GitHub Actions

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

GitHub Actions

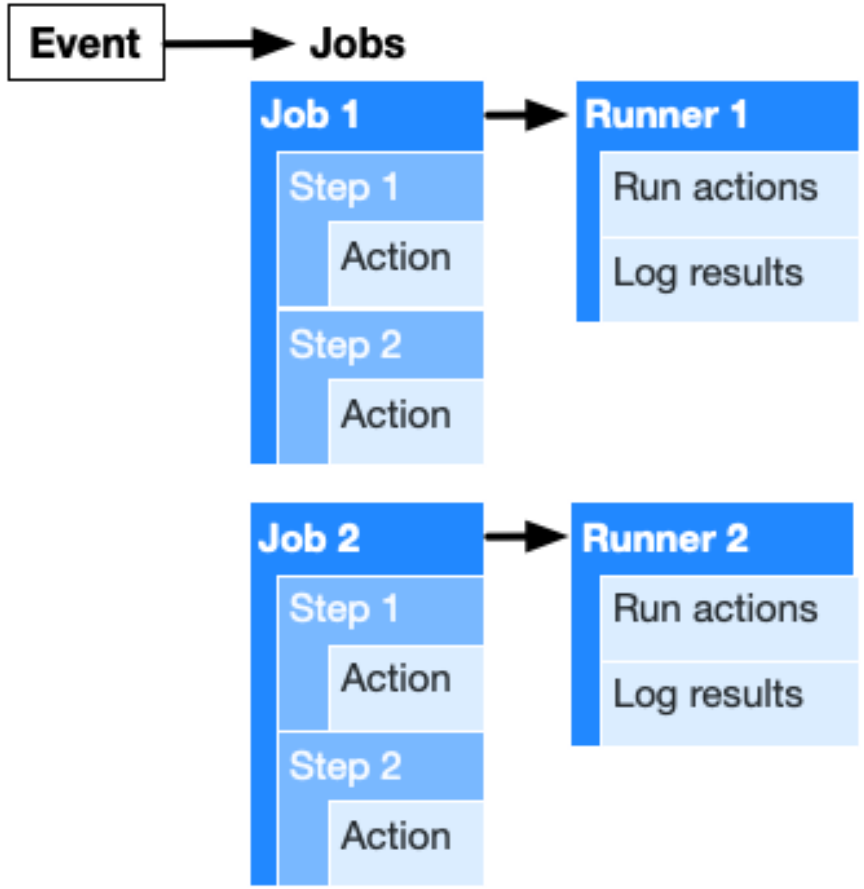
Automate your workflow from idea to production

GitHub Actions makes it easy to automate all your software workflows, now with world-class CI/CD. Build, test, and deploy your code right from GitHub. Make code reviews, branch management, and issue triaging work the way you want.

GitHub Actions

- GitHub Actions help you automate tasks within your software development life cycle.
- GitHub Actions are event-driven, meaning that you can run a series of commands after a specified event has occurred.
- For example, every time someone creates a pull request for a repository, you can automatically run a command that executes a software testing script.

The components of GitHub Actions



- Workflows
- Events
- Jobs
- Steps
- Actions
- Runners

Workflow

- The workflow is an automated procedure that you add to your repository.
- Workflows are made up of one or more jobs and can be scheduled or triggered by an event.
- The workflow can be used to build, test, package, release, or deploy a project on GitHub.

Events

- An event is a specific activity that triggers a workflow.
- Examples
  - Activity can originate from GitHub when someone pushes a commit to a repository or when an issue or pull request is created.
  - You can also use the repository dispatch webhook to trigger a workflow when an external event occurs.

Events That Trigger Workflows

- Scheduled events
  - schedule
- Manual events
  - workflow_dispatch
  - repository_dispatch
- Webhook events
  - check_run
  - check_suite
  - create
  - delete
  - deployment
  - deployment_status
  - fork
  - gollum
- Webhook events (contd.)
  - issue_comment
  - issues
  - label
  - milestone
  - page_build
  - project
  - project_card
  - project_column
  - public
  - pull_request
  - pull_request_review
  - pull_request_review_comment
  - pull_request_target
  - push
  - registry_package
  - release
  - status
  - watch
  - workflow_run

Jobs

- A job is a set of steps that execute on the same runner.
- By default, a workflow with multiple jobs will run those jobs in parallel.
- You can also configure a workflow to run jobs sequentially.
- Example
  - A workflow can have two sequential jobs that build and test code, where the test job is dependent on the status of the build job. If the build job fails, the test job will not run.

