(/)

OFFICIAL REPOSITORY

postgres (/r/_/postgres/) ☆

Last pushed: 5 days ago

Repo Info (/_/postgres/)

Short Description

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

Full Description

Supported tags and respective `Dockerfile` links

- `11.1`, `11`, `latest` (*11/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/11/Dockerfile)
- `11.1-alpine`, `11-alpine`, `alpine` (*11/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/112bf5ee8455f930354bc90f32d6a1c830525ed9/11/alpine/Dockerfile)
- `10.6`, `10` (*10/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/10/Dockerfile)
- `10.6-alpine`, `10-alpine` (*10/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/7c287e7800dc08743da8d6372ab752e5438f662b/10/alpine/Dockerfile)
- `9.6.11`, `9.6`, `9` (*9.6/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/9.6/Dockerfile)
- `9.6.11-alpine`, `9.6-alpine`, `9-alpine` (*9.6/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/cb8d873277a533b2f1140a3f3624321a0c53f45d/9.6/alpine/Dockerfile)
- `9.5.15`, `9.5` (*9.5/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/9.5/Dockerfile)
- `9.5.15-alpine`, `9.5-alpine` (*9.5/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/d48c7ca3c7b8c573a33b9f0598839aa7a06318ff/9.5/alpine/Dockerfile)
- `9.4.20`, `9.4` (*9.4/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/9.4/Dockerfile)
- `9.4.20-alpine`, `9.4-alpine` (*9.4/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/d564da5142b2e5fa235707b05d8aa0fc76250418/9.4/alpine/Dockerfile)
- `9.3.25`, `9.3` (*9.3/Dockerfile*) (https://github.com/docker-library/postgres/blob/d61fd19b699b2c380da08c3a95f7272137a182bb/9.3/Dockerfile)
- `9.3.25-alpine`, `9.3-alpine` (*9.3/alpine/Dockerfile*) (https://github.com/docker-library/postgres/blob/4955a99b600fa0badb1e707285617a23315421af/9.3/alpine/Dockerfile)

Quick reference

- **Where to get help**:
  the Docker Community Forums (https://forums.docker.com/), the Docker Community Slack (https://blog.docker.com/2016/11/introducing-docker-community-directory-docker-community-slack/), or Stack Overflow (https://stackoverflow.com/search?tab=newest&q=docker)

- **Where to file issues**:
  https://github.com/docker-library/postgres/issues (https://github.com/docker-library/postgres/issues)

- **Maintained by**:
  the PostgreSQL Docker Community (https://github.com/docker-library/postgres)

- **Supported architectures**: (more info (https://github.com/docker-library/official-images#architectures-other-than-amd64))
  `amd64` (https://hub.docker.com/r/amd64/postgres/), `arm32v5` (https://hub.docker.com/r/arm32v5/postgres/), `arm32v6` (https://hub.docker.com/r/arm32v6/postgres/), `arm32v7` (https://hub.docker.com/r/arm32v7/postgres/), `arm64v8` (https://hub.docker.com/r/arm64v8/postgres/), `i386` (https://hub.docker.com/r/i386/postgres/), `ppc64le` (https://hub.docker.com/r/ppc64le/postgres/), `s390x` (https://hub.docker.com/r/s390x/postgres/)

- **Published image artifact details**:
  repo-info repo's `repos/postgres/` directory (https://github.com/docker-library/repo-info/blob/master/repos/postgres) (history (https://github.com/docker-library/repo-info/commits/master/repos/postgres))
  (image metadata, transfer size, etc)

- **Image updates**:
  official-images PRs with label `library/postgres` (https://github.com/docker-library/official-images/pulls?q=label%3Alibrary%2Fpostgres)
  official-images repo's `library/postgres` file (https://github.com/docker-library/official-images/blob/master/library/postgres) (history (https://github.com/docker-library/official-images/commits/master/library/postgres))

- **Source of this description**:
  docs repo's `postgres/` directory (https://github.com/docker-library/docs/tree/master/postgres) (history (https://github.com/docker-library/docs/commits/master/postgres))

- **Supported Docker versions**:
  the latest release (https://github.com/docker/docker-ce/releases/latest) (down to 1.6 on a best-effort basis)

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 (https://en.wikipedia.org/wiki/PostgreSQL)

How to use this image

start a postgres instance

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

This image includes `EXPOSE 5432` (the postgres port), so standard container linking will make it automatically available to the linked containers. 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 (http://www.postgresql.org/docs/9.5/interactive/app-initdb.html)

connect to it from an application

```
$ docker run --name some-app --link some-postgres:postgres -d application-that-uses-postgr
```

... or via `psql`

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

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

... via `docker stack deploy` (https://docs.docker.com/engine/reference/commandline/stack_deploy/) or `docker-compose` (https://github.com/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 (http://play-with-docker.com?stack=https://raw.githubusercontent.com/docker-library/docs/9efeec18b6b2ed232cf0fbd3914b6211e16e242c/postgres/stack.yml)

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).

Environment Variables

The PostgreSQL image uses several environment variables which are easy to miss. While none of the variables are required, they may significantly aid you in using the image.

POSTGRES_PASSWORD

This environment variable is recommended for you to use the PostgreSQL image. This environment variable sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable. In the above example, it is being set to "mysecretpassword".

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 inital 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/10/static/libpq-envars.html (https://www.postgresql.org/docs/10/static/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.

