

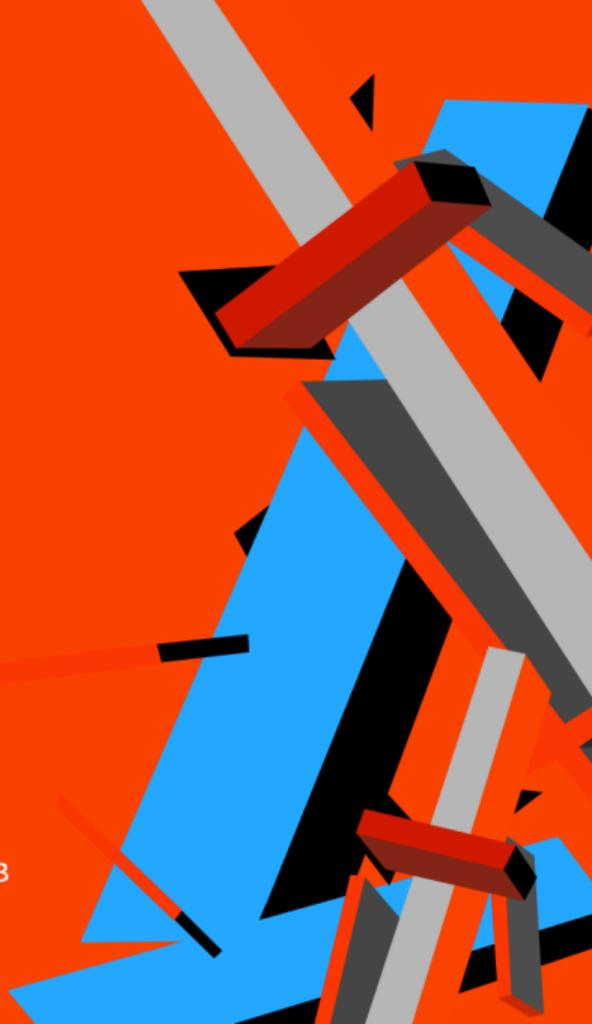
# Тестирование ClickHouse которого мы заслуживаем

Александр Сапин



**BackendConf**  
РИТ 2019

Профессиональная  
конференция  
для веб-разработчиков



# Про себя

- › Работаю в Яндексе
- › Разрабатываю ClickHouse
- › Иногда пишу тесты

# Про ClickHouse

# ClickHouse: Поколоночная СУБД

- › Не тормозит.
- › Масштабируется линейно
- › Позволяет хранить петабайты данных
- › Поддерживает гибкий SQL с расширениями
- › Отказоустойчив при работе в разных ДЦ

