

Евгений Аристов

Сравнение PostgreSQL и других решений



<https://aristov.tech>

Все материалы защищены авторским правом. Использование полностью или частично разрешено только с письменного разрешения Аристов Е.Н. (с)





25 лет занимаюсь разработкой БД и ПО

Архитектор высоконагруженных баз данных и инфраструктуры

Спроектировал и разработал сотни проектов для финансового сектора, сетевых магазинов, фитнес-центров, отелей

Деплой БД как on-premise Google Cloud Platform, AWS, Azure, Yandex Cloud, так и Kubernetes, DBaaS, MultiCloud.

Автор курсов и книг по PostgreSQL.

Недавно вышла книга "[PostgreSQL 16. Лучшие практики оптимизации](#)"

Правила вебинара

<https://aristov.tech>

Задаем вопрос в чат

Можно общаться голосом, для этого поставьте знак ? в чат

Вопросы вижу, отвечу в момент логической паузы

Если остались вопросы, можно написать мне через сайт aristov.tech

<https://aristov.tech>

Маршрут вебинара

<https://aristov.tech>

- ❖ Разобрать основные фишки и преимущества с недостатками:
 - PostgreSQL
 - MongoDB
 - CRDB
 - ClickHouse
 - Elasticsearch и альтернативы
- ❖ Коротко о проекте aristov.tech
- ❖ Розыгрыш скидки 30% на курс
- ❖ Ответ на вопросы

<https://aristov.tech>

PostgreSQL

PostgreSQL. Фичи

<https://aristov.tech>

<https://www.postgresql.org/>

Плюсы:

- ❖ [ACID](#) - всё оборачивается в транзакции
- ❖ огромное комьюнити
- ❖ легковесность
- ❖ открытость
- ❖ бесплатность
- ❖ хорошая производительность [OLTP](#)
- ❖ постоянный выход новых версий

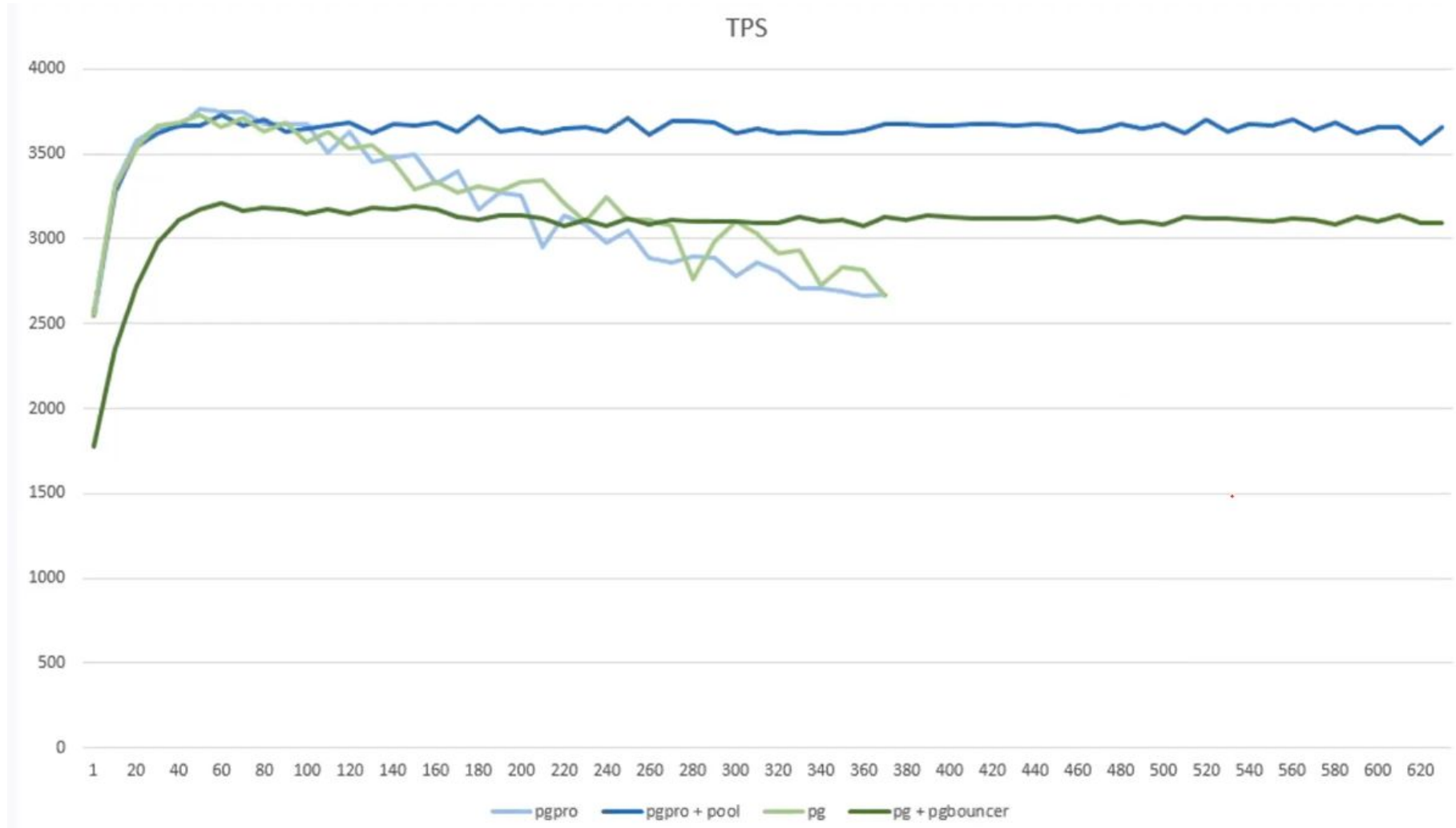
Минусы:

- ❖ производительность ниже конкурентов
- ❖ высокий порог вхождения из-за особенностей архитектуры
- ❖ хороший тюнинг и решение основных тонких мест - требует просто [огромной квалификации](#)
- ❖ плохая производительность [OLAP](#)
- ❖ ужасно масштабируется - нет кластеров из коробки
- ❖ проблемы с коннектингом
- ❖ настройка отказоустойчивости имеет ооочень высокий порог для входа

