Для прохождения практики тебе потребуется 2-3 машины с linux и docker на борту.

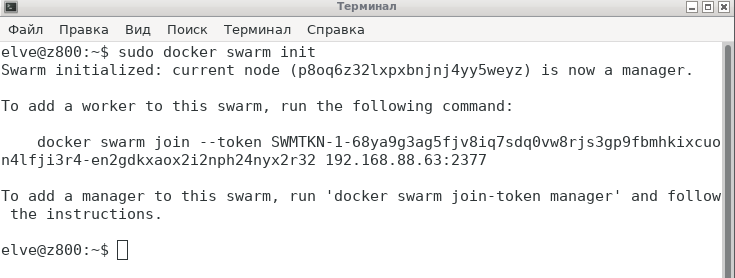
Сборка кластера идет последовательно.

Сначала инициализируем кластер на первой мастер-ноде. Она будет главной.

А потом добавляем остальные ноды.

Инцициализация кластера делается вот так:

sudo docker swarm init



Если у тебя на компьютере несколько интерфейсов и ты хочешь конкретный закрепить за кластером, то команда будет чуть длинее:

sudo docker swarm init --advertise-addr твой\_IP-address

Дальше

Как видишь, при создании кластера/роя выводится подсказка с командой. Ее надо вводить на остальных нодах, чтобы добавить их в рой/кластер.

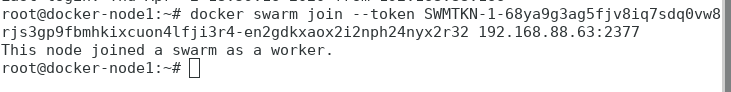
Если ты случайно закрыл окно и забыл что там было написано, то получить это сообщение можно с помощью команды:

sudo docker swarm join-token manager

Дальше

Заходи на вторую ноду и подключай к рою.

sudo docker swarm join --token SWMTKN-1-68ya9g3ag5fjv8iq7sdq0vw8rjs3gp9fbmhkixcuon4lfji3r4-en2gdkxaox2i2nph24nyx2r32 192.168.88.63:2377



И так по аналогии добавляй остальных членов роя. Команда будет для всех одна и та же. Она указывает ip менеджера и токен для авторизации.

У меня будут две ноды:

**z800**

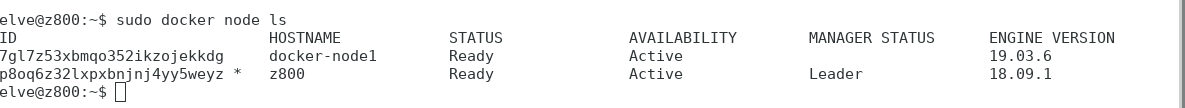
**docker-node1**

Дальше

Посмотреть состояние кластера и список подключенных **нод** можно командой:

sudo docker node ls

Выполнять ее нужно на главной **ноде** (на той где мы делали **init** ).



Дальше

Думаю, что у тебя все получилось с первого раза.

Но если вдруг ты столкнулся с проблемой добавления ноды, то проблема может заключаться в некорректных настройках файрвола.

Docker Swarm для своей работы использует следующие порты:

**2377/tcp**

**7946/tcp**

**7946/udp**

**4889/udp**

Эти порты должны быть доступны от каждой ноды к любой другой.

Дальше

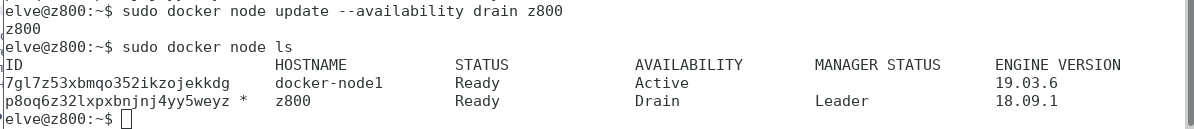
И до запуска сервисов хочу рассказать о еще одной возможности.

