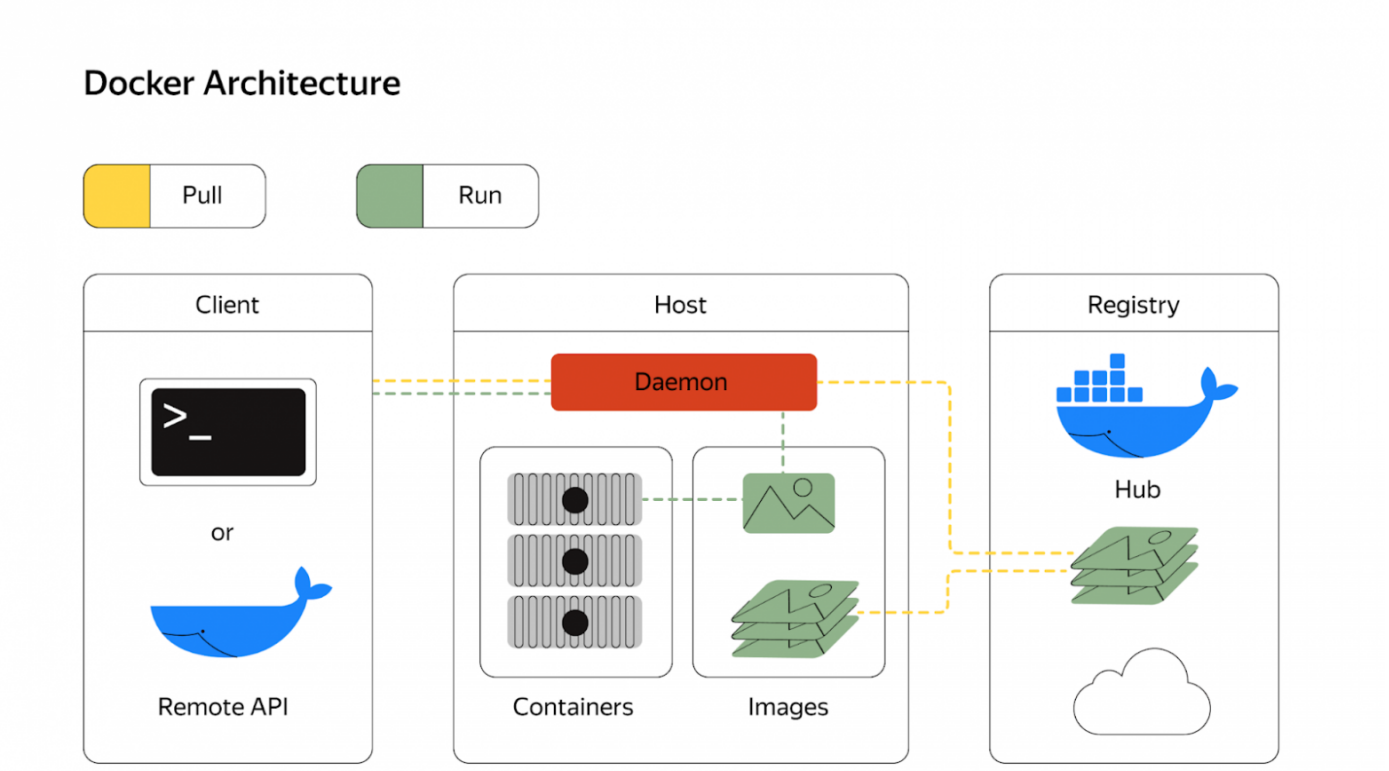
mtm

DOCKER



Основным элементом являются Docker-**образы** (**Images**) — это заранее подготовленные шаблоны, которые включают в себя все необходимое для корректной и быстрой работы приложения, включая библиотеки, сам код и все зависимости, а также произведенные настройки. Эти образы используются для создания контейнеров — изолированных исполняемых единиц, в которых запускается приложение.

Реестр Docker (Docker **Registry**) представляет собой удалённую платформу, используемую для хранения образов Docker. В ходе работы с Docker образы отправляют в реестр и загружают из него. Подобный реестр может быть организован тем, кто пользуется Docker. Кроме того, поставщики облачных услуг могут поддерживать и собственные реестры. Например, это касается AWS и Google Cloud.

Хаб Docker (Docker **Hub**) — это самый крупный реестр образов Docker. Кроме того, именно этот реестр используется при работе с Docker по умолчанию. Пользоваться хабом Docker можно бесплатно.

