

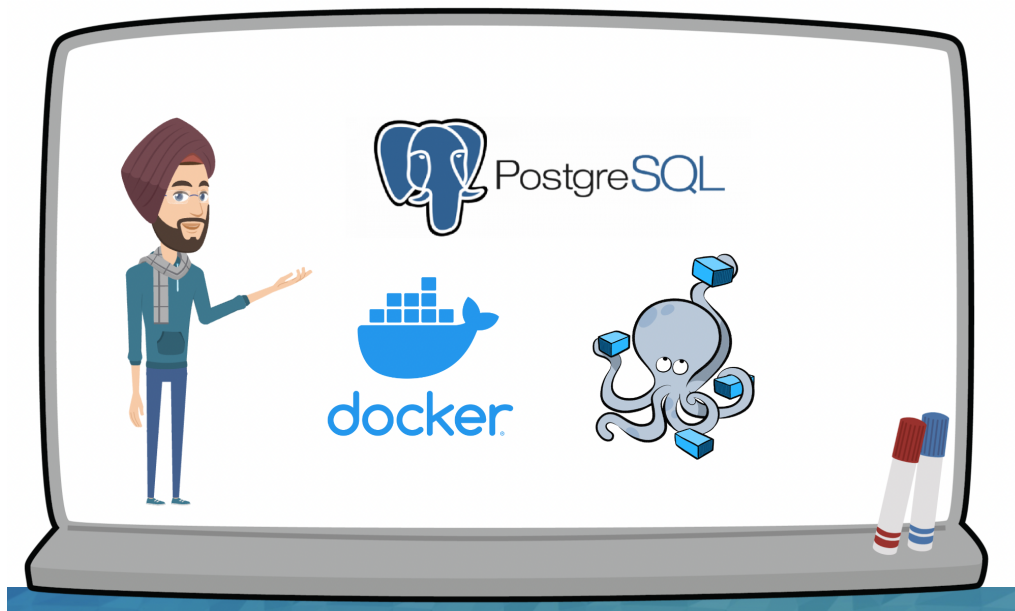
[Follow](#)

Collabnix



# Getting Started with Docker and PostgreSQL

14th February 2023 4 min read



[Join our Discord Server](#)

PostgreSQL is an open-source relational database management system (RDBMS) that is known for its reliability, robustness, and scalability. It is widely used in enterprise applications and offers features such as data integrity, transactions, and SQL support.

## Why is PostgreSQL so damn popular?



# Collabnix



complex databases with high concurrency.

## 2. Open source

PostgreSQL is free and open source, which makes it accessible to anyone and allows for a large community of contributors and users.

## 3. Advanced features

PostgreSQL offers advanced features such as support for advanced data types, JSON, and full-text search.

## 4. Standards compliance

PostgreSQL is known for its high level of standards compliance, which makes it easy to integrate with other systems and applications.

## 5. Extensibility

PostgreSQL is highly extensible, which means it can be easily customized and extended to meet specific needs and requirements.

## PostgreSQL Docker Image

The PostgreSQL Docker image is a pre-configured version of PostgreSQL that can be run inside a Docker container. It allows developers and system administrators to quickly set up and run a PostgreSQL instance without having to install it directly on their local machines.

The official PostgreSQL Docker image is available on the Docker Hub, and it is maintained by the PostgreSQL community. It includes



# Collabnix



configuration options. It can also be used in conjunction with other Docker images and services to create scalable and highly available database clusters.

Using the PostgreSQL Docker image can help to simplify the deployment and management of PostgreSQL instances, and it can make it easier to develop, test, and deploy applications that rely on PostgreSQL as their backend database.

## Steps to install PostgreSQL using Docker Desktop

here are the general steps to install PostgreSQL using Docker Desktop:

### Step 1. Install Docker Desktop

First, you'll need to download and install Docker Desktop on your machine. You can download Docker Desktop from the Docker website, and installation instructions vary depending on your operating system.

### Step 2. Pull the PostgreSQL image

Once Docker Desktop is installed, open a terminal or command prompt and run the following command to download the latest PostgreSQL image from Docker Hub:

 Follow

# Collabnix



After the image is downloaded, run the following command to start a PostgreSQL container:

```
docker run --name my-postgres -e  
POSTGRES_PASSWORD=mysecretpassword -d postgres
```

This command will start a new PostgreSQL container with the name my-postgres and the password mysecretpassword. The -d option runs the container in detached mode, which means it runs in the background.

## Step 4. Access PostgreSQL

You can access the PostgreSQL container using the psql command-line utility or a GUI tool like pgAdmin. To access the container using psql, run the following command:

```
docker exec -it my-postgres psql -U postgres
```

This command connects to the running PostgreSQL container and opens the psql shell as the postgres user.

That's it! You should now have a working PostgreSQL instance running in a Docker container on your machine.

## Setup PostgreSQL Using Customised Dockerfile

To set up PostgreSQL using a customized Dockerfile, you can follow



# Collabnix



```
FROM postgres
ENV POSTGRES_DB mydb
ENV POSTGRES_USER myuser
ENV POSTGRES_PASSWORD mysecretpassword
COPY init.sql /docker-entrypoint-initdb.d/
```

This file starts from the postgres image and sets some environment variables for the database name, user, and password. It also copies an init.sql file to the /docker-entrypoint-initdb.d/ directory, which will be used to initialize the database.

## Create the init.sql file

In the same directory as your Dockerfile, create an init.sql file and add any SQL commands you want to execute when the container starts.

For example, you could create a table:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE
);
```

## Build the image

Run the following command in the same directory as your Dockerfile to build the custom image:



# Collabnix



## Run the container

Finally, run the following command to start a new container based on your custom image:

```
docker run --name my-postgres -p 5432:5432 -d my-postgres
```

This command starts a new container with the name my-postgres, maps the container's port 5432 to the host machine's port 5432, and runs the container in detached mode.

That's it! You should now have a working PostgreSQL instance running in a Docker container based on your custom image. You can connect to the database using a client like psql or any other PostgreSQL client.

## Using Docker Compose

To use PostgreSQL with Docker Compose, you can define a service for PostgreSQL in your docker-compose.yml file. Here are the general steps to do that:

Create a docker-compose.yml file: In the root directory of your project, create a file called docker-compose.yml and add the following content:



# Collabnix



```
restart: always
environment:
  POSTGRES_PASSWORD: mysecretpassword
```

This file defines a service called db that uses the postgres image, sets the POSTGRES\_PASSWORD environment variable, and specifies that the container should always be restarted if it fails.

Start the service: To start the PostgreSQL service, run the following command in the same directory as your docker-compose.yml file:

```
docker-compose up -d
```

This command starts the service in the background (-d option) and creates a new container based on the postgres image.

## Access PostgreSQL

You can access the PostgreSQL container using the psql command-line utility or a GUI tool like pgAdmin. To access the container using psql, run the following command:

```
docker-compose exec db psql -U postgres
```

This command connects to the running PostgreSQL container and opens the psql shell as the postgres user.

That's it! You should now have a working PostgreSQL instance



## Collabnix



# High Availability PostgreSQL with Replication using Docker

Here is an example YAML file for setting up a High Availability  
PostgreSQL database with replication using Docker Compose:





# Collabnix



```
image: postgres:latest
environment:
  POSTGRES_USER: your_username
  POSTGRES_PASSWORD: your_password
volumes:
  - ./master/data:/var/lib/postgresql/data
  - ./master/init.sql:/docker-entrypoint-initdb.d
/init.sql
ports:
  - "5432:5432"
restart: always

slave:
  image: postgres:latest
  environment:
    POSTGRES_USER: your_username
    POSTGRES_PASSWORD: your_password
    POSTGRES_MASTER_HOST: master
    POSTGRES_MASTER_PORT: 5432
  volumes:
    - ./slave/data:/var/lib/postgresql/data
  restart: always
```




In this example, we have two services defined: master and slave.

The master service is the primary database node. It uses the postgres Docker image and sets the environment variables `POSTGRES_USER` and `POSTGRES_PASSWORD` to define the superuser account for the

 Follow


# Collabnix






 Docker image, and sets the POSTGRES\_USER and POSTGRES\_PASSWORD environment variables. We also define the POSTGRES\_MASTER\_HOST, POSTGRES\_MASTER\_PORT,

**Kubernetes  
CRUD  
Operation  
using  
Go on  
Docker  
Desktop**



Ajeet

Raina

May 6, 2023 5

min read 

© Copyright Collabnix Inc

 Follow

**5  
Minutes  
to  
Strapi  
– A  
Headless  
Content  
Management**



Ajeet

Raina

May 5, 2023 6

min read 

**Kubernetes  
for  
Python  
Developers**



Ajeet

Raina

Apr 29, 2023 3

min read 

Built for Collabnix Community, by Community

Docker Captain, ARM Innovator, &amp; Docker Bangalore Community Leader.

Join our Discord Server

- [Docker](#)
- [#Docker](#)
- [#PostgreSQL](#)

 « [Top 10 Security Practices for Enhancing DevOps Security](#)

How to find investors for a college IT startup? »