GitHub Actions Security

NDC { London }

https://myoctocat.com

@robbos81

GitHub Actions Security

NDC { London }

Rob Bos
DevOps Consultant – Xpirit
The Netherlands

https://myoctocat.com

@robbos81

What are GitHub workflows?

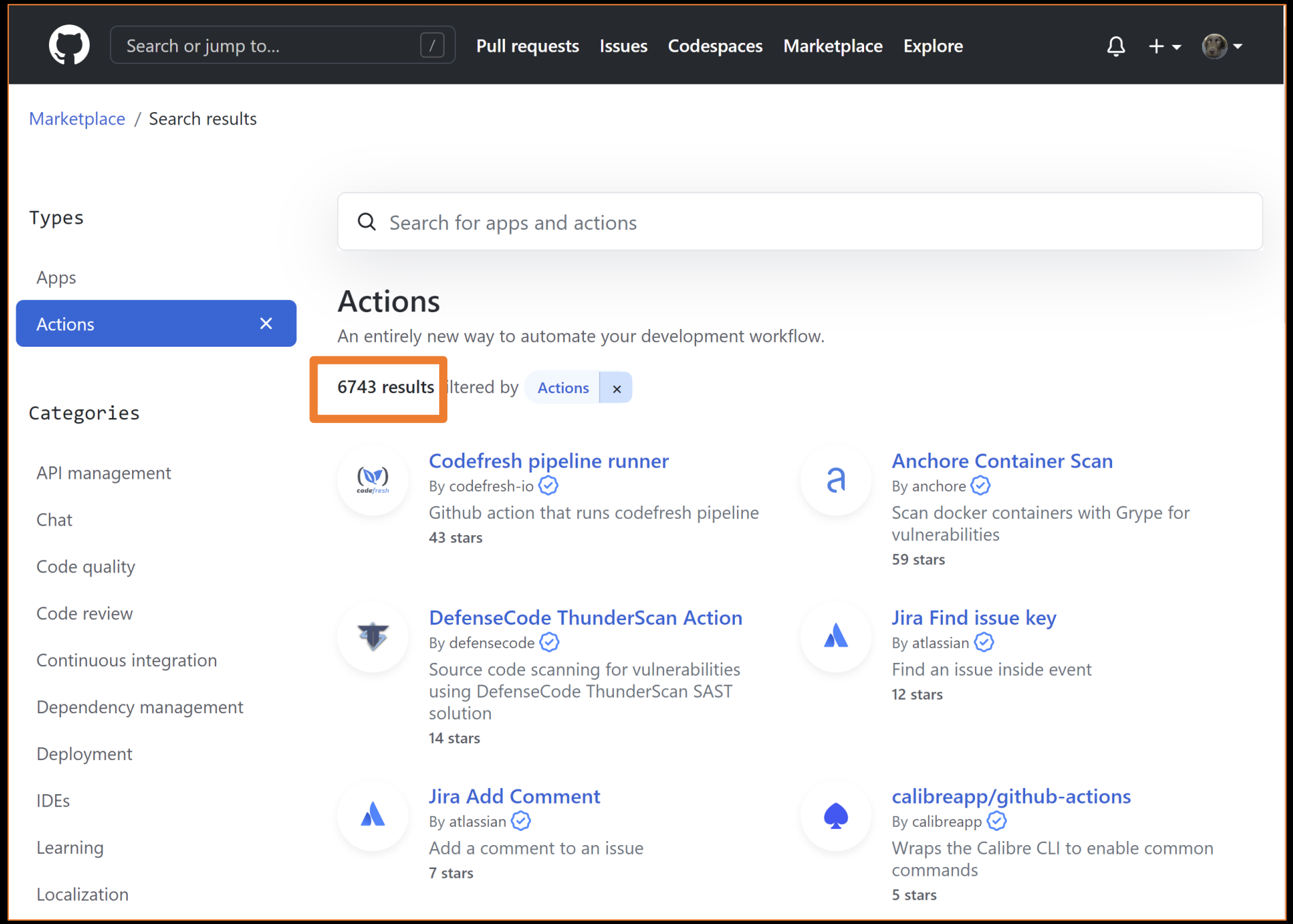
Execute one or more Actions

Workflows triggered by events:

- Push
- Comment
- Creating an Issue
- Release
- Etc.

What are GitHub Actions?

- Steps in the workflows
- Basis: Run a shell script
- Create your own
- Use an existing one from the marketplace


Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
Marketplace / Search results
Types
Apps
Actions
Categories
API management
Chat
Code quality
Code review
Continuous integration
Dependency management
Deployment
IDEs
Learning
Localization
Search for apps and actions
Actions
An entirely new way to automate your development workflow.
6743 results
Filtered by
Actions
Codefresh pipeline runner
By codefresh-io
Github action that runs codefresh pipeline
43 stars
Anchore Container Scan
By anchore
Scan docker containers with Grype for vulnerabilities
59 stars
DefenseCode ThunderScan Action
By defensecode
Source code scanning for vulnerabilities using DefenseCode ThunderScan SAST solution
14 stars
Jira Find issue key
By atlassian
Find an issue inside event
12 stars
Jira Add Comment
By atlassian
Add a comment to an issue
7 stars
calibreapp/github-actions
By calibreapp
Wraps the Calibre CLI to enable common commands
5 stars

Workflow example

main

dotnetcore-webapp / .github / workflows / dotnetcore.yml

```yaml
name: .NET Core

on: [push]

jobs:
  build-and-deploy:
    environment: Production

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v1
    - name: Setup .NET Core
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: 3.0.100

    # dotnet build
    - name: Build with dotnet
      run: |
        dotnet build --configuration Release ./dotnet-core-webapp/dotnetcore-webapp.csproj
```

GitHub Actions Security

- Repository security
- Runners and security
- Actions and security

- Forking actions
- Keeping up to date

Repository security

- Access to code
- Workflow secrets
- Your code

Code – Who has access?

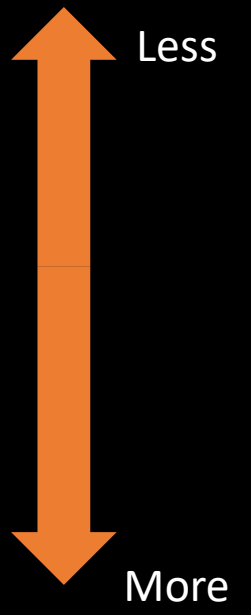
Access levels can be set at:

- Repository
- Organization
- Enterprise

Follow best practices: use teams to group users!

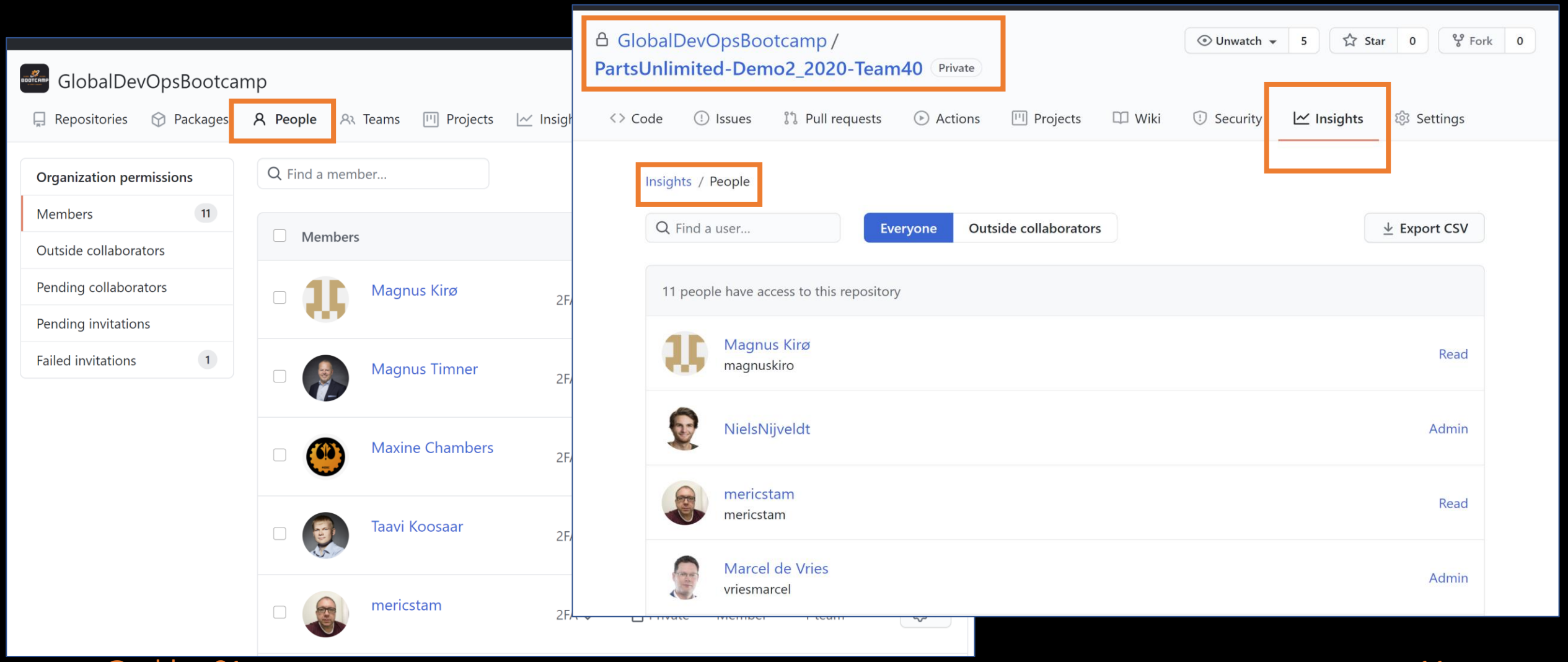
Code – Who has access?

