ДЗ: Отказоустойчивость приложений

Для реализации отказоусточивости и ее проверки в проект были внесены следующие изменения:

1. На СУБД постгрес включена репликация (кворумная синхронизация)
2. Подняты 3 дополнительных ноды Postgres и подключены к кластеру (node2,node3,node4)
3. Для балансировки подключений к readonly БД поднят контейнер с haproxy, который слушает порт 5342 и перенаправляет подключение к одной из реплик.

Конфигурация ha proxy:

global

    maxconn 256

    log stdout format raw local0

defaults

    log global

    mode    tcp

    option  tcplog

    option  dontlognull

    option  logasap

    log-format "%H[%pid]\t[%t]\t%ft ->\t%ci:%cp %s %Tw/%Tc/%Tt %B %ts %ac/%fc/%bc/%sc/%rc %sq/%bq"

    retries 2

    timeout client 1m

    timeout connect 4s

    timeout server 1m

    timeout check 5s

listen stats

    mode http

    bind \*:7000

    stats enable

    stats uri /

    stats refresh 5s

listen pgReadOnly

    bind \*:5432

    option pgsql-check user haproxycheck

    default-server inter 3s fall 3

    server postgres.node2 postgres.node2:5432 check port 5432

    server postgres.node3 postgres.node3:5432 check port 5432

    server postgres.node4 postgres.node4:5432 check port 5432

В конфигурации также настроено логирование подключений к сервисам.

1. В коде сервиска один из ендпоинтов на чтение данных переключен на подключение к readonly реплики через haproxy (метод «/user/get/{userid}»).
2. Основной контейнер сервиса переименован в node1, дополнительно добавлен контейнер с его копией node2
3. Для балансировки подключений к сервису развернут контейнер nginx с конфигурацией:

upstream nginx-backend {

    server                      otuslab\_social.node1:8080 fail\_timeout=0 max\_fails=1;

    server                      otuslab\_social.node2:8080 fail\_timeout=0 max\_fails=1;

}

server {

    listen        9090;

    location /nginx\_status {

        stub\_status;

        allow 127.0.0.1;

        allow 10.0.0.0/8;

        allow 172.16.0.0/12;

        allow 192.168.0.0/16;

        deny all;

    }

}

log\_format compression '[$time\_local]\t$proxy\_add\_x\_forwarded\_for ->\t$upstream\_addr\t"$request" $status $body\_bytes\_sent "$http\_user\_agent"';

server {

    listen                      80;

    server\_name                 otuslab;

    location / {

        proxy\_read\_timeout 1s;

        proxy\_connect\_timeout 1s;

        proxy\_set\_header   Host             $host:$server\_port;

        proxy\_set\_header   X-Real-IP        $remote\_addr;

        proxy\_set\_header   X-Forwarded-For  $proxy\_add\_x\_forwarded\_for;

        proxy\_pass http://nginx-backend;

        access\_log /dev/stdout compression;

    }

}

1. В настройках nginx включено логирование
2. Для создании нагрузки был запущен python скрипт, который в течение 10 минут каждый 250ms делает обращение к методу «http://localhost/user/get/8eb54d56-15eb-4305-a2e2-c2d09ababe8f» и выводит в консоль код ответа.

**Эксперимент:**

Запущен скрипт нагрузки.

По данным из docker были выбрана нода реплики и сервиса, на которой есть нагрузка (загружен CPU выше других нод) для удаления (node3 БД и node2 сервиса).

1. Остановлена node3 реплика СУБД:
   1. Сервис работает, ответы идут с 200 кодом и данными пользователя.
   2. В логах haproxy node3 перешла в состояние DOWN, нагрузка переключилась на node2
2. Остановлена node2 сервиса:
   1. Скрипт продолжает получать ответы с кодом 200 и данными пользователя
   2. В логах nginx информация о недоступности node2, все запросы уходят к node1 и обрабатываются там.
3. Реплика и отключенная нода сервиса включены обратно, в логах балансировщиков информация о доступности всех нод, данные распределяются равномерно.

Эксперемент показал, что балансировщики нагрузки могут сохранять общую работоспособность сервиса, даже если какие-то из нод выйдут из строя, при балансировки нагрузки как на L7 по http, так и на L4 по TCP.