?????????? ???????? ????????????????

Здесь описано, как настроить балансировку по схеме "клиент-прокси-шлюз-прокси-приложение" и почему я выбрал именно эту схему для своей инфраструктуры

?? ??? ???????? ??? ???????????

Перед тем, как переходить к вопросам настройки балансировки, важно понимать: а что именно мы баллансироуем? С чем мы работаем? Какие проблемы мы решаем?

Без ответа на эти вопросы, вряд-ли у нас получится сделать грамотную балансировку. Так, какие проблемы сы сегодня будем решать?

???????? 1

В моей инфраструктуре существует единственная проблема, помимо моих собственных кривых рук, - это мой провайдер.

У МГТС, как известно, достаточно закрытая экосистема. Все мои попытки общения с ними сводились к примерно таким диалогам:

- Хочешь установить свое оборудование? Нет, у нас уже есть наше;
- Хочешь настроить свои правила маршрутизации? Нет, не хочешь;
- Хочешь управлять маршрутизатором, который ты у нас когда-то купил? Любой каприз, но не с нашим сетевым каналом.

Это привело меня к мысли, что настраивать что-то сложное мне просто нет смысла - даже самый простой, казалось бы, функционал, работает через очко на моем F660.

???????? 2

Более того, такая технология как "объединение VLAN" для этой рокетки - слишком сложая. Я не могу объединить 2 или более физических "очка" в 1 сеть. Допустим, у меня 2 основных хоста в LAN:

- Мой основной ПК (допустим, 192.168.1.99);
- Мой сервер виртуализации (допустим, 192.168.1.100).

И оба подключены физически, через провод в разные порты маршрутизатора. Так вот при такой схеме, я не могу отправлять запросы между портами маршрутизатора - это может делать только сам F660. Это значит, что при попытке отправить любой пакет с одного хоста в сторону второго, он потеряется.

На первый взгляд это не кажется проблемой. Но на практике это совсем не так. Например, чтобы просто попасть в панель управления виртуализатором, мне нужно ходить к нему через WAN, ведь соединение LAN-LAN не пройдет. Это означает, что она должна быть открыта для всего интернета. То же самое касается и вопросов удаленного доступа к ВМ - это становится просто невозможным.

??? ?? ???????

Благо, железо, с которым я работаю, имеет ограниченно-рабочую возможность проброса портов. Почему ограниченную? Потому что диапазон больше 32 значений для маршрутизатора считается невалидным значением. Вы не можете одним правилом прокинуть, например, такой диапазон портов: [8001-8128], потому что это приведет к ошибке.

Но вы можете создать несколько правил, прокидывая диапазоны постепенно, мало-помалу. Правда, и здесь есть ограничение - вы не сможете создать больше пары десятков правил, потому что вам не хватит памяти устройства, выделенной на их хранение. Здесь нужно действовать аккуратнее и без фанатизма.

??? ?????? ??? ????????? ?? ?????????

Решая эту проблему, я придумал другую схему, которая не требует сложной настройки. **Я взял за основу 3 принципа**:

1. Касательно публикации сервисов во вне: Один сервис в LAN - 1 порт в WAN (не совпадающий с реальным портом приложения)
2. Касательно ИБ:
   - Действие внешнего пользователя в инфраструктуре может сломать только ее часть, но не вывести из строя весь ЗК;
   - Нет смысла прятать скрывать информацию о системе, которая открывается для WAN - важнее позаботиться о невозможности (по крайней мере, усложнении) ее эксплуатации;
3. Касательно удаленного доступа: одна "магистральная" ВМ, с которой осуществляется доступ в закрытый контур. Для SSH эта ВМ служит JUMP-хостом.

Для остальных протоколов - аналогично, это обертка с единственным пользователем без sudo и права входа в шелл (nologin-user). Получается вот такая схема:

```
WAN User => Gateway F660 => LAN jump-host => Target application
```

Эти принципы не единственные, но на мой взгляд основные - на них строится моя инфраструктура.

Если же мы говорим о публичных сервисах (опубликованных в WAN), то для доступа к ним используется внешний прокси - единственный, который может обращаться на открытые порты на маршрутизаторе. Это одновременно и единственная точка отказа (ака - бутылочное горлышко, но никто не запрещает иметь сразу несколько WAN-прокси с балансировкой). Вот простая схема для понимания того, как это работает:

basic-infra-schema-principals.png

??? ?? ????????????

