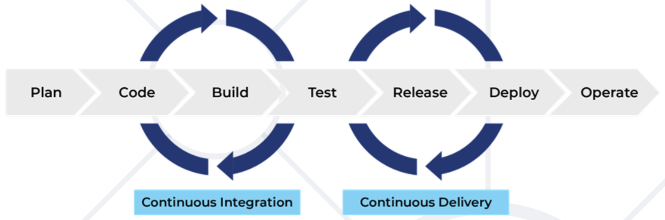
Continuous Integration (CI)

Continuous Integration, GitHub Actions, Jenkins



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

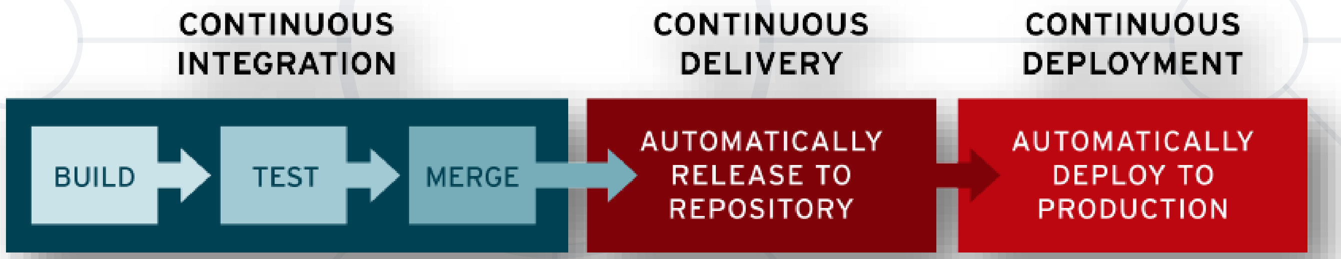
#QA-BackEnd

Table of Contents

The CI/CD Pipelines

What is CI/CD?

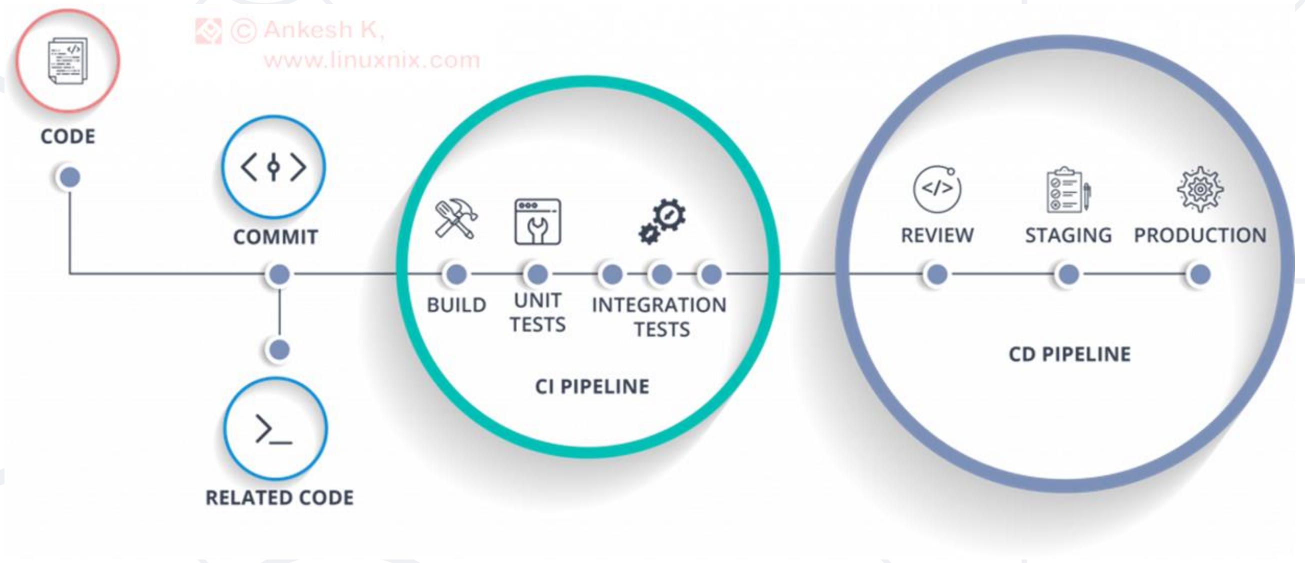
- **CI/CD** = **Continuous Integration** + **Continuous Delivery** (+ **Continuous Deployment**)



- **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)

