Forgejo Actions | Advanced features

- pre-commit hook
- Artifacts
- Uploading to a package registry
- Cache
  - Caching with Podman
- Services
- Grouping of log messages
- Workflow Run Prioritization
- Job Summaries

This page covers the more advanced features of Forgejo Actions.

pre-commit hook

A `pre-commit` hook is provided to assist in validating Forgejo Actions files in your projects. See pre-commit.com for detailed instructions on how to set it up, as well as more details on how it works, and then add this to your `.pre-commit-config.yaml` file:

```
repos:
  - repo: https://code.forgejo.org/forgejo/runner
    rev: v11.1.1
    hooks:
      - id: forgejo-runner-validate
```

Artifacts

`Artifacts` let you persist data after a job has completed. An artifact is a file or collection of files produced during a workflow run.

This has two main uses:

- Storing non-permanent data after a workflow finishes, like test results.
- Storing data generated in one job, so another job later in the same workflow can use it.

Artifacts expire after a delay of 90 days by default, but this may be changed by the instance admin.

The artifacts created by a workflow can be downloaded from the web interface that shows the details of the jobs for a workflow.

download artifacts

Check out the example based on the upload-artifact action and the download-artifact action.

> The name of an artifact is limited to 255 characters.
>
> It is not currently possible to access artifacts from other workflow runs within Forgejo #8004.
>
> When using upload and download actions you must use v3 or patched v4 as demonstrated here.

Uploading to a package registry

Forgejo offers built-in package registries. You can use these to publish packages for various package managers, or just generic build output.

Cache

You can use the `Cache` to reuse commonly used files from previous workflow runs, like package downloads or intermediary build results.

Some Actions will automatically use the cache. For instance the setup-go action will do that by default to save downloading and compiling packages found in `go.mod`.

It is also possible to explicitly control what is cached (and when) by using the actions/cache action. Caching makes no guarantees that a cache will be populated, even when two `jobs` are run in sequence. If you want to reliably exchange information between `jobs`, see `artifacts`.

See also the set of examples.

> actions/cache will use `zstd` if present when compressing files to be sent to the cache. It is faster than the default compression. A container which does not have `zstd` installed *will not* be able to decompress the cache and will continue without. In general it is recommended to ensure that if you're using `zstd` in one place in your project, you use it everywhere.
>
> If the runner is not configured to provide a cache, actions/cache will fail with the following error: `Cache action is only supported on GHES version >= 3.5`.

Caching with Podman

Since version 5, when used with Pasta, which is the default, Podman copies the IP of the primary interface of the host into containers. Therefore, it is not possible to connect to the host using the host's primary IP address, which is the method used by `actions/cache`. It manifests in errors like `::warning::Failed to restore: getCacheEntry failed: connect ECONNREFUSED 192.168.124.60:41275`.

There are multiple options to deal with it.

One option is to configure Podman not to use the host's primary IP address for containers; see the article Podman 5.0 breaking changes in detail, section "Pasta default for rootless networking".

Another option is to change the Forgejo Runner configuration to use the "magic" host name `host.containers.internal`:

```
cache:
  proxy_port: 4000
  actions_cache_url_override: 'http://host.containers.internal:4000'
```

However, that is only an option if a particular Forgejo Runner instance only uses Podman as a back-end and not, for example, Podman and LXC, because all back-ends would receive

`http://host.containers.internal:4000` as cache URL.

Services

Services like databases can be started from container images and are then available to all steps in a particular job:

```
jobs:
  my-job:
    runs-on: docker
    services:
      pgsql:
        image: postgres:15
        env:
          POSTGRES_DB: test
          POSTGRES_PASSWORD: postgres
    ...
```

A container with the specified `image:` is run before the job starts and is terminated when it completes. The job can address the service using its name, in this case `pgsql`.

The IP address of `pgsql` is on the same network as the container running the **steps** and there is no need for port binding. The `postgres:15` image exposes the PostgreSQL port 5432 and a client will be able to connect as shown in this example.

> Services are only supported with the `docker` backend, not with the `lxc` backend.

> Services require `[container].network` to be empty so that a new network can be temporarily created to connect the container running the workflow with the services.

See the reference for the list of supported options. There are also a number of examples that illustrate the configuration and usage of services.

Grouping of log messages

To create an expandable section of the logs, use `##[group]` followed by a headline and end the section with `##[endgroup]`. Expansions can be nested.

```
jobs:
  fold-example:
    runs-on: docker
    steps:
      - name: Expandable log lines
        run: |
          echo "##[group]Headline"
          echo "Initially, this text is not visible"
          echo "##[group]Subheadline"
          echo "This text isn't even visible after the first expansion! "
          echo "##[endgroup]"
          echo "This is still hidden at first"
          echo "##[endgroup]"
          echo "This is never hidden."
```

As can be seen in the example, the `##[endgroup]` lines are not shown themselves, but are counted in the line numbers.

grouped log message

Workflow Run Prioritization

By default, Forgejo Actions jobs are run in the order they arrive. However, Forgejo offers the ability to manually mark entire workflow runs as prioritized. Prioritized workflow runs are executed before all others.

List of workflow runs with one that is prioritized

Workflow runs can be prioritized by clicking on *Options* in the workflow run list followed by *Prioritize run*. If a run should no longer be prioritized, the flag can be cleared by clicking on *Options* followed by *Do not prioritize run*.

When a workflow run is prioritized, Forgejo will first run the jobs that belong to that particular workflow run. When multiple runs are prioritized, Forgejo will run them in the order they arrived, not in the order they have been prioritized.

Prioritization is best-effort. That means that it may not have an effect in some circumstances, for example, when a workflow run is already running. Also, the run order is currently not exposed over the HTTP API or in webhook messages. That means that external systems cannot participate.

Job Summaries

A job summary is a composition of the summaries produced by each step. The content (Markdown) is rendered on top of the log output as soon as it is appended by a step.

actionSummarySample

> Each step can append up to 1MiB of Markdown. Surplus content is automatically truncated.

Each individual step of a Forgejo Actions job can render additional Markdown content. Using Mermaid and KaTeX is also supported.

Creating a summary is done by writing it into the file behind `FORGEJO_STEP_SUMMARY`. This example can be produced by:

```
on:
  push:
    branches:
      - '*'

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v7

      - name: Run tests
        run: |
          # This application writes into $GITHUB_STEP_SUMMARY which is respected here as well
          dotnet run -c Release --project "./tests/UnitTests/UnitTests.csproj"

      - name: Add simple mermaid chart
        run: |
          cat << 'EOF' >> "$FORGEJO_STEP_SUMMARY"

          Mermaid's official [git graph sample](https://mermaid.ai/open-source/syntax/gitgraph.html)

          ```mermaid
```
```

```
gitGraph
   commit
   commit
   branch develop
   commit
   commit
   commit
   checkout main
   commit
   commit
```

EOF

□□□ #2
□ □□□ □□□ 2026-10-31 13:35:45 UTC
□ □□□ □□□ 2026-10-31 13:35:45 UTC