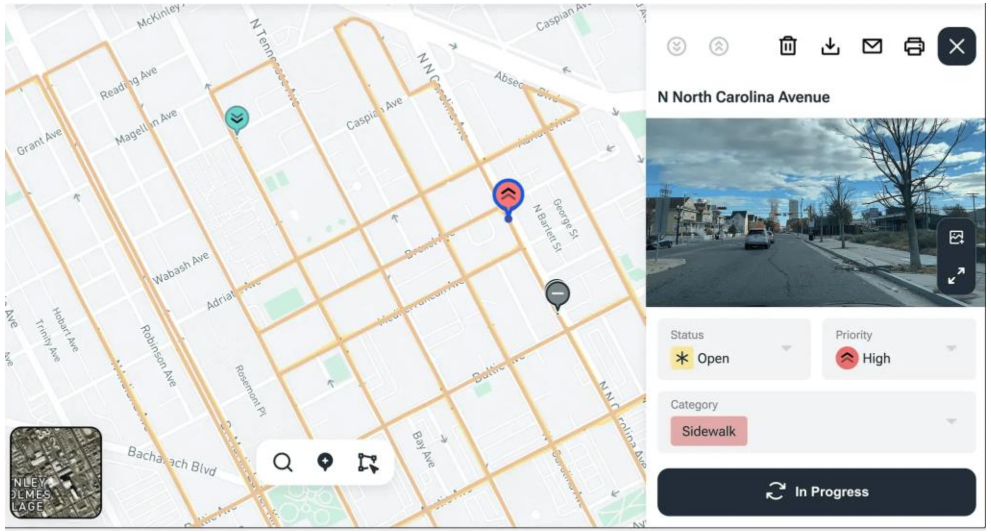
جميع الشخصيات الآتية من خيال المؤلف
و أيِ تشابه بينها و بين الواقع هو محض الصدفة

