# Creating Redis service containers

**In this article**



You can use service containers to create a Redis client in your workflow. This guide shows examples of creating a Redis service for jobs that run in containers or directly on the runner machine.

**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.

## Introduction

This guide shows you workflow examples that configure a service container using the Docker Hub `redis` image. The workflow runs a script to create a Redis client and populate the client with data. To test that the workflow creates and populates the Redis client, the script prints the client's data to the console.

**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.

## Prerequisites

You should be familiar with how service containers work with GitHub Actions and the networking differences between running jobs directly on the runner or in a container. For more information, see "About service containers."

You may also find it helpful to have a basic understanding of YAML, the syntax for GitHub Actions, and Redis. For more information, see:

- "Learn GitHub Actions"
- "Getting Started with Redis" in the Redis documentation

## Running jobs in containers

Configuring jobs to run in a container simplifies networking configurations between the job and the service containers. Docker containers on the same user-defined bridge

network expose all ports to each other, so you don't need to map any of the service container ports to the Docker host. You can access the service container from the job container using the label you configure in the workflow.

You can copy this workflow file to the `.github/workflows` directory of your repository and modify it as needed.

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:10.18-jessie

    # Service containers to run with `container-job`
    services:
      # Label used to access the service container
      redis:
        # Docker Hub image
        image: redis
        # Set health checks to wait until redis has started
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      # Downloads a copy of the code in your repository before running CI tests
      - name: Check out repository code
        uses: actions/checkout@v4

      # Performs a clean installation of all dependencies in the `package.json`
file
      # For more information, see https://docs.npmjs.com/cli/ci.html
      - name: Install dependencies
        run: npm ci

      - name: Connect to Redis
        # Runs a script that creates a Redis client, populates
        # the client with data, and retrieves data
        run: node client.js
        # Environment variable used by the `client.js` script to create a new
Redis client.
        env:
          # The hostname used to communicate with the Redis service container
          REDIS_HOST: redis
          # The default Redis port
          REDIS_PORT: 6379
```

## Configuring the container job

This workflow configures a job that runs in the `node:10.18-jessie` container and uses the `ubuntu-latest` GitHub-hosted runner as the Docker host for the container. For more information about the `node:10.18-jessie` container, see the node image on Docker Hub.

The workflow configures a service container with the label `redis`. All services must run in a container, so each service requires that you specify the container `image`. This example uses the `redis` container image, and includes health check options to make sure the service is running. Append a tag to the image name to specify a version, e.g.

`redis:6`. For more information, see the redis image on Docker Hub.

YAML

```yaml
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:10.18-jessie

    # Service containers to run with `container-job`
    services:
      # Label used to access the service container
      redis:
        # Docker Hub image
        image: redis
        # Set health checks to wait until redis has started
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
```

## Configuring the steps for the container job

The workflow performs the following steps:

1. Checks out the repository on the runner
2. Installs dependencies
3. Runs a script to create a client

YAML

```yaml
steps:
  # Downloads a copy of the code in your repository before running CI tests
  - name: Check out repository code
    uses: actions/checkout@v4

  # Performs a clean installation of all dependencies in the `package.json` file
  # For more information, see https://docs.npmjs.com/cli/ci.html
  - name: Install dependencies
    run: npm ci

  - name: Connect to Redis
    # Runs a script that creates a Redis client, populates
    # the client with data, and retrieves data
    run: node client.js
    # Environment variable used by the `client.js` script to create a new Redis
client.
    env:
      # The hostname used to communicate with the Redis service container
      REDIS_HOST: redis
      # The default Redis port
      REDIS_PORT: 6379
```

The *client.js* script looks for the `REDIS_HOST` and `REDIS_PORT` environment variables to create the client. The workflow sets those two environment variables as part of the "Connect to Redis" step to make them available to the *client.js* script. For more information about the script, see "Testing the Redis service container."

The hostname of the Redis service is the label you configured in your workflow, in this case, `redis`. Because Docker containers on the same user-defined bridge network open all ports by default, you'll be able to access the service container on the default Redis port 6379.

## Running jobs directly on the runner machine

When you run a job directly on the runner machine, you'll need to map the ports on the service container to ports on the Docker host. You can access service containers from the Docker host using `localhost` and the Docker host port number.

You can copy this workflow file to the `.github/workflows` directory of your repository and modify it as needed.

YAML

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

jobs:
  # Label of the runner 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
        # Set health checks to wait until redis has started
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          # Maps port 6379 on service container to the host
          - 6379:6379

    steps:
      # Downloads a copy of the code in your repository before running CI tests
      - name: Check out repository code
        uses: actions/checkout@v4

      # Performs a clean installation of all dependencies in the `package.json`
file
      # For more information, see https://docs.npmjs.com/cli/ci.html
      - name: Install dependencies
        run: npm ci

      - name: Connect to Redis
        # Runs a script that creates a Redis client, populates
        # the client with data, and retrieves data
        run: node client.js
        # Environment variable used by the `client.js` script to create
        # a new Redis client.
        env:
          # The hostname used to communicate with the Redis service container
          REDIS_HOST: localhost
          # The default Redis port
          REDIS_PORT: 6379
```

## Configuring the runner job

The example uses the `ubuntu-latest` GitHub-hosted runner as the Docker host.

