B0 - HUB

B-INN-000

GitHub Actions

Because GitHub is always up!

1.0

Foreword

Before starting the workshop, make sure you have followed all the steps described below.

Create a GitHub account

If you already have a GitHub account, go to the next step.

- Go to `https://github.com/join` and follow the steps provided by the form.

Fork the model repository

- Go to `https://github.com/CyberryTeam/epitech-workshop-github-actions`
- Click on the **Fork** button
- Select your GitHub account

Add collaborators

To verify your work, we will need access to your depot.

From your repository:

- Go to the Settings tab
- Click on the Manage access button
- Click on the Invite a collaborator button.
- Add the following people:
    - Madfish5415
    - MrSquaare

You are not judged on your success but on your investment.

Home

DevOps is a set of practices combining software development and information technology operations that aims to shorten the systems development lifecycle and provide tools to ensure software quality and deployment.The most recurring terms in this area are those of "continuous integration" and "continuous deployment".

Continuous integration

A set of practices consisting of verifying with each source code modification that the result of the modifications does not produce a regression in the developed application. The main purpose of this practice is to detect integration problems as early as possible in the development process. In addition, it makes it possible to automate the execution of test suites and to see the progress of software development.

Continuous deployment

A set of practices consisting of frequent delivery through automated deployments.

The goal of this workshop is to introduce you to these two sets of practices in the context of your Epitech projects. To do so, we will use GitHub, and particularly its services GitHub Actions and GitHub Marketplace.

Make sure you have followed the steps in the Foreword

Hello, World!

You **MUST** name your file: ex00.yml

Getting started

From your repository:

- Go to the Actions tab.
- A list of pre-configured workflows is displayed:
    - If you are an adventurer, click on the button **Set up a workflow yourself** button.
    - If you are an explorer, click on the **Set up this workflow** button of the **Simple workflow** card.

The editor provides auto-completion and syntax analysis.

- Click on the **Start commit** button and the **Commit a new file** button when everything is ready.
- Go back to the **Actions** tab.
- Click on the name of the commit you sent previously.
- Watch the output of your job.

Goal

Launch a workflow that displays `Hello, World!` when someone stars the repository.

Help

```
on:
  watch:
    types: [started] # Star event (no there's no mistake, yes it's strange.)

jobs:
  # ...
```

Useful references

- GitHub Actions documentation
    - Create a workflow file

CONFIGURE YOUR WORKFLOW

You **MUST** name your file: ex01.yml

GitHub Actions provides several features to define your workflow:

- Target one or more events
- Target one or more types of event
- Target at one or more branches
- Target (or ignore) one or more files (or folders)
- Planning routines

In fact, the following workflow:

```
on:
  push:
    branches:
      - master
    paths:
      - 'src/**'
  pull_request:
    branches:
      - master
      - develop
    types:
      - opened
      - edited
  schedule:
    - cron: '* * /1 * * '
```

will have the effect of triggering the workflow when:

- Any modification in the src folder has been pushed to the master branch.
- Any Pull Request on master or develop has been opened or edited.
- A day has passed.

Goal

Launch a workflow that displays `Hello, World! Today is {date}` where `{date}` is in the following format: `{month}-{day}-{year} {hour}:{minute}:{second}`.
The workflow will be triggered under the following conditions:

- When someone stars the repository
- When an issue is opened, edited or closed
- When a pull request is ready to be reviewed
- When a Wiki page has been created, edited or deleted
- Every beginning of the month

Example

```
Terminal
Hello, World! Today is 01-01-2020 23:42:00
```

Useful resources

- GitHub Actions documentation
    - on
- Ubuntu documentation
    - date

Choose your runner

You **MUST** name your file: ex02.yml

GitHub Actions also offers several ways to test your projects on multiple operating systems simultaneously:

- macOS
- Ubuntu
- Windows

Goal

Create a workflow that displays the version of each of these virtual environments when someone stars the repository:

- `macos-latest`
- `ubuntu-latest`
- `windows-latest`

Example

macOS

```
~/B-INN-000> 10-15.1
```

Ubuntu

```
~/B-INN-000> 18.04
```

Windows

```
C:\B-INN-000> 10.0.17763
```

Useful resources

- GitHub Actions documentation
    - runs-on
- macOS documentation
    - sw_vers
- Ubuntu documentation
    - lsb_release
- Windows documentation
    - Get-ComputerInfo
    - Select-Object

Go further (optional)

Do the same exercise, but using matrices.

