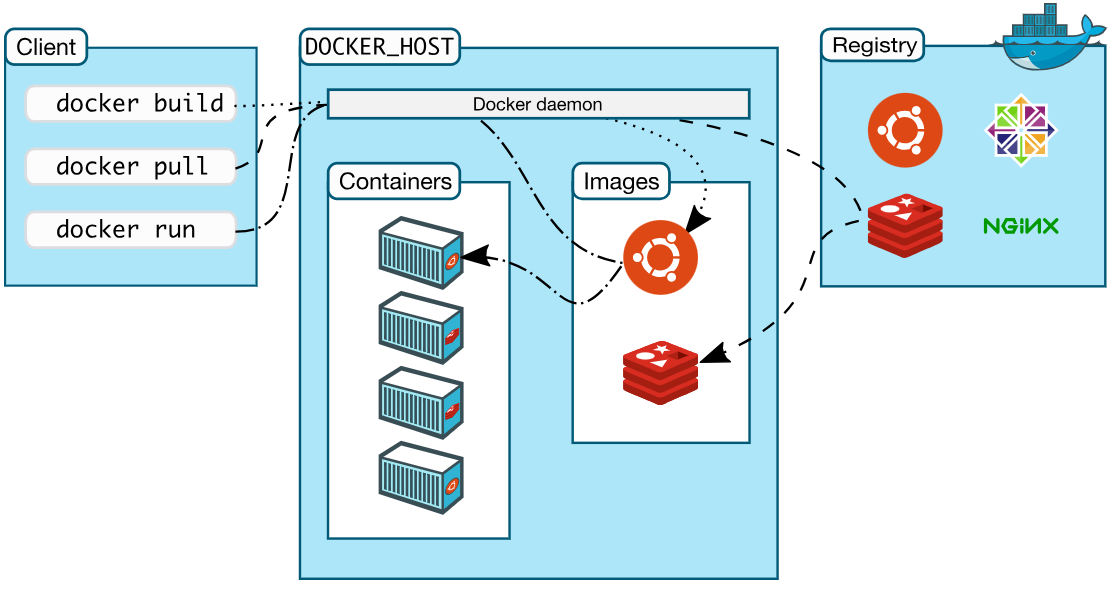
**Лабораторная работа №8. Базовые команды Docker**

****

**Docker демон**

Docker демон (containerd или dockerd) обслуживает Docker API запросы (через сокет или по http) и управляет Docker-объектами (образа, контейнеры, сети, тома, и т.д.). Проверить: systemctl status dockerd.

**Docker клиент**

Docker клиент (docker) основной способ взаимодействия пользователя с Docker демоном (против работы напрямую с API). Когда вы запускаете команду docker run, клиент отправляет сообщение процессу dockerd (или containerd), который его обрабатывает. Утилита docker использует Docker API и может работать сразу с несколькими Docker демонами, не обязательно на локальной машине.

**Docker Desktop**

Docker Desktop приложение для Mac, Windows или Linux окружений, которое позволяет вам собирать и публиковать ваши контейнеризованные приложения и сервисы. Docker Desktop включает в себя Docker демон, Docker клиент, Docker Compose (с ним будем работать отдельно позже), Docker Content Trust, Kubernetes, и Credential Helper.

**Docker registry**

Docker *registry* (реестр) хранит Docker образа. Docker Hub - публичный реестр для общего пользования, и Docker настроен искать образа там по умолчанию. Также вы можете использовать свой личный реестр.

**Подготовка рабочего окружения**

В лабораторной работе по Linux мы уже настроили рабочее окружение в ОС Ubuntu и установили Docker. Если по каким-либо причинам вы этого еще не сделали - проделайте прямо сейчас:

Установка Docker в Ubuntu 22.04:

sudo apt install apt-transport-https ca-certificates curl software-properties-common  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"  
sudo apt update  
sudo apt install docker-ce  
sudo systemctl status docker

Запустим [hello-world](https://hub.docker.com/_/hello-world):

docker run hello-world

После запуска контейнер отрабатывает, выводит результат работы в STDOUT и сразу завершается, т.к. основной процесс печати руководства внутри него после выполнения завершился.

Если вы хотите, чтобы ваш пользователь мог пользоваться docker без ограничений и необходимости прав суперпользователя - вы можете добавить его в группу docker:

usermod -aG docker $username

Не забудьте после этого перелогиниться, чтобы процесс командного интерпретатора (в моем случае bash) запустился с новыми правами (с правами на группу docker)

**Команды для работы с образами - docker pull, docker images**

Для того, чтобы запустить "контейнер" нам нужна сущность под названием образ. Подробно все сущности мы разбирали на второй лекции. Остановимся на том, что это заранее упакованный слепок состояния некоторого сервиса или окружения (бибилиотеки и исполняемые файлы), которое можно использовать без дополнительных зависимостей.

Для загрузки образа используется команда docker pull, загрузим образ mysql (найти вручную подходящие образа можно на [docker hub](https://hub.docker.com/search?q=mysql" \t "_blank)):

docker pull mysql



Когда загрузка и распаковка будет завершена, утилита вернет вам управление. Теперь проверим, что же мы скачали с помощью команды docker image ls или docker images:

docker image ls  
docker images

Но что, если мы хотим скачать определенную версию образа? Нам помогут метки. Загрузим mysql версии 5.7.39:

docker pull mysql:5.7.39

Убедимся, что теперь у нас есть 2 образа с разными метками:

docker images

**Метки и удаление образа - docker tag**

Чтобы управлять нашими образами мы можем назначать им метки - тэги, с помощью команды docker tag:

docker tag mysql mysql:8.0-mak



Это не создало новый образ (image ID одинаковый), но сделало еще одну ссылку на него.

**Удалить образ можно удалив все ссылки на него командой** docker rmi или аналогом docker image rm:

docker rmi mysql:8.0-mak  
docker image rm mysql:5.7.39

Если вы не хотите вручную выяснять, какие образа вам более не требуются воспользуйтесь командой docker image prune:

docker image prune  
*# docker image prune -a # удалить все, на которых не запущены контейнеры*

**Запускаем контейнер - docker run, docker logs**[**​**](https://iu5edu.ru/wiki/devops/docs/labs/docker/commands/#%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA%D0%B0%D0%B5%D0%BC-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80---docker-run-docker-logs)

Запустим контейнер с СУБД командой docker run:

docker run mysql

mak@overmind:~$ docker run mysql  
2022-09-21 23:33:40+00:00 [Note] [Entrypoint]: Entrypoint script *for* MySQL Server 8.0.30-1.el8 started.  
2022-09-21 23:33:40+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'  
2022-09-21 23:33:40+00:00 [Note] [Entrypoint]: Entrypoint script *for* MySQL Server 8.0.30-1.el8 started.  
2022-09-21 23:33:40+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified  
 You need to specify one of the following:  
 - MYSQL\_ROOT\_PASSWORD  
 - MYSQL\_ALLOW\_EMPTY\_PASSWORD  
 - MYSQL\_RANDOM\_ROOT\_PASSWORD

Окей, исправляемся, добавим аргумент -e для передачи переменной окружения, зададим пароль:

docker run -e MYSQL\_ROOT\_PASSWORD=password mysql

База запустилась, но как будто мы просто запустили ее командой mysql из консоли.