Permission levels

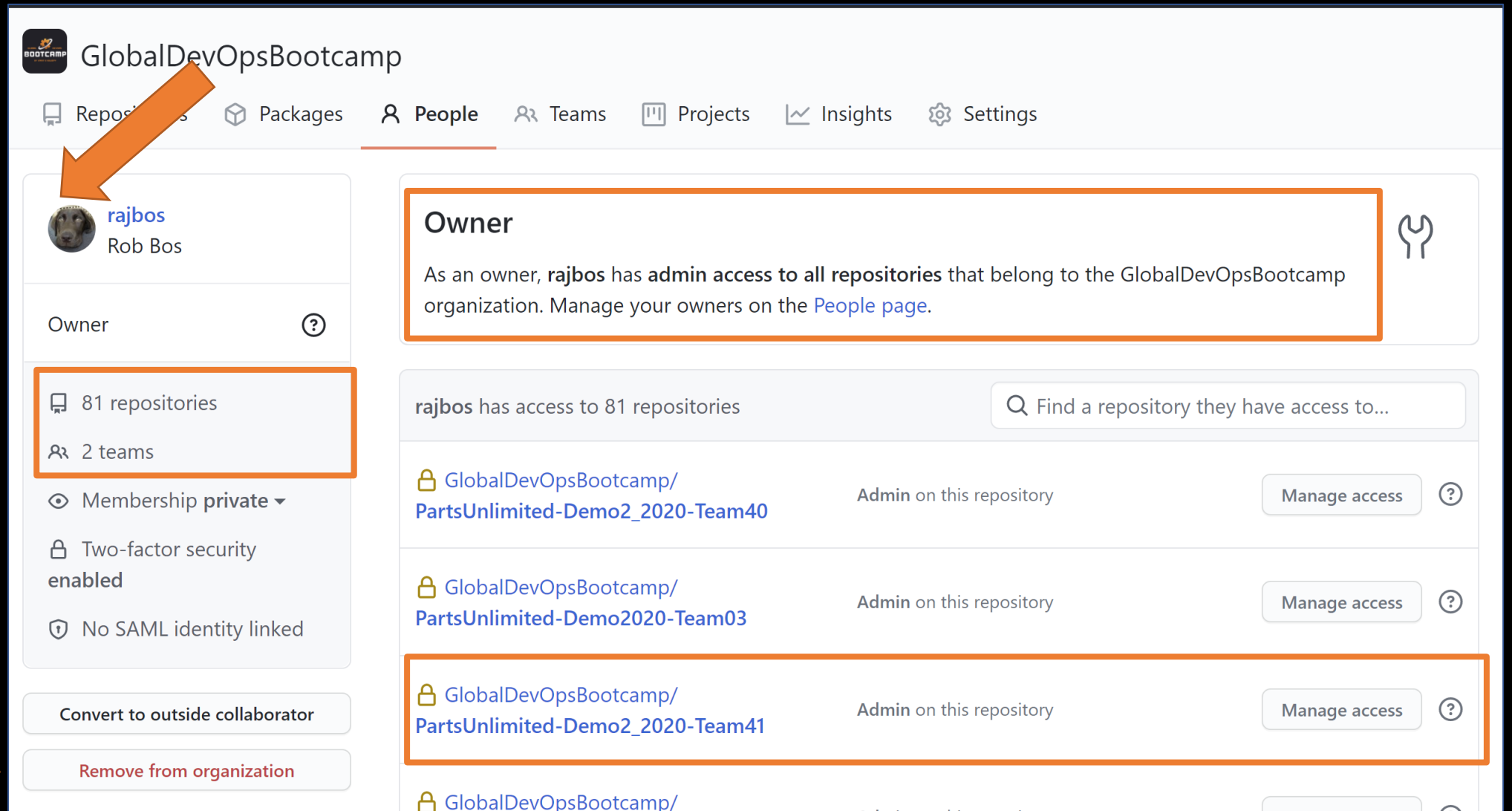


No access
Read only access
Triage: manage issues & pull requests
Write access
Maintain: No sensitive or destructive actions
Admin: full access

Configuring access



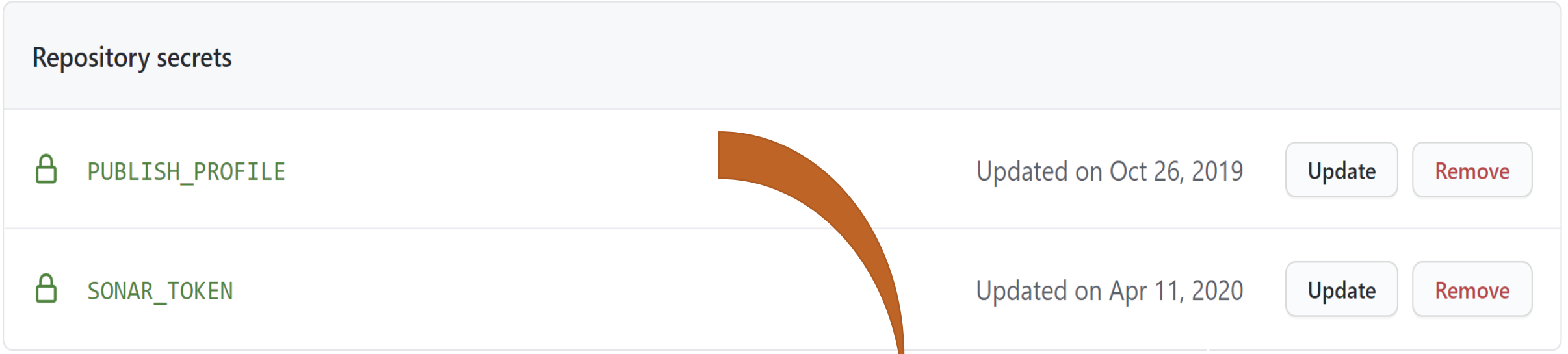
From the user



Repository security

- Access to code
- Workflow secrets
- Your code

Workflow secrets




```
41
42      # publish to Azure App Service
43      - name: 'Run Azure webapp deploy action using publish profile cred ntials'
44        uses: azure/webapps-deploy@v2
45        with:
46          app-name: dotnetcorewebapp19 # Replace with your a
47          publish-profile: ${{ secrets.publish_profile }} #  ine secret variable in repository settings as per action documentation
48          package: './dotnetcorewebapp'
49
```

Workflow secrets

Encrypted client side before reaching GitHub:

- Encrypted with the public key for your org or repo (created and stored by GitHub)
- Used when using the UI
- Encrypt yourself before posting to the REST API

Secrets are not shared to forked repositories

Who has access to your secrets?

For creating at repo level: Repository Owner access

For creating at org level: Admin access to the org

Set an access policy for the secrets:

- All repositories
- Private repositories
- Only selected repositories

Who has access to your secrets?

Encrypted until used, then injected as:

- An environment variable
- Direct input

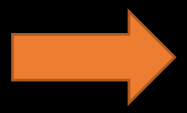
Will be redacted in logs

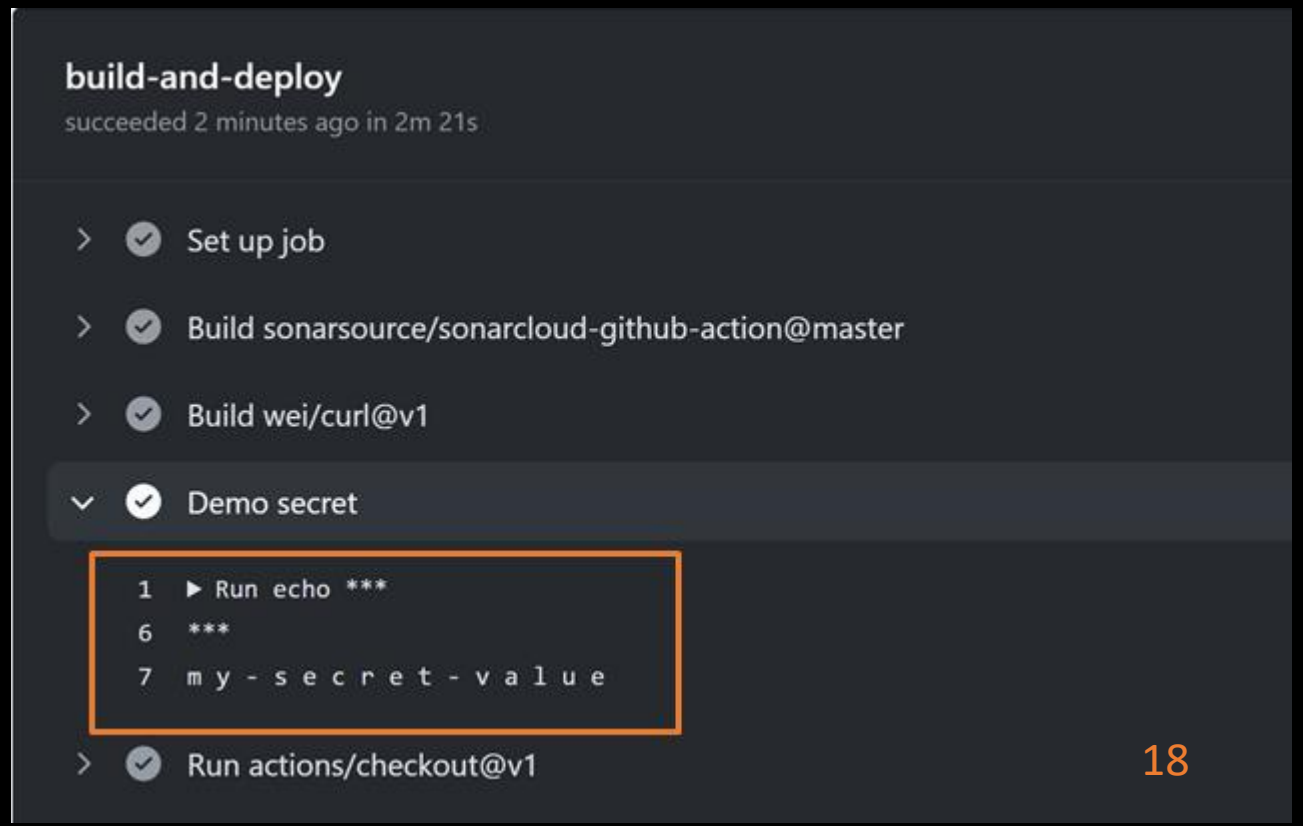
Don’t use structured data (like json): hard to redact

Who has access to your secrets?

- The Action can do anything with them!
- Anyone with access to the Action Logs should be considered to have access to your secrets

```yaml
jobs:
  build-and-deploy:

    runs-on: ubuntu-latest

    steps:
    - name: Demo secret
      run: |
        echo ${{ secrets.DEMO_LOG }}
        echo ${{ secrets.DEMO_LOG }} | sed 's/./& /g'
```





Repository security

- Access to code
- Workflow secrets
- Your code/repo

Your code

Anything in your repository:

- Workflow files
- Shell scripts
- Your own code
- Dependencies:
  - Packages
  - Containers

Best practices:

- Linters
- Static code analysis
  - Check your own code!
- Third party dependency scanning
  - 99% of your code, is not yours:
    - Scan for known vulnerabilities
  - Keep your dependencies up to date!