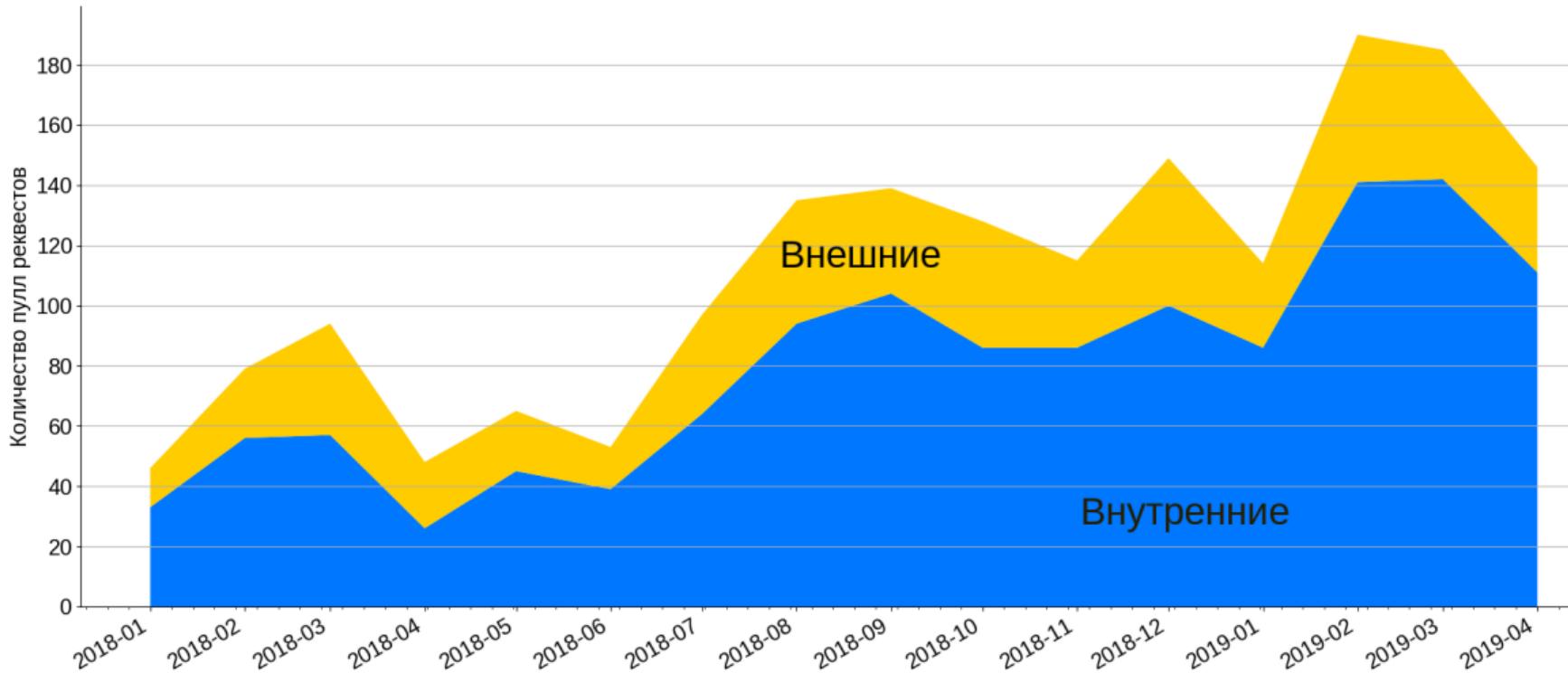


# ClickHouse: как проект

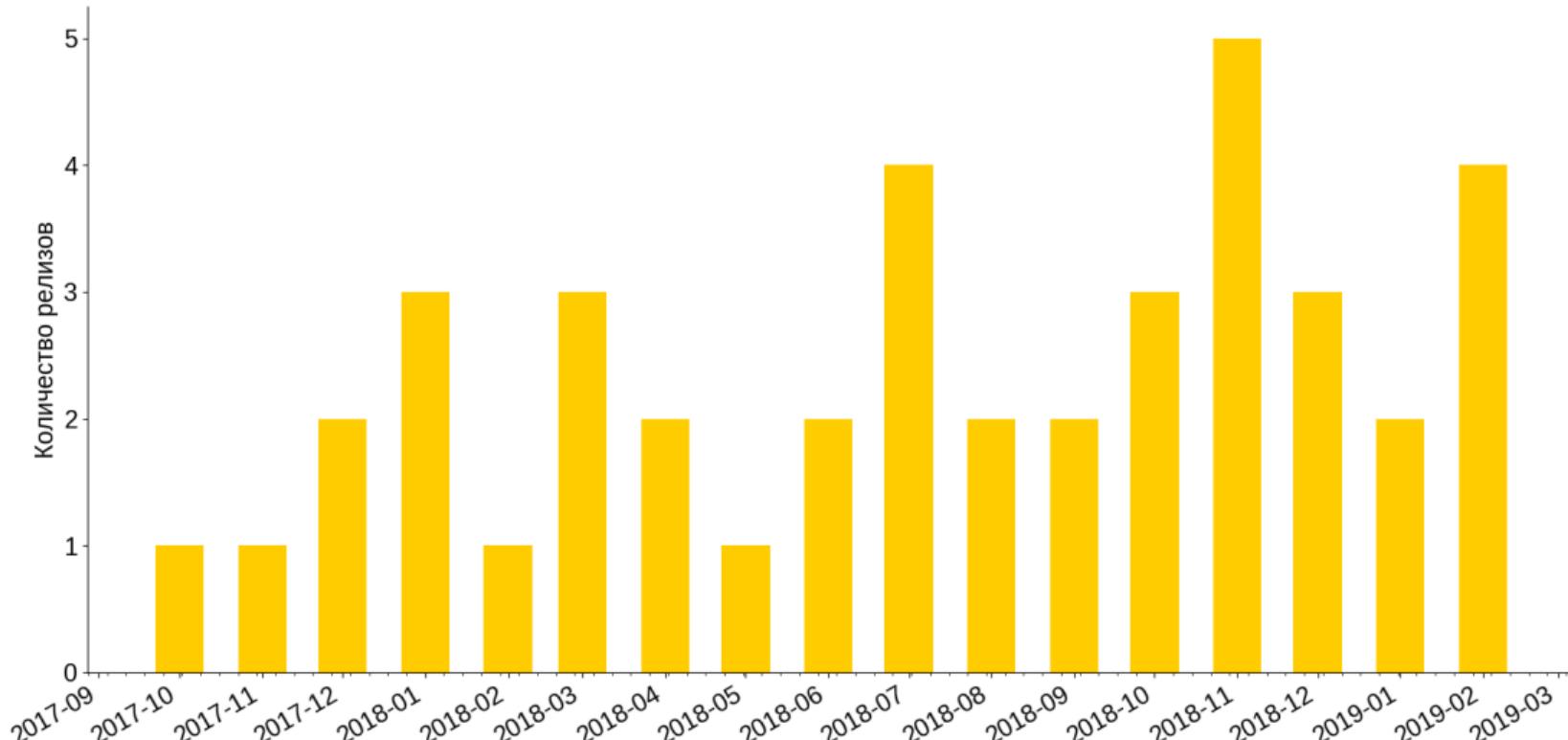
- › Открытый исходный код на C++
- › Больше 300 тысяч строк кода
- › Открытый репозиторий на GitHub
- › Изменения через пулл реквесты
- › В неделю вливается 40 пулл реквестов
- › 20% изменений от внешних разработчиков
- › Уже 6900 звездочек!



# Пулл реквесты



# Релизы



# Issues

## ❗ SQL parser error bug

#4858 opened 15 hours ago

## ❗ Error: Type mismatch for column because of INTERVAL bug

#4854 opened 17 hours ago

## ❗ Abnormally long merge bug

#4853 opened 18 hours ago

## ❗ Segment Fault of arrayIntersect bug

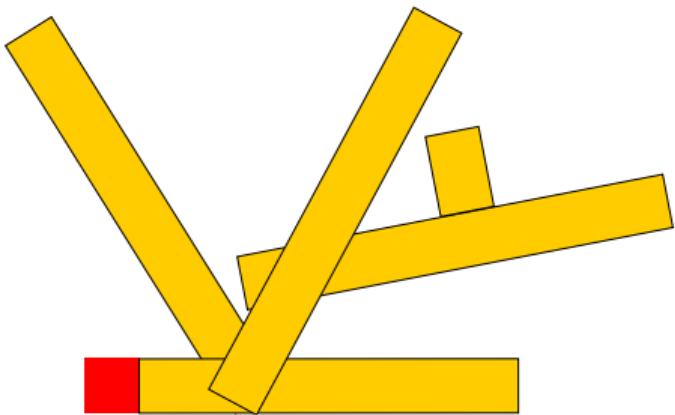
#4843 opened a day ago

## ❗ Segfault with Distributed tables bug

#5327 opened 2 days ago

# Что можно сломать

- › Управление памятью
- › Язык запросов
- › Производительность
- › Построение плана запроса
- › Формат данных на диске
- › Репликацию данных
- › Интеграцию с другими системами



# Про тесты

# Сборка: необходимое

- › Разные компиляторы
- › Простой переход на новую версию
- › Разные OS (Linux, FreeBSD, macOS)
- › Строгие флаги
- › Минимум зависимостей
- › Воспроизводимость
- › Быстро!

# Сборка: особенности

- > C++17, clang, gcc
- > -Wall -Werror -Wpedantic -Wextra
- > Настроенный -Weverything
- > Система сборки CMake + Ninja
- > Статический бинарник под все Linux
- > Запуск в Docker и Vagrant
- > 6 млн строк в contrib
- > Сборки для энтузиастов



# Сборка: санитайзеры

**Sanitizers** – библиотеки из поставки компиляторов, помогают найти проблемы в коде на C++ или Go.

## Виды:

- › Address – неправильная работа с адресацией и утечки
- › Thread – гонки и взаимные блокировки
- › Undefined – неопределенное поведение
- › Memory – использование неинициализированной памяти

## Ссылки:

- › Основной репозиторий: <https://github.com/google/sanitizers>
- › Внутри LLVM: <http://compiler-rt.llvm.org/>

# Сборка: конфигурации

Компилятор	Тип сборки	Санитайзер	Библиотеки	Бинарник
clang-7	debug	—	contrib	static
clang-7	release	address	contrib	static
clang-7	release	undefined	contrib	static
clang-7	release	—	contrib	shared
clang-8	release	thread	contrib	static
gcc-8	release	—	contrib	static
gcc-8	release	—	system	static

И это не все...

# ClickHouse не тормозит, сборка тормозит

- › Время одной сборки – 20 минут на 32 ядрах + 128 RAM
- › 10 вариантов сборок на каждый коммит

# Сборка: ускорение

## Не сработало:

- › Кэширование с помощью distcc
- › Unity builds + precompiled headers
  - <https://github.com/sakra/cotire>
  - Возможно недотюнили

# Сборка: ускорение

## Не сработало:

- › Кэширование с помощью distcc
- › Unity builds + precompiled headers
  - <https://github.com/sakra/cotire>
  - Возможно недотюнили

## Сработало (bruteforce):

- › ccache размером 5GB
- › хранение кэша на каждый коммит
- › использование кэша родительского коммита
- › ускорение в 2.5 раза

## Ссылка:

- › [http://slides.com/onqtam/faster\\_builds](http://slides.com/onqtam/faster_builds)

# Совместимость с версиями Linux

- › Одна сборка под все дистрибутивы
- › Зависимость от libc
  - Протекает через contrib
  - Символы версионируются
- › Зависимость от ядра
  - Системные вызовы
  - Фичи (AIO, taskstats)



# Тесты совместимости

## проверяем:

- › Версии символов @GLIBC\_
- › Запуск в docker на CentOS 5 и Ubuntu 12.04

## чиним:

- › Заменяем функции из glibc на куски из musl
- › Убираем слишком новые системные вызовы

# Про настоящие тесты

# Джентельменский набор тестов

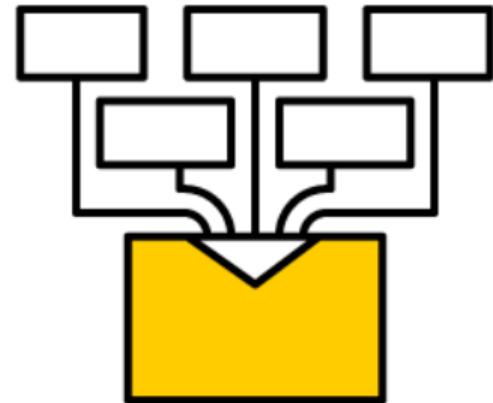
- › Статический анализ кода с PVS-studio
- › Unit-тесты на gtest
- › Функциональные тесты
  - .sql на входе + файл с ожиданием
  - запуск произвольных скриптов
  - с санитайзерами
  - с данными

# Недостатки обычных тестов

- › Их мало
- › Покрыто около 70% кода
- › Не тестируют пересечения фич
- › Долго писать
- › Не тестируют одновременные запросы

# Стресс-тест

- › *n* копий функциональных тестов
- › Нагрузка из мусорных запросов
- › С **address** и **thread** санитайзерами
- › Проверяется живость сервера



