Extending GitHub Actions Importer with custom transformers

In this article

GitHub Actions Importer offers the ability to extend its built-in mapping.

Legal notice

About custom transformers

GitHub Actions Importer offers the ability to extend its built-in mapping by creating custom transformers. Custom transformers can be used to:

- Convert items that GitHub Actions Importer does not automatically convert, or modify how items are converted. For more information, see "Creating custom transformers for items."
- Convert references to runners to use different runner labels. For more information, see "Creating custom transformers for runners."
- Convert environment variable values from your existing pipelines to GitHub Actions workflows. For more information, see "Creating custom transformers for environment variables."

Using custom transformers with GitHub Actions Importer

A custom transformer contains mapping logic that GitHub Actions Importer can use to transform your plugins, tasks, runner labels, or environment variables to work with GitHub Actions. Custom transformers are written with a domain-specific language (DSL) built on top of Ruby, and are defined within a file with the `.rb` file extension.

You can use the `--custom-transformers` CLI option to specify which custom transformer files to use with the `audit`, `dry-run`, and `migrate` commands.

For example, if custom transformers are defined in a file named `transformers.rb`, you can use the following command to use them with GitHub Actions Importer:

```
gh actions-importer ... --custom-transformers transformers.rb
```

Alternatively, you can use the glob pattern syntax to specify multiple custom transformer

files. For example, if multiple custom transformer files are within a directory named `transformers`, you can provide them all to GitHub Actions Importer with the following command:

```
gh actions-importer ... --custom-transformers transformers/*.rb
```

Note: When you use custom transformers, the custom transformer files must reside in the same directory, or in subdirectores, from where the `gh actions-importer` command is run.

Creating custom transformers for items

You can create custom transformers that GitHub Actions Importer will use when converting existing build steps or triggers to their equivalent in GitHub Actions. This is especially useful when:

- GitHub Actions Importer doesn't automatically convert an item.
- You want to change how an item is converted by GitHub Actions Importer.
- Your existing pipelines use custom or proprietary extensions, such as shared libraries in Jenkins, and you need to define how these steps should function in GitHub Actions.

GitHub Actions Importer uses custom transformers that are defined using a DSL built on top of Ruby. In order to create custom transformers for build steps and triggers:

- Each custom transformer file must contain at least one `transform` method.
- Each `transform` method must return a `Hash`, an array of `Hash`'s, or `nil`. This returned value will correspond to an action defined in YAML. For more information about actions, see "Understanding GitHub Actions."

Example custom transformer for a build step

The following example converts a build step that uses the "buildJavascriptApp" identifier to run various `npm` commands:

Ruby

```ruby
transform "buildJavascriptApp" do |item|
  command = ["build", "package", "deploy"].map do |script|
    "npm run #{script}"
  end

  {
    name: "build javascript app",
    run: command.join("\n")
  }
end
```

The above example results in the following GitHub Actions workflow step. It is comprised of converted build steps that had a `buildJavascriptApp` identifier:

```yaml
- name: build javascript app
  run: |
    npm run build
    npm run package
    npm run deploy
```

The `transform` method uses the identifier of the build step from your source CI/CD instance in an argument. In this example, the identifier is `buildJavascriptLibrary`. You

can also use comma-separated values to pass multiple identifiers to the `transform` method. For example, `transform "buildJavascriptApp", "buildTypescriptApp" { |item| ... }`.

> **Note**: The data structure of `item` will be different depending on the CI/CD platform and the type of item being converted.

Creating custom transformers for runners

You can customize the mapping between runners in your source CI/CD instance and their equivalent GitHub Actions runners.

GitHub Actions Importer uses custom transformers that are defined using a DSL built on top of Ruby. To create custom transformers for runners:

- The custom transformer file must have at least one `runner` method.
- The `runner` method accepts two parameters. The first parameter is the source CI/CD instance's runner label, and the second parameter is the corresponding GitHub Actions runner label. For more information on GitHub Actions runners, see "Using GitHub-hosted runners."

Example custom transformers for runners

The following example shows a `runner` method that converts one runner label to one GitHub Actions runner label in the resulting workflow.

Ruby

```ruby
runner "linux", "ubuntu-latest"
```

You can also use the `runner` method to convert one runner label to multiple GitHub Actions runner labels in the resulting workflow.

Ruby

```ruby
runner "big-agent", ["self-hosted", "xl", "linux"]
```

GitHub Actions Importer attempts to map the runner label as best it can. In cases where it cannot do this, the `ubuntu-latest` runner label is used as a default. You can use a special keyword with the `runner` method to control this default value. For example, the following custom transformer instructs GitHub Actions Importer to use `macos-latest` as the default runner instead of `ubuntu-latest`.

Ruby

```ruby
runner :default, "macos-latest"
```

Creating custom transformers for environment variables

You can customize the mapping between environment variables in your source CI/CD pipelines to their values in GitHub Actions.

GitHub Actions Importer uses custom transformers that are defined using a DSL built on

top of Ruby. To create custom transformers for environment variables:

- The custom transformer file must have at least one `env` method.
- The `env` method accepts two parameters. The first parameter is the name of the environment variable in the original pipeline, and the second parameter is the updated value for the environment variable for GitHub Actions. For more information about GitHub Actions environment variables, see "Variables."

Example custom transformers for environment variables

There are several ways you can set up custom transformers to map your environment variables.

- The following example sets the value of any existing environment variables named `OCTO`, to `CAT` when transforming a pipeline.

  ```ruby
  env "OCTO", "CAT"
  ```

  You can also remove all instances of a specific environment variable so they are not transformed to an GitHub Actions workflow. The following example removes all environment variables with the name `MONA_LISA`.

  ```ruby
  env "MONA_LISA", nil
  ```

- You can also map your existing environment variables to secrets. For example, the following `env` method maps an environment variable named `MONALISA` to a secret named `OCTOCAT`.

  ```ruby
  env "MONALISA", secret("OCTOCAT")
  ```

  This will set up a reference to a secret named `OCTOCAT` in the transformed workflow. For the secret to work, you will need to create the secret in your GitHub repository. For more information, see "Using secrets in GitHub Actions."

- You can also use regular expressions to update the values of multiple environment variables at once. For example, the following custom transformer removes all environment variables from the converted workflow:

  ```ruby
  env /.*/, nil
  ```

  The following example uses a regular expression match group to transform environment variable values to dynamically generated secrets.

  ```ruby
  env /^(.+)_SSH_KEY/, secret("%s_SSH_KEY)
  ```

Note: The order in which `env` methods are defined matters when using regular expressions. The first `env` transformer that matches an environment variable name takes precedence over subsequent `env` methods. You should define your most specific environment variable transformers first.

Legal notice

Portions have been adapted from https://github.com/github/gh-actions-importer/ under the MIT license: