**GitHub Actions**

**Table of Contents**

[**Introduction of CI/CD** 3](#_Toc111632795)

[**GitHub Actions** 3](#_Toc111632796)

[**Workflow of GitHub Actions** 4](#_Toc111632797)

[**Integrating test cases in the CI Flow** 5](#_Toc111632798)

[**Architecture** 5](#_Toc111632799)

[**CI Flow** 6](#_Toc111632800)

[**Project Folder Structure** 6](#_Toc111632801)

[**Git Flow Architecture – Branch Level** 8](#_Toc111632802)

[**Git Flow** 9](#_Toc111632803)

[**PyTest** 9](#_Toc111632804)

[**Understanding the output - PyTest** 10](#_Toc111632805)

[**AWS – GitHub Actions Authentication** 11](#_Toc111632806)

[**GitHub Action Workflow Script** 12](#_Toc111632807)

[**GitHub Action Workflow Logs** 14](#_Toc111632808)

[**Pull Request Template** 17](#_Toc111632809)

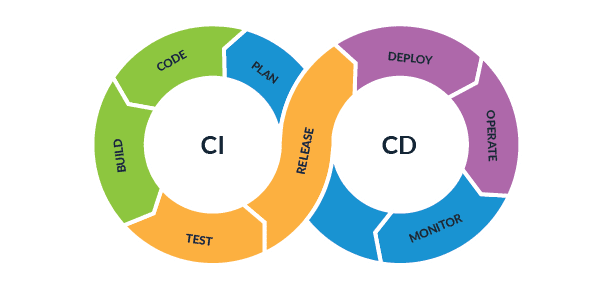
[**Slack Notification** 18](#_Toc111632810)

[**Skipping of Workflow jobs** 19](#_Toc111632811)

# **Introduction of CI/CD**

**What is CI/CD?**

* Strategy to automate the development process to speed up turnaround time.
* With CI/CD, development cycles are more frequent, meaningful and faster.



**Purpose:**

Build a pipeline which integrates code changes from developers into multiple branches and tests against requirements, if prior condition success, then packages the validated code and deploys the new version.

# **GitHub Actions**

* GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform which allows us to automate the build, test, and deployment phases of pipeline.
* We can create workflows to build and test when events happen in the repository.
* Workflow can contain multiple jobs which can be run in sequential order or in parallel.

**Sample flow of CI/CD with GitHub Actions:**

Timeline

Description automatically generated

# **Workflow of GitHub Actions**

Workflow is a configurable automated process that will run one or more jobs.

**Flow:**

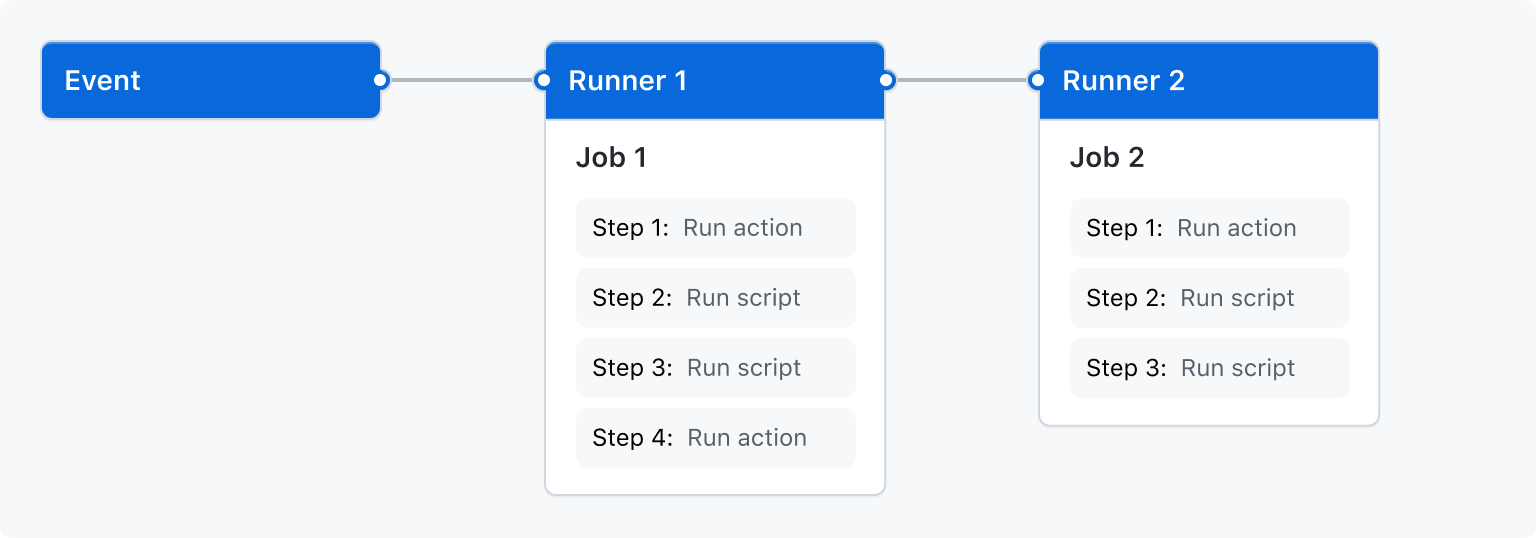
Write the workflows under .github/workflows folder to trigger the CI for any event, ex: Push.