Steps

- A step is an individual task that can run commands in a job.
- A step can be either an action or a shell command.
- Each step in a job executes on the same runner, allowing the actions in that job to share data with each other.

Actions

- Actions are standalone commands that are combined into steps to create a job.
- Actions are the smallest portable building block of a workflow.
- You can create your own actions, or use actions created by the GitHub community.
- To use an action in a workflow, you must include it as a step.

Runners

- A runner is a server that has the GitHub Actions runner application installed.
- You can use a runner hosted by GitHub, or you can host your own.
- A runner listens for available jobs, runs one job at a time, and reports the progress, logs, and results back to GitHub.
- GitHub-hosted runners are based on Ubuntu Linux, Microsoft Windows, and macOS, and each job in a workflow runs in a fresh virtual environment.

Writing GitHub Actions

- GitHub Actions uses **`YAML`** syntax to define the events, jobs, and steps.
- GitHub Actions YAML files are stored in your code repository, in a directory called **`.github/workflows`**.

Sample Workflow

```yaml
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
      - run: npm install -g bats
      - run: bats -v
```

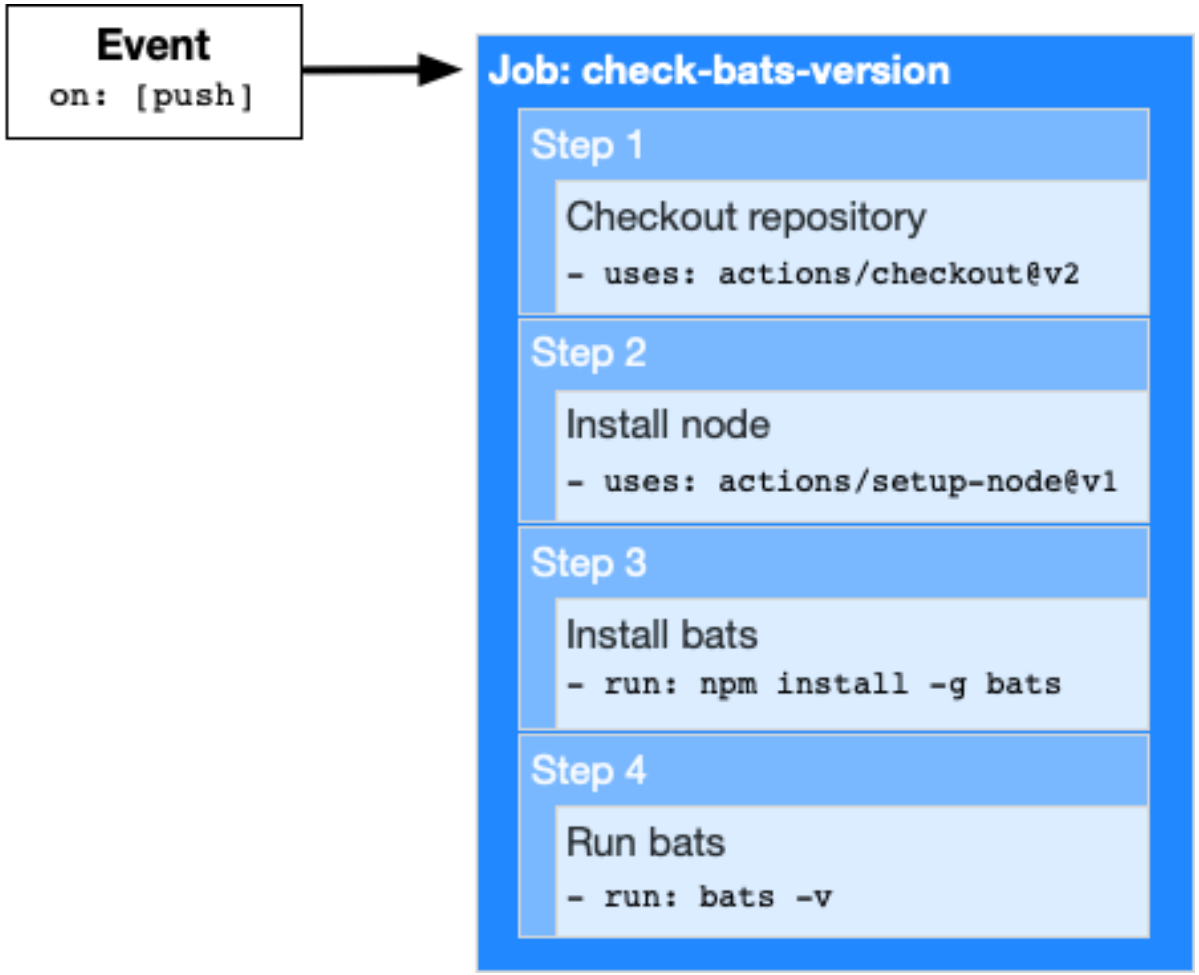
Understanding The Workflow File (1)

