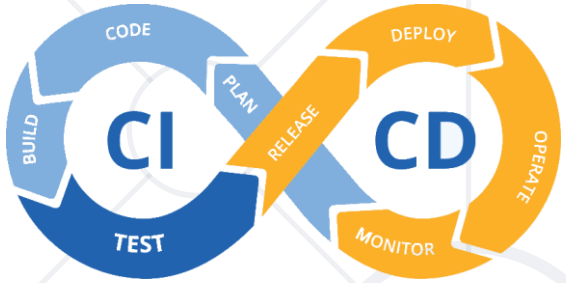
# Continuous Integration, CI / CD Pipelines

## Continuous Integration, Continuous Delivery, Continuous Deployment, GitHub Actions, YAML




**SoftUni Team**

**Technical Trainers**

Software University

SoftUni

Software University

https://softuni.bg

# Have a Question?

Software University

# sli.do

# #Dev-Ops

# Table of Contents



# The CI/CD Pipelines

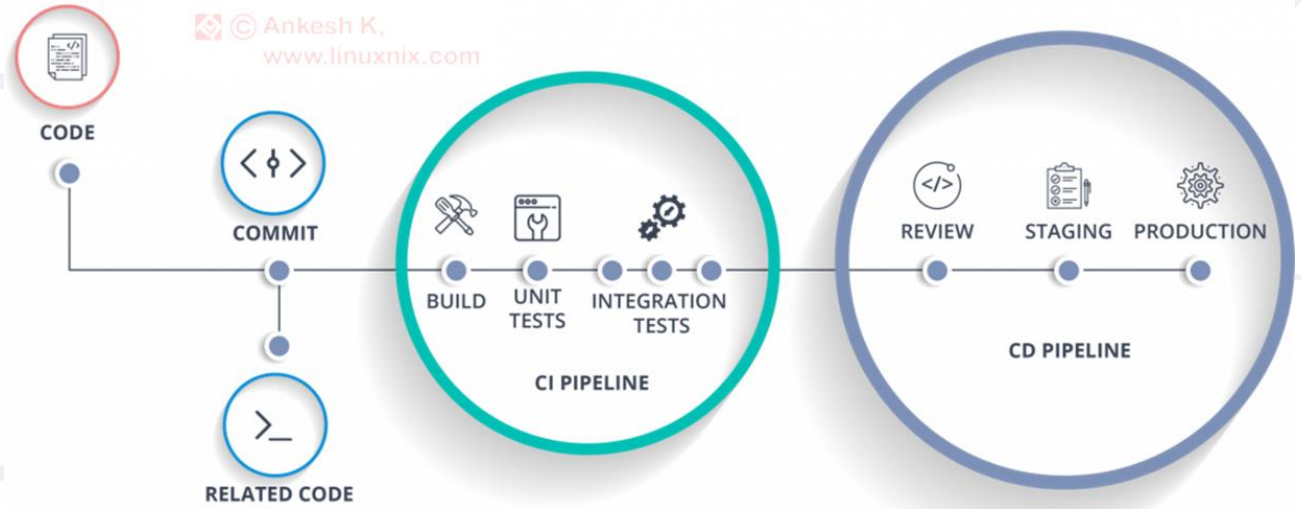
# What is CI/CD?

- **CI/CD** = **Continuous Integration** + **Continuous Delivery**
  ( + **Continuous Deployment**)


- **Automates** much of the process to get new code from a **commit** into **production**
  - Developers **regularly** merge their code changes into a **central** repository, which is then **automatically tested** and **deployed** to **production** to ensure frequent and reliable software updates

# CI/CD Overview

- **CI/CD pipeline**
  - Continuously **integrate** and **release** new features
- **Continuous integration** (**CI**)
  - Write code, test it and **integrate** it in the product
- **Continuous delivery** (**CD**)
  - Continuously **release** new features
- **QAs** monitor and sometimes maintain the CI/CD pipeline



# Continuous Integration (CI)

- Integrating the code from different developers frequently (at least once a day)
- **Automated building** and **testing** the code
  - Typically, at **Git push** in certain branch
- **Finding integration problems** and bugs early
  - Continuously maintain software quality
- CI is implemented by a **CI system** (like Jenkins, GitHub Actions, TeamCity, Azure Pipelines)

# Continuous Testing (CT)

- Regularly execute **automated tests** as part of the software delivery pipeline
  - Ensures consistent software quality
- Implemented with a **CI system**
  - **Unit tests** executed at each commit / push
  - **Integration tests** executed at each major commit / push
  - **End-to-end tests** executed every night (execution takes hours)