В дисциплине программирования существует раздел, именуемый "Правила построения чистой архитектуры". На Habr есть статья, посвященная этому. Но моя позиция состоит в том, что "чистая архитектура" не ограничивается в применении только областью кода. Рассматривая его в контексте, один из принципов, который я стараюсь соблюсти и в своей инстраструктуре - невозможность влияния компонентов на уровень, от которых они зависят.

Тут получается интересная ситуация. Возьмем, приложение А. Оно выполняет свою задачу, но имеет дополнительный функционал, который завязан на дугих службах в инфраструктуре.

Например:

1. **Nextcloud** имеет функционал хранения файлов, но использует почтовый сервер для отправки уведомлений;
2. **Grafana** имеет функционал для отображения графиков, но использует Prometheus для их сбора и LDAP для аутентификации пользователей;
3. Написанный мной **сервер телеграмм-бота** умеет выполнять разные виды сценариев, но использует СУБД для хранения информации.

Задача - сделать так, чтобы в ситауции "приложение А недоступно", сервис, который оно использует, не встал раком. Как это сделать проще всего? Как такового единого стандарта в сфере не существует, и уж тем более - для домашних инсталляций. Поэтому, наше любимое - **натянуть сову на глобус**. Взять набор принципов, которые формулировались совсем для другой индустрии, и применить его в нашем контексте, практически без изменений.

Я не говорю, что это - лучший подход, но выбирая между "строить инфраструктуру как карта ляжет или левая пятка скажет" и "попробовать структурировать подход к построению инфраструктуры", я смело выбираю второе, а именно - **SOLID**. Если вы не знакомы с этой метой - вот статья на Habr, в которой автор достаточно подробно рассказывает о том, что это такое.

А мы вернемся к инфраструктурному вопросу и переопределим значения принципов SOLID, используя наш конеткст:

Что переопределяем	**Как определено изначально**	**На что мы заменяем**
Single Responsibility Principle	Класс должен выполнять только одну задачу. У него может быть несколько методов, но все они работают вместе, чтобы выполнять одну основную задачу.	Приложение, используемое для решения одной задачи, должно решать ее без дополнительных модулей и интеграций с другими системами Например, если я хочу развернуть у себя в инфраструктуре мессенджер, его минимальная конфигурация на 1 сервере (ВМ) должна иметь функционал мессенджера, без необходимости подключения дополнительных служб или модулей
Open-Closed Principle	Сущности программы (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменений	Здесь мы рассматриваем "сущность программы" как сервис, на котором завязано приложение. Если переформулировать: Приложения могут использовать функционал внешних сервисов для расширения своего функционала, но не могут на них основываться
Liskov Substitution Principle	Объекты в программе должны быть заменяемы экземплярами их подтипов без изменения базовой логики работы программы	Оригинальная формулировка принципа звучит сложно, но проще для понимания на практике: При замене какого-то сервиса на аналогичный, функциональность инфраструктуры может только расширяться (но не уменьшаться) Например, я могу соблюсти этот принцип, если я захочу заменить RocketChat в своей инфраструктуре на связку Matrix-Synapse, т.к. он имеет тот же функционал, что и RC, но расширяет его наличием шифрования и отсутствием лицензионных ограничений на кол-во пользователей в системе

Interface Segregation Principle	Ни один клиент не должен зависеть от методов, которые он не использует	Подробнее я рассмотрел его выше, в начале этой главы. Переформулировав: В случае отказа системы, с которой интегрировано приложение, допустима потеря только той части функционала, которая завязана на этой системе (недопускается отказ всего функционала приложения) Этот принцип так же ссылается на мою формулировку для Open-Closed Principle - при его соблюдении, отказ системы в такой описанной ситуации (по этим причинам) не должен быть невозможен в целом
Dependency Inversion Principle	1. Модуль высокого уровня не должен зависеть от модулей низкого уровня. И то, и другое должно зависеть от абстракций; 2. Абстракции не должны зависеть от деталей реализации. Детали реализации должны зависеть от абстракций.	Это - единственный принцип, который мы **выбрасываем из списка**, потому что в инфраструктуре не может быть сущностей, которые безпроблемно соотносились с теми же сущностями в коде. Вышеописанное "SOLI", при его соблюдени, уже по-умолчанию реализует все необходимые принципы "чистой архитектуры", а при аналогии с кодом, когда вы добираетесь до этого пункта, вы лишь проверяете выполнение предыдущих правил, потому что они уже неявно регламентируют его определение в конкретной реализации *(бля, я лучше чем это сейчас не сформулирую, вот честно)*

Держите это в голове, если думаете о моей инфраструктуре - именно так я ее вижу. Дальшейшие статьи - пример, как я применяю их у себя в инфраструктуре, чем я руководствуюсь при расширении/изменении существующего функционала

????? ????? ??????? ? ????????