**Create a workflow script:**

Open Git Repository 🡪 Open GitHub Actions 🡪 New Workflow 🡪 Select and update template

**Components of workflow:**

* Events
* Jobs
* Runner



# **Integrating test cases in the CI Flow**

Diagram

Description automatically generated

1. Test cases can be written in test\_\*.py or \*\_test.py file format and pushed into github branch.
2. Once PUSH event is created, it will trigger the Github action workflow which has a series of steps to execute testcases.
3. GitHub Action steps:
   1. Checkout the code of specific branch where PUSH event is created.
   2. Run the pre-requisites to run test cases. For example: check test frame is installed, any necessary dependencies were present, etc.
   3. Execute test cases by using PYTEST framework which by default check for test files in the provided path and starts executing it.
   4. Do any post-checks if needed.

# **Architecture**

**Diagram, engineering drawing

Description automatically generated**

## **CI Flow**

* Individual users (data scientists) create a user branch from the DEV branch and work on it to create a model/make change to an existing model.
* Once changes are made locally, manual testing can be done in the local environment and then code is pushed onto the user branch.
* Once code is pushed, then the user can raise Pull Request (PR) to the Dev branch.
* Upon initiation of the Pull Request (PR), a separate (mandatory) GitHub Actions workflow gets triggered which will run the predefined code build tests on project level. Upon successful execution of the workflow, the model codes need to be tested on the Airflow development environment by running the DAG(s). This DAG(s) will run on a small dataset for testing.
* Upon success of the DAG(s), the PR can be approved manually by the reviewer. Once approved, it can be merged with the Dev branch.
* The model codes again need to be tested on the Airflow Dev environment by running the DAG(s).
* Upon success of the DAG(s), PR can be raised from Dev to Prod branch. Upon initiation of the Pull Request (PR), a GitHub Actions workflow gets triggered which has code changes made by all the users.
* Upon successful execution of the workflow, PR can be merged after approval from the reviewer.

# **Project Folder Structure**

Diagram

Description automatically generated with medium confidence

**requirements.txt:** pip install arguments placed in this file to install with command

“pip install -r requirements.txt”

you can see the precise version of each dependency installed in the virtual environment by running “pip freeze” command.

A picture containing graphical user interface

Description automatically generated

**Note:**

1. requirements.txt file placed in allbirds\_ml\_models/<project\_folder> should have the dependencies related to project.
2. requirements.txt file placed in allbirds\_ml\_models folder should have dependencies related to utils.

**setup.py:**Contains various information related to the package including options and metadata. But it is advised to keep the package and metadata in setup.cfg file and can create setup.py which simply calls setup() method.

