GitHub Actions

Fundamentals

Presented by GitHub Professional Services

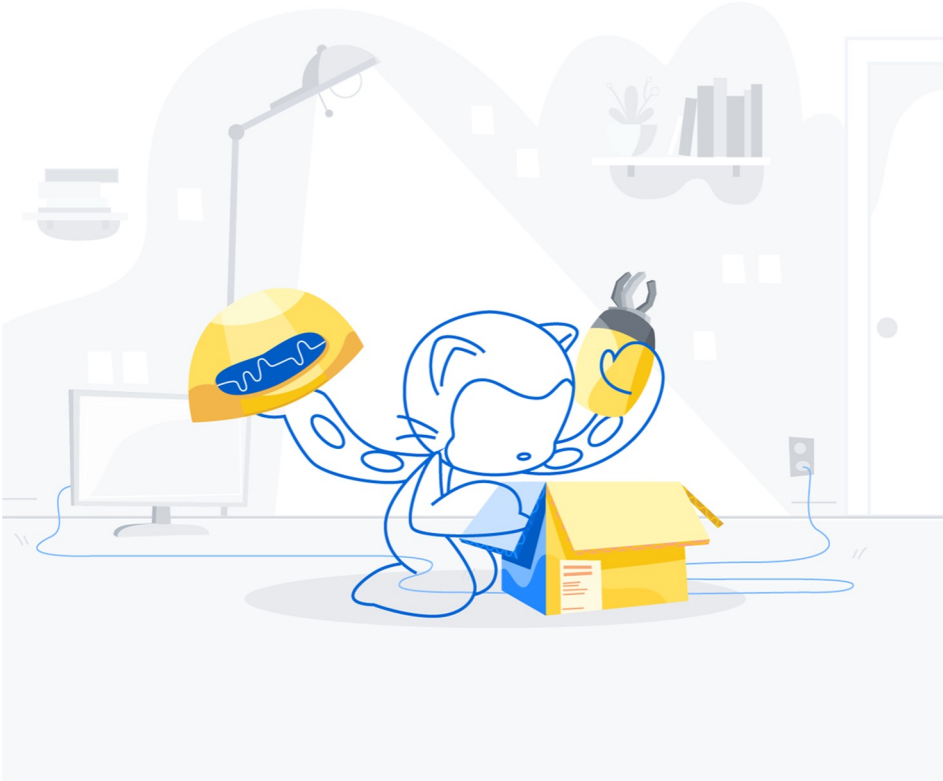
Introduction

Objectives

- Understand the **basic components** and vocabulary of GitHub Actions
- Understand the **YAML** syntax
- Understand the **workflow syntax** and how to write workflows using intellisense
- Understand events that can **trigger** workflows
- Learn the **context and expression syntax** as well as workflow commands
- Know the different types of **actions** and how to create/publish them
- Understand the different hosting options for **runners**
- Use **Secrets, Variables** and **Environments** for staged deployments
- Workflow templates and **reusable workflows**

Agenda – day 1

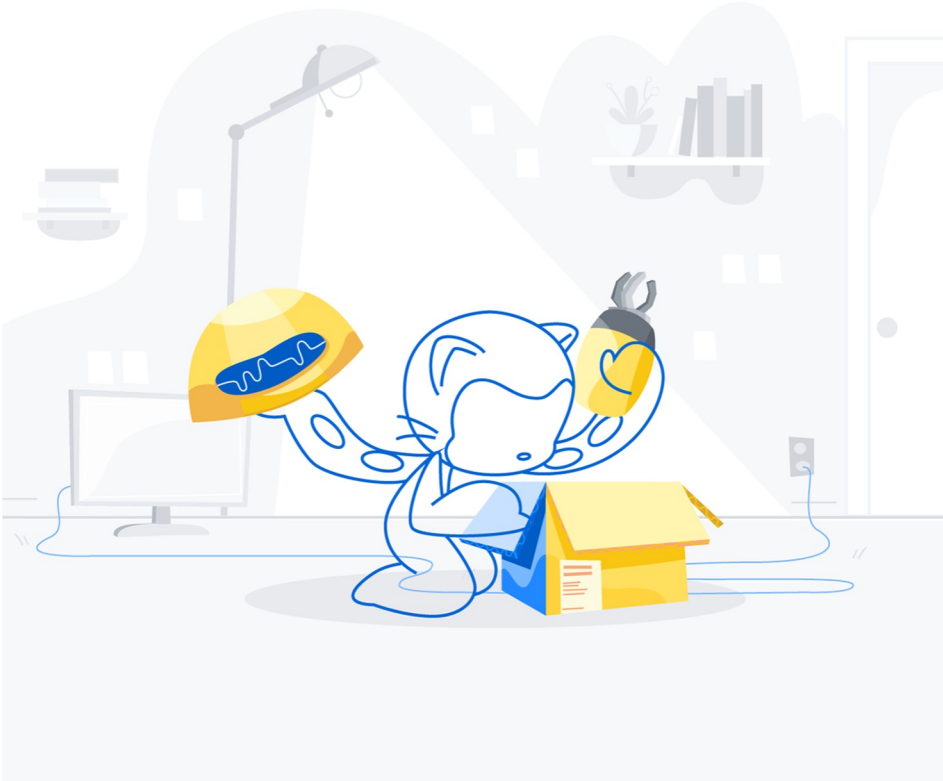
- Workflow fundamentals
- YAML and workflow syntax
- Writing workflows
- GitHub Actions
- Authoring Actions



Agenda – day 2

- CI/CD with GitHub Actions
- Secrets, Variables and Environments

  (staged deployments)
- Running workflows (runners)
- Action policies
- Sharing workflows (reusable workflows)



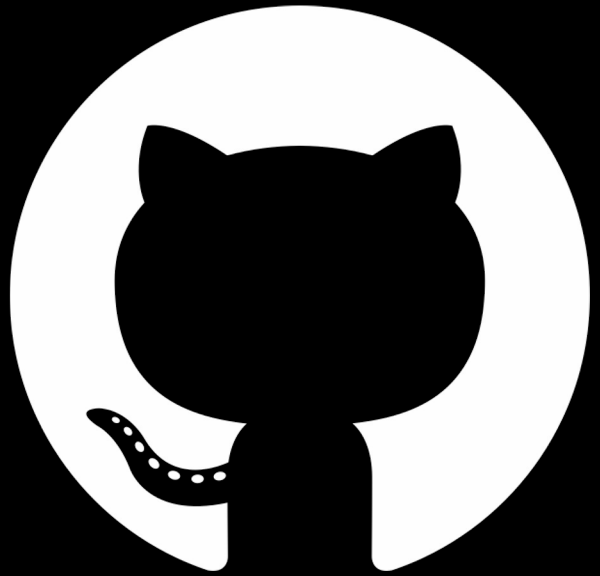
Icebreaker

GitHub Actions Fundamentals

Where the world builds software

100M+
Developers

200M+
Repositories



4M+
Organizations

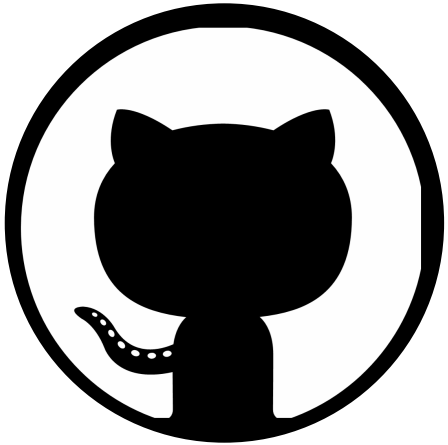
1,000s
Open-Source Communities

2.6B+
Contributions / Year

84%
Fortune 500 companies

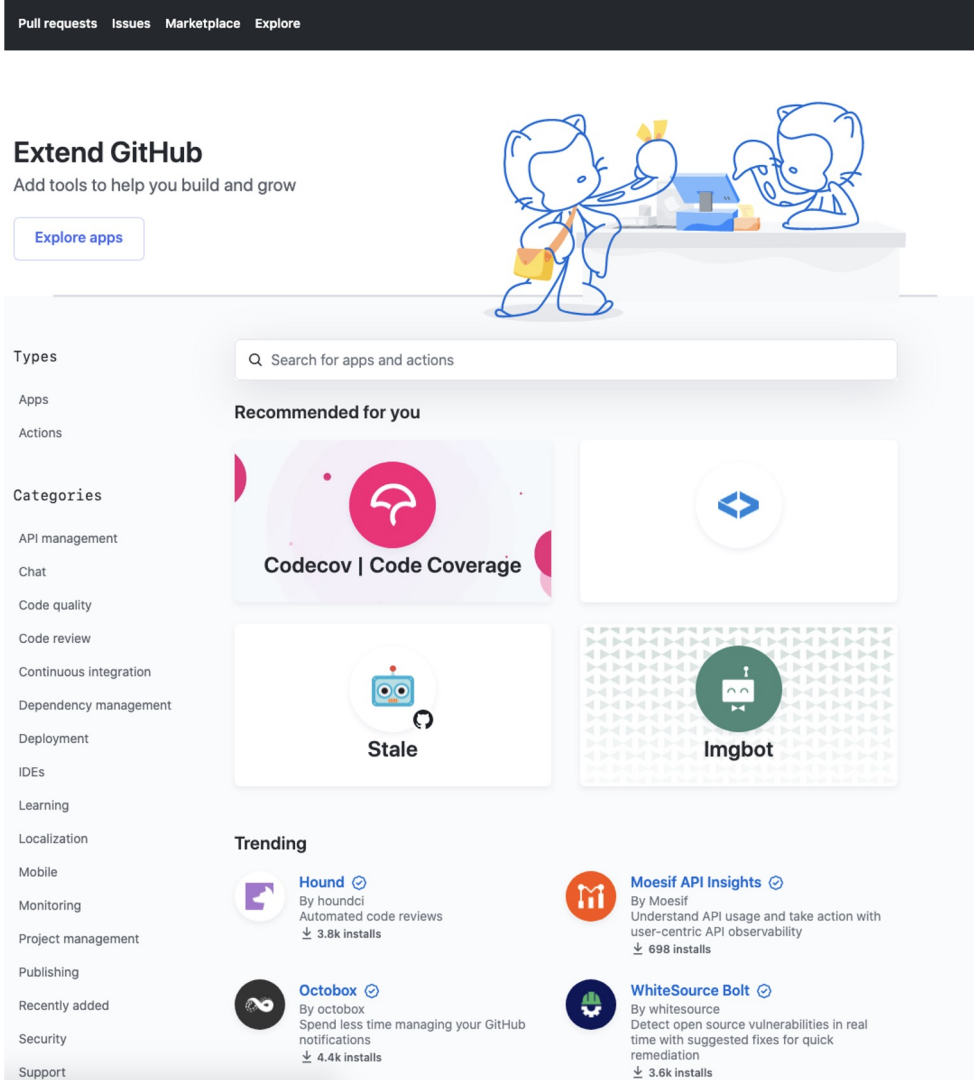
A different approach…

- Leverage the power of the open-source community
- Platform
- Flexible



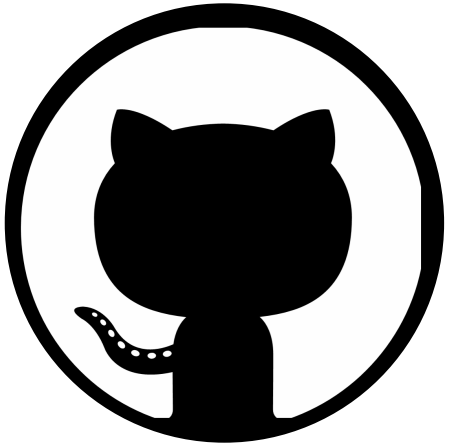
GitHub Marketplace

- Discover open-source Actions across multiple domains
- ~15,000 Actions (and counting...)
- Verified creators
  (*Publisher domain and email verified*)
- Reference these Actions directly in your workflow
- Integrated into the GitHub editor



GitHub Actions – More than CI/CD

- Generic workflow engine
- Automate everything with workflows
- 35 events can trigger a workflow
- GitHub Token and Workflow Permissions
- Community-powered workflows
- Any platform, any language, any cloud



Workflow Fundamentals

- A text file in your repository (`.github/workflows`)
- YAML Ain't Markup Language (**YAML**)
- Events trigger workflows (`on:`)
- One or multiple jobs
- Executed on a runner
- Contains steps
- A reusable step is action

AccelerateDevOps / .github / workflows / pr-validation.yml in main

Edit file | Preview changes | Spaces | 2 | No wrap

```yaml
name: PR-Validation

on: [pull_request]

jobs:
  Build:
    runs-on: ubuntu-latest

    steps:
    - name: 'Checkout Github Action'
      uses: actions/checkout@master

    - name: Set up .NET Core
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: '5.0.x'

    - name: Setup Node
      uses: actions/setup-node@v2.5.1
      with:
        node-version: 10.16.3

    - name: Install dependencies in client app
      working-directory: src/Tailwind.Traders.Web/ClientApp
      run: npm install

    - name: Build and publish with dotnet
      working-directory: src/Tailwind.Traders.Web
      run: |
        dotnet build --configuration Release

```

Use Control + Space to trigger autocomplete in most situations.

YAML

YAML basics

- Extension: `.yml` or `.yaml`
- A strict superset of **JSON**
- Contains syntactically relevant **newlines** and **indentation** instead of braces
- Data-serialization language writable and readable by humans

YAML basics

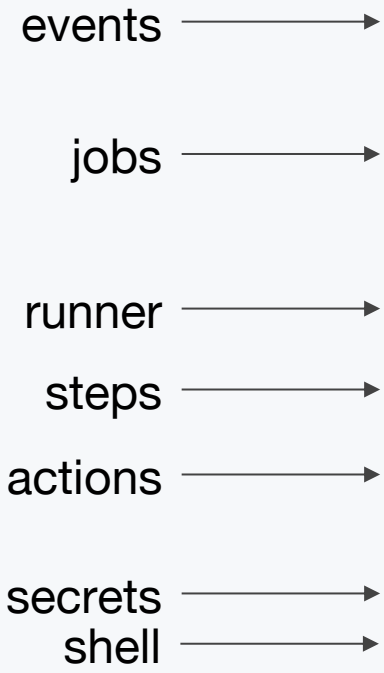
```yaml
# This is a comment in yaml

# Scalar types:
key: value

# Data types:
integer: 42
float: 42.0
string: a text value
boolean: true
null value: null
datetime: 1999-12-31T23:59:43.1Z

# Keys and values can contain spaces and do not need quotation.
# You can quote both with single or double quotes:
'single quotes': 'have ''one quote'' as the escape pattern'
"double quotes": "have the \"backslash \" escape pattern"

# Literal blocks:
literal_block: |
    Text blocks use 4 spaces as indentation. The entire
    block is assigned to the key 'literal_block' and keeps
    line breaks and empty lines.

    The block continuous until the next element.