**Задача**: поднять еще одно окно и прервать процесс контейнера, который подвесил вашу консоль без ее завершения. Пригодится: tmux, ps aux, kill

Теперь попробуем запустить в фоновом режиме с аргументом -d, кроме того, дадим ему вменяймое имя --name $NAME:

docker run -d -e MYSQL\_ROOT\_PASSWORD=password --name db1 mysql

Получилось, в ответ Docker выдал нам хэш-ID контейнера (далее $ID).

Посмотрим логи контейнера по его ID командой docker logs:

docker logs $ID  
*# Это можно сделать и по имени*  
docker logs db1

**Списки контейнеров - docker ps**

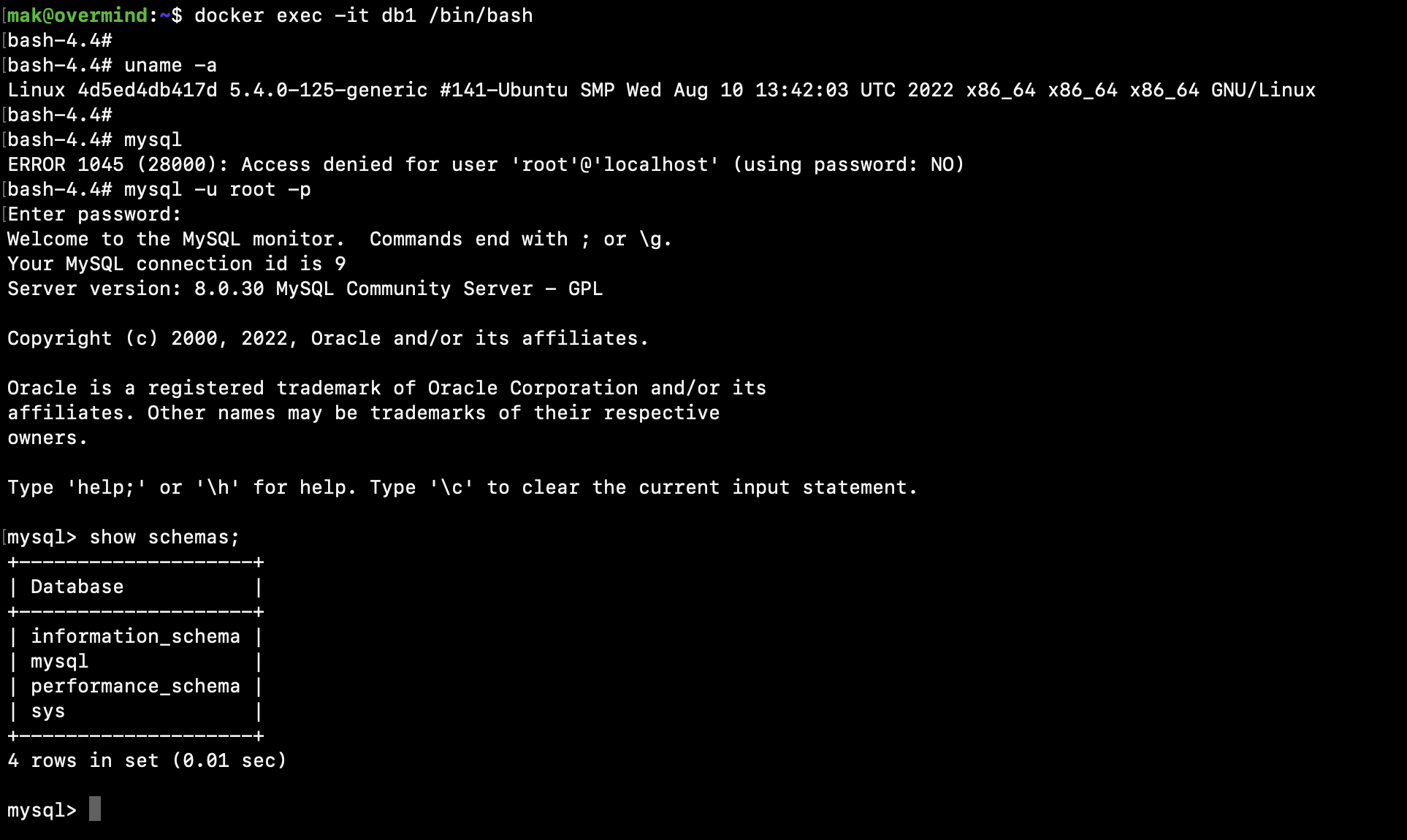
Посмотрим списки контейнеров с помощью команды docker ps:

docker ps  
docker ps -a *# включая завершенные*

**Подключаемся к работающему контейнеру - docker exec**

Подключимся (запустим еще один процесс внутри контейнера) с помощью команды docker exec:

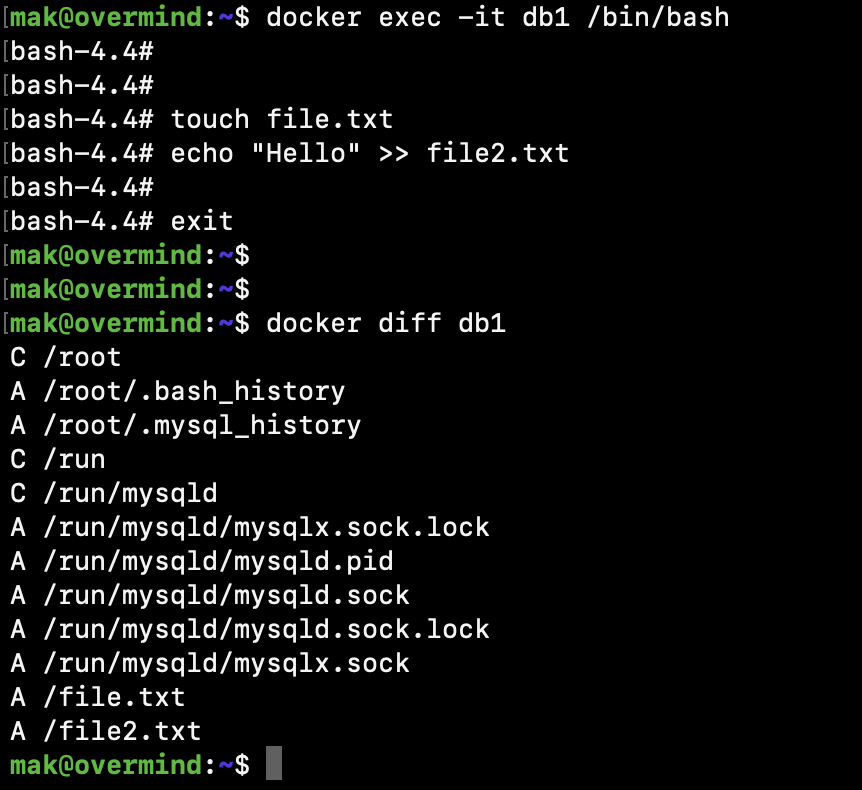
*# i - interactive - держать STDIN открытым*  
*# t - tty - создать псевдо-tty*  
docker exec -it db1 /bin/bash



**Список изменений - docker diff[​](https://iu5edu.ru/wiki/devops/docs/labs/docker/commands/" \l "%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA-%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9---docker-diff" \o "Прямая ссылка на список-изменений---docker-diff)**

Просмотрим список изменений в слое на файловой системе с помощью команды docker diff:

docker diff db1



Видимо наличие новых файлов file.txt и file2.txt.