`name: learn-github-actions`	*Optional* - The name of the workflow as it will appear in the Actions tab of the GitHub repository.
`on: [push]`	Specify the event that automatically triggers the workflow file. This example uses the `push` event, so that the jobs run every time someone pushes a change to the repository. You can set up the workflow to only run on certain branches, paths, or tags. For syntax examples including or excluding branches, paths, or tags, see "Workflow syntax for GitHub Actions."
`jobs:`	Groups together all the jobs that run in the `learn-github-actions` workflow file.
`check-bats-version:`	Defines the name of the `check-bats-version` job stored within the `jobs` section.
`runs-on: ubuntu-latest`	Configures the job to run on an Ubuntu Linux runner. This means that the job will execute on a fresh virtual machine hosted by GitHub. For syntax examples using other runners, see "Workflow syntax for GitHub Actions."

Understanding The Workflow File (contd)

`steps:`	Groups together all the steps that run in the `check-bats-version` job. Each item nested under this section is a separate action or shell command.
`- uses: actions/checkout@v2`	The `uses` keyword tells the job to retrieve `v2` of the community action named `actions/checkout@v2`. This is an action that checks out your repository and downloads it to the runner, allowing you to run actions against your code (such as testing tools). You must use the checkout action any time your workflow will run against the repository's code or you are using an action defined in the repository.
`- uses: actions/setup-node@v1`	This action installs the `node` software package on the runner, giving you access to the `npm` command.
`- run: npm install -g bats`	The `run` keyword tells the job to execute a command on the runner. In this case, you are using `npm` to install the `bats` software testing package.
`- run: bats -v`	Finally, you'll run the `bats` command with a parameter that outputs the software version.

Visualizing The Workflow File

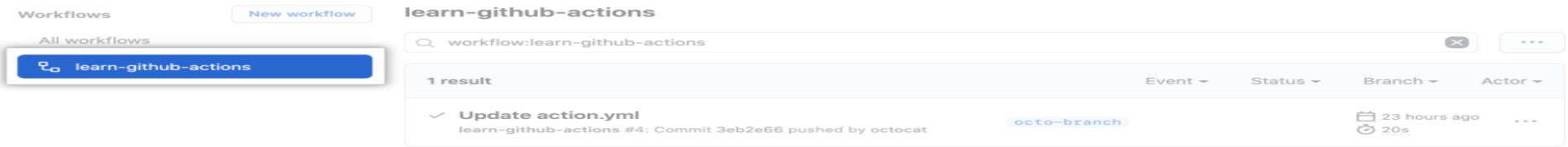


Viewing The Job’s Activity

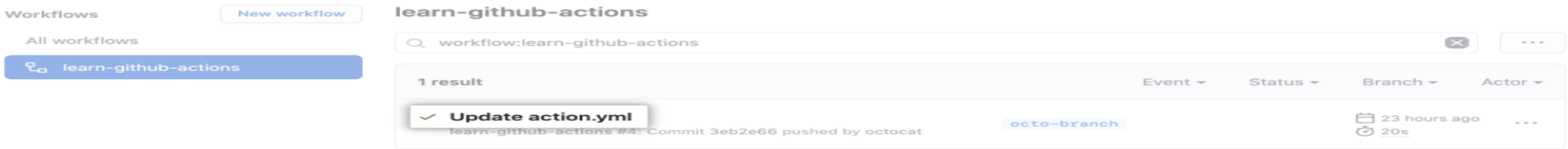
Once your job has started running, you can see a visualization graph of the run's progress and view each step's activity on GitHub.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.