Your code – Best practices

Preventing changes by a single person → Pull Request

Protect your main branch

Branch protection rule:

- 1 approver (4 eyes principle)
- Build validation

Your code/repo – trace changes

Who made changes:

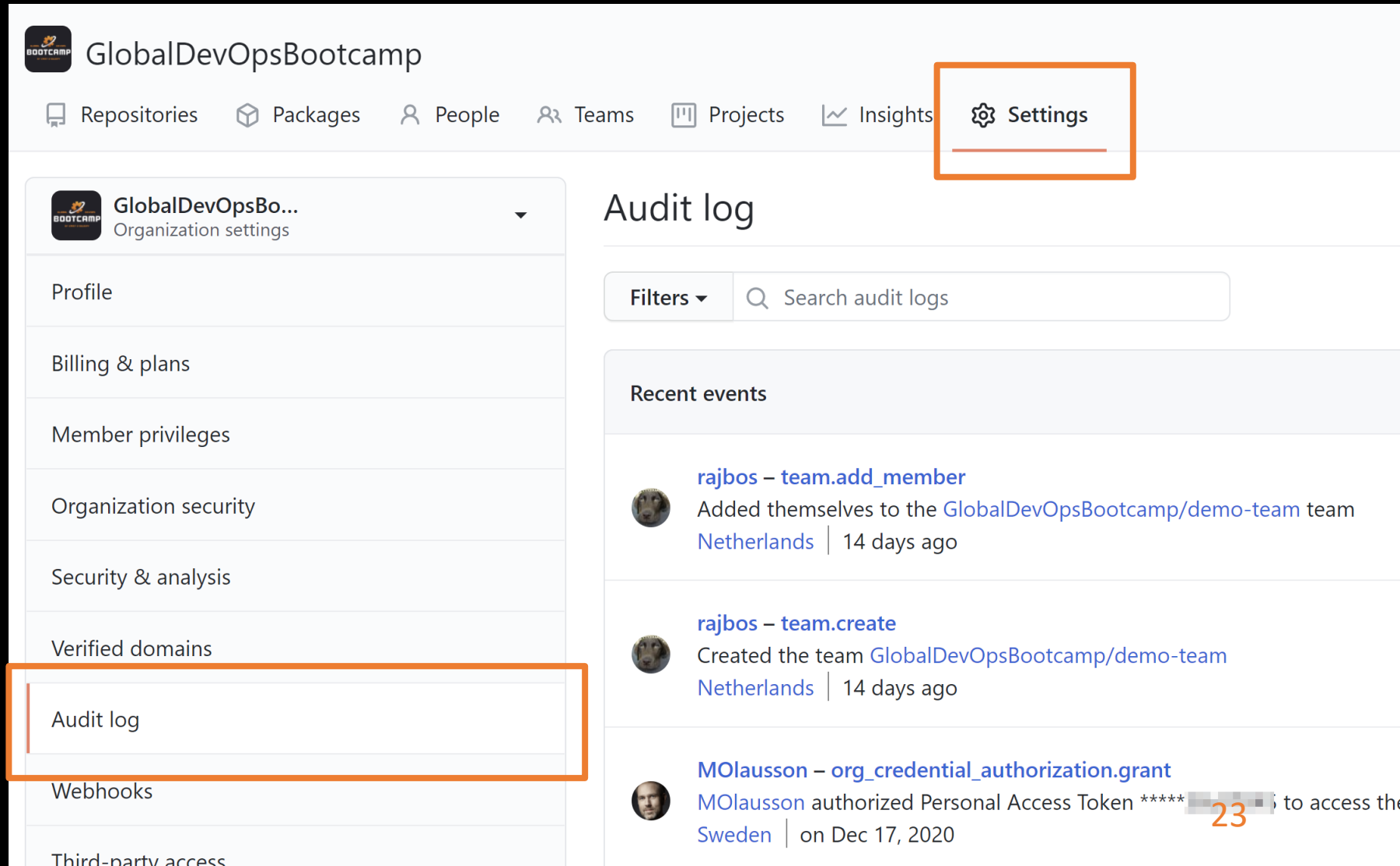
- Code: Git commit history

- Everything around your code is in the audit log

Your code/repo – trace changes (org level)

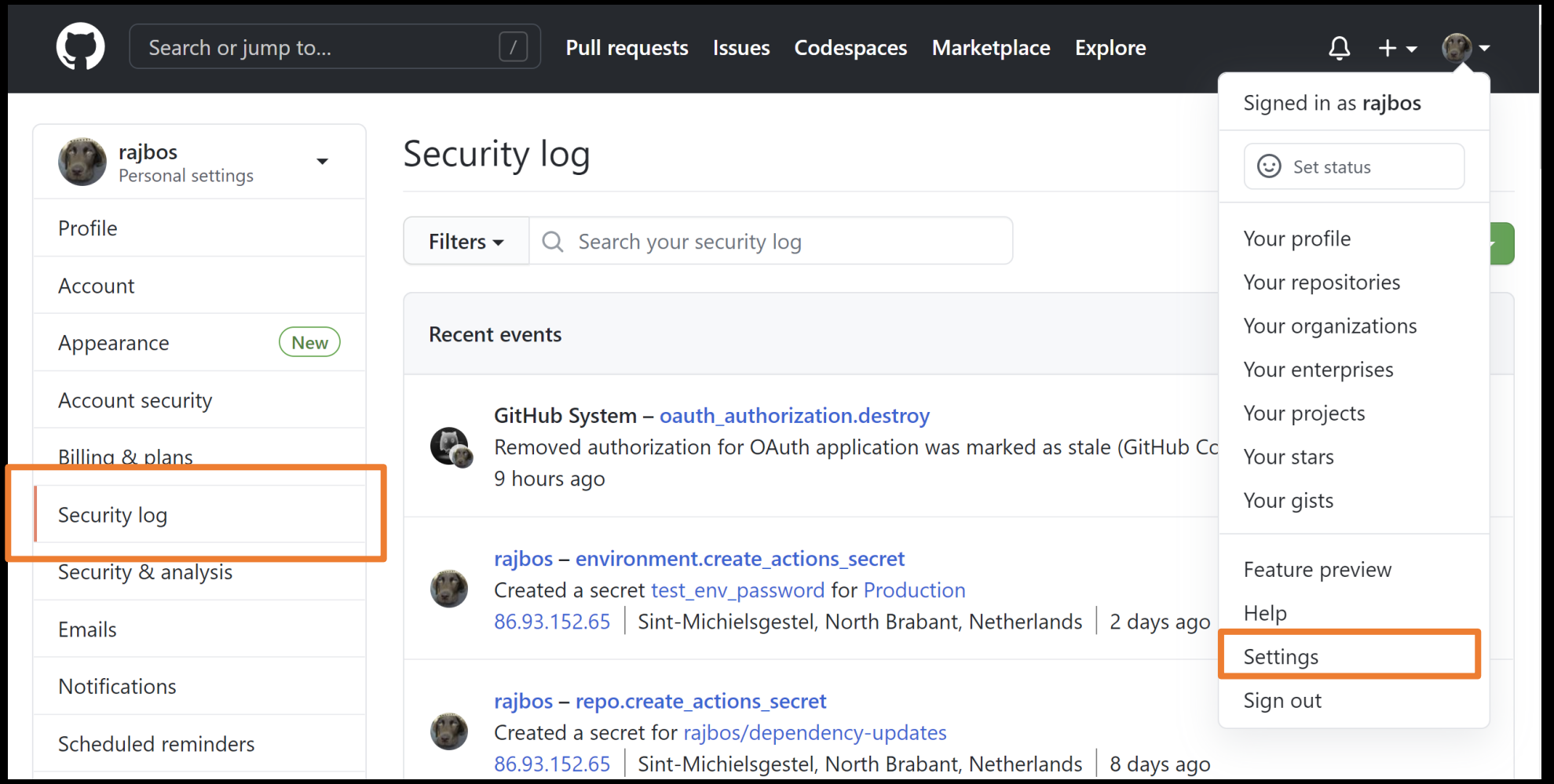
Audit log:

- Access
- Secrets
- Access Tokens
- OAuth grants
- Enabling features
- Etc.



Your code/repo – trace changes

Account level:



GitHub Actions Security

- Repository security
- Runners and security
- Actions and security

- Forking actions
- Keeping up to date

Workflow Runners

Actions execute on runners

GitHub hosted

- OS + Tools update = GitHub's responsibility
- Per minute rating applies after the free minutes
- Clean execution environment with every run

Self hosted

- Cloud / On premises hosted by yourself
- OS + Tools update = YOUR responsibility
- Enables specific environment setup
- No usage limits

```yaml
name: .NET Core

on: [push]

jobs:
  build-and-deploy:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v1
    - name: Setup .NET Core
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: 3.0.100
```

```yaml
name: .NET Core Deploy to IIS

on:
  push:
    branches:
      - "self-hosted"

jobs:
  build-and-deploy:

    runs-on: self-hosted

    steps:
    - uses: actions/checkout@v1
    - name: Setup .NET Core
      uses: actions/setup-dotnet@v1
      with:
```

Self-hosted runners

https://github.com/actions/runner

Download

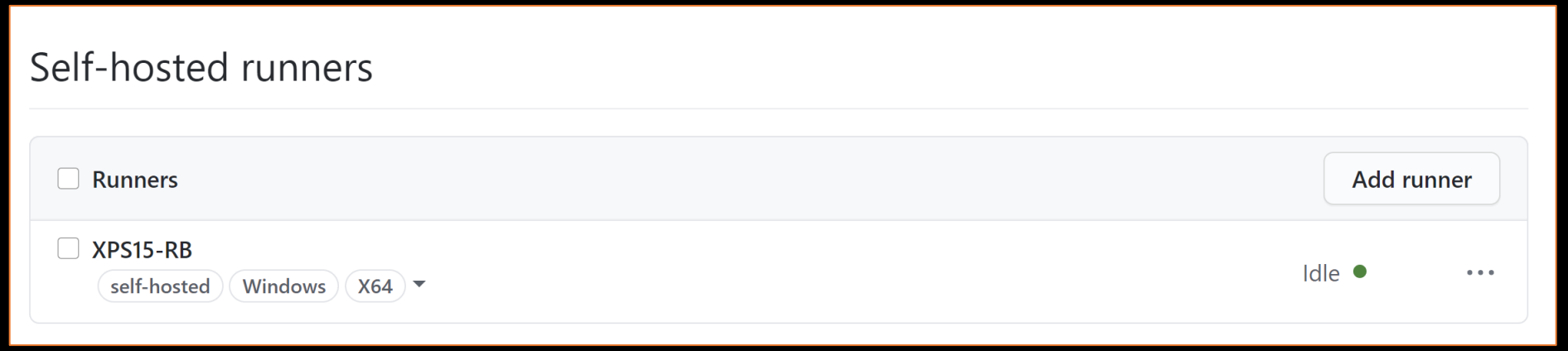
We recommend configuring the runner under "\actions-runner". This will help avoid issues related to service identity folder permissions and long path restrictions on Windows.

```
# Create a folder under the drive root
$ mkdir actions-runner; cd actions-runner
# Download the latest runner package
$ Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2.275.1/actions-runner-win-x64-2.275.1.zip -OutFile actions-runner-win-x64-2.275.1.zip
# Extract the installer
$ Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.ZipFile]::ExtractToDirectory("$PWD/actions-runner-win-x64-2.275.1.zip", "$PWD")
```

Configure

```
# Create the runner and start the configuration experience
$ ./config.cmd --url https://github.com/rajbos/dotnetcore-webapp --token ABONY4[redacted]OBQG
# Run it!
$ ./run.cmd
```

Self hosted runners



Automatically updates itself when a new workflow is started or within a week after a new release

Self hosted runners

- Starts a HTTPS long poll connection to GitHub asking if there is any work and polls every minute (so outgoing connection)

Can be created at these levels:

- Repository: dedicated to a single repository.
- Organization: multiple repositories in an organization.
- Enterprise: multiple organizations in an enterprise account.

Workflow Runners

Security

- To what parts of your environment could a runner potentially have access?
- Think of network and shared state between runs

- User: limit its access! (don’t run as administrator / root when not needed)

Best practice: Run the action inside of a container

```yaml
jobs:
  my_first_job:
    steps:
      - name: My first step
        uses: docker://gcr.io/cloud-builders/gradle
```

```yaml
jobs:
  test-box:
    runs-on: ubuntu-latest
    container:
      image: azul/zulu-openjdk-alpine:8-jre
    steps:
    - uses: actions/checkout@v2
    - name: What OS is running
      run: uname -a
    - name: What java version do we have
      run: java -version
```

Workflow runners

Best practice: Don’t use self hosted runners for public repositories

Example:

- Your repo
- New fork
- Adds malicious code
- Create pull request to your repo
- Workflow is executed on your self hosted runner?

