Set 1:

1. Create an HTML registration form (username, password, confirm password) and upload it to GitHub.

```
<html>
<body>
<form onsubmit="alert('Successfully Registered!'); return false;">
  <input placeholder="Username" required><br><br>
  <input type="password" placeholder="Password" required><br><br>
  <input type="password" placeholder="Confirm Password" required><br><br>
  <button>Register</button>
</form>
</body>
</html>
```

```
git init
git add register.html
git commit -m "Add registration form"
git branch -M main
git remote add origin https://github.com/<your-username>/registration-form.git
git push -u origin main
```

2. Create a Docker Compose setup to run two simple applications together.

```
# ----------------------------------------
# 1. CREATE PROJECT FOLDER
# ----------------------------------------
mkdir two-apps
cd two-apps

# ----------------------------------------
# 2. APP 1 (PYTHON)
# ----------------------------------------
mkdir app1
cat << 'EOF' > app1/app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello from Python App 1!"

app.run(host='0.0.0.0', port=5000)
EOF

echo "flask" > app1/requirements.txt

cat << 'EOF' > app1/Dockerfile
FROM python:3.9
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
```

```
CMD ["python", "app.py"]
EOF

# ----------------------------------------
# 3. APP 2 (NODE.JS)
# ----------------------------------------
mkdir app2
cat << 'EOF' > app2/server.js
const http = require('http');
const server = http.createServer((req, res) => {
  res.end("Hello from Node App 2!");
});
server.listen(6000, () => console.log("Node App running on port 6000"));
EOF

cat << 'EOF' > app2/package.json
{
  "name": "app2",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {}
}
EOF

cat << 'EOF' > app2/Dockerfile
FROM node:18
WORKDIR /app
COPY . .
CMD ["node", "server.js"]
EOF

# ----------------------------------------
# 4. DOCKER COMPOSE FILE
# ----------------------------------------
cat << 'EOF' > docker-compose.yml
version: "3.9"

services:
  app1:
    build: ./app1
    ports:
      - "5000:5000"

  app2:
    build: ./app2
    ports:
      - "6000:6000"
EOF

# ----------------------------------------
# 5. START BOTH APPS
# ----------------------------------------
```

```
docker-compose up --build

# ----------------------------------------
# 6. ACCESS
# ----------------------------------------
# http://localhost:5000  → Python App
# http://localhost:6000  → Node.js App

# ----------------------------------------
# 7. STOP APPS
# ----------------------------------------
# Press CTRL + C
docker-compose down
```

3. Create a Jenkins Freestyle job that prints "Hello STUDENT_NAME, Welcome to Jenkins!" using a string parameter.

1️⃣ Open Jenkins

Go to:

`http://localhost:8080`

2️⃣ Create a New Job

- Click **New Item**
- Enter name → `hello-job`
- Select **Freestyle Project**
- Click **OK**

3️⃣ Add String Parameter

- Scroll to **Build Parameters**
- Click **Add Parameter → String Parameter**
- Name:

  `STUDENT_NAME`

- Default Value (optional):

  `Vishwa`

4️⃣ Add Build Step

Go to **Build → Add build step → Execute shell**

Paste this shortest script:

```
echo "Hello ${STUDENT_NAME}, Welcome to Jenkins!"
```

5️⃣ Save and Build

- Click **Save**
- Click **Build with Parameters**
- Enter your name → Run

6️⃣ Check Console Output

You will see:

```
Hello Vishwa, Welcome to Jenkins!
```

Set 2:

1. Create an HTML feedback/contact form and push it to GitHub.

HTML file (contact.html):

```
<!DOCTYPE html>
<html>
<body>
<form>
<input placeholder="Name" required><br>
<input type="email" placeholder="Email" required><br>
<textarea placeholder="Message" required></textarea><br>
<button>Send</button>
</form>
</body>
</html>
```

Commands to push it to GitHub:

```
mkdir contact-form
cd contact-form
nano contact.html      # paste the above HTML, save and exit

git init
git add contact.html
git commit -m "Add simple contact/feedback form"

# create an empty repo on GitHub, e.g. contact-form
git branch -M main
git remote add origin https://github.com/<your-username>/contact-form.git
git push -u origin main
```

