

# GitHub Actions

Automation for Modern Development

# Agenda

The Big Picture: Basic Architecture of GitHub Actions

The Building Blocks: Workflows, Jobs, Steps & Triggers

Scaling Up: The Matrix Build Concept

Hands-On #2: A Secure Java Matrix Build

The FIL Way: Specific Best Practices

Our Environment: Self-Hosted Runners at FIL

Hands-On #1: Your First Python Workflow

Security First: Managing Secrets

When Things Go Wrong: Troubleshooting & Debugging

Wrapping Up: Conclusion & Key Takeaways



Python\_CI\_Starter.zip

# Basic Architecture - The Big Picture

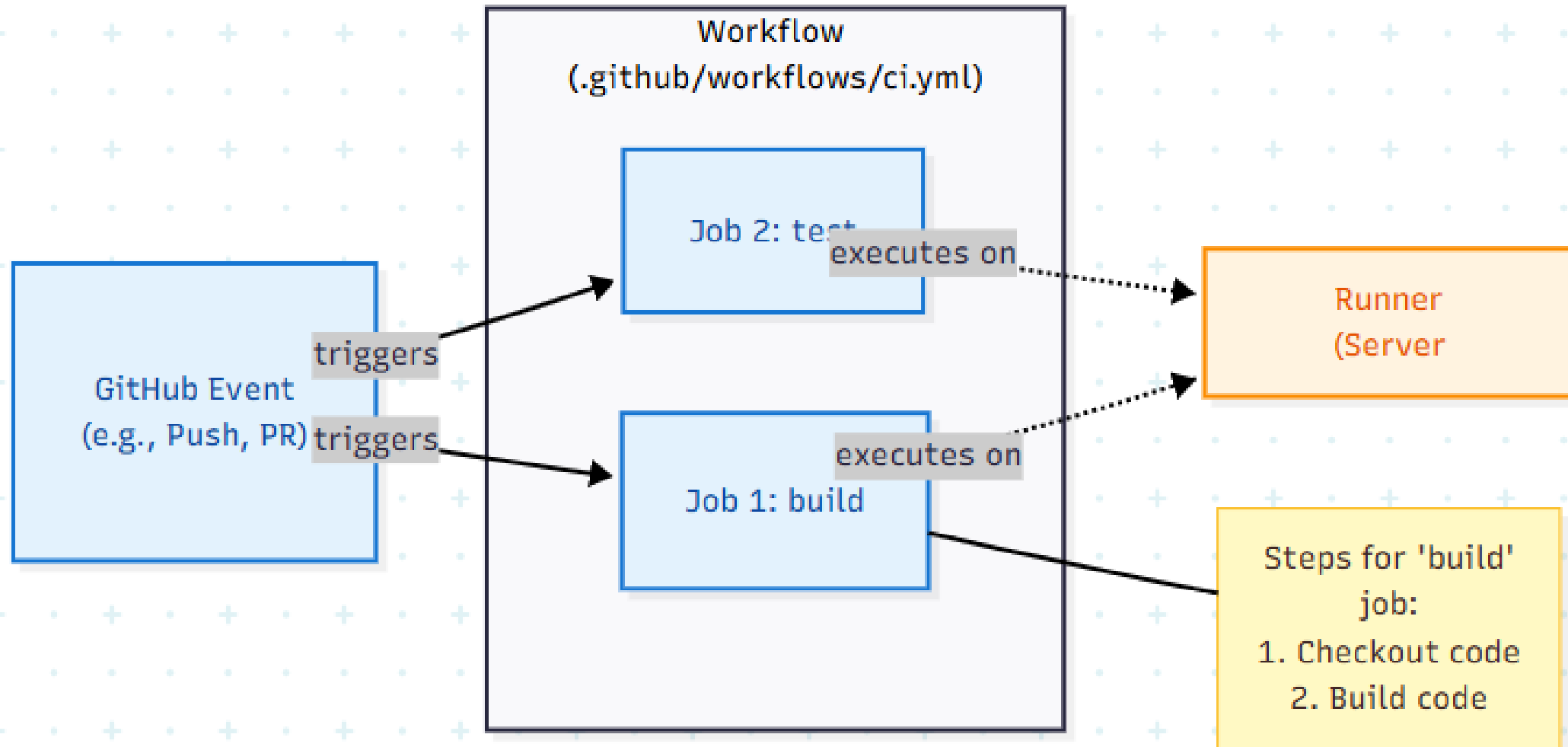
---

GitHub Actions is an automation platform that allows you to build, test, and deploy your code right from GitHub.



- **Event-Driven:** Workflows are triggered by events in your repository (e.g., push, pull\_request, schedule)
- **Workflows:** An automated process defined by a YAML file (.github/workflows/\*.yaml)
- **Runners:** A server that runs your workflow jobs. Can be GitHub-hosted or FaaS self-hosted
- **Jobs:** A set of steps that execute on the same runner
- **Steps:** Individual tasks. Can be shell commands (run) or pre-packaged scripts (uses)
- **Actions:** Reusable units of code shared across the community or built internally

# Basic Architecture - The Big Picture



# Self-Hosted Runners in FIL

---

## Why do we use Self-Hosted Runners?

For an environment like Personal Banking, security, control, and performance are non-negotiable.

