Search for great content Explore Pricing Sign In Sign U

Explore Official Images postgres

postgres

Official Image

The PostgreSQL object-relational database system provides reliability and data integrity.

1B+

Container Linux mips64le 386 PowerPC 64 LE ARM 64 x86-64 ARM IBM Z

Databases Official Image

Copy and paste to pull this image

```
docker pull postgres
```

View Available Tags

Description Reviews Tags

Note: the description for this image is longer than the Hub length limit of 25000, so has been trimmed. The full description can be found at https://github.com/docker-library/docs/tree/master/postgres/README.md. See docker/hub-beta-feedback#238 for more information.

Quick reference

- **Maintained by**: the PostgreSQL Docker Community
- **Where to get help**: the Docker Community Forums, the Docker Community Slack, or Stack Overflow

Sponsored Resources

- Running PostgreSQL Database in a Cloud Native Environment with Kubernetes
- Installing Cloud Native Postgres Operator from EDB
- Gartner Report: CTO's Guide to Containers and Kubernetes

Supported tags and respective `Dockerfile` links

- `14.0`, `14`, `latest`, `14.0-bullseye`, `14-bullseye`, `bullseye`
- `14.0-alpine`, `14-alpine`, `alpine`, `14.0-alpine3.14`, `14-alpine3.14`, `alpine3.14`
- `13.4`, `13`, `13.4-bullseye`, `13-bullseye`
- `13.4-alpine`, `13-alpine`, `13.4-alpine3.14`, `13-alpine3.14`
- `12.8`, `12`, `12.8-bullseye`, `12-bullseye`
- `12.8-alpine`, `12-alpine`, `12.8-alpine3.14`, `12-alpine3.14`
- `11.13-bullseye`, `11-bullseye`
- `11.13`, `11`, `11.13-stretch`, `11-stretch`
- `11.13-alpine`, `11-alpine`, `11.13-alpine3.14`, `11-alpine3.14`

- `10.18-bullseye` , `10-bullseye`
- `10.18` , `10` , `10.18-stretch` , `10-stretch`
- `10.18-alpine` , `10-alpine` , `10.18-alpine3.14` , `10-alpine3.14`
- `9.6.23-bullseye` , `9.6-bullseye` , `9-bullseye`
- `9.6.23` , `9.6` , `9` , `9.6.23-stretch` , `9.6-stretch` , `9-stretch`
- `9.6.23-alpine` , `9.6-alpine` , `9-alpine` , `9.6.23-alpine3.14` , `9.6-alpine3.14` , `9-alpine3.14`

Quick reference (cont.)

- **Where to file issues**: https://github.com/docker-library/postgres/issues
- **Supported architectures**: (more info) `amd64` , `arm32v5` , `arm32v6` , `arm32v7` , `arm64v8` , `i386` , `mips64le` , `ppc64le` , `s390x`
- **Published image artifact details**: repo-info repo's `repos/postgres/` directory (history) (image metadata, transfer size, etc)
- **Image updates**: official-images repo's `library/postgres` label
  official-images repo's `library/postgres` file (history)
- **Source of this description**: docs repo's `postgres/` directory (history)

What is PostgreSQL?

PostgreSQL, often simply "Postgres", is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards-compliance. As a database server, its primary function is to store data, securely and supporting best practices, and retrieve it later, as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet). It can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users. Recent versions also provide replication of the database itself for security and scalability.

PostgreSQL implements the majority of the SQL:2011 standard, is ACID-compliant and transactional (including most DDL statements) avoiding locking issues using multiversion concurrency control (MVCC), provides immunity to dirty reads and full serializability; handles complex SQL queries using many indexing methods that are not available in other databases; has updateable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other expandability, and has a large number of extensions written by third parties. In addition to the possibility of working with the major proprietary and open source databases, PostgreSQL supports migration from them, by its extensive standard SQL support and available migration tools. And if proprietary extensions had been used, by its extensibility that can emulate many through some built-in and third-party open source compatibility extensions, such as for Oracle.

> wikipedia.org/wiki/PostgreSQL

How to use this image

start a postgres instance

```
$ docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -d postgres
```

The default `postgres` user and database are created in the entrypoint with `initdb` .