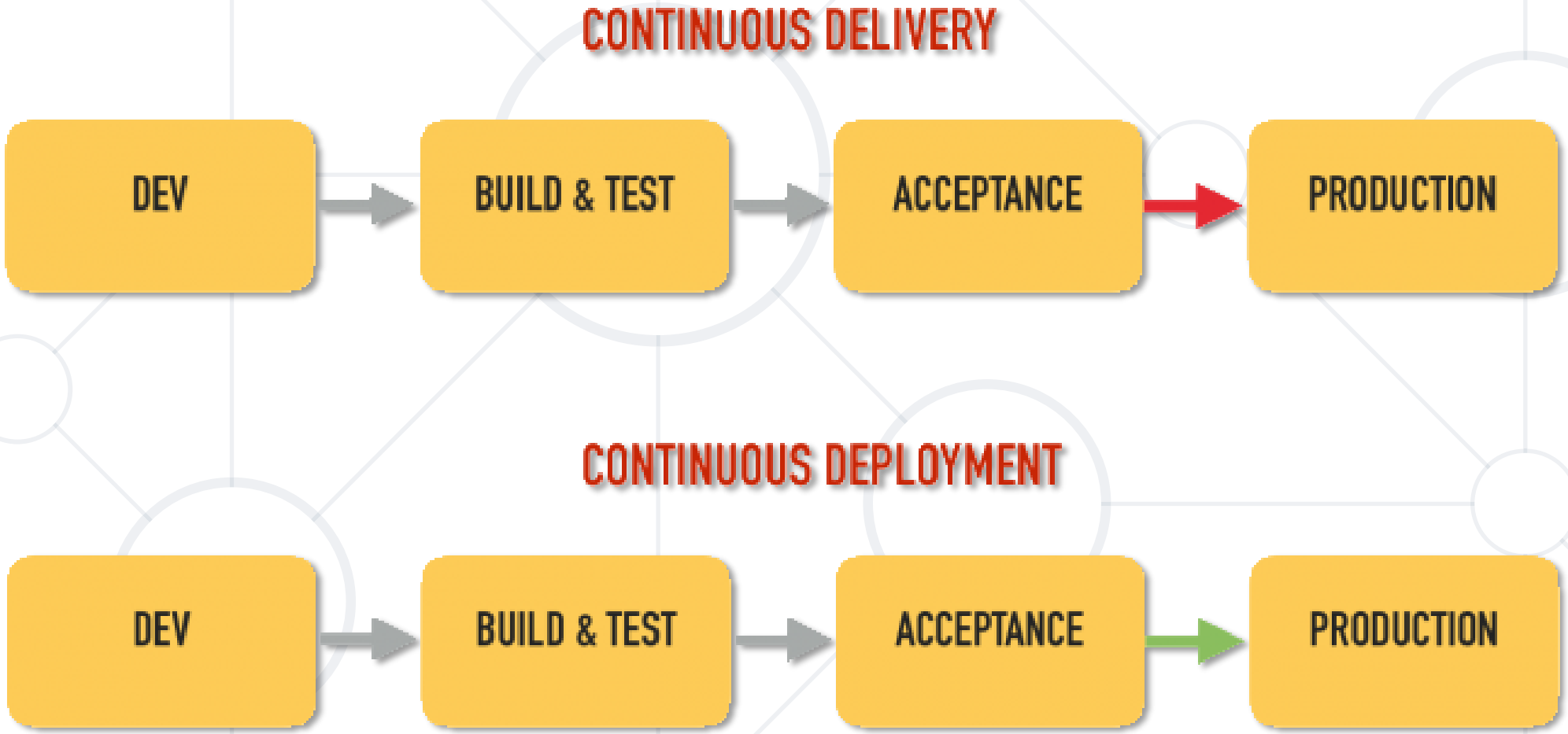
# Continuous Delivery (CD)

- Keeping your codebase **deployable at any point**
- **CD** continuously verifies that
  - Software **builds** correctly
  - Passes the **automated tests**
  - Has all the necessary **configuration** and assets for **deployment in production**
- E.g., build an **.apk** package for Android apps

# Continuous Deployment (CD)

- Continuous **automated deployment**
- E.g., after each **git push** in certain branch
  - The software is **built**, the **tests** are executed, and binaries are **deployed** and configured correctly
- Automated deployment typically uses a **testing environment**
  - Sometimes directly to the **production** servers
- Deployment should be done by script (not by hand)

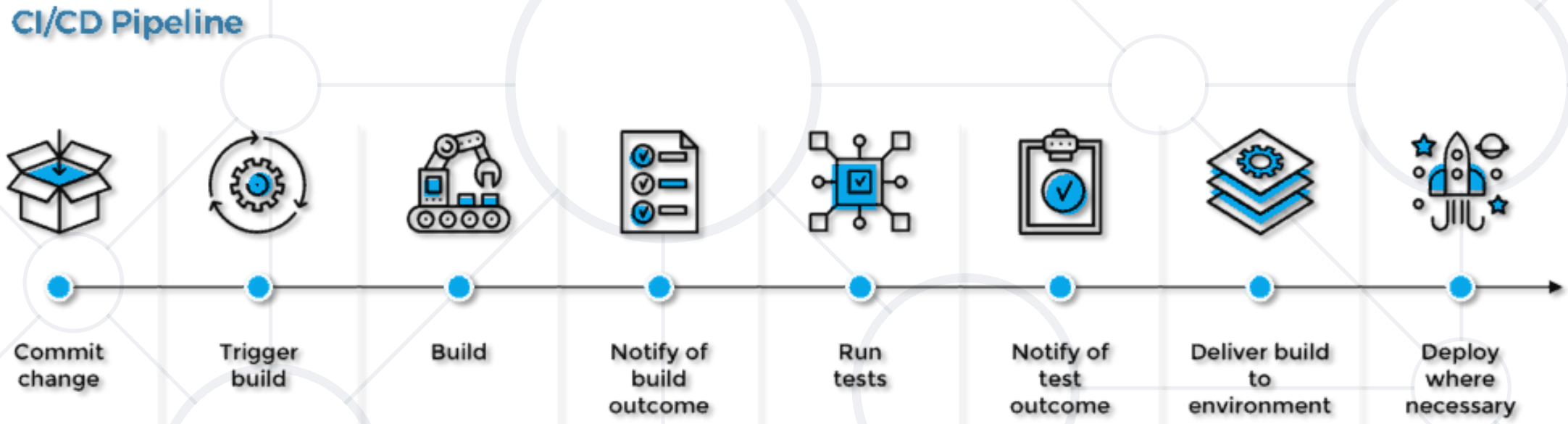
# Continuous Delivery vs. Continuous Deployment



