FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on lab#3

Performed by

Daniil Nelubin

J41322c

Accepted by

Teryoshkin Sergey

St. Petersburg

2021

1. Прописываем порты sudo nano /etc/hosts

Пароль от локального убунту

Дописывание адресов

10.32.7.101 gateway.st

10.32.7.200 docker.st

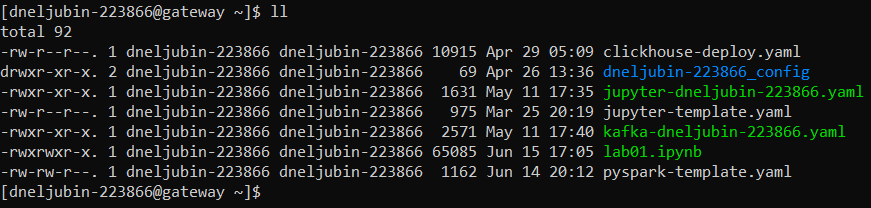
10.32.7.103 node03.st

Логинимся на сервер с включенным forticlient

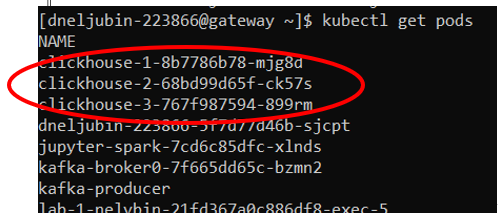
$ ssh dneljubin-223866@gateway.st

Ввод пароля

1. Убедится, что имеется файл-шаблон yaml



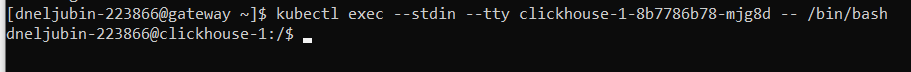
1. Создать pod выполнив kubectl apply -f clickhouse-deploy.yaml Теперь по команде kubectl get pods появится три новых poda



1. Логинимся, выполнив

kubectl exec --stdin --tty clickhouse-1-8b7786b78-mjg8d -- /bin/bash

(выбрать можно любой доступный pod)



1. Подключаемся к кликхаусу через клиент, выполнив clickhouse-client

clickhouse-client 
ClickHouse client version 20.5.2.7 (official build). 
Connecting to localhost:9øøø as user default. 
onnected to ClickHouse server version 2m 5.2 revision 54435. 
clickhouse-l : ) 

Кликхаус запущен

Посмотреть доступные базы данных через show databases

lickhouse-l show databases; 
DATABASES 
_tempora I_tables 
default 
system 
3 rows in set. 
Elapsed: 
ø.øø2 sec. 

1. Структура кластера по SELECT cluster, shard\_num, host\_name, host\_address, port, is\_local FROM system.clusters;

SELECT 
cluster, 
shard num, 
host name, 
host address, 
port, 
is local 
system. clusters 
cluste 
kube clickhouse cluster 
kube clickhouse cluster 
kube clickhouse cluster 
13 rows in set. Elapsed: 0 
hard 
.021 sec. 
st n 
clickhouse-l 
clickhouse-2 
clickhouse-3 
st address 
10.128.232.52 
10.129.gø.253 
10.129.108.73 
is local 
gøøø 
gøøø 
gøøø 

Кластер разделен на шарды, каждый со своим адресом

1. Создадим базу данных выполнив:

CREATE DATABASE IF NOT EXISTS vk ON CLUSTER kube\_clickhouse\_cluster;

Где vk название базы данных

После выполнения получаем ответ от каждого шарда

clickhouse-l : ) 
CREATE DATABASE 
hos 
clickh0use-3 
ho s 
clickhouse-2 
clickhouse-l 
CREATE 
IF NOT 
9øøø 
9000 
9øøø 
DATABASE IF NOT EXISTS vk ON CLUSTER 
EXISTS vk ON CLUSTER kube clickhouse 
kube clickhouse cluster; 
cluster 
tatus 
tat u s 
um_host s _ 
hosts 
um 
2 
rema in in 
um 
hosts 
acti 

В show databases появится наша БД

clickhouse-l show databases; 
SHOU DATABASES 
_tempora I_tables 
default 
SY stem 
vk 

