Task 2: Setting Up Jenkins for Continuous Integration

Step 1: Install Jenkins

- Install Jenkins on your laptop from https://jenkins.io

- Install Git, Python, and pip as prerequisites

- Start Jenkins locally using http://localhost:8080

Step 2: Create a Simple Pipeline Project

- Create a GitHub repository with these files:

  * math_ops.py

  * test_sample.py

  * requirements.txt

  * Jenkinsfile

- Sample Python Code:

math_ops.py:

```
def add(a, b): return a + b
```

test_sample.py:

```
from math_ops import add

def test_add(): assert add(2, 3) == 5
```

re.txt:

```
pytest
```

- **Jenkinsfile content:**

```
- pipeline {
      agent any stages
      {
        stage('Clean Workspace') { steps { deleteDir() } } stage('Clone
        Repo') {
            steps { git branch: 'main', url: 'https://github.com/RETHESSHED/DevOps-Project.git' }}
        stage('Set Up Environment') { steps
          {
            bat '''
              python -m venv venv  call
              venv\Scripts\activate
              pip install -r requirements.txt
            '''
          }
        }
        stage('Run Tests') { steps {
            bat '''
              call venv\Scripts\activate
              pytest
            '''
          }
        }
      }
    }
```

EXPLANATION:

```
pipeline {
  agent any
```

- **pipeline**: Declares the beginning of a declarative Jenkins pipeline.
- **agent any**: Tells Jenkins to run this pipeline on **any available agent/node**. If you have multiple agents, Jenkins will pick one.

📁 Stage 1: Clone the GitHub Repository

```
  stages {
    stage('Clone Repo') {
      steps {
        git branch: 'main', url: 'https://github.com/RETHESSHED/DevOps-Project.git'
      }
    }
```

- **stage('Clone Repo')**: A named step in the pipeline to organize and label output.
- **git**: This is the Jenkins Git plugin.
  - branch: 'main': Specifies which branch to clone.
  - url: 'https://github.com/RETHESSHED/DevOps-Project.git': Your GitHub repo URL.
- Jenkins automatically checks out the repo into the workspace.

🛍 Stage 2: Set Up Python Environment & Install Dependencies

```
    stage('Install Dependencies') {
      steps {
        bat '''
          python -m venv venv
          call venv\\Scripts\\activate
          pip install -r re.txt
        '''
      }
    }
```

- **stage('Install Dependencies')**: Prepares the Python environment.
- **bat**: Runs a **Windows batch command**.
- Inside the batch command:
  1. python -m venv venv: Creates a virtual environment named venv in the current directory.
  2. call venv\\Scripts\\activate: Activates the virtual environment. call is used so the script continues after activation.
  3. pip install -r re.txt: Installs all packages listed in re.txt. (This is typically named requirements.txt, but you can name it anything.)

 Stage 3: Run Tests:

```
    stage('Run Tests') {
      steps {
        bat '''
          call venv\\Scripts\\activate
          pytest
        '''
      }
```

}

- **stage('Run Tests')**: Executes unit or integration tests.
- **pytest**: Runs tests written using the pytest framework. Jenkins will capture the output and display it in the console log.

Step 3: Auto-trigger Build on Git Push

- Install and run ngrok:

ngrok http 8080

- Copy the HTTPS forwarding URL from ngrok

- In GitHub repo > Settings > Webhooks > Add Webhook:

* Payload URL: https://your-ngrok-url/github-webhook/

* Content-Type: application/json

* Trigger: Just the push event

- In Jenkins job config:

* Check: GitHub hook trigger for GITScm polling

Step 4: Run and Analyze the Build

- Make a change and push to GitHub: git

commit -am "trigger build"

git push origin main

- Jenkins will auto-run the pipeline

- Check the Console Output for pytest results

- If tests pass, build shows SUCCESS

- If tests fail, the error will be logged and build shows FAILURE

Summary of Flow

1. **Clone** your code from GitHub.
2. **Create and activate** a virtual environment.
3. **Install dependencies** like Flask, Django, or Pytest from re.txt.
4. **Run tests** and check results.