Мастер-нода по умолчанию не отлынивает от работы и выполняет все те же задачи, что и подчиненные ноды.

Но можно принудительно вывести ее в чистые начальники. Нода будет заниматься управлением, но ни один контейнер на ней выполняться не будет.

Делается это так:

docker node update --availability drain z800



Чтобы опять сделать мастер-ноду работягой можно выполнить вот такую команду:

sudo docker node update --availability active z800

Если нода хочет выйти из кластера, то на ней надо выполнить команду:

sudo docker swarm leave

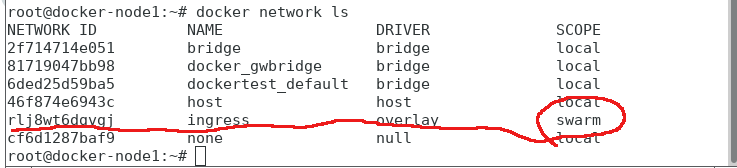
Дальше

Теперь пару слов о сети. Ты же помнишь, что для контейнеров на одной машине сеть делается с помощью **bridge**?

Так вот. Для кластеров придуман свой тип сети - **overlay**.

При инициализации кластера такая сеть создается автоматически. И добавляется на каждую **ноду**, при ее подключении к кластеру.

docker network ls



Дальше

Вот теперь все готово для запуска первого контейнера в кластере - т.е. сервиса.

sudo docker service create --replicas 1 --name web nginx

Что делает эта команда?

Естественно создает сервис =). Со следующими параметрами:

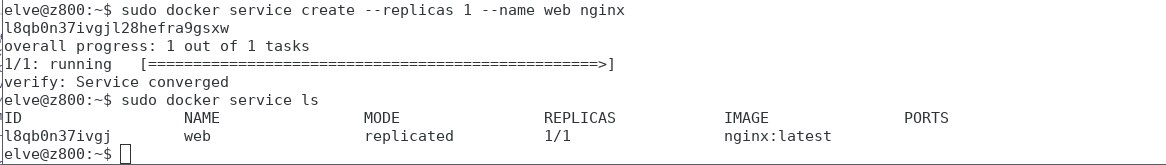
- количество реплик (т.е. на каком количестве нод будет запущен сервис) = **1**

- имя сервиса = **web**

- образ из которого будет собран сервис = **nginx**

А чтобы посмотреть список запущенных сервисов, надо ввести вот такую команду:

sudo docker service ls

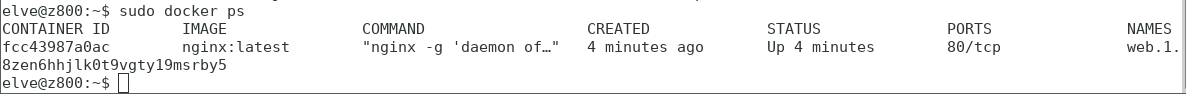


Дальше

Через

sudo docker ps

контейнер, принадлежащий сервису тоже виден



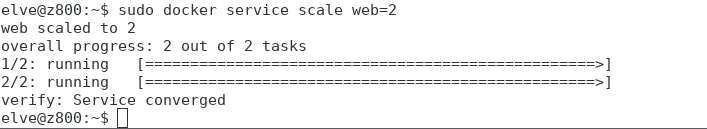
Дальше

Вот запустили мы с тобой сервис на одну **реплику**. А что если теперь хочется его расширить на две реплики, но так, чтобы не выключать?

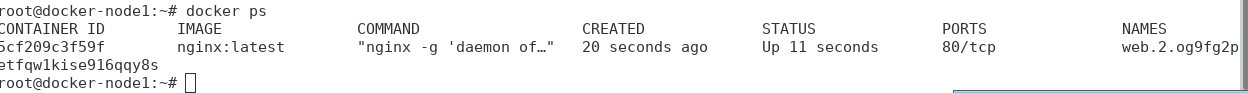
Такой механизм в swarm есть.