Напомню, как выглядит наша схема.

basic-infra-schema-principals.png

Если рассматривать сетевой путь от клиента до конечного сервиса, здесь есть 3 основных узла: WAN-proxy, сам маршрутизатор (A660) и LAN-proxy.

В случае с **маршрутизатором** - здесь все просто:

1. Входим в ПУ вашего маршрутизатора
2. Идем в "Приложение" -> "Переадресация порта"
3. Настраиваем переадресацию на хост-балансировщик, находящийся в вашей домашней сети. Например, для адреса 192.168.1.102 и диапазона портов 50001-50032, настройка будет примерно такой:

image.png

На места этого IP-адреса может быть любой из вашей сети, единственный критерий - к нему должен быть доступ от маршрутизатора. Проверить это можно в "Администрирование" -> "Диагностика Ping".

А вот дальше уже сложнее...

Многие задаются вопросом: а зачем вообще открывать диапазоны портов и потом как-то сложно их маппить? Можно же просто открыть, например, 443 порт, и хостить балансировку сразу за ним.

Да, можно, но только не на F660 - эта рокетка уже использует 443 порт в LAN и, открывая его и в WAN, происходит ситуация, которая решается только физическим сбросом маршрутизатора. ПУ перестает открываться, а перенаправление не работает. Как? Не спрашивайте, потому что на нормальном железе (например, на Keenetic или TPlink) такой ситуации не происходит: они видят разницу между LAN и WAN подключениями.

Помимо этого, вы не сможете скрыть своей реальный IP-адрес, тем самым вам придется столкнуться с 2 нюансами:

1. Вам придется реализовать блокировку доступа к ПУ не на уровне маршрутизатора, а на уровне Nginx на самой ноде PVE (или прокси). Казалось, бы, это не проблема? Я напоминаю - сам хост стоит в LAN, за маршрутизатором, и заголовок Source-EA у всех пакетов, которые будут к нему приходить, всегда будут содержать только 1 IP-

адрес - вашего маршрутизатора. При таком подходе и на этом железе вы не сможете "отсеять" нежелательные подключения к ПУ, как следствие - она будет торчать в интернет as-is.
2. При попытке положить вам хост, вы рискуете лишиться не только его самого, но и всего интернета - маршрутизатор уйдет в перезагрузку, а вместе с ним и весь ваш канал.

Балансировщик решает эту проблему и дает возможность тонкой настройки единой точки входа:

- Хотите общий WAF для всех приложений? Настраивайте его на балансировщике, ведь именно через него проходит весь трафик от пользователя до приложения;
- Хотите запретить доступ сразу ко всем пользователям? Просто погасите балансировщик, и хуй кто из пользователей воспользуется вашей инфраструктурой (моментами, может сыграть против вас);
- Такая архитектура - один из моментов, когда мы применяем наш костыльный "SOLI", а именно - Single Responsibility. Функционал сервисов - на самих сервисах, а функционал доступа к ним - на отдельных хостах.

Чтобы настроить балансировку со стороны WAN

????????? ??????????????

В этой главе описана настройка внешней балансировки (на практике)

?????????????? ????? ???????

Общая схема прохождения трафика выглядит следующим образом:

Proxy-Route-MAP.png

Здесь все пользователи, для доступа к целевому ресурсу, проходят через единую точку входа - WAN-proxy.

?????????? ?????????????? ????????

В общем случае, мы будем использовать HAProxy как решение для балансировки нагрузки. Установка для большинства Debian-based дистрибутивов Linux будет следующей:

```
# Синхронизация даты и времени (для примера - Мск_)
timedatectl set-timezone Europe/Moscow
# Обновление индекса пакетного менеджера и установленных пакетов
apt update && apt upgrade -y
# Установка
apt install -y haproxy
```

А дальнейшая настройка зависит от того, где именно будет выполняться балансировка: в WAN или LAN-сегменте.

????????? HAProxy ? WAN-????????

Для WAN-прокси основной задачей является соотношение доменного имени (SNI), к которому обращается клиент, с целевым ресурсом, который находится в LAN-сегменте за нашим шлюзом (маршрутизатором). В общем случае конфигурация будет состоять из 2 этапов:

????????? ????????? ???????????????? ?????

