# Introduction to CI/CD

## Why CI/CD?

- Manual builds and deployments are slow and error-prone.
- CI/CD automates building, testing, and deploying software.
- Ensures faster feedback and reliable releases.

## CI vs CD

- Continuous Integration (CI): Automatically build & test every commit.
- Continuous Delivery (CD): Always have deployable builds.
- Continuous Deployment: Automatically release every passing build.

# The CI/CD Pipeline

- Code → Build → Test → Deploy → Monitor.
- Each stage ensures quality and reliability.
- Common tools: GitHub Actions, Jenkins, GitLab CI.

## Tools in the Ecosystem

- Version Control: Git, GitHub, GitLab.
- CI/CD: Jenkins, GitHub Actions, GitLab CI, CircleCI.
- Containers: Docker.
- Deployment: AWS, Azure, Netlify, Heroku.

# Example: GitHub Actions Workflow (JavaScript)

- A minimal CI pipeline for a Node.js project.
- Automatically runs on each push.

```
yaml
name: CI Pipeline
on: [push]
jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
```

# Example: GitHub Actions Workflow (Python)

- A simple pipeline for Python projects using pytest.
- Runs automatically on
- push or pull request.

```
yaml
name: Python CI
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'
      - name: Install dependencies
        run: pip install -r requirements.tx
      - name: Run tests
        run: pytest
```

## Example Flow

- 1. Developer pushes code to GitHub.
- 2. Pipeline triggers automatically.
- 3. Tests run and report feedback.
- 4. Fix and push → Re-run automatically.
- 5. Passing tests = ready to deploy.

# Sample Test File (Python)

It checks out the repo (so it can see the test file).
It sets up Python and installs dependencies.
It executes pytest in the repo folder.
Pytest automatically discovers any files named test_*.py or *_test.py.
It runs those tests and reports success/failure.

```
- name: Run tests
  run: pytest
```

```
python
# simple_test.py
def add(a, b):
   return a + b

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

```
================= test session starts ==================
collected 1 item

simple_test.py .                                    [100%]

================== 1 passed in 0.03s ===================
```

## Common Pitfalls

- No tests → CI gives no feedback.
- Slow pipelines discourage use.
- Flaky tests cause confusion.
- Skipping deployment verification.

## Best Practices

- Keep pipelines fast (<10 min).
- Run tests and linting on every push.
- Protect main branches with required CI checks.
- Use environment variables for secrets.
- Make build status visible to all developers.