Вводи:

sudo docker service scale web=2



И вот контейнер на второй ноде (обрати внимание на поле **NAME** на обеих нодах =) )



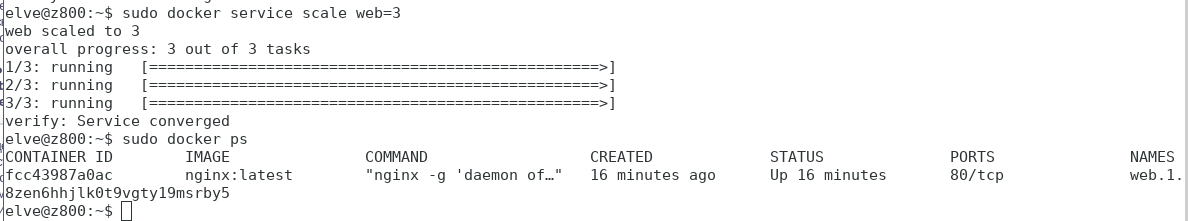
Дальше

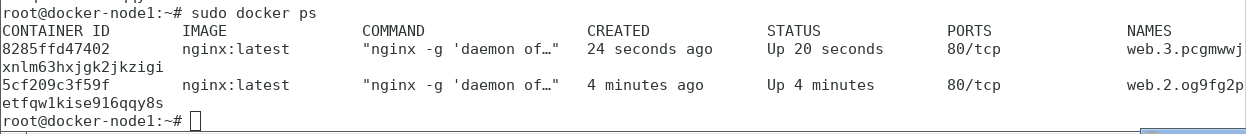
Ты же помнишь, что в моем распоряжении всего две **ноды**?

Так вот. А что будет если запустить 3 реплики сервиса?

Давай попробуем.

sudo docker service scale web=3





Как видишь на одной из нод поднялся лишний экземпляр контейнера.

Ограничения нет. Можно хоть до 100 реплик увеличивать. Но смысл такого действия туманен =))))

Дальше

В случае, если количество реплик равно количеству нод и одна из нод пропала из сети, то на одной из нод запустится второй экземпляр контейнера.

При включении ноды он обратно уползет на нее.

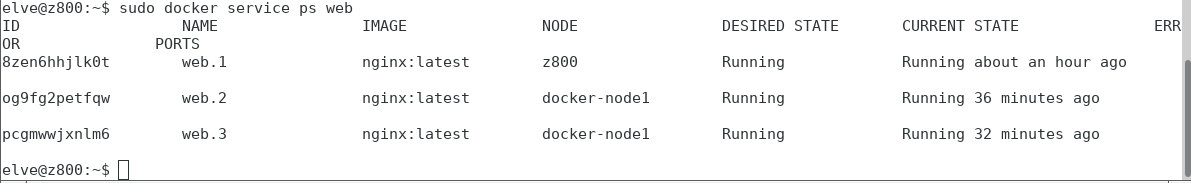
Можешь поэкспериментировать.

Сразу предупреждаю. Реплика не в прямом смысле слова переползает. Контейнер тушится на одной ноде и запускается на другой. Несохраненные данные при этом будут потеряны.

Это надо понимать при разработке контейнера, который будет работать в кластере.

Кстати, чтобы не лазать по всем нодам, чтобы посмотреть сколько и чего там запущено, можно ввести вот такую команду:

sudo docker service ps web



Дальше

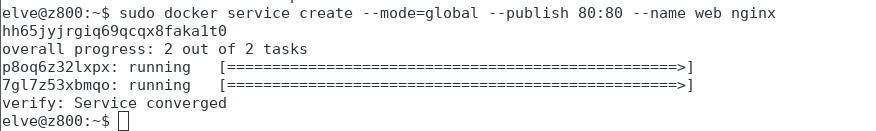
А теперь удалим сервис. Тем более он ничего полезного и не делает, да и попасть на контейнеры проблематично =).

Делается это вот так:

sudo docker service rm web

А теперь соберем сервис заново, но с другими параметрами. Как минимум неплохо бы 80-й порт вытащить наружу ;)

sudo docker service create --mode=global --publish 80:80 --name web nginx



Что изменилось?

**--mode=global** - этим параметром я сказал рою, что контейнер должен быть запущен на каждой ноде. По умолчанию сервис создается в режиме replicated

**--publish 80:80** - тут должно быть понятно. Я пробросил **80**-й порт на хост-машину. Теперь сервис виден снаружи. Подключаться можно к любой **ноде**.

Дальше

Попробуй теперь уменьшить количество реплик =).

sudo docker service scale web=1

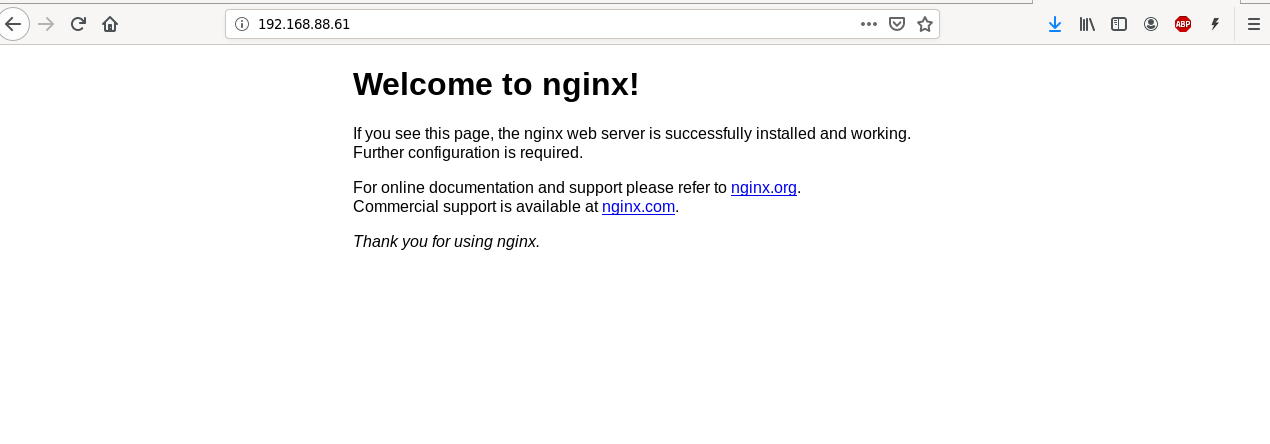
Не вышло? Ну так и должно быть. В этом режиме сервис заполоняет все доступные ноды.

Придется снова разобрать и собрать сервис.

sudo docker service rm web

sudo docker service create --replicas 1 --publish 80:80 --name web nginx

Ну и попробуй открыть веб-сайт на первой и второй нодах.



А какая же из нод тогда открывалась, когда их было две?

Все очень просто - случайная.

Балансировщик равномерно распределяет запросы между **репликами**.

Дальше

Благодаря этому балансировщику можно выполнять т.н. **rollup-update** для сервисов.

Да, я знаю что докер это не продакшн. Однако функция такая есть и я не могу о ней не упомянуть (да и много кто все же использует докер в продакшн =) ).

В чем задумка? Ну смотри. Вот к примеру используем мы для сервиса образ с встроенным приложением **webapp:1.0**. И выкатили разработчики свежий образ - **2.0**. Он оттестирован и его надо накатить так, чтобы никто не заметил простоя.

Так вот. Мы выполняем команду:

docker service update --update-parallelism 1 --update-delay 5s --image webapp:2.0 web

И каждая реплика обновляется по очереди с интервалом в 5 секунд, в то время как другие реплики остаются доступными.

Никакого простоя. Ни один пользователь ничего не заметит.

Дальше

Простои. Для современных сервисов это крайне неприятно. Желательно чтобы их не было вообще.

Так вот - а что будет, если единственная управлющая нода сдохнет? =).

Без рулевого коллектив пойдет вразнос и будет простой сервиса.

А вот чтобы этого избежать нужны резервные менеджеры. Да хоть все ноды можно объявить менеджерами. Тогда при кончине одной из управляющих нод, один из резервных менеджеров возьмет управление на себя.

Можно повышать воркеров до менеджеров:

docker node promote docker-node1



Дальше

Но можно ноду сразу подключать в рой в качестве менеджера.

Делается это так:

На мастер-ноде запрашивается отдельный токен для подключения менеджеров:

sudo docker swarm join-token manager

А после этого показанная команда используется для подключения новых нод.

А если надо менеджера разжаловать до воркера, то делается это вот так:

sudo docker demote docker-node1

Дальше

Теперь поговорим о стратегиях размещения реплик на нодах.

В общем и целом у **Docker Swarm** есть две стратегии:

1.Доступность ресурсов – планировщик Docker Swarm имеет информацию о доступных ресурсах на рабочих нодах и запускает задачу на наименее загруженной рабочей ноде  
2.Метки(labels) и ограничение(constrains)

С ресурсами понятно. А что это за метки такие?

Один сервер может быть добавлен только в один рой. А иногда требуется внутри роя как-то разграничить ресурсы самостоятельно. Для этого и применяются метки.

Метки делятся на стандартные, автоматические и заданные пользователем.

Стандартные метки есть всегда (к примеру node.hostname). В терминологии swarm это **node.labels.** Эти метки либо задаются докером, либо собираются с конкретного контейнера.

Но когда их недостаточно, то можно задать свои - **engine.labels.**

Дальше

Чтобы воспользоваться механизмом меток, нужно при создании сервиса указать требуемую метку для размещения приложения.

Создадим еще один сервис - **web2** с образом nginx, но жить он должен всегда на ноде **z800**

sudo docker service create --constraint 'node.hostname == z800' --name web2 --replicas 1 --publish 81:80 nginx

А как сделать свою метку?

Вот примерно вот так:

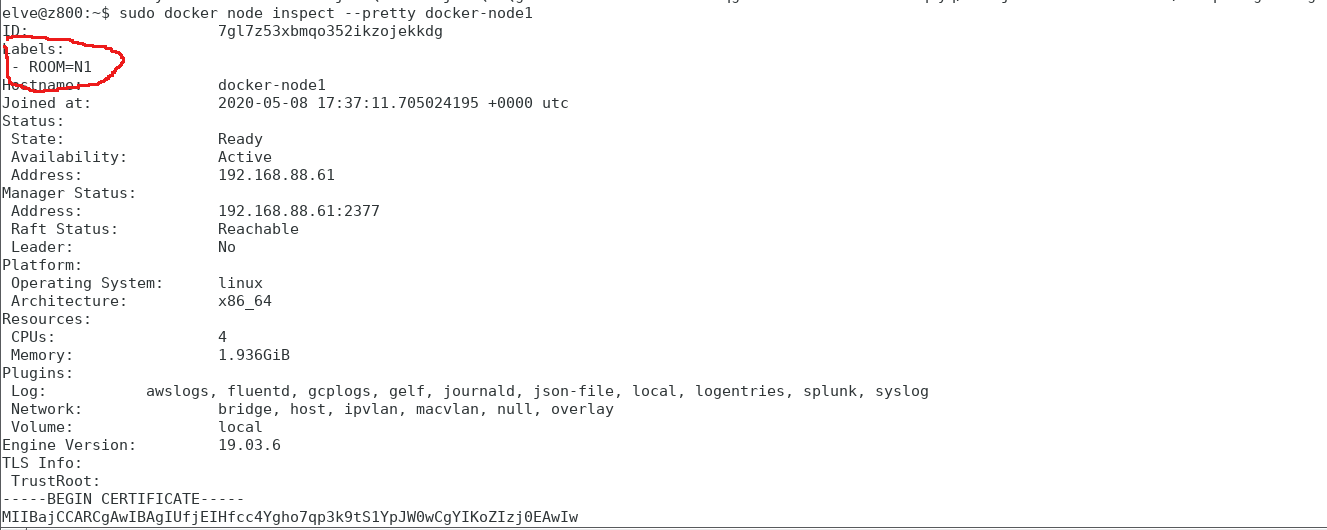
sudo docker node update --label-add ROOM=N1 docker-node1

