Migrating from Bitbucket Pipelines with GitHub Actions Importer

In this article

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

Legal notice

About migrating from Bitbucket Pipelines with GitHub Actions Importer

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

Prerequisites

- 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 Bitbucket Pipelines to GitHub Actions with GitHub Actions Importer.

- Images in a private AWS ECR are not supported.
- The Bitbucket Pipelines option `size` is not supported. If additional runner resources are required in GitHub Actions, consider using larger runners. For more information, see "About larger runners."

- Metrics detailing the queue time of jobs is not supported by the `forecast` command.
- Bitbucket after-scripts are supported using GitHub Actions `always()` in combination with checking the `steps.<step_id>.conclusion` of the previous step. For more information, see "Contexts."

  The following is an example of using the `always()` with `steps.<step_id>.conclusion`.

```
- name: After Script 1
  run: |-
    echo "I'm after the script ran!"
    echo "We should be grouped!"
  id: after-script-1
  if: "${{ always() }}"
- name: After Script 2
  run: |-
    echo "this is really the end"
    echo "goodbye, for now!"
  id: after-script-2
  if: "${{ steps.after-script-1.conclusion == 'success' && always() }}"
```

Manual tasks

Certain Bitbucket Pipelines constructs must be migrated manually. These include:

- Secured repository, workspace, and deployment variables
- SSH keys

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
authenticate with your CI server(s).
  audit      Plan your CI/CD migration by analyzing your current CI/CD
footprint.
  forecast   Forecast GitHub Actions usage from historical pipeline
utilization.
  dry-run    Convert a pipeline to a GitHub Actions workflow and output its
yaml file.
  migrate    Convert a pipeline to a GitHub Actions workflow and open a pull
request with the changes.
```

Configuring credentials

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

1. Create a GitHub personal access token (classic). 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 Workspace Access Token for Bitbucket Pipelines. For more information, see Workspace Access Token permissions in the Bitbucket documentation. Your token must have the `read` scope for pipelines, projects, and repositories.

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 `Bitbucket`, press `Space` to select it, then press `Enter`.
   - For "Personal access token for GitHub", enter the value of the personal access token (classic) that you created earlier, and press `Enter`.
   - For "Base url of the GitHub instance", press `Enter` to accept the default value (`https://github.com`).
   - For "Personal access token for Bitbucket", enter the Workspace Access Token that you created earlier, and press `Enter`.
   - For "Base url of the Bitbucket instance", enter the URL for your Bitbucket instance, and press `Enter`.

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

   ```
   $ gh actions-importer configure
   ✔ Which CI providers are you configuring?: Bitbucket
   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 Bitbucket: ********************
   ✔ Base url of the Bitbucket instance: https://bitbucket.example.com
   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 the Bitbucket instance

You can use the audit command to get a high-level view of pipelines in a Bitbucket instance.

The audit command performs the following steps.

1. Fetches all of the pipelines for a workspace.
2. Converts 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 run the following command in your terminal, replacing `:workspace` with the name of the Bitbucket workspace to audit.

```
gh actions-importer audit bitbucket --workspace :workspace--output-dir tmp/audit
```

Optionally, a `--project-key` option can be provided to the audit command to limit the results to only pipelines associated with a project.

In the below example command `:project_key` should be replaced with the key of the project that should be audited. Project keys can be found in Bitbucket on the workspace projects page.

```
gh actions-importer audit bitbucket --workspace :workspace --project-key
:project_key --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 "Using secrets in GitHub Actions."
- 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.

Forecasting usage

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

Running the forecast command

To perform a forecast of potential GitHub Actions usage, run the following command in your terminal, replacing `:workspace` with the name of the Bitbucket workspace to forecast. By default, GitHub Actions Importer includes the previous seven days in the forecast report.

```
gh actions-importer forecast bitbucket --workspace :workspace --output-dir
tmp/forecast_reports
```

Forecasting a project

To limit the forecast to a project, you can use the `--project-key` option. Replace the value for the `:project_key` with the project key for the project to forecast.

```
gh actions-importer forecast bitbucket --workspace :workspace --project-key
:project_key --output-dir tmp/forecast_reports
```

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.
- **Concurrent jobs** metrics describe the amount of jobs running at any given time.

Performing a dry-run migration

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

Running the dry-run command

To perform a dry run of migrating a Bitbucket pipeline to GitHub Actions, run the following command in your terminal, replacing `:workspace` with the name of the workspace and `:repo` with the name of the repository in Bitbucket.

```
gh actions-importer dry-run bitbucket --workspace :workspace --repository :repo -
-output-dir tmp/dry-run
```

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."

Performing a production migration

You can use the migrate command to convert a Bitbucket pipeline and open a pull request with the equivalent GitHub Actions workflow(s).

Running the migrate command

To migrate a Bitbucket pipeline to GitHub Actions, run the following command in your terminal, replacing the following values.

- Replace `target-url` value with the URL for your GitHub repository.
- Replace `:repo` with the name of the repository in Bitbucket.
- Replace `:workspace` with the name of the workspace.

```
gh actions-importer migrate bitbucket --workspace :workspace --repository :repo -
-target-url https://github.com/:owner/:repo --output-dir tmp/dry-run
```

The command's output includes the URL of 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 bitbucket --workspace actions-importer --repository
custom-trigger --target-url https://github.com/valet-dev-testing/demo-private --
output-dir tmp/bitbucket
[2023-07-18 09:56:06] Logs: 'tmp/bitbucket/log/valet-20230718-165606.log'
[2023-07-18 09:56:24] Pull request: 'https://github.com/valet-dev-testing/demo-
private/pull/55'
```

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 repository.

