1. **Цель работы**
   1. Развернуть приложение в отказоустойчивой масштабируемой облачной-инфраструктуре на платформе OpenNebula.
2. **Выполнение работы** 
   1. Подключимся по ssh к хосту OpenNebula с учетными данными devops : devops (логин : пароль) и сразу же посмотрим логин и пароль для авторизации в OpenNebula командой «cat /var/lib/one/.one/one\_auth» предварительно получив права суперпользователя «su -» (рисунок 1).

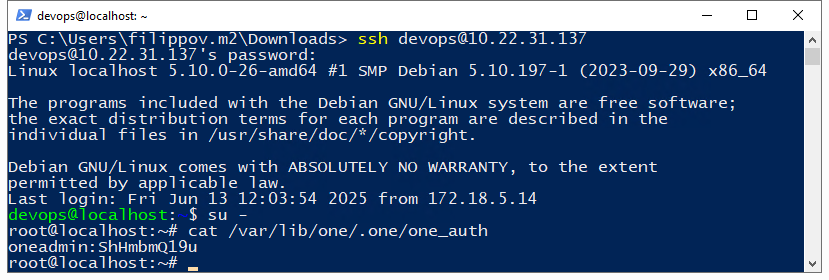


Рисунок 1 – Получение учетных данных для OpenNebula по ssh

* 1. На рисунках 2-3 изображен графический интерфейс OpenNebula.



Рисунок 2 – Графический интерфейс OpenNebula (авторизация)

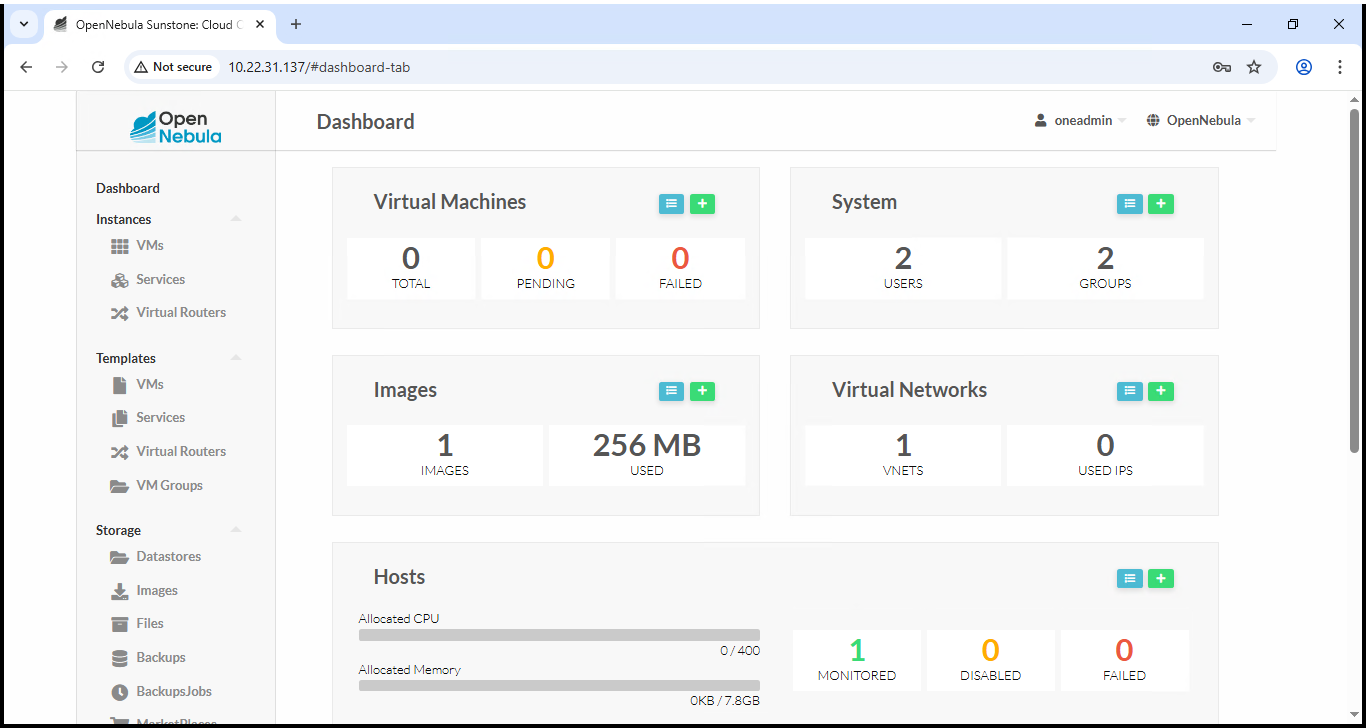


Рисунок 3 – Графический интерфейс OpenNebula (главная страница)

* 1. Перейдём к Storage – Images (рисунок 4). В системе уже есть образ, который можно использовать для создания виртуальных машин Alpine Linux. Создадим новый образ, для этого выберем + Create image и заполним поля (рисунок 5).

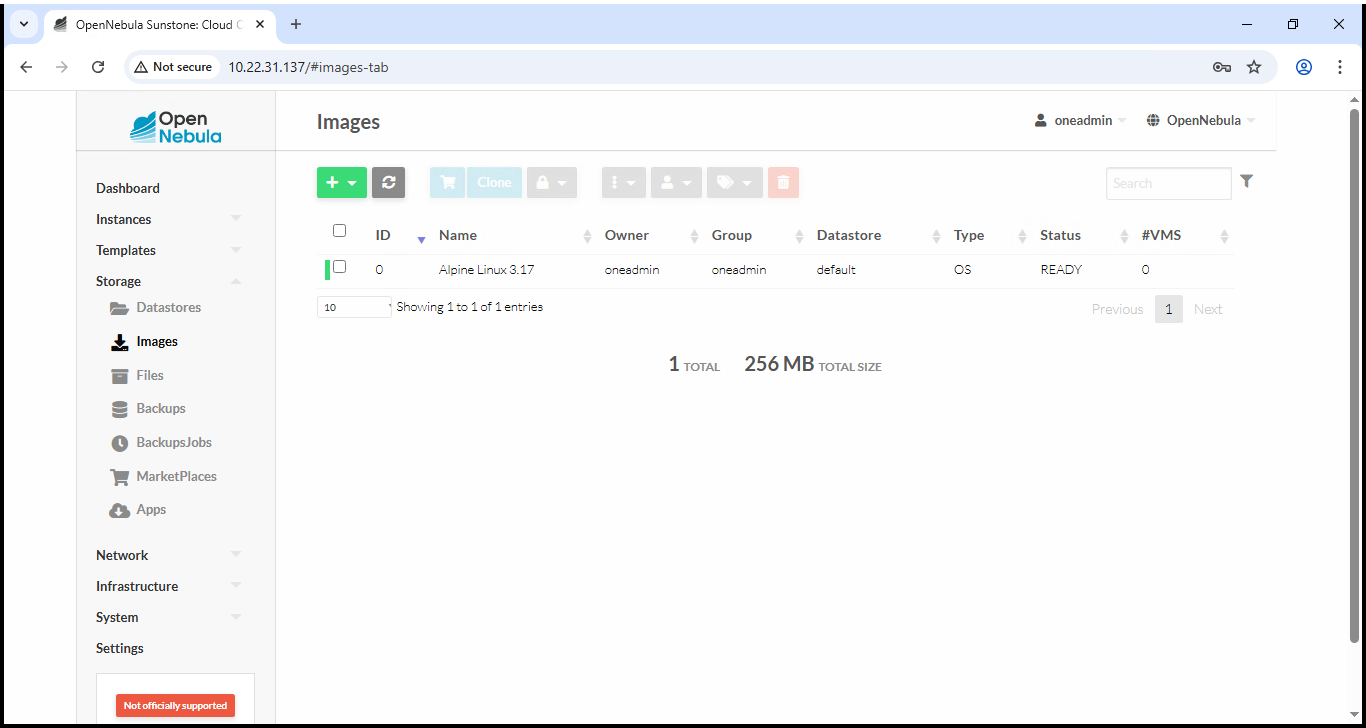


Рисунок 4 – Раздел образов в хранилище

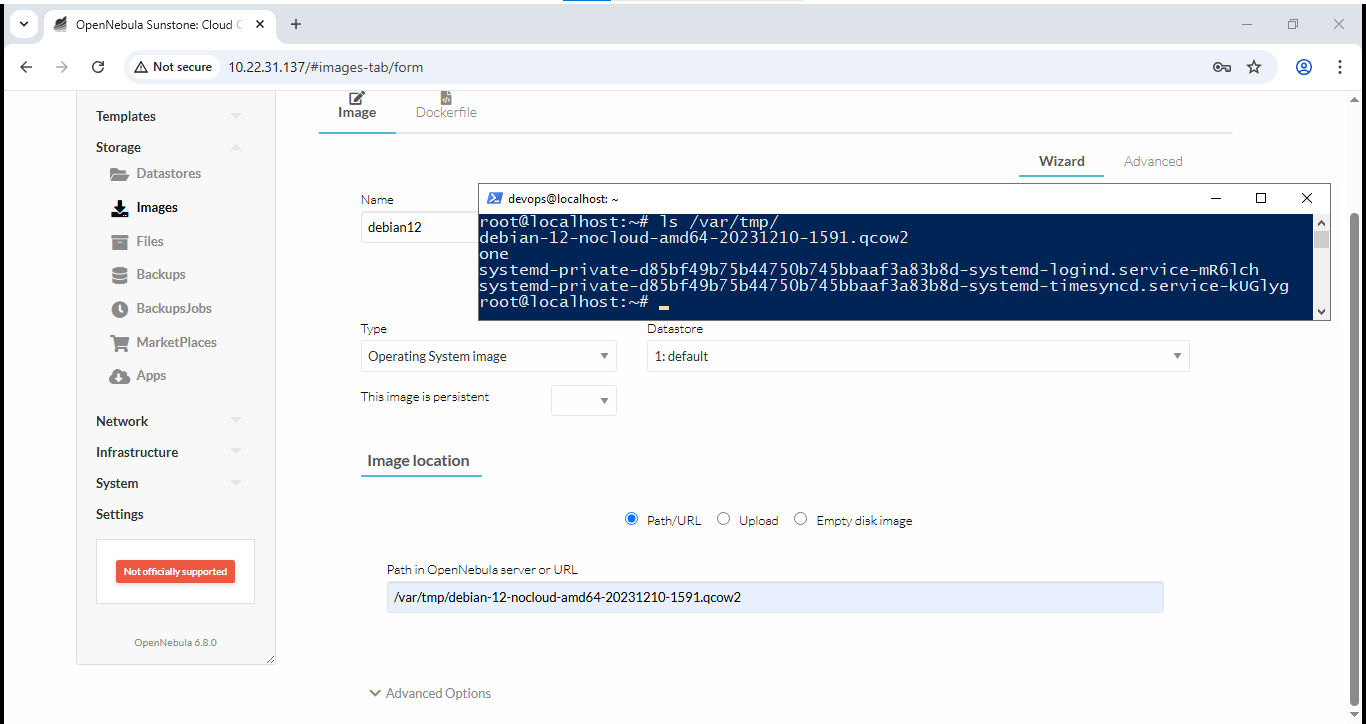


Рисунок 5 – Создание нового образа

* 1. Теперь этот образ можно использовать для создания будущих виртуальных машин. Перейдём в раздел Templates – VM Templates (Шаблоны – Шаблоны виртуальных машин) и в списке уже будет шаблон для Alpine Linux (рисунок 6).

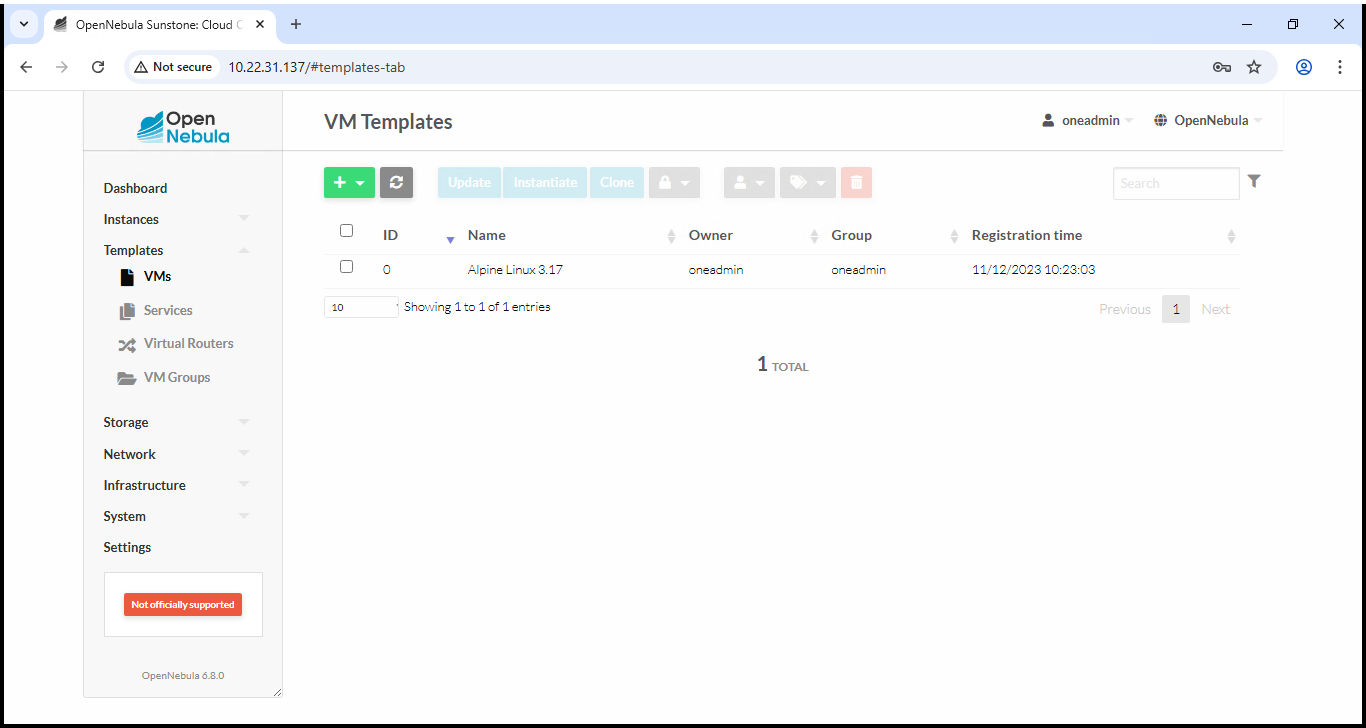


Рисунок 6 – Шаблоны виртуальных машин

* 1. Воспользуемся + для создания нового шаблона виртуальной машины и заполним необходимые поля, как показано на рисунках 7-9. Выделим 1Gb памяти и 1CPU. В качестве гипервизора выбираем KVM или QEMU.