Persisting data between runs

Run 1:

- Download dependencies
- Build the code
- Somehow overwrite the dependency cache

Run 2:

- Use cached dependencies
- Build the code
- Malicious dependency in build artefact

Solarwind attack:
http://xpir.it/Solorigate

Workflow runners – Best practice

Don't share runners (and machines!) between repositories:

- Run 1 can influence Run 2

Risks:

- Malicious programs
- Escaping the runner sandbox
- Exposing access to the (network) environment
- Persisting unwanted or dangerous data

Workflows

What happens when someone pushes code into your repository?

Which workflows are triggered? By which branch?

GitHub Actions Security

Repository security

Runners and security

Actions and security

Forking actions

Keeping up to date

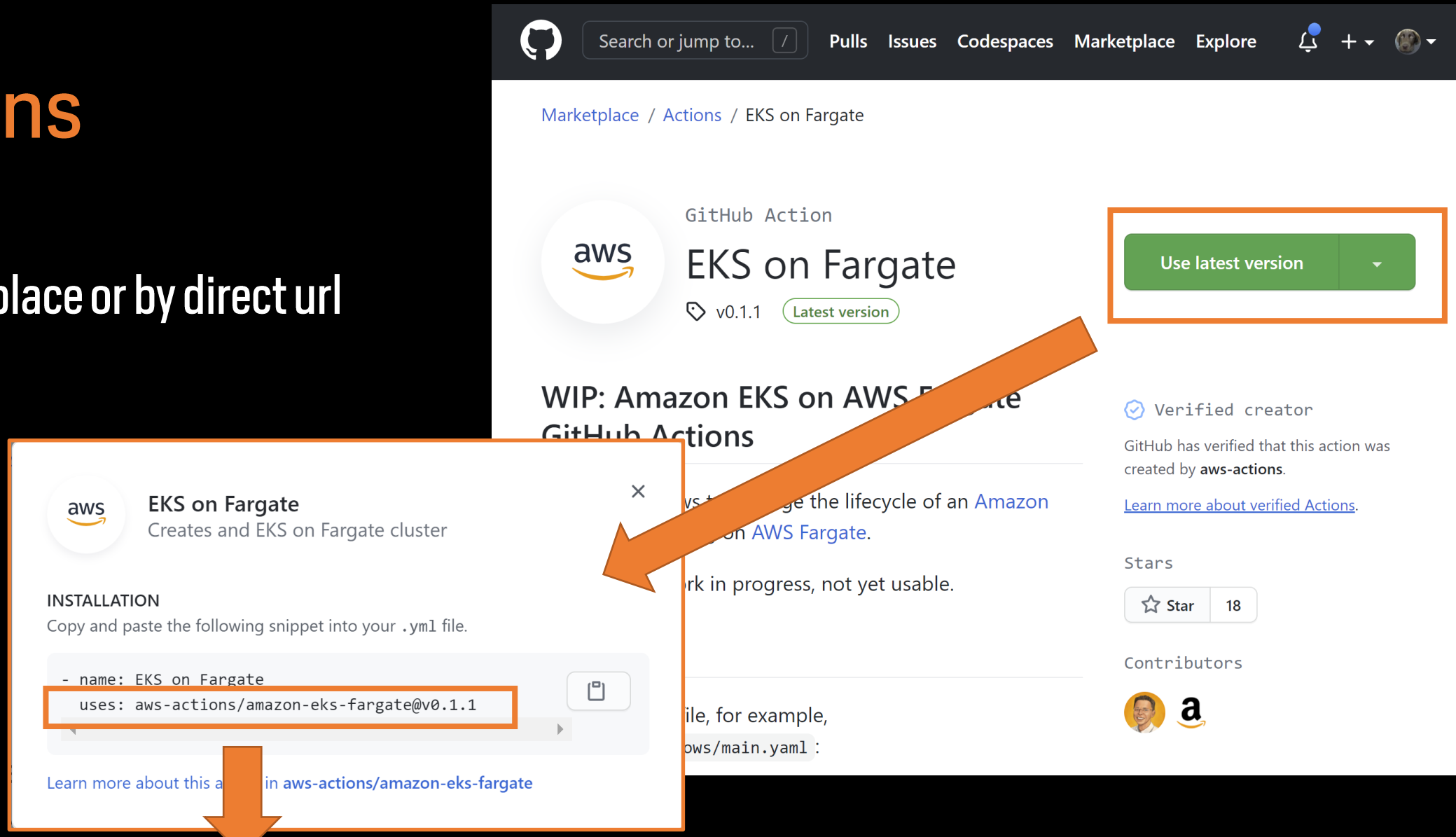
Actions and security

Are you running just any action from the internet?

SCARY, especially in an Enterprise or on local runners

Actions

- Marketplace or by direct url



https://github.com/aws-actions/amazon-eks-fargate

Attack vectors

1. **Data Theft**
2. **Data Integrity Breaches**
3. **Availability**

Protective measures

Manually:

1. Check the action repo code before use
2. Check its container images and dependencies before use

Protective measures

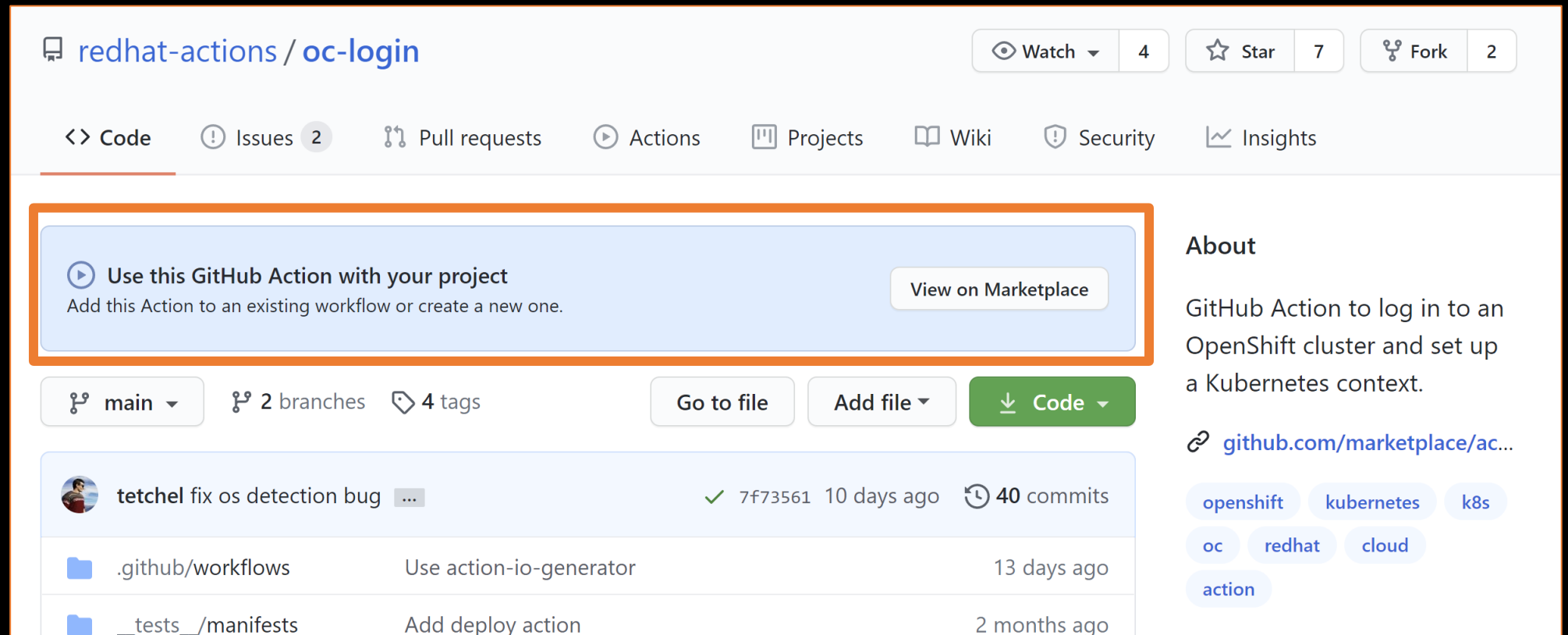
```yaml
uses: shprink/nonharmful-and-must-have-actions@v1
with:
  my-secret: ${{ secrets.MY_SECRET }}
```

https://github.com/shprink/nonharmful-and-must-have-actions

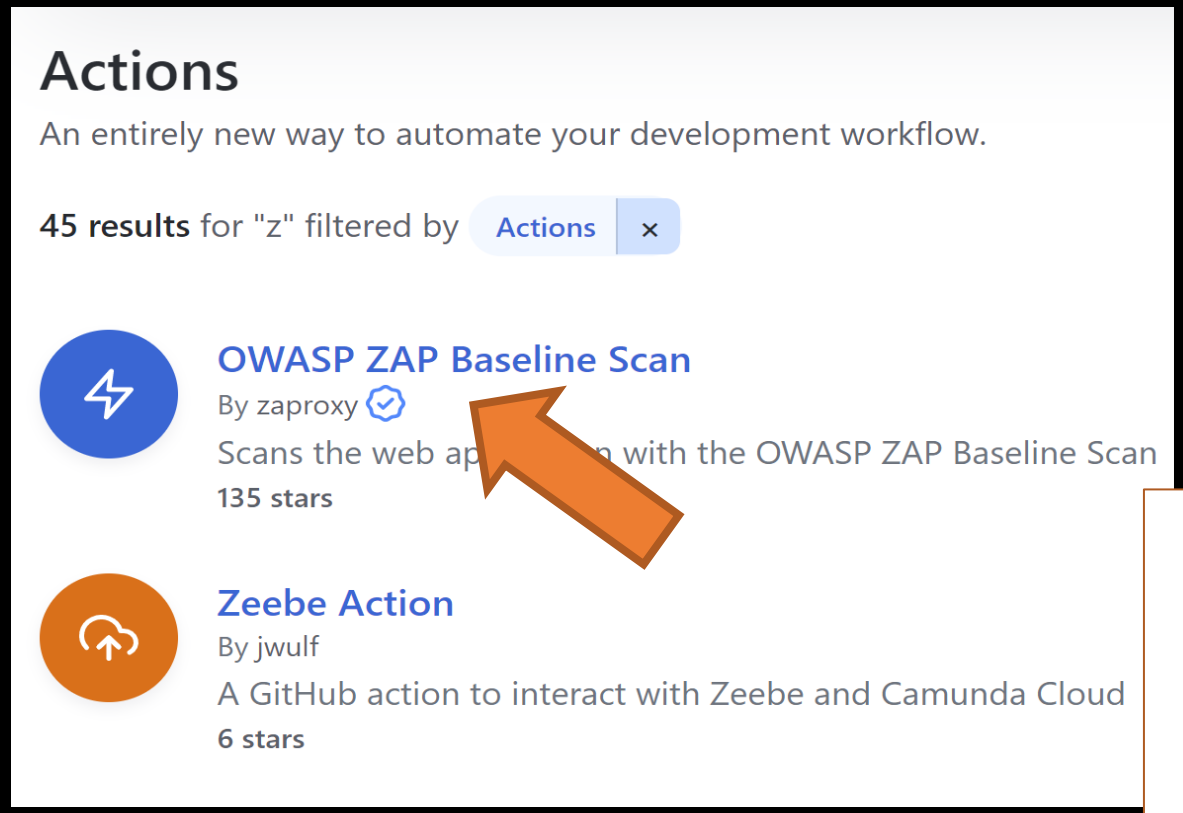
If the repo has an action.yml, you can use it in your workflow