# CI/CD Pipelines

- **CI/CD pipeline** == **CI** + **CD**
  - Continuously integrate, test and release new features
- On **git push**, the CI/CD pipeline does automatically
  - **Build** the software (compile, package, sign, etc.)
  - Run the **automated tests** (unit & integration)
  - **Deploy** in the testing environment & run E2E tests
    - Or only prepare for deployment
    - Or deploy directly on production

# CI/CD Pipeline View



# CI/CD & Software Development Environments



- **Development** environment
  - Code commit
- **Testing** environment
  - Continuous integration, automated testing
- **Staging** environment
  - Continuous delivery, user acceptance test
- **Production** environment
  - Continuous deployment, monitoring

# CI/CD Principles

- A **single source repository**, which contains everything needed for the build
  - Source code, database structure, libraries, scripts, etc.
- **Frequent iterations and check-ins** to the main branch
  - Use small segments of code and merge them into the branch often
- **Automated** and **self-testing builds**

# CI/CD Benefits

- Higher efficiency of web deployment
- Reduced risk of defects
- Faster product delivery
- Exclusive log generation
- Easier rollback of code changes
- More test liability
- Customer satisfaction

# CI/CD Systems

Software University

CI → CI → CD → CD

## Source Code Control

Automatically trigger CI/CD pipeline based on code check-in.




GitLab

GitHub

## Build & Test Automation

Start automated build and test, including functional, security and performance tests.

Jenkins

GitLab


Selenium

docker

Maven

## Release Automation

Update artifact repository with latest successful code artifacts or containers for record-keeping and accessibility.

JFrog Artifactory

docker

Nexus

## Deploy to Staging & Production

Deploy applications to staging and migrate to production using either a blue/green or canary process.

Microsoft Azure

Amazon AWS

Google Cloud Platform

Physical

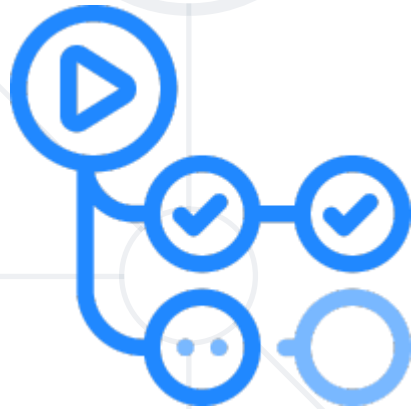
Virtual

openstack

# CI/CD Tools

# GitHub Actions

- GitHub Actions == powerful **CI/CD platform**
  - Integrated directly into GitHub repos
- Enables developers to **automate** workflows, build, test and code deployment
- Large **library** of pre-built actions and custom workflows
- Free for public repos + 2000 mins per month for private repos with the **free plan**



