CI/CD: THE SIMPLE WAY

From Code to Production

"How to stop worrying and love automation"

The "Cooking" Analogy

Building software is like running a restaurant kitchen.

Code = Ingredients

(Raw materials)

Build = Cooking

(Compile/Bundle)

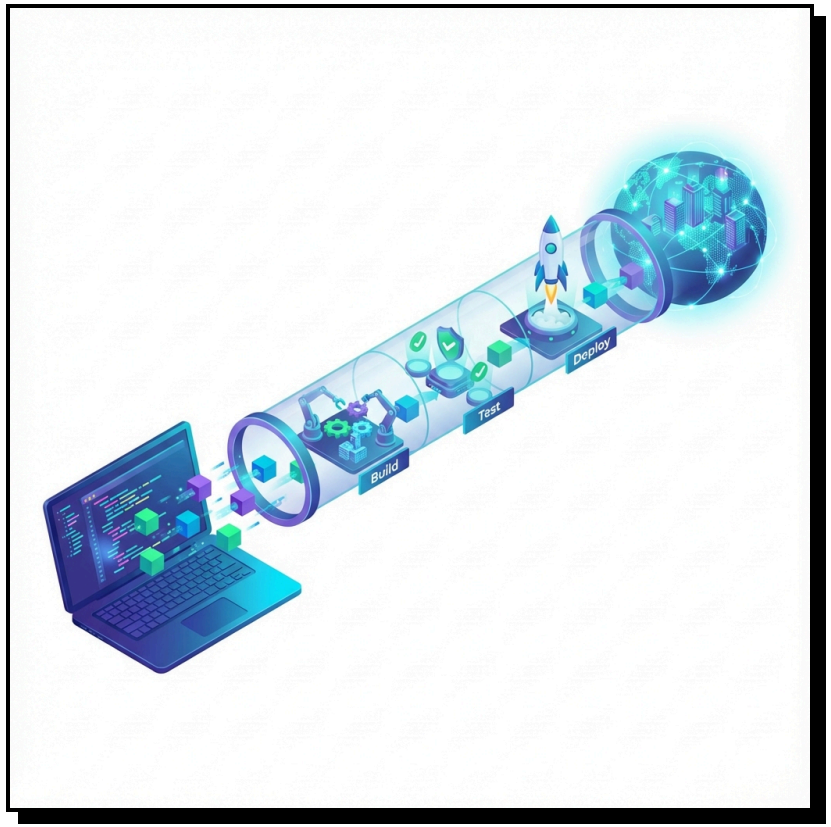
Deploy = Serving

(Give to customer)

CI/CD is an **automated chef** that cooks and serves the food for you, perfectly, every single time.

The Pipeline Vision

Imagine a tube connecting your laptop to the world.



1. **Build Factory**: Assembles the code.
2. **Scanner Gate**: Checks for bugs/quality.
3. **Launch Pad**: Blasts it to the internet.

The "Old Way" (Manual)

1. Developer writes code
2. Developer emails code to Bob
3. Bob puts it on a USB stick
4. Bob walks to the server room
5. **CRASH!** "It doesn't work!"

The Problem?

- **Slow**: Takes days/weeks.
- **Risky**: Human error (Bob forgot a file).
- **Stressful**: Deploying on Friday is terrifying.

The "New Way" (CI/CD)

1. Developer pushes code to GitHub
2. **Automation wakes up**
3. Automation tests the code
4. Automation deploys the code
5. **Success!**

The Benefit?

- **Fast**: Takes minutes.
- **Safe**: Scripts don't forget files.
- **Reliable**: Deploy on Friday with confidence.

PART 2: GITHUB ACTIONS

Your First Automation

Setup in 3 Steps

GitHub Actions is **free** and built right into GitHub.

1. Create the File

Inside your repo, make a folder path:
`.github/workflows/pipeline.yml`

2. Add the Instructions

Write **YAML** (it's list of instructions).

3. Push and Watch

Commit your code, and see the indicators turn green!