USER CONTAINERS

You **MUST** name your file: ex03.yml

Although GitHub Actions offers ready-to-use operating systems, the list is limited. To counteract this problem, it is possible to use Docker containers.

GOAL

Create a workflow that displays the version of each of these Docker images when someone stars the repository:

- `alpine`
- `archlinux`
- `fedora`

EXAMPLE

Alpine

```
~/B-INN-000> 3.11.5
```

CentOS

```
~/B-INN-000> 8
```

Fedora

```
~/B-INN-000> 31
```

Useful references

- GitHub Actions documentation
    - container
- Manual pages documentation
    - os-release

Go further (optional)

Do the same exercise, but using matrices.

Use services

You **MUST** name your file: ex04.yml

GitHub Actions also allows the deployment of services, themselves deployed as a Docker container. This allows you, for example, to deploy a database needed to test your application.

Goal

Deploy a nginx service with exposed port `4242`. Send a `GET` request with `curl` or `wget` and `grep` to retrieve only the response code when someone stars the repository.

Example

```
HTTP/1.1 200 OK
```

Useful references

- GitHub Actions documentation
    - services
- Manual pages documentation
    - curl
    - grep
    - wget

USE VARIABLES

You **MUST** name your file: ex05.yml

GitHub Actions offers two ways of variables:

- Environment variables, declared in the workflow
- Secrets, declared in the repository settings

The advantage of secrets is that they allow the storage of sensitive data (for example: an SSH private key).

GOAL

Deploy a nginx service (with the following Docker image: `mrsquaare/nginx-htpasswd`) with exposed port `4242`. Send a GET request with `curl` or `wget` and `grep` to retrieve only the response code when someone stars the repository. It will requires a username and password to login. The username will be stored in the `NGINX_USER` environment variable and its value will be `john`. The password will be stored in the secret `NGINX_PASSWORD` and its value will be `doe`.

EXAMPLE

Terminal

```
HTTP/1.1 200 OK
```

USEFUL REFERENCES

- GitHub Actions documentation
  - services
  - Environment variables
  - Secrets
- Manual pages documentation
  - curl
  - grep
  - wget

Use actions

You **MUST** name your file: ex06.yml

The great thing about GitHub Actions is the ability to use out-of-the-box automation scripts called... action. In addition to the actions offered by GitHub, you can also use the thousands of actions created by the community.

Goal

Use the action `actions/checkout` to retrieve the repository, checkout to the `sort` branch and display the list of files sorted by ASCII order of the `files` directory.

`files` is a git submodule

Example

Terminal

```
this-is-a-file
thisafile
```

Useful references

- GitHub marketplace
  - actions/checkout
- Manual pages documentation
  - ls
  - sort

Test automatically a project

You **MUST** name your file: ex07.yml

It is now time to implement continuous integration for our project.

Goal

Retrieve the repository, checkout to the `project` branch and check the following steps when a pull request is made on any branch:

- Project compiles
- Functional tests are valid
- Unit tests are valid
- Epitech coding style is met

These tests will have to be performed using the Epitech container.

Useful references

- GitHub marketplace
  - actions/checkout
- Docker Hub
  - cyberryteam/normez-docker
  - cyberryteam/epitech-docker
  - epitestcontent/epitech-docker

Deploy automatically a project

You **MUST** name your file: ex08.yml

Now that we have our continuous integration in place, we're going to put in place continuous deployment for our project.

Prerequisites

- Create a repository on Epitech's git server.

Goal

Retrieve the repository, go to the `project` branch, add your private SSH key in the runner and push to the Epitech repository when a push is made on the master branch.

Private SSH key is a very sensitive data.
Don't trust community actions and use the commit hash to specify the version you want to use. See more information

Useful references

- GitHub marketplace
  - actions/checkout
  - mrsquaare/ssh-setup-action
- Git documentation
  - push

Go further

Congratulations to you, you've finished this workshop!

If you want to learn more about how to work on GitHub, here are some links:

- Managing your projects with GitHub
- Discover GitHub in its entirety via interactive tutorials

If you would like to get involved, here are a few suggestions:

- Create your own action
- Automate via GitHub Apps
- Create your own GitHub Apps