DevOps Fundamentals

Continuous Integration

Agenda

Continuous Integration

Pipeline As Code

GitHub Actions

Continuous Integration

What happens without Continuous Integration

Long Development Cycles

- Non-compiling code. Code may not be functional

High Build Failures/Bug Count

- Long living branches causing bigger merge effort
- Code missing from source control
- Security flaws found late
- Overall reduced quality

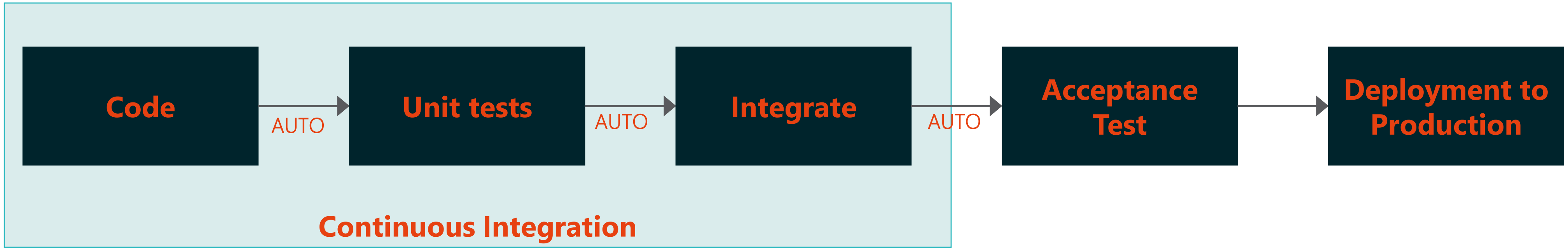
Less communication and collaboration

- Not following standards (code, docs, etc.)
- Few (or no) code reviews
- Testing done late (if done…) and mostly manual

What is Continuous Integration

CI is a mindset, team strategy and a capability. Is not a tool!

"... a **software development practice** where members of a team **integrate their work frequently**, usually each person integrates **at least daily** – leading to multiple integrations per day. Each integration is **verified by an automated build** (including test) to detect integration errors as quickly as possible. Many teams find that this **approach leads to significantly reduced integration problems** and allows a team to **develop cohesive software more rapidly**." (Martin Fowler)



Continuous Integration: Goals

1) Leverage team collaboration
2) Enable parallel development
3) Minimize integration debt
4) Act as a quality gate
5) Automate everything!

Continuous Integration: Main elements

Version Control

- Source code, configurations, scripts
- Infra, pipelines, everything as code!

Branching Strategy

- First, have one ☺
- Work with small batches
- Branch often, commit always

Automated Build

- Runs after every commit/merge
- Shift-left Testing
- Must run fast!

Continuous Integration: Are your team doing it?

Are all the developers on the team checking into trunk at least once a day? (**Trunk-based development** and **working in small batches**)

Does every change to trunk kick off a build process, including running a set of automated tests to detect regressions? (**Automated builds**)

When the build and test process fail, does the team fix the build within a few minutes, either by fixing the breakage or by reverting the change that caused the build to break? (**Focus on DevOps Metrics** [Deployment frequency, MTTR, Lead time for change])

Reference: Humble, Jez. Farley, David. (2010) Continuous delivery: Reliable software releases through build, test, and deployment automation

Continuous Integration: Practices

Mono-repos vs Multi-repos

All your code on a single repo or each component (service, microservice) on its own repo

Mono-repo: Improved Developer Testing, Reduce Code Complexity, Effective Code Reviews, Share of Common Components, Easy Refactoring

Multi-repo: Clear Ownership, Improved Scale

Branching Strategies

KISS, keep it stupid simple

Main/trunk is always buildable and (production) deployable

Keep feature/topic branches as short lived as possible

Merge to main/trunk as fast as possible

Continuous Integration: Practices

Branching policies

- Require a minimum number of reviewers on Git pull requests
- Check for linked work items
- Check for comment resolution
- Enforce a merge strategy
- Build validation
- Automatically include specific code reviewers