Nehal

RoadBotics™



N North Carolina Avenue

Status
Open

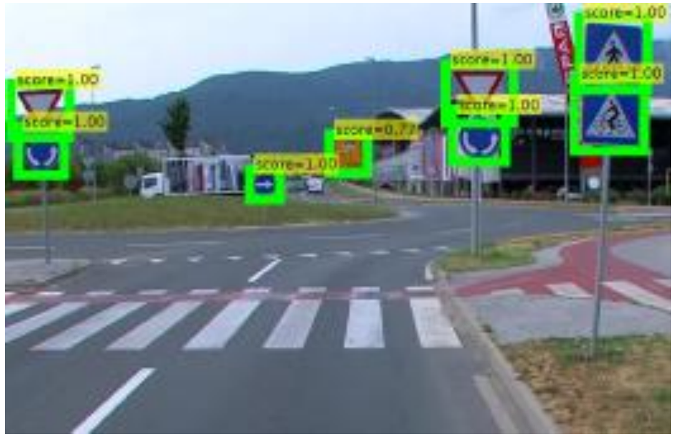
Priority
High

Category
Sidewalk

In Progress

Toqa

vialytics



Nehal

RoadBotics™

Toqa

vialytics

ultralytics YOLOv8

Edit the bounding box thickness

2 min

20 min

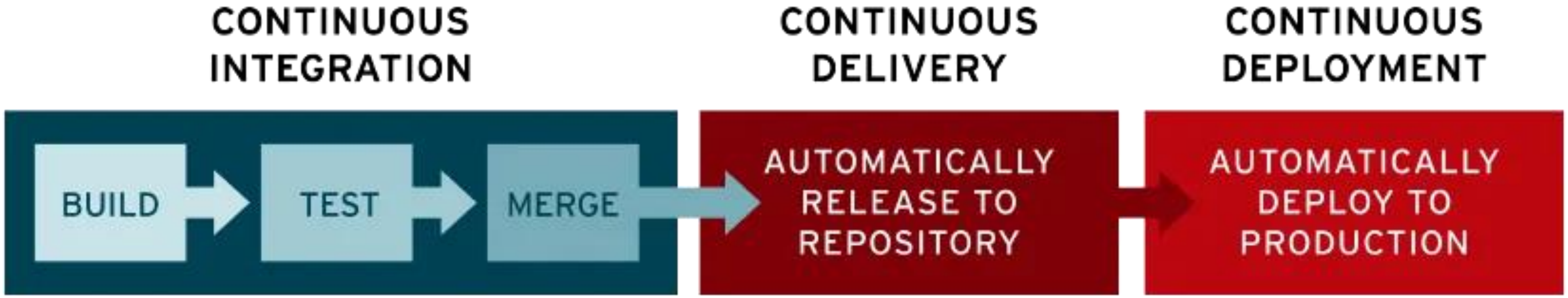
CI/CD

Agenda

- What’s CI/CD
- CI/CD Tools
    - GitHub Actions

CI/CD

CI/CD, which stands for **Continuous Integration** and **Continuous Delivery/Deployment**, is the practice of **automating** the **integration** of code changes from multiple developers into a single codebase. It aims to streamline and accelerate the software development lifecycle.



CI/CD – Continuous Integration (CI)

- **Continuous Integration (CI)** is the practice of **automatically and frequently integrating code changes** into a shared repository. Its main goals are to:
  - detect bugs early
  - simplify collaboration among developers
  - improve software quality
  - accelerate feature delivery

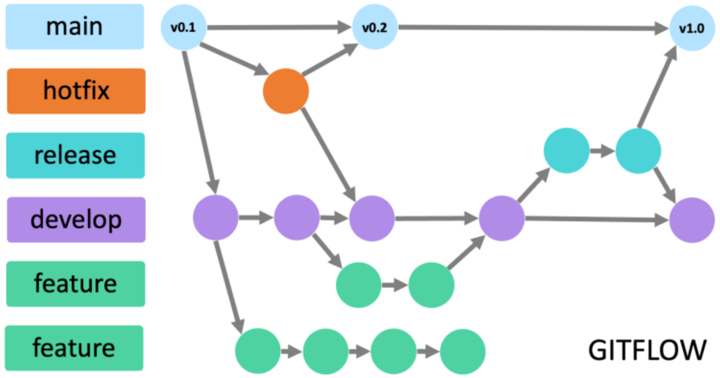
- A typical CI pipeline is **triggered by every code commit or merge**. It automatically builds the application, stores artifacts, runs **unit and integration tests**, and can perform code analysis using tools like Sonar.
  This helps validate changes quickly and ensures they don’t break existing functionality. Automated testing allows teams to catch and fix issues early, making development faster and more reliable.

CI/CD – Continuous Delivery (CD)

- **Continuous delivery** helps developers **test their code in a production-similar environment**, hence preventing any last-moment or post-production surprises. These tests may include UI testing, load testing, integration testing, etc. It helps developers discover and resolve bugs pre-emptively.

- By automating the software release process, CD contributes to low-risk releases, lower costs, better software quality, improved productivity levels, and most importantly, **it helps us deliver updates to customers faster and more frequently**. If Continuous Delivery is implemented properly, we will always have a deployment-ready code that has passed through a standardized test process.

- Some key aspects of Continuous Delivery:
  - ○ **Automated** release process that can deploy code to production environments
  - ○ **Shortens** the release cycle and provides **faster feedback**
  - ○ Allows for more **incremental updates** rather than big bang releases
  - ○ Releases still **require manual approval** before going live



CI/CD – Continuous Deployment (CD)

- **Continuous Deployment** is the final stage in the CI/CD pipeline and goes one step further than Continuous Delivery. It automates the entire process of releasing validated code changes directly to production, **without any manual approval**. Once code passes all automated tests, it is immediately deployed to the live environment. This requires a high level of confidence in the test coverage and the stability of the deployment process.
- In simple terms, Continuous Deployment means that a developer's change — once committed and tested — can go live within minutes, typically through a cloud-based infrastructure. It enables rapid feedback, faster innovation, and a truly seamless release process.

- **Key aspects of Continuous Deployment:**
  - Fully automated pipeline from code commit to production
  - Code is automatically tested and deployed if it passes all checks
  - Supports rapid and frequent release cycles
  - Relies on robust test automation and continuous monitoring
  - Ideal for web services and cloud-based applications
  - May not be suitable for all systems, especially those requiring regulatory approval or manual QA steps

CI/CD Tools

GitHub Actions

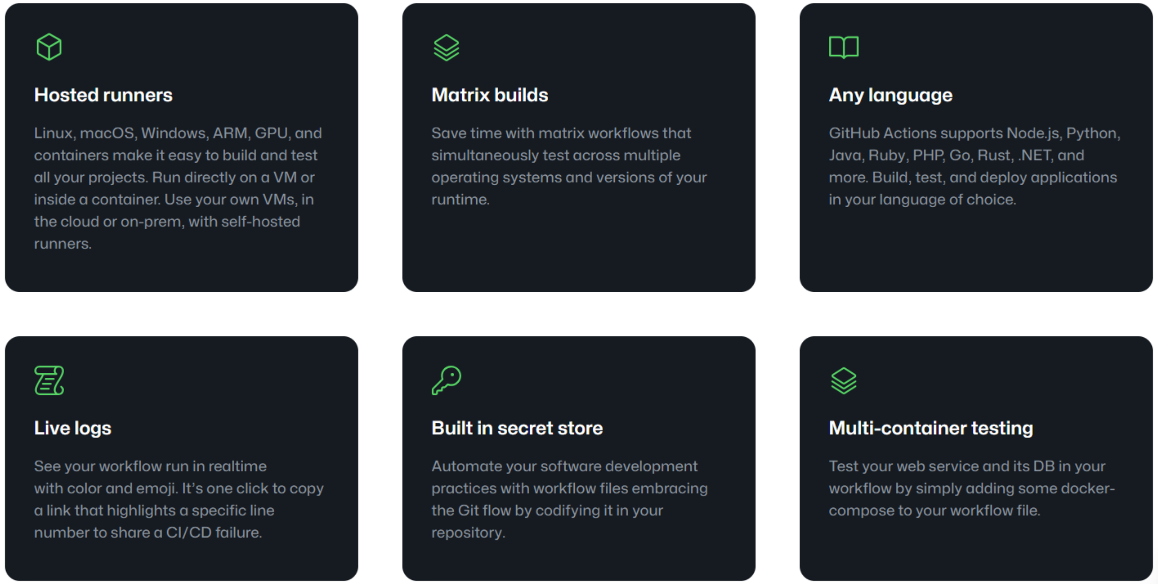
Azure Pipelines

Jenkins

GitHub Actions

CI/CD – GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.



Hosted runners

Linux, macOS, Windows, ARM, GPU, and containers make it easy to build and test all your projects. Run directly on a VM or inside a container. Use your own VMs, in the cloud or on-prem, with self-hosted runners.

Matrix builds

Save time with matrix workflows that simultaneously test across multiple operating systems and versions of your runtime.

Any language

GitHub Actions supports Node.js, Python, Java, Ruby, PHP, Go, Rust, .NET, and more. Build, test, and deploy applications in your language of choice.

Live logs

See your workflow run in realtime with color and emoji. It's one click to copy a link that highlights a specific line number to share a CI/CD failure.

Built in secret store

Automate your software development practices with workflow files embracing the Git flow by codifying it in your repository.