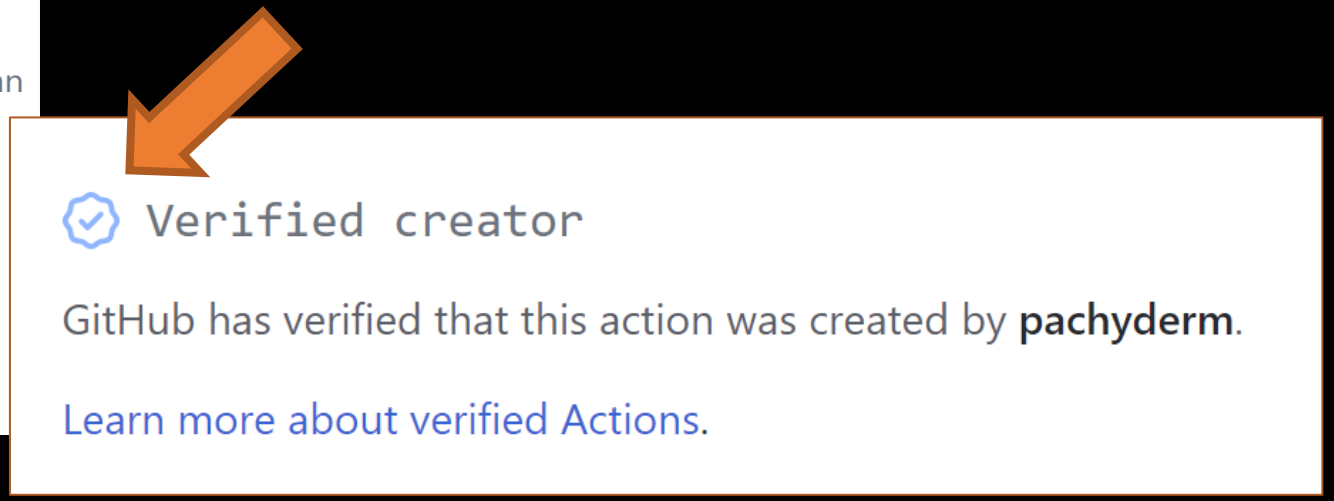
Protective measures

- Only use actions listed in the marketplace?
  - There is no real verification process for it ☹



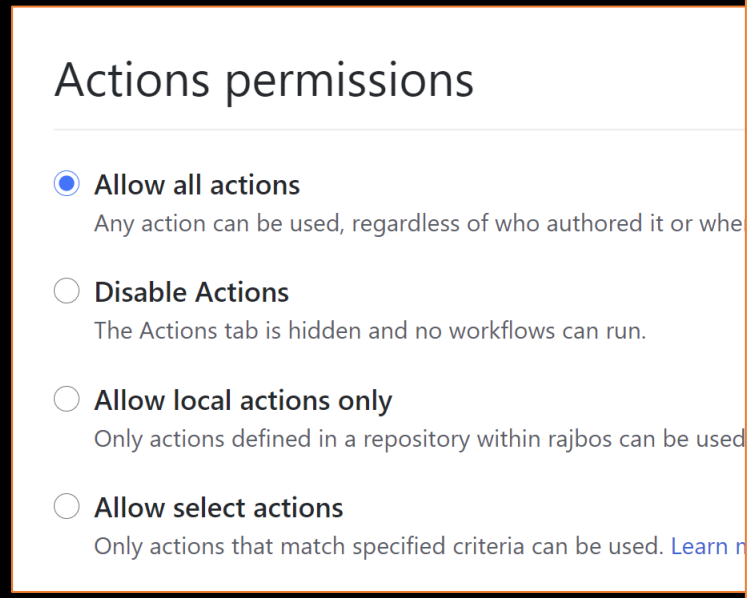
Protective measures

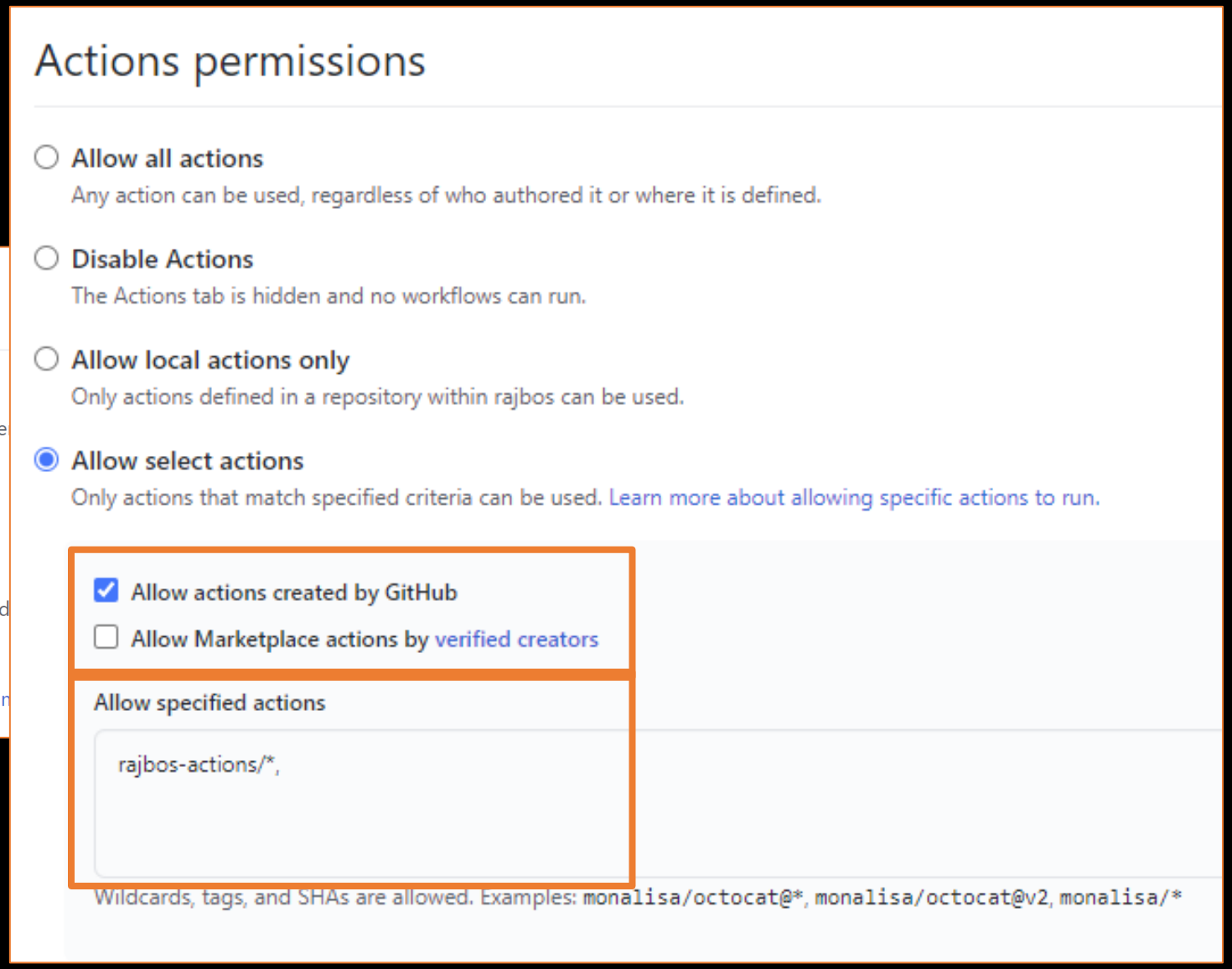




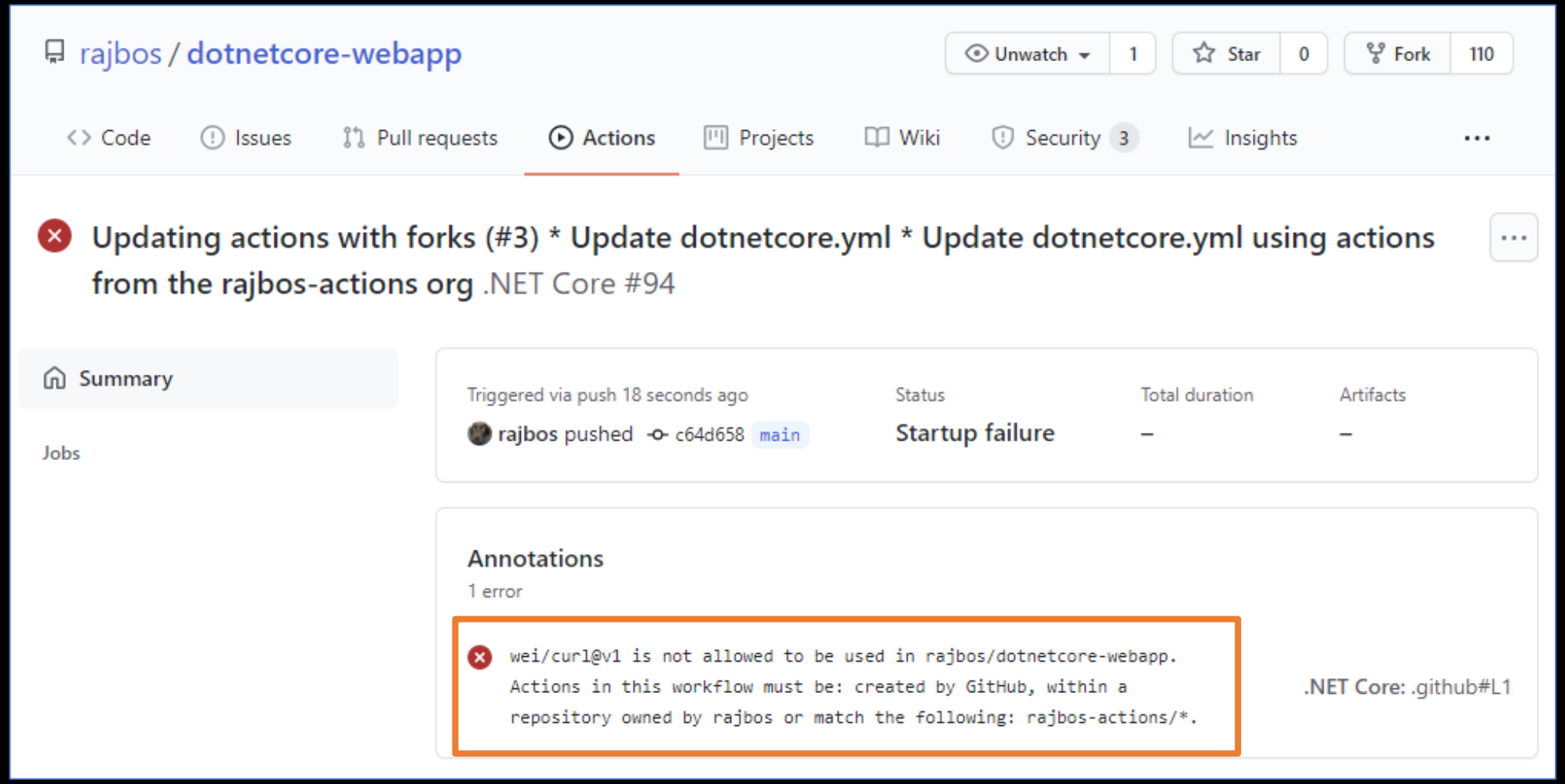
Protective measures

Limiting actions altogether





Protective measures



Protective measures

Pin the action version:

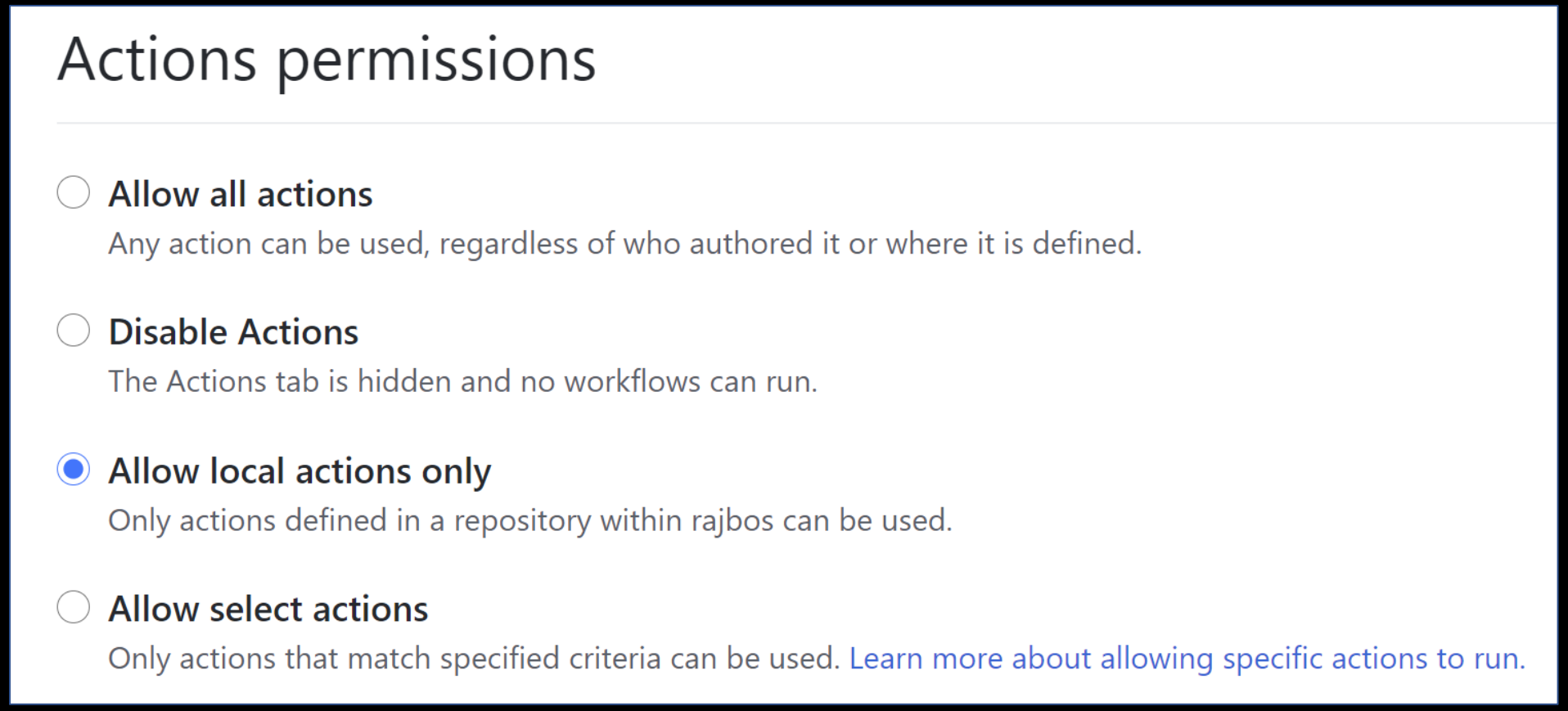
uses: gaurav-nelson/github-action-markdown-link-check@v1

uses: gaurav-nelson/github-action-markdown-link-check@v1.0.1

Best practice: Pin the Action's commit SHA:

uses: gaurav-nelson/github-action-markdown-link-check@44a942b2f7ed0dc101d556f281e906fb79f1f478

Protective measures



Recommendation

- Best practice: Limit to local actions and fork action repositories

- Also create a separate org to test actions in, before forking them
  - To enable DevOps teams to have the autonomy to test and verify themselves

GitHub Actions Security

Repository security

Runners and security

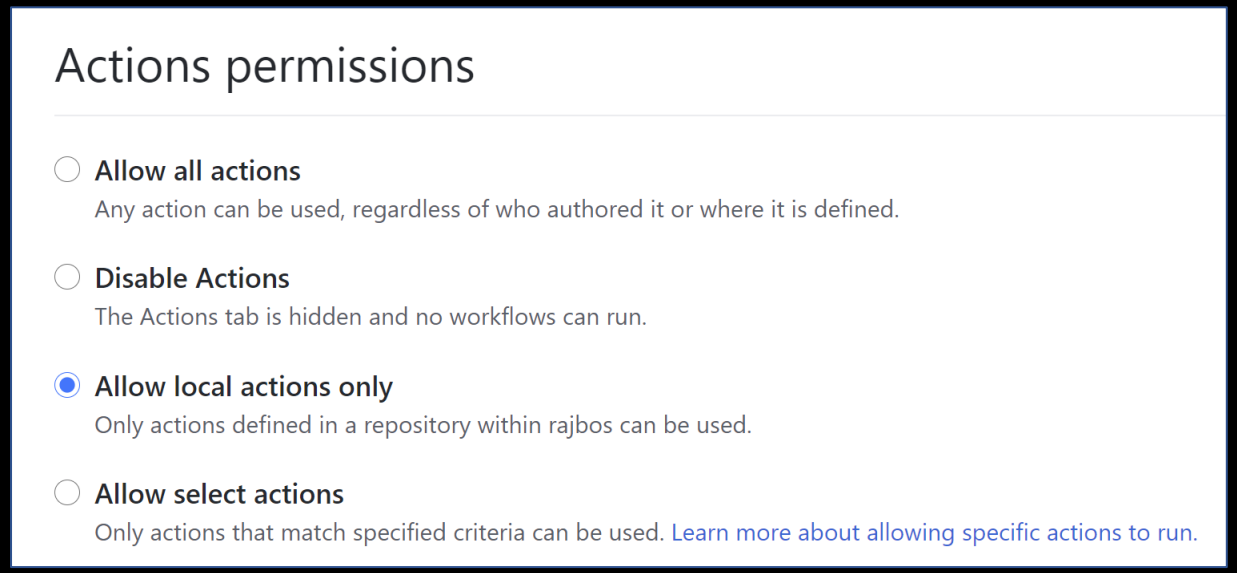
Actions and security

Forking actions

Keeping up to date

Forking actions

- Best practice: fork the action to a local (org) repo
- Limit actions to only local actions



Forking actions

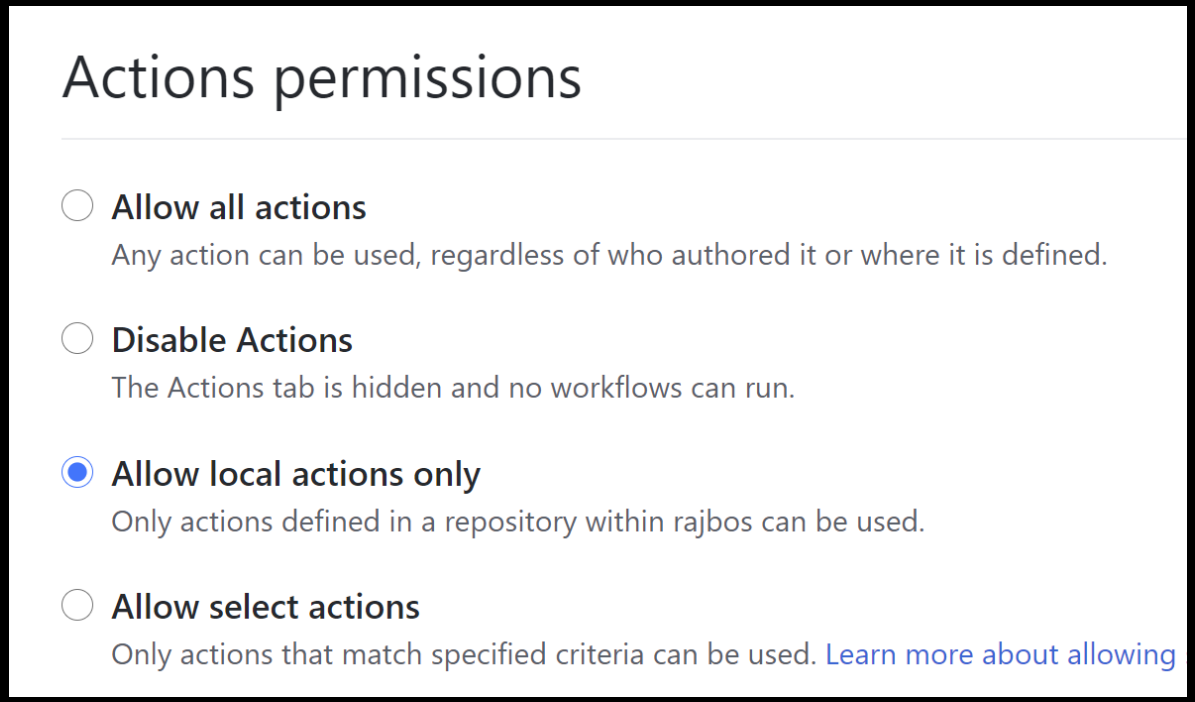
Pros:

- More secure
- Backup of actions that can be deleted or moved to a different org/repo

Cons:

- More maintenance work
  - Fork needs to be created
  - Kept up to date
- Limits the usage of new actions in your org, as someone create the new action (and by that take responsibility for enabling its use)

Enable DevOps teams to test actions



Be aware: setting this on the organization, will prevent any repository in that org to change that setting to all actions.

Create a test org so teams can test other actions to prevent them from being completely blocked.