2. Develop a containerized web application in Docker consisting of Registration + Login form.

1. Create project folder:
   mkdir webapp
   cd webapp

2. Create index.html with Registration + Login form:
   cat << 'EOF' > index.html
   <form>
    <h3>Registration</h3>
    <input placeholder="Username" required>
    <input type="password" placeholder="Password" required>

    <h3>Login</h3>
    <input placeholder="Username" required>
    <input type="password" placeholder="Password" required>

    <button>Submit</button>
   </form>
   EOF

3. Create Dockerfile:
   cat << 'EOF' > Dockerfile
   FROM nginx:alpine
   COPY index.html /usr/share/nginx/html/index.html
   EOF

4. Build Docker image:
   docker build -t webapp-image .

5. Run container:
   docker run -d -p 8080:80 --name webapp-container webapp-image

6. Access application:
   http://localhost:8080

3. Create a Jenkins Pipeline with Build → Test → Deploy stages, each printing its stage name.

1. Open Jenkins → New Item → Pipeline → OK.

2. Go to Pipeline section → Paste this script:

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo "Build Stage"
      }
    }
    stage('Test') {
      steps {
        echo "Test Stage"
```

```
      }
    }
    stage('Deploy') {
      steps {
        echo "Deploy Stage"
      }
    }
  }
}
```

3. Click Save → Build Now.

4. Console Output shows:
   Build Stage
   Test Stage
   Deploy Stage

Set 3:

1.Create a Git branching story project where each branch adds a chapter and merge all into a final book.html.

1. Create project folder:
   mkdir story-book
   cd story-book

2. Initialize Git:
   git init

3. Create main file:
   echo "<h1>My Story Book</h1>" > book.html
   git add book.html
   git commit -m "Create book.html"

4. Create branch for Chapter 1:
   git checkout -b chapter1
   echo "<h2>Chapter 1</h2> This is chapter one. " >> book.html
   git add book.html
   git commit -m "Add Chapter 1"

5. Create branch for Chapter 2:
   git checkout main
   git checkout -b chapter2
   echo "<h2>Chapter 2</h2> This is chapter two. " >> book.html
   git add book.html
   git commit -m "Add Chapter 2"

6. Create branch for Chapter 3:
   git checkout main
   git checkout -b chapter3
   echo "<h2>Chapter 3</h2> This is chapter three. " >> book.html
   git add book.html

