**Лекция – 1 урок**

**Docker** – программное обеспечение для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации.

Все современные компьютеры поддерживают контейнеризацию.

Также используется для тестирования.

Первый релиз докера был в 2013 году.

Docker != Virtual Machine

Преимущества:

* Docker запускает только те процессы, которые нужны для запуска приложения. Запускает в отдельном потоке.
* Запускается на любом компьютере довольно быстро.

`Docker hub??

Особенности:

* Open-source
* Enterprise
* Cross-platform
* Написан на языке Go
* Малое потребление ресурсов
* Огромное количество готовых докер-образов

Области применения:

* Контейнеризация Веб-приложения
* Построения отказоуствойчивых систем
* Kuberneter – для косторизации. Для девопса
* Тестирование
* CI/CD

Запускается контейнер – запускается приложение.

* Установка для виндовс: 10 версия, 64 бита. 4 гб памяти под запуск докер-тестера.

Будет установлено:

* Docker Engine
* Docker CLI client
* Docker Compose
* Docker Machine

Команды:

* Docker version
* Docker run hello-world – показывает, что мы установили докер правильно

**Docker DAEMON** – демон выполняет команды, которые были отправлены клиентом docker. Сборка образов, запуск контейнеров.

**Dockerfile** – файл с набором инструкций, который используется для сборки образов (docker image). Собирают образы по докер-файлу.

**Docker Image** – файл состоящий из множества слоев, который используется для выполнения кода в докер контейнерах. Каждый слой – набор данных. Используется для запуска контейнера. Каждая команда в докер файле создает новый слой.

**Union File Systems** – своего рода объединяемая файловая система, которая содержит файлы и каталоги разных файловых систем. Они прозрачно накладываются друг на друга, образуя единую файловую систему.

Docker Volumes – часть данных контейнеров, которые ссылаются на внешние носители. Сохранять данные внутри контейнеров можно при наличии

Контейнер без образа существовать не может.

2 урок (Докер-контейнеры)

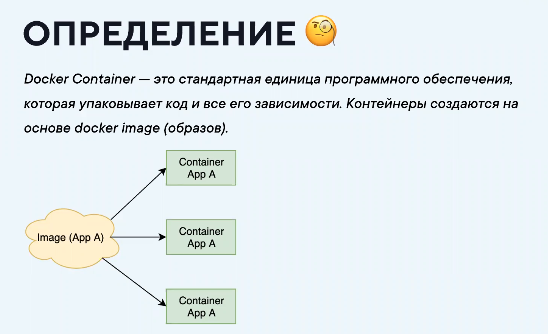
Содержание:

* Работа с линукс и видовс контейнерами
* Разобраться с docker network
* Познакомиться с Nginx
* Научимся настраивать взаимодействие меду контейнерами
* Масштабирование контейнеров

На основе docker image создается docker container.

Проверяем работоспособность докера:

* docker version (client and server)
* docker info – проверка конфигов
* docker run –rm hello-world
* docker container run –rm hello-world



Создание контейнера:

Docker create <options><image name:tag>

docker container create <options><image name:tag>

docker info (information about container):

docker ps <options>

docker container ls <options>

Container stats:

* Container ID – идентификатор контейнера
* Image – образ на основании которого был создан контейнер
* Command – команда, которая используется как основной процесс
* Created – время когда был создан контейнер
* Status – статус контейнера(запущен, на паузе, остановлен с ошибкой и тд)
* Ports – внутренние порты и мапинг портов
* Names – имя контейнера

Запуск nginx:

* docker run --publish 80:80 nginx
* docker run -p 80:80 nginx (сокращенная версия сроки выше)
* docker container run --publish 80:80 nginx
* docker container run -p 80:80 nginx (сокращенная версия)

1. Загружается образ из докер хаб
2. Запускает контейнер для этого образа
3. Открывает 80 порт для host IP
4. Перенаправляет трафик на 80-й порт контейнера

Run Nginx: detached:

Add:

* -detach nginx
* -d nginx

Name:

* docker run --name webhost nginx

remove container:

* docker rm <options> <container name> or <hash>
* docker container rm <options> <container name> or <hash>

Stop and pause, start, unpause:

* docker stop <container na me> or <hash>
* docker pause <container name> or <hash>

Inspect – подробная информация о контейнере, включая хэлс чеки.

* docker inspect <container name> or <hash>
* docker container inspect <container name> or <hash>

Container processes – показывает процессы, которые запущены внутри контейнера.

* docker top <container name> or <hash>

STATS – в режиме реального времени увидеть потребляемые ресурсы контейнерами:

* docker stats <container name> or <hash>

**Docker Networks**

Docker Network – внутренняя сеть докер которая служит для объединения контейнеров в разные подсети и изоляции контейнеров на уровне сети.

