Migrating from Azure DevOps with GitHub Actions Importer

In this article

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

Legal notice

About migrating from Azure DevOps with GitHub Actions Importer

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

Prerequisites

- An Azure DevOps account or organization with projects and pipelines that you want to convert to GitHub Actions workflows.
- Access to create an Azure DevOps personal access 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 Azure DevOps to GitHub Actions with GitHub Actions Importer:

- GitHub Actions Importer requires version 5.0 of the Azure DevOps API, available in either Azure DevOps Services or Azure DevOps Server 2019. Older versions of Azure DevOps Server are not compatible.
- Tasks that are implicitly added to an Azure DevOps pipeline, such as checking out source code, may be added to a GitHub Actions Importer audit as a GUID name. To find the friendly task name for a GUID, you can use the following URL: `https://dev.azure.com/:organization/_apis/distributedtask/tasks/:guid`.

Manual tasks

Certain Azure DevOps constructs must be migrated manually from Azure DevOps into GitHub Actions configurations. These include:

- Organization, repository, and environment secrets
- Service connections such as OIDC Connect, GitHub Apps, and personal access tokens
- Unknown tasks
- Self-hosted agents
- Environments
- Pre-deployment approvals

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

Unsupported tasks

GitHub Actions Importer does not support migrating the following tasks:

- Pre-deployment gates
- Post-deployment gates
- Post-deployment approvals
- Some resource triggers

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 Azure DevOps 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 an Azure DevOps personal access token. For more information, see Use personal access tokens in the Azure DevOps documentation. The token must have the following scopes:

   - Agents Pool: `Read`
   - Build: `Read`
   - Code: `Read`
   - Release: `Read`
   - Service Connections: `Read`
   - Task Groups: `Read`
   - Variable Groups: `Read`

   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 `Azure DevOps`, 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 Azure DevOps", enter the value for the Azure DevOps personal access token that you created earlier, and press `Enter`.
   - For "Base url of the Azure DevOps instance", press `Enter` to accept the default value (`https://dev.azure.com`).
   - For "Azure DevOps organization name", enter the name for your Azure DevOps organization, and press `Enter`.
   - For "Azure DevOps project name", enter the name for your Azure DevOps project, and press `Enter`.

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

     ```
     $ gh actions-importer configure
     ✔ Which CI providers are you configuring?: Azure DevOps
     ```

```
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 Azure DevOps: ***************
✔ Base url of the Azure DevOps instance: https://dev.azure.com
✔ Azure DevOps organization name: :organization
✔ Azure DevOps project name: :project
Environment variables successfully updated.
```

4. In your terminal, run the GitHub Actions Importer `update` CLI command to connect to the 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 Azure DevOps

You can use the `audit` command to get a high-level view of all projects in an Azure DevOps organization.

The `audit` command performs the following steps:

1. Fetches all of the projects defined in an Azure DevOps organization.
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 an Azure DevOps organization, run the following command in your terminal:

```
gh actions-importer audit azure-devops --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 GitHub Actions usage

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

Running the forecast command

To perform a forecast of potential GitHub Actions usage, 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 azure-devops --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.
- **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 Azure DevOps. 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

You can use the `dry-run` command to convert an Azure DevOps 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.

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."

Running the dry-run command for a build pipeline

To perform a dry run of migrating your Azure DevOps build pipeline to GitHub Actions, run the following command in your terminal, replacing `pipeline_id` with the ID of the pipeline you are converting.

```
gh actions-importer dry-run azure-devops pipeline --pipeline-id :pipeline_id --outpu
```

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

Running the dry-run command for a release pipeline

To perform a dry run of migrating your Azure DevOps release pipeline to GitHub Actions, run the following command in your terminal, replacing `pipeline_id` with the ID of the pipeline you are converting.

```
gh actions-importer dry-run azure-devops release --pipeline-id :pipeline_id --output
```

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

Perform a production migration

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

Running the migrate command for a build pipeline

To migrate an Azure DevOps build pipeline to GitHub Actions, run the following command in your terminal, replacing the `target-url` value with the URL for your GitHub repository, and `pipeline_id` with the ID of the pipeline you are converting.