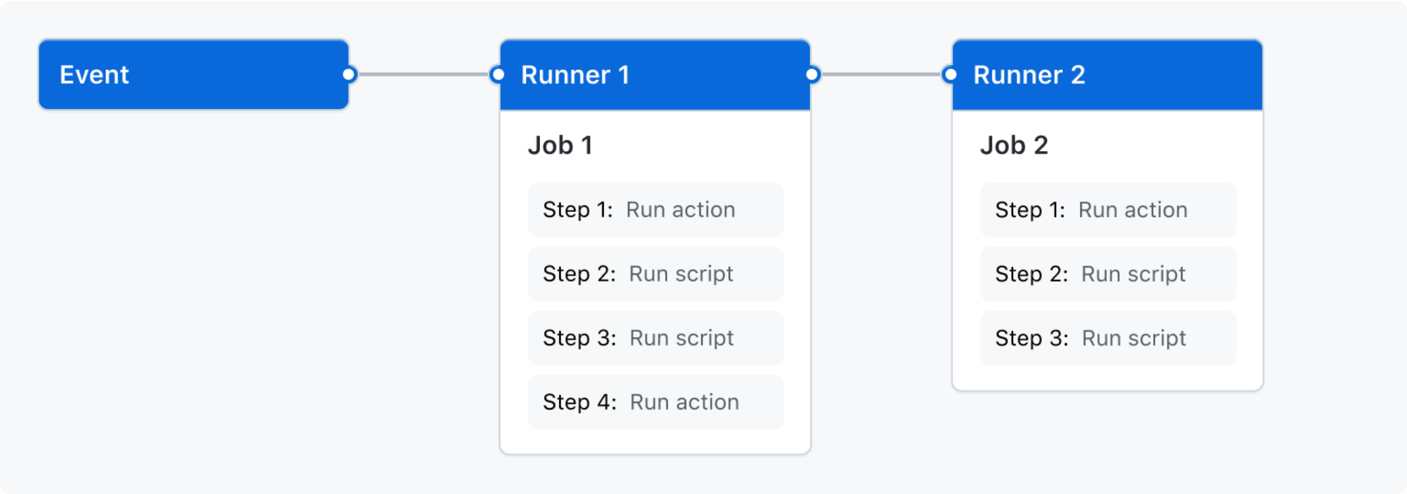
Multi-container testing

Test your web service and its DB in your workflow by simply adding some docker-compose to your workflow file.

CI/CD – GitHub Actions

The components of GitHub Actions: [workflow, event, job, runner, steps, action]

- You can configure a GitHub Actions **workflow** to be triggered when an **event** occurs in your repository, such as a pull request being opened or an issue being created, or they can be triggered manually, or at a defined schedule.
- Your workflow contains one or more **jobs** which can run in sequential order or in parallel.
- Each job will run inside its own virtual machine **runner**, or inside a container, and has one or more **steps** that either run a script that you define or run an **action**, which is a reusable extension that can simplify your workflow.



CI/CD – GitHub Actions

Workflow:

- is a configurable automated process that will run one or more jobs. Workflows are defined by a **YAML** file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

Workflows are defined in the .github/workflows directory in a repository. A repository can have multiple workflows, each of which can perform a different set of tasks such as:

  - Building and testing pull requests
  - Deploying your application every time a release is created
  - Adding a label whenever a new issue is opened

CI/CD – GitHub Actions

Events

- is a specific activity in a repository that triggers a **workflow** run. For example, an activity can originate from GitHub when someone creates a pull request, opens an issue, or pushes a commit to a repository. You can also trigger a workflow to run on a schedule, by posting to a REST API, or manually.
- For a complete list of events that can be used to trigger workflows, see Events that trigger workflows.

CI/CD – GitHub Actions

Runners

- A **runner** is a server that runs your workflows when they're triggered. Each runner can run a single **job** at a time. GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners to run your **workflows**. Each workflow run executes in a fresh, newly-provisioned virtual machine.
- If you need a different operating system or require a specific hardware configuration, you can host your own runners.

CI/CD – GitHub Actions

Jobs

- A set of **steps** in a workflow that is executed on the same **runner**.

- Each step is either a shell script that will be executed, or an **action** that will be run. Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another. For example, you can have a step that builds your application followed by a step that tests the application that was built.