```

```yaml
# Collection types

# Maps
# Maps use 2 spaces of indentation:
nested_type:
  key1: value1
  key2: value2
  another_nested_type:
    key1: value1

#JSON syntax:
map: {key: value}

# Sequence
# Uses a dash before each item:
sequence:
  - item1
  - item2

#JSON syntax:
sequence: [item1, item2, item3]
```

Basic workflow syntax

Basic syntax

`./.github/workflows/workflow-file-name.yml`

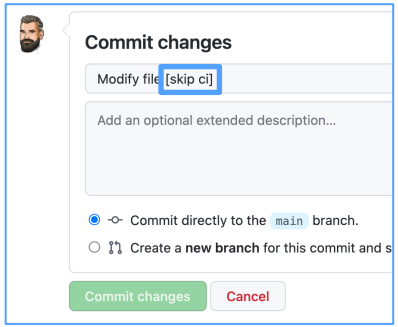



```yaml
name: Super Linter workflow

on:
  push:

jobs:
  lint:
    name: Lint Code Base

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - uses: github/super-linter@v3
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      - run: echo "Hello world"
```

Workflow triggers

Events that triggers workflows

- Trigger:
  - Webhook events
  - Scheduled events
  - Manual event



AccelerateDevOps / .github / workflows / starter.yml in main

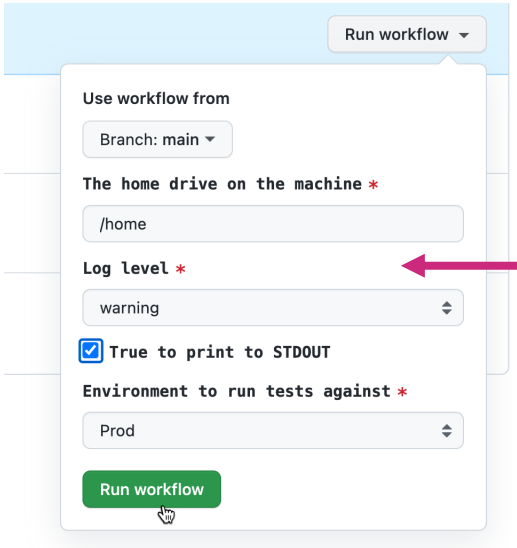
Edit file | Preview changes

```yaml
name: Starter Workflow

on:
  # Webhook events
  push:
    branches:
      - main
  issues:
    types: [opened, edited, milestoned]

  # Scheduled events

  schedule:
    - cron: '*/15 * * * *'
    - cron: '0 9-17 * * *'
    - cron: '11 11 * * 5'

  # Manual events
  workflow_dispatch:
    inputs:
      homedrive:
        description: 'The home drive on the machine'
        required: true
        default: '/home'
```

Runs every 15 minutes.
Actions schedules run at most every 5 minutes. Learn more

Events that triggers workflows

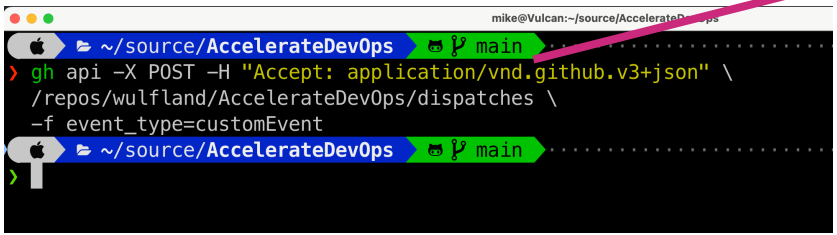
Manual events




```yaml
# Manual events
workflow_dispatch:
  inputs:
    homedrive:
      description: 'The home drive on the machine'
      required: true
      default: '/home'
    logLevel:
      description: 'Log level'
      required: true
      default: 'warning'
      type: choice
      options:
      - info
      - warning
      - debug
    print_tags:
      description: 'True to print to STDOUT'
      required: true
      type: boolean
    environment:
      description: 'Environment to run tests against'
      type: environment
      required: true
```

Events that triggers workflows

Manual events: trigger using the API
(curl, octokit, GitHub CLI)




```yaml
# Trigger using the API
repository_dispatch:
  types: [customEvent]

# Call for example using GitHub CLI:
# $ gh api -X POST -H "Accept: application/vnd.github.v3+json" \
# /repos/wulfland/AccelerateDevOps/dispatches \
# -f event_type=customEvent
```

Jobs and steps

Workflow jobs

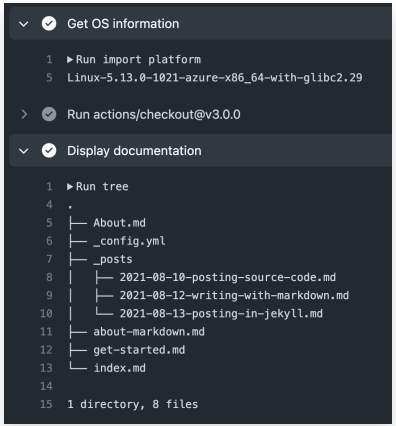
- Map – run in parallel by default
- Can be chained using **needs** keyword
- Runs on a **runner** in one process
- Contains a sequence of steps
- Steps can be a shell command (run) or an action (uses)

```yaml
jobs:
  job_1:
    runs-on: ubuntu-latest

    steps:
      - run: echo "🎉 The job was triggered by a ${{ github.event_name }} event."
      - run: echo "🏠 drive is `${{ github.event.inputs.homedrive }}`."
      - run: echo "🎯 environment is `${{ github.event.inputs.environment }}`."
      - run: echo "📖 log level is `${{ github.event.inputs.logLevel }}`."
      - run: echo "➿ Run the matrix? `${{ github.event.inputs.run_matrix }}`."

  job_2:
    runs-on: ubuntu-latest
    needs: job_1
    steps:
      - run: echo "Status ${{ job.status }}"

  job_3:
    runs-on: ubuntu-latest
    needs: job_1
    steps:
      - run: echo "Services ${{ job.services }}"

  job_4:
    runs-on: ubuntu-latest
    needs: [job_2, job_3]
    steps:
      - run: echo "Status ${{ job.status }}"
```

Workflow steps

- Sequence in a job
- Runs in the same process / same directory
- Runs in a shell

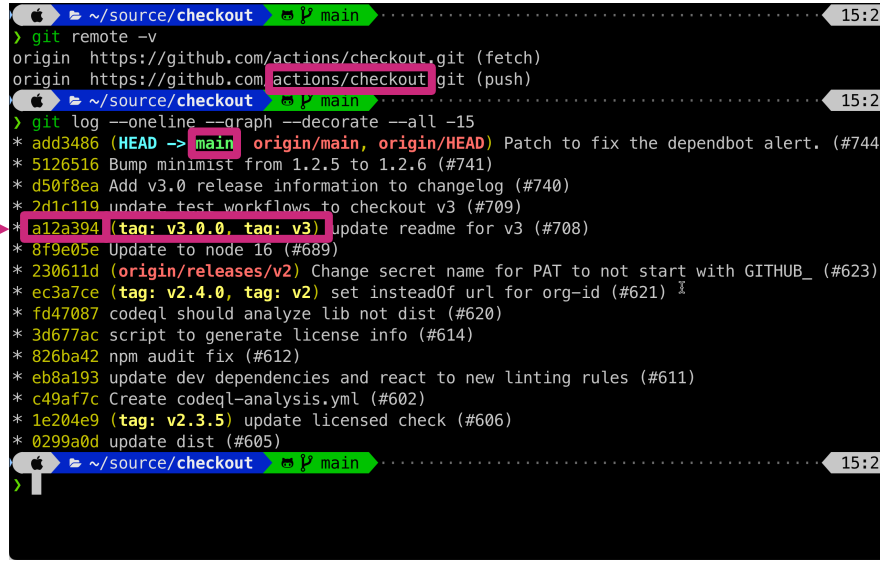
Parameter	Description
`bash`	Bash shell. The default shell on all non-Windows platforms with a fallback to sh. When specified on Windows, the bash shell included with Git is used.
`pwsh`	PowerShell Core. Default on the Windows platform.
`python`	The python shell. Allows you to run python scripts
`cmd`	Windows only! The windows command prompt.
`powershell`	Windows only! The classical Windows PowerShell.




```yaml
job_1:
  runs-on: ubuntu-latest

  steps:
    - run: echo "🎉 The job was triggered by a ${{ github.event_name }} event."
    - run: echo "🏠 drive is `${{ github.event.inputs.homedrive }}`."
    - run: echo "🎯 environment is `${{ github.event.inputs.environment }}`."
    - run: echo "📖 log level is `${{ github.event.inputs.logLevel }}`."
    - run: echo "👓 Run the matrix? `${{ github.event.inputs.run_matrix }}`."

    - name: Get OS information
      run: |
        import platform
        print(platform.platform())
      shell: python

    - uses: actions/checkout@v3.0.0
    - name: Display documentation
      run: tree
      working-directory: docs
```

Actions

- A reusable step
- Lives in a git repo
- Syntax
  - `{owner}/{repo}@{ref}`
  - `{owner}/{repo}/{path}@{ref}`
  - `./.github/actions/my-action`
- References
  - `SHA / Tag / Branch`
- Pass variables to Action
  - `with:`
  - `env:`




```
- uses: actions/checkout@a12a3943b4bdde767164f792f33f40b04645d846
- uses: actions/checkout@v3.0.0
- uses: actions/checkout@v3
- uses: actions/checkout@main
```

Actions

User docker images as actions

```yaml
- name: Run a docker containers as an action
  uses: docker://alpine:3.8

- uses: docker://ghcr.io/wulfland/container-demo:latest
```

main container-demo / Dockerfile

wulfland Update Dockerfile

1 contributor

2 lines (2 sloc) | 48 Bytes

```
1  FROM alpine:3.14.2
2  CMD ["echo", "Hello World!"]
```

Run ghcr.io/wulfland/container-demo:latest

```
1  ▶ Run docker://ghcr.io/wulfland/container-demo:latest
2  /usr/bin/docker run --name ghcriowulflandcontainerdemolate
   e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e GITHUB_RETENTION_D
   GITHUB_API_URL -e GITHUB_GRAPHQL_URL -e GITHUB_REF_NAME -e
   GITHUB_PATH -e GITHUB_ENV -e GITHUB_STEP_SUMMARY -e RUNNER
   ACTIONS_CACHE_URL -e GITHUB_ACTIONS=true -e CI=true -v "/v
   "/home/runner/work/_temp/_github_workflow":"/github/workfl
   ghcr.io/wulfland/container-demo:latest
3  Hello World!
```

Demo

Hands-on:
My first workflow

Contexts and expression syntax

Contexts and expressions syntax

- `${{ <expression> }}`
- Context syntax
  - `context['key']` (if key starts with number or contains special characters)
  - `context.key`
- Context
  - `matrix`
  - `github`
  - `env`
  - `runner`

```yaml
steps:
  - name: Dump runner context
    run: echo '${{ toJSON(runner) }}'
  - name: Dump GitHub context
    run: echo '${{ toJSON(github) }}'
```

Dump runner context

```
▶ Run echo '{
{
  "os": "Linux",
  "arch": "X64",
  "name": "GitHub Actions 2",
  "tool_cache": "/opt/hostedtoolcache",
  "temp": "/home/runner/work/_temp",
  "workspace": "/home/runner/work/AccelerateDevOps"
}
```

Context

succeeded 1 minute ago in 3s

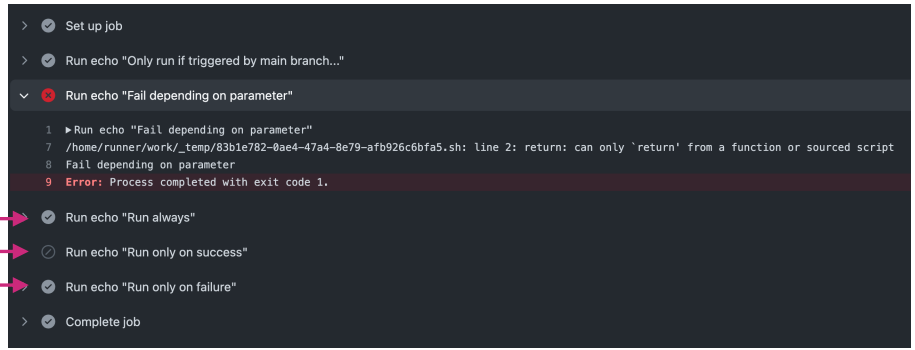
Set up job

Dump GitHub context


```
▶ Run echo '{
{
  "token": "***",
  "job": "context_job",
  "ref": "refs/heads/main",
  "sha": "c610cff739f85138a175c892651d204e71cedb43",
  "repository": "wulfland/AccelerateDevOps",
  "repository_owner": "wulfland",
  "repository_owner_id": "5276337",
  "repositoryUrl": "git://github.com/wulfland/AccelerateDevOps.git",
  "run_id": "2161816664",
  "run_number": "32",
  "retention_days": "90",
  "run_attempt": "1",
  "artifact_cache_size_limit": "10",
  "repository_id": "383720539",
  "actor_id": "5276337",
  "actor": "wulfland",
  "workflow": "Starter Workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch",
  "event": {
    "inputs": {
      "environment": "github-pages",
      "homedrive": "/home",
      "logLevel": "warning",
      "run_matrix": "false"
```

Contexts and expressions syntax

```yaml
expression_job:
  runs-on: ubuntu-latest
  name: Expressions
  if: ${{ github.ref == 'refs/heads/main' && github.event.inputs.logLevel == 'debug' }}

  steps:
    - run: echo "Only run if triggered by main branch..."
      if: contains(github.ref, 'main')

    - run: |
        echo "Fail depending on parameter"
        return 1
      if: github.event.inputs.run_matrix == 'true'

    - run: echo "Run always"
      if: always()
    - run: echo "Run only on success"
      if: success()
    - run: echo "Run only on failure"
      if: failure()
```



Contexts and expressions syntax

Function	Description
`success()`	Returns true if none of the previous steps have failed or been cancelled.
`always()`	Returns true even if a previous step was cancelled and causes the step to always get executed anyway.
`cancelled()`	Returns only true if the workflow was canceled.
`failure()`	Returns true if a previous step of the job had failed.