Посмотрев на данные из json файлов, понимаем какой тип будет иметь каждый столбец в нашей будущей таблице в базе данных vk

1. Создаем таблицу, прописав типы переменных в столбцах

CREATE TABLE vk.likes ON CLUSTER kube\_clickhouse\_cluster (item\_type String, owner\_id Int64, item\_id Int64, liker\_id Int64, ctime DateTime, like\_date DateTime, post\_date DateTime) ENGINE = MergeTree() PARTITION BY toYear(post\_date) ORDER BY (liker\_id, owner\_id, item\_id);

CREATE TABLE vk.likes ON CLUSTER kube clickhouse cluster 
item_type• String, 
owner id- Int64, 
item id- Int64, 
liker id- Int64, 
ctime• Datetime, 
like date" DateTime, 
post_date- DateTime 
ENGINE = hergeTree() 
PARTITION BY 
ORDER BY (liker_id, owner_id, item_id) 
host 
clickhouse 
-2 
clickhouse-l 
host 
clickhouse-3 
rt 
status 
status 
0.982 sec. 
num hosts remainin 
num_hosts_remaining 
un hosts acti 
un hosts acti 
13 rows in set. Elapsed: 

**Аналогично для остальных таблиц**

CREATE TABLE vk.followers ON CLUSTER kube\_clickhouse\_cluster (profile\_id Int64, follower\_id Int64, ctime DateTime) ENGINE = MergeTree() Order BY (profile\_id, follower\_id);

CREATE TABLE vk.friends ON CLUSTER kube\_clickhouse\_cluster (user\_id Int64, friend\_id Int64, ctime DateTime) ENGINE = MergeTree() Order BY (user\_id, friend\_id);

CREATE TABLE vk.posts ON CLUSTER kube\_clickhouse\_cluster (ctime DateTime, date DateTime, post\_id Int64, from\_id Int64, owner\_id Int64, comments\_count Int64, likes\_count Int64, reposts\_count Int64, views\_count Int64, text String, signed\_by Int64, post\_type String, reposted\_from\_owner\_id Float64, reposted\_from\_post\_id Float64, geo Int64, geo\_lat Float64, geo\_lon Float64, geo\_hash Float64, photo\_attachments Nested(owner\_id Nullable(Int64), photo\_id Nullable(Int64), size Nullable(Float64), url Nullable(String), geo\_lat Nullable(Float64), geo\_lon Nullable(Float64), geo\_hash Nullable(Float64)), video\_attachments Nested (owner\_id Nullable(Int64), video\_id Nullable(Int64), views Nullable(Int64)), audio\_attachments Nested (owner\_id Nullable(Int64), audio\_id Nullable(Int64), artist Nullable(String), title Nullable(String)), doc\_attachments Nested (owner\_id Nullable(Int64), doc\_id Nullable(Int64), title Nullable(String), size Nullable(Int64), url Nullable(String), type Nullable(Int64)), link\_attachments Nested (title Nullable(String), url Nullable(String)), page\_attachments Nested (group\_id Nullable(Int64), page\_id Nullable(Int64)), sticker\_attachments Nested (sticker\_id Nullable(Int64)), photos\_attachments\_count Nullable(Int64), videos\_attachments\_count Nullable(Int64), audios\_attachments\_count Nullable(Int64), docs\_attachments\_count Nullable(Int64), links\_attachments\_count Nullable(Int64), pages\_attachments\_count Nullable(Int64), stickers\_attachments\_count Nullable(Int64)) ENGINE = MergeTree() Order BY (owner\_id, post\_id);

