This version of GitHub Enterprise was discontinued on 2023-03-15. No patch releases will be made, even for critical security issues. For better performance, improved security, and new features, upgrade to the latest version of GitHub Enterprise. For help with the upgrade, contact GitHub Enterprise support.

Migrating from Travis CI to GitHub Actions

In this article

GitHub Actions and Travis CI share multiple similarities, which helps make it relatively straightforward to migrate to GitHub Actions.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide helps you migrate from Travis CI to GitHub Actions. It compares their concepts and syntax, describes the similarities, and demonstrates their different approaches to common tasks.

Before you start

Before starting your migration to GitHub Actions, it would be useful to become familiar with how it works:

- For a quick example that demonstrates a GitHub Actions job, see "Quickstart for GitHub Actions."
- To learn the essential GitHub Actions concepts, see "Understanding GitHub Actions."

Comparing job execution

To give you control over when CI tasks are executed, a GitHub Actions *workflow* uses *jobs* that run in parallel by default. Each job contains *steps* that are executed in a sequence that you define. If you need to run setup and cleanup actions for a job, you can define steps in each job to perform these.

Key similarities

GitHub Actions and Travis CI share certain similarities, and understanding these ahead of time can help smooth the migration process.

Using YAML syntax

Travis CI and GitHub Actions both use YAML to create jobs and workflows, and these files are stored in the code's repository. For more information on how GitHub Actions uses YAML, see "Understanding GitHub Actions."

Custom variables

Travis CI lets you set variables and share them between stages. Similarly, GitHub Actions lets you define variables for a workflows. For more information, see "Variables."

Default variables

Travis CI and GitHub Actions both include default environment variables that you can use in your YAML files. For GitHub Actions, you can see these listed in "Variables."

Parallel job processing

Travis CI can use `stages` to run jobs in parallel. Similarly, GitHub Actions runs `jobs` in parallel. For more information, see "About workflows."

Status badges

Travis CI and GitHub Actions both support status badges, which let you indicate whether a build is passing or failing. For more information, see "Adding a workflow status badge."

Using a matrix

Travis CI and GitHub Actions both support a matrix, allowing you to perform testing using combinations of operating systems and software packages. For more information, see "Using a matrix for your jobs."

Below is an example comparing the syntax for each system.

Travis CI syntax for a matrix

```
matrix:
  include:
    - rvm: 2.5
    - rvm: 2.6.3
```

GitHub Actions syntax for a matrix

```
jobs:
  build:
    strategy:
      matrix:
        ruby: [2.5, 2.6.3]
```

Targeting specific branches

Travis CI and GitHub Actions both allow you to target your CI to a specific branch. For more information, see "[Workflow syntax for GitHub Actions](#)."

Below is an example of the syntax for each system.

Travis CI syntax for targeting specific branches

```
branches:
  only:
    - main
    - 'mona/octocat'
```

GitHub Actions syntax for targeting specific branches

```
on:
  push:
    branches:
      - main
      - 'mona/octocat'
```

Checking out submodules

Travis CI and GitHub Actions both allow you to control whether submodules are included in the repository clone.

Below is an example of the syntax for each system.

Travis CI syntax for checking out submodules

```
git:
  submodules: false
```

GitHub Actions syntax for checking out submodules

```
- uses: {% data reusables.actions.action-checkout %}
  with:
    submodules: false
```

Using environment variables in a matrix

Travis CI and GitHub Actions can both add custom variables to a test matrix, which allows you to refer to the variable in a later step.

In GitHub Actions, you can use the `include` key to add custom environment variables to a matrix. In this example, the matrix entries for `node-version` are each configured to use different values for the `site` and `datacenter` environment variables. The `Echo site` `details` step then uses `env: ${{ matrix.env }}` to refer to the custom variables:

```yaml
name: Node.js CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
       include:
         - node-version: 10.x
           site: "prod"
           datacenter: "site-a"
         - node-version: 12.x
           site: "dev"
           datacenter: "site-b"
    steps:
      - name: Echo site details
        env:
          SITE: ${{ matrix.site }}
          DATACENTER: ${{ matrix.datacenter }}
        run: echo $SITE $DATACENTER
```

Key features in GitHub Actions

When migrating from Travis CI, consider the following key features in GitHub Actions:

Storing secrets

GitHub Actions allows you to store secrets and reference them in your jobs. GitHub Actions organizations can limit which repositories can access organization secrets. Environment protection rules can require manual approval for a workflow to access environment secrets. For more information, see "Encrypted secrets."

