

Отчет по нагрузочному тестированию с реализацией

Дата проведения: 07.06.2023

Подготовительный этап №1: настройка БД с асинхронной репликацией.

Редактируем `docker-compose.yml` (см. код репозитория):

1. Создаем сеть с заданной маской подсети - добавляем секцию `networks` (см. код приложения).
2. Редактируем основной инстанс - добавляем `networks` (чтобы инстансы были в одной сети), добавляем волюм со скриптом создания пользователя для репликации:

```
volumes:
  - ./create_replica_user.sql:/docker-entrypoint-initdb.d/create_replica_user.sql
  - $PWD/volumes/pgmaster:/var/lib/postgresql/data
```

где скрипт `create_replica_user.sql` содержит следующий запрос:

```
create role replicator with login replication password 'pass';
```

Так при создании контейнера будет автоматически создан пользователь для репликации.

3. Меняем `postgresql.conf` на мастере - меняем соответствующие строки на нужные значения:

postgres.conf	104	105	ssl = off
postgres.auto.conf	204		
postgres.conf	205	wal_level = replica	# minimal, replica, or logical
postmaster.opts	206		# (change requires restart)
postgres.conf	307		
postgres.conf	308	max_wal_senders = 4	# max number of <u>walsender</u> processes
postmaster.opts	309		# (change requires restart)

4. Добавляем запись в pgmaster/pg hba.conf с subnet с первого шага:

pg_hba.conf	99	host	replication	replicator	192.168.0.0/24	md5
pg_ident.conf	100	host	all	all	all	scram-sha-256

- ## 5. Перезапустим мастер:

docker restart pgmaster

- ## 6. Сделаем бэкап для реплик:

```
docker exec -it pgmaster bash
```

```
mkdir /pgslave
```

```
pg_basebackup -h pgmaster -D /pgslave -U replicator -v -P --wal-method=stream
exit
```

- ## 7. Копируем директорию себе:

```
docker cp pgmaster:/pgslave volumes/pgslave/
```

8. Создадим файл, чтобы реплика узнала, что она реплика:

touch volumes/pqslave/standby.signal

9. Меняем postgresql.conf на реплике pgslave:

```
332 primary_conninfo = 'host=pgmaster port=5432 user=replicator password=pass application_name=pgslave'
```

10. Добавляем секцию `pg-slave` в файл `docker-compose.yaml`, указав ссылку на скопированный волком с нужным дампом и конфиг файлами. Важно - профиль указан **donotstart** чтобы не запускать автоматически контейнер при первичном запуске.

- ### 11. Запускаем реплику из docker-compose.yml.

12. Убеждаемся, что репликация настроена в асинхронном режиме:

```
test_user=# select application_name, sync_state from pg_stat_replication;
application_name | sync_state
-----+-----
pgslave          | async
(1 row)

test_user=#
```

Подготовительный этап №2: Доработка методов поиска для работы со slave-инстансом (См. последний коммит в репозитории):

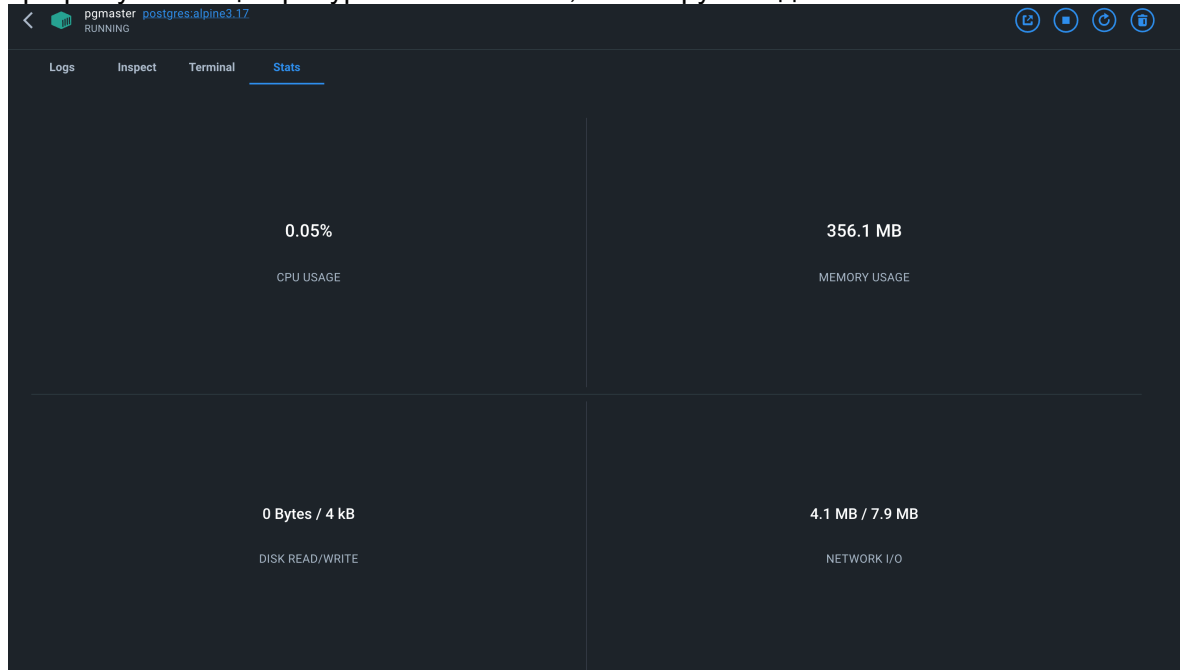
1. Настроим два новых метода по аналогии с `/user/get/{id}` и `/user/search: /user/slave/get/{id}` и `/user/slave/search`, сохраним их в коллекцию Postman.
2. Доработаем конфигурацию приложения - настроим два датасурса.