Git pull requests

- Promote code reviews by your peers
- Allow to define the flow of changes
- With automatics validation allow to increase (a lot) quality

Continuous Integration: Practices

Automated Builds

The way you build your product is consistent
Any change in the way you build is tracked
The build results are triaged for quality
Linking artifacts to build events for traceability
The code is valid before merging into main

Automated Builds: Common Activities

Coding Standards Checking

Download Dependent Packages

Build Code

Unit Testing

Code Coverage Analysis

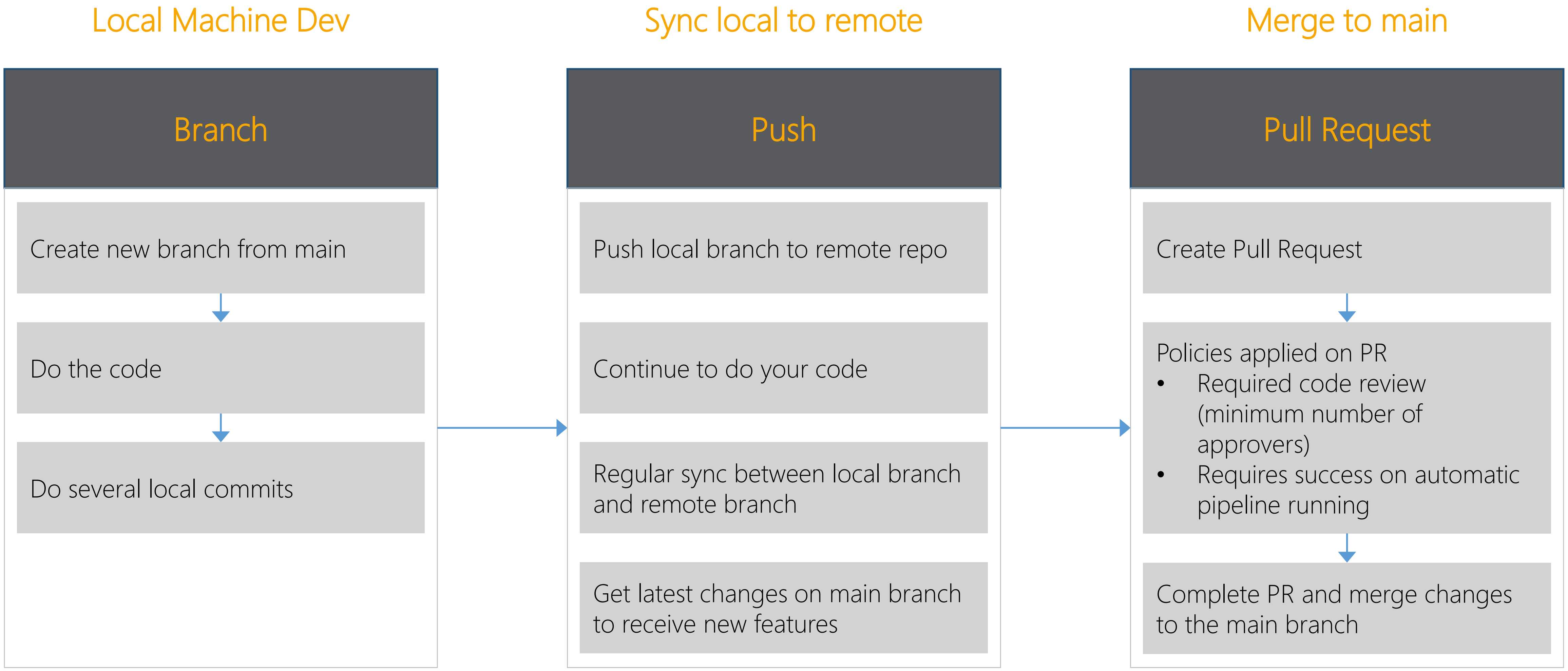
CredScan

Static Code Analysis

Open Source Component Scan

Create Deployable Package

Continuous Integration: E2E process



Local Machine Dev

Branch

Create new branch from main

Do the code

Do several local commits

Sync local to remote

Push

Push local branch to remote repo

Continue to do your code

Regular sync between local branch and remote branch

Get latest changes on main branch to receive new features

Merge to main

Pull Request

Create Pull Request

Policies applied on PR

- Required code review (minimum number of approvers)
- Requires success on automatic pipeline running

Complete PR and merge changes to the main branch

Pipeline As Code

Pipeline As Code

Practice of defining deployment pipelines through source code

Pipeline as code is part of a larger “as code” movement that includes infrastructure as code, docs as code, etc.

