



# Онлайн-образование

# Меня хорошо видно && слышно?

Ставьте + , если все хорошо  
Напишите в чат, если есть проблемы  
заодно проверяем, включена ли запись занятия

*Включил Юджин запись ли ты*



# ClickHouse

Аристов Евгений

telegram @AEugene

<https://aristov.tech>

# Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в #канал группы



Вопросы вижу в чате, могу ответить не сразу



Колоночные СУБД



Использование ClickHouse



Основные возможности



Установка и практика

# Цели вебинара | После занятия вы сможете

1

Понять что вам нужен ClickHouse на вашем проекте (или посмеяться над тем кто его «притянул за уши»)

2

Развернуть СН для своего проекта

3

Использовать интеграционные и управляющие интерфейсы СН

# Смысл | Зачем вам это уметь

1

СН сегодня одна из самых быстрых СУБД с  
дисковым хранением

2

Объём аналитических задач вырастает в  
современных системах. Надо уметь с ним  
работать

Погнали



Learn  
ClickHouse

# Когда нужен СН

- 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>

# Когда нужен СН

ClickHouse   Vertical   Vertical (x3)   Vertical (x6)   InfiniDB   MonetDB   Infobright   Hive   MySQL   MemSQL   Greenplum   Greenplum(x2)   OmniSci

Dataset size  
10 mln.   100 mln.   1 bn.

Run  
first (cold cache)   second   third

Relative query processing time (lower is better)

System	1 bn.	second	third	1.00
ClickHouse	~0.25	~0.25	~0.25	1.00
Vertica	~1.5	~1.5	~1.5	4.30

Full results

Query	ClickHouse (19.1.6)			Vertica (7.1.1)	
SELECT count() FROM hits	×7.85 (0.747 s.)	(0.090 s.)	(0.072 s.)	(0.095 s.)	×1.05 (0.094 s.)   ×1.23 (0.092 s.)
SELECT count() FROM hits WHERE AdvEngine	(0.197 s.)	(0.074 s.)	(0.061 s.)	×4.14 (0.816 s.)	×6.03 (0.446 s.)   ×7.44 (0.454 s.)
SELECT sum(AdvEngineID), count(), avg(Resoli	(1.370 s.)	(0.410 s.)	(0.334 s.)	×1.54 (2.111 s.)	×4.23 (1.773 s.)   ×5.36 (1.791 s.)
SELECT sum(UserID) FROM hits	×1.84 (3.911 s.)	(0.380 s.)	(0.365 s.)	(2.320 s.)	×2.65 (1.033 s.)   ×2.80 (1.020 s.)
SELECT uniq(UserID) FROM hits	(4.096 s.)	(0.635 s.)	(0.638 s.)	×1.99 (8.150 s.)	×12.17 (7.741 s.)   ×12.01 (7.660 s.)
SELECT uniq(SearchPhrase) FROM hits	(3.739 s.)	(1.626 s.)	(1.542 s.)	×5.51 (26.119 s.)	×15.75 (25.611 s.)   ×16.65 (25.676 s.)
SELECT min(EventDate), max(EventDate) FRO	(0.270 s.)	(0.172 s.)	(0.168 s.)	×3.83 (1.035 s.)	×4.76 (0.818 s.)   ×4.37 (0.822 s.)

Быстрее некуда

<https://clickhouse.tech/benchmark/dbms>

# Когда нужен СН

Раньше: <https://clickhouse.tech/docs/ru/introduction/adopters/>

- Yandex
- Avito
- Badoo
- Cloudflare
- Cisco
- eBay
- Ivi
- Mail.ru
- Deutsche Bank

Теперь: <https://clickhouse.com/use-cases/>

# Когда нужен СН

## За что любят СН

- Жутко быстрые OLAP запросы
- Данные хранятся на диске
- C++
- SQL
- Кластеризация/репликация

## А за что не любят

- Нет транзакций
- Медленные & Delete
- Медленные одиночные Insert & Select

# Столбцовые СУБД



Google BigQuery



Hybrid columnar compression



Columnstore index



# Столбцовые СУБД

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.

## Столбец VS Стока

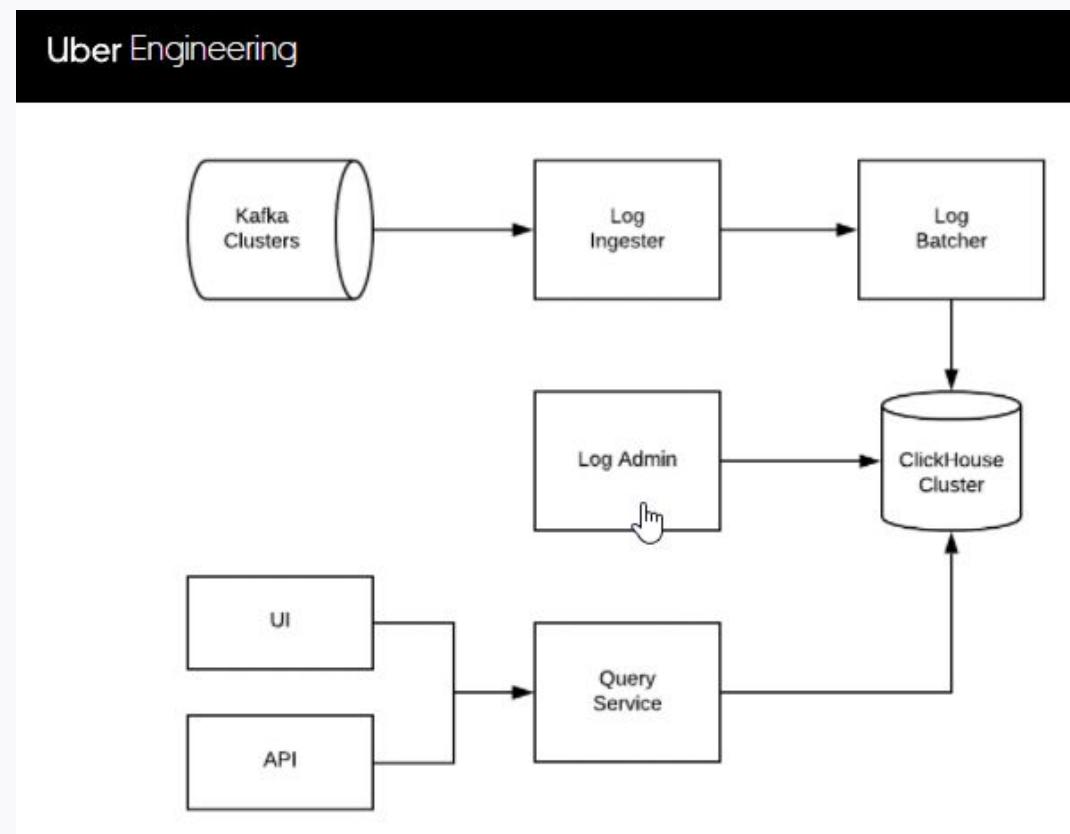
- Блок меньше размером
- Проще сжимается
- Вертикальное секционирование
- Быстрее запись

<https://clickhouse.tech/docs/en/> - тут есть gif, которая неплохо илюстрирует процесс

# Хранение логов в ClickHouse

Убер

<https://eng.uber.com/logging/>



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 querifiable. 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.

# Хранение логов в ClickHouse

Как VK вставляет данные в ClickHouse с десятков тысяч серверов  
Юрий Насретдинов (ВКонтакте)

HL HighLoad++

### ElasticSearch

1   Простота эксплуатации	4   Сжатие данных
2   Высокая скорость записи и чтения	5   Запись длинных строк (>4 Кб)
3   Долговременное хранение (месяцы, годы)	6   Очень много серверов (UDP)

HighLoad++

@HighLoadTalks @HighLoadChannel vk.com/HighLoadConference @profyclub fb.com/HighLoadConference

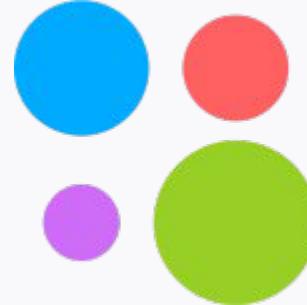
Как VK вставляет данные в ClickHouse с десятков тысяч серверов  
Юрий Насретдинов (ВКонтакте)

HL HighLoad++

### ClickHouse

1   Простота эксплуатации	4   Сжатие данных
2   Высокая скорость записи и чтения	5   Запись длинных строк (>4 Кб)
3   Долговременное хранение (месяцы, годы)	6   Очень много серверов (UDP)

HighLoad++



# Avito

Nginx-log-collector утилита от Авито для отправки логов nginx в Clickhouse

<https://temofeev.ru/info/articles/nginx-log-collector-utilita-ot-avito-dlya-otpravki-logov-nginx-v-clickhouse/>

# Хранение логов в ClickHouse

[https://nastachku.ru/var/files/1/presentation/backend/2\\_Backend\\_6.pdf](https://nastachku.ru/var/files/1/presentation/backend/2_Backend_6.pdf)

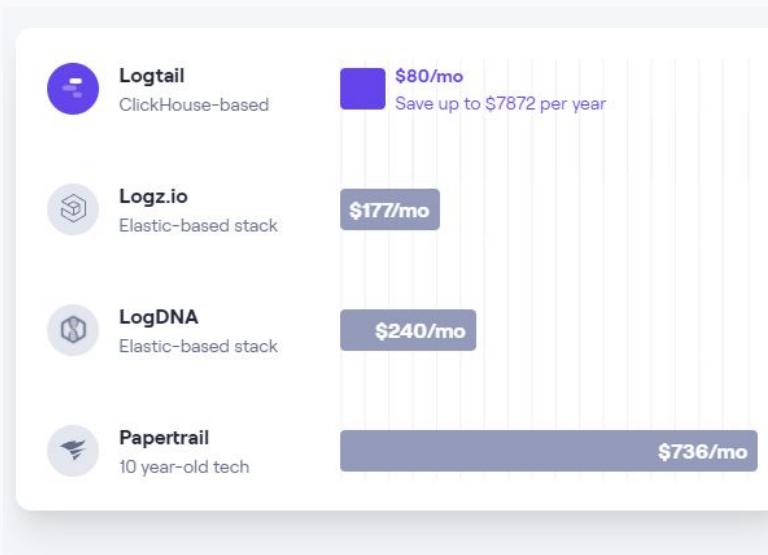


ecwid

## 20Тб логов в ClickHouse

Василий Васильков, Ecwid

<https://logtail.com/>



A screenshot of the Logtail website showing a price comparison chart. It lists four log processing services: Logtail, Logz.io, LogDNA, and Papertrail. Logtail is described as ClickHouse-based and costs \$80/mo, saving up to \$7872 per year. Logz.io is an Elastic-based stack at \$177/mo. LogDNA is an Elastic-based stack at \$240/mo. Papertrail is 10 year-old tech at \$736/mo. To the right, a section titled "Log more, pay less" claims Logtail's custom-built technology is significantly more efficient and cheaper than conventional Elastic stack, passing savings to the user. It also mentions 80GB per month with 30 day retention.

Service	Type	Price	Savings
Logtail	ClickHouse-based	\$80/mo	Save up to \$7872 per year
Logz.io	Elastic-based stack	\$177/mo	
LogDNA	Elastic-based stack	\$240/mo	
Papertrail	10 year-old tech	\$736/mo	

**Log more, pay less**

Logtail uses a custom-built technology to process your logs and stores them in ClickHouse. Our stack is significantly more efficient and thus cheaper than the conventional Elastic stack. We pass these savings to you.