Diagram

Description automatically generated

**setup.cfg:** Best approach in terms of code quality and design is to specify the package details in setup.cfg file. Mention the Python version required for the project at python\_requires in options section and place the setup.cfg file in project folder.

**Note:** Setup.cfg file in root directory is the default file and needs to be replaced with project folder setup.cfg file while working in local.

**pyproject.toml:**

* One of the biggest problems with setuptools is that the use of an executable file cannot be executed without knowing its dependencies.
* Pyproject.toml file is supposed to solve  the build-tool dependency problem since pip itself can read the pyproject.toml file.
* To use the pyproject.toml file run “python -m pip install .” command.
* Data scientist can include PyTest framework options in toml file under tool.pytest.ini\_options section. These options will be pickup by pytest framework while executing test cases with pytest command.

Text

Description automatically generated

**Note:** If Data Scientist wants to install packages in editable mode by running “pip install -e .” command, repo must contain setup.py file apart from setup.cfg and pyproject.toml.

# **Git Flow Architecture – Branch Level**

Diagram

Description automatically generated

## **Git Flow**

* Consider V1 is version of project which runs on production and two individual Data scientists working on 2 features i.e., feature-1 and feature-2.
* Individual Data scientists creates feature branches from DEV branch naming feature-1 and feature-2 branch from same commit.
* Data scientist-1 works on feature-1 branch and merges code to DEV branch by raising Pull Request (PR).
* Every day, data scientists need to pull the latest code changes from DEV branch into feature branches.
* So Data scientist-2 pulls the latest code from DEV branch (which also includes commits of feature-1 branch) and works on the feature by making changes in the code.
* Once feature is ready, data scientist-2 merges the feature-2 branch with DEV branch by raising PR.
* Now the Dev branch has changes (feature-1 and feature-2) made by both data scientists.
* Every Friday Release Manager checks the relevant code changes in DEV branch and merges with PROD branch by raising PR.
* Now the new version(V2) of project includes features feature-1 and feature-2.

# **PyTest**

**Installation: “**pip install pytest” command can be used to install the pytest.

Execution of test cases can be done using command “pytest”.

1. Pytest initially looks for a directory i.e “tests”.
2. Then looks for any module starting with test\_\*.py or \*\_test.py.
3. Then runs the functions inside the module which are prefixed with “test” keyword.

**Example:**

Write the python function and save the file in allbirds\_ml\_models/<project\_folder>/src directory.

A screenshot of a computer

Description automatically generated with low confidence

Write the test case in test file with format test\_<file\_name>.py and save it in allbirds\_ml\_models/<project\_folder>/tests directory.

A screenshot of a computer

Description automatically generated with medium confidence

## **Understanding the output - PyTest**

Text

Description automatically generated

Data Scientist can find the config file [pyproject.toml] which was picked up by PyTest framework.

Collected 3 items -> Pytest has found 3 test cases

Representation of tests results in serial way with path:

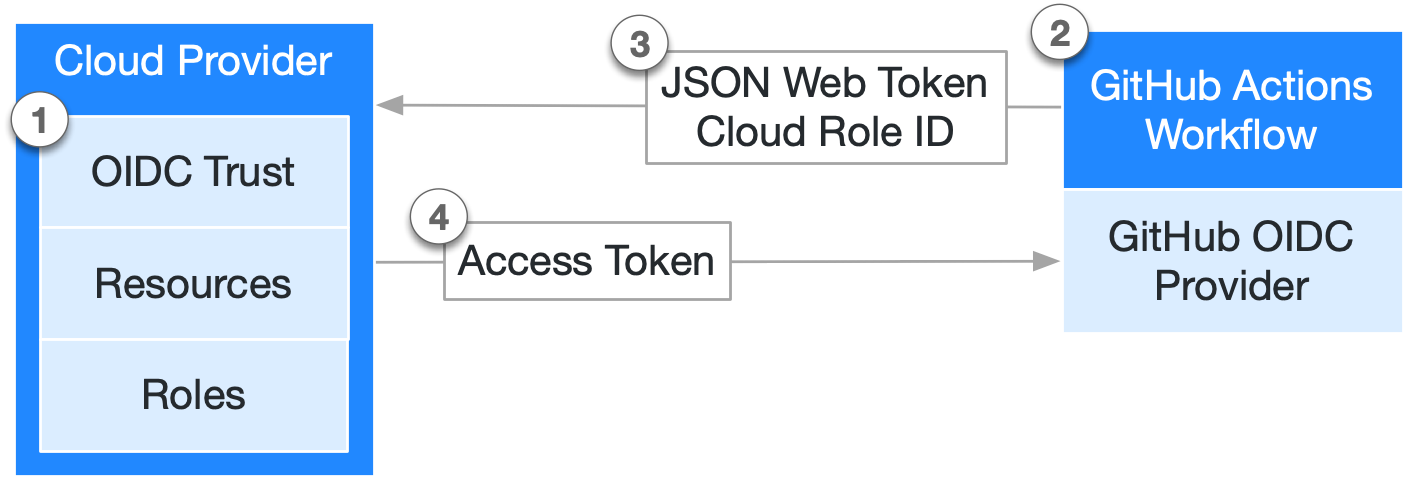
. -> Test case Passed

F -> Test case Failed

S -> Test case Skipped

# **AWS – GitHub Actions Authentication**

* Accessing AWS services from GitHub actions using assume role created with GitHub Identity Provider. OpenID Connect (OIDC) allows GitHub actions workflows to access AWS resources without needing to store AWS Credentials anywhere in GitHub.



* Assume role needs to be mentioned in the GitHub workflow script which will send the JSON web token with cloud Role ID to the cloud provider (AWS), AWS verifies the role and sends back the Access token to the runner machine with which runner machine can access AWS services.
* The token generated in this flow is a temporary and can be used only in a single session. Token gets expired once GitHub workflow job is completed.

Steps to be followed to Configure OpenID Connect (OIDC) in AWS:

1. Add Identity provider to AWS.

AWS console -> IAM -> Identity provider -> Add provider

**Provider type:** OpenID Connect

**Provider URL:** <https://token.actions.githubusercontent.com>

**Audience:** sts.amazonaws.com

1. Create a Role with necessary access (S3 access)

IAM -> Roles -> Create Role

* Select trusted entity
  + **Trusted entity type:** Web Identity (assume role to perform actions)
  + **Identity Provider:** token.actions.githubusercontent.com
  + **Audience:** sts.amazonaws.com
* Add permission:
  + Assign required Permission policies (S3 access)
* Name, review and create role

# **GitHub Action Workflow Script**

Refer [here](https://github.com/allbirds/data-science-c360/blob/feature/cicd/.github/workflows/pullrequest_workflow.yml) for workflow script.

**Explanation:**

**name:** Name of the workflow script

**on**: GitHub event to be mentioned here to auto-trigger Workflow.

**GitHub Event configured in workflow**: pull\_request

Types of pull\_request configured in workflow script:

* **opened**: When pull request is open from feature branch to dev branch.
* **reopened**: When closed pull request is reopened which has base branch as dev.
* **synchronize**: When PR gets updated with new commits, workflow job executes upon the commit.

Text

Description automatically generated

**permissions:** Permissions of GITHUB\_TOKEN needed to be provided for the workflow job for accessing repository and performing actions.

Permissions configured in workflow script:

* **id-token: write 🡪** Write access for the token is required for the authentication between GitHub actions workflow and AWS.
* **contents: read 🡪** Read access is provided for the token to read the content of the repository.

Text

Description automatically generated with low confidence

**runs-on:** Specify OS (Windows | Linux | Mac) upon which series of steps defined in the job executes.

All the workflows run on GitHub-hosted runner machines (managed by GitHub) and workflows runs on **ubuntu-latest** machine.



