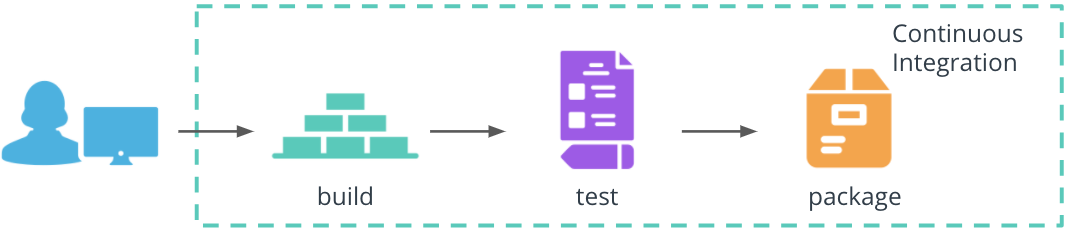
**Summary**

Continuous Integration (CI) is a mechanism that produces the package of an application that can be deployed to consumers. As such, every commit to the main branch is built, tested, and packaged, if it meets the expected behavior. Within the cloud-native, the result of Continuous Integration represents a Docker image or an artifact that is OCI compliant.

Let's explore the commands and tools associated with each Continuous Integration stage!



Continuous Integration stages

**Build**

The build stage compiles the application source code and associated dependencies. We have explored this stage as part of the Dockerfile. For example, the Dockerfile for the Python hello-world application instructed the installation of dependencies from *requirements.txt* file and execution of the *app.py* at the container start.

**Build**

The build stage compiles the application source code and associated dependencies. We have explored this stage as part of the Dockerfile. For example, the Dockerfile for the Python hello-world application instructed the installation of dependencies from *requirements.txt* file and execution of the *app.py* at the container start.

# use pip to install any application dependencies

RUN pip install -r requirements.txt

# execute command on the container start

CMD [ "python", "app.py" ]

**Test**

The technology landscape provides an abundance of frameworks, libraries, and tools to test and validate the behavior of an application. For Python application, some of the well-known frameworks are:

* [pytest](https://docs.pytest.org/en/stable/) - for functional testing of the application
* [pylint](https://pypi.org/project/pylint/) - for static code analysts of the application codebase

**Package**

The result of the package stage is an executable that contains the latest features and their dependencies. With Docker, this stage is implemented using docker build and docker push commands. These create a Docker image using a Dockerfile, and stores the image to a registry, such as DockerHub. For example, to create and store the image for Python hello-world application, the following commands are used:

# build an image using a Dockerfile

docker build -t python-helloworld .

# store and distribute an image using DockerHub

docker push pixelpotato/python-helloworld:v1.0.0

*Note:* Buildbacks does not require a Dockerfile to create an OCI compliant image or artifact.

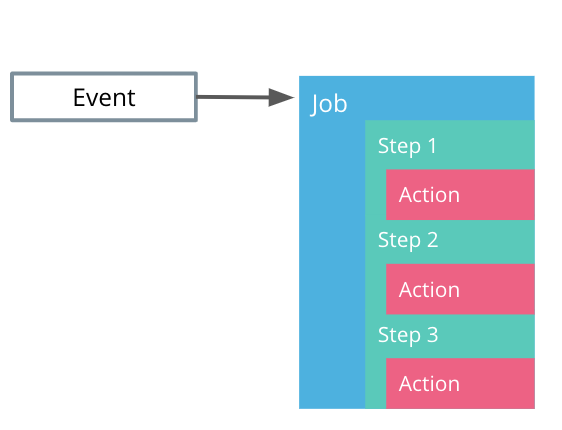
**Github Actions**

**Summary**

There are plenty of tools that automate the Continuous Integration stages, such as Jenkins, CircleCI, Concourse, and Spinnaker. However, in this lesson, we will explore GitHub Actions to build, test, and package an application.

GitHub Actions are event-driven workflows that can be executed when a new commit is available, on external or scheduled events. These can be easily integrated within any repository and provide immediate feedback if a new commit passes the quality check. Additionally, GitHub Actions are supported for multiple programming languages and can offer tailored notifications (e.g. in Slack) and status badges for a project.