80GB per month with 30 day retention

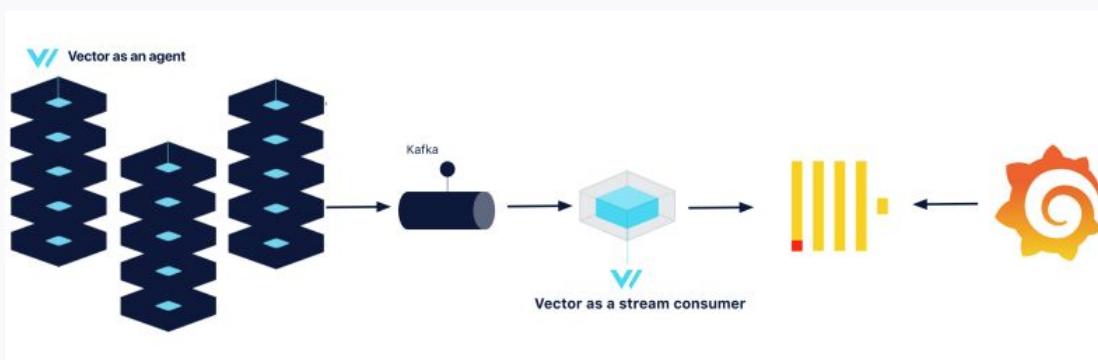
# Хранение логов в ClickHouse

## Есть ли жизнь без ELK?

Как снизить стоимость Log Management, используя Kafka, ClickHouse и Vector

Денис Безкоровайный,  
Директор подразделения DevOps/DevSecOps  
Proto Group

 Saint HighLoad++  
 PROTO Services for Modern Stack



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

## 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/>

# Хранение логов в ClickHouse



> Percon

Services ▾      Product

## Massive Parallel Log Processing with ClickHouse

2 MARCH 2021

**Clickhouse as an alternative to ElasticSearch and MySQL, for log storage and analysis, in 2021**

<https://pixeljets.com/blog/clickhouse-vs-elasticsearch/>

# Хранение логов в ClickHouse

<https://github.com/YuriyNasretdinov/logscli>

Использование ClickHouse в VK, или Зачем мы написали KittenHouse

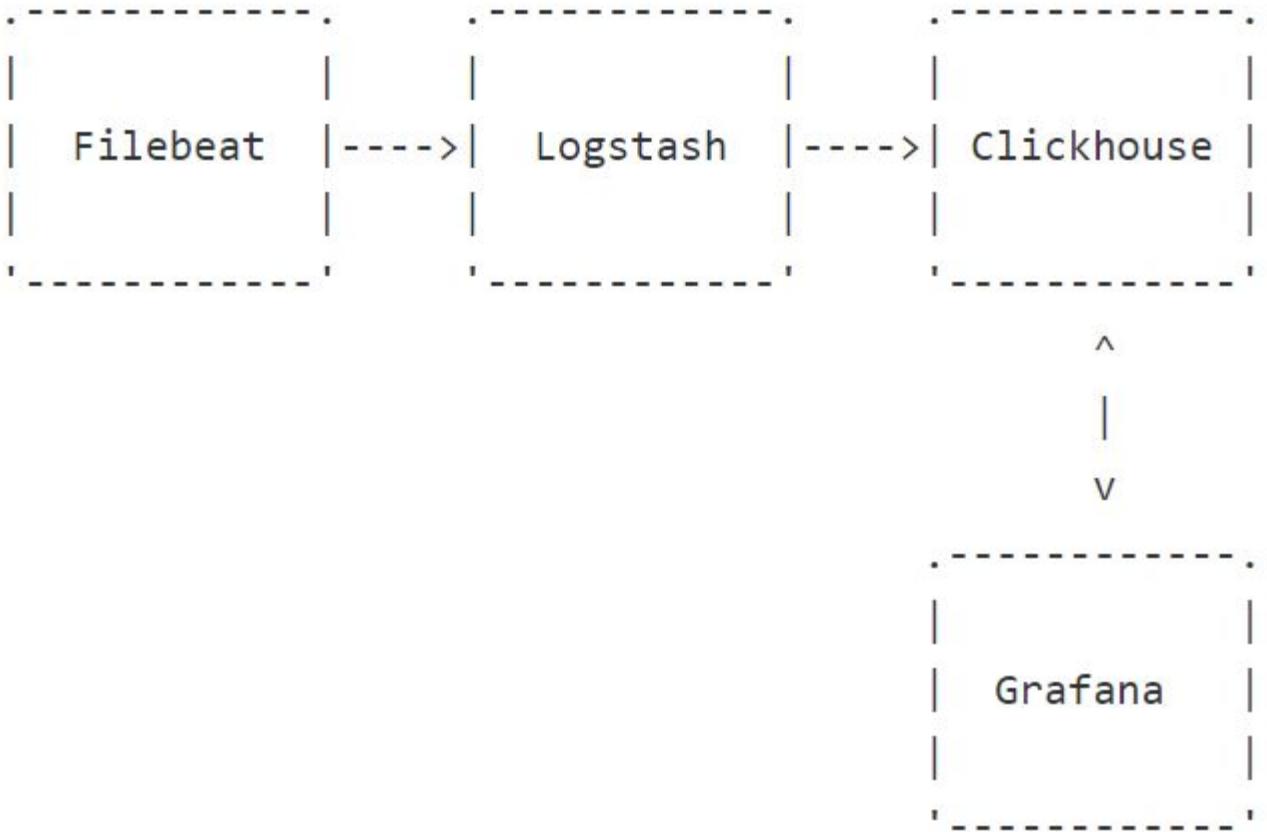
Блог компании VK , Высокая производительность \*, Программирование \*



В начале года мы решили научиться хранить и читать отладочные логи ВКонтакте более эффективно, чем раньше. Отладочные логи — это, к примеру, логи конвертации видео (в основном вывод команды ffmpeg и список шагов по предварительной обработке файлов), которые иногда бывают нам нужны лишь спустя 2-3 месяца после обработки проблемного файла.

<https://github.com/VKCOM/kittenhouse>

[https://blog.luisico.net/2019/03/17/testing\\_clickhouse\\_as\\_logs\\_analysis\\_storage/](https://blog.luisico.net/2019/03/17/testing_clickhouse_as_logs_analysis_storage/)



# Хранение логов в ClickHouse

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

# Создание структуры

# Создание базы данных

<https://clickhouse.com/docs/en/sql-reference/statements/create/database>

`CREATE DATABASE [IF NOT EXISTS] db_name  
[ON CLUSTER cluster] [ENGINE = engine(...)] [COMMENT 'Comment']`

Движки БД

<https://clickhouse.com/docs/en/engines/database-engines/>

Например для подключения к Постгресу (заодно посмотрим типы данных)

<https://clickhouse.com/docs/en/engines/database-engines/postgresql>

# Деплой кластера

<https://clickhouse.com/docs/ru/getting-started/tutorial#cluster-deployment>

# Создание таблицы

<https://clickhouse.com/docs/en/sql-reference/statements/create/table>

```
CREATE TABLE events (
    account_id UInt64,
    device_id UInt64,
    event_type Enum8(
        'Login' = 1,
        'Logout' = 2),
    country String,
    time_ns Int64,
    date_time DateTime MATERIALIZED toDateTime(time_ns / 1000000000)
)
ENGINE = MergeTree()
ORDER BY (account_id)
PARTITION BY toYYYYMM(date_time)
```