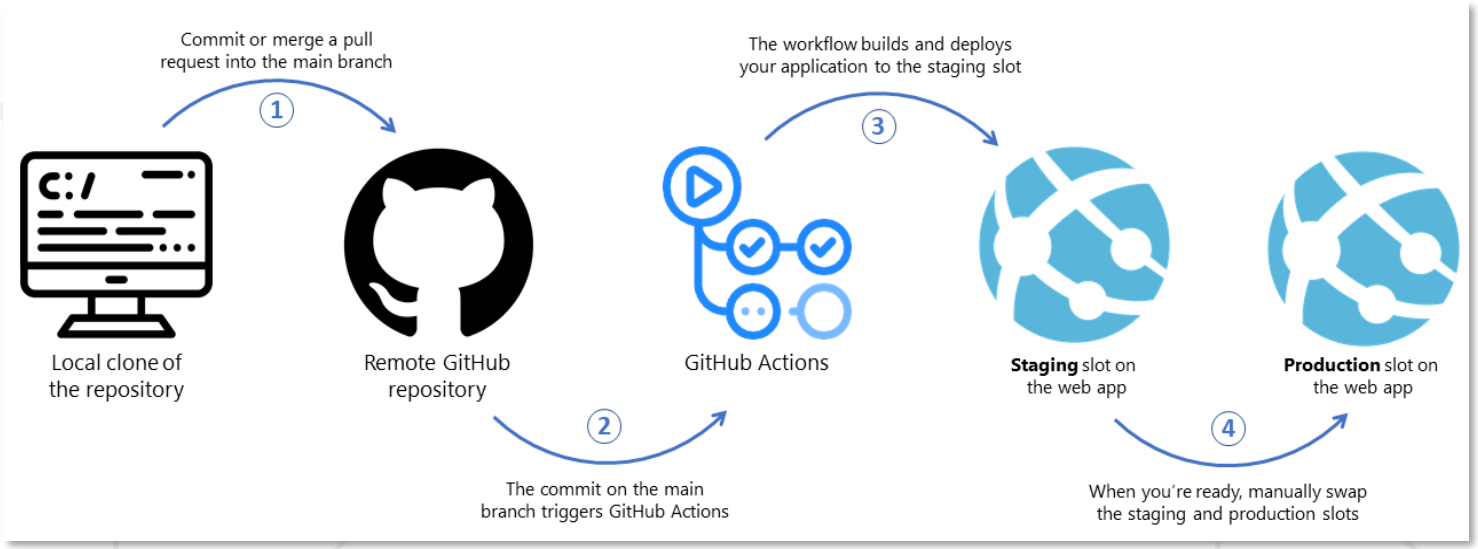
# GitHub Actions

Software University

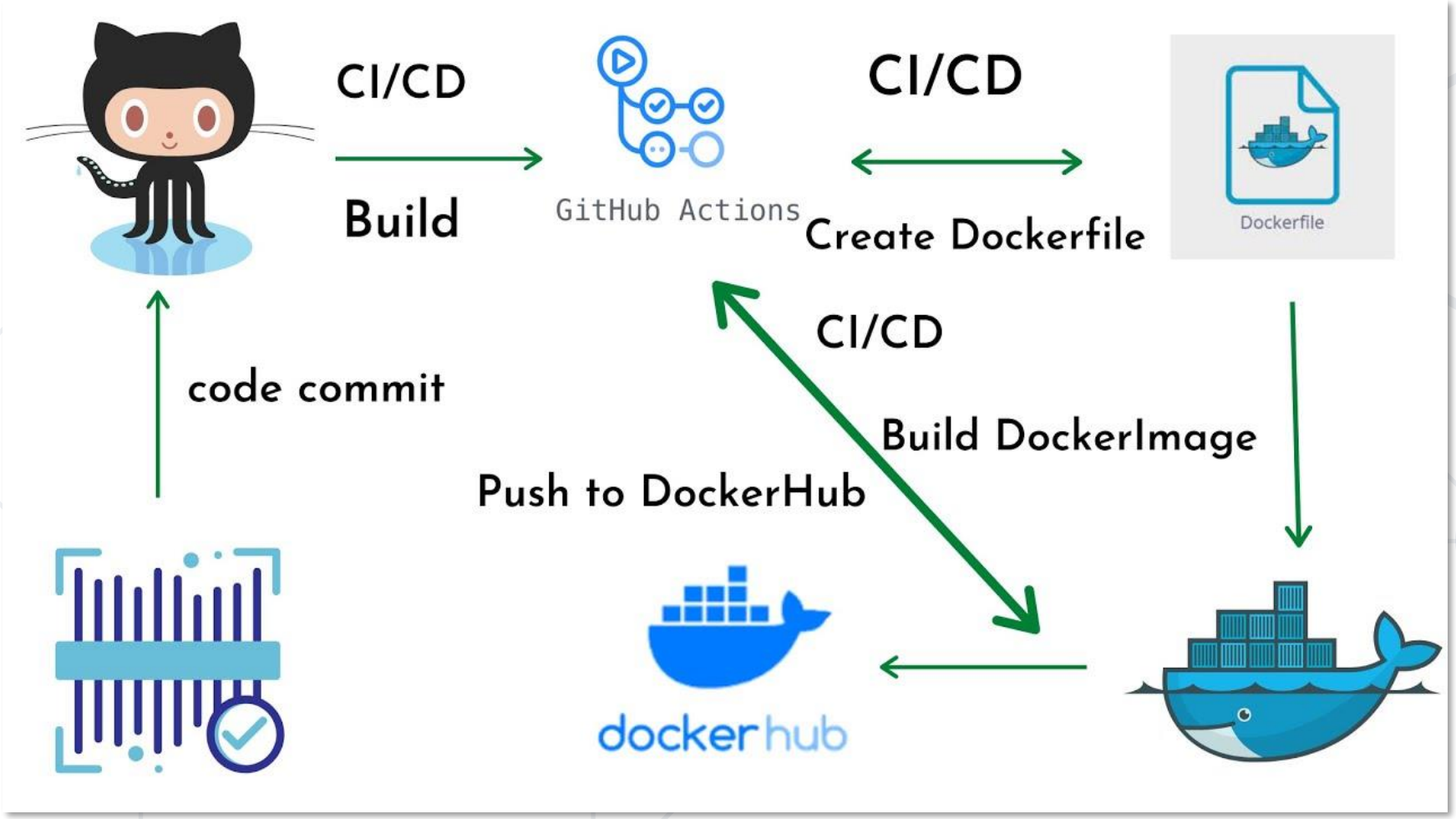
- Flexible environment
  - Supports **various** programming languages
- Allows developers to **trigger workflows**
  - Based on events like code commits, pull requests, issue updates
- Allows defining **custom** workflows
- YAML syntax

# GitHub Actions and Other Platforms

- You can use it to integrate and deploy code changes to a **cloud application platform** and test, track, and manage these changes
- With **GitHub Actions for Azure**, you can deploy to Azure
- GitHub Actions also supports other **CI/CD tools**, **Docker**, and **automation platforms**



# GitHub Actions and Other Platforms



# CircleCI



- CircleCI == **CI/CD platform**
- Supports a wide range of programming languages
- Integrates with **version control systems**
  - e.g., GitHub and Bitbucket
- User-friendly configuration file
  - Allows custom workflows

# Azure DevOps

- Azure DevOps == set of deployment tools that enables
  - Planning
  - Developing
  - Testing
  - Deployment
- Facilitates creation of CI/CD pipelines

