About service containers

In this article

About service containers
Communicating with service containers
Creating service containers
Mapping Docker host and service container ports
Further reading

You can use service containers to connect databases, web services, memory caches, and other tools to your workflow.

About service containers

Service containers are Docker containers that provide a simple and portable way for you to host services that you might need to test or operate your application in a workflow. For example, your workflow might need to run integration tests that require access to a database and memory cache.

You can configure service containers for each job in a workflow. GitHub creates a fresh Docker container for each service configured in the workflow, and destroys the service container when the job completes. Steps in a job can communicate with all service containers that are part of the same job. However, you cannot create and use service containers inside a composite action.

Note: If your workflows use Docker container actions, job containers, or service containers, then you must use a Linux runner:

- If you are using GitHub-hosted runners, you must use an Ubuntu runner.
- If you are using self-hosted runners, you must use a Linux machine as your runner and Docker must be installed.

Communicating with service containers

You can configure jobs in a workflow to run directly on a runner machine or in a Docker container. Communication between a job and its service containers is different depending on whether a job runs directly on the runner machine or in a container.

Running jobs in a container

When you run jobs in a container, GitHub connects service containers to the job using Docker's user-defined bridge networks. For more information, see "Use bridge networks" in the Docker documentation.

Running the job and services in a container simplifies network access. You can access a service container using the label you configure in the workflow. The hostname of the service container is automatically mapped to the label name. For example, if you create a service container with the label `redis`, the hostname of the service container is

`redis`.

You don't need to configure any ports for service containers. By default, all containers that are part of the same Docker network expose all ports to each other, and no ports are exposed outside of the Docker network.

Running jobs on the runner machine

When running jobs directly on the runner machine, you can access service containers using `localhost:<port>` or `127.0.0.1:<port>`. GitHub configures the container network to enable communication from the service container to the Docker host.

When a job runs directly on a runner machine, the service running in the Docker container does not expose its ports to the job on the runner by default. You need to map ports on the service container to the Docker host. For more information, see "About service containers."

Creating service containers

You can use the `services` keyword to create service containers that are part of a job in your workflow. For more information, see `jobs.<job_id>.services`.

This example creates a service called `redis` in a job called `container-job`. The Docker host in this example is the `node:16-bullseye` container.

YAML

```yaml
name: Redis container example
on: push

jobs:
  # Label of the container job
  container-job:
    # Containers must run in Linux based operating systems
    runs-on: ubuntu-latest
    # Docker Hub image that `container-job` executes in
    container: node:16-bullseye

    # Service containers to run with `container-job`
    services:
      # Label used to access the service container
      redis:
        # Docker Hub image
        image: redis
```

Mapping Docker host and service container ports

If your job runs in a Docker container, you do not need to map ports on the host or the service container. If your job runs directly on the runner machine, you'll need to map any required service container ports to ports on the host runner machine.

You can map service containers ports to the Docker host using the `ports` keyword. For more information, see `jobs.<job_id>.services`.

Value of `ports`	Description
`8080:80`	Maps TCP port 80 in the container to port 8080 on the Docker host.
`8080:80/udp`	Maps UDP port 80 in the container to port 8080 on the Docker host.

	on the Docker host.
`8080/udp`	Map a randomly chosen UDP port in the container to UDP port 8080 on the Docker host.

When you map ports using the `ports` keyword, GitHub uses the `--publish` command to publish the container's ports to the Docker host. For more information, see "Docker container networking" in the Docker documentation.

When you specify the Docker host port but not the container port, the container port is randomly assigned to a free port. GitHub sets the assigned container port in the service container context. For example, for a `redis` service container, if you configured the Docker host port 5432, you can access the corresponding container port using the `job.services.redis.ports[5432]` context. For more information, see "Contexts."

Example mapping Redis ports

This example maps the service container `redis` port 6379 to the Docker host port 6379.

YAML

```yaml
name: Redis Service Example
on: push

jobs:
  # Label of the container job
  runner-job:
    # You must use a Linux environment when using service containers or container
jobs
    runs-on: ubuntu-latest

    # Service containers to run with `runner-job`
    services:
      # Label used to access the service container
      redis:
        # Docker Hub image
        image: redis
        #
        ports:
          # Opens tcp port 6379 on the host and service container
          - 6379:6379
```

Further reading

- "Creating Redis service containers"
- "Creating PostgreSQL service containers"