# Сортировка

- в ORDER BY указывается ключ сортировки
- по нему ФИЗИЧЕСКИ будут уложены данные на диске
- можно отдельно указать PRIMARY KEY (если отличается, но только как префикс полей для сортировки)

<https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree/#choosing-a-primary-key-that-differs-from-the-sorting-key>

- физически поменять сортировку (при этом данные будут **физически переупорядочены**)

`ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY ORDER BY new_expression`

только на движке MergeTree

# Сжатие

## Кодеки сжатия столбцов

По умолчанию, ClickHouse применяет к столбцу метод сжатия, определённый в [конфигурации сервера](#). Кроме этого, можно задать метод сжатия для каждого отдельного столбца в запросе CREATE TABLE.

```
CREATE TABLE codec_example
(
    dt Date CODEC(ZSTD),
    ts DateTime CODEC(LZ4HC),
    float_value Float32 CODEC(NONE),
    double_value Float64 CODEC(LZ4HC(9)),
    value Float32 CODEC(Delta, ZSTD)
)
ENGINE = <Engine>
...
```

Если кодек Default задан для столбца, используется сжатие по умолчанию, которое может зависеть от различных настроек (и свойств данных) во время выполнения. Пример: value UInt64 CODEC(Default) — то же самое, что не указать кодек.

Также можно подменить кодек столбца сжатием по умолчанию, определенным в config.xml:

```
ALTER TABLE codec_example MODIFY COLUMN float_value CODEC(Default);
```

Кодеки можно последовательно комбинировать, например, CODEC(Delta, Default)

<https://clickhouse.com/docs/ru/sql-reference/statements/create/table#create-query-common-purpose-codecs>

Вообще столбцовая СУБД, да и любая аналитическая СУБД без сжатия это очень странное дело.

Вопрос в зал – почему?

# Гибридное хранилище

Парты старше определенной даты можно переносить в дешевое S3

```
CREATE TABLE tutorial.hits_v1
(
    <схема>
)
ENGINE = MergeTree()
PARTITION BY EventDate
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID)
TTL EventDate + toIntervalDay(dateDiff('day', toDate('2014-03-20'), now())) TO DISK 'object_storage'
SETTINGS index_granularity = 8192
```

```
TTL expr
[DELETE|RECOMPRESS codec_name1|TO DISK 'xxx'|TO VOLUME 'xxx'][, DELETE|RECOMPRESS codec_name2|TO DISK 'aaa'|TO VOLUME 'bbb'] ...
[WHERE conditions]
[GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]] ]
```

<https://cloud.yandex.ru/docs/managed-clickhouse/tutorials/hybrid-storage>

# Движки таблиц

СН это MergeTree

- ReplacingMergeTree
- SummingMergeTree
- AggregatingMergeTree
- CollapsingMergeTree ...

<https://clickhouse.com/docs/ru/engines/table-engines/>

Дополнительные плюсы:  
TTL для столбцов и таблиц

# MergeTree

## Конкурентный доступ к данным

Для конкурентного доступа к таблице используется мультиверсионность. То есть, при одновременном чтении и обновлении таблицы, данные будут читаться из набора кусочков, актуального на момент запроса. Длинных блокировок нет. Вставки никак не мешают чтениям.

Чтения из таблицы автоматически распараллеливаются.

## Вертикальное партиционирование

<https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/custom-partitioning-key/>

## Сэмплирование

<https://habr.com/ru/post/519168/>

# Движки таблиц

## LOG – Ещё один популярный тип таблиц

- StripeLog
- Log
- TinyLog

<https://clickhouse.com/docs/ru/engines/table-engines/log-family/>

основной минус - Не поддерживают операции мутации.