```
gh actions-importer migrate azure-devops pipeline --pipeline-id :pipeline_id --targe
```

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 azure-devops pipeline --target-url https://github.com/
[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'
```

Running the migrate command for a release pipeline

To migrate an Azure DevOps release pipeline to GitHub Actions, run the following command in your terminal, replacing the `target-url` value with the URL for your GitHub repository, and `pipeline_id` with the ID of the pipeline you are converting.

```
gh actions-importer migrate azure-devops release --pipeline-id :pipeline_id --target
```

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 azure-devops release --target-url https://github.com/
[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 Azure DevOps.

Configuration 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 Azure DevOps instance:

- `GITHUB_ACCESS_TOKEN`: The personal access token used to create pull requests with a converted workflow (requires the `workflow` scope).
- `GITHUB_INSTANCE_URL`: The URL to the target GitHub instance (for example, `https://github.com`).
- `AZURE_DEVOPS_ACCESS_TOKEN`: The personal access token used to authenticate with your Azure DevOps instance. This token requires the following scopes:
  - Build: `Read`
  - Agent Pools: `Read`
  - Code: `Read`
  - Release: `Read`
  - Service Connections: `Read`
  - Task Groups: `Read`

    - Variable Groups: `Read`

- `AZURE_DEVOPS_PROJECT`: The project name or GUID to use when migrating a pipeline. If you'd like to perform an audit on all projects, this is optional.
- `AZURE_DEVOPS_ORGANIZATION`: The organization name of your Azure DevOps instance.
- `AZURE_DEVOPS_INSTANCE_URL`: The URL to the Azure DevOps instance, such as `https://dev.azure.com`.

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

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 `migrate` 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.

For example:

```
gh actions-importer dry-run azure-devops --output-dir ./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.

The `--config-file-path` argument can also be used to specify which repository a converted reusable workflow or composite action should be migrated to.

Audit example

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

```
gh actions-importer audit azure-devops pipeline --output-dir ./output/ --config-file
```

To audit an Azure DevOps instance using a configuration file, the configuration file must be in the following format and each `repository_slug` must be unique:

```yaml
source_files:
  - repository_slug: azdo-project/1
    path: file.yml
  - repository_slug: azdo-project/2
    paths: path.yml
```

You can generate the `repository_slug` for a pipeline by combining the Azure DevOps organization name, project name, and the pipeline ID. For example, `my-organization-name/my-project-name/42`.

Dry run example

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

The pipeline is selected by matching the `repository_slug` in the configuration file to the value of the `--azure-devops-organization` and `--azure-devops-project` option. The `path` is then used to pull the specified source file.

```
gh actions-importer dry-run azure-devops pipeline --output-dir ./output/ --config-f
```

Specify the repository of converted reusable workflows and composite actions

GitHub Actions Importer uses the YAML file provided to the `--config-file-path` argument to determine the repository that converted reusable workflows and composite actions are migrated to.

To begin, you should run an audit without the `--config-file-path` argument:

```
gh actions-importer audit azure-devops --output-dir ./output/
```

The output of this command will contain a file named `config.yml` that contains a list of all the reusable workflows and composite actions that were converted by GitHub Actions Importer. For example, the `config.yml` file may have the following contents:

```yaml
reusable_workflows:
  - name: my-reusable-workflow.yml
    target_url: https://github.com/octo-org/octo-repo
    ref: main

composite_actions:
  - name: my-composite-action.yml
    target_url: https://github.com/octo-org/octo-repo
    ref: main
```

You can use this file to specify which repository and ref a reusable workflow or composite action should be added to. You can then use the `--config-file-path` argument to provide the `config.yml` file to GitHub Actions Importer. For example, you can use this file when running a `migrate` command to open a pull request for each unique repository defined in the config file:

```
gh actions-importer migrate azure-devops pipeline  --config-file-path config.yml --
```

Supported syntax for Azure DevOps pipelines

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

Azure Pipelines	GitHub Actions	Status
condition	• `jobs.<job_id>.if` • `jobs.<job_id>.steps[*].if`	Supported

container	• `jobs.<job_id>.container` • `jobs.<job_id>.name`	Supported
continuousIntegration	• `on.<push>.<branches>` • `on.<push>.<tags>` • `on.<push>.paths`	Supported
job	• `jobs.<job_id>`	Supported
pullRequest	• `on.<pull_request>.<branches>` • `on.<pull_request>.paths`	Supported
stage	• `jobs`	Supported
steps	• `jobs.<job_id>.steps`	Supported
strategy	• `jobs.<job_id>.strategy.fail-fast` • `jobs.<job_id>.strategy.max-parallel` • `jobs.<job_id>.strategy.matrix`	Supported
timeoutInMinutes	• `jobs.<job_id>.timeout-minutes`	Supported
variables	• `env` • `jobs.<job_id>.env` • `jobs.<job_id>.steps.env`	Supported
manual deployment	• `jobs.<job_id>.environment`	Partially supported
pool	• `runners` • `self hosted runners`	Partially supported
services	• `jobs.<job_id>.services`	Partially supported
strategy	• `jobs.<job_id>.strategy`	Partially supported
triggers	• `on`	Partially supported
pullRequest	• `on.<pull_request>.<tags>`	Unsupported
schedules	• `on.schedule` • `on.workflow_run`	Unsupported
triggers	• `on.<event_name>.types`	Unsupported

For more information about supported Azure DevOps tasks, see the `github/gh-actions-importer` repository.

Environment variable mapping

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

Azure Pipelines	GitHub Actions
`$(Agent.BuildDirectory)`	`${{ runner.workspace }}`
`$(Agent.HomeDirectory)`	`${{ env.HOME }}`
`$(Agent.JobName)`	`${{ github.job }}`
`$(Agent.OS)`	`${{ runner.os }}`
`$(Agent.ReleaseDirectory)`	`${{ github.workspace}}`
`$(Agent.RootDirectory)`	`${{ github.workspace }}`
`$(Agent.ToolsDirectory)`	`${{ runner.tool_cache }}`
`$(Agent.WorkFolder)`	`${{ github.workspace }}`
`$(Build.ArtifactStagingDirectory)`	`${{ runner.temp }}`
`$(Build.BinariesDirectory)`	`${{ github.workspace }}`
`$(Build.BuildId)`	`${{ github.run_id }}`
`$(Build.BuildNumber)`	`${{ github.run_number }}`
`$(Build.DefinitionId)`	`${{ github.workflow }}`
`$(Build.DefinitionName)`	`${{ github.workflow }}`
`$(Build.PullRequest.TargetBranch)`	`${{ github.base_ref }}`
`$(Build.PullRequest.TargetBranch.Name)`	`${{ github.base_ref }}`
`$(Build.QueuedBy)`	`${{ github.actor }}`
`$(Build.Reason)`	`${{ github.event_name }}`
`$(Build.Repository.LocalPath)`	`${{ github.workspace }}`
`$(Build.Repository.Name)`	`${{ github.repository }}`
`$(Build.Repository.Provider)`	`GitHub`
`$(Build.Repository.Uri)`	`${{ github.server.url }}/${{ github.repository }}`
`$(Build.RequestedFor)`	`${{ github.actor }}`
`$(Build.SourceBranch)`	`${{ github.ref }}`
`$(Build.SourceBranchName)`	`${{ github.ref }}`
`$(Build.SourceVersion)`	`${{ github.sha }}`
`$(Build.SourcesDirectory)`	`${{ github.workspace }}`
`$(Build.StagingDirectory)`	`${{ runner.temp }}`
`$(Pipeline.Workspace)`	`${{ runner.workspace }}`
`$(Release.DefinitionEnvironmentId)`	`${{ github.job }}`

`$(Release.DefinitionId)`	`${{ github.workflow }}`
`$(Release.DefinitionName)`	`${{ github.workflow }}`
`$(Release.Deployment.RequestedFor)`	`${{ github.actor }}`
`$(Release.DeploymentID)`	`${{ github.run_id }}`
`$(Release.EnvironmentId)`	`${{ github.job }}`
`$(Release.EnvironmentName)`	`${{ github.job }}`
`$(Release.Reason)`	`${{ github.event_name }}`
`$(Release.RequestedFor)`	`${{ github.actor }}`
`$(System.ArtifactsDirectory)`	`${{ github.workspace }}`
`$(System.DefaultWorkingDirectory)`	`${{ github.workspace }}`
`$(System.HostType)`	`build`
`$(System.JobId)`	`${{ github.job }}`
`$(System.JobName)`	`${{ github.job }}`
`$(System.PullRequest.PullRequestId)`	`${{ github.event.number }}`
`$(System.PullRequest.PullRequestNumber)`	`${{ github.event.number }}`
`$(System.PullRequest.SourceBranch)`	`${{ github.ref }}`
`$(System.PullRequest.SourceRepositoryUri)`	`${{ github.server.url }}/${{ github.repository }}`
`$(System.PullRequest.TargetBranch)`	`${{ github.event.base.ref }}`
`$(System.PullRequest.TargetBranchName)`	`${{ github.event.base.ref }}`
`$(System.StageAttempt)`	`${{ github.run_number }}`
`$(System.TeamFoundationCollectionUri)`	`${{ github.server.url }}/${{ github.repository }}`
`$(System.WorkFolder)`	`${{ github.workspace }}`

Templates

You can transform Azure DevOps templates with GitHub Actions Importer.

Limitations

GitHub Actions Importer is able to transform Azure DevOps templates with some limitations.

- Azure DevOps templates used under the `stages`, `deployments`, and `jobs` keys are converted into reusable workflows in GitHub Actions. For more information, see "Reusing workflows."
- Azure DevOps templates used under the `steps` key are converted into composite

actions. For more information, see "Creating a composite action."

- If you currently have job templates that reference other job templates, GitHub Actions Importer converts the templates into reusable workflows. Because reusable workflows cannot reference other reusable workflows, this is invalid syntax in GitHub Actions. You must manually correct nested reusable workflows.
- If a template references an external Azure DevOps organization or GitHub repository, you must use the `--credentials-file` option to provide credentials to access this template. For more information, see "Supplemental arguments and settings."
- You can dynamically generate YAML using `each` expressions with the following caveats:
  - Nested `each` blocks are not supported and cause the parent `each` block to be unsupported.
  - `each` and contained `if` conditions are evaluated at transformation time, because GitHub Actions does not support this style of insertion.
  - `elseif` blocks are unsupported. If this functionality is required, you must manually correct them.
  - Nested `if` blocks are supported, but `if/elseif/else` blocks nested under an `if` condition are not.
  - `if` blocks that use predefined Azure DevOps variables are not supported.

Supported templates

GitHub Actions Importer supports the templates listed in the table below.

Azure Pipelines	GitHub Actions	Status
Extending from a template	`Reusable workflow`	Supported
Job templates	`Reusable workflow`	Supported
Stage templates	`Reusable workflow`	Supported
Step templates	`Composite action`	Supported
Task groups in classic editor	Varies	Supported
Templates in a different Azure DevOps organization, project, or repository	Varies	Supported
Templates in a GitHub repository	Varies	Supported
Variable templates	`env`	Supported
Conditional insertion	`if` conditions on job/steps	Partially supported
Iterative insertion	Not applicable	Partially supported
Templates with parameters	Varies	Partially supported

Template file path names

GitHub Actions Importer can extract templates with relative or dynamic file paths with variable, parameter, and iterative expressions in the file name. However, there must be a default value set.

Variable file path name example

