МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»

ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №8**

**«Знакомство с Docker Swarm»**

Практическая работа

по дисциплине «Современные технологии программирования»

студента 1 курса группы ПИ-б-о-231

Покидько Максим Сергеевич

направления подготовки 09.03.04 «Программная инженерия»

|  |  |  |
| --- | --- | --- |
|  |  |  |

Симферополь, 2024

**Цель:**

ознакомиться на практике с инструментом оркестрации контейнеризированных приложений Docker Swarm.

**Ход выполнения задания.**

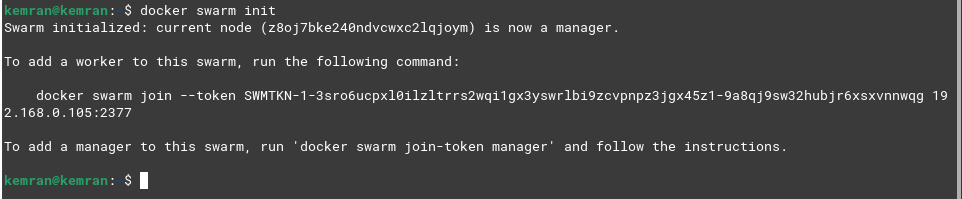
1. Изначально режим Docker Swarm отключён (inactive). Проверьте это выполнив команду:

docker info | grep Swarm



1. Чтобы переключиться в режим Docker Swarm и инициализировать текущий узел (node) как менеджер выполите команду:

docker swarm init

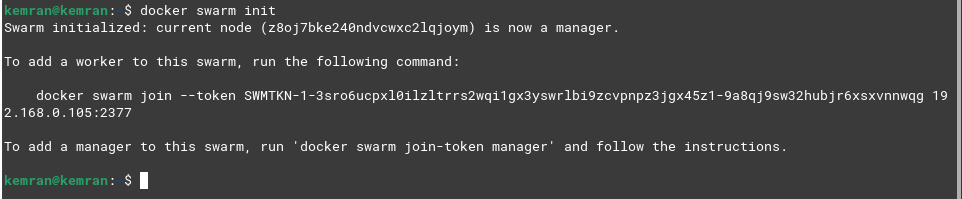


В результате вы увидите идентификатор текущего узла и команду позволяющую другим машинам подключиться к текущему кластеру в роли рабочего узла (worker node). Машину так же можно подключить в роли менеджера. Как видно, в команде указан токен доступа и сокет для подключения. Любой, у кого есть эта информация, сможет подключиться к кластеру, поэтому токен должен держаться в секрете.

1. Чтобы посмотреть команду подключения позже, введите:

docker swarm join-token worker

docker swarm join-token manager



1. Убедимся, что Docker переключился в режим роя (swarm):

docker info | grep Swarm





1. Получить информацию по всем подключённым к кластеру узлам можно командой:

docker node ls





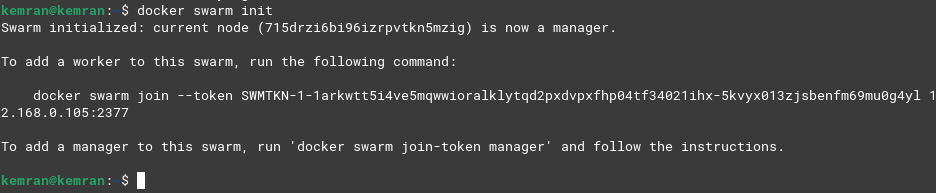
1. Чтобы покинуть кластер и выйти из режима Swarm введите:

docker swarm leave —force

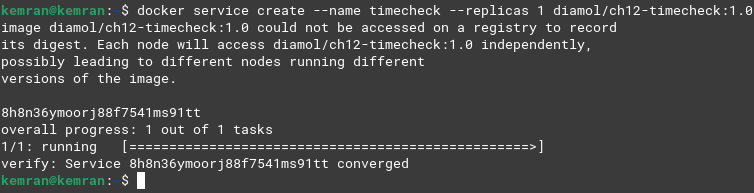




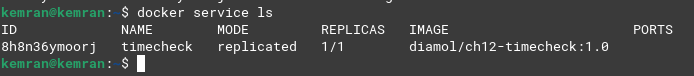
1. Активируйте режим Swarm;



1. Swarm, вместо отдельных контейнеров оперирует сервисами (service). Запустим сервис вручную:

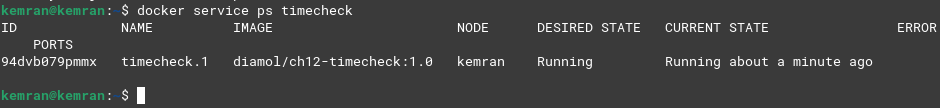
docker service create --name timecheck --replicas 1 diamol/ch12-timecheck:1.0

1. Посмотрим список сервисов запущенных в кластере:

docker service ls

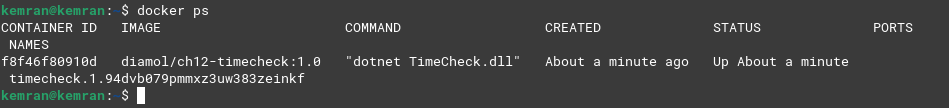
1. Посмотрим список контейнеров запущенных сервисом timecheck:

docker service ps timecheck



1. Т.к. в кластере, сейчас присутствует только один узел, то и контейнер запущен на нём. Проверим это при помощи команды, которая покажет список контейнеров работающих на текущей машине:

docker ps



1. Принудительно (-f) удалим этот контейнер, чтобы проверить, как поведёт себя кластер, в случае внезапного падения контейнера:

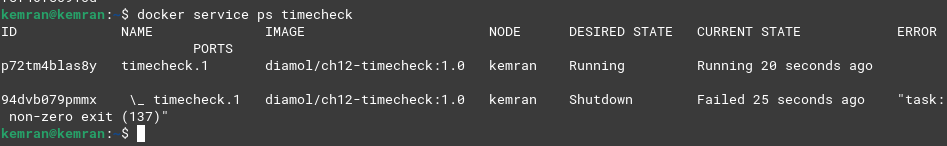
docker rm -f $(docker ps --last 1 -q)





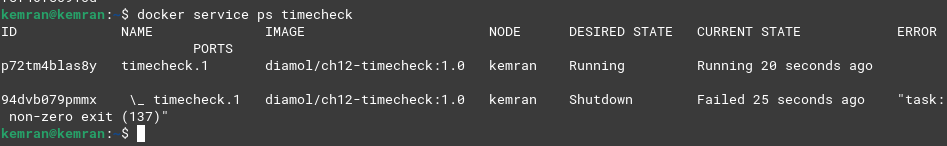
1. Снова проверим контейнеры сервиса timecheck:

docker service ps timecheck



1. Посмотреть логи сервиса можно командой:

docker service logs --since 10s timecheck



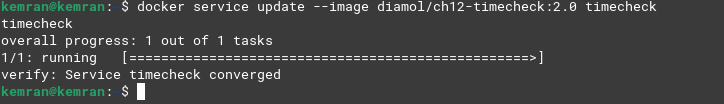
1. Получить информацию по сервису можно командой:

docker service inspect timecheck



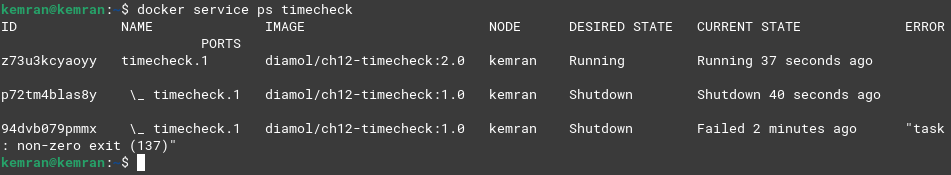
1. В уже запущенный сервис можно вносить изменения:

docker service update --image diamol/ch12-timecheck:2.0 timecheck



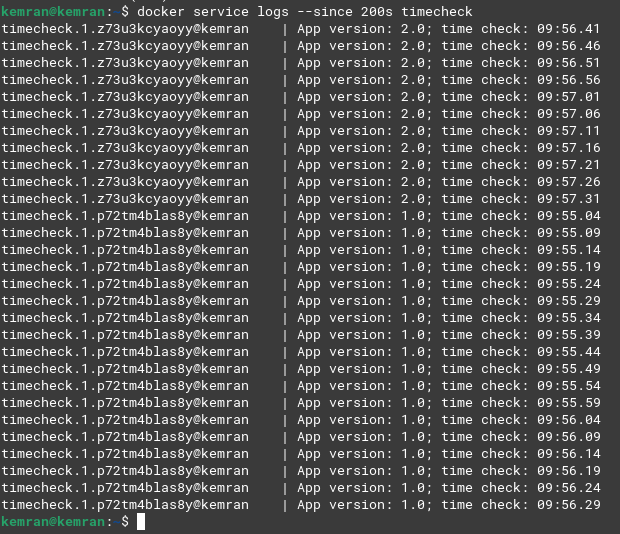
1. Проверим список контейнеров сервиса:

docker service ps timecheck



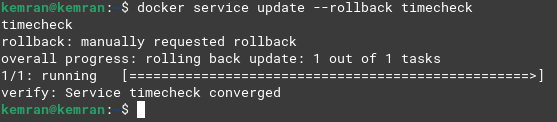
1. Если заглянуть в логи достаточно далеко назад, то можно обнаружить момент, когда произошла замена контейнера:

docker service logs --since 200s timecheck



1. В некоторых случаях возникает необходимость вернуться назад к предыдущей версии. Swarm может это сделать, при этом нет необходимости указывать старую конфигурацию, т.к. она сохранена в базе данных кластера. Достаточно выполнить команду:

docker service update --rollback timecheck

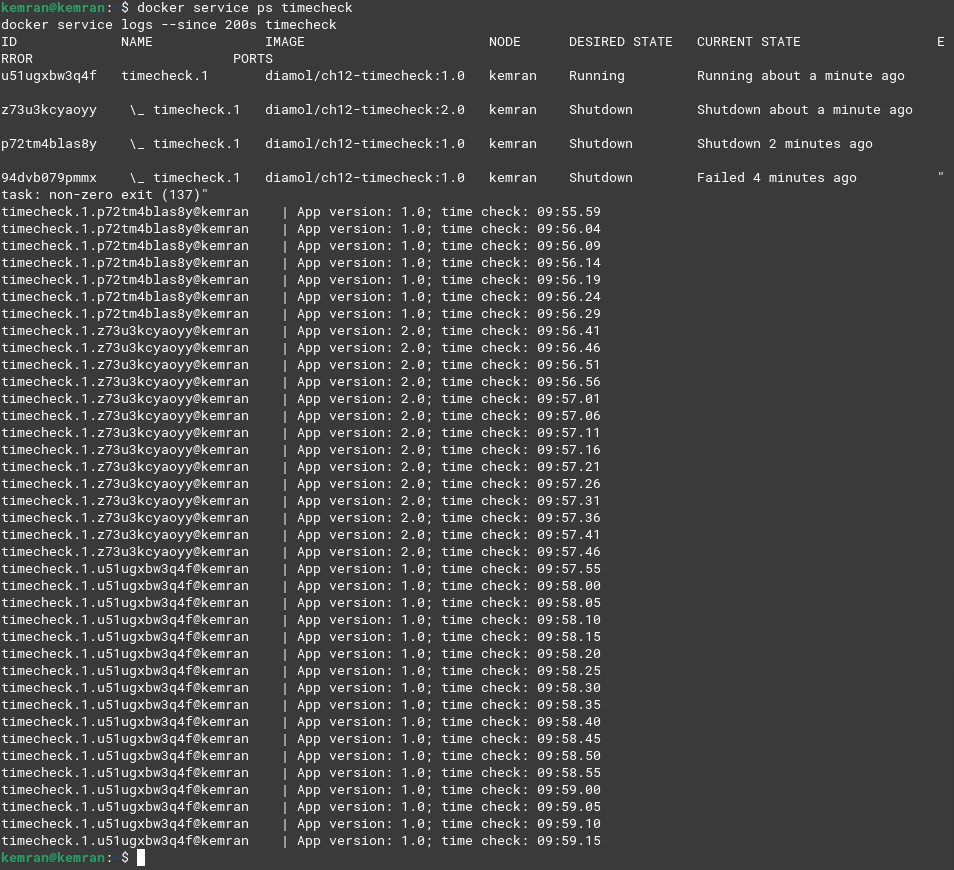




1. Снова проверим список контейнеров логи:

docker service ps timecheck

docker service logs --since 200s timecheck



1. Этот сервис нам больше не нужен, поэтому его можно удалить:

docker service rm timecheck





1. Создадим сеть типа overlay вручную:

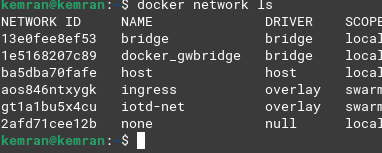
docker network create --driver overlay iotd-net





1. Проверьте, что новая сеть появилась в списке:

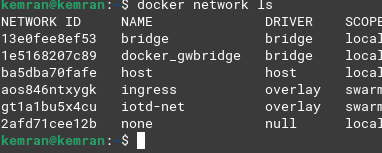
docker network ls





1. Запустим новый сервис и подключим его к сети с названием iotd-net

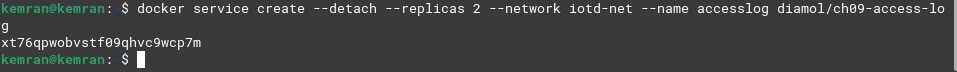
docker service create --detach --replicas 3 --network iotd-net --name iotd diamol/ch09-image-of-the-day





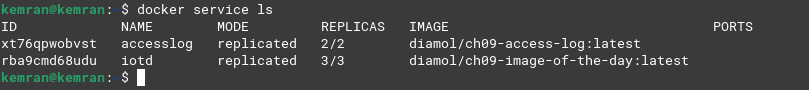
1. Запустим ещё один сервис и назовём его accesslog:

docker service create --detach --replicas 2 --network iotd-net --name accesslog diamol/ch09-access-log



1. Посмотрим список сервисов:

docker service ls



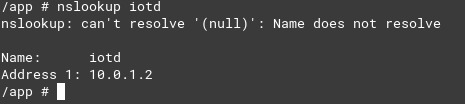
1. Зайдём в контейнер сервиса accesslog. Точнее в какой-то из его двух контейнеров, который запустился последним:

docker exec -it $(docker ps --last 1 -q) sh



1. Находясь в контейнере выполним команду nslookup, которая позволит получить список ip адресов соответствующих указанному доменному имени:

nslookup iotd

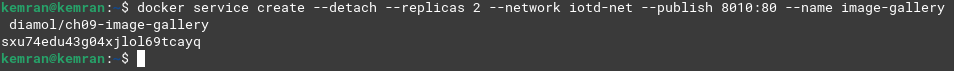


1. Отключитесь от контейнера;



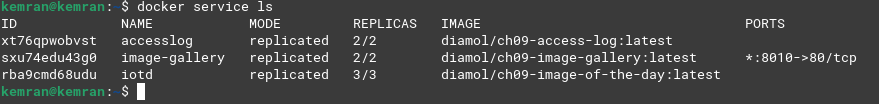
1. Запустим третий сервис:

docker service create --detach --replicas 2 --network iotd-net --publish 8010:80 --name image-gallery diamol/ch09-image-gallery

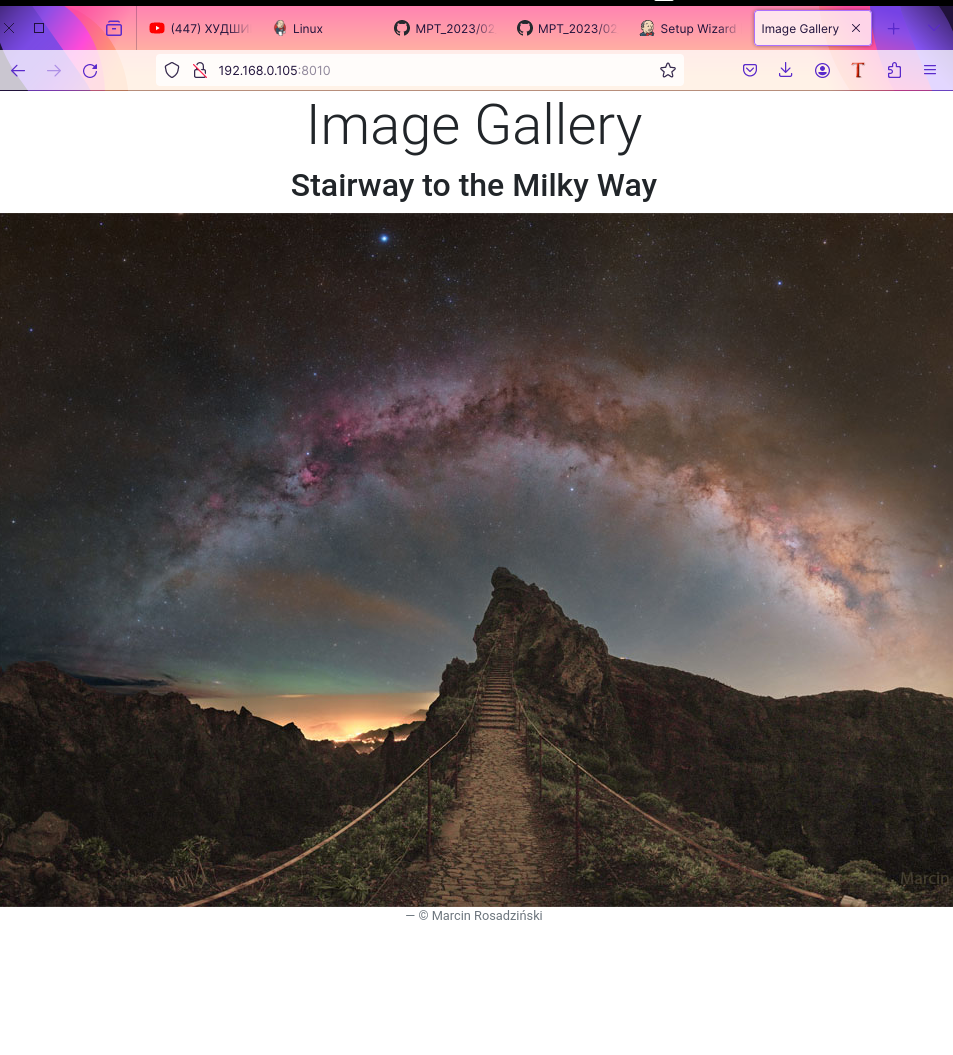


1. Посмотрим список сервисов:

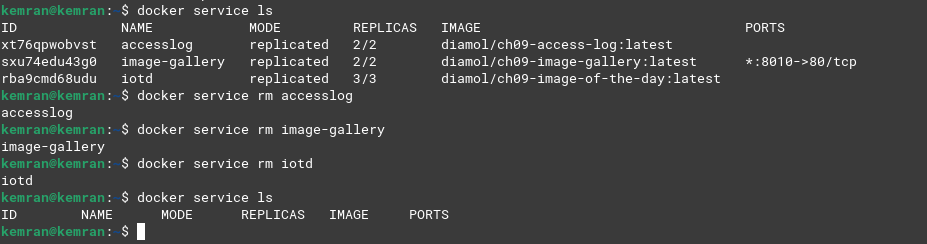
docker service ls



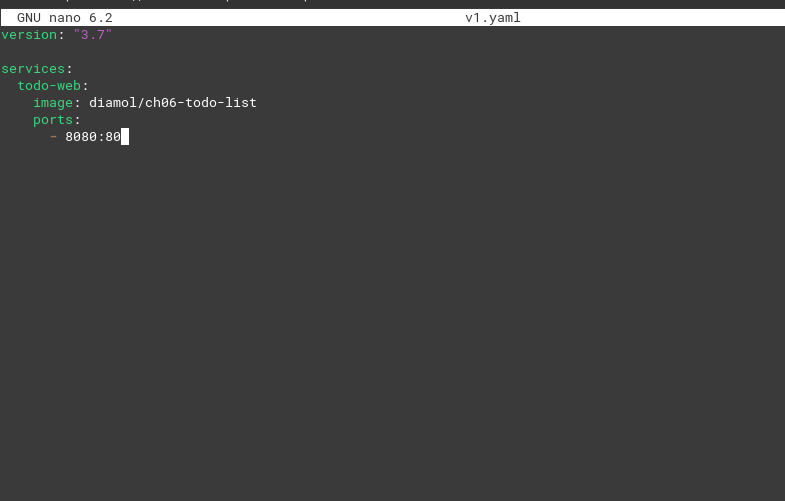
1. Откройте браузер на хостовой машине и введите в строку адреса ip-адрес виртуальной машины и порт 8010. Обновите вкладку несколько раз.  
   В результате вы должны увидеть изображение с серверов NASA на космическую тематику. Изображение обновляется один раз в день.



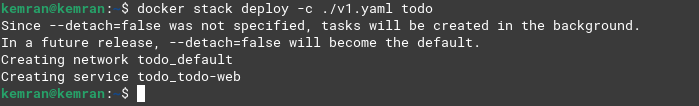
1. Остановите все запущенные сервисы.



1. Сохраните приведённый выше текст в файл "v1.yaml".

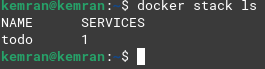


1. Чтобы развернуть описанное данным манифестом приложение в кластере выполните команду:

docker stack deploy -c ./v1.yaml todo  


1. Посмотрим список стеков:

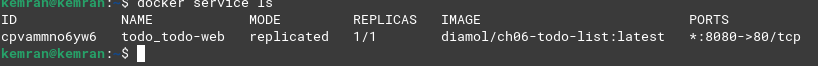
docker stack ls



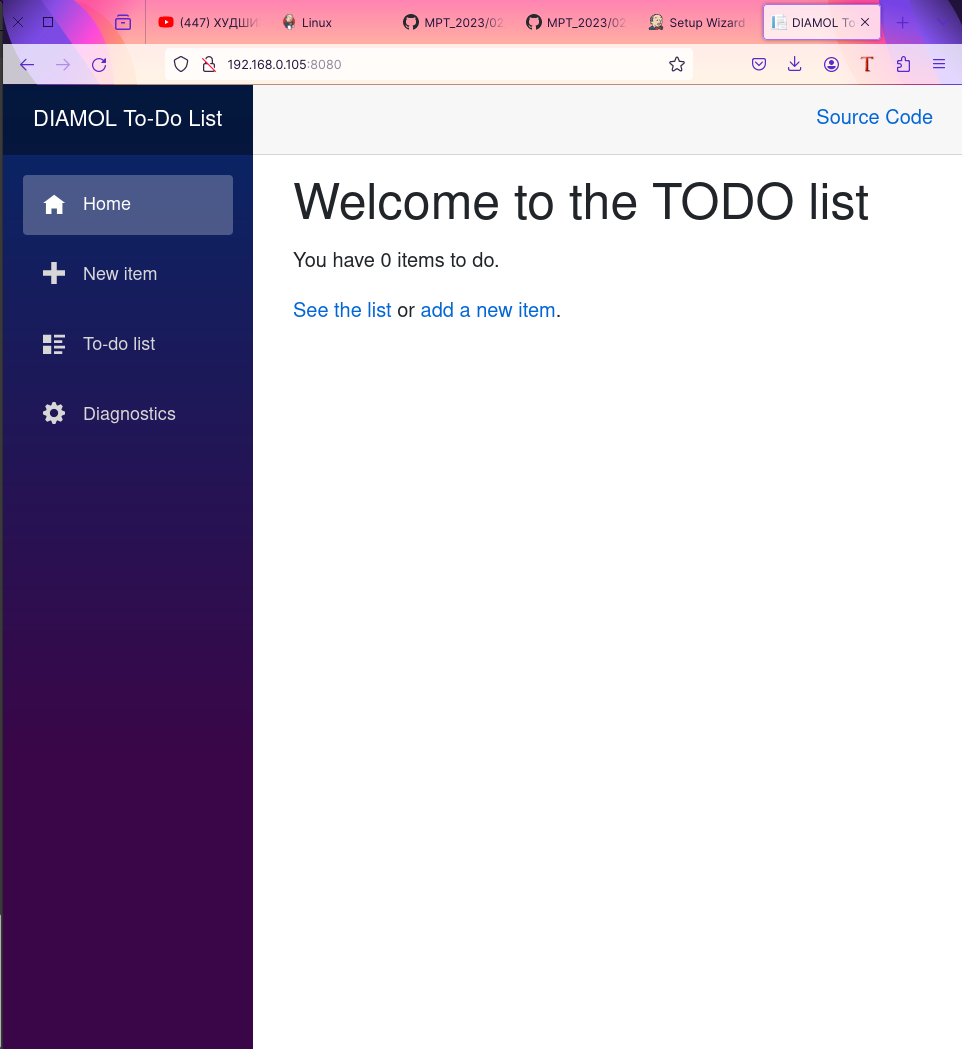


1. Посмотрим список сервисов работающих в кластере:

0docker service ls



1. Откройте браузер на хостовой машине и введите в строку адреса ip-адрес виртуальной машины и порт 8080. Вы должны увидеть веб интерфейс приложения;



1. Создайте файл "v2.yaml" содержащий:

version: "3.7"

services:

todo-web:

image: diamol/ch06-todo-list

ports:

- 8080:80

deploy:

replicas: 2

resources:

limits:

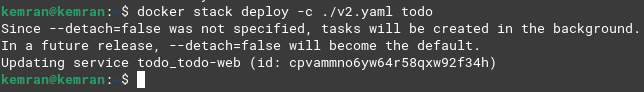
cpus: "0.50"

memory: 100M



1. Выполните команду:

docker stack deploy -c ./v2.yaml todo



1. Чтобы посмотреть список сервисов стека воспользуемся командой:

docker stack services todo



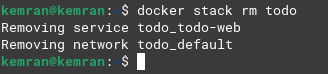
1. Посмотрим на список контейнеров этого сервиса:

docker service ps todo\_todo-web



1. Удалим стек "todo":

docker stack rm todo





1. Создайте каталог "configs" и в нем файл "config.json" содержащий:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Warning"

}

},

"AllowedHosts": "\*",

"Database": {

"Provider": "Postgres"

}

}



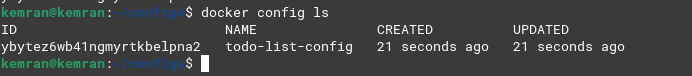
1. Создадим объект конфигурации Docker:

docker config create todo-list-config ./congif.json



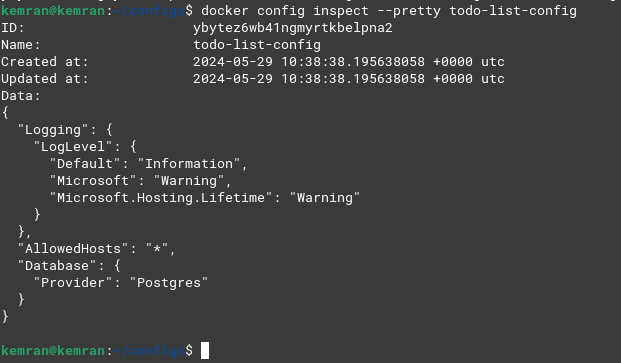
1. Посмотрим список доступных конфигов:

docker config ls



1. Посмотрим содержимое конфига:

docker config inspect --pretty todo-list-config



1. Выйдите из каталога "configs" и создайте файл "v3.yaml" содержащий:

version: "3.7"

services:

todo-web:

image: diamol/ch06-todo-list

ports:

- 8080:80

configs:

- source: todo-list-config

target: /app/config/config.json

deploy:

replicas: 1

resources:

limits:

cpus: "0.50"

memory: 100M

networks:

- app-net

todo-db:

image: diamol/postgres:11.5

deploy:

replicas: 1

resources:

limits:

cpus: "0.50"

memory: 500M

networks:

- app-net

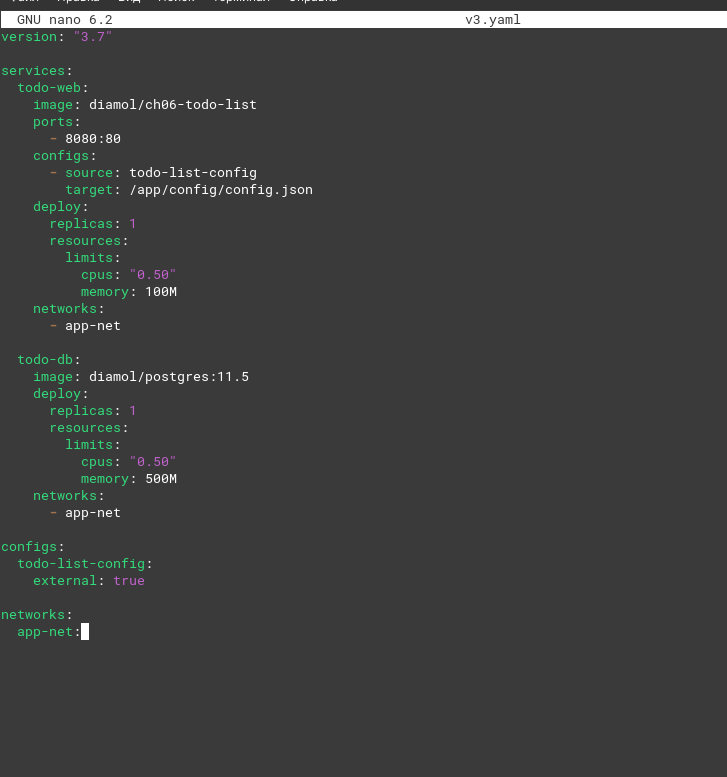
configs:

todo-list-config:

external: true

networks:

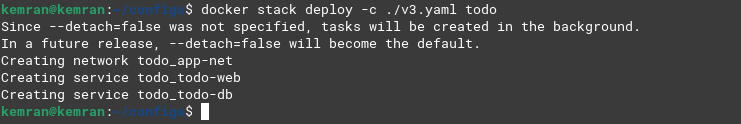
app-net:

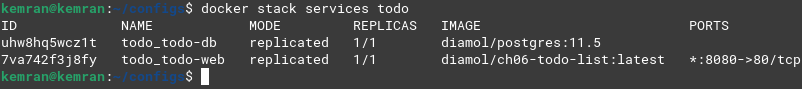


1. Обновим приложение:

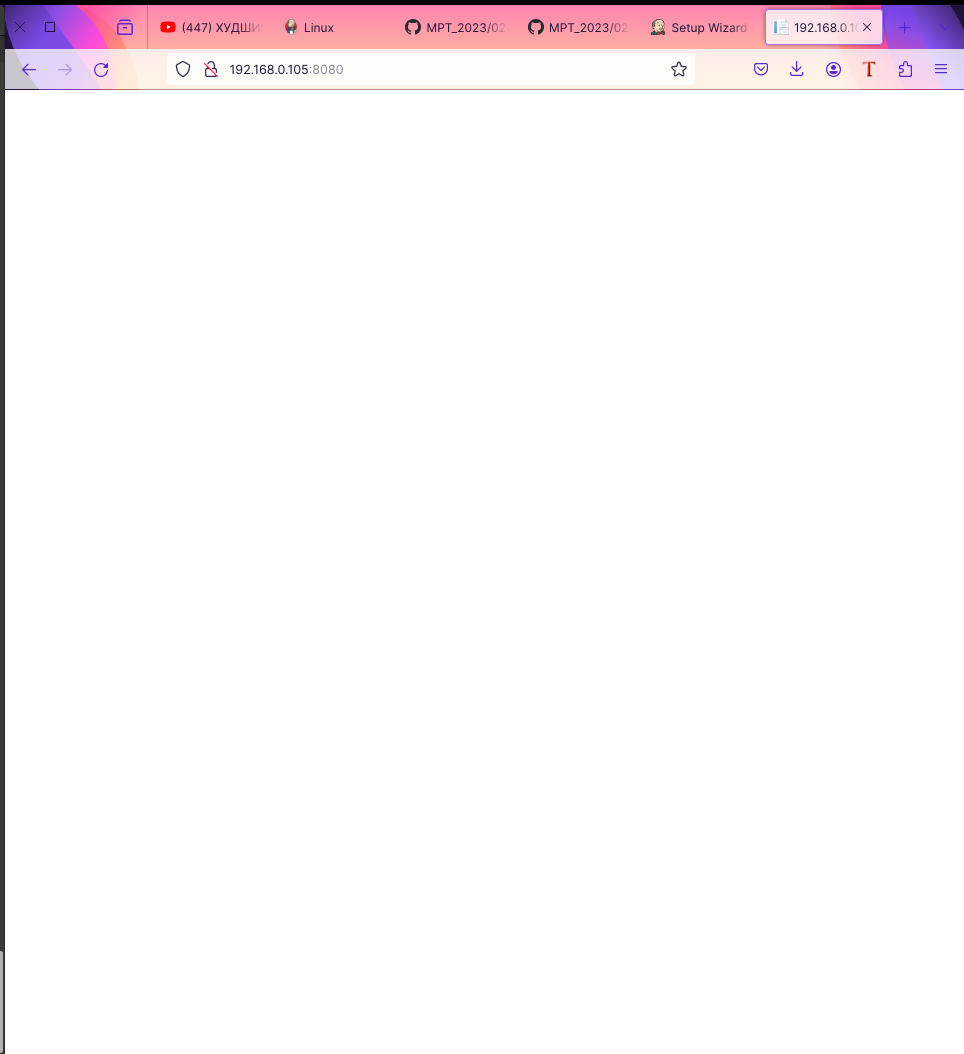
docker stack deploy -c ./v3.yaml todo



1. Убедитесь, что оба сервиса запущены и в каждом работает по одному контейнеру:

docker stack services todo

1. Откройте браузер на хостовой машине и введите в строку адреса ip-адрес виртуальной машины и порт 8080. Сейчас приложение должно быть не доступно;



1. Если изучить логи приложения (сервис "todo\_todo-web"):

docker service logs todo\_todo-web



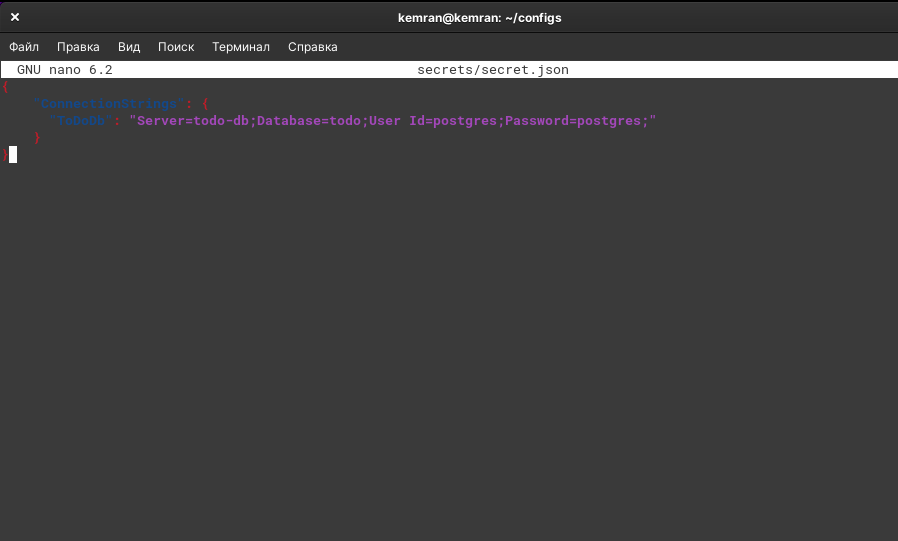
1. Создайте каталог "secrets" и в нём файл "secret.json" содержащий:

{

"ConnectionStrings": {

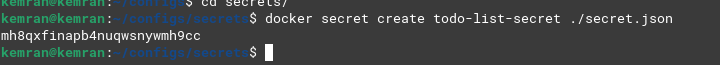
"ToDoDb": "Server=todo-db;Database=todo;User Id=postgres;Password=postgres;"