Нагрузочное тестирование, сценарий:

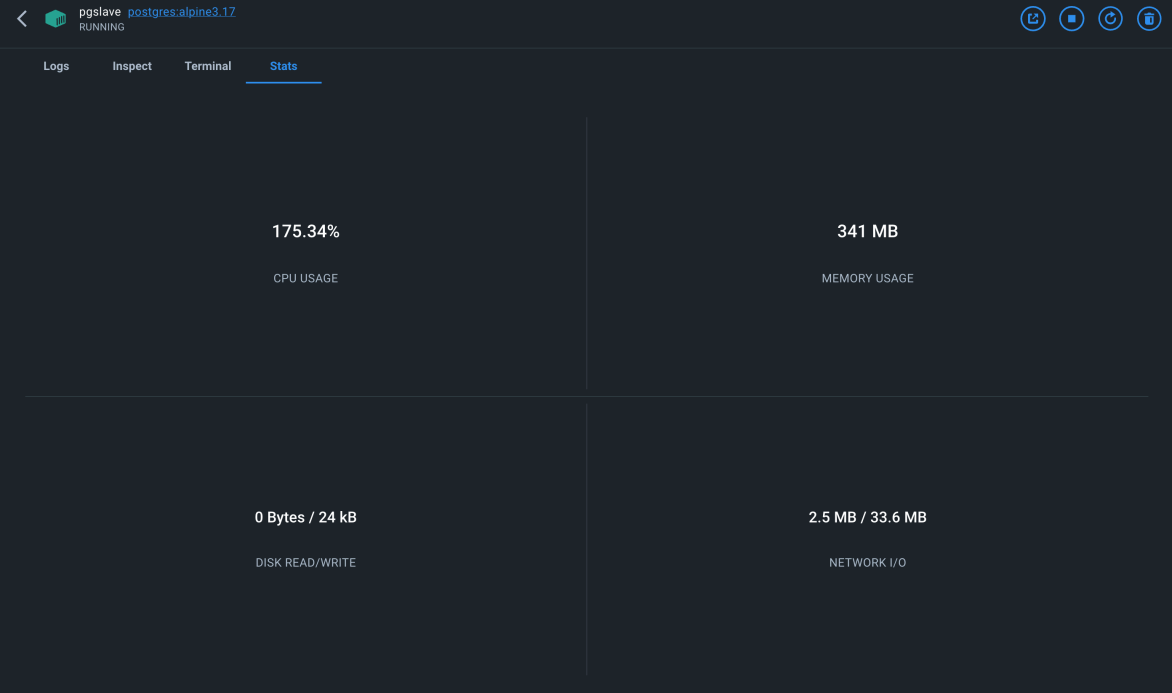
1. Создание нового пользователя: в один поток отправляется запрос на сервер для регистрации нового пользователя.
2. Авторизация, получение токена: обращение к серверу по полученному в п. 1 идентификатору.
3. Поиск случайного пользователя из заданного набора данных в течении 10 минут: 20 пользователей параллельно (по 10 на каждый запрос). Старт с 0 до 10 в течении минуты. Верификация, что каждый ответ получил 200 Ок.
 - Поиск по имени и фамилии.
 - Поиск по идентификатору.
 - Все ответы должны быть ошибочными (код 404) до проведения вставки данных в мастер и репликации.
4. На мастере проводим вставку данных.
5. После репликации ошибки уходят, все ответы приходят с кодом 200 Ок.

Нагрузочное тестирование, результат:

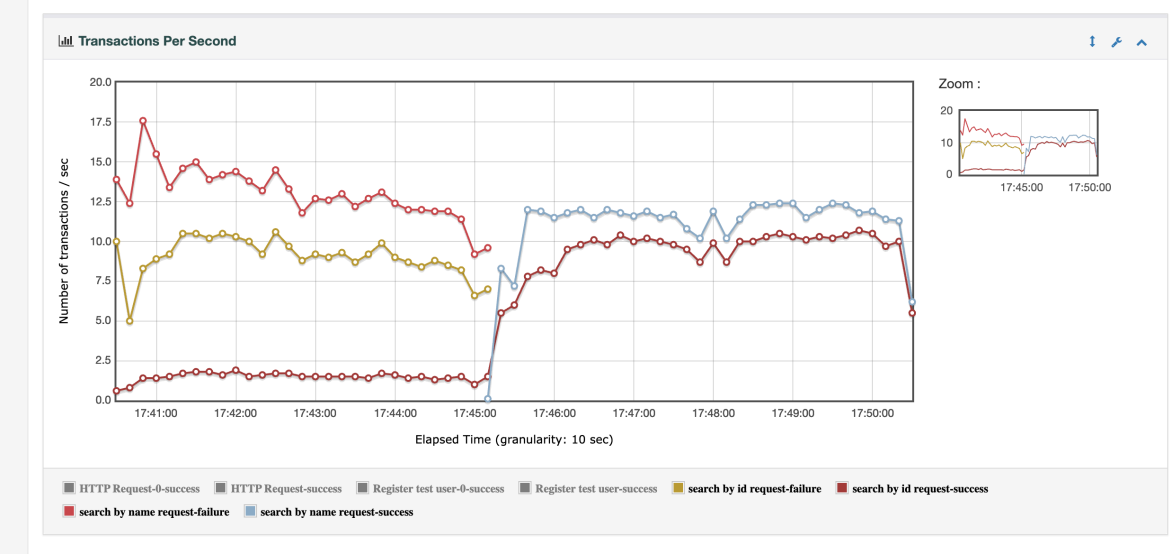
График утилизации ресурсов показывает, что нагрузка идет на слейв:



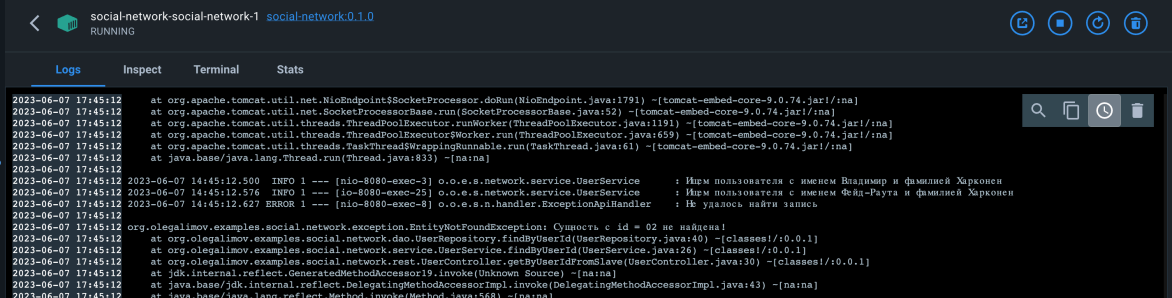
Так как утилизация ресурсов под нагрузкой на слей - сильно выше:



Из графика ТПС видно, что до момента вставки все запросы были с ошибками.



В логах приложения можно видеть также ошибки:



Через 5 минут после начала теста выполнен скрипт инсерта значений в мастер:

```
INSERT INTO users (user_id, password, second_name,
first_name, age, city)
VALUES ('01','pass01','Атрейдеc', 'Пауль', 15, 'Каладан'),
('02','pass02','Атрейдеc', 'Джессика', 42, 'Каладан'),
('03','pass03','Атрейдеc', 'Лето', 42, 'Каладан'),
('04','pass04','Харконен', 'Владимир', 42, 'Джеди Прайм'),
('05','pass05','Харконен', 'Раббан', 32, 'Джеди Прайм'),
('06','pass06','Харконен', 'Фейд-Раута', 15, 'Джеди Прайм');
```

В результате - из логов пропали ошибки, по графику ТПС выше видно падение числа ошибок:

The screenshot shows a terminal window with the title 'social-network-social-network-1' and 'social-network-0.1.0' in the subtitle. The 'Logs' tab is selected. The log entries show a sequence of INFO messages from 'nio-8080-exec-2' to 'nio-8080-exec-24', all from 'o.o.e.s.network.service.UserService'. The messages indicate successful user lookups for various user IDs (01, 05, 06) and confirm the existence of specific entities (id=1000026, id=1000022).