Operator	Description
`( )`	Logical group
`!`	Not
`< , <=`	Less than, less than or equal
`> , >=`	Greater than, greater than or equal
`==`	Equal
`!=`	Not equal
`&&`	And
`||`	Or

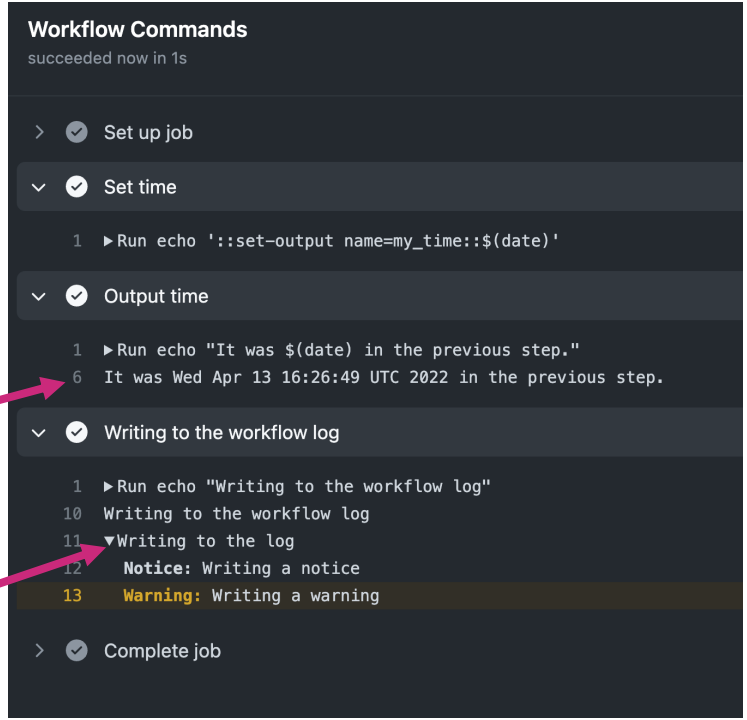
Function	Description
`contains(search, item)`	Returns true if search contains item.
`startsWith(search, item)`	Returns true if search start with item.
`endsWith(search, item)`	Returns true if search ends with item
`format(' {0} ', item)`	Replaces placeholders in a string.
`join(array, seperator)`	All values in array are concatenated into a string.
`toJSON(value)`	Returns a pretty-print JSON representation of value.
`fromJSON(value)`	Returns a JSON object or JSON data type for value.

Workflow commands

Workflow commands

- Interact with the workflow from within your steps
- Write command to output (normally using `echo`)
- Examples
  - `set-output`
  - `error`

```yaml
steps:
  - name: Set time
    run: echo '::set-output name=my_time::$(date)'
    id: set-time

  - name: Output time
    run: echo "It was ${{ steps.set-time.outputs.my_time }} in the previous step."

  - name: Writing to the workflow log
    run: |
      echo "Writing to the workflow log"
      echo "::group::Writing to the log"
      echo "::notice file=starter.yml,line=169,col=19,endColumn=22::Writing a notice"
      echo "::warning file=starter.yml,line=171,col=19,endColumn=22::Writing a warning"
      echo "::endgroup::"
```



Workflow commands

Toolkit function	Equivalent workflow command
`core.addPath`	Accessible using environment file `GITHUB_PATH`
`core.debug`	`debug`
`core.notice`	`notice`
`core.error`	`error`
`core.endGroup`	`endgroup`
`core.exportVariable`	Accessible using environment file `GITHUB_ENV`
`core.getInput`	Accessible using environment variable `INPUT_{NAME}`
`core.getState`	Accessible using environment variable `STATE_{NAME}`

Toolkit function	Equivalent workflow command
`core.isDebug`	Accessible using environment variable `RUNNER_DEBUG`
`core.saveState`	`save-state` **Deprecated**
`core.setCommandEcho`	`echo`
`core.setFailed`	Used as a shortcut for `::error` and `exit 1`
`core.setOutput`	`set-output` **Deprecated**
`core.setSecret`	`add-mask`
`core.startGroup`	`group`
`core.warning`	`warning`

Workflow commands

Examples

A workflow using `save-state` or `set-output` like the following

```
- name: Save state
  run: echo "::save-state name={name}::{value}"

- name: Set output
  run: echo "::set-output name={name}::{value}"
```

should be updated to write to the new `GITHUB_STATE` and `GITHUB_OUTPUT` environment files:

```
- name: Save state
  run: echo "{name}={value}" >> $GITHUB_STATE

- name: Set output
  run: echo "{name}={value}" >> $GITHUB_OUTPUT
```

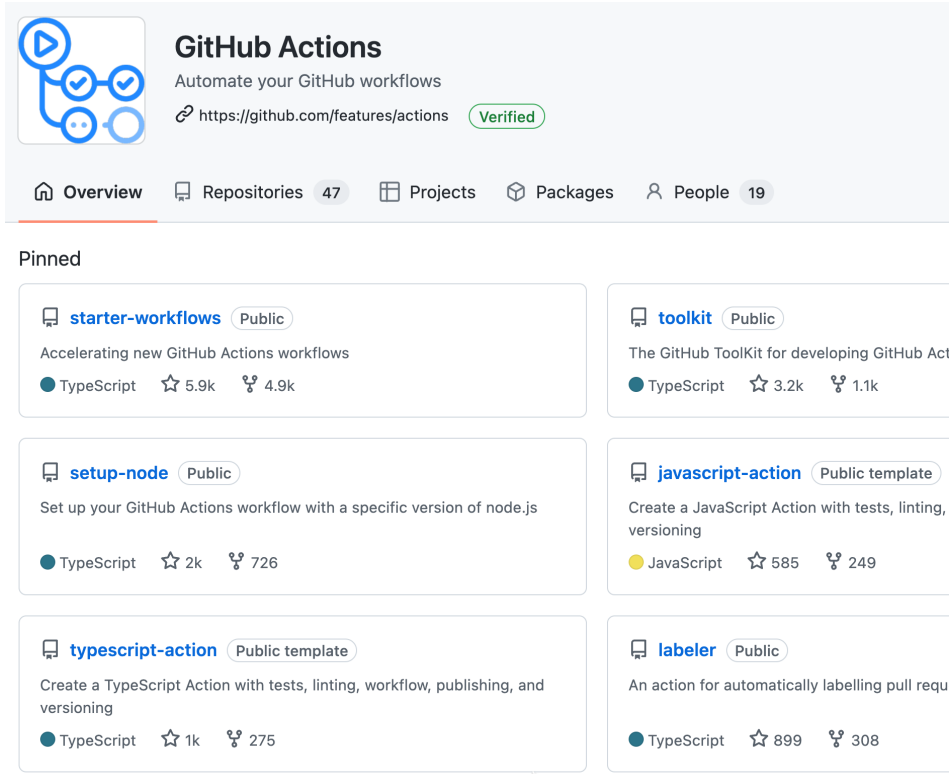
More advanced syntax elements

Syntax element	Description
`permissions`	Set workflow permissions for GITHUB_TOKEN
`env`	Set environment variable for all run steps
`defaults`	Set the shell and working directory for the run
`concurrency`	Manage workflows running concurrently
`needs`	Make job dependent of each other. Share outputs
`if`	Check whether a job should run based on variables. Options are: success() always() cancelled() failure()
`timeout`	Limit runtime
`continue-on-error`	Handle termination of workflows
`container`	Use a container for the steps execution
`services`	Use a container for the steps execution

Actions

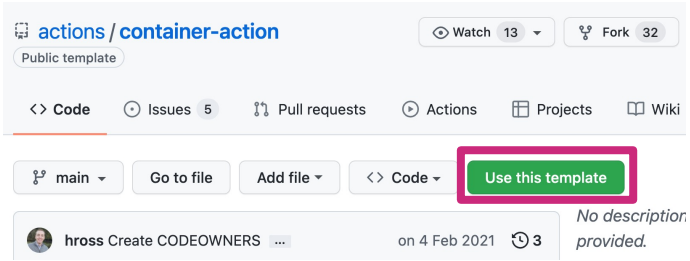
GitHub Actions

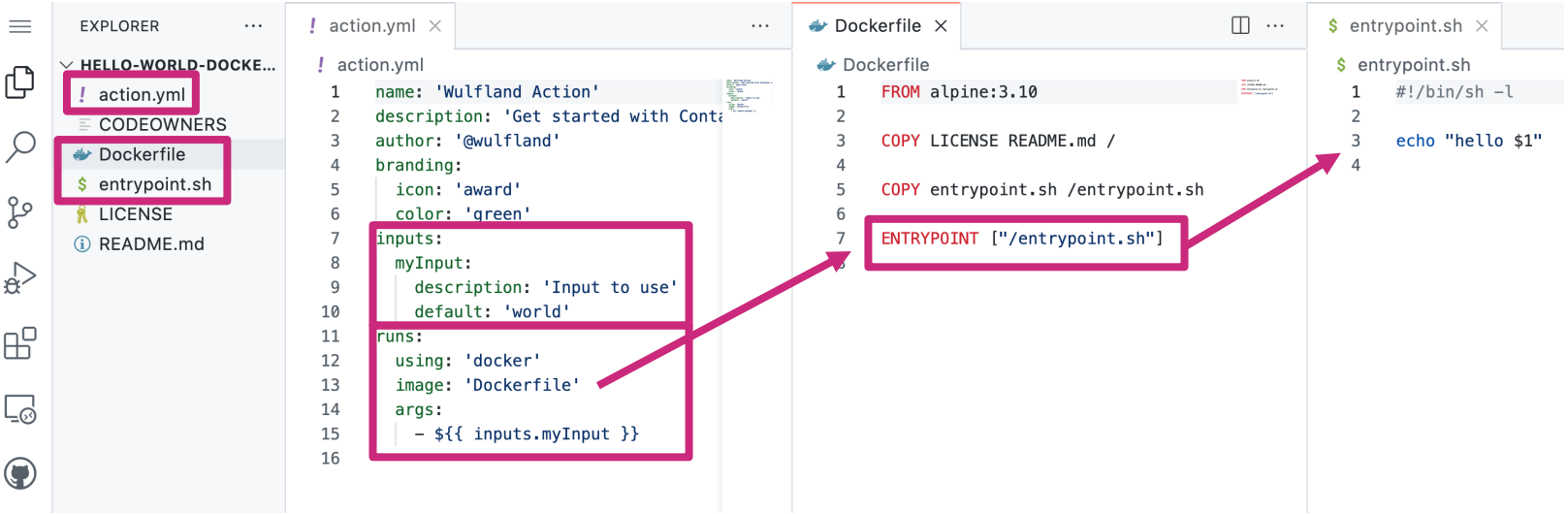
- Actions are reusable
- 3 kinds of Actions:
  - Container
  - JavaScript / Typescript
  - Composite Actions



Container Actions

- Dockerfile or existing image
- Inputs



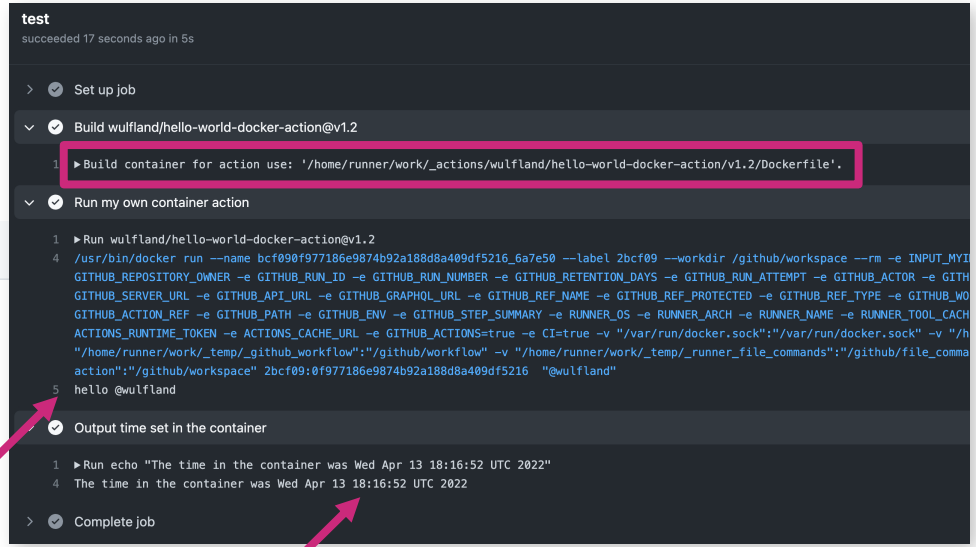


Container Actions

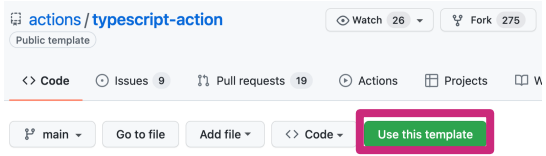
- Dockerfile or existing image
- inputs

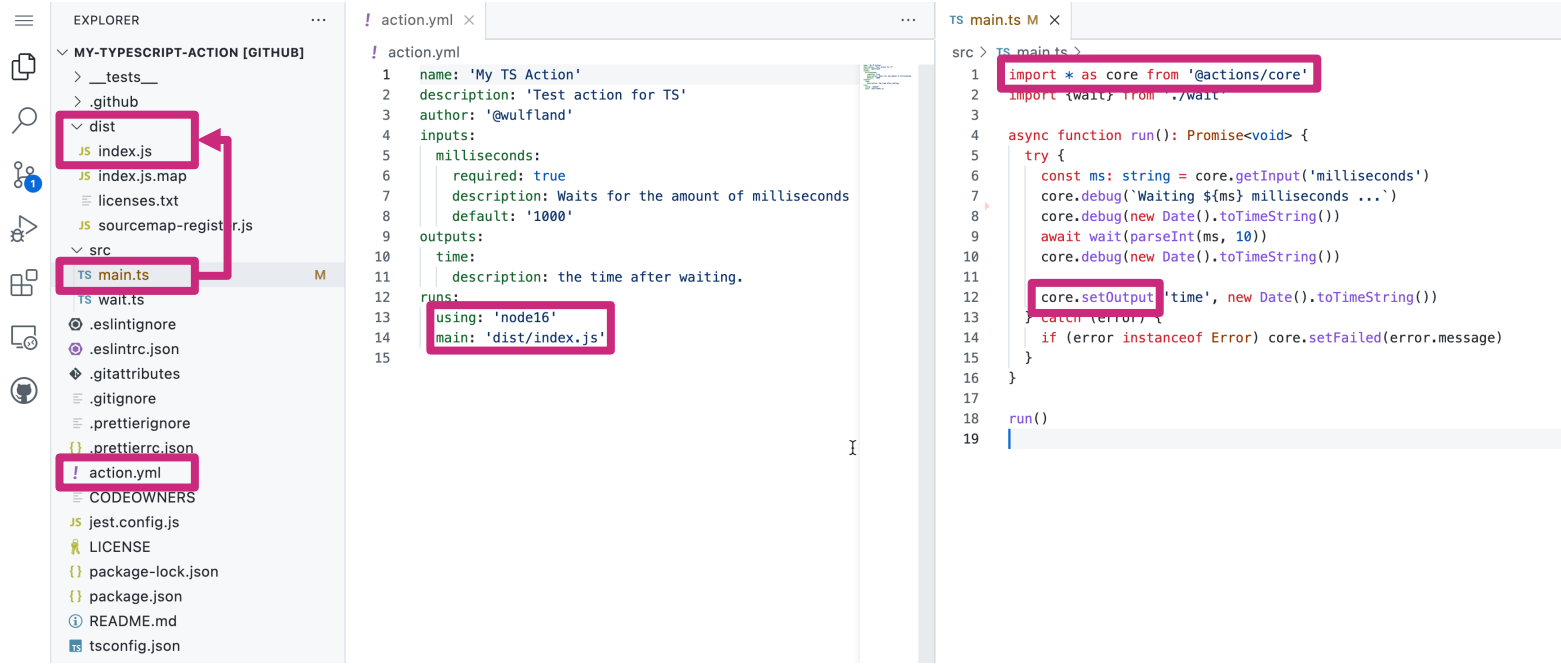
15 lines (13 sloc) | 398 Bytes

```yaml
name: Test Action

on: [workflow_dispatch]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Run my own container action
        id: hello-action
        uses: wulfland/hello-world-docker-action@v1.2
        with:
          myInput: '@wulfland'
      - name: Output time set in the container
        run: echo "The time in the container was ${{ steps.hello-action.outputs.time }}"
```