Teams can configure builds, tests, and deployment in code that is trackable and stored in a centralized source repository.

Use a declarative language, like YAML (more open), or vendor-specific language (such as Jenkins and Groovy)

Tries to remove the challenges that UI interfaces bring like limited auditing, hard to versioning, difficult to rollback and prone to break on code changes

Pipeline As Code: Benefits

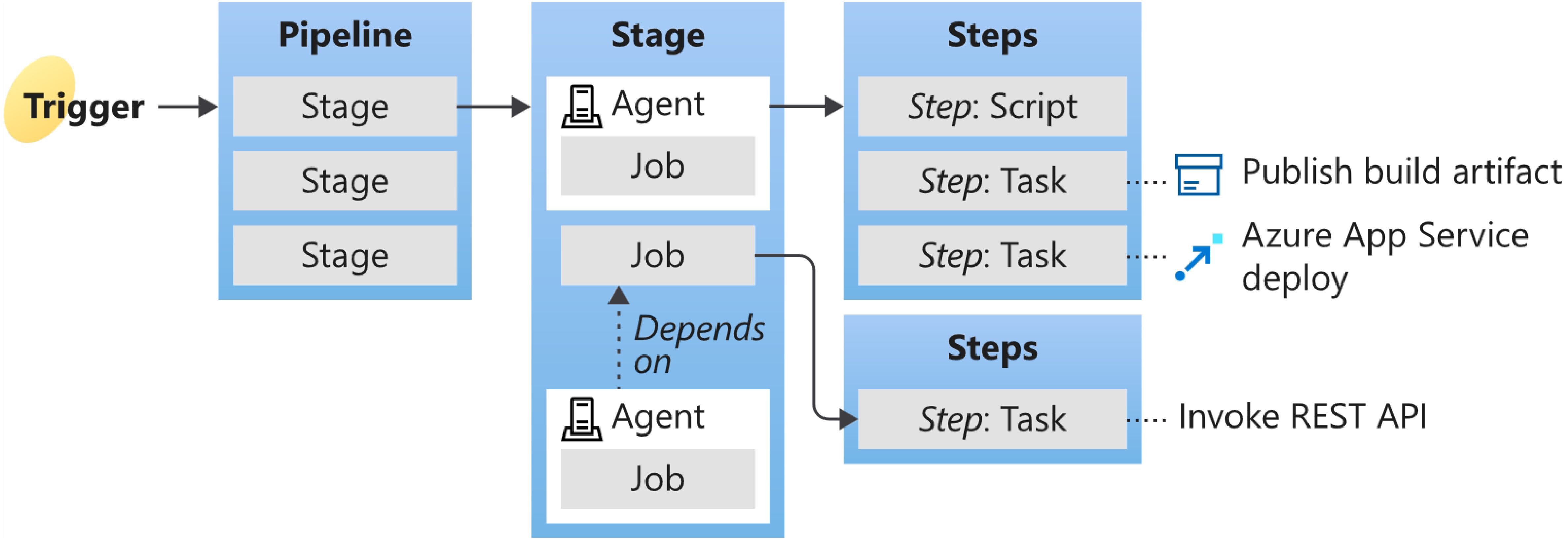
CI pipelines and application code are stored in the same source repository. All the information teams need is located in the same place.

Developers can make changes without additional permissions and can work in the tools they're already using.

Teams can collaborate more efficiently. Keeping information accessible means teams can collaborate and then act on their decisions.

Pipeline changes go through a code review process, avoiding any break in the pipeline integration.

