Creating PostgreSQL service containers

In this article

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

Introduction

This guide shows you workflow examples that configure a service container using the Docker Hub `postgres` image. The workflow runs a script that connects to the PostgreSQL service, creates a table, and then populates it with data. To test that the workflow creates and populates the PostgreSQL table, the script prints the data from the table 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 PostgreSQL. For more information, see:

- "Learn GitHub Actions"
- "PostgreSQL tutorial" in the PostgreSQL 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: PostgreSQL service 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
      postgres:
        # Docker Hub image
        image: postgres
        # Provide the password for postgres
        env:
          POSTGRES_PASSWORD: postgres
        # Set health checks to wait until postgres has started
        options: >-
          --health-cmd pg_isready
          --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 PostgreSQL
        # Runs a script that creates a PostgreSQL table, populates
        # the table with data, and then retrieves the data.
        run: node client.js
        # Environment variables used by the `client.js` script to create a new
PostgreSQL table.
        env:
          # The hostname used to communicate with the PostgreSQL service
container
          POSTGRES_HOST: postgres
          # The default PostgreSQL port
          POSTGRES_PORT: 5432
```

Configuring the runner job for jobs in containers

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 `postgres`. All services must run in a container, so each service requires that you specify the container `image`. This

example uses the `postgres` container image, provides the default PostgreSQL password, and includes health check options to make sure the service is running. For more information, see the postgres 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
      postgres:
        # Docker Hub image
        image: postgres
        # Provide the password for postgres
        env:
          POSTGRES_PASSWORD: postgres
        # Set health checks to wait until postgres has started
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
```

Configuring the steps for jobs in containers

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 PostgreSQL
    # Runs a script that creates a PostgreSQL table, populates
    # the table with data, and then retrieves the data.
    run: node client.js
    # Environment variable used by the `client.js` script to create
    # a new PostgreSQL client.
    env:
      # The hostname used to communicate with the PostgreSQL service container
      POSTGRES_HOST: postgres
      # The default PostgreSQL port
      POSTGRES_PORT: 5432
```

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

The hostname of the PostgreSQL service is the label you configured in your workflow, in this case, `postgres`. 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 PostgreSQL port 5432.

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: PostgreSQL Service 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
      postgres:
        # Docker Hub image
        image: postgres
        # Provide the password for postgres
        env:
          POSTGRES_PASSWORD: postgres
        # Set health checks to wait until postgres has started
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          # Maps tcp port 5432 on service container to the host
          - 5432:5432

    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 PostgreSQL
        # Runs a script that creates a PostgreSQL table, populates
        # the table with data, and then retrieves the data
        run: node client.js
        # Environment variables used by the `client.js` script to create
```

```
        # a new PostgreSQL table.
        env:
          # The hostname used to communicate with the PostgreSQL service
container
          POSTGRES_HOST: localhost
          # The default PostgreSQL port
          POSTGRES_PORT: 5432
```

Configuring the runner job for jobs directly on the runner machine

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

The workflow configures a service container with the label `postgres`. All services must run in a container, so each service requires that you specify the container `image`. This example uses the `postgres` container image, provides the default PostgreSQL password, and includes health check options to make sure the service is running. For more information, see the postgres image on Docker Hub.

The workflow maps port 5432 on the PostgreSQL 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
      postgres:
        # Docker Hub image
        image: postgres
        # Provide the password for postgres
        env:
          POSTGRES_PASSWORD: postgres
        # Set health checks to wait until postgres has started
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          # Maps tcp port 5432 on service container to the host
          - 5432:5432
```

Configuring the steps for jobs directly on the runner machine

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 PostgreSQL
    # Runs a script that creates a PostgreSQL table, populates
    # the table with data, and then retrieves the data
    run: node client.js
    # Environment variables used by the `client.js` script to create
    # a new PostgreSQL table.
    env:
      # The hostname used to communicate with the PostgreSQL service container
      POSTGRES_HOST: localhost
      # The default PostgreSQL port
      POSTGRES_PORT: 5432
```

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

The hostname is `localhost` or `127.0.0.1`.

Testing the PostgreSQL service container

You can test your workflow using the following script, which connects to the PostgreSQL service and adds a new table with some placeholder data. The script then prints the values stored in the PostgreSQL table to the terminal. Your script can use any language you'd like, but this example uses Node.js and the `pg` npm module. For more information, see the npm pg module.

You can modify *client.js* to include any PostgreSQL operations needed by your workflow. In this example, the script connects to the PostgreSQL service, adds a table to the `postgres` database, inserts some placeholder data, and then retrieves the data.

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

JavaScript

```javascript
const { Client } = require('pg');

const pgclient = new Client({
    host: process.env.POSTGRES_HOST,
    port: process.env.POSTGRES_PORT,
    user: 'postgres',
    password: 'postgres',
    database: 'postgres'
});

pgclient.connect();

const table = 'CREATE TABLE student(id SERIAL PRIMARY KEY, firstName VARCHAR(40)
NOT NULL, lastName VARCHAR(40) NOT NULL, age INT, address VARCHAR(80), email
VARCHAR(40))'
const text = 'INSERT INTO student(firstname, lastname, age, address, email)
VALUES($1, $2, $3, $4, $5) RETURNING *'
const values = ['Mona the', 'Octocat', 9, '88 Colin P Kelly Jr St, San Francisco,
CA 94107, United States', 'octocat@github.com']
```

```
pgclient.query(table, (err, res) => {
    if (err) throw err
});

pgclient.query(text, values, (err, res) => {
    if (err) throw err
});

pgclient.query('SELECT * FROM student', (err, res) => {
    if (err) throw err
    console.log(err, res.rows) // Print the data in student table
    pgclient.end()
});
```

The script creates a new connection to the PostgreSQL service, and uses the `POSTGRES_HOST` and `POSTGRES_PORT` environment variables to specify the PostgreSQL service IP address and port. If `host` and `port` are not defined, the default host is `localhost` and the default port is 5432.

The script creates a table and populates it with placeholder data. To test that the `postgres` database contains the data, the script prints the contents of the table to the console log.

When you run this workflow, you should see the following output in the "Connect to PostgreSQL" step, which confirms that you successfully created the PostgreSQL table and added data:

```
null [ { id: 1,
    firstname: 'Mona the',
    lastname: 'Octocat',
    age: 9,
    address:
     '88 Colin P Kelly Jr St, San Francisco, CA 94107, United States',
    email: 'octocat@github.com' } ]
```