<https://aristov.tech>

600++ conn

<https://aristov.tech>

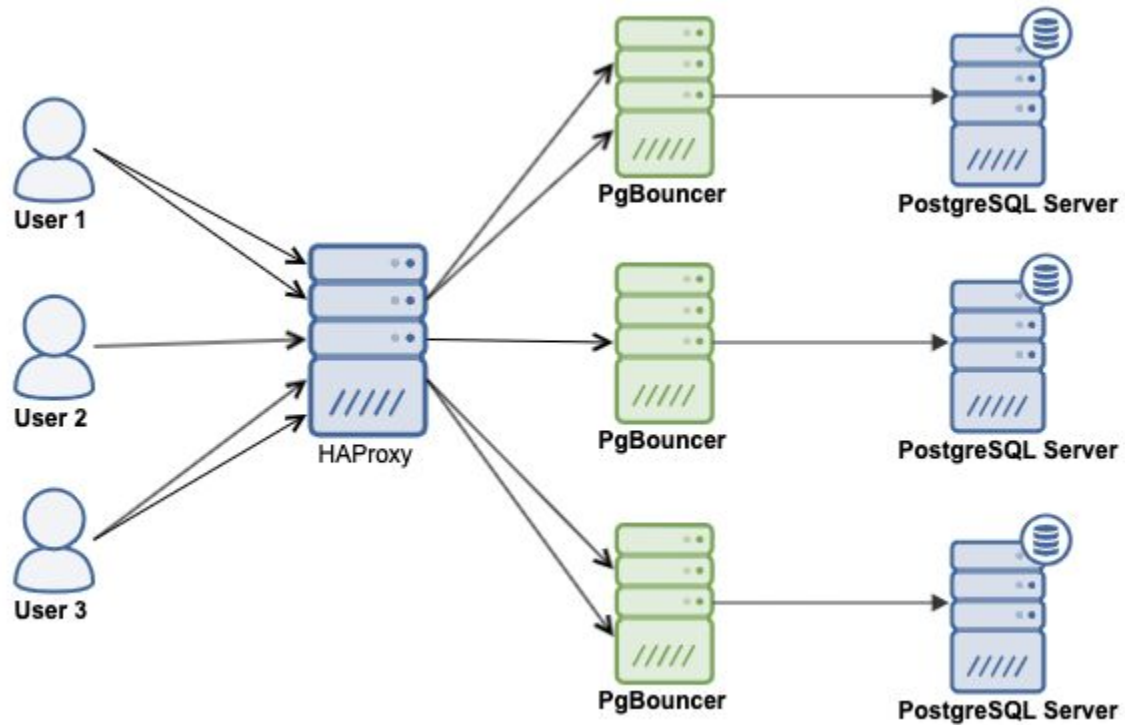


<https://aristov.tech>

<https://postgrespro.ru/docs/enterprise/15/connection-pooler-configuration>

Балансируем нагрузку

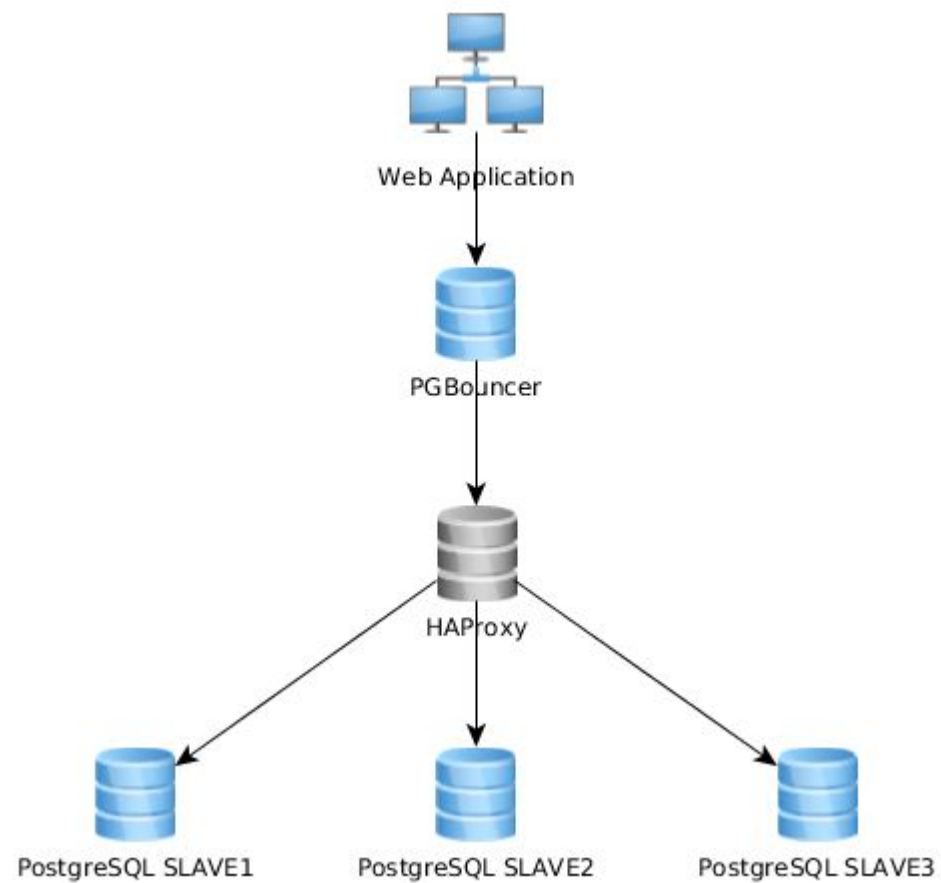
<https://aristov.tech>



<https://aristov.tech>

Балансируем нагрузку

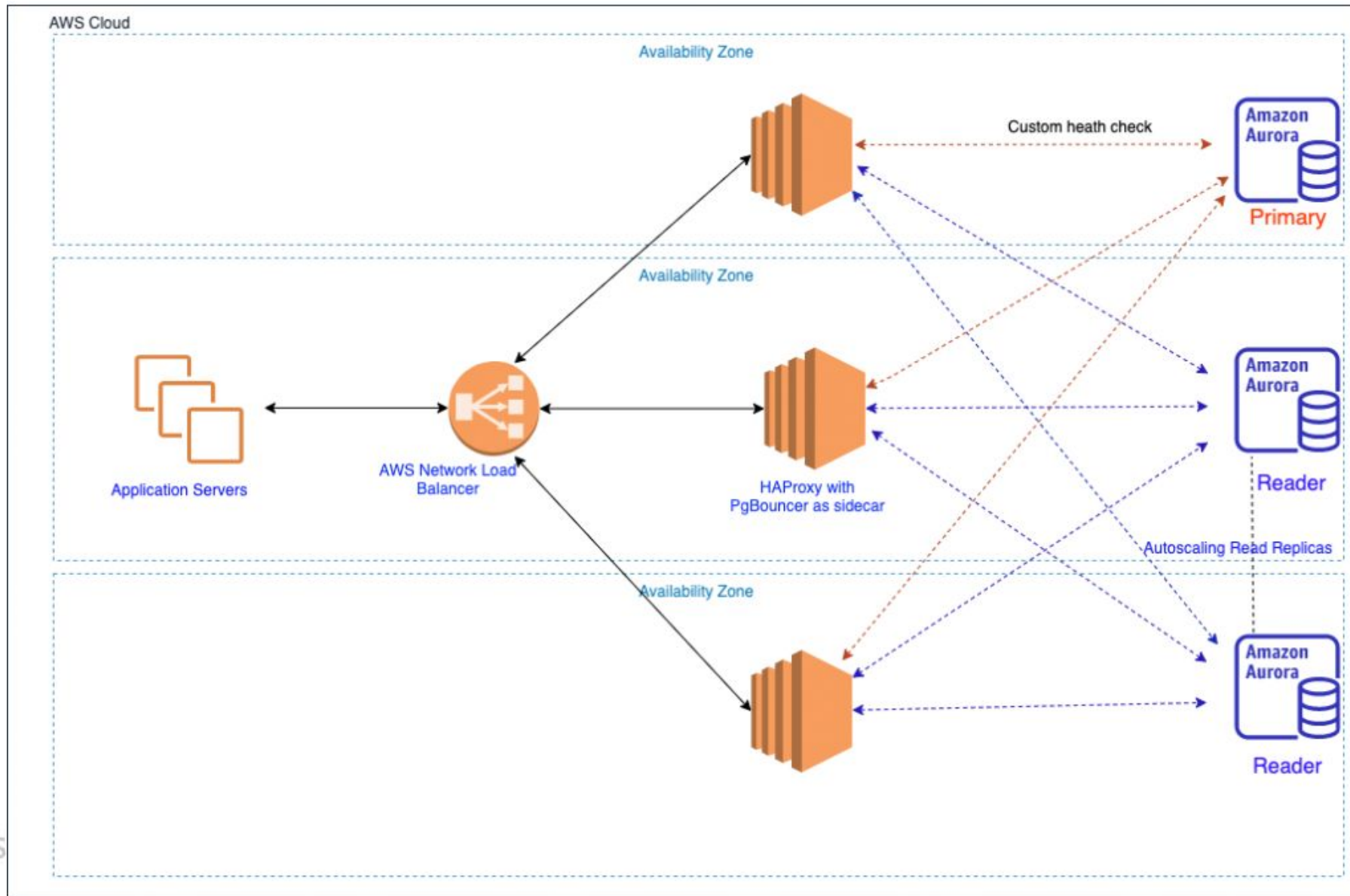
<https://aristov.tech>



<https://aristov.tech>

Балансируем нагрузку

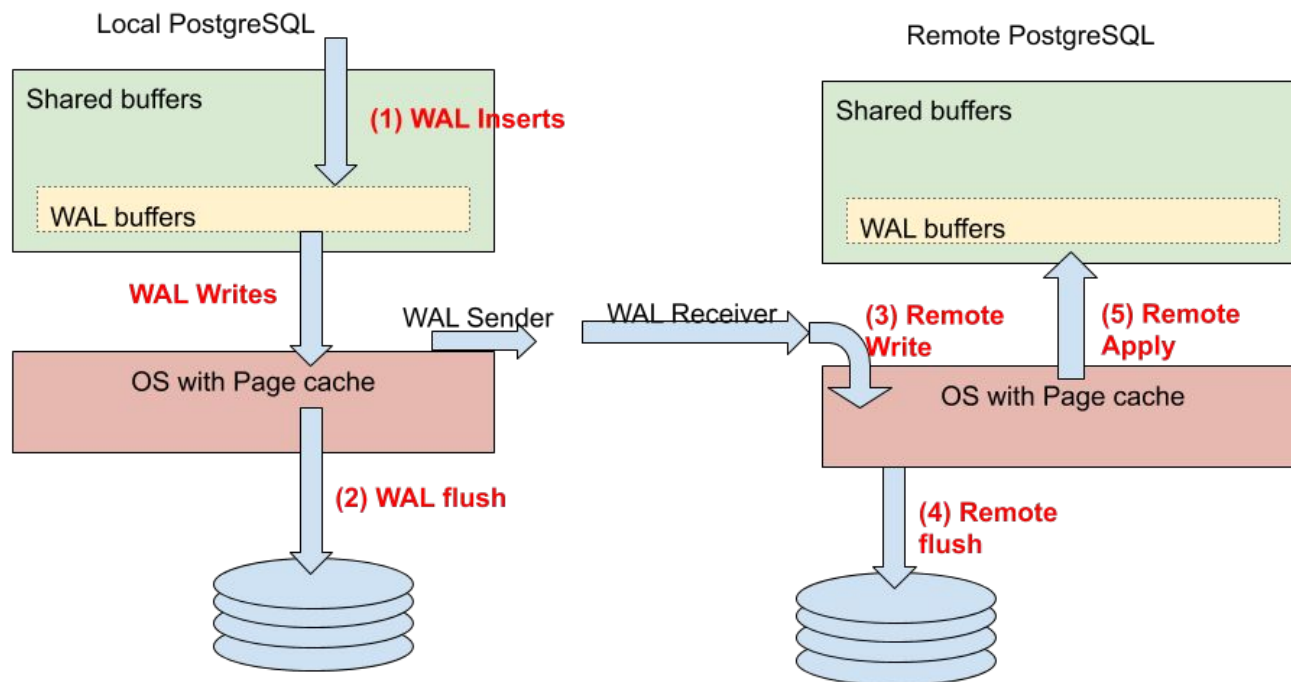
<https://aristov.tech>



<https://aristov.tech>

Физическая репликация

Особенности репликации:



https://www.percona.com/blog/postgresql-synchronous_commit-options-and-synchronous-standby-replication/