```
# File: azure-pipelines.yml
variables:
- template: 'templates/vars.yml'

steps:
- template: "./templates/$"
```

```
# File: templates/vars.yml
variables:
  one: 'simple_step.yml'
```

Parameter file path name example

```
parameters:
- name: template
  type: string
  default: simple_step.yml

steps:
- template: "./templates/${{ parameters.template }}"
```

Iterative file path name example

```
parameters:
- name: steps
  type: object
  default:
  - build_step
  - release_step
steps:
- ${{ each step in parameters.steps }}:
    - template: "$-variables.yml"
```

Template parameters

GitHub Actions Importer supports the parameters listed in the table below.

Azure Pipelines	GitHub Actions	Status
string	`inputs.string`	Supported
number	`inputs.number`	Supported
boolean	`inputs.boolean`	Supported
object	`inputs.string` with `fromJSON` expression	Partially supported
step	`step`	Partially supported
stepList	`step`	Partially supported
job	`job`	Partially supported
jobList	`job`	Partially supported

deployment	`job`	Partially supported
deploymentList	`job`	Partially supported
stage	`job`	Partially supported
stageList	`job`	Partially supported

Note: A template used under the `step` key with this parameter type is only serialized as a composite action if the steps are used at the beginning or end of the template steps. A template used under the `stage`, `deployment`, and `job` keys with this parameter type are not transformed into a reusable workflow, and instead are serialized as a standalone workflow.

Legal notice

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