ООО «Астра Консалтинг»

Астра ИС МД (Инфраструктурные Сервисы)

Модуль Astra ClickHouse

ДОКУМЕНТАЦИЯ, СОДЕРЖАЩАЯ ИНФОРМАЦИЮ, НЕОБХОДИМУЮ ДЛЯ ЭКСПЛУАТАЦИИ ЭКЗЕМПЛЯРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ПРЕДОСТАВЛЕННОГО ДЛЯ ПРОВЕДЕНИЯ ЭКСПЕРТНОЙ ПРОВЕРКИ

Москва, 2026

Содержание

1 Область применения

Настоящий документ содержит описание функциональных характеристик программного обеспечения, модуль Astra ClickHouse, далее Astra ClickHouse развернутого в архитектуре с 3-мя серверными узлами и 3-мя координаторами с использованием Astra ClickHouse Keeper на тех же серверах.

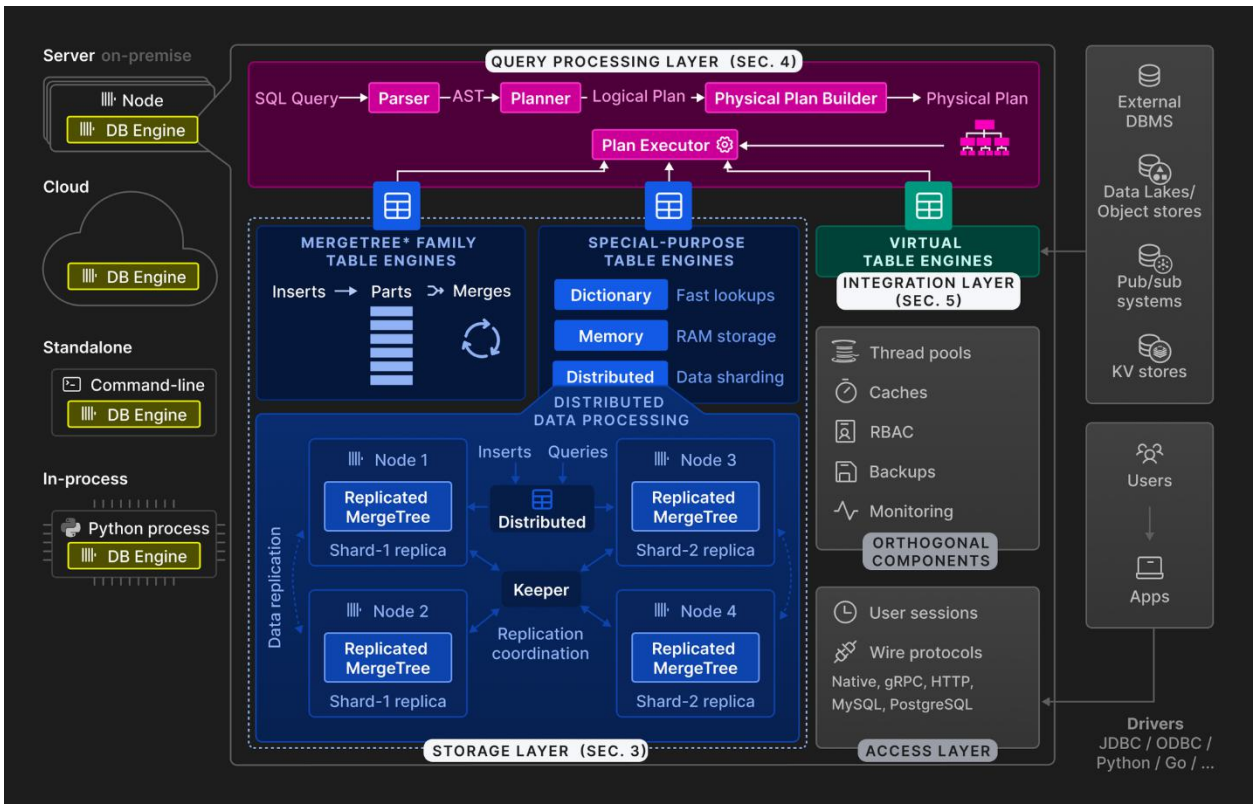
Документ предназначен для системных администраторов и разработчиков, осуществляющих эксплуатацию и обслуживание системы управления базами данных Astra ClickHouse.

1.1 Архитектура системы

Astra ClickHouse — это популярная OLAP СУБД с открытым исходным кодом, предназначенная для высокопроизводительной аналитики наборов данных петабайтного масштаба с интенсивными вставками данных. Ее система хранения комбинирует формат данных на основе традиционных LSM-деревьев (log-structured merge trees) с современными техниками с постоянной трансформацией (т.е. агрегацией, архивированием) исторических данных в фоновом режиме. Запросы пишутся на удобном диалекте языка SQL и обрабатываются векторным ядром выполнения запросом с опциональной компиляцией кода. Astra ClickHouse выполняет агрессивные техники исключения обработки ненужных для результата данных. Остальные СУБД могут быть интегрированы на уровне табличных функций, табличных или уровня базы ядер обработки (table или database engine). Бенчмарки на реальных данных демонстрируют, что Astra ClickHouse является одной из самых быстрых аналитических СУБД на рынке.

Astra ClickHouse предназначен для решения пяти ключевых вызовов при обработке современных аналитических данных:

- Огромные наборы данных с высокой скоростью вставки данных
- Много параллельных запросом с ожиданием низких задержек от них
- Разнообразный ландшафт хранилищ данных, их расположение и форматы
- Удобный язык запросов с возможностью анализа их производительности
- Надёжность и разнообразное развёртывание



На рисунке выше показана высокоуровневая архитектура СУБД Astra ClickHouse.

Ядро Astra ClickHouse разбито на 3 главных уровня: уровень выполнения запросов (описан в Секции 4), уровень хранения (Секция 3) и интеграционный уровень (Секция 5). Кроме того, уровень доступа управляет пользовательскими сессиями и коммуникацией с приложениями с помощью различных протоколов. Есть компоненты для управления потоками, кэшированием, контролем доступа на основе ролей, архивированием и постоянным мониторингом. Astra ClickHouse собран на языке C++ как единый статически связанный бинарный файл без зависимостей.

Система выполнения запросов следует традиционной парадигме разбора входящих запросов, построения и оптимизации логических и физических планов запросов. Запросы могут быть написаны на диалекте SQL с богатым функционалом под названием PRQL или Kusto's KQL.

Уровень хранения данных состоит из различных Табличных Ядер (table engines), которые включают формат и расположение данных таблицы.

Табличные Ядра подразделяются на 3 категории.

Первая категория это т.н. MergeTree* семейство Табличных Ядер, представляющая из себя основной формат хранения постоянных данных в Astra ClickHouse, основываясь на идее LSM деревьев, где таблицы разбиты на горизонтальные отсортированные Куски (parts), которые постоянно сливаются фоновыми процессами. Индивидуальные Табличные Ядра семейства MergeTree* отличаются способом, который используется для слияния строк из входных Кусков. Например, записи могут агрегироваться или перезаписываться при их устаревании.

Вторая категория — это специальные Табличные Ядра, предназначенные для ускорения или распределения запросов. Этак категория включает ядра обработки данных

типа ключ-значение в памяти. Словарь кэширует результат запроса и периодически выполняет его на внутренний или внешний источник данных. Это значительно уменьшает задержки доступа к данным в сценариях, где можно мириться с некоторой степенью устаревания данных. Другими примерами специализированных Табличных Ядер являются Ядра обработки данных только в памяти, используемых для обработки временных данных, и Распределённые (Distributed) Табличные Ядра для горизонтального масштабирования.

Третья категория — это виртуальные Табличные Ядра для двустороннего обмена данных с внешними системами, такими как реляционные СУБД (например, PostgreSQL, MySQL), системы подписок/публикаций (например, Kafka, RabbitMQ) или хранилища типа ключ/значение (например, Redis). Виртуальные Ядра могут также взаимодействовать с озёрами данных (data lakes, например, Iceberg, DeltaLake, Hudi) или файлами в объектном хранилище (например, AWS S3, Google GCP).

Astra ClickHouse поддерживает секционирование (sharding) и репликацию таблиц по нескольким узлам кластера для масштабирования и высокой доступности. Секционирование делит таблицу на набор секций согласно выражению распределения. Все секции являются взаимно независимыми друг от друга таблицами, которые обычно располагаются на разных узлах кластера. Клиенты могут читать и писать секции напрямую, т.е. работать с ними как с независимыми таблицами, или использовать специальное Табличное Ядро Distributed, которое обеспечивает глобальное представление всех секций таблицы. Основное предназначение секционирования — это обработка наборов данных, которые превышают возможности отдельных узлов. Другое использование секционирования - распределения нагрузки на чтение-запись для таблицы по нескольким узлам, т.е. балансировка нагрузки. Кроме того, секцию можно реплицировать на несколько узлов для защиты от отказов узла. Для этого каждое Merge-Tree* Табличное Ядро имеет соответствующее ReplicatedMergeTree* Ядро, которое использует много-мастерную (multi-master) схему координации на основе консенсуса Raft (реализовано с помощью Astra ClickHouse Keeper - замены Apache Zookeeper, написанной на C++) для гарантии того, что каждая секция в каждый момент времени имеет сконфигурированное число копий. На рисунке выше, например, показана таблица из двух секций, каждая из которых имеет по 2 копии на разных узлах.

Кластер Astra ClickHouse может быть развёрнут на мощностях заказчика (on-premise), в облаке, как утилита (standalone) или во внутри-процессном (in-process) режимах. В on-premise режиме Astra ClickHouse устанавливается локально на одном или нескольких узлах с секционированием и/или репликацией. Клиенты работают с системой с помощью приватного протокола или через HTTP REST API. Режим утилиты превращает Astra ClickHouse в утилиту командной строки для анализа и преобразования файлов, предоставляя основанную на языке SQL альтернативу утилитам Unix, таким как cat и grep. While this requires no prior configuration, the standalone mode is restricted to a single server. Недавно разработан внутри-процессный режим (называемый chDB) для интерактивного анализа данных, например, с Jupyter notebooks, Pandas dataframes. chDB встраивает в процесс хозяина Astra ClickHouse как высоко производительное OLAP ядро. По сравнению

с остальными режимами это позволяет эффективнее передавать данные между ядром СУБД и приложением без их копирования, т.к. они работают в одном адресном пространстве.

1.2 Характеристики системы

На всех 3-х узлах.

Параметр	Значение
Версия Astra ClickHouse	25.8.18.1
Web интерфейс порт	8443
Координаторы	Astra ClickHouse Keeper
Конфигурационные каталоги	/etc/clickhouse-server/
Каталоги с данными	/var/lib/clickhouse
Каталоги логов	/var/log/clickhouse-{server,keeper}

2 Подключение к модулю Astra ClickHouse

Компоненты модуля Astra ClickHouse могут быть загружены с сайта *https://github.com/ClickHouse/ClickHouse/releases*

Это 3 пакета для соответствующего пакетного менеджера для ОС, на которую производится установка, с схематичными именами:

clickhouse-{client,common,server}_{{ версия }}_amd64.{deb,rpm}

2.1 Подключение через clickhouse-client

Утилита использует в качестве одного из параметров файл конфигурации, который по умолчанию имеет имя ~/.clickhouse-client/config.xml. Удобно создать его с содержимым ниже:

```
<config>
  <user>default</user>
  <password>P@ssw0rd</password>
  <secure>true</secure>
  <openSSL>
    <client>
      <caConfig>/etc/security/certs/truststore.pem</caConfig>
      <invalidCertificateHandler>
        <name>RejectCertificateHandler</name>
          </invalidCertificateHandler>
    </client>
  </openSSL>
</config>
```

И алиас:

```
     $ alias cc='clickhouse-client -h haribda-01.internal -f
PrettyCompactMonoBlock'
      25.8.18.1
```

Тогда команда подключения будет выглядеть так:

```
     $ cc -q "select version()"
        ┌─version()─┐
```

```
1. │ 25.8.18.1 │
   └─────────────┘
```

2.2 Подключение через curl

Удобно создать алиас для **curl**:

```
$ alias curl_ch='curl https://haribda-01.internal:8443 \
--cacert /etc/security/certs/truststore.pem \
--user "default:P@ssw0rd" --data-binary @-'
```

Тогда команда с использованием алиаса будет выглядеть так:

```
$ echo "select version()" | curl_ch
25.8.18.1
```

3 Основные команды модуля Astra ClickHouse

3.1 Получение информации о сервере

3.1.1 Общая информация о кластерах

Так, как их видит каждый хост (host_).

```
$ cc -q "select hostName() as host_, host_name, cluster, shard_num, replica_num, is_local, errors_count from clusterAllReplicas('{cluster}', system.clusters) order by host_, cluster, shard_num, replica_num"
```

	host_	host_name	cluster	shard_num	replica_num	is_local	errors_count
1.	haribda-01.internal	haribda-01.internal	s1r3	1	1	1	0
2.	haribda-01.internal	haribda-02.internal	s1r3	1	2	0	0
3.	haribda-01.internal	haribda-03.internal	s1r3	1	3	0	0
4.	haribda-01.internal	haribda-01.internal	s3r1	1	1	1	0
5.	haribda-01.internal	haribda-02.internal	s3r1	2	1	0	0
6.	haribda-01.internal	haribda-03.internal	s3r1	3	1	0	0
7.	haribda-02.internal	haribda-01.internal	s1r3	1	1	0	0
8.	haribda-02.internal	haribda-02.internal	s1r3	1	2	1	0
9.	haribda-02.internal	haribda-03.internal	s1r3	1	3	0	0
10.	haribda-02.internal	haribda-01.internal	s3r1	1	1	0	0
11.	haribda-02.internal	haribda-02.internal	s3r1	2	1	1	0
12.	haribda-02.internal	haribda-03.internal	s3r1	3	1	0	0
13.	haribda-03.internal	haribda-01.internal	s1r3	1	1	0	0
14.	haribda-03.internal	haribda-02.internal	s1r3	1	2	0	0
15.	haribda-03.internal	haribda-03.internal	s1r3	1	3	1	0
16.	haribda-03.internal	haribda-01.internal	s3r1	1	1	0	0
17.	haribda-03.internal	haribda-02.internal	s3r1	2	1	0	0
18.	haribda-03.internal	haribda-03.internal	s3r1	3	1	1	0

3.1.2 Вывод всех пользователей, ролей, профилей, всех прав

Выдаётся информация в виде SQL команд, которые выполнялись явно или неявно (согласно конфигурационным настройкам) ранее.

```
$ cc -q "show access"
```

	ACCESS
1.	CREATE USER default IDENTIFIED WITH sha256_password SETTINGS PROFILE `default`
2.	CREATE USER grafana IDENTIFIED WITH sha256_password SETTINGS PROFILE `readonly`
3.	CREATE SETTINGS PROFILE `default`

```
4. | CREATE SETTINGS PROFILE `readonly` SETTINGS readonly = 1, max_execution_time CHANGEABLE_IN_READONLY |
5. | CREATE QUOTA default KEYED BY user_name FOR INTERVAL 1 hour TRACKING ONLY TO default                 |
6. | GRANT ALL ON *.* TO default WITH GRANT OPTION                                                        |
7. | GRANT SHOW, SELECT ON *.* TO grafana                                                                 |
   └──────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

3.1.3 Список Баз Данных

```
$ cc -q "show databases"

   ┌─name──────────────┐
1. | INFORMATION_SCHEMA |
2. | default            |
3. | information_schema |
4. | system             |
   └───────────────────┘
```

3.1.4 Создание Базы Данных

База данных **testdb**.

```
$ echo "create database testdb on cluster '{cluster}'" | curl_ch

haribda-02.internal     9440    0               2       0
haribda-03.internal     9440    0               1       0
haribda-01.internal     9440    0               0       0
```

3.1.5 Создание пользователя и выдача ему прав

Создание пользователя **test**:

```
$ cc -q "CREATE USER test ON CLUSTER '{cluster}' IDENTIFIED BY 'P@ssw0rd'"

   ┌─host──────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
1. | haribda-01.internal | 9440 |      0 |       |                   2 |                0 |
2. | haribda-03.internal | 9440 |      0 |       |                   1 |                0 |
3. | haribda-02.internal | 9440 |      0 |       |                   0 |                0 |
   └───────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘
```

3.1.6 Выдача прав пользователю

Пользователю test в базе testdb.

```
$ cc -q "GRANT ALL ON testdb.* to test on cluster '{cluster}'"
```

```
   ┌─host──────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
1. │ haribda-01.internal │ 9440 │      0 │       │                   2 │                0 │
2. │ haribda-03.internal │ 9440 │      0 │       │                   1 │                0 │
3. │ haribda-02.internal │ 9440 │      0 │       │                   0 │                0 │
   └───────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘
```

3.1.7 Информация о кластере

Имя кластера - s1r3.

```
$ cc -q "show cluster s1r3"

   ┌─cluster─┬─shard_num─┬─replica_num─┬─host_name─────────┬─host_address─┬─port─┐
1. │ s1r3    │         1 │           1 │ haribda-01.internal │ 10.128.0.11  │ 9440 │
2. │ s1r3    │         1 │           2 │ haribda-02.internal │ 10.128.0.12  │ 9440 │
3. │ s1r3    │         1 │           3 │ haribda-03.internal │ 10.128.0.13  │ 9440 │
   └─────────┴───────────┴─────────────┴─────────────────────┴──────────────┴──────┘
```

3.1.8 Информация о таблицах в Базе Данных

Имя БД - default.

```
$ cc -q "show tables in default"

   ┌─name──────────────────┐
1. │ asynchronous_metric_log │
   └─────────────────────────┘
```

3.2 Работа с таблицами

3.2.1 Загрузка данных

Создадим тестовую таблицу **trips** в базе данных **testdb**. База данных должна быть предварительно создана (см. выше).

В текущем каталоге создадим файл **trips_create.sql** с содержимым ниже:

```
CREATE TABLE testdb.trips on cluster '{cluster}'

(

    `trip_id` UInt32,

    `vendor_id` Enum8('1' = 1, '2' = 2, '3' = 3, '4' = 4,
'CMT' = 5, 'VTS' = 6, 'DDS' = 7, 'B02512' = 10, 'B02598' = 11,
'B02617' = 12, 'B02682' = 13, 'B02764' = 14, '' = 15),

    `pickup_date` Date,

    `pickup_datetime` DateTime,
```

```
        `dropoff_date` Date,
        `dropoff_datetime` DateTime,
        `store_and_fwd_flag` UInt8,
        `rate_code_id` UInt8,
        `pickup_longitude` Float64,
        `pickup_latitude` Float64,
        `dropoff_longitude` Float64,
        `dropoff_latitude` Float64,
        `passenger_count` UInt8,
        `trip_distance` Float64,
        `fare_amount` Float32,
        `extra` Float32,
        `mta_tax` Float32,
        `tip_amount` Float32,
        `tolls_amount` Float32,
        `ehail_fee` Float32,
        `improvement_surcharge` Float32,
        `total_amount` Float32,
        `payment_type` Enum8('UNK' = 0, 'CSH' = 1, 'CRE' = 2,
'NOC' = 3, 'DIS' = 4),
        `trip_type` UInt8,
        `pickup` FixedString(25),
        `dropoff` FixedString(25),
        `cab_type` Enum8('yellow' = 1, 'green' = 2, 'uber' = 3),
        `pickup_nyct2010_gid` Int8,
        `pickup_ctlabel` Float32,
        `pickup_borocode` Int8,
        `pickup_ct2010` String,
```

```
        `pickup_boroct2010` String,
        `pickup_cdeligibil` String,
        `pickup_ntacode` FixedString(4),
        `pickup_ntaname` String,
        `pickup_puma` UInt16,
        `dropoff_nyct2010_gid` UInt8,
        `dropoff_ctlabel` Float32,
        `dropoff_borocode` UInt8,
        `dropoff_ct2010` String,
        `dropoff_boroct2010` String,
        `dropoff_cdeligibil` String,
        `dropoff_ntacode` FixedString(4),
        `dropoff_ntaname` String,
        `dropoff_puma` UInt16
    )
    ENGINE = ReplicatedMergeTree
    PARTITION BY toYYYYMM(pickup_date)
    ORDER BY pickup_datetime;
```

Выполним его:

```
    $ cc --queries-file trips_create.sql
   ┌─host──────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
1. │ haribda-01.internal │ 9440 │      0 │       │                   2 │                0 │
2. │ haribda-03.internal │ 9440 │      0 │       │                   1 │                0 │
3. │ haribda-02.internal │ 9440 │      0 │       │                   0 │                0 │
   └───────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘
```

Загрузим 2 файла с данными в текущий каталог:

```
    $ curl -LO https://datasets-documentation.s3.eu-west-3.amazonaws.com/nyc-taxi/trips_1.gz
```

```
$ curl -LO https://datasets-documentation.s3.eu-west-3.amazonaws.com/nyc-taxi/trips_2.gz
```

Создадим файл **trips_insert.sql** с командой вставки данных в таблицу с содержимым:

```
INSERT INTO testdb.trips
SELECT *
FROM input("
    `trip_id` UInt32,
    `vendor_id` Enum8('1' = 1, '2' = 2, '3' = 3, '4' = 4, 'CMT' = 5, 'VTS' = 6, 'DDS' = 7, 'B02512' = 10, 'B02598' = 11, 'B02617' = 12, 'B02682' = 13, 'B02764' = 14, '' = 15),
    `pickup_date` Date,
    `pickup_datetime` DateTime,
    `dropoff_date` Date,
    `dropoff_datetime` DateTime,
    `store_and_fwd_flag` UInt8,
    `rate_code_id` UInt8,
    `pickup_longitude` Float64,
    `pickup_latitude` Float64,
    `dropoff_longitude` Float64,
    `dropoff_latitude` Float64,
    `passenger_count` UInt8,
    `trip_distance` Float64,
    `fare_amount` Float32,
    `extra` Float32,
    `mta_tax` Float32,
    `tip_amount` Float32,
    `tolls_amount` Float32,
    `ehail_fee` Float32,
```

```
        `improvement_surcharge` Float32,
        `total_amount` Float32,
        `payment_type` Enum8('UNK' = 0, 'CSH' = 1, 'CRE' = 2,
'NOC' = 3, 'DIS' = 4),
        `trip_type` UInt8,
        `pickup` FixedString(25),
        `dropoff` FixedString(25),
        `cab_type` Enum8('yellow' = 1, 'green' = 2, 'uber' = 3),
        `pickup_nyct2010_gid` Int8,
        `pickup_ctlabel` Float32,
        `pickup_borocode` Int8,
        `pickup_ct2010` String,
        `pickup_boroct2010` String,
        `pickup_cdeligibil` String,
        `pickup_ntacode` FixedString(4),
        `pickup_ntaname` String,
        `pickup_puma` UInt16,
        `dropoff_nyct2010_gid` UInt8,
        `dropoff_ctlabel` Float32,
        `dropoff_borocode` UInt8,
        `dropoff_ct2010` String,
        `dropoff_boroct2010` String,
        `dropoff_cdeligibil` String,
        `dropoff_ntacode` FixedString(4),
        `dropoff_ntaname` String,
        `dropoff_puma` UInt16
    ")
    FORMAT TabSeparatedWithNames
```

```
    SETTINGS input_format_try_infer_datetimes = 0
    ;
```

Выполним его дважды с использованием обоих загруженных файлов:

```
    $ gzip -dc trips_1.gz | cc --queries-file trips_insert.sql
    $ gzip -dc trips_2.gz | cc --queries-file trips_insert.sql
```

3.2.2 Вывод количества записей в таблице

Запрос на локальную таблицу.

Т.к. таблица имеет 3 реплики (создана как ReplicatedMergeTree в кластере **s1r3** из одной секции и 3-х реплик), то на 3-х узлах системы должно содержаться одинаковые данные. Проверить можно так:

```
    $ cc -q "select hostName() as host, *
    from clusterAllReplicas(
      '{cluster}'
    , view(
        select count()
        from testdb.trips
      )
    )"
   ┌─host──────────────────┬─count()─┐
1. │ haribda-01.internal │ 1999658 │
2. │ haribda-02.internal │ 1999658 │
3. │ haribda-03.internal │ 1999658 │
   └───────────────────────┴─────────┘
```

3.2.3 Список таблиц в Базе Данных

Для базы данных **testdb**.

```
    $ echo "show tables in testdb format PrettyCompactMonoblock"
| curl_ch
       ┌─name─┐
```

```
1. │ trips │
   └───────┘
```

3.2.4 Вывод N первых записей таблицы

```
$ $ cc -q "select * from testdb.trips where trip_id <> 0 limit
2"
Row 1:
──────
trip_id:                1201746944 -- 1.20 billion
vendor_id:              2
pickup_date:            2015-07-01
...
dropoff_ntacode:        MN09
dropoff_ntaname:        Morningside Heights
dropoff_puma:           3802

Row 2:
──────
trip_id:                1200864931 -- 1.20 billion
vendor_id:              2
pickup_date:            2015-07-01
..
dropoff_ntacode:        MN12
dropoff_ntaname:        Upper West Side
dropoff_puma:           3806
```

3.2.5 Средняя стоимость поездки

```
$ $ cc -q "SELECT round(avg(tip_amount), 2) FROM testdb.trips"
   ┌─round(avg(tip_amount), 2)─┐
```

```
    1. │                     1.68 │
       └──────────────────────────┘
```

3.2.6 Средняя цена с разбивкой по кол-ву пассажиров

```
    $ cc -q "SELECT
        passenger_count,
        ceil(avg(total_amount),2) AS average_total_amount
    FROM testdb.trips
    GROUP BY passenger_count"
    ┌─passenger_count─┬─average_total_amount─┐
 1. │               0 │                22.49 │
 2. │               1 │                15.97 │
 3. │               2 │                17.15 │
 4. │               3 │                16.76 │
 5. │               4 │                17.33 │
 6. │               5 │                16.35 │
 7. │               6 │                16.04 │
 8. │               7 │                 59.8 │
 9. │               8 │                36.41 │
10. │               9 │                 9.81 │
    └─────────────────┴──────────────────────┘
```

3.2.7 Удаление таблицы

Удаление таблицы trips из базы **testdb**.

```
    $ cc -q "drop table testdb.trips on cluster '{cluster}'"
```

3.2.8 Удаление Базы Данных

Удаление базы **testdb**.

```
    $ cc -q "drop database testdb on cluster '{cluster}'"
```

4 Мониторинг и статистика

4.1 Метрики в формате Prometheus

Экземпляры Astra ClickHouse экспортируют показатели производительности в формате Prometheus.

Для включения экспорта метрик необходимо на сервере поместить в каталог /etc/clickhouse-server/config.d/ файл с содержимым ниже.

```
<clickhouse>
    <prometheus>
        <endpoint>/metrics</endpoint>
        <port>9363</port>
        <metrics>true</metrics>
        <events>true</events>
        <asynchronous_metrics>true</asynchronous_metrics>
    </prometheus>
</clickhouse>
```

После этого метрики на соответствующем сервере, например, можно получить с помощью команды ниже:

```
$ curl -s haribda-01.internal:9363/metrics
```

Доступны дашборды для визуализации этих метрик в Grafana.

Например: *https://grafana.com/grafana/dashboards/14192-clickhouse/*

4.2 Метрики запросами

Для анализа метрик с помощью SQL система Astra ClickHouse предоставляет таблицы с большим количеством:

- текущих метрик в таблицах:
  - system.metrics
  - system.asynchronous_metrics
- исторических метрик в таблицах:
  - system.metric_log
  - system.asynchronous_metric_log

Для включения истории метрик необходимо на сервере поместить в каталог /etc/clickhouse-server/config.d/ файл с примерным содержимым ниже.

Здесь задаются имена соответствующих таблиц и правила хранения данных в них. В примере данные в таблицах хранятся 14 дней, после чего автоматически удаляются.

```
</clickhouse>
  <asynchronous_metric_log replace="1">
    <database>system</database>
    <table>asynchronous_metric_log</table>
    <engine>ENGINE = MergeTree
            PARTITION BY (event_date)
            ORDER BY (event_time)
            TTL event_date + INTERVAL 14 DAY DELETE
    </engine>

<flush_interval_milliseconds>7500</flush_interval_milliseconds>
  </asynchronous_metric_log>


  <metric_log replace="1">
    <database>system</database>
    <table>metric_log</table>
    <engine>ENGINE = MergeTree
            PARTITION BY (event_date)
            ORDER BY (event_time)
            TTL event_date + INTERVAL 14 DAY DELETE
    </engine>

<flush_interval_milliseconds>7500</flush_interval_milliseconds>
  </metric_log>
</clickhouse>
```

4.3 Уровни логирования

Для изменения уровня и других параметров журналирования необходимо на сервере поместить в каталог /etc/clickhouse-server/config.d/ файл с примерным содержимым ниже.

```
<clickhouse>
  <logger>
    <level>information</level>
    <log>/var/log/clickhouse-server/clickhouse-server.log</log>
    <errorlog>/var/log/clickhouse-server/clickhouse-
server.err.log</errorlog>
    <size>50M</size>
    <count>2</count>
  </logger>
</clickhouse>
```

4.4 Журнальные записи

Есть возможность получать и анализировать журнальные записи с помощью SQL. Для этого необходимо на сервере поместить в каталог /etc/clickhouse-server/config.d/ файл с примерным содержимым ниже.

```
<clickhouse>
  <text_log replace="1">
    <level>notice</level>
    <database>system</database>
    <table>text_log</table>

<flush_interval_milliseconds>7500</flush_interval_milliseconds>
    <max_size_rows>1048576</max_size_rows>
    <reserved_size_rows>8192</reserved_size_rows>

<buffer_size_rows_flush_threshold>524288</buffer_size_rows_flush
_threshold>
```

```
    <flush_on_crash>false</flush_on_crash>
    <!-- <partition_by>event_date</partition_by> -->
    <engine>Engine = MergeTree
            PARTITION BY event_date
            ORDER BY event_time
            TTL event_date + INTERVAL 30 day
    </engine>
  </text_log>
</clickhouse>
```

В примере конфигурируется таблица system.text_log и параметры хранения в ней данных. Данные будут храниться 30 дней, после чего будут удалены автоматически.

Пример запроса на вывод 10 последних сообщений за сегодня с уровнем Warning или Error.

```
$ cc -q "select *
from system.text_log
where event_date = today() and level in ('Warning', 'Error')
order by event_time desc
limit 10"
```

4.5 Другие _log таблицы

В дополнение к определяемым пользователем таблицам логирования в системе по умолчанию доступны некоторые другие. Их список можно получить так:

```
$ cc -q "select name
from system.tables where database = 'system' and name like
'%_log'
order by name"
   ┌─name──────────────────┐
1. │ asynchronous_metric_log │
2. │ error_log             │
3. │ metric_log            │
```

```
4. | part_log                 |
5. | processors_profile_log   |
6. | query_log                |
7. | query_metric_log         |
8. | text_log                 |
9. | trace_log                |
   └──────────────────────────┘
```

Среди них, например, таблицы с историей:

- сообщений об ошибках сервере (error_log)
- выполнения запросов (query_log)
- метрик выполненных запросов (query_metric_log)
- операций с кусками таблиц (part_log).

5 Резервное копирование и восстановление

5.1 Общая информация

Резервные копии в Astra ClickHouse могут быть:

- полными или инкрементальными
- синхронными или асинхронными
- конкурентными с другими аналогичными процессами или нет
- сжатыми или нет
- использовать именованные коллекции
- защищены паролем
- содержать системные, журнальные таблицы и таблицы управления доступом

Архивировать/восстанавливать данные можно с использованием:

- локального или S3 диска
- S3 endpoint'a
- AzureBlobStorage
- альтернативных методов типа снимков файловых систем или манипуляций с кусками таблиц, для чего со стороны Astra ClickHouse есть поддерживающие команды.

Для выполнения архивирования/восстановления на локальный диск необходимо на сервере поместить в каталог /etc/clickhouse-server/config.d/ файл с примерным содержимым ниже.

```
<clickhouse>
    <storage_configuration>
        <disks>
            <backups>
                <type>local</type>
                <path>/var/lib/clickhouse/backup/</path>
            </backups>
        </disks>
    </storage_configuration>
    <backups>
        <allowed_disk>backups</allowed_disk>
        <allowed_path>/var/lib/clickhouse/backup/</allowed_path>
    </backups>
```

```
</clickhouse>
```

Указанный в конфигурации выше каталог /var/lib/clickhouse/backup/ должен существовать на всех узлах системы и быть доступным на запись пользователю clickhouse.

Указанный диск с именем backups должен появиться в системных таблицах:

```
$ cc -q "select type, path from system.disks where name = 'backups'"

   ┌─type──┬─path──────────────────────────┐
1. │ Local │ /var/lib/clickhouse/backup/ │
   └───────┴───────────────────────────────┘
```

5.2 Создание резервной копии

Для создания резервной копии используется команда **BACKUP**.

```
$ cc -q "BACKUP TABLE testdb.trips

TO Disk('backups', ' testdb.data.zip')

SETTINGS compression_method='lzma', compression_level=3

ASYNC

"

    ┌─id─────────────────────────────────────┬─status─────────┐
1. │ 26dfd81f-47bd-4348-bfd1-1ba543a4e5ca │ CREATING_BACKUP │
   └────────────────────────────────────────┴────────────────┘
```

В примере выше производится запуск резервной копии одной таблицы **trips** в базе **testdb**. Запуск происходит асинхронно, команда завершает работу немедленно, выводится идентификатор резервной копии, с помощью которого можно получать информацию о результате выполнения задания.

```
$ cc -q " SELECT event_time, status

FROM system.backup_log

WHERE id = '26dfd81f-47bd-4348-bfd1-1ba543a4e5ca'

ORDER BY event_time DESC"

   ┌──────────event_time─┬─status─────────┐
1. │ 2026-03-13 15:03:37 │ BACKUP_CREATED  │
2. │ 2026-03-13 15:02:33 │ CREATING_BACKUP │
   └─────────────────────┴─────────────────┘
```

На узле, на котором запускалось получение резервной копии, в каталоге **/var/lib/clickhouse/backup/** (путь используется в определении диска **backups**, согласно конфигурации) появится файл **testdb.data.zip** с содержимым копии.

5.3 Восстановление из резервной копии

Для демонстрации сначала удалим **trips** из базы данных **testdb**.

```
$ cc -q "truncate table testdb.trips"
```

Идентификатор копии можно найти, например, листингом каталога, в который делалась копия.

Восстановление из копии. Указываются те же параметры в Disk, которые использовались при создании резервной копии, чтобы восстановиться из нее.

```
$ cc -q "RESTORE TABLE testdb.trips

FROM Disk('backups', ' testdb.data.zip')

ASYNC"


   ┌─id─────────────────────────────────────┬─status────┐
1. │ 6f896d4e-7cd6-45f6-affd-86973e363cd4 │ RESTORING │
   └────────────────────────────────────────┴───────────┘
```

Запуск происходит асинхронно, команда завершает работу немедленно, выводится идентификатор резервной копии, с помощью которого можно получать информацию о результате выполнения задания.

```
$ cc -q " SELECT event_time, status

FROM system.backup_log

WHERE id = '6f896d4e-7cd6-45f6-affd-86973e363cd4'

ORDER BY event_time DESC"

   ┌──────────event_time─┬─status────┐
1. │ 2026-03-13 15:55:21 │ RESTORED  │
2. │ 2026-03-13 15:55:17 │ RESTORING │
   └─────────────────────┴───────────┘
```

Проверка кол-ва записей в таблице.

Таблица реплицируемая (использует ядро ReplicatedMergeTree), поэтому, хотя и восстановление происходило на одном из узлов, все 3 копии таблицы получили одинаковое количество записей.

```
$ cc -q "select hostName() as host, *
from clusterAllReplicas(
  '{cluster}'
, view(
    select count()
    from testdb.trips
  )
)"
┌─host──────────────────┬─count()─┐
1. │ haribda-01.internal │ 1999658 │
2. │ haribda-02.internal │ 1999658 │
3. │ haribda-03.internal │ 1999658 │
└───────────────────────┴─────────┘
```

Приложение А Часто задаваемые вопросы

А.1 Как подключиться к модулю Astra ClickHouse извне?

Astra ClickHouse предоставляет 2 сетевых интерфейса (они могут быть использованы также с TLS для дополнительной безопасности):

- HTTP, который хорошо документирован и который легко использовать напрямую
- Native TCP, который имеет меньше накладных расходов

В большинстве случаев рекомендуется использовать соответствующие утилиты или библиотеки вместо использования их напрямую.

Следующие инструменты официально поддерживаются Astra ClickHouse:

- клиент командой строки (clickhouse-client)
- JDBC драйвер
- ODBC драйвер
- клиентские библиотеки C++

Также Astra ClickHouse поддерживает 2 протокола RPC:

- специально разработанный для Astra ClickHouse протокол gRPC
- Apache Arrow Flight

Astra ClickHouse сервер предоставляет встроенные визуальные интерфейсы для продвинутых пользователей:

- Play UI: откройте дополнительный к основному путь /play в браузере
- Продвинутый дашборд: откройте /dashboard в браузере
- Просмотр бинарных символов для инженеров Astra ClickHouse: откройте /binary в браузере
- ClickStack UI для осведомлённости: откройте /clickstack в браузере

Также есть множество библиотек третьих сторон для работы с Astra ClickHouse:

- *клиентские библиотеки*
- *интеграции*
- *визуальные интерфейсы*

А.2 Как изменить пароль пользователя

От пользователя с соответствующими правами:

```
$ cc -q "ALTER USER test IDENTIFIED BY 'NewP@ssw0rd'"
```

А.3 Установка модуля Astra ClickHouse с использованием инсталлятора

А.3.1 Установка модуля Astra ClickHouse с использованием инсталлятора

Для использования инсталлятора требуется предварительная подготовка виртуальных машин и серверов, включая настройку операционной системы, установку необходимых пакетов и инструментов, а также настройку сетевых подключений.

А.3.1.1 Установка Ansible:

– Убедитесь, что Ansible версии 2.9 и выше установлен.

Рекомендуется использовать Python 3.8 и выше.

```
sudo apt update
sudo apt install -y ansible python3-pip
```

А.3.1.2 Установка необходимых Python-модулей

```
pip3 install --user ansible
```

А.3.2 Инструкция по установке Astra ClickHouse с использованием плейбуков Ansible

Инструкция описывает установку Astra ClickHouse кластера из 3-х узлов с использованием Astra ClickHouse Keeper в качестве координатора репликации данных и выполнения распределённых запросов.
https://clickhouse.com/docs/guides/sre/keeper/clickhouse-keeper

А.3.2.1 Шаг 1: Настройка файла инвентаря

Создайте файл **`inv `** с именами хостов ваших узлов:

```
[clickhouse]
host-01.internal
host-02.internal
host-03.internal
```

```
[zookeepers]
host-01.internal
host-02.internal
host-03.internal
```

A.3.2.2 Шаг 2: Настройка плейбука

Основной плейбук для запуска установки находится в файле `**clickhouse.yml**`.

Соответствующий целевой платформе дистрибутив ClickHouse или ссылка на него должны располагаться в каталоге **roles/clickhouse/files/distrib/**. Установка будет произведена стандартным для этой платформы пакетным менеджером с использованием указанного каталога.

A.3.2.3 Шаг 3: Настройка переменных

Файл **`group_vars/all/ansible.yml`** содержит настройки удалённых подключений к хостам кластера. Пользователь, указанный в параметре ansible_user, должен иметь sudo права на удалённых хостах с беспарольным выполнением. Жирным шрифтом указаны поля, которые могут быть изменены.

ansible_connection: ssh

ansible_user: remote_username

ansible_become: yes

ansible_ssh_private_key_file: ~/.ssh/id_rsa

Файл `**roles/clickhouse/defaults/main.yml**` содержит основные переменные для настройки кластера:

ch_default_password: "P@ssw0rd" - Пароль административного пользователя с именем default.

ch_dataDir: "/var/lib/clickhouse" - Путь к основному каталогу данных ClickHouse.

ch_with_ssl: "yes" - Если этот параметр имеет значение "yes", то на локальном хосте должны быть подготовлены файлы ssl сертификатов и ключей в каталоге **roles/clickhouse/files/certs/** со следующими правилами именования и содержимым:

- **hostname.crt** - для каждого hostname из инвентарного файла **inv**; содержит сертификат для хоста hostname в формате pem
- **hostname.key** - для каждого hostname из инвентарного файла **inv**; содержит приватный ключ для хоста hostname в формате pem
- **truststore.pem** - содержит CA сертификаты в формате pem

Остальные параметры обычно не меняются.

zk_group_name: "zookeepers" - имя группы хостов Zookeeper / ClickHouse Keeper в инвентарном файле

ch_group_name: "clickhouse" - имя группы хостов серверов ClickHouse в инвентарном файле

ch_cfgDir: "/etc/clickhouse-server" - Путь к основному каталогу конфигурационных файлов ClickHouse

ch_logDir: "/var/log/clickhouse-server" - Путь к каталогу с журналами

ch_certDir: "{{ ch_cfgDir ~ '/certs' }}" - Путь к каталогу с сертификатами и ключами

ch_zk_identity: - Если переменная определена, то указанные имя и пароль используются для использования аутентификационной схемы digest для соединений с кластером Zookeeper

ch_with_zk_ssl: "yes" - использовать ли ssl соединения с кластером Zookeeper

ch_cert_file: "{{ ch_cfgDir ~ '/node.crt' }}" - Путь к файлу ssl сертификата хоста ClickHouse

ch_keystore_file: "{{ ch_certDir ~ '/node.key' }}" - Путь к файлу ssl ключа хоста ClickHouse

ch_truststore_file: "{{ ch_certDir ~ '/truststore.pem' }}" - Путь к файлу CA сертификатов

ch_service_name: "clickhouse-server.service" - Имя файла systemd сервиса ClickHouse

ch_use_keeper: "yes/no" - Использовать ClickHouse Keeper или Zookeeper в качестве координатора.

Если используется значение "yes", то каждый хост из группы zookeepers инвентарного файла должен входить в группу серверов ClickHouse clickhouse. ClickHouse Keeper будет запущен на хостах группы clickhouse в дополнение к серверу ClickHouse на хостах группы zookeepers.

Если используется значение "no", то на хостах группы zookeepers должен быть сконфигурирован и запущен кластер Zookeeper отдельно.

ch_default_cluster: "s1r3" - Имя кластера ClickHouse по умолчанию. Используется для удобства. Это значение получает макрос {**cluster**}, и он же используется в параметре **default_replica_path** Astra ClickHouse. Обычно устанавливается в значение имени одного из кластеров, в которых наиболее часто создаются реплицируемые таблицы для того, чтобы можно было создавать реплицируемые таблицы без указания параметров движка ReplicatedMergeTree. Например:

CREATE TABLE ... ON CLUSTER '{cluster}' ... ENGINE = ReplicatedMergeTree()

ch_clusters: - Описание кластеров ClickHouse. В одном из name должно быть значение, указанное в **ch_default_cluster**. Имена хостов каждого кластера указываются с помощью их индексов в группе серверов ClickHouse инвентарного файла.

Пример 2-х кластеров:

- s3r1: 3 раздела (shards) по 1 реплике
- s1r3: 1 раздел (shards) по 3 реплики (это же имя указано в ch_default_cluster)

```
ch_clusters:
 - name: s3r1
```

```
    shards:
      - replicas:
        - "{{ groups[ch_group_name][0] }}"
      - replicas:
        - "{{ groups[ch_group_name][1] }}"
      - replicas:
        - "{{ groups[ch_group_name][2] }}"
  - name: "{{ ch_default_cluster }}"
    shards:
      - replicas:
        - "{{ groups[ch_group_name][0] }}"
        - "{{ groups[ch_group_name][1] }}"
        - "{{ groups[ch_group_name][2] }}"
```

A.3.2.4 Шаг 4: Запуск плейбука в режиме установки

Для запуска установки выполните команду:

```
ansible-playbook clickhouse.yml -e "ch_config_only=no"
```

A.3.2.5 Дополнительные шаги

После успешного завершения плейбука убедитесь, что все узлы работают корректно, проверив статус службы Astra ClickHouse:

```
ansible -m shell -a "systemctl status clickhouse-server.service"
clickhouse
```

A.3.3 Проверка состояния установленного экземпляра ПО

После завершения установки и настройки кластера Astra ClickHouse, вы можете проверить состояние всех узлов с помощью утилиты ОС **curl** с использованием HTTP API. Для удобного форматирования результатов вызова **curl** может использоваться утилита jq.

Если кластер сконфигурирован с использованием SSL/TLS, то в параметре --cacert необходимо указывать тот же файл с CA сертификатами, который использовался для установки.

A.3.3.1 Получение состояния всех кластеров на всех хостах

С помощью утилиты **curl** можно получить информацию о текущем состоянии всех кластеров системы. Для выполнения этой команды на клиентской машине:

```
$ echo -n "select hostName() as host_, host_name, cluster, shard_num,
replica_num, is_local, errors_count from clusterAllReplicas('{cluster}',
system.clusters) order by host_, cluster, shard_num, replica_num format
PrettyCompactMonoblock" \
| curl_ch
```

A.3.3.2 Пример вывода команды получения состояния всех кластеров на всех хостах

Вывод команды выше для кластера из 3-х хостов и 2-х кластеров (поле cluster).

Показывается список кластеров на каждом хосте (поле host_) так, как этот хост видит эти кластеры.

Порядок следования полей в записи совпадает с порядком следования полей в команде.

```
┌─host_─────────────────┬─host_name─────────────┬─cluster─┬─shard_num─┬─replica_n
um─┬─is_local─┬─errors_count─┐
 1. │ haribda-01.internal │ haribda-01.internal │ s1r3    │         1 │         1
│         1 │            0 │
 2. │ haribda-01.internal │ haribda-02.internal │ s1r3    │         1 │         2
│         0 │            0 │
 3. │ haribda-01.internal │ haribda-03.internal │ s1r3    │         1 │         3
│         0 │            0 │
 4. │ haribda-01.internal │ haribda-01.internal │ s3r1    │         1 │         1
│         1 │            0 │
 5. │ haribda-01.internal │ haribda-02.internal │ s3r1    │         2 │         1
│         0 │            0 │
 6. │ haribda-01.internal │ haribda-03.internal │ s3r1    │         3 │         1
│         0 │            0 │
 7. │ haribda-02.internal │ haribda-01.internal │ s1r3    │         1 │         1
│         0 │            0 │
 8. │ haribda-02.internal │ haribda-02.internal │ s1r3    │         1 │         2
│         1 │            0 │
 9. │ haribda-02.internal │ haribda-03.internal │ s1r3    │         1 │         3
│         0 │            0 │
10. │ haribda-02.internal │ haribda-01.internal │ s3r1    │         1 │         1
│         0 │            0 │
11. │ haribda-02.internal │ haribda-02.internal │ s3r1    │         2 │         1
│         1 │            0 │
```

```
12. | haribda-02.internal | haribda-03.internal | s3r1    |        3 |          1
|        0 |            0 |
13. | haribda-03.internal | haribda-01.internal | s1r3    |        1 |          1
|        0 |            0 |
14. | haribda-03.internal | haribda-02.internal | s1r3    |        1 |          2
|        0 |            0 |
15. | haribda-03.internal | haribda-03.internal | s1r3    |        1 |          3
|        1 |            0 |
16. | haribda-03.internal | haribda-01.internal | s3r1    |        1 |          1
|        0 |            0 |
17. | haribda-03.internal | haribda-02.internal | s3r1    |        2 |          1
|        0 |            0 |
18. | haribda-03.internal | haribda-03.internal | s3r1    |        3 |          1
|        1 |            0 |

└─────────────────────┴─────────────────────┴─────────┴──────────┴──────────
──┴──────────┴──────────────┘
```

A.3.4 Операции с кластером

A.3.4.1 Реконфигурация с перезапуском

При изменениях параметров кластера, требующих перезапуск служб Astra ClickHouse.

```
ansible-playbook clickhouse.yml
```

A.3.4.2 Реконфигурация без перезапуска

При изменениях параметров кластера, не требующих перезапуск служб Astra ClickHouse.

```
ansible-playbook clickhouse.yml -e "ch_reconfig_restart=no"
```

A.3.4.3 Останов кластера

```
ansible-playbook clickhouse.yml -e "play_action=stop"
```

A.3.4.4 Старт кластера

```
ansible-playbook clickhouse.yml -e "play_action=start"
```

A.3.5 Заключение

После развертывания и настройки кластера с помощью Ansible и запуска модуля Astra ClickHouse, регулярная проверка состояния узлов с помощью утилиты curl является ключевым шагом в мониторинге здоровья и производительности кластера. Этот инструмент позволяет вам оперативно обнаружить проблемы и принять необходимые меры для их устранения.

Если запрос выше выполняется успешно, и поле errors_count во всех записях равно 0, то это означает, что ваш кластер функционирует корректно.

A.4 Инструкция по подключению к стенду

Доступ к стенду предоставляется с помощью ssh подключения:

```
ssh 4Check@80.85.253.160
```

На этом сервере:

- установлены утилиты для работы с Astra ClickHouse, такие как **clickhouse-client** и **curl**
- создан файл **~/.clickhouse-client/config.xml** для подключения к кластеру для примера

Пример получения соединения с кластером и выполнения команды:

```
$ clickhouse-client -h haribda-01.internal -q "select version()"

25.8.18.1
```

Пример использования утилиты curl с запросом на проверку жизнеспособности всех компонентов кластера:

```
$ echo 'SELECT version()' | curl 'https://haribda-01.internal:8443' \
--cacert /etc/security/certs/truststore.pem \
--user "default:P@ssw0rd" --data-binary @-

25.8.18.1
```