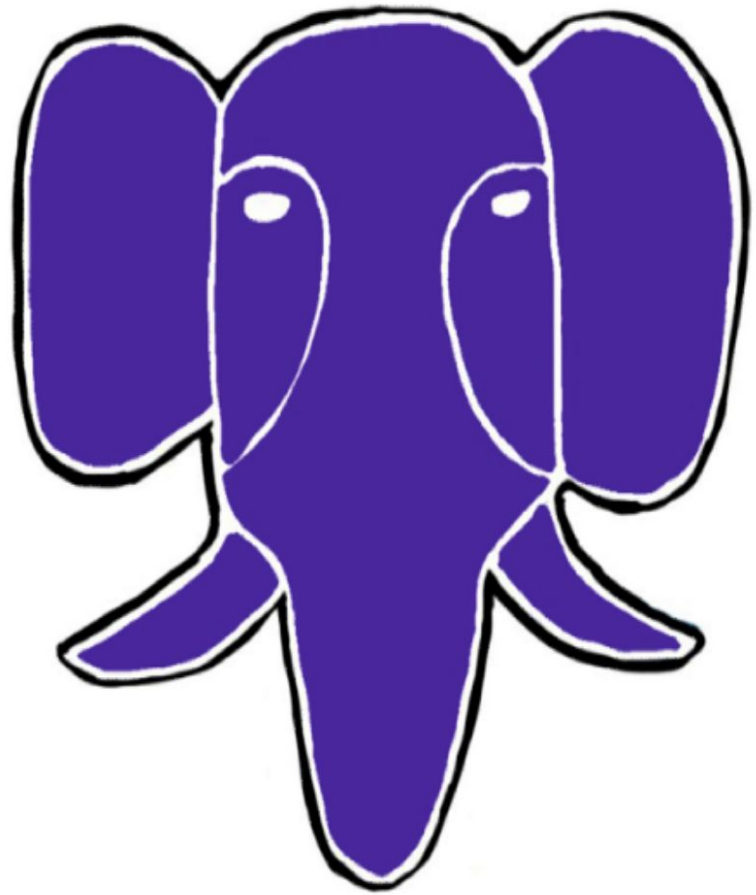
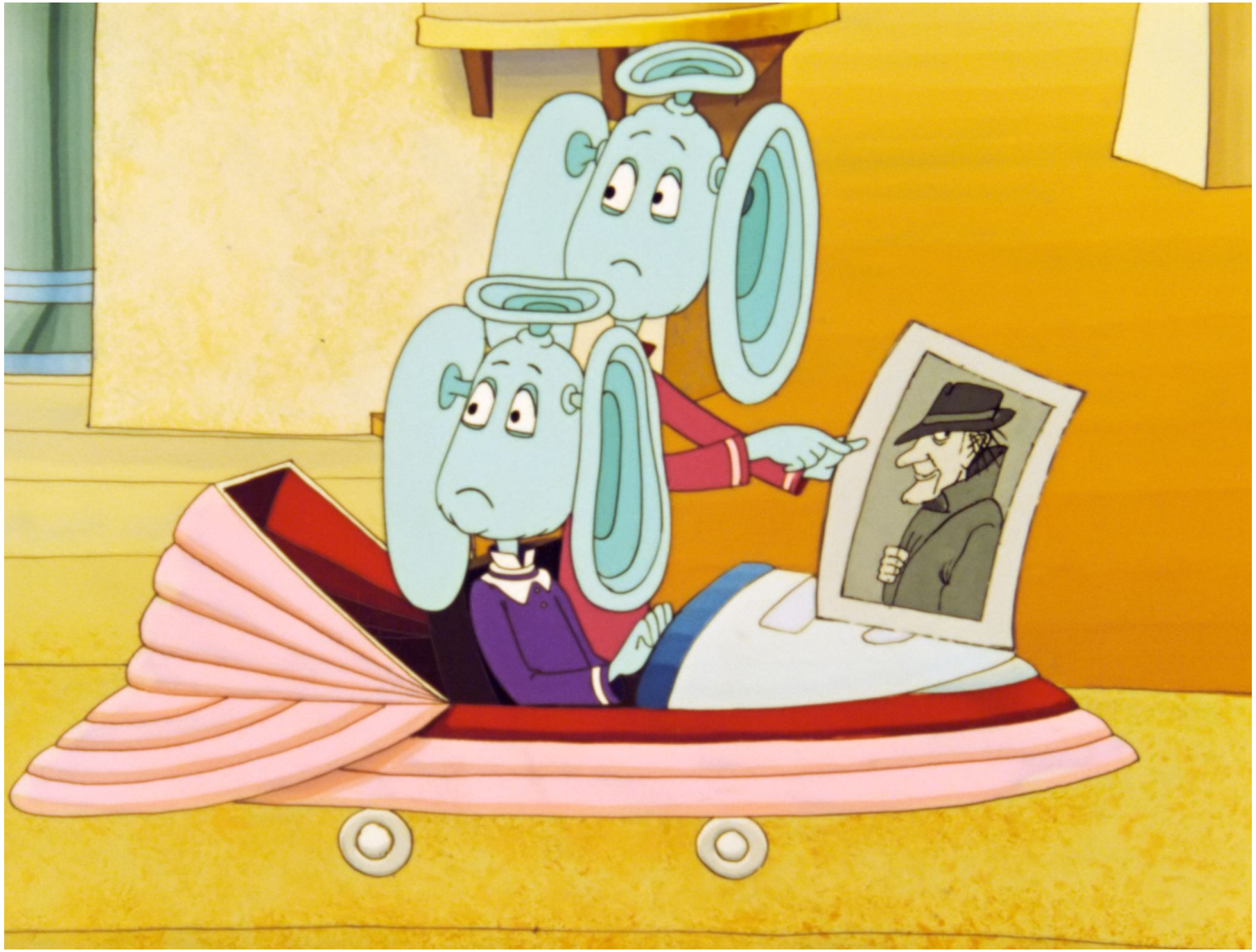
Евгений Аристов

Архитектура PostgreSQL



https://aristov.tech



Аристов
Евгений
Николаевич

Founder & CEO aristov.tech
25 лет занимаюсь разработкой БД и ПО

Архитектор высоконагруженных баз данных и инфраструктуры

Спроектировал и разработал более ста проектов для финансового сектора, сетевых магазинов, фитнес-центров, отелей.

Сейчас решаю актуальные для бизнеса задачи: аудит и оптимизация БД и инфраструктуры, миграция на PostgreSQL, обучение сотрудников.

Автор более 10 практических курсов по PostgreSQL, MySQL, Mongo и др..

Автор книг по PostgreSQL. Новинка **PostgreSQL 16: лучшие практики оптимизации**

Цель курса

В сжатые сроки обучить работе с PostgreSQL.

Курс практико ориентирован с большим количеством ссылок на доп. изучение.

Моя задача дать вам максимум материала и практических заданий - ваша задача тренироваться.

Часть 1 (март) 7 крутых лекций + 7 домашних заданий
Часть 2 (апрель-май) 8 крутых лекций + 6 домашних заданий

https://aristov.tech

Правила вебинара

Задаем вопрос в чат

Вопросы вижу, отвечу в момент логической паузы

Если есть вопрос голосом - поставьте знак ? в чат

Если остались вопросы, можно их задать на следующем занятии

https://aristov.tech

Маршрут вебинара

- Основные концепции PostgreSQL - плюсы и минусы, OLTP vs OLAP
- Новые функции и улучшения в последних версиях PostgreSQL
- Архитектура PostgreSQL - серверные процессы и память, фоновые процессы
- Физическая структура данных - табличные пространства, файловое хранение
- Логический уровень - базы данных, схемы, объекты и search_path, соотношение с физическим уровнем
- Ограничения архитектуры
- Построение отказоустойчивой инфраструктуры
- Облачная архитектура

PostgreSQL

https://aristov.tech

Где используется PostgreSQL

PostgreSQL предназначен в основном для OLTP (**O**n**l**ine **T**ransaction **P**rocessing) нагрузки - много небольших запросов, обычно возвращающих небольшое количество строк.

Примеры:

- Магазин
- Биржа
- Биллинг в телекоме
- Банковская система
- ERP (Система планирования ресурсов предприятия)
- CRM (Система управления взаимоотношениями с клиентами)
- Витрины товаров
- Каталоги
- БД для микросервисов
- Системы логирования

https://aristov.tech

Плюсы PostgreSQL

- Традиционная популярная реляционная модель
- Поддержка множества типов данных, в том числе JSON (jsonb)
- Открытый исходный код
- Работа с большими объемами
- Поддержка сложных запросов, объединений десятков таблиц
- Написание функций на нескольких языках, например можно установить расширение и использовать Python для написания функций и хранимых процедур
- Одновременный параллельный доступ к БД с сотен устройств благодаря системе **MVCC** (Multiversion Concurrency Control) - уникальный принцип **copy-on-write**, что накладывает много ограничений
- Поддержка ACID - Atomicity, Consistency, Isolation, Durability — атомарность, согласованность, изолированность, надежность
- Возможность включать и создавать расширения (extension) - микропрограммы, расширяющие функционал Постгреса
- Высокая мощность и широкая функциональность
- Кроссплатформенность

Минусы PostgreSQL

- внутренняя архитектура отличается от стандартных СУБД, необходимо это учитывать
- сложная система настройки
- соответственно высокий порог вхождения
- нет кластеров “из коробки”
- проблемы с онлайн обновлением на новую версию
- производительность ниже по сравнению с Oracle & MSSql до 30% на сложных запросах
- под OLAP запросы нужно использовать специальные подходы

https://aristov.tech

Что нового в PostgreSQL 17?

https://aristov.tech/blog/chto-novogo-v-postgresql-17/

https://aristov.tech

Немного статистики

Подробнее можно почитать: https://statisticsanddata.org/data/the-most-popular-databases-2006-2023/

Most Popular Databases 2023

Search...