}

}

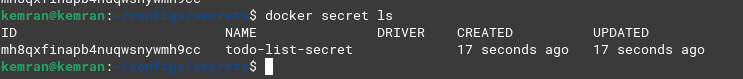
1. Создадим объект секрета Docker:

docker secret create todo-list-secret ./secret.json



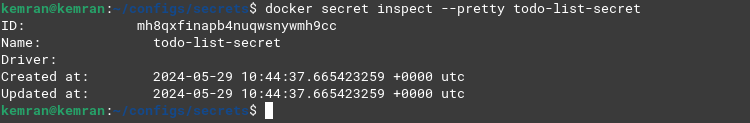
1. Посмотрим список доступных секретов:

docker secret ls



1. Посмотрим содержимое секрета:

docker secret inspect --pretty todo-list-secret



1. Выйдите из каталога "secrets" и создайте файл "v4.yaml" содержащий:

version: "3.7"

services:

todo-web:

image: diamol/ch06-todo-list

ports:

- 8080:80

configs:

- source: todo-list-config

target: /app/config/config.json

secrets:

- source: todo-list-secret

target: /app/config/secrets.json

deploy:

replicas: 1

resources:

limits:

cpus: "0.50"

memory: 100M

networks:

- app-net

todo-db:

image: diamol/postgres:11.5

deploy:

replicas: 1

resources:

limits:

cpus: "0.50"

memory: 500M

networks:

- app-net

configs:

todo-list-config:

external: true

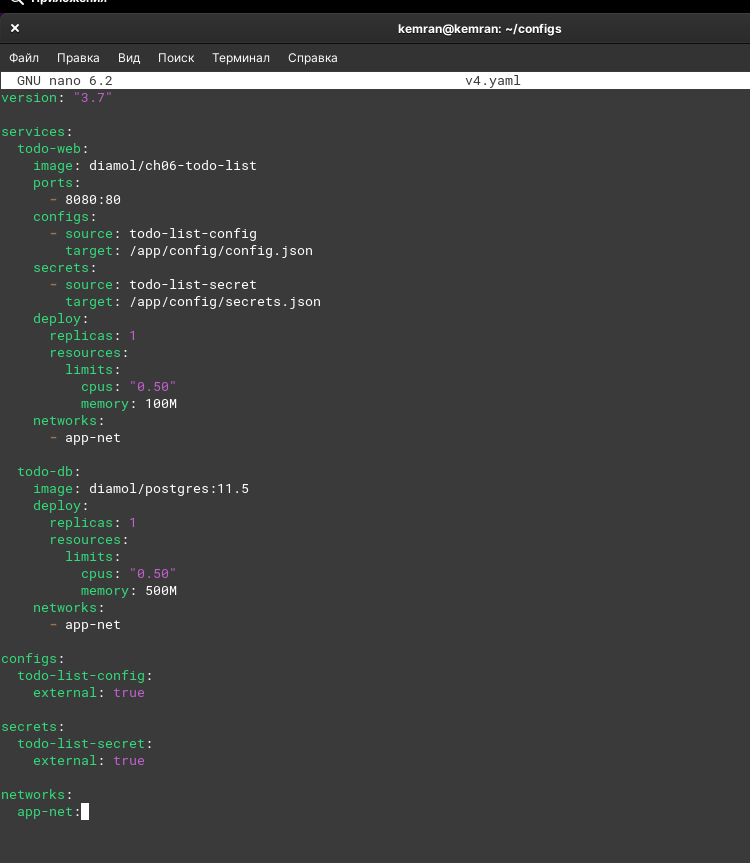
secrets:

todo-list-secret:

external: true

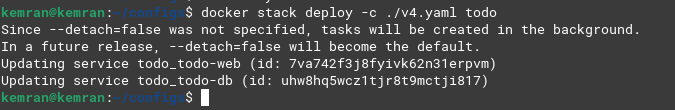
networks:

app-net:



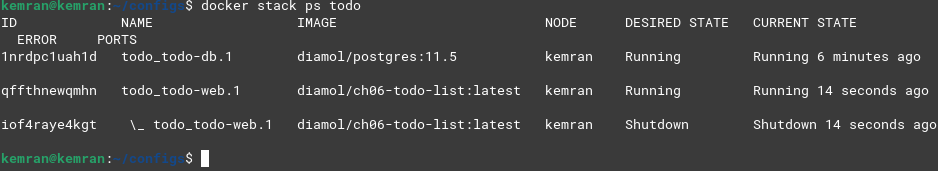
1. Обновим приложение:

docker stack deploy -c ./v4.yaml todo

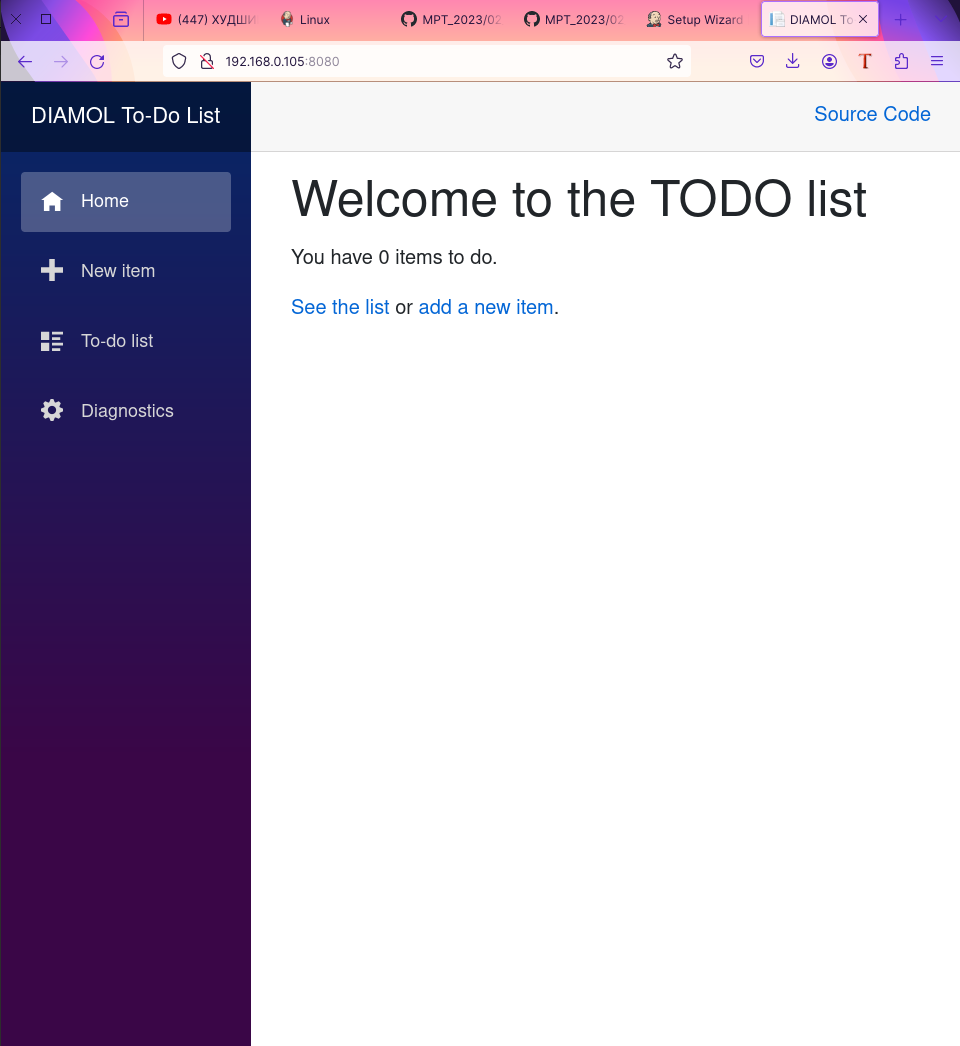


1. Посмотрим контейнеры стека "todo":

docker stack ps todo



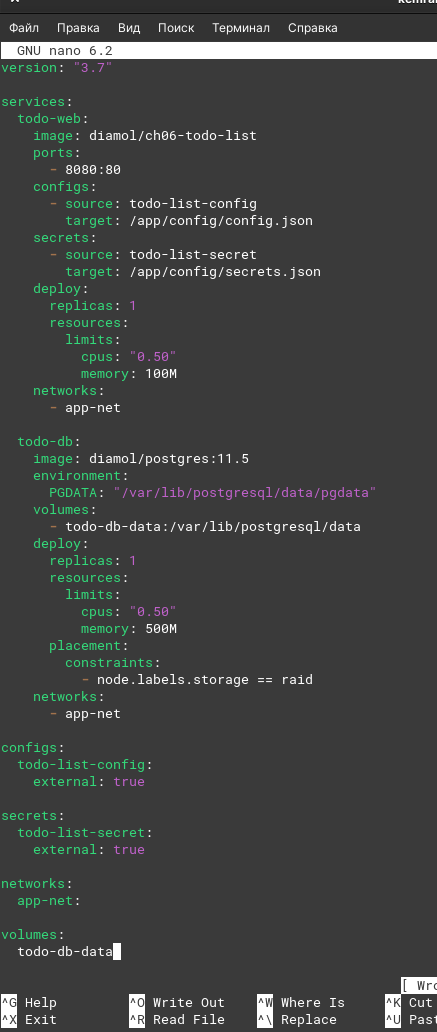
1. Откройте браузер на хостовой машине и введите в строку адреса ip-адрес виртуальной машины и порт 8080. Теперь приложение работает корректно и описывается схемой представленной на рисунке.



