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

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

Revision #14
Created 2025-12-01 02:11:05 UTC by root
Updated 2025-12-01 04:49:27 UTC by root