> The postgres database is a default database meant for use by users, utilities and third party applications.
>
> postgresql.org/docs

... or via `psql`

```
$ docker run -it --rm --network some-network postgres psql -h some-postgres -U postgres
psql (9.5.0)
Type "help" for help.

postgres=# SELECT 1;
 ?column?
----------
        1
(1 row)
```

... via `docker stack deploy` or `docker-compose`

Example `stack.yml` for `postgres` :

```
# Use postgres/example user/password credentials
version: '3.1'

services:

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

Try in PWD

Run `docker stack deploy -c stack.yml postgres` (or `docker-compose -f stack.yml up`), wait for it to initialize completely, and visit `http://swarm-ip:8080` , `http://localhost:8080` , or `http://host-ip:8080` (as appropriate).

How to extend this image

There are many ways to extend the `postgres` image. Without trying to support every possible use case, here are just a few that we have found useful.

Environment Variables

The PostgreSQL image uses several environment variables which are easy to miss. The only variable required is `POSTGRES_PASSWORD` , the rest are optional.

Warning: the Docker specific variables will only have an effect if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup.

`POSTGRES_PASSWORD`

This environment variable is required for you to use the PostgreSQL image. It must not be empty or undefined. This environment variable sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable.

Note 1: The PostgreSQL image sets up `trust` authentication locally so you may notice a password is not required when connecting from `localhost` (inside the same container). However, a password will be required if connecting from a different host/container.

Note 2: This variable defines the superuser password in the PostgreSQL instance, as set by the `initdb` script during initial container startup. It has no effect on the `PGPASSWORD` environment variable that may be used by the `psql` client at runtime, as described at https://www.postgresql.org/docs/current/libpq-envars.html. `PGPASSWORD`, if used, will be specified as a separate environment variable.

`POSTGRES_USER`

This optional environment variable is used in conjunction with `POSTGRES_PASSWORD` to set a user and its password. This variable will create the specified user with superuser power and a database with the same name. If it is not specified, then the default user of `postgres` will be used.

Be aware that if this parameter is specified, PostgreSQL will still show `The files belonging to this database system will be owned by user "postgres"` during initialization. This refers to the Linux system user (from `/etc/passwd` in the image) that the `postgres` daemon runs as, and as such is unrelated to the `POSTGRES_USER` option. See the section titled "Arbitrary `--user` Notes" for more details.

`POSTGRES_DB`

This optional environment variable can be used to define a different name for the default database that is created when the image is first started. If it is not specified, then the value of `POSTGRES_USER` will be used.

`POSTGRES_INITDB_ARGS`

This optional environment variable can be used to send arguments to `postgres initdb`. The value is a space separated string of arguments as `postgres initdb` would expect them. This is useful for adding functionality like data page checksums: `-e POSTGRES_INITDB_ARGS="--data-checksums"`.

`POSTGRES_INITDB_WALDIR`

This optional environment variable can be used to define another location for the Postgres transaction log. By default the transaction log is stored in a subdirectory of the main Postgres data folder (`PGDATA`). Sometimes it can be desireable to store the transaction log in a different directory which may be backed by storage with different performance or reliability characteristics.

Note: on PostgreSQL 9.x, this variable is `POSTGRES_INITDB_XLOGDIR` (reflecting the changed name of the `--xlogdir` flag to `--waldir` in PostgreSQL 10+).

`POSTGRES_HOST_AUTH_METHOD`

This optional variable can be used to control the `auth-method` for `host` connections for `all` databases, `all` users, and `all` addresses. If unspecified then `md5` password authentication is used. On an uninitialized database, this will populate `pg_hba.conf` via this approximate line:

```
echo "host all all all $POSTGRES_HOST_AUTH_METHOD" >> pg_hba.conf
```

See the PostgreSQL documentation on `pg_hba.conf` for more information about possible values and their meanings.

Note 1: It is not recommended to use `trust` since it allows anyone to connect without a password, even if one is set (like via `POSTGRES_PASSWORD`). For more information see the PostgreSQL documentation on *Trust Authentication*.

Note 2: If you set `POSTGRES_HOST_AUTH_METHOD` to `trust`, then `POSTGRES_PASSWORD` is not required.

Note 3: If you set this to an alternative value (such as `scram-sha-256`), you might need additional `POSTGRES_INITDB_ARGS` for the database to initialize correctly (such as `POSTGRES_INITDB_ARGS=--auth-host=scram-sha-256`).

`PGDATA`

This optional variable can be used to define another location - like a subdirectory - for the database files. The default is `/var/lib/postgresql/data`. If the data volume you're using is a filesystem mountpoint (like with GCE persistent disks) or remote folder that cannot be chowned to the `postgres` user (like some NFS mounts), Postgres `initdb` recommends a subdirectory be created to contain the data.

For example:

```
$ docker run -d \
    --name some-postgres \
    -e POSTGRES_PASSWORD=mysecretpassword \
    -e PGDATA=/var/lib/postgresql/data/pgdata \
    -v /custom/mount:/var/lib/postgresql/data \
    postgres
```

This is an environment variable that is not Docker specific. Because the variable is used by the `postgres` server binary (see the PostgreSQL docs), the entrypoint script takes it into account.

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to some of the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

```
$ docker run --name some-postgres -e POSTGRES_PASSWORD_FILE=/run/secrets/postgres-passwd -d postgr
```

Currently, this is only supported for `POSTGRES_INITDB_ARGS` , `POSTGRES_PASSWORD` , `POSTGRES_USER` , and `POSTGRES_DB` .

Initialization scripts

If you would like to do additional initialization in an image derived from this one, add one or more `*.sql` , `*.sql.gz` , or `*.sh` scripts under `/docker-entrypoint-initdb.d` (creating the directory if necessary). After the entrypoint calls `initdb` to create the default `postgres` user and database, it will run any `*.sql` files, run any executable `*.sh` scripts, and source any non-executable `*.sh` scripts found in that directory to do further initialization before starting the service.

Warning: scripts in `/docker-entrypoint-initdb.d` are only run if you start the container with a data directory that is empty; any pre-existing database will be left untouched on container startup. One common problem is that if one of your `/docker-entrypoint-initdb.d` scripts fails (which will cause the entrypoint script to exit) and your orchestrator restarts the container with the already initialized data directory, it will not continue on with your scripts.

For example, to add an additional user and database, add the following to `/docker-entrypoint-initdb.d/init-user-db.sh` :

```
#!/bin/bash
set -e

psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
    CREATE USER docker;
    CREATE DATABASE docker;
    GRANT ALL PRIVILEGES ON DATABASE docker TO docker;
EOSQL
```

These initialization files will be executed in sorted name order as defined by the current locale, which defaults to `en_US.utf8` . Any `*.sql` files will be executed by `POSTGRES_USER` , which defaults to the `postgres` superuser. It is recommended that any `psql` commands that are run inside of a `*.sh` script be executed as `POSTGRES_USER` by using the `--username "$POSTGRES_USER"` flag. This user will be able to connect without a password due to the presence of `trust` authentication for Unix socket connections made inside the container.

Additionally, as of docker-library/postgres#253, these initialization scripts are run as the `postgres` user (or as the "semi-arbitrary user" specified with the `--user` flag to `docker run` ; see the section titled "Arbitrary `--user` Notes" for more details). Also, as of docker-library/postgres#440, the temporary daemon started for these initialization scripts listens only on the Unix socket, so any `psql` usage should drop the hostname portion (see docker-library/postgres#474 (comment) for example).

Database Configuration

There are many ways to set PostgreSQL server configuration. For information on what is available to configure, see the postgresql.org docs for the specific version of PostgreSQL that you are running. Here are a few options for setting configuration:

- Use a custom config file. Create a config file and get it into the container. If you need a starting place for your config file you can use the sample provided by PostgreSQL which is available in the container at `/usr/share/postgresql/postgresql.conf.sample` (`/usr/local/share/postgresql/postgresql.conf.sample` in Alpine variants).
    - **Important note:** you must set `listen_addresses = '*'` so that other containers will be able to access postgres.

```
$ # get the default config
$ docker run -i --rm postgres cat /usr/share/postgresql/postgresql.conf.sample > my-postgres

$ # customize the config

$ # run postgres with custom config
$ docker run -d --name some-postgres -v "$PWD/my-postgres.conf":/etc/postgresql/postgresql.c
```

- Set options directly on the run line. The entrypoint script is made so that any options passed to the docker command will be passed along to the `postgres` server daemon. From the docs we see that any option available in a `.conf` file can be set via `-c` .