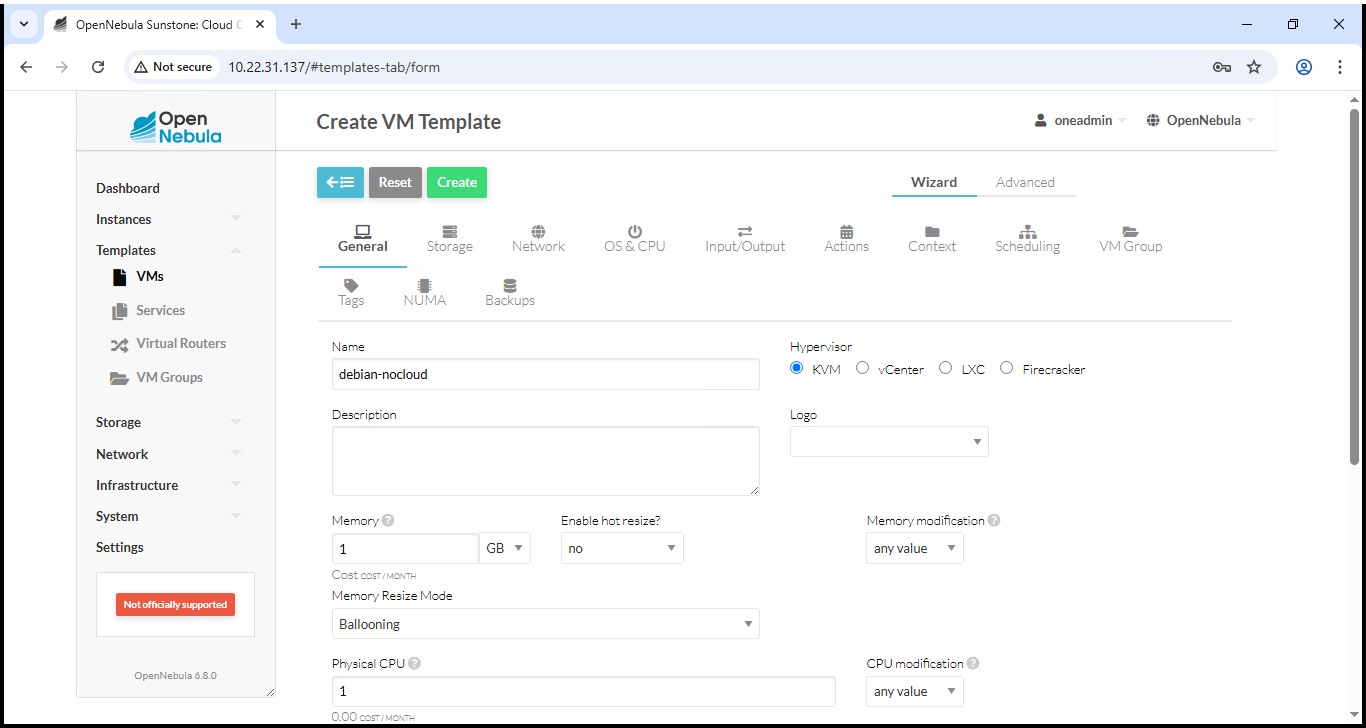


Рисунок 7 – Создание нового шаблона виртуальной машины

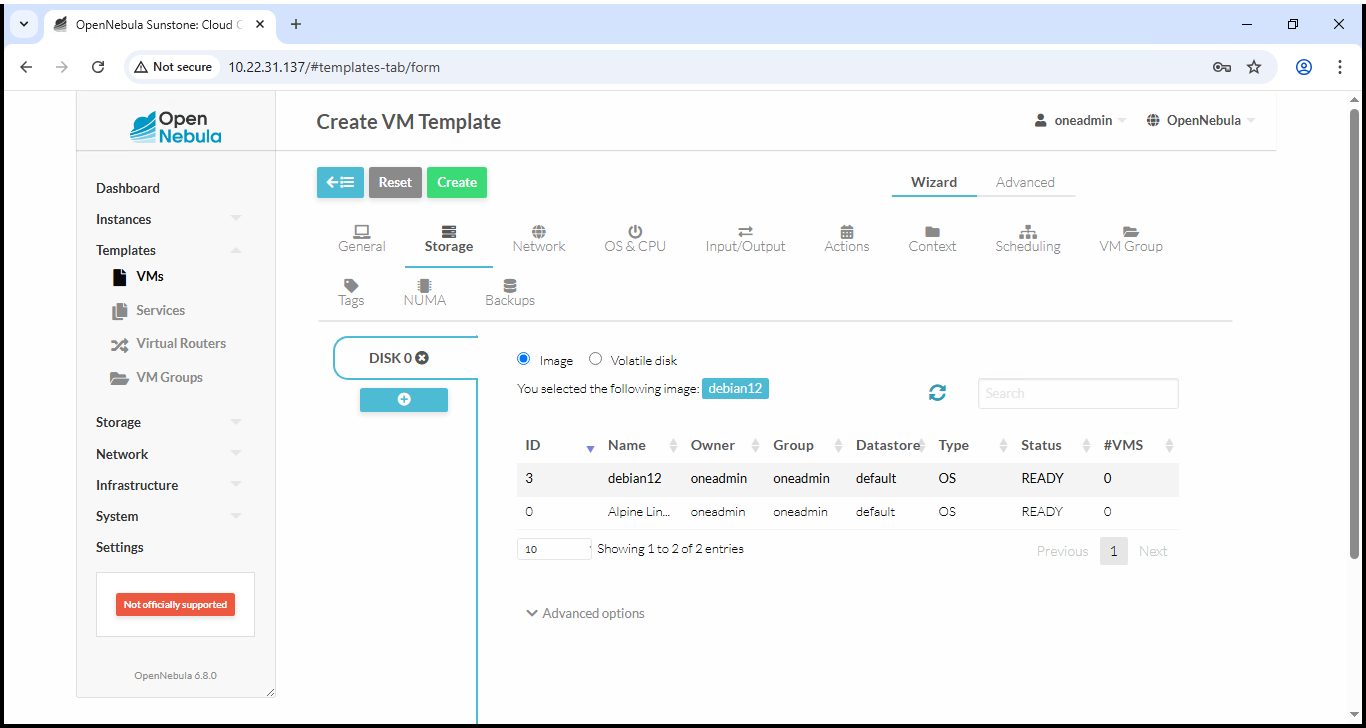


Рисунок 8 – Выбор образа debian12

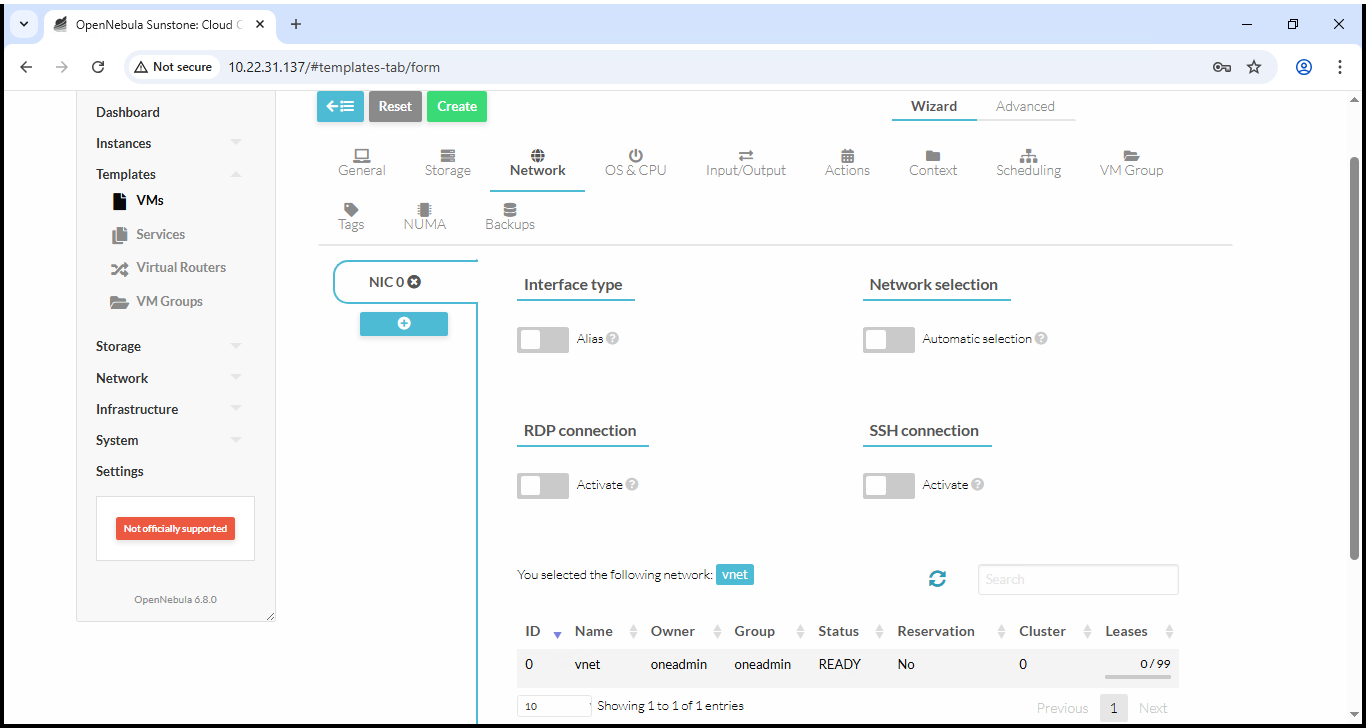


Рисунок 9 – Выбор сети vnet

* 1. Для создания виртуальной машины на основе шаблона необходимо перейти в раздел Instances (Экземпляры) и воспользоваться + (рисунок 8), выбрав ранее созданный шаблон debian-nocloud (рисунок 10).

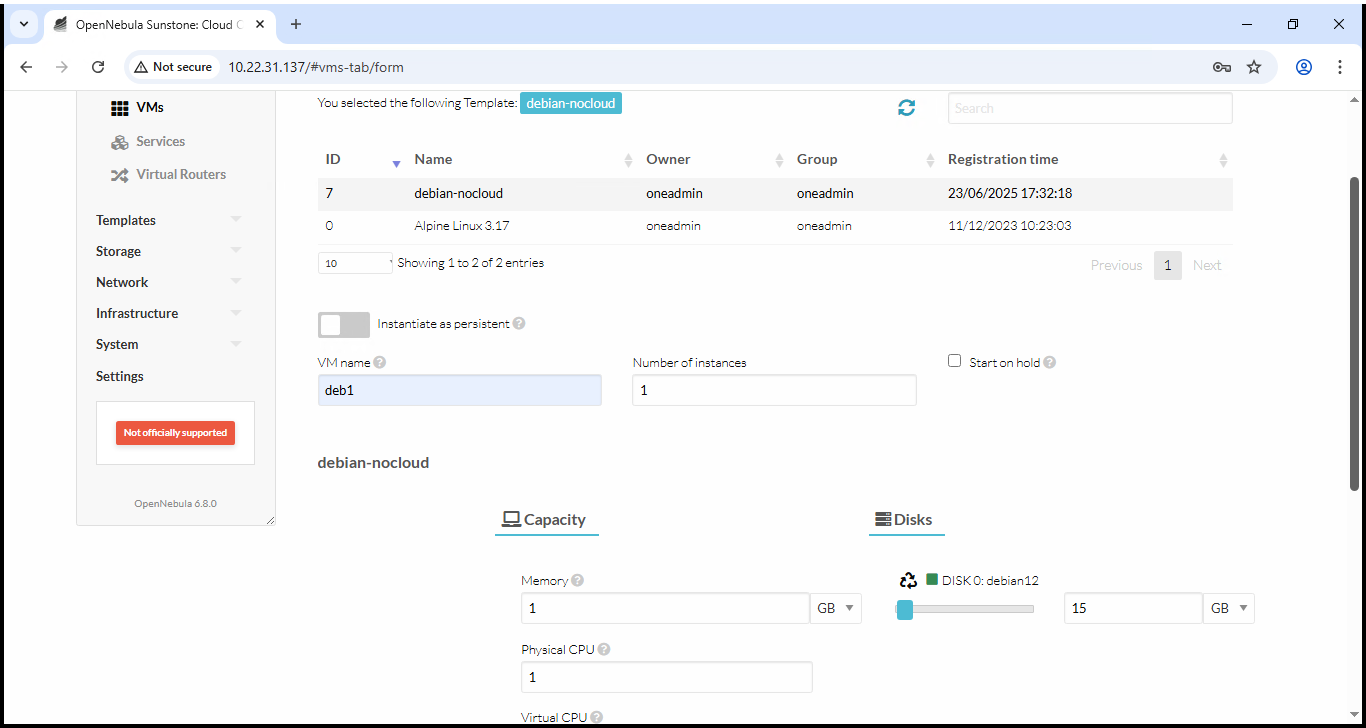


Рисунок 10 – Создание экземпляра виртуальной машины

* 1. При нажатии на Create будет создан экземпляр deb1 (рисунок 11).

