Customizing the containers used by jobs

In this article

About container customization
Container customization commands
Generating the customization script
Triggering the customization script
Troubleshooting

You can customize how your self-hosted runner invokes a container for a job.

Note: This feature is currently in beta and is subject to change.

About container customization

GitHub Actions allows you to run a job within a container, using the `container:` statement in your workflow file. For more information, see "Running jobs in a container." To process container-based jobs, the self-hosted runner creates a container for each job.

GitHub Actions supports commands that let you customize the way your containers are created by the self-hosted runner. For example, you can use these commands to manage the containers through Kubernetes or Podman, and you can also customize the `docker run` or `docker create` commands used to invoke the container. The customization commands are run by a script, which is automatically triggered when a specific environment variable is set on the runner. For more information, see "Triggering the customization script" below.

This customization is only available for Linux-based self-hosted runners, and root user access is not required.

Container customization commands

GitHub Actions includes the following commands for container customization:

- `prepare_job`: Called when a job is started.
- `cleanup_job`: Called at the end of a job.
- `run_container_step`: Called once for each container action in the job.
- `run_script_step`: Runs any step that is not a container action.

Each of these customization commands must be defined in its own JSON file. The file name must match the command name, with the extension `.json`. For example, the `prepare_job` command is defined in `prepare_job.json`. These JSON files will then be run together on the self-hosted runner, as part of the main `index.js` script. This process is described in more detail in "Generating the customization script."

These commands also include configuration arguments, explained below in more detail.

`prepare_job`

The `prepare_job` command is called when a job is started. GitHub Actions passes in any job or service containers the job has. This command will be called if you have any service or job containers in the job.

GitHub Actions assumes that you will do the following tasks in the `prepare_job` command:

- Prune anything from previous jobs, if needed.
- Create a network, if needed.
- Pull the job and service containers.
- Start the job container.
- Start the service containers.
- Write to the response file any information that GitHub Actions will need:
    - Required: State whether the container is an `alpine` linux container (using the `isAlpine` boolean).
    - Optional: Any context fields you want to set on the job context, otherwise they will be unavailable for users to use. For more information, see "Contexts."
- Return `0` when the health checks have succeeded and the job/service containers are started.

Arguments for `prepare_job`

- `jobContainer`: **Optional**. An object containing information about the specified job container.
    - `image`: **Required**. A string containing the Docker image.
    - `workingDirectory`: **Required**. A string containing the absolute path of the working directory.
    - `createOptions`: **Optional**. The optional *create* options specified in the YAML. For more information, see "Running jobs in a container."
    - `environmentVariables`: **Optional**. Sets a map of key environment variables.
    - `userMountVolumes`: **Optional**. An array of user mount volumes set in the YAML. For more information, see "Running jobs in a container."
        - `sourceVolumePath`: **Required**. The source path to the volume that will be mounted into the Docker container.
        - `targetVolumePath`: **Required**. The target path to the volume that will be mounted into the Docker container.
        - `readOnly`: **Required**. Determines whether or not the mount should be read-only.
    - `systemMountVolumes`: **Required**. An array of mounts to mount into the container, same fields as above.
        - `sourceVolumePath`: **Required**. The source path to the volume that will be mounted into the Docker container.
        - `targetVolumePath`: **Required**. The target path to the volume that will be mounted into the Docker container.
        - `readOnly`: **Required**. Determines whether or not the mount should be read-only.
    - `registry` **Optional**. The Docker registry credentials for a private container registry.

        - `username`: **Optional**. The username of the registry account.
        - `password`: **Optional**. The password to the registry account.
        - `serverUrl`: **Optional**. The registry URL.
    - `portMappings`: **Optional**. A key value hash of *source:target* ports to map into the container.

- `services`: **Optional**. An array of service containers to spin up.
    - `contextName`: **Required**. The name of the service in the Job context.
    - `image`: **Required**. A string containing the Docker image.
    - `createOptions`: **Optional**. The optional *create* options specified in the YAML. For more information, see "Running jobs in a container."
    - `environmentVariables`: **Optional**. Sets a map of key environment variables.
    - `userMountVolumes`: **Optional**. An array of mounts to mount into the container, same fields as above.
        - `sourceVolumePath`: **Required**. The source path to the volume that will be mounted into the Docker container.
        - `targetVolumePath`: **Required**. The target path to the volume that will be mounted into the Docker container.
        - `readOnly`: **Required**. Determines whether or not the mount should be read-only.
    - `registry` **Optional**. The Docker registry credentials for the private container registry.
        - `username`: **Optional**. The username of the registry account.
        - `password`: **Optional**. The password to the registry account.
        - `serverUrl`: **Optional**. The registry URL.
    - `portMappings`: **Optional**. A key value hash of *source:target* ports to map into the container.

Example input for `prepare_job`

JSON


```
{
  "command": "prepare_job",
  "responseFile": "/users/octocat/runner/_work/{guid}.json",
  "state": {},
  "args": {
    "jobContainer": {
      "image": "node:18"
      "workingDirectory": "/__w/octocat-test2/octocat-test2",
      "createOptions": "--cpus 1",
      "environmentVariables": {
        "NODE_ENV": "development"
      },
      "userMountVolumes": [
        {
          "sourceVolumePath": "my_docker_volume",
          "targetVolumePath": "/volume_mount",
          "readOnly": false
        }
      ],
      "systemMountVolumes": [
        {
          "sourceVolumePath": "/home/octocat/git/runner/_layout/_work",
          "targetVolumePath": "/__w",
          "readOnly": false
        },
```

```
        {
          "sourceVolumePath": "/home/octocat/git/runner/_layout/externals",
          "targetVolumePath": "/__e",
          "readOnly": true
        },
        {
          "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_temp",
          "targetVolumePath": "/__w/_temp",
          "readOnly": false
        },
        {
          "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_actions",
          "targetVolumePath": "/__w/_actions",
          "readOnly": false
        },
        {
          "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_tool",
          "targetVolumePath": "/__w/_tool",
          "readOnly": false
        },
        {
          "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_home",
          "targetVolumePath": "/github/home",
          "readOnly": false
        },
        {
          "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_workflow",
          "targetVolumePath": "/github/workflow",
          "readOnly": false
        }
      ],
      "registry": {
        "username": "octocat",
        "password": "examplePassword",
        "serverUrl": "https://index.docker.io/v1"
      },
      "portMappings": { "80": "801" }
    },
    "services": [
      {
        "contextName": "redis",
        "image": "redis",
        "createOptions": "--cpus 1",
        "environmentVariables": {},
        "userMountVolumes": [],
        "portMappings": { "80": "801" },
        "registry": {
          "username": "octocat",
          "password": "examplePassword",
          "serverUrl": "https://index.docker.io/v1"
        }
      }
    ]
  }
}
```


Example output for `prepare_job`

This example output is the contents of the `responseFile` defined in the input above.

JSON


```
{
  "state": {
    "network": "example_network_53269bd575972817b43f7733536b200c",
    "jobContainer":
```

```
"82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
    "serviceContainers": {
      "redis": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105"
    }
  },
  "context": {
    "container": {
      "id": "82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
      "network": "example_network_53269bd575972817b43f7733536b200c"
    },
    "services": {
      "redis": {
        "id": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105",
        "ports": {
          "8080": "8080"
        },
        "network": "example_network_53269bd575972817b43f7733536b200c"
      }
    },
    "isAlpine": true
  }
}
```

`cleanup_job`

The `cleanup_job` command is called at the end of a job. GitHub Actions assumes that you will do the following tasks in the `cleanup_job` command:

- Stop any running service or job containers (or the equivalent pod).
- Stop the network (if one exists).
- Delete any job or service containers (or the equivalent pod).
- Delete the network (if one exists).
- Cleanup anything else that was created for the job.

Arguments for `cleanup_job`

No arguments are provided for `cleanup_job`.

Example input for `cleanup_job`

JSON

```
{
  "command": "cleanup_job",
  "responseFile": null,
  "state": {
    "network": "example_network_53269bd575972817b43f7733536b200c",
    "jobContainer":
"82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
    "serviceContainers": {
      "redis": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105"
    }
  },
  "args": {}
}
```

Example output for `cleanup_job`

No output is expected for `cleanup_job`.

`run_container_step`

The `run_container_step` command is called once for each container action in your job. GitHub Actions assumes that you will do the following tasks in the `run_container_step` command:

- Pull or build the required container (or fail if you cannot).
- Run the container action and return the exit code of the container.
- Stream any step logs output to stdout and stderr.
- Cleanup the container after it executes.

Arguments for `run_container_step`

- `image`: **Optional**. A string containing the docker image. Otherwise a dockerfile must be provided.
- `dockerfile`: **Optional**. A string containing the path to the dockerfile, otherwise an image must be provided.
- `entryPointArgs`: **Optional**. A list containing the entry point args.
- `entryPoint`: **Optional**. The container entry point to use if the default image entrypoint should be overwritten.
- `workingDirectory`: **Required**. A string containing the absolute path of the working directory.
- `createOptions`: **Optional**. The optional *create* options specified in the YAML. For more information, see "Running jobs in a container."
- `environmentVariables`: **Optional**. Sets a map of key environment variables.
- `prependPath`: **Optional**. An array of additional paths to prepend to the `$PATH` variable.
- `userMountVolumes`: **Optional**. an array of user mount volumes set in the YAML. For more information, see "Running jobs in a container."
  - `sourceVolumePath`: **Required**. The source path to the volume that will be mounted into the Docker container.
  - `targetVolumePath`: **Required**. The target path to the volume that will be mounted into the Docker container.
  - `readOnly`: **Required**. Determines whether or not the mount should be read-only.
- `systemMountVolumes`: **Required**. An array of mounts to mount into the container, using the same fields as above.
  - `sourceVolumePath`: **Required**. The source path to the volume that will be mounted into the Docker container.
  - `targetVolumePath`: **Required**. The target path to the volume that will be mounted into the Docker container.
  - `readOnly`: **Required**. Determines whether or not the mount should be read-only.
- `registry` **Optional**. The Docker registry credentials for a private container registry.
  - `username`: **Optional**. The username of the registry account.
  - `password`: **Optional**. The password to the registry account.
  - `serverUrl`: **Optional**. The registry URL.
- `portMappings`: **Optional**. A key value hash of the *source:target* ports to map into the container.

Example input for image

If you're using a Docker image, you can specify the image name in the `"image":` parameter.

JSON


```
{
  "command": "run_container_step",
  "responseFile": null,
  "state": {
    "network": "example_network_53269bd575972817b43f7733536b200c",
    "jobContainer":
"82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
    "serviceContainers": {
      "redis": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105"
    }
  },
  "args": {
    "image": "node:18",
    "dockerfile": null,
    "entryPointArgs": ["-f", "/dev/null"],
    "entryPoint": "tail",
    "workingDirectory": "/__w/octocat-test2/octocat-test2",
    "createOptions": "--cpus 1",
    "environmentVariables": {
      "NODE_ENV": "development"
    },
    "prependPath": ["/foo/bar", "bar/foo"],
    "userMountVolumes": [
      {
        "sourceVolumePath": "my_docker_volume",
        "targetVolumePath": "/volume_mount",
        "readOnly": false
      }
    ],
    "systemMountVolumes": [
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work",
        "targetVolumePath": "/__w",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/externals",
        "targetVolumePath": "/__e",
        "readOnly": true
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_temp",
        "targetVolumePath": "/__w/_temp",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_actions",
        "targetVolumePath": "/__w/_actions",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_tool",
        "targetVolumePath": "/__w/_tool",
        "readOnly": false
      },
      {
        "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_home",
        "targetVolumePath": "/github/home",
        "readOnly": false
      },
      {
        "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_workflow",
        "targetVolumePath": "/github/workflow",
        "readOnly": false
      }
    ],
```

```
    "registry": null,
    "portMappings": { "80": "801" }
  }
}
```

Example input for Dockerfile

If your container is defined by a Dockerfile, this example demonstrates how to specify the path to a `Dockerfile` in your input, using the `"dockerfile":` parameter.

JSON


```
{
  "command": "run_container_step",
  "responseFile": null,
  "state": {
    "network": "example_network_53269bd575972817b43f7733536b200c",
    "jobContainer":
"82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
    "services": {
      "redis": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105"
    }
  },
  "args": {
    "image": null,
    "dockerfile": "/__w/_actions/foo/dockerfile",
    "entryPointArgs": ["hello world"],
    "entryPoint": "echo",
    "workingDirectory": "/__w/octocat-test2/octocat-test2",
    "createOptions": "--cpus 1",
    "environmentVariables": {
      "NODE_ENV": "development"
    },
    "prependPath": ["/foo/bar", "bar/foo"],
    "userMountVolumes": [
      {
        "sourceVolumePath": "my_docker_volume",
        "targetVolumePath": "/volume_mount",
        "readOnly": false
      }
    ],
    "systemMountVolumes": [
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work",
        "targetVolumePath": "/__w",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/externals",
        "targetVolumePath": "/__e",
        "readOnly": true
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_temp",
        "targetVolumePath": "/__w/_temp",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_actions",
        "targetVolumePath": "/__w/_actions",
        "readOnly": false
      },
      {
        "sourceVolumePath": "/home/octocat/git/runner/_layout/_work/_tool",
        "targetVolumePath": "/__w/_tool",
        "readOnly": false
      },
```

```
      {
        "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_home",
        "targetVolumePath": "/github/home",
        "readOnly": false
      },
      {
        "sourceVolumePath":
"/home/octocat/git/runner/_layout/_work/_temp/_github_workflow",
        "targetVolumePath": "/github/workflow",
        "readOnly": false
      }
    ],
    "registry": null,
    "portMappings": { "80": "801" }
  }
}
```

Example output for `run_container_step`

No output is expected for `run_container_step`.

`run_script_step`

GitHub Actions assumes that you will do the following tasks:

- Invoke the provided script inside the job container and return the exit code.
- Stream any step log output to stdout and stderr.

Arguments for `run_script_step`

- `entryPointArgs`: **Optional**. A list containing the entry point arguments.
- `entryPoint`: **Optional**. The container entry point to use if the default image entrypoint should be overwritten.
- `prependPath`: **Optional**. An array of additional paths to prepend to the `$PATH` variable.
- `workingDirectory`: **Required**. A string containing the absolute path of the working directory.
- `environmentVariables`: **Optional**. Sets a map of key environment variables.

Example input for `run_script_step`

JSON

```
{
  "command": "run_script_step",
  "responseFile": null,
  "state": {
    "network": "example_network_53269bd575972817b43f7733536b200c",
    "jobContainer":
"82e8219701fe096a35941d869cf3d71af1d943b5d8bdd718857fb87ac3042480",
    "serviceContainers": {
      "redis": "60972d9aa486605e66b0dad4abb678dc3d9116f536579e418176eedb8abb9105"
    }
  },
  "args": {
    "entryPointArgs": ["-e", "/runner/temp/example.sh"],
    "entryPoint": "bash",
    "environmentVariables": {
      "NODE_ENV": "development"
    },
    "prependPath": ["/foo/bar", "bar/foo"],
```

```
    "workingDirectory": "/__w/octocat-test2/octocat-test2"
  }
}
```

Example output for `run_script_step`

No output is expected for `run_script_step`.

Generating the customization script

GitHub has created an example repository that demonstrates how to generate customization scripts for Docker and Kubernetes.

> **Note:** The resulting scripts are available for testing purposes, and you will need to determine whether they are appropriate for your requirements.

1. Clone the actions/runner-container-hooks repository to your self-hosted runner.
2. The `examples/` directory contains some existing customization commands, each with its own JSON file. You can review these examples and use them as a starting point for your own customization commands.
   - `prepare_job.json`
   - `run_script_step.json`
   - `run_container_step.json`
3. Build the npm packages. These commands generate the `index.js` files inside `packages/docker/dist` and `packages/k8s/dist`.

   ```
   npm install && npm run bootstrap && npm run build-all
   ```

When the resulting `index.js` is triggered by GitHub Actions, it will run the customization commands defined in the JSON files. To trigger the `index.js`, you will need to add it your `ACTIONS_RUNNER_REQUIRE_JOB_CONTAINER` environment variable, as described in the next section.

Triggering the customization script

The custom script must be located on the runner, but should not be stored in the self-hosted runner application directory (that is, the directory into which you downloaded and unpacked the runner software). The scripts are executed in the security context of the service account that's running the runner service.

> **Note**: The triggered script is processed synchronously, so it will block job execution while running.

The script is automatically executed when the runner has the following environment variable containing an absolute path to the script:

- `ACTIONS_RUNNER_CONTAINER_HOOKS`: The script defined in this environment variable is triggered when a job has been assigned to a runner, but before the job starts running.

To set this environment variable, you can either add it to the operating system, or add it

to a file named `.env` within the self-hosted runner application directory. For example, the following `.env` entry will have the runner automatically run the script at `/Users/octocat/runner/index.js` before each container-based job runs:

```
ACTIONS_RUNNER_CONTAINER_HOOKS=/Users/octocat/runner/index.js
```

If you want to ensure that your job always runs inside a container, and subsequently always applies your container customizations, you can set the `ACTIONS_RUNNER_REQUIRE_JOB_CONTAINER` variable on the self hosted runner to `true`. This will fail jobs that do not specify a job container.

Troubleshooting

No timeout setting

There is currently no timeout setting available for the script executed by `ACTIONS_RUNNER_CONTAINER_HOOKS`. As a result, you could consider adding timeout handling to your script.

Reviewing the workflow run log

To confirm whether your scripts are executing, you can review the logs for that job. For more information on checking the logs, see "Using workflow run logs."