JavaScript Actions





Composite Actions

- Just a `action.yml` file
- Inputs
- Outputs
- Runs

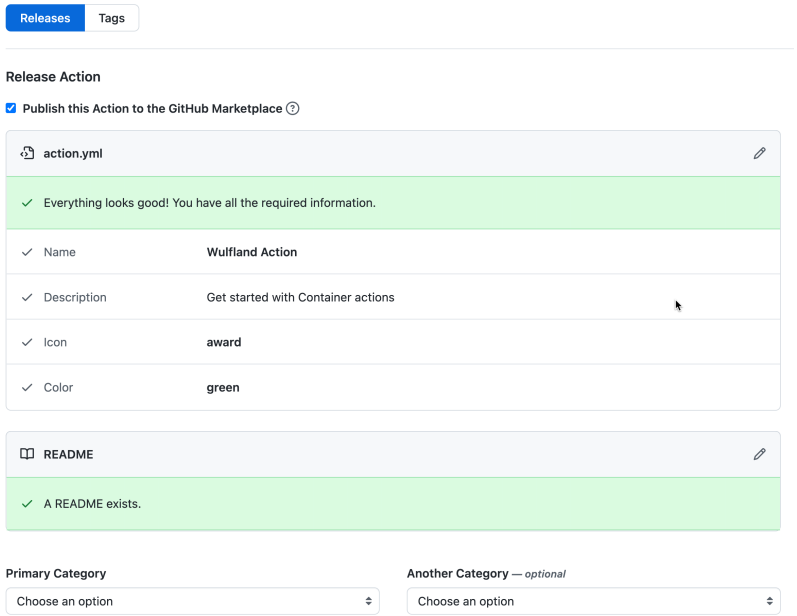
28 lines (25 sloc) | 689 Bytes

```yaml
name: 'Hello World'
description: 'Greet someone'
inputs:
  who-to-greet:
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  random-number:
    description: "Random number"
    value: ${{ steps.random-number-generator.outputs.random-id }}
runs:
  using: "composite"
  steps:
    - run: echo Hello ${{ inputs.who-to-greet }}.
      shell: bash

    - id: random-number-generator
      run: echo "::set-output name=random-id::$(echo $RANDOM)"
      shell: bash

    - run: echo "${{ github.action_path }}" >> $GITHUB_PATH
      shell: bash

    - run: echo "Goodbuye $YOU"
      shell: bash
      env:
        YOU: ${{ inputs.who-to-greet }}
```

Share your action in the marketplace

- Really easy to share
- Draft a release
- Unique name
- Check for README, Icon Color

```yaml
name: 'Wulfland Action'
description: 'Get started with Container actions'
author: '@wulfland'
branding:
  icon: 'award'
  color: 'green'
```



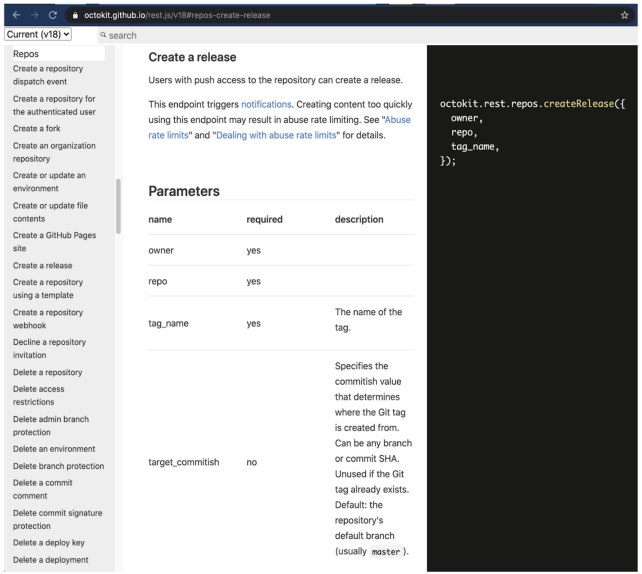
Demo

Hands-on: My first Action

The GitHub API and Octokit

Using the GitHub API

- REST API (v3)
  - Libraries available for most languages
  - Octokit
- GraphQL (v4)
  - The future of the GitHub API
  - A query language allowing granular control of request and response



Job Summaries

```yaml
jobs:
  job1:
    runs-on: ubuntu-latest

    steps:
      - run: echo '### Hello world! :rocket:' >> $GITHUB_STEP_SUMMARY
      - run: echo '### Love this feature! :medal_sports:' >> $GITHUB_STEP_SUMMARY
```

job1 summary

Hello world! 🚀

Love this feature! 🎖️

Job summary generated at run-time

```yaml
    steps:
      - name: Write Summary from Action
        uses: actions/github-script@v6.1.0
        with:
          script: |
            await core.summary
            .addHeading('Test Results')
            .addTable([
              [{data: 'File', header: true}, {data: 'Result', header: true}],
              ['foo.js', 'Pass ✅'],
              ['bar.js', 'Fail ❌'],
              ['test.js', 'Pass ✅']
            ])
            .addLink('View staging deployment!', 'https://github.com')
            .write()
```

job2 summary

Test Results

File	Result
foo.js	Pass ✅
bar.js	Fail ❌
test.js	Pass ✅

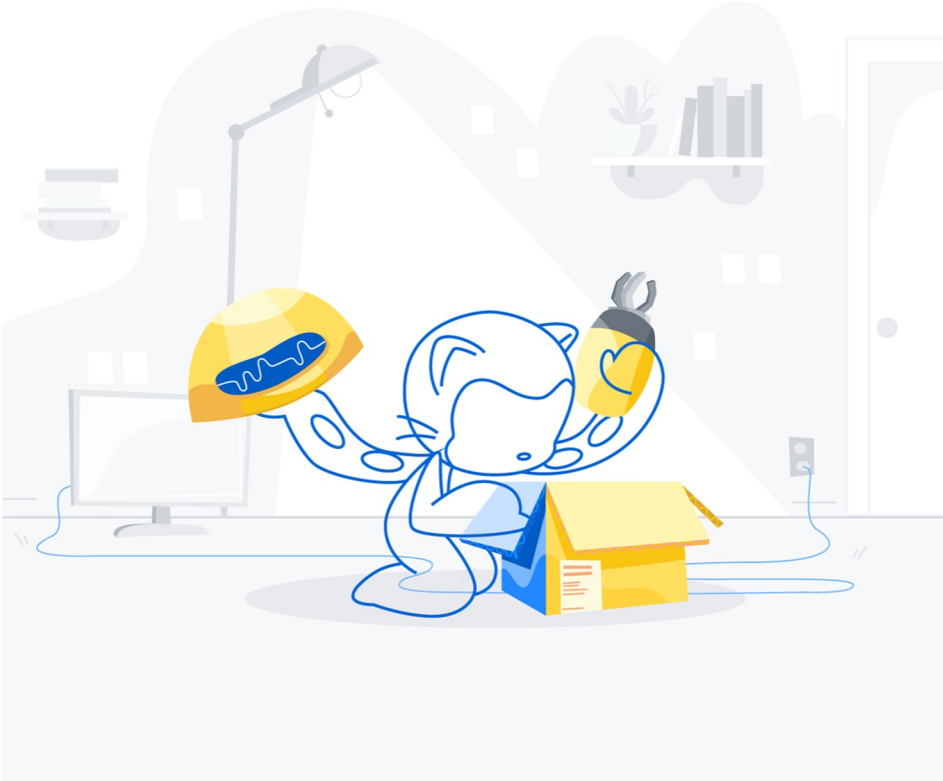
View staging deployment!

Job summary generated at run-time

Writing Actions

Best Practices

- Design for reusability
- Small and focused (Single Responsibility Principle)
- Write tests and a test workflow
- Semantic versioning
- Documentation
- Proper `action.yml` metadata
- github.com/actions/toolkit
- Publish the Action to the marketplace

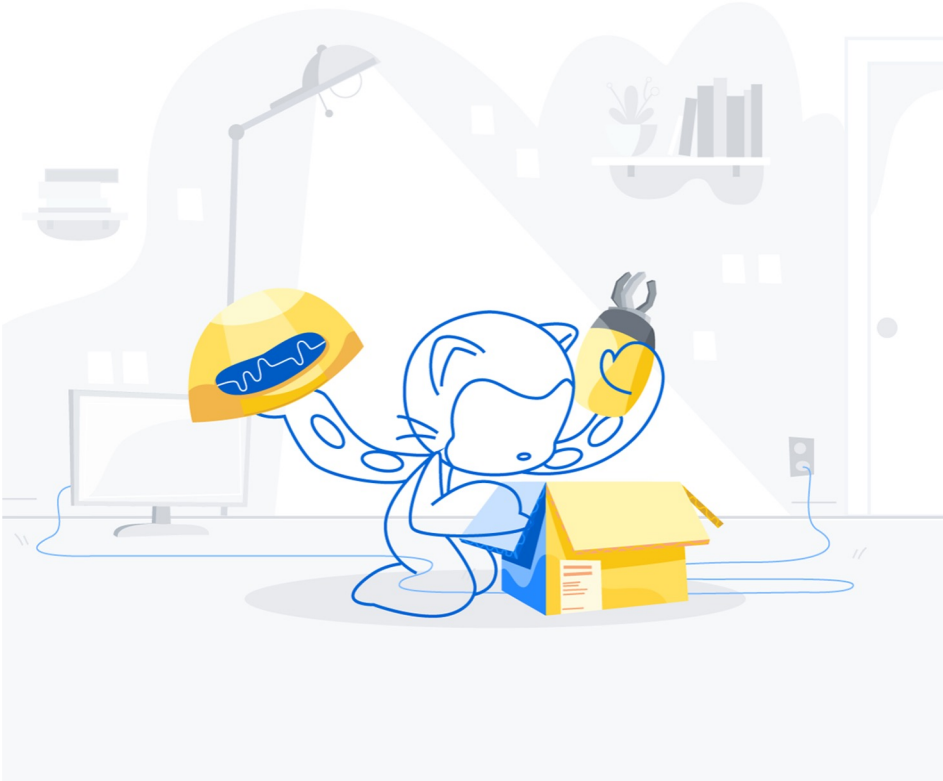


Retro Day 1

Introduction Day 2

Agenda – day 1

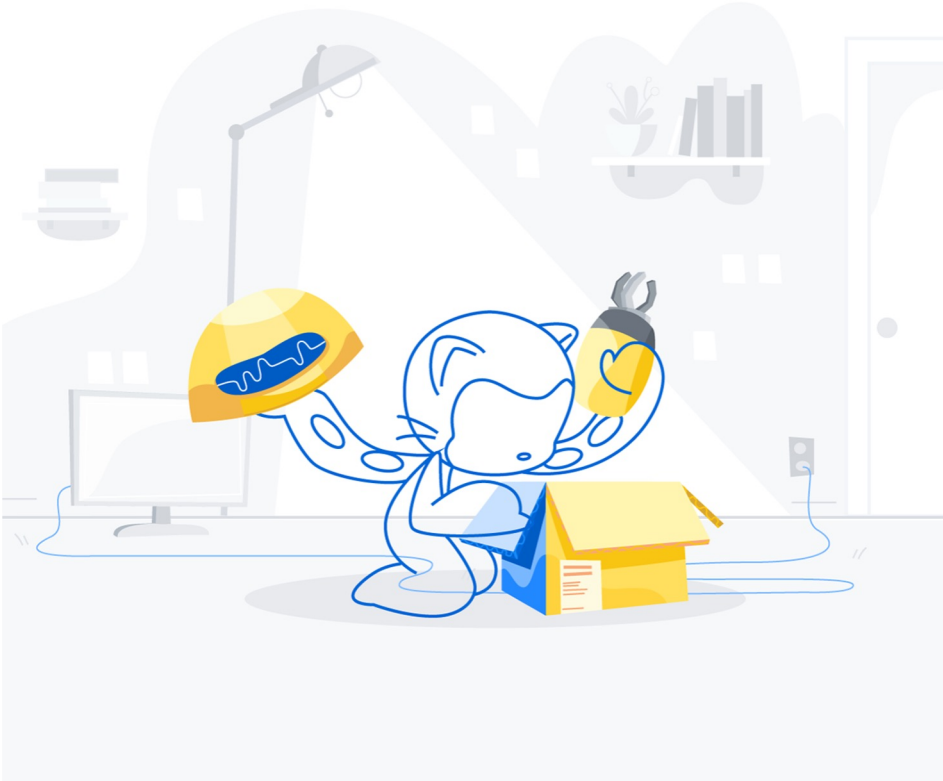
- Workflow fundamentals
- YAML and workflow syntax
- Writing workflows
- GitHub Actions
- Authoring Actions