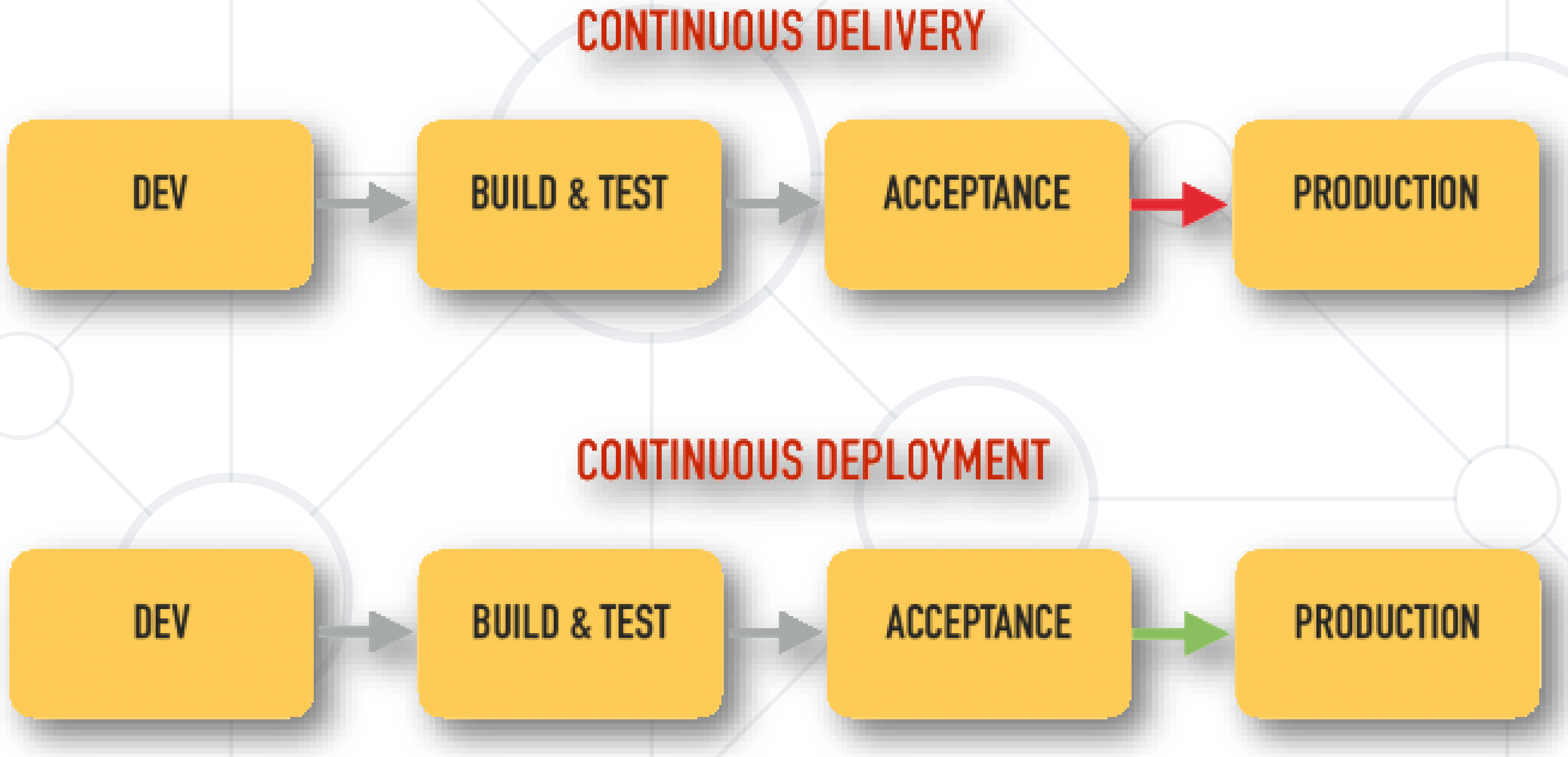
Continuous Delivery (CD)

- 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)

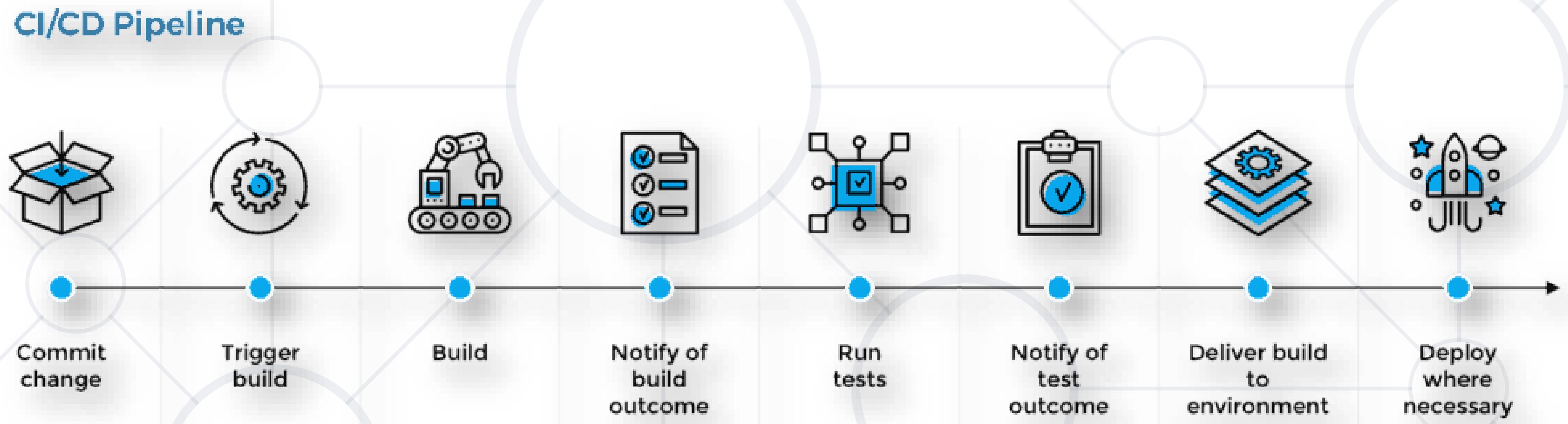
Continuous Delivery vs. Continuous Deployment



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

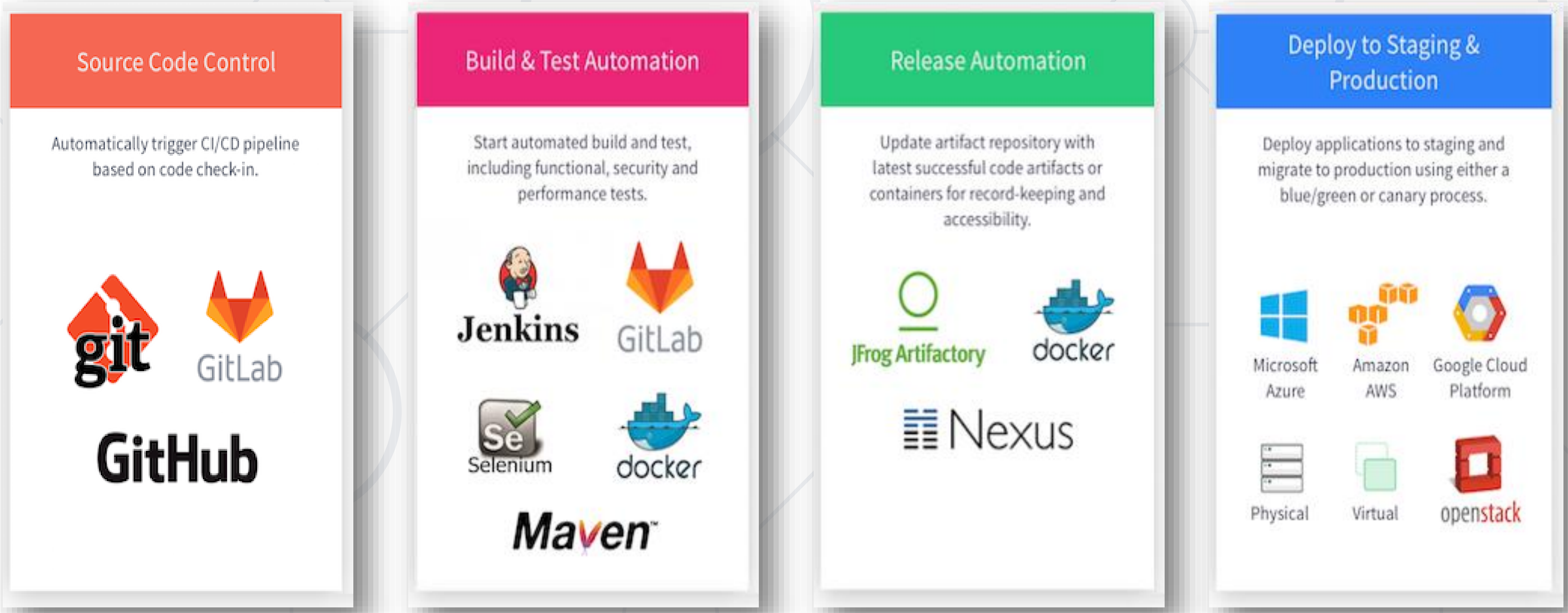
- A **single source repository**, which contains everything needed for the build
  - Source code, database structure, libraries, scripts, etc.
- **Frequent iterations and check-ins** to the main branch
  - Use small segments of code and merge them into the branch often
- **Automated** and **self-testing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