# Гибкий язык запросов

- › Больше 600 функций
- › 38 типов данных
- › Десятки операторов
- › Нестандартный диалект SQL



[https://www.kubomarket.ru/product/moyu-13x13832/](https://www.kubomarket.ru/product/moyu-13x13x13832/)

# Fuzz-тест

- › Имеет словарь типов данных и выражений
- › Список функций получает от ClickHouse
- › Генерирует случайные запросы
- › Проверяет живость сервера
- › Запуск с UB и address санитайзерами

Примеры запросов:

```
SELECT metroHash64(uniqUpTo('\0', '2[Vu]', 'Y&d');  
SELECT joinGet(toDateTimeOrNull((CAST(([885455.14523]) AS String))));  
SELECT (SELECT 1) FROM remote('127.0.0.{1,2}', system.one);
```

# Про интеграцию

# С чем интегрируется ClickHouse

## Внешние системы:

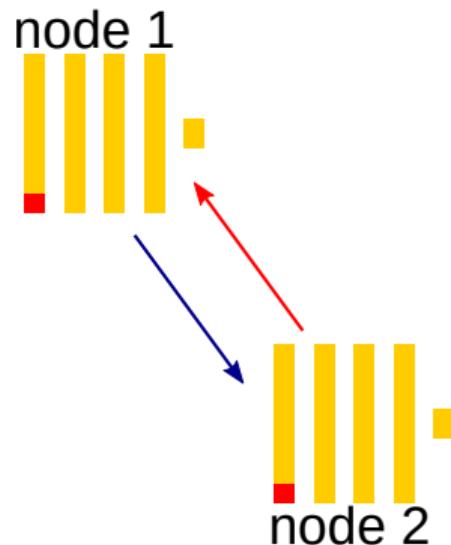
- › СУБД: MySQL, MongoDB, PostgreSQL, ...
- › Распределенные системы: Kafka, ZK, HDFS, ...

## С самим собой:

- › При распределенных запросах
- › Через реплицируемые таблицы

# Особенности репликации

- › Асинхронная мастер-мастер
- › Zookeeper как хранилище метаданных
- › Работает потаблично
- › Eventually Consistent
- › **Highly Available:**
  - Переживает разрывы между узлами
  - Потерю соединения с ZK

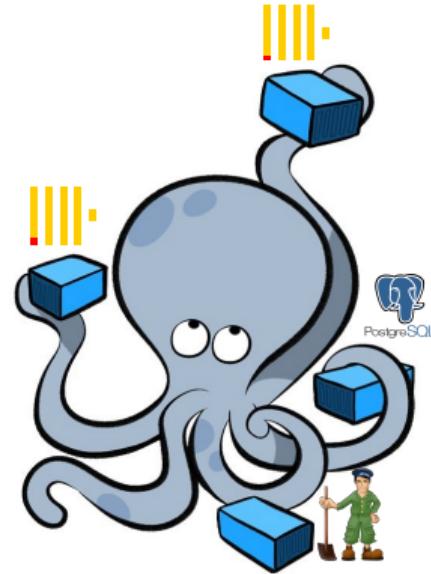


# Интеграционные тесты: что хотим

- › Моделировать взаимодействие с другими системами
- › Разрывать сеть между компонентами
- › Запуск окружения одной командой
- › Сценарий теста на языке программирования
- › Минимум требуемых зависимостей
- › Без мусора на хосте

# Интеграционные тесты: фреймворк

- › docker-compose для окружения
- › python + pytest для описания сценария
- › Конфигурация в коде



# Интеграционные тесты: сеть

## iptables для управления сетью

```
$ iptables -I DOCKER-USER 1 -s 192.168.2.1 -d 192.168.2.2 -j DROP  
$ iptables -D DOCKER-USER -s 192.168.2.1 -d 192.168.2.2 -j DROP
```

# Интеграционные тесты: сеть

## iptables для управления сетью

```
$ iptables -I DOCKER-USER 1 -s 192.168.2.1 -d 192.168.2.2 -j DROP  
$ iptables -D DOCKER-USER -s 192.168.2.1 -d 192.168.2.2 -j DROP  
$ iptables -I DOCKER-USER 1 -m statistic --mode random --probability 0.2 \  
      -s 192.168.2.1 -d 192.168.2.2 -j DROP
```

# Интеграционные тесты: сеть

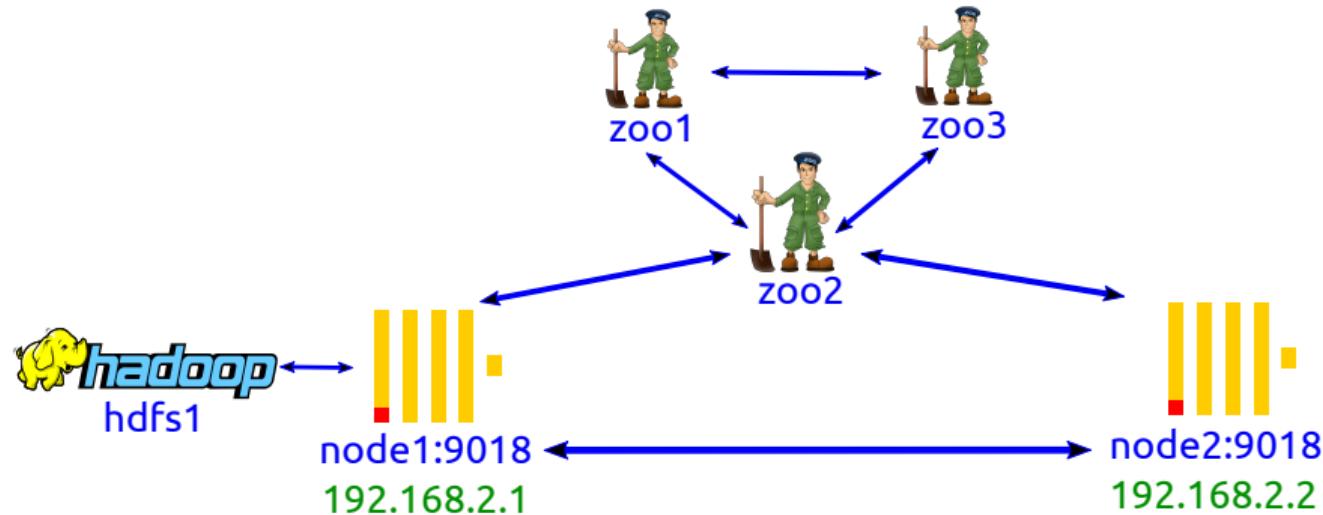
## iptables для управления сетью

```
$ iptables -I DOCKER-USER 1 -s 192.168.2.1 -d 192.168.2.2 -j DROP  
$ iptables -D DOCKER-USER -s 192.168.2.1 -d 192.168.2.2 -j DROP  
$ iptables -I DOCKER-USER 1 -m statistic --mode random --probability 0.2 \  
      -s 192.168.2.1 -d 192.168.2.2 -j DROP
```

