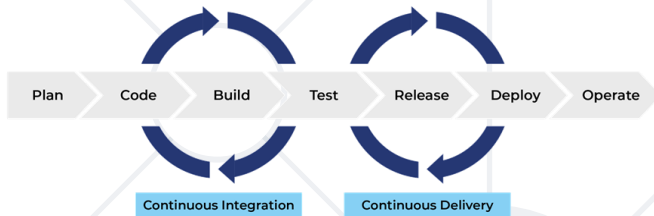


Continuous Integration (CI)

Continuous Integration, GitHub Actions, Jenkins



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

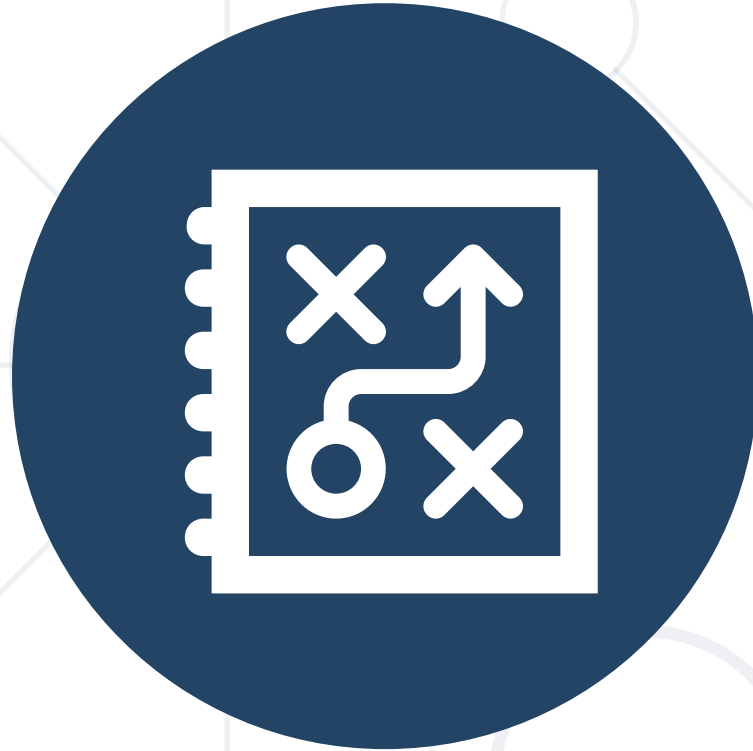
sli.do

#QA-Auto-FrontEnd

Table of Contents

1. The CI/CD Pipelines
2. GitHub Actions
3. Jenkins





The CI/CD Pipelines

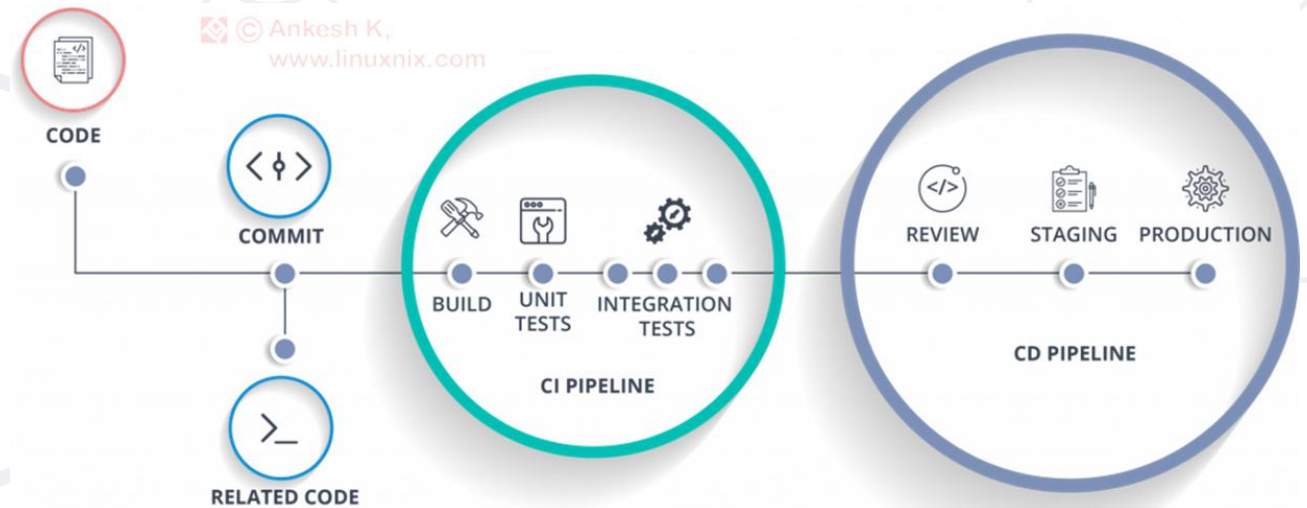
What is CI/CD?