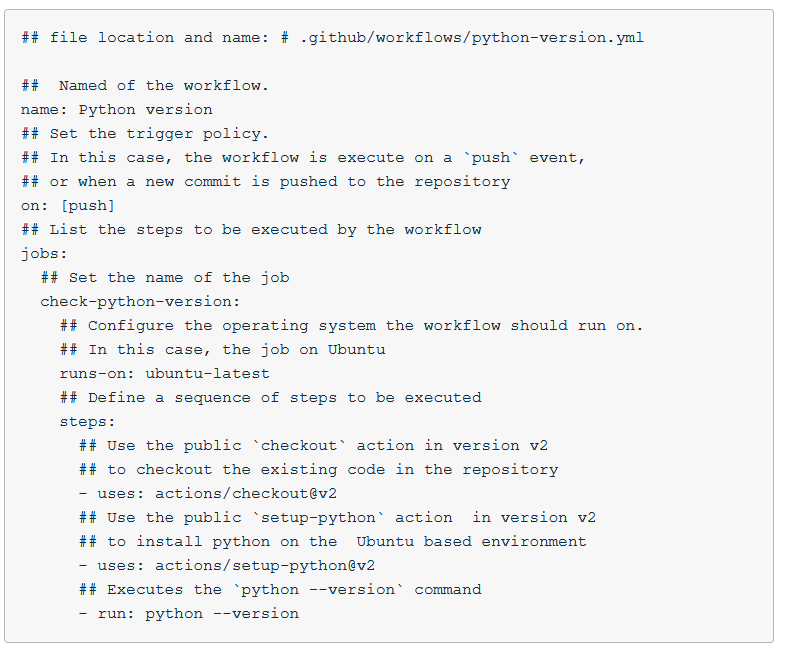
A GitHub Action consists of one or more jobs. A job contains a sequence of steps that execute standalone commands, known as actions. When an event occurs, the GitHub Action is triggered and executes the sequence of commands to perform an operation, such as code build or test.



GitHub Actions structure

Let's configure a GitHub Action that prints the Python version!

GitHub Actions are configured using YAML syntax, hence uses the .yaml or .yml file extensions. These files are stored in .github/workflows directory within the code repository. Within this folder, the python-version.yml contains the following sections:



On the successful execution of the workflow, the Python version is printed. For example, Python 3.9.0.

**Github Actions Walkthrough**

*Note:* To follow this demo make sure you fork the [course repository](https://github.com/udacity/nd064_course_1) and reference the [python-helloworld application](https://github.com/udacity/nd064_course_1/tree/main/solutions/python-helloworld).

This demo provides a step-by-step guide of how to write a GitHub Action to run a suite of pytests.

Pytest is a framework that is used to write functional testing for an application. In this example, within the [test\_with\_pytest.py](https://github.com/udacity/nd064_course_1/blob/main/solutions/python-helloworld/test_with_pytest.py) we create a mock test that will always execute successfully.

**Summary**

## Define a test that will always pass successfully

def test\_always\_passes():

assert True

Once we have completed writing the test, we can construct a GitHub Action that will execute the tests using pytest tool. For this purpose, a new workflow is defined in the [pytest.yml file](https://github.com/udacity/nd064_course_1/blob/main/.github/workflows/pytest.yml) within the .github/workflows directory.

To trigger the workflow a new commit is necessary (e.g. editing the README.md fie). On the successful execution of the GitHub Action, we should see an output mentioning that all the test passed successfully (as expected):

Run pytest

============================= test session starts ==============================

platform linux -- Python 3.8.6, pytest-6.2.1, py-1.10.0, pluggy-0.13.1

rootdir: /home/runner/work/nd064\_course\_1/nd064\_course\_1

collected 1 item

solutions/python-helloworld/test\_with\_pytest.py . [100%]

============================== 1 passed in 0.02s ===============================

Also, this demo showcases the "Python version" GitHub Action that was used as an example in the slides.

**Further reading**

Explore GitHub Actions in more detail:

* [Introduction to GitHub Actions](https://docs.github.com/en/free-pro-team@latest/actions/learn-github-actions/introduction-to-github-actions)
* [Create an example workflow](https://docs.github.com/en/free-pro-team@latest/actions/learn-github-actions/introduction-to-github-actions#create-an-example-workflow)
* Inspect public workflows on [GitHub Marketplace](https://github.com/marketplace?type=actions)