Обычно логи в MergeTree пишут

# Движки таблиц

Но как бы... <https://habr.com/ru/company/flant/blog/341386/>

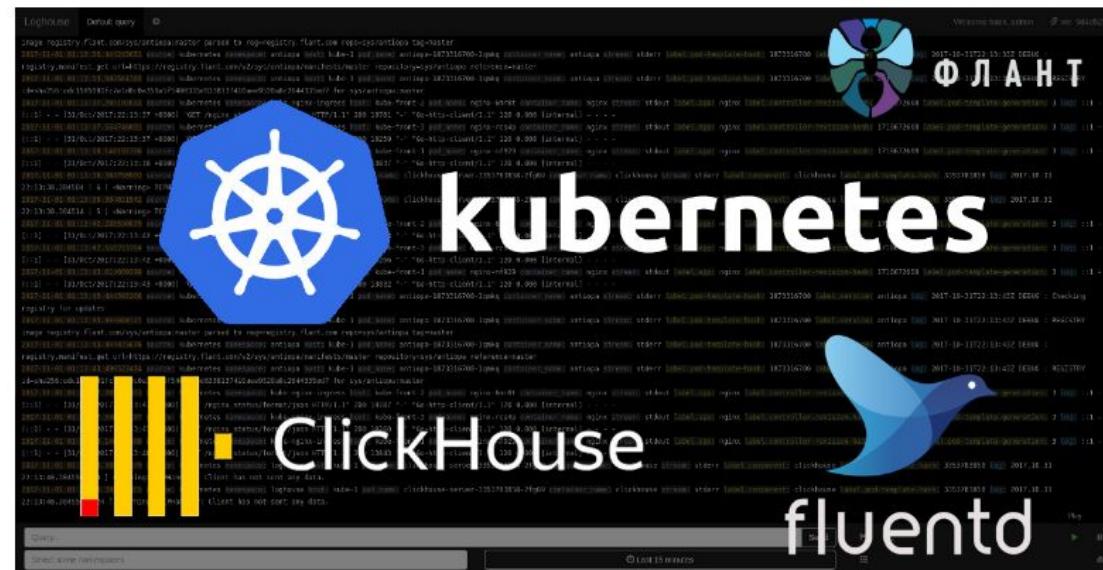
414,47  
Рейтинг

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



749a56bd27 → loghouse / docs / en / schemas / original / db.sql

gyrter New clickhouse schema (#137) ... ✓

2 contributors

50 lines (46 sloc) | 1.22 KB

```
1 CREATE TABLE logs
2 (
3     `date` Date MATERIALIZED toDate(timestamp),
4     `timestamp` DateTime,
5     `nsec` UInt32,
6     `source` String,
7     `namespace` String,
8     `host` String,
9     `pod_name` String,
10    `container_name` String,
11    `stream` String,
12    `labels.names` Array(String),
13    `labels.values` Array(String),
14    `string_fields.names` Array(String),
15    `string_fields.values` Array(String),
16    `number_fields.names` Array(String),
17    `number_fields.values` Array(Float64),
18    `boolean_fields.names` Array(String),
19    `boolean_fields.values` Array(Float64),
20    `null_fields.names` Array(String)
21 )
22 ENGINE = MergeTree()
23 PARTITION BY (date)
24 ORDER BY (timestamp, nsec, namespace, container_name)
25 TTL date + toIntervalDay(14)
26 SETTINGS index_granularity = 32768;
```

Нет, Флант знают что делают. И это лучше чем ElasticSearch

# Репликация

- только для семейства MergeTree
- на уровне отдельных таблиц
- репликация на уровне сжатых данных для **INSERT, ALTER**
- для координации используется **ZooKeeper**
- асинхронная
- master master
- блоки записываются атомарно и дедуплицируются

<https://clickhouse.com/docs/en/engines/database-engines/replicated>

<https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/replication>

Распределенные DDL

<https://clickhouse.com/docs/ru/sql-reference/distributed-ddl>

on-premise та еще история - до сих пор в статусе беты

посмотрим DBaaS

<https://console.cloud.yandex.ru/folders/b1ggepi457i7gntd5u09/managed-clickhouse/clusters>

# Проекции

<https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family/mergetree/#projections>

# Представления & materialized

<https://clickhouse.com/docs/ru/sql-reference/statements/create/view>

# Временные таблицы

<https://clickhouse.com/docs/ru/sql-reference/statements/create/table#temporary-tables>

- Временные таблицы исчезают после завершения сессии, в том числе при обрыве соединения.
- Временная таблица использует только модуль памяти.
- Невозможно указать базу данных для временной таблицы. Она создается вне баз данных.
- Невозможно создать временную таблицу распределенным DDL запросом на всех серверах кластера (с опцией ON CLUSTER): такая таблица существует только в рамках существующей сессии.
- Если временная таблица имеет то же имя, что и некоторая другая, то, при упоминании в запросе без указания БД, будет использована временная таблица.
- При распределённой обработке запроса, используемые в запросе временные таблицы, передаются на удалённые серверы.

# Настройки

- **max\_memory\_usage**
- **max\_execution\_time**
- **force\_primary\_key**

<https://clickhouse.com/docs/ru/operations/settings/query-complexity>

# Вставка данных

- на каждую вставку создается свой парт
- внутри парты для каждой колонки как минимум 1 файл
- парты мержатся в фоне
- частые мелкие вставки - плохо!

# Парти

```
SELECT
    table,
    partition,
    name,
    rows,
    disk_name
FROM system.parts
```

```
ubuntu2004.linuxvmimages.local : ) show databases
SHOW DATABASES
Query id: 49945ddc-d436-42dd-999c-c3200d094316
+-----+
| name |
+-----+
| datasets |
| default |
| system |
+-----+
3 rows in set. Elapsed: 0.019 sec.
```

	table	partition	name	rows	disk_name
1	visits_v1	201403	201403_1_6_1	1,194,218	default
2	visits_v1	201403	201403_7_7_0	199,246	default
3	visits_v1	201403	201403_8_8_0	199,534	default
4	visits_v1	201403	201403_9_9_0	86,793	default

# Парти

Внутри каждого парты файл с данными и файл с засечками

	UP - -DIR
..	
AdvEngineID.bin	58420
AdvEngineID.mrk2	4320
Age.bin	685938
Age.mrk2	4320
Attendance.bin	9685894
Attendance.mrk2	4320
BrowserCountry.bin	748948
BrowserCountry.mrk2	4320
BrowserLanguage.bin	336653
BrowserLanguage.mrk2	4320
CLID.bin	230087
CLID.mrk2	4320
ClickAWAPSCampaignName.bin	5320
ClickAWAPSCampaignName.mrk2	4320
ClickAttempt.bin	6063
ClickAttempt.mrk2	4320
ClickBannerID.bin	85823
ClickBannerID.mrk2	4320
ClickClientIP.bin	91251
ClickClientIP.mrk2	4320
ClickContextType.bin	35769
ClickContextType.mrk2	4320

# Парти

Но самые интересные файлы в конце:

Primary.idx – первичный разреженый индекс, который хранится в памяти.  
Чем меньше гранулярность, тем больше в нём данных. Тем он объёмнее.

В columns.txt – информация по колонкам

```
YCLID.mrk2
checksums.txt
columns.txt
count.txt
default_compression_codec.txt
minmax_StartDate.idx
partition.dat
primary.idx
```

# Index\_granularity

- максимальное количество строк в колонке (помним, что это отдельный файл) между засечками
- всегда читается не меньше index\_granularity строк данных
- выбирать аккуратно под ваши нужды - это важно
- по умолчанию 8192
- adaptive index\_granularity с 19.7, включен по умолчанию с 19.11

Несколько способов задания настроек движка при создании таблицы

```
SETTINGS index_granularity=8192
```

```
MergeTree(EventDate, intHash32(UserID), (CounterID, EventDate, intHash32(UserID)), 8192)
```