3. In the left sidebar, click the workflow you want to see.

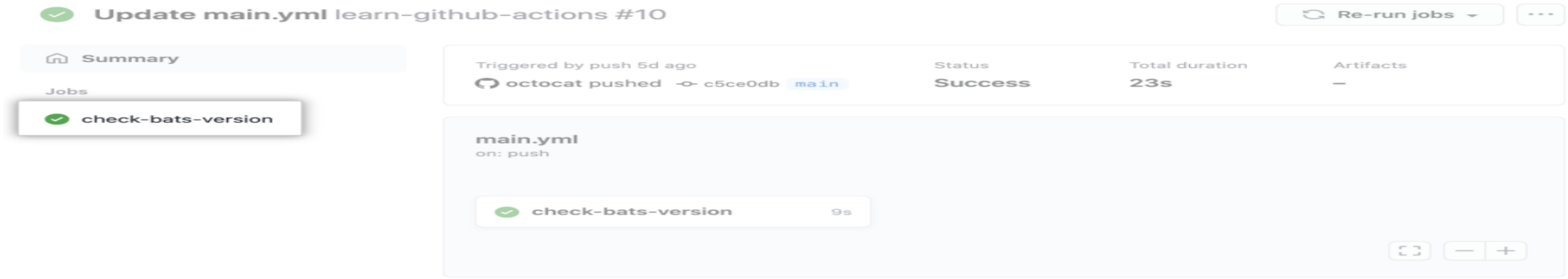


4. Under "Workflow runs", click the name of the run you want to see.

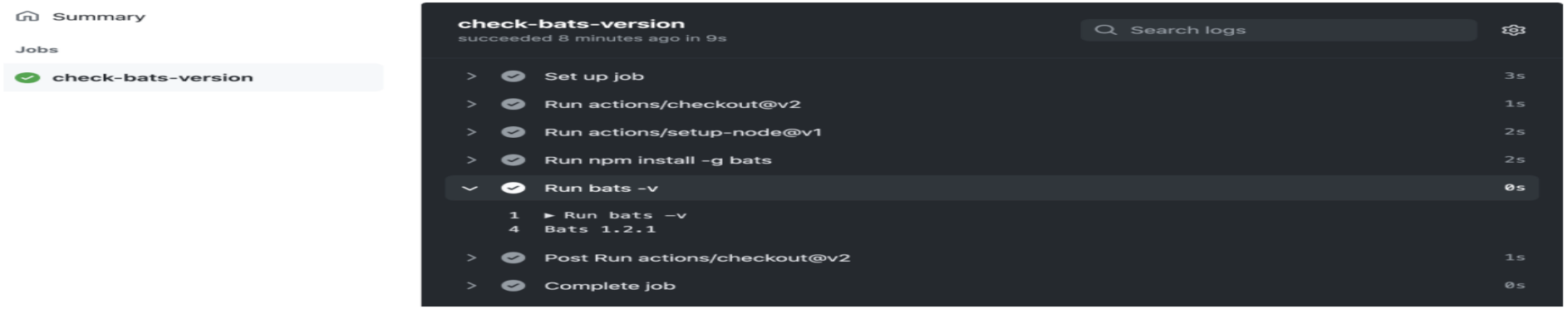


Viewing The Job’s Activity (contd.)

5 Under **Jobs** or in the visualization graph, click the job you want to see.



6 View the results of each step.



Managing Workflow Runs

- You can view the status and results of each step in your workflow, cancel a pending workflow, review deployments, view billable job execution minutes, debug and re-run a failed workflow, search and download logs, and download artifacts.
- https://docs.github.com/en/actions/managing-workflow-runs

