6 
root 
adnn 
exit: 
—IS Is -tah 'nfs/shared/clickhouse data/ 
Apr 29 
Aur 29 
Apr 29 
r oot 
admn 
n 
Apr 
3.60 
Apr 
Apr 
Aor 
29 
29 
29 
29 
29 
42 
44 
Practice Oita 
likes, ison 

* + Прописываем порты Sudo nano /etc/hosts
    - Пароль от локального убунту
    - Дописывание адресов
    - 10.32.7.101 gateway.st
    - clickhouse\_data
    - 10.32.7.103 node03.st
  + Логинимся на сервер с включенным forticlient

$ ssh vvoronin-306285@gateway.st

* + Ввод пароля
  + VbM4nh84R8

* + Убедится, что имеется файл-шаблон yaml

total 24 
-rw-r--r--. 
[vvoronin-3ß6285@gateway —J$ 11 
17:35 
1 vvoronin-306285 vvoronin 
1 vvoronin 306285 vvoronin 
-306285 
le896 Apr 
29 
05 
clickhouse-deploy. yaml 

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

[vvoronin-3e6285@gateway kubectl 
pod/ c lickhouse -I- 7b6a5d5ba7- kgbht 
pod/ clickhouse -2-578df67å6d- j q rtn 
pod/ clickhouse -3-558f878bc -6hcwp 
pod/ jupyter- spa rk - 75c 86b8afb -grxm2 
pod/ la b2-97fff7777 -altgm 
pod/ zookeeper -7f68bd 57b9-pqv56 
get all 
READY 
1/1 
1/1 
1/1 
1/1 
1/1 
1/1 
STATUS 
Running 
Running 
Running 
Running 
Running 
Running 
RESTARTS 
AGE 
5d23h 

* 1. Логинимся, выполнив

**kubectl exec --stdin --tty clickhouse-1-7b64545b47-kgbht -- /bin/bash**

(выбрать можно любой доступный pod)

Cvvoronin-306285@eateway 
kubectl exec - -stdin 
hostname 
clickhouse-l 
--tty clickhouse-1-7b64545b47-kgbht /bin/bash 

* 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 

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

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(String)), 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.distr\_likes ON CLUSTER kube\_clickhouse\_cluster AS vk.likes

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

Без хеширования указанного в конце, раскидать таблицу по трем шардам и затем найти записи в нужном порядке будет невозможно.

clickhouse-l 
:) CREATE TABLE vk.distr_likes ON CLUSTER kube_clickhouse_cluster AS vk.likes 
- Distributed(kube_clickhouse_cluster, vk, likes, xxHash64(owner_id)); 
ENGINE 
CREATE TABLE 
vk.distr likes ON CLUSTER kube clickhouse cluster AS vk.likes 
ENGINE = Distributed(kube_clickhouse_cluster, vk, likes, xxHash64(owner_id)) 
host 
clickhouse 
-2 
clickhouse-l 
host 
clickhouse-3 
13 rows in set. 
port 
9000 
9øøø 
port 
9øøø 
Elapsed: 
tat u 
tatus 
o. 727 sec. 
um hosts remainin 
um hosts remainin 
um hosts activ 
um hosts activ 

**Аналогично для остальных таблиц**

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

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

REATE 
ENGINE - 
hos 
clickhouse-2 
clickhouse-3 
clickhouse-l 
3 rows in set. 
TABLE vk.distr followers ON CLUSTER kube clickhouse cluster AS vk.followers 
- Distributed(kube_clickhouse_cluster, vk, followers, 
9øøø 
9øøø 
9øøø 
tatus 
erro 
0 
0 
hosts remainin 
num hosts acti 
e 
Elapsed: 0.126 sec. 

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

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

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

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

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

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

* 1. В соседнем терминале под @clickhouse-1 выполняем вставку данных в созданную таблицу и файла, выполнив

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

Выполняется до 7 минут.