CI → CI → CD → CD



Source Code Control

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

Build & Test Automation

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

Release Automation

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

Deploy to Staging & Production

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

GitHub Actions

Introduction

GitHub Actions

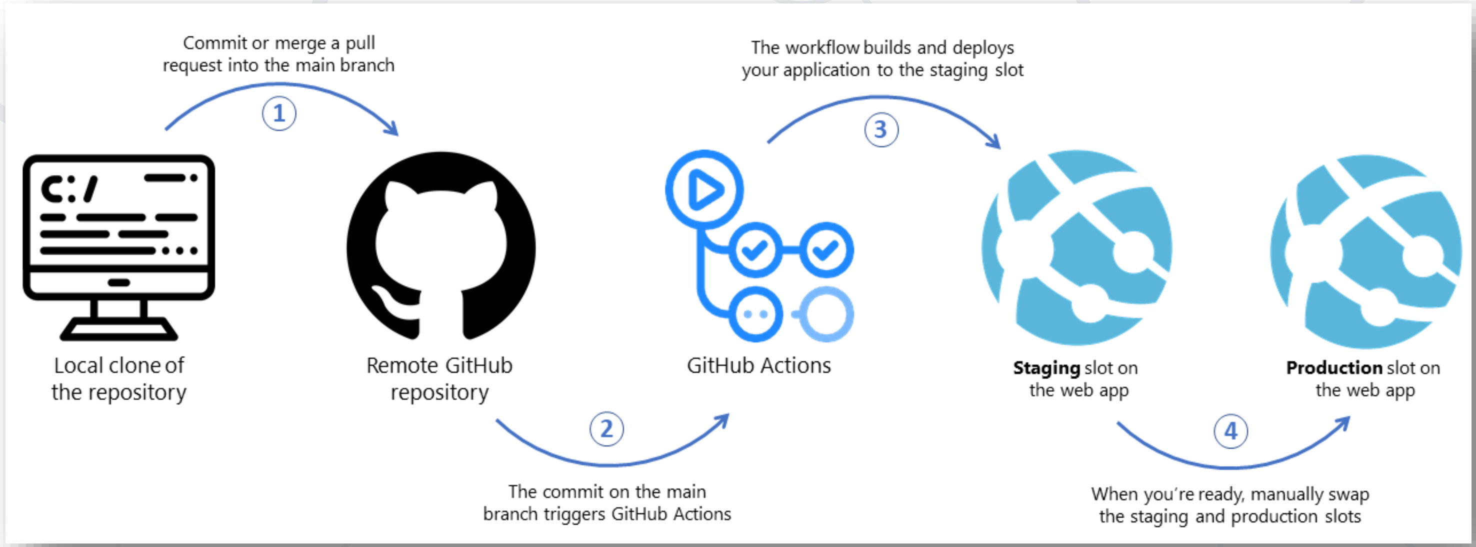
- GitHub Actions
  - Powerful **CI/CD platform**
  - Integrated directly into GitHub repos
- Enables developers to **automate** workflows, build, test and code deployment
- Free for public repos + 2000 mins per month for private repos with the **free plan**

GitHub Actions

- 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
- Large **library** of pre-built actions and custom workflows

