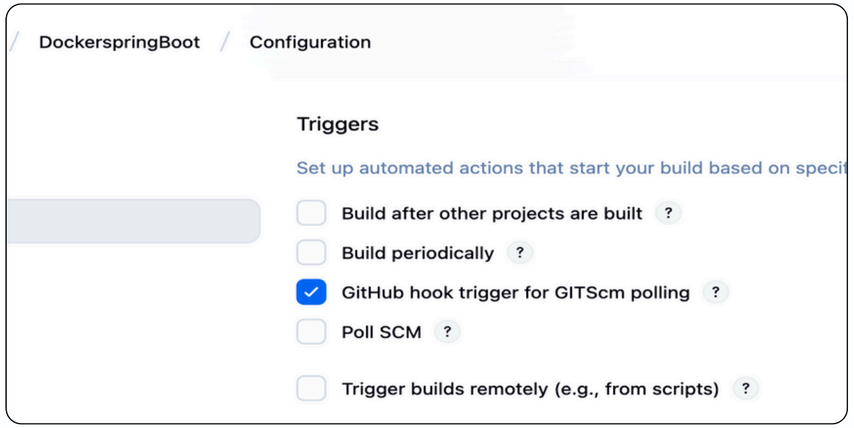
  - ngrok config add-authtoken <token>

SECURITY


Security
Authentication
Disable "Keep me signed in"
Security Realm
Jenkins' own user database
Allow users to sign up
Authorization
Logged-in users can do anything
Allow anonymous read access
Git plugin notifyCommit access tokens
Current access tokens
rajeev-jenkins-remote
Add new access token
Git Hooks
Allow on Controller
Allow git hooks to run on the Jenkins Controller
Allow on Agents
Allow git hooks to run on Jenkins Agents

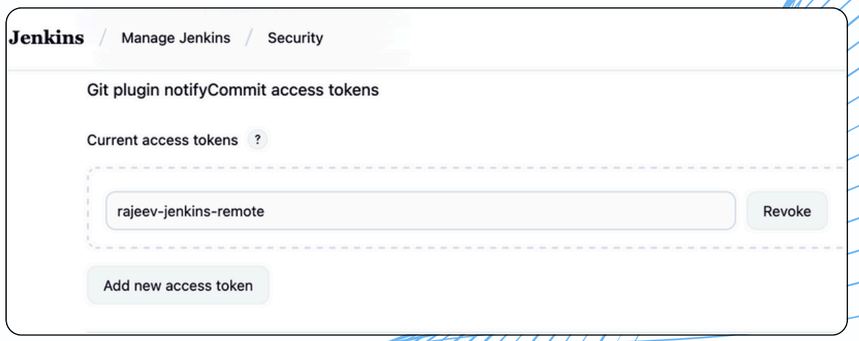
ENSURE JENKINS JOB AND SECURITY ARE CORRECT

- **Jenkins job configuration → Build Triggers:**
  - Check “GitHub hook trigger for GITScm polling”



- **Manage Jenkins → Security → Git Hooks:**
  - Check Allow on Controller

- **Security / authentication:**
  - Use a webhook secret in GitHub, or temporarily allow anonymous access for local testing.



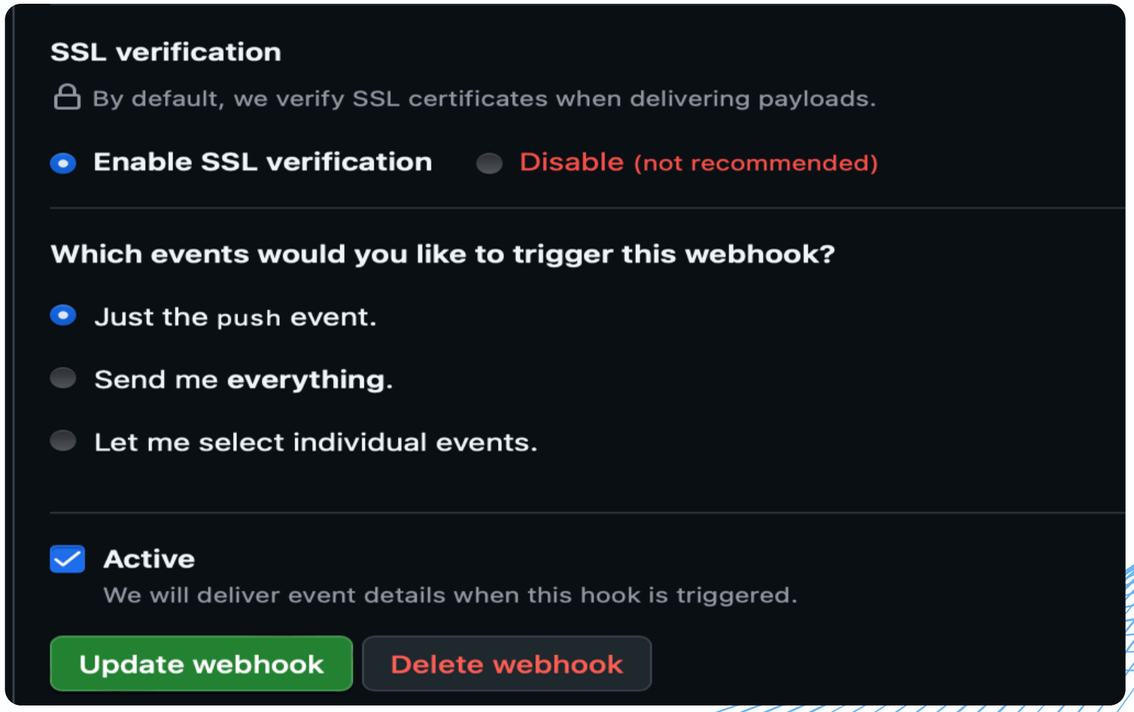
CONFIGURING GITHUB

- **Now, in GitHub**
  - select your repository —> settings —> web hooks

- **Payload [as in terminal — see above]**
  - https://refusable-lela-unsharing.ngrok-free.dev//github-webhook/

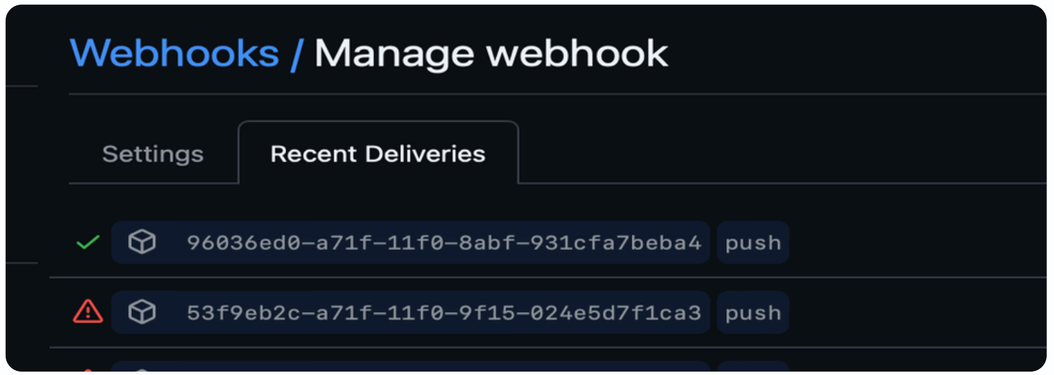
- **Content type ***
  - application/json

- Secret



CONFIGURING GITHUB

- **Jenkins**
  - You can check Recent Deliveries tab
  - In case it does not work, you will get Error 403 Forbidden



- **Github**
  - We have to configure SECRET settings