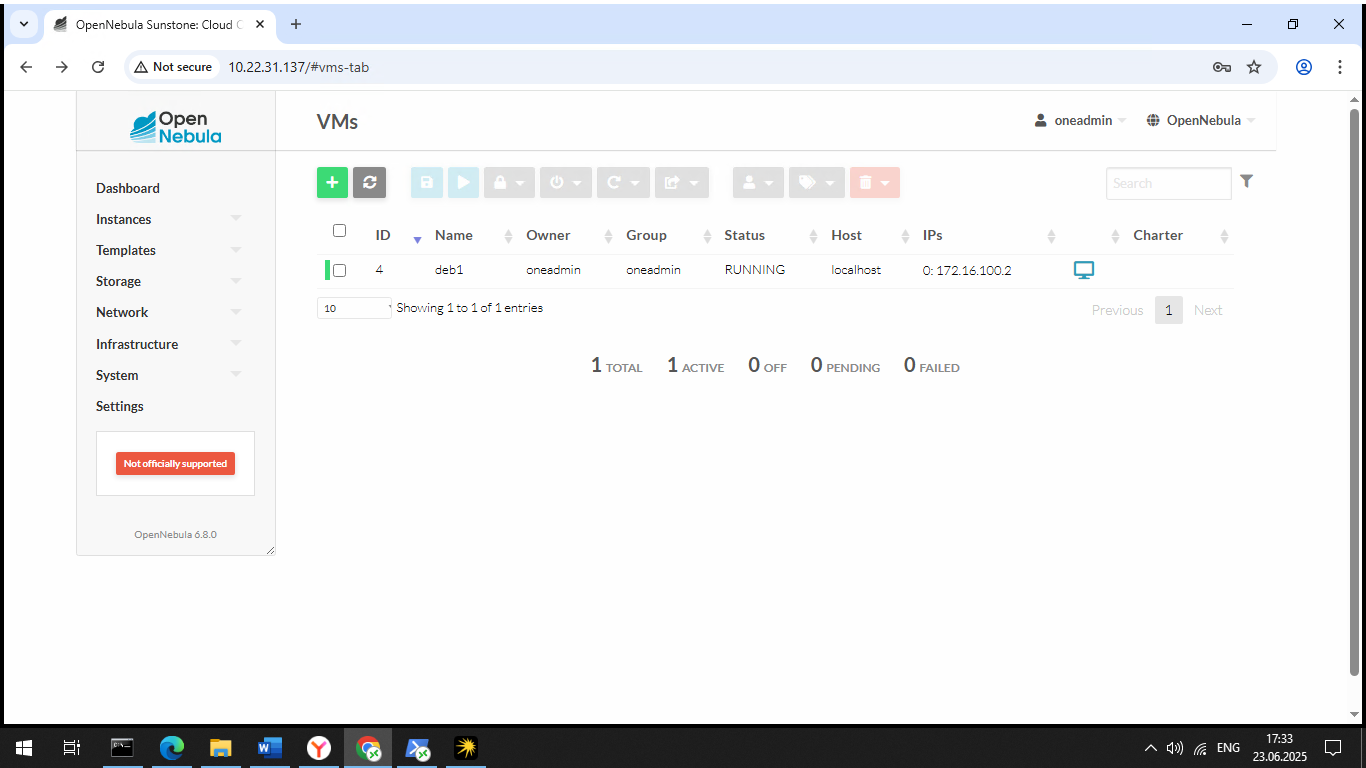


Рисунок 11 – Экземпляр deb1

* 1. Откроем созданную виртуальную машину с использованием VNC, lоступ к системе возможен под пользователем root без пароля. Проверив сетевые настройки, видно что у deb1 нет адреса 172.16.100.2, поэтому эта виртуальная машина недоступна (рисунок 12).

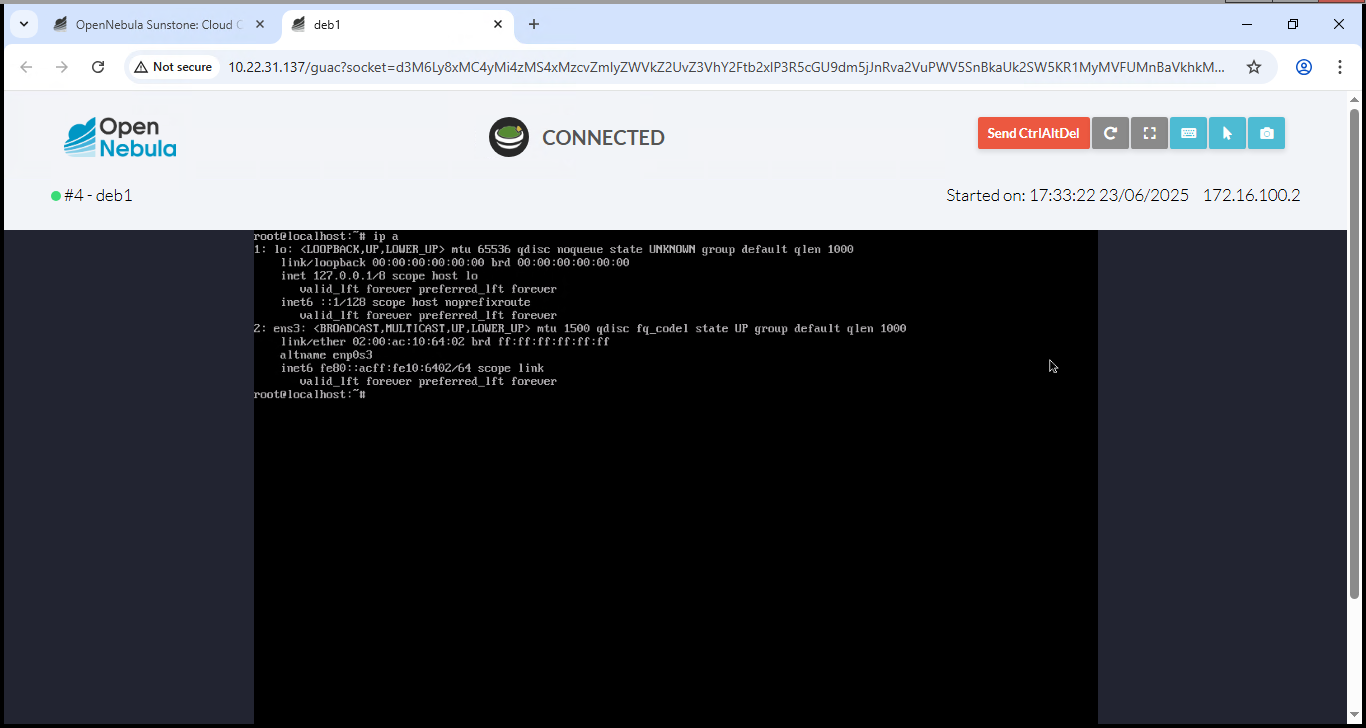


Рисунок 12 – Консоль deb1 в vnc

* 1. Сначала удалим на виртуальной машине файл, используемый по умолчанию для сетевых настроек командой «rm /run/systemd/network/10-netplan-all-en.network», а также удалим Netplan, чтобы при перезапуске не сбрасывались настройки сети командой «apt purge netplan.io -y» и создадим файл «/etc/systemd/network/wire.network» с необходимыми данными (рисунок 13).

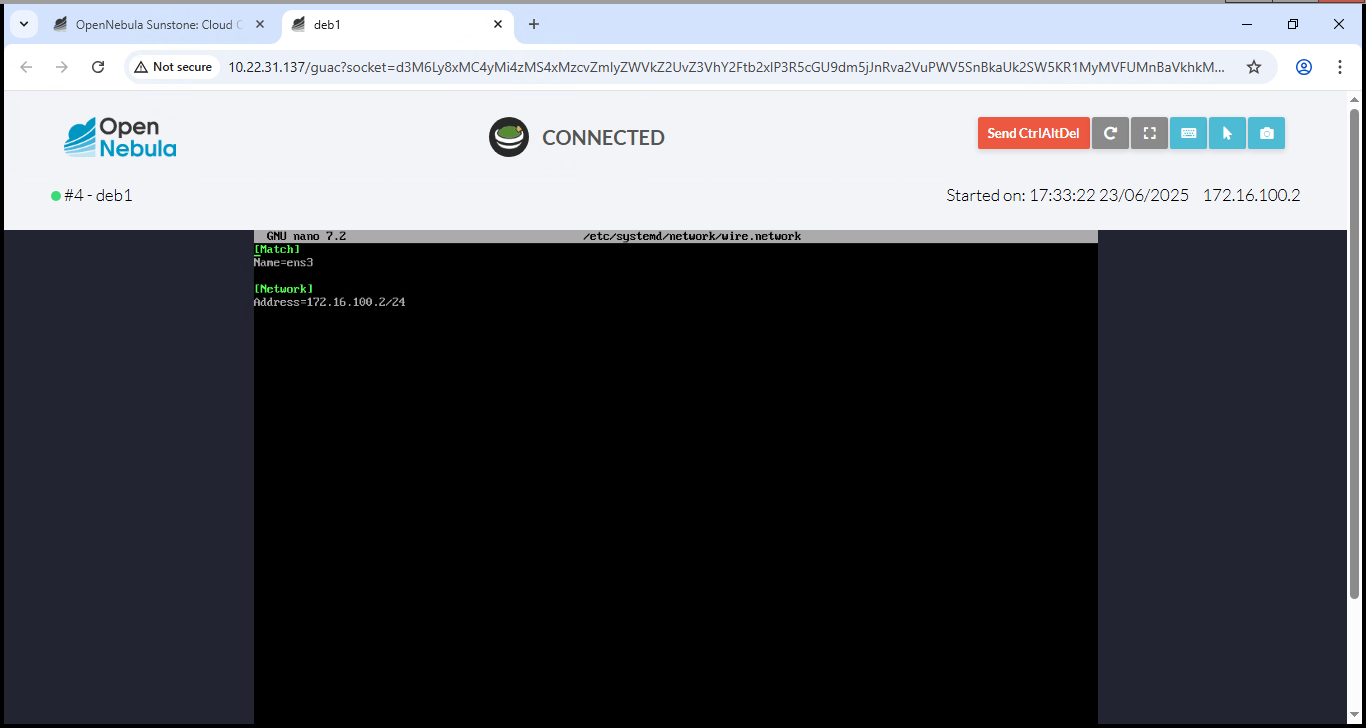


Рисунок 13 – Задание сетевых настроек

* 1. Перезапустим сервис, отвечающий за сеть и проверим появился ли адрес (рисунок 14).

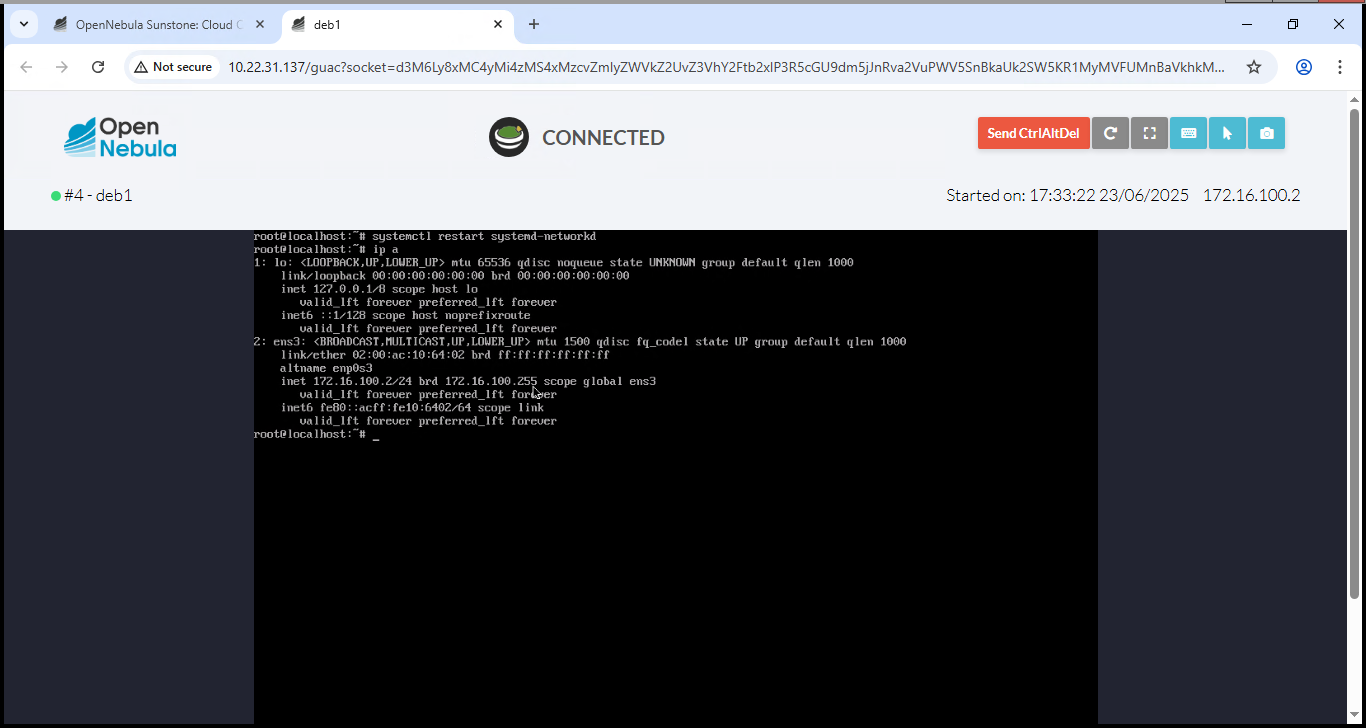


Рисунок 14 – Перезапуск сервиса

* 1. Как видно из рисунка на виртуальной машине есть адрес и она доступна с другого хоста (рисунок 15).

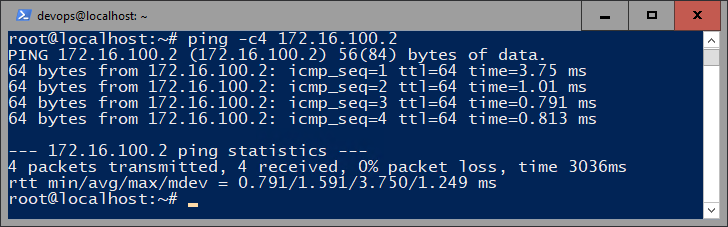


Рисунок 15 – Проверка доступности виртуальной машины

* 1. Виртуальная машина deb1 была подключена к предустановленной виртуальной сети vnet c адресом 172.16.100.0/24. Создадим ещё одну сеть, подключим к ней другой экземпляр виртуальной машины и обеспечим взаимодействие между ними. Для создания новой виртуальной сети перейдём к разделу Networks – Virtual networks (Сети – Виртуальные сети). Для добавления новой сети воспользуемся + и заполним необходимые поля как показано на рисунках 16-18 и нажмём кнопку Create.

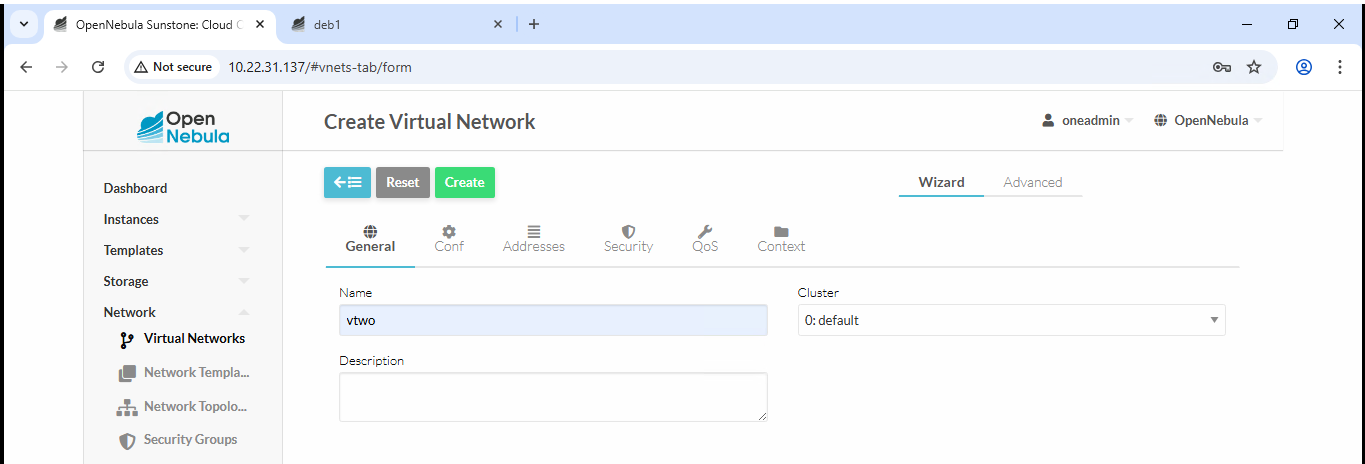


Рисунок 16 – Указание имени сети

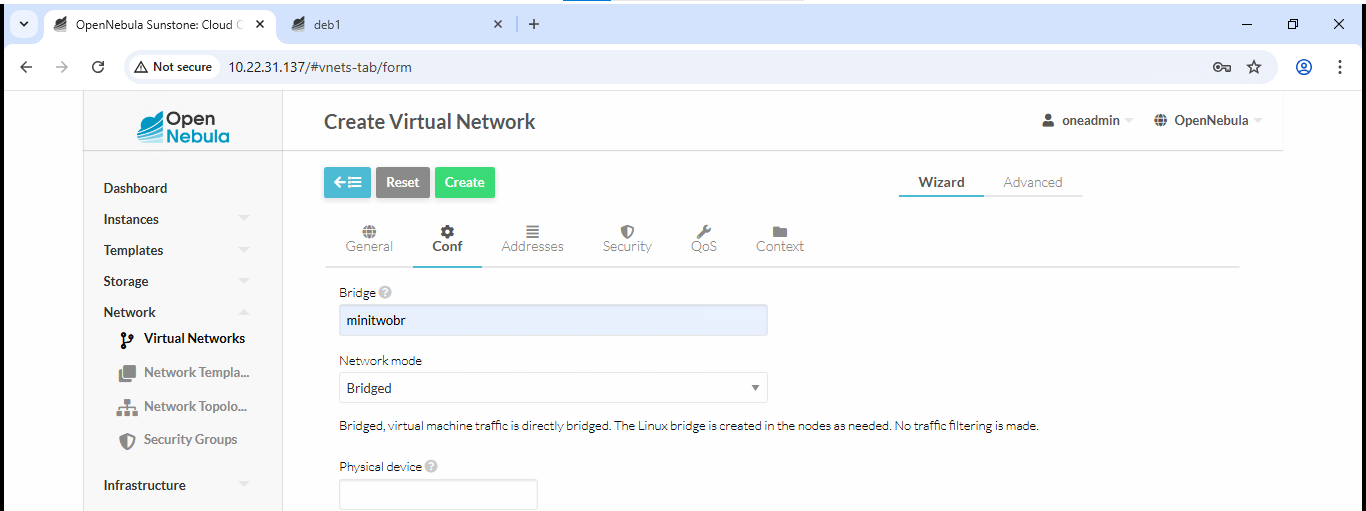


Рисунок 17 – Название физического моста и сетевой режим

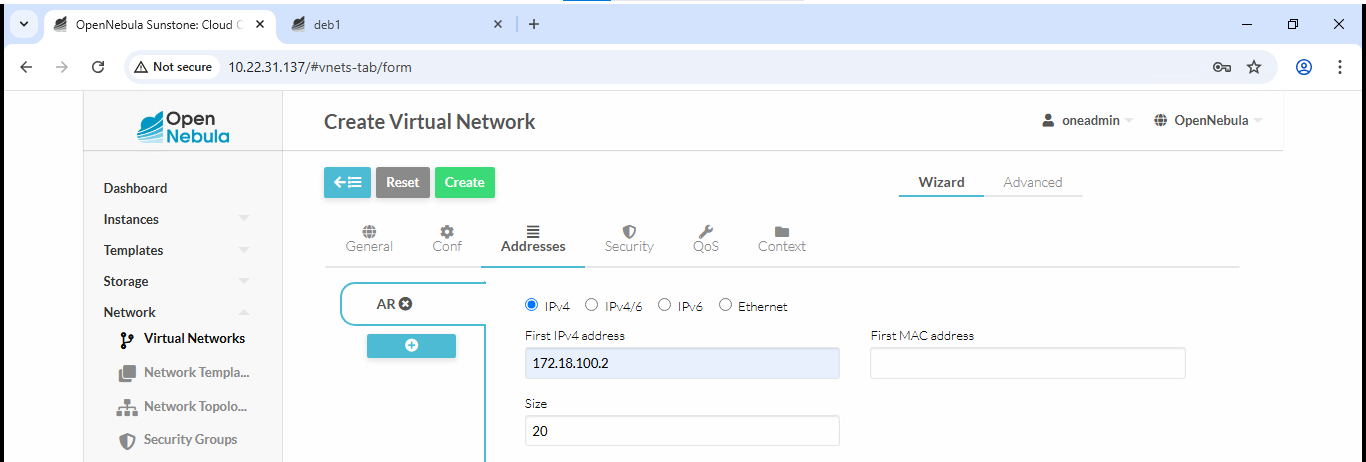


Рисунок 18 – Создание адреса сети

* 1. При создании виртуальной сети был указан мост minitwobr и его необходимо создать. Создадим его на основе существующего моста, скопировав (рисунок 19) и отредактировав настройки (рисунок 20).

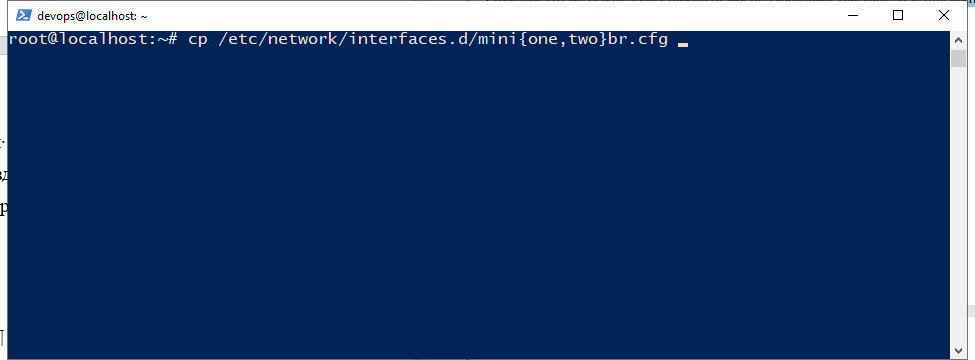


Рисунок 19 – Подготовка файла конфигурации

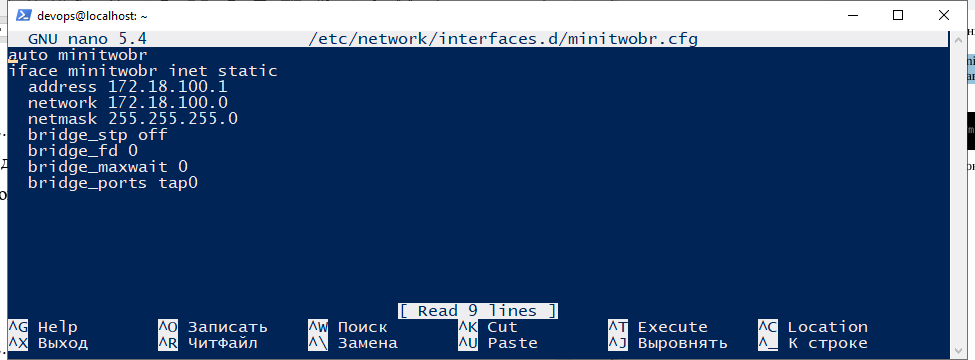


Рисунок 20 – Изменение конфигурации minitwobr

* 1. Перезапустим сервис командой «systemctl restart networking» и проверим получение адреса созданным мостом minitwobr командой «ip a» (рисунок 21).

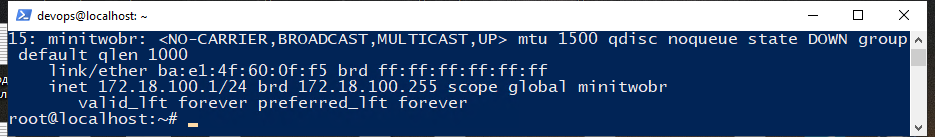


Рисунок 21 – Список сетевых устройств

* 1. Создадим ещё один экземпляр виртуальной машины debian-nocloud, но в этот раз подключим к сети vtwo (рисунок 22).

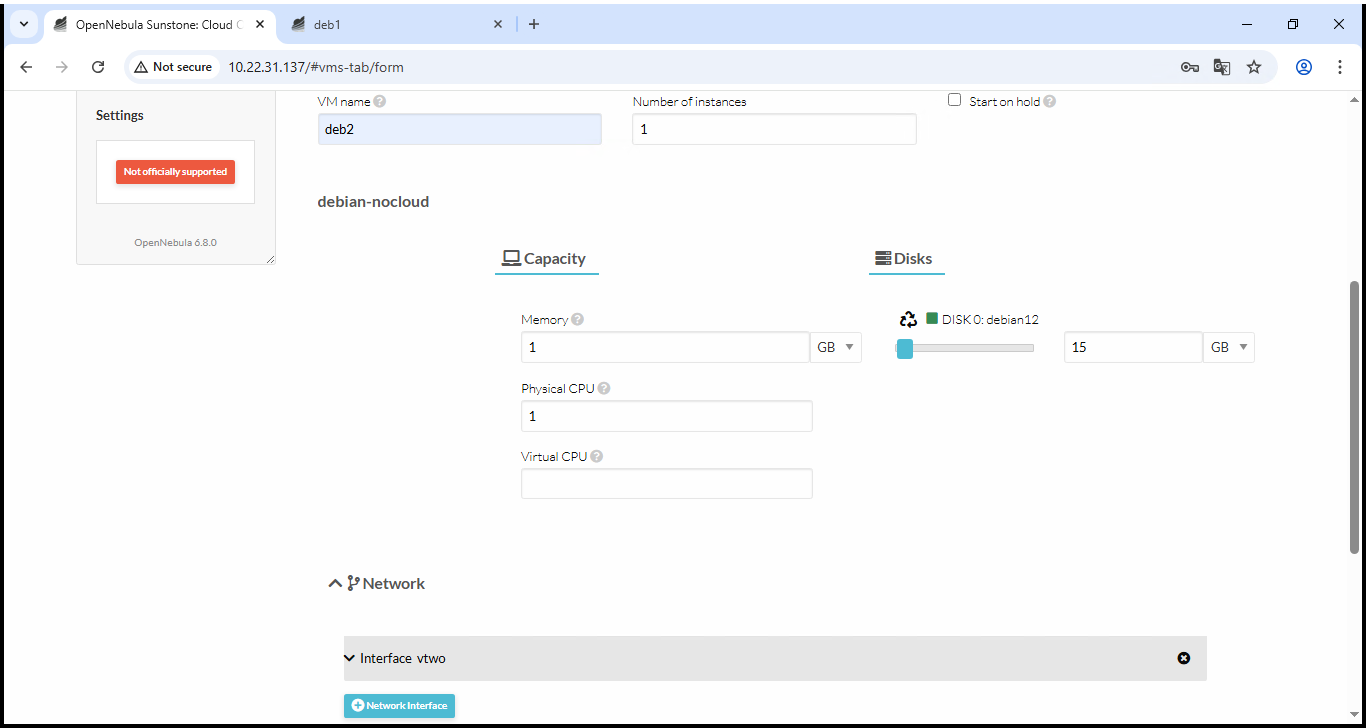


Рисунок 22 – Выбор подключения к виртуальной сети

* 1. Теперь есть две виртуальные машины в разных сетях (рисунок 23).

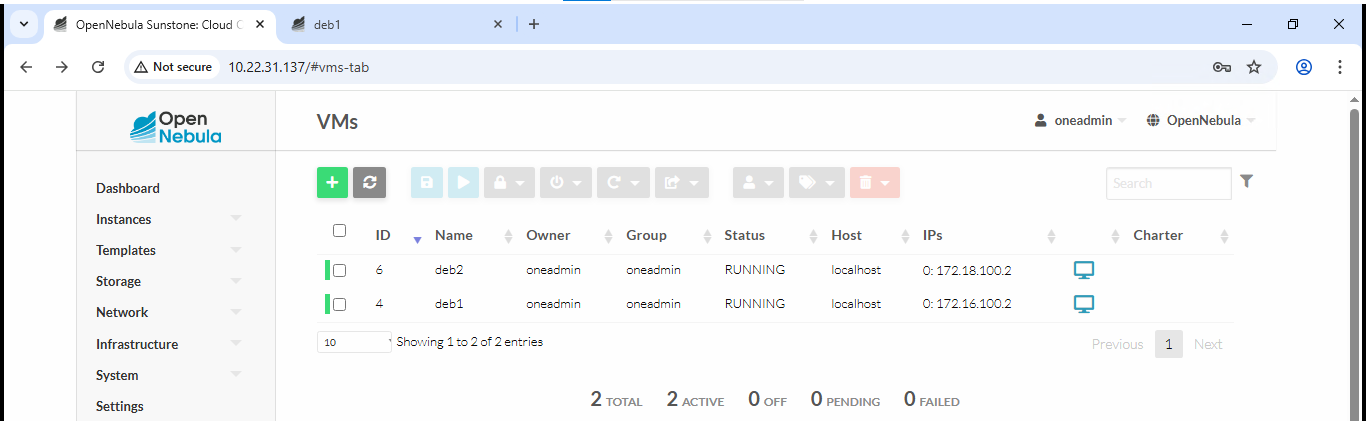


Рисунок 23 – Список виртуальных машин

* 1. На рисунке 24 изображена настройка сети аналогично deb1 (см. пункт 2.9).

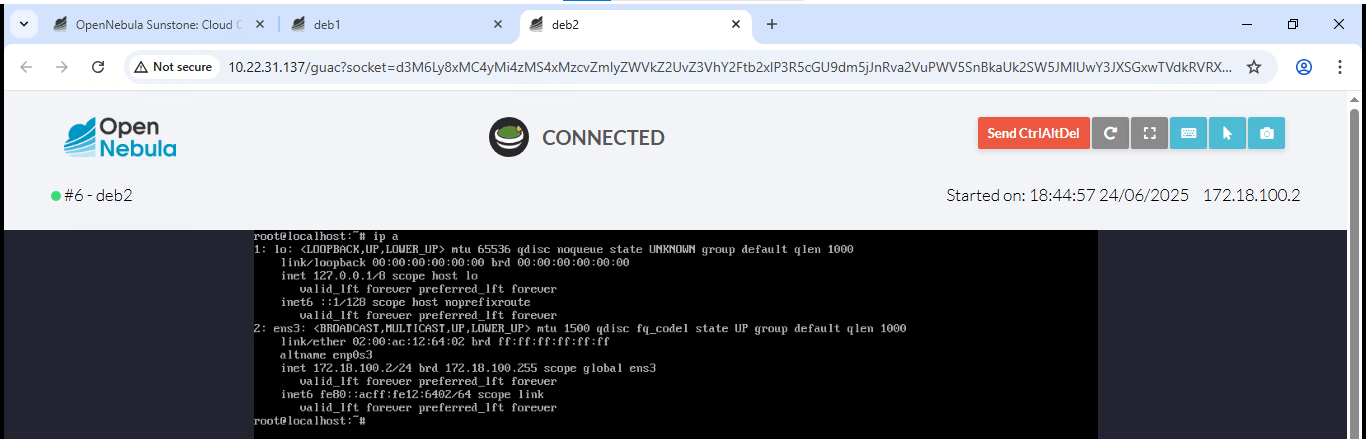


Рисунок 24 – Задание сетевых настроек

* 1. Создадим deb3 для того чтобы зарезервировать ip-адрес, для будущего автомасштабирования веб-приложения, а также создадим deb4 и настроим статическую адресацию для будущего PostgreSQL сервера, удалим deb3. Поскольку deb1 и deb2 находятся в разных сетях, чтобы они могли между собой взаимодействовать нужен маршрутизатор. В составе opennebula есть магазин приложений Storage – Apps, откуда можно установить необходимый сервис. Выберем Service Virtual Router и кнопку Import to datastore (рисунок 25).

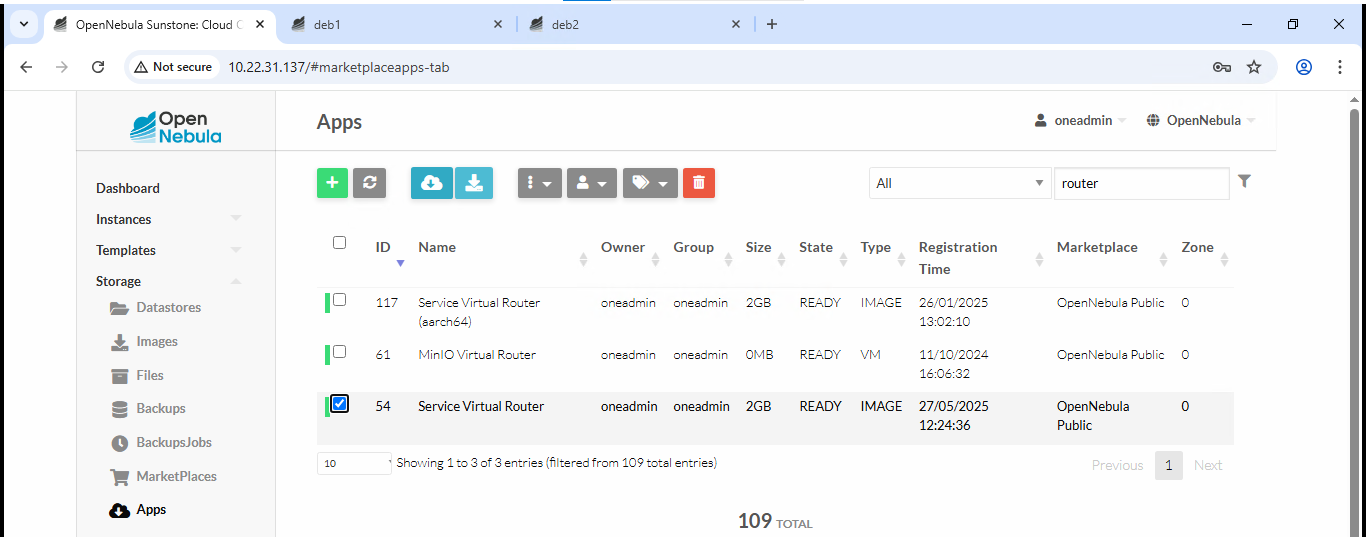


Рисунок 25 – Магазин приложений

* 1. Выберем хранилище по умолчанию и нажмем Download (рисунок 26).

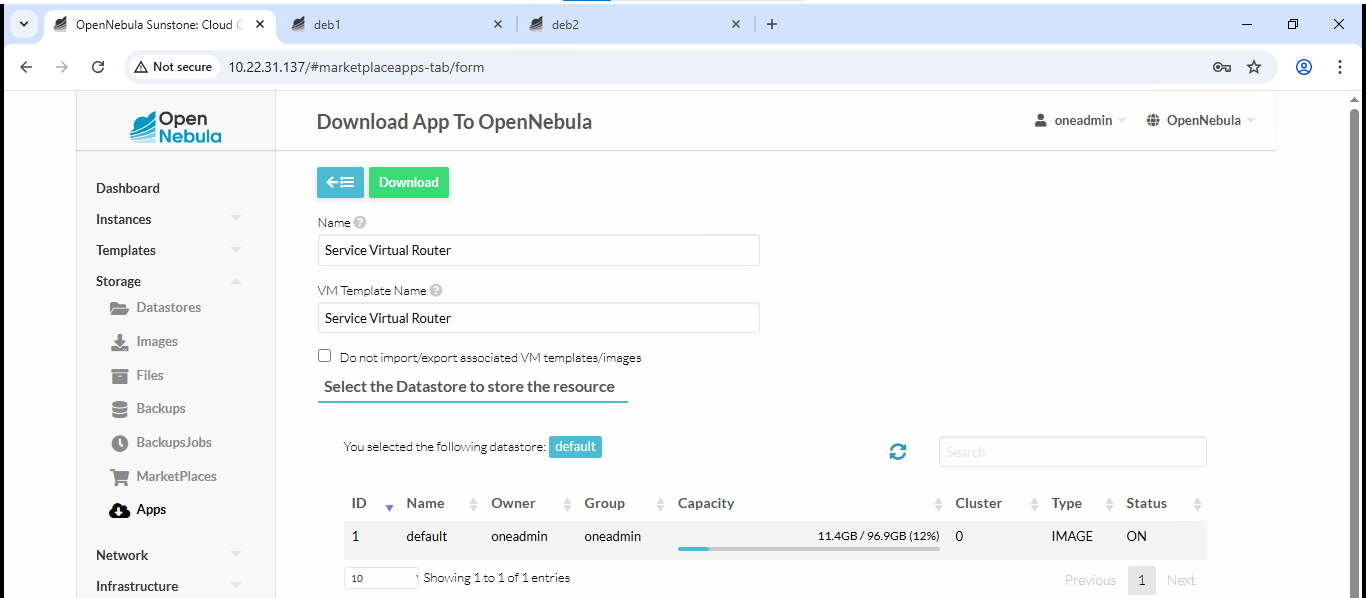


Рисунок 26 – Скачивание приложение в хранилище

* 1. Создадим экземпляр виртуального маршрутизатора, для этого воспользуемся + в пункте Instances – Virtual Routers и выберем шаблон Service Virtual Router. Далее укажем имя роутера и подключим маршрутизатор к двум сетям. Или если создание шаблона заканчивается неудачей, то магазин приложений блокируется сетью университета или провайдера, в этом случае создадим обычную виртуальную машину на основе debian-nocloud и настроим её в качестве маршрутизатора (рисунок 27).