GitHub Actions Security

Repository security

Runners and security

Actions and security

Forking actions

Keeping up to date

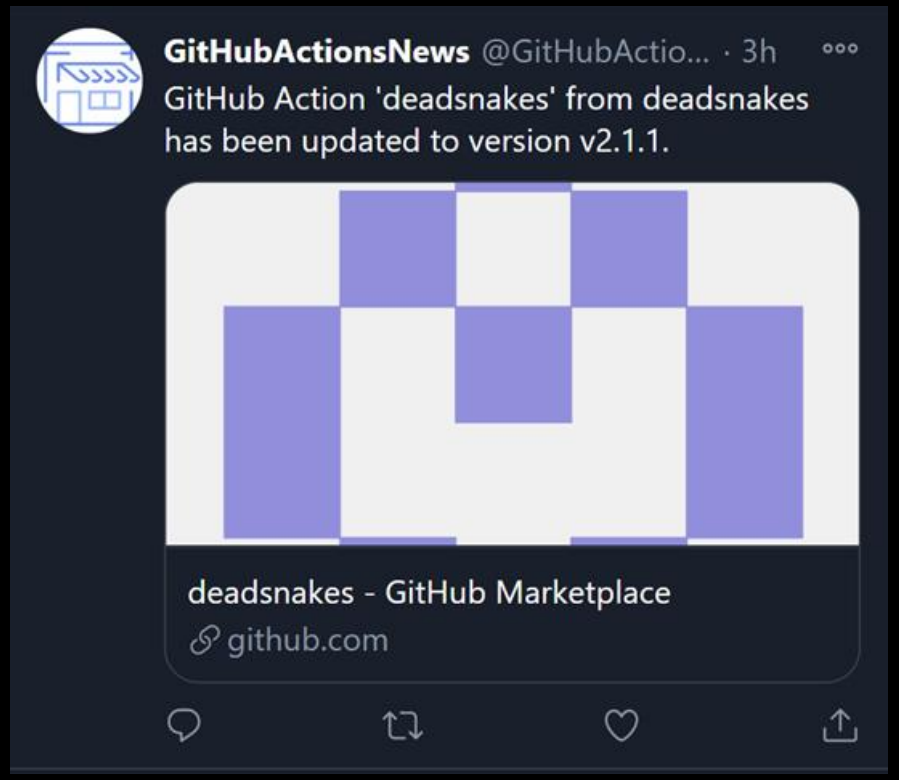
Updates

- Actions are updated regularly
- Wait for a deprecation message?
- How do you stay up to date?

- Auto update with a PR?
- Read the changes in the source repo

Staying up to date

Follow @githubactions on Twitter!

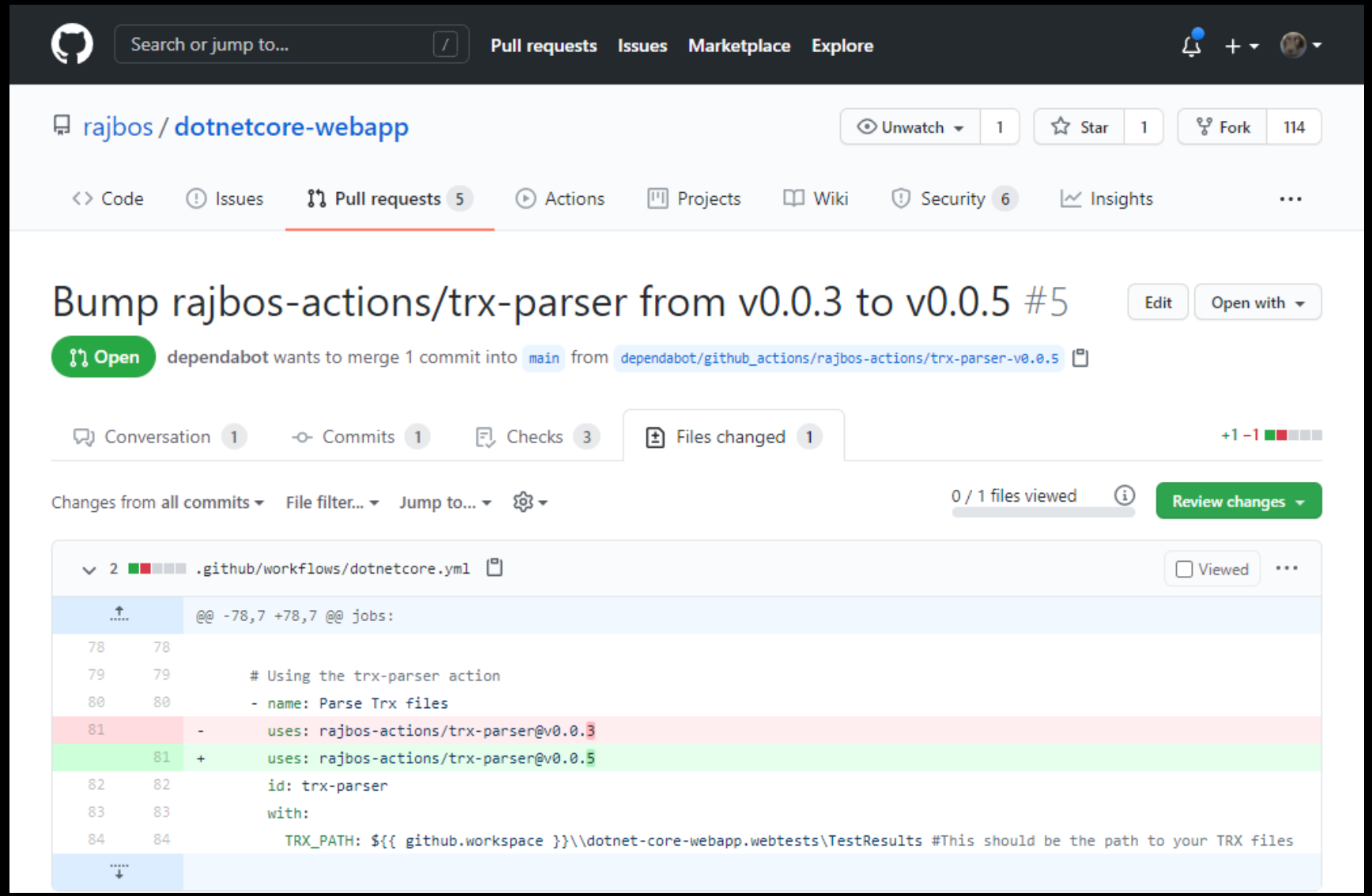


Use Dependabot

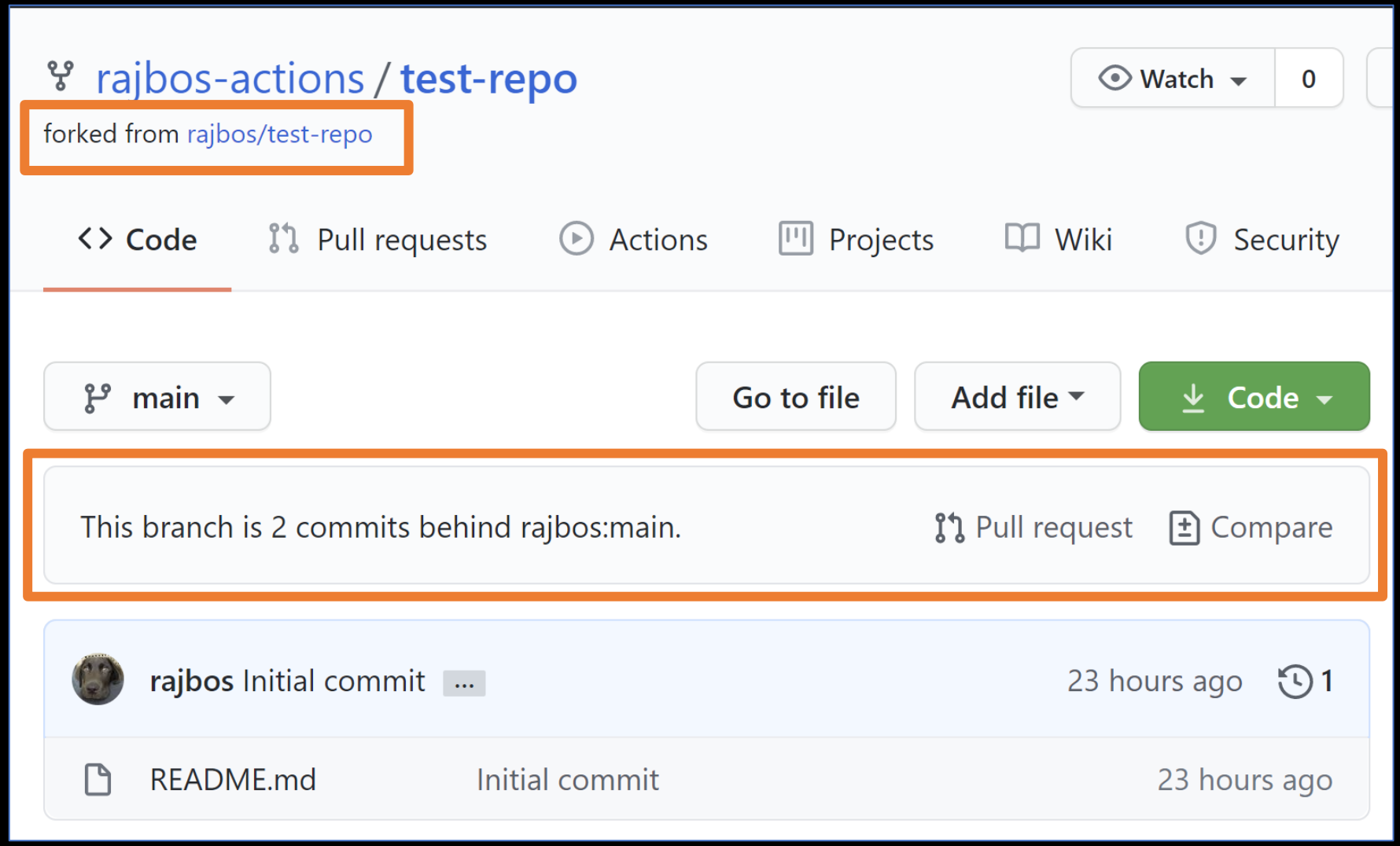
Add .github/dependabot.yml to the repo

```yaml
#Dependabot will check the dependencies in this repo for updates

version: 2
updates:
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      # Check for updates to GitHub Actions every weekday
      interval: "daily"


  - package-ecosystem: "nuget"
    directory: "/"
    schedule:
      # Check for updates to on nuget packages every weekday
      interval: "daily"
```

Use Dependabot



Keep you forked action up to date



Create an update process yourself

Add the parent repo as a remote

Fetch the changes from the parent

Merge in any changes from the branch

Push the changes back to your repo

Gist: http://xpir.it/UpdateRepo

Automate the update

Use a workflow!
https://github.com/aormsby/Fork-Sync-With-Upstream-action

Downsides:

- Needs to be added to each fork and on a new branch
- Workflows on non-default branch issue

- You're still pulling in all changes: how do you handle reviews?