- **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 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



- 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)

- 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)

- 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 **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)

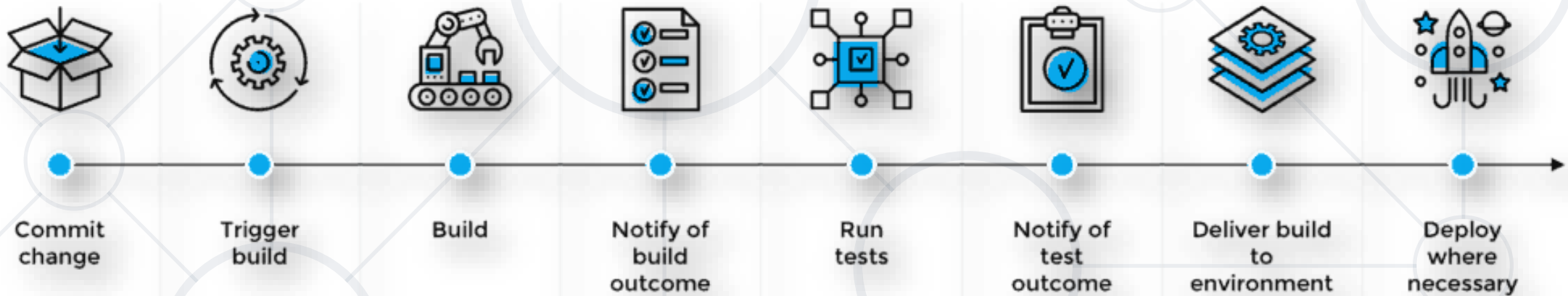
Continuous Delivery vs. Continuous Deployment