Agenda – day 2

- CI/CD with GitHub Actions
- Secrets, Variables and Environments

  (staged deployments)
- Running workflows (runners)
- Action policies
- Sharing workflows (reusable workflows)



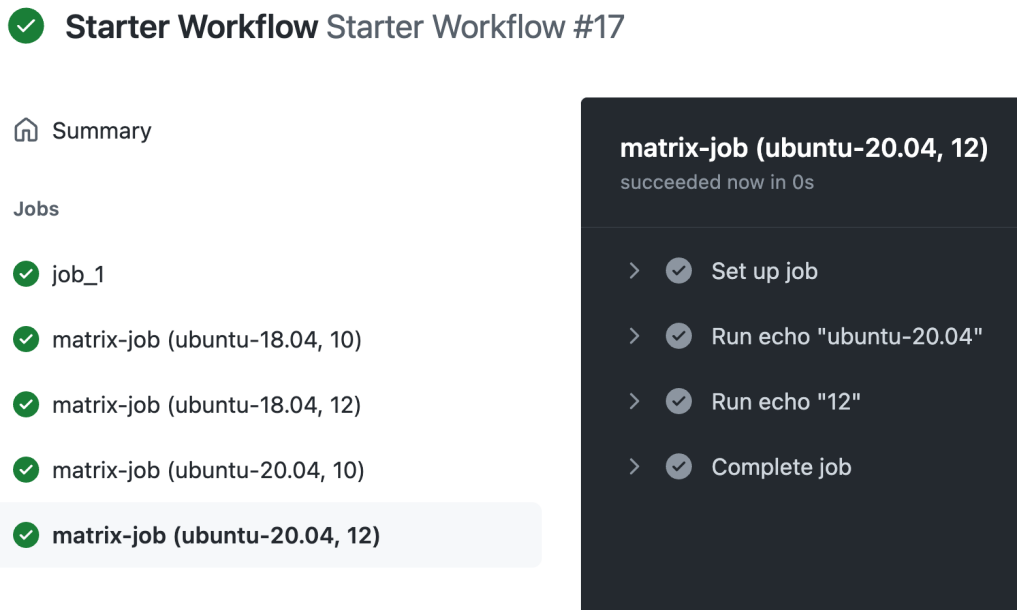
Icebreaker

Actions for CI / CD

Strategy

- For-loop – array
- Nested for-loops: multidimensional array
- Runs for all combinations in all dimensions
- Fail-fast (yes/no)
- Max 256 parallel jobs

```yaml
matrix_job:
  name: matrix-job
  runs-on: ${{ matrix.runner }}
  if: github.event.inputs.run_matrix

  strategy:
    matrix:
      runner: [ubuntu-18.04, ubuntu-20.04]
      node: [10,12]

  steps:
    - run: echo "${{ matrix.runner }}"
    - run: echo "${{ matrix.node }}"
```



Basic CI workflow

- Uses a build matrix across multiple node versions
- Runs on the VM
  - Ubuntu in this case
- Actions are composable
- Checkout is separate
- Setup for most languages in github.com/actions
- `npm run` by shell
- Artifact upload is a separate action

```yaml
name: Node CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [10.x, 12.x]

    steps:
      - uses: actions/checkout@v2
      - name: Use Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.node-version }}
      - name: Install and test
        run: |
          npm ci
          npm run build --if-present
          npm test
      - uses: actions/upload-artifact@v2
        with:
          name: artifact
          path: dist/
```

Linting

- Linting as part of CI runs
- See e.g. the super-linter
  - github.com/github/super-linter
  - Supports ~45 different languages
- Easily added as a new step to an existing workflow

```yaml
name: Lint Code Base

on:
  push:
    branches-ignore: [main]
  pull_request:
    branches: [main]

jobs:
  build:
    name: Lint Code Base
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: Lint Code Base
        uses: github/super-linter@v4
        env:
          VALIDATE_ALL_CODEBASE: false
          DEFAULT_BRANCH: main
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Caching

Optimizing your workflow performance with caching:

- Temporarily save files between workflow runs
- 10GB max cache size per repo
- 7 days retention
- Scoped to key and branch
- Never cache sensitive data

Caching dependencies to speed up workflows

Caching can help with speeding up workflows when you need to install dependencies. NPM, Python, Ruby, etc… these are simple examples of applications that require dependencies to be built. But there are more complex scenarios, such as Java, C/C++ and modularized microservices that often require downstream artifacts. Caching can speed up your builds when your dependencies have not changed

Caching

```yaml
steps:
- uses: actions/checkout@v3

- name: Cache Primes
  id: cache-primes
  uses: actions/cache@v3
  with:
    path: primes
    key: ${{ runner.os }}-primes

- name: Generate Prime Numbers
  if: steps.cache-primes.outputs.cache-hit != 'true'
  run: |
    sleep 60
    echo "1 2 3..." > primes

- name: Use Prime Numbers
  run: cat primes
```

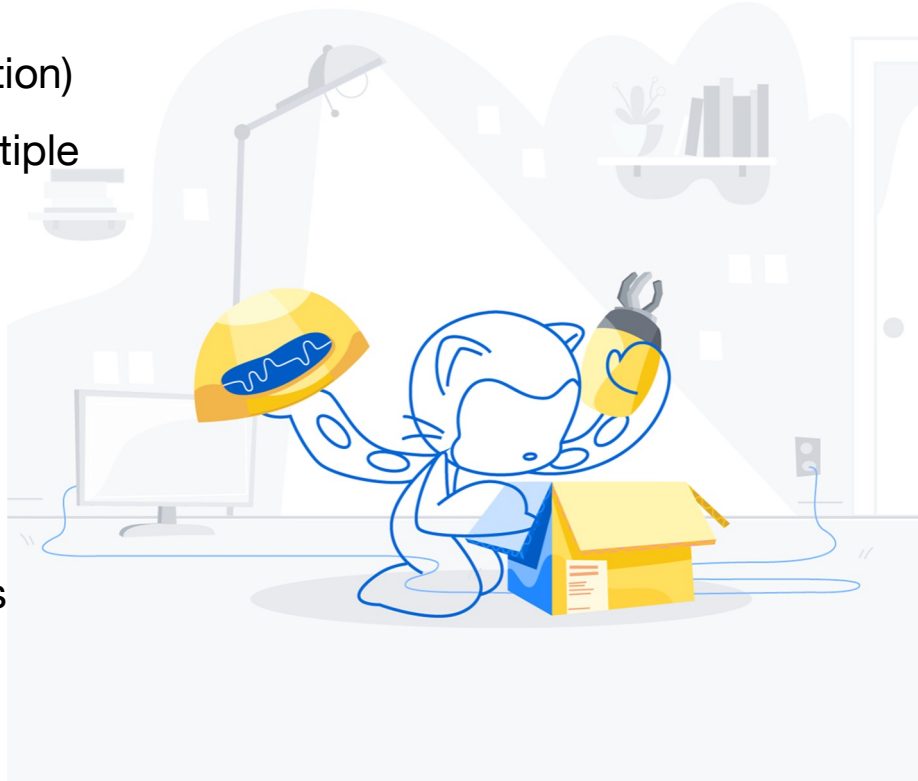
Java - Maven

```yaml
- name: Cache local Maven repository
  uses: actions/cache@v3
  with:
    path: ~/.m2/repository
    key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
    restore-keys: |
      ${{ runner.os }}-maven-
```

CI with Actions

Best Practices

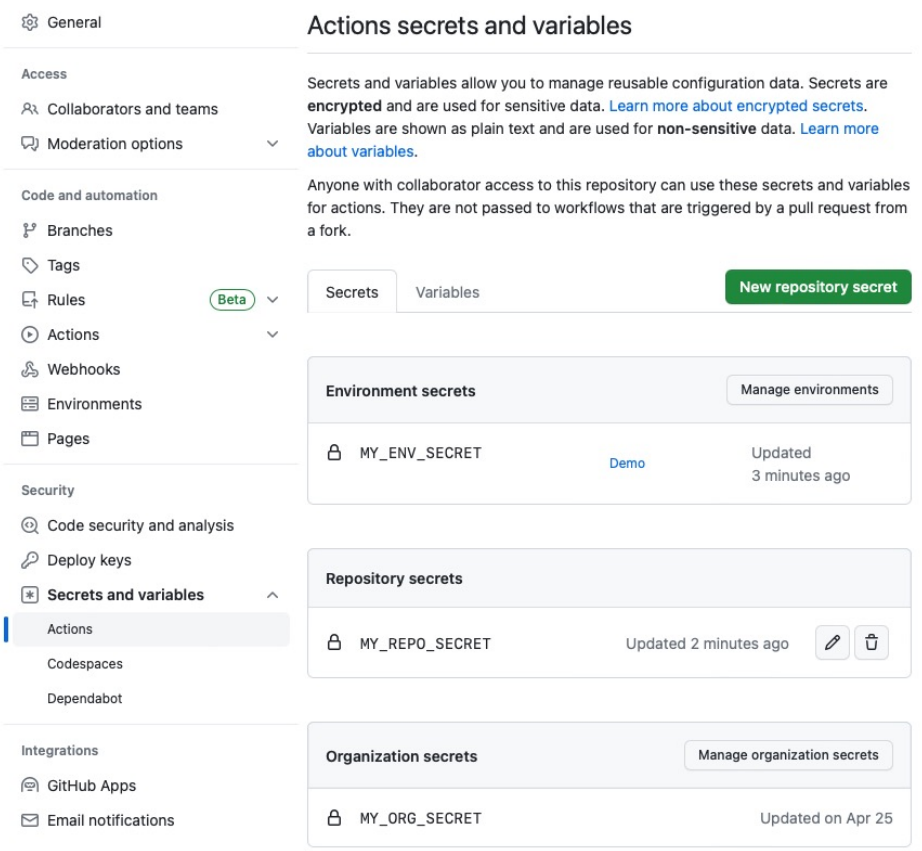
- Always use **`setup`** actions
- Implement caching if (only) needed (**`cache`** action)
- Use the **`matrix`** strategy to build and test multiple versions
- Use **`upload-artifact`**
- Use the super-linter:
  - `github/`**`super-linter:v4`**
  - `github/`**`super-linter:slim-v4`**
- Use tests and job summaries to display results
- Require status checks for pull requests



Secrets & variables

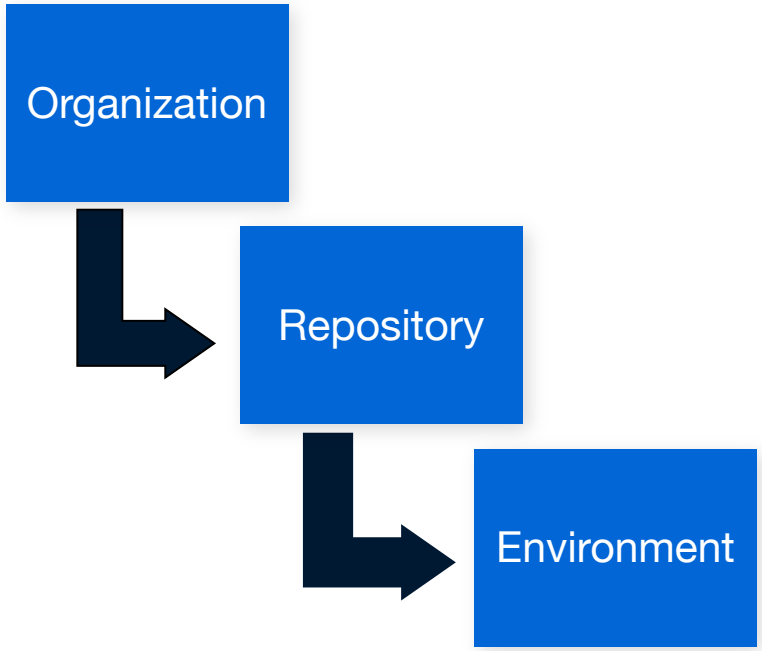
GitHub Secret store

- Built-in secret store
- Encrypted
  - LibSodium sealed box
- Use directly from your workflow
- Redacted in workflow logs
- API support
- Organization / repository / environment level secrets
- Do not use structured data!



Secrets

- Defined on org, repo, or environment level
- Secret context
  - `${{ secrets.MY_SECRET }}`
  - Set as input (`with:`) or environment (`env:`) for actions
- Set in UI or CLI
  - `$ gh secret set MY_SECRET -body "$value"`
  - `$ gh secret set MY_SECRET --env Prod`
  - `$ gh secret set MY_SECRET --org my-org`
- Masked in log



The `GITHUB_TOKEN`

- `${{ secrets.GITHUB_TOKEN }}` or `${{ github.token }}`
- Authenticate to GitHub to perform automation inside the workflow's repo
- Default permission read/write for all scopes (old default) or set to read

```yaml
permissions:
  contents: read
  pull-requests: write
```

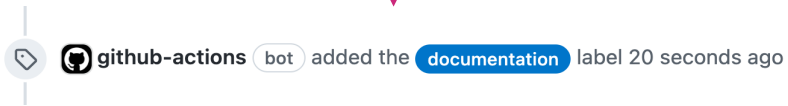
```yaml
permissions: read-all
```

```yaml
permissions:
  actions: read|write|none
  checks: read|write|none
  contents: read|write|none
  deployments: read|write|none
  issues: read|write|none
  packages: read|write|none
  pull-requests: read|write|none
  repository-projects: read|write|none
  security-events: read|write|none
  statuses: read|write|none
```

The GITHUB_TOKEN

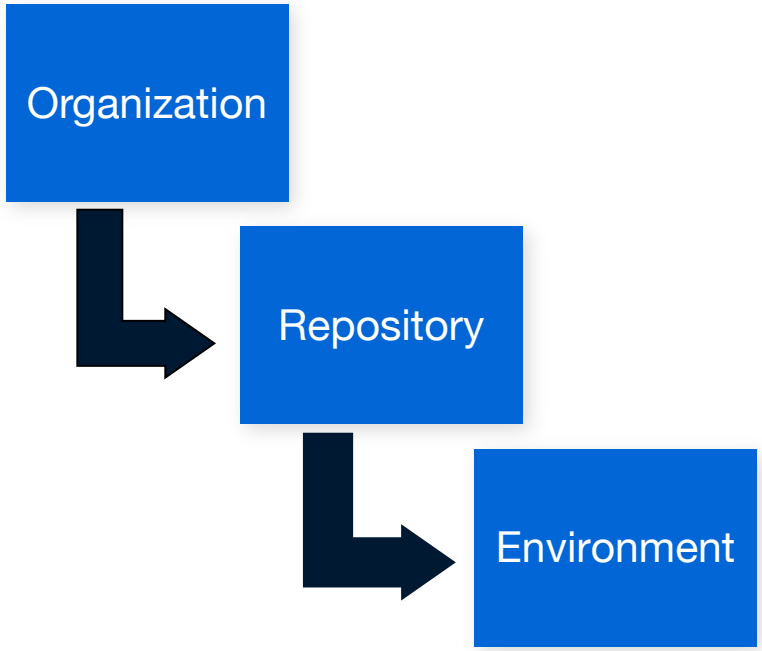
Perform actions as “github-actions”:

```yaml
permissions:
  contents: read
  issues: write


 label_issues:
   runs-on: ubuntu-latest
   if: github.event_name == 'issues'

   steps:
     - uses: andymckay/labeler@e6c4322d0397f3240f0e7e30a33b5c5df2d39e90
       with:
         add-labels: documentation
         repo-token: ${{ secrets.GITHUB_TOKEN }}
```



Variables

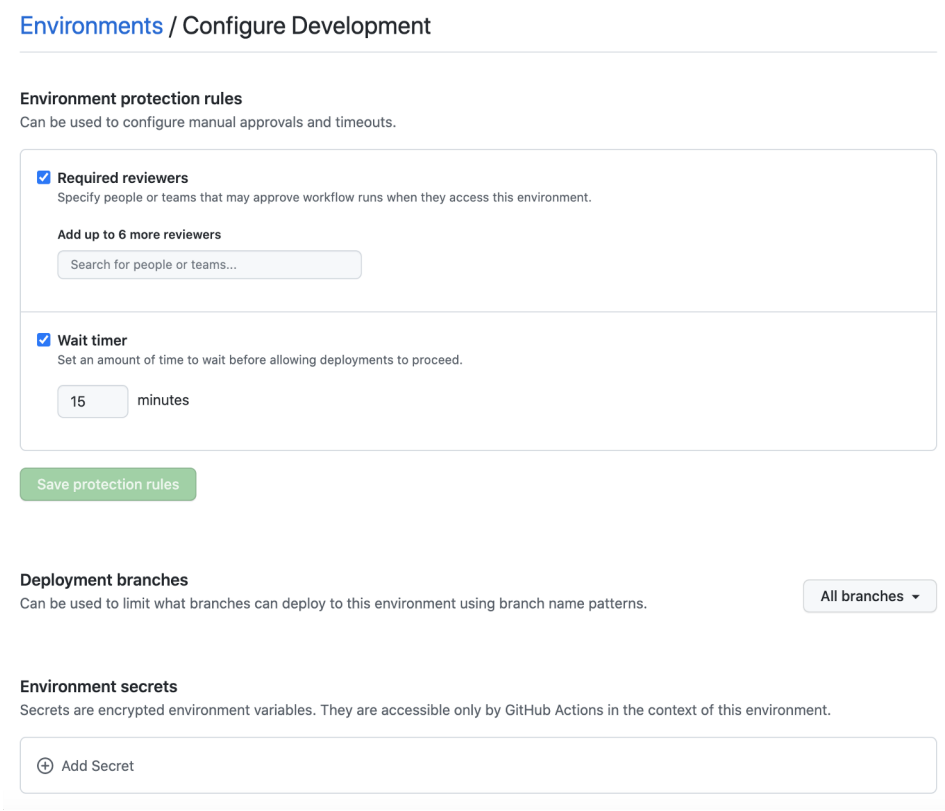
- Same setup as secrets, but no redacting
- Defined on org, repo, or environment level
- vars context
  - `${{ vars.MY_VAR }}`
  - Set as input (`with:`) or environment (`env:`) for actions
- **Not** masked in log



Environments

Environments

- Control deployments
- Add gated deployments with approvals
- Control secrets
- Review all deployments to an env
- Navigate directly to urls for deployments
- Fully integrated with the checks API (previously called deployment API)
- Supports matrix for gated deployments

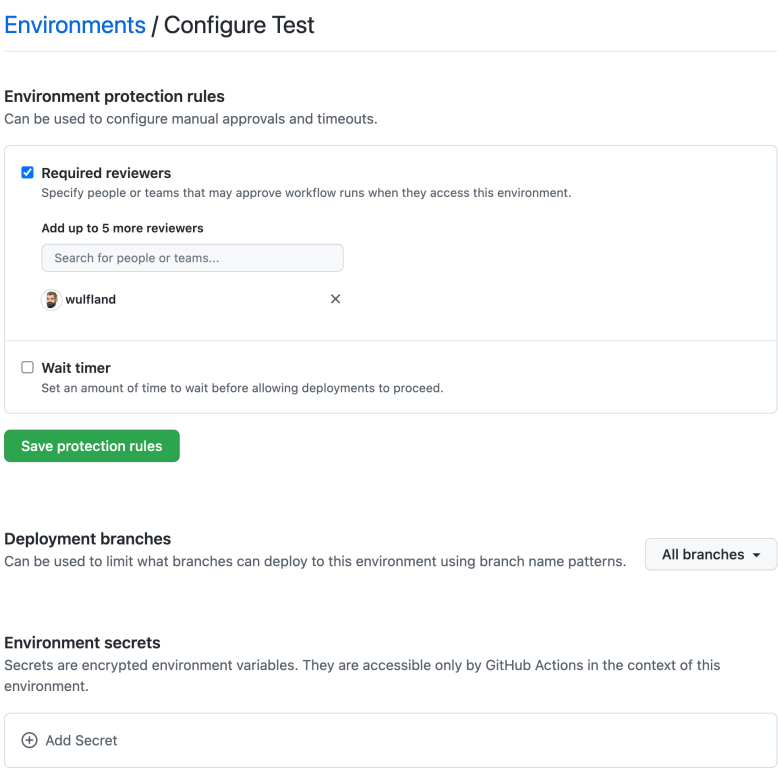


Environments

- Environments
  - Reviewers / Approvers
  - Wait timer (until 30 days)
  - Branches (→ branch protection!)
  - Deployment branches
  - Secrets

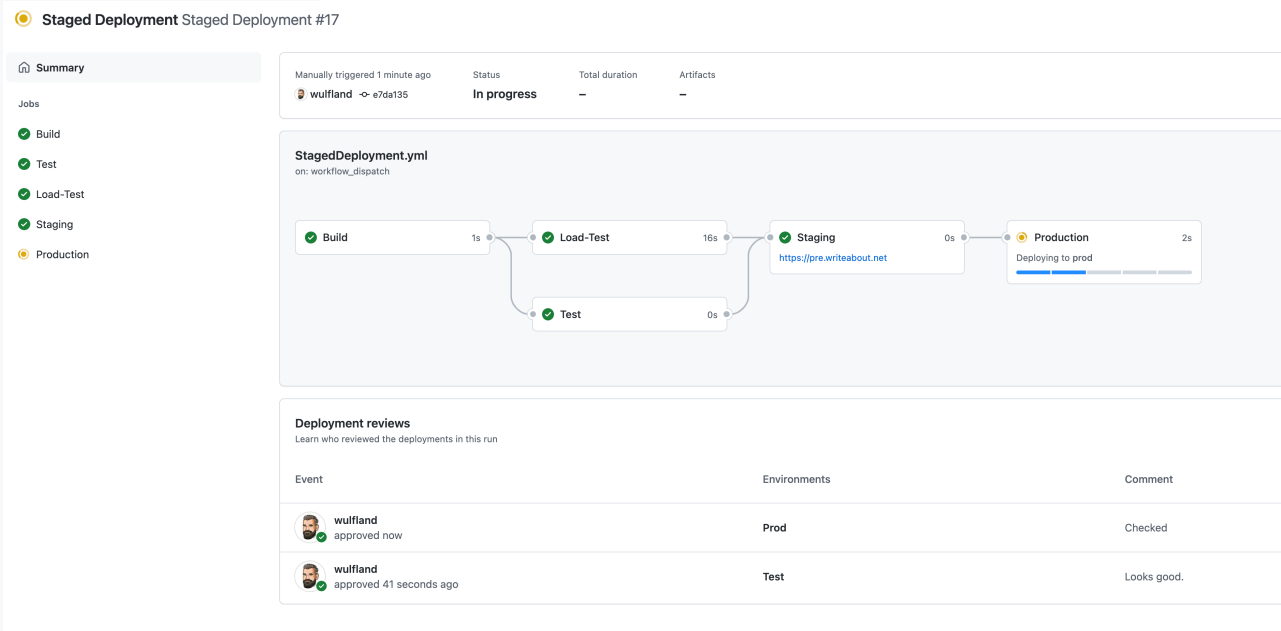
```yaml
Test:
  runs-on: ubuntu-latest
  environment: Test
  needs: Build
  steps:
  - name: Test app
    run: echo "Testing..."
```

```yaml
Production:
  runs-on: ubuntu-latest
  environment:
    name: prod
    url: https://writeabout.net
  needs: Staging
  steps:
  - name: Deploy app
    run: echo "Deploying..."
```



Environments

- Approvals
- Secrets after approval
- Set URL from output of other job/step
- Progress



Demo

Hands-on:
Staged deployments

Basic CD workflow

- Starter workflows available for most cloud providers
- Store the images in GitHub
- Jobs run on different envs
  - Uses the Docker image
  - Deploys the container image to Azure
- Use **`login`** actions
- Security hardening with **OpenID Connect**

```yaml
  Build-Docker-Image:
    runs-on: ubuntu-latest
    needs: build
    name: Build image and store in GitHub Packages
    steps:
      - name: Checkout
        uses: actions/checkout@v1

      - name: Download built artifact
        uses: actions/download-artifact@master
        with:
          name: webpack artifacts
          path: public

      - name: create image and store in Packages
        uses: mattdavis0351/actions/docker-gpr@1.3.0
        with:
          repo-token: ${{secrets.GITHUB_TOKEN}}
          image-name: ${{env.DOCKER_IMAGE_NAME}}

  Deploy-to-Azure:
    runs-on: ubuntu-latest
    needs: Build-Docker-Image
    name: Deploy app container to Azure
    steps:
      - name: "Login via Azure CLI"
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - uses: azure/docker-login@v1
        with:
          login-server: ${{env.IMAGE_REGISTRY_URL}}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Deploy web app container
        uses: azure/webapps-container-deploy@v1
        with:
          app-name: ${{env.AZURE_WEBAPP_NAME}}
          images: ${{env.IMAGE_REGISTRY_URL}}/${{ github.repository }}/${{env.DOCKER_IMAGE_NAME}}:${{ github.sha }}

      - name: Azure logout
        run: |
          az logout
```

Concurrency

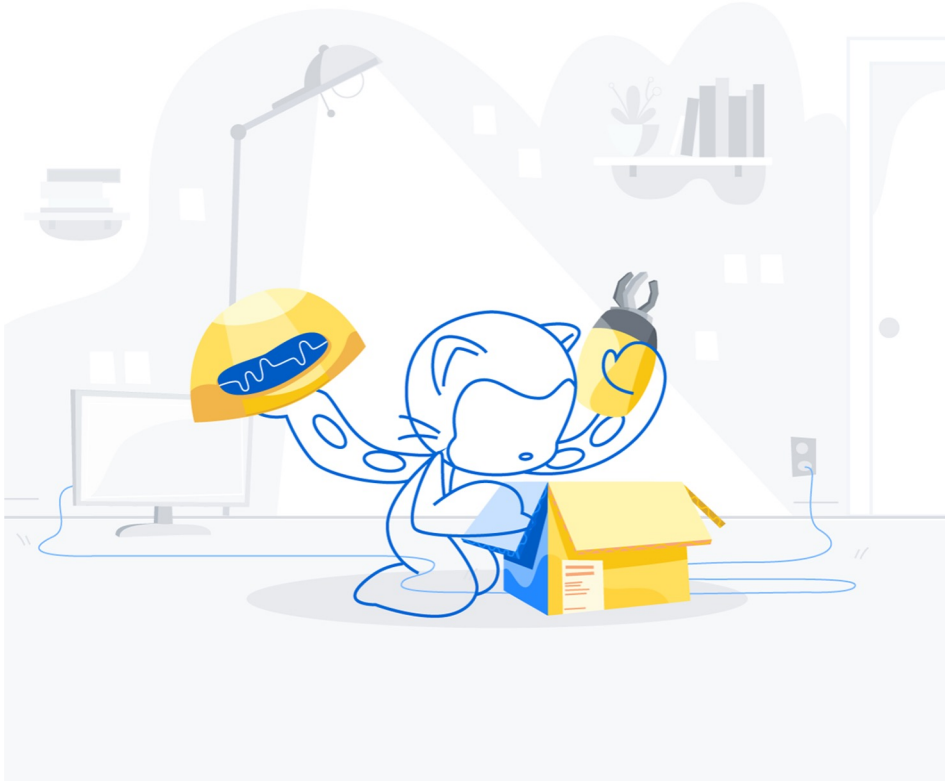
- Workflow or job
- Optional: cancel in-progress jobs
- Use cases:
  - Wait job/workflow until deployment completed
  - Cancel deployment and deploy newer version instead