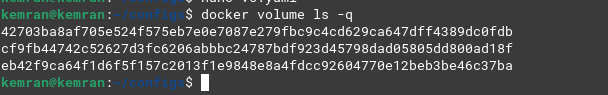
1. Выполните команду:
2. Список меток узла можно найти в разделе "Spec" -> "Labels":



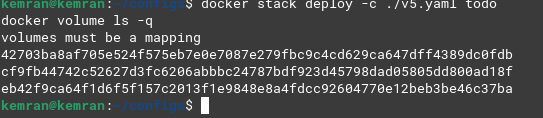
1. Создайте файл "v5.yaml" содержащий:



1. Посмотрите на имена томов, которые у вас уже существуют:

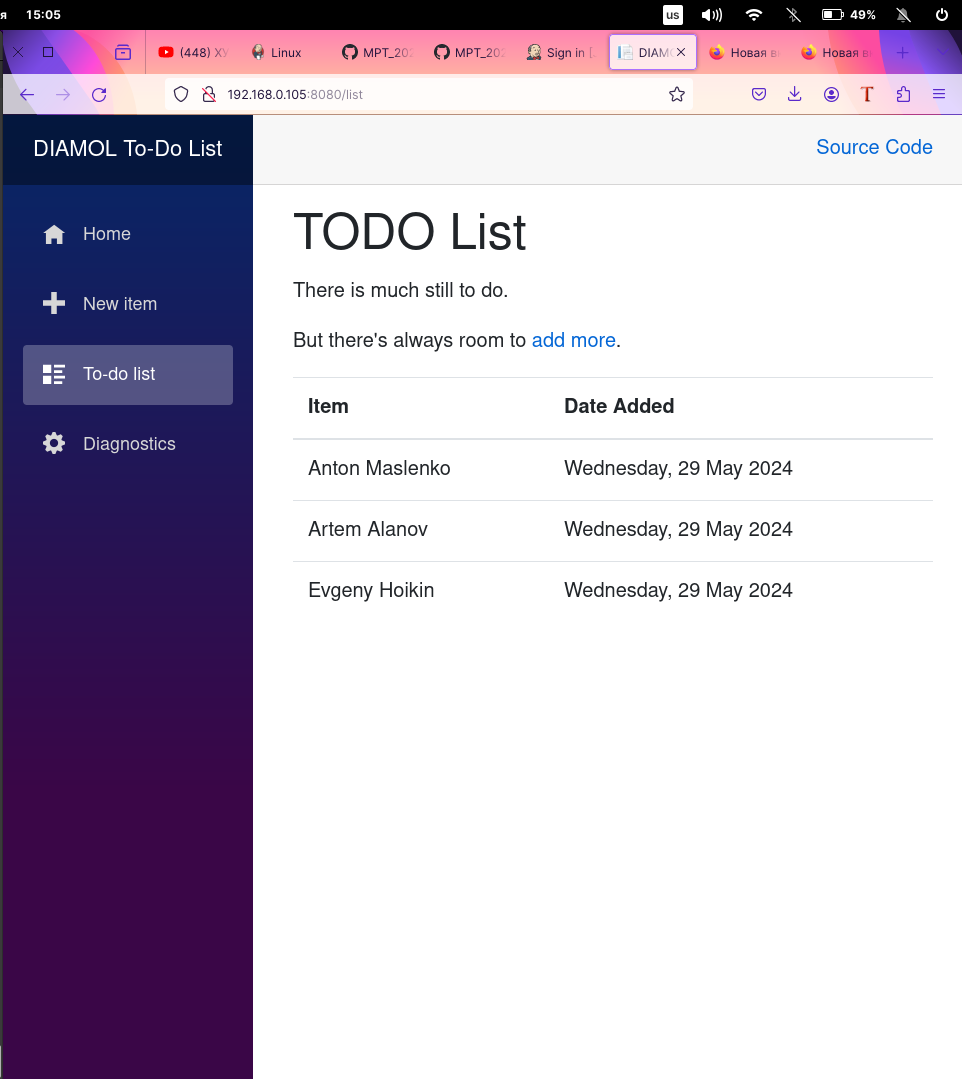


1. Обновите приложение и снова проверьте список томов:

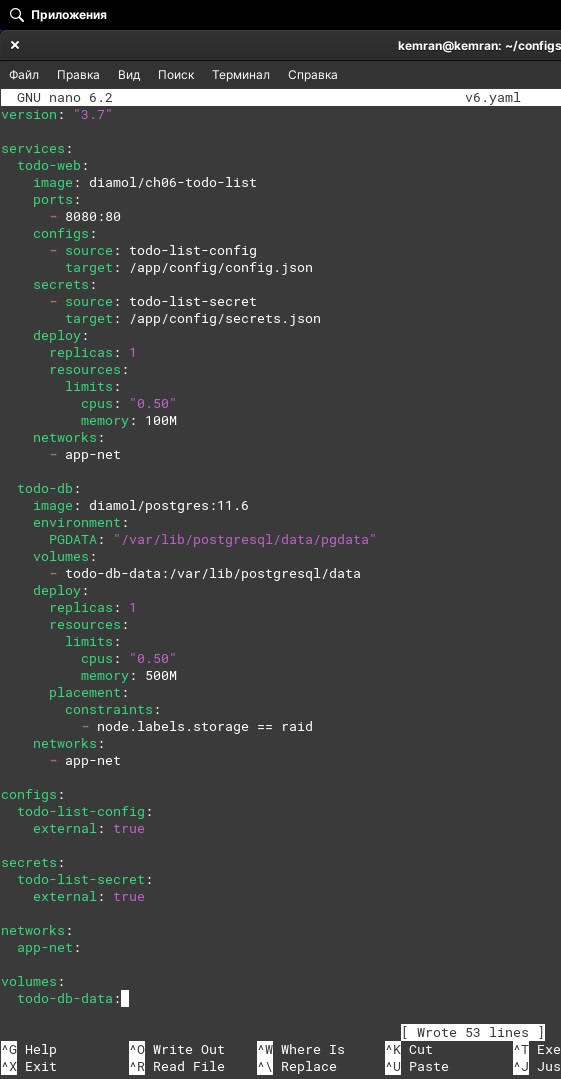




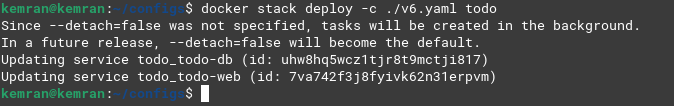
1. Перейдите в браузер и добавьте несколько элементов в список дел;



1. На сервере создайте файл "v6.yaml" содержащий:

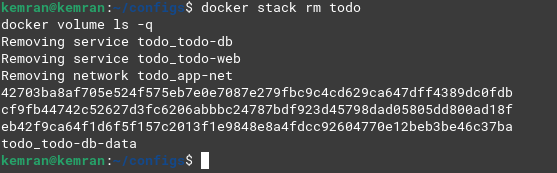


1. Обновите приложение:



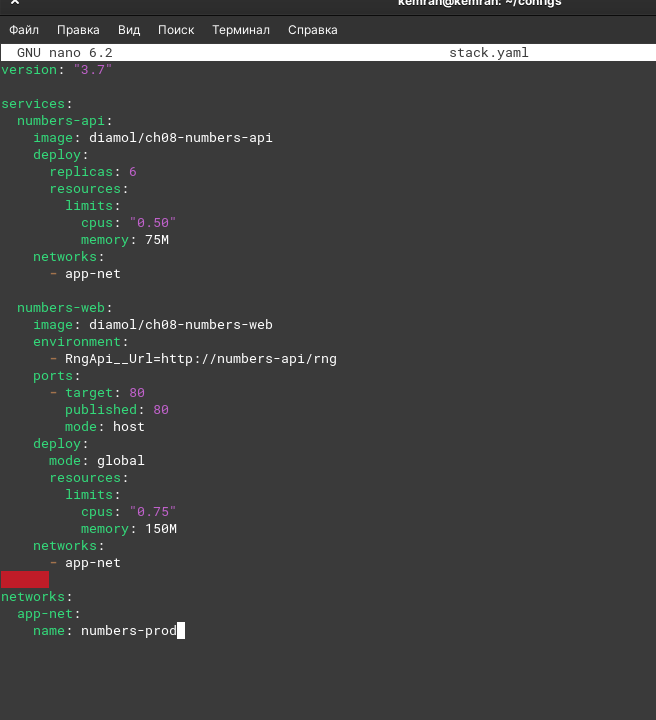
1. Перейдите в браузер и проверьте, что составленный вами список дел всё ещё на месте. Даже если бы в кластере было больше одного узла, контейнер с базой данных всё равно запустился бы на этом, т.к. только у него есть метка storage=raid.  
   Текущая версия приложения описывается схемой представленной на рисунке.
2. Удалите стек и проверьте список томов:



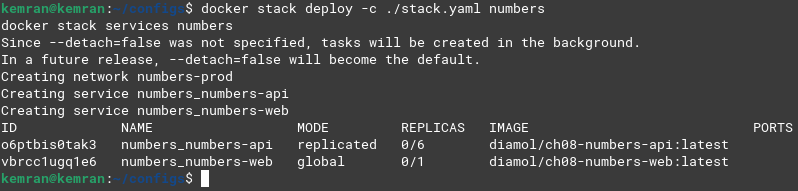




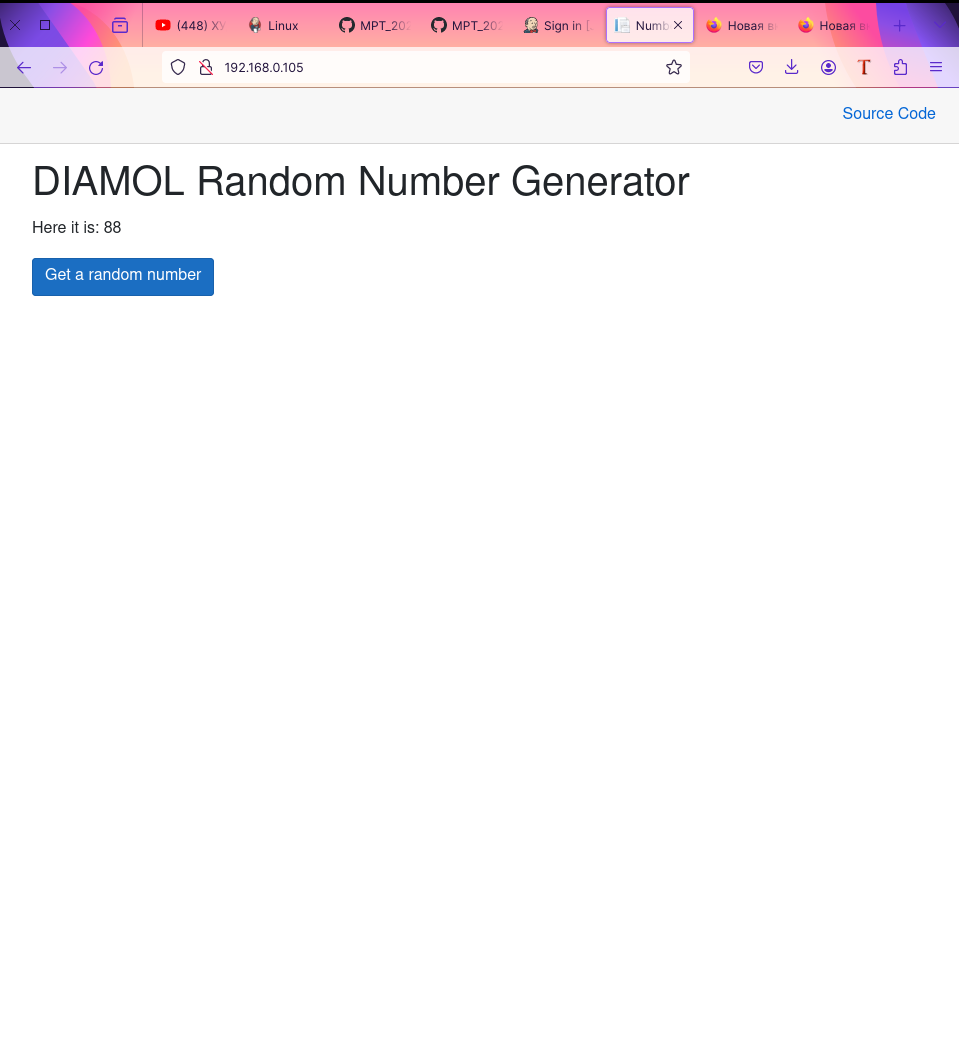
1. Создайте файл "stack.yaml" содержащий:



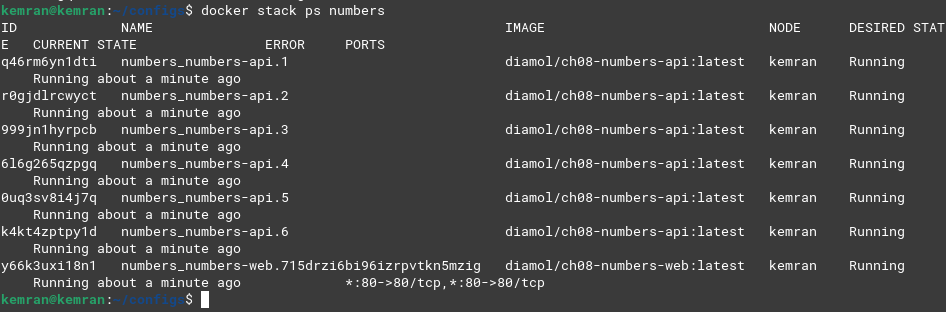
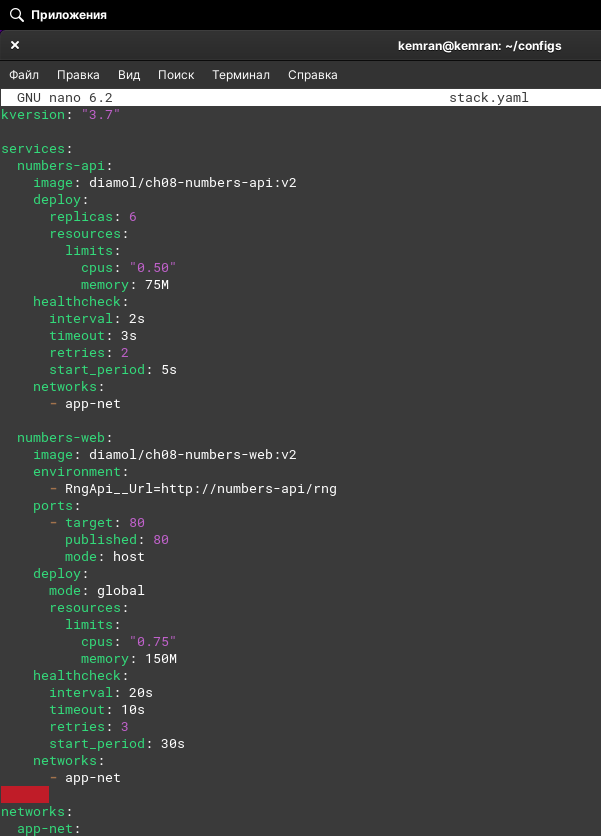
1. Запустите приложение и проверьте состояние сервисов текущего стека:



1. Откройте браузер на хостовой машине и введите в строку адреса ip-адрес виртуальной машины и порт 80. Нажмите несколько раз на кнопку генерации случайных чисел, пока не получите сообщение "RNG service unavailable!". Теперь, сколько бы вы ни нажимали кнопку сообщение не изменится;



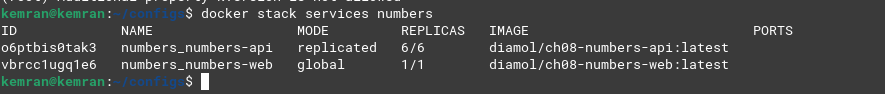
1. Посмотрите список контейнеров стека:



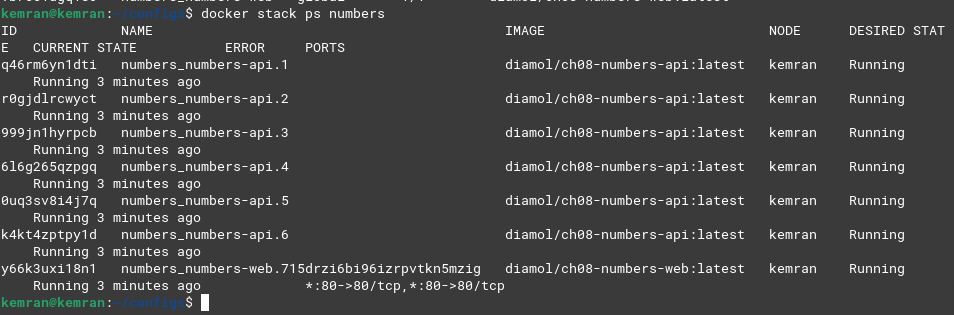
1. Откройте файл "stack.yaml" и замените его содержимое на:
2. Обновите приложение:



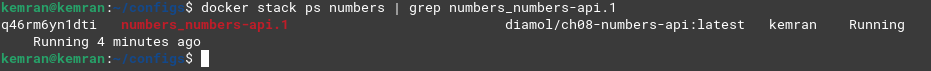
1. Выполните команду:



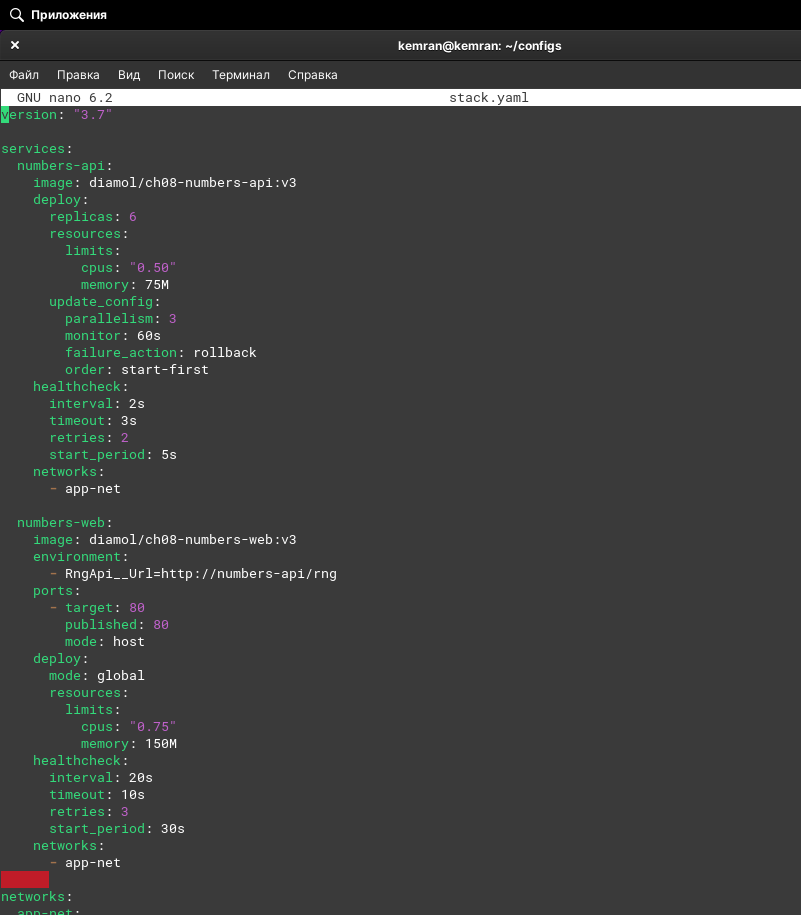
1. Посмотрите список контейнеров стека:



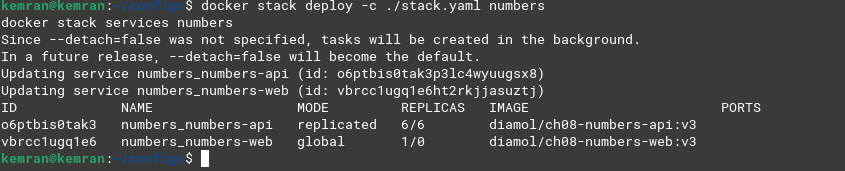
1. Перейдите в браузер, обновите страницу и снова понажимайте на кнопку "Get a random number" до тех пор, пока не получите сообщение "RNG service unavailable!". После этого нажмите на эту кнопку ещё 6+ раз.  
   После небольшого ожидания снова начните нажимать на кнопку генерации случайных чисел. Вы увидите, что приложение опять работает нормально;
2. Снова посмотрите список контейнеров стека:



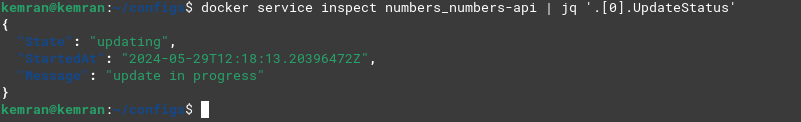
1. Откройте файл "stack.yaml" и замените его содержимое на:



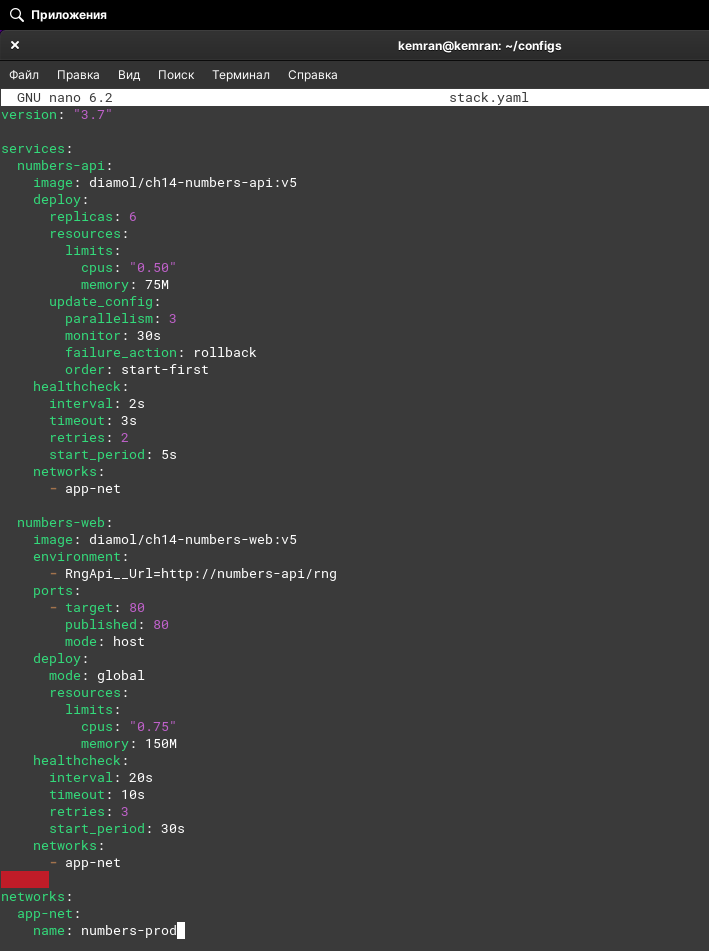
1. Выполните команды:

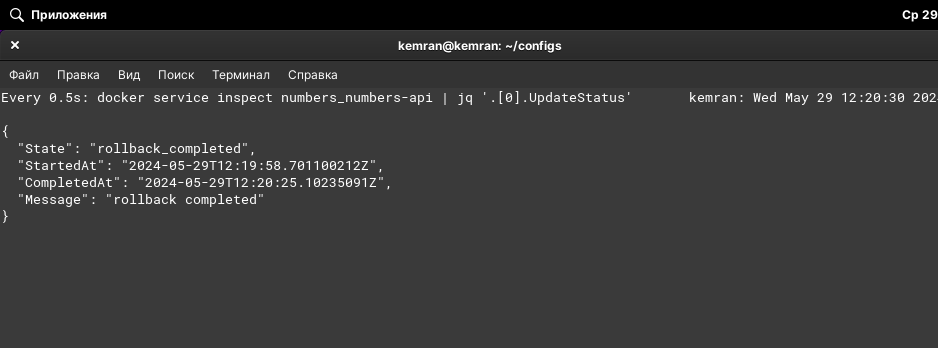


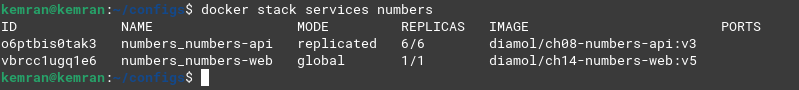
1. Примерно через 3 минуты после начала обновления выполните команду:



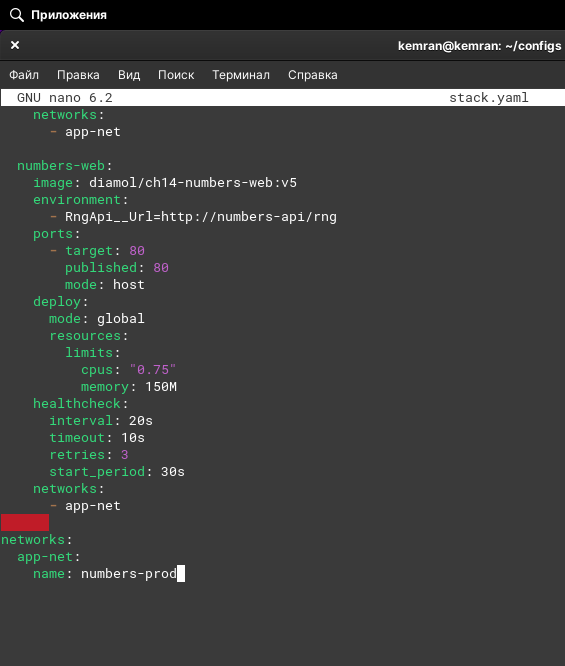
1. Откройте файл "stack.yaml" и замените его содержимое на:



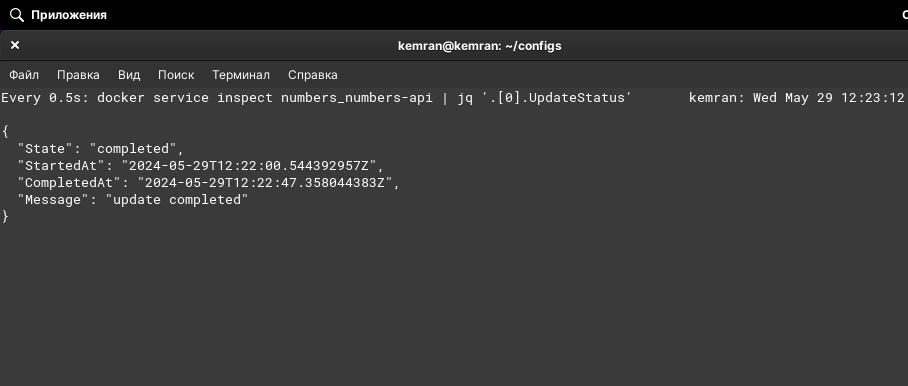
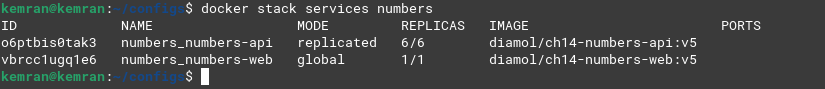
1. Выполните команды:
2. Выполните команду:



1. Откройте файл "stack.yaml" и замените его содержимое на:



1. Выполните команды:
2. Выполните команду:



**Ответы на вопросы.**

[Ваш текст здесь]

**Вывод:**

Я ознакомился на практике с инструментом оркестрации контейнеризированных приложений Docker Swarm.