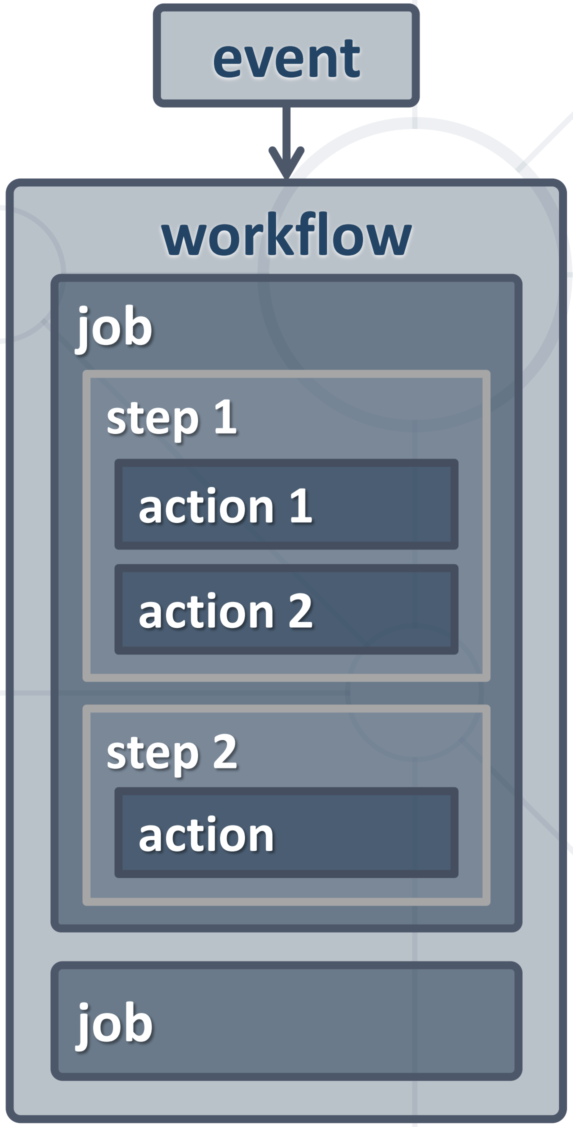
GitHub Actions and Other Platforms

- 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**


Commit or merge a pull request into the main branch
1
Local clone of the repository
Remote GitHub repository
2
The commit on the main branch triggers GitHub Actions
GitHub Actions
3
The workflow builds and deploys your application to the staging slot
Staging slot on the web app
Production slot on the web app
4
When you're ready, manually swap the staging and production slots

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

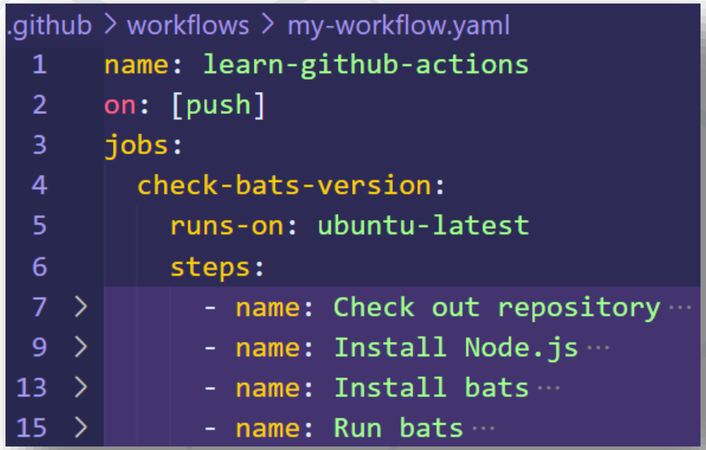
- Specific **triggers** that can activate workflows in a repository
- Allow **automation** of various **tasks** and **actions** based on different types of events that occur in the within the repository
- Each event can be used to **start a workflow** that performs **specific action**, e.g.
  - Running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

Events Types

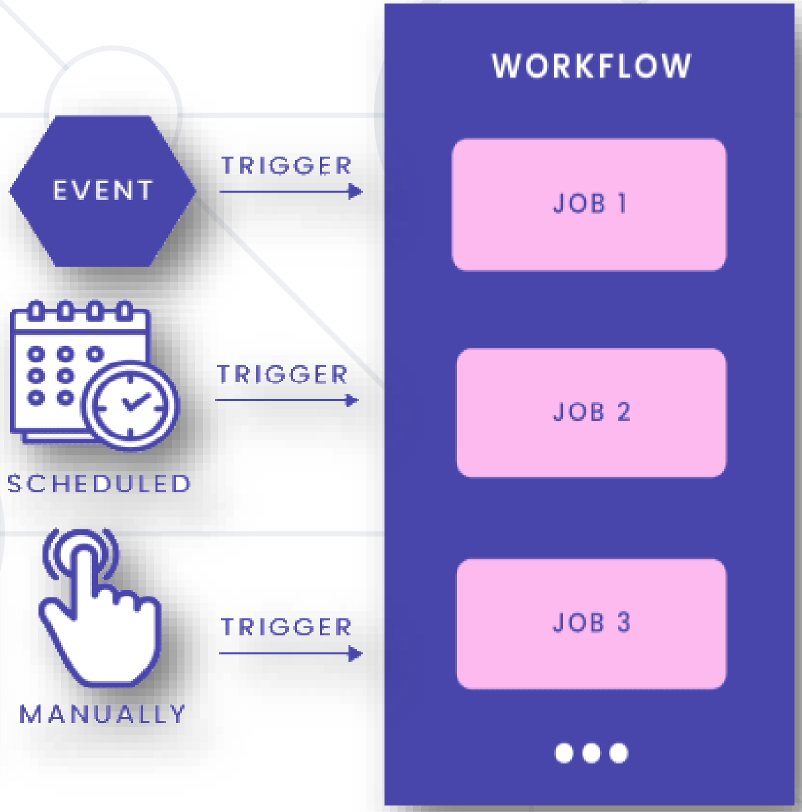
- **Repository**
  - Specific to the repository and are triggered by actions like code pushes, pull requests, etc.
- **Workflow**
  - Related to the workflows themselves and are triggered by workflow-specific events
- **Webhook**
  - Triggered by external services integrated with GitHub using webhooks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

Workflow

- **GitHub Actions workflow** == 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**
- A **GitHub repository** can have **multiple workflows**

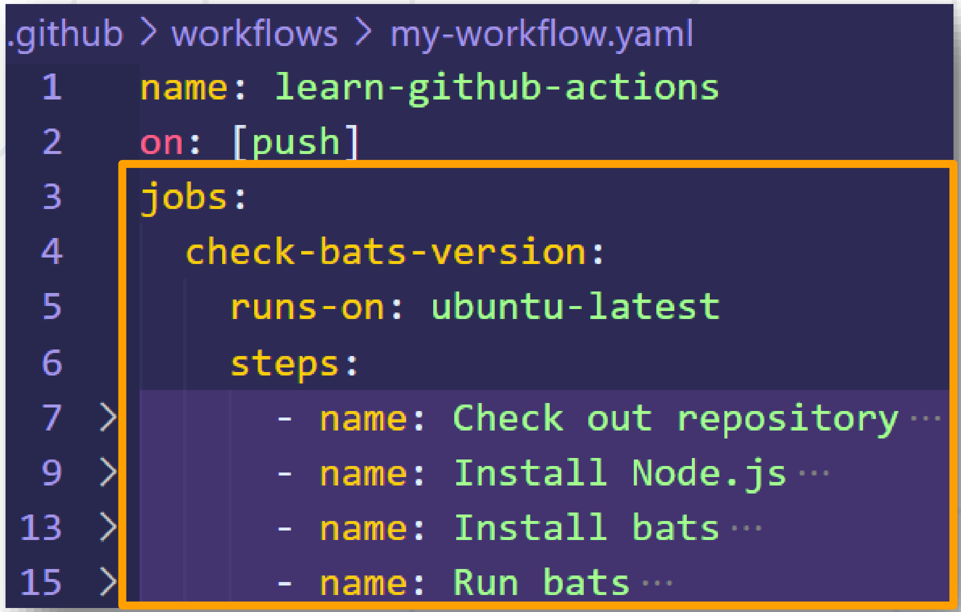

.github > workflows > my-workflow.yaml

```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 ···
```