Контейнеры Docker по своей природе неизменяемы. Это означает, что при перезапуске контейнера все данные, сохраненные в нем, **сотрутся**. Но сохранить эти данные можно. Для этого Docker предоставляет возможность подключения к контейнеру локальных директорий.

b4d699d2189c --------> id контейнера

`docker exec -it b4d699d2189c bash` ------> команда для входа в контейнер
`docker start b4d699d2189c` ----------> Запуск сущ-ся контейнера
`docker ps` --------> Посмотреть список запущеных контейнеров
`docker ps -a` ---------> Посмотреть список всех контейнеров
.
.

```
root@nashimudes:~# docker cp /media/nashimudes/New\ Volume/Downloads/demo-small/demo-small-20170815.sql test_postgres1:/
```

------> Копируем файл demo-small-20170815.sql в контейнер test_postgres1
.
.

```
root@nashimudes:~# docker run --name small_postgres -p 5411:5432 -e POSTGRES_PASSWORD=1234 -d postgres
```

------> Создание контейнера с именем small_postgres с внешним портом 5411 и внутреним 5432 с паролем 1234 образ которого являетя postgres
.
.

```
root@nashimudes:~# docker cp small_postgres:"/backup/mydb.sql" "/home/nashimudes/Desktop"
```

--------> Копируем файл mydb.sql из контейнера в локальную машину
.
.

```
root@283018d372df:/# pg_dump -U postgres demo >  /backup/mydb.sql
```

--------> Создание backup-а базы demo в папку /backup под именем postgres.
283018d372df----> контейнер внутри коготоро мы находимся
.
.

```
root@283018d372df:/test# pg_dump -d demo -U postgres > /test/demo_back.sql
```

-----> -d demo указываем, что нужно скопировать бд demo
.
.

```
version: '2'
services:
```

```
  db:
    image: postgres
    container_name: test_cont1
    ports:
      - 5467:5432
    environment:
              POSTGRES_PASSWORD: 1234
              POSTGRES_USER: postgres
              POSTGRES_DB: postgres
    volumes:
      - /home/nashimudes/Desktop/test_cont1:/var/lib/postgresql/data
```

-----> Create compose.yml

.

.

```
version: '2'

services:
      postgresql_01:
        image: postgres
        container_name: postgres_master
        restart: always
        networks:
          - app-network
        ports:
          - 5411:5432
        environment:
             POSTGRES_PASSWORD: 1234
             POSTGRES_USER: postgres
             POSTGRES_DB: postgres
        volumes:
            - /home/nashimudes/Desktop/test_cont1:/var/lib/postgresql/data

      postgresql_02:
        image: postgres
        container_name: postgres_slave
        restart: always
        networks:
          - app-network
        ports:
          - 5422:5432
        environment:
```

```
            POSTGRES_PASSWORD: 1234
            POSTGRES_USER: postgres
            POSTGRES_DB: postgres
        volumes:
            - /home/nashimudes/Desktop/test_cont2:/var/lib/postgresql/data

networks:
    app-network:
        driver: bridge
```

.

.

version: '2.1'

```
services:
      postgresql_01:
          image: postgres
          container_name: postgres_01
          restart: always
          ports:
            - 5411:5432
          environment:
              POSTGRES_PASSWORD: 1234
              POSTGRES_USER: postgres
          volumes:
            - /home/nashimudes/Desktop/test_01:/var/lib/postgresql/data

      postgresql_02:
          image: postgres
          container_name: postgres_02
          restart: always
          ports:
            - 5422:5432
          environment:
              POSTGRES_PASSWORD: 1234
              POSTGRES_USER: postgres
          volumes:
            - /home/nashimudes/Desktop/test_02:/var/lib/postgresql/data
```

.

.

```
root@nashimudes:~# docker inspect postgres_02
```

-----> Find IP of a PostgreSQL database running as a docker container

.

.

```
root@ef9c8a944a6c:/# pg_basebackup  --host=172.24.0.2 --username=repl_nassim --pgdata=/mount_folders/dump_test1/pgdata2 --wal-method=stream --write-recovery-conf
```

-----> Make replica database

.

.

su - postgres -c "pg_basebackup --host=172.25.0.2 --username=repluser --pgdata=/mount_folders/rep_test2/postgresql2 --wal-method=stream --write-recovery-conf"

.

.

```
sudo docker run --name test_postgres1 -p 5455:5432 -v /home/nashimudes/Desktop/test:/var/lib/postgresql/data -e POSTGRES_PASSWORD=1234 postgres
```

-----> Монтируем папку test которая находится в домашней директории, в контейнер где запущен postgresql

.