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

Reference

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

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 Bitbucket instance.

- `GITHUB_ACCESS_TOKEN`: The personal access token (classic) used to create pull requests with a transformed workflow (requires `repo` and `workflow` scopes).
- `GITHUB_INSTANCE_URL`: The url to the target GitHub instance. (e.g. `https://github.com`)
- `BITBUCKET_ACCESS_TOKEN`: The workspace access token with read scopes for pipeline, project, and repository.

These environment variables can be specified in a `.env.local` file that will be loaded by GitHub Actions Importer at run time. The distribution archive contains a `.env.local.template` file that can be used to create these files.

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 `dry-run` or `migrate` subcommands.

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

For example:

```
gh actions-importer dry-run bitbucket --workspace :workspace --repository :repo -
-output-dir tmp/dry-run --source-file-path path/to/my/pipeline/file.yml
```

`--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 the Bitbucket instance. The `--config-file-path` argument tells GitHub Actions Importer to use the specified source files instead.

Audit example

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

```
gh actions-importer audit bitbucket --workspace :workspace --output-dir tmp/audit
--config-file-path "path/to/my/bitbucket/config.yml"
```

To audit a Bitbucket instance using a config file, the config file must be in the following

format, and each `repository_slug` must be unique:

```
source_files:
  - repository_slug: repo_name
    path: path/to/one/source/file.yml
  - repository_slug: another_repo_name
    path: path/to/another/source/file.yml
```

Supported syntax for Bitbucket Pipelines

The following table shows the type of properties that GitHub Actions Importer is currently able to convert.

Bitbucket	GitHub Actions	Status
`after-script`	`jobs.<job_id>.steps[*]`	Supported
`artifacts`	`actions/upload-artifact` & `download-artifact`	Supported
`caches`	`actions/cache`	Supported
`clone`	`actions/checkout`	Supported
`condition`	`job.<job_id>.steps[*].run`	Supported
`deployment`	`jobs.<job_id>.environmen`	Supported
`image`	`jobs.<job_id>.container`	Supported
`max-time`	`jobs.<job_id>.steps[*].timeout-minutes`	Supported
`options.docker`	None	Supported
`options.max-time`	`jobs.<job_id>.steps[*].timeout-minutes`	Supported
`parallel`	`jobs.<job_id>`	Supported
`pipelines.branches`	`on.push`	Supported
`pipelines.custom`	`on.workflow_dispatch`	Supported
`pipelines.default`	`on.push`	Supported
`pipelines.pull-requests`	`on.pull_requests`	Supported
`pipelines.tags`	`on.tags`	Supported
`runs-on`	`jobs.<job_id>.runs-on`	Supported
`script`	`job.<job_id>.steps[*].run`	Supported
`services`	`jobs.<job_id>.service`	Supported
`stage`	`jobs.<job_id>`	Supported

`step`	`jobs.<job_id>.steps[*]`	Supported
`trigger`	`on.workflow_dispatch`	Supported
`fail-fast`	None	Unsupported
`oidc`	None	Unsupported
`options.size`	None	Unsupported
`size`	None	Unsupported

Environment variable mapping

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

Bitbucket	GitHub Actions
`CI`	`true`
`BITBUCKET_BUILD_NUMBER`	`${{ github.run_number }}`
`BITBUCKET_CLONE_DIR`	`${{ github.workspace }}`
`BITBUCKET_COMMIT`	`${{ github.sha }}`
`BITBUCKET_WORKSPACE`	`${{ github.repository_owner }}`
`BITBUCKET_REPO_SLUG`	`${{ github.repository }}`
`BITBUCKET_REPO_UUID`	`${{ github.repository_id }}`
`BITBUCKET_REPO_FULL_NAME`	`${{ github.repository_owner }}` / `${{ github.repository }}`
`BITBUCKET_BRANCH`	`${{ github.ref }}`
`BITBUCKET_TAG`	`${{ github.ref }}`
`BITBUCKET_PR_ID`	`${{ github.event.pull_request.number }}`
`BITBUCKET_PR_DESTINATION_BRANCH`	`${{ github.event.pull_request.base.ref }}`
`BITBUCKET_GIT_HTTP_ORIGIN`	`${{ github.event.repository.clone_url }}`
`BITBUCKET_GIT_SSH_ORIGIN`	`${{ github.event.repository.ssh_url }}`
`BITBUCKET_EXIT_CODE`	`${{ job.status }}`
`BITBUCKET_STEP_UUID`	`${{ job.github_job }}`
`BITBUCKET_PIPELINE_UUID`	`${{ github.workflow }}`
`BITBUCKET_PROJECT_KEY`	`${{ github.repository_owner }}`
`BITBUCKET_PROJECT_UUID`	`${{ github.repository_owner }}`
`BITBUCKET_STEP_TRIGGERER_UUID`	`${{ github.actor_id }}`

`BITBUCKET_SSH_KEY_FILE`	`${{ github.workspace }}/.ssh/id_rsa`
`BITBUCKET_STEP_OIDC_TOKEN`	No Mapping
`BITBUCKET_DEPLOYMENT_ENVIRONMENT`	No Mapping
`BITBUCKET_DEPLOYMENT_ENVIRONMENT_UUID`	No Mapping
`BITBUCKET_BOOKMARK`	No Mapping
`BITBUCKET_PARALLEL_STEP`	No Mapping
`BITBUCKET_PARALLEL_STEP_COUNT`	No Mapping

System Variables

System variables used in tasks are transformed to the equivalent bash shell variable and are assumed to be available. For example, `${system.<variable.name>}` will be transformed to `$variable_name`. We recommend you verify this to ensure proper operation of the workflow.

Legal notice

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