Задание 1.

1. Создание и настройка ВМ. Ставим Debian, без графического интерфейса (только консоль). Далее нужно настроить удаленное подключение по SSH. Для подключения из Windows можно использовать MobaXterm или его аналоги. После настройки можно будет работать с сервером через SSH (с подсветкой текста и прочими удобствами).

2. Понять, в чем различие виртуальных машин NAT и Bridge

3. Установить Wireshark (или tshark, если мы в консоли). Поучиться захватывать трафик с разных адаптеров, почитать про фильтрацию пакетов.

4. Узнать, что такое утилита nc. То есть, создаются две ВМ — одна будет сервером, другая клиентом. Откроется чат и можно туда что-то послать. Нужно переслать сообщение "secretkey" с одной ВМ на другую ВМ. В этот же момент с помощью Wireshark отлавливаем пакет. Ожидаем увидеть, что он передается в открытом виде.

5. Необязательно: установить cryptcat. В Wireshark ожидаем увидеть, что все данные шифруются.

1. Создание и настройка ВМ.

1) Настройка Debian без графического интерфейса:

**hostname** - любое (например debian-1)

**domain name** - local.domain

**full name** - Test User

Guided - use entire disk

**Scan extra installation media**? - no

**Debian archive mirror country** - Russian Federation, затем ftp.ru.debian.org

**HTTP proxy** – пропускается, оставляем поле пустым

**software to install** - галочки только на SSH и Standard system utilities, со всего остального убрать

**GRUB** – yes

2. Настройка удаленного подключения по SSH.

На созданной ВМ производим ряд настроек:

а) Установка sudo:

su -

apt update

apt install sudo -y

usermod -aG sudo <username>

exit

б) Установка SSH:

sudo apt update

sudo apt install openssh-server -y

sudo systemctl start ssh

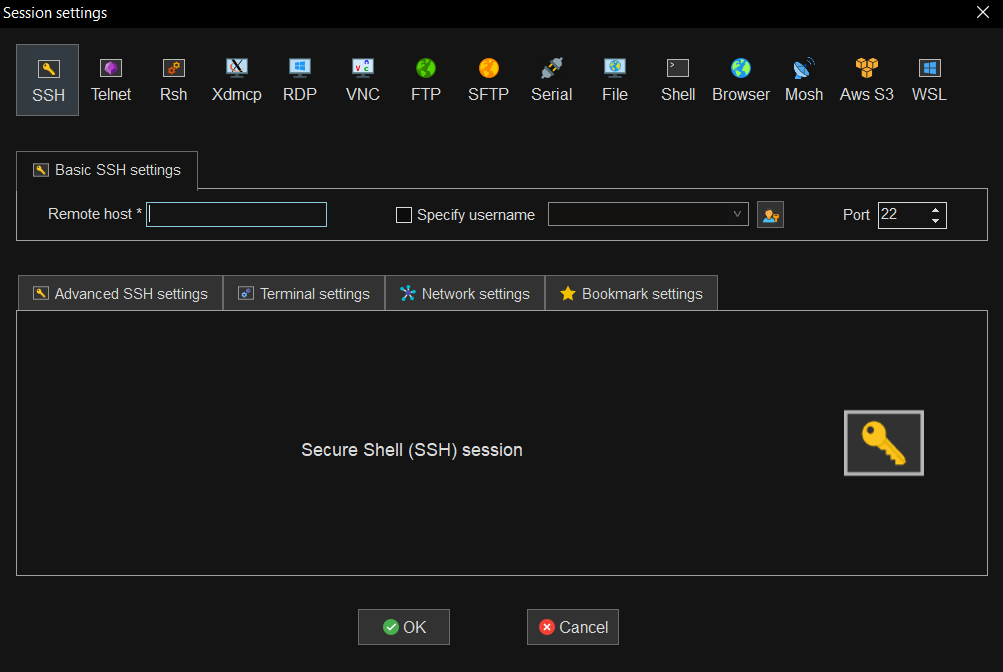
sudo systemctl enable ssh

Проверка, работает ли SSH: sudo systemctl status ssh

Теперь нужно узнать IP-адрес ВМ: ip a

В выводе ищем интерфейс (например, у меня ens33) и IP-адреса (например, у меня 198.168.0.106).

Скачивается MobaXterm, запускается. Далее “Session” -> “SSH”.



**Remote host** – сюда пишется IP-адрес ВМ.

**Specify username** – ставится галочка, вводится имя пользователя, установленное на VMware

**Port** – не трогать.

Нажимаем ОК, открывается окошко, вводим туда пароль, установленный в VMware. В принципе на этом всё, теперь мы в консоли Debian через SSH.

2. В чем разница виртуальных машин NAT и Bridged?

**Bridged:**

* ВМ получает IP из той же сети, что и наш компьютер. Допустим, если у нас 192.168.1.10, то ВМ получит 192.168.1.11.
* ВМ ведет себя как отдельное устройство в сети.
* Выбираем Bridged, если нужно взаимодействие между ВМ и другими устройствами в локальной сети.

**NAT (Network Address Translation):**

* ВМ получает IP из внутренней сети VMware.
* Внешний мир видит только IP нашего компьютера, а не ВМ.
* Выбираем NAT, если нужно, чтобы ВМ имела интернет, но была скрыта от локальной сети.

В данном задании нам нужны ВМ именно Bridged, поэтому выбираем его.

3. Установка Wireshark (tshark) и захват трафика.

Мы работаем в консоли, поэтому устанавливаем tshark:

sudo apt update

sudo apt install tshark –y

Чтобы захватить трафик в tshark, пишем следующее:

sudo tshark -i ens33

тут ens33 – это имя интерфейса, которое можно узнать с помощью “ip a”.

4. Работа с Netcat (nc). Перехват сообщений.

Для этого пункта нам нужны две ВМ. Одна у нас уже есть и она настроена так, как нам надо. Создадим вторую: ВМ -> “Manage” -> “Clone”.

Выбираем “Create a full clone”, указываем новое имя (дадим первой ВМ имя Debian-1, а второй – Debian-2).

Получаем IP-адрес второй ВМ и открываем вторую SSH-сессию в MovaXterm с этим IP. Теперь у нас два окошка для двух ВМ.

Проверка, что ВМ видят друг друга.

С первой ВМ: ping <IP\_Адрес\_Debian-2> и наоборот.

Устанавливаем netcat на обе ВМ:

sudo apt update

sudo apt install netcat –y

На первой ВМ (допустим, Debian-1) запускается сервер:

nc -l -p 1234 (слушаем порт 1234)

Подробнее о параметрах:

* -l — режим сервера (listen).
* -p 1234 — порт.

На второй ВМ (Debian-2) подключаем клиента:

nc <IP\_Адрес\_Debian-1> 1234

Теперь открываем еще одно окно MobaXterm и подключаемся к Debian-1 второй раз.

Выполним захват трафика на интерфейсе:

sudo tshark -i ens33 -f "tcp port 1234" -Y "tcp.payload" –V

Подробнее про параметры:

* -i ens33 — захват трафика на интерфейсе ens (смотрим интерфейс через  “ip a”).
* -f "tcp port 1234" — фильтр для захвата только TCP-пакетов на порту 1234.
* -Y "tcp.payload" — показывать только полезную нагрузку (само сообщение).
* -V — подробный вывод (чтобы видеть содержимое пакета).

Возвращаемся в окошко Debian-2, где запущен nc-клиент.

Отправляем какое-нибудь сообщение, например, “secretkey” -> Enter.

Открываем окно с Debian-1, где ранее выполняли захват трафика. Там увидим вывод tshark, в том числе и наш отправленный текст “secretkey”, который, как наглядно видно, передается в открытом виде.

Если бы данные шифровались, вместо “secretkey” мы бы увидели какие-нибудь случайные байты. Но в netcat трафик всегда открытый, поэтому tshark показывает сообщения такими, какие они есть.

5. Работа с Cryptcat . Перехват сообщений.

Устанавливаем cryptcat на обе ВМ:

sudo apt update

sudo apt install cryptcat –y

Теперь подобно тому, как мы работали с netcat, будем работать с cryptcat.

На первой ВМ (Debian-1) запускается сервер:

cryptcat -l -p 1234 -k mysecretkey

Подробнее о параметрах:

* -l — режим сервера (listen).
* -p 1234 — порт.
* -k mysecretkey — ключ шифрования (можно задать любой, но он должен совпадать на сервере и клиенте).

На второй ВМ (Debian-2) подключаем клиента:

cryptcat <IP\_Адрес\_Debian-1> 1234 -k mysecretkey

Теперь открываем еще одно окно MobaXterm и подключаемся к Debian-1 второй раз.

Выполним захват трафика на интерфейсе:

sudo tshark -i ens33 -f "tcp port 1234" -Y "tcp.payload" –V

Возвращаемся в окошко Debian-2, где запущен nc-клиент.

Снова отправляем какое-нибудь сообщение: “secretkey” -> Enter.

Открываем окно с Debian-1, где ранее выполняли захват трафика. Там увидим вывод tshark, но теперь сообщение будет передаваться в зашифрованном виде. Вместо “secretkey” увидим случайные байты.

Само переданное расшифрованное сообщение можем увидеть на Debian-1 (где запускался сервер).

Вывод по netcat и cryptcat:

|  |  |  |
| --- | --- | --- |
| **Критерий** | **netcat** | **cryptcat** |
| Шифрование | Открытый текст | Зашифрованный текст |
| Отображение в tshark | secretkey | Случайные байты |
| Ключ | Не нужен | Обязательный (-k) |

Задание 2.

1. Виртуальные машины vs контейнеры.

2. Что такое Docker и для чего он нужен? Необходимо установить его и попробовать какие-то базовые команды.

3. Необходимо создать контейнеры на двух ВМ и настроить IPSec между этими контейнерами.

1. Виртуальные машины vs контейнеры.

Виртуализация – это технология, которая позволяет запускать несколько виртуальных машин (ВМ) на одном физическом сервере. Каждая ВМ – это полноценная копия операционной системы со своим ядром, диском, сетью и ресурсами.

Основной принцип:

1. Гипервизор (VMware, VirtualBox и др.) создает изолированные ВМ.

2. Каждая ВМ эмулирует реальное железо (CPU, RAM и др.).

3. Гостевая ОС внутри ВМ не знает, что работает на виртуальной среде.

Контейнеризация – это технология, которая позволяет запускать изолированные процессы (контейнеры) на одном ядре ОС. Контейнеры делят ядро хоста, но имеют свои файловые системы и сетевые настройки.

Основной принцип:

1. Движок контейнеризации (Docker, LXC) создает “легкие” изолированные среды.

2. Контейнеры используют ядро хостовой ОС (нет эмуляции железа).

3. Всё, что нужно приложению, упаковывается в образ (image).

Сравнение виртуальных машин и контейнеров:

|  |  |  |
| --- | --- | --- |
| Критерий | Виртуальные машины | Контейнеры |
| Изоляция | Полная (своя ОС, ядро) | Частичная (общее ядро) |
| Загрузка | Дольше | Быстрее |
| Ресурсы | Требуют много CPU, RAM, диска | Экономные |
| Безопасность | Высокая | Более низкая |
| Использование | Запуск разных ОС (Windows/Linux, macOS) | Только ОС с тем же ядром |
| Примеры | VMware, VirtualBox, Hyper-V | Docker, LXC |

2. Что такое Docker и для чего он нужен

Docker – это платформа для контейнеризации приложений.

Контейнер – это “легкая виртуальная машина”, которая запускает приложение в изолированной среде, использует ядро хоста и запускается за секунды, потребляя минимум ресурсов.

Условно: ВМ это здание или дом (ОС + приложения), а контейнер – это комната в доме (только приложение + его зависимости).

Устанавливаем Docker на обеих ВМ:

sudo apt update

sudo apt install docker.io -y

Можно также добавить нашего пользователя в группу docker (это нужно для того, чтобы не писать sudo перед каждой командой):

sudo usermod -aG docker $USER

newgrp docker (применение изменений)

Внимание!! При команде ‘sudo usermod -aG docker $USER’ во второй ВМ может возникнуть проблема: "sudo: unable to resolve host debian-2: Name or service not known".

Чтобы ее решить, необходимо выполнить команду на второй ВМ:

sudo nano /etc/hosts

Откроется файл, где будет указано имя debian-1 (возможно даже не один раз). Нужно просто в этом файле заменить имя debian-1 (имя первой ВМ) на debian-2 (имя второй ВМ). Содержимое файла получилось идентичным из-за того, что вторая ВМ создавалась путем клонирования первой. Просто меняем неправильное имя на нужное, сохраняем файл и снова пытаемся ввести команду. На этот раз должно быть всё хорошо.

После установки проверим, что всё ок:

docker –version

Теперь, когда Docker установлен, можно попробовать базовые команды.

1. **docker run** – запустить контейнер.

Пример: docker run hello-world

2. **docker ps** – посмотреть список запущенных контейнеров.

Пример: docker ps – a (посмотреть все контейнеры)

3. **docker images** – посмотреть список образов.

Пример: docker images (так и пишется)

4. **docker stop** – остановить контейнер.

Пример: docker stop <ID\_контейнера>

5. **docker rm** – удалить контейнер.

Пример: docker rm <ID\_контейнера>

3. Создание контейнеров и настройка IPSec

Запускаем две ВМ (Debian-1 и Debian-2)

Задание 3.

Настройка кластера.

1. Настройка Corosync

2. Настройка Apache

3. Настройка HAProxy

4. Настройка Pacemaker

Имеются две виртуальные машины:

Debian-1 – 192.168.0.103

Debian-2 – 192.168.0.102

Начинаем базовую настройку. На обеих ВМ выполняется следующее:

1. Обновление пакетов:

sudo apt update && sudo apt upgrade -y

2. Установка всего, что нам нужно:

sudo apt install -y corosync pacemaker haproxy apache2 crmsh

3. Перезапуск:

sudo reboot

1. Настройка Corosync

1. Сначала надо сгенерировать ключ. На Debian-1:

sudo corosync-keygen

2. Ключ копируется на Debian-2:

sudo scp /etc/corosync/authkey wwoxero@192.168.0.102:/tmp/

3. Теперь на Debian-2 берем скопированный ключ и перемещаем в нужную папку:

sudo mv /tmp/authkey /etc/corosync/

4. На обеих ВМ нужно настроить corosync.conf (это конфигурация).

Открываем:

sudo nano /etc/corosync/corosync.conf

Редактируем конфигурацию и получается примерно что-то вот такое:

totem {

version: 2

cluster\_name: debian

transport: udpu

crypto\_cipher: none

crypto\_hash: none

interface {

ringnumber: 0

bindnetaddr: 192.168.0.0 (Подсеть, где работают узлы)

broadcast: yes

}

}

logging {

fileline: off

to\_stderr: yes

to\_logfile: yes

logfile: /var/log/corosync/corosync.log

to\_syslog: yes

debug: off

logger\_subsys {

subsys: QUORUM

debug: off

}

}

nodelist {

node { (Информация о первой ВМ)

name: debian-1

nodeid: 1

ring0\_addr: 192.168.0.103

}

node { (Информация о второй ВМ)

name: debian-2

nodeid: 2

ring0\_addr: 192.168.0.102

}

}

quorum {

provider: corosync\_votequorum

two\_node: 1 (Разрешить работу без quorum для 2 узлов)

}

5. Теперь Corosync включается и запускается:

sudo systemctl enable corosync

sudo systemctl restart corosync

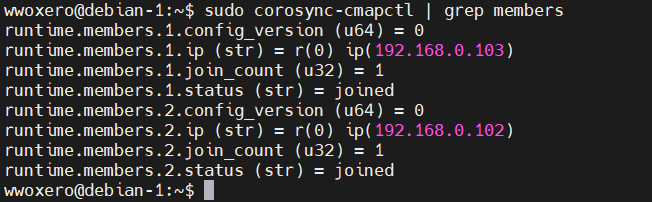
6. Проверка, что узлы друг друга видят:

sudo corosync-cmapctl | grep members

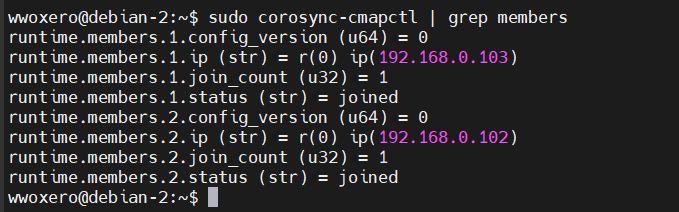
(Вывод должен содержать IP-адреса обеих машин).

Вот результаты:

С Debian-1:

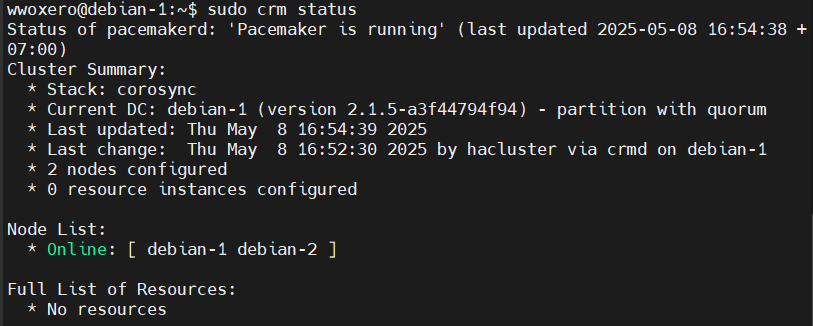


С Debian-2:



Еще можно проверить статус, оба узла должны быть Online:

sudo crm status



Соединение есть.

2. Настройка Apache

1. Меняем порт Apache (на обеих ВМ!!):

sudo nano /etc/apache2/ports.conf

Тут нужно заменить Listen 80 на Listen 81.

2. Создаем тестовую страницу.

Для Debian-1:

sudo mkdir -p /var/www/mysite

echo "<h1>Hello from Debian-1</h1>" | sudo tee /var/www/mysite/index.html

Для Debian-2:

sudo mkdir -p /var/www/mysite

echo "<h1>Hello from Debian-2</h1>" | sudo tee /var/www/mysite/index.html

3. Перезапускаем Apache (на обеих ВМ!!):

sudo systemctl restart apache2

4. Теперь проверки:

Можно проверить на обеих ВМ:

curl -I http://localhost:81

Для Debian-1:

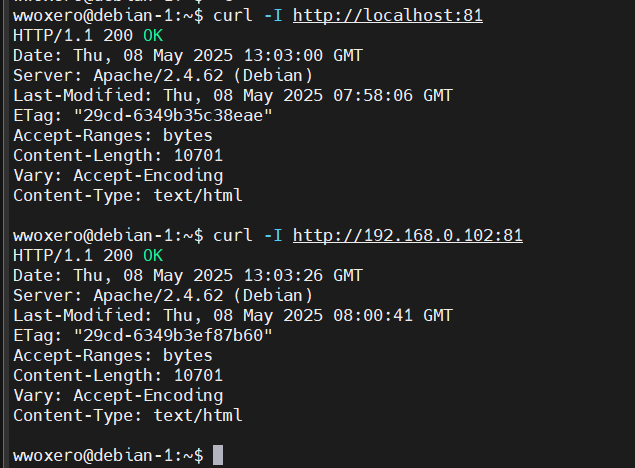
curl -I http://192.168.0.102:81

Для Debian-2:

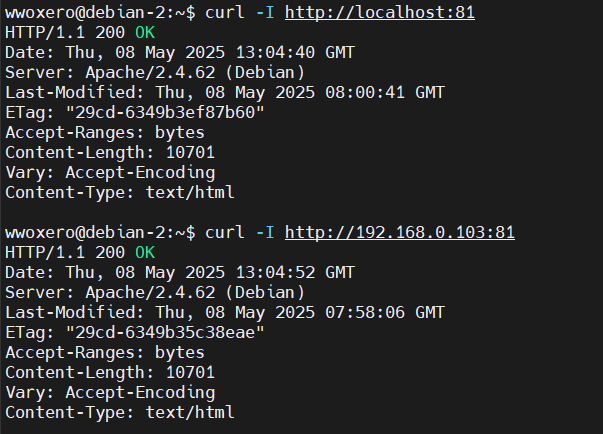
curl -I http://192.168.0.103:81

Результаты:

На Debian-1:



На Debian-2:



3. Настройка HAProxy

На обеих ВМ нужно сделать следующее:

1. Отредактируем конфигурацию:

sudo nano /etc/haproxy/haproxy.cfg

В конец всего нужно добавить:

frontend http\_front

bind 192.168.0.200:80

default\_backend http\_back

backend http\_back

balance roundrobin

server node1 192.168.0.103:81 check

server node2 192.168.0.102:81 check

3. Разрешаем привязку нелокального IP-адреса:

echo "net.ipv4.ip\_nonlocal\_bind=1" | sudo tee -a /etc/sysctl.conf

sudo sysctl –p

4. Проверяем конфигурацию вот этой командой:

sudo haproxy -c -f /etc/haproxy/haproxy.cfg

Если всё ок, то в выводе должно появиться “”.

5. Запускаем HAProxy:

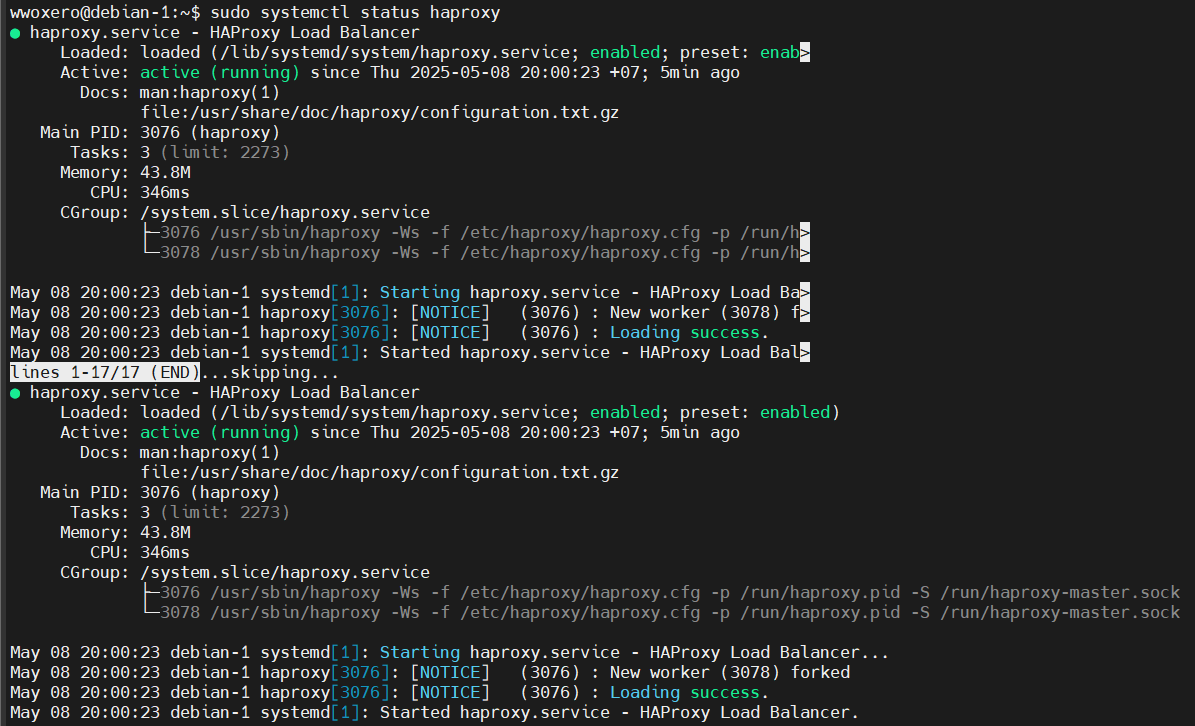
sudo systemctl restart haproxy

6. Проверка статуса:

sudo systemctl status haproxy

Результаты:

С Debian-1:



С Debian-2 аналогично. Как видно, статус Active (running), значит всё ок.

4. Настройка Pacemaker

Настраиваем только для Debian-1:

1. Сначала нужно установить crmsh:

sudo apt update

sudo apt install -y crmsh

2. Теперь переходим в режим конфигурации:

sudo crm configure

3. Теперь вводим команды последовательно (!!):

property no-quorum-policy=ignore

property stonith-enabled=false

primitive VirtualIP ocf:heartbeat:IPaddr2 params ip=192.168.0.200 cidr\_netmask=24 op monitor interval=10s

primitive HAProxy lsb:haproxy op monitor interval=10s

colocation HAProxy-with-VirtualIP inf: VirtualIP HAProxy

order VirtualIP-before-HAProxy inf: VirtualIP HAProxy

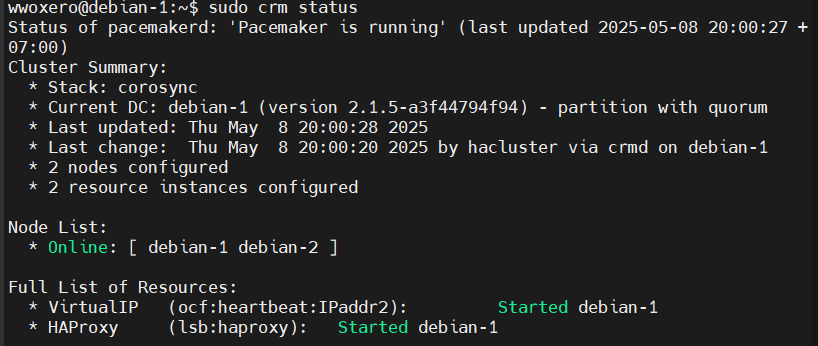
commit

exit

4. Проверка статуса:

sudo crm status

Результаты:



Задание 4.

Compose HAProxy.

Создание демонстрационной системы балансировки нагрузки с использованием Docker Compose, включающей HAProxy как балансировщик и несколько экземпляров nginx в качестве бэкенд-серверов. Система должна быть готова к проведению нагрузочного тестирования с помощью утилиты hping3.

1. Создание самой системы

1. Установка всего необходимого для задания:

sudo apt update

sudo apt install -y docker.io docker-compose

sudo systemctl enable --now docker

2. Создание структуры.

