

ClickHouse для инженеров и архитекторов БД

Построение production-ready хранилища данных Такси на базе ClickHouse



**Меня хорошо видно
& слышно?**



Защита проекта

Тема: Построение production-ready хранилища данных Такси на базе ClickHouse



Константин Соколов

Дата инженер в компании Wildberries

План защиты

Цель и задачи проекта

Какие технологии использовались

Что получилось

Выводы

Вопросы и рекомендации

Цель и задачи проекта

Цель проекта: Построение production-ready хранилища данных Такси на базе ClickHouse

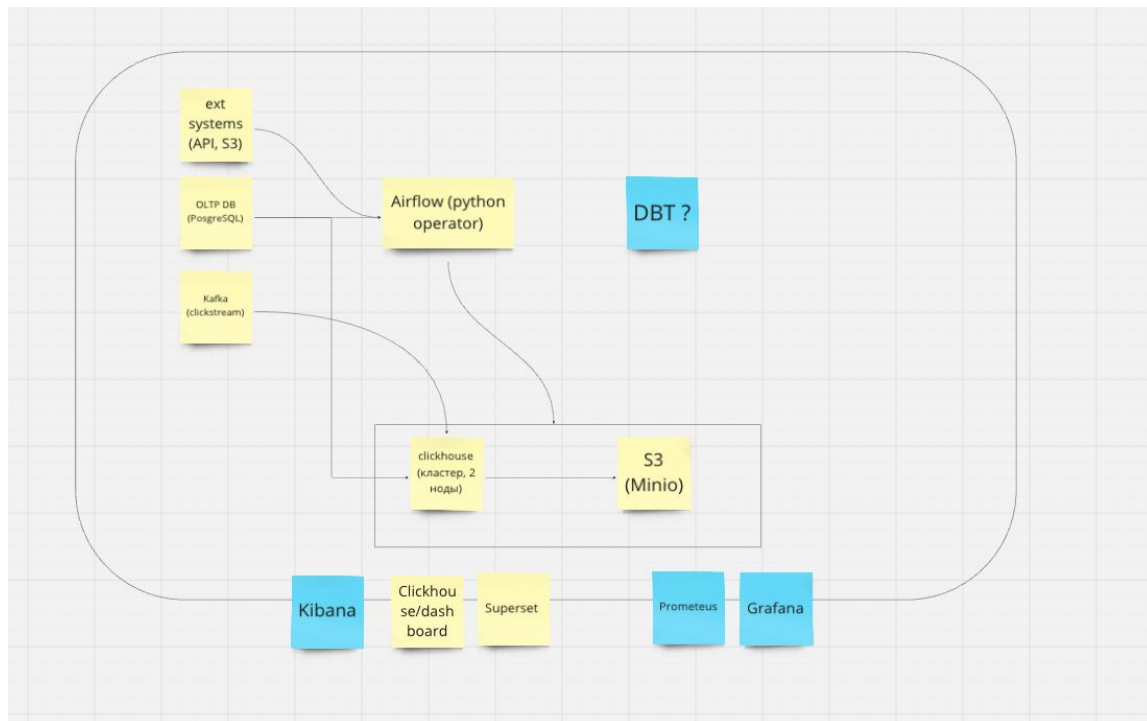
1. Построение production-ready хранилища данных
2. Визуализация данных в виде дашборда
3. Настройка RBAC, бэкапов
4. Интеграция с внешними системами

Какие технологии использовались

1. СУБД - Clickhouse, PostgreSQL
2. Airflow - оркестрация
3. Kafka - шина данных
4. Minio - для бэкапов и для storage policies
5. Superset - для визуализаций