# Azure DevOps

Software University

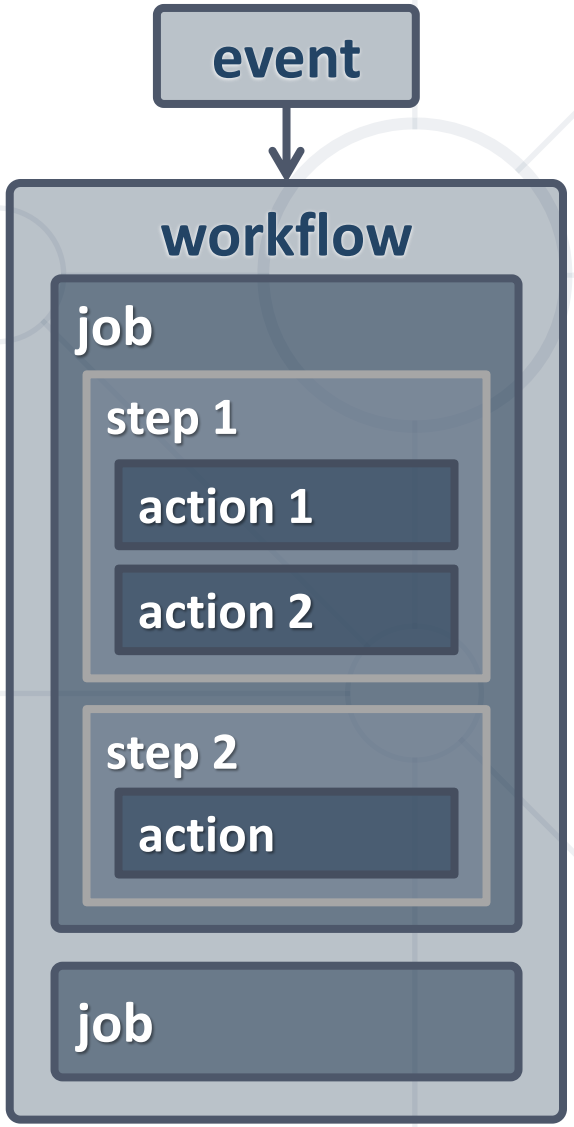
- Offers **services** for
  - Version control
  - Agile project management
  - CI/CD
  - Application monitoring
  - and many more…
- Supports **integration** with **version control systems**, e.g., Git
- Provides **code repository** for managing source code

# GitHub Actions

Introduction

# Concepts

- **Events** execute **workflows**
  (one or several jobs, running in parallel)
- **Workflows** hold **jobs**
  (e.g., build, check security, deploy)
- **Jobs** hold **steps** (e.g.. "checkout the code", "install .NET", "run tests", …)
- **Steps** hold **actions**
  (commands like `` `dotnet test` ``)



# Events

- Specific **triggers** that can activate workflows in a repository
- Allow **automation** of various **tasks** and **actions** based on different types of events that occur in the within the repository
- Each event can be used to **start a workflow** that performs **specific action**, e.g.,
  - Running tests
  - Deploy code
  - Sending notifications

# Events Types

- **Repository**
  - Specific to the repository and are triggered by actions like code pushes, pull requests, etc.
- **Workflow**
  - Related to the workflows themselves and are triggered by workflow-specific events
- **Webhook**
  - Triggered by external services integrated with GitHub using webhooks
- **External**
  - Specific to actions taken by external services
- **Internal**
  - Related to actions within the GitHub repository or organization

# Workflow

- **GitHub Actions workflow** is a configurable automated procedure
- Made of one or many **jobs**
- Defined by a **YAML file** in **.github/workflows** folder in your repo
- Can be triggered by **events** in the repo, on **schedule** or **manually**

.github > workflows > my-workflow.yaml

```yaml
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository ···
      - name: Install Node.js ···
      - name: Install bats ···
      - name: Run bats ···
```

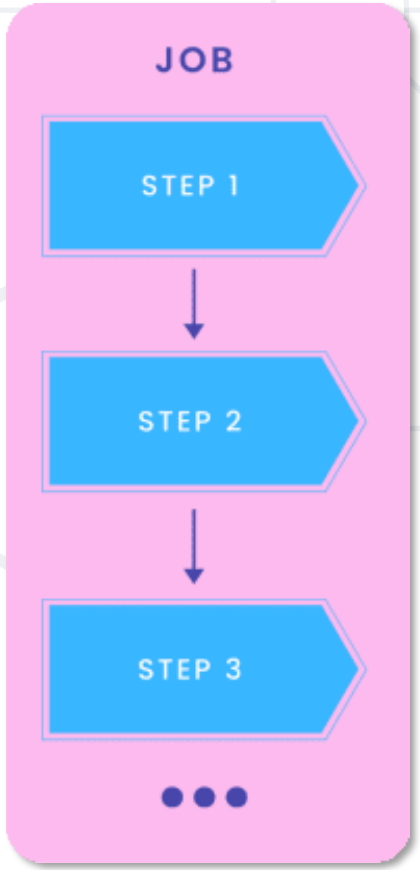

- A **GitHub repository** can have **multiple workflows**

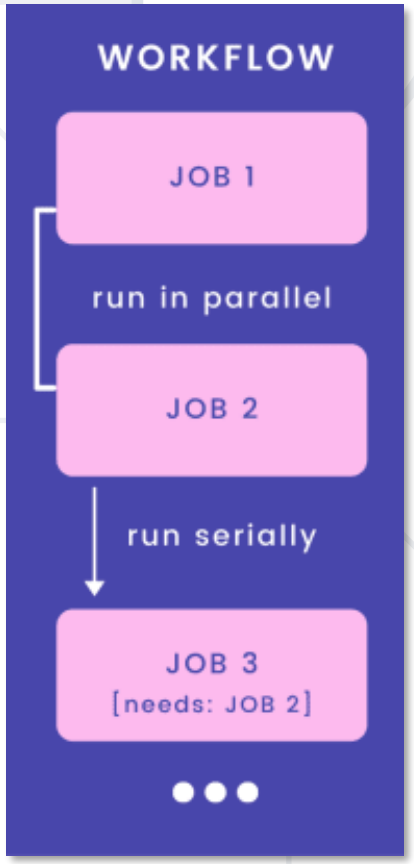
# Jobs

- **Job** == a **set of steps** that will be executed on **the same runner**
- All **jobs in the workflow** normally run in **parallel**
- When you have **jobs that depend on each other**, they run **serially**