.

.

.

.

.

.

POSTGRES

```
root@283018d372df:/# psql -U postgres < demo-small-20170815.sql
```

------> Распокавываем файл demo-small-20179815.sql внутрь бд под пользователем postgres

.

.

`pg_config --version` ---> проверка версии postgresql

.

.

linux

root

abdu0102A

.

.

`sudo -u postgr es psql template1`

-----> Эта команда подключится к PostgreSQL базе данных template1 как пользователь postgres.

.

.

```
test_db1=# ALTER GROUP readonly_users ADD USER test_user1, test_user2;
```

----> Добавить поль-лей test_user1, test_user2 в группу readonly_users.

.

.

.

.

```
test_db1=# \du
```

-----> Посмотреть список пользователей

.

.

`CTRL + SHIFT + L` -----> clear psql terminal

.

.

```
test_db1=# SELECT usename FROM pg_catalog.pg_user;
```

-----> Посмотреть список пользователей

.

.

```
test_db1=# ALTER GROUP readonly_users ADD USER test_user1, test_user2;
```

-----> Добавляем в группу readonly_users польз-лей test_user1, test_user2

.

.

```
SELECT r.rolname as username,r1.rolname as "role"
FROM pg_catalog.pg_roles r JOIN pg_catalog.pg_auth_members m
ON (m.member = r.oid)
JOIN pg_roles r1 ON (m.roleid=r1.oid)
WHERE r.rolcanlogin
ORDER BY 1;
```

.

.

```
select * from current_user, session_user;
```

.
.

```
SELECT current_user, session_user;
```

-----> Посмтореть текущего поль-ля и кто создал эту сессию

.
.

```
SELECT * FROM information_schema.role_table_grants WHERE grantee =
'readonly_user';
```

------> Посмотреть список привелегий пользователя
'readonly_user'

.
.

```
REVOKE SELECT ON bookings.boarding_passes, bookings.flights,
bookings.airports_data FROM readonly_user;
```

------> убрать привилегии у readonly_user

.
.

```
GRANT INSERT, DELETE ON bookings.tickets TO test_user1;
```

----------> Даём привилегий на добавление
и удаление в таблице tickets для поль-ля
test_user1.

.
.

```
demo=# CREATE USER nassim WITH PASSWORD '1234';
```

------> Создание поль-ля с именем nassim и паролем 1234
demo-->название базы в которой мы находимся

.
.

`psql -U test_user1 demo;`
-------> подключится под пользователем test_user1 к бд demo

.
.

To clear the screen using the Ctrl + L keyboard shortcut, simply press Ctrl + L while you are in psql.

Sequences в PostgreSQL - генераторы уникальных последовательностей чисел, использовать можно аналогичным auto_increment полям образом в MySQL. Т.е. создаете sequence для таблицы, при вставке в нее новой записи в поле, которое должно быть уникальным (primary key) пишете значение, которое выдает sequence, созданный для этой таблицы.

Кластеризация, если говорить простыми словами, - это разбиение на группы, по определенным критериям. Кластер это группа объектов.

Попытаюсь объяснить на пальцах
В большинстве случаев основная нагрузка идет на чтение БД, часто бывает, что одна машина не справляется с существующей нагрузкой, тогда поднимают кластер — запускают СУБД на нескольких машинах, одна из них объявляется мастером, остальные репликами
Мастер занимается только записью и распространением готовых изменений по репликам
А читаем мы только из реплик, балансируя нагрузку между ними, тем самым снижая нагрузку на каждую из них и уменьшая время отклика.

Схема в POSTGRES
Схема в PostgreSQL - это логическая структура, которая представляет собой набор таблиц, индексов, представлений, функций и других объектов базы данных, организованных в единую группу. Схема помогает организовать данные в базе данных, делая их более удобными для управления и обеспечивая логическую разделяемость объектов базы данных.

С помощью схемы можно создавать отдельные пространства имен для объектов базы данных, что позволяет избежать конфликтов имен и обеспечивает более удобную организацию базы данных. Каждая таблица, индекс или другой объект базы данных может принадлежать к определенной схеме, что позволяет структурировать базу данных и упрощает ее обслуживание.

Схемы также играют важную роль при управлении правами доступа к данным, так как позволяют назначать различные права на доступ к объектам базы данных в зависимости от их принадлежности к определенной схеме.