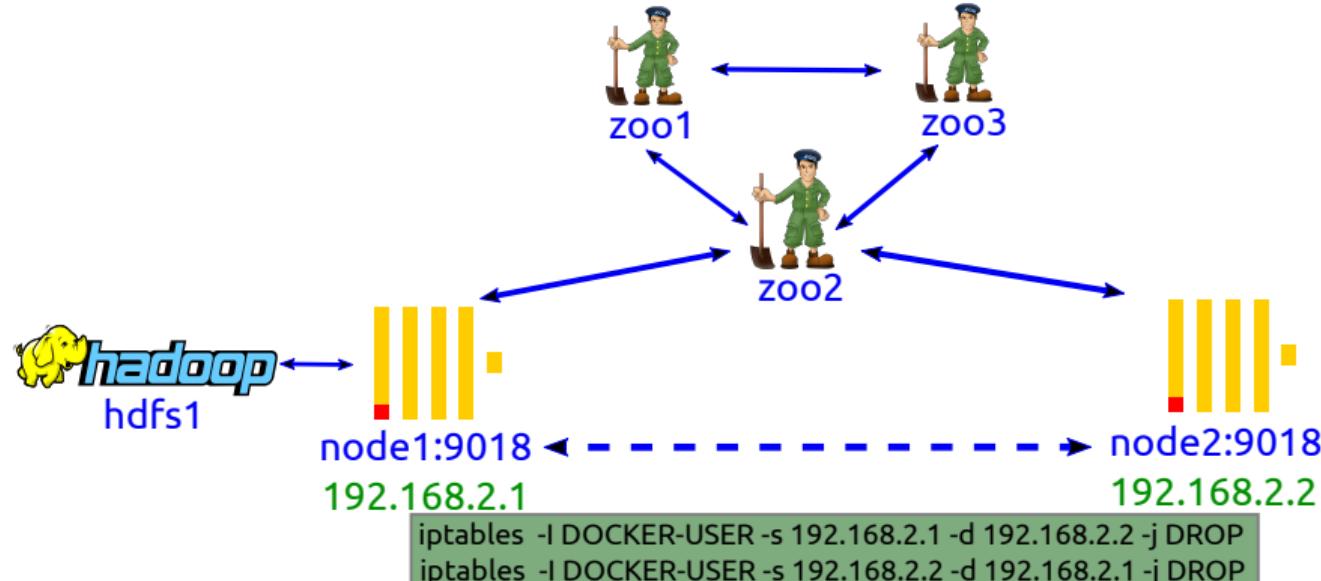
## Без sudo

- › Отдельный контейнер с Alpine Linux
- › Запуск в хостовой сети (--net=host)
- › Периодический перезапуск

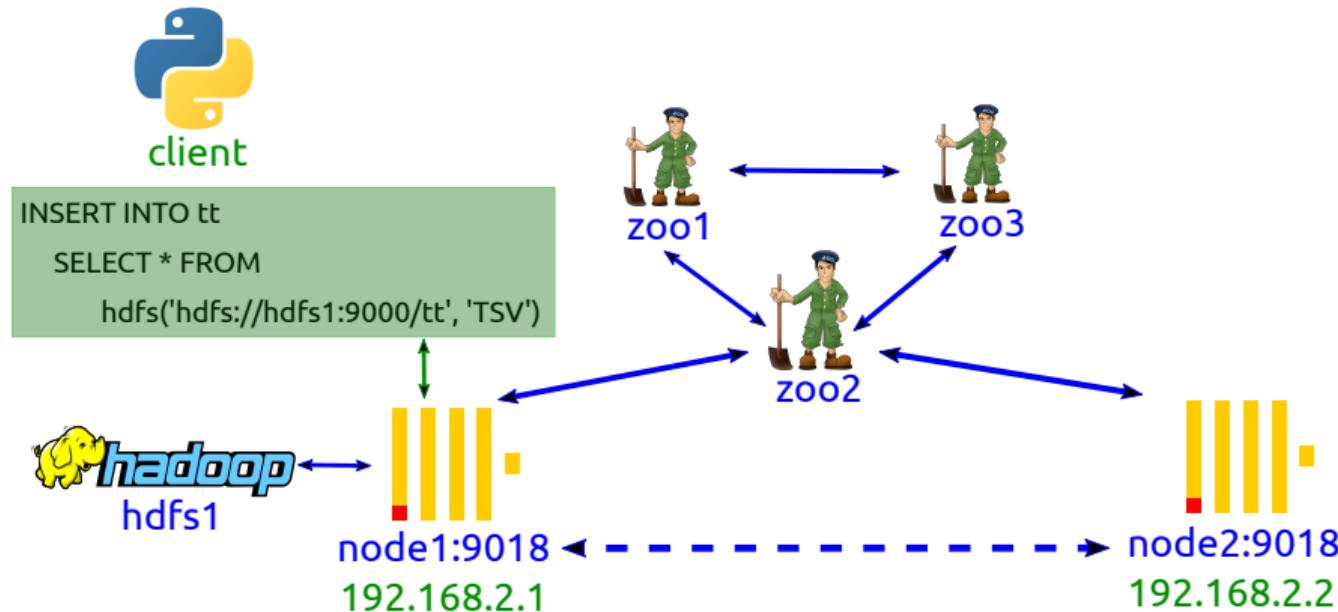
# Интеграционные тесты: пример



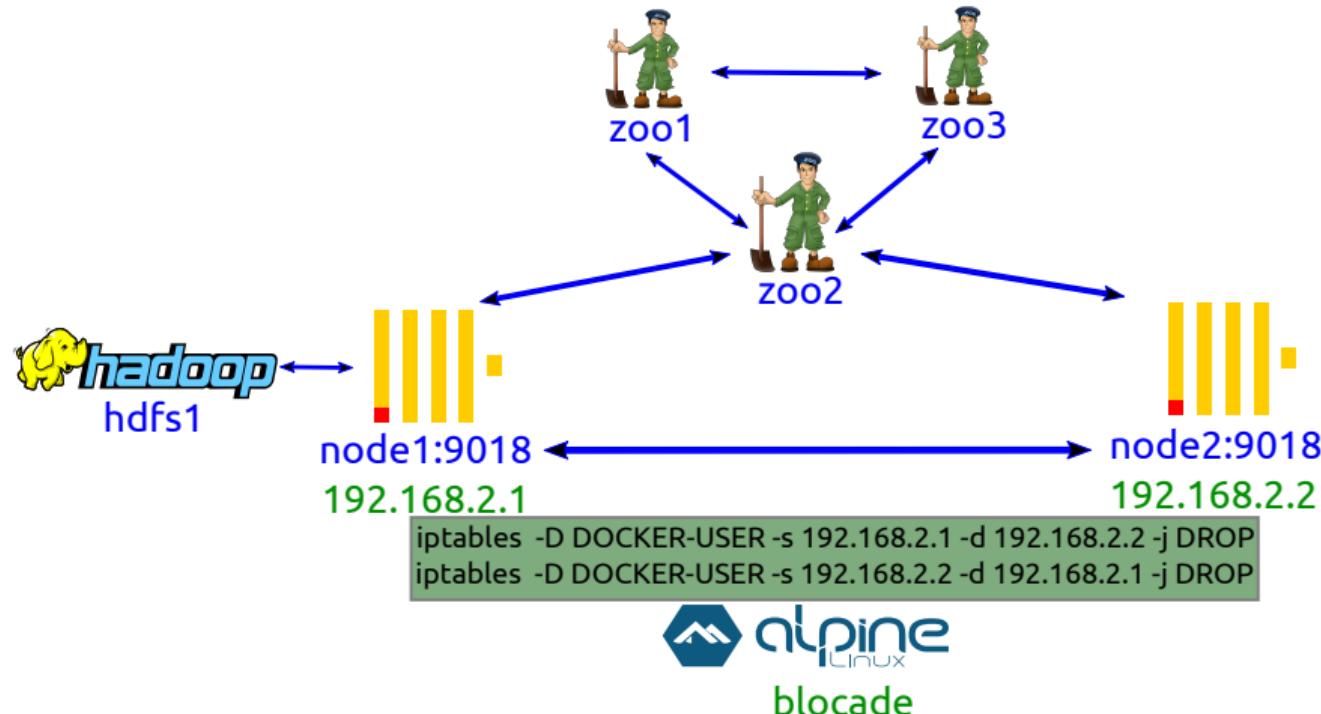
# Интеграционные тесты: пример



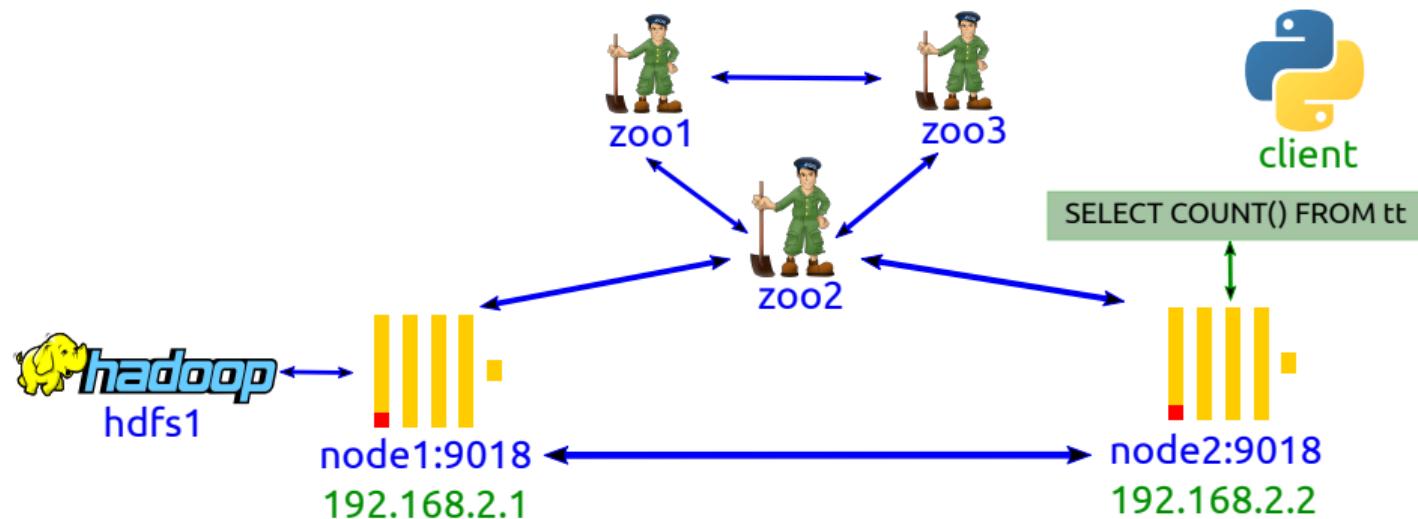
# Интеграционные тесты: пример



# Интеграционные тесты: пример



# Интеграционные тесты: пример



# Интеграционные тесты: код

```
cluster = ClickHouseCluster('__file__')
node1 = cluster.add_instance(name='node1', with_hdfs=True, with_zk=True)
node2 = cluster.add_instance(name='node2', with_zk=True)
```

# Интеграционные тесты: код

```
cluster = ClickHouseCluster('__file__')
node1 = cluster.add_instance(name='node1', with_hdfs=True, with_zk=True)
node2 = cluster.add_instance(name='node2', with_zk=True)

# Create tables here
```

# Интеграционные тесты: код

```
cluster = ClickHouseCluster('__file__')
node1 = cluster.add_instance(name='node1', with_hdfs=True, with_zk=True)
node2 = cluster.add_instance(name='node2', with_zk=True)

# Create tables here

def test_drop_failover(started_cluster):
    with PartitionManager() as pm:
        pm.partition_instances(node1, node2, port=9009) #drop connection
        node1.query("INSERT INTO tt \
                    SELECT * FROM hdfs('hdfs://hdfs1:9000/tt', 'TSV')")
    assert_with_retry(node2, "SELECT COUNT() FROM tt", "1")
```

# Интеграционные тесты: запуск

## Проблема:

- › Нужны python-библиотеки
- › Может остаться мусор от iptables

# Интеграционные тесты: запуск

## Проблема:

- › Нужны python-библиотеки
- › Может остаться мусор от iptables

## Решение:

- › Docker in docker!
- › Образ с библиотеками
- › Запуск в изолированной сети
- › <https://github.com/jpetazzo/dind>



Inception (2010)

# Про производительность

# Особенности исполнения запросов

- › Утилизируется весь пул treadов
- › Треды воруют задачи друг у друга
- › Используется умный аллокатор
- › Нет внутреннего кэша
- › Запросы могут разгоняться

# Что влияет на производительность

## Диск:

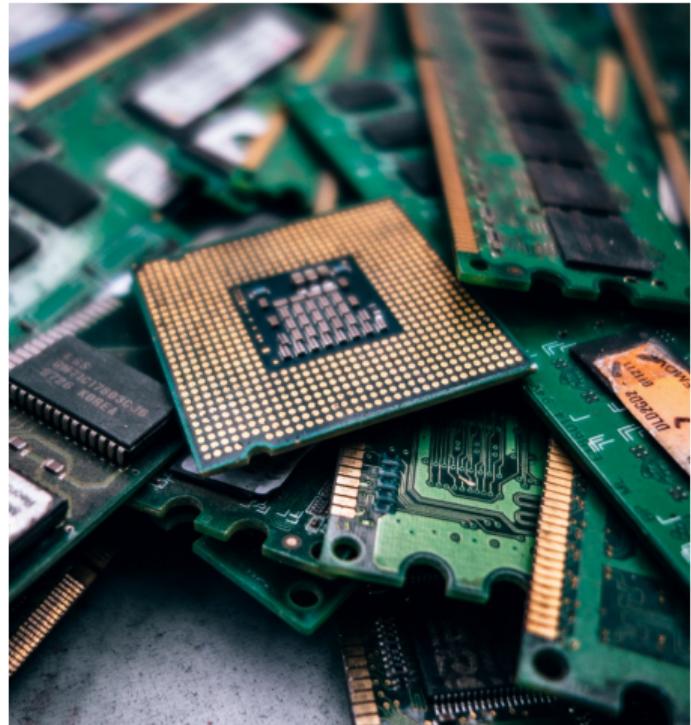
- › Кэш OS
- › Уровень RAID и состояние диска
- › Положение данных на диске

## Память:

- › Аллокатор
- › Объем

## CPU:

- › Количество ядер
- › Размер кэшей
- › Планирование



[https://cdn-images-1.medium.com/max/2600/1\\*10rhXUnMtGFXrRqcaEcoA.jpeg](https://cdn-images-1.medium.com/max/2600/1*10rhXUnMtGFXrRqcaEcoA.jpeg)

# Как измерять производительность

- › Исключить влияние диска и кэшей
- › Фиксировать CPU и память
- › Подбирать условия останова
- › Проверять запросы дольше 10ms
- › Использовать реальные данные
- › Точно замерять время



# Тесты производительности: инструмент

## [clickhouse-performance-test](#)

Программа на C++ – альтернативный клиент ClickHouse.

- › Декларативная конфигурация тестов
- › Выполнение запросов с подстановками
- › Режимы выполнения
  - В цикле
  - Бесконечно

# Тесты производительности: инструмент

## [clickhouse-performance-test](#)

Программа на C++ – альтернативный клиент ClickHouse.

- › Декларативная конфигурация тестов
- › Выполнение запросов с подстановками
- › Режимы выполнения
  - В цикле
  - Бесконечно
- › Метрики запросов
  - Максимальный/средний RPS
  - Квантили времени