```yaml
name: Concurrency

on: [workflow_dispatch]

jobs:
  job1:
    concurrency: group1
    runs-on: ubuntu-latest
    steps:
      - run: sleep 60
      - run: echo "Hello World! $(date)"

  job2:
    concurrency: group1
    runs-on: ubuntu-latest
    steps:
      - run: sleep 60
      - run: echo "Hello World! $(date)"

  job3:
    concurrency:
      group: group2
      cancel-in-progress: true

    runs-on: ubuntu-latest
    steps:
      - run: sleep 60
      - run: echo "Hello World! $(date)"

  job4:
    concurrency:
      group: group2
      cancel-in-progress: true

    runs-on: ubuntu-latest
    steps:
      - run: sleep 60
      - run: echo "Hello World! $(date)"
```

CD with Actions

Best Practices

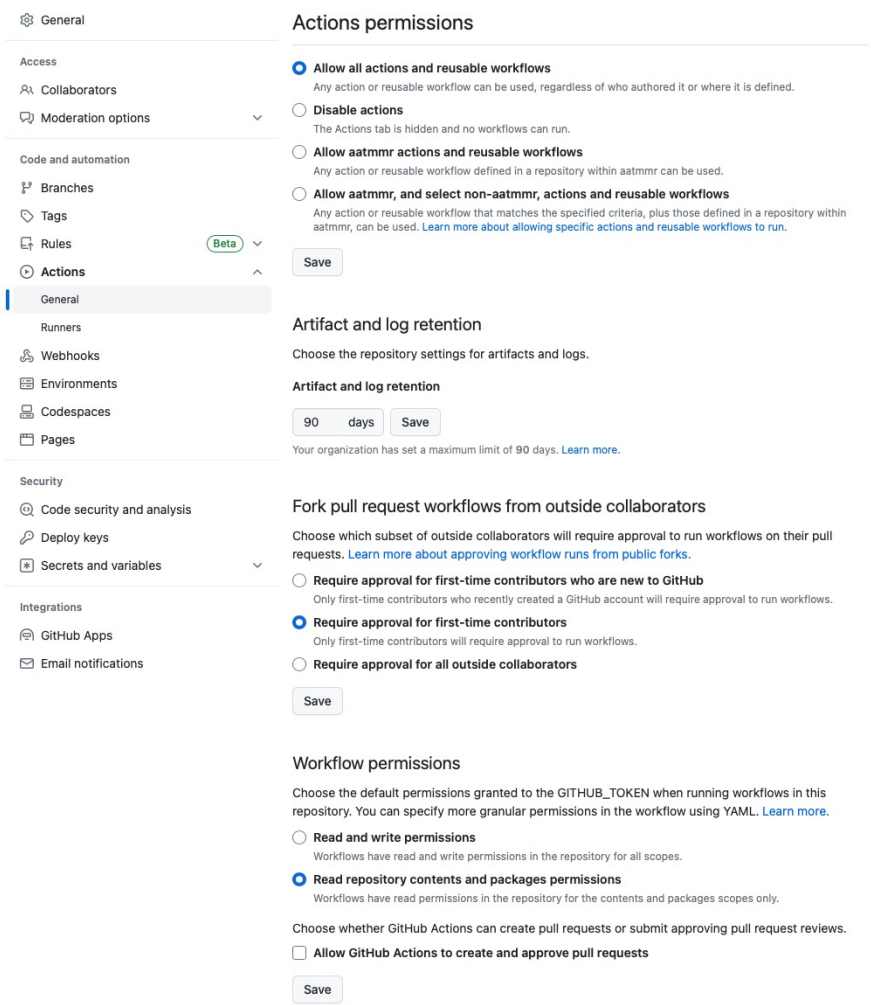
- Use `download-artifact`
- Use `concurrency` groups
- Use staged deployments (**Environments**)
- Pull request: require deployments to succeed before merging
- Use container registry
- Use login actions
- Use OpenID Connect (**OIDC**)



Action Policies

Actions policies

- Configure Actions policies on enterprise / organization / repository level
  - Which Actions are allowed
  - Artifact retention period
  - Running workflows from fork PRs
  - Permissions of `GITHUB_TOKEN`



Demo

Running your workflows

Runners

GitHub-hosted

- Receive automatic updates for the operating system, pre-installed packages and tools, and the self-hosted runner application.
- Are managed and maintained by GitHub.
- Provide a clean instance for every job execution.
- Use free minutes on your GitHub plan, with per-minute rates applied after surpassing the free minutes.

Self-hosted

- Receive automatic updates for the self-hosted runner application only. You are responsible updating the operating system and all other software.
- Can use cloud services or local machines that you already pay for.
- Are customizable to your hardware, operating system, software, and security requirements.
- Don't need to have a clean instance for every job execution.
- Are free to use with GitHub Actions, but you are responsible for the cost of maintaining your runner machines.

GitHub hosted runners

Linux Windows	MacOS
Hardware: • **Standard_DS2_v2** virtual machines in Microsoft **Azure** • 2-core CPU • 7 GB of RAM • 14 GB of SSD disk space	**Hardware**: • 3-core CPU • 14 GB of RAM • 14 GB of SSD disk space
Passwordless sudo / UAC disabled	Passwordless sudo

Runner Images

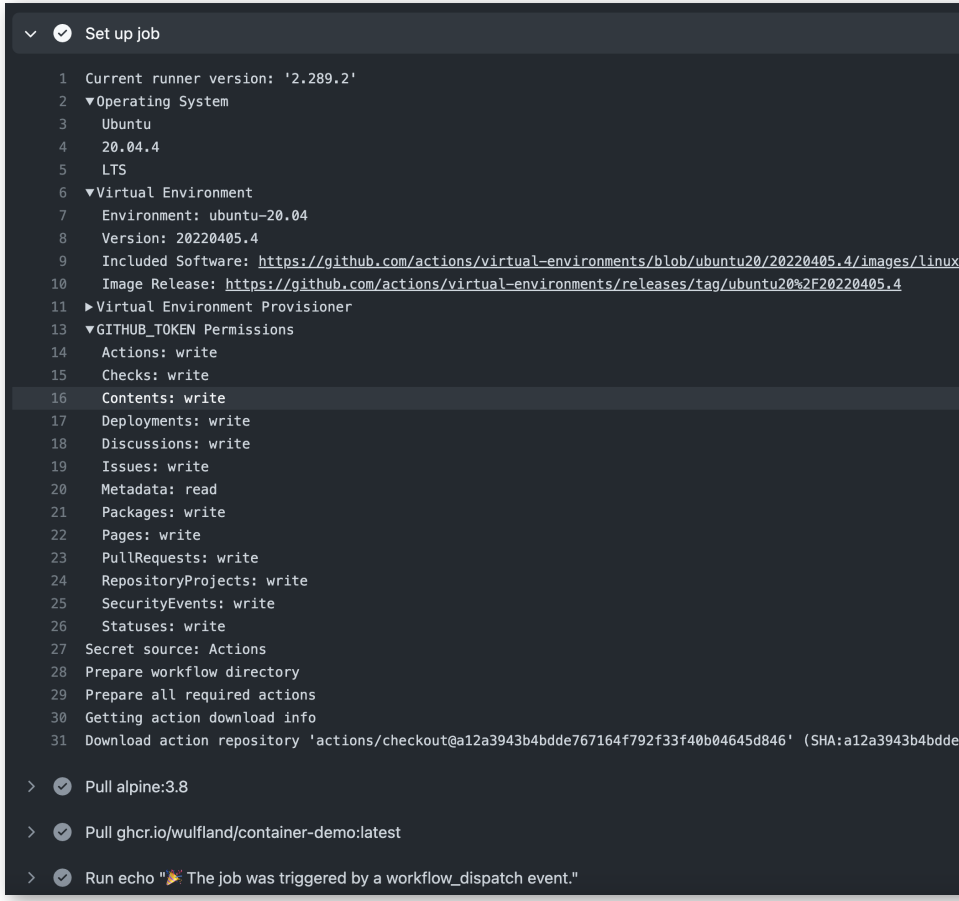
Environment	YAML label	Included Software
Ubuntu 22.04	ubuntu-latest or ubuntu-22.04	ubuntu-22.04
Ubuntu 20.04	ubuntu-20.04	ubuntu-20.04
macOS 11	macos-latest or macos-11	macOS-11
macOS 10.15	macos-10.15	macOS-10.15
Windows Server 2022	windows-latest or windows-2022	windows-2022
Windows Server 2019	windows-2019	windows-2019
Windows Server 2016	windows-2016	windows-2016

https://github.com/actions/runner-images

GitHub hosted runners pricing

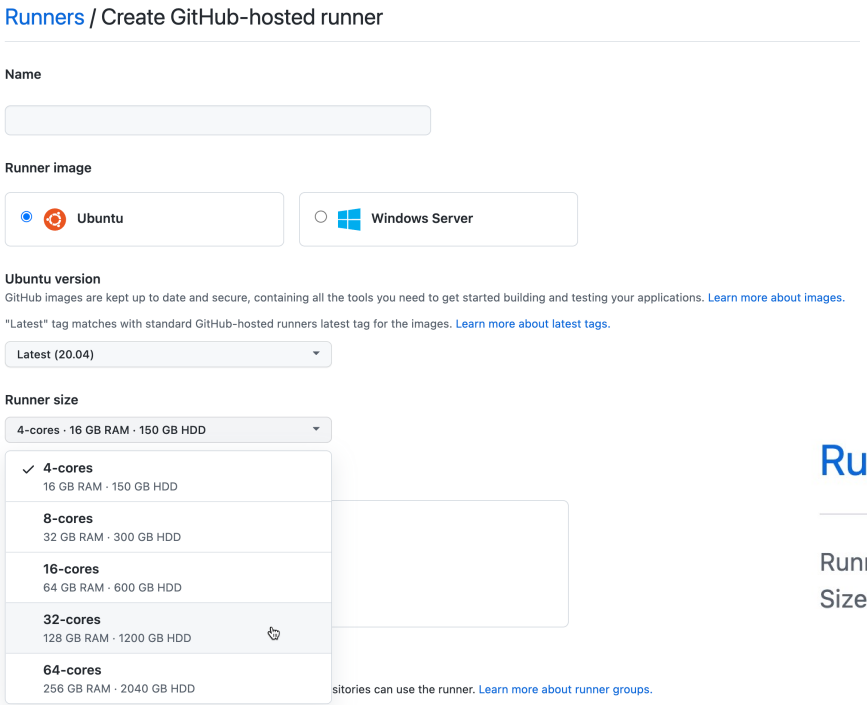
- Build minutes
  - On Linux `$0.008`
  - On Windows `$0.016 = x2`
  - On macOS `$0.080 = x10`

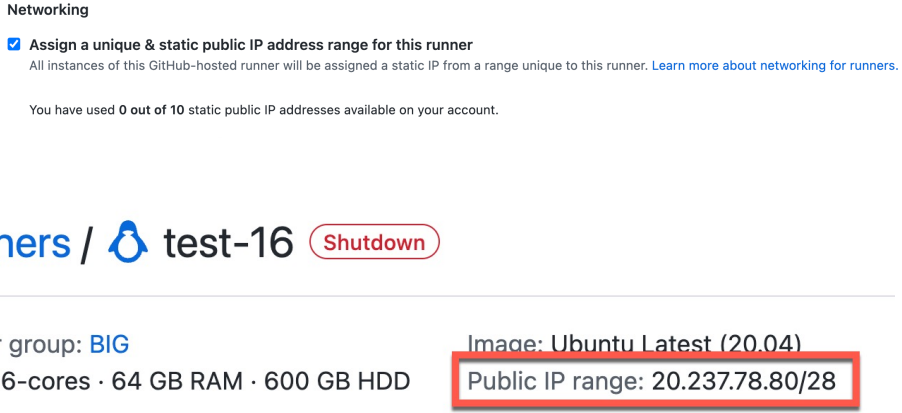
GitHub edition	Storage	Minutes	Max concurrent jobs
GitHub Free	500 MB	2,000	20 (5 for macOS)
GitHub Pro	1 GB	3,000	40 (5 for macOS)
GitHub Free for organizations	500 MB	2,000	20 (5 for macOS)
GitHub Team	2 GB	3,000	60 (5 for macOS)
GitHub Enterprise Cloud	50 GB	50,000	180 (50 for macOS)



https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions

Larger runners (beta)



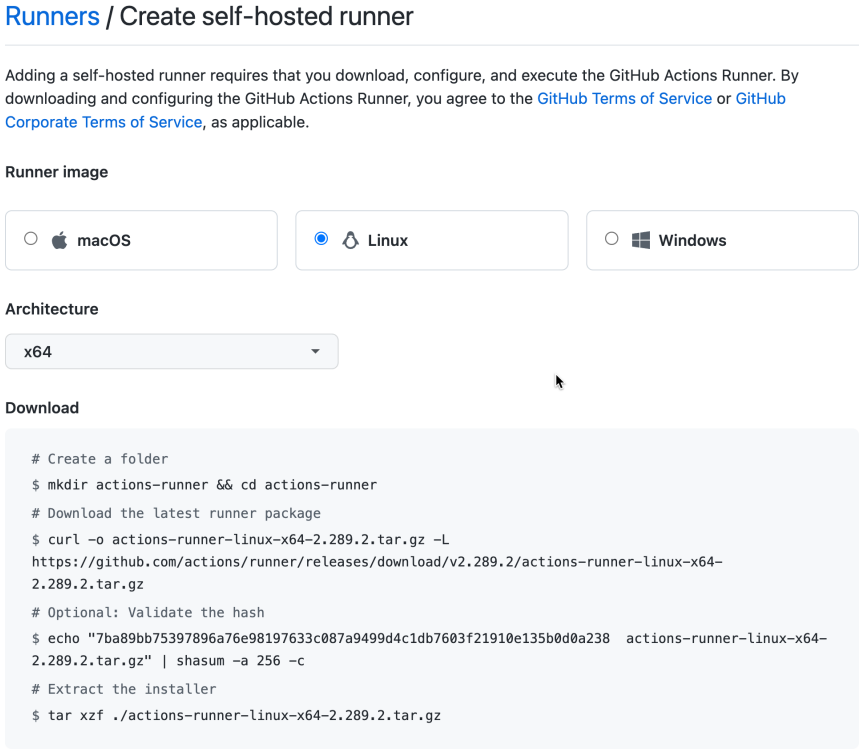


https://docs.github.com/en/actions/using-github-hosted-runners/using-larger-runners

Demo

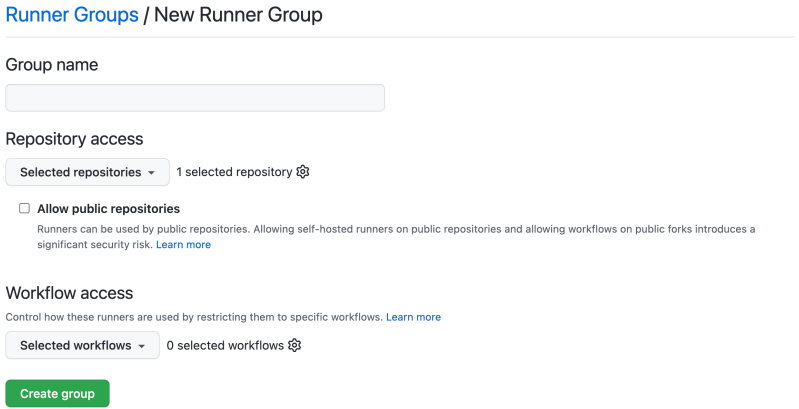
Self-hosted runners

- **Free**
- **Any platform** (x64: Linux, macOS, Windows. ARM64 and ARM32 on Linux)
- **HTTPS long polling** port 443 – 50 seconds
- Can be used to **deploy to local resources**
- Can be added at **Enterprise, Organization, and Repository level**



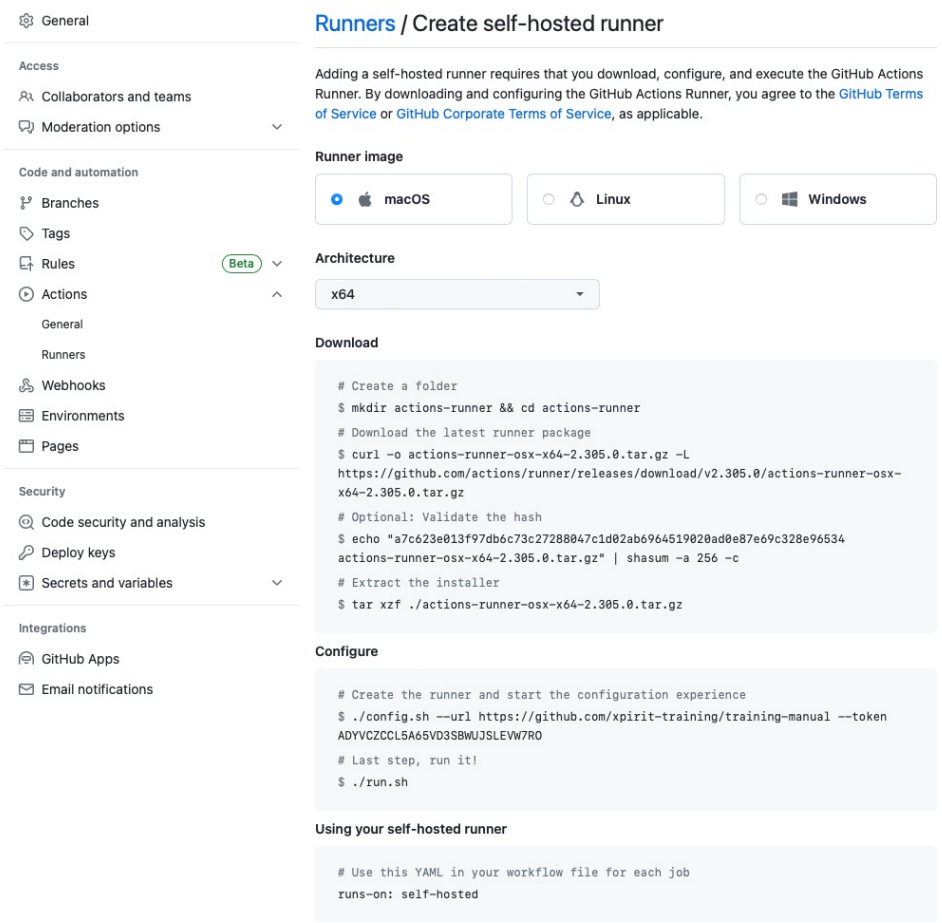
Self-hosted runners

- Access: Runner Groups
- A runner can only be in 1 group
- Can be moved to another group
- Apply labels
  - `$ ./config.sh --labels self-hosted,x64,linux,`**`matlab`**
  - `runs-on: [self-hosted, linux, X64, `**`matlab`**`]`



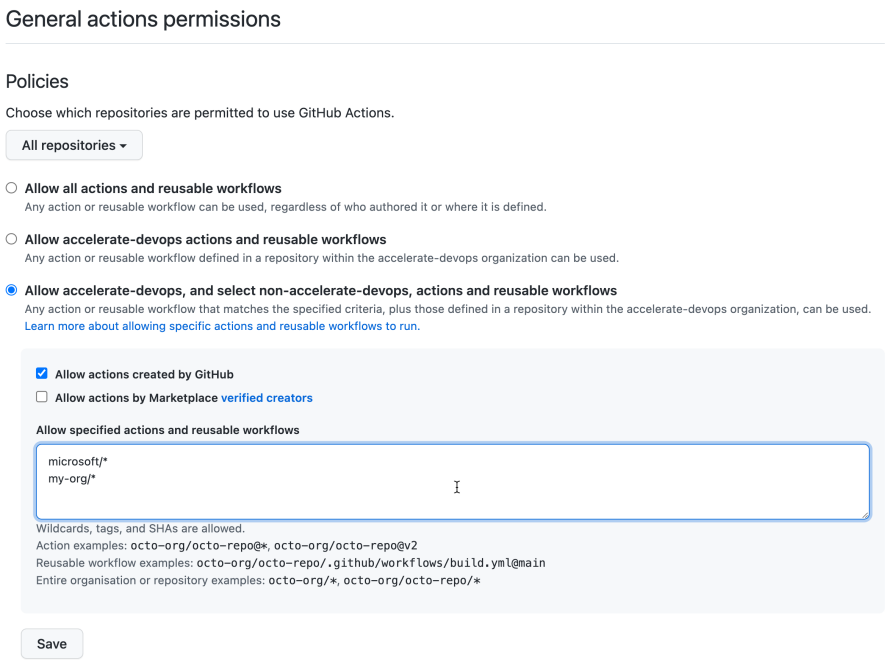
Adding self-hosted runners

- Configure on enterprise / organization / repository level
- Download and extract the scripts
- Configure and authenticate the runner with the token
- Start listening for jobs
- For GHES: Blob storage must be provided (Azure Blob storage, Amazon S3, MinIO)



Self-hosted runners
Gotchas

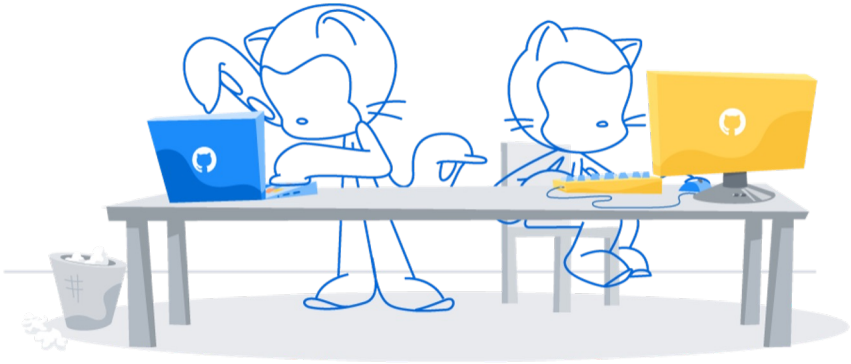
- Runners are not ephemeral per default – you have to clean up after a build yourself
  - `$ ./config.sh --ephemeral`
- Use web hooks to auto scale (github.com/jonico/awesome-runners)
- Do **not** allow public repositories!
- Limit Actions and use SHA or fork
- Create a company marketplace (github.com/rajbos/actions-marketplace)



Security with self-hosted runners

Public repositories with self-hosted runners pose potential risks:

- Malicious programs running on the machine
- Escaping the machine's runner sandbox
- Exposing access to the machine's network
- Persisting unwanted or dangerous data on the machine



Self-hosted runners and Security

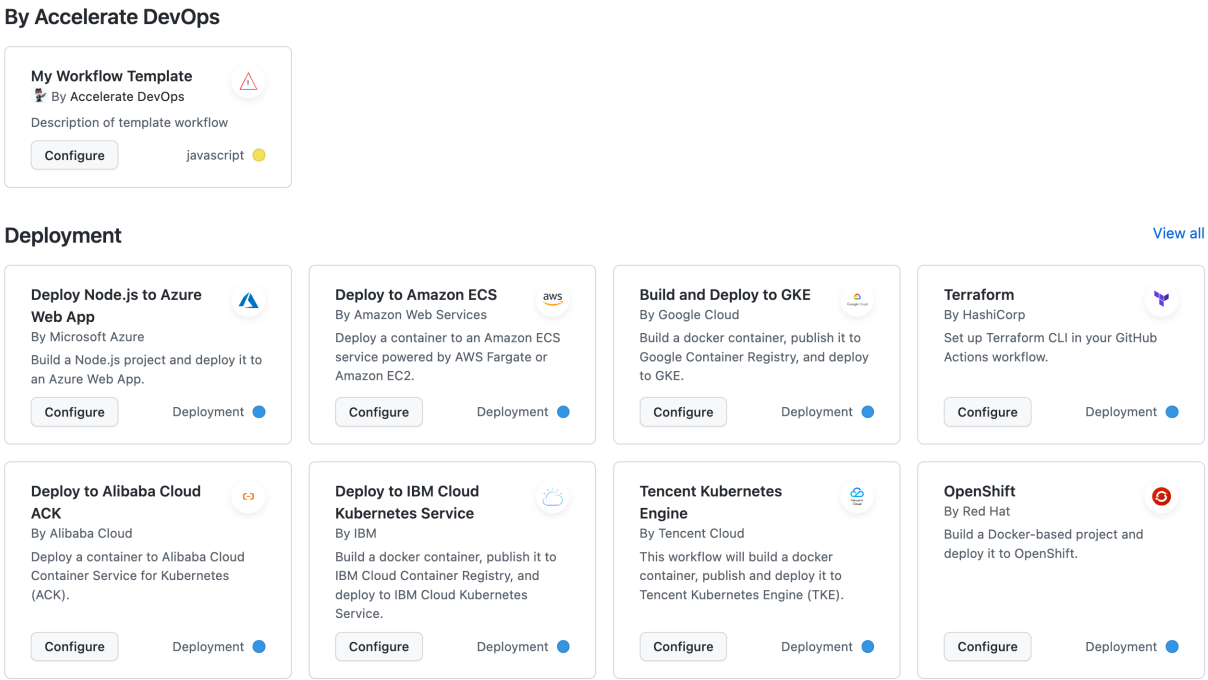
Forked repositories will contain the same Actions configuration as the parent repository, including the self-hosted runners. Creates the potential for a fork to run malicious code on a runner inside your network. For this reason, it is highly recommended to use self-hosted runners only with **private** repositories.

Sharing Workflows

Workflow templates

Workflow templates

- Available in Actions / New workflow
- Get copied one time
- Starter workflows



Workflow templates

- `<org>/.github/workflow-templates`

Reusable Workflows

Reusable workflows