- You can configure a job's dependencies with other jobs; by default, jobs have no dependencies and run in parallel. When a job takes a dependency on another job, it waits for the dependent job to complete before running.
  - For example, you might configure multiple build jobs for different architectures without any job dependencies and a packaging job that depends on those builds. The build jobs run in parallel, and once they complete successfully, the packaging job runs.
  - You can use the **needs** keyword to create this dependency. If one of the jobs fails, all dependent jobs are skipped; however, if you need the jobs to continue, you can define this using the if conditional statement.

CI/CD – GitHub Actions

Actions

- An **action** is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. Use an action to help reduce the amount of repetitive code that you write in your **workflow** files. An action can pull your Git repository from GitHub, set up the correct toolchain for your build environment, or set up the authentication to your cloud provider.

- You can write your own actions, or you can find actions to use in your workflows in the GitHub Marketplace.

CI/CD – GitHub Actions

About YAML syntax for workflows

- **Using multiple events**
  - You can specify a single event or multiple events. For example, a workflow with the following on value will run when a push is made to any branch in the repository or when someone forks the repository:
    on: [push, fork]
    - If you specify multiple events, only one of those events needs to occur to trigger your workflow. If multiple triggering events for your workflow occur at the same time, multiple workflow runs will be triggered.
- **Using activity types**
  - Some events have activity types that give you more control over when your workflow should run.
    Use on.<event_name>.types to define the type of event activity that will trigger a workflow run.
  - For example, the issue_comment event has the created, edited, and deleted activity types. If your workflow triggers on the label event, it will run whenever a label is created, edited, or deleted. If you specify the created activity type for the label event, your workflow will run when a label is created but not when a label is edited or deleted.
    on: label: types: - created

CI/CD – GitHub Actions

About YAML syntax for workflows

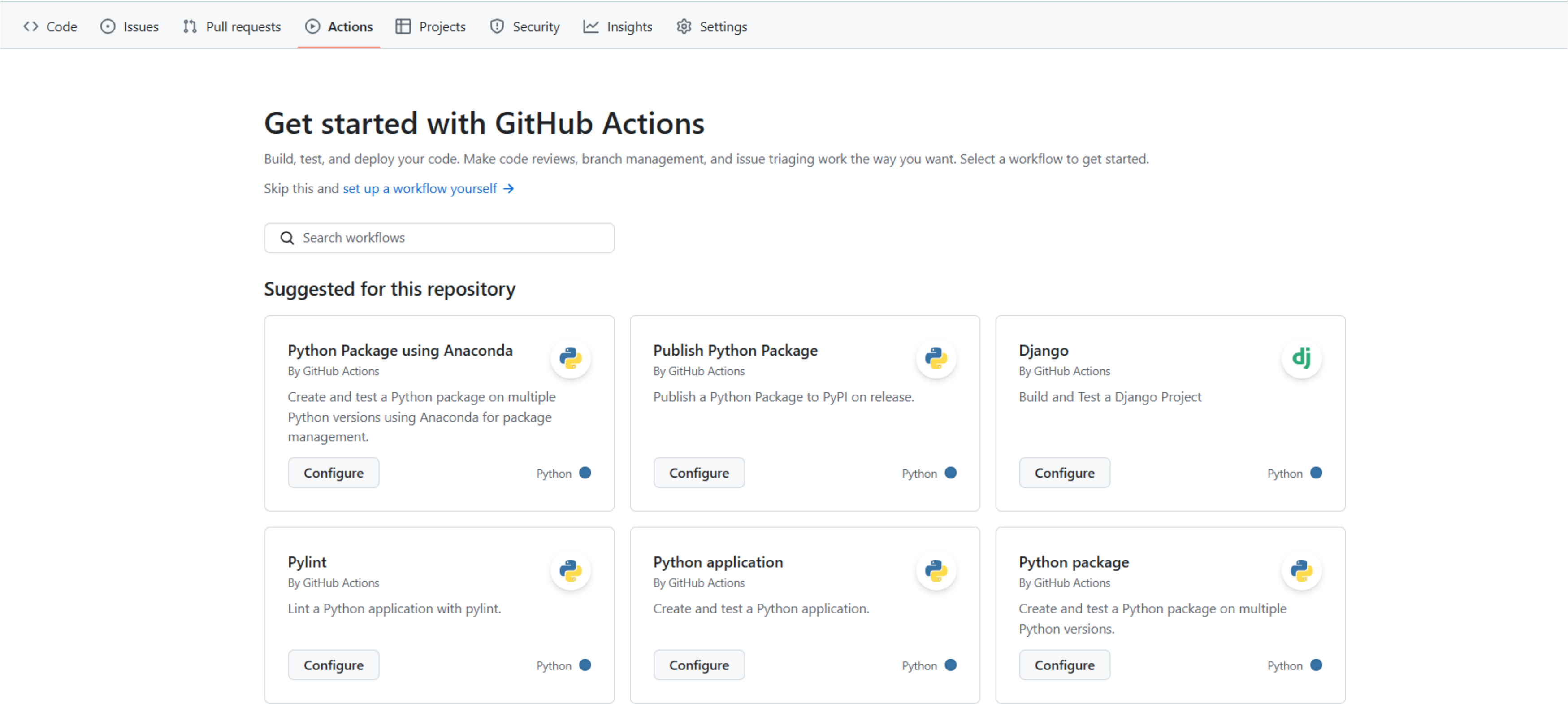
- **Using filters**
  - Some events have filters that give you more control over when your workflow should run.
  - For example, the push event has a branches filter that causes your workflow to run only when a push to a branch that matches the branches filter occurs, instead of when any push occurs.