- **CI/CD 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 Pipeline



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

- 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**

- 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

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

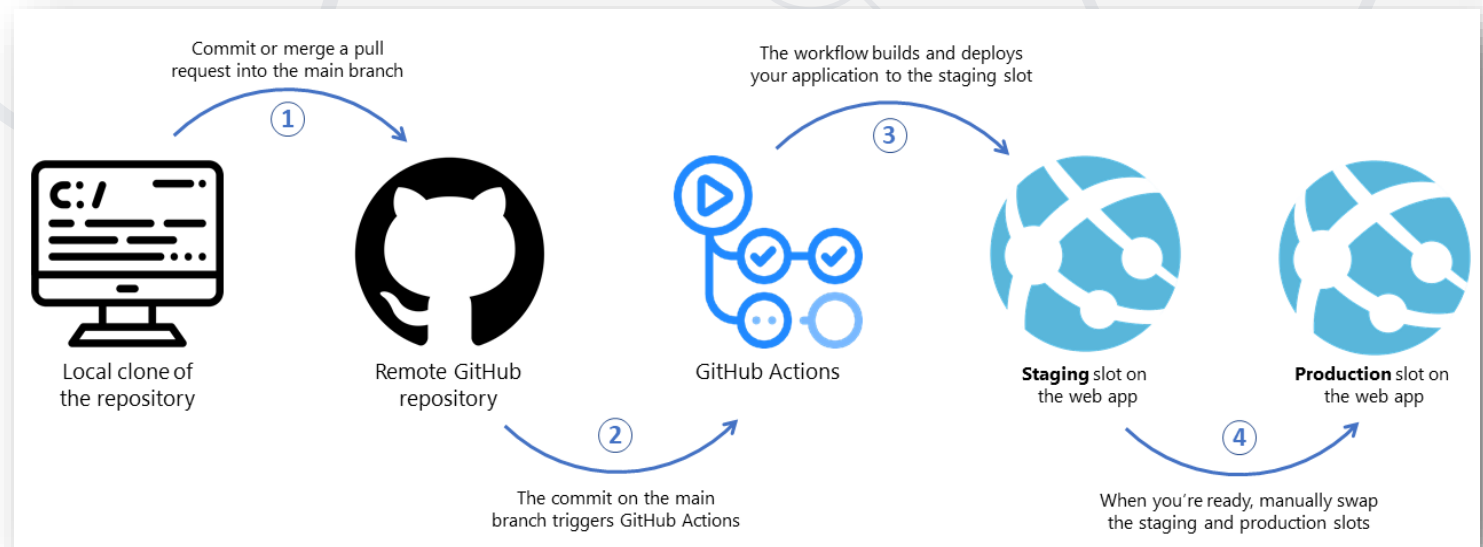
- GitHub Actions

- 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**

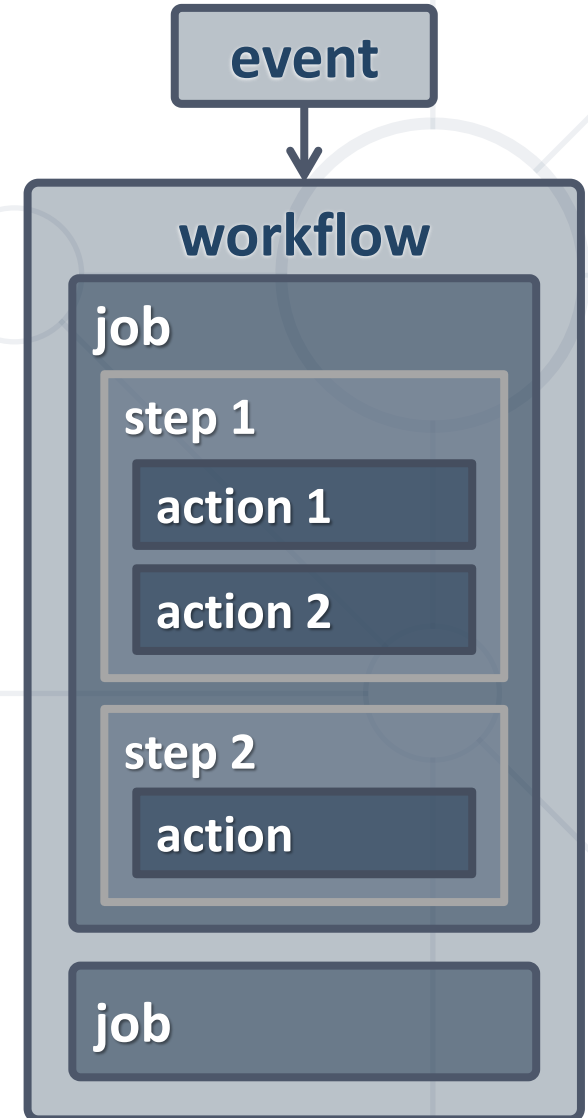


- Flexible environment
 - Supports **various** programming languages
- Allows developers to **trigger workflows**
 - Based on events like code commits, pull requests, issue updates
- Allows defining **custom** workflows
- YAML syntax
- Large **library** of pre-built actions and custom workflows

- 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**



- **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`**)