Проверим, что искомые значения действительно присутствуют в слейв-инстансе:

The screenshot shows a terminal window with the title 'pgslave postgres:alpine3.17'. The 'Terminal' tab is selected. A SQL query is executed: 'test_user# select * from users where user_id in ('01','02','03','04','05','06');'. The result is a table with 6 rows and 7 columns: id, user_id, password, first_name, second_name, age, sex, interests, city. The data matches the values inserted in the first screenshot.

id	user_id	password	first_name	second_name	age	sex	interests	city
1000025	04	pass04	Владимир	Харконен	42			Джеди Прайм
1000026	05	pass05	Раббан	Харконен	32			Джеди Прайм
1000027	06	pass06	Фейд-Раута	Харконен	15			Джеди Прайм
1000022	01	pass01	Пауль	Атрейдеc	15			Каладан
1000023	02	pass02	Джессика	Атрейдеc	42			Каладан
1000024	03	pass03	Лето	Атрейдеc	42			Каладан

Вывод: в ходе теста репликация сработала успешно.

Подготовительный этап №3:

1. Добавим еще одну реплику аналогично инструкции выше.
2. Включаем синхронный режим репликации:

The screenshot shows a configuration file with two lines: 'synchronous_standby_names = 'FIRST 1 (pgslave, pgasynslave)'' and a comment '# standby servers that provide sync rep'. Below it, another line shows '# method to choose sync standbys, number of sync standbys,'.

Нагрузочное тестирование, сценарий:

1. Создание нового пользователя: в один поток отправляется запрос на сервер для регистрации нового пользователя.
2. Авторизация, получение токена: обращение к серверу по полученному в п. 1 идентификатору.
3. Создание нового пользователя на основе случайного набора данных - в течении 5 минут 10 потоков параллельно отсылают POST-запрос на сервер. Старт с 0 до 10 в течении минуты. Верификация, что каждый ответ получил 200 Ок.
4. На третьей минуте теста останавливаем реплику pgslave.
5. Фиксируем значения по данным теста и в двух оставшихся инстансах.
6. Останавливаем мастер, промоутируем pgasynslave, целим на него pgslave и проверяем, есть ли потеря транзакции.


Нагрузочное тестирование, результат:

В каталоге **replication-test/sync/config** создан нагрузочный тест для Jmeter, запустим его и зафиксируем результат - выполнено 2596 успешных запросов.

Requests	Executions			
Label ▼	#Samples ◆	FAIL ◆	Error % ◆	
Total	2600	0	0.00%	.
Register user request	2596	0	0.00%	.

Аналогичное количество записей в БД на мастере:

```

<  pgmaster postgres:alpine3.17
RUNNING

Logs Inspect Terminal Stats


test_user=# select count(*) from users where first_name like '%John%';
count
-----
 2596
(1 row)

test_user=#

```

И на работающем slave:

```

<  pgasyncslave postgres:alpine3.17
RUNNING

Logs Inspect Terminal Stats

test_user=# select count(*) from users where first_name like '%John%';
count
-----
 2596
(1 row)

test_user=# █

```

На момент старта этих записей в БД не существовало.

Остановим мастер, промоутируем командой **select pg_promote();** реплику pgasyncslave до мастера и настроим репликацию:

```

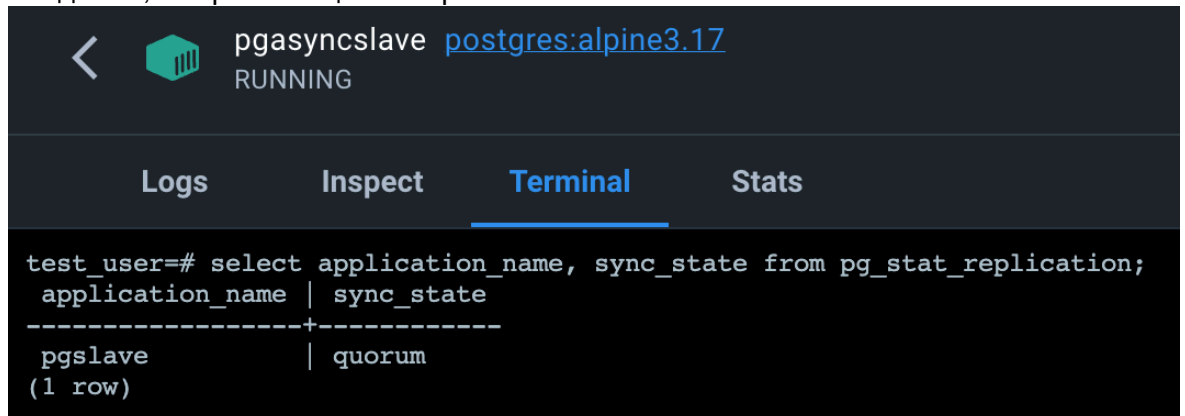
postgres@auto.com 321
postgres@auto.com 322 postgresql.conf synchronous_standby_names = 'ANY 1 (pgmaster, pgslave)' # standby servers that provide sync rep
postgres@auto.com