CREATE TABLE vk.user\_profiles ON CLUSTER kube\_clickhouse\_cluster(ctime DateTime, id Int64, first\_name String, last\_name String, screen\_name Nullable(String), maiden\_name Nullable(String), nickname Nullable(String), bdate Nullable(String), birth\_date Nullable(UInt32), sex Int16, deactivated Int8, is\_closed Nullable(Int8), verified Int8, followers\_count Nullable(Int64), status Nullable(String), city\_id Int32, city\_title Nullable(String), country\_id Int32, country\_title Nullable(String), mobile\_phone Nullable(String), home\_phone Nullable(String), tv Nullable(String), twitter Nullable(String), livejournal Nullable(String), facebook Nullable(String), site Nullable(String), skype Nullable(String), instagram Nullable(String), about Nullable(String), activities Nullable(String), books Nullable(String), home\_town Nullable(String), interests Nullable(String), movies Nullable(String), music Nullable(String), games Nullable(String), quotes Nullable(String), domain Nullable(String), personal\_alcohol Nullable(Int8), personal\_inspired\_by Nullable(String), personal\_langs Array(Nullable(String)), personal\_life\_main Nullable(Int8), personal\_people\_main Nullable(Int8), personal\_political Nullable(Int16), personal\_religion Nullable(String), personal\_smoking Nullable(Int8), relation Nullable(Int8), relation\_partner\_first\_name Nullable(String), relation\_partner\_id Nullable(Int64), relation\_partner\_last\_name Nullable(String), photo\_id Nullable(String), photo\_max\_url Nullable(String), crop\_photo\_album\_id Nullable(Int64), crop\_photo\_date Nullable(DateTime), crop\_photo\_id Nullable(Int64), crop\_photo\_lat Nullable(Float64), crop\_photo\_long Nullable(Float64), crop\_photo\_owner\_id Nullable(Int64), crop\_photo\_max Nullable(Int32), crop\_photo\_max\_url Nullable(String), crop\_photo\_post\_id Nullable(Int64), crop\_photo\_text Nullable(String), occupation\_id Nullable(Int64), occupation\_name Nullable(String), occupation\_type Nullable(String), education\_form Nullable(Int8), education\_status Nullable(String), faculty Nullable(Int32), faculty\_name Nullable(String), graduation Nullable(UInt32), university Nullable(Int32), university\_name Nullable(String), relatives Nested(user\_id Nullable(Int64), name Nullable(String), type Nullable(String)), career Nested(company Nullable(String), group\_id Nullable(Int64), city\_id Nullable(Int32), country\_id Nullable(Int32), position Nullable(String), from Nullable(UInt32), until Nullable(UInt32)), schools Nested (city\_id Nullable(Int32), class Nullable(String), country\_id Nullable(Int32), id Nullable(Int32), name Nullable(String), speciality Nullable(String), type Nullable(UInt8), type\_str Nullable(String), year\_from Nullable(UInt16), year\_graduated Nullable(UInt16), year\_to Nullable(UInt16)), universities Nested (chair Nullable(UInt32), chair\_name Nullable(String), education\_form Nullable(String), education\_status Nullable(String), faculty Nullable(UInt32), faculty\_name Nullable(String), graduation Nullable(UInt32), id Nullable(Int32), name Nullable(String))) ENGINE = MergeTree() ORDER BY (id);

1. Создаем распределенную таблицу выполнив

CREATE TABLE vk.distributed\_likes ON CLUSTER kube\_clickhouse\_cluster AS vk.likes

ENGINE = Distributed(kube\_clickhouse\_cluster, vk, likes, xxHash64(owner\_id));

1. **Аналогично для остальных таблиц**

CREATE TABLE vk.distributed\_followers ON CLUSTER kube\_clickhouse\_cluster AS vk.followers

ENGINE = Distributed(kube\_clickhouse\_cluster, vk, followers, xxHash64(profile\_id));

CREATE TABLE vk. distributed \_friends ON CLUSTER kube\_clickhouse\_cluster AS vk.friends

ENGINE = Distributed(kube\_clickhouse\_cluster, vk, friends, xxHash64(user\_id));

CREATE TABLE vk. distributed \_posts ON CLUSTER kube\_clickhouse\_cluster AS vk.posts

ENGINE = Distributed(kube\_clickhouse\_cluster, vk, posts, xxHash64(owner\_id));

CREATE TABLE vk. distributed \_user\_profiles ON CLUSTER kube\_clickhouse\_cluster AS vk.user\_profiles

ENGINE = Distributed(kube\_clickhouse\_cluster, vk, user\_profiles, xxHash64(id));

1. Создание буфера

Запускаем кафку, выполнив в терминале под {login}@gateway.st

kubectl apply -f kafka- dneljubin\_223866.yaml

To read data from a Kafka topic to a ClickHouse table, we need three things:

* A target MergeTree table to provide a home for ingested data
* A Kafka engine table to make the topic look like a ClickHouse table
* A materialized view to move data automatically from Kafka to the target table

[Source|](https://altinity.com/blog/2020/5/21/clickhouse-kafka-engine-tutorial)

Первый пункт уже есть (все нераспределенные таблицы до этого создавались через MergeTree)

Второй пункт: делаем кафка-таблицу на основе уже существующей, но можно и вручную определить типы и названия столбцов, создавая таблицу не по аналогии (с помощью AS vk.table), а с нуля.