.github > workflows > my-workflow.yaml

```yaml
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository ···
      - name: Install Node.js ···
      - name: Install bats ···
      - name: Run bats ···
```

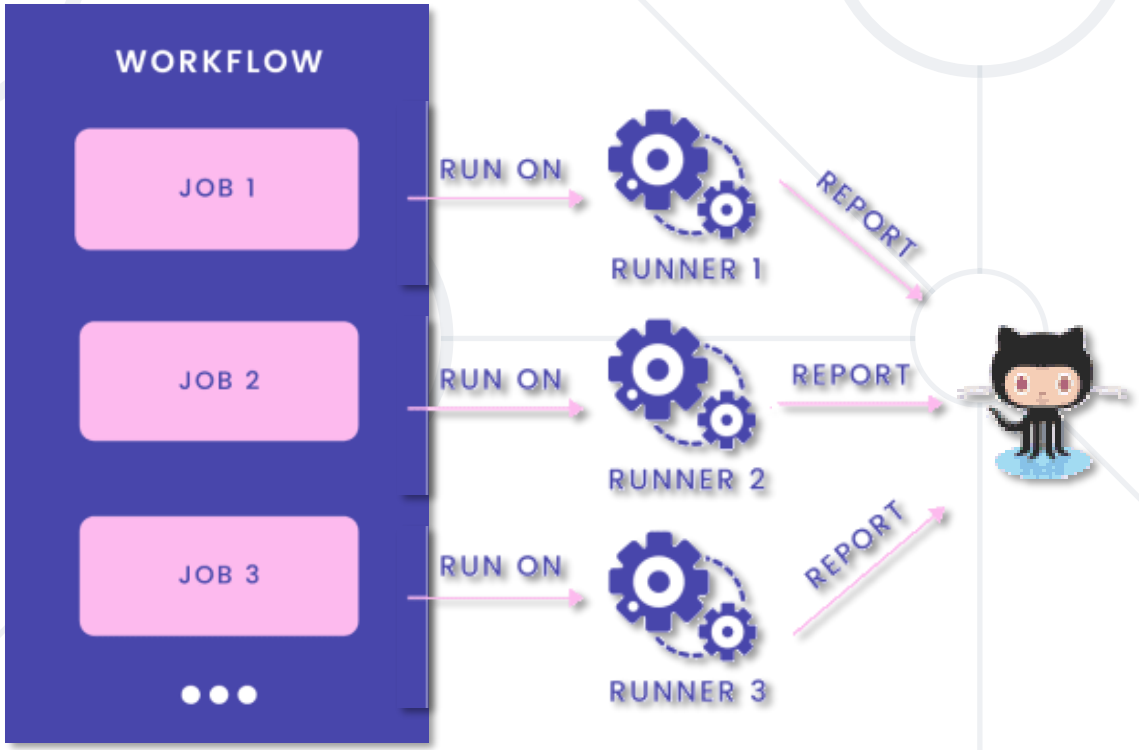


# Runners

- To **run jobs**, we must specify a **runner** for each of them
- A **runner** is a **server that runs jobs**
- Runs only **1 job at a time**
- Reports **job progress**, **logs**, and **results** back to GitHub
  - We can look at them in the UI of the repository
- Two types: **GitHub hosted** or **self-hosted**

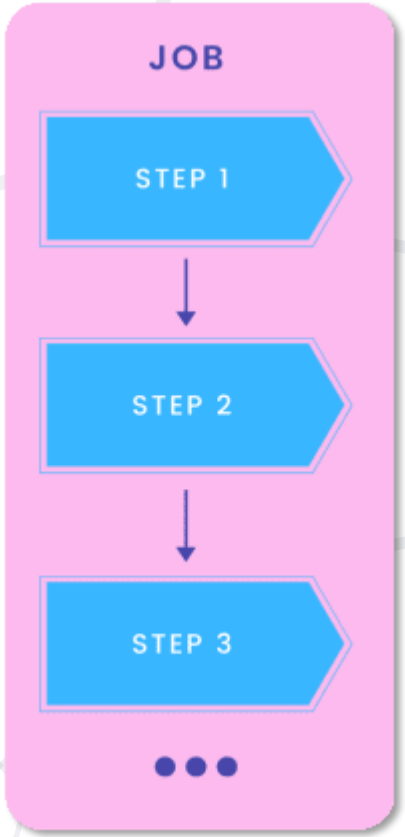
```
3   jobs:
4     check-bats-version:
5       runs-on: ubuntu-latest
```