The workflow configures a service container with the label `redis`. All services must run in a container, so each service requires that you specify the container `image`. This example uses the `redis` container image, and includes health check options to make sure the service is running. Append a tag to the image name to specify a version, e.g. `redis:6`. For more information, see the redis image on Docker Hub.

The workflow maps port 6379 on the Redis service container to the Docker host. For more information about the `ports` keyword, see "About service containers."

YAML

```yaml
jobs:
  # Label of the runner 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
        # Set health checks to wait until redis has started
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          # Maps port 6379 on service container to the host
          - 6379:6379
```

## Configuring the steps for the runner job

The workflow performs the following steps:

1. Checks out the repository on the runner
2. Installs dependencies
3. Runs a script to create a client

YAML

```yaml
steps:
  # Downloads a copy of the code in your repository before running CI tests
  - name: Check out repository code
    uses: actions/checkout@v4

  # Performs a clean installation of all dependencies in the `package.json` file
  # For more information, see https://docs.npmjs.com/cli/ci.html
  - name: Install dependencies
    run: npm ci

  - name: Connect to Redis
    # Runs a script that creates a Redis client, populates
    # the client with data, and retrieves data
```

```
    run: node client.js
    # Environment variable used by the `client.js` script to create
    # a new Redis client.
    env:
      # The hostname used to communicate with the Redis service container
      REDIS_HOST: localhost
      # The default Redis port
      REDIS_PORT: 6379
```

The *client.js* script looks for the `REDIS_HOST` and `REDIS_PORT` environment variables to create the client. The workflow sets those two environment variables as part of the "Connect to Redis" step to make them available to the *client.js* script. For more information about the script, see "Testing the Redis service container."

The hostname is `localhost` or `127.0.0.1`.

## Testing the Redis service container

You can test your workflow using the following script, which creates a Redis client and populates the client with some placeholder data. The script then prints the values stored in the Redis client to the terminal. Your script can use any language you'd like, but this example uses Node.js and the `redis` npm module. For more information, see the npm redis module.

You can modify *client.js* to include any Redis operations needed by your workflow. In this example, the script creates the Redis client instance, adds placeholder data, then retrieves the data.

Add a new file called *client.js* to your repository with the following code.

JavaScript

```
const redis = require("redis");

// Creates a new Redis client
// If REDIS_HOST is not set, the default host is localhost
// If REDIS_PORT is not set, the default port is 6379
const redisClient = redis.createClient({
  url: `redis://${process.env.REDIS_HOST}:${process.env.REDIS_PORT}`
});

redisClient.on("error", (err) => console.log("Error", err));

(async () => {
  await redisClient.connect();

  // Sets the key "octocat" to a value of "Mona the octocat"
  const setKeyReply = await redisClient.set("octocat", "Mona the Octocat");
  console.log("Reply: " + setKeyReply);
  // Sets a key to "species", field to "octocat", and "value" to "Cat and
Octopus"
  const SetFieldOctocatReply = await redisClient.hSet("species", "octocat", "Cat
and Octopus");
  console.log("Reply: " + SetFieldOctocatReply);
  // Sets a key to "species", field to "dinotocat", and "value" to "Dinosaur and
Octopus"
  const SetFieldDinotocatReply = await redisClient.hSet("species", "dinotocat",
"Dinosaur and Octopus");
  console.log("Reply: " + SetFieldDinotocatReply);
  // Sets a key to "species", field to "robotocat", and "value" to "Cat and
Robot"
  const SetFieldRobotocatReply = await redisClient.hSet("species", "robotocat",
"Cat and Robot");
  console.log("Reply: " + SetFieldRobotocatReply);
```

```
  try {
    // Gets all fields in "species" key
    const replies = await redisClient.hKeys("species");
    console.log(replies.length + " replies:");
    replies.forEach((reply, i) => {
        console.log("    " + i + ": " + reply);
    });
    await redisClient.quit();
  }
  catch (err) {
    // statements to handle any exceptions
  }
})();
```

The script creates a new Redis client using the `createClient` method, which accepts a `host` and `port` parameter. The script uses the `REDIS_HOST` and `REDIS_PORT` environment variables to set the client's IP address and port. If `host` and `port` are not defined, the default host is `localhost` and the default port is 6379.

The script uses the `set` and `hset` methods to populate the database with some keys, fields, and values. To confirm that the Redis client contains the data, the script prints the contents of the database to the console log.

When you run this workflow, you should see the following output in the "Connect to Redis" step confirming you created the Redis client and added data:

```
Reply: OK
Reply: 1
Reply: 1
Reply: 1
3 replies:
    0: octocat
    1: dinotocat
    2: robotocat
```