Физическая репликация

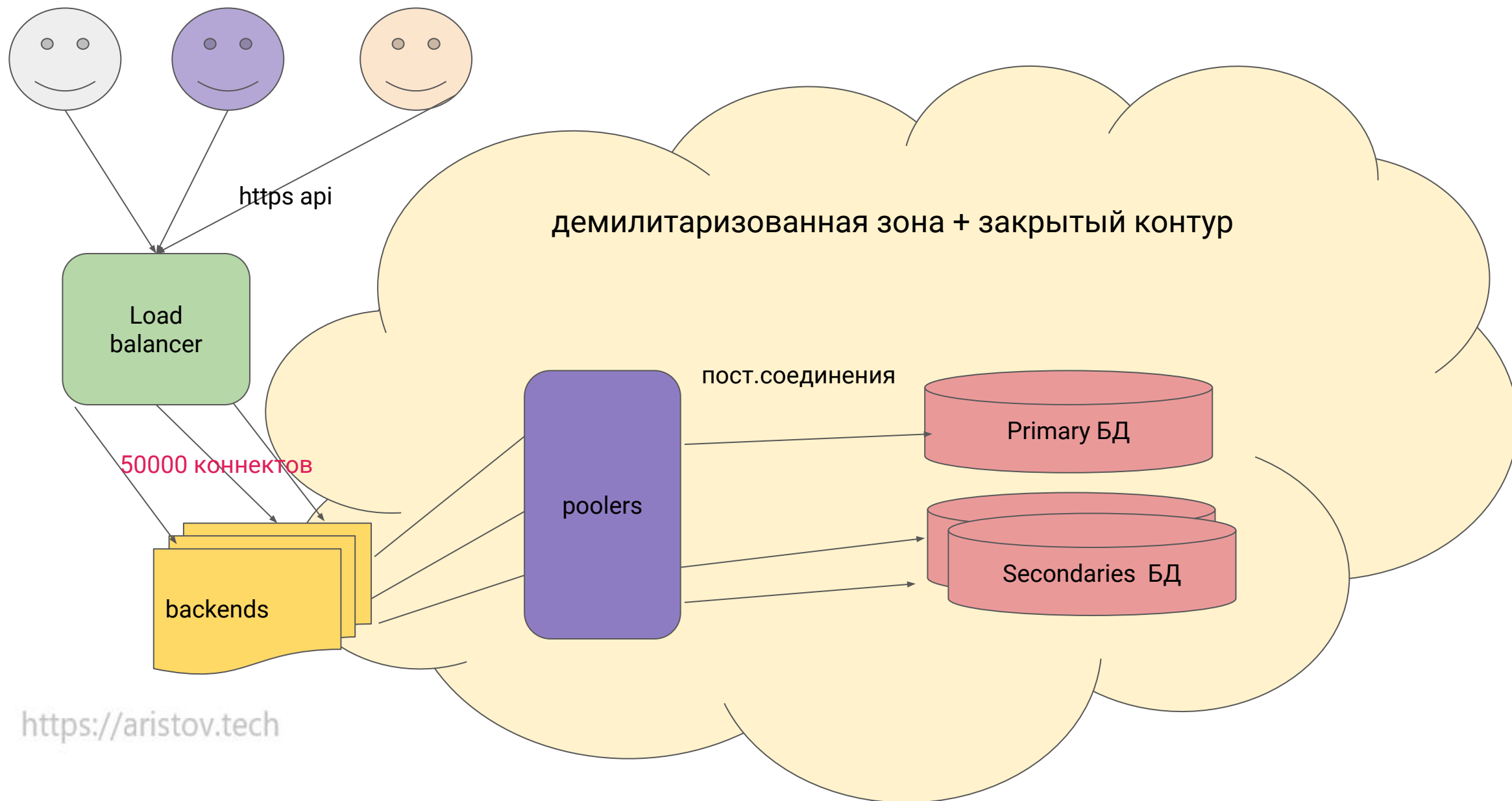
Особенности синхронной репликации:

synchronous_commit Modes				
synchronous_commit setting	local durable commit	standby durable commit after PG crash	standby durable commit after OS crash	standby query consistency
remote_apply	•	•	•	•
on	•	•	•	
remote_write	•	•		
local	•			
off				

https://postgresqlco.nf/doc/en/param/synchronous_commit/

Ну теперь то все хорошо?

<https://aristov.tech>

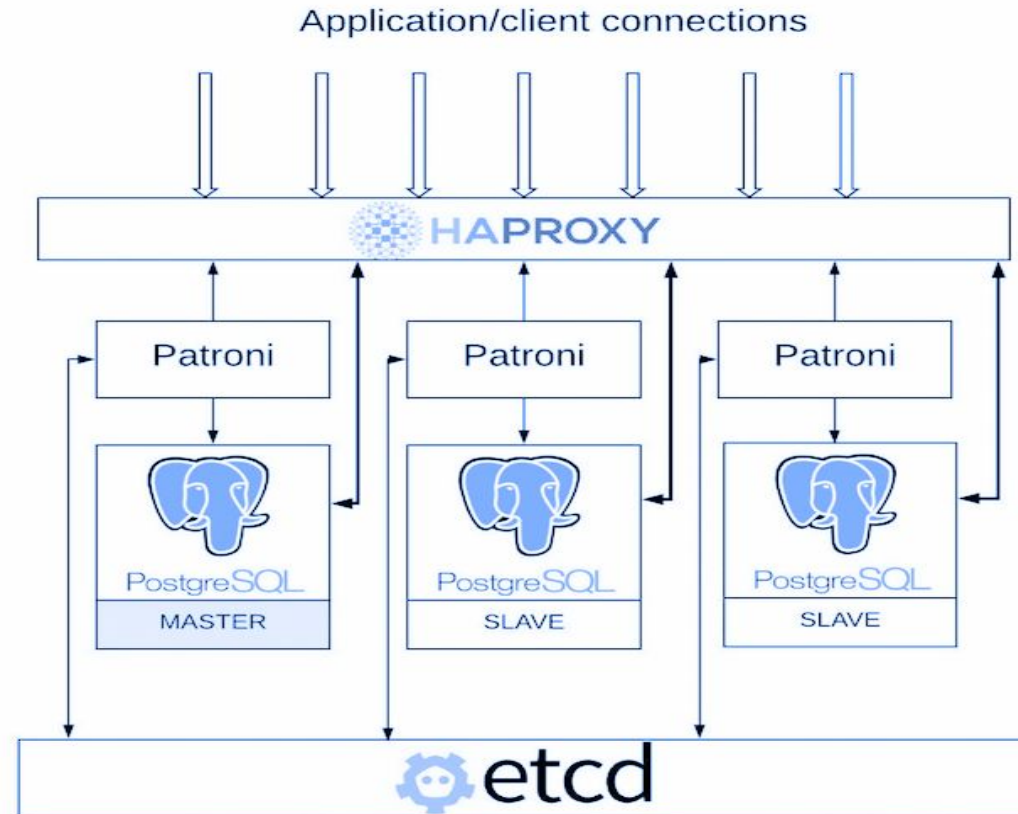


<https://aristov.tech>

Patroni

<https://aristov.tech>

А кто будет переключать мастера при падении?



<https://aristov.tech>

можно еще добавить pgbouncer, keepalived+2 HAProxy для HA

Что делать с OLAP

<https://aristov.tech>

- ❖ промежуточные агрегаты, избыточность
- ❖ материализованные представления

... может быть использование Хранимых Процедур?

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

... может быть готовые OLAP решения (Pentaho, GreenPlum, ArenaData, ClickHouse, [Citus](#))?

... может есть смысл fdw? cstore? pgmemcache? timescaledb?

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

<https://aristov.tech>

CRDB

CockroachDB. Фичи

<https://aristov.tech>

Основатели [Google Spanner](#) создали свой продукт - <https://www.cockroachlabs.com/>

Trusted by enterprises for mission-critical use cases

- ❖ Scale to meet demand
- ❖ SQL that scales horizontally - ACID-compliant transactions
- ❖ Built for transaction-heavy (OLTP) workloads
- ❖ Highly available by design
- ❖ Keep mission-critical applications online
- ❖ No database downtime
- ❖ Multi-active availability
- ❖ Multi-region, multi-cloud deployments
- ❖ Put data in its place
- ❖ Pricing
- ❖ <https://www.cockroachlabs.com/pricing/>
- ❖ <https://github.com/cockroachdb/cockroach>

<https://aristov.tech>

CockroachDB

<https://aristov.tech>

- ❖ Архитектура - протокол PostgreSQL, под капотом KV

<https://www.cockroachlabs.com/docs/stable/architecture/overview.html>

- ❖ Обработка параллельных транзакций - в т.ч. пишущих

<https://www.cockroachlabs.com/blog/local-and-distributed-processing-in-cockroachdb/>

- ❖ Github

<https://github.com/cockroachdb/cockroach>

- ❖ Сравнение с другими продуктами

<https://www.cockroachlabs.com/compare/>

Итого - Постгрес на максималках - комбайн, но крутилок не так много

<https://aristov.tech>

MongoDB

- ❖ **Кроссплатформенность.** СУБД разработана на языке программирования C++, поэтому с легкостью интегрируется под любую операционную систему (Windows, Linux, MacOS и др.).
- ❖ **Формат данных.** MongoDB использует собственный формат хранения информации — Binary JavaScript Object Notation (BSON), который построен на основе языка JavaScript.
- ❖ **Документ.** Если реляционные БД используют строки, то MongoDB — документы, которые хранят значения и ключи
- ❖ Вместо таблиц MongoDB использует **коллекции**. Они содержат разные типы наборов данных
- ❖ **Индексация.** Технология применяется к любому полю в документе на усмотрение пользователя

- ❖ **Репликация.** Система хранения информации в СУБД представлена узлами. Существует один главный и множество вторичных. Данные реплицируются между точками. Если один первичный узел выходит из строя, то вторичный становится главным.
- ❖ **Шардирование.** Распределение их частей по различным узлам кластера. Благодаря тому, что каждый узел кластера может принимать запросы, обеспечивается балансировка нагрузки.
- ❖ **Map Reduce framework** - ускорение при работе с данными на шардированных кластерах. Каждая нода отдельно обрабатывает запрос, потом данные объединяются
- ❖ Для сохранения данных большого размера MongoDB использует собственную технологию **GridFS**, состоящую из двух коллекций. В первой (files) содержатся имена файлов и метаданные по ним. Вторая (chunks) сохраняет сегменты информации, размер которых не превышает 256 Кб.