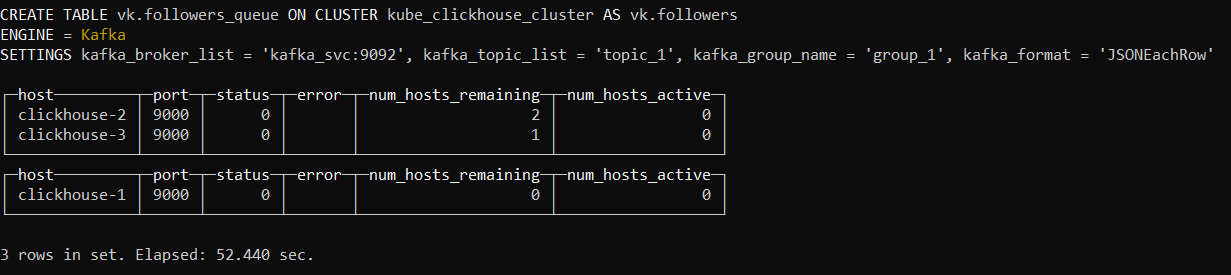
CREATE TABLE vk.followers\_kafka ON CLUSTER kube\_clickhouse\_cluster AS vk.followers

ENGINE = Kafka

SETTINGS kafka\_broker\_list = 'kafka\_svc:9092', kafka\_topic\_list = 'topic\_1', kafka\_group\_name = 'group\_1', kafka\_format = 'JSONEachRow'

Буфер для других таблиц

CREATE TABLE vk.friends\_kafka ON CLUSTER kube\_clickhouse\_cluster AS vk.friends ENGINE = Kafka SETTINGS kafka\_broker\_list = 'kafka\_svc:9092', kafka\_topic\_list = 'topic\_2', kafka\_group\_name = 'group\_1', kafka\_format = 'JSONEachRow'



CREATE TABLE vk.likes\_kafka ON CLUSTER kube\_clickhouse\_cluster AS vk.likes ENGINE = Kafka SETTINGS kafka\_broker\_list = 'kafka\_svc:9092', kafka\_topic\_list = 'topic\_3', kafka\_group\_name = 'group\_1', kafka\_format = 'JSONEachRow'

CREATE TABLE vk.posts\_kafka ON CLUSTER kube\_clickhouse\_cluster AS vk.posts ENGINE = Kafka SETTINGS kafka\_broker\_list = 'kafka\_svc:9092', kafka\_topic\_list = 'topic\_4', kafka\_group\_name = 'group\_1', kafka\_format = 'JSONEachRow'

CREATE TABLE vk.user\_kafka ON CLUSTER kube\_clickhouse\_cluster AS vk.user\_profiles ENGINE = Kafka SETTINGS kafka\_broker\_list = 'kafka\_svc:9092', kafka\_topic\_list = 'topic\_5', kafka\_group\_name = 'group\_1', kafka\_format = 'JSONEachRow'

Наконец, третий пункт: делаем M\_View, что передачи данных от кафки в кликхаус. Данные отправляем в распределенную версию таблицы для балансировки данных из буфера между шардами

CREATE MATERIALIZED VIEW vk.followers\_kafka\_mv ON CLUSTER kube\_clickhouse\_cluster TO vk.distributed\_followers AS SELECT \* FROM vk.followers\_kafka

CREATE MATERIALIZED VIEW vk.friends\_kafka\_mv ON CLUSTER kube\_clickhouse\_cluster TO vk.distributed\_friends AS SELECT \* FROM vk.friends\_kafka

CREATE MATERIALIZED VIEW vk.likes\_kafka\_mv ON CLUSTER kube\_clickhouse\_cluster TO vk.distributed\_likes AS SELECT \* FROM vk.likes\_kafka