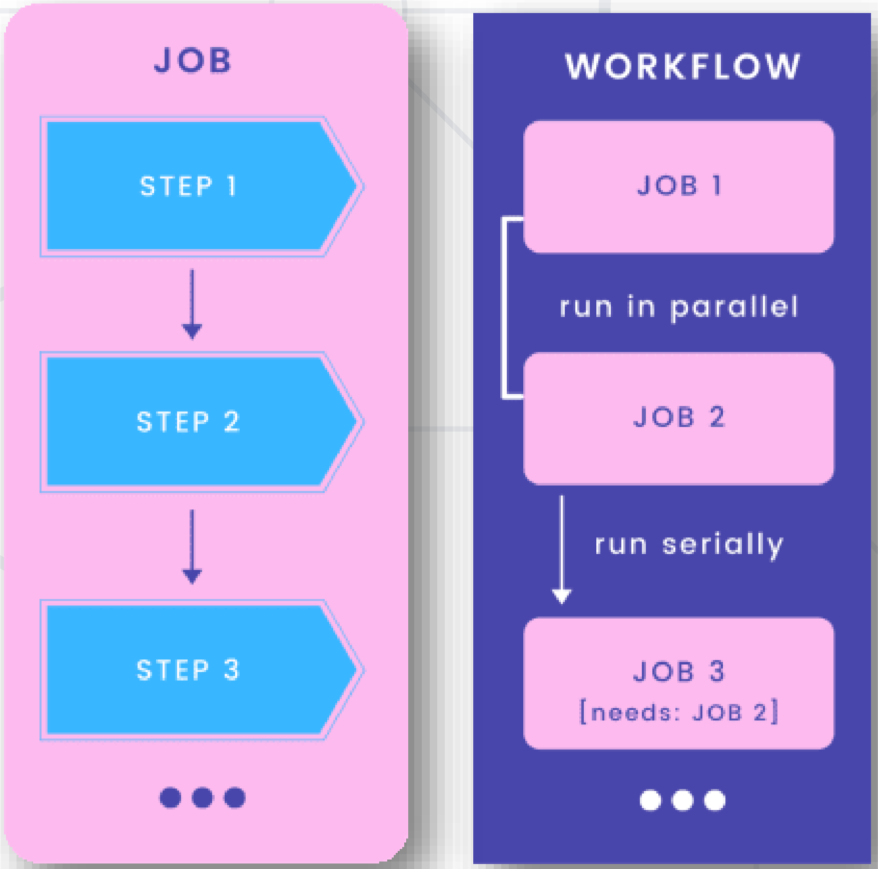
Jobs

- **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**



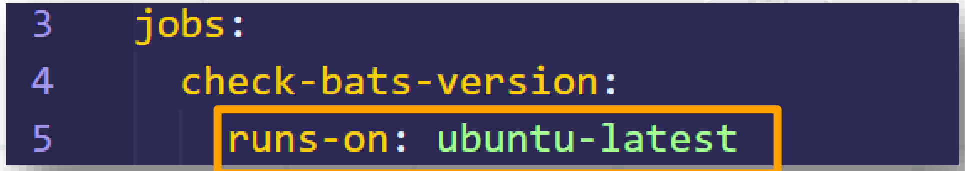
.github > workflows > my-workflow.yaml

```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 ···
```

