Learn GitHub Actions

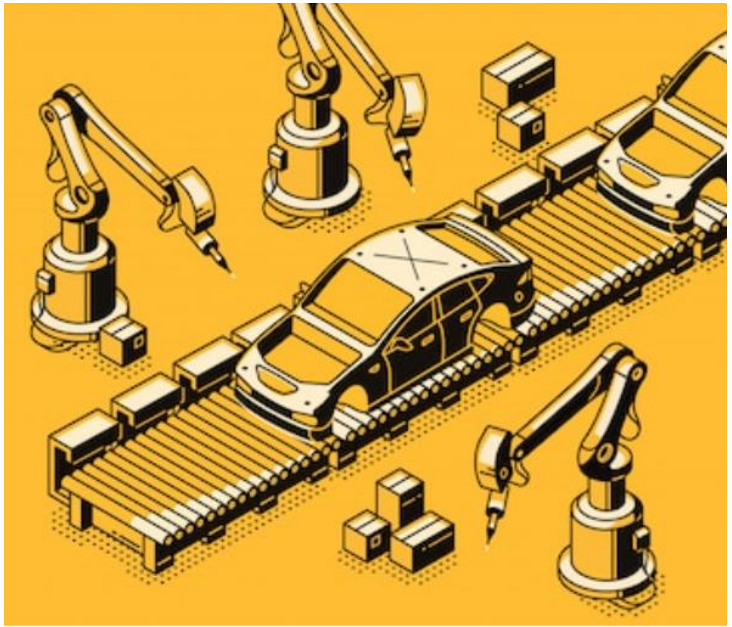
Instructor: Paul Furlan

Release: September 2022

https://www.linkedin.com/in/peafl

Twitter: @DeusPaul

What is DevOps?



Previously…

Stages:

- Planning
- Architectural Design
- Development
- Testing
- Deployment

Enter Agile


THE AGILE MANIFESTO
HTTP://WWW.AGILEMANIFESTO.ORG
INDIVIDUALS AND INTERACTIONS
OVER PROCESSES AND TOOLS
WORKING SOFTWARE
OVER COMPREHENSIVE DOCUMENTATION
CUSTOMER COLLABORATION
OVER CONTRACT NEGOTIATION
RESPONDING TO CHANGE
OVER FOLLOWING A PLAN

DevOps LifeCycle

- Continuous planning
- Continuous integration
- Continuous delivery & deployment
- Continuous operations

DevOps as a Service

GitHub

+

GitLab

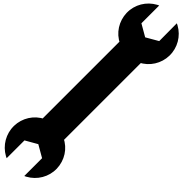
Version control / Source code management

- Collaborate on features
  - Feature branch workflow
  - Distributed development

- Revert to a previous state

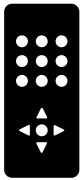
- Compare Code Changes

Git Commands: Setting up & logs



CONFIG

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```



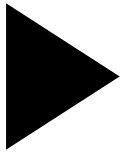
REMOTE

```
$ git remote -v
$ git remote add <name> <url>
```

LOG

```
$ git log
```

Git Commands: Getting started with repos



Init

```
$ git init
```

Clone

```
$ git clone <repo> <directory>
$ git clone http://example.com/gitproject.git
```

Git Commands: Git Files

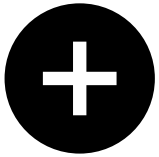
Tracked

```
$ git add
```

Untracked

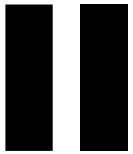
```
$ git rm --cached [filename]
```

Git Commands: Changes



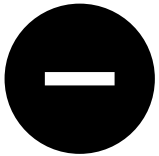
STAGED CHANGES

Changes added to staging area
use "$ git add" command to add files to staging



UNSTAGED CHANGES

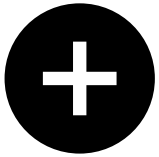
Files that have been changed but not added to staging