CREATE MATERIALIZED VIEW vk.posts\_kafka\_mv ON CLUSTER kube\_clickhouse\_cluster TO vk.distributed\_posts AS SELECT \* FROM vk.posts\_kafka

CREATE MATERIALIZED VIEW vk.users\_kafka\_mv ON CLUSTER kube\_clickhouse\_cluster TO vk.distributed\_user\_profiles AS SELECT \* FROM vk.user\_kafka

Буфер создан

1. Запускаем еще один терминал под @clickhouse-1 (пункт 3) выполняем вставку данных в созданную таблицу из файла через буфер кафки, выполнив

cat /shared-data/clickhouse\_data/likes.json | clickhouse-client --query="INSERT INTO vk.likes\_kafka FORMAT JSONEachRow"

Где cat - чтение файла, | означает передачу данных в то, что справа. Запрос содержит ключевое слово INSERT INTO, затем ссылка на таблицу и указание на формат. В нашем случае JSON в каждой строке содержит отдельный объект (entity)

**Аналогично для других таблиц**

cat /shared-data/clickhouse\_data/followers.json | clickhouse-client --query="INSERT INTO vk.followers\_kafka FORMAT JSONEachRow"

cat /shared-data/clickhouse\_data/friends.json | clickhouse-client --query="INSERT INTO vk.friends\_kafka FORMAT JSONEachRow"

cat /shared-data/clickhouse\_data/posts.json | clickhouse-client --query="INSERT INTO vk.posts\_kafka FORMAT JSONEachRow"

cat /shared-data/clickhouse\_data/user\_profiles.json | clickhouse-client --query="INSERT INTO vk.user\_kafka FORMAT JSONEachRow"

Можно заливать напрямую в распределенную таблицу по команде вида

cat /shared-data/clickhouse\_data/{file}.json | clickhouse-client --query="INSERT INTO vk.{distributed\_table} FORMAT JSONEachRow"

По cat /shared-data/clickhouse\_data/likes.json | wc -l можно передать данные из файла на подсчет слов (wc), флаг -l задаст счет строк. Всего там 20.500.000 строк. Очевидно, что столько же должно было импортироваться в нашу базу данных. Выполняем в терминале под clickhouse:) команду SELECT count() FROM vk.distr\_likes и получаем те же 20млн строк на всем кластере. Для получения размера таблицы на конкретном шарде выполняется select count() from vk.likes; (скорее всего под первым будет 6827520)

**Контрольные числа для таблиц (получают из терминала с clickhouse bin/bash)**

cat /shared-data/clickhouse\_data/likes.json | wc -l число записей: 20500000

cat /shared-data/clickhouse\_data/followers.json | wc -l число записей: 50000000

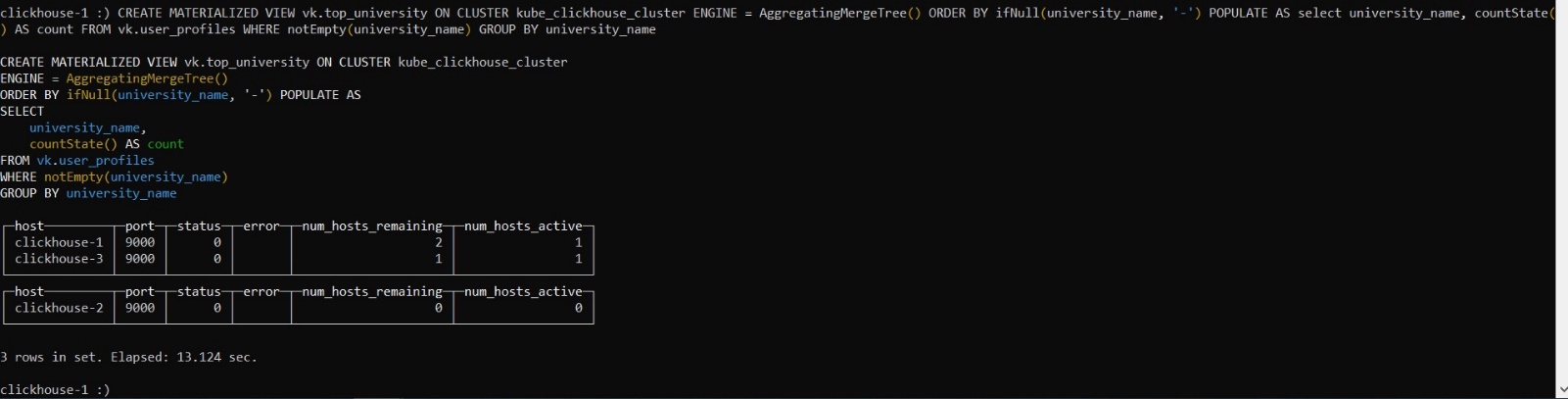
cat /shared-data/clickhouse\_data/friends.json | wc -l число записей: 50000000