Mongo. Примеры

<https://aristov.tech>

- ❖ хранение и регистрация событий
- ❖ системы управления документами и контентом
- ❖ электронная коммерция
- ❖ игры
- ❖ данные мониторинга, датчиков
- ❖ мобильные приложения
- ❖ хранилище оперативных данных веб-страниц (например, хранение комментариев, рейтингов, профилей пользователей, сеансы пользователей)

<https://aristov.tech>

Mongo. Утечки информации

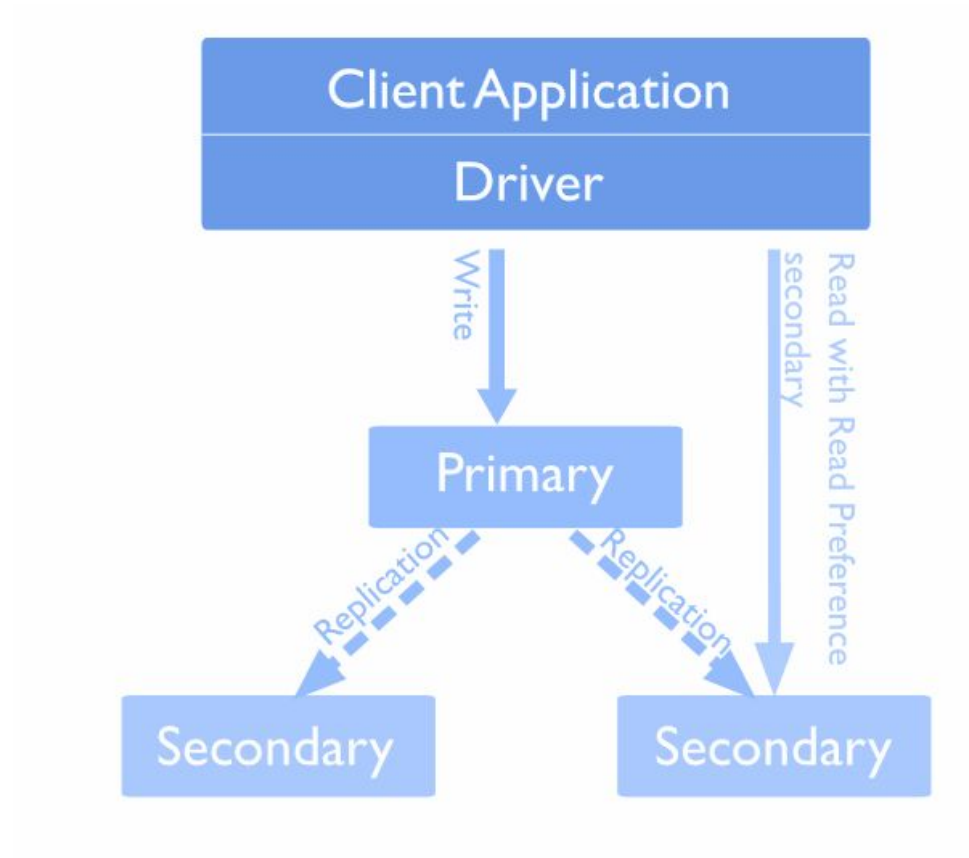
<https://aristov.tech>

- ❖ <https://securitydiscovery.com/800-million-emails-leaked-online-by-email-verification-service/>
- ❖ <https://www.opennet.ru/opennews/art.shtml?num=49259>

<https://aristov.tech>

Read from secondary

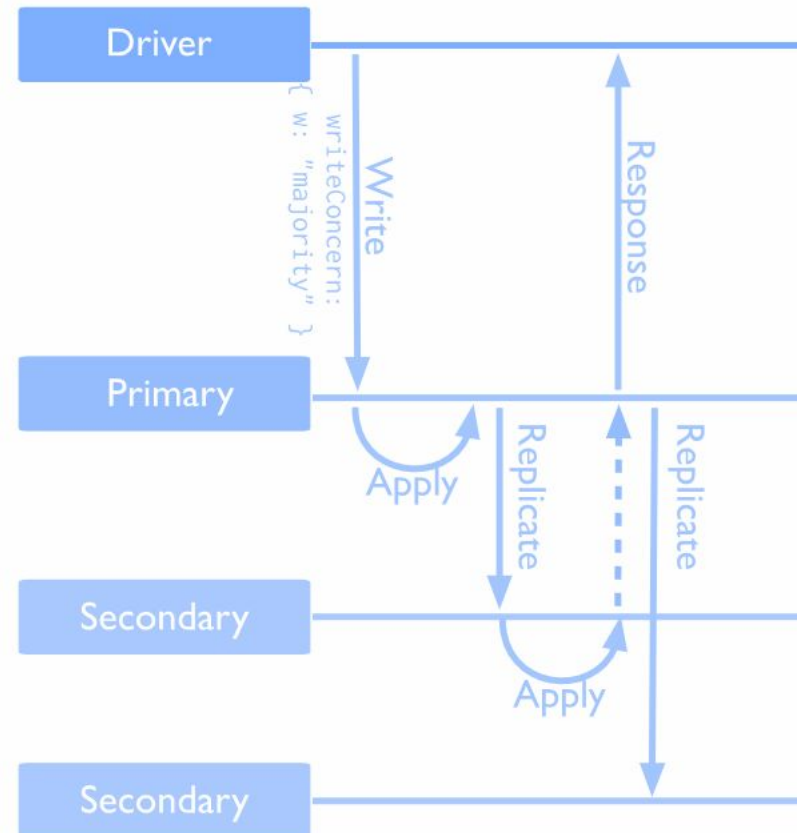
<https://aristov.tech>



<https://aristov.tech>

WriteConcern/ Majority

<https://aristov.tech>



<https://docs.mongodb.com/manual/core/replica-set-write-concern/>

<https://aristov.tech>

Масштабирование

<https://aristov.tech>

Можем комбинировать чтение/запись.

1. подтверждение записи большинством голосов и журналирование {w:majority, j:true} - максимальная консистентность на чтение, максимальная отказоустойчивость
2. отключаем подтверждение записи большинством голосов {w:1, j:true}
3. отключаем журналирование {w:1, j:false}
4. отключаем любые проверки {w:0, j:false} - максимальная скорость, можем потерять несколько транзакций при краше
5. что дальше?

[Write Concern – MongoDB Manual](#)

[Journaling – MongoDB Manual](#)

[Монго на вырост](#)

<https://aristov.tech>

Sharding. Минусы

<https://aristov.tech>

- ❖ **С шардированием обязательно будет усложняться код** — во многие места придется протащить ключ шардирования. Это не всегда удобно, и не всегда возможно. Некоторые запросы пойдут либо broadcast, либо multicast, что тоже не добавляет масштабируемости. Подходите к выбору ключа по которому пойдет шардирование аккуратнее.
- ❖ **В шардированных коллекциях может сломаться операция count** - начинает возвращать число больше, чем в действительности — может соврать по размеру мигрируемых чанков. Причина лежит в процессе балансировки, когда документы переливаются с одного шарда на другой. Когда документы перелились на соседний шард, а на исходном еще не удалились — count их все равно посчитает.
- ❖ **Значительный рост потребляемых ресурсов** - для минимальной отказоустойчивой работы необходимо 11 инсталляций Монго. Да их можно разложить на одни и те же хосты - но ресурсы то будем делить.
- ❖ **Шардированный кластер гораздо тяжелее в администрировании** - процедура снятия бэкапа становится радикально сложнее. Необходимо большая автоматизация работы.

<https://aristov.tech>

Шардирование. Проблемы

<https://aristov.tech>

При включении шардинга у нас есть 2 основные группы проблем:

1. Шардинг коллекции не возможен, если:

- ❖ требуется обновление полей, входящих в ключ шардирования
- ❖ на коллекции есть несколько уникальных ключей
- ❖ под результат запроса findAndModify попадают данные на разных шардах