**Завершаем контейнер - docker stop, docker kill, docker rm[​](https://iu5edu.ru/wiki/devops/docs/labs/docker/commands/" \l "%D0%B7%D0%B0%D0%B2%D0%B5%D1%80%D1%88%D0%B0%D0%B5%D0%BC-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80---docker-stop-docker-kill-docker-rm" \o "Прямая ссылка на завершаем-контейнер---docker-stop-docker-kill-docker-rm)**

Завершим контейнер нежно с помощью stop, сразу завершим с помощью [kill](https://docs.docker.com/engine/reference/commandline/kill/" \t "_blank) и удалим остатки (из спика завершенных, включая логи контейнера) c помощью [rm](https://docs.docker.com/engine/reference/commandline/rm/" \t "_blank):

docker stop db1  
docker kill db1  
docker rm db1

**Не теряем данные - docker volume[​](https://iu5edu.ru/wiki/devops/docs/labs/docker/commands/" \l "%D0%BD%D0%B5-%D1%82%D0%B5%D1%80%D1%8F%D0%B5%D0%BC-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5---docker-volume" \o "Прямая ссылка на не-теряем-данные---docker-volume)**

Запустим контейнер обратно:

docker run -d -e MYSQL\_ROOT\_PASSWORD=password --name db1 mysql  
*# И тут я спалил пароль в истории в открытом виде*

Как избежать утечки пароля через историю bash?

*# Грузите из переменных окружения из файла:*  
nano .docker\_mysql\_rc  
*# Вписываем необходимые переменные*  
  
*# Передаем как переменную (если одна)*  
source .docker\_mysql\_rc  
docker run -d -e MYSQL\_ROOT\_PASSWORD=${MYSQL\_ROOT\_PASSWORD} --name db1 mysql  
  
*# Или целиком файл (если много):*  
docker run -d --env-file ./.docker\_mysql\_rc --name db1 mysql

Выведем содержимое файла:

docker exec -it db1 cat file2.txt

О ужас! Его нет. Как и всего содержимого базы. **Все данные удалились при завершении контейнера**.

Чтобы этого избежать запустим контейнер с томом:

docker run --rm -d \  
 -v mysql:/var/lib/mysql \  
 -v mysql\_config:/etc/mysql \  
 --name db1 \  
 -e MYSQL\_ROOT\_PASSWORD=password \  
 mysql

Внесем изменения:

docker exec -it db1 mysql -ppassword  
*# И тут я снова спалил пароль в истории в открытом виде*  
*# Лучше написать -p, а дальше вводить интерактивно*  
*# Но так как задача образовательная - пока игнорируем*

*create* *database* testdb;  
*create* *database* blog;  
*show* schemas;

Завершим контейнер:

docker stop db1

А теперь перезапустим и убедимся, что базы testdb и blog не исчезли (как и все изменения на файловую систему):

docker run --rm -d \  
 -v mysql:/var/lib/mysql \  
 -v mysql\_config:/etc/mysql \  
 --name db1 \  
 -e MYSQL\_ROOT\_PASSWORD=password \  
 mysql  
   
docker exec -it db1 mysql -ppassword -e "show schemas;"

Просмотреть список томов:

docker volume ls

Создать новый:

docker volume create test

Посмотреть информацию о томе:

docker volume inspect test

Удалить том:

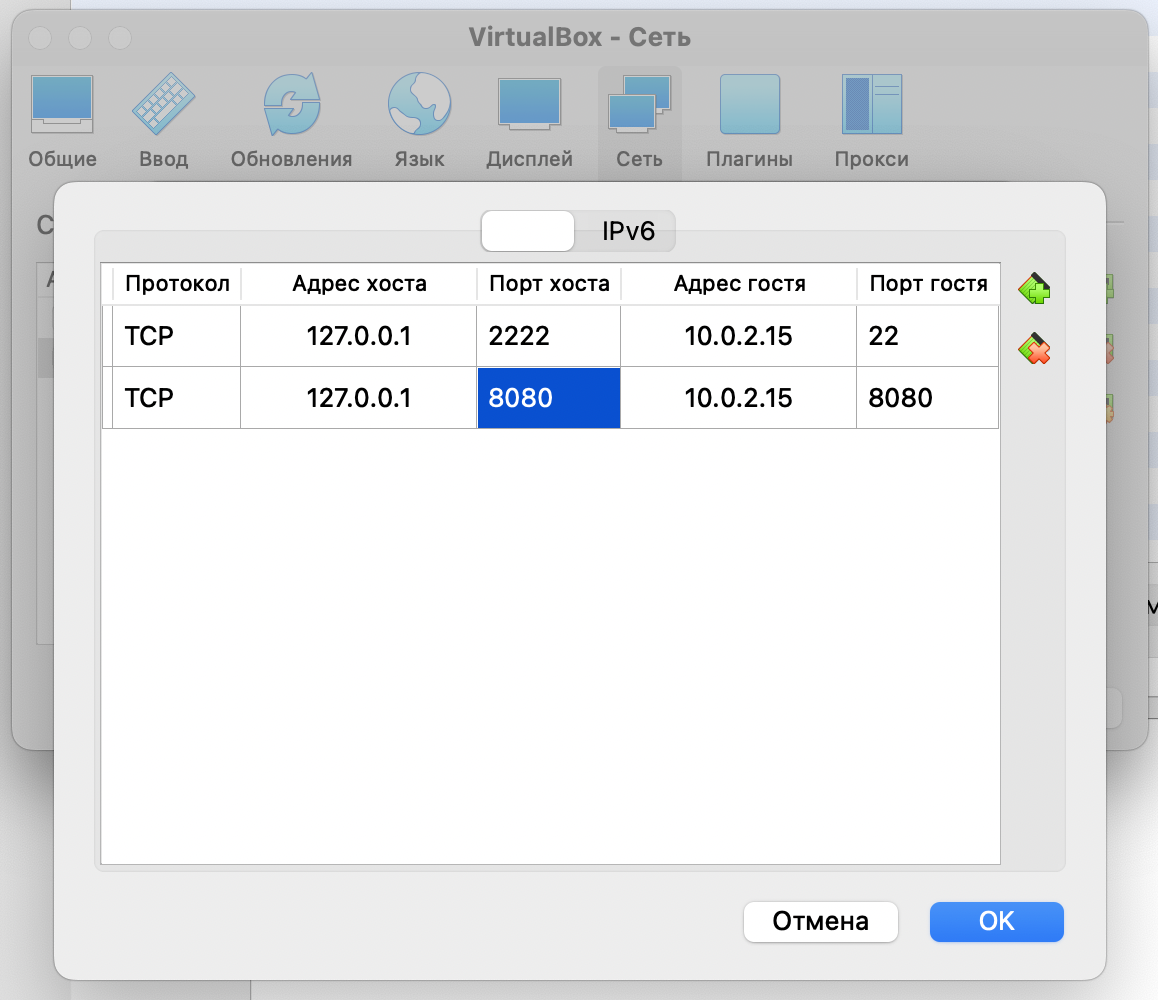
docker volume rm test

Очистить лишние тома:

docker volume prune

**Контейнер Adminer**

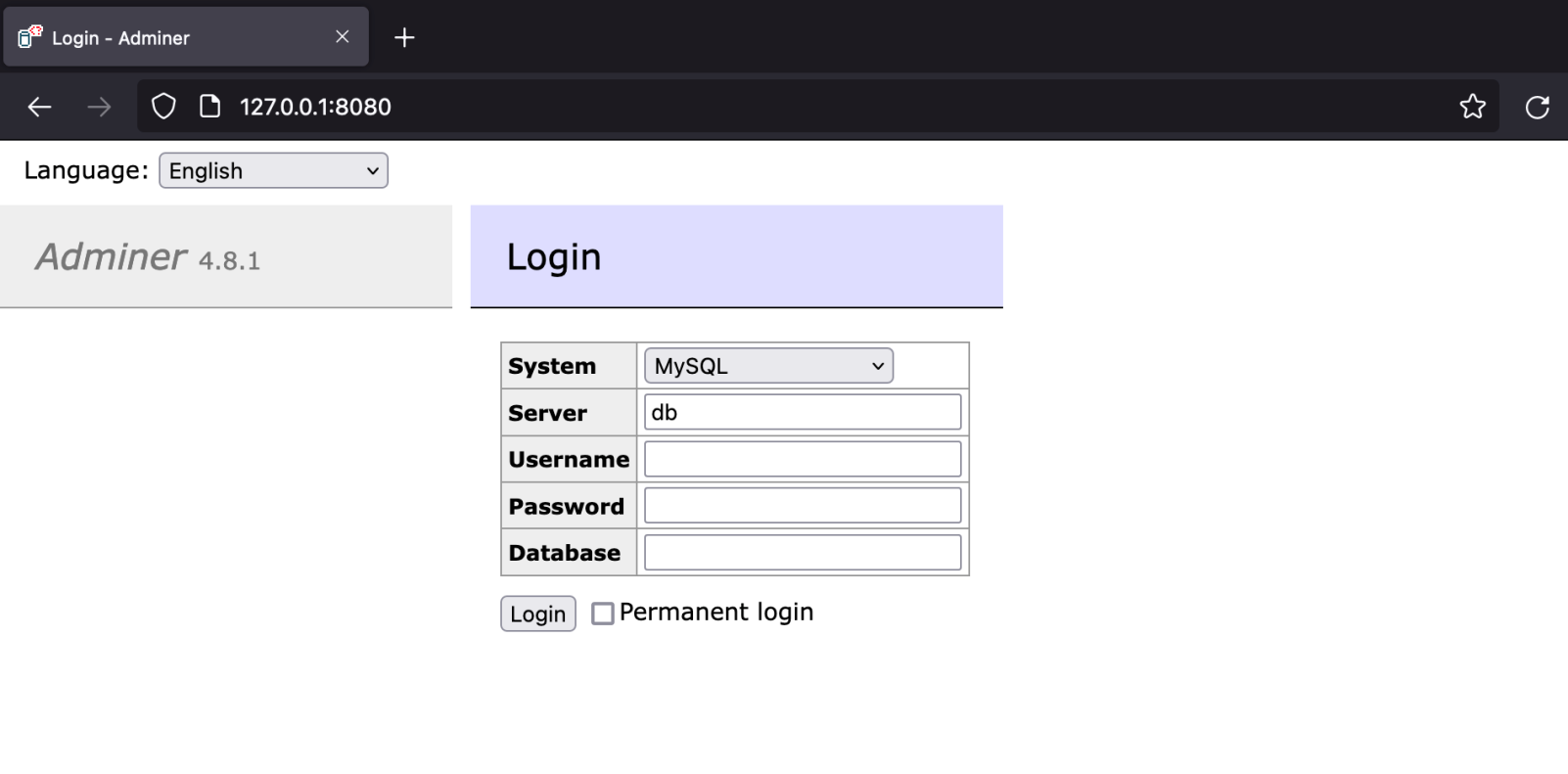
Настроим дополнительный проброс портов (если кто-то настрил bridge - то просто идем на IP виртуальной машины):



Запустим образ Adminer (скачается автоматически). Для того, чтобы попасть из виртуальки внутрь контейнера на порт 8080 укажем ключик -p HostPort:ContainerPort:

docker run -d -p 8080:8080 --name adminer adminer

Подключимся в браузере на хосте к http://127.0.0.1:8080/:



**Adminer** (бывший phpMinAdmin) — это легковесный инструмент администрирования [MySQL](https://ru.wikipedia.org/wiki/MySQL" \t "_blank), [PostgreSQL](https://ru.wikipedia.org/wiki/PostgreSQL" \t "_blank), [SQLite](https://ru.wikipedia.org/wiki/SQLite" \t "_blank), [MS SQL](https://ru.wikipedia.org/wiki/MS_SQL) и [Oracle](https://ru.wikipedia.org/wiki/Oracle_(%D0%A1%D0%A3%D0%91%D0%94)" \t "_blank). Проект родился как «облегчённый» вариант [phpMyAdmin](https://ru.wikipedia.org/wiki/PhpMyAdmin" \t "_blank). Распространяется в форме одиночного [PHP](https://ru.wikipedia.org/wiki/PHP)-файла размером около 380 KB, который является результатом компиляции исходных php- и js-файлов с помощью специального PHP-скрипта. Т.о. контейнер с ним содержит php-сервер и один php-скрипт.