Sharing files between jobs and workflows

GitHub Actions includes integrated support for artifact storage, allowing you to share files between jobs in a workflow. You can also save the resulting files and share them with other workflows. For more information, see "Essential features of GitHub Actions."

Hosting your own runners

If your jobs require specific hardware or software, GitHub Actions allows you to host your own runners and send your jobs to them for processing. GitHub Actions also lets you use policies to control how these runners are accessed, granting access at the organization or repository level. For more information, see "Hosting your own runners."

Using different languages in GitHub Actions

When working with different languages in GitHub Actions, you can create a step in your job to set up your language dependencies. For more information about working with a particular language, see the specific guide:

- Building and testing Node.js
- Building and testing Python
- Building and testing PowerShell
- Building and testing Java with Maven
- Building and testing Java with Gradle

- Building and testing Java with Ant

Executing scripts

GitHub Actions can use `run` steps to run scripts or shell commands. To use a particular shell, you can specify the `shell` type when providing the path to the script. For more information, see "Workflow syntax for GitHub Actions."

For example:

```
steps:
  - name: Run build script
    run: ./.github/scripts/build.sh
    shell: bash
```

Error handling in GitHub Actions

When migrating to GitHub Actions, there are different approaches to error handling that you might need to be aware of.

Script error handling

GitHub Actions stops a job immediately if one of the steps returns an error code. For more information, see "Workflow syntax for GitHub Actions."

Job error handling

GitHub Actions uses `if` conditionals to execute jobs or steps in certain situations. For example, you can run a step when another step results in a `failure()`. For more information, see "Workflow syntax for GitHub Actions." You can also use `continue-on-error` to prevent a workflow run from stopping when a job fails.

Migrating syntax for conditionals and expressions

To run jobs under conditional expressions, Travis CI and GitHub Actions share a similar `if` condition syntax. GitHub Actions lets you use the `if` conditional to prevent a job or step from running unless a condition is met. For more information, see "Expressions."

This example demonstrates how an `if` conditional can control whether a step is executed:

```
jobs:
  conditional:
    runs-on: ubuntu-latest
    steps:
      - run: echo "This step runs with str equals 'ABC' and num equals 123"
        if: env.str == 'ABC' && env.num == 123
```

Migrating phases to steps

Where Travis CI uses *phases* to run *steps*, GitHub Actions has *steps* which execute *actions*. You can find prebuilt actions in the GitHub Marketplace, or you can create your own actions. For more information, see "Creating actions."

Below is an example of the syntax for each system.

Travis CI syntax for phases and steps

```
language: python
python:
  - "3.7"

script:
  - python script.py
```

GitHub Actions syntax for steps and actions

```
jobs:
  run_python:
    runs-on: ubuntu-latest
    steps:
      - uses: {% data reusables.actions.action-setup-python %}
        with:
          python-version: '3.7'
          architecture: 'x64'
      - run: python script.py
```

Caching dependencies

Travis CI and GitHub Actions let you manually cache dependencies for later reuse.

GitHub Actions caching is only available for repositories hosted on GitHub.com or GitHub Enterprise Server 3.5 and later. For more information, see "Caching dependencies to speed up workflows."

Examples of common tasks

This section compares how GitHub Actions and Travis CI perform common tasks.

Configuring environment variables

You can create custom environment variables in a GitHub Actions job.

Travis CI syntax for an environment variable

```
env:
  - MAVEN_PATH="/usr/local/maven"
```

GitHub Actions workflow with an environment variable

```
jobs:
  maven-build:
    env:
      MAVEN_PATH: '/usr/local/maven'
```

Building with Node.js

Travis CI for building with Node.js

```
install:
  - npm install
script:
  - npm run build
  - npm test
```

GitHub Actions workflow for building with Node.js

```
name: Node.js CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Use Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '12.x'
      - run: npm install
      - run: npm run build
      - run: npm test
```

Next steps

To continue learning about the main features of GitHub Actions, see "Learn GitHub Actions."