The Recipe (YAML)

This is a real workflow file. It is simple instructions.

```yaml
name: My First Automation

on: [push]                  # Trigger: When code is pushed

jobs:
  check-code:               # Job Name
    runs-on: ubuntu-latest  # Computer Type (Cloud)

    steps:
      - uses: actions/checkout@v3  # Step 1: Get Code

      - name: Run Tests            # Step 2: Test It
        run: npm test
```

Hands-On Time

Let's build a safety net.

We want a workflow that runs our tests automatically. If the tests pass, we're happy. If they fail, we want to know!

Your Mission:

1. Open VS Code.
2. Find `.github/workflows/basic-ci.yml`.
3. Read the comments.
4. **Action**: Push it to GitHub and see it run!

PART 3: THE NEXT STEPS

Moving Beyond Basic Testing

Level Up: Docker (The Package)

Testing code is good. Packaging it is better.

Goal: Create a "Container" that runs anywhere (laptop, server, cloud).

The Recipe Update

1. **Test** the code (Ingredients are fresh?)
2. **Build** the Docker Image (Cook the meal)
3. **Push** to Registry (Put it in the freezer for later)

```
      - name: Build & Push Docker Image
        run: |
          docker build -t my-app .
          docker push my-repo/my-app
```

Level Up: Pull Requests (The Shield)

Don't push straight to Production!

The "Branch Protection" Workflow

1. Developer creates a **Branch** (`feature-login`).
2. Developer opens a **Pull Request (PR)**.
3. **GitHub Actions** runs automatically on the PR.
4. If tests Pass
   
   → **Merge** allowed.
5. If tests Fail
   
   → **Merge** blocked.

This prevents bad code from ever reaching the main branch.

Level Up: Security (DevSecOps)

Hacker-proof your code automatically.

"Shift Left" Security

Find vulnerabilities **before** you deploy, not after.

```
      - name: Run Security Scan
        run: npm audit  # Checks for known security holes
```

If a security hole is found, the robot stops the line.

PART 4: JENKINS

The Enterprise Standard

Smart Home vs Factory

GitHub Actions

"The Smart Home"

- Easy to set up (Plug & Play).
- Great for most families (Startups/Projects).
- Lives in the Cloud.
- **Zero maintenance.**

Jenkins

"The Industrial Factory"

- Powerful, heavy machinery.
- Great for massive scale (Enterprises).
- You host it yourself (On-premise).
- **Requires a maintenance crew.**

Why "Enterprise"? (The 3 Reasons)

Why do big banks and legacy companies love Jenkins?

1. **Private Networks (Air-gapped)**
    - Some companies have servers with **NO Internet**.
    - GitHub Actions can't reach them. Jenkins lives inside the wall.

2. **Infinite Customization**
    - Need to restart a server in 1998? There is a plugin for that.
    - GitHub is opinionated; Jenkins lets you do *anything* (even bad things).

3. **Complex Auditing**
    - "Who approved this deploy?"
    - Jenkins allows complex "Manual Approval" gates required by law.

How They Run: A Deep Dive

Where does your code actually execute?

GitHub Actions

Runner: "Ephemeral VM"

1. GitHub creates a fresh VM.
2. Runs your code.
3. **Destroys the VM.**

- *Clean slate every time.*

Jenkins

Architecture: "Controller & Agents"

1. **Controller**: The boss (assigns work).
2. **Agent**: The worker node (runs code).

- *Agents are often persistent servers.*

> ***Analogy**: GitHub is an Uber (new car every time). Jenkins is owning a fleet of trucks (you maintain the tires).*

Jenkins Speaks a Different Language

The concepts are the same, but the syntax is different.

```
// Jenkinsfile (Groovy)

pipeline {
    agent any           // "runs-on: ubuntu-latest"

    stages {
        stage('Test') {
            steps {
                sh 'npm test'  // "run: npm test"
            }
        }
    }
}
```

It is just a different dialect for the same instructions.

READY TO LAUNCH?

"Automation is good, so long as you know exactly where to put the machine."

– Eliyahu Goldratt

Your Cheat Sheet

1. **CI (Continuous Integration)**:
    - Merging code often + Auto-testing.
    - *Goal: Don't break the app.*
2. **CD (Continuous Delivery)**:
    - Auto-deploying to users.
    - *Goal: Ship features fast.*
3. **YAML**:
    - The language we use to command the automation.

Now go automate something!