Perfect 👍
Here's a **complete, beginner-friendly, detailed guide** for doing **DevOps practicals on Ubuntu**, from **Set 1 → Set 3**.
Everything is explained from scratch, in step-by-step format — so even if you've just installed Ubuntu, you can follow along easily.

□ SET 1 — Basic Software & Source Code Management

◈ Q1 – Create a local Git repository, add a Java file, commit and push it to GitHub

□ Concept

You'll write a simple Java program, use Git to track changes, and upload (push) it to GitHub so it's stored safely online.

🔧 Steps

Step 1 – Create a working folder

```
mkdir ~/devpractice
cd ~/devpractice
```

This makes a folder called devpractice inside your home directory, and then you move into it.

Step 2 – Create a simple Java program

Open a text editor (nano comes pre-installed):

```
nano HelloWorld.java
```

Paste:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

Save → Ctrl + O, Enter.
Exit → Ctrl + X.

Step 3 – Compile & run the program

javac HelloWorld.java

java HelloWorld

✅ Output:

Hello, world!

If this works, your Java setup is correct.

Step 4 – Initialize Git & commit your code

git init

git add HelloWorld.java

git commit -m "First commit - HelloWorld"

This creates a .git folder that tracks changes and saves the first version (commit).

Step 5 – Create a GitHub repository

1. Go to https://github.com → Sign in.
2. Click **New Repository** → Name: myapp-example.
3. Keep defaults → Click **Create repository**.
   You'll now see the remote URL like:

https://github.com/<your-username>/myapp-example.git

Step 6 – Push your code to GitHub

git branch -M main

git remote add origin https://github.com/<your-username>/myapp-example.git

git push -u origin main

✅ Your HelloWorld.java file now appears on your GitHub repo page.

🔷 **Q2 – Create a Maven project and build it**

□ Concept

Maven is a Java build-automation tool. It compiles code, runs tests, and packages everything into a .jar file automatically.

🔧 **Steps**

Step 1 – Generate a new Maven project

cd ~/devpractice

mvn archetype:generate -DgroupId=com.example \

-DartifactId=myapp \

-DarchetypeArtifactId=maven-archetype-quickstart \

-DinteractiveMode=false

✅ This creates a folder myapp/ with a ready-to-use Java project.

Step 2 – Build the project

cd myapp

mvn clean package

- clean = deletes old build files
- package = compiles + packages into a JAR

Step 3 – Run the built program

java -cp target/myapp-1.0-SNAPSHOT.jar com.example.App

✅ You'll see "Hello World!" printed again, this time from the packaged JAR.

🔷 **Q3 – Create a Dockerfile and build an image**

□ Concept

Docker wraps your app + its dependencies into a container so it runs identically anywhere.

🔧 **Steps**

Step 1 – Create a Dockerfile

nano Dockerfile

Paste:

FROM eclipse-temurin:17-jre

WORKDIR /app

COPY target/myapp-1.0-SNAPSHOT.jar app.jar

ENTRYPOINT ["java","-jar","/app/app.jar"]

Save & exit.

Step 2 – Build the Docker image

docker build -t myapp:1.0 .

✅ Docker creates an image named myapp:1.0.

Step 3 – Run the container

docker run --rm myapp:1.0

✅ You'll see your program output inside the container.

⚙️ **SET 2 — Continuous Integration with Jenkins**

🔷 **Q1 – Create a Jenkins Freestyle job to build Maven project**

□ Concept

Jenkins automates building/testing so you don't have to run commands manually.

🔧 **Steps**

Step 1 – Run Jenkins in Docker

docker run -d --name jenkins -p 8080:8080 -p 50000:50000 \

-v jenkins_home:/var/jenkins_home jenkins/jenkins:lts

Step 2 – Open Jenkins in browser

Go to:
👉 http://localhost:8080

Get admin password:

docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword

Copy and paste it on the setup screen → Install “Suggested Plugins”.

Step 3 – Add tools

Go to:
Manage Jenkins → Global Tool Configuration

- Add Maven installation (Name = Maven)
- Ensure Git plugin is enabled

Step 4 – Create a Freestyle project

1. **New Item** → enter BuildMyApp → choose **Freestyle Project**
2. **Source Code Management** → Git → enter your GitHub repo URL + credentials
3. **Build → Invoke top-level Maven targets** → Goals: clean package
4. Click **Save → Build Now**
   ✅ You’ll see “BUILD SUCCESS” in the console.

🔷 Q2 – Set up GitHub Webhook to trigger build on push

□ Concept

A webhook tells Jenkins to build automatically whenever you push new code to GitHub.

🔧 Steps

1. In Jenkins job → **Configure → Build Triggers** → check ✅ “GitHub hook trigger for GITScm polling”
2. In GitHub → Repo → **Settings → Webhooks → Add webhook**
   - Payload URL: http://<your-IP>:8080/github-webhook/
   - Content type: application/json
   - Event: **Just the push event**
3. Push a change to repo → Jenkins builds automatically.

◈ Q3 – Run Docker container and view logs

docker run -d --name myapp -p 8080:8080 myapp:1.0

docker ps

docker logs myapp

docker stop myapp

✅ docker ps shows running container; docker logs shows app output.

⚙ SET 3 — Pipelines and Version Control Flow

◈ Q1 – Clone repo, edit file, commit and push changes

Steps

cd ~/devpractice

git clone https://github.com/<your-username>/myapp-example.git

cd myapp-example

git checkout -b change-message

nano HelloWorld.java

(change the text)

git add HelloWorld.java

git commit -m "Change greeting message"

git push -u origin change-message

Go to GitHub → create Pull Request → merge to main.

◈ Q2 – Create Jenkins Pipeline with Stages

□ Concept

A pipeline is a scripted automation file (Jenkinsfile) defining each step of CI/CD.

🔧 Steps

Create Jenkinsfile:

nano Jenkinsfile

Paste:

```
pipeline {
  agent any
  tools { maven 'Maven' }
  stages {
    stage('Checkout') { steps { checkout scm } }
    stage('Build')    { steps { sh 'mvn clean package' } }
    stage('Test')     { steps { sh 'mvn test' } }
    stage('Archive')  { steps { archiveArtifacts artifacts: 'target/*.jar' } }
  }
}
```

Save → exit.

Commit and push:

```
git add Jenkinsfile
git commit -m "Add Jenkins pipeline"
git push
```

In Jenkins:
New Item → Pipeline → Pipeline script from SCM → Git → repo URL → Save → Build Now.

✅ You'll see all pipeline stages run in order.

🔷 Q3 – Build Docker image inside pipeline

Steps

Add a new stage in Jenkinsfile below the existing ones:

```
stage('Docker Build') {
  steps {
    sh 'docker build -t myapp:latest .'
  }
```

}

Re-commit:

git add Jenkinsfile

git commit -m "Add Docker build stage"

git push

Re-run pipeline → after success:

docker images | grep myapp

✅ You'll see myapp:latest listed.

🎯 **Summary of What You Achieved**

Tool	You Learned To Do
Java	Write and run programs
Git + GitHub	Track and upload code
Maven	Build and package apps
Docker	Containerize your application
Jenkins	Automate builds and tests
Webhook	Trigger builds automatically
Pipeline	Script complete CI/CD flows

Would you like me to now make this full detailed version (with these explanations) into a **nicely formatted PDF** for you?