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.distr\_followers FORMAT JSONEachRow" время выполнения до 4 минут

cat /shared-data/clickhouse\_data/friends.json | clickhouse-client --query="INSERT INTO vk.distr\_friends FORMAT JSONEachRow" время выполнения до 2 минут

cat /shared-data/clickhouse\_data/posts.json | clickhouse-client --query="INSERT INTO vk.distr\_posts FORMAT JSONEachRow" время выполнения до 2 минут

cat /shared-data/clickhouse\_data/user\_profiles.json | clickhouse-client --query="INSERT INTO vk.distr\_user\_profiles FORMAT JSONEachRow" время выполнения до 20 секунд

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

Проверять через терминал с clickhouse-client запрашивая из **всего** кластера, а не одного

clickhouse-l :) select count() from vk.distr_followers; 
count() 
SELECT 
vk.distr followers 
ount() 

Получить размер таблиц в Мб:

SELECT table, formatReadableSize(sum(data\_comperssed\_bytes)) FROM system.parts WHERE database='vk' AND active=1 GROUP BY database, table;

Если что-то пошло не так, сносим таблицы, создаем их и заново импортируем данные

DROP TABLES vk.posts FROM vk ON CLUSTER kube\_clickhouse\_cluster

Следом

DROP TABLES vk.distr\_posts FROM vk ON CLUSTER kube\_clickhouse\_cluster

Контроль через show tables from vk;

* 1. Создаем materialized view

Топ-10 постов по числу комментов

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\_likes);

Смотрим результат

SELECT post\_id, countMerge(comments\_count) as comments\_count FROM vk.mv\_comments GROUP BY post\_id LIMIT 10;

lickhouse-l : ) SELECT post_id, 
p BY post_id LIMIT 10; 
ELECT 
post _ id , 
AS comments_count 
FRCY•I vk.mv comments 
ROUP BY post_id 
as FRCY•I vk .mv_conrnents GR 
LIMIT le 
pos 
46Ø85 
13289 
230654 
sego 
379Ø2 
38598 
431917 
4394 
co—ents coun 
45 
2 
6 
1 
1 
35 
1 
1 
27 

Аналогично еще делаем еще два

Самая популярное имя

CREATE MATERIALIZED VIEW vk.mv\_first\_name ON CLUSTER kube\_clickhouse\_cluster ENGINE = AggregatingMergeTree() ORDER BY (first\_name) POPULATE AS SELECT first\_name, countState() AS frequency FROM vk.user\_profiles GROUP BY first\_name;

Распределяем

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

Смотрим результат

SELECT first\_name, countMerge(frequency) AS frequency FROM vk.distr\_mv\_first\_name GROUP by first\_name ORDER BY frequency DESC LIMIT 10;

lickhouse-l : ) SELECT first_name, countMerge(frequency) AS frequency FRCY•I G 
P by first_name ORDER BY frequency DESC LIMIT la; 
ELECT 
first name, 
countmerge(frequency) AS frequency 
FRCY•I vk.distr mv first name 
ROUP BY first name 
RDER BY frequency DESC 
LIMIT le 
irst na 
Elena 
Alexander 
Olea 
Tatyana 
Irina 
Anna 
Anastasia 
Natalya 
Yulia 
Sergey 
requen 
6162 
5181 
5237 
a736 
4720 
4574 
4392 
4368 
azøø 

Считаем алкоголиков в новом MV

CREATE MATERIALIZED VIEW vk.mv\_alcohol ON CLUSTER kube\_clickhouse\_cluster ENGINE = AggregatingMergeTree() ORDER BY alcohol\_interest POPULATE AS SELECT personal\_alcohol, countState() AS alcohol\_interest FROM vk.user\_profiles GROUP BY personal\_alcohol;

Распределяем

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

Смотрим результат

SELECT personal\_alcohol, countMerge(alcohol\_interest) AS frequency FROM vk.distr\_mv\_alcohol GROUP by personal\_alcohol ORDER BY frequency DESC;