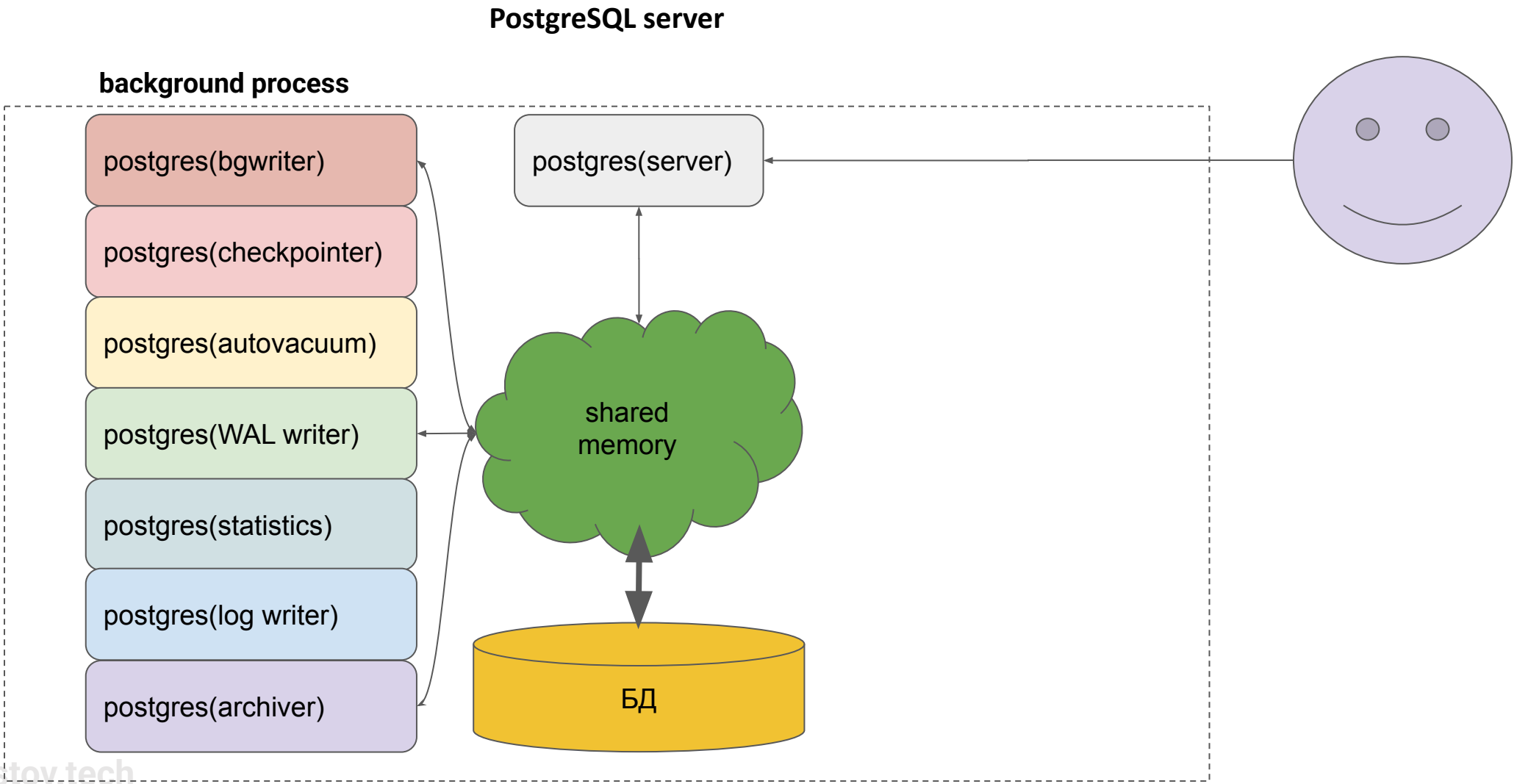
Rank	Name	February 2023	Last month	Last year
1.	Oracle	1,247.52	2.35	-9.31
2.	MySQL	1,195.45	-16.51	-19.23
3.	Microsoft SQL Server	929.09	9.70	-19.96
4.	PostgreSQL	616.50	1.65	7.12
5.	MongoDB	452.77	-2.42	-35.88
6.	Redis	173.83	-3.72	-1.96
7.	IBM Db2	142.97	-0.60	-19.91
8.	Elasticsearch	138.60	-2.56	-23.70