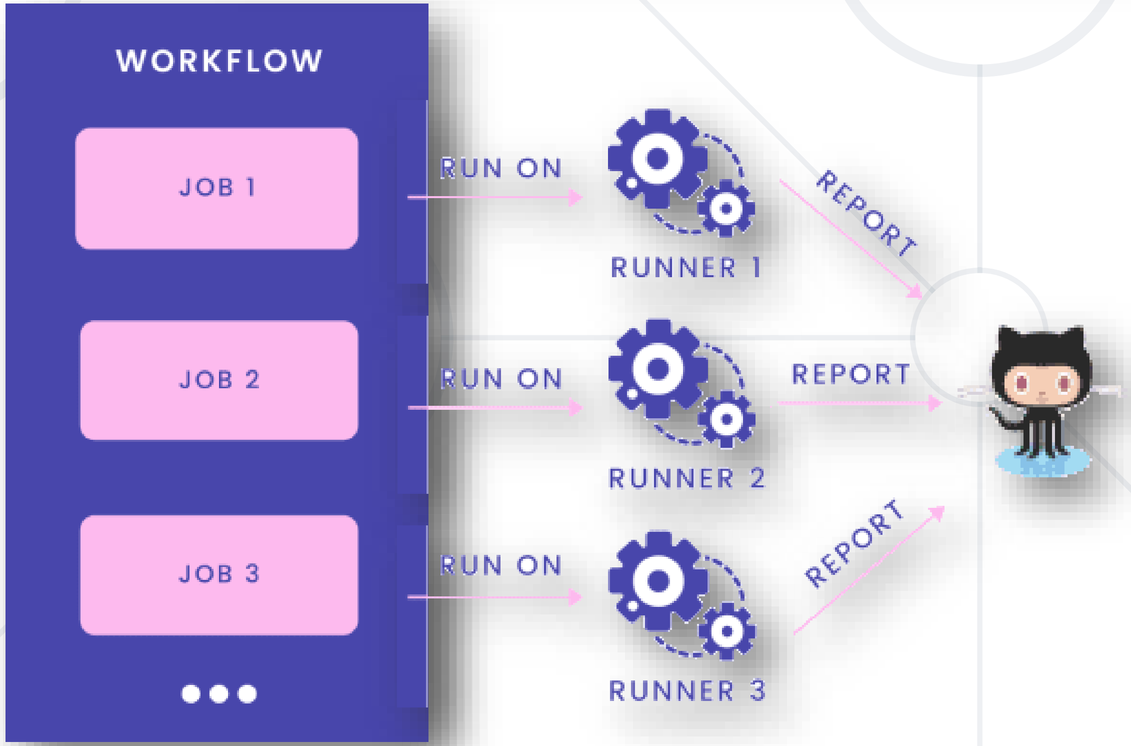


Runners

- 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**

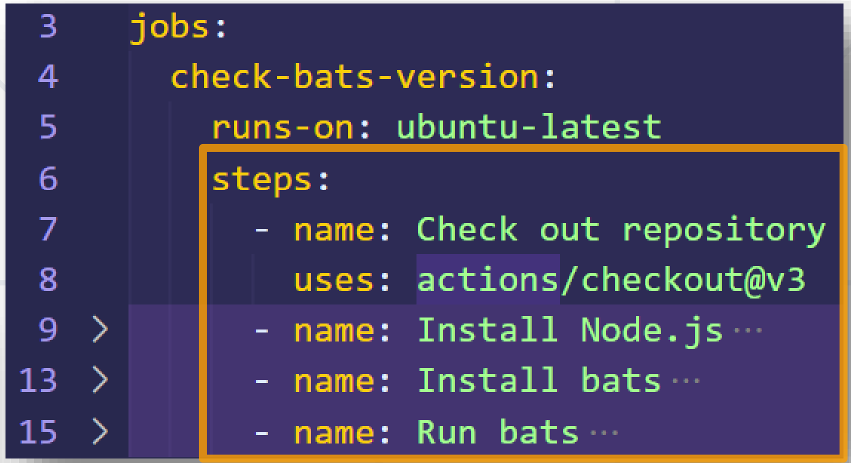


```
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
```

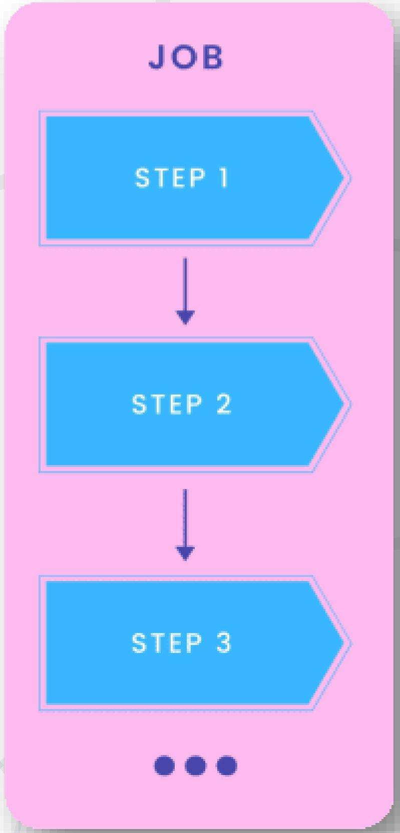


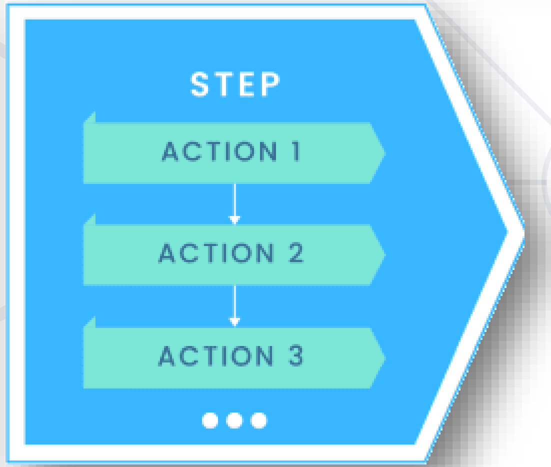
Steps and Actions

- **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



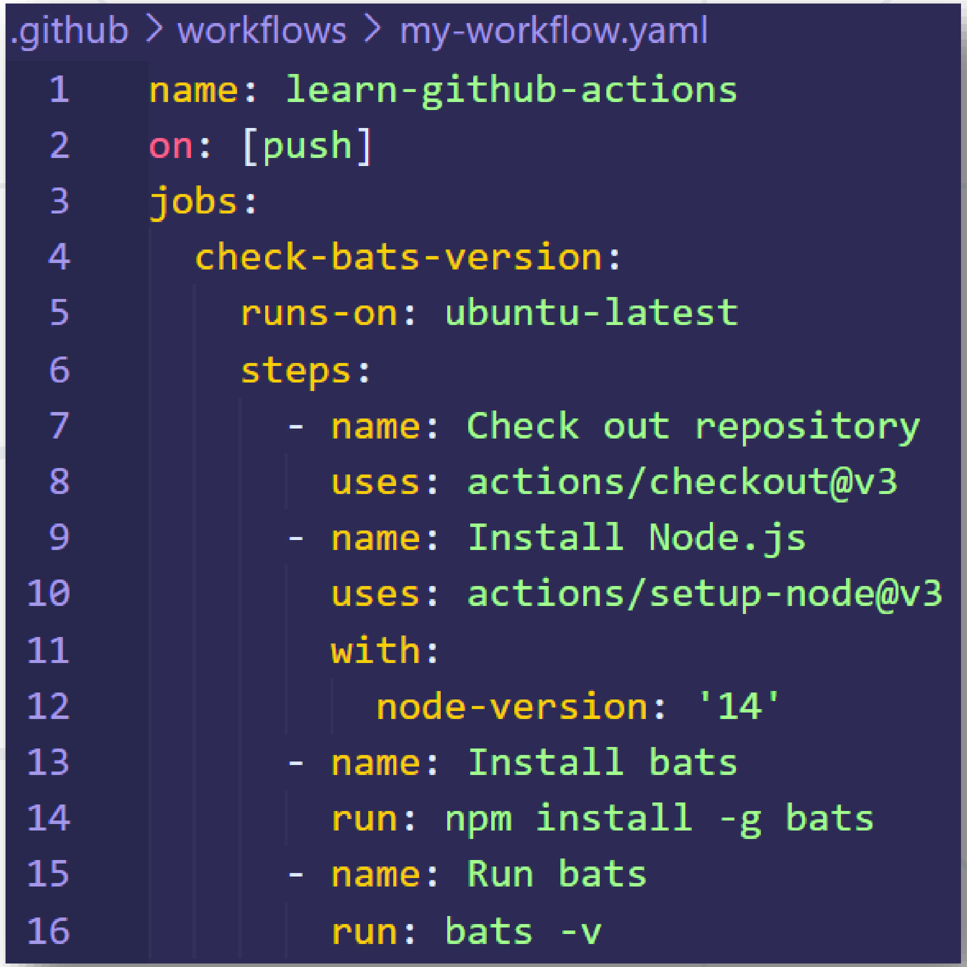
```yaml
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository
        uses: actions/checkout@v3
      - name: Install Node.js ···
      - name: Install bats ···
      - name: Run bats ···
```