# Дополнительные индексы

Они как бы есть, но зачем? у нас обычно выборка по всей БД

<https://clickhouse.com/docs/ru/sql-reference/statements/alter/index/>

# Вставка данных

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

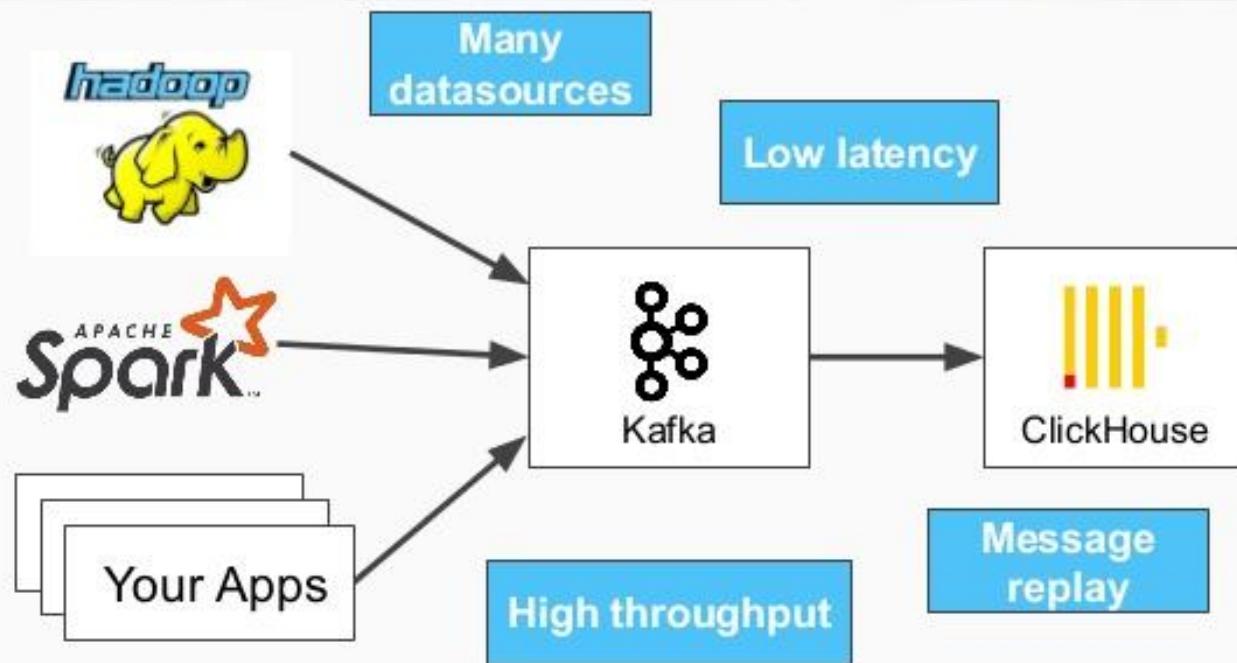
Как достигнуть:

- на стороне приложения
- open source решения (kittenhouse, clickhouse-bulk, ...)
- **kafka engine**
- buffer table
- http-chunk (осторожно) и можно придумать еще!

# Kafka

## Reasons to use Kafka with ClickHouse

© 2020, Altinity LTD



# Kafka

Чтение сообщения с помощью `SELECT` не слишком полезно (разве что для отладки), поскольку каждое сообщения может быть прочитано только один раз. Практичнее создавать потоки реального времени с помощью материализованных представлений. Для этого:

1. Создайте потребителя Kafka с помощью движка и рассматривайте его как поток данных.
2. Создайте таблицу с необходимой структурой.
3. Создайте материализованное представление, которое преобразует данные от движка и помещает их в ранее созданную таблицу.

[https://clickhouse.tech/docs/r  
u/enGINes/table-ENGINes/integrations/kafka/](https://clickhouse.tech/docs/ru/enGINes/table-ENGINes/integrations/kafka/)

Когда к движку присоединяется материализованное представление (`MATERIALIZED VIEW`), оно начинает в фоновом режиме собирать данные. Это позволяет непрерывно получать сообщения от Kafka и преобразовывать их в необходимый формат с помощью `SELECT`.

Материализованных представлений у одной kafka таблицы может быть сколько угодно, они не считывают данные из таблицы kafka непосредственно, а получают новые записи (блоками), таким образом можно писать в несколько таблиц с разным уровнем детализации (с группировкой - агрегацией и без).

Пример:

```
CREATE TABLE queue (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');

CREATE TABLE daily (
    day Date,
    level String,
    total UInt64
) ENGINE = SummingMergeTree(day, (day, level), 8192);

CREATE MATERIALIZED VIEW consumer TO daily
    AS SELECT toDate(toDateTime(timestamp)) AS day, level, count() as total
    FROM queue GROUP BY day, level;

SELECT level, sum(total) FROM daily GROUP BY level;
```

Для улучшения производительности полученные сообщения группируются в блоки размера `max_insert_block_size`. Если блок не удалось сформировать за `stream_flush_interval_ms` миллисекунд, то данные будут сброшены в таблицу независимо от полноты блока.

# Интерфейсы подключения

- Нативный (для тех кто любит...)
- Clickhouse-client – cli (стандарт)
- Http (хороший и удобный)
- Mysql (для ленивых)
- JDBC/ODBC – с ограничениями конечно
- Клиенты есть почти для любого языка (go, python, java, C#... )  
<https://clickhouse.tech/docs/ru/interfaces/third-party/client-libraries/>
- Kafka, postgresql, mysql, mssql, flink, grafana, k8s, m puppet  
<https://clickhouse.tech/docs/ru/interfaces/third-party/integrations/>

# CH Grafana

Access to CH via HTTP

Page configuration is standard

### Add data source

Name	My data source name	<small>i</small>	Default	<input type="checkbox"/>
Type	ClickHouse			
Http settings				
Url	http://localhost:8123			
Access	proxy			
Http Auth				
Basic Auth	<input type="checkbox"/>	With Credentials	<small>i</small>	<input type="checkbox"/>
TLS Client Auth	<input type="checkbox"/>	With CA Cert	<small>i</small>	<input type="checkbox"/>
Additional				
Add CORS flag to requests	<small>i</small>	<input type="checkbox"/>		
Use POST method to send queries	<small>i</small>	<input type="checkbox"/>		

Вообщем совместно с Grafana Clickhouse явно может заменить ElasticSearch для хранения логов

# CH DataGrip

Ну и для тех кто без ума от творчества JetBrains

The screenshot displays the ClickHouse DataGrip interface, which is a part of the ClickHouse ecosystem. It includes:

- Database Browser:** On the left, a sidebar shows available data sources: Data Source (selected), DDL Data Source, Data Source from URL, Data Source from Path, Driver and Data Source, Driver, and Import from Sources... The ClickHouse entry is highlighted.
- SQL Console:** The main area shows a database tree under "MyDatabases". A connection to "Postgres in Docker" is selected. The tree shows databases (guest), schemas (public), tables (32), views (5), routines (10), trigger functions, aggregates (1), sequences (17), object types (3), languages (4), and roles (11). Other databases listed include Docker SQL Server (2 of 13) and Postgres in Docker (2 of 4).
- Results Viewer:** Below the tree, a table lists actor records from the "actor" table. The columns are actor\_id, first\_name, last\_name, and last\_update. The data is as follows:

	actor_id	first_name	last_name	last_update
1	56	DAN	HARRIS	2006-02-15 04:34:33.00
2	113	MORGAN	HOPKINS	2006-02-15 04:34:33.00
3	93	ELLEN	PRESLEY	2006-02-15 04:34:33.00
4	27	JULIA	MCQUEEN	2006-02-15 04:34:33.00
5	131	JANE	JACKMAN	2006-02-15 04:34:33.00
6	196	BELA	WALKEN	2006-02-15 04:34:33.00
7	68	RIP	WINSLET	2006-02-15 04:34:33.00
8	95	DARYL	WAHLBERG	2006-02-15 04:34:33.00
9	79	MAE	HOFFMAN	2006-02-15 04:34:33.00

# Бэкапы

<https://github.com/AlexAkulov/clickhouse-backup>

<https://clickhouse.com/docs/ru/operations/utilities/clickhouse-copier/>

- 1- я работает с партами
- 2 - я в новый кластер

# Вопросы на понимание - где СН?

- Хочу постоянно искать пользователя по id?
- Хочу писать информацию о банковских транзакциях?
- Хочу строить месячные отчеты по широкой таблице, при этом самих данных очень много?
- Хочу нормализовать таблицу?
- Хочу записать связанные данные в две и более таблицы?
- Хочу использовать ML?
- Хочу организовать прокси/кэш?

# Вопросы на понимание - где СН?

- Хочу постоянно искать пользователя по id?
- Хочу писать информацию о банковских транзакциях?
- Хочу строить месячные отчеты по широкой таблице, при этом самих данных очень много?
- Хочу нормализовать таблицу?
- Хочу записать связанные данные в две и более таблицы?
- Хочу использовать ML?
- Хочу организовать прокси/кэш?

# Вопросы



**МИНИТЕСТ**

**<https://forms.gle/aA28pTpFxTSohTpZ7>**

**2-3 минуты**

# Варианты установки

# Варианты установки

на сайте производителя (из исходных кодов, архивов и тд)

<https://clickhouse.com/docs/ru/getting-started/install>

DBaaS от Yandex

<https://console.cloud.yandex.ru/folders/b1qgepi457i7gntd5u09/managed-clickhouse/clusters>

Докер

<https://hub.docker.com/r/clickhouse/clickhouse-server/>

Кубер (единого варианта нет)

<https://github.com/plutov/clickhouse-helm>

<https://pliutau.com/ultimate-clickhouse-helm-chart/>

# Установка и использование

## Docker

```
docker pull yandex/clickhouse-server
```

```
docker run -d --name some-clickhouse-server --ulimit nofile=262144:262144 yandex/clickhouse-server
```

```
docker run -it --rm --link some-clickhouse-server:clickhouse-server yandex/clickhouse-client --host clickhouse-server
```

# Установка и использование

**Cloud.yandex.ru** - я думаю вряд ли будут сомнения что СН лучше разворачивать в Yandex.Cloud

← Создание платежного аккаунта

**Создайте платёжный аккаунт и получите грант не менее чем на 4000 ₽ сроком действия 60 дней, чтобы познакомиться с облачными сервисами в рамках пробного периода. Подробнее об условиях использования гранта читайте в [документации](#).**

Страна [?](#) Россия

Имя аккаунта

Тип плательщика

Физическое лицо  Юридическое лицо или ИП

Данные плательщика

Имя Олег

Фамилия Филиппов

Отчество Александрович

Банковская карта [?](#)

Добавить карту

Нажимая кнопку «Активировать», вы принимаете [Оферту](#)

**Активировать**

# Установка и использование

Поскольку все любим CLI:

<https://cloud.yandex.ru/docs/cli/quickstart> - скачать и установить Yandex CLI

# Визуальный интерфейс

comoltest3 — Alive 11 декабря 2020, в 03:26 20.8 PRODUCTION ...

Подключиться Переместить Остановить Удалить

### Подключиться к comoltest3

Shell Python PHP Java Node.js Go ODBC

Инструкции по использованию клиента командной строки  
Установите сертификат:

```
mkdir -p ~/.clickhouse-client /usr/local/share/ca-certificates/Yandex && \
wget "https://storage.yandexcloud.net/cloud-certs/CA.pem" -O /usr/local/share/ca-certificates/Yandex/YandexInternal
wget "https://storage.yandexcloud.net/mdb/clickhouse-client.conf.example" -O ~/.clickhouse-client/config.xml
```

Пример строки подключения:

```
clickhouse-client --host rc1a-xu1xoaxa9o7utima.mdb.yandexcloud.net \
--secure \
--user user1 \
--database <dbname> \
--port 9440 \
--ask-password
```

# Визуальный интерфейс

## Создание кластера ClickHouse

### Базовые параметры

Имя кластера  ?

Описание

Окружение  ?

Версия  ?

Сервисный аккаунт

### Класс хоста

Платформа  ?

Тип

s2.micro	s2.small	s2.medium	s2.large
2 ядра vCPU	8 ГБ Память	4 ядра vCPU	16 ГБ Память

s2.5xlarge	s2.6xlarge
48 ядра vCPU	192 ГБ Память

### Размер хранилища

10 ГБ  
 ?

2053 ГБ

4096 ГБ

Макс. IOPS  ? Чтение 400 Запись 1000

Макс. bandwidth  ? Чтение 15 МБ/с Запись 15 МБ/с

### База данных

Имя БД  ?

Имя пользователя  ?

Пароль

### Сеть

?

### Хосты

#	Зона доступности	Подсеть	Публичный доступ
1	ru-central1-a	default-ru-central1-a	нет

### Дополнительные настройки

Начало резервного копирования (UTC)  ?

Окно обслуживания

Гибридное хранилище ? Функциональность находится в стадии Preview. [Запросить доступ.](#)

Доступ из DataLens

Доступ из консоли управления

Доступ из Метрики и AppMetrica

Доступ из Serverless

### Настройки СУБД

? Расширенные настройки СУБД могут значительно повлиять на производительность кластера. Меняйте их только если вы точно знаете, что хотите сделать.