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

- **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

- **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
1  name: learn-github-actions
2  on: [push]
3  jobs:
4    check-bats-version:
5      runs-on: ubuntu-latest
6      steps:
7 >   - name: Check out repository ...
9 >   - name: Install Node.js ...
13 >  - name: Install bats ...
15 >  - name: Run bats ...
```



TRIGGER



SCHEDULED

TRIGGER



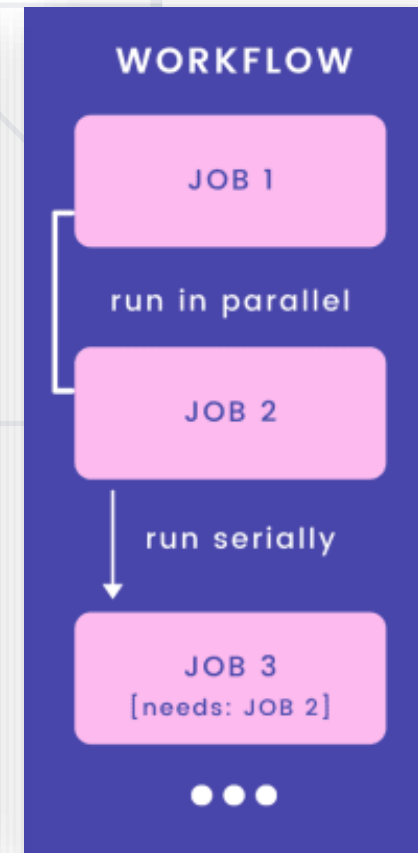
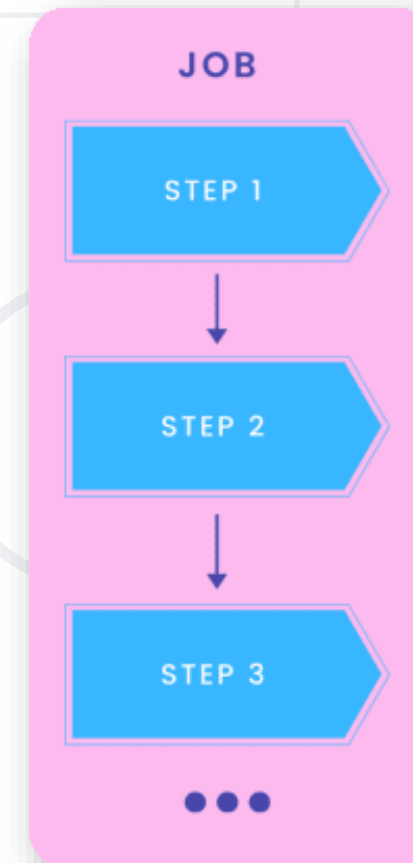
MANUALLY

TRIGGER



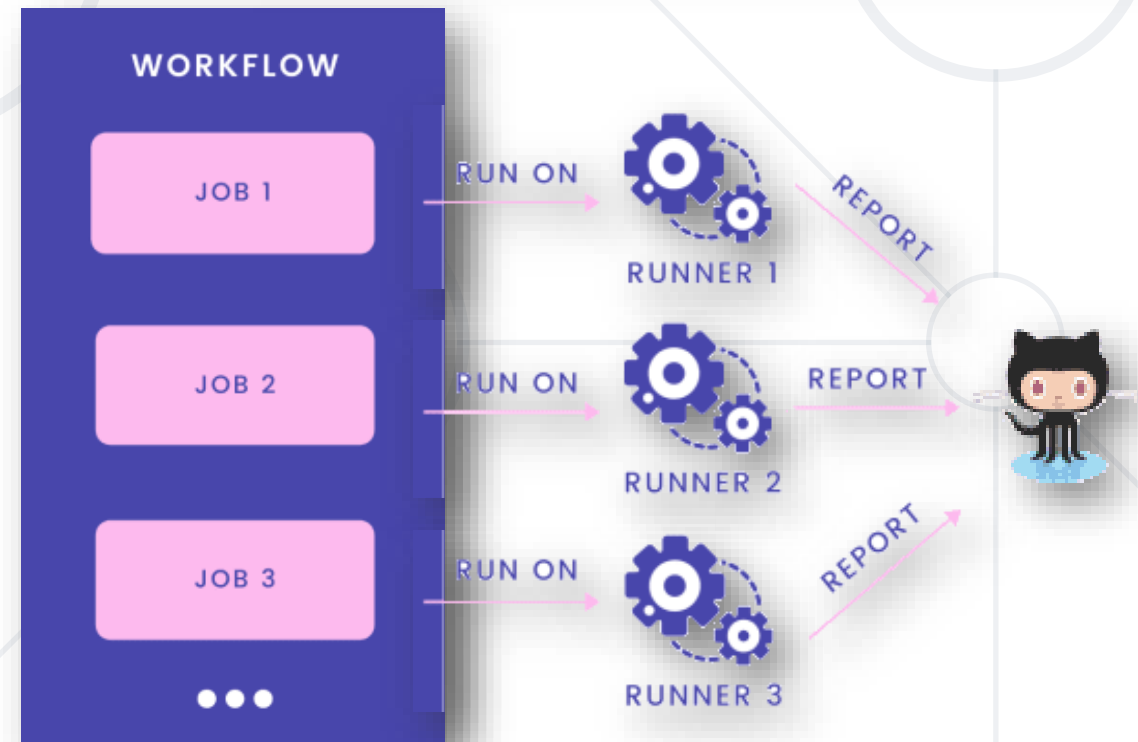
- **Job** == a **set of steps** that will be executed on **the same runner**
- All **jobs in the workflow** normally run in **parallel**
- When you have **jobs that depend on each other**, they run **serially**