Pipeline As Code: Basic (common) concepts



Pipeline As Code: Basic (common) concepts

A **trigger** tells a Pipeline to run.

A **pipeline** is made up of **one or more stages**

A **stage** is a **way of organizing jobs** in a pipeline and each stage can have one or more jobs.

Each **job runs on one agent**. Each agent runs a job that contains one or more steps.

A **step** can be a **task or script** and is the smallest building block of a pipeline.

An **artifact** is a collection of files or packages **published by a run**.

GitHub Actions

GitHub Actions

Automate from code to cloud

Flexible automation triggered by events

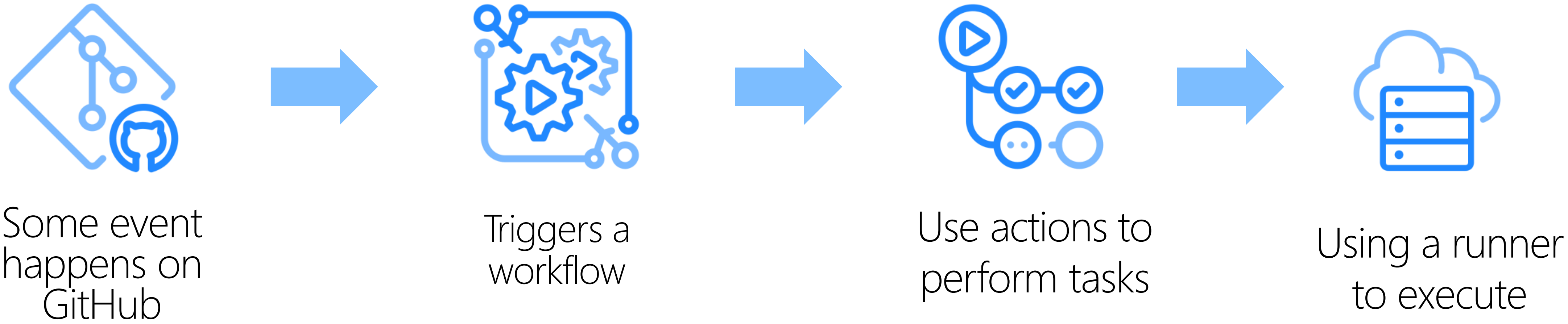
Powerful CI/CD

Any OS, any language and any cloud

Secure and automated workflows at scale

Faster innovation

GitHub Actions: Complete Flow



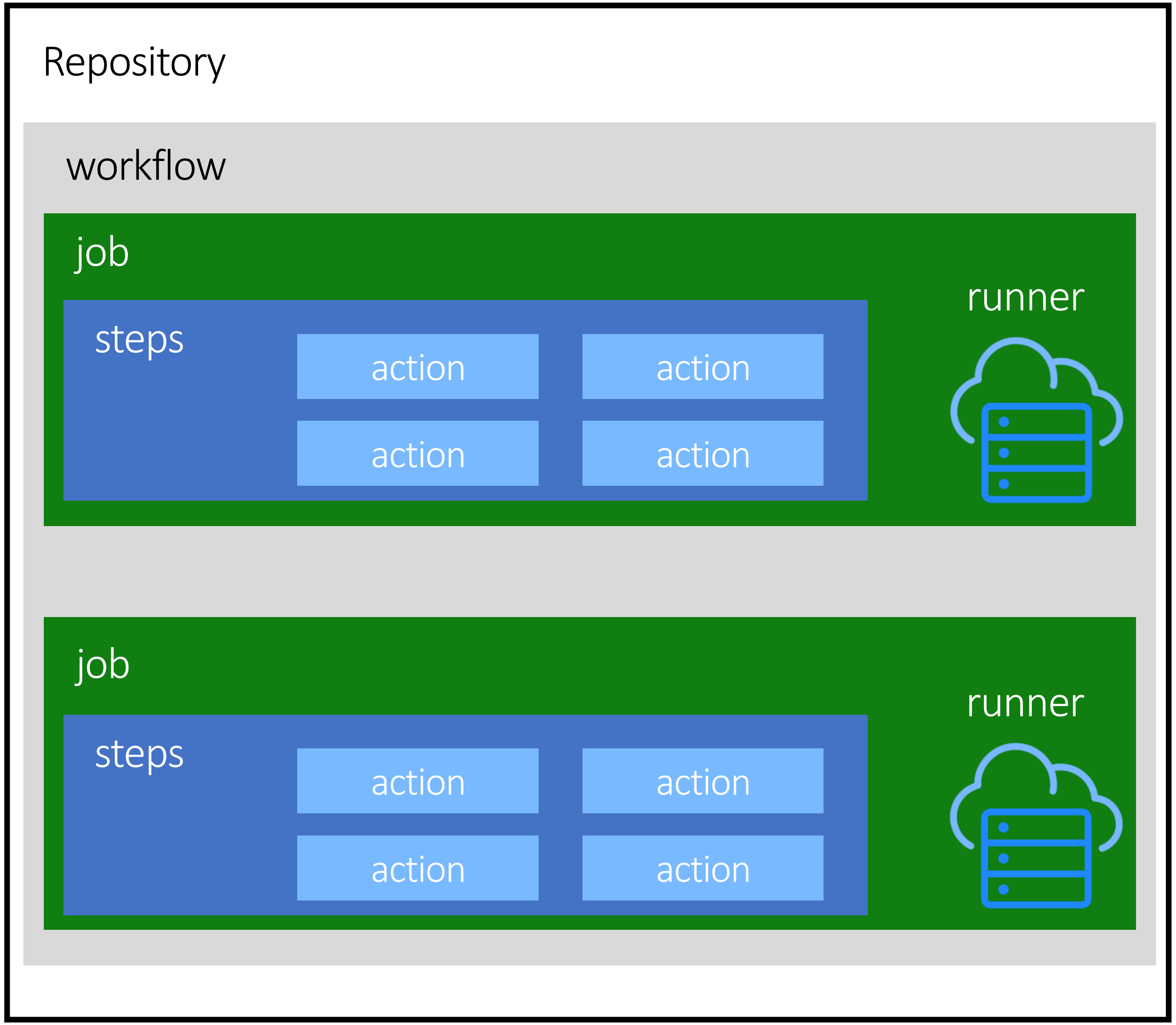
Some event happens on GitHub

Triggers a workflow

Use actions to perform tasks

Using a runner to execute

GitHub Actions: Composition



GitHub Actions: Triggers

Events that occur in your workflow's repository

Events to explicitly trigger an workflow from other workflow

Scheduled times

Manual

GitHub Actions: Workflow

Automated process available on your repository

Defined in yaml on .github/workflows folder (filename don’t have a rule)