```
git commit -m "Add Chapter 3"
```

7. Merge all chapters into main:
```
git checkout main
git merge chapter1
git merge chapter2
git merge chapter3
```

8. Final book.html after merge contains:
```
<h1>My Story Book</h1>
<h2>Chapter 1</h2><p>This is chapter one.</p>
<h2>Chapter 2</h2><p>This is chapter two.</p>
<h2>Chapter 3</h2><p>This is chapter three.</p>
```

2. Develop a simple Java user-interactive containerized application (e.g., calculator) using Docker.

1. Create project folder:
```
mkdir java-calculator
cd java-calculator
```
2. Create Java file:
```
cat << 'EOF' > Calculator.java
import java.util.*;
public class Calculator {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a: ");
    int a = sc.nextInt();
    System.out.print("Enter b: ");
    int b = sc.nextInt();
    System.out.println("Sum = " + (a + b));
  }
}
EOF
```
3. Create Dockerfile:
```
cat << 'EOF' > Dockerfile
FROM openjdk:17
WORKDIR /app
COPY Calculator.java .
RUN javac Calculator.java
CMD ["java", "Calculator"]
EOF
```
4. Build Docker image:
```
docker build -t calc-app .
```

5. Run container:
```
docker run -it calc-app
```

6. User interaction example:
```
Enter a: 5
Enter b: 7
Sum = 12
```

3. Automate deployment of that containerized application using Kubernetes Deployment + Service.

1. Build and push Docker image to Docker Hub:
   docker build -t <your-dockerhub-username>/calc-app .
   docker push <your-dockerhub-username>/calc-app
2. Create Kubernetes Deployment:
   cat << 'EOF' > deployment.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: calc-deploy
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: calc
     template:
       metadata:
         labels:
           app: calc
       spec:
         containers:
         - name: calc
           image: <your-dockerhub-username>/calc-app
           stdin: true
           tty: true
   EOF

3. Create Service:
   cat << 'EOF' > service.yaml
   apiVersion: v1
   kind: Service
   metadata:
     name: calc-service
   spec:
     type: ClusterIP
     selector:
       app: calc
     ports:
     - port: 80
       targetPort: 8080
   EOF
4. Apply Kubernetes files:
   kubectl apply -f deployment.yaml
   kubectl apply -f service.yaml

5. Check resources:
   kubectl get pods
   kubectl get svc
6. Connect to interactive calculator:
   kubectl exec -it <pod-name> -- java Calculator

<u>Set 4:</u>

1. Build an HTML page with a profile card (photo, name, description) and push it to GitHub.

1. Create project folder:
```
mkdir profile-card
cd profile-card
```

2. Create index.html:
```
cat << 'EOF' > index.html
<!DOCTYPE html>
<html>
<body>
  <div style="width:200px;padding:10px;border:1px solid #000;text-align:center">
    <img src="https://via.placeholder.com/150" width="150">
    <h3>Your Name</h3>
    <p>Short description about you.</p>
  </div>
</body>
</html>
EOF
```

3. Initialize Git:
```
git init
```

4. Add file:
```
git add index.html
```

5. Commit:
```
git commit -m "Add profile card page"
```

6. Create GitHub repo (e.g., profile-card)

7. Connect local → GitHub:
```
git branch -M main
git remote add origin https://github.com/<your-username>/profile-card.git
```

8. Push:
```
git push -u origin main
```

2. List the installation steps for Docker and demonstrate container/content management commands

1. Install Docker on Mac:
   - Download & install Docker Desktop from:
     https://www.docker.com/products/docker-desktop
   - Open Docker Desktop
   - Verify installation:
     ```
     docker --version
     ```

2. Basic Docker Container Commands:
   - Pull image:
     ```
     docker pull nginx
     ```

- Run container:
  docker run -d --name mynginx -p 8080:80 nginx

- List running containers:
  docker ps

- List all containers:
  docker ps -a

- Stop container:
  docker stop mynginx

- Start container:
  docker start mynginx

- Remove container:
  docker rm mynginx

3. Basic Image & Content Commands:
- List images:
  docker images

- Remove image:
  docker rmi nginx

- Execute inside container:
  docker exec -it mynginx sh

- Copy file from container:
  docker cp mynginx:/path/in/container/file.txt .

- Copy file to container:
  docker cp test.txt mynginx:/test.txt

3. Run a Java Selenium script to test Google and MGIT website.

1. Install Java & Maven:
   brew install openjdk
   brew install maven

2. Create Maven project:
   mkdir selenium-test
   cd selenium-test
   mvn archetype:generate \
     -DgroupId=demo \
     -DartifactId=selenium-test \
     -DarchetypeArtifactId=maven-archetype-quickstart \
     -DinteractiveMode=false
   cd selenium-test

3. Add Selenium dependency:

Open pom.xml and add inside <dependencies>:

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>4.20.0</version>
</dependency>
```

4. Create Selenium test file:
   nano src/main/java/demo/App.java

   Paste this code:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class App {
  public static void main(String[] args) {
    WebDriver d = new ChromeDriver();

    // Test Google
    d.get("https://www.google.com");
    System.out.println("Google Title: " + d.getTitle());

    // Test another website (PUT YOUR LINK HERE)
    d.get("https://example.com");
    System.out.println("Second Website Title: " + d.getTitle());

    d.quit();
  }
}
```

5. Run the script:
   mvn clean install
   mvn exec:java -Dexec.mainClass=demo.App

Set 5:

1. Develop an HTML page showing your favorite books/movies and push it to GitHub.

1. Create project folder:
   mkdir favorites-page
   cd favorites-page

2. Create HTML file:
   cat << 'EOF' > index.html