```
github > workflows > my-workflow.yaml
1  name: learn-github-actions
2  on: [push]
3  jobs:
4    check-bats-version:
5      runs-on: ubuntu-latest
6      steps:
7 >     - name: Check out repository ...
9 >     - name: Install Node.js ...
13 >    - name: Install bats ...
15 >    - name: Run bats ...
```



- 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**

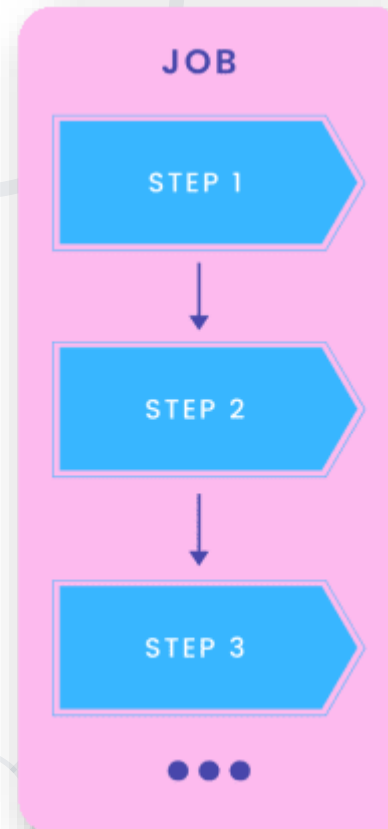
```
3 jobs:  
4   check-bats-version:  
5     runs-on: ubuntu-latest
```



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

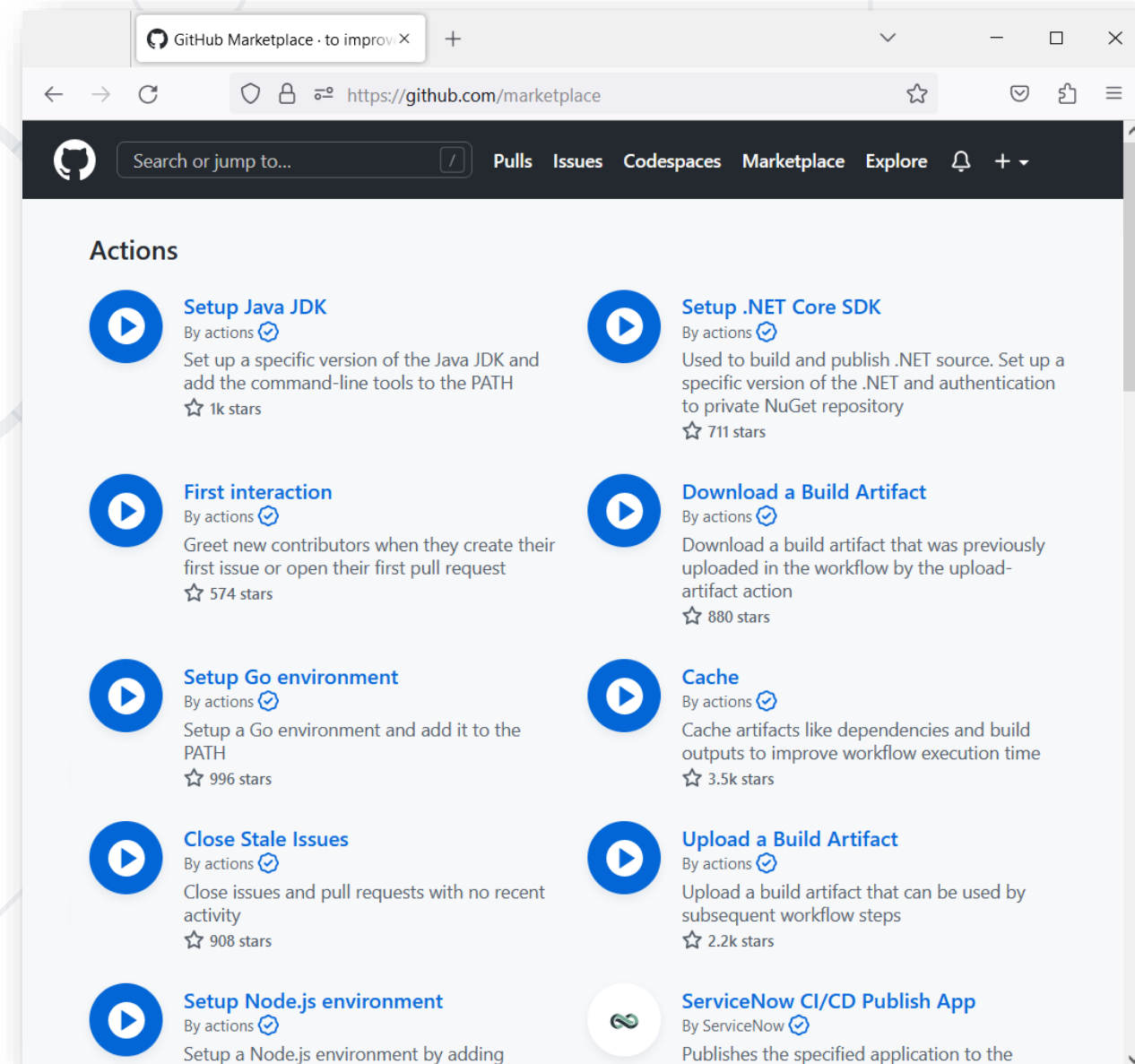
- **name**
 - for **names** of workflows, steps, which GitHub Action displays
- **on**
 - used to define which events can cause the workflow to run (**triggers**)
- **jobs**
 - used to **list jobs**
- **runs-on**
 - specify **runner environment**

