Отчет по домашнему заданию №10 курса Highload Architect

[Реализованная схема 2](#__RefHeading___1)

[Запуск: 2](#__RefHeading___2)

[Конфиг nginx 3](#__RefHeading___7)

[Конфиг haproxy 4](#__RefHeading___8)

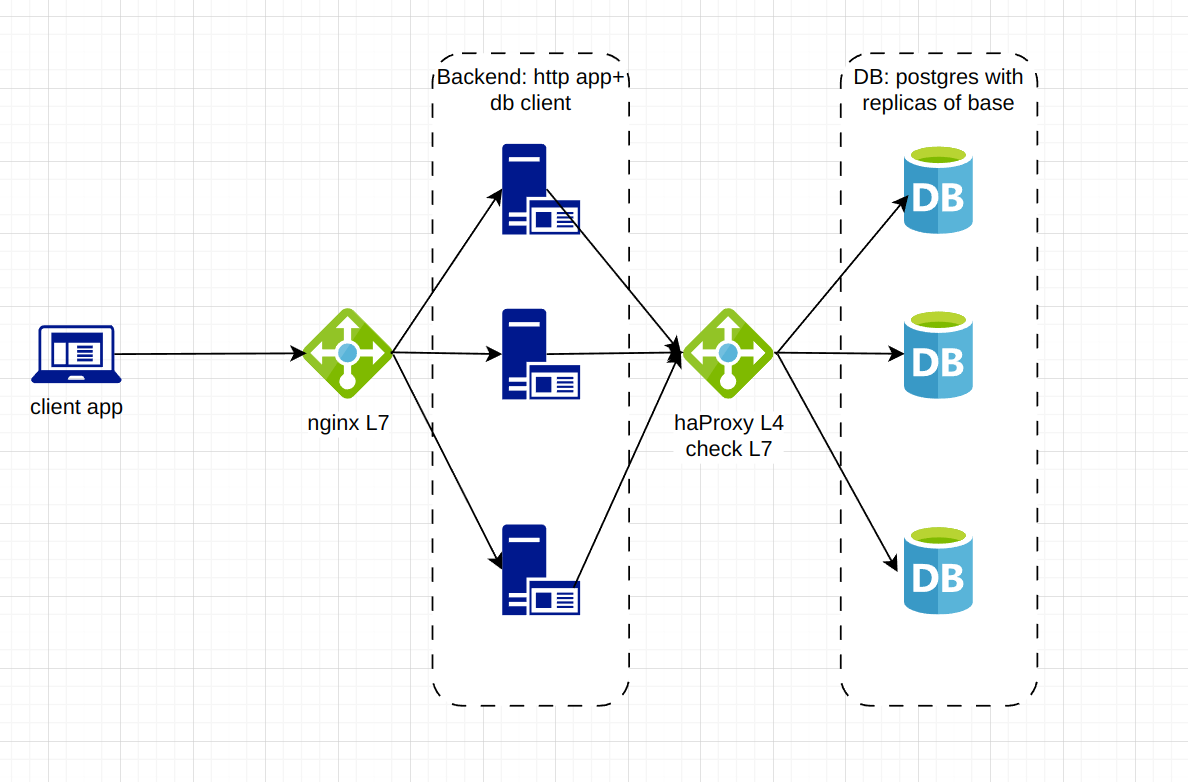
[Проверка доступа к БД через haproxy 5](#__RefHeading___3)

[Проверка доступа к Backend через nginx 7](#__RefHeading___4)

[Останов приложений и очистка: 10](#__RefHeading___5)

[Выводы 10](#__RefHeading___6)

# Реализованная схема



# Запуск:

git clone <https://github.com/filatkinen/ha-proxy-postgres>

cd cd ha-proxy-postgres/

make run

## Конфиг nginx

upstream nginx-backend {

server lab10\_backend01:8080;

server lab10\_backend02:8080;

server lab10\_backend03:8080;

}

server {

listen 8090;

location /nginx\_status {

stub\_status;

}

}

# HTTP Server Configuration

server {

listen 8080;

# Your website name goes here.

server\_name localhost nginx-lb.test;

# pass requests for dynamic content to rails/turbogears/zope, et al

location / {

proxy\_read\_timeout 1s;

proxy\_pass http://nginx-backend;

}

}

## Конфиг haproxy

global

maxconn 256

log stdout format raw local0

defaults

log global

mode tcp

timeout connect 10s

timeout client 50000ms

timeout server 50000ms

frontend stats

bind \*:8404

mode http

stats enable

stats uri /stats

stats refresh 10s

stats admin if TRUE

frontend postgres-proxy

bind \*:5432

mode tcp

default\_backend postgres

backend postgres

mode tcp

option pgsql-check user muser

balance leastconn

#default-server inter 2s downinter 5s rise 3 fall 2 slowstart 10s maxconn 64

server lab10\_pgsql01 lab10\_pgsql01:5432 check

server lab10\_pgsql02 lab10\_pgsql02:5432 check

server lab10\_pgsql03 lab10\_pgsql03:5432 check

## Проверка доступа к БД через haproxy

В скрипты запуска сервисов добавлены таймауты, поэтому после запуска необходимо подождать 1 минуту.

Проще воспользоваться каким ни-будь приложением, обеспечивающим удобный доступ к контейнерам докера и к консоли запущенных приложений.

Подключаемся к консоли клиента:

docker logs lab10\_client --follow

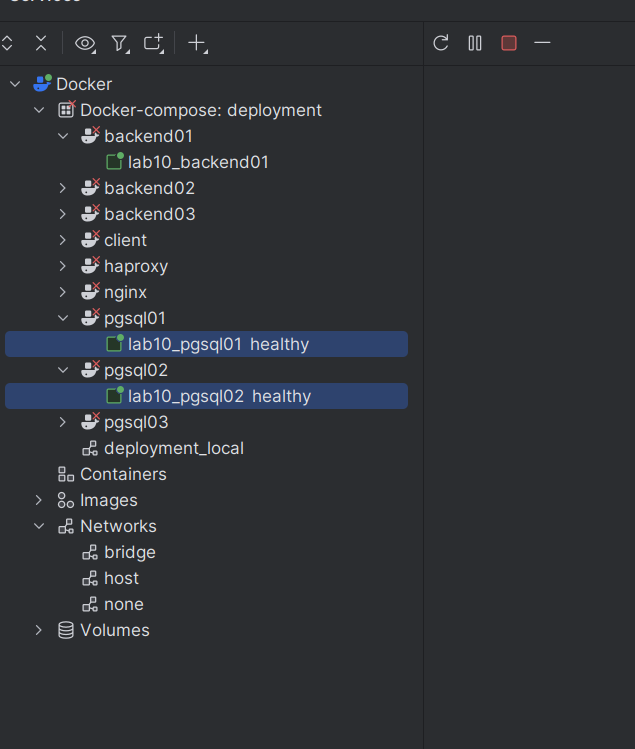
Видим бесконечные сообщения вида: это статусы потоков, которые опрашивают ручку приложения, находящегося за nginx, которые обращаются через haproxy к репликам БД. Статистика отдается без накопленного эффекта за 2 секунды. statusOK – количество выполненных запросов, status500 – ошибки сервера, если он получил ошибку от БД. err - ошибка модуля http клиента

Thread 2, statusOK=703, status500=0, err=0

Thread 3, statusOK=362, status500=0, err=0

Thread 0, statusOK=362, status500=0, err=0

Затем останавливаем 2 сервера БД через управялку докером(например плагин к vscode или другой среде разработки)



Что видим в логах клиента:

Thread 0, statusOK=1739, status500=0, err=0

Thread 2, statusOK=1733, status500=0, err=0

Thread 4, statusOK=1737, status500=0, err=0

Thread 0, statusOK=852, status500=2, err=0

Thread 2, statusOK=860, status500=2, err=0

Thread 1, statusOK=859, status500=1, err=0

Thread 4, statusOK=846, status500=4, err=0

Thread 3, statusOK=2190, status500=3, err=0

Thread 0, statusOK=1902, status500=0, err=0

Thread 3, statusOK=1919, status500=0, err=0

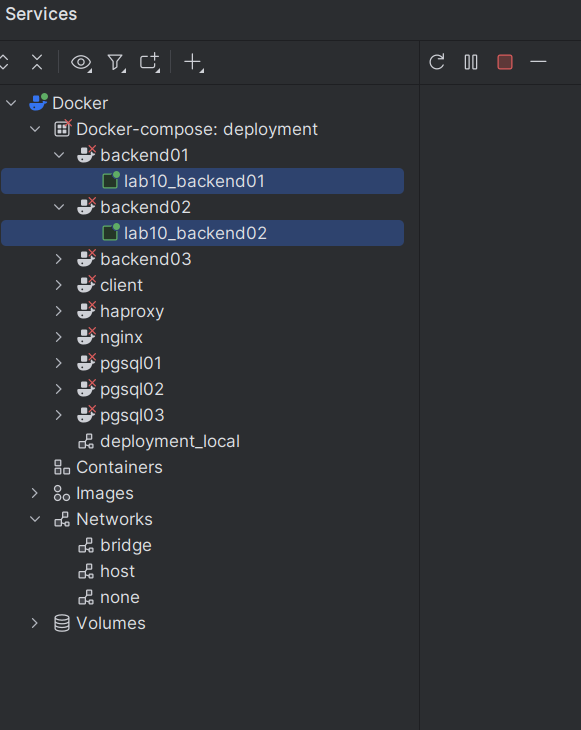
Thread 4, statusOK=1899, status500=0, err=0

Thread 2, statusOK=1899, status500=0, err=0

Появились 500-ки, и немного, и они быстро уходят, ну это нормально.

## Проверка доступа к Backend через nginx

Останавливаем 2 бэкенда:



Что видим:

Thread 3, statusOK=0, status500=2800, err=0

Thread 0, statusOK=3, status500=999, err=0

Thread 4, statusOK=1793, status500=1566, err=0

Thread 0, statusOK=298, status500=0, err=0

Thread 3, statusOK=1855, status500=1198, err=0

Thread 1, statusOK=197, status500=908, err=0

Thread 0, statusOK=1507, status500=0, err=0

Thread 1, statusOK=1510, status500=0, err=0

Thread 3, statusOK=1519, status500=0, err=0

Thread 4, statusOK=1509, status500=0, err=0

Thread 2, statusOK=4, status500=1196, err=0

Thread 2, statusOK=605, status500=0, err=0

Thread 4, statusOK=573, status500=0, err=0

Thread 3, statusOK=574, status500=0, err=0

Thread 0, statusOK=195, status500=0, err=0

Thread 2, statusOK=1046, status500=0, err=0

Идет рваный трафик. Я не понимал почему пока не ознакомился с документацией ngix:

<http://nginx.org/en/docs/http/load_balancing.html#nginx_load_balancing_health_checks>

Написано следующее:

**Health checks**

Reverse proxy implementation in nginx includes in-band (or passive) server health checks. If the response from a particular server fails with an error, nginx will mark this server as failed, and will try to avoid selecting this server for subsequent inbound requests for a while.

The [max\_fails](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive sets the number of consecutive unsuccessful attempts to communicate with the server that should happen during [fail\_timeout](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server). By default, [max\_fails](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) is set to 1. When it is set to 0, health checks are disabled for this server. The [fail\_timeout](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) parameter also defines how long the server will be marked as failed. After [fail\_timeout](http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) interval following the server failure, nginx will start to gracefully probe the server with the live client’s requests. If the probes have been successful, the server is marked as a live one.

**т.е. этот «продукт» чекает upsream с помощью живых запросов от пользователя.**

**Ну это как, Карл? Одно расстройство.**

Вообщем лучше haproxy юзать.

# Останов приложений и очистка:

make down

# Выводы

1. nginx очень разочаровал в части работы с upstream. Healthchek работает наживую от запросов пользователя. Можно перевести хосты в down в опции server, но какая это отказоустойчивость?
2. Странные логи в postgress. Вероятно это связано с тем, что запросы приходят от разных хостов с разными id. Могу только предполагать. Если есть ответ в отношении данного нюанса, то прошу дать.

2023-10-12T17:24:25.693191087Z 2023-10-12 17:24:25.693 UTC [227] LOG: PID 94 in cancel request did not match any process

2023-10-12T17:24:25.693671597Z 2023-10-12 17:24:25.693 UTC [228] LOG: PID 95 in cancel request did not match any process

2023-10-12T17:24:25.694338724Z 2023-10-12 17:24:25.694 UTC [229] LOG: PID 91 in cancel request did not match any process

2023-10-12T17:24:25.700864580Z 2023-10-12 17:24:25.700 UTC [230] LOG: PID 97 in cancel request did not match any process

2023-10-12T17:24:25.701536106Z 2023-10-12 17:24:25.701 UTC [231] LOG: PID 94 in cancel request did not match any process

2023-10-12T17:24:25.701905778Z 2023-10-12 17:24:25.701 UTC [232] LOG: PID 90 in cancel request did not match any process

2023-10-12T17:26:28.591470226Z 2023-10-12 17:26:28.591 UTC [497] LOG: PID 367 in cancel request did not match any process

2023-10-12T17:26:28.591901568Z 2023-10-12 17:26:28.591 UTC [498] LOG: PID 363 in cancel request did not match any process

2023-10-12T17:26:28.592715641Z 2023-10-12 17:26:28.592 UTC [499] LOG: PID 364 in cancel request did not match any process