```
<!DOCTYPE html>
<html>
<body>
  <h2>My Favorite Books</h2>
  <ul>
    <li>Book 1</li>
    <li>Book 2</li>
```

```
    <li>Book 3</li>
  </ul>

  <h2>My Favorite Movies</h2>
  <ul>
    <li>Movie 1</li>
    <li>Movie 2</li>
    <li>Movie 3</li>
  </ul>
</body>
</html>
EOF
```

3. Initialize Git:
   git init

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add favorite books and movies page"

6. Create a new empty repo on GitHub (for example: favorites-page)

7. Connect local repo to GitHub:
   git branch -M main
   git remote add origin https://github.com/<your-username>/favorites-page.git

8. Push to GitHub:
   git push -u origin main

2. Create a branch bio-update, add 3 hobbies to a file and push the branch.

1. Create a file with hobbies:
   echo "My Hobbies:
   1. Reading
   2. Gaming
   3. Traveling" > hobbies.txt

2. Initialize Git (if not already):
   git init

3. Create and switch to the branch bio-update:
   git checkout -b bio-update

4. Add the file:
   git add hobbies.txt

5. Commit the file:
   git commit -m "Add 3 hobbies"

6. Connect to GitHub repo (if not already connected):

git remote add origin https://github.com/<your-username>/<repo-name>.git

7. Push the branch:
git push -u origin bio-update

3. Create and run a Simple Python containerized application (e.g., number guessing) using Docker.

1. Create project folder:
mkdir python-guess
cd python-guess

2. Create Python script:
cat << 'EOF' > app.py
import random
n = random.randint(1,10)
g = int(input("Guess (1–10): "))
print("Correct!" if g == n else f"Wrong, number was {n}")
EOF

3. Create Dockerfile:
cat << 'EOF' > Dockerfile
FROM python:3.10
WORKDIR /app
COPY app.py .
CMD ["python", "app.py"]
EOF

4. Build image:
docker build -t guess-app .

5. Run container (interactive):
docker run -it guess-app

Set 6:

1. Design an HTML Weekly Timetable and upload it to GitHub.

1. Create project folder:
mkdir weekly-timetable
cd weekly-timetable

2. Create HTML file:
cat << 'EOF' > index.html
<!DOCTYPE html>
<html>
<body>
<h2>Weekly Timetable</h2>
<table border="1" cellpadding="8">
<tr>
<th>Day / Time</th>
<th>9–10</th>
<th>10–11</th>

```
      <th>11–12</th>
      <th>12–1</th>
    </tr>
    <tr>
      <td>Monday</td>
      <td>Maths</td>
      <td>Physics</td>
      <td>Break</td>
      <td>CS</td>
    </tr>
    <tr>
      <td>Tuesday</td>
      <td>English</td>
      <td>Maths</td>
      <td>Break</td>
      <td>Electronics</td>
    </tr>
    <tr>
      <td>Wednesday</td>
      <td>CS</td>
      <td>Physics</td>
      <td>Break</td>
      <td>Lab</td>
    </tr>
    <tr>
      <td>Thursday</td>
      <td>Maths</td>
      <td>English</td>
      <td>Break</td>
      <td>CS</td>
    </tr>
    <tr>
      <td>Friday</td>
      <td>Electronics</td>
      <td>Maths</td>
      <td>Break</td>
      <td>Sports</td>
    </tr>
    <tr>
      <td>Saturday</td>
      <td>CS</td>
      <td>Lab</td>
      <td>Break</td>
      <td>Club</td>
    </tr>
  </table>
</body>
</html>
EOF
```

3. Initialize Git:

```
git init
```

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add weekly timetable page"

6. Create a new empty repo on GitHub (for example: weekly-timetable)

7. Connect local repo to GitHub:
   git branch -M main
   git remote add origin https://github.com/<your-username>/weekly-timetable.git

8. Push to GitHub:
   git push -u origin main

2. Write steps to integrate Kubernetes with Docker, and explore basic kubectl commands.

1. Install Docker:
   brew install --cask docker
   open /Applications/Docker.app

2. Install Kubernetes (kubectl + cluster):
   brew install kubectl
   brew install minikube

3. Start Kubernetes cluster (with Docker as driver):
   minikube start --driver=docker

4. Verify integration:
   kubectl version
   kubectl get nodes

5. Basic kubectl commands:
   kubectl get pods
   kubectl get deployments
   kubectl get services
   kubectl create deployment myapp --image=nginx
   kubectl expose deployment myapp --type=NodePort --port=80
   kubectl delete deployment myapp
   kubectl delete service myapp

3. Create a job that runs every 2 minutes and prints the system date/time in Jenkins.

1. Open Jenkins → New Item → Pipeline → OK.

2. In Pipeline script box paste:

```
pipeline {
  agent any
  triggers {
    cron('H/2 * * * *')
  }
```

```
  stages {
    stage('Show Time') {
      steps {
        sh 'date'
      }
    }
  }
}
```

3. Save → Jenkins will run the job every 2 minutes and print the system date/time.

Set 7:

1. Create an HTML Resume page (Education, Skills, and Projects) and push it.

1. Create folder:
   mkdir resume
   cd resume

2. Create easiest HTML resume:
   cat << 'EOF' > index.html
   <h2>My Resume</h2>

   <h3>Education</h3>

     B.Tech CSE
     Intermediate

   <h3>Skills</h3>

     HTML
     Python
     Git

   <h3>Projects</h3>

     Project 1
     Project 2

   EOF

3. Initialize Git:
   git init

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add simple resume page"

6. Create new empty GitHub repo (name: resume)

7. Connect local to GitHub:
   git branch -M main
   git remote add origin https://github.com/<your-username>/resume.git

8. Push:
   git push -u origin main

2. Build a containerized version of the application from using a Dockerfile.

1. Create project folder:
   mkdir app-container
   cd app-container

2. Create a simple application (app.py):
   cat << 'EOF' > app.py
   print("Hello from containerized app!")
   EOF

3. Create the shortest Dockerfile:
   cat << 'EOF' > Dockerfile
   FROM python:3.10
   COPY app.py .
   CMD ["python", "app.py"]
   EOF

4. Build Docker image:
   docker build -t myapp .

5. Run the container:
   docker run myapp

3. Write a JavaScript program for Calculator and test it using Selenium WebDriver.

1. Create project folder:
   mkdir calc-selenium
   cd calc-selenium

2. Create smallest calculator (supports + - * /):
   cat << 'EOF' > calc.html
   <input id=a>
   <select id=o>
     <option>+</option>
     <option>-</option>
     <option>*</option>
     <option>/</option>
   </select>
   <input id=b>
   <button onclick="out.innerText = eval(a.value + o.value + b.value)">Calc</button>

   EOF

3. Start local server:
```
python3 -m http.server 8000
```

4. Create smallest Maven Selenium project:
```
mkdir -p test/src/main/java/demo
cd test
cat << 'EOF' > pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>demo</groupId>
  <artifactId>calc-test</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.20.0</version>
    </dependency>
  </dependencies>
</project>
EOF
```

5. Create Selenium WebDriver test:
```
cat << 'EOF' > src/main/java/demo/App.java
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;

public class App {
  public static void main(String[] args) {
    WebDriver d = new ChromeDriver();
    d.get("http://localhost:8000/calc.html");

    d.findElement(By.id("a")).sendKeys("8");
    d.findElement(By.id("b")).sendKeys("4");
    d.findElement(By.id("o")).sendKeys("*");   // choose operator
    d.findElement(By.tagName("button")).click();