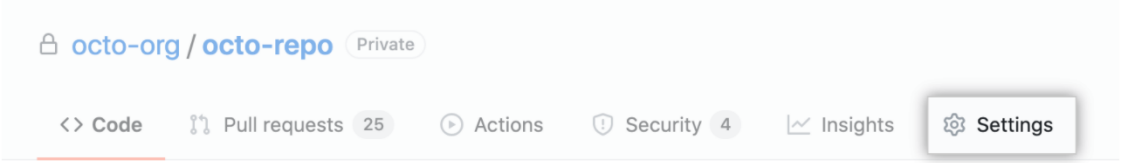
Encrypted Secrets

- Encrypted secrets allow you to store sensitive information in your organization, repository, or repository environments.
- Secrets are encrypted environment variables that you create in an organization, repository, or repository environment.
- The secrets that you create are available to use in GitHub Actions workflows.
- GitHub uses a libsodium sealed box to help ensure that secrets are encrypted before they reach GitHub and remain encrypted until you use them in a workflow.
- https://docs.github.com/en/actions/reference/encrypted-secrets

Creating Encrypted Secrets For A Repository

To create secrets for a user account repository, you must be the repository owner. To create secrets for an organization repository, you must have `admin` access.

1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click ⚙ **Settings**.



3. In the left sidebar, click **Secrets**.
4. Click **New repository secret**.
5. Type a name for your secret in the **Name** input box.
6. Enter the value for your secret.
7. Click **Add secret**.

Using Encrypted Secrets In A Workflow

- To provide an action with a secret as an input or environment variable, you can use the `secrets` context to access secrets you've created in your repository.
- Avoid passing secrets between processes from the command line, whenever possible.
- Command-line processes may be visible to other users (using the ps command) or captured by security audit events.
- To help protect secrets, consider using environment variables, STDIN, or other mechanisms supported by the target process.
- If you must pass secrets within a command line, then enclose them within the proper quoting rules.
- Secrets often contain special characters that may unintentionally affect your shell.
- To escape these special characters, use quoting with your environment variables.
- **Secrets are limited to 64 KB in size.**

```yaml
steps:
  - name: Hello world action
    with: # Set the secret as an input
      super_secret: ${{ secrets.SuperSecret }}
    env: # Or as an environment variable
      super_secret: ${{ secrets.SuperSecret }}
```

Example using Bash

```yaml
steps:
  - shell: bash
    env:
      SUPER_SECRET: ${{ secrets.SuperSecret }}
    run: |
      example-command "$SUPER_SECRET"
```

GitHub Actions Marketplace

- https://github.com/marketplace?type=actions

GitHub Action for AMI Pull Request

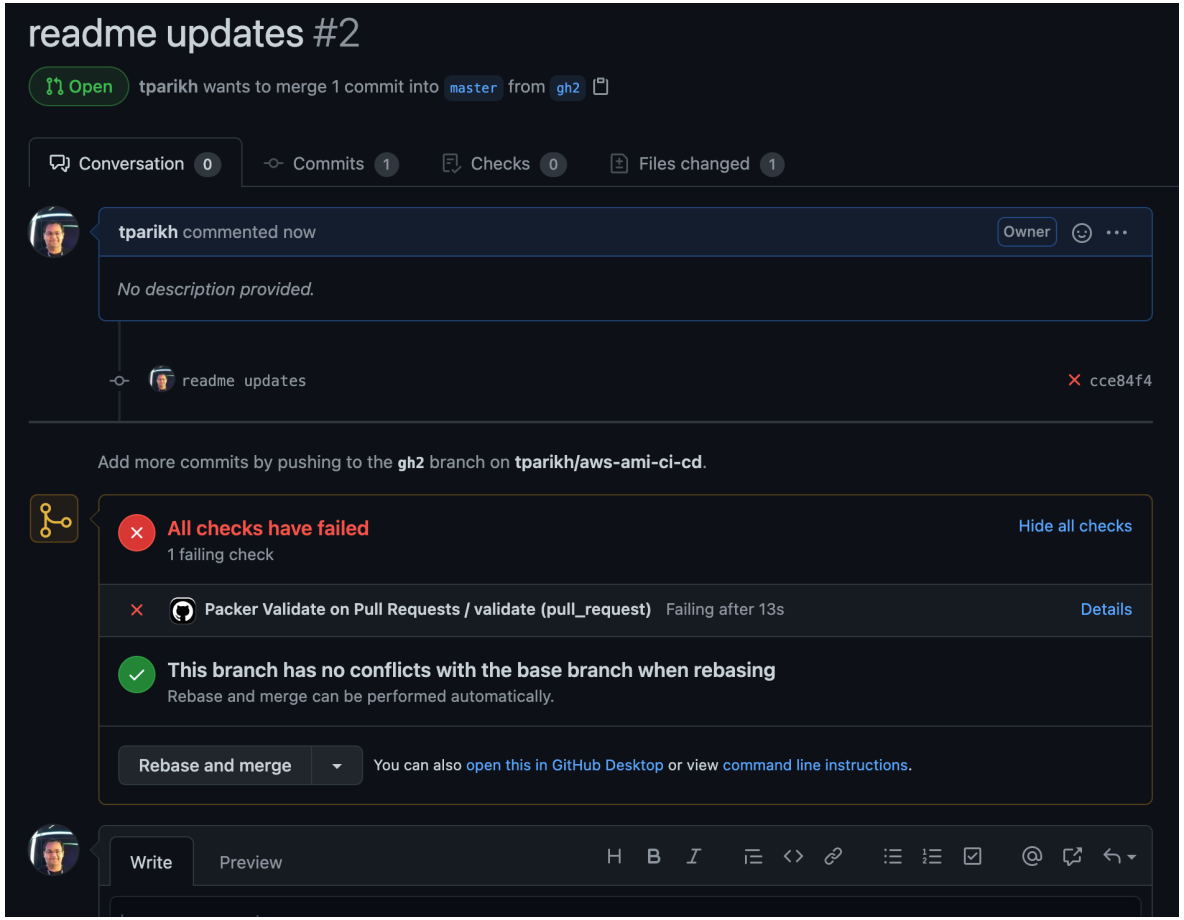
.github > workflows > pull-requests.yml > {} jobs > {} validate > [ ] steps