Типы драйверов:

* Bridge – для изолирования контейнеров
* Host – для присоединения сети контейнеров к хостовой системе
* Overlay – тип драйвера для соединения нескольких докер демонов
* Macvlan – редко используется. позволяет приматить
* None
* Network plugins – добавляется все остальное (редко)

DOCKER NETWORK:

* Connect
* Create
* Disconnect
* Inspect
* Ls – output list of networks
* Prune – удалить неиспользуемые сети
* Rm

Один контейнер может принадлежать нескольким сетям

**3 урок**

**Registry & Images**

**Содержание:**

* Докер хаб
* Docker Registry
* Docker Images
* Научимся создавать собственные образы Dockerfile
* Познакомимся в версионированием образов, Image Tags

1. Docker Hub & Docker Registry

Docker Hub – это облачное хранилище, предназначенное для создания публичных и приватных репозиториев для образов. Репозиторий для работы с Докер по умолчанию. Самый популярный публичный репозиторий докер образов. Подключается в ГитХаб, БитБакет для авто-сборки. Создает цепочки для сборки образов.

Docker Store – магазин, где можно приобрести образы или купить подписку на официальные, сертифицированные образы.

Docker Registry – репозиторий для хранения докер образов

Запуск доке реджистри:

docker container run -d -p 5000:5000 - -restart always - -name registry registry:2

Запушить в локальный репозиторий:

docker push localhost: 5000/ubuntu

1. **Images & Tags**

Репозиторий = хранилище образов.

Docker images – файл, состоящий из множества слоев, который используется для выполнения кода в докер контейнерах. Read-only template.

Docker Image Tag – лейбл, используемый для версионирования докер образов.

Просмотреть все локальные образы:

* docker images <options> <image name:tag>
* docker image ls <options> <image name:tag>

Image info:

* Repository
* Tag
* Image ID
* Created – относительное время когда был создан имаджи
* Size – размер образа

Скачивание образа:

* docker pull <options> <image name:tag>
* docker image pull <options> <image name:tag>

Запушивание образа:

* docker push <options> <image name: tag>
* docker image push <options> <image name:tag>
* docker inspect <options> <image name:tag>
* docker history <options> <image name:tag>
* docker tag <source image name: tag> <image name:tag> - **версия образа**

**Dockerfile** – файл с набором инструкций предназначенный для создания образов.

Создать образ:

* docker build.-t <image name:tag>
* docker image build.-t<image name:tag>

Директивы

* Escape
* Syntax

Команды:

From

Run – запуск

Cmd

Label – добавление мета информации

Expose – открывает внутренние порты контейнера (наружу порты)

Env – переменные среди окружения

Add – добавить инфу внутрь контейнера

Copy – скопировать в хотстовую систему

Entrypoint

Volume -

User – под именем какого пользователя будут запускаться

Workdir

Arg – аргументы внутри образа

Onbuld -

Stopsignal – записывается сигнал

Healthcheck – хэлз чеки, проверка жив или не жив контейнер

Shell – выбрать шэл-скрипт

**4 урок**

**Volumes & compose**

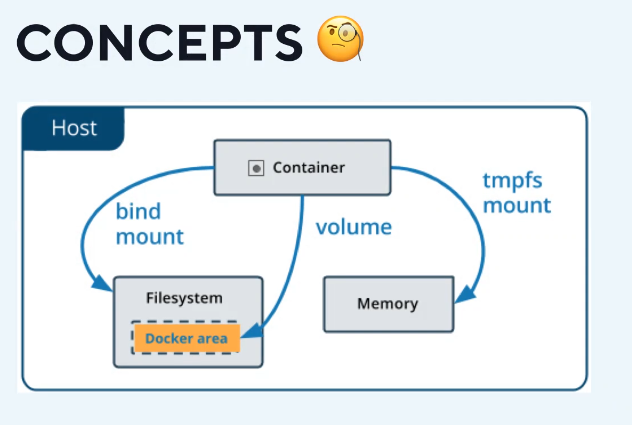
**Содержание:**

* Жизннный цикл контейнера - Lifetime
* Как мониторить логи контейнера
* Docker Volumes
* Поднимание инфраструктуры с docker-compose
* Управление ресурсами с docker-compose, image building
* Запуск fullstack приложения

Docker bind mount – файл или директория на диске host системы.

Docker Volume – механизм для постоянного сохранения данных генерируемых и используемых контейнером.

TMPFS – механизм для временного сохранения данных генерируемых и используемых контейнеров в оперативной памяти host системы.



Docker volume CMD:

* Create – создает вольюм
* Inspect – выводит информацию
* Ls – все значения
* Prune – удаляет сети и контейнеры, которые не используются. Не трогают вольюм.
* Rm - удаление

При удалении контейнера вольюм не удаляется.

* docker volume create <options> <volume name>

В опциях драйвер.

Usage (путь внутри контейнера):

* docker container run -d -v <volume name>:/container/path nginx

Volume: ANONYMOUS

docker container run -d -v /usr/share/nginx/html nginx:alpine

Volume:NAMED

* docker container run -d \
* -- mount type = volume, source = nginx -vol, destination = /usr/share/nginx/html \
* nginx:alpine

Запуск докер-компоус:

Переходим в нужную папку, где хранится файл .yml

Docker-compose up -d - запуск

docker-compose down -остановка