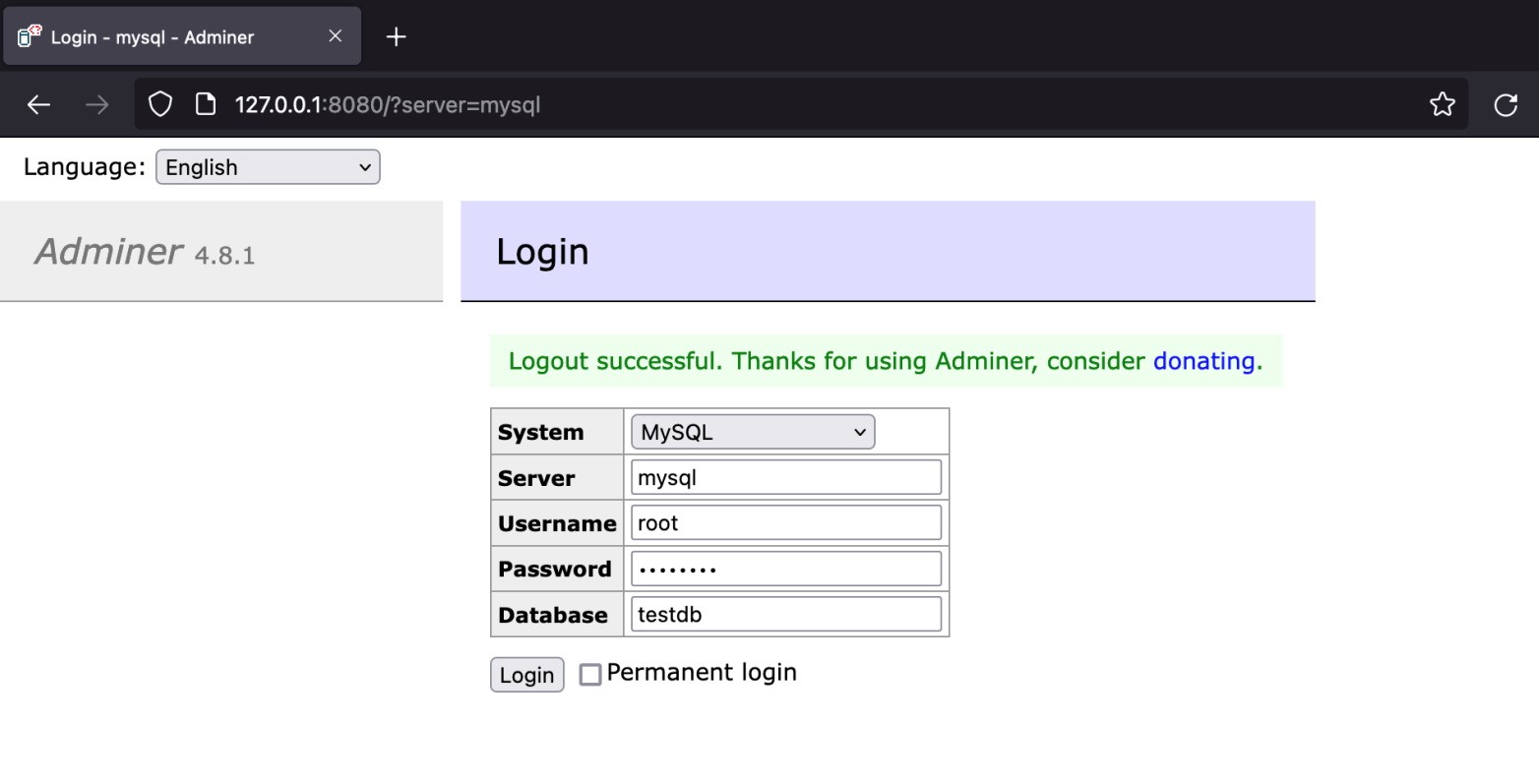
Однако как бы мы не пытались подключиться к базе - ничего не выйдет. Котейнеры не связаны по сети.

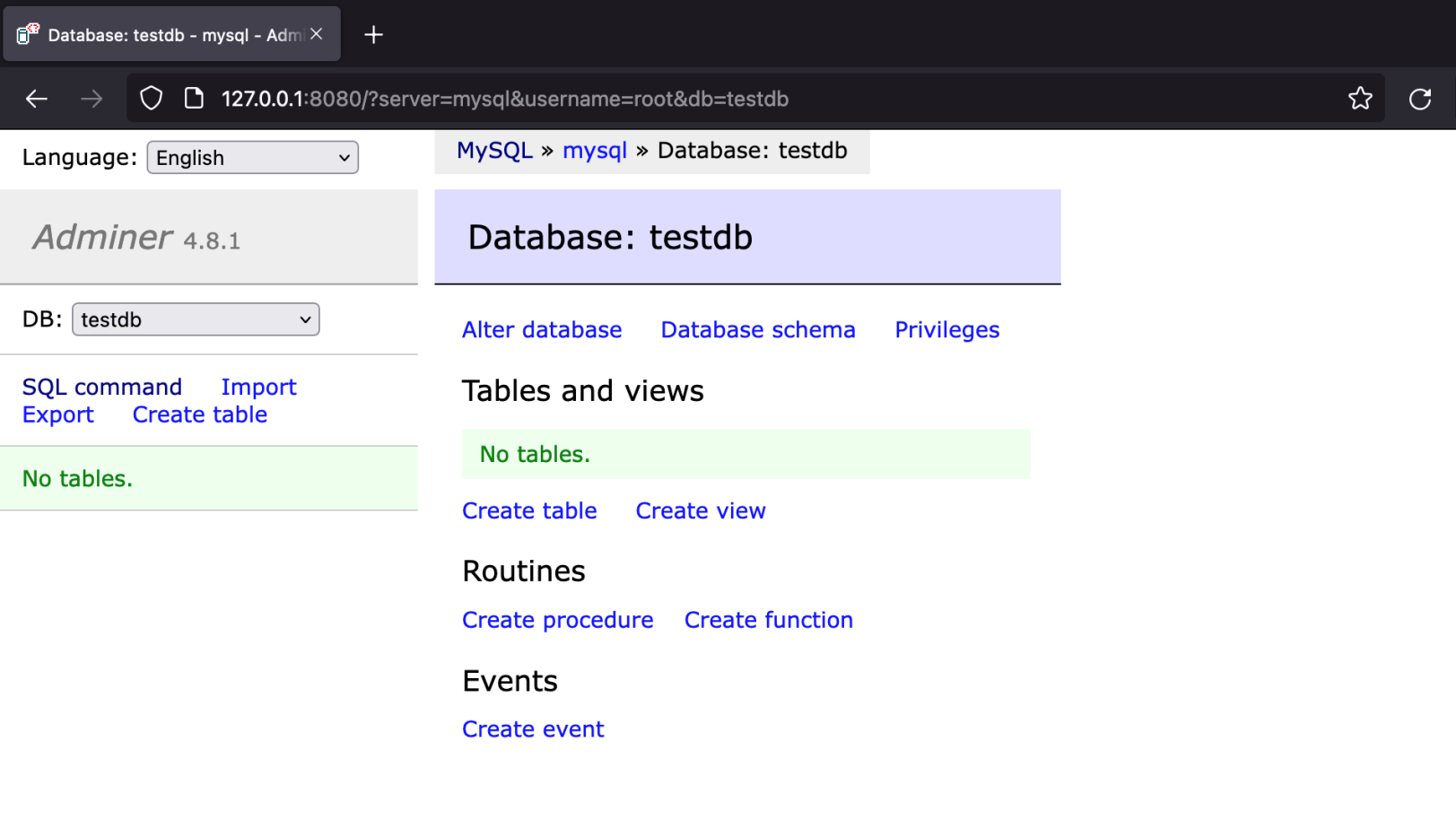
**Сети - docker network[​](https://iu5edu.ru/wiki/devops/docs/labs/docker/commands/" \l "%D1%81%D0%B5%D1%82%D0%B8---docker-network" \o "Прямая ссылка на сети---docker-network)**

Для начала попробуем связать контейнеры простым способом, завершим предыдущий контейнер Adminer и запустим новый, с параметром --link Container:AliasName.

docker rm -f adminer  
docker run -d -p 8080:8080 --link db1:mysql --name adminer adminer

Теперь подключимся к базе по ее Alias из контейнера adminer:



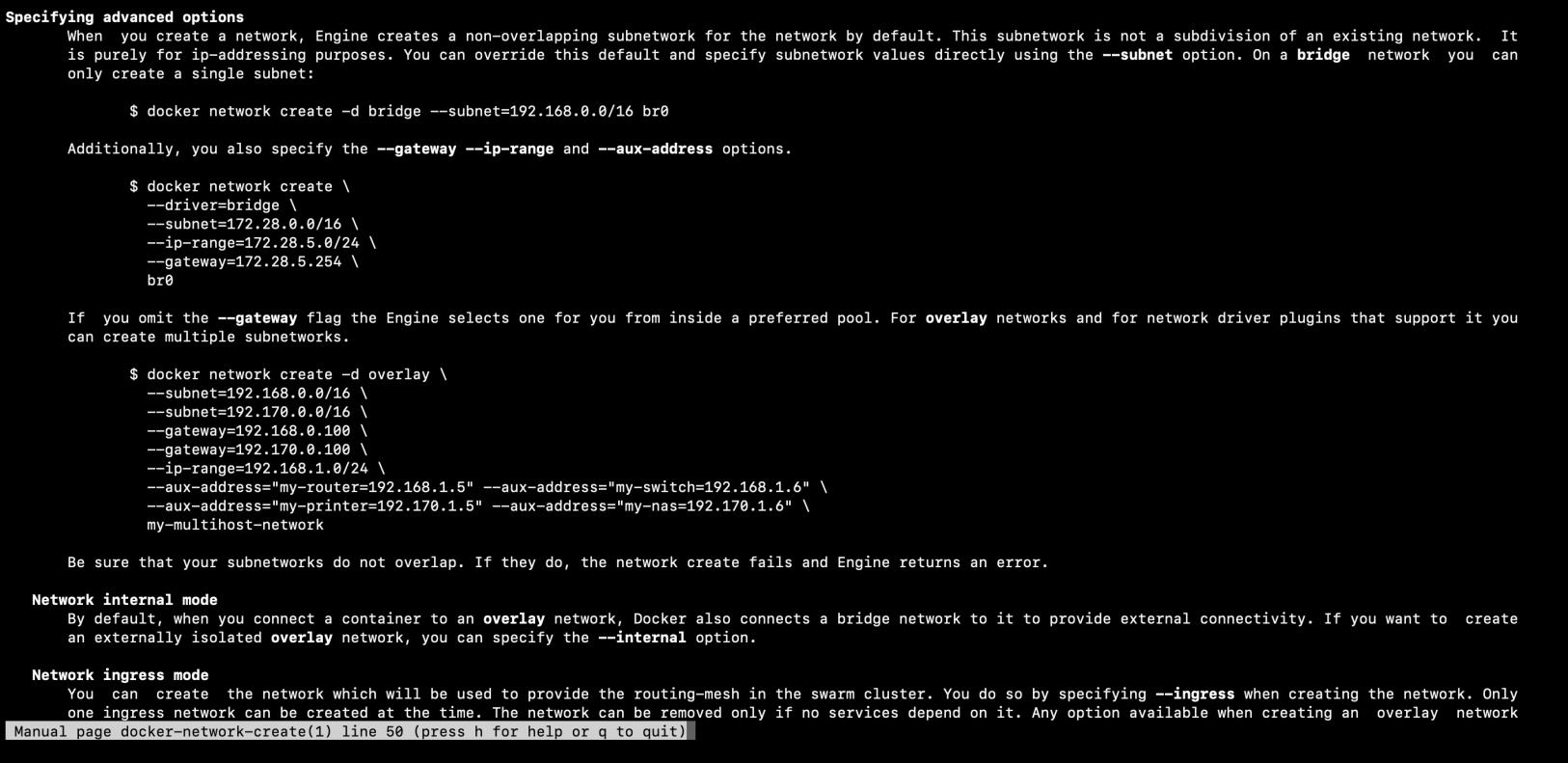


Неплохо, но такой способ считается устаревшим. К тому же, это может быть не всегда удобно. Теперь создадим новую сеть:

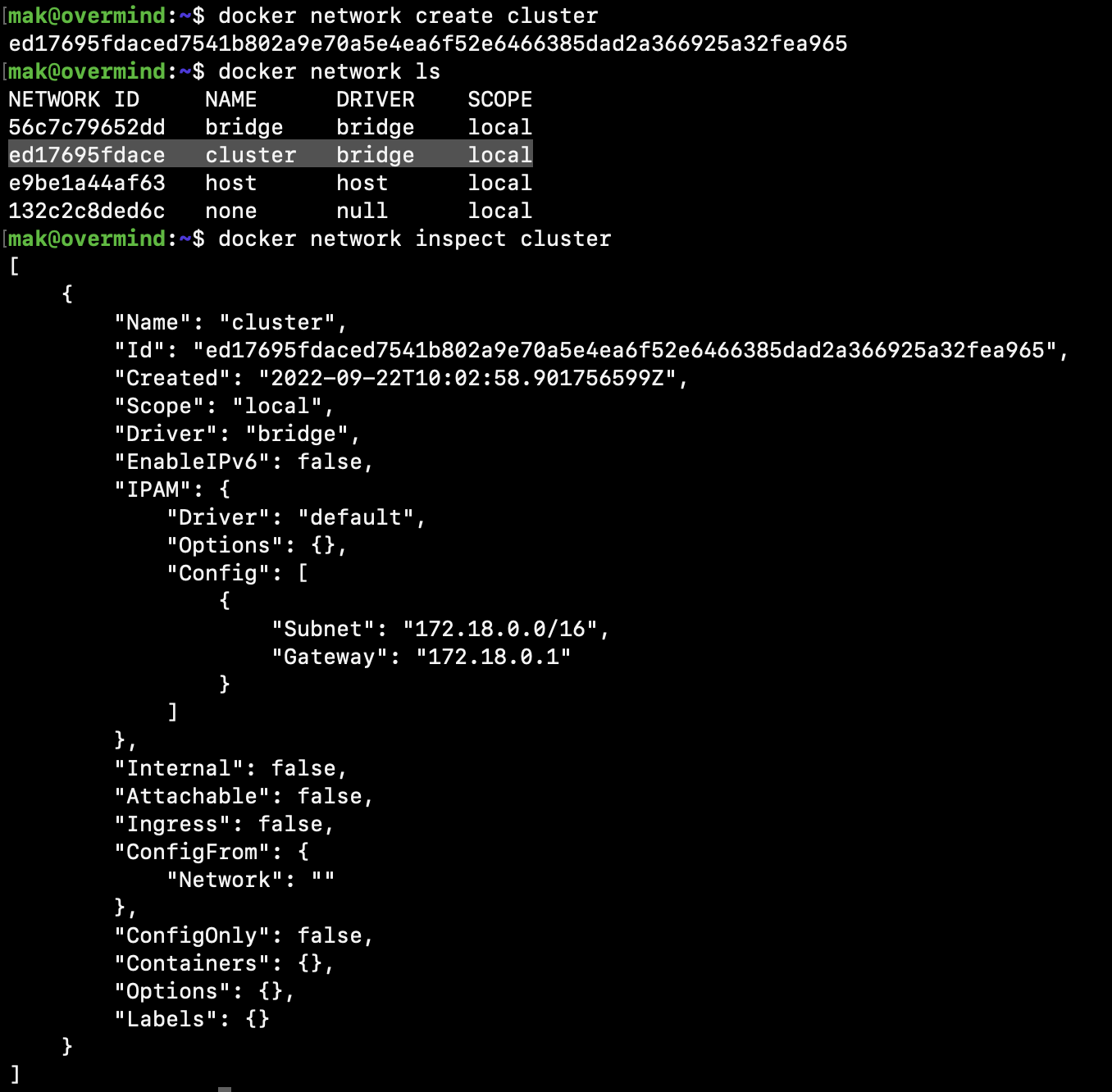
docker network create cluster

Кстати в man-странице по этой команде есть много интересного про параметры:

man docker-network-create

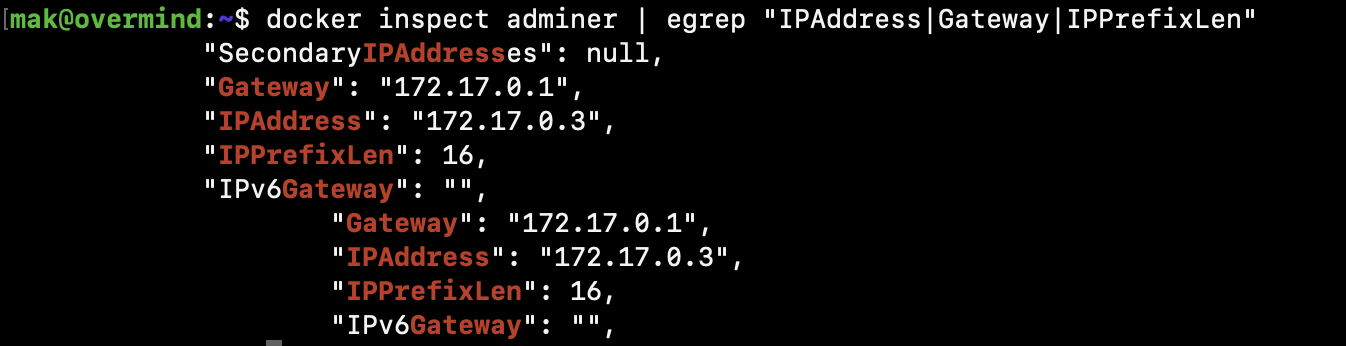


Проверим как создалась сеть с параметрами по умолчанию:



Проверим в какой сети работают сейчас Adminer и MySQL:

docker inspect db1  
docker inspect adminer | egrep "IPAddress|Gateway|IPPrefixLen"



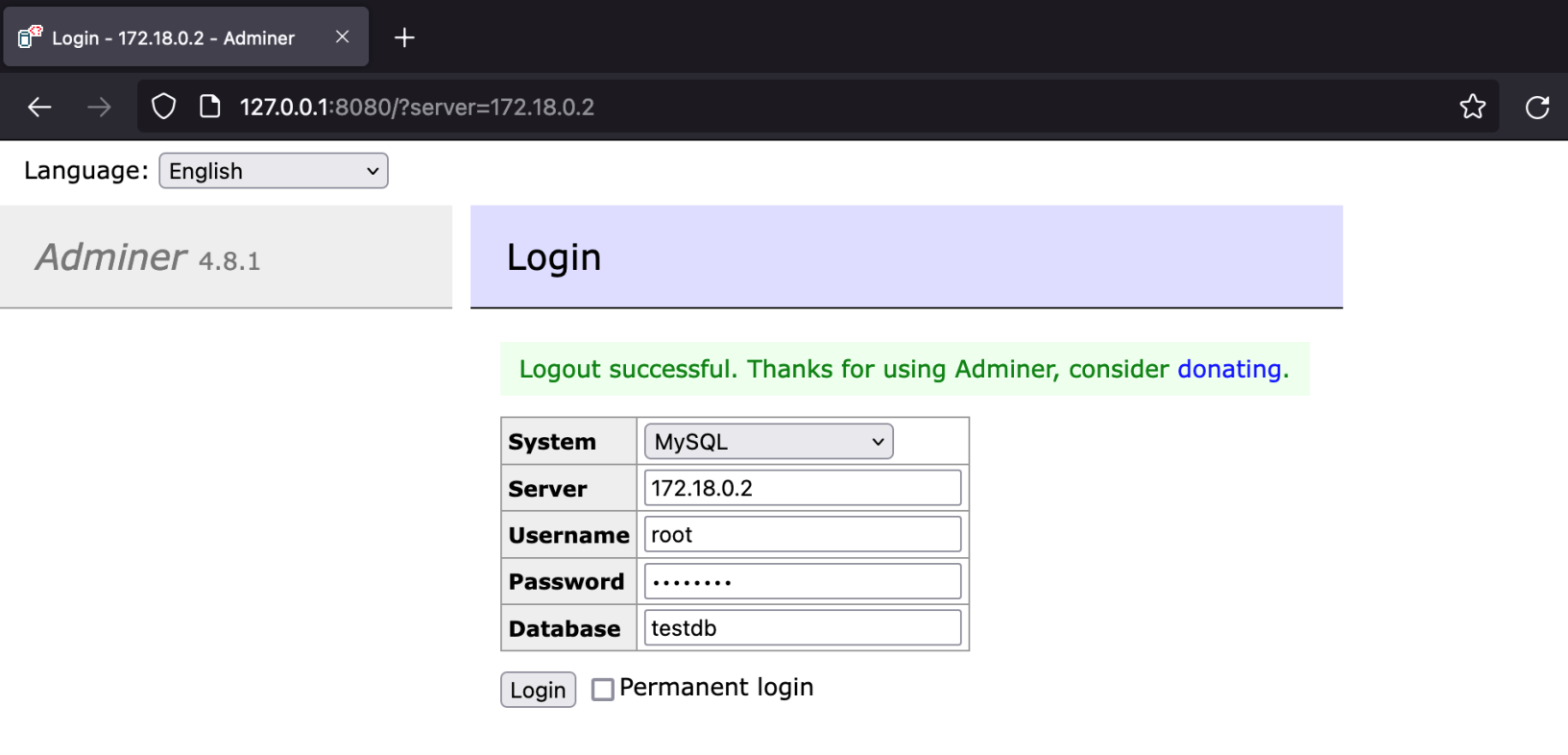
Теперь пересоздадим контейнеры СУБД и Adminer в этой сети:

docker rm -f db1  
docker rm -f adminer  
  
*# Добавим к предыдущей команде запуска: --net NetName*  
docker run --rm -d \  
 -v mysql:/var/lib/mysql \  
 -v mysql\_config:/etc/mysql \  
 --name db1 \  
 -e MYSQL\_ROOT\_PASSWORD=password \  
 --net cluster \  
 mysql  
  
*# А теперь аналогично для Adminer*  
docker run -d -p 8080:8080 --net cluster --name adminer adminer