Создаем папку, где будем работать – haproxy-demo. Структура у нее будет вот такая:

1. haproxy-demo

1.1. docker-compose.yml

1.2. haproxy.cfg

1.3. nginx1

1.3.1. html

1.3.1.1. index.html

1.4. nginx2

1.4.1. html

1.4.1.1. index.html

Вот как выглядит:



3. Создание файлов:

1) docker-compose.yml:

version: '3.8'

services:

haproxy:

image: haproxy:latest

container\_name: haproxy

ports:

- "8080:80"

volumes:

- ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg

networks:

- haproxy-network

nginx1:

image: nginx:latest

container\_name: nginx1

volumes:

- ./nginx1/html:/usr/share/nginx/html

networks:

- haproxy-network

nginx2:

image: nginx:latest

container\_name: nginx2

volumes:

- ./nginx2/html:/usr/share/nginx/html

networks:

- haproxy-network

networks:

haproxy-network:

driver: bridge

2) haproxy.cfg:

global

log /dev/log local0

log /dev/log local1 notice

daemon

defaults

log global

mode http

timeout connect 5000

timeout client 50000

timeout server 50000

frontend http\_front

bind \*:80

default\_backend http\_back

backend http\_back

balance roundrobin

server nginx1 nginx1:80 check

server nginx2 nginx2:80 check

3) index.html для nginx1:

<html>

<body>

<h1>Hello from Nginx 1</h1>

</body>

</html>

4) index.html для nginx2:

<html>

<body>

<h1>Hello from Nginx 2</h1>

</body>

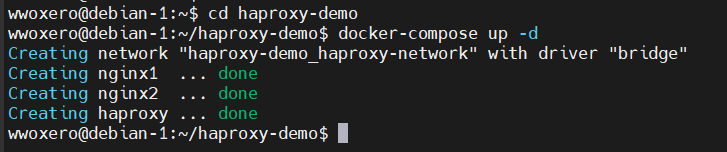
</html>

4. Теперь можно запустить созданную систему.

Запускаем контейнеры (из папки haproxy-demo !!!):

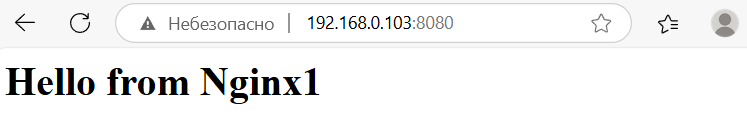
docker-compose up –d

Вывод:

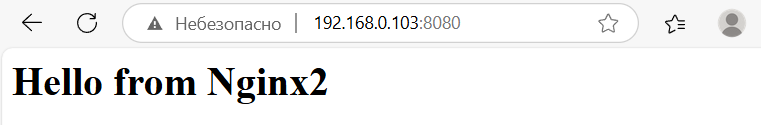


Для проверки откроем в браузере http://192.168.0.103:8080.

Вот как выглядит первый заход на страницу:



Перезагрузка страницы:

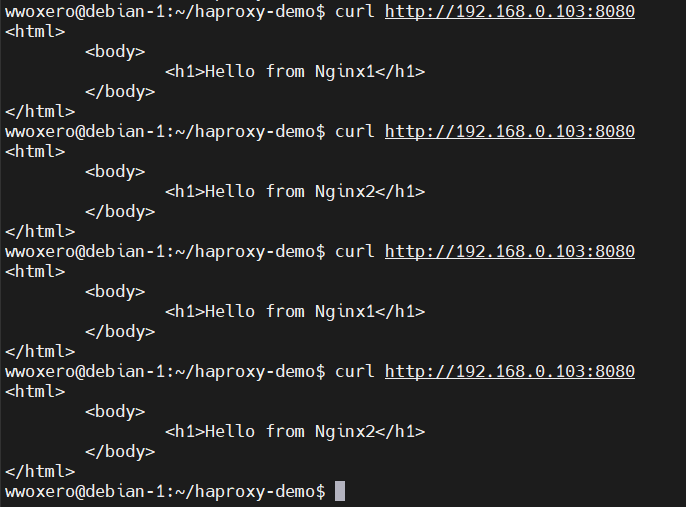


При повторных перезагрузках надписи чередуются.

Еще можно проверить с помощью команды:

curl http://192.168.0.103:8080

Вывод вот такой:



Надписи точно так же чередуются.

2. Нагрузочное тестирование

Не получилось ☹☹☹