https://aristov.tech

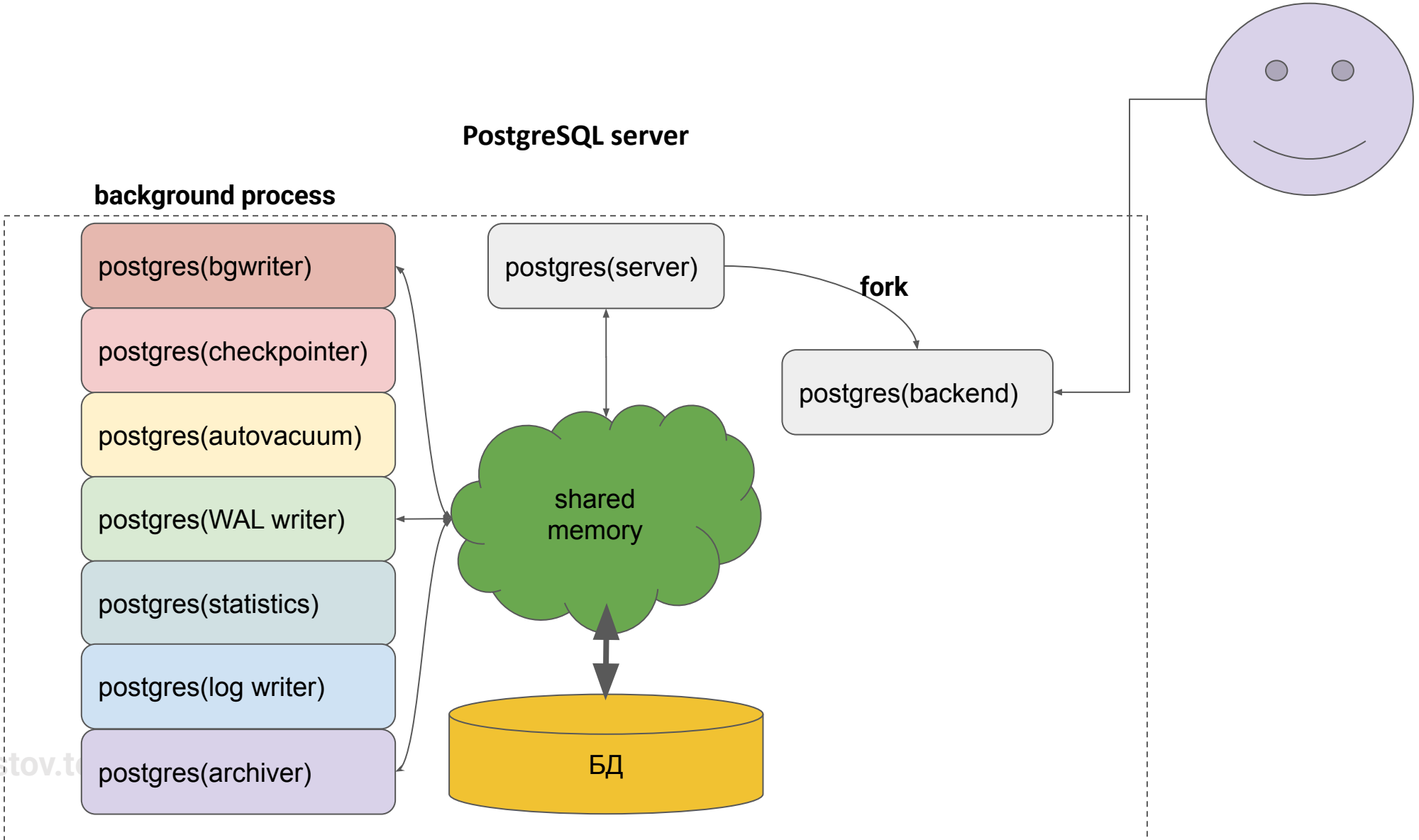
Серверные процессы и память

https://aristov.tech

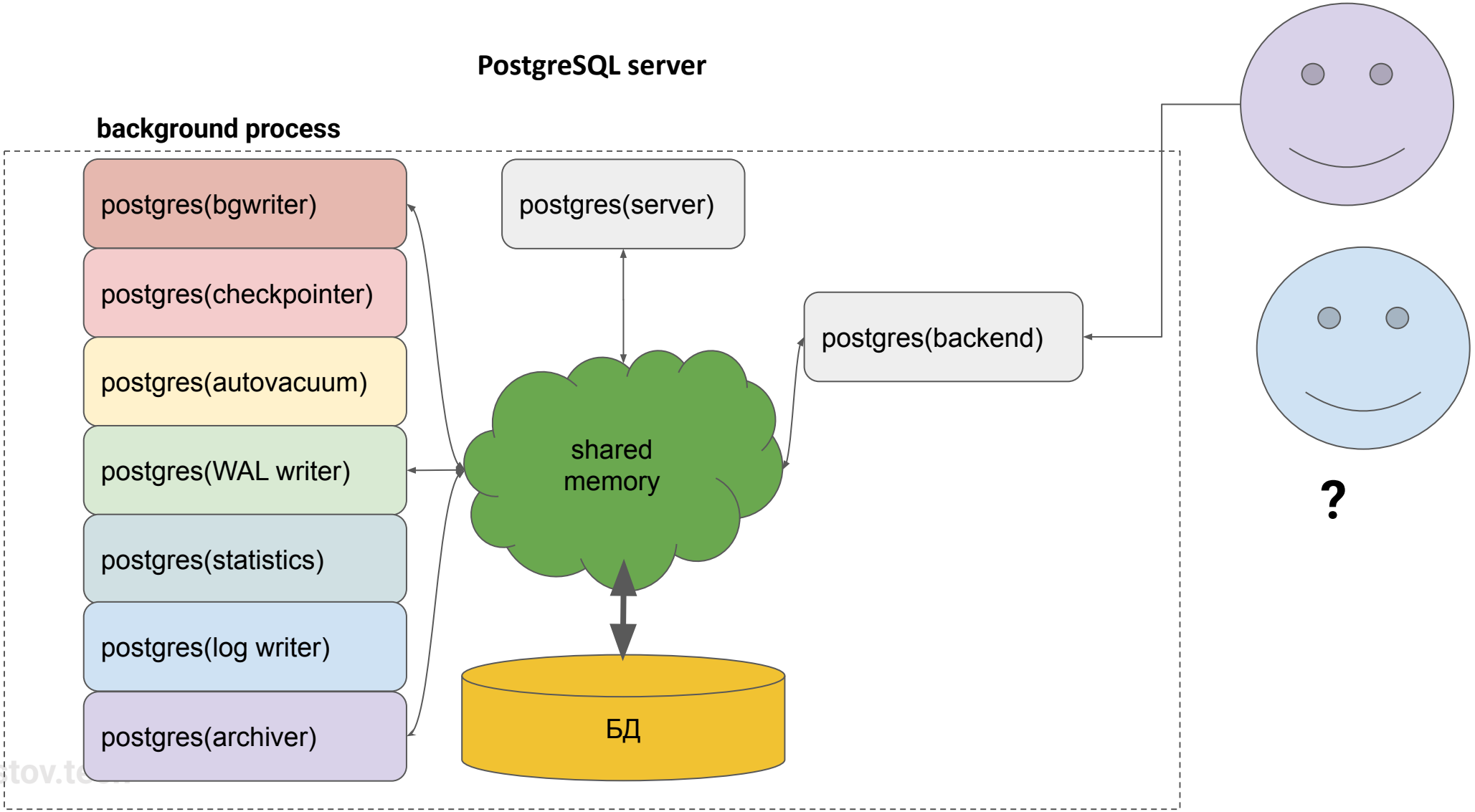
Серверные процессы и память



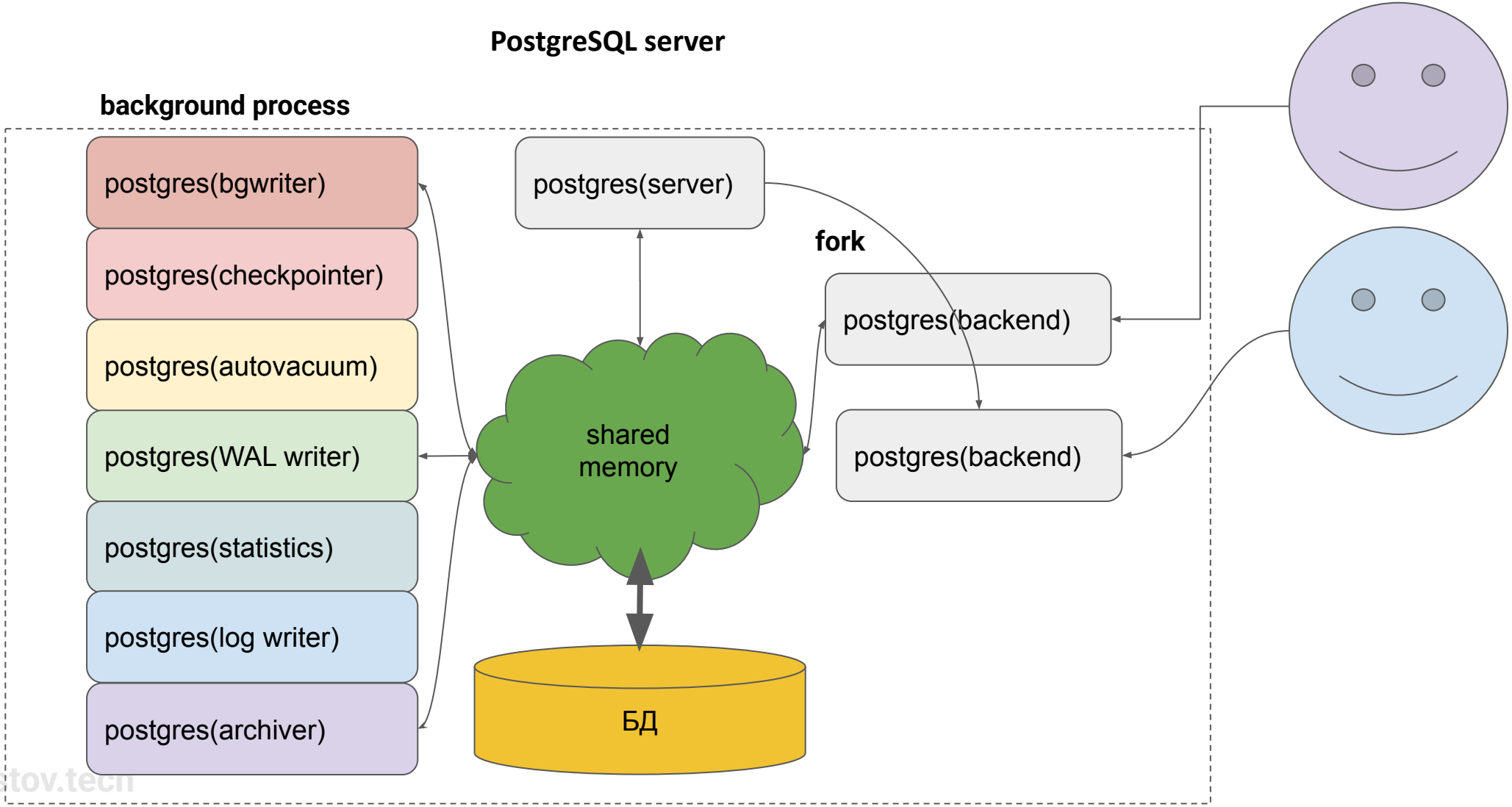
Серверные процессы и память



Серверные процессы и память



Серверные процессы и память



Установка PostgreSQL

https://aristov.tech

Варианты установки

- Собрать из исходников
- on-premise
- **Установка из линукс репозитория** (классика на BM - **VirtualBox**, ЯО, GCP, Hyper-V, ESXi, Proxmos, etc)
- Используем docker
- Используем k8s
- DBaaS

Описаны популярные варианты установки:

https://aristov.tech/blog/

https://aristov.tech

Oracle VirtualBox

Сайт компании https://www.virtualbox.org/

Поддерживаются все современные системы:

- Windows
- Os X
- Linux

Скачать версию под вашу ОС:

https://www.virtualbox.org/wiki/Downloads

Если возникли проблемы с установкой под Windows 10:

https://itigic.com/ru/use-virtualbox-and-vmware-alongside-hyper-v/

https://aristov.tech

Oracle VirtualBox

При создании ВМ указываем:

- диск не меньше 15 Гигабайт
- ОЗУ не меньше 4 Гигабайт

Преимущества Ubuntu 24.04 LTS

Дистрибутивов очень много. Почему Ubuntu?

- Безопасность. Ubuntu— один из самых самых безопасных Линуксов
- Бесплатность
- Большое дружелюбное сообщество
- Простота в использовании
- Удобный центр приложений
- Релизы выходят каждые полгода

Почему 24.04?

- он самый современный
- этот дистрибутив является LTS - Long Time Support, то есть обновления этого дистрибутива будут выходить еще 5 лет. У остальных (не LTS) не более 2 лет, обычно намного меньше

Физическая структура

Физическая структура данных

PostgreSQL работает с данными на дисках только через файловую систему.

Одной таблице соответствует минимум 3 файла:

- с данными (нарезка по 2Гб)
- файл с картой видимости (для мультиверсионности)
- файл с картой свободных строк (при мультиверсионности новая версия строки добавляется в конец, старая помечается на удаление)

Особенности:

- EXT3/4 и XFS наиболее популярны
- Raw devices (like Oracle) не поддерживаются

Best practices:

- не хранить данные в корневой файловой системе
- отдельная файловая система для каждого табличного пространства
- в случае внешнего файлового хранилища - отдельный каталог для каждого **табличного пространства**

https://aristov.tech

Конфигурационные файлы PostgreSQL

Как посмотреть конфигурационные файлы?

- # show hba_file;
- # show config_file;

Путь зависит от ОС. В Ubuntu они расположены:

- /etc/postgresql/17/main/pg_hba.conf
- /etc/postgresql/17/main/postgresql.conf

Посмотрим где в postgresql.conf физически хранятся данные:

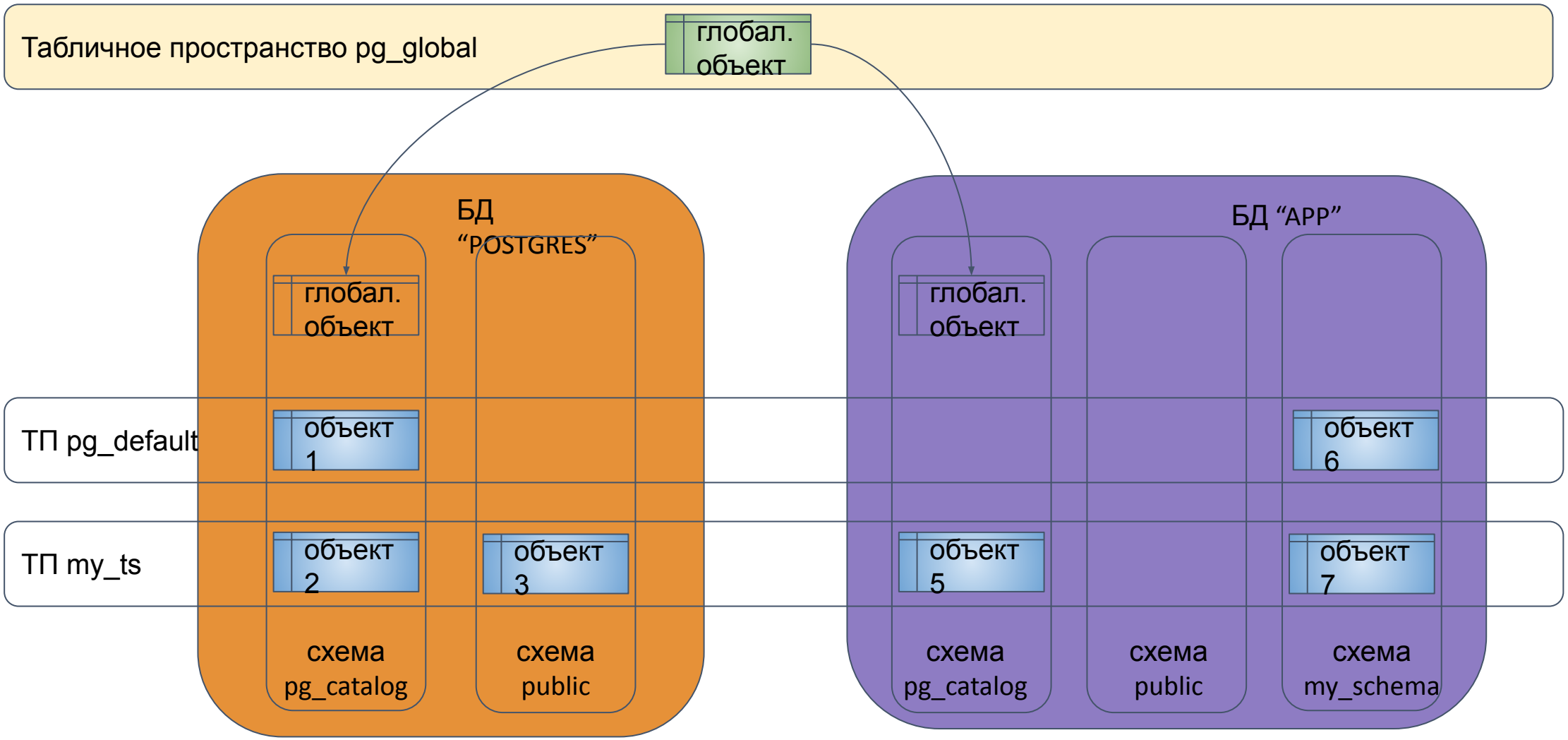
Выполним команду прямо из оболочки psql используя \! :