- **Enhanced Security:** Runners are within FIL's network perimeter, reducing exposure. Configured to meet strict compliance standards (PCI DSS, GDPR)
- **Custom Hardware:** Specific CPU, memory, or GPU configurations for intensive build or testing jobs
- **Controlled Environment:** Access to internal resources like private repositories, databases, or secret vaults
- **Cost Management:** Optimized for our usage patterns, potentially lowering costs for high-volume jobs

💡 **Pro Tip:** Treat your self-hosted runners like production servers. They should be patched, monitored, and have clear lifecycle management. Use labels (e.g., windows-gpu, linux-large-memory) to route specific jobs to the right runners.

# The Building Blocks

## Workflows, Jobs, Steps, Triggers

Let's break down the syntax of a typical workflow file (.github/workflows/ci.yml):

```
# 1. Workflow Name
name: Basic CI Workflow

# 2. Trigger
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

# 3. Jobs
jobs:
  # Job ID
  build-and-test:
    # 4. Runner
    runs-on: ubuntu-latest # Or a FIL self-hosted runner label

    # 5. Steps
    steps:
      # Each '-' is a step
      - name: Check out repository code
        uses: actions/checkout@v4

      - name: Run a one-line script
        run: echo "Hello, FIL!"

      - name: Run a multi-line script
        run: |
          echo Add other build steps here
          ls -l
```

# Hands-On #1: A Simple Python CI Workflow

## Objective

Create a GitHub Action to automatically test a simple Python application on every push.

### Setup Requirements:

1. Create a new repository on GitHub
2. Add `app.py`:

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

3. Add `test_app.py`:


```
import unittest from app import add, subtract class TestMath(unittest.TestCase): def test_add(self): self.assertEqual(add(2, 3), 5) def test_subtract(self): self.assertEqual(subtract(5, 2), 3) if __name__ == '__main__': unittest.main()
```

4. Add `requirements.txt` (even if empty - good practice)

# Hands-On #1: Exercise Steps

## Your Task:

1. In your repository, create the directory path `.github/workflows/`
2. Inside that path, create a new file named `python-ci.yml`
3. **Write a workflow that does the following:**
  - Triggers on a push to the main branch
  - Runs on an ubuntu-latest runner
  - Checks out your repository's code
  - Sets up Python (version 3.9)
  - Installs any dependencies from `requirements.txt`
  - Runs the tests using the unittest module
4. Commit the file and push it to main
5. Check the "Actions" tab in your repository to see it run!

 **Need a hint?** Look for the `actions/setup-python` action.



# Hands-On #1: Solution

---

**File:** .github/workflows/python-ci.yml

```
name: Python CI

on:
  push:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set up Python 3.9
        uses: actions/setup-python@v5
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

      - name: Run Tests
        run: python -m unittest test_app.py
```

# The Matrix Build Concept

## What is it?


A way to run the same job multiple times with different configurations in parallel.

## Why use it?

Ensure your application works across different environments:



```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.8', '3.9', '3.10']
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}
      - name: Run tests
        run: python my_tests.py
```

 **Pro Tips:** Use a matrix build for any core, shared libraries to guarantee compatibility for all consuming services.

# Managing Secrets

 **Important:** NEVER hardcode credentials or sensitive information in your workflow files.

GitHub provides a secure way to store and use secrets:

- **Storage:** Secrets can be stored at the Repository, Environment, or Organization level
- **FIL Standard:** We primarily use Environment Secrets and FIL's central vault
- **Usage:** Secrets are passed as environment variables but not printed in logs (automatically redacted)

## How to Use:

```
steps:
  - name: Use the secret
    env:
      MY_SECRET_TOKEN: ${ secrets.API_TOKEN }
    run: |
      echo "Calling an API with a token..."
      # Your script would use the MY_SECRET_TOKEN env var
```

**Best Practice:** In GitHub, go to Settings > Secrets and variables > Actions to add repository secrets

# Troubleshooting & Debugging

---

## When a workflow fails, don't panic.

- **Check the Logs:** The Actions tab provides detailed logs for every step. The error is usually in the last few lines
- **Increase Verbosity:** Enable step debug logging by setting ACTIONS\_STEP\_DEBUG secret to true
- **Use the tmate Action:** For tricky issues, SSH directly into the runner session

⚠ **Important:** Using tmate or similar actions requires extreme caution and should only be done on non-production, isolated runners with explicit security approval. It creates a temporary public connection to our internal environment.

💡 **Pro Tip:** Before pushing a complex workflow, consider using a tool like 'act' to run your GitHub Actions locally. This can save time by catching syntax errors early.

# The FIL Way - Specific Practices

---

To maintain security, compliance, and consistency, we follow these practices:

- **Default to Self-Hosted:** All workflows for Personal Banking services must run on FIL-provisioned runners
- **Use Curated Actions:** Prefer actions from GitHub official (actions/) or FIL's security-vetted collection
- **Mandatory Security Scans:** Every workflow building artifacts must include SAST and dependency scanning
- **Environment Protection Rules:** Production deployments require specific branch triggers and manual approval
- **Standardized Naming:** Follow team conventions for workflow files and job names


# Conclusion & Key Takeaways

---

- **Automate Everything:** GitHub Actions increases speed and reduces human error
- **Security is Paramount:** Use Self-Hosted Runners and strict secret management
- **Start Simple, Scale Smart:** Begin with basic CI, then add complex features
- **Actions are Code:** Treat your .yml workflow files with the same care as application code

## Your Next Steps:

- Add a basic CI workflow to your service's repository
- Review your project's security and add scanning steps
- Explore the GitHub Marketplace for useful actions (check FIL's approved list!)

 **Pro Tip: Questions?**