```

Подключаем реплику pgslave - настроим на новый мастер, включим обратно инстанс и перечитаем конфиг:

```
postgresql.auto.conf 332 primary_conninfo = 'host=pgasyncslave port=5432 user=replicator password=pass application_name=pgslave'
postgresql.conf      333 # connection string to sending server
```

Убедимся, что репликация настроена:



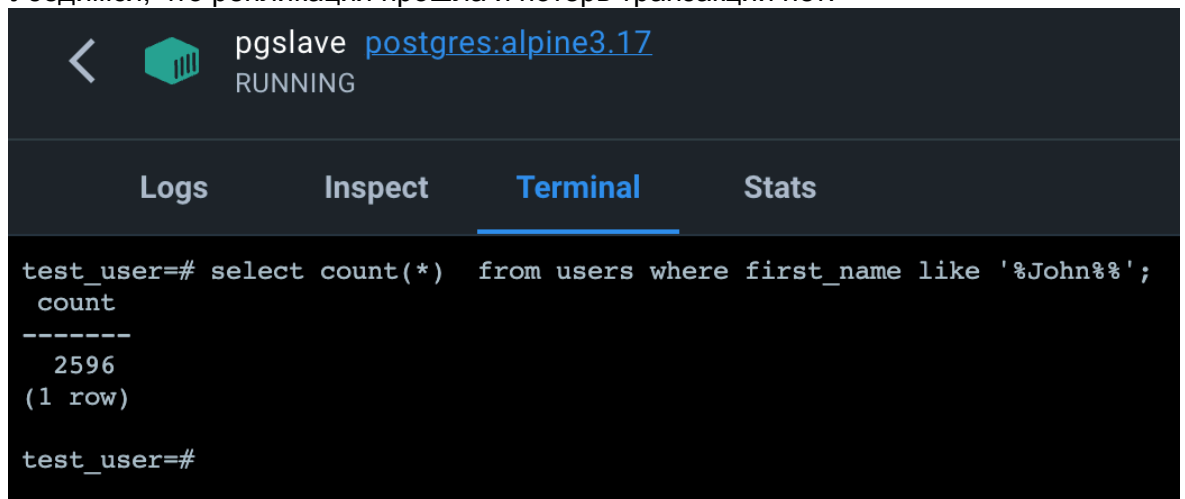
The screenshot shows a terminal window for the 'pgasyncslave' instance, which is in a 'RUNNING' state. The terminal has tabs for 'Logs', 'Inspect', 'Terminal' (which is selected), and 'Stats'. The command executed is 'select application_name, sync_state from pg_stat_replication;'. The output shows a single row with 'pgslave' as the application_name and 'quorum' as the sync_state.

```
< [icon] pgasyncslave postgres:alpine3.17
RUNNING

Logs    Inspect  Terminal  Stats

test_user=# select application_name, sync_state from pg_stat_replication;
 application_name | sync_state 
-----+-----
 pgslave          | quorum
(1 row)
```

Убедимся, что репликация прошла и потерь транзакций нет:



The screenshot shows a terminal window for the 'pgslave' instance, which is in a 'RUNNING' state. The terminal has tabs for 'Logs', 'Inspect', 'Terminal' (which is selected), and 'Stats'. The command executed is 'select count(*) from users where first_name like '%John%';'. The output shows a single row with the count '2596'.

```
< [icon] pgslave postgres:alpine3.17
RUNNING

Logs    Inspect  Terminal  Stats

test_user=# select count(*) from users where first_name like '%John%';
 count
-----
 2596
(1 row)

test_user=#
```

Вывод: нагрузочное тестирование показало, что при отказе реплики, отказе мастера с последующим промоутом актуальной реплики до мастера, после репликации потери транзакции отсутствуют.