UNTRACKED CHANGES

Files not tracked by git
you can use the $git rm --cached <filename>

Git Commands: Working with Git



ADD

```
$ git add <filename>
```

COMMIT

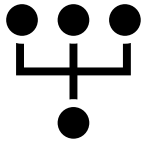
```
$ git commit –m "commit message"
```

STATUS

Provides list of staged, tracked and untracked files as well as commit status and merge conflicts.

```
$ git status
```

Git Commands: Branches



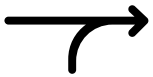
Branch

Create a copy of the main branch from your repository.
$ git branch <branch>

Checkout

Switch to another branch
$ git checkout <branch>

Merge & Merge Conflict

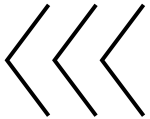


Used to combine one branch into another
Same line of code changed in 2 different commits causes a conflict
$ git merge <branch>

Git Commands: Remote repositories

Push

Upload local repository content to remote repository
$ git push <remote> <branch>



Fetch

Downloads content from a remote repository without updating local
$ git fetch <remote>

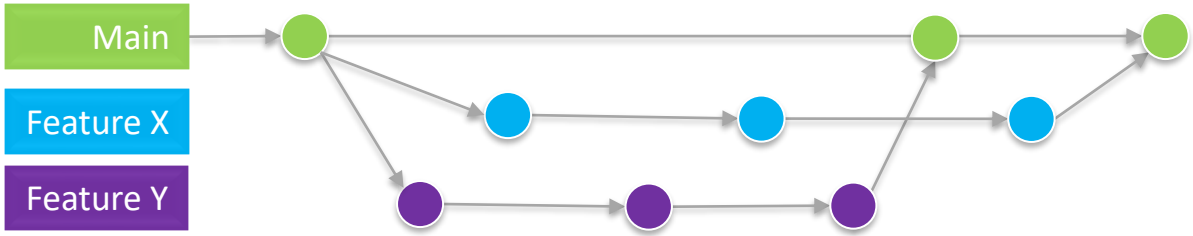
Pull

Downloads and merges changes to your local repository
$ git pull <remote>

GitHub flow

No concept of release; every feature is pushed live immediately
- Simpler and easier to adopt
- Main and release branches are combined into “trunk” branch
- New short-lived branches are created for every new feature
- Hotfixes are treated just like feature branches

“Commit early, commit often”



GitHub Overview

Exercise #1: GitHub Overview

Exercise #1: GitHub Overview

What is the GitHub actions platform?

GitHub Actions

Runners

Runners:

- Run workflow jobs
- GitHub-Hosted or Self-Hosted

GitHub-Hosted Runners:

- Hosted in Azure (Standard_DS2_v2) VM
- Ubuntu, Windows Server, MacOs
- Preinstalled software (Languages, tools, Package management)

macOS

YAML

YAML:

- Format & Represent Data
- Superset of JSON
- Easy to read / designed with human readability in mind
- Convert between JSON and YAML

Use Cases:

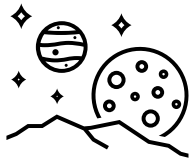
- Applications:
  - Ansible
  - Kubernetes
  - Gitlab
  - Openstack & More...

- Transmit data from 2 application components

- Import/Export data

Basically, almost everything related to Infrastructure as code

YAML



Indentation:

- Uses spaces for indentation

Example:

```
Jobs:
  first_job:
    runs-on: ubuntu-latest
```

Comments:

- Require the "#" hash/number symbol at the start of the line

Example:

```
#this is a comment
```

Note: Multiline comments are not accepted

YAML

Data Types:

- **Nulls**
  variable: null
  (null | Null | NULL | ~)

- **Numbers**
  variable: 123

- **Strings**
  variable: "123"

- **Booleans**
  Variable: True
  (True/False, Yes/No, On/Off)

Lists:

- Elements will be added as long as indentation is in place

- Lists can be defined in block or square brackets syntax:

variable:
- one
- two
- three

variable: [one, two, three]

YAML

Maps:

- Requires indentation and a key followed by a colon and space

- Can be defined in block or round bracket syntax:

cat:
  name: Mirana
  color: brown
  age: 3 years
  eye-color: green
  weight: 3.3kg

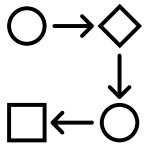
Cat: {name: Mirana, color: brown,...}

Nesting:

- Can nest maps in maps; lists in maps; maps in lists

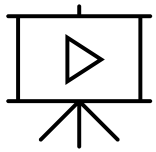
cat:
  name: Mirana
  characteristics:
    color: brown
    age: 3 years
    weight: 3.3kgs
  activities:
    - attack
    - jump
    - sleep

GitHub Actions Workflows



Workflow:

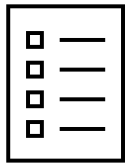
- File stored in ".github/workflows"
- Defines jobs and steps (automation procedures, instructions and actions)
- Defines a "pipeline" to build, test and deploy your GitHub projects



"on":

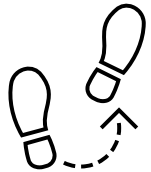
- Keyword to define the trigger that will initiate the workflow
- Examples: "push", "pull_request", "scheduled", "fork", "issue"…..

GitHub Actions Workflows: jobs



“jobs”:

- Group jobs to divide workflow into a list of activities (build, test, deploy)
- Defined with “jobs” keyword
- Jobs run in parallel by default
- “needs” keyword can be used to arrange jobs sequentially



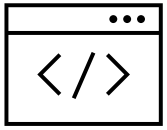
“job”:

- Group of steps to be executed by a runner within “job” code block
- Run instructions, procedures and actions within code block of “steps”
- Specify runner with “runs-on” keyword

GitHub Actions Workflows: steps

“steps”:

- Define automation procedures, instructions and actions to build, test and deploy your projects
- Identify step with “name” keyword
- “Id” used to reference



“shell”:

- Override default shell in the runner
- Pwsh, bash, sh, cmd, python

“run”:

- Run commands in the operating system shell

Exercise #2: Setup a runner and a Workflow

Exercise #2: Setup a self-hosted runner and run a Workflow on it

GitHub Actions Workflows: actions

GitHub Actions

“uses”:

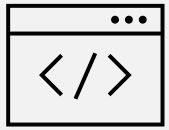
- Reusable unit of code (premade tasks/scripts)
- Pass input parameters with the “with” keyword

- Syntax:
- "actions/<action_name>@<version/commit/branch>"

```
...
jobs:
  job_with_GH_actions:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2

    - uses: actions/setup-node@v3
      with:
        node-version: 16

    - uses: wxdlong/hello-action@3dc69a523f937b57d06445e71f237b19565fb830
      with:
        who-to-greet: ‘live-lesson attendees’
```



GitHub Actions Workflows: variables

Secrets:

- Never commit secrets or sensitive information into your repository, use secrets instead
- ${{ secrets.<SECRET_NAME> }}

Variables (environment variables):

- Known as environment variables
- Can be defined at workflow level; job level or step level
- Syntax $(<variable_name>)

Naming Secrets:

- Only alphanumeric or underscores
- Must not start with “GITHUB_” prefix
- Must not start with a number
- Not case-sensitive

GitHub Actions Workflows: variables

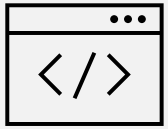
(x)
$(x)(x)$

Default environment variables:

- Predefined variables
- Contain useful values such as jobId, ref, runnerOS, paths & more..