Versioned as code

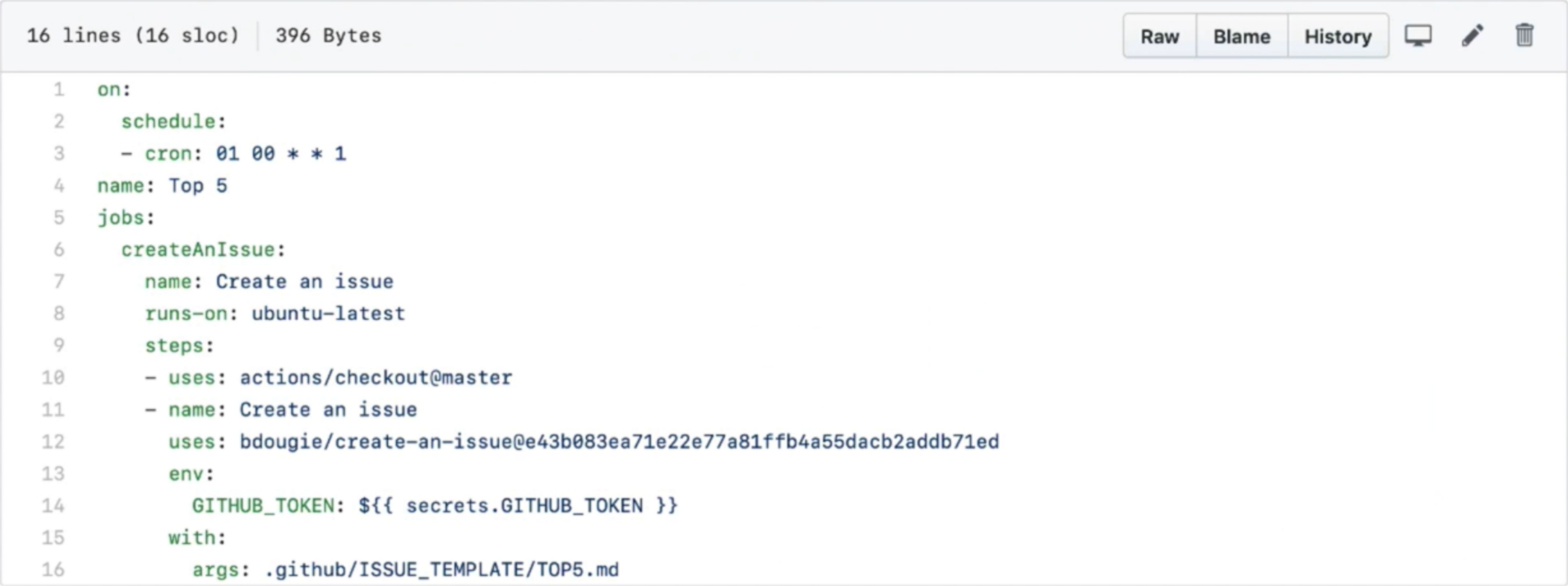
Triggered by any event

- Push, Pull request, Label on PR, …
- New issue, new comment, …

Different triggers makes GitHub Actions to be much more than a CI automation tool

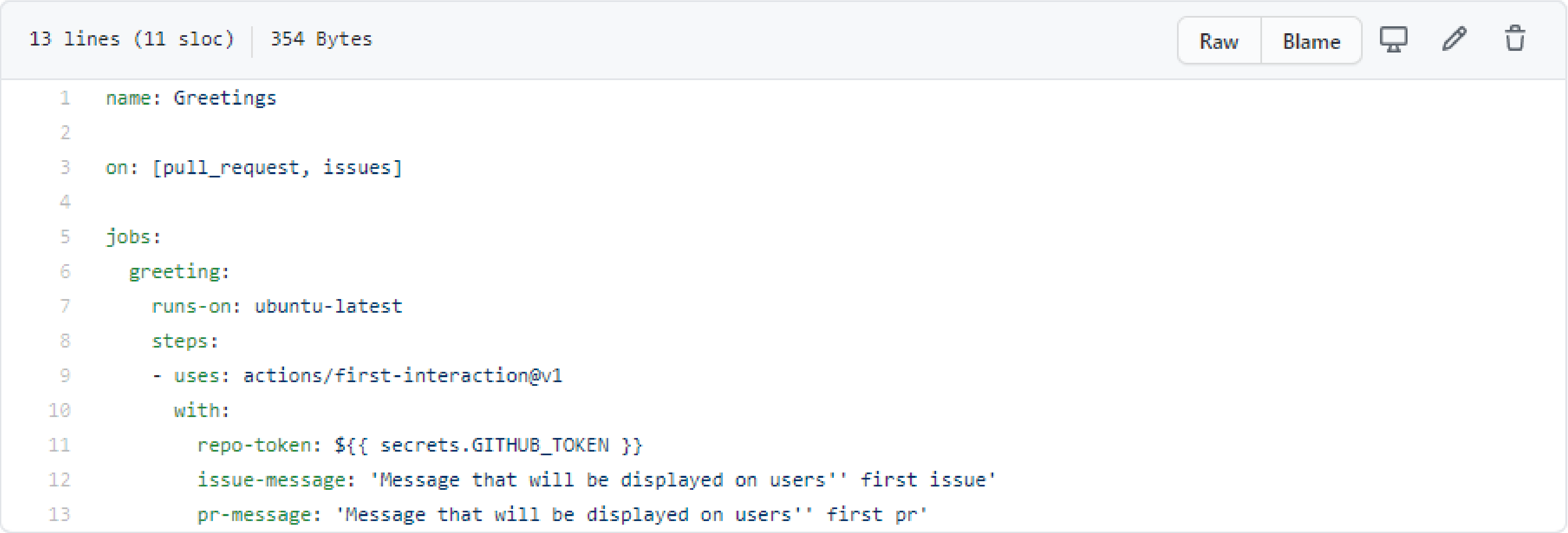
- Can create a “bot” behavior on your issues or PRs