```yaml
name: Reusable workflow

on:
  workflow_call:
    inputs:
      who-to-greet:
        description: 'The person to greet'
        type: string
        required: true
        default: World
    outputs:
      current-time:
        description: 'The time when greeting.'
        value: ${{ jobs.reusable-job.outputs.current-time }}

jobs:
  reusable-job:
    runs-on: ubuntu-latest
    outputs:
      current-time: ${{ steps.time.outputs.current-time }}
    steps:
      - name: Greet someone
        run: echo "Hello ${{ inputs.who-to-greet }}"
      - name: Set time
        id: time
        run: echo "::set-output name=current-time::$(date)"


```

```yaml
name: Reuse other workflow

on: [workflow_dispatch]

jobs:
  call-workflow:
    uses: ./.github/workflows/reusable.yml
    with:
      who-to-greet: '@wulfland'

  use-output:
    runs-on: ubuntu-latest
    needs: [call-workflow]
    steps:
      - run: echo "Time was ${{ needs.call-workflow.outputs.current-time }}"

```

Reusable workflow vs Composite action

Reusable workflow:

- Defines the entire job
- Can enforce runner labels
- No option to do something before and after the steps

Composite action:

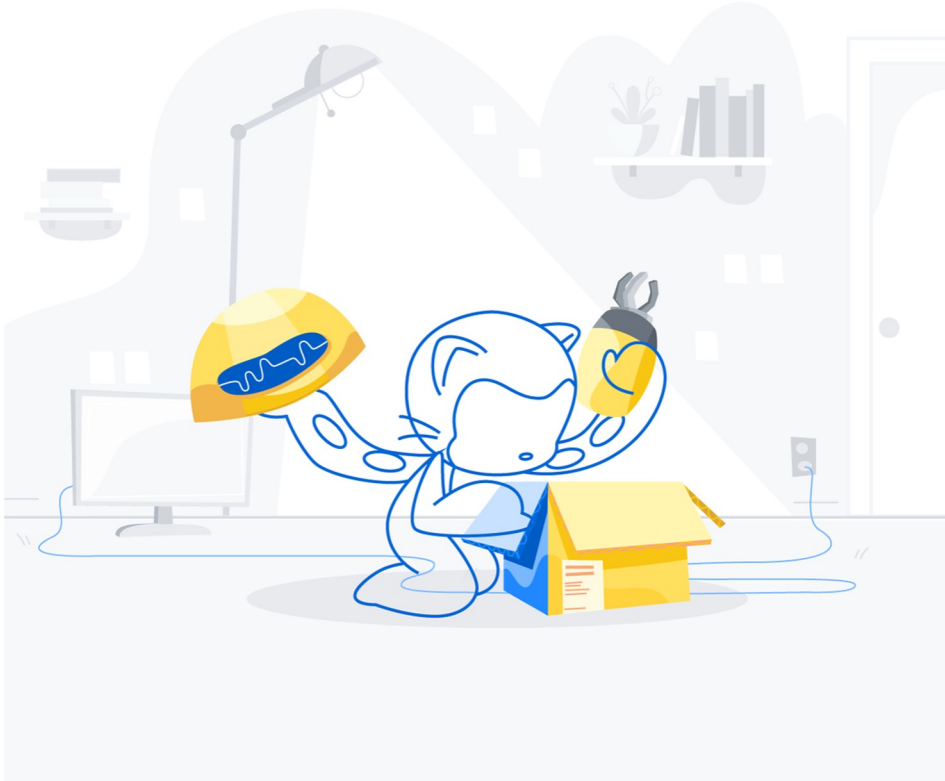
- Defines the list of steps
- Full flexibility to do something before and after the steps in the composite action

Hands-on:
Reusable workflows

Sharing workflows

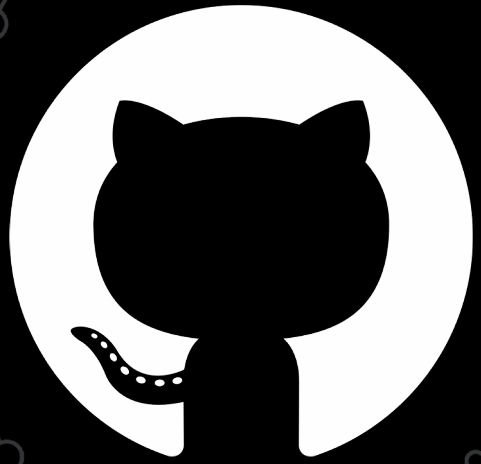
Best Practices

- Use **actions** and composite actions as building blocks
- Use **workflow templates** and **template repositories** for discoverability
- Use **reusable workflows** for complex scenarios
- Share actions and reusable workflows in internal repositories



Q&A

Retro Day 2



Thank you