Examples:
GITHUB_ACTION: name of action currently running
GITHUB_EVENT_NAME: event that triggered the workflow
GITHUB_JOB: job id of current job
GITHUB_REF: branch/tag that triggered workflow

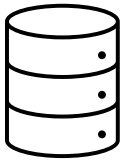
```
name: workflow name
env:
  WF_VAR: “Workflow Variable”
jobs:
  SomeJob:
  env:
    JOB_VAR: “Job Variable”
  steps:
    - name: test variables
      env:
        STEP_VAR: “Step Variable”
      run: |
        echo $WF_VAR
        echo $JOB_VAR
        echo $STEP_VAR
        echo ${{ secrets.A_SECRET }}
```



Exercise #3: Advanced Workflow

Exercise #3: Create a multi-job workflow and use actions and secrets

Artifacts & Cache



Cache:

- Reuse files such as dependencies to speed up jobs
- “actions/cache”
- “action/setup-<package_manager>”:

Node
Python
Java
Ruby
Go

Path: of files you want to add to the cache
Key: to refer to the cache

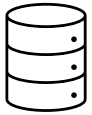
Artifacts:

- Collection of files generated by a job
- Log files, test results, screenshots, binaries, compressed files, reports, etc.
- “actions/upload-artifact”
- “actions/download-artifact”

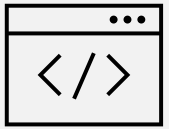
Retention:

- Artifacts: up to 90 days
- Cache: removed if not accessed within 7 days

Cache & Artifacts example

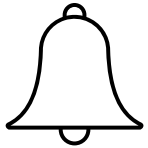


```
...
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      ...
      - name: cache_node_modules
        uses: actions/cache@v3
        with:
          path: my-cache-key
          key: path/to/dependencies
          restore-keys: |
            ${{ runner.os }}-
  ...
```



```
...
jobs:
  buildJob:
    runs-on: ubuntu-latest
    steps:
      ...
      - name: upload build files
        uses: actions/upload-artifact@v3
        with:
          name: my-build-name
          path: path/to/buildfiles
  deployJob:
    steps:
      - name: download build files
        uses: actions/download-artifact@v3
        with:
          name: my-build-name
```

Troubleshooting & Notifications



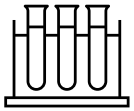
Troubleshooting:

- Reviewing activity logs
- Reviewing visualization graph
- Reviewing workflow history
- “ACTIONS_STEP_DEBUG”: “true” to enable debug logging
- “ACTIONS_RUNNER_DEBUG”: “true” to enable runner job execution diagnostic log

Notifications:

- Email/message
- GitHub notifications inbox/GitHub mobile app
- Subscriptions:
  - Issue
  - Pull request
  - Gist
  - Repository and workflow activity

Continuous Integration: Testing



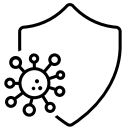
Testing:

- Increase quality and security
- “Stopped/Static” testing:

Static application security testing
Source code analysis
Mostly “white box” testing

Bug:

- Coding error; unreliable/broken code



Vulnerability:

- Prone to being hacked or attacked

Code smell:

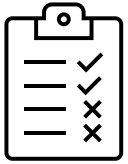
- Complex/repeated/dead/bloated code
- Results in technical debt

Unit Testing & Code Coverage

Unit testing:

- Validate code works as expected
- Test smallest piece of code independent/isolated from rest of codebase
- Tools:

Python: Pytest, unittest
Javascript: Mocha, Jasmine
Java: jUnit

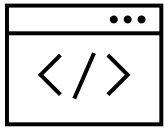


Code coverage:

- Monitors execution of test suite
- Report on what % of code has been tested
- Tools:

Python: Coverage.py
Javascript: instanbul
Java: Cobertura, JaCoCo

GitHub Code Scanning & Secret Scanning



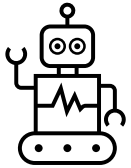
Code scanning:

- GitHub feature to analyze code
- Coding errors & security vulnerabilities
- Setup with CodeQL analysis workflow
- Scan on push, pull request or schedule
- Code scanning pull request checks
- View alerts in Security > Code scanning

Secret scanning:

- Scans code for patterns that match secrets
- Scans entire git history on all branches
- View alerts in Security > Secret scanning alerts

GitHub Dependabot & Sonarcloud



Dependabot:

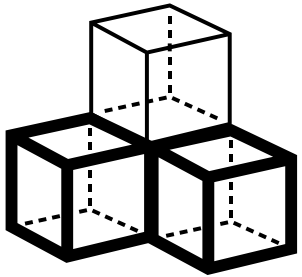
- Helps keep dependencies up to date
- Monitors vulnerabilities in dependencies
- Can create Pull request with solution
- Alerts can be viewed in Security > Dependabot alerts

sonarcloud

Sonarcloud:

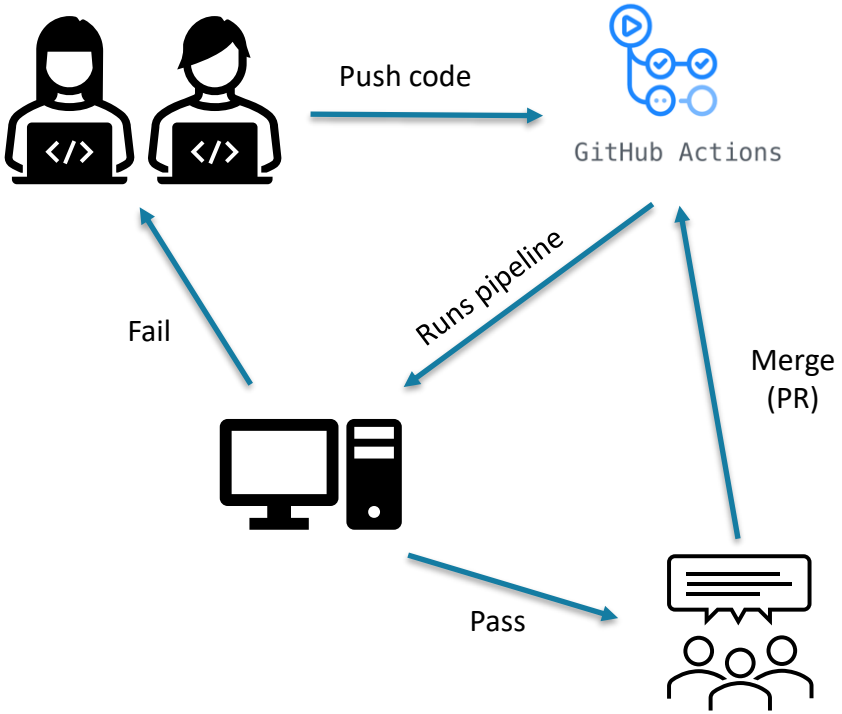
- Leading static code analysis service
- Identify:
  - code smells
  - bugs
  - vulnerabilities
- Integrates with DevOps platforms:
  - GitHub
  - Azure DevOps
  - GitLab
  - Bitbucket

Continuous Integration: Recap



Continuous Integration Phase:

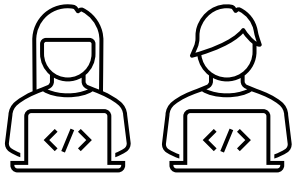
- Commit early, commit often
- Identify bugs or issues
- Developers submit their code
- If their code passes testing and reviewers approve, it can be merged



Exercise #4: Building & testing

Exercise #4: Building & testing

Environments

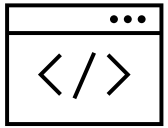


Dev:

- Replica of the production environment
- Same server configuration & software
- Mock data
- Useful for testing code
- Environment local in dev's computer as well as in dev servers to include all code that was merged to the development branch

Test:

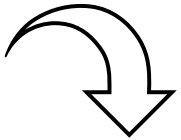
- Replica of the production environment
- Includes new features ready to be released into production
- Used for manual & automated testing

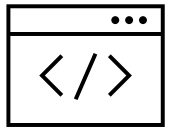


Prod:

- Environment running the application that your end users are using

GitHub deployment environments



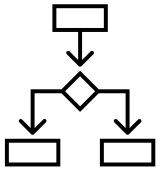


Deployment environments:

- Created in settings > environments
- Each job can reference a single environment
- Each environment can have its own environment variables
- Protection rules:
  - required reviewers
  - wait timer
  - deployment branches

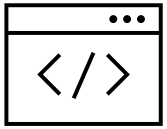
```
...
jobs:
  test_deployment_job:
    runs-on: ubuntu-latest
    environment: test
    steps:
      - name: deploySomewhere
        env:
          SOME_SECRET: ${{ secrets.A_SECRET }}
...
```

GitHub conditions



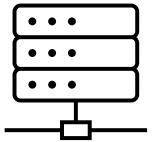
Conditions:

- Prevent job/step execution
- Create conditions with secrets, predefined variables, status checks and more.
- Include operators to expand on the condition (!, &&, !=, ||, ==, and more..)



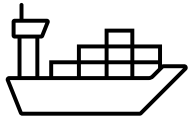
```
...
jobs:
  job1:
    ...
  job2:
    if:  always()
    needs: job1
    steps:
      - name: someStep
        if: github.ref == 'refs/heads/main' &&
github.event_name == ‘pull_request’
...
```

Deployment targets



Servers:

- Physical
- Virtual



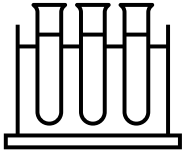
Containers:

- Container instance
- Kubernetes

Others:

- App services
- Functions as a service

Dynamic testing



Testing:

- Confirm expectations are met
- “Running/Dynamic” testing:

Smoke testing
End to end testing
Performance testing
Mostly “black box” testing
Dynamic application security testing

Automated testing:

- Ensure application works as intended
- Faster test execution
- Reliable, consistent and accurate results
- Selenium, testcafe, cypress, appium

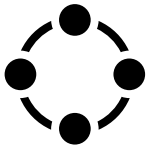
Manual testing:

- Tests carried out by designated testers, developers & users to gather feedback

Deployment Strategies

Big Bang Deployment:

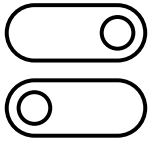
- Updates whole application in one go
- Oldest form of deploying software
- Type of deployment where you provide customer with installation media
- Often requires downtime
- Rolling back requires reinstalling



Rolling Deployment:

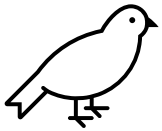
- Group of servers/containers
- Install new version in one group and continue with the next groups
- No downtime
- Key concept: 2 software versions coexisting

Deployment Strategies



Blue-Green:

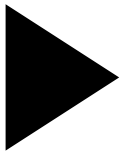
- Also known as Red-Black or A-B
- Two separate environments
- Previous environment is kept live but disconnected in case we need to failback



Canary:

- Similar to Blue-Green + Rolling
- % of servers/containers running previous version + smaller % of servers running new version
- Specify specific ip range/region for new version
- "Real production" test without affecting production as a whole

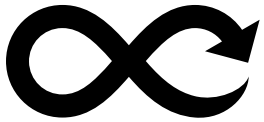
Continuous Delivery & Deployment



Continuous Delivery:

- Manually push a button to run the last stage

- Considered as a way to safely deploy into production



Continuous Deployment:

- Automatically deploys the software

- Tests and stages in CI/CD pipeline must pass

- If a stage fails the whole pipeline fails

Exercise #5: Deploy

Exercise #5: Deploy

Thank you!

Social Media:

https://www.linkedin.com/in/peafl

Twitter: @DeusPaul

Pulse Check Question #1

Question:
What Operating System are you using?

Options:
Linux
macOS
Windows
Other

Pulse Check Question #2

Question:
How much experience do you have with git?

Options:
What is git?
Basic
Intermediate
High

Pulse Check Question #3

Question:
Are you familiar with YAML?

Options:
No
Yes
I prefer JSON

Pulse Check Question #4

Question:
Which programming language do you use the most?

Options:
C++
Java
Javascript
Go
PHP
Python
Ruby
Rust
Typescript
Other