- Основной конфигурационный файл HAproxy располагается по следующему пути: /etc/haproxy/haproxy.cfg
- Пример основной конфигурации:

```
global
    log /var/log/haproxy local0 info
    log /var/log/haproxy local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

    # Оптимизация производительности
    maxconn 100000
    tune.ssl.default-dh-param 2048
    ssl-default-bind-options no-sslv3 no-tlsv10 no-tlsv11
    ssl-default-bind-ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384
    ssl-default-server-options no-sslv3 no-tlsv10 no-tlsv11
    ssl-default-server-ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384

    # Увеличенные буферы для больших загрузок
    tune.bufsize 32768
    tune.maxrewrite 1024

defaults
```

```
    log     global
    mode    tcp
    option  tcplog
    option  dontlognull
    option  tcpka
    option  clitcpka
    option  srvtcpka
    timeout connect 10s
    timeout client 1h
    timeout server 1h
    timeout tunnel 1h
    retries 3

# Мониторинг HAProxy (опционально)
frontend stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 10s
    stats admin if LOCALHOST

# Основной фронтенд для HTTPS трафика
frontend https-in
    bind *:443

    # ACL для определения сервисов по SNI
    acl host_nextcloud req.ssl_sni -i nextcloud.yourdomain.com
    acl host_service1 req.ssl_sni -i service1.yourdomain.com
    acl host_service2 req.ssl_sni -i service2.yourdomain.com

    # Используем таблицу для динамического маршрутизации (альтернативный вариант)
    use_backend %[req.ssl_sni,map_str(/etc/haproxy/sni-map.map,unknown_backend)]

    # ИЛИ использовать статические правила (раскомментировать если нужно):
    # use_backend nextcloud_backend if host_nextcloud
    # use_backend service1_backend if host_service1
    # use_backend service2_backend if host_service2

    # TCP-инспекция для извлечения SNI
    tcp-request inspect-delay 5s
```

```
    tcp-request content accept if { req_ssl_hello_type 1 }

# Бэкенд по умолчанию для неизвестных доменов
backend unknown_backend
    mode tcp
    option tcplog
    timeout server 5s
    server unknown 127.0.0.1:81

# Шаблон для обычных сервисов
backend service_template
    mode tcp
    option tcplog

    # Параметры балансировки (если несколько серверов)
    balance roundrobin

    # Настройки проксирования
    option tcp-check
    tcp-check connect

    # Поддержка Keep-Alive
    option clitcpka
    option srvtcpka

    # Увеличенные таймауты
    timeout connect 10s
    timeout server 1h
    timeout tunnel 1h

    # Сервер шаблона (заменить в конфигах сервисов)
    server template-server GATEWAY_IP:TEMPLATE_PORT check inter 2s fall 3 rise 2
```

Вот основные моменты этой конфигурации:

1. Настраиваем логирование (/var/log/haproxy)
2. Настраиваем бэкенд по-умолчанию (127.0.0.1:81, что в обычной конфигурации приведет к ошибке ConnectionTimeout)
3. Используем карту SNI, которая располагается здесь: /etc/haproxy/sni-map.map;
4. Разрешаем публичную статистику (она доступна на порту этого сервера, *:8404);
5. Мы используем TCP-прокси, т.е. не инкапсулируем входящие пакеты.

Логика работы со стороны Haproxy такая:

1. Возьми SNI из пакета, который тебе пришел от клиента;
2. Соотнеси полученный SNI с твоей базой (картой), которая лежит в /etc/haproxy/sni-map.map;
3. Если ты нашел доменное имя, на которое обращается клиент - проксируй весь трафик туда (как есть);
4. Если клиент общается на SNI, которого нет в твоей карте - отправляй его на самого себя же, на порт *:81 (там может стоять Nginx с сайтом-заглушкой).

????????? ????? ???????? (SNI)

Файл маппинга, согласно конфигурации выше, лежит по пути /etc/haproxy/sni-map.map. Изменим его, примерно следующим образом:

```
# Формат: доменное_имя бэкенд
nextcloud.yourdomain.com nextcloud_backend
service1.yourdomain.com service1_backend
service2.yourdomain.com service2_backend
# Добавляйте новые сервисы здесь
```

????????? ???????????????? ????? ??? ?????? ?? ????? (????? ???????)

??? **LAN**

(???????????) ????????? ???????????? ? ??????? ???????????????? ? LAN-????????