



Highload Architect



Проверить, идет ли запись

Меня хорошо видно && слышно?



Ставим “+”, если все хорошо
“-”, если есть проблемы

Репликация: практическое применение

М1: Введение в высокие нагрузки

Проблемы
высоких нагрузок

Нагрузочное
тестирование

Введение в
высокие нагрузки

М2: СУБД в высоконагруж. проектах

Индексы 1/2

Индексы 2/2

Репликация 1/3

Репликация 2/3

Репликация 3/3

Кеширование

Транзакции в рел.
СУБД

Шардирование
1/2

Шардирование
2/2

Очереди и отл.
выполнение 1/2

Очереди и отл.
выполнение 1/2

In-Memory СУБД

OLAP и OLTP

Обзор ClickHouse

М3: Разработка бэкенда высоконагр. сервисов

Декомпозиция на
микросервисы

Микросервисы и
монолиты

Протокол HTTP
1/2

Протокол HTTP
2/2

Организация
микросервисов

Балансировка и
отказоуст. 1/2

Балансировка и
отказоуст. 2/2

Балансировка и
отказоуст. 2/2

Асинхронность
обработки

Распределенные
транзакции

Инфраструктура
микросервисов

Системы
конфигурации

Docker. docker-
compose

Мониторинг и
алертинг

М4: Типовые архитектуры

System Design

Новостной портал

Рекламная
система

Почтовый сервис

Облачное
хранилище

Сайт знакомств

М5: Итоговый проект

Выбор темы
проекта

Консультация по
проектам и дз

Защита
проектных работ

Тема вебинара

Репликация: практическое применение



Преподаватель
Игорь Золотарев

github.com/yngvar-antonsson

Внешний консультант в Tarantool (VK)

Разрабатываю инструменты для управления кластерами на Tarantool, веду тренинги и пишу статьи.



Правила вебинара



**Активно
участвуем**



**Off-topic обсуждаем
в Telegram**



**Задаем вопрос
в чат**



**Вопросы в чате вижу, но
могу ответить не сразу**

Условные обозначения



Индивидуально



**Время, необходимое
на активность**



Пишем в чат



Говорим голосом



Документ



**Ответьте себе или
задайте вопрос**

Маршрут вебинара



Цели вебинара

После занятия вы:

1. Проанализировать отличия high availability и disaster recovery
 2. Настроить репликацию в Postgres
-

Смысл

Зачем вам это уметь

1. Готовимся к авариям
2. Понимаем, что будет в случае аварии

Домашнее задание

- Поднять Postgres - docker / k8s / local
- Или воспользоваться кластерным расширением Postgres (Patroni, Citus, BDR...)
- Настроить асинхронную и синхронную репликацию
- Провести тесты

Были ли у вас аварии и какими силами вы их решали?



Пишем в чат



Отвечаем устно

High Availability & Disaster Recovery

High Availability

- измеряется в 9-ках
- все хотят 99,999 - 5 минут простоя в год
- метод измерения и критерий доступности - важен!
- работает, но тормозит

Performance

- время отклика на запрос
- как измерить?
- эмуляция запроса?
- мониторинг?

Что влияет на доступность?

- отказы или аварии
- запланированные и незапланированные
- по вине человека (чаще всего), техники (редко) или природы (совсем редко)
- метод повышения доступности - защита от отказов

Reliability

- метод повышения доступности
- устойчивость к отказам
- не путать с Disaster Recovery
- обеспечивается техническими и организационными средствами
- техническое средство - избыточность
- отказоустойчивая архитектура

Disaster recovery

- **Время восстановления?**
- **Количество потерянных данных?**
- **Дежурная смена**
- **Алгоритмы и документы**

Избыточность

- сервисов (compute)
 - active-passive кластеры
 - параллельные кластеры
- данных (data)
 - реплики

Что поможет при аварии?



Пишем в чат



Отвечаем устно

Технологии

Еще раз про репликацию

- мастер-реплика
- мастер-мастер
- асинхронная
- синхронная

Синхронная

- подтверждается после фиксации на реплике
- более надежная, чем асинхронная (durable)
- медленная
- выше шанс отказа в обслуживании
- RPO = 0

Асинхронная

- подтверждается после записи на одну ноду
- распространяет изменения асинхронно
- $RPO > 0$

Мастер-реплика

- Можно писать только в один узел
- Как правило не решает вопрос переключения нагрузки с master на slave
- Почти никогда не решат вопрос обратного переключения на master
- $RTO > 0$

Мастер-мастер

- Можно писать в несколько узлов сразу
- Или только один, но все доступны для записи
- Нужно обрабатывать конфликты
- Подходит не для всех
- $RTO = 0$
- Не поддерживается в Postgres, нужна кластерная обертка

Были у вас самописные/дописанные инструменты для БД?



Пишем в чат



Отвечаем устно

Практика

Практика

1. Создаем сеть, запоминаем адрес

```
docker network create pgnet  
docker network inspect pgnet | grep Subnet
```

2. Поднимаем мастера

```
docker run -dit -v $PWD/pgmaster:/var/lib/postgresql/data -e  
POSTGRES_PASSWORD=pass -p 5432:5432 --restart=unless-stopped --network=pgnet  
--name=pgmaster postgres
```

3. Меняем postgresql.conf на мастере

```
ssl = off  
wal_level = replica  
max_wal_senders = 4 # expected slave num
```

4. Подключаемся к мастеру и создаем пользователя для репликации

```
docker exec -it pgmaster su - postgres -c psql  
  
create role replicator with login replication password 'pass';
```

Практика

6. Добавляем запись в `pg_hba.conf` с `ip` с первого шага

```
host    replication  replicator  172.16.0.0/16  md5
```