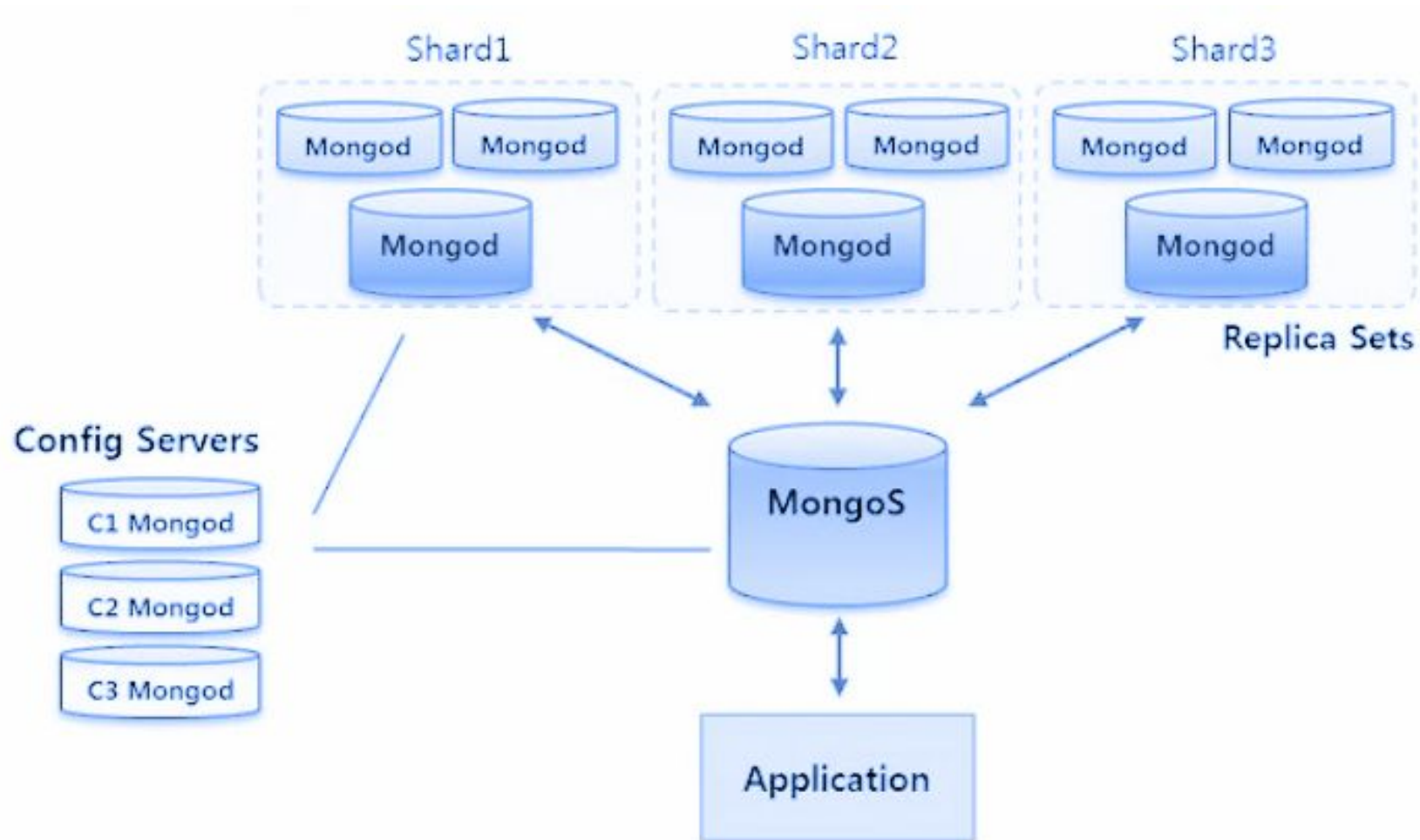
2. Несбалансированная нагрузка по шардам, если:

- ❖ слабая селективность ключа шардирования
- ❖ запросы без значений **ключа шардирования**

<https://aristov.tech>

Шардирование. Архитектура

<https://aristov.tech>



<https://aristov.tech>

ClickHouse

ClickHouse. Зачем?

<https://aristov.tech>

- ❖ Данных очень много – скорость их обработки штатными средствами не устраивает
- ❖ Нужно получать агрегированную информацию по этим данным
- ❖ Нужны оперативные и исторические данные – действительно Online
- ❖ Данные нужны на диске
- ❖ <https://clickhouse.com/docs/ru/faq/general/why-clickhouse-is-so-fast>
- ❖ <https://clickhouse.com/use-cases>
- ❖ <https://clickhouse.com/docs/ru/introduction/distinctive-features>

Плюсы:

- ❖ Жутко быстрые OLAP запросы
- ❖ Данные хранятся на диске
- ❖ Сжатие 2++ раза
- ❖ SQL
- ❖ Кластеризация/репликация

Минусы:

- ❖ Нет транзакций
- ❖ Очень медленные & Delete - это не DML, а DDL - мутации
- ❖ Медленные одиночные Insert & Select - пишем только пачками

<https://aristov.tech>

Столбцовые СУБД

<https://aristov.tech>



ORACLE®



<https://aristov.tech>

ClickHouse. Архитектура

<https://aristov.tech>

ClickHouse - столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP).

В обычной, «строковой» СУБД, данные хранятся в таком порядке:

Строка	WatchID	JavaEnable	Title	GoodEvent	EventTime
#0	89354350662	1	Investor Relations	1	2016-05-18 05:19:20
#1	90329509958	0	Contact us	1	2016-05-18 08:10:20
#2	89953706054	1	Mission	1	2016-05-18 07:38:00
#N

То есть, значения, относящиеся к одной строке, физически хранятся рядом.

Примеры строковых СУБД: MySQL, Postgres, MS SQL Server.

В столбцовых СУБД, данные хранятся в таком порядке:

Строка:	#0	#1	#2	#N
WatchID:	89354350662	90329509958	89953706054	...
JavaEnable:	1	0	1	...
Title:	Investor Relations	Contact us	Mission	...
GoodEvent:	1	1	1	...
EventTime:	2016-05-18 05:19:20	2016-05-18 08:10:20	2016-05-18 07:38:00	...

В примерах изображён только порядок расположения данных.

То есть, значения из разных столбцов хранятся отдельно, а данные одного столбца - вместе.

Плюсы колонок:

- ❖ Блок меньше размером
- ❖ Проще сжимается
- ❖ Вертикальное секционирование

Плюсы строк:

- ❖ Быстрее одиночная запись

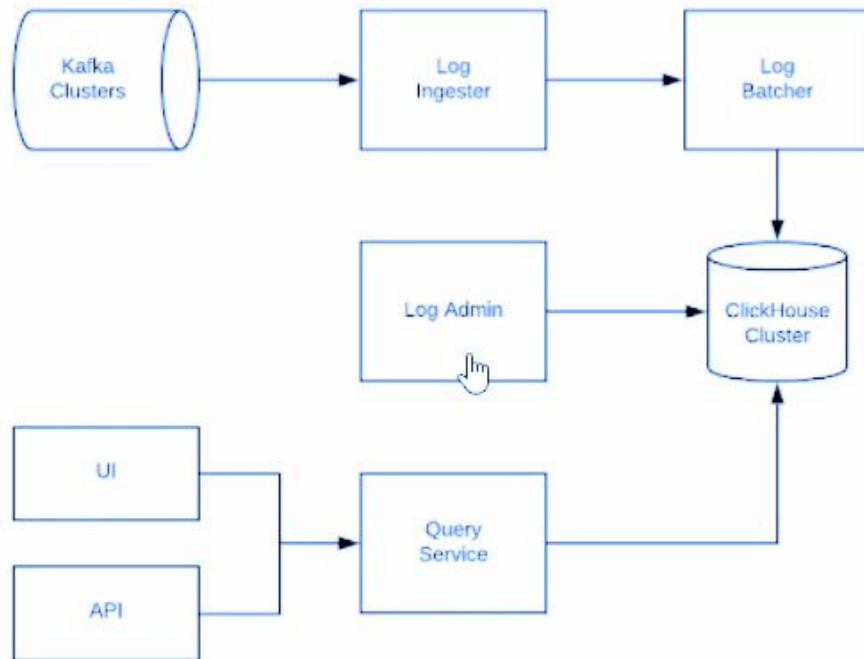
<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>

Uber

Uber Engineering



1. **Log schema:** Our logs are semi-structured. ES (Elasticsearch) infers schemas automatically, keeps it consistent across the cluster, and enforces it on following logs. Incompatible field types cause type conflict error in ES, which drops the offending logs. While enforcing a consistent schema is reasonable for business events with well-defined structures, for logging this leads to significant reduction in developer productivity because log schema organically evolves over time. For example, a large API platform can be updated by hundreds of engineers, accumulating thousands of fields in the ES index mapping and there is a high chance for different engineers to use the same field name, but with different types in field values, which leads to type conflicts in ES. Forcing engineers to learn about the existing schema and maintain its consistency just to print some service logs is inefficient. Ideally, the platform should treat schema changes as a norm and be able to handle fields with multiple types for the users.

2. **Operational cost:** We had to run 20+ ES clusters in each region to limit the blast radius if a cluster was affected negatively by heavy queries or mapping explosions. The number of Logstash pipelines was even higher with 50+ per region to accommodate special use cases and custom configurations. Both expensive queries and mapping explosions could degrade the ES cluster severely and sometimes even "freeze" it, when we had to restart the cluster to bring it back. As the number of clusters increased, this kind of outage grew more frequent. Despite our best efforts to automate processes such as detecting and disabling fields that would cause mapping explosions and type conflicts, rebalancing traffic among ES clusters, and so on, manual intervention for resolving issues like type conflicts was still inevitable. We would like a platform that is able to support our organization's large scale and new use cases without incurring as much operational overhead.

3. **Hardware Cost:** Indexing fields is pretty costly in ES, because it needs to build and maintain sophisticated inverted indexing and forward indexing structures, write it to transaction log, flush the in-memory buffer periodically to disk, and perform regular background merges to keep the number of flushed index segments from growing unbounded, all of which requires significant processing power and increases the latency for logs to become queryable. So instead of indexing all fields in the logs, our ES clusters were configured to index those up to three levels. But ingesting all of the generated logs still consumed a lot of hardware resources and was too expensive to scale.

4. **Aggregation:** As found in our production environment, more than 80% queries are aggregation queries, such as terms, histogram and percentile aggregations. While ES has made improvements to optimize forward indexing structures, it is nonetheless not designed to support fast aggregations across large datasets. The performance inefficiency led to an unpleasant user experience. For example, the critical dashboards for a service with large log volume loaded very slowly when querying last 1h logs (around 1.3TB) and frequently timed out when querying last 6h logs, making it impossible to diagnose production issues.