```
$ docker run -d --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword postgres -c share
```

Locale Customization

You can extend the Debian-based images with a simple `Dockerfile` to set a different locale. The following example will set the default locale to `de_DE.utf8` :

```
FROM postgres:9.4
RUN localedef -i de_DE -c -f UTF-8 -A /usr/share/locale/locale.alias de_DE.UTF-8
ENV LANG de_DE.utf8
```

Since database initialization only happens on container startup, this allows us to set the language before it is created.

Also of note, Alpine-based variants do *not* support locales; see "Character sets and locale" in the musl documentation for more details.

Additional Extensions

When using the default (Debian-based) variants, installing additional extensions (such as PostGIS) should be as simple as installing the relevant packages (see github.com/postgis/docker-postgis for a concrete example).

When using the Alpine variants, any postgres extension not listed in postgres-contrib will need to be compiled in your own image (again, see github.com/postgis/docker-postgis for a concrete example).

Arbitrary `--user` Notes

As of docker-library/postgres#253, this image supports running as a (mostly) arbitrary user via `--user` on `docker run`.

The main caveat to note is that `postgres` doesn't care what UID it runs as (as long as the owner of `/var/lib/postgresql/data` matches), but `initdb` *does* care (and needs the user to exist in `/etc/passwd`):

```
$ docker run -it --rm --user www-data -e POSTGRES_PASSWORD=mysecretpassword postgres
The files belonging to this database system will be owned by user "www-data".
...

$ docker run -it --rm --user 1000:1000 -e POSTGRES_PASSWORD=mysecretpassword postgres
initdb: could not look up effective user ID 1000: user does not exist
```

The three easiest ways to get around this:

1. use the Debian variants (not the Alpine variants) and thus allow the image to use the `nss_wrapper` library to "fake" `/etc/passwd` contents for you (see docker-library/postgres#448 for more details)

2. bind-mount `/etc/passwd` read-only from the host (if the UID you desire is a valid user on your host):

```
$ docker run -it --rm --user "$(id -u):$(id -g)" -v /etc/passwd:/etc/passwd:ro -e POSTGRES_P/
The files belonging to this database system will be owned by user "jsmith".
...
```

3. initialize the target directory separately from the final runtime (with a `chown` in between):

```
$ docker volume create pgdata
$ docker run -it --rm -v pgdata:/var/lib/postgresql/data -e POSTGRES_PASSWORD=mysecretpasswo
The files belonging to this database system will be owned by user "postgres".
...
( once it's finished initializing successfully and is waiting for connections, stop it )
$ docker run -it --rm -v pgdata:/var/lib/postgresql/data bash chown -R 1000:1000 /var/lib/po
$ docker run -it --rm --user 1000:1000 -v pgdata:/var/lib/postgresql/data postgres
LOG:  database system was shut down at 2017-01-20 00:03:23 UTC
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

Caveats

If there is no database when `postgres` starts in a container, then `postgres` will create the default database for you. While this is the expected behavior of `postgres`, this means that it will not accept incoming connections during that time. This may cause issues when using automation tools, such as `docker-compose`, that start several containers simultaneously.

Also note that the default `/dev/shm` size for containers is 64MB. If the shared memory is exhausted you will encounter `ERROR: could not resize shared memory segment . . . : No space left on device`. You will want to pass `--shm-size=256MB` for example to `docker run`, or alternatively in `docker-compose`

See "IPVS connection timeout issue" in the Docker Success Center for details about IPVS connection timeouts which will affect long-running idle connections to PostgreSQL in Swarm Mode using overlay networks.

Where to Store Data

Important note: There are several ways to store data used by applications that run in Docker containers. We encourage users of the `postgres` images to familiarize themselv

What is a Container

Product Offerings

Docker Desktop

Docker Hub

Features

Container Runtime

Developer Tools

Docker App

Kubernetes

Play with Docker

Community

Open Source

Docs

Hub Release Notes

Resources

Blog

Customers

Partners

Newsroom

Events and Webin

Careers

Contact Us