You, seconds ago | 1 author (You)

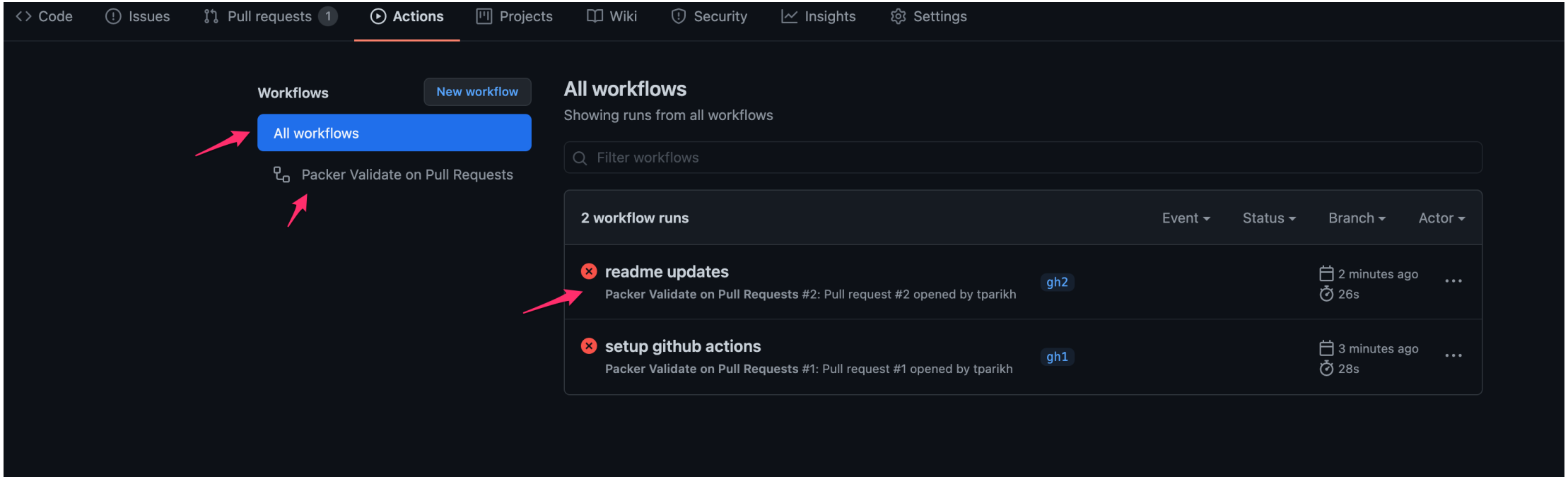
```yaml
---
name: Packer Validate on Pull Requests

# Controls when the action will run.
# Triggers the workflow on push or pull request events but only for the main branch
on:
  # Workflow is triggered on pull requests
  pull_request:
    # Workflow will only be triggered for pull requests on "master" branch
    branches: [ master ]

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  validate:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
    # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
    - name: Checkout Repository
      uses: actions/checkout@v2

    # Ref: https://github.com/marketplace/actions/packer-github-actions
    - name: Validate Packer Template
      uses: hashicorp/packer-github-actions@master
      with:
        command: validate
        arguments: -syntax-only # only validate syntax
        target: ami.json
      env:
        PACKER_LOG: 1 # enable debug log for packer
```