Sample Workflow: Create Issue on Schedule



```yaml
on:
  schedule:
  - cron: 01 00 * * 1
name: Top 5
jobs:
  createAnIssue:
    name: Create an issue
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@master
    - name: Create an issue
      uses: bdougie/create-an-issue@e43b083ea71e22e77a81ffb4a55dacb2addb71ed
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      with:
        args: .github/ISSUE_TEMPLATE/TOP5.md
```

Sample Workflow: Greets user on his first comment



13 lines (11 sloc) | 354 Bytes

```yaml
name: Greetings

on: [pull_request, issues]

jobs:
  greeting:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/first-interaction@v1
      with:
        repo-token: ${{ secrets.GITHUB_TOKEN }}
        issue-message: 'Message that will be displayed on users'' first issue'
        pr-message: 'Message that will be displayed on users'' first pr'
```

GitHub Actions: Jobs

Defines a list of steps (actions) to be performed in sequence

A job runs on a GitHub Runner

Defines a virtual environment inside the runner

Selects the runner using tags to matching

Sharing between jobs needs to be done explicitly

GitHub Actions: Actions

Unit of work that represents one task to be performed

Can be implemented in JavaScript, using a container (Linux) or a composite (template) action

Marketplace, community or own actions

- actions/checkout
- GitHub Marketplace
- Your action available on your repo

Any action available on a repo can be selected by its version

- Git branch
- Commit SHA
- Git tag (human-readable name for a commit)

Demo: GitHub Marketplace

GitHub Actions: Runners

GitHub Runners available as GitHub-Hosted and self-hosted

GitHub-Hosted Runners

- Available on Windows, Linux and MacOs
- For each run, a new runner is created with clean state
- New versions (at least very week)
- Simplicity and quickness

Self-Hosted Runners

- Can be installed on Windows, Linux MacOS
- Infra and all management is done by “you”
- Control and specific scenarios

GitHub Actions: Additional Features

Matrix builds

Live logs for real-time feedback

Built-in variables, secret store and workflow artifact store

GitHub Packages to share libraries, packages and containers

Demo: GitHub Actions

GitHub Actions: Custom Actions

Uses custom code that interacts with your repository

Can interact with GitHub or third-party APIs

Can be used on your workflow or shared with GitHub community

Three types: JavaScript, Docker container or Composite

Needs to be available on public repos for community sharing

Inside an organization can be use internal repos, making available for everyone inside your organization

Action to be used on your workflow can be placed together your software code

Recommended folder: `.github/actions`

Custom Actions: JavaScript/TypeScript

Runs on Windows, Linux or MacOs runners

Must use only pure JavaScript to ensure compatibility with all runners

GitHub Actions Toolkit (actions/toolkit) can be used to speed up development

Custom Actions: Docker

Allows to use specific versions of operating system, dependencies, tools and code

Need to write your code and Dockerfile

Slower than JavaScript actions due to latency to build and retrieve container

Can only be executed on Linux runners (GitHub or self-hosted)

Custom Actions: Composite

Define actions templates to be reused on several workflows

A composite action is a set of tasks, another actions and scripts

Easier to update if some change is needed

Good approach to enforce the execution of specific actions/scripts (like security related, auditing related, etc)

Enable creation of standard actions to be used for several teams enforcing standards, security concerns and optimization (on creation and running)

Demo: Composite Action

moOngy.
Minds on the move

Rua Sousa Martins, nº 10
1050-218 Lisboa | Portugal