А просмотреть метки можно вот так:

sudo docker node inspect --pretty docker-node1

или

sudo docker node inspect --pretty z800



Дальше

И теперь можно запускать приложение, точно зная, что оно запустится на ноде с меткой.

sudo docker service create --constraint '[node.labels.ROOM](https://node.labels.room/" \t "_blank) == N1' --name TestLabels --replicas 1 --publish 8889:8080 tomcat:latest

Это все, что я хотел тебе показать из возможностей **docker swarm** при управлении в ручном режиме.

С этими знаниями тебе будет проще отлаживать ошибки своих кластерных конфигураций.

В том числе и в той, которую мы с тобой соберем в следующей части урока.

Автоматизированная сборка группы связанных контейнеров внутри роя/кластера выполняется с помощью механизма **docker stack.**

По сути **Docker** Stack это как раз интеграция Composer в Swarm. Даже синтаксис конфигурационных файлов почти тот же.

Сразу предупреждаю насчет конфигурационных файлов. В уроке про **Docker** мы подключали локальную папку в контейнер. Эта папка никуда не пробрасывается в кластере. Т.е. если ты используешь подобную конструкцию, то должен позаботиться о том, чтобы на всех **нодах** эта папка была откуда-то примонтирована (либо прямо из контейнера монтировать папку по **nfs** например).

Дальше

Сейчас мы с тобой будем поднимать ту же финальную конфигурацию из докера - **nginx** + **php-fpm** + **mysql**

С учетом специфики роя/кластера хранилище с сайтом будет лежать на файлсервере и подключаться по **nfs**.

Для повторения эксперимента тебе нужно настроить файлсервер и по **nfs** расшарить папку - пусть будет **/docker/learning**

Я в уроке буду использовать эту. Ты же можешь ее поменять.

Дальше

В эту папку на файлсервере нужно будет сложить файлы сайта. Как минимум простенький **index.php**

Это будет общая папка с сайтом, доступная для изменений извне.

Образ **nginxtest/user1** надо будет собрать на всех нодах, либо поместить в публичное хранилище, чтобы все ноды могли его скачать.

В идеале образ всегда должен лежать в общем хранилище. Иначе удобство резко становится не совсем и удобством =).

Дальше

Ты можешь создать новый файл **docker-compose.yml** или же дополнить старый.

Начнем мы с подключаемого хранилища. Если ты успел почитать дополнительную литературу по работе с docker-compose, то наверное знаешь, что хранилище можно объявить отдельно и подключать в каждый контейнер по имени.

Если ты еще не в курсе, то тебя ждет сюрприз =).

Вот такой вот кусок кода надо добавить в конец docker-compose.yml

**volumes:**

**mywww:**

**driver\_opts:**

**type: "nfs"**

**o: "addr=192.168.88.241,nolock,soft,rw"**

**device: ":/docker/learning"**

Тут как видишь создается общее хранилище "mywww" и описываются параметры его подключения.

Дальше

Каждому контейнеру добавляется еще один раздел для конфигурирования - **deploy**, где описывается сколько реплик надо запускать.

И меняем данные по подключению хранилища.