7. Перезапустим мастера

```
docker restart pgmaster
```

8. Сделаем бэкап для реплик

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

```
mkdir /pgslave
```

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

9. Копируем директорию себе

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

10. Создадим файл, чтобы реплика узнала, что она реплика

```
touch pgslave/standby.signal
```

Практика

11. Меняем postgresql.conf на реплике

```
hot_standby = on #is pgslave available to perform queries
primary_conninfo = 'host=pgmaster port=5432 user=replicator password=pass
application_name=pgslave'
```

12. Запускаем реплику

```
docker run -dit -v $PWD/pgslave:/var/lib/postgresql/data -e
POSTGRES_PASSWORD=pass -p 15432:5432 --network=pgnet --restart=unless-stopped
--name=pgslave postgres
```

13. Запустим вторую реплику

```
docker cp pgmaster:/pgslave pgasyncslave
```

```
hot_standby = on #is pgslave available to perform queries
primary_conninfo = 'host=pgmaster port=5432 user=replicator password=pass
application_name=pgasyncslave'
```

```
touch pgasyncslave/standby.signal
```

```
docker run -dit -v $PWD/pgasyncslave:/var/lib/postgresql/data -e
POSTGRES_PASSWORD=pass -p 25432:5432 --network=pgnet --restart=unless-stopped
--name=pgasyncslave postgres
```

Практика

14. Включаем синхронную репликацию

```
synchronous_commit = on  
synchronous_standby_names = 'FIRST 1 (pgslave, pgasyncslave)'  
  
select pg_reload_conf();
```

15. Создадим тестовую таблицу и проверим репликацию

```
docker exec -it pgmaster su - postgres -c psql  
  
select application_name, sync_state from pg_stat_replication;  
  
create table test(id bigint primary key not null);  
insert into test(id) values(1);
```

16. Запромоутим реплику

```
docker stop pgmaster  
  
select * from pg_promote();  
  
synchronous_commit = on  
synchronous_standby_names = 'ANY 1 (pgmaster, pgasyncslave)'
```


Практика

17. Подключим вторую реплику к новому мастеру

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

18. Восстановим мастер в качестве реплики

```
touch pgmaster/standby.signal
```

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

19. Настроим логическую репликацию с текущего мастера на новый сервер

```
wal_level = logical
```

```
restart
```

Практика

20. Создадим публикацию

```
GRANT CONNECT ON DATABASE postgres TO replicator;  
GRANT SELECT ON ALL TABLES IN SCHEMA public TO replicator;  
create publication pg_pub for table test;
```

21. Создадим новый сервер для логической репликации

```
docker run -dit -v $PWD/pgstandalone:/var/lib/postgresql/data -e  
POSTGRES_PASSWORD=pass -p 5432:5432 --restart=unless-stopped --network=pgnet  
--name=pgstandalone postgres
```

22. Копируем файлы

```
pg_dumpall -U postgres -r -h pgslave -f /var/lib/postgresql/roles.dmp  
pg_dump -U postgres -Fc -h pgslave -f /var/lib/postgresql/schema.dmp -s  
postgres
```

```
docker cp pgslave:/var/lib/postgresql/roles.dmp .  
docker cp roles.dmp pgstandalone:/var/lib/postgresql/roles.dmp  
docker cp pgslave:/var/lib/postgresql/schema.dmp .  
docker cp schema.dmp pgstandalone:/var/lib/postgresql/schema.dmp
```

```
psql -f roles.dmp  
pg_restore -d postgres -C schema.dmp
```

Практика

23. Создаем подписку

```
CREATE SUBSCRIPTION pg_sub CONNECTION 'host=pgslave port=5432 user=replicator
password=pass dbname=postgres' PUBLICATION pg_pub;
```

24. Сделаем конфликт в данных

На sub:

```
insert into test values(9);
```

На pub:

```
insert into test values(9);
```

В логах видим:

```
2023-03-27 16:15:02.753 UTC [258] ERROR:  duplicate key value violates unique
constraint "test_pkey"
```

```
2023-03-27 16:15:02.753 UTC [258] DETAIL:  Key (id)=(9) already exists.
```

```
2023-03-27 16:15:07.802 UTC [259] CONTEXT:  processing remote data for
replication origin "pg_16397" during message type "INSERT" for replication
target relation "public.test" in transaction 743, finished at 0/30288F8
```

25. Исправляем конфликт

```
SELECT pg_replication_origin_advance('pg_16397', '0/30288F9'::pg_lsn); <-
message from log + 1
ALTER SUBSCRIPTION pg_sub ENABLE;
```

LIVE

Ваши вопросы?



Список материалов для изучения

- <https://www.postgresql.fastware.com/blog/how-to-handle-logical-replication-conflicts-in-postgresql>
- <https://severalnines.com/blog/converting-asynchronous-synchronous-replication-postgresql/>
- <https://arctype.com/blog/postgres-patroni/>
- <https://habr.com/ru/post/227959/>
- https://docs.citusdata.com/en/v5.2/installation/production_deb.html
- <https://youtu.be/rIKfnT7jN2g>

Рефлексия

Цели вебинара

Проверка достижения целей

1. Проанализировать отличия high availability и disaster recovery
 2. Настроить репликацию в Postgres
-

Рефлексия



Что нового вы узнали сегодня?



**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Следующий вебинар



Модуль 2: СУБД в высоконагруженных проектах Кеширование



Ссылка на
вебинар будет
в ЛК за 15 минут



Материалы
к занятию в ЛК —
можно изучать



Обязательный
материал обозначен
красной лентой



Спасибо за внимание!

Приходите на следующие вебинары



Игорь Золотарев

Внешний консультант в Tarantool (VK)

<https://github.com/yngvar-antonsson>