<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>



[Nginx-log-collector утилита от Авито для отправки логов nginx в Clickhouse](#)

<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>

https://nastachku.ru/var/files/1/presentation/backend/2_Backend_6.pdf



20Тб логов в ClickHouse

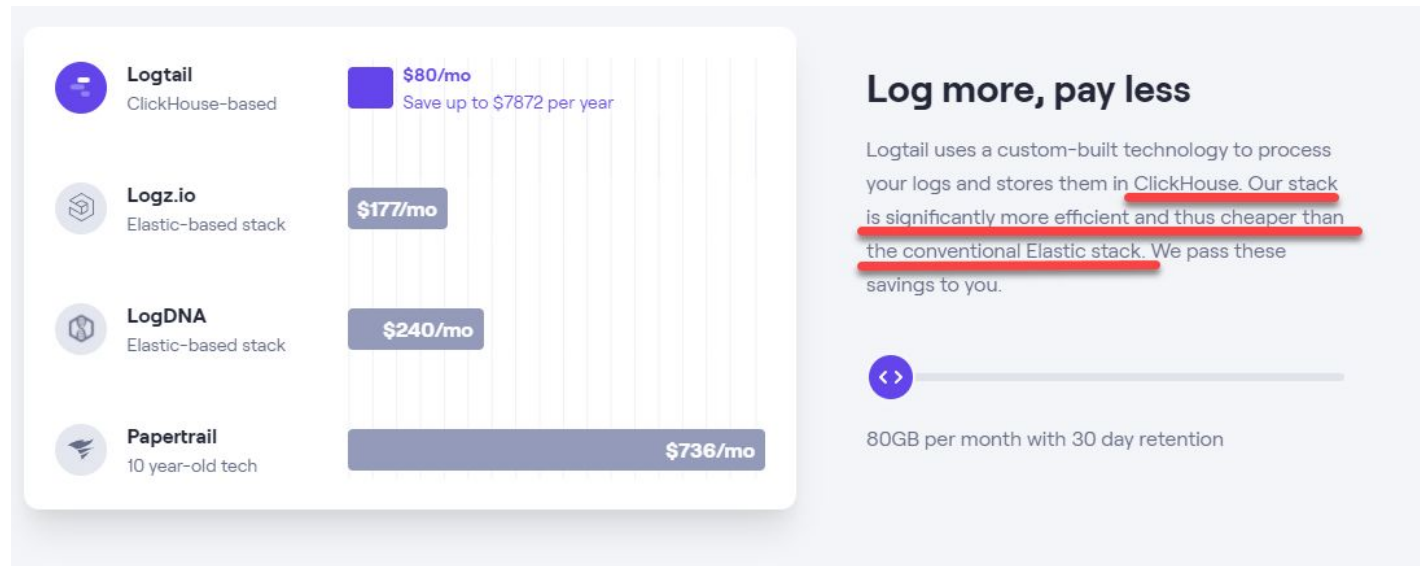
Василий Васильков, Ecwid

<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>

<https://logtail.com/>

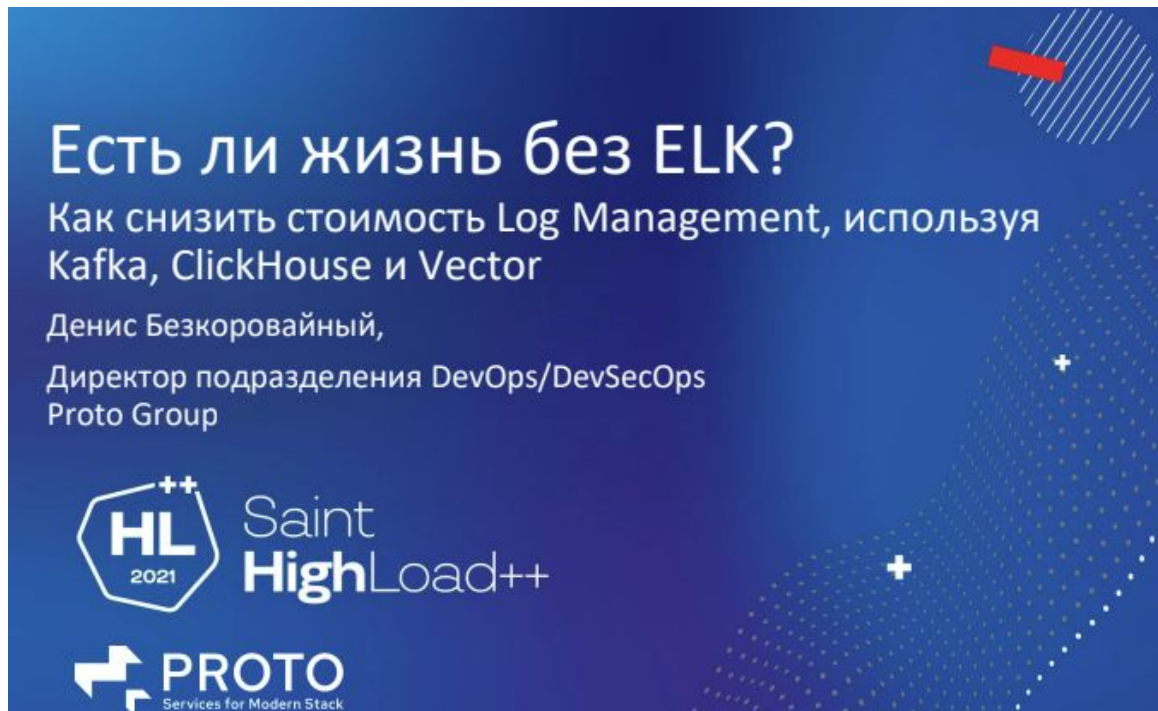


<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>

<https://drive.google.com/file/d/1nrZccTZziCcQaLsu-vae9H4ZwbT8qTfR/view>



Есть ли жизнь без ELK?
Как снизить стоимость Log Management, используя
Kafka, ClickHouse и Vector

Денис Безкоровайный,
Директор подразделения DevOps/DevSecOps
Proto Group

HL 2021 Saint HighLoad++

PROTO
Services for Modern Stack

К чему пришли – итоги

Экономия ресурсов на «железо»

- Экономия места за счет компрессии около 7x
- 120 ГБ логов в сутки примерно равно 520 ГБ за месяц в ClickHouse (вместо 8000 ГБ в Elasticsearch)
Нужно меньше RAM

Быстро!

- Нет долгой индексации
- Данные доступны сразу

Все это можно попробовать самому!

- <https://vector.dev/>
- <https://github.com/Vertamedia/clickhouse-grafana/>

<https://aristov.tech>

ClickHouse. Почему логи?

<https://aristov.tech>

Так почему столько компаний хранят Логи в ClickHouse?

В современном мире очень часто логи собираются не только для поиска по ним источников проблем.

Для поиска ClickHouse плохо подходит в целом – это не замена `grep`, `awk`, `cut`, `tail` (в основном расследование инцидентов)

По логам (особенно `nginx`) собираются метрики, и вот для метрик уже ClickHouse подходит как никогда лучше:

- Сколько было ошибок
- Сколько пользователей не аутентифицировалось
- Сколько мы потеряли пользователей из за падения сервиса
- Какой процент деградации производительности был из за тормозов эластика...
- прочие

<https://aristov.tech>

ClickHouse. Сценарии. Логи

<https://aristov.tech>

Замена ELK - <https://habr.com/ru/company/flant/blog/341386/>



414,47

Рейтинг

Флант

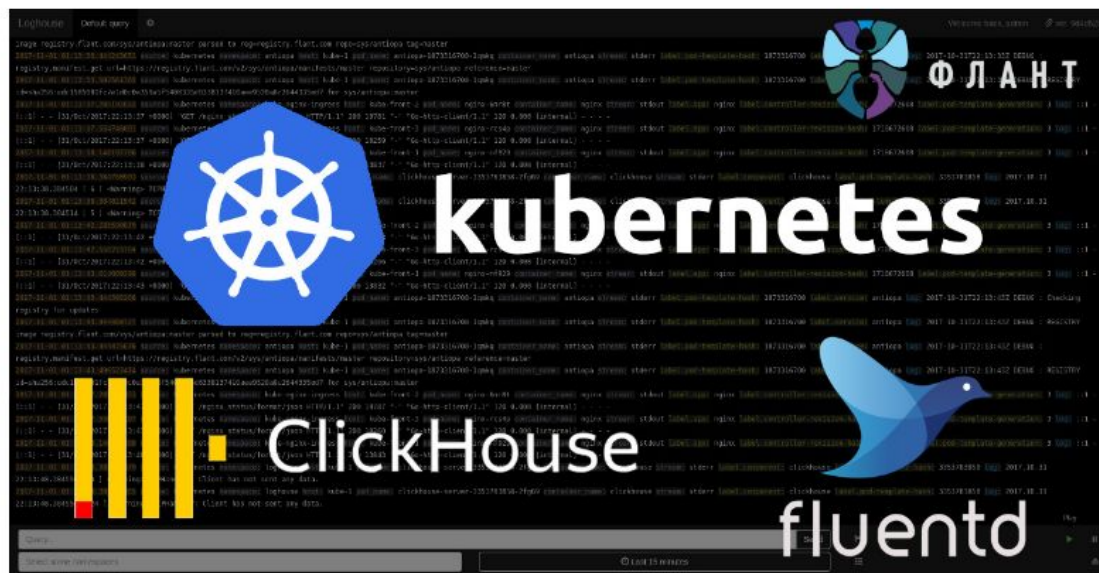
DevOps-as-a-Service, Kubernetes, обслуживание 24x7