Вот так они станут выглядеть в результате:

**mysql:**

**deploy:**

**replicas: 1**

**image: mysql**

**ports:**

**- 3306:3306**

**environment:**

**- MYSQL\_ROOT\_PASSWORD=vecrek**

**nginx:**

**deploy:**

**replicas: 2**

**image: elve/nginxtest**

**ports:**

**- 80:80**

**links:**

**- php**

**volumes:**

**- mywww:/var/www**

**php:**

**deploy:**

**replicas: 1**

**image: macedigital/phpfpm**

**volumes:**

**- mywww:/var/www**

Дальше

Если у тебя не сохранился файл и тебе нечего редактировать, то вот я тебе его приложил сюда целиком.

[Open file](https://storage.googleapis.com/media.helloumi.com/85100/channels/3PWWSES9G7B0SGNJCKBSGDP5PDR7FF95.yml)

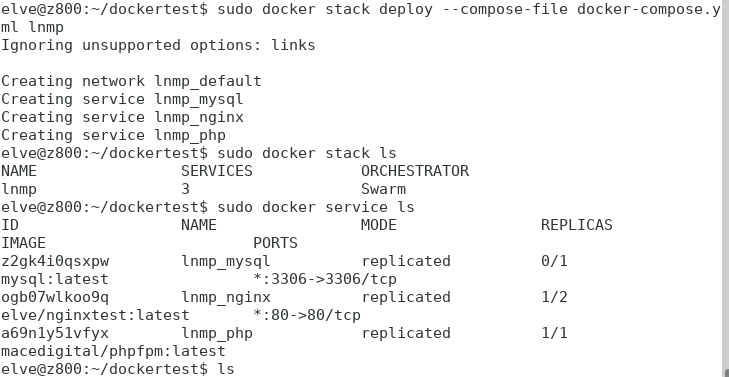
Сразу предупреждаю. Если хранилище по **nfs** недоступно, то будет создано хранилище типа local. Пустое. На каждой ноде свое.

Чтобы точно быть уверенным, что все смонтируется на всех нодах надо сделать:

apt install nfs-common

Все настроил и установил? Тогда запускаем!

sudo docker stack deploy --compose-file docker-compose.yml lnmp



Мы запустили стэк сервисов из **complose**-файла **docker-compose.yml**

Этому стэку мы дали кодовое название **lnmp**.

Дальше

Ты помнишь какие у нас там были настройки веб-сервера?

Имя сайта [**test.ru**](https://test.ru/)

Прописывай любую ноду на адрес [**test.ru**](https://test.ru/)и открывай в браузере [**http://test.ru**](http://test.ru/)

Заработало!

Ты обратил внимание на вывод **docker stack**?

Он написал, что опцию links не знает и проигнорил. Так почему же **index.php** выполнился?

А ты прочти внимательнее. Под набор сервисов была создана персональная **overlay**-сеть. И в каждый контейнер имена всех сервисов вписались в файл **/etc/hosts**. Т.е. они все друг друга видят, а не по линкам, как было в **композе**.

Дальше

Давай теперь посмотрим как это выглядит с точки зрения более низкоуровневых инструментов:

на обеих нодах:

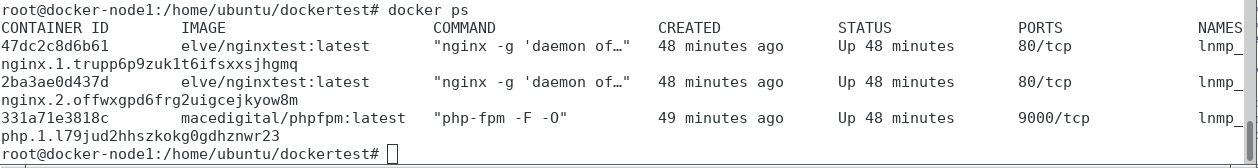
sudo docker ps

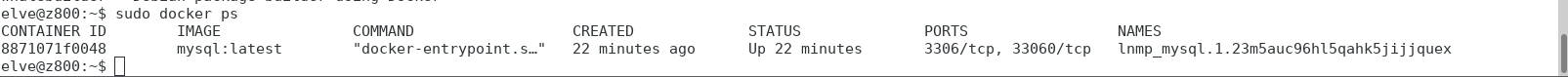
на мастер-ноде:

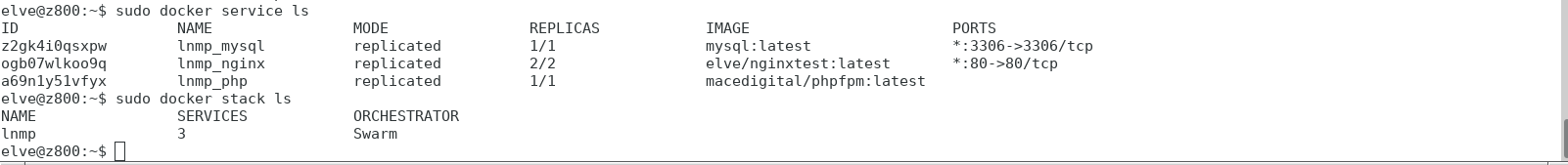
sudo docker service ls

sudo docker ps

sudo docker stack ls

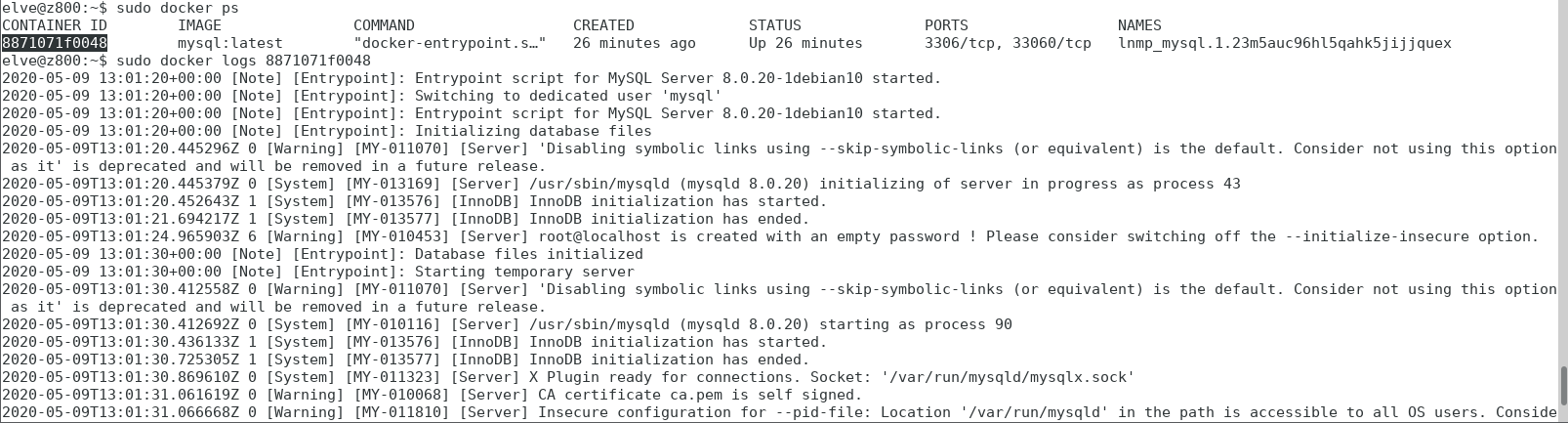






Дальше

И каждый контейнер ты можешь потыкать палочкой. Вот например:



Дальше

И удаляется стэк аналогично другим сущностям докера.

sudo docker stack rm lnmp

Ну что ж. Это все, что я хотел тебе сегодня рассказать.

Дальше

Конечно же, как и всегда, это не полное подробное описание всех опций всех конфигов, но практический урок, дающий тебе смелость тыкать во все эти кнопочки и решать встающие перед тобой проблемы. Как минимум ты уже без труда сможешь диагностировать имеющиеся кластеры и поддерживать их в рабочем состоянии.