Repo: https://github.com/tparikh/aws-ami-ci-cd

GitHub Status Checks



Debugging Failed Workflows



Workflow Run Logs

Code | Issues | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

readme updates Packer Validate on Pull Requests #2

Re-run jobs

Summary

Jobs

validate

validate
failed 43 seconds ago in 13s

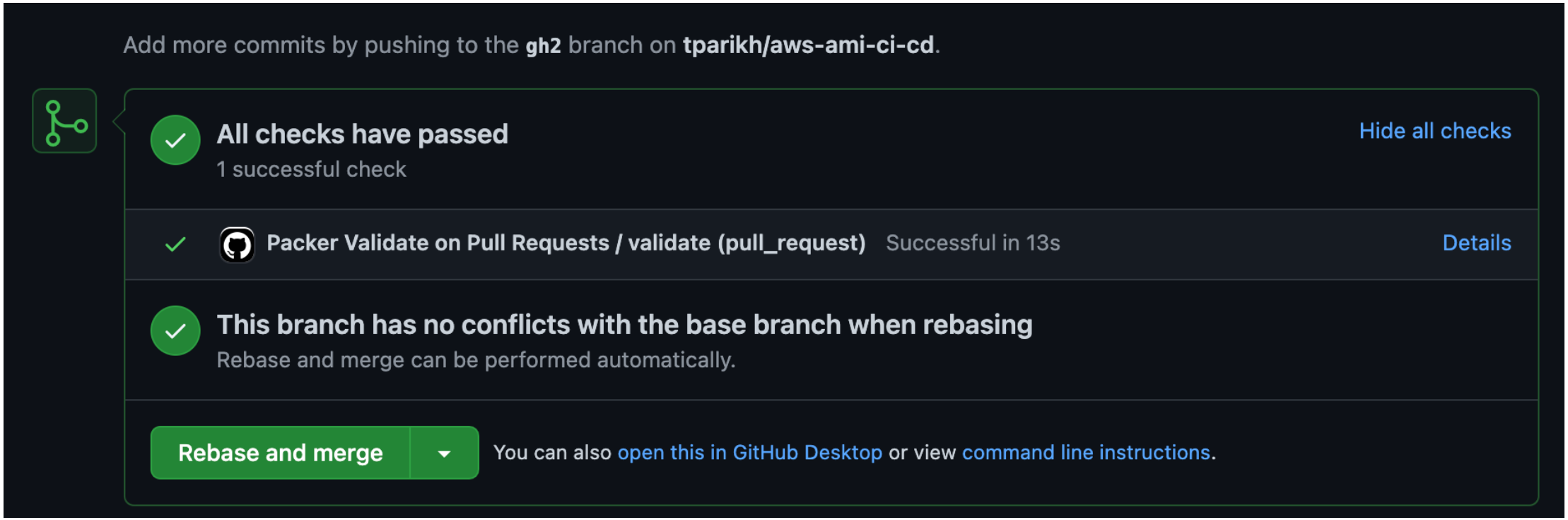
Search logs

- Set up job 3s
- Build hashicorp/packer-github-actions@master 8s
- Checkout Repository 0s
- Validate Packer Template 1s

```
1  ▶ Run hashicorp/packer-github-actions@master
5  /usr/bin/docker run --name e4d771a57bea1e4607ba17b7c5f14816ee_9dcfef --label 5588e4 --workdir /github/workspace --rm -e INPUT_COMMAND -e INPUT_TARGET -e INPUT_ARGUMENTS -e
   INPUT_WORKING_DIRECTORY -e HOME -e GITHUB_JOB -e GITHUB_REF -e GITHUB_SHA -e GITHUB_REPOSITORY -e GITHUB_REPOSITORY_OWNER -e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e
   GITHUB_RETENTION_DAYS -e GITHUB_ACTOR -e GITHUB_WORKFLOW -e GITHUB_HEAD_REF -e GITHUB_BASE_REF -e GITHUB_EVENT_NAME -e GITHUB_SERVER_URL -e GITHUB_API_URL -e GITHUB_GRAPHQL_URL