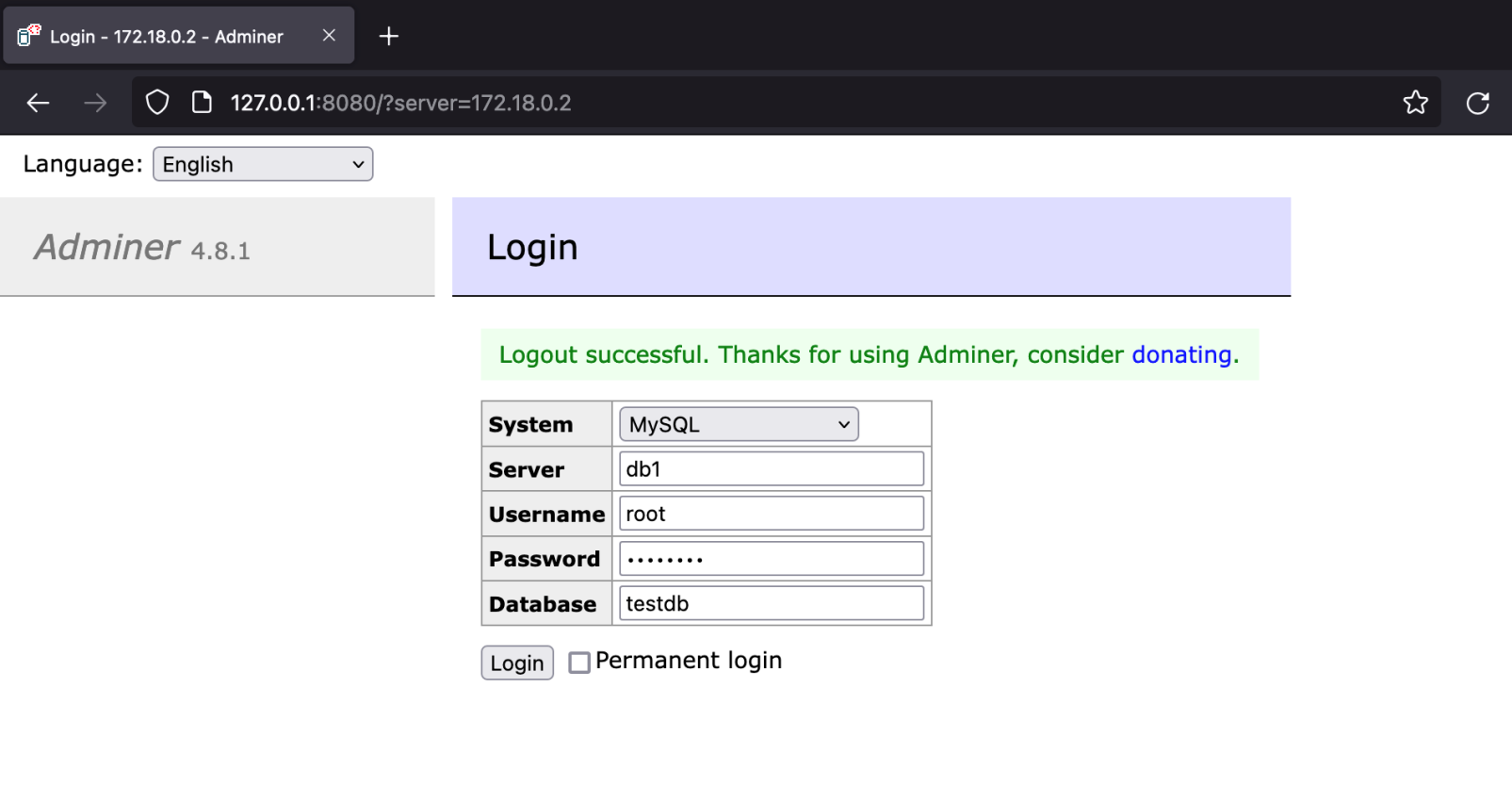
Проверим их IP-адреса:

docker inspect db1 | egrep "IPAddress|Gateway|IPPrefixLen"  
docker inspect adminer | egrep "IPAddress|Gateway|IPPrefixLen"

Пробуем подключиться к базе по IP-адресу:

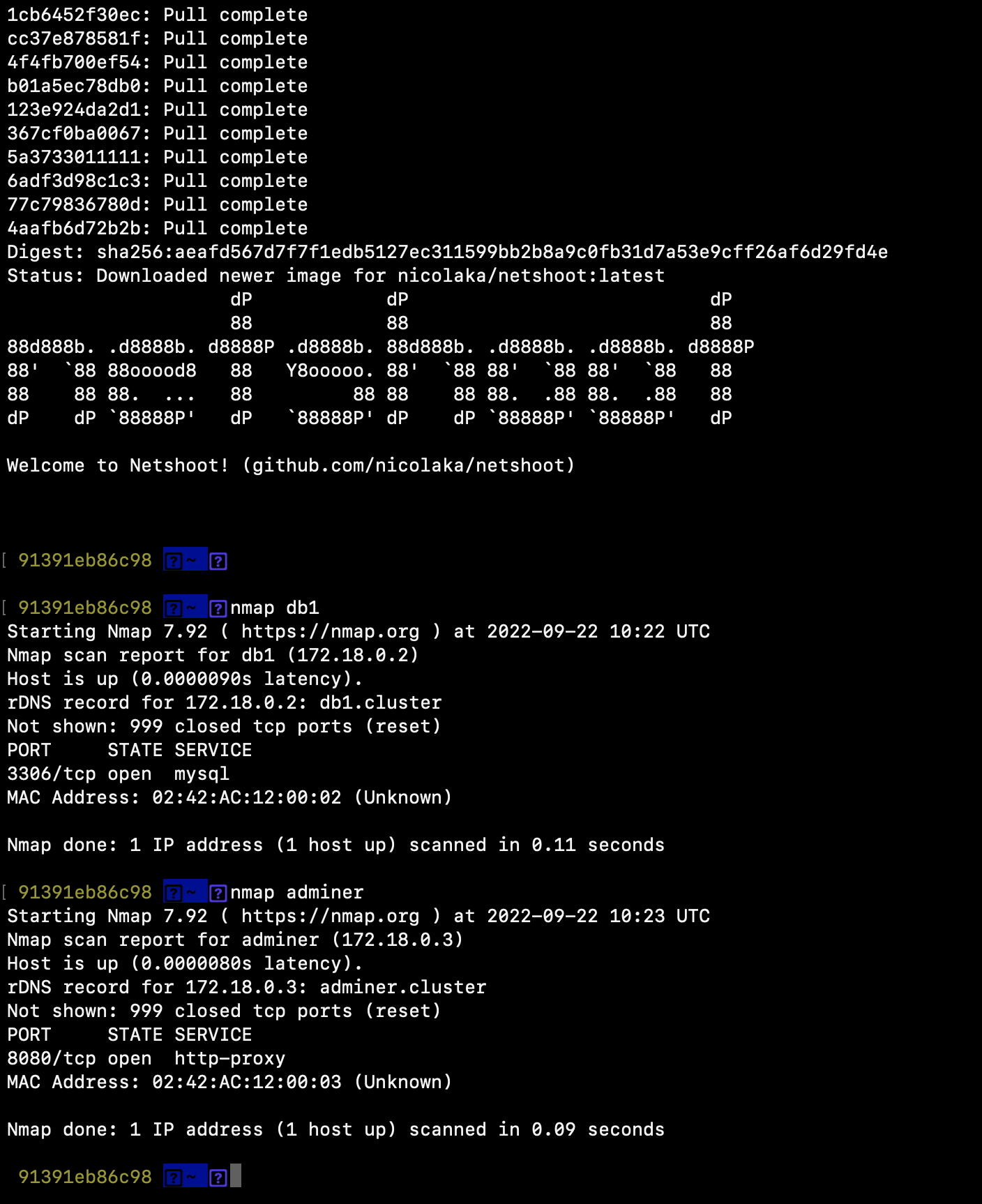


Кроме того, docker предоставляет dns-записи внутри своих сетей, поэтому мы можем обратиться по имени:



Для диагностики сетей есть [полезный образ](https://hub.docker.com/r/nicolaka/netshoot), подробное применение рассматривается по ссылке. Попробуем запустить его в интерактивном режиме и проверить сеть контейнеров с помощью nmap:

docker run -it --net cluster nicolaka/netshoot



Теперь выйдем и грохнем все контейнеры:

docker stop $(docker ps -a -q)