Что получилось

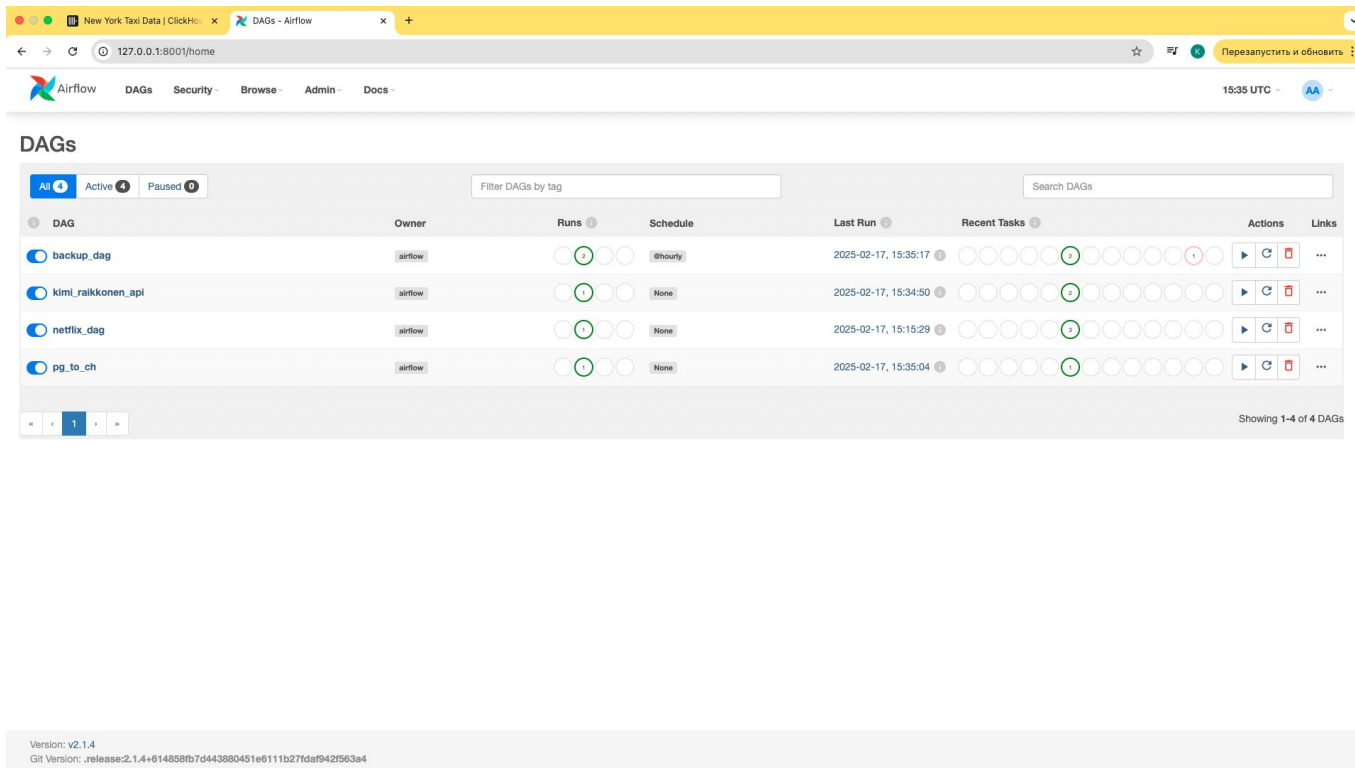


Структура БД

- streams - База данных с консьюмерами кафки
- raw - База данных с сырыми данными из кафки (данные хранятся как json в строке + метаданные кафки)
- parsed - База данных с распаршенными данными из кафки
- dict - База данных со словарями
- airflow_metadata - База данных с метаданными Airflow (движок PostgreSQL)
- ext - База данных, куда складываем данные из внешних систем (апи, парсинг и тд)
- prod - База данных, куда складываем данные из прода (ОЛТП базы данных)
- datamart - Основная БД для запросов со стороны BI. Единственная таблица - datamart.trips
- dashboard - БД для мониторинга



ETL pipelines



The screenshot displays the Apache Airflow web interface. The top navigation bar includes links for DAGs, Security, Browse, Admin, and Docs. The main content area is titled "DAGs" and features a table listing four DAGs: backup_dag, kimi_raikkonen_api, netflix_dag, and pg_to_ch. Each row shows the DAG name, owner (airflow), status (Active), schedule, last run time, and recent task status. The backup_dag is scheduled @hourly and last ran on 2025-02-17 at 15:35:17. The kimi_raikkonen_api, netflix_dag, and pg_to_ch are scheduled None and last ran on 2025-02-17 at 15:34:50, 15:15:29, and 15:35:04 respectively. The interface also includes a search bar for DAGs and a pagination control at the bottom.

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
backup_dag	airflow	1	@hourly	2025-02-17, 15:35:17	1	[Play] [Refresh] [Stop]	...
kimi_raikkonen_api	airflow	1	None	2025-02-17, 15:34:50	1	[Play] [Refresh] [Stop]	...
netflix_dag	airflow	1	None	2025-02-17, 15:15:29	1	[Play] [Refresh] [Stop]	...
pg_to_ch	airflow	1	None	2025-02-17, 15:35:04	1	[Play] [Refresh] [Stop]	...

Showing 1-4 of 4 DAGs

Version: v2.1.4
Git Version: .release:2.1.4+614858fb7d443880451e6111b27fdaf942f963a4



ETL pipelines

Также данные поступают из:

- Кафки (KafkaEngine + каскад MV)
- PostgreSQL - за счет словарей и за счет таблицы с движком PostgreSQL
- S3 (разовая вставка за счет табличной функции s3)

RBAC

До создания объектов базы раскатываются объекты RBAC. В проекте представлены следующие сущности:

- пользователи
- роли
- права
- квоты
- сеттинги

Также создал именованную коллекцию для упрощения доступа

Backups and Storage policies

Для холодного хранения данных и для бэкапов были подключено объектное хранилище. В качестве объектного хранилища остановился на Minio (легкая настройка, open-source). Конфиги представлены в примонтированных директориях, чтобы убедиться, что все работает, проверим системные таблицы:

```
select policy_name, volume_name from system.storage_policies;
```

```
/*
```

	policy_name	volume_name
1.	default	default
2.	default	s3_backup
3.	s3_backup	s3_backup

```
3 rows in set. Elapsed: 0.009 sec.
```

```
*/
```



Backups and Storage policies

Бэкапы осуществляются ежечасно по расписанию через эирфлоу. Раз в неделю в 3 ночи по субботам снимаются полные бэкапы, остальные - инкрементальные. Бэкапы выполняются за счет sql (в эирфлоу):

```
-- full backup
BACKUP ALL ON CLUSTER otus TO Disk('s3_backup', 'backup_20250216');

-- incremental
BACKUP ALL ON CLUSTER otus TO Disk('s3_backup', 'backup_20250216_08')
  SETTINGS base_backup = Disk('s3_backup', 'test_backups')
```



Backups and Storage policies

Для холодного хранения настроен сторадж.

```
SELECT
    table,
    disk_name,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size,
    formatReadableSize(sum(data_uncompressed_bytes)) AS uncompressed_size,
    sum(rows) AS total_rows
FROM system.parts
WHERE active = 1 and table = 'trips'
GROUP BY table, disk_name
ORDER BY table, disk_name;
/*


|    | table | disk_name | compressed_size | uncompressed_size | total_rows |
|----|-------|-----------|-----------------|-------------------|------------|
| 1. | trips | default   | 674.00 B        | 4.72 KiB          | 100        |
| 2. | trips | s3_cold   | 120.71 MiB      | 206.83 MiB        | 3000317    |


2 rows in set. Elapsed: 0.089 sec.
*/
```



Backups and Storage policies

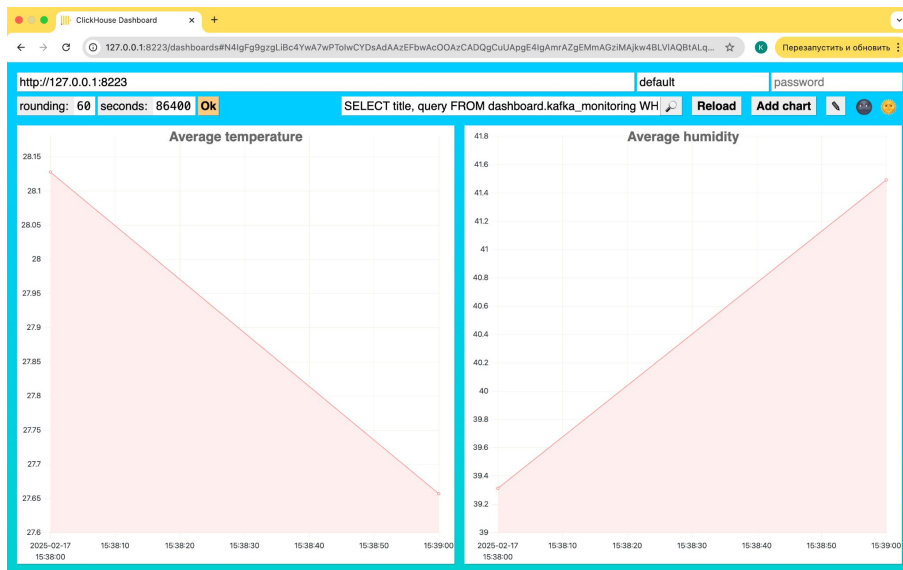


The screenshot shows the MiniIO Console interface in a web browser. The address bar displays the URL `127.0.0.1:10001/browser/clickhouse`. The page header includes the MiniIO logo, "OBJECT STORE", and a "Login" link. A search bar at the top contains the text "Start typing to filter objects in the bucket". Below the search bar, the bucket name "clickhouse" is displayed. A table lists the objects in the bucket, with columns for "Name", "Last Modified", and "Size". The table contains 16 rows of objects, each with a folder icon and a name starting with 'a' followed by two letters. The "Last Modified" and "Size" columns show dashes. At the top right of the table, there are "Refresh" and "Upload" buttons.

Name	Last Modified	Size
adi	-	-
ady	-	-
afg	-	-
agc	-	-
aob	-	-
apd	-	-
anq	-	-
asa	-	-
aui	-	-
baf	-	-
bag	-	-
bbq	-	-
bcq	-	-
bhn	-	-
bho	-	-
bhz	-	-

Мониторинг

Промежуточный вариант - встроенный мониторинг на эндпоинте dashboards. Допустим, мы следим за событиями из кафки (данные с каких то датчиков). Для мониторинга заходим в браузере на `127.0.0.1:8223/dashboards`



Что дальше?

- Мониторинг системы: логи сейчас идут в специальные таблицы, можно рассмотреть использование ELK-стека либо использовать отдельный клик для логов кластера
- мониторинг в Графанае
- Vault для хранения секретов
- DBT для Data Lineage
- сбор витрин для кликхауса - вынести (Vertica/Greenplum или Trino + Iceberg или Spark on K8S)



Выводы

1. Кликхаус отлично интегрируется с внешними системами
2. До определенных объемов можно прекрасно обходиться встроенным мониторингом
3. Анализ логов кликхауса можно также выполнять внутри самого кликхауса



Вопросы и рекомендации



если есть вопросы



если вопросов нет

Спасибо за внимание!