Wimbo 1 ноября 2017 в 11:04

Представляем loghouse — Open Source-систему для работы с логами в Kubernetes

Блог компании Флант, Open source, IT-инфраструктура, DevOps, Kubernetes



<https://aristov.tech>

Вставка данных

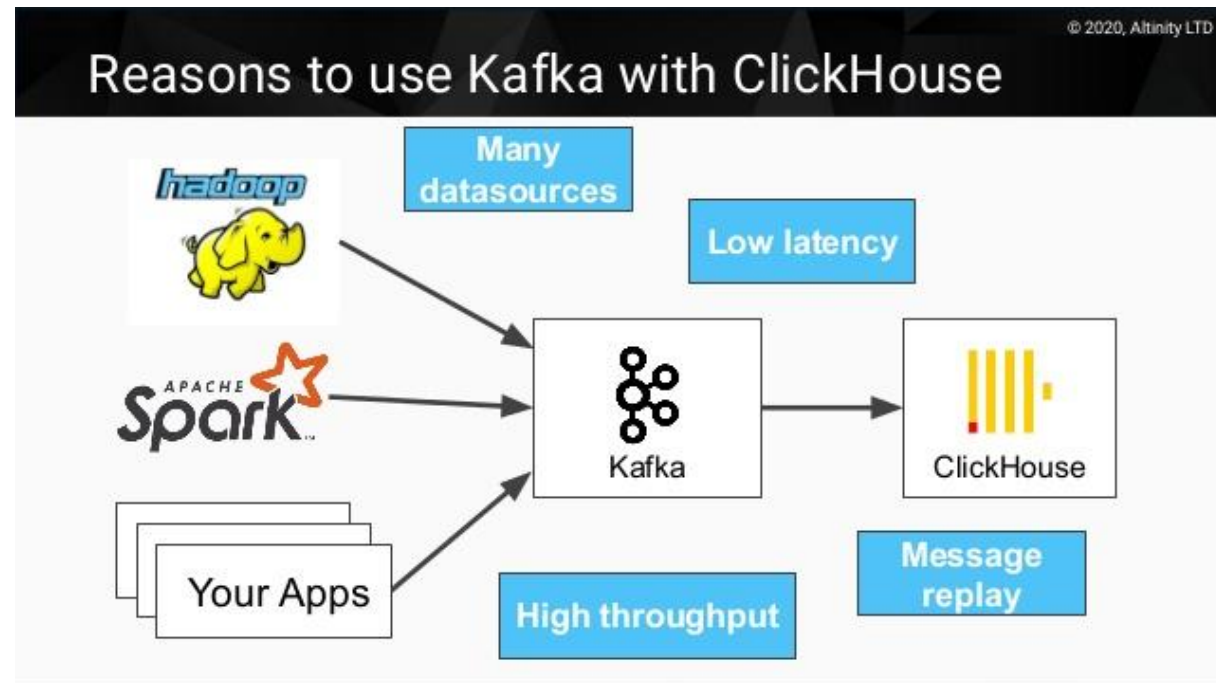
<https://aristov.tech>

Особенности:

- ❖ вставка больших кусков данных за раз (1 млн строк)
- ❖ чем реже - тем лучше (хотя бы раз в секунду)

Как достигнуть:

- ❖ на стороне приложения
- ❖ open source решения
 - kittenhouse
 - clickhouse-bulk
- ❖ buffer table
- ❖ **kafka engine**



<https://aristov.tech>

Elasticsearch

- ❖ Предназначен для полнотекстового поиска
- ❖ ELK стек
Особенности:
- ❖ Поиск в Elastic состоит происходит согласно условиям выборки в двух контекстах — Filter и Query.
 - Filter — отвечает на вопрос “подходит ли документ под условия поиска”
 - Query — “насколько хорошо документ подходит под условия поиска”. Ранжирует документы по релевантности
- ❖ поддерживает сложные агрегации
 - с поминутной разбивкой и подсчетом кол-ва документов
 - вычислением максимального значения поля на заданном диапазоне
- ❖ СТОИМОСТЬ
 - <https://www.elastic.co/pricing>
 - community <https://github.com/elastic/elasticsearch>

Elasticsearch

<https://aristov.tech>

- ❖ **Написан на Java**
- ❖ Синхронизация на Raft
- ❖ Плюсы - из коробки:
 - Платформонезависимость + Масштабируемость
 - Отказоустойчивость
- ❖ Минусы
 - Любит оочень много памяти
 - гео распределенные базы работают плохо
- ❖ Что делать?
 - Оставлять много памяти под page cache
 - Делать много шардов

<https://aristov.tech>

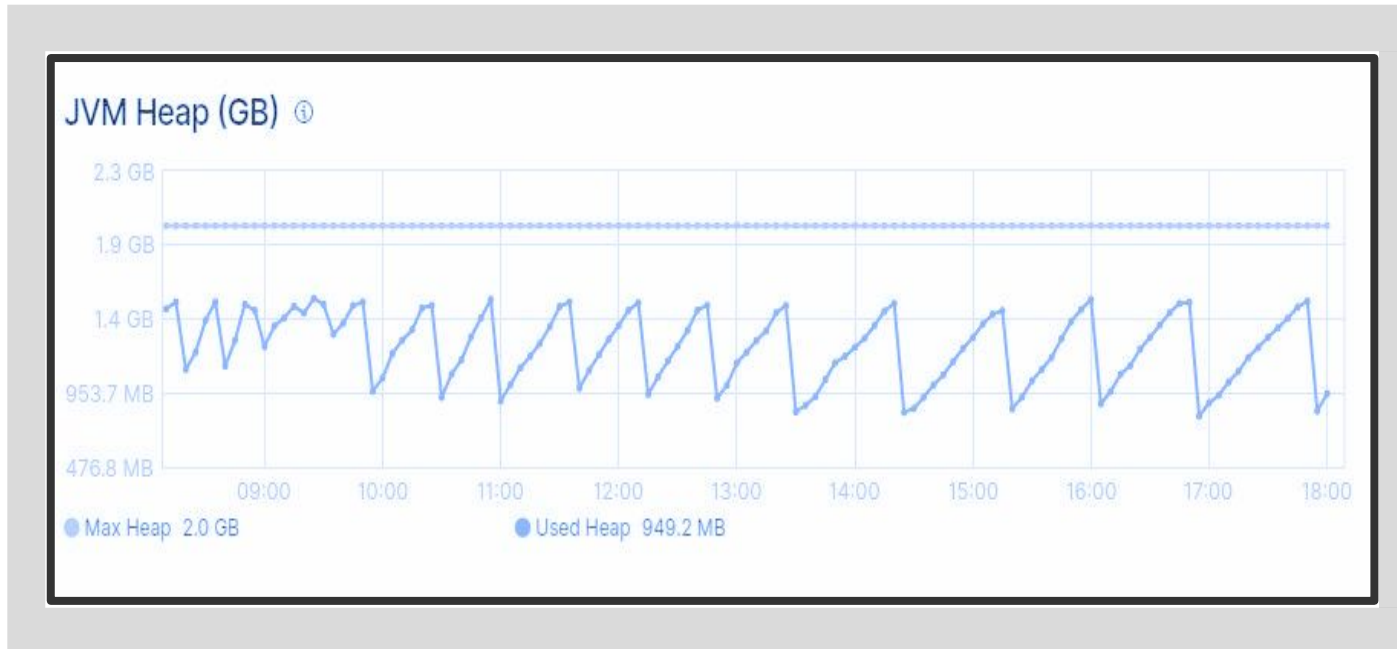
Elasticsearch. Профили нагрузок

<https://aristov.tech>

- ❖ Редкая запись/обновление и активное чтение
 - поисковые индексы для порталов
- ❖ Активная запись - редкое чтение
 - сбор логов и метрик (как мы видели Clockhouse лучше в большинстве сценариев)

Elasticsearch. Мониторинг памяти

<https://aristov.tech>



Garbage collector + generations

<https://aristov.tech>

Elasticsearch - так ли всё гуд?

<https://aristov.tech>

Вроде бы всё понятно, но есть альтернативы:

- ❖ [Opensearch](#)
- ❖ [Mantikore search](#)

<https://aristov.tech>

OpenSearch — поисковая и аналитическая система с открытым исходным кодом в облаке | Yandex Cloud

- ❖ функциональность машинного обучения
- ❖ быстрый поиск по векторам с помощью k-NN (k-Nearest Neighbor)
- ❖ управление безопасностью, индексами
- ❖ поиск и обнаружение аномалий
- ❖ поддержка SQL
- ❖ генерация уведомлений
- ❖ диагностика производительности кластера
- ❖ шифрование трафика
- ❖ разграничение доступа на основе ролей (RBAC)
- ❖ аутентификация через SAML и другие внешние источники авторизации
- ❖ реализация единой точки входа (SSO)
- ❖ ведение детального лога для аудита.

Manticore Search. История

<https://aristov.tech>

- ❖ 2001 - Старт разработки Sphinx (SQL Phrase Index).
- ❖ 2004 - Первая версия [Solr](#)
- ❖ 2010 - Sphinx является популярным полнотекстовым движком, его используют Хабрахабр, Викимапия и другие. Первый выпуск Elasticsearch
- ❖ 2017 - Код новой версии Sphinx 3 становится закрытым. Ключевые сотрудники Sphinx создают форк под названием Manticore Search, так как развитие приостановилось — Elasticsearch давно обогнал Sphinx

<https://aristov.tech>

Manticore Search. Возможности

<https://aristov.tech>

- ❖ Полнотекстовый, фасетный и гео поиск
- ❖ Можно использовать без полнотекстового поиска
- ❖ Построчное и колоночное хранение данных
- ❖ MPP архитектура и распараллеливания запросов
- ❖ Синтаксис SQL (клиент mysql), SQL over HTTP, JSON over HTTP
- ❖ Декларативное и императивное управление схемой
- ❖ Изолированные транзакции для атомарных изменений
- ❖ Ведение двоичного журнала для безопасной записи
- ❖ Репликация (galera cluster) и балансировка нагрузки из коробки
- ❖ Percolate индексы (обратный поиск)
- ❖ Кеширование в RAM и малое потребление памяти
- ❖ И другое...