    System.out.println("Result: " + d.findElement(By.id("out")).getText());
    d.quit();
  }
}
EOF
```

6. Run the Selenium test:
```
mvn clean install
mvn exec:java -Dexec.mainClass=demo.App
```

Set 8:

1. Create a simple HTML page to display tables and lists (any structure) and push it to GitHub.

1. Create project folder:
   mkdir tables-lists
   cd tables-lists

2. Create simple HTML page (tables + lists):
   cat << 'EOF' > index.html
   <!DOCTYPE html>
   <html>
   <body>
    <h2>My Table</h2>
    <table border="1">
      <tr><th>Name</th><th>Age</th></tr>
      <tr><td>Alice</td><td>20</td></tr>
      <tr><td>Bob</td><td>22</td></tr>
    </table>

    <h2>My List</h2>

      Item 1
      Item 2
      Item 3

   </body>
   </html>
   EOF

3. Initialize Git:
   git init

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add simple tables and lists page"

6. Create a new empty repo on GitHub (example name: tables-lists)

7. Connect local repo to GitHub:
   git branch -M main
   git remote add origin https://github.com/<your-username>/tables-lists.git

8. Push to GitHub:
   git push -u origin main

2. Build a multi-stage Git branching task where each branch stores a part of a secret message and merge into main.

1. Create project folder:
   mkdir secret-branches
   cd secret-branches

2. Initialize Git:

```
git init
echo "" > secret.txt
git add secret.txt
git commit -m "Base file"
```

3. Branch 1 – part of message:

```
git checkout -b part1
echo "The secret is" >> secret.txt
git add secret.txt
git commit -m "Add part 1"
```

4. Branch 2 – part of message:

```
git checkout main
git checkout -b part2
echo "hidden inside" >> secret.txt
git add secret.txt
git commit -m "Add part 2"
```

5. Branch 3 – part of message:

```
git checkout main
git checkout -b part3
echo "these branches." >> secret.txt
git add secret.txt
git commit -m "Add part 3"
```

6. Merge all parts into main:

```
git checkout main
git merge part1
git merge part2
git merge part3
```

7. Final secret.txt contains:

```
The secret is
hidden inside
these branches.
```

3. Write a JavaScript program for Calculator and test it using Selenium WebDriver.

1. Create project folder:

```
mkdir js-calc-test
cd js-calc-test
```

2. Create smallest calculator (HTML + JS):

```
cat << 'EOF' > calc.html
<input id=a>
<select id=o>
  <option>+</option>
  <option>-</option>
  <option>*</option>
  <option>/</option>
</select>
<input id=b>
<button onclick="out.innerText = eval(a.value + o.value + b.value)">Calc</button>
```

```
<p id=out></p>
EOF
```

3. Start a simple web server:
```
python3 -m http.server 8000
```

4. Create simplest Maven Selenium project:
```
mkdir -p test/src/main/java/demo
cd test
```

5. Create pom.xml:
```
cat << 'EOF' > pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>demo</groupId>
  <artifactId>selenium-test</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.20.0</version>
    </dependency>
  </dependencies>
</project>
EOF
```

6. Create shortest Selenium test:
```
mkdir -p src/main/java/demo
cat << 'EOF' > src/main/java/demo/App.java
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;

public class App {
  public static void main(String[] args) {
    WebDriver d = new ChromeDriver();
    d.get("http://localhost:8000/calc.html");

    d.findElement(By.id("a")).sendKeys("8");
    d.findElement(By.id("b")).sendKeys("4");
    d.findElement(By.id("o")).sendKeys("*");
    d.findElement(By.tagName("button")).click();