```
.github > workflows > my-workflow.yaml
1  name: learn-github-actions
2  on: [push]
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
10               uses: actions/setup-node@v3
11               with:
12                   node-version: '14'
13              - name: Install bats
14                run: npm install -g bats
15              - name: Run bats
16                run: bats -v
```

- **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





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

- **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

- **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

- 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**

- 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

- **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  
    }  
  }  
}
```

- 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  
    }  
}
```


- 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

- 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.

- 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**

- **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

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

- **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

- **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 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**
 - 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

- 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

- **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

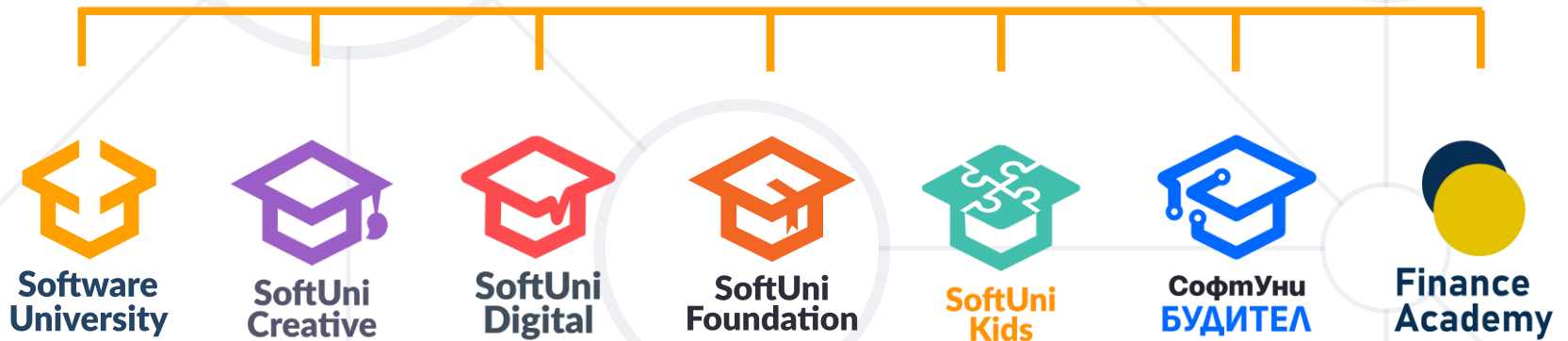
- **CI/CD** == a method to **frequently deliver apps** by introducing **automation** into **continuous integration, continuous delivery** and **continuous deployment**
- There are a lot of **CI/CD platforms**
 - **GitHub Actions**, in which you can create **workflows** to **automate** your **build, test** and **deployment pipeline**
 - **Jenkins**, which is an **open-source server**, that simplifies the CI/CD pipelines



Questions?



SoftUni



Diamond Partners



THE CROWN IS YOURS



- 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



- 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>