<https://aristov.tech>

Manticore Search. Применение

<https://aristov.tech>

- ❖ Полнотекстовый поиск, например, электронная библиотека
- ❖ Фасетный поиск — интернет-магазин
- ❖ Геопространственный поиск — карты
- ❖ Онлайн обработка аналитических запросов (OLAP)
- ❖ Автозаполнение - фильтрация потока данных

<https://aristov.tech>

Manticore Search. Тестирование

<https://aristov.tech>

- ❖ **ClickHouse** 22.7.5.13 vs **Elasticsearch** 7.17.6 vs **Manticore Search** 5.0.2.
- ❖ [Hacker News Curated Comments Dataset](#), 1 165 439 rows.
- ❖ Размер индекса ClickHouse = 454.555 MB.
- ❖ Размер индекса Elasticsearch = 911.613 MB.
- ❖ Размер индекса Manticore Search = 1037.85 MB.
- ❖ Аналитические запросы и полнотекстовой поиск.

<https://aristov.tech>

Manticore Search. Тестирование. Результаты

<https://aristov.tech>

Query	clickhouse Fast avg	elasticsearch Fast avg	manticoresearch Fast avg
select * from hn_small order by comment_ranking asc limit 20	x69.5 (417 ms)	x3.83 (23 ms)	6 ms
select * from hn_small order by comment_ranking asc, story_id asc limit 20	x71 (426 ms)	x4.33 (26 ms)	6 ms
select * from hn_small order by comment_ranking desc limit 20	x70.5 (423 ms)	x3.83 (23 ms)	6 ms
select * from hn_small where match('elon musk') limit 20	x27 (81 ms)	x4 (12 ms)	3 ms
select * from hn_small where match('abc -google') limit 20	x23.75 (95 ms)	x3.75 (15 ms)	4 ms
select * from hn_small where match('abc') limit 20	x30.33 (91 ms)	x4.33 (13 ms)	3 ms
select * from hn_small where match('abc') order by comment_ranking asc limit 20	x244 (732 ms)	x4.33 (13 ms)	3 ms
select * from hn_small where match('abc') order by comment_ranking asc, story_id desc limit 20	x250.33 (751 ms)	x4.33 (13 ms)	3 ms
select comment_ranking from hn_small order by comment_ranking asc limit 20	x2 (10 ms)	x4.6 (23 ms)	5 ms
select comment_ranking, avg(author_comment_count) avg from hn_small group by comment_ranking order by avg desc, comment_ranking desc limit 20	x3.3 (33 ms)	x17.4 (174 ms)	10 ms
select comment_ranking, avg(author_comment_count) avg from hn_small where match('google') and comment_ranking > 200 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x97.5 (390 ms)	x4.75 (19 ms)	4 ms

<https://aristov.tech>

Manticore Search. Тестирование. Результаты

<https://aristov.tech>

Query	clickhouse	elasticsearch	manticoresearch
	Fast avg	Fast avg	Fast avg
select comment_ranking, avg(author_comment_count) avg from hn_small where match('google') group by comment_ranking order by avg desc, comment_ranking desc limit 20	x126.83 (761 ms)	x4.17 (25 ms)	6 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small group by comment_ranking order by avg desc, comment_ranking desc limit 20	x3.45 (38 ms)	x28.45 (313 ms)	11 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small where comment_ranking < 10 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x4.11 (37 ms)	x21 (189 ms)	9 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small where match('google') and comment_ranking > 200 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x100.25 (401 ms)	x5 (20 ms)	4 ms
select comment_ranking, count(*) from hn_small group by comment_ranking order by count(*) desc limit 20	x4 (24 ms)	x15.17 (91 ms)	6 ms
select comment_ranking, story_text from hn_small order by comment_ranking asc limit 20	x11.33 (68 ms)	x3.83 (23 ms)	6 ms
select count(*) from hn_small	x3.75 (15 ms)	x2 (8 ms)	4 ms
select count(*) from hn_small where comment_ranking > 300 and comment_ranking < 500	x2.56 (23 ms)	9 ms	x1.56 (14 ms)
select count(*) from hn_small where comment_ranking in (100,200)	x7.67 (23 ms)	x3.33 (10 ms)	3 ms

<https://aristov.tech>

Manticore Search. Тестирование. Результаты

<https://aristov.tech>

Query	clickhouse	elasticsearch	manticoresearch
	Fast avg	Fast avg	Fast avg
select count(*) from hn_small where comment_ranking=100	x7 (21 ms)	x3 (9 ms)	3 ms
select count(*) from hn_small where comment_ranking=500	x11.5 (23 ms)	x4.5 (9 ms)	2 ms
select count(*) from hn_small where match('google') and comment_ranking > 200	x95.25 (381 ms)	x3 (12 ms)	4 ms
select story_author, avg(comment_ranking) avg from hn_small group by story_author order by avg desc limit 20	x1.18 (40 ms)	x128.09 (4355 ms)	34 ms
select story_author, count(*) from hn_small group by story_author order by count(*) desc limit 20	33 ms	x124.55 (4110 ms)	33 ms
select story_id from hn_small order by comment_ranking asc, author_comment_count asc, story_comment_count asc, comment_id asc limit 20	x7 (35 ms)	x5.4 (27 ms)	5 ms
select story_id from hn_small where match('me') order by comment_ranking asc limit 20	x87.75 (702 ms)	x2.25 (18 ms)	8 ms
select story_id, comment_id, comment_ranking, author_comment_count, story_comment_count, story_author, comment_author from hn_small where match('abc') limit 20	x49 (98 ms)	x6.5 (13 ms)	2 ms

<https://aristov.tech>

Итоги основной части

Основные фичи

<https://aristov.tech>

	PG	CRDB	MDB	CH	ES
community	+	+-	+	+	+-
Поддержка	+	-	-	-	-
Cloud/Dedic	+	+	+	+	+
Own cloud	-	+	+	-	-
Cluster	-	+	+	+	+
Перспективы	+	+	+-	+	-
OLTP	+	+	+	-	+-
OLAP	-	+	+(shard)	+	+-
Полнотекст	+-	+-	+-	+-	+
JSON	+-	+-	+	-	+
JOIN	+	+	-	-	-
Назначение	ACID	parallel ACID	document	column	textsearch

<https://aristov.tech>

Бонус

<https://aristov.tech>

Если нужно сравнить функционал и скорость разных СУБД:

<https://db-benchmarks.com>

<https://benchmark.clickhouse.com/>

All AlloyDB Athena (partitioned) Athena (single) Aurora for MySQL Aurora for PostgreSQL ByConity ByteHouse chDB Citus ClickHouse Cloud (aws) ClickHouse Cloud (aws) Parallel Replicas ON ClickHouse Cloud (Azure) ClickHouse Cloud (Azure) Parallel Replica ON ClickHouse Cloud (Azure) Parallel Replicas ON ClickHouse Cloud (gcp) ClickHouse Cloud (gcp) Parallel Replicas ON ClickHouse (data lake, partitioned) ClickHouse (data lake, single) ClickHouse (Parquet, partitioned) ClickHouse (Parquet, single) ClickHouse (web) ClickHouse ClickHouse (tuned) ClickHouse (tuned, memory) CrateDB Databend DataFusion (Parquet, partitioned) DataFusion (Parquet, single) Apache Doris Druid DuckDB (Parquet, partitioned) DuckDB Elasticsearch Elasticsearch (tuned) GlareDB Greenplum HeavyAI Hydra Infobright Kinetica MariaDB ColumnStore MariaDB MonetDB MongoDB Motherduck MySQL (MyISAM) MySQL Oxa ParadeDB Pinot PostgreSQL (tuned) PostgreSQL QuestDB (partitioned) QuestDB Redshift SelectDB SingleStore Snowflake SQLite StarRocks Tablespace Tembo OLAP (columnar) TimescaleDB (compression) TimescaleDB Umbra

<https://aristov.tech>

Бонус

<https://aristov.tech>

Используем [реплику](#) ClickHouse для OLAP PostgreSQL

Если нужно переливать постоянно - [CDC](#), например [Debezium](#)

<https://aristov.tech>

Проект aristov.tech

aristov.tech

- ❖ Книга по 14 Постгресу (Архитектура) и 16 Постгресу (Оптимизация)

<https://aristov.tech/#orderbook>

- ❖ Эксклюзивный курс по Оптимизации Постгреса

<https://aristov.tech/blog/kurs-po-optimizaczii-postgresql/>

- ❖ Блог с популярными темами

<https://aristov.tech/blog>

- ❖ ТГ канал с новостями блога и проекта

https://t.me/aristov_tech

- ❖ Ютуб/Рутуб канал с интересными видео + [курс SQL с 0](#)

<https://www.youtube.com/@aristovtech> <https://rutube.ru/u/aristovtech/>

- ❖ Моя группа ДБА

<https://t.me/+S-twn4foKFAtUNil>

- ❖ *Всегда можно со мной связаться через сайт для консультация, аудита вашего проекта, личного менторинга, обучения и многого другого*

PostgreSQL 16 под капотом: архитектура и методы оптимизации

Оглавление

Об авторе	4
1. ПостгресQL 16. Настройка VM, ОС и БД	6
2. Коннектинг к ПостгресQL. Права пользователя	40
3. Настройка файловой системы.....	63
4. Настройка бэкапов и репликации	83
5. Мониторинг, профилирование и логирование	113
6. Тюнинг WAL & shared_buffers	140
7. Vacuum, background writer, work_mem, statistic collector, locks	164
8. Схема данных.....	188
9. Оптимизация запросов	222
10. Обслуживание СУБД. Работа на стенде.....	250
Заключение	274

aristov.tech

Розыгрыш книги и скидки на очный вариант курса по оптимизации
регистрируемся по форме:

<https://forms.gle/w4iPn1c3QsX4LRjq8>

Сам розыгрыш в db-fiddle

<https://www.db-fiddle.com/f/43wbuUzv7YUAcS3aypkvvC/1>

ДЗ

ДЗ

1. Подписаться на канал
2. Подписаться на Ютуб
3. Вступить в мою группу ДБА
4. Посещать открытые уроки
5. Прийти на курс %)

Спасибо за внимание!

<https://aristov.tech/>

Аристов Евгений