    ```
    on: push: branches: - main - 'releases/**'
    ```

Let’s Get Started

CI/CD – GitHub Actions

The main tab is Actions to monitor all your workflows, and you can use one of the provided templates suggested by GitHub.



Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

Suggested for this repository

Python Package using Anaconda
By GitHub Actions
Create and test a Python package on multiple Python versions using Anaconda for package management.

Publish Python Package
By GitHub Actions
Publish a Python Package to PyPI on release.

Django
By GitHub Actions
Build and Test a Django Project

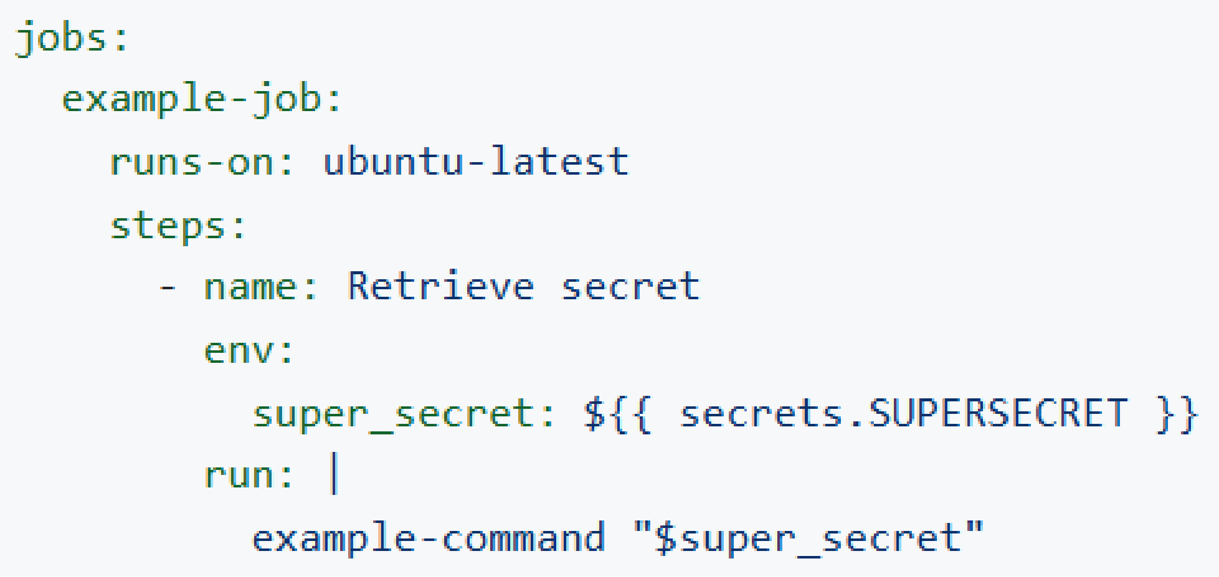
Pylint
By GitHub Actions
Lint a Python application with pylint.