   -e GITHUB_WORKSPACE -e GITHUB_ACTION -e GITHUB_EVENT_PATH -e GITHUB_ACTION_REPOSITORY -e GITHUB_ACTION_REF -e GITHUB_PATH -e GITHUB_ENV -e RUNNER_OS -e RUNNER_TOOL_CACHE -e
   RUNNER_TEMP -e RUNNER_WORKSPACE -e ACTIONS_RUNTIME_URL -e ACTIONS_RUNTIME_TOKEN -e ACTIONS_CACHE_URL -e GITHUB_ACTIONS=true -e CI=true -v
   "/var/run/docker.sock":"/var/run/docker.sock" -v "/home/runner/work/_temp/_github_home":"/github/home" -v "/home/runner/work/_temp/_github_workflow":"/github/workflow" -v
   "/home/runner/work/_temp/_runner_file_commands":"/github/file_commands" -v "/home/runner/work/aws-ami-ci-cd/aws-ami-ci-cd":"/github/workspace"
   5588e4:d771a57bea1e4607ba17b7c5f14816ee
6  Error: Failed to prepare build: "amazon-ebs"
7
8  2 error(s) occurred:
9
10 * A source_ami or source_ami_filter must be specified
11 * For security reasons, your source AMI filter must declare an owner.
12
13
```

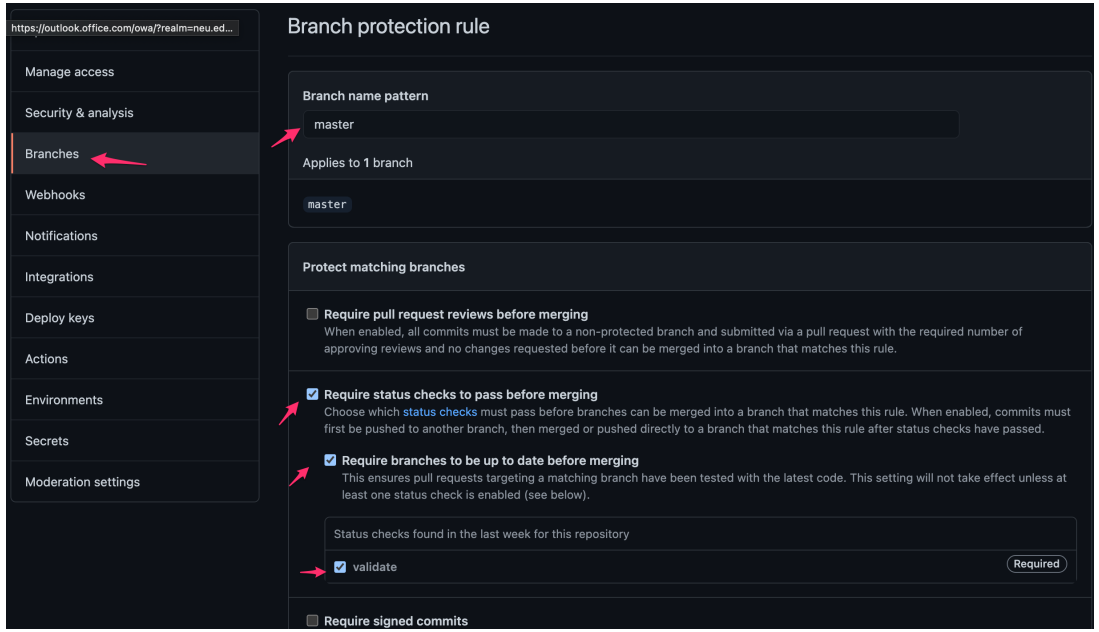
- Post Checkout Repository 0s
- Complete job 1s

Successful Status Checks



Enforcing Status Checks

Note that you will need GitHub Action workflow already committed to the repo to enable status checks.



Additional Resources

See Lecture Page