# Тесты производительности: инструмент

## [clickhouse-performance-test](#)

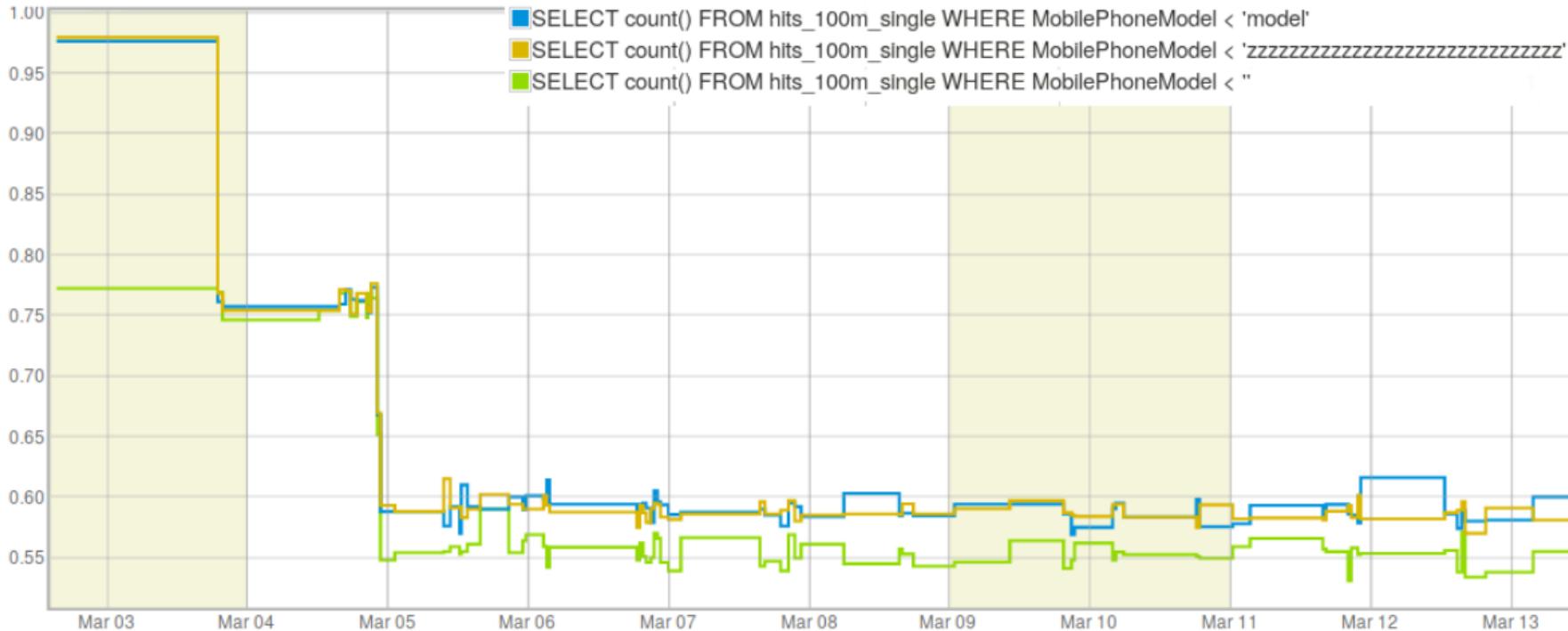
Программа на C++ – альтернативный клиент ClickHouse.

- › Декларативная конфигурация тестов
- › Выполнение запросов с подстановками
- › Режимы выполнения
  - В цикле
  - Бесконечно
- › Метрики запросов
  - Максимальный/средний RPS
  - Квантили времени
- › Интервальные условия останова
  - Максимальный/средний RPS
  - Минимальное время

# Тесты производительности: запуск

- › Синтетические запросы для отдельных функций
- › Реальные запросы к реальным данным:
  - NYC Taxi
  - Airline On-Time
  - Данные Яндекс.Метрики
- › Сохраняются результаты лучшего запуска
- › Сравнивается разница между коммитами
- › На что реагируем:
  - просадка на отдельных запросах  $> 7\%$
  - просадка по серверу целиком  $> 1\%$

# Тесты производительности: графики



# Тесты производительности: борьба с шумом

- › Правильные условия останова
- › Данные для запросов в *tmpfs*
- › Ограничение размера пула treadов
- › Запуск на нескольких хостах
- › Перезапуски упавших запросов

# Тесты производительности: анализ запросов

## Запрос:

```
SELECT count() FROM system.numbers
WHERE
    NOT ignore(
        materialize('xxxxxxxxxxxxxxxxxxxx') AS s,
        concat(s, s, s, s, s, s, s, s, s) AS t,
        concat(t, t, t, t, t, t, t, t) AS u);
```

## Условия останова:

```
<any_of>
    <max_speed_not_changing_ms>3000</max_speed_not_changing_ms>
    <total_time_ms>10000</total_time_ms>
</any_of>
```

# Тесты производительности: анализ запросов

```
:) SELECT count() FROM numbers(5000000) WHERE NOT ignore ...  
Elapsed: 4.123 sec, processed 40.00 MB (1.21 million rows/s.)
```

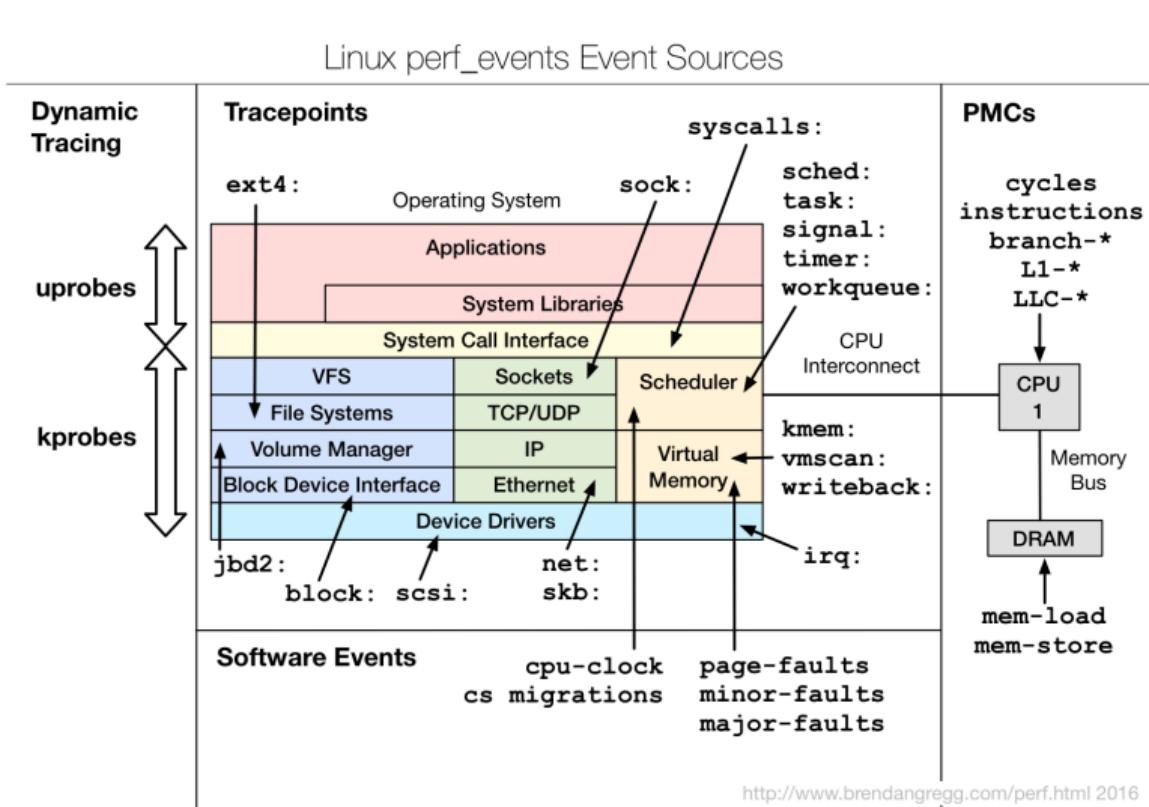
```
:) SELECT count() FROM numbers(5000000) WHERE NOT ignore ...  
Elapsed: 2.822 sec, processed 40.00 MB (1.77 million rows/s.)
```

```
:) SELECT count() FROM numbers(5000000) WHERE NOT ignore ...  
Elapsed: 2.648 sec, processed 40.00 MB (1.89 million rows/s.)
```

```
:) SELECT count() FROM numbers(5000000) WHERE NOT ignore ...  
Elapsed: 3.440 sec, processed 40.00 MB (1.45 million rows/s.)
```

```
:) SELECT count() FROM numbers(5000000) WHERE NOT ignore ...  
Elapsed: 3.261 sec, processed 40.00 MB (1.53 million rows/s.)
```

# Тесты производительности: анализ запросов



# Тесты производительности: анализ запросов

```
SELECT
    PE.Names, anyIf(v, slow) AS v_slow,
    anyIf(v, NOT slow) AS v_fast, v_slow / v_fast AS ratio
FROM
(
    SELECT
        PE.Names,
        query_duration_ms > 3000 AS slow,
        avg(PE.Values) AS value,
        FROM system.query_thread_log
        WHERE query = '...'
        GROUP BY PE.Names, slow
)
HAVING ratio > 1.1 ORDER BY ratio DESC
```

# Тесты производительности: анализ запросов

PE.Names	v_slow	v_fast	ratio
SoftPageFaults	10226.823	6060.816	1.6873
SystemTimeMicroseconds	10613.019	7759.116	1.368
RealTimeMicroseconds	2735841.077	2199059.022	1.244

## Выводы:

- › Аллокатор по другому выделяет память
- › Сделали запрос конечным
- › Выполняем в цикле

Про CI

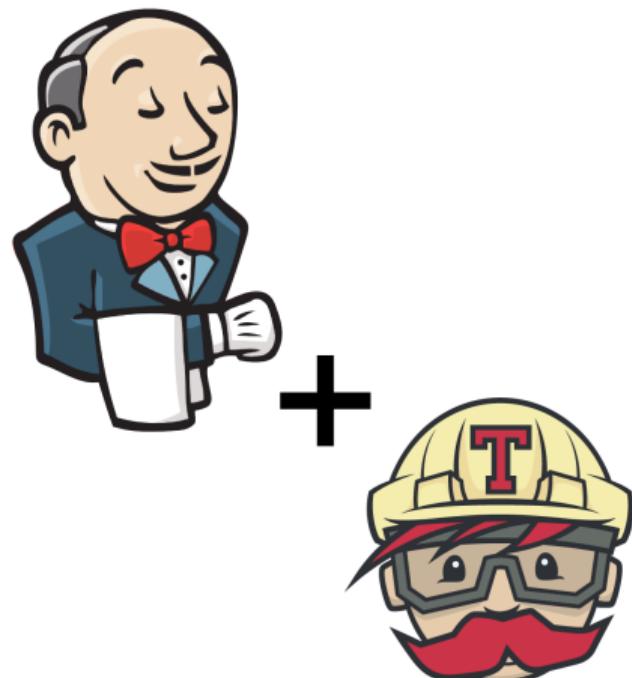
# CI в ClickHouse раньше

## | Внутренняя инсталляция Jenkins

- › Железные хосты
- › Сборки под разные версии Ubuntu
- › Функциональные тесты

## | Бесплатный план в Travis CI

- › 2 джоба по 2 ядра
- › Сборки под macOS и Ubuntu
- › Функциональные тестов



# Проблемы старого CI

## Недостатки Jenkins:

- › Не тестировались внешние пулл реквесты
- › Не было возможности запуска тяжелых тестов
- › Сборки не сохранялись

# Проблемы старого CI

## Недостатки Jenkins:

- › Не тестировались внешние пулл реквесты
- › Не было возможности запуска тяжелых тестов
- › Сборки не сохранялись

## Фундаментальные недостатки Jenkins:

- › Job DSL или программирование на HTML
- › Не самый приятный интерфейс
- › Система плагинов

# Проблемы старого CI

## Недостатки Jenkins:

- › Не тестировались внешние пулл реквесты
- › Не было возможности запуска тяжелых тестов
- › Сборки не сохранялись

## Фундаментальные недостатки Jenkins:

- › Job DSL или программирование на HTML
- › Не самый приятный интерфейс
- › Система плагинов

## Фундаментальные недостатки Travis:

- › Очень маленькие мощности, даже за деньги
- › Трудновоспроизводимые запуски тестов

# CI в open-source СУБД

# MongoDB

- › Разработка на GitHub
- › Изменения через пулл реквесты
- › Для CI используется Evergreen

## Понравилось:

- ✓ Артефакты доступны для скачивания
- ✓ Удобный интерфейс

## Не понравилось:

- ✗ Внутренняя JIRA
- ✗ Тесты запускаются на ветки



# PostgreSQL

- › Разработка в собственном git-репозитории
- › Изменения через mailing-листы
- › Распределенная система BuildFarm

*This software is written in Perl. If you're not comfortable with Perl then you possibly don't want to run this ...*

— PostgreSQL Buildfarm Howto

## Понравилось:

- ± Тестирование и сборка на хостах мейнтайнеров
- ✓ Можно смотреть логи (много логов)

## Не понравилось:

- ✗ Тесты запускаются на ветки
- ✗ Артефакты скачать нельзя

# Apache Ignite

- › Разработка на GitHub
- › Изменения через пулл реквесты
- › Для CI используется TeamCity

## Понравилось:

- ✓ Тесты на каждый пулл реквест
- ✓ Артефакты доступны для скачивания

## Не понравилось:

- ✗ Сложный интерфейс, тормозит
- ✗ Сотни падающих тестов
- ✗ Большая очередь, долгие запуски



# Антипаттерны CI

- › Инструкция для контрибьютора
  - CI нужно искать
  - Отдельный issue-трекер

# Антипаттерны CI

- › Инструкция для контрибьютора
  - CI нужно искать
  - Отдельный issue-трекер
- › Флатающие тесты
- › Недоступные артефакты
- › Невозможность повторить запуск

# Антипаттерны CI

- › Инструкция для контрибьютора
  - CI нужно искать
  - Отдельный issue-трекер
- › Флатающие тесты
- › Недоступные артефакты
- › Невозможность повторить запуск
- › Часы на прогон тестов
- › Тонны нерелевантных логов
- › Запуск на выбранные бранчи

# CI в ClickHouse

# Основные требования

- › Интерфейс очевиден для контрибьютора
- › Артефакты сборок и тестов доступны всем
- › Долгие и тяжелые тесты на каждый коммит
- › Масштабируемо на десятки активных разработчиков
- › Сборки и тесты воспроизводимы локально
- › Код, тесты и данные доступны открыто

## Цель

Сделать возможным для внешних контрибьюторов самостоятельно работать над задачами, используя возможности нашей инфраструктуры.

# GitHub как интерфейс CI

- › Очевиден для контрибуторов
- › Мощный API
- › Статусы коммитов
- › Ссылки в статусах
- › Множество способов взаимодействия

 **Some checks haven't completed yet**  
4 pending and 19 successful checks

	 <b>Functional stateless tests (thread)</b> Pending — Started	<a href="#">Details</a>
	 <b>Integration tests (asan)</b> Pending — Started	<a href="#">Details</a>
	 <b>Integration tests (release)</b> Pending — Started	<a href="#">Details</a>
	 <b>Performance test</b> Pending — Started	<a href="#">Details</a>
	 <b>ClickHouse build check</b> — 10/10 builds are OK	<a href="#">Details</a>

# Отчеты и артефакты

- › Хранилище – внутренний S3
- › Статические HTML-страницы
- › Хранение на каждый коммит
- › Ссылка на отчет в статусах
- › Артефакты сборки становятся релизными

## ClickHouse Unit Tests for PR #5267

Test name	Test status
zkutil.zookeeper_connected	OK
zkutil.multi_nice_exception_msg	OK
zkutil.multi_async	OK
zkutil.watch_get_children_with_chroot	OK
zkutil.multi_create_sequential	OK
Common.RWLock_1	OK
Common.RWLock_Recursive	OK

[test\\_run.txt.out.txt](#) [PR #5267](#) [Commit](#) [test\\_result.txt](#) [Task \(private network\)](#)

# Инфраструктурное облако поиска

- › Как TeamCity, но проще
- › Запуск произвольного кода на python (задачи)
- › Фиксирование характеристик хостов (CPU, RAM, OS)
- › Устойчивость к выпадению хостов
- › Хранение и поиск артефактов
- › Запуск задач по таймеру (cron)
- › Произвольные атрибуты артефактов

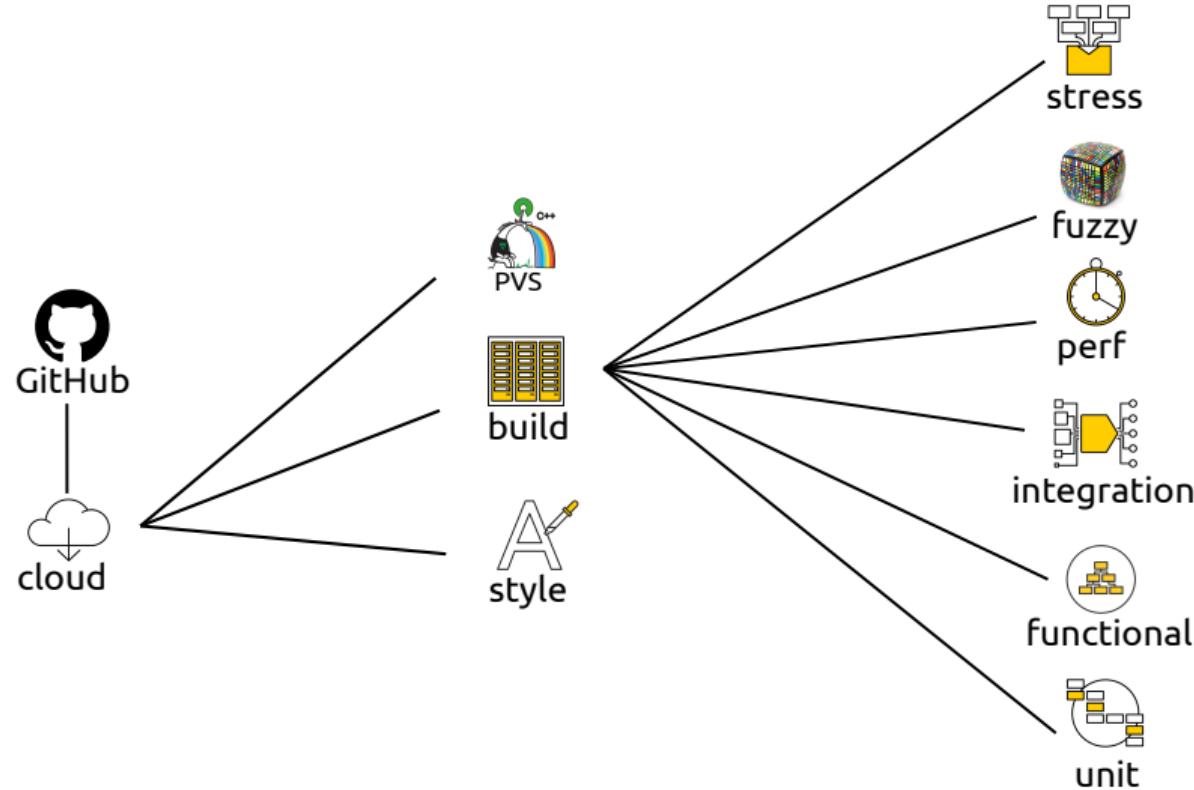
# Безопасность

- › Запуск после подтверждения
- › Задачи изолированы по диску
- › Все в Docker
- › Таймауты на выполнение задач
- › Ограничение сетевого доступа



[https://energyassault.com/wp-content/uploads/2018/03/shutterstock\\_66670864.jpg](https://energyassault.com/wp-content/uploads/2018/03/shutterstock_66670864.jpg)

# Pipeline



# Результаты



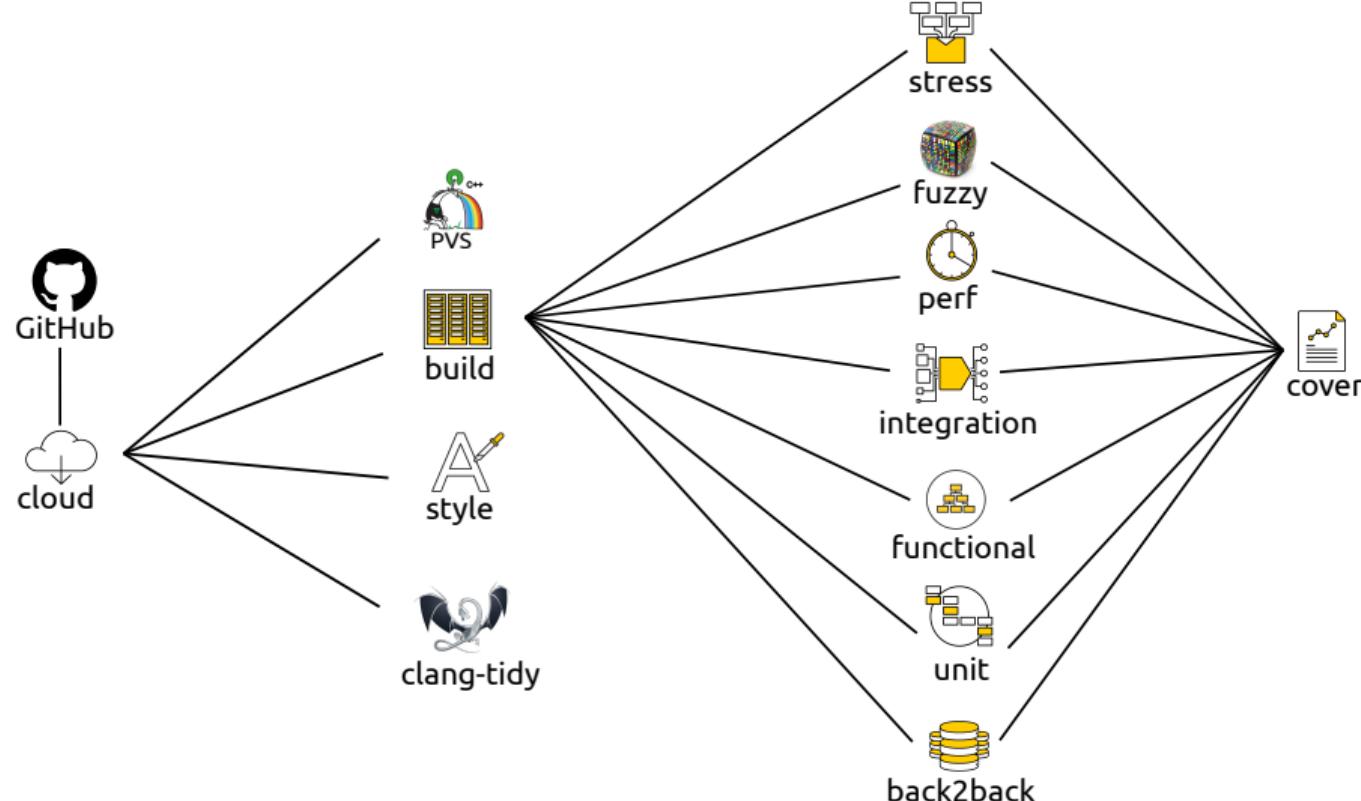
# Issues

- ➊ Excessive connections when using foreign tables. comp-foreign-db performance  
#5336 opened 15 hours ago
- ➊ clickhouse-local without parameters does nothing easy task prio-minor usability  
#5335 opened 16 hours ago
- ➊ S3 table function and storage engine. feature  
#5333 opened 18 hours ago
- ➊ Query optimization for GROUP BY with ORDER BY keys and LIMIT. feature  
#5329 opened a day ago
- ➊ enable\_optimize\_predicate\_expression & arrayJoin can be improved feature  
#5327 opened 2 days ago

# Чего не хватает

- › Тестовое покрытие кода
- › Страница со статистикой по тестам
- › Сборка и тесты с memory-санитайзером
- › **Back2Back-тесты** с запросами и данными от пользователей
- › Комбинирование настроек в функциональных тестах
- › Одновременный тест производительности двух версий

# Full Pipeline



# Но есть и хорошие новости

- › Чаще обнаруживаем баги до релиза
- › Мержим больше пулл реквестов
- › Не допускаем просадок производительности
- › Ускорили выкладку релиза до единиц минут
- › Поддерживаем двухнедельный релизный цикл
- › Бекпортируем изменения в старые версии
- › Стали появляться контрибьюты в CI!

Спасибо

QA