**jobs:** A workflow script can have multiple jobs defined and each job runs on different runner machine.

Jobs configured in workflow:

1. verify\_projects
2. Execute-project:

**verify\_projects:**

Text

Description automatically generated

* This job clones the repository and finds the projects present in repo by searching for the setup.cfg file defined in project folder.
* Stores all the project names in variable and formats it with JSON format.
* This variable is mapped with output of the workflow job.

**Execute-project:**

* This job utilises matrix strategy to execute the projects in parallel way.
* For each project, a new runner machine is created, clones the repository, install the dependencies and package, and executes the test cases using pytest.

Text

Description automatically generated

**needs:** We need to mention the job name here so that the current job (Execute-project) waits until the job value mentioned here (verify\_projects) is completed its execution. This job is dependent on the previous job.

**strategy:** Workflow job uses the strategy provided here.

* **fail-fast: false 🡪** If one of the parallel job fails, all the jobs stops executing by default.

We need to override this by setting the value as **false** so that if one job fails, other jobs which were running in parallel continues with execution.

* **matrix:** We need to define multiple values here so that for each value, it will create a runner machine and executes the steps defined in this job. We take the JSON output (project names) from the previous job (verify\_projects) and pass it to matrix strategy here to run workflows for each project in parallel.

**AWS Authentication:** Authentication between AWS and GitHub actions is done by using git repo aws-actions/configure-aws-credentials. We need to add few attributes to this step such as role-to-assume (IAM role), role-session-name (session for the logs) and aws-region.

# **GitHub Action Workflow Logs**

* Workflow logs can be found in Actions tab.

Graphical user interface, application

Description automatically generated

* User can click on All workflows in the Workflows section on the left side panel to view the jobs of all workflows.
* User can click on workflow name to view the jobs related only to that workflow.

Graphical user interface, text

Description automatically generated

* Click on any of the workflow job to view the summary of it.

A screenshot of a computer

Description automatically generated with medium confidence

* User can view the jobs present in the workflow on the left panel and execution flow on the right side.
  + In the above job, verify\_projects and Execute-project were 2 jobs executing in sequential way.
  + Execute-project job using matrix strategy and executes 3 jobs in parallel.
    - Execute-project(Exp\_FW)
    - Execute-project(SPR\_LightFM)
    - Execute-project(NPR\_LightFM)
* Each job in workflow executes in a different runner machine.
* Also, User can find the Status, Total Duration of workflow, Billable time in the summary of the workflow job.
* Now click on any of the job name to view the steps executed by that job.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

* User can find the steps on the right-side panel.
* Click on the step to expand the logs and find the details.
* If the Data Scientist wants to re-run the job, Workflow job can be executed again by clicking on the **circle icon** showed beside the job name in the left panel.

A screenshot of a computer

Description automatically generated with medium confidence

* A dialogue box pops-up showing the jobs which will be executed again, click on **Re-run jobs** button to start the workflow job again.

A screenshot of a computer

Description automatically generated with medium confidence

* Data Scientist can download the log of workflow job into local by clicking on **Download log archive** button under settings of job.

A screenshot of a computer

Description automatically generated with medium confidence

# **Pull Request Template**

* Template is created with content which needs to be updated while raising the Pull Request by Data Scientists.
* This template needs to be placed in .github folder in default branch of the repository.
* This will auto-populate the description while raising the PR.

A screenshot of a computer

Description automatically generated with medium confidence

* Data Scientist needs to update the info related to the changes in the PR and create pull request.

# **Slack Notification**

* Upon creation of pull request, slack channel gets notified with the PR info along with redirect URL of pull request, description, checks if any (GitHub actions workflow jobs).

Text

Description automatically generated

* By integration slack channel with github repository, Data Scientists can be updated with the changes happening in the repo, get to know the status of workflow jobs which got executed upon raising PR.

# **Skipping of Workflow jobs**

* Data scientists can skip workflow runs triggered by GitHub events by including a command in commit message.
* Workflows will not be triggered if you add **[skip ci]** string to the commit message in a push, or the HEAD commit of a pull request.
* Example commit message: “updated dependencies of model [skip ci]”