Запускаем PostgreSQL в Docker: от простого к сложному

[PostgreSQL\*](https://habr.com/ru/hub/postgresql/)[IT-инфраструктура\*](https://habr.com/ru/hub/it-infrastructure/)

Tutorial

Трудно представить современную разработку без контейнеризации. Docker и Kubernetes прочно обосновались на рынке, и, если вы ещё не знакомы с этими технологиями, им явно стоит уделить внимание.

Запуск баз данных и других stateful приложений в контейнере – это тема интересная, но способная вызвать очередной Большой взрыв в комментариях. Оговорюсь сразу, мы не используем в production окружении PostgreSQL в Docker. Но делаем это в локальной разработке и на dev-стендах. Почему? Потому что это чертовски удобно!

В этой статье я хочу рассмотреть типовые способы запуска ванильного (то есть чистого/оригинального) PostgreSQL в контейнере, а также проблемы и их возможные решения, с которыми может столкнуться software engineer. Статья задумывалась как небольшое руководство для новых ребят, приходящих в мою команду, но, уверен, будет полезна и более широкой аудитории.

Прежде чем мы начнём

Вам понадобится Docker на машине, где вы будете проводить эксперименты. Скачайте его с [официального сайта](https://www.docker.com/products/docker-desktop) и установите. Настоятельно рекомендуется использовать актуальную версию.

Все приведенные в этой статье команды и скрипты запускались на версии Docker Desktop 3.5.2 и выше. Их работоспособность на старых версиях Docker не гарантируется.

Работоспособность скриптов на более старых версиях Docker не гарантируется

Так же подразумевается, что у вас есть базовые навыки работы с Docker. Если ещё нет, то для начала крайне желательно ознакомиться со статьями [один](https://habr.com/ru/company/ruvds/blog/438796/) и [два](https://habr.com/ru/post/310460/).

Где взять образы PostgreSQL?

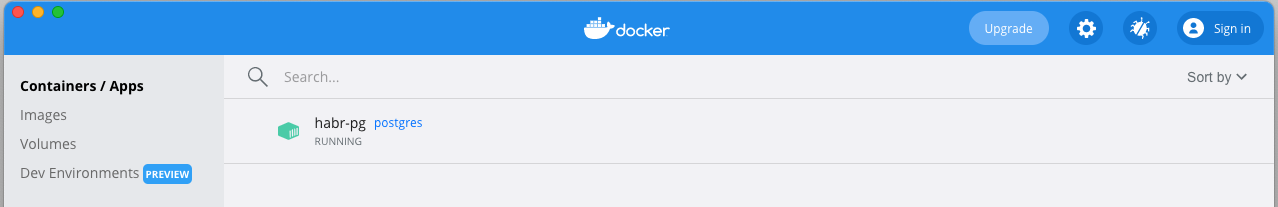
Официальные образы PostgreSQL опубликованы на [Docker Hub](https://hub.docker.com/_/postgres). Там же можно найти базовые инструкции по использованию этих образов. Я буду опираться в том числе и на них.

Hello Postgres

Официальный образ Постгреса очень продвинутый и позволяет настраивать множество параметров. Для быстрого старта большинство из них можно оставить как есть, но вот пароль суперпользователя придётся задать явно:

docker run --name habr-pg -e POSTGRES\_PASSWORD=pgpwd4habr -d postgres

Эта команда запустит нам контейнер PostgreSQL в фоновом (detached) режиме и присвоит ему имя habr-pg:

Контейнер с PostgreSQL, запущенный в Docker

Классно, не правда ли? А что мы можем делать с этой базой данных? К сожалению, на текущий момент не так уж и много. Через интерфейс Docker можно запустить CLI, подключиться к контейнеру и уже оттуда запустить, например, ***psql***:

psql --username=postgres --dbname=postgres

Далее я буду использовать сокращенный вариант этой команды:

psql -U postgres -d postgres

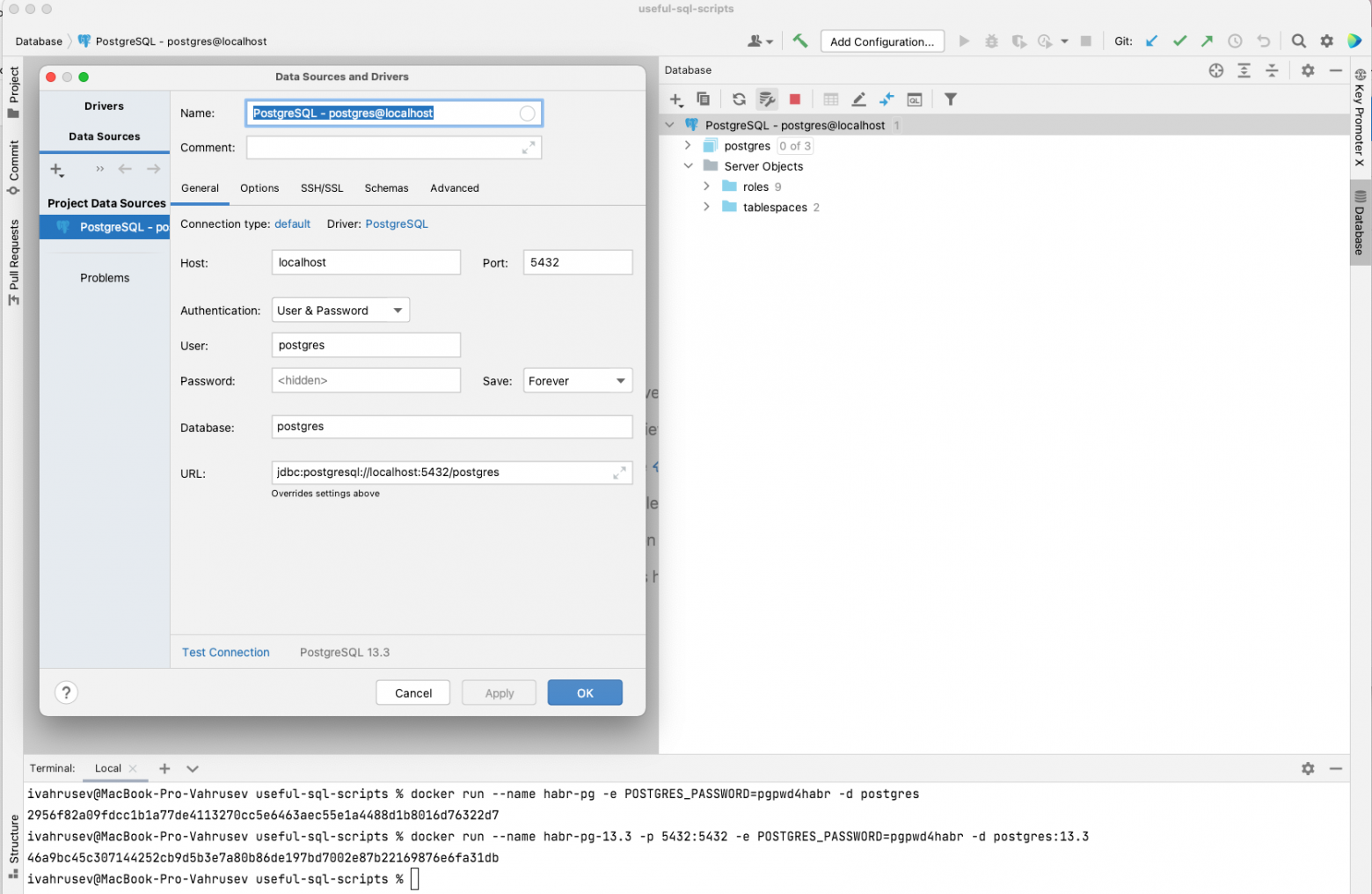
И тут мы сталкиваемся с первой проблемой: что вернёт нам запрос select version();, выполненный в консоли? Мы не указали явным образом версию БД, которую хотим использовать. Давайте это исправим:

docker run --name habr-pg-13.3 -e POSTGRES\_PASSWORD=pgpwd4habr -d postgres:13.3

Теперь вопросов об используемой версии БД не возникает, но работать с ней по-прежнему не очень удобно. Нам нужно сделать эту БД доступной извне, чтобы к ней могли подключаться приложения и IDE. Для этого нужно выставить наружу порт:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_PASSWORD=pgpwd4habr -d postgres:13.3

Отлично! С этого момента к базе данных можно подключиться, например, из IntelliJ IDEA:

Настройка подключения к БД в IntelliJ IDEA

Сейчас мы используем пользователя и базу данных в контейнере, создаваемых по умолчанию, я же предпочитаю указывать их явно. Финальная версия команды для запуска будет иметь вид:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -d postgres:13.3

***psql*** можно запустить так: psql -U habrpguser -d habrdb

И соответствующий compose-файл:

version: "3.9"

services:

postgres:

image: postgres:13.3

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

ports:

- "5432:5432"

Инициализация структуры БД

К текущему моменту мы научились запускать в контейнере необходимую нам версию PostgreSQL, переопределять суперпользователя и создавать базу данных с нужным именем.

Это хорошо, но чистая база данных вряд ли будет сильно полезна. Для работы/тестов/экспериментов нужно наполнить эту базу таблицами и другими объектами. Разумеется, всё это можно сделать вручную, но, согласитесь, гораздо удобнее, когда сразу после запуска вы автоматически получаете полностью готовую БД.

Разработчики официального образа PostgreSQL естественно предусмотрели этот момент и предоставили нам специальную точку входа для инициализации базы данных - **docker-entrypoint-initdb.d**. Любые \*.sql или \*.sh файлы в этом каталоге будут рассматриваться как скрипты для инициализации БД. Здесь есть несколько нюансов:

1. если БД уже была проинициализирована ранее, то никакие изменения к ней применяться не будут;
2. если в каталоге присутствует несколько файлов, то они будут отсортированы по имени с использованием текущей локали (по умолчанию en\_US.utf8).

Инициализацию БД можно запустить через однострочник, но в этом случае требуется указывать абсолютный путь до каталога со скриптами:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -d -v "/absolute/path/to/directory-with-init-scripts":/docker-entrypoint-initdb.d postgres:13.3

Например, на моей машине это выглядит так:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -d -v "/Users/ivahrusev/src/useful-sql-scripts/running\_pg\_in\_docker/2. Init Database":/docker-entrypoint-initdb.d postgres:13.3

В качестве обходного варианта можно использовать макрос, на лету определяя рабочую директорию, и запускать команду из каталога со скриптами:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -d -v "$(pwd)":/docker-entrypoint-initdb.d postgres:13.3

Использование docker-compose файла в этом случае более удобно и позволяет указывать относительные пути:

version: "3.9"

services:

  postgres:

    image: postgres:13.3

    environment:

      POSTGRES\_DB: "habrdb"

      POSTGRES\_USER: "habrpguser"

      POSTGRES\_PASSWORD: "pgpwd4habr"

    volumes:

      - .:/docker-entrypoint-initdb.d

    ports:

      - "5432:5432"

Здесь хотелось бы акцентировать ваше внимание на одной простой вещи, о которой уже говорил в [предыдущей статье](https://habr.com/ru/post/501454/): при создании миграций БД для ваших приложений отдавайте предпочтение чистому (plain) SQL. В этом случае их можно будет переиспользовать с минимальными затратами.

А куда сохраняются мои данные?