Python application
By GitHub Actions
Create and test a Python application.

Python package
By GitHub Actions
Create and test a Python package on multiple Python versions.

CI/CD – GitHub Actions

If your workflows use sensitive data, such as passwords or certificates, you can save these in GitHub as *secrets* and then use them in your workflows as environment variables. This means that you will be able to create and share workflows without having to embed sensitive values directly in the workflow's YAML source.

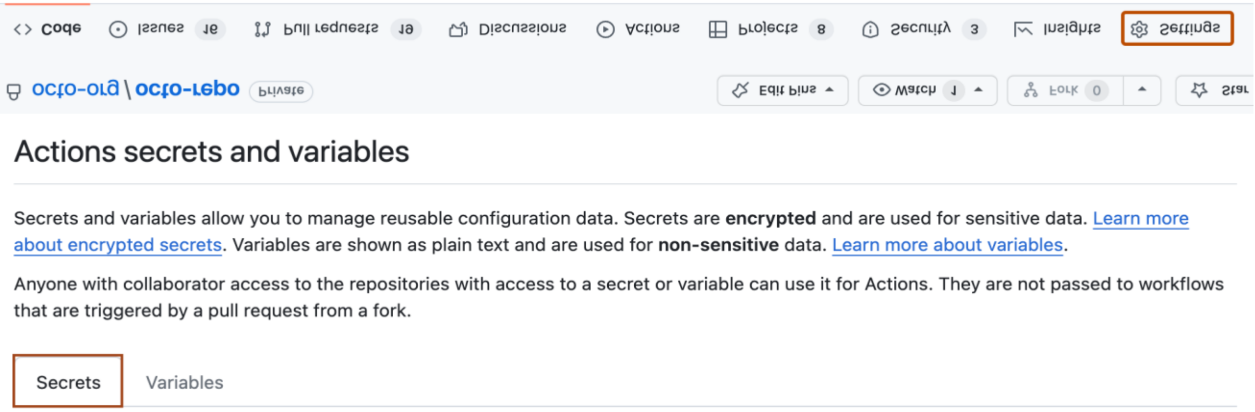


```yaml
jobs:
  example-job:
    runs-on: ubuntu-latest
    steps:
      - name: Retrieve secret
        env:
          super_secret: ${{ secrets.SUPERSECRET }}
        run: |
          example-command "$super_secret"
```

CI/CD – GitHub Actions

To add a secret:

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Settings**. If you cannot see the "Settings" tab, select the dropdown menu, then click **Settings**.
3. In the "Security" section of the sidebar, select **Secrets and variables**, then click **Actions**.
4. Click the **Secrets** tab.
5. Click **New repository secret**.
6. In the **Name** field, type a name for your secret.
7. In the **Secret** field, enter the value for your secret.
8. Click **Add secret**.



Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for **non-sensitive** data. Learn more about variables.

Anyone with collaborator access to the repositories with access to a secret or variable can use it for Actions. They are not passed to workflows that are triggered by a pull request from a fork.

Example

TAKE A BREAK!

Mind Sprint

LAB

CI/CD – Lab

- What you’ll do is to deploy the churn model you containerized in the previous lab:
    - Add the workflow YAML file (github/workflows/actions.yml)
        - Add a test stage
        - Add docker image building and pushing to AWS ECR or Docker Hub
        - Optional: Deploy to a t2.micro EC2 instance
    - If you can’t use AWS, you can just deploy manually on claw cloud

Additional Resources

- CI/CD
- CI/CD ML Project