Running jobs in a container

In this article

Use a container to run the steps in a job.

Overview

Use `jobs.<job_id>.container` to create a container to run any steps in a job that don't already specify a container. If you have steps that use both script and container actions, the container actions will run as sibling containers on the same network with the same volume mounts.

If you do not set a `container`, all steps will run directly on the host specified by `runs-on` unless a step refers to an action configured to run in a container.

Note: The default shell for `run` steps inside a container is `sh` instead of `bash`. This can be overridden with `jobs.<job_id>.defaults.run` or `jobs.<job_id>.steps[*].shell`.

Example: Running a job within a container

YAML

```yaml
name: CI
on:
  push:
    branches: [ main ]
jobs:
  container-test-job:
    runs-on: ubuntu-latest
    container:
      image: node:18
      env:
        NODE_ENV: development
      ports:
        - 80
      volumes:
        - my_docker_volume:/volume_mount
      options: --cpus 1
    steps:
      - name: Check for dockerenv file
        run: (ls /.dockerenv && echo Found dockerenv) || (echo No dockerenv)
```

When you only specify a container image, you can omit the `image` keyword.

```yaml
jobs:
  container-test-job:
    runs-on: ubuntu-latest
    container: node:18
```

Defining the container image

Use `jobs.<job_id>.container.image` to define the Docker image to use as the container to run the action. The value can be the Docker Hub image name or a registry name.

Defining credentials for a container registry

If the image's container registry requires authentication to pull the image, you can use `jobs.<job_id>.container.credentials` to set a `map` of the `username` and `password`. The credentials are the same values that you would provide to the `docker login` command.

Example: Defining credentials for a container registry

```yaml
container:
  image: ghcr.io/owner/image
  credentials:
     username: ${{ github.actor }}
     password: ${{ secrets.github_token }}
```

Using environment variables with a container

Use `jobs.<job_id>.container.env` to set a `map` of environment variables in the container.

Exposing network ports on a container

Use `jobs.<job_id>.container.ports` to set an `array` of ports to expose on the container.

Mounting volumes in a container

Use `jobs.<job_id>.container.volumes` to set an `array` of volumes for the container to use. You can use volumes to share data between services or other steps in a job. You can specify named Docker volumes, anonymous Docker volumes, or bind mounts on the host.

To specify a volume, you specify the source and destination path:

`<source>:<destinationPath>`.

The `<source>` is a volume name or an absolute path on the host machine, and `<destinationPath>` is an absolute path in the container.

Example: Mounting volumes in a container

```yaml
volumes:
  - my_docker_volume:/volume_mount
  - /data/my_data
```

```
    - /source/directory:/destination/directory
```

Setting container resource options

Use `jobs.<job_id>.container.options` to configure additional Docker container resource options. For a list of options, see "`docker create` options."

Warning: The `--network` and `--entrypoint` options are not supported.