Базы данных – это в первую очередь история про персистентность. И,.. Хьюстон, кажется у нас проблема… К настоящему моменту мы никак не управляем долговременным хранением нашей базы данных. Эту задачу целиком на себя берёт Docker, автоматически создавая volume для контейнера с БД. Есть целый ворох причин, почему это плохо, начиная от банальной невозможности просматривать содержимое volume’ов в бесплатной версии Docker Desktop и заканчивая лимитами дискового пространства.

Разумеется, хорошей практикой является полностью ручное управление физическим размещением создаваемых баз данных. Для этого нам нужно подмонтировать соответствующий каталог (куда будут сохраняться данные) в контейнер и при необходимости переопределить переменную окружения **PGDATA**:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -e PGDATA=/var/lib/postgresql/data/pgdata -d -v "/absolute/path/to/directory-with-data":/var/lib/postgresql/data -v "/absolute/path/to/directory-with-init-scripts":/docker-entrypoint-initdb.d postgres:13.3

Вариант с макросом, использующий для инициализации БД скрипты из предыдущего раздела:

docker run --name habr-pg-13.3 -p 5432:5432 -e POSTGRES\_USER=habrpguser -e POSTGRES\_PASSWORD=pgpwd4habr -e POSTGRES\_DB=habrdb -e PGDATA=/var/lib/postgresql/data/pgdata -d -v "$(pwd)":/var/lib/postgresql/data -v "$(pwd)/../2. Init Database":/docker-entrypoint-initdb.d postgres:13.3

С однострочниками на этом закончим. Все дальнейшие шаги будем осуществлять только через compose-файл:

version: "3.9"

services:

postgres:

image: postgres:13.3

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ../2. Init Database:/docker-entrypoint-initdb.d

- .:/var/lib/postgresql/data

ports:

- "5432:5432"

При запуске этого скрипта рядом с ним создастся директория ***pgdata***, где будут располагаться файлы БД.

Healthcheck? Нет, не слышал…

Проверка состояния/работоспособности – ***healthcheck*** – вполне устоявшийся архитектурный шаблон, который вы должны взять на вооружение для всех ваших приложений. База данных, запускаемая в контейнере, не является исключением.

Основная задача healthcheck’а – как можно скорее уведомить среду, управляющую контейнером, о том, что с контейнером что-то не так. И самая простая стратегия решения проблемы – перезапуск контейнера.

Так же стоит сразу позаботиться об ограничении ресурсов для контейнера с БД. Для экспериментов и локального запуска вполне подойдёт секция ***resources*** (флаг --compatibility [больше не требуется](https://github.com/docker/compose/issues/4513)).

Healthcheck для PostgreSQL обычно основывается на использовании утилиты [pg\_isready](https://postgrespro.ru/docs/postgresql/13/app-pg-isready) как показано ниже:

version: "3.9"

services:

postgres:

image: postgres:13.3

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ../2. Init Database:/docker-entrypoint-initdb.d

- .:/var/lib/postgresql/data

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U habrpguser -d habrdb"]

interval: 10s

timeout: 5s

retries: 5

start\_period: 10s

restart: unless-stopped

deploy:

resources:

limits:

cpus: '1'

memory: 4G

А если хочу изменить параметры БД?

Для администраторов PostgreSQL не секрет, что конфигурация СУБД из коробки далека от идеальной и не очень подходит для эксплуатации. Я немного рассказывал об этом в своей [статье про pg-index-health](https://habr.com/ru/post/490824/): ряд параметров нужно изменить в обязательном порядке, так как они влияют на производительность. Так же существует большое количество расширений для Постгреса, которые сделают эксплуатацию БД более удобной, наблюдаемой и управляемой. Одно из таких расширений - ***pg\_stat\_statements*** (кстати, оно пригодится нам позднее для мониторинга БД).