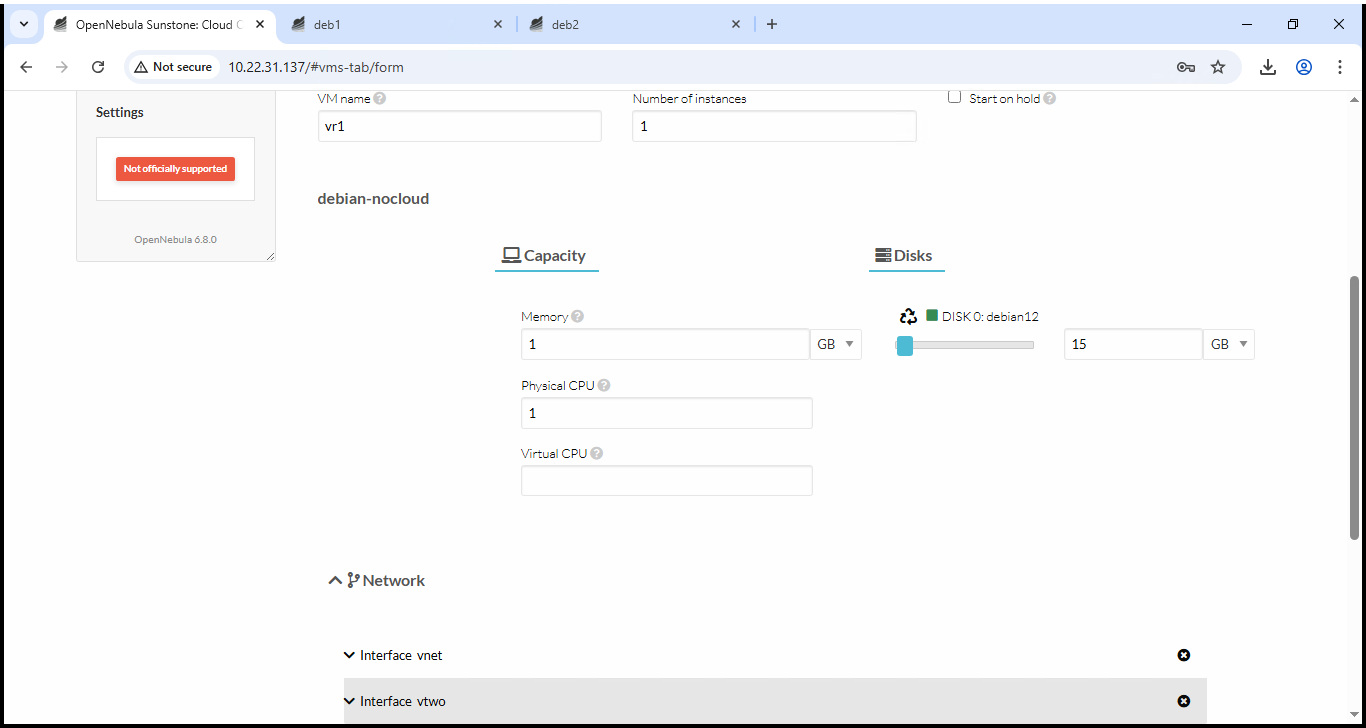


Рисунок 27 – Создание экземпляра виртуального маршрутизатора

* 1. Теперь виртуальная машина успешна запущена (рисунок 28).

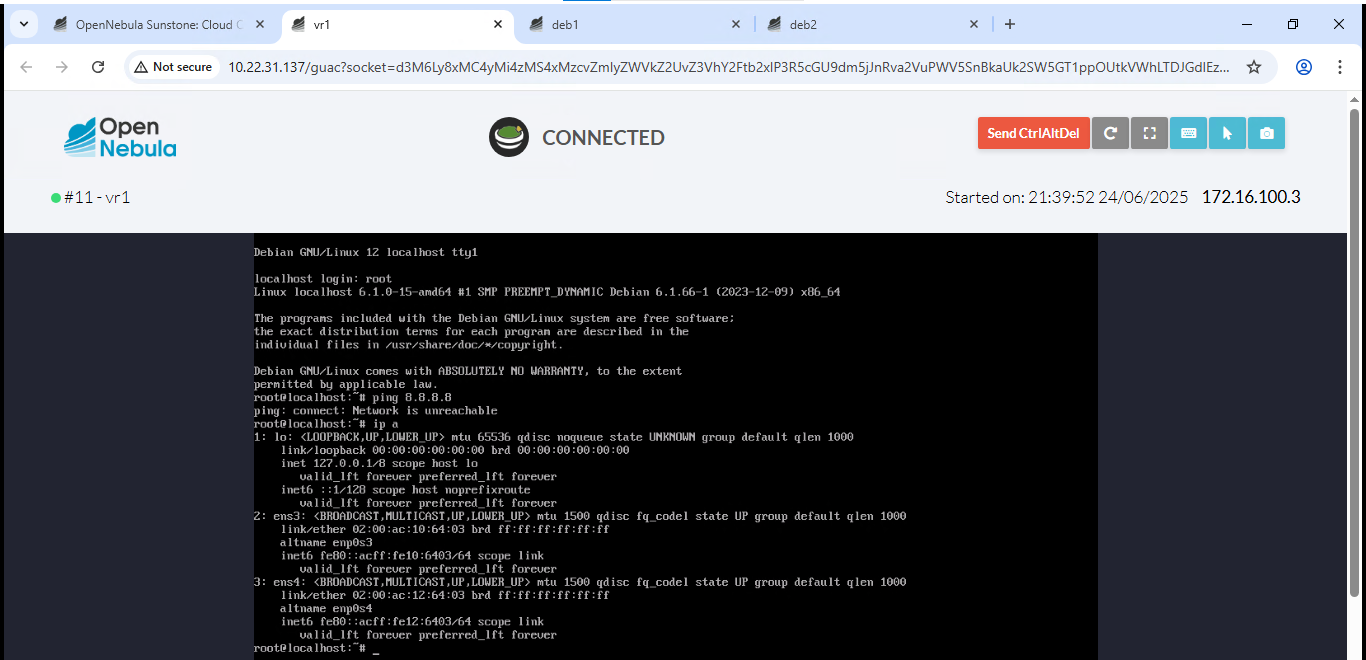


Рисунок 28 – Успешный запуск маршрутизатора

* 1. Как видно из рисунка 28 нет адресов на сетевых интерфейсах. Назначим адреса на всех виртуальных машинах согласно рисунка 29, как это было продемонстрировано ранее. У маршрутизатора две сетевых карты, у обычных машин по одной.



Рисунок 29 – Сетевые адреса на маршрутизаторе

* 1. Для виртуальных машин deb1, deb2, deb4 необходимо добавить шлюз, используя файл .network. Для deb1 это будет выглядеть, как показано на рисунке 30.



Рисунок 30 – Сетевые настройки на deb1

* 1. Осталось включить форвардинг пакетов на маршрутизаторе (рисунок 31) и виртуальные машины deb1, deb2 и deb4 смогут взаимодействовать друг с другом. Если форвардинг слетает после перезагрузки лучше раскомментировать строку «net.ipv4.ip\_forward = 1» в etc/sysctl.conf и применить изменения командой «sysctl -p».

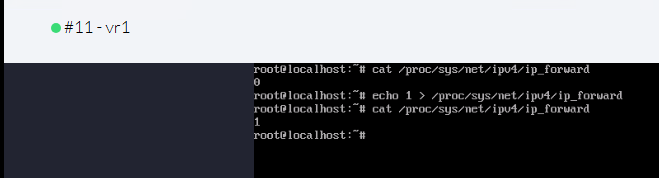


Рисунок 31 – Включение форвардинга на маршрутизаторе

* 1. Как видно из рисунка 32, пакет проходит через маршрутизатор и поступает на целевую виртуальную машину.

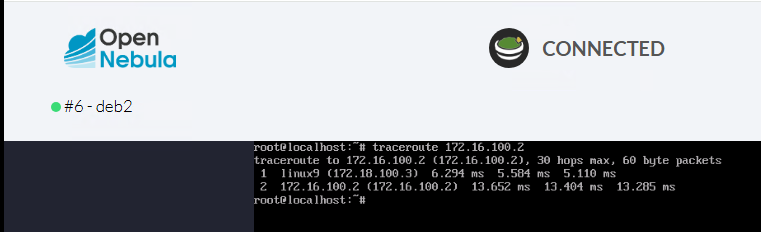


Рисунок 32 – Трассировка

* 1. Для стабильной работы перенастроим deb1 и vr1 на сетевой адаптер vthree вместо встроенного vnet, предварительно создав и настроив виртуальную сеть vthree и сетевой мост minithreebr, аналогично vtwo и minitwobr, как это было показано ранее, а также настроим выход в интернет, изменив конфигурационные файлы (рисунки 33-34) и перезапустив systemd-networkd.

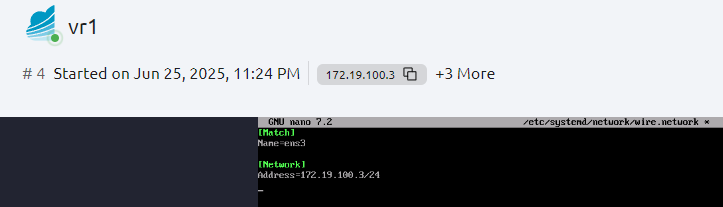


Рисунок 33 – Настройки на ens3

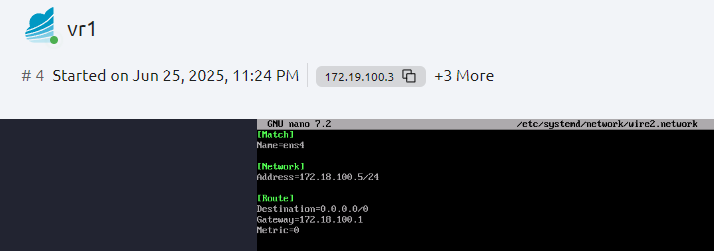


Рисунок 34 – Указание маршрута по умолчанию на ens4

* 1. Осталось включить маскарадинг на хосте OpenNebula, сохранить правила и восстанавливать их при перезагрузке через crontab (рисунки 35-37).

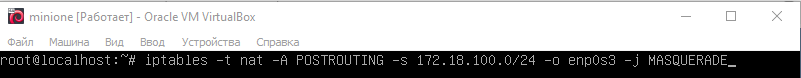


Рисунок 35 – Включение маскарадинга на хосте OpenNebula

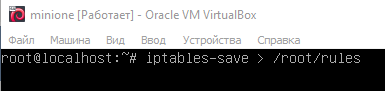


Рисунок 36 – Сохранение правил iptables

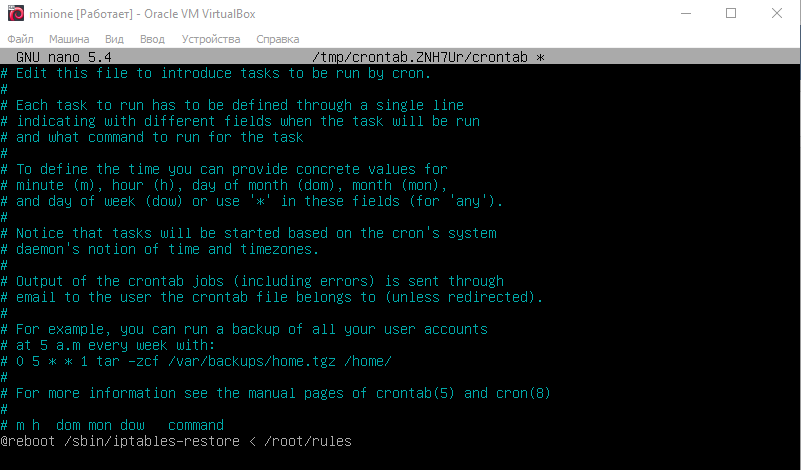


Рисунок 37 – Восстановление правил при перезагрузке через crontab

* 1. На хостах приватной сети появился интернет (рисунок 38).

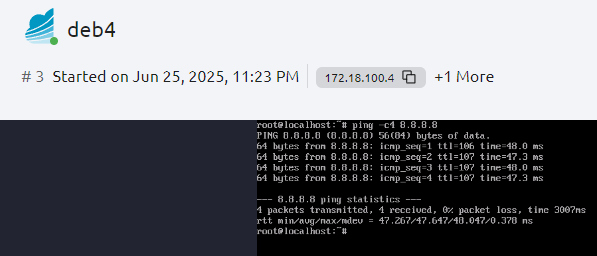


Рисунок 38 – Проверка интернета

* 1. Включим маскарадинг на vr1 для доступа в Интернет из публичной и приватной сети (рисунок 39).

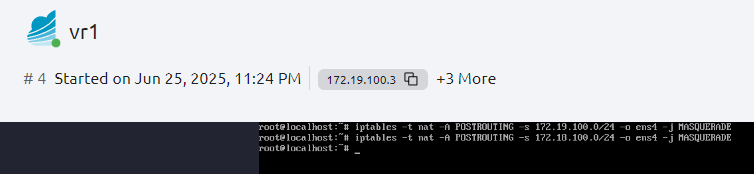


Рисунок 39 – Включение маскарадинга на vr1

* 1. Сохраним правила (рисунок 40) аналогично тому как это было показано ранее для хоста OpenNebula, предварительно добавив «DNS=8.8.8.8» в настройки .network и установив crontab командой «apt update && apt install cron».

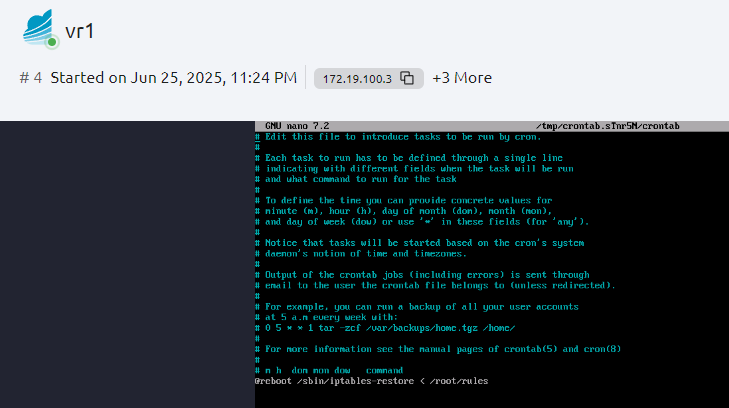


Рисунок 40 – Установка crontab

* 1. Теперь и на хостах публичной сети появился интернет (рисунок 41).

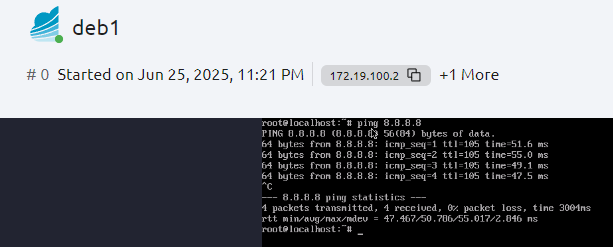


Рисунок 41 – Проверка интернета

* 1. Развернем СУБД PostgreSQL на deb4. Для этого установим пакеты postgresql и postgresql-contrib, предварительно добавив «DNS=8.8.8.8» в настройки .network и перезапустив systemd-networkd (рисунок 42).

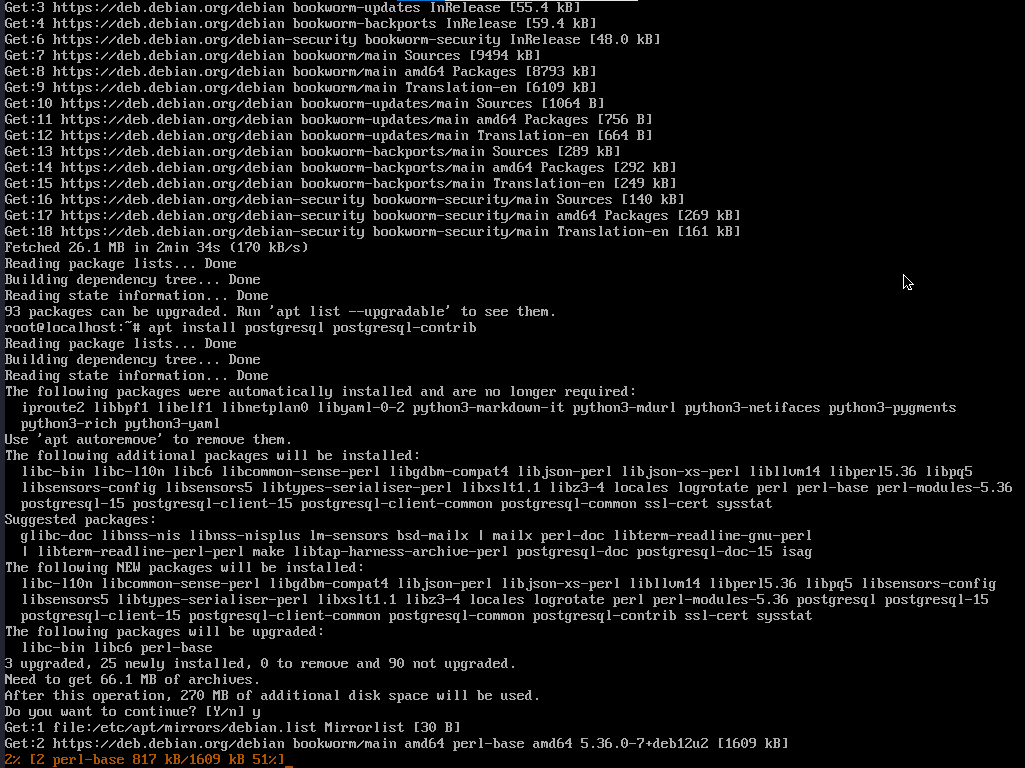


Рисунок 42 – Установка PostgreSQL

* 1. Переключимся на пользователя postgres и создадим новую роль и базу данных командами «createuser -P --interactive» и «psql», «CREATE DATABASE test OWNER devops;» (рисунок 43).

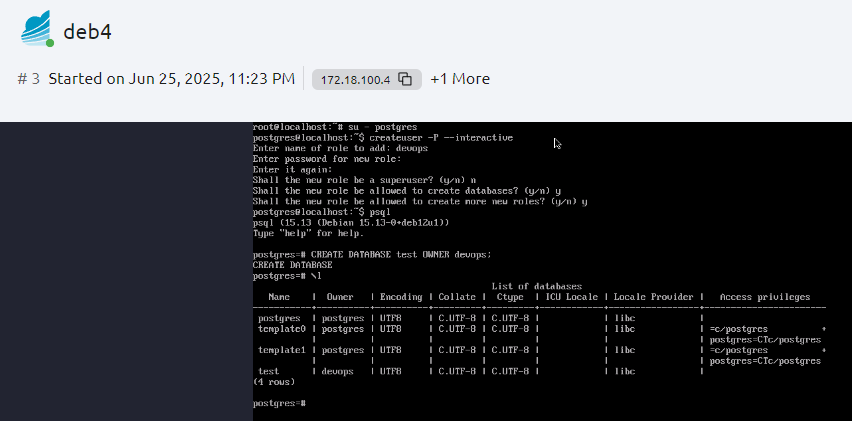


Рисунок 43 – Создание роли и базы данных

* 1. Для доступа к базам данных извне изменим файл /etc/postgresql/15/main/postgresql.conf, разрешив прослушивание любого адреса, раскомментировав и приведя к следующему виду строку «listen\_addresses = '\*'» (рисунок 44).

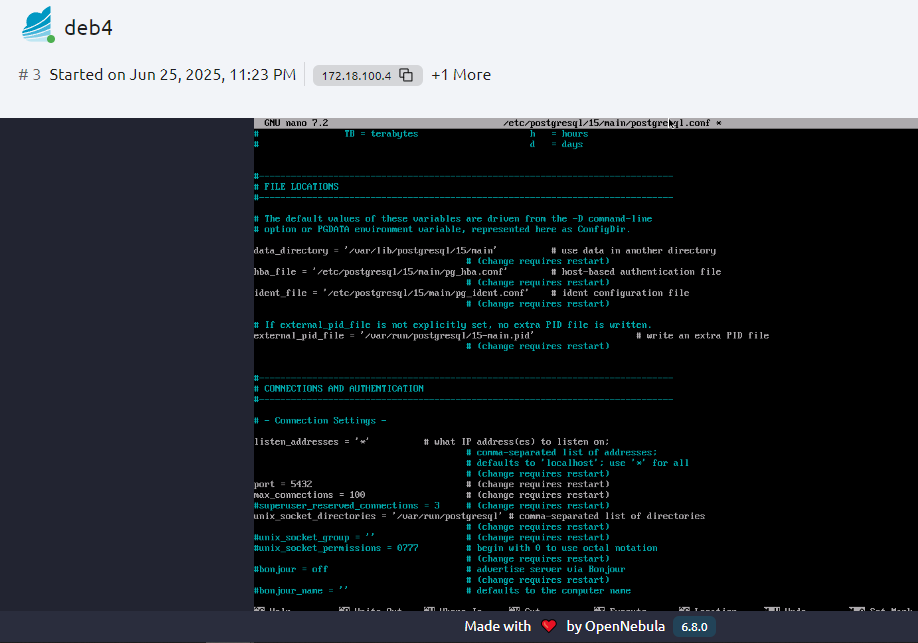


Рисунок 44 – Настройка адресов прослушивания

* 1. Также добавим в /etc/postgresql/15/main/pg\_hba.conf строку, разрешающую удаленное подключение с хоста с адресом 172.18.100.2/32, которым в дальнейшем будет машина с веб-приложением (рисунок 45).

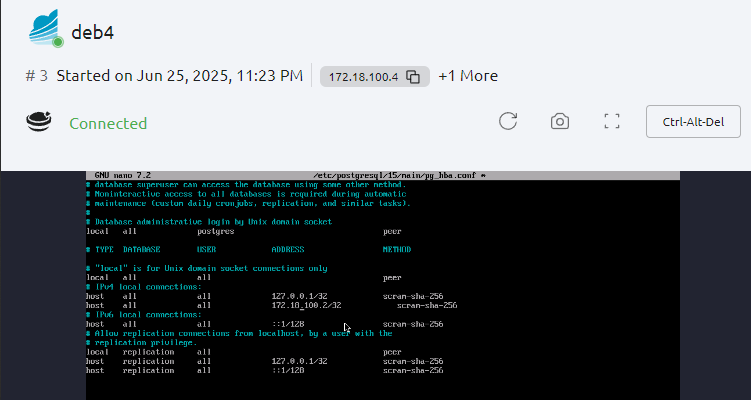


Рисунок 45 – Настройки разрешения удаленного подключения

* 1. Настройка PostgreSQL завершена, перезапустим сервис postgresql для применения всех настроек (рисунок 46).

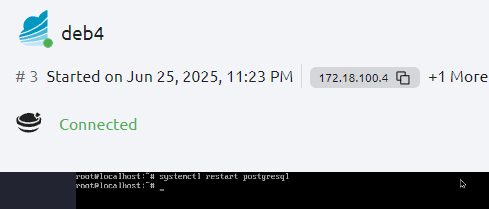


Рисунок 46 – Перезапуск postgresql

* 1. Развернем веб-приложение на основе Flask в Docker на deb2, для этого сначала установим Docker командой «apt update && apt install docker.io», предварительно добавив «DNS=8.8.8.8» в настройки .network (рисунок 47).

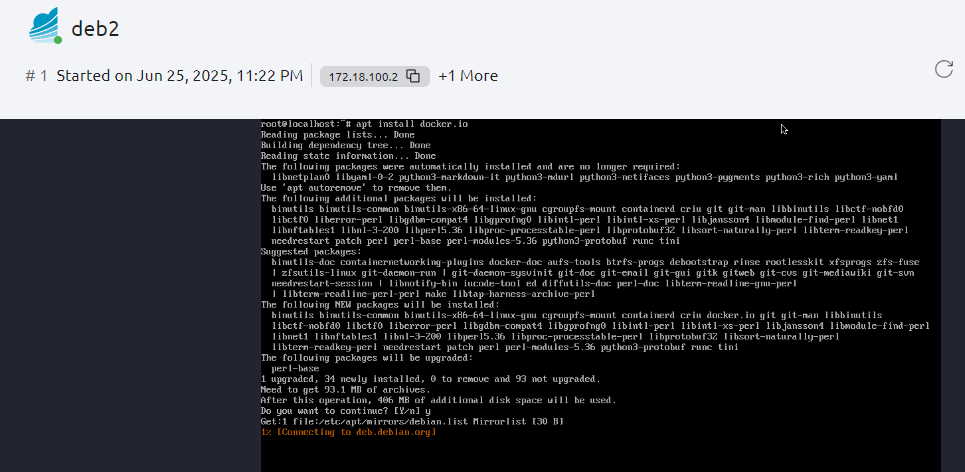


Рисунок 47 – Установка Docker

* 1. Создадим файл «/root/web-app.py», в котором будет создаваться веб-приложение Flask со стартовой страницей и страницей /status, в которой будет выполняться подключение к базе данных и выводиться сообщение об его успешности (рисунок 48).

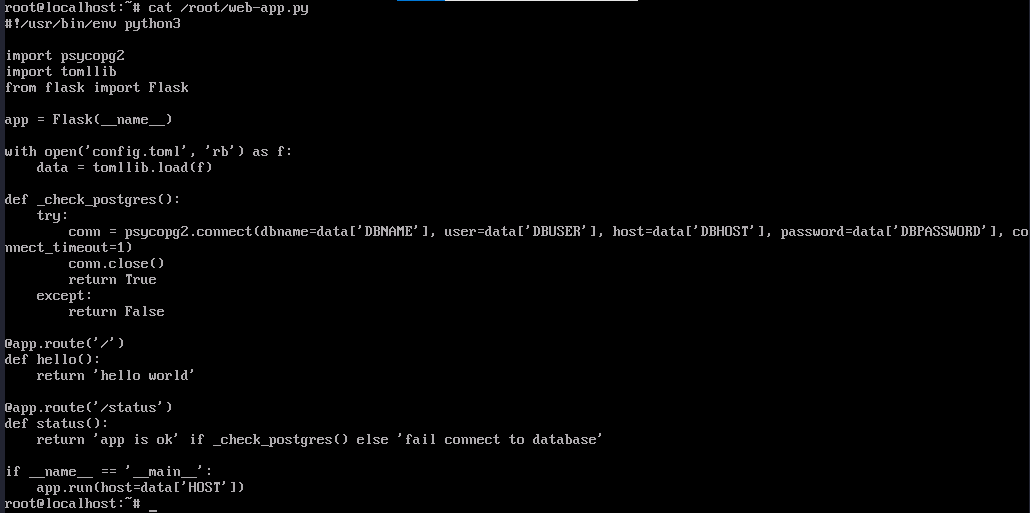


Рисунок 48 – Файл «/root/web-app.py»

* 1. Создадим файл «/root/config.toml», в котором хранятся данные для подключения к базе данных (рисунок 49). Данный файл будет импортироваться в web-app.py.



Рисунок 49 – Файл «/root/config.toml»

* 1. Создадим Dockerfile. Укажем в нем использование образа python:3.11-alpine, установку библиотек для работы приложения. Также настроим копирование файлов web-app.py и config.toml в образ и запуск приложения при запуске контейнера (рисунок 50).

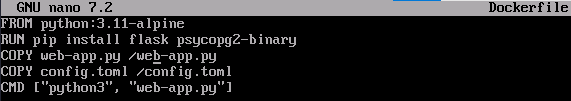


Рисунок 50 – Файл Dockerfile

* 1. Соберем образ под названием webapp (рисунок 51).

