This version of GitHub Enterprise was discontinued on 2023-03-15. No patch releases will be made, even for critical security issues. For better performance, improved security, and new features, upgrade to the latest version of GitHub Enterprise. For help with the upgrade, contact GitHub Enterprise support.

Dockerfile support for GitHub Actions

In this article

When creating a `Dockerfile` for a Docker container action, you should be aware of how some Docker instructions interact with GitHub Actions and an action's metadata file.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

About Dockerfile instructions

A `Dockerfile` contains instructions and arguments that define the contents and startup behavior of a Docker container. For more information about the instructions Docker supports, see "Dockerfile reference" in the Docker documentation.

Dockerfile instructions and overrides

Some Docker instructions interact with GitHub Actions, and an action's metadata file can override some Docker instructions. Ensure that you are familiar with how your Dockerfile interacts with GitHub Actions to prevent any unexpected behavior.

USER

Docker actions must be run by the default Docker user (root). Do not use the `USER` instruction in your `Dockerfile`, because you won't be able to access the `GITHUB_WORKSPACE` directory. For more information, see "Variables" and USER reference in the Docker documentation.

FROM

The first instruction in the `Dockerfile` must be `FROM`, which selects a Docker base image. For more information, see the FROM reference in the Docker documentation.

These are some best practices when setting the `FROM` argument:

- It's recommended to use official Docker images. For example, `python` or `ruby`.
- Use a version tag if it exists, preferably with a major version. For example, use

`node:10` instead of `node:latest`.

- It's recommended to use Docker images based on the Debian operating system.

WORKDIR

GitHub Enterprise Server sets the working directory path in the `GITHUB_WORKSPACE` environment variable. It's recommended to not use the `WORKDIR` instruction in your `Dockerfile`. Before the action executes, GitHub Enterprise Server will mount the `GITHUB_WORKSPACE` directory on top of anything that was at that location in the Docker image and set `GITHUB_WORKSPACE` as the working directory. For more information, see "Variables" and the WORKDIR reference in the Docker documentation.

ENTRYPOINT

If you define `entrypoint` in an action's metadata file, it will override the `ENTRYPOINT` defined in the `Dockerfile`. For more information, see "Metadata syntax for GitHub Actions."

The Docker `ENTRYPOINT` instruction has a *shell* form and *exec* form. The Docker `ENTRYPOINT` documentation recommends using the *exec* form of the `ENTRYPOINT` instruction. For more information about *exec* and *shell* form, see the ENTRYPOINT reference in the Docker documentation.

You should not use `WORKDIR` to specify your entrypoint in your Dockerfile. Instead, you should use an absolute path. For more information, see WORKDIR.

If you configure your container to use the *exec* form of the `ENTRYPOINT` instruction, the `args` configured in the action's metadata file won't run in a command shell. If the action's `args` contain an environment variable, the variable will not be substituted. For example, using the following *exec* format will not print the value stored in `$GITHUB_SHA`, but will instead print `"$GITHUB_SHA"`.

```
ENTRYPOINT ["echo $GITHUB_SHA"]
```

If you want variable substitution, then either use the *shell* form or execute a shell directly. For example, using the following *exec* format, you can execute a shell to print the value stored in the `GITHUB_SHA` environment variable.

```
ENTRYPOINT ["sh", "-c", "echo $GITHUB_SHA"]
```

To supply `args` defined in the action's metadata file to a Docker container that uses the *exec* form in the `ENTRYPOINT`, we recommend creating a shell script called `entrypoint.sh` that you call from the `ENTRYPOINT` instruction:

Example *Dockerfile*

```
# Container image that runs your code
FROM debian:9.5-slim

# Copies your code file from your action repository to the filesystem path `/` of th
COPY entrypoint.sh /entrypoint.sh

# Executes `entrypoint.sh` when the Docker container starts up
ENTRYPOINT ["/entrypoint.sh"]
```

Example *entrypoint.sh* file

Using the example Dockerfile above, GitHub Enterprise Server will send the `args` configured in the action's metadata file as arguments to `entrypoint.sh`. Add the `#!/bin/sh` shebang at the top of the `entrypoint.sh` file to explicitly use the system's POSIX-compliant shell.

```
#!/bin/sh

# `$#` expands to the number of arguments and `$@` expands to the supplied `args`
printf '%d args:' "$#"
printf " '%s'" "$@"
printf '\n'
```

Your code must be executable. Make sure the `entrypoint.sh` file has `execute` permissions before using it in a workflow. You can modify the permission from your terminal using this command:

```
chmod +x entrypoint.sh
```

When an `ENTRYPOINT` shell script is not executable, you'll receive an error similar to this:

```
Error response from daemon: OCI runtime create failed: container_linux.go:348: start
```

CMD

If you define `args` in the action's metadata file, `args` will override the `CMD` instruction specified in the `Dockerfile`. For more information, see "Metadata syntax for GitHub Actions".

If you use `CMD` in your `Dockerfile`, follow these guidelines:

1. Document required arguments in the action's README and omit them from the `CMD` instruction.
2. Use defaults that allow using the action without specifying any `args`.
3. If the action exposes a `--help` flag, or something similar, use that to make your action self-documenting.

Supported Linux capabilities

GitHub Actions supports the default Linux capabilities that Docker supports. Capabilities can't be added or removed. For more information about the default Linux capabilities that Docker supports, see "Runtime privilege and Linux capabilities" in the Docker documentation. To learn more about Linux capabilities, see "Overview of Linux capabilities" in the Linux man-pages.

Legal

© 2023 GitHub, Inc. Terms Privacy Status Pricing Expert services Blog