Ванильный образ PostgreSQL позволяет тюнить параметры и добавлять расширения на старте контейнера БД:

version: "3.9"

services:

postgres:

image: postgres:13.3

command:

- "postgres"

- "-c"

- "max\_connections=50"

- "-c"

- "shared\_buffers=1GB"

- "-c"

- "effective\_cache\_size=4GB"

- "-c"

- "work\_mem=16MB"

- "-c"

- "maintenance\_work\_mem=512MB"

- "-c"

- "random\_page\_cost=1.1"

- "-c"

- "temp\_file\_limit=10GB"

- "-c"

- "log\_min\_duration\_statement=200ms"

- "-c"

- "idle\_in\_transaction\_session\_timeout=10s"

- "-c"

- "lock\_timeout=1s"

- "-c"

- "statement\_timeout=60s"

- "-c"

- "shared\_preload\_libraries=pg\_stat\_statements"

- "-c"

- "pg\_stat\_statements.max=10000"

- "-c"

- "pg\_stat\_statements.track=all"

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ../2. Init Database:/docker-entrypoint-initdb.d

- .:/var/lib/postgresql/data

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U habrpguser -d habrdb"]

interval: 10s

timeout: 5s

retries: 5

start\_period: 10s

restart: unless-stopped

deploy:

resources:

limits:

cpus: '1'

memory: 4G

Разумеется, можно указать свой ***postgresql.conf***. Оставлю это в качестве домашнего задания.

Посмотреть список установленных расширений можно с помощью запроса select \* from pg\_extension;.

А команда ***show*** позволит узнать текущее значение того или иного параметра, например: show random\_page\_cost;.

Не люблю консоль; дайте мне человеческий UI!

Далеко не все пользователи любят работать с БД из командной строки. Очень многие предпочитают использовать для этого продвинутый графический интерфейс, например pgAdmin.

Запустить ещё один контейнер, в котором будет бежать GUI, не сложно, но для удобной коммуникации с БД их лучше объединить в одну сеть:

version: "3.9"

services:

postgres:

container\_name: postgres\_container

image: postgres:13.3

command:

- "postgres"

- "-c"

- "max\_connections=50"

- "-c"

- "shared\_buffers=1GB"

- "-c"

- "effective\_cache\_size=4GB"

- "-c"

- "work\_mem=16MB"

- "-c"

- "maintenance\_work\_mem=512MB"

- "-c"

- "random\_page\_cost=1.1"

- "-c"

- "temp\_file\_limit=10GB"

- "-c"

- "log\_min\_duration\_statement=200ms"

- "-c"

- "idle\_in\_transaction\_session\_timeout=10s"

- "-c"

- "lock\_timeout=1s"

- "-c"

- "statement\_timeout=60s"

- "-c"

- "shared\_preload\_libraries=pg\_stat\_statements"

- "-c"

- "pg\_stat\_statements.max=10000"

- "-c"

- "pg\_stat\_statements.track=all"

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ../2. Init Database:/docker-entrypoint-initdb.d

- .:/var/lib/postgresql/data

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U habrpguser -d habrdb"]

interval: 10s

timeout: 5s

retries: 5

start\_period: 10s

restart: unless-stopped

deploy:

resources:

limits:

cpus: '1'

memory: 4G

networks:

- postgres

pgadmin:

container\_name: pgadmin\_container

image: dpage/pgadmin4:5.7

environment:

PGADMIN\_DEFAULT\_EMAIL: "habrpguser@habr.com"

PGADMIN\_DEFAULT\_PASSWORD: "pgadminpwd4habr"

PGADMIN\_CONFIG\_SERVER\_MODE: "False"

volumes:

- ./pgadmin:/var/lib/pgadmin

ports:

- "5050:80"

restart: unless-stopped

deploy:

resources:

limits:

cpus: '0.5'

memory: 1G

networks:

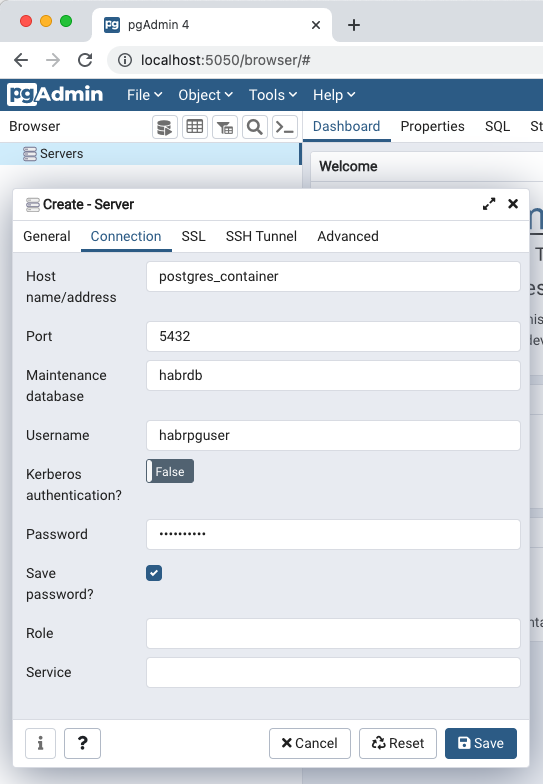
- postgres

networks:

postgres:

driver: bridge

pgAdmin стартует на порту [5050](http://localhost:5050/): перейдя на нужный адрес, можно будет настроить подключение к БД.

К БД можно подключиться как по имени контейнера, так и по имени сервиса

А как насчёт мониторинга?

В современной разработке любой микросервис или инфраструктурный компонент должен быть поставлен на мониторинг, то есть непрерывно отдавать метрики - ключевые показатели, позволяющие определить, как ведёт себя система в данный момент времени.

PostgreSQL не имеет встроенной интеграции с системами мониторинга наподобие Prometheus (или Zabbix). Вместо этого он полагается на использование внешних агентов - экспортеров. Мы у себя активно используем [postgres\_exporter](https://github.com/prometheus-community/postgres_exporter), позволяющий добавлять свои собственные sql-запросы и кастомные метрики на их основе:

version: "3.9"

services:

postgres:

container\_name: postgres\_container

image: postgres:13.3

command:

- "postgres"

- "-c"

- "max\_connections=50"

- "-c"

- "shared\_buffers=1GB"

- "-c"

- "effective\_cache\_size=4GB"

- "-c"

- "work\_mem=16MB"

- "-c"

- "maintenance\_work\_mem=512MB"

- "-c"

- "random\_page\_cost=1.1"

- "-c"

- "temp\_file\_limit=10GB"

- "-c"

- "log\_min\_duration\_statement=200ms"

- "-c"

- "idle\_in\_transaction\_session\_timeout=10s"

- "-c"

- "lock\_timeout=1s"

- "-c"

- "statement\_timeout=60s"

- "-c"

- "shared\_preload\_libraries=pg\_stat\_statements"

- "-c"

- "pg\_stat\_statements.max=10000"

- "-c"

- "pg\_stat\_statements.track=all"

environment:

POSTGRES\_DB: "habrdb"

POSTGRES\_USER: "habrpguser"

POSTGRES\_PASSWORD: "pgpwd4habr"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ../2. Init Database:/docker-entrypoint-initdb.d

- .:/var/lib/postgresql/data

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U habrpguser -d habrdb"]

interval: 10s

timeout: 5s

retries: 5

start\_period: 10s

restart: unless-stopped

deploy:

resources:

limits:

cpus: '1'

memory: 4G

networks:

- postgres

pgadmin:

container\_name: pgadmin\_container

image: dpage/pgadmin4:5.7

environment:

PGADMIN\_DEFAULT\_EMAIL: "habrpguser@habr.com"

PGADMIN\_DEFAULT\_PASSWORD: "pgadminpwd4habr"

PGADMIN\_CONFIG\_SERVER\_MODE: "False"

volumes:

- ./pgadmin:/var/lib/pgadmin

ports:

- "5050:80"

restart: unless-stopped

deploy:

resources:

limits:

cpus: '0.5'

memory: 1G

networks:

- postgres

postgres\_exporter:

container\_name: exporter\_container

image: prometheuscommunity/postgres-exporter:v0.10.0

environment:

DATA\_SOURCE\_URI: "postgres:5432/habrdb?sslmode=disable"

DATA\_SOURCE\_USER: "habrpguser"

DATA\_SOURCE\_PASS: "pgpwd4habr"

PG\_EXPORTER\_EXTEND\_QUERY\_PATH: "/etc/postgres\_exporter/queries.yaml"

volumes:

- ./queries.yaml:/etc/postgres\_exporter/queries.yaml:ro

ports:

- "9187:9187"

restart: unless-stopped

deploy:

resources:

limits:

cpus: '0.2'

memory: 500M

networks:

- postgres

networks:

postgres:

driver: bridge

После запуска скрипта экспортер будет доступен на порту [9187](http://localhost:9187/metrics) и отдавать метрики в формате Prometheus:

...

# HELP pg\_stat\_user\_tables\_n\_tup\_upd Number of rows updated

# TYPE pg\_stat\_user\_tables\_n\_tup\_upd counter

pg\_stat\_user\_tables\_n\_tup\_upd{datname="habrdb",relname="first\_table",schemaname="public",server="postgres:5432"} 0

pg\_stat\_user\_tables\_n\_tup\_upd{datname="habrdb",relname="second\_table",schemaname="public",server="postgres:5432"} 0

# HELP pg\_stat\_user\_tables\_seq\_scan Number of sequential scans initiated on this table

# TYPE pg\_stat\_user\_tables\_seq\_scan counter

pg\_stat\_user\_tables\_seq\_scan{datname="habrdb",relname="first\_table",schemaname="public",server="postgres:5432"} 1

pg\_stat\_user\_tables\_seq\_scan{datname="habrdb",relname="second\_table",schemaname="public",server="postgres:5432"} 1

# HELP pg\_stat\_user\_tables\_seq\_tup\_read Number of live rows fetched by sequential scans

# TYPE pg\_stat\_user\_tables\_seq\_tup\_read counter

pg\_stat\_user\_tables\_seq\_tup\_read{datname="habrdb",relname="first\_table",schemaname="public",server="postgres:5432"} 0

pg\_stat\_user\_tables\_seq\_tup\_read{datname="habrdb",relname="second\_table",schemaname="public",server="postgres:5432"} 0

# HELP pg\_stat\_user\_tables\_vacuum\_count Number of times this table has been manually vacuumed (not counting VACUUM FULL)

# TYPE pg\_stat\_user\_tables\_vacuum\_count counter

pg\_stat\_user\_tables\_vacuum\_count{datname="habrdb",relname="first\_table",schemaname="public",server="postgres:5432"} 0

pg\_stat\_user\_tables\_vacuum\_count{datname="habrdb",relname="second\_table",schemaname="public",server="postgres:5432"} 0

# HELP pg\_static Version string as reported by postgres

# TYPE pg\_static untyped

pg\_static{server="postgres:5432",short\_version="13.3.0",version="PostgreSQL 13.3 (Debian 13.3-1.pgdg100+1) on x86\_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit"} 1

...

Разумеется, для полноценной постановки на мониторинг нужно ещё поднять сам Prometheus + Grafana, а так же загрузить подходящий dashboard, но это уже выходит за рамки данной статьи. Более того, если ваша служба информационной безопасности исповедует Zero Trust, то экспортер придётся прикрыть с помощью nginx и настроить mTLS...

В качестве заключения

На этом у меня всё. Приведенной выше конфигурации более чем достаточно для развёртывания БД PostgreSQL в Docker-контейнере на стенде разработки или локально.

Все приведённые в статье команды и ***docker-compose*** файлы также доступны на [GitHub](https://github.com/mfvanek/useful-sql-scripts/tree/master/running_pg_in_docker).