\! nano /etc/postgresql/17/main/postgresql.conf

https://aristov.tech

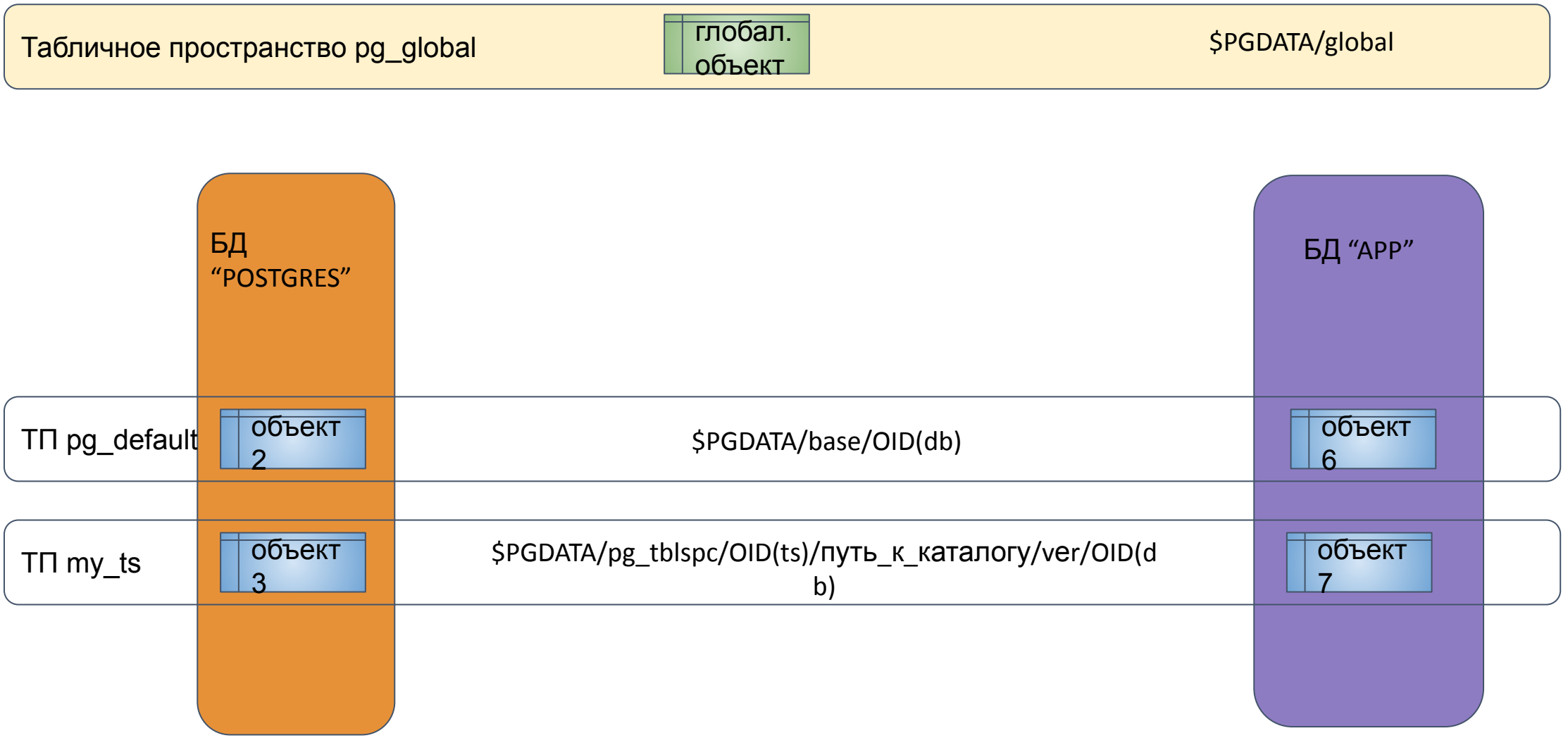
Табличные пространства

Устройство ТП



https://aristov.tech

Табличное пространство. Где файлы?



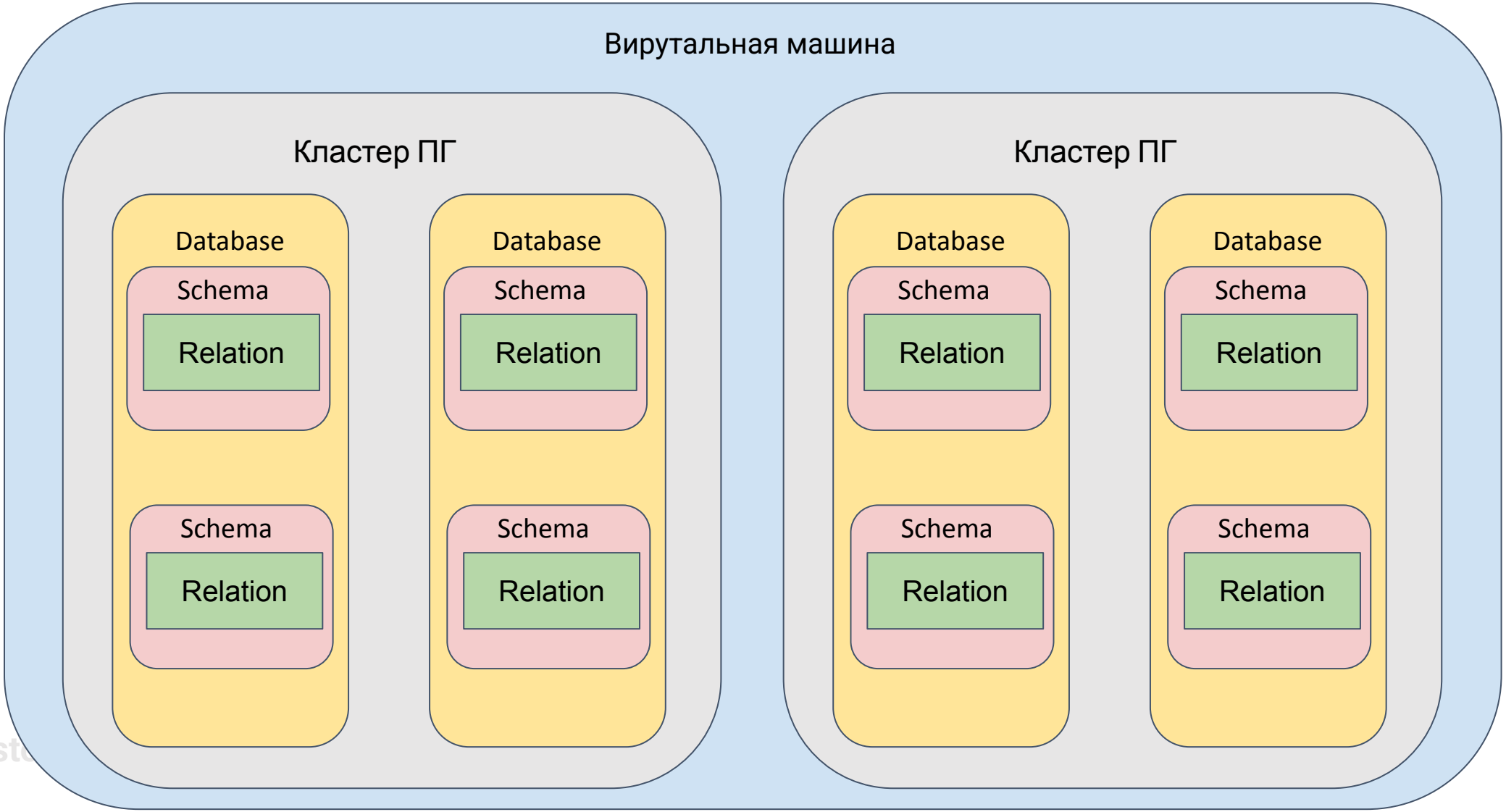
Табличное пространство. Оптимизация

Варианты:

- отдельные объекты (БД, таблицы) размещаем в разных ТП, расположенных на разных дисках, дисковых массивах - таким образом распараллеливаем нагрузку
- можно из оперативной памяти часть смонтировать как файловую систему, создать ТП и держать там, например, материализованные представления, временные таблицы, индексы
- вариант установить локально на сервер ssd диск под такие же некритичные данные

Логический уровень

Общее логическое устройство PostgreSQL



Виды отношений в БД

r = ordinary table

Кроме простых таблиц, также существуют и другие отношения:
i = index,
S = sequence,
v = view,
m = materialized view,
c = composite type,
t = TOAST table,
f = foreign table,

https://www.postgresql.org/docs/15/catalog-pg-class.html

Database

- Является контейнером самого верхнего уровня
- По умолчанию в любом кластере есть как минимум 3 БД:
  - postgres
  - template0
  - template1
- Присутствует на логическом и физическом уровне

template0

- для восстановления из резервной копии
- по умолчанию даже нет прав на connect
- лучше всего не создавать в ней никаких объектов
- а еще лучше про нее забыть и не вспоминать ;)

template1

- используется как шаблон для создания новых баз данных
- в нем имеет смысл делать некие действия, которые не хочется делать каждый раз при создании новых баз данных
- например create extension или create schema
- но (как мне кажется) лучше не создавать объектов, так как для других пользователей это будет неочевидно

https://aristov.tech

postgres

- первая база данных для регулярной работы
- создается по умолчанию
- хорошая практика - также не использовать, но и не удалять - *иногда* нужна для различных утилит (pgbench без указания БД %)

Create Database

```
CREATE DATABASE name
    [ WITH ] [ OWNER [=] user_name ]
    [ TEMPLATE [=] template ]
    [ ENCODING [=] encoding ]
    [ STRATEGY [=] strategy ]
    [ LOCALE [=] locale
..
    [ TABLESPACE [=] tablespace_name ]
    [ ALLOW_CONNECTIONS [=] allowconn ]
    [ CONNECTION LIMIT [=] connlimit ]
    [ IS_TEMPLATE [=] istemplate ]
```

https://www.postgresql.org/docs/current/sql-createdatabase.html

https://aristov.tech

Create Schema

Контейнер 2 уровня.

CREATE SCHEMA **IF NOT EXISTS** *имя_схемы* [AUTHORIZATION *указание_роли*]

https://www.postgresql.org/docs/current/sql-createschema.html

https://aristov.tech

Create Table

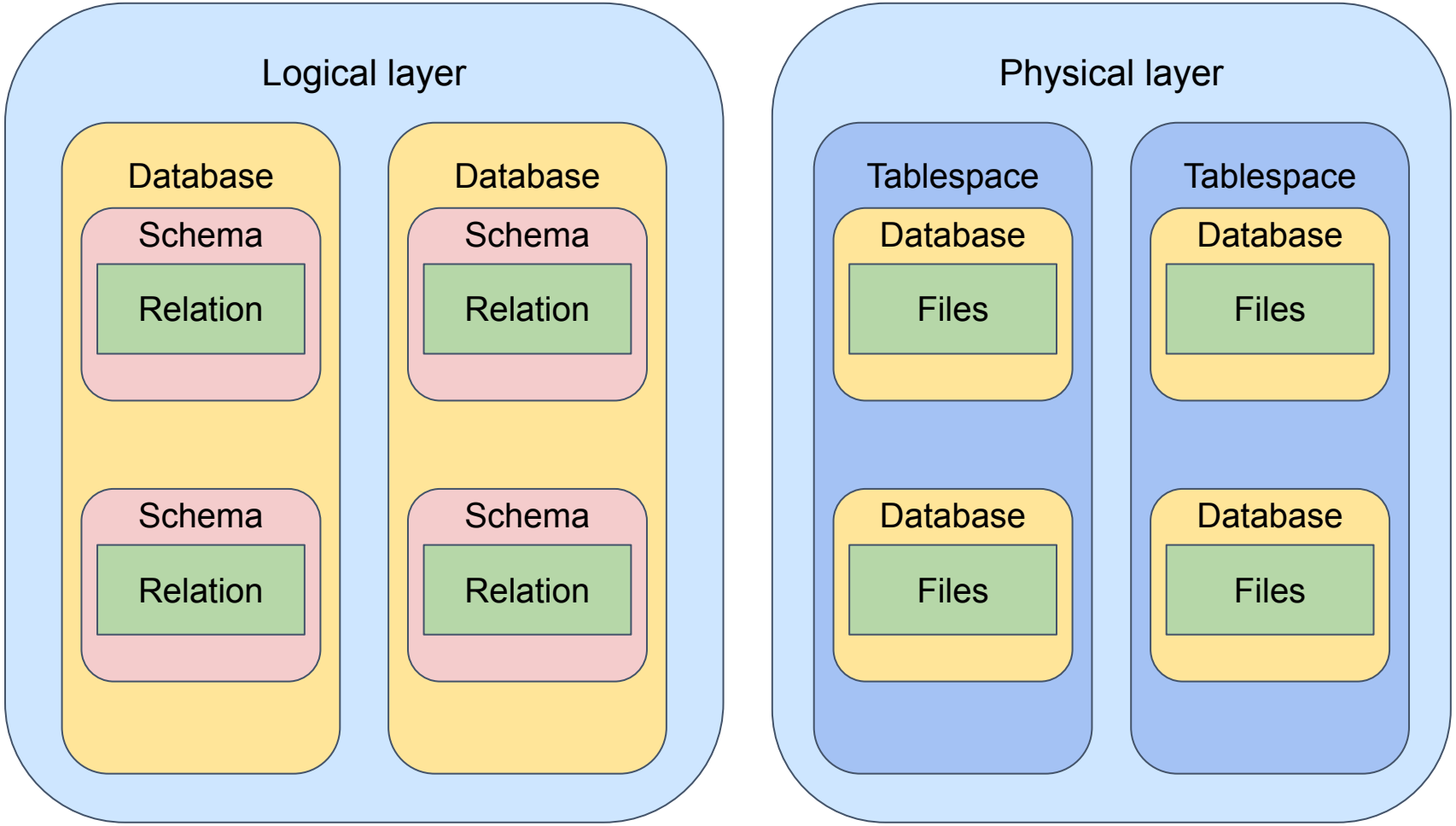
Контейнер 3 уровня.

CREATE [{ TEMPORARY | TEMP } | UNLOGGED] TABLE [IF NOT EXISTS] *имя_таблицы* (
[
  { *имя_столбца тип_данных* [*ограничение_столбца* [...]]

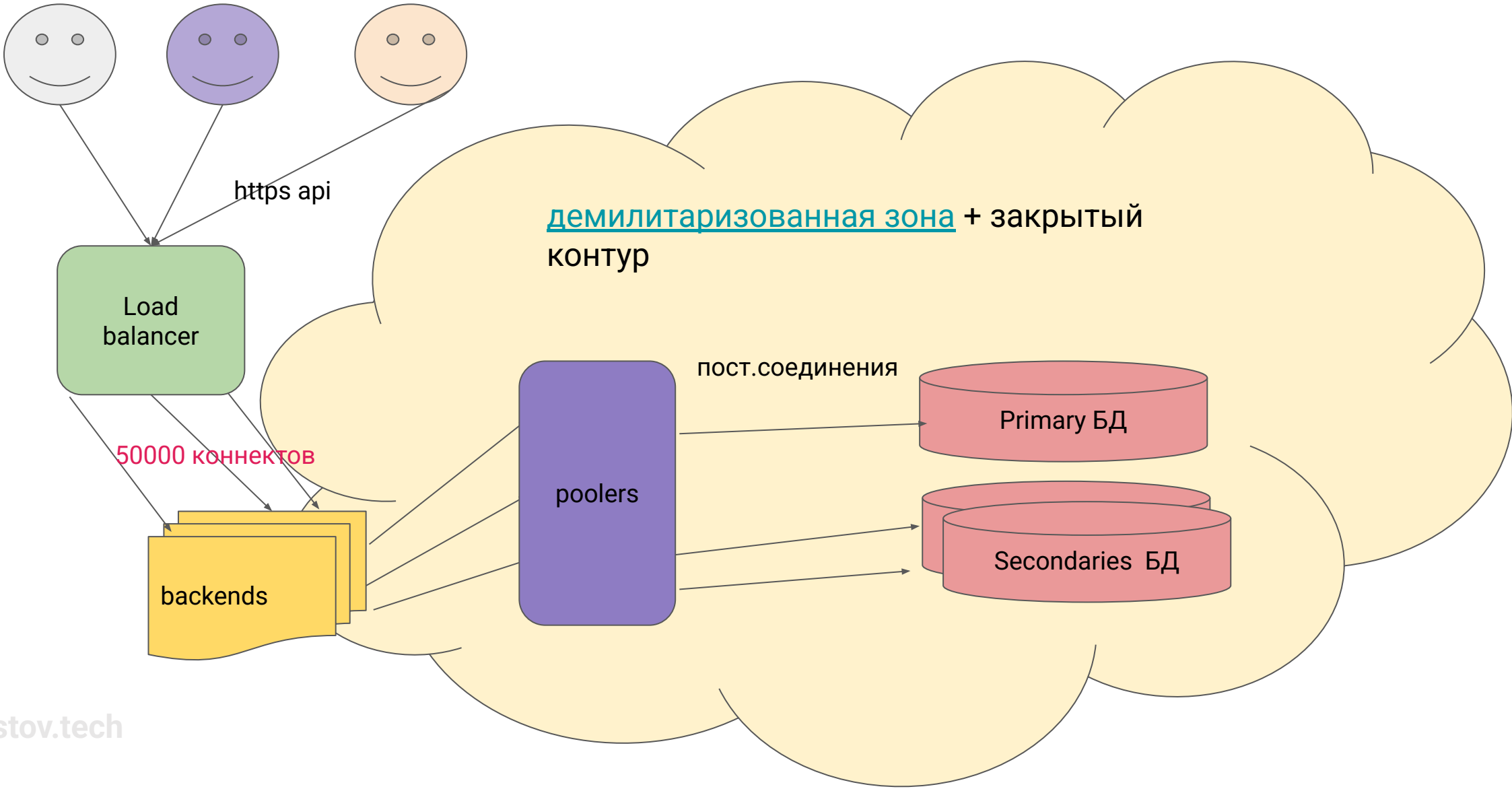
https://www.postgresql.org/docs/current/sql-createtable.html

https://aristov.tech

Соответствие физического и логического уровней PostgreSQL



Ну теперь то все хорошо?



Снова проблема

Вроде все хорошо, но..

кто будет переключать SECONDARY в PRIMARY при проблемах?

Вариант 1

Вроде все хорошо, но..

кто будет переключать SECONDARY в PRIMARY при проблемах?

PgPool II

Вариант 2

Вроде все хорошо, но..

кто будет переключать SECONDARY в PRIMARY при проблемах?

Pg_auto_failover:

- HA
- отказоустойчивость
- синхронный и асинхронный режим
- открытый исходный код

https://github.com/citusdata/pg_auto_failover

Вариант 3

Вроде все хорошо, но..

кто будет переключать SECONDARY в PRIMARY при проблемах?

Repmgr:

- открытый исходный код
- автоматическое переключение основного сервера при падении
- очень простая установка и масштабирование

https://github.com/EnterpriseDB/repmgr

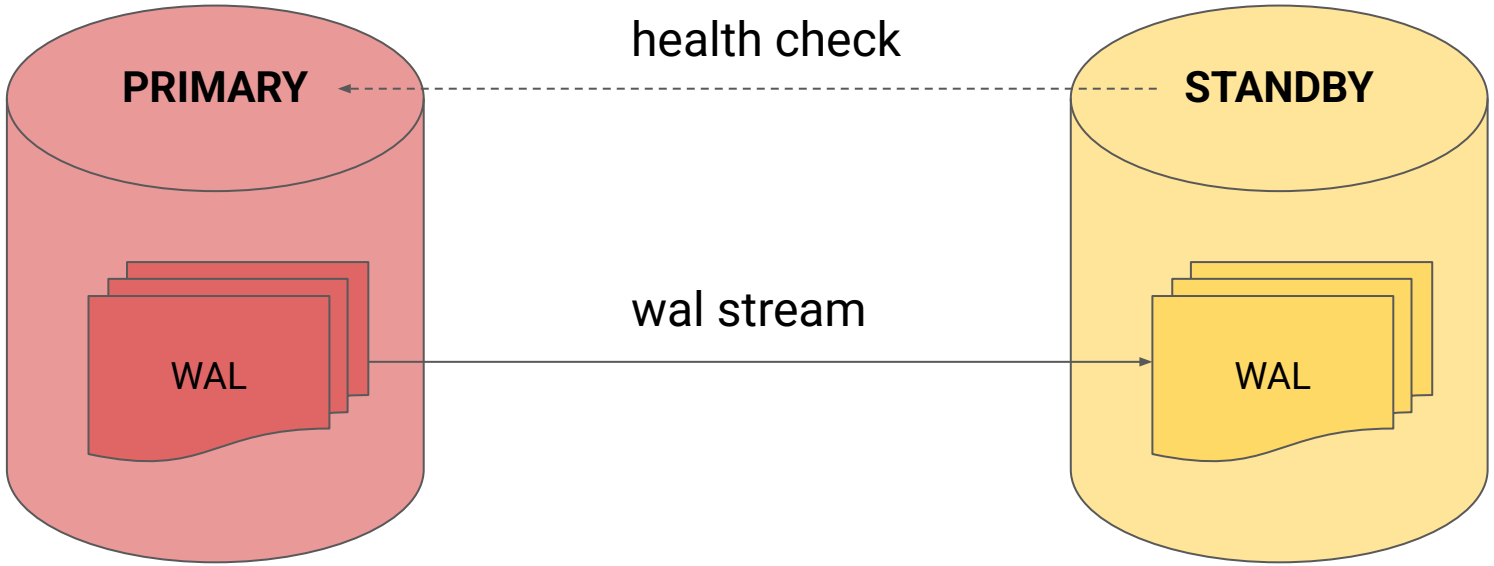
Вариант X

Вроде все хорошо, но..

кто будет переключать SECONDARY в PRIMARY при проблемах?

А кстати, мы ничего не забыли?)

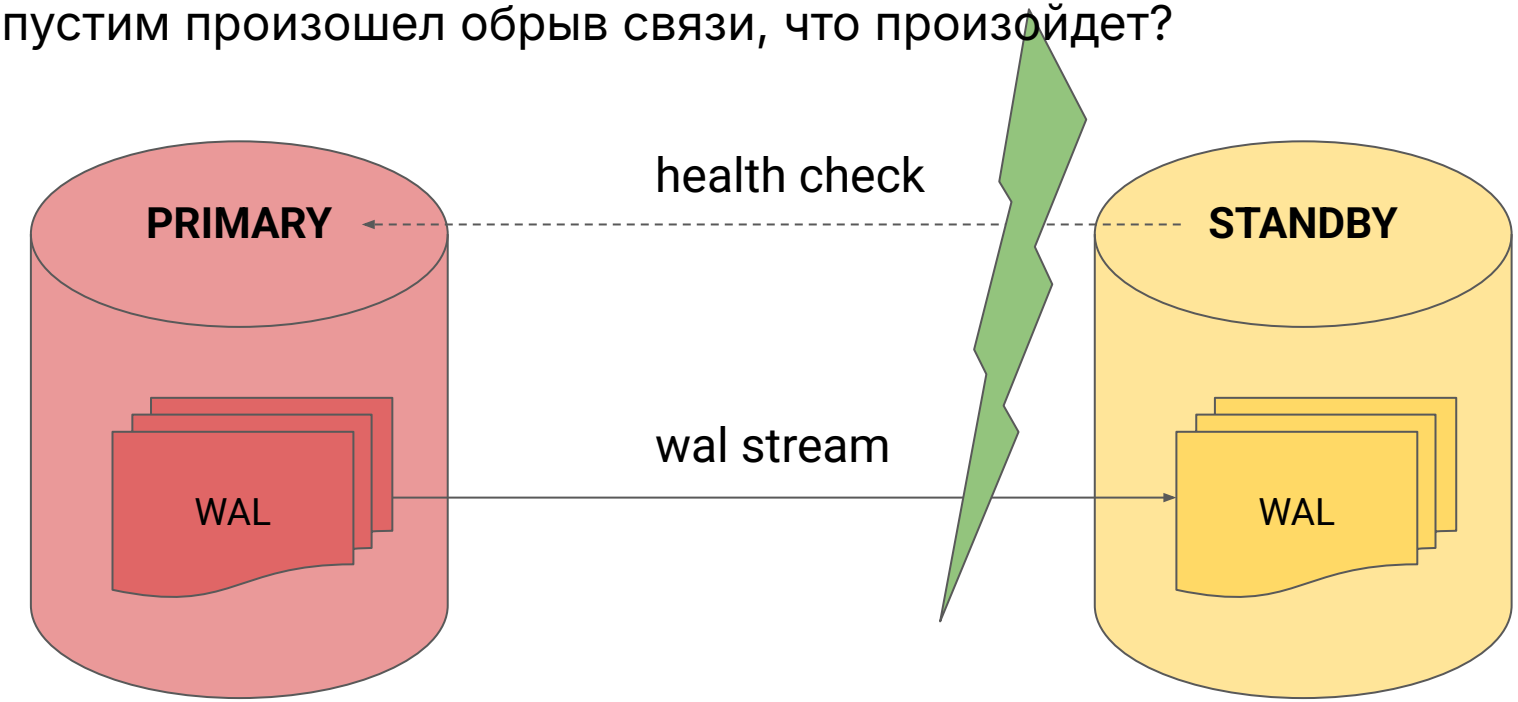
Кластер из 2 нод



https://aristov.tech

Кластер из 2 нод

Допустим произошел обрыв связи, что произойдет?



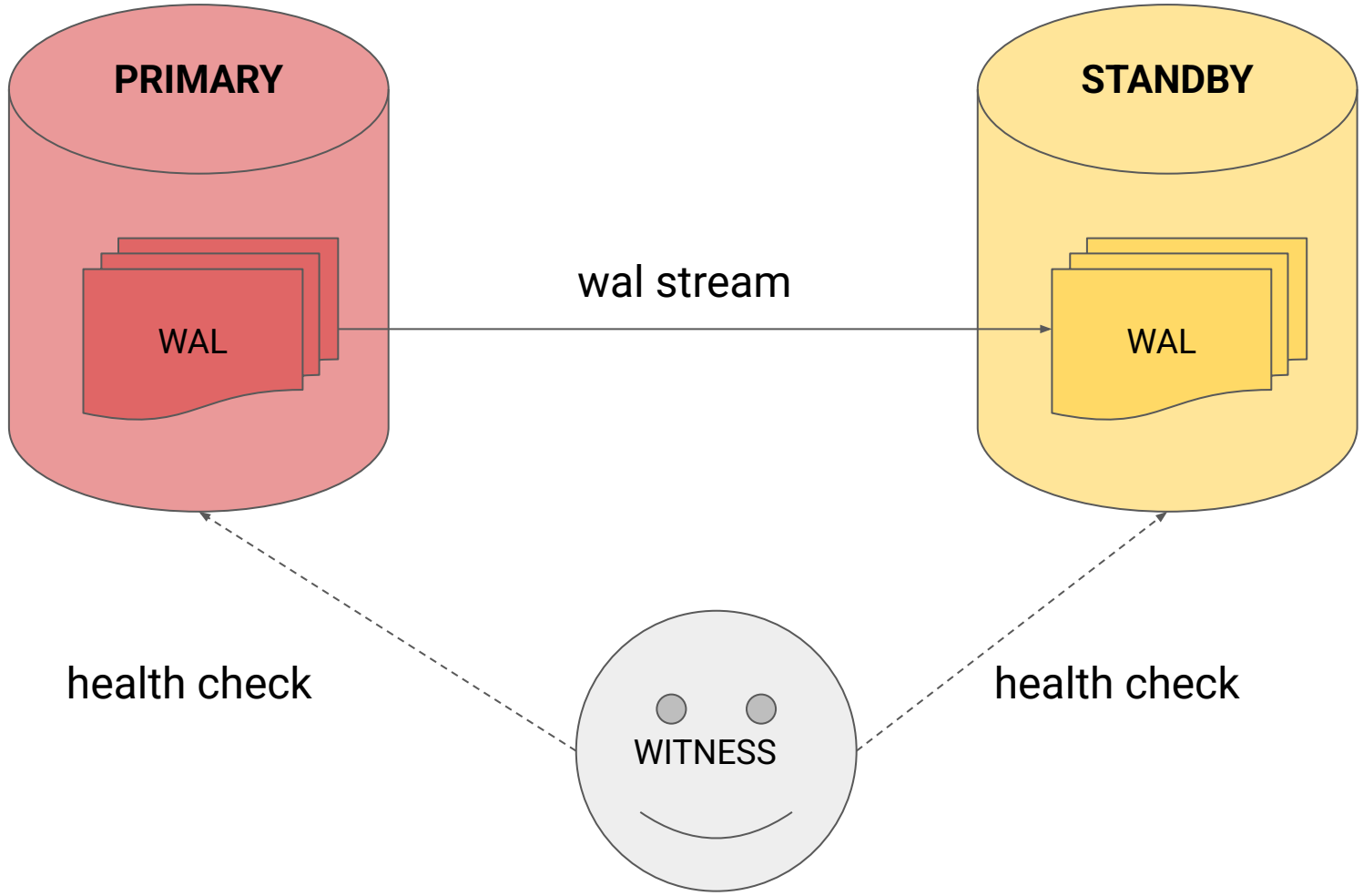
Кластер из 2 нод

Допустим произошел обрыв связи, что произойдет?

Сплитбрейн !!!

Кластер из 2 нод with Witness

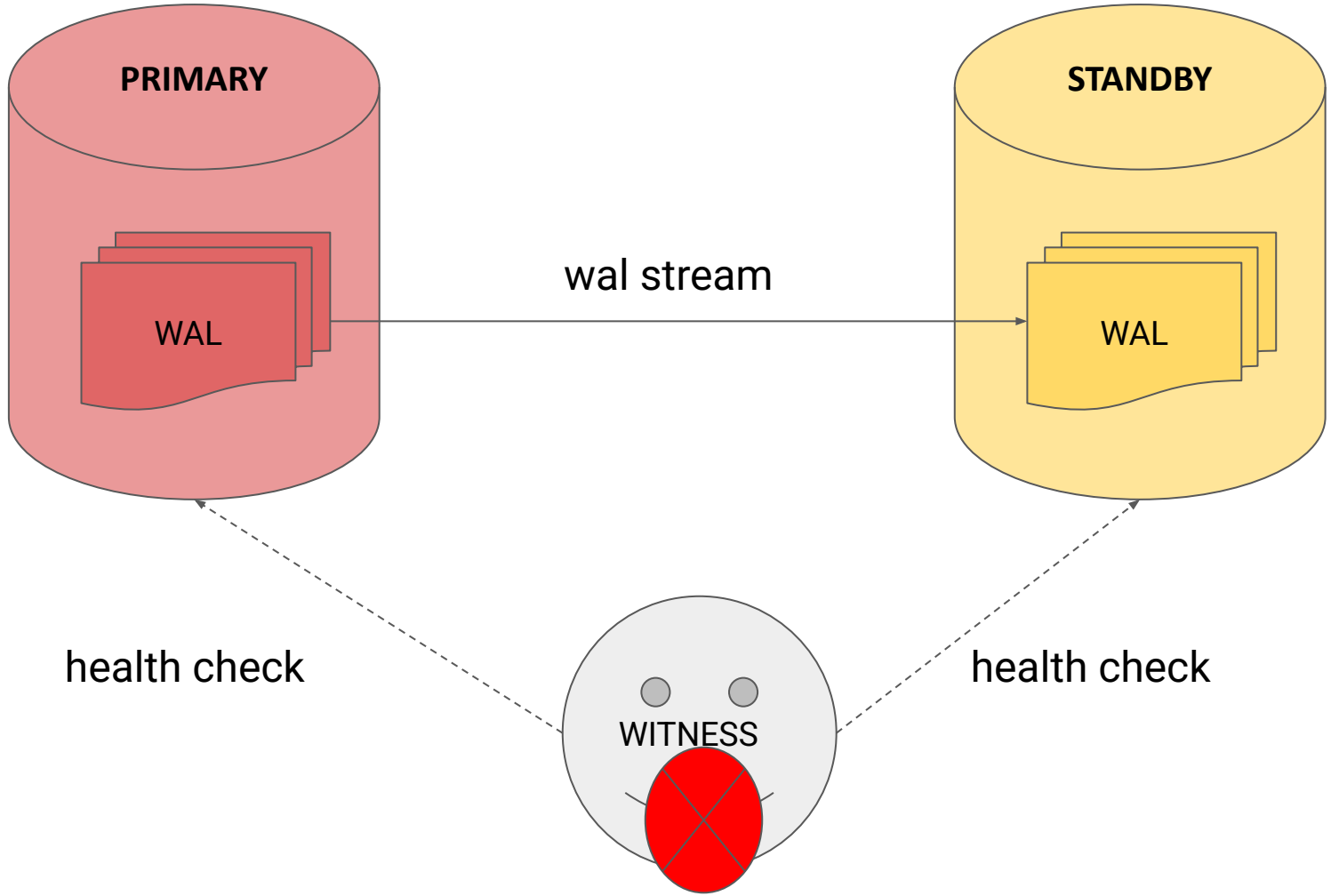
Что может пойти не так??



https://aristov.tech

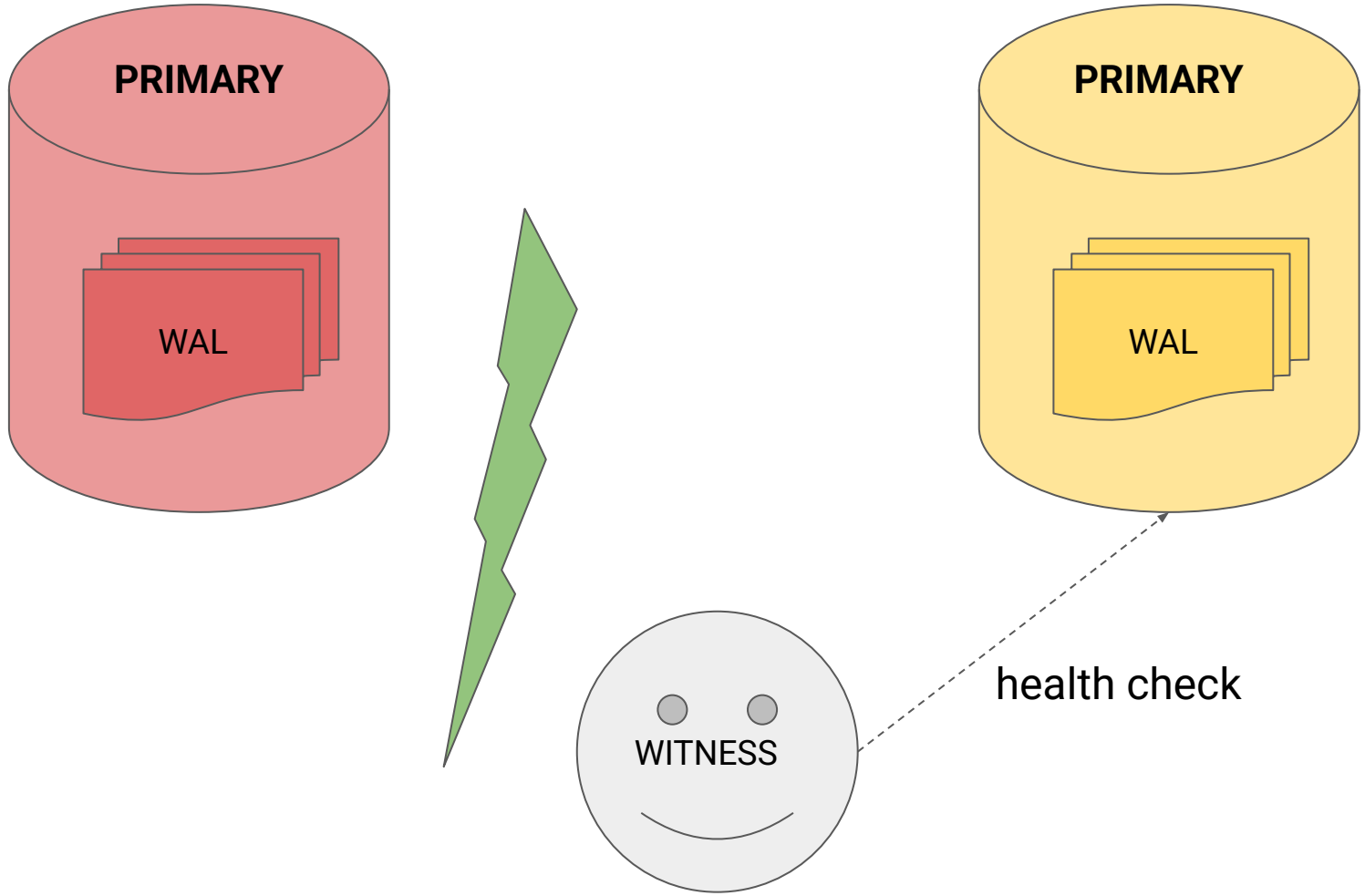
Кластер из 2 нод with Witness

Умер наблюдатель



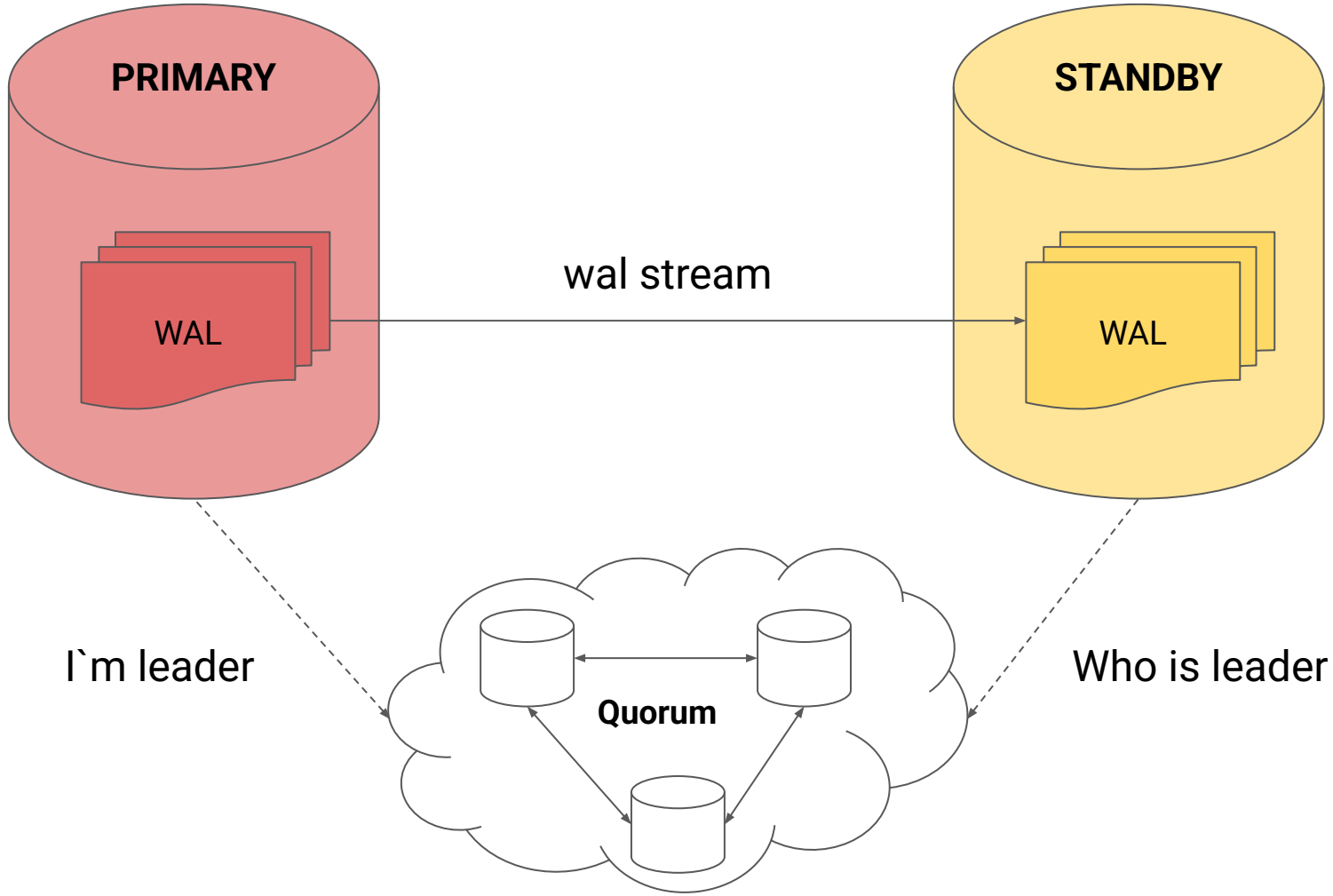
Кластер из 2 нод with Witness

Обрыв соединения с основной нодой. Казалось бы и что? А снова сплитбрейн(



Кластер из 2 нод с ETCD/CONSUL/ZOOKEEPER

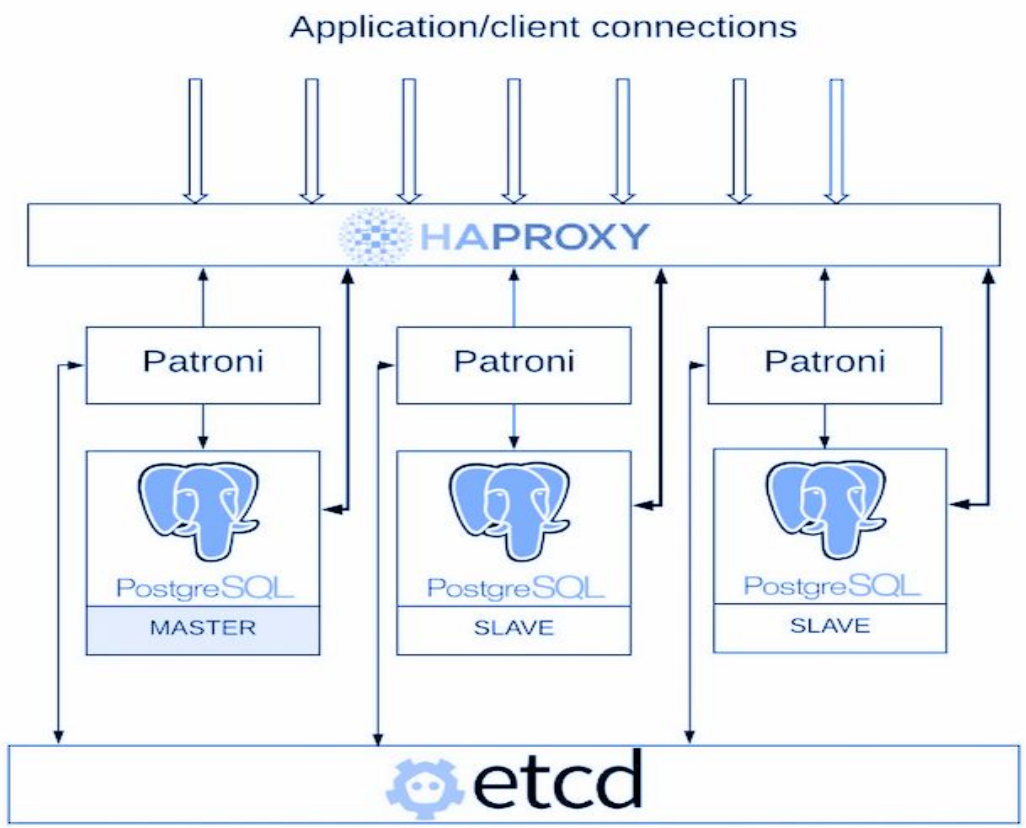
Одно из решений (Patroni)



Patroni

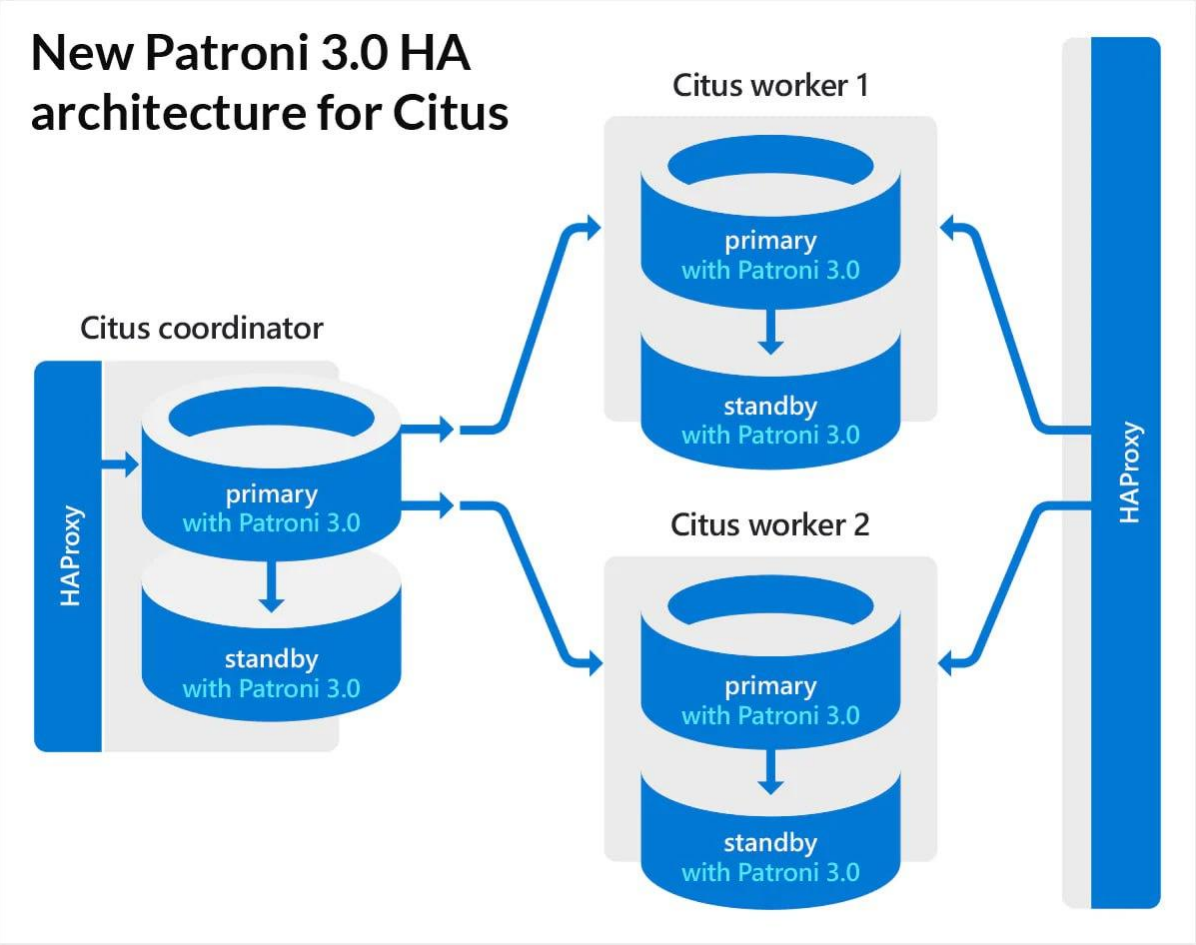
Самый популярный кластер

Можно еще добавить pgbouncer, keepalived+2 HAPROXY для HA

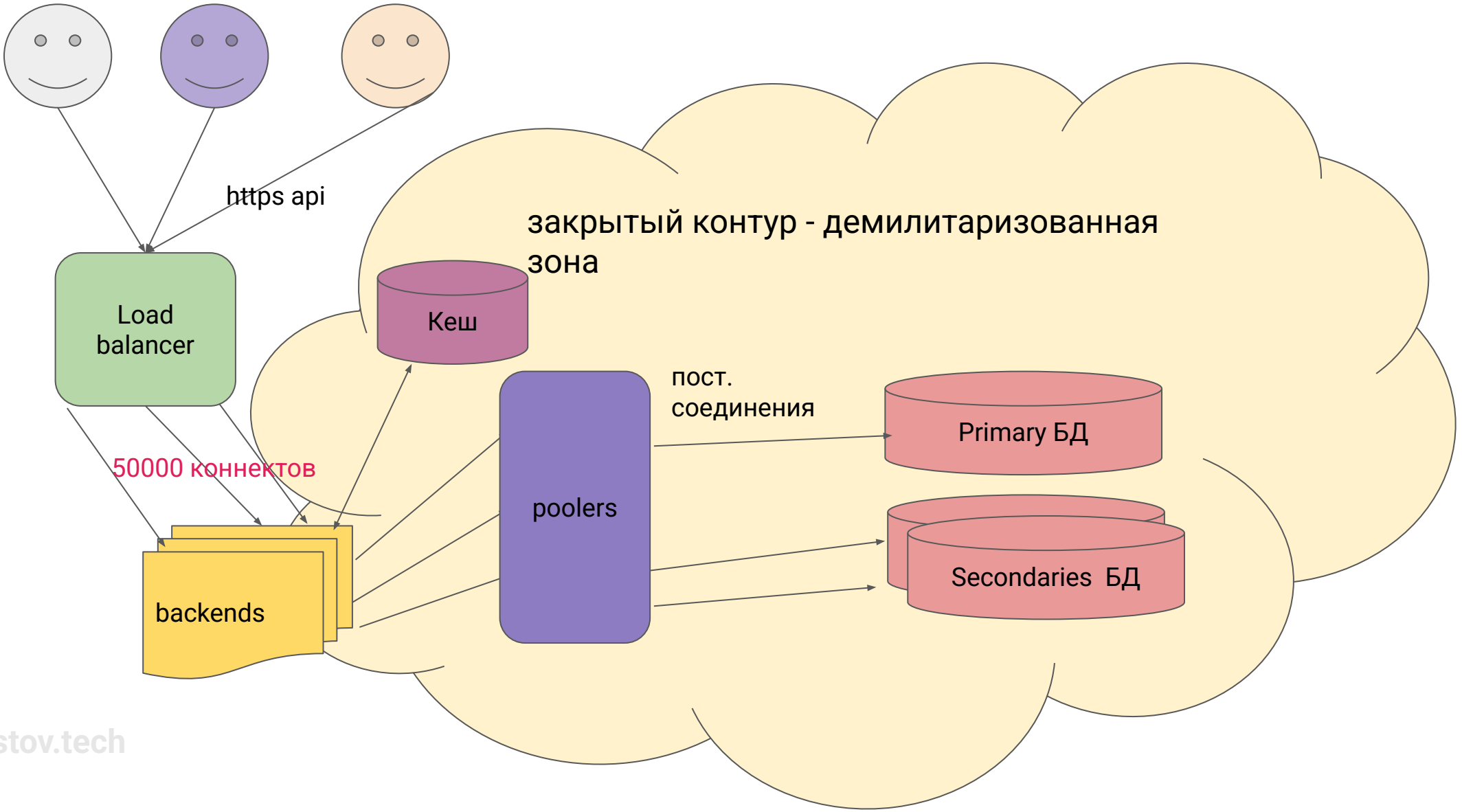


Пример CITUS + HA Patroni

P.S. вообще в 3 Patroni протокол raft derpracated



Добавим кэш. Какие проблемы остались?



Какие проблемы при этом есть?

Если делать локальный кэш, то лоад балансер должен запросы с 1 фронта всегда отправлять на нужный бэкенд.

Если делать общий tarantool/redis (не совсем кэш), то сетевые задержки.

И общие проблемы:

- TTL
- инвалидация кэша

Предпочтительно использовать кластера

HA:

- Patroni
- Stolon
- Slony
- ClusterControl
- Kubegres

Параллельные кластера:

- Postgres-BDR
- CitusData
- Bucardo
- CockroachDB
- Yogabyte
- Greenplum

Опять же как развернуть.. on-premise, docker, k8s...

А уж сколько облачных решений...

https://aristov.tech

Ограничения архитектуры

Общие проблемы

- огриничения коннектинга!
- ограничение 32Тб на одну таблицу - заранее думаем про секционирование!!!
- бэкап снимаем с secondary и не забываем про валидацию бэкапа
- возможно есть смысл в каскадной репликации при проблемах с производительностью сети
- синхронная реплика снижает производительность, но обеспечивает надежность
- георасределение нагрузки
- строим планы развития системы (краткосрочный, долгосрочный) - прогноз обязателен!!!
- мониторим пустое место !!! размер WAL и не только !!!
- **Не забываем включать pg_rewind !!! В stolon по умолчанию отключен!!!**

https://aristov.tech

Что делать с OLAP

- *отдельная реплика для OLAP!!*
- промежуточные агрегаты, избыточность
- материализованные представления

... может быть использование Хранимых Процедур?

... message broker для накопления данных и их ночная обработка?

... может быть готовые OLAP решения (GreenPlum, ArenaData, ClickHouse, Citus)?

... может есть смысл fdw? timescaledb?

Как мы понимаем, серебряной пули, к сожалению, не существует(

https://aristov.tech

Облачные и кубер возможности

Облачные варианты

DBaaS у любого провайдера.

Плюсы:

- кластер за пару кликов в GUI
- бэкапы, репликация за 1 нажатие

Минусы:

- повышенная стоимость - только платные лицензии
- нет контроля над инстансом
- минимум настроек
- дебаг запросов превращается в многоуровневый квест
- обычно HA реплику не получится использовать как read реплику
- ооочень огромная сложность построения мултиклауд конфигураций и даже реплики у другого провайдера
- время развертывания инстанса значительно превышает вариант on premise

https://aristov.tech

Kubernetes

Как в облачном так и on premise кластере.

Плюсы - гибкость и простота развертывания HA конфигураций со встроенными пулерами и т.д., обычно есть GUI для управления.

Варианты:

- свой релиз обернуть в чарт
- **cloud native**
  - **patroni**
  - **stolon**
  - **ClusterControl**
- операторы (для YAML разработчиков новый тип ресурса - postgresql)
  - от авторов Патрони https://github.com/zalando/postgres-operator
  - от дистрибьютора Crunchy https://github.com/CrunchyData/postgres-operator
  - от контрибьютора EnterpriseDB https://cloudnative-pg.io/
  - на хабе операторов 15 вариантов

Kubernetes operator

Посмотрим архитектуру на примере: https://github.com/zalando/postgres-operator

```yaml
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: acid-minimal-cluster
spec:
  teamId: "acid"
  volume:
    size: 1Gi
  numberOfInstances: 2
  users:
    zalando:  # database owner
    - superuser
    - createdb
    foo_user: []  # role for application foo
  databases:
    foo: zalando  # dbname: owner
  preparedDatabases:
    bar: {}
  postgresql:
    version: "16"
```

https://aristov.tech

Kubernetes

В статье на Хабре собраны и протестированы основные операторы:

	Stolon	Crunchy Data	Zalando	KubeDB	StackGres	CloudNativePG
Текущая версия	0.17.0	5.1.2	1.8.0	0.17	1.2.0	1.16.0
Версии PostgreSQL	9.6-14	10-14	9.6-14	9.6-14	12, 13	10-14
Общие возможности						
Кластеры PgSQL	✓	✓	✓	✓	✓	✓
Теплый и горячий резерв	✓	✓	✓	✓	✓	✓
Синхронная репликация	✓	✓	✓	✓	✓	✓
Потоковая репликация	✓	✓	✓	✓	✓	✓

Итоги

Итоги

Остались ли вопросы?

Увидимся на следующем занятии

ДЗ

ДЗ

1. Развернуть ВМ (Linux) с PostgreSQL (у вас есть ВМ в ВБ, любой другой способ, в т.ч. докер)
2. Залить Тайские перевозки в минимальном варианте

https://github.com/aeuge/postgres16book/tree/main/database

3. Посчитать количество поездок - select count(*) from book.tickets;
4. Не забываем ВМ остановить/удалить

Сдача в формате markdown на github/gitlab/etc на почту doit.ifti@yandex.ru

Не забываем указать ФИ и номер задания - ответ в течении обычно рабочих суток, стараюсь быстрее

Посмотрим как развернуть в ЯО

Спасибо за внимание!

Когда дальше и куда?
вт/чт в 19

Аристов Евгений

https://aristov.tech