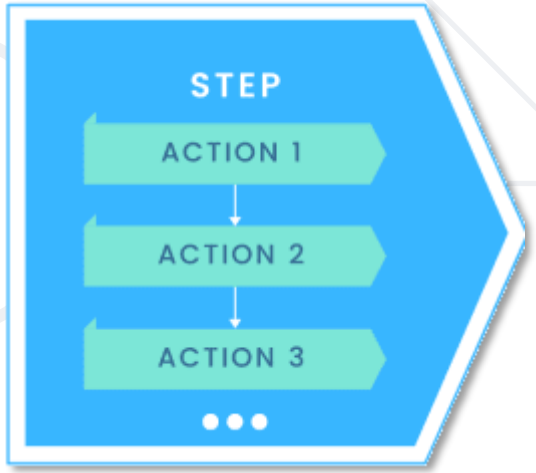


# Steps and Actions

- **Steps** are **individual tasks** within a **job**
- They run serially, one after another
- Each **step** is either a **shell script** that will be executed, or an **action** that will be run
- An **action** is basically a **standalone command**
- **Actions** run serially within a step
- **Actions** can be reused

```
3   jobs:
4     check-bats-version:
5       runs-on: ubuntu-latest
6       steps:
7         - name: Check out repository
8           uses: actions/checkout@v3
9 >       - name: Install Node.js ···
13 >      - name: Install bats ···
15 >      - name: Run bats ···
```



# Workflow Syntax Keywords

- **name**
  - For **names** of workflows, steps, which GitHub Action displays
- **on**
  - Used to define which events can cause the workflow to run (**triggers**)
- **jobs**
  - Used to **list jobs**
- **runs-on**
  - Specify **runner environment**

.github > workflows > my-workflow.yaml

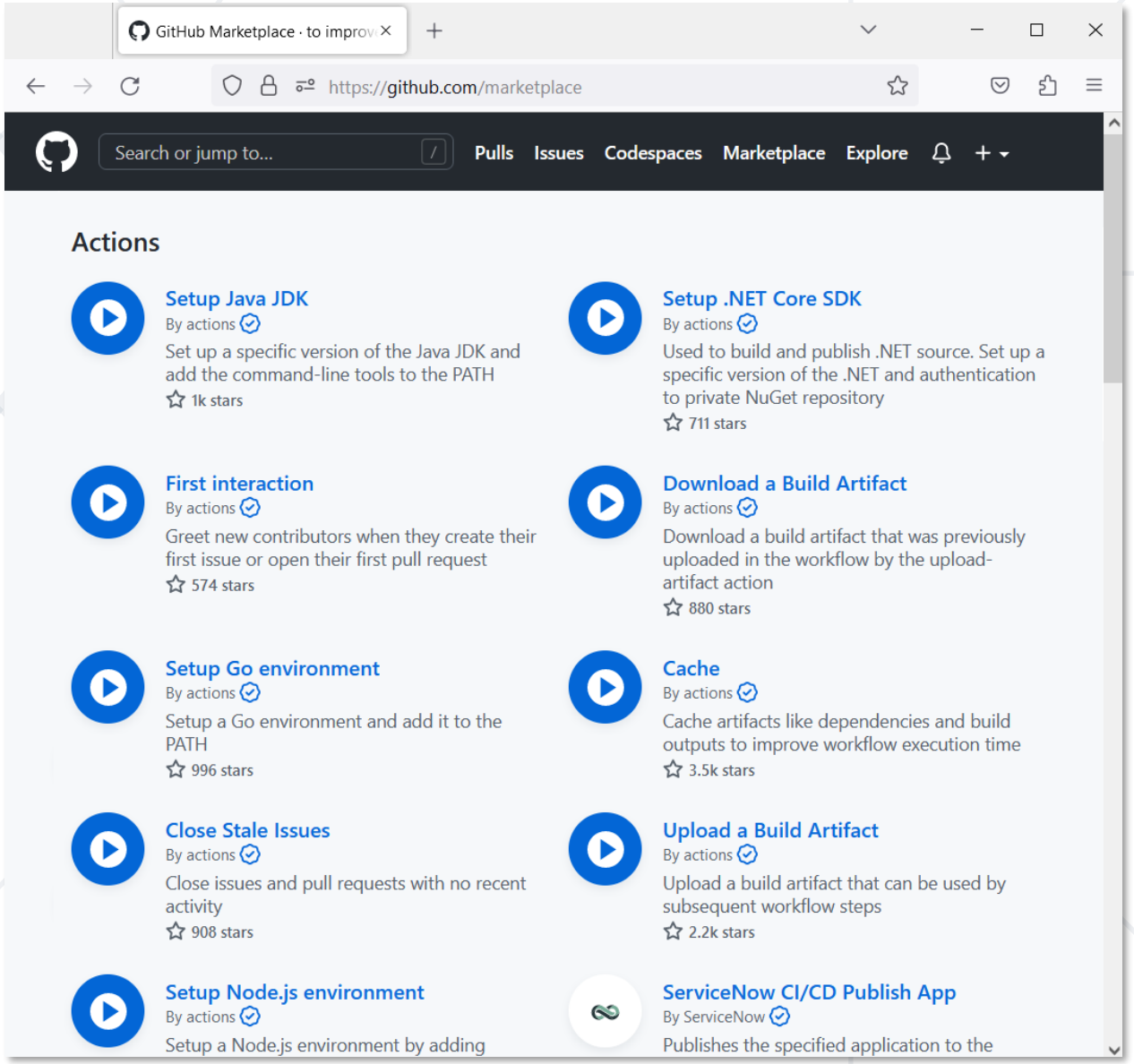
```yaml
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository
        uses: actions/checkout@v3
      - name: Install Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '14'
      - name: Install bats
        run: npm install -g bats
      - name: Run bats
        run: bats -v
```