    System.out.println("Result = " + d.findElement(By.id("out")).getText());
    d.quit();
  }
}
EOF
```

7. Run the Selenium test:
```
mvn clean install
mvn exec:java -Dexec.mainClass=demo.App
```

Set 9:

1. Build an HTML page with a gallery of 3–4 images with captions and upload it.

1. Create folder:
   mkdir gallery
   cd gallery

2. Create the smallest HTML gallery:
   cat << 'EOF' > index.html
   <img src="https://via.placeholder.com/150"> Caption 1
   <img src="https://via.placeholder.com/150"> Caption 2
   <img src="https://via.placeholder.com/150"> Caption 3
   EOF

3. Initialize Git:
   git init

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add small image gallery"

6. Create a new empty GitHub repo (name: gallery)

7. Connect repo:
   git branch -M main
   git remote add origin https://github.com/<your-username>/gallery.git

8. Push:
   git push -u origin main

2. Create a Docker image for the HTML gallery application and run it with port mapping.

1. Create project folder:
   mkdir gallery-docker
   cd gallery-docker

2. Create smallest HTML gallery:
   cat << 'EOF' > index.html
   <img src="https://via.placeholder.com/150"> Caption 1
   <img src="https://via.placeholder.com/150"> Caption 2
   <img src="https://via.placeholder.com/150"> Caption 3
   EOF

3. Create the shortest Dockerfile:
   cat << 'EOF' > Dockerfile
   FROM nginx:alpine
   COPY index.html /usr/share/nginx/html/index.html
   EOF

4. Build Docker image:
   docker build -t gallery-app .

5. Run container with port mapping:
   docker run -d -p 8080:80 --name gallery gallery-app

6. Access in browser:
   http://localhost:8080

3. Automate the gallery application deployment using a Kubernetes Deployment + Service YAML.

1. Push your gallery image:
   docker build -t <your-user>/gallery .
   docker push <your-user>/gallery

2. Create smallest Deployment + Service:
```
   cat << 'EOF' > gallery.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata: { name: gallery }
   spec:
     selector: { matchLabels: { app: g } }
     replicas: 1
     template:
       metadata: { labels: { app: g } }
       spec:
         containers:
         - name: g
           image: <your-user>/gallery
   ---
   apiVersion: v1
   kind: Service
   metadata: { name: gallery-svc }
   spec:
     type: NodePort
     selector: { app: g }
     ports:
     - port: 80
       targetPort: 80
       nodePort: 30080
   EOF
```

3. Apply it:
   kubectl apply -f gallery.yaml

4. Check:
   kubectl get pods
   kubectl get svc

5. Access:
   http://localhost:30080

Set 10:

1. Create an HTML page with a Pizza Order form and commit it to GitHub.

1. Create folder:
   mkdir pizza
   cd pizza

2. Create smallest Pizza Order HTML:
   cat << 'EOF' > index.html
   <form>
    <input placeholder="Name">
    <select><option>S</option><option>M</option><option>L</option></select>
    <input placeholder="Toppings">
    <textarea placeholder="Address"></textarea>
    <button>Order</button>
   </form>
   EOF

3. Initialize Git:
   git init

4. Add file:
   git add index.html

5. Commit:
   git commit -m "Add pizza form"

6. Connect to GitHub:
   git branch -M main
   git remote add origin https://github.com/<user>/pizza.git

7. Push:
   git push -u origin main

2. Write installation steps for Selenium and execute a JavaScript (Node.js) Selenium script for UI testing.

1. Install Node.js:
   brew install node

2. Create project folder:
   mkdir selenium-js
   cd selenium-js

3. Install Selenium WebDriver:
   npm init -y
   npm install selenium-webdriver

4. Create UI test script:
   cat << 'EOF' > test.js
   const {Builder, By} = require('selenium-webdriver');

```
(async () => {
  let d = await new Builder().forBrowser('chrome').build();
  await d.get("https://www.google.com");
  let title = await d.getTitle();
  console.log("Google Title:", title);
  await d.quit();
})();
EOF
```

5. Run the Selenium test:
```
node test.js
```

3. Develop Selenium test cases for the containerized application you created in the previous questions.

1. Ensure your gallery container is running:
```
docker run -d -p 8080:80 --name gallery <your-image>
```

2. Create test folder:
```
mkdir gallery-test
cd gallery-test
```

3. Install Node + Selenium:
```
brew install node
npm init -y
npm install selenium-webdriver
```

4. Create shortest Selenium test (test.js):
```
cat << 'EOF' > test.js
const {Builder, By} = require('selenium-webdriver');

(async () => {
  let d = await new Builder().forBrowser('chrome').build();
  await d.get("http://localhost:8080");

  console.log("Page Title:", await d.getTitle());
  console.log("First Image Present:",
    await d.findElements(By.tagName("img")).then(e => e.length > 0)
  );

  d.quit();
})();
EOF
```

5. Run Selenium test:
```
node test.js
```