cat /shared-data/clickhouse\_data/posts.json | wc -l число записей: 2000000

cat /shared-data/clickhouse\_data/user\_profiles.json | wc -l число записей: 286013

1. Создаем materialized view

CREATE MATERIALIZED VIEW vk.top\_university ON CLUSTER kube\_clickhouse\_cluster ENGINE = AggregatingMergeTree() ORDER BY ifNull(university\_name, ‘‒’) POPULATE AS SELECT university\_name, countState() AS count FROM vk.user\_profiles GROUP BY university\_name

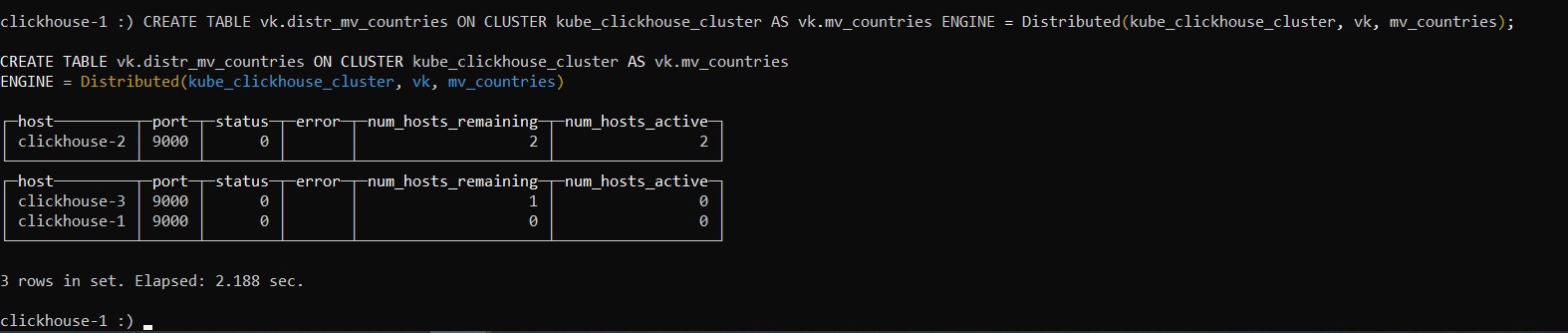


Делаем его распределённым

CREATE TABLE vk.distr\_mv\_univeristy ON CLUSTER kube\_clickhouse\_cluster AS vk.top\_university ENGINE = Distributed(kube\_clickhouse\_cluster, vk, too\_university);

Аналогично еще один

CREATE MATERIALIZED VIEW vk.mv\_countries ON CLUSTER kube\_clickhouse\_cluster ENGINE = AggregatingMergeTree() ORDER BY country\_id POPULATE AS SELECT country\_id, country\_title, countState() AS country FROM vk.user\_profiles GROUP BY country\_title, country\_id



Распределяем

CREATE TABLE vk.distr\_mv\_countries ON CLUSTER kube\_clickhouse\_cluster AS vk.mv\_countries ENGINE = Distributed(kube\_clickhouse\_cluster, vk, mv\_countries);

И еще один был

CREATE MATERIALIZED VIEW vk.mv\_comments ON CLUSTER kube\_clickhouse\_cluster ENGINE = AggregatingMergeTree() ORDER BY (post\_id, comments\_count) POPULATE AS SELECT post\_id, countState() AS comments\_count FROM vk.posts GROUP BY post\_id;

Делаем его распределенным

CREATE TABLE vk.distr\_mv\_comments ON CLUSTER kube\_clickhouse\_cluster AS vk.mv\_comments ENGINE = Distributed(kube\_clickhouse\_cluster, vk, mv\_comments);