PGDATA

This optional environment variable can be used to define another location - like a subdirectory - for the database files. The default is `/var/lib/postgresql/data` , but if the data volume you're using is a fs mountpoint (like with GCE persistent disks), Postgres `initdb` recommends a subdirectory (for example `/var/lib/postgresql/data/pgdata`) be created to contain the data.

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+ (https://wiki.postgresql.org/wiki/New_in_postgres_10#Renaming_of_.22xlog.22_to_.22wal.22_Globally_.28and_location.2Flsn.29)).

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to 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 -
```

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

Arbitrary `--user` Notes

As of docker-library/postgres#253 (https://github.com/docker-library/postgres/pull/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 postgres
The files belonging to this database system will be owned by user "www-data".
...

$ docker run -it --rm --user 1000:1000 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 (https://cwrap.org/nss_wrapper.html) to "fake" `/etc/passwd` contents for you (see docker-library/postgres#448 (https://github.com/docker-library/postgres/pull/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 postgres
   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 postgres
   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/lil
   $ 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
   ```

How to extend this image

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.

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 (https://github.com/docker-library/postgres/pull/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 (https://github.com/docker-library/postgres/pull/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) (https://github.com/docker-library/postgres/issues/474#issuecomment-416914741) for example).

You can also extend the image 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.

Database Configuration

There are many ways to set PostgreSQL server configuration. For information on what is available to configure, see the postgresql.org docs (https://www.postgresql.org/docs/current/static/runtime-config.html) 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-postg

$ # customize the config

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

- 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 (https://www.postgresql.org/docs/current/static/app-postgres.html) we see that any option available in a `.conf` file can be set via `-c`.

```
$ docker run -d --name some-postgres postgres -c 'shared_buffers=256MB' -c 'max_connecti
```

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/appropriate/docker-postgis (https://github.com/appropriate/docker-postgis/blob/f6d28e4a1871b1f72e1c893ff103f10b6d7cb6e1/10-2.4/Dockerfile) for a concrete example).

When using the Alpine variants, any postgres extension not listed in postgres-contrib (https://www.postgresql.org/docs/10/static/contrib.html) will need to be compiled in your own image (again, see github.com/appropriate/docker-postgis (https://github.com/appropriate/docker-postgis/blob/f6d28e4a1871b1f72e1c893ff103f10b6d7cb6e1/10-2.4/alpine/Dockerfile) for a concrete example).

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` (https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources) for example to `docker run`, or alternatively in `docker-compose` (https://docs.docker.com/compose/compose-file/#shm_size)

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 themselves with the options available, including:

- Let Docker manage the storage of your database data by writing the database files to disk on the host system using its own internal volume management (https://docs.docker.com/engine/tutorials/dockervolumes/#adding-a-data-volume). This is the default and is easy and fairly transparent to the user. The downside is that the files may be hard to locate for tools and applications that run directly on the host system, i.e. outside containers.
- Create a data directory on the host system (outside the container) and mount this to a directory visible from inside the container (https://docs.docker.com/engine/tutorials/dockervolumes/#mount-a-host-directory-as-a-data-volume). This places the database files in a known location on the host system, and makes it easy for tools and applications on the host system to access the files. The downside is that the user needs to make sure that the directory exists, and that e.g. directory permissions and other security mechanisms on the host system are set up correctly.

The Docker documentation is a good starting point for understanding the different storage options and variations, and there are multiple blogs and forum postings that discuss and give advice in this area. We will simply show the basic procedure here for the latter option above:

1. Create a data directory on a suitable volume on your host system, e.g. `/my/own/datadir`.
2. Start your `postgres` container like this:

```
$ docker run --name some-postgres -v /my/own/datadir:/var/lib/postgresql/data -d postgre
```

The `-v /my/own/datadir:/var/lib/postgresql/data` part of the command mounts the `/my/own/datadir` directory from the underlying host system as `/var/lib/postgresql/data` inside the container, where PostgreSQL by default will write its data files.

Image Variants

The `postgres` images come in many flavors, each designed for a specific use case.

`postgres:<version>`

This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as the base to build other images off of.

`postgres:<version>-alpine`

This image is based on the popular Alpine Linux project (http://alpinelinux.org), available in the `alpine` official image (https://hub.docker.com/_/alpine). Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general.

This variant is highly recommended when final image size being as small as possible is desired. The main caveat to note is that it does use musl libc (http://www.musl-libc.org) instead of glibc and friends (http://www.etalabs.net/compare_libcs.html), so certain software might run into issues depending on the depth of their libc requirements. However, most software doesn't have an issue with this, so this variant is usually a very safe choice. See this Hacker News comment thread (https://news.ycombinator.com/item?id=10782897) for more discussion of the issues that might arise and some pro/con comparisons of using Alpine-based images.

To minimize image size, it's uncommon for additional related tools (such as `git` or `bash`) to be included in Alpine-based images. Using this image as a base, add the things you need in your own Dockerfile (see the `alpine` image description (https://hub.docker.com/_/alpine/) for examples of how to install packages if you are unfamiliar).

License

View license information (https://www.postgresql.org/about/licence/) for the software contained in this image.

As with all Docker images, these likely also contain other software which may be under other licenses (such as Bash, etc from the base distribution, along with any direct or indirect dependencies of the primary software being contained).

Some additional license information which was able to be auto-detected might be found in the `repo-info` repository's `postgres/` directory (https://github.com/docker-library/repo-info/tree/master/repos/postgres).

As for any pre-built image usage, it is the image user's responsibility to ensure that any use of this image complies with any relevant licenses for all software contained within.

Docker Pull Command

```
docker pull postgres
```