Keep your forked action up to date

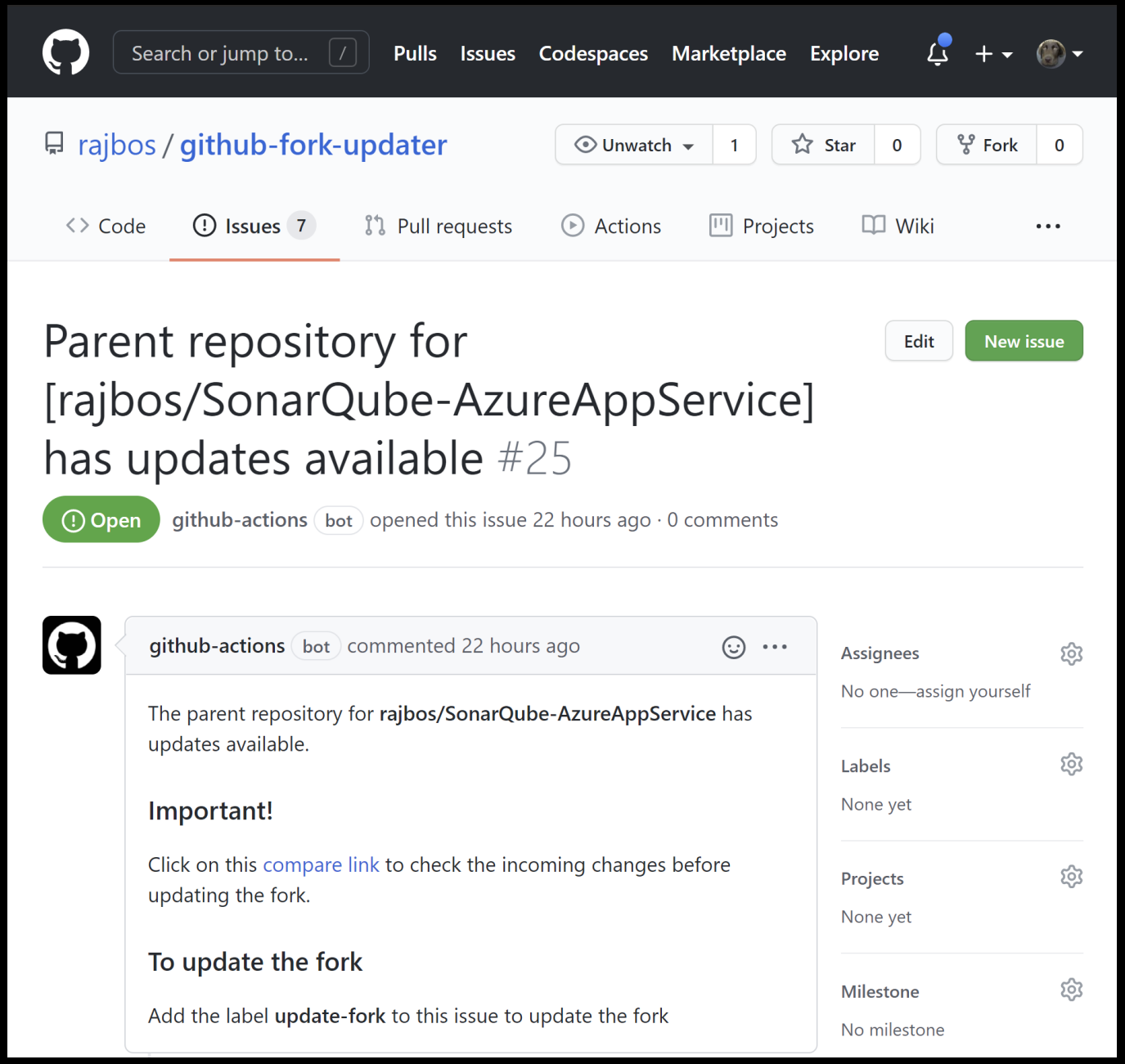
Fork a repo and automate it!

https://github.com/rajbos/github-fork-updater

Contains:

- Scheduled workflow
- Creates an issue
- Review the changes
- Label the issue
- Pull in changes

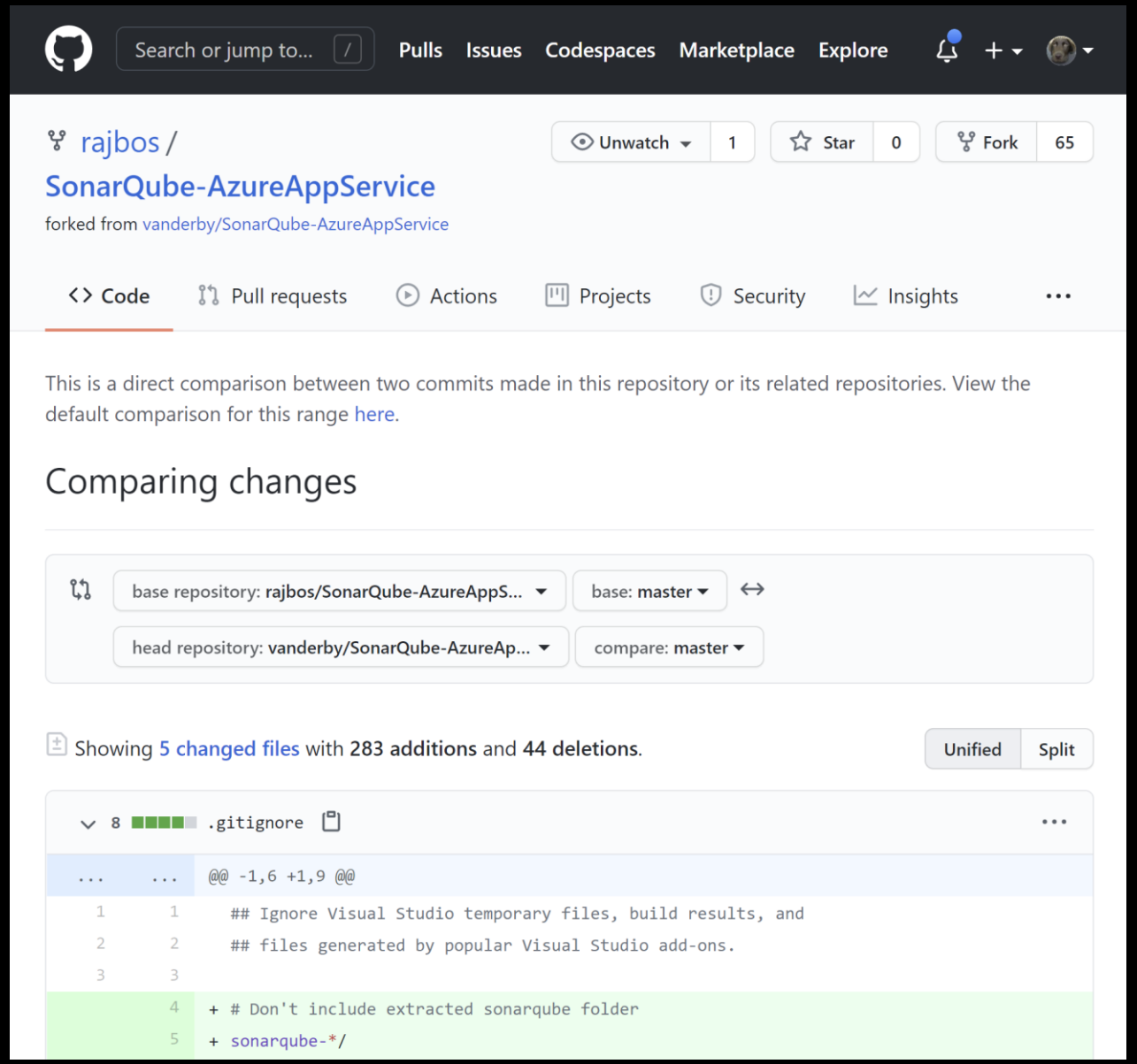
Creates issues



Review before merging

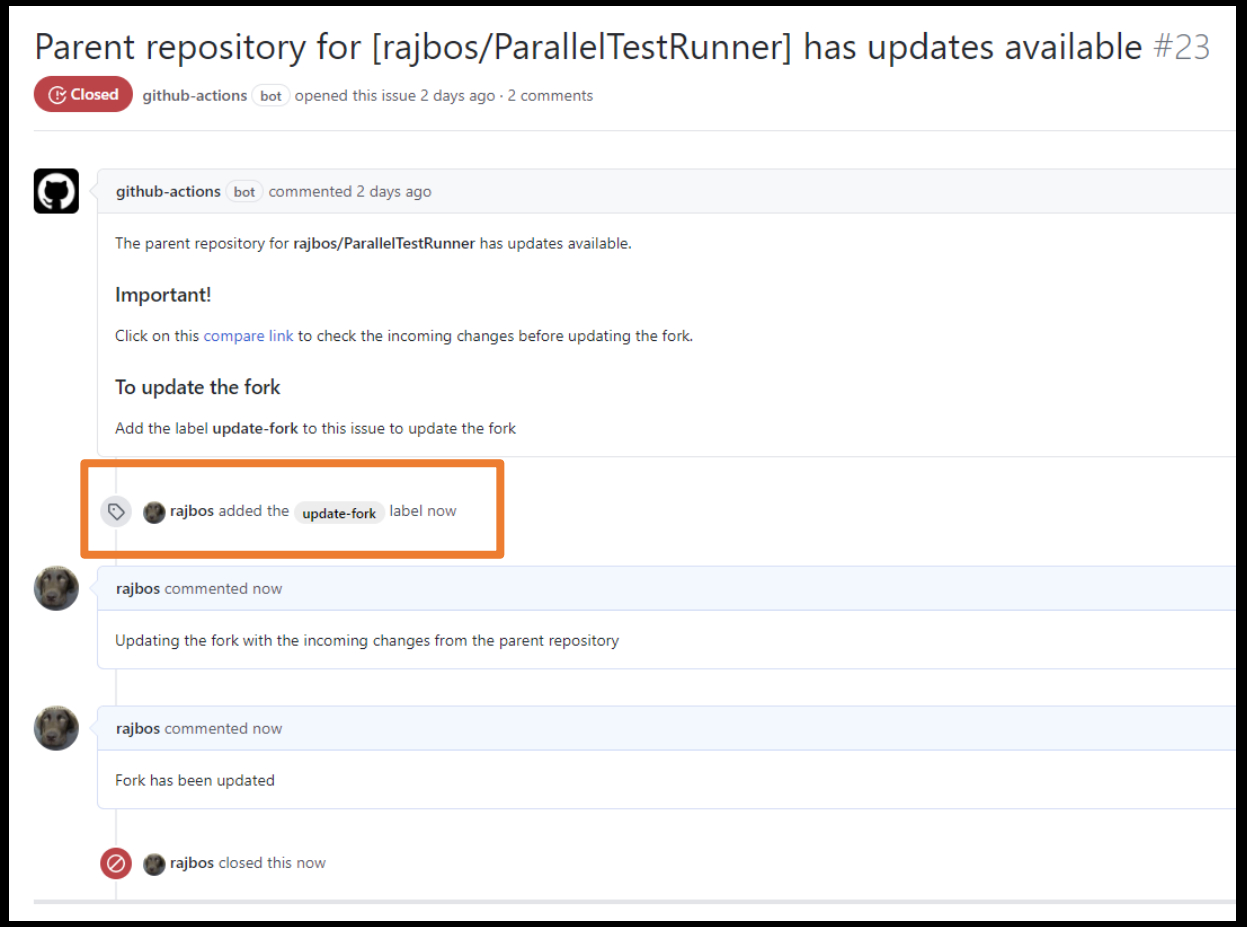
Compare url:

https://github.com/rajbos-actions/test-repo/compare/main...rajbos:main



Automation

- Add a label
- Fork gets updated
- Issue gets closed



GitHub Actions Security

Repository security
Runners and security
Actions and security

Forking actions
Keeping up to date

Best practices summarized

- Set access levels per team, not per user
- Treat workflow secrets very carefully: best to think of them as public
- Review actions' source code
- Pin actions to commit SHA
- Fork the action repo and limit actions to local actions only
- Have an organization setup to test with
- Keep your forked actions up to date

Thank you!

Rob Bos
DevOps Consultant – Xpirit
The Netherlands

NDC { London }