Таким образом, схема в PostgreSQL является важным средством организации данных и обеспечивает более эффективное управление базой данных.

База данных (БД) — это набор данных, который как-то структурирован. Например, можно взять сто картинок с котами и отсортировать их по цвету или по позе.
Система управления базами данных (СУБД) — это набор инструментов, которые позволяют удобно управлять базами данных: удалять, добавлять, фильтровать и находить элементы, менять их структуру и создавать резервные копии.

Популярные **СУБД** — PostgreSQL, MySQL, Microsoft SQL Server, SQLite, MongoDB, Redis, Oracle Database.

База данных — это набор элементов, которые сгруппированы по определённым правилам. Они бывают реляционными, графовыми и иерархическими.

СУБД — это инструменты, которые помогают управлять базами данных. Например, с их помощью можно удалять, изменять и находить элементы.

Базы данных отличаются от **СУБД** тем, что сами по себе представляют лишь файл на компьютере. Базы данных не умеют ничего делать с этими данными — только хранить. А вот СУБД уже предоставляют возможности по манипуляции ими.

Репликация БД в PostgreSQL

Репликация базы данных - это процесс создания точной копии базы данных и ее содержимого на другом сервере или в другом месте. Это делается для повышения отказоустойчивости и производительности системы, так как реплика (копия) может использоваться для чтения данных, тогда как основная база данных продолжает обслуживать запись и другие операции. Если основная база данных выходит из строя, реплика может взять на себя работу, чтобы система продолжала работать без перерывов.

Репликация базы данных — это процесс создания и обслуживания нескольких копий базы данных в разных местах или на разных серверах. Основная цель репликации — повысить доступность данных, распределить рабочую нагрузку и повысить общую производительность системы базы данных. Он также обеспечивает избыточность и отказоустойчивость, гарантируя, что данные реплицируются на несколько узлов синхронно или асинхронно.

И завершая теоретическую часть этой статьи хотелось бы прояснить различия между понятиями резервирование и резервное копирование. Готовя материал к предыдущей статье я сталкивался с тем, что на некоторых ресурсах бэкап тоже называют термином резервирование. Это неверно, резервирование это обеспечение отказоустойчивости работы ресурсов в режиме реального времени. Резервирование не допускает простоев системы, по крайней мере таких масштабных как при восстановлении из бэкапов. Так что резервирование это репликации и кластеры, а резервное копирование это бэкап.

Репликация на серверах СУБД PostgreSQL бывает двух видов: **физическая** и **логическая**. При физической репликации у нас на сервер реплики передается поток WAL записей. Одним из основных достоинств физической репликации является простота в конфигурировании и использовании, так как используется простое побайтовое копирование с одного сервера на другой. Также из-за своей простоты, физическая репликация потребляет меньше ресурсов.

Но у физических репликаций есть и свои недостатки. По аналогии с физическим бэкапом здесь также требуются одинаковые версии PostgreSQL и операционной системы.

При этом, также должны быть идентичны в том числе и аппаратные компоненты, такие как архитектура процессора. Также при физических бэкапах возможна репликация только всего кластера, на подчиненном инстансе нельзя создать никакую отдельную таблицу, даже временную. Полная идентичность основному серверу.

В зависимости от архитектуры самой БД возможна существенная нагрузка на инфраструктуру, так как нужно передавать все изменения в файлах данных полностью.

[(https://habr.com/ru/companies/otus/articles/710956/)]

LINUX

`$ cp опции /путь/к/файлу/источнику /путь/к/директории/назначения`

.

.

How Do I Use VI?
To use vi: vi filename
To exit and save changes: :wq!
To exit without saving changes: :q!
To save changes without quitting: :w!
To get out of insert mode: [esc]

.

.

Команда mv позволяет изменить имя файла, не перемещая его в другой каталог.

`mv appendix apndx.a`

.

.

```
root@nashimudes:/home/nashimudes# chown nashimudes:nashimudes ./Desktop/
```

```
root@nashimudes:/home/nashimudes# chown -Rv nashimudes:nashimudes ./Desktop/
```

.

.

`chmod 777 foldername` will give read, write, and execute permissions for everyone.

.

.

Для удаления папки и ее содержимого (рекурсивное удаление) воспользуйтесь командной rm с опцией -R:

```
rm -R название-папки
```

.

.

```
root@nashimudes:/docker_composes# cd ..
```

-----> пойти на один шаг назад

.

.

```
sudo apt-get update
sudo apt-get install vim
```

-----> install vim for editing files

.

.