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**



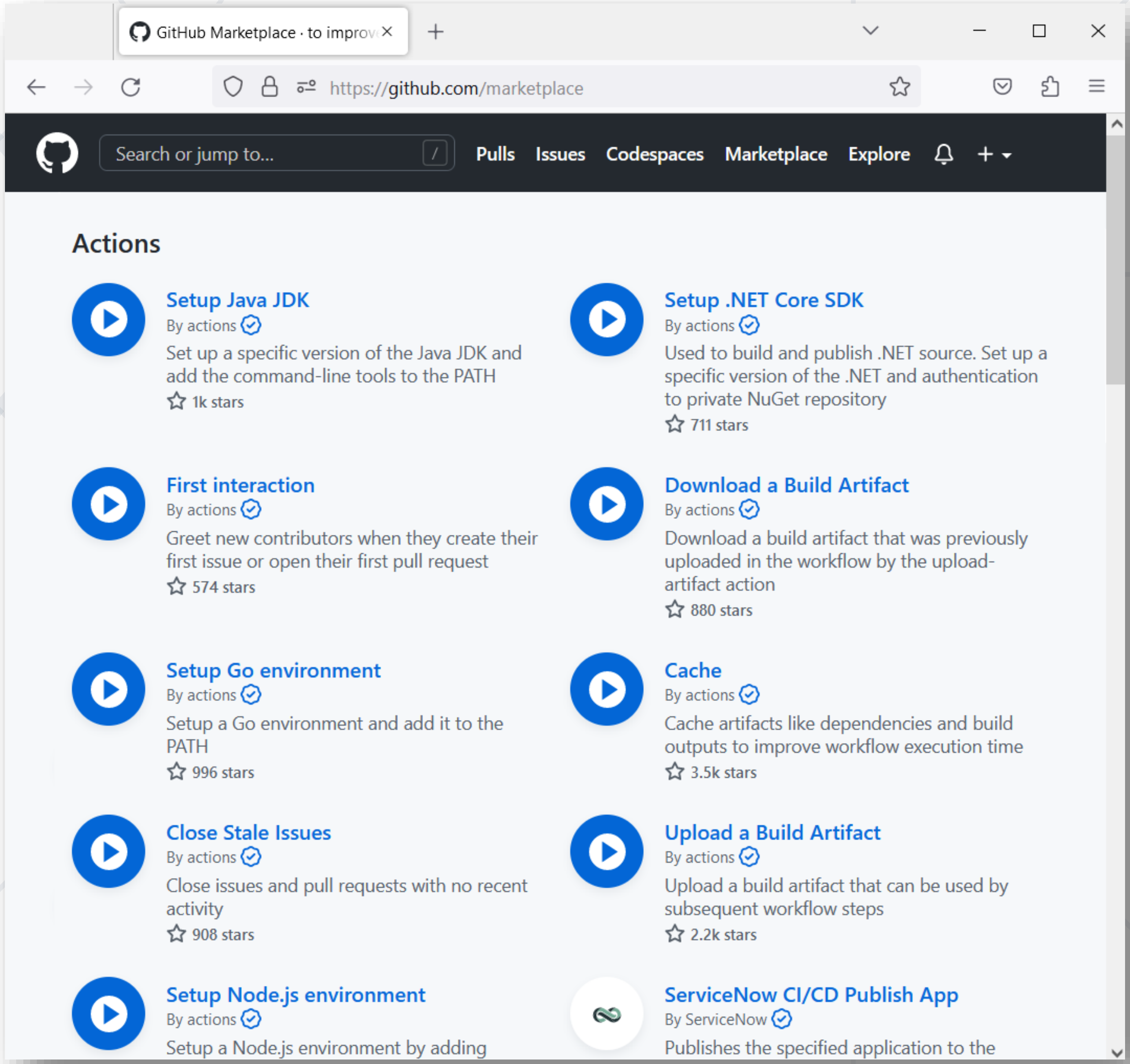
```yaml
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository
        uses: actions/checkout@v3
      - name: Install Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '14'
      - name: 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 **workflow templates** based on code in your repo


GitHub Marketplace · to improv
https://github.com/marketplace
Search or jump to...
Pulls
Issues
Codespaces
Marketplace
Explore

Actions

Setup Java JDK
By actions
Set up a specific version of the Java JDK and add the command-line tools to the PATH
1k stars

Setup .NET Core SDK
By actions
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository
711 stars

First interaction
By actions
Greet new contributors when they create their first issue or open their first pull request
574 stars

Download a Build Artifact
By actions
Download a build artifact that was previously uploaded in the workflow by the upload-artifact action
880 stars

Setup Go environment
By actions
Setup a Go environment and add it to the PATH
996 stars

Cache
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time
3.5k stars

Close Stale Issues
By actions
Close issues and pull requests with no recent activity
908 stars

Upload a Build Artifact
By actions
Upload a build artifact that can be used by subsequent workflow steps
2.2k stars

Setup Node.js environment
By actions
Setup a Node.js environment by adding

ServiceNow CI/CD Publish App
By ServiceNow
Publishes the specified application to the

Jenkins

Jenkins

- Jenkins
  - Open-source automation server
- Used for facilitating CI/CD
- Supports various platforms and languages
- Large ecosystem of plugins
  - Allows users to integrate it with various tools and technologies
- Simplifies CI/CD pipeline

Key Features and Benefits

- **Web-based interface**
  - Easier configuration and management of CI/CD
  - No need for extensive scripting
- Extensible
  - Through its plugin architecture
    - Providing a wide range of options for task completion
- Supports **distributed** builds
  - Allows **multiple** build agents to work in parallel
  - Optimizes resource utilization
  - Speeds up development process

Jenkinsfile Pipelines

- **Set of plugins** that support the integration and implementation of CD pipelines
- Provides a **domain-specific language** (DSL) for **defining steps** involved in the software delivery process
  - Automates the entire process of software delivery
- Ensures that software is **always** in a **releasable state** through its lifecycle

Pipeline Components

- **Stages**
  - High-level phases that organize the main activities in a pipeline
    - Build, test, deploy, etc.
- **Steps**
  - Concrete tasks within each stage
- **Nodes**
  - Define the system or agent where the pipeline or a specific stage will run
- **Agents**
  - Direct the pipeline where to run

Pipeline as Code

- Practice that treats the continuous integration, continuous delivery and continuous deployment as **part** of the **application code**
- Enables **collaboration** on design and changes
- Facilitates tracking **changes** and reviewing previous versions
- Improves transparency
  - All team members can see the pipeline's **logic** and understand the delivery **process**

Jenkinsfile

- Core component representing the "**Pipeline as Code**" philosophy
- Defines the **pipeline configuration as code**
- Outlines the **stages**, **steps** and **actions** that Jenkins will execute during the build, test and deploy processes
- Usually, placed at the **root** of the project repository
  - Allows revision and versioning
- Two main **types** of syntax, written in **Groovy** (optionally typed and dynamic language)
  - Choice between the two types depends on project's complexity and team's preferences

Declarative Syntax

- **Newer** and **simplified** way of defining the pipelines
- Aims to provide more readable way to define pipeline configuration
  - Easy to read and write
- **Pre-defined** structure

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
            // Commands to build
        }
    }
    post {
        always {
            // actions to perform after 
the pipeline runs
        }
    }
}
```