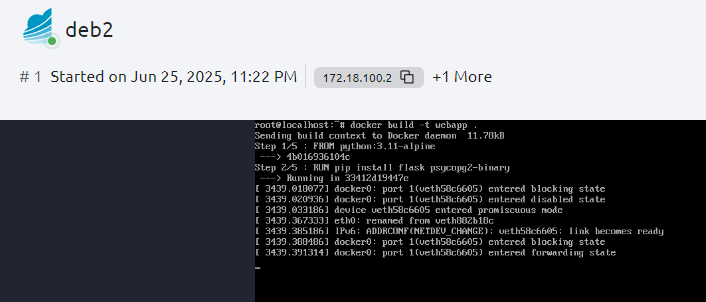


Рисунок 51 – Сборка образа

* 1. Создадим два контейнера на основе образа. Для контейнера webapp1 укажем проброс порта 5000 на 5001, для webapp2 – на 5002 командами «docker run -p 5001:5000 --name=webapp1 --restart=always -d webapp» и «docker run -p 5002:5000 --name=webapp2 --restart=always -d webapp» (рисунок 52).

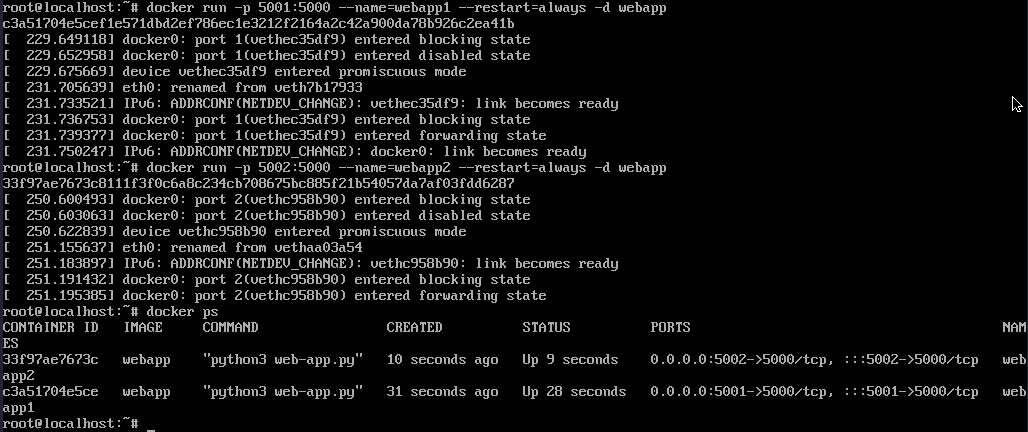


Рисунок 52 – Создание контейнеров

* 1. Проверим работоспособность приложения, обратившись к нему с помощью curl (рисунок 53).

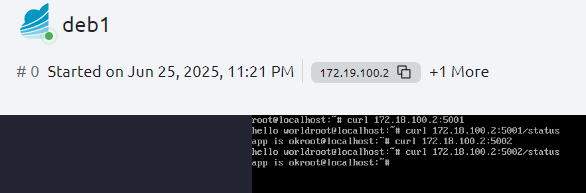


Рисунок 53 – Проверка приложения

* 1. Вернемся к машине vr1 и настроим балансировщик нагрузки с помощью nginx. Для начала установим его командой «apt update && apt install nginx» (рисунок 54).

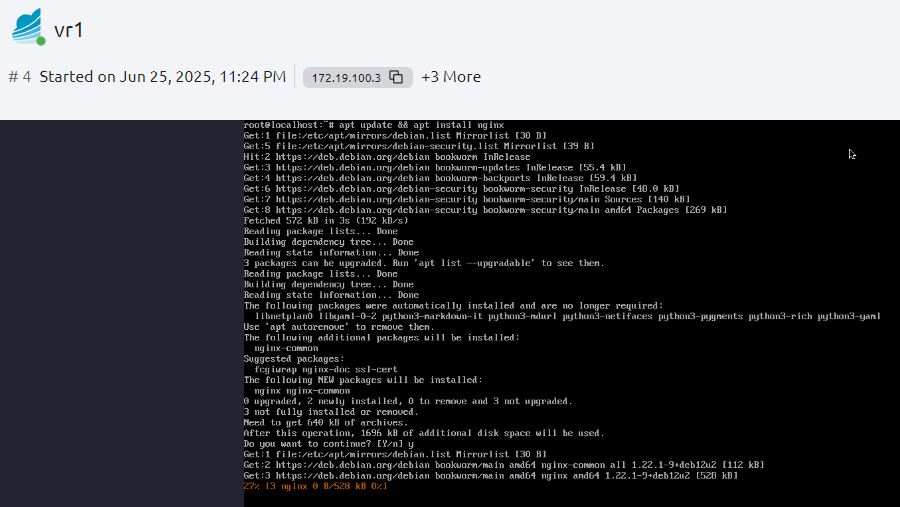


Рисунок 54 – Установка nginx

* 1. Отредактируем файл /etc/nginx/sites-available/default. В блоке upstream укажем сервера 172.18.100.2:5001 и 172.18.100.2:5002, на которых работают веб-приложения. Создадим 2 сервера. Один прослушивает порт 80 и доступен только для публичной сети. Другой прослушивает порт 5000 и доступен только для приватной сети. Оба сервера выполняют роль обратного прокси-сервера (рисунок 55).

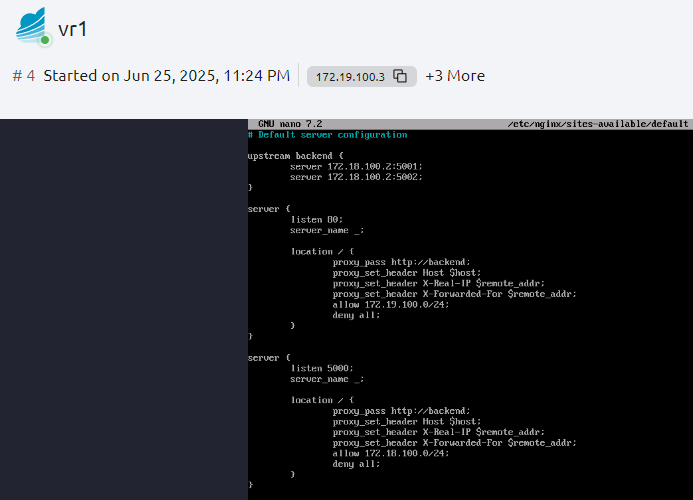


Рисунок 55 – Файл /etc/nginx/sites-available/default

* 1. Перезапустим сервис nginx и проверим работоспособность приложения. Обратимся к нему с любой машины из приватной сети (рисунок 56). Приложение доступно только при обращении по адресу с использованием 5000 порта, т.к. машина находится в приватной сети.



Рисунок 56 – Проверка приложения из приватной сети

* 1. Настроим машину deb1 для приложения locust, для начала проверим доступность приложения из публичной сети (рисунок 57).

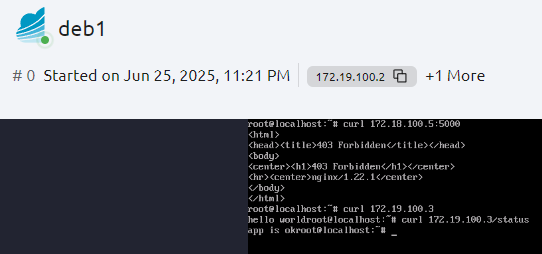


Рисунок 57 – Проверка приложения из публичной сети

* 1. Теперь развернем python приложение Locust. Установим утилиту pipx командой «apt update && apt install pipx» (рисунок 58), предварительно добавив «DNS=8.8.8.8» в настройки .network и перезапустив systemd-networkd.

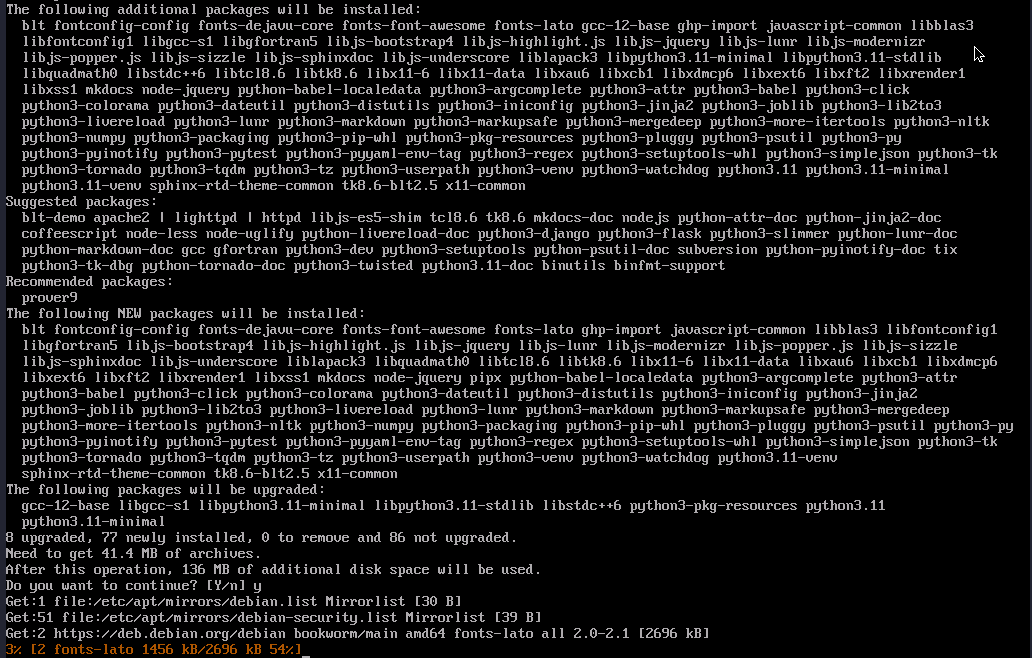


Рисунок 58 – Установка pipx

* 1. Выполним обновление переменной PATH командой «pipx ensurepath» (рисунок 59).

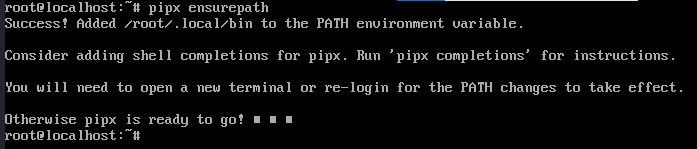


Рисунок 59 – Обновление переменной PATH

* 1. Нажмем «Ctrl+d» чтобы завершить сеанс и применить изменения. Снова входим в учетную запись root и устанавливаем locust командой «pipx install locust» (рисунок 60).

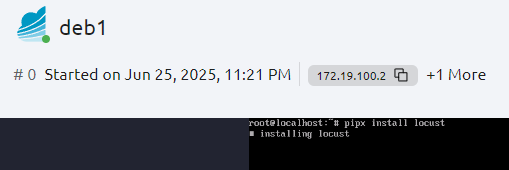


Рисунок 60 – Установка locust

* 1. Создадим файл «/root/locustfile.py» со следующим содержимым (рисунок 61).

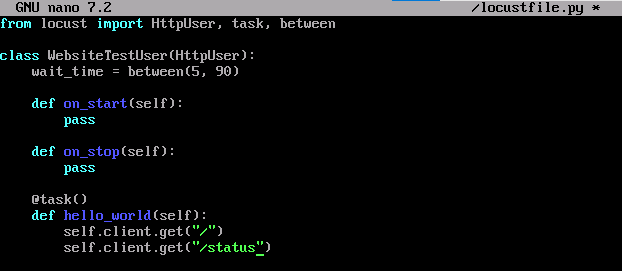


Рисунок 61 – Файл «/root/locustfile.py»

* 1. Теперь запустим приложение, выполнив команду «locust --headless --users 100 --spawn-rate 10 -H http://172.19.100.3 --run-time 10s» (рисунок 62).



Рисунок 62 – Запуск Locust

* 1. Приложение исправно работает и отправляет запросы (рисунок 63).

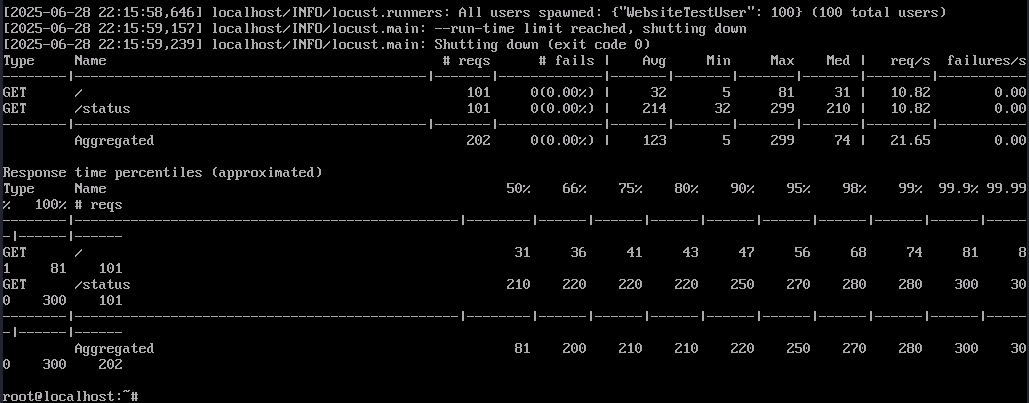


Рисунок 63 – Процесс работы Locust

1. **Вывод**
   1. Мы развернули приложение в отказоустойчивой масштабируемой облачной-инфраструктуре на платформе OpenNebula.