# Загружаем данные

```
curl https://clickhouse-datasets.s3.yandex.net/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
```

```
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
```

```
clickhouse-client --query "CREATE TABLE datasets.visits_v1 ( CounterID UInt32, StartDate Date, Sign Int8, IsNew UInt8, VisitID UInt64, UserID UInt64, StartTime DateTime, Duration UInt32, UTCStartTime DateTime, PageViews Int32, Hits Int32, IsBounce UInt8, Referer String, StartURL String, RefererDomain String, StartURLDomain String, EndURL String, LinkURL String, IsDownload UInt8, TraficSourceID Int8, SearchEngineID UInt16, SearchPhrase String, AdvEnginID UInt8, PlaceID Int32, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions Array(UInt32), IsYandex UInt8, GoalReachesDepth Int32, GoalReachesURL Int32, GoalReachesAny Int32, SocialSourceNetworkID UInt8, SocialSourcePage String, MobilePhoneModel String, ClientEventTime DateTime, RegionID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RemoteIP UInt32, RemoteIP6 FixedString(16), IPNetworkID UInt32, SilverlightVersion3 UInt32, CodeVersion UInt32, ResolutionWidth UInt16, ResolutionHeight UInt16, UserAgentMajor UInt16, UserAgentMinor UInt16, WindowClientWidth UInt16, WindowClientHeight UInt16, SilverlightVersion2 UInt8, SilverlightVersion4 UInt16, FlashVersion3 UInt16, FlashVersion4 UInt16, ClientTimeZone Int16, OS UInt8, UserAgent UInt8, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, NetMajor UInt8, NetMinor UInt8, MobilePhone UInt8, SilverlightVersion1 UInt8, Age UInt8, Sex UInt8, Income UInt8, JavaEnable UInt8, CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, BrowserLanguage UInt16, BrowserCountry UInt16, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), Params Array(String), Goals Nested(ID UInt32, Serial UInt32, EventTime DateTime, Price Int64, OrderID String, CurrencyID UInt32), WatchIDs Array(UInt64), ParamSumPrice Int64, ParamCurrency FixedString(3), ParamCurrencyID UInt16, ClickLogID UInt64, ClickEventID Int32, ClickGoodEvent Int32, ClickEventTime DateTime, ClickPriorityID Int32, ClickPhraseID Int32, ClickPageID Int32, ClickPlaceID Int32, ClickTypeID Int32, ClickResourceID Int32, ClickCost UInt32, ClickClientIP UInt32, ClickDomainID UInt32, ClickURL String, ClickAttempt UInt8, ClickOrderID UInt32, ClickBannerID UInt32, ClickMarketCategoryID UInt32, ClickMarketPP UInt32, ClickMarketCategoryName String, ClickMarketPPName String, ClickAWAPSCampaignName String, ClickPageName String, ClickTargetType UInt16, ClickTargetPhraseID UInt64, ClickContextType UInt8, ClickSelectType Int8, ClickOptions String, ClickGroupBannerID Int32, OpenstatServiceName String, OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String, UTMCampaign String, UTMTerm String, FromTag String, HasGCLID UInt8, FirstVisit DateTime, PredLastVisit Date, LastVisit Date, TotalVisits UInt32, TraficSource Nested(ID Int8, SearchEngineID UInt16, AdvEnginID UInt8, PlaceID UInt16, SocialSourceNetworkID UInt8, Domain String, SearchPhrase String, SocialSourcePage String), Attendance FixedString(16), CLID UInt32, YCLID UInt64, NormalizedRefererHash UInt64, SearchPhraseHash UInt64, RefererDomainHash UInt64, NormalizedStartURLHash UInt64, StartURLDomainHash UInt64, NormalizedEndURLHash UInt64, TopLevelDomain UInt64, URLscheme UInt64, OpenstatServiceNameHash UInt64, OpenstatCampaignIDHash UInt64, OpenstatAdIDHash UInt64, OpenstatSourceIDHash UInt64, UTMSourceHash UInt64, UTMMediumHash UInt64, UTMCampaignHash UInt64, UTMCContentHash UInt64, UTMTermHash UInt64, FromHash UInt64, WebVisorEnabled UInt8, WebVisorActivity UInt32, ParsedParams Nested(Key1 String, Key2 String, Key3 String, Key4 String, Key5 String, ValueDouble Float64), Market Nested(Type UInt8, GoalID UInt32, OrderID String, OrderPrice Int64, PP UInt32, DirectPlaceID UInt32, DirectOrderID UInt32, DirectBannerID UInt32, GoodID String, GoodName String, GoodQuantity Int32, GoodPrice Int64), IslandID FixedString(16)) ENGINE = CollapsingMergeTree(Sign) PARTITION BY toYYYYMM(StartDate) ORDER BY (CounterID, StartDate, intHash32(UserID), VisitID) SAMPLE BY intHash32(UserID) SETTINGS index_granularity = 8192"
```

```
cat visits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.visits_v1 FORMAT TSV" --max_insert_block_size=100000
```

```
clickhouse-client --query "OPTIMIZE TABLE datasets.visits_v1 FINAL"
clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1"
```

# P.S.

<https://prohoster.info/blog/administrirovanie/1-1-milliard-poezdok-na-taksi-108-yadernyj-klaster-clickhouse>

# Огромный FAQ

# Вопросы

# Домашнее задание для курса NoSQL

Необходимо, используя тьюториал <https://clickhouse.tech/docs/ru/getting-started/tutorial/> :

- развернуть инстанс;
- выполнить импорт тестовой БД;
- выполнить несколько запросов и оценить скорость выполнения.

\* развернуть дополнительно одну из тестовых БД

<https://clickhouse.com/docs/en/getting-started/example-datasets> , протестировать скорость запросов

\*\* развернуть Кликхаус в кластерном исполнении, создать распределенную таблицу, заполнить данными и протестировать скорость по сравнению с 1 инстансом

Дз сдается в виде миниотчета.



Заполните, пожалуйста,  
опрос о занятии по ссылке в чате  
<https://otus.ru/polls/75229/>



Спасибо за внимание!  
Приходите на следующие вебинары

Аристов Евгений