Scripted Syntax

- Traditional way of scripting the Jenkinsfile
- Based on Groovy
  - Provides **more flexibility** and control
- Complete control over the script
- Allows **more complex logic**

```
node {
    stage('Build') {
        // Commands to build
    }
    stage(Test') {
        // Commands to build
    }
    stage('Deploy') {
        // Commands to build
    }
}
```

Events

- Start a Jenkins **job** or **pipeline**
- Executed by external **triggers**
  - Source code changes
    - Commit or merge to a version control system, e.g., Git
  - Manual initiation
    - Started through the Jenkins UI
  - Upstream or downstream triggers
    - Completion of another job
  - Scheduled event

Workflows

- High-level **definition** of the **entire process** for deployment
- Described in a **Jenkinsfile**
  - Defines one or more **pipeline** jobs
  - Stored in source control
    - Enables versioning and review
- Supports complex logic
  - Conditional execution
  - Parallel steps
  - Etc.

Jobs

- Runnable **tasks** in Jenkins
  - Basic unit of functionality
  - Defined in a pipeline
  - Can include stages
- Accept various **parameters** in order to modify the build process
- Store **artifacts** (binaries, reports, etc.) and record build **results**

Steps

- **Individual tasks** within a Jenkins job
- **Command** or a series of **commands**
- In declarative syntax
  - Script commands
    - Shell scripts or batch commands
  - Tool invocation
  - File operations

Actions

- **Operations** that are performed by steps
- Actual command executions or function calls that
  - Interact with the workspace
  - Modify the build state
  - Send notifications

Jenkins Pipeline Syntax Keywords

- `pipeline`
  - Defines the block where the pipeline process is described
- `agent`
  - Specifies where the entire pipeline or a specific stage will execute in the Jenkins environment
- `stages`
  - Sequence of one or more stages that are to be executed in a defined order
- `stage`
  - Defines a conceptually distinct subset of tasks performed through the entire pipeline

Jenkins Pipeline Syntax Keywords

- `steps`
  - Defines a series of one or more steps to be executed in a given stage
- `script`
  - Allows for the inclusion of arbitrary Groovy code to be executed
- `environment`
  - Defines a set of environment variables for the steps to use
- `post`
  - Determines one or more additional steps that are run upon the completion of the pipeline's or stage's execution

Jenkins Architecture

- Jenkins follows a distributed architecture
- **Main component → controller**
  - Responsible for scheduling jobs, dispatching builds to nodes (agents) and monitoring them
- **Distributed nature**
  - Jenkins can run jobs on different machines (**nodes** or **agents**)
    - Allows scaling as the workload increases

Controller/Agent Model

- **Controller**
  - Manages entire Jenkins environment
  - Previously known as **master**
- **Agents**
  - Machines or virtual instances that execute the jobs, dispatched by the controller
  - Allow builds and test to run in different environment
- **Distributed builds**
  - Multiple agents can run concurrently
    - Optimizes the utilization of resources

Scalability, Load Balancing and Security

- Jenkins scales **horizontally** by adding more agents
- Automatically **distributes jobs** among **available agents** based on their configurations and capabilities
- Supports various **authentication mechanisms**
- Communication between controller and agents can be **encrypted**
  - Ensures code and build results are securely transmitted

Plugin Architecture

- **Plugins** == **primary** method extending Jenkins
  - Thousands of plugins available in the **ecosystem**
- Plugin architecture makes Jenkins highly **extensible** and **customizable**
  - Plugins can be chosen based on the user's specific requirements
- Allows for a lightweight and lean core with ability to expand capabilities if needed
  - Helps Jenkins evolve with the changing technology

Summary

- **CI/CD** == a method to **frequently deliver apps** by introducing **automation** into **continuous integration**, **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**
  - **Jenkins**, which is an **open-source server,** that simplifies the CI/CD pipelines

Questions?

Software University

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

createX

VIVACOM

INDEAVR
Serving the high achievers

Coca-Cola HBC
Bulgaria

Postbank
Решения за твоето утре

AMBITIONED

SUPER
HOSTING
.BG

DRAFT
KINGS
THE CROWN IS YOURS

PHAR
VISION

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