# Workflow Syntax Keywords

- `steps`
  - Used to **list steps** to run in the job
- `uses`
  - **Use an action** which is already defined with its version (v3)
- `with`
  - **Input parameters** required by some actions
- `run`
  - Tells the job to **execute a Shell command** on the runner

# GitHub Marketplace

- GitHub Marketplace contains tools that add functionality and improve your workflow
- You can discover, browse, and install tools, including **GitHub Actions**
- GitHub uses it to suggest you **workflow templates** based on code in your repo




# YAML

## Definition and Syntax

# YAML

- YAML (**Y**AML **A**in't **M**arkup **L**anguage)
  - Originally – **Y**et **A**nother **M**arkup **L**anguage
  - Human-readable data serialization **standard**
  - Focused on clear **data serialization**
- Widespread adoption
  - Common in configuration and system management
  - Favored in DevOps for complex systems
  - Principal format for deployment and setup

# YAML Syntax

- **Indentation**
  - **Spaces** are used to denote structure
  - **Tabs** are not allowed
- **Key-Value Pairs**
  - The basic structure of YAML is **map** of **key-value** pairs
  - Separated by colons ( **:** )
- **Lists**
  - Represented by a **dash**, followed by a **space** ( **-** )
- **Comments** (**#**)
  - Used to add comments within YAML files

# Advanced Structures

- **Nested Structures**
  - Consist of maps and lists

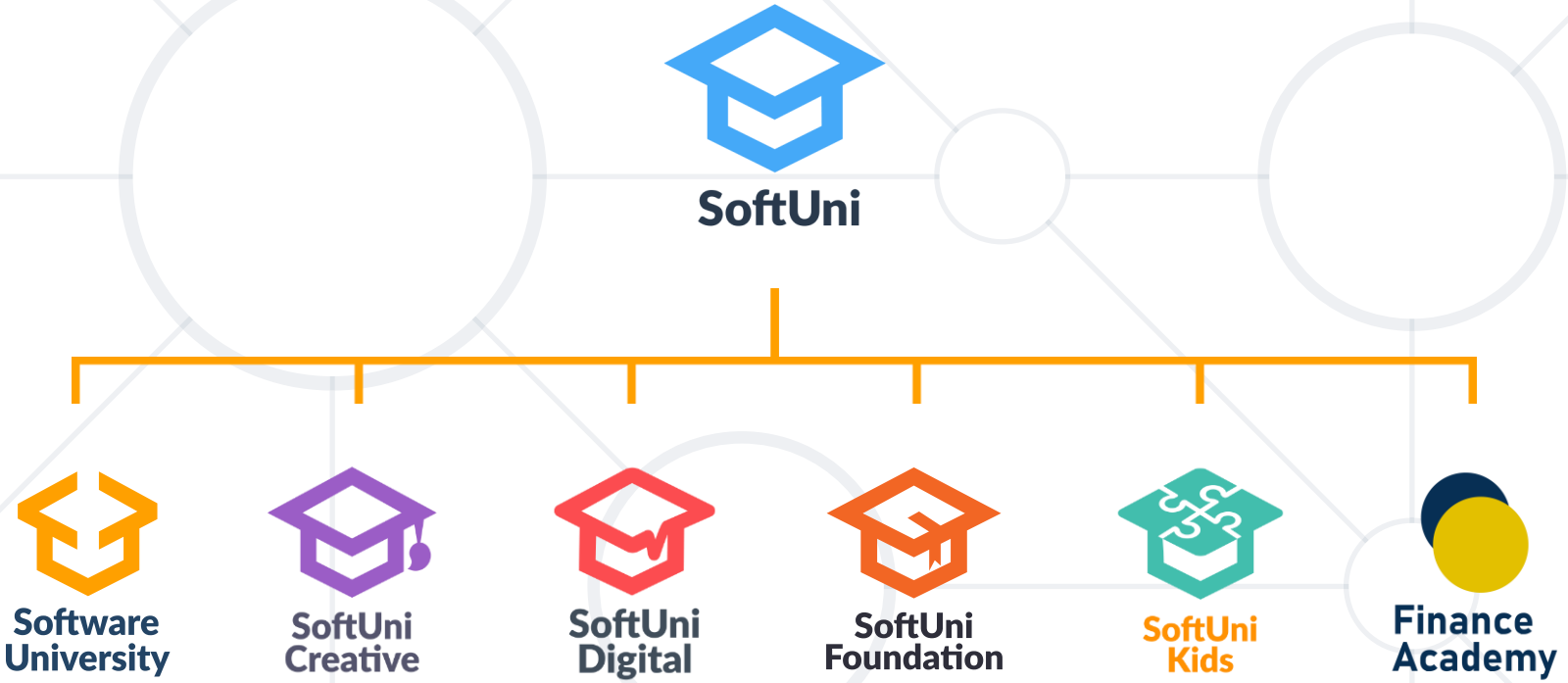
```
parentKey:
  childKey: value
  childList:
    - Item 1
    - Item 2
```

- **Scalars**
  - Strings, numbers and Booleans
    - No need of **quotes around strings** unless the string contains special characters

# Summary

- **CI/CD** == a method to **frequently deliver apps** by introducing **automation** into **continuous delivery** and **continuous deployment**
- There are a lot of **CI/CD platforms**
  - **GitHub Actions,** in which you can create **workflows** to **automate** your **build**, **test** and **deployment pipeline,** using **YAML**

# Questions?



# SoftUni Diamond Partners

Software University












# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg