Migrating from Jenkins with GitHub Actions Importer

In this article

Learn how to use GitHub Actions Importer to automate the migration of your Jenkins pipelines to GitHub Actions.

Legal notice

About migrating from Jenkins with GitHub Actions Importer

The instructions below will guide you through configuring your environment to use GitHub Actions Importer to migrate Jenkins pipelines to GitHub Actions.

Prerequisites

- A Jenkins account or organization with pipelines and jobs that you want to convert to GitHub Actions workflows.
- Access to create a Jenkins personal API token for your account or organization.
- An environment where you can run Linux-based containers, and can install the necessary tools.
  - Docker is installed and running.
  - GitHub CLI is installed.

Note: The GitHub Actions Importer container and CLI do not need to be installed on the same server as your CI platform.

Limitations

There are some limitations when migrating from Jenkins to GitHub Actions with GitHub Actions Importer. For example, you must migrate the following constructs manually:

- Mandatory build tools
- Scripted pipelines
- Secrets
- Self-hosted runners
- Unknown plugins

For more information on manual migrations, see "Migrating from Jenkins to GitHub Actions."

Installing the GitHub Actions Importer CLI extension

1. Install the GitHub Actions Importer CLI extension:

   Bash

   ```
   gh extension install github/gh-actions-importer
   ```

2. Verify that the extension is installed:

   ```
   $ gh actions-importer -h
   Options:
     -?, -h, --help  Show help and usage information

   Commands:
     update     Update to the latest version of GitHub Actions Importer.
     version    Display the version of GitHub Actions Importer.
     configure  Start an interactive prompt to configure credentials used to authe
     audit      Plan your CI/CD migration by analyzing your current CI/CD footprir
     forecast   Forecast GitHub Actions usage from historical pipeline utilizatior
     dry-run    Convert a pipeline to a GitHub Actions workflow and output its yam
     migrate    Convert a pipeline to a GitHub Actions workflow and open a pull re
   ```

Configuring credentials

The `configure` CLI command is used to set required credentials and options for GitHub Actions Importer when working with Jenkins and GitHub.

1. Create a GitHub personal access token. For more information, see "Managing your personal access tokens."

   Your token must have the `workflow` scope.

   After creating the token, copy it and save it in a safe location for later use.

2. Create a Jenkins API token. For more information, see Authenticating scripted clients in the Jenkins documentation.

   After creating the token, copy it and save it in a safe location for later use.

3. In your terminal, run the GitHub Actions Importer `configure` CLI command:

```
gh actions-importer configure
```

The `configure` command will prompt you for the following information:

- For "Which CI providers are you configuring?", use the arrow keys to select `Jenkins`, press `Space` to select it, then press `Enter`.
- For "Personal access token for GitHub", enter the value of the personal access token that you created earlier, and press `Enter`.
- For "Base url of the GitHub instance", enter the URL for your GitHub Enterprise Server instance, and press `Enter`.
- For "Personal access token for Jenkins", enter the value for the Jenkins personal API token that you created earlier, and press `Enter`.
- For "Username of Jenkins user", enter your Jenkins username and press `Enter`.
- For "Base url of the Jenkins instance", enter the URL of your Jenkins instance, and press `Enter`.

  An example of the `configure` command is shown below:

  ```
  $ gh actions-importer configure
  ✔ Which CI providers are you configuring?: Jenkins
  Enter the following values (leave empty to omit):
  ✔ Personal access token for GitHub: ***************
  ✔ Base url of the GitHub instance: https://github.com
  ✔ Personal access token for Jenkins: ***************
  ✔ Username of Jenkins user: admin
  ✔ Base url of the Jenkins instance: https://localhost
  Environment variables successfully updated.
  ```

4. In your terminal, run the GitHub Actions Importer `update` CLI command to connect to GitHub Packages Container registry and ensure that the container image is updated to the latest version:

   ```
   gh actions-importer update
   ```

   The output of the command should be similar to below:

   ```
   Updating ghcr.io/actions-importer/cli:latest...
   ghcr.io/actions-importer/cli:latest up-to-date
   ```

Perform an audit of Jenkins

You can use the `audit` command to get a high-level view of all pipelines in a Jenkins server.

The `audit` command performs the following steps:

1. Fetches all of the projects defined in a Jenkins server.
2. Converts each pipeline to its equivalent GitHub Actions workflow.
3. Generates a report that summarizes how complete and complex of a migration is possible with GitHub Actions Importer.

Running the audit command

To perform an audit of a Jenkins server, run the following command in your terminal:

```
gh actions-importer audit jenkins --output-dir tmp/audit
```

Inspecting the audit results

The files in the specified output directory contain the results of the audit. See the `audit_summary.md` file for a summary of the audit results.

The audit summary has the following sections.

Pipelines

The "Pipelines" section contains a high-level statistics regarding the conversion rate done by GitHub Actions Importer.

Listed below are some key terms that can appear in the "Pipelines" section:

- **Successful** pipelines had 100% of the pipeline constructs and individual items converted automatically to their GitHub Actions equivalent.
- **Partially successful** pipelines had all of the pipeline constructs converted, however, there were some individual items that were not converted automatically to their GitHub Actions equivalent.
- **Unsupported** pipelines are definition types that are not supported by GitHub Actions Importer.
- **Failed** pipelines encountered a fatal error when being converted. This can occur for one of three reasons:
  - The pipeline was misconfigured and not valid in Bamboo.
  - GitHub Actions Importer encountered an internal error when converting it.
  - There was an unsuccessful network response that caused the pipeline to be inaccessible, which is often due to invalid credentials.

Build steps

The "Build steps" section contains an overview of individual build steps that are used across all pipelines, and how many were automatically converted by GitHub Actions Importer.

Listed below are some key terms that can appear in the "Build steps" section:

- A **known** build step is a step that was automatically converted to an equivalent action.
- An **unknown** build step is a step that was not automatically converted to an equivalent action.
- An **unsupported** build step is a step that is either:
  - Fundamentally not supported by GitHub Actions.
  - Configured in a way that is incompatible with GitHub Actions.
- An **action** is a list of the actions that were used in the converted workflows. This can be important for:
  - If you use GitHub Enterprise Server, gathering the list of actions to sync to your instance.
  - Defining an organization-level allowlist of actions that are used. This list of actions is a comprehensive list of actions that your security or compliance teams may need to review.

Manual tasks

The "Manual tasks" section contains an overview of tasks that GitHub Actions Importer is not able to complete automatically, and that you must complete manually.

Listed below are some key terms that can appear in the "Manual tasks" section:

- A **secret** is a repository or organization-level secret that is used in the converted pipelines. These secrets must be created manually in GitHub Actions for these pipelines to function properly. For more information, see "Encrypted secrets."
- A **self-hosted runner** refers to a label of a runner that is referenced in a converted pipeline that is not a GitHub-hosted runner. You will need to manually define these runners for these pipelines to function properly.

Files

The final section of the audit report provides a manifest of all the files that were written to disk during the audit.

Each pipeline file has a variety of files included in the audit, including:

- The original pipeline as it was defined in GitHub.
- Any network responses used to convert the pipeline.
- The converted workflow file.
- Stack traces that can be used to troubleshoot a failed pipeline conversion.

Additionally, the `workflow_usage.csv` file contains a comma-separated list of all actions, secrets, and runners that are used by each successfully converted pipeline. This can be useful for determining which workflows use which actions, secrets, or runners, and can be useful for performing security reviews.

Forecast potential build runner usage

You can use the `forecast` command to forecast potential GitHub Actions usage by computing metrics from completed pipeline runs in your Jenkins server.

Prerequisites for running the forecast command

In order to run the `forecast` command against a Jenkins instance, you must install the `paginated-builds` plugin on your Jenkins server. This plugin allows GitHub Actions Importer to efficiently retrieve historical build data for jobs that have a large number of builds. Because Jenkins does not provide a method to retrieve paginated build data, using this plugin prevents timeouts from the Jenkins server that can occur when fetching a large amount of historical data. The `paginated-builds` plugin is open source, and exposes a REST API endpoint to fetch build data in pages, rather than all at once.

To install the `paginated-builds` plugin:

1. On your Jenkins instance, navigate to `https://<your-jenkins-instance>/pluginManager/available`.
2. Search for the `paginated-builds` plugin.
3. Check the box on the left and select **Install without restart**.

Running the forecast command

To perform a forecast of potential GitHub Actions, run the following command in your

terminal. By default, GitHub Actions Importer includes the previous seven days in the forecast report.

```
gh actions-importer forecast jenkins --output-dir tmp/forecast
```

Inspecting the forecast report

The `forecast_report.md` file in the specified output directory contains the results of the forecast.

Listed below are some key terms that can appear in the forecast report:

- The **job count** is the total number of completed jobs.
- The **pipeline count** is the number of unique pipelines used.
- **Execution time** describes the amount of time a runner spent on a job. This metric can be used to help plan for the cost of GitHub-hosted runners.
    - This metric is correlated to how much you should expect to spend in GitHub Actions. This will vary depending on the hardware used for these minutes. You can use the GitHub Actions pricing calculator to estimate the costs.
- **Queue time** metrics describe the amount of time a job spent waiting for a runner to be available to execute it.
- **Concurrent jobs** metrics describe the amount of jobs running at any given time. This metric can be used to define the number of runners you should configure.

Additionally, these metrics are defined for each queue of runners in Jenkins. This is especially useful if there is a mix of hosted or self-hosted runners, or high or low spec machines, so you can see metrics specific to different types of runners.

Perform a dry-run migration of a Jenkins pipeline

You can use the `dry-run` command to convert a Jenkins pipeline to its equivalent GitHub Actions workflow.

Running the dry-run command

You can use the `dry-run` command to convert a Jenkins pipeline to an equivalent GitHub Actions workflow. A dry-run creates the output files in a specified directory, but does not open a pull request to migrate the pipeline.

To perform a dry run of migrating your Jenkins pipelines to GitHub Actions, run the following command in your terminal, replacing `my-jenkins-project` with the URL of your Jenkins job.

```
gh actions-importer dry-run jenkins --source-url my-jenkins-project --output-dir tmp
```

Inspecting the converted workflows

You can view the logs of the dry run and the converted workflow files in the specified output directory.

If there is anything that GitHub Actions Importer was not able to convert automatically, such as unknown build steps or a partially successful pipeline, you might want to create custom transformers to further customize the conversion process. For more information, see "Extending GitHub Actions Importer with custom transformers."

Perform a production migration of a Jenkins pipeline

You can use the `migrate` command to convert a Jenkins pipeline and open a pull request with the equivalent GitHub Actions workflow.

Running the migrate command

To migrate a Jenkins pipeline to GitHub Actions, run the following command in your terminal, replacing the `target-url` value with the URL for your GitHub Enterprise Server repository, and `my-jenkins-project` with the URL for your Jenkins job.

```
gh actions-importer migrate jenkins --target-url https://github.com/:owner/:repo --
```

The command's output includes the URL to the pull request that adds the converted workflow to your repository. An example of a successful output is similar to the following:

```
$ gh actions-importer migrate jenkins --target-url https://github.com/octo-org/octo
[2022-08-20 22:08:20] Logs: 'tmp/migrate/log/actions-importer-20220916-014033.log'
[2022-08-20 22:08:20] Pull request: 'https://github.com/octo-org/octo-repo/pull/1'
```

Inspecting the pull request

The output from a successful run of the `migrate` command contains a link to the new pull request that adds the converted workflow to your repository.

Some important elements of the pull request include:

- In the pull request description, a section called **Manual steps**, which lists steps that you must manually complete before you can finish migrating your pipelines to GitHub Actions. For example, this section might tell you to create any secrets used in your workflows.
- The converted workflows file. Select the **Files changed** tab in the pull request to view the workflow file that will be added to your GitHub Enterprise Server repository.

When you are finished inspecting the pull request, you can merge it to add the workflow to your GitHub Enterprise Server repository.

Reference

This section contains reference information on environment variables, optional arguments, and supported syntax when using GitHub Actions Importer to migrate from Jenkins.

Using environment variables

GitHub Actions Importer uses environment variables for its authentication configuration. These variables are set when following the configuration process using the `configure` command. For more information, see the "Configure credentials for GitHub Actions Importer" section.

GitHub Actions Importer uses the following environment variables to connect to your Jenkins instance:

- `GITHUB_ACCESS_TOKEN`: The personal access token used to create pull requests with a

converted workflow (requires `repo` and `workflow` scopes).

- `GITHUB_INSTANCE_URL`: The URL to the target GitHub instance (for example, `https://github.com`).
- `JENKINS_ACCESS_TOKEN`: The Jenkins API token used to view Jenkins resources.

  > **Note**: This token requires access to all jobs that you want to migrate or audit. In cases where a folder or job does not inherit access control lists from their parent, you must grant explicit permissions or full admin privileges.

- `JENKINS_USERNAME`: The username of the user account that created the Jenkins API token.
- `JENKINS_INSTANCE_URL`: The URL of the Jenkins instance.
- `JENKINSFILE_ACCESS_TOKEN` (Optional) The API token used to retrieve the contents of a `Jenkinsfile` stored in the build repository. This requires the `repo` scope. If this is not provided, the `GITHUB_ACCESS_TOKEN` will be used instead.

These environment variables can be specified in a `.env.local` file that is loaded by GitHub Actions Importer when it is run.

Using optional arguments

There are optional arguments you can use with the GitHub Actions Importer subcommands to customize your migration.

`--source-file-path`

You can use the `--source-file-path` argument with the `forecast`, `dry-run`, or `migration` subcommands.

By default, GitHub Actions Importer fetches pipeline contents from source control. The `--source-file-path` argument tells GitHub Actions Importer to use the specified source file path instead. You can use this option for Jenkinsfile and multibranch pipelines.

If you would like to supply multiple source files when running the `forecast` subcommand, you can use pattern matching in the file path value. For example, `gh forecast --source-file-path ./tmp/previous_forecast/jobs/*.json` supplies GitHub Actions Importer with any source files that match the `./tmp/previous_forecast/jobs/*.json` file path.

Jenkinsfile pipeline example

In this example, GitHub Actions Importer uses the specified Jenkinsfile as the source file to perform a dry run.

```
gh actions-importer dry-run jenkins --output-dir path/to/output/ --source-file-path
```

`--config-file-path`

You can use the `--config-file-path` argument with the `audit`, `dry-run`, and `migrate` subcommands.

By default, GitHub Actions Importer fetches pipeline contents from source control. The `--config-file-path` argument tells GitHub Actions Importer to use the specified source files instead.

When you use the `--config-file-path` option with the `dry-run` or `migrate` subcommands, GitHub Actions Importer matches the repository slug to the job represented by the `--source-url` option to select the pipeline. It uses the `config-file-path` to pull the specified source file.

Audit example

In this example, GitHub Actions Importer uses the specified YAML configuration file to perform an audit.

```
gh actions-importer audit jenkins --output-dir path/to/output/ --config-file-path pa
```

To audit a Jenkins instance using a config file, the config file must be in the following format, and each `repository_slug` value must be unique:

```yaml
source_files:
  - repository_slug: pipeline-name
    path: path/to/Jenkinsfile
  - repository_slug: multi-branch-pipeline-name
    branches:
      - branch: main
        path: path/to/Jenkinsfile
      - branch: node
        path: path/to/Jenkinsfile
```

Supported syntax for Jenkins pipelines

The following tables show the type of properties GitHub Actions Importer is currently able to convert. For more details about how Jenkins pipeline syntax aligns with GitHub Actions, see "Migrating from Jenkins to GitHub Actions".

For information about supported Jenkins plugins, see the `github/gh-actions-importer` repository.

Supported syntax for Freestyle pipelines

Jenkins	GitHub Actions	Status
docker template	`jobs.<job_id>.container`	Supported
build	`jobs`	Partially supported
build environment	`env`	Partially supported
build triggers	`on`	Partially supported
general	`runners`	Partially supported

Supported syntax for Jenkinsfile pipelines

Jenkins	GitHub Actions	Status
docker	`jobs.<job_id>.container`	Supported
stage	`jobs.<job_id>`	Supported
agent	`runners`	Partially supported

environment	`env`	Partially supported
stages	`jobs`	Partially supported
steps	`jobs.<job_id>.steps`	Partially supported
triggers	`on`	Partially supported
when	`jobs.<job_id>.if`	Partially supported
inputs	`inputs`	Unsupported
matrix	`jobs.<job_id>.strategy.matrix`	Unsupported
options	`jobs.<job_id>.strategy`	Unsupported
parameters	`inputs`	Unsupported

Environment variables syntax

GitHub Actions Importer uses the mapping in the table below to convert default Jenkins environment variables to the closest equivalent in GitHub Actions.

Jenkins	GitHub Actions
`${BUILD_ID}`	`${{ github.run_id }}`
`${BUILD_NUMBER}`	`${{ github.run_id }}`
`${BUILD_TAG}`	`${{ github.workflow }}-${{ github.run_id }}`
`${BUILD_URL}`	`${{ github.server_url }}/${{ github.repository }}/actions/runs/${{ github.run_id }}`
`${JENKINS_URL}`	`${{ github.server_url }}`
`${JOB_NAME}`	`${{ github.workflow }}`
`${WORKSPACE}`	`${{ github.workspace }}`

Legal notice

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

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```