Docker

General

- Container - is a sandboxed process on your machine that is isolated from all other processes on the host machine. That isolation `lerages kernel` `namespaces` and `cgroups`, features that have been in Linux for a long time. Docker has worked to make these capabilities approachable and easy to use.
  To summarize, a container:
  - is a runnable instance of an image. You can create, start, stop, move, or delete a container using the DockerAPI or CLI.
  - can be run on local machines, virtual machines or deployed to the cloud.
  - is portable (can be run on any OS).
  - Containers are isolated from each other| and run their own software, binaries, and configurations.
- When running a container, it uses an isolated filesystem. This custom filesystem is provided by a container image.
- Image contains all dependencies, configuration, scripts, binaries, etc. The image also contains other configuration for the container, such as environment variables, a default command to run, and other metadata.

One container - one process!

- There`s a good chance you had have to scale APIs and front-ends differently than databases
- Separate containers let you version and update versions in isolation.
- While you may use a container for the database locally, you may want to use a managed service for the database in production. You don`t want to ship your database engine with your app then.
- Running multiple processes will require a process manager (the container only starts one process), which adds complexity to container startup/shutdown

To connect containers with each other use networking

- If two containers are on the same network, they can talk` to each other. If they aren`t, they can not.
- Create a network | `docker network create <network name>`
- Start container and attach it to the network | `docker run -d --network <network name> --network-alias` | <it will become a hostname>
- Start other container in the same network

Commands:

Command	Describe
`docker images`	какие вообще есть image на данный момент
`docker run <image name>`	Download and start your container
`docker search <name>`	Search image in registry
`docker pull`	Download image from registry to local machine

`docker build`
`docker build -t <tag>:<version>`
`docker build <path to docker file>`
	Build image
	Build image + tag and version
	Build an image from a specified docker file

`docker logs`	Container logs
`docker ps` `docker ps -a`	List all running containers To show all the running and exited containers

`docker start/stop/restart <name>`
`docker start/stop/restart <id>`
`docker pause/unpause <id>`
	To work with containers

`docker inspect <container name/id>`	View information about container
`docker run --publish 443:443/80:80/24:22`	Port forward
`docker -p <local port>:<container port>`	Run a container on certain port
`docker exec -it <container id> bash`	Terminal inside a container
`docker exec <container-id> <command that you need to run>`	Execute command inside a container
`docker kill <container name/id>`	This command kills the container by stopping its execution immediately

`docker rm <container name/id>` `docker rm -f <container name/id>`	Remove the container after IT STOPS. You can not stop the container before removal
`docker rmi <image name/id>`	Delete an image from local storage
`docker image prune`	Remove unused containers
`docker network ls`	Display all networks/valumes/images, etc.
`sudo docker tag nginx-reverse_nginx-reverse-proxy:latest nginx-reverse_nginx-reverse-proxy:old`	take tag "old"
`sudo docker tag kh-rdua-mentor-r1-devops_mentor-jms:latest kh-rdua-mentor-r1-devops_mentor-jms:old`	
About docker build: https://docs.docker.com/engine/reference/commandline/build/	
`docker build --tag test --file test.dockerfile .`	Билд из докерфайла с определенным именем
`docker build --tag test:version1.2 --file test.dockerfile .`	Билд из докерфайла с определенным именем + сразу дать тег: "version1.2"
`docker tag test webserver` `docker rmi test`	rename image and del old name (untagged)
`docker rename task-01_db_1 database`	rename container

RUN	The `RUN` instruction will execute any commands in a new layer on top of the current image and commit the results.

CMD	There can only be one `CMD` instruction in a `Dockerfile`. If you list more than one `CMD` then only the last `CMD` will take effect. Основная цель CMD - предоставить параметры по умолчанию для исполняемого контейнера. Эти значения по умолчанию могут включать в себя исполняемый файл, а могут и не включать его, в этом случае вы должны указать инструкцию ENTRYPOINT.
LABEL	The `LABEL` instruction adds metadata to an image. A `LABEL` is a key-value pair. To include spaces within a `LABEL` value, use quotes and backslashes as you would in command-line parsing. An image can have more than one label. You can specify multiple labels on a single line.
MAINTAINER	The `MAINTAINER` instruction sets the *Author* field of the generated images.
EXPOSE	The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.
	The `ENV` instruction sets the environment variable `<key>` to the value `<value>`. The `ENV` instruction allows for multiple `<key>=<value> ...` variables to be set at one time
ADD	Инструкция ADD копирует новые файлы, каталоги или URL удаленных файлов из <src> и добавляет их в файловую систему образа по пути <dest>.

ENTRYPOINT	You can use the *exec* form of `ENTRYPOINT` to set fairly stable default commands and arguments and then use either form of `CMD` to set additional defaults that are more likely to be changed.
	`FROM ubuntu`
	`ENTRYPOINT ["top", "-b"]`
	`CMD ["-c"]`
	
	`When you run the container, you can see that top is the only process:`
	`docker run -it --rm --name test  top -H`

VOLUME	The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log /var/db.
	`FROM ubuntu`
	`RUN mkdir /myvol`
	`RUN echo "hello world" > /myvol/greeting`
	`VOLUME /myvol`

USER	The `USER` instruction sets the user name (or UID) and optionally the user group (or GID) to use as the default user and group for the remainder of the current stage.
WORKDIR	The `WORKDIR` instruction sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY` and `ADD` instructions that follow it in the `Dockerfile`. If the `WORKDIR` doesn't exist, it will be created even if it's not used in any subsequent `Dockerfile` instruction.

ARG	Инструкция ARG определяет переменную, которую пользователь может передать сборщику во время сборки с помощью команды docker build, используя флаг --build-arg <varname>=<value>. Если пользователь указывает аргумент сборки, который не был определен в Dockerfile, сборка выдает предупреждение. Определение переменной ARG вступает в силу с той строки, на которой оно определено в Dockerfile, а не с момента использования аргумента в командной строке или где-либо еще. Например, рассмотрим этот Dockerfile:
	`FROM busybox`
	`USER ${user:-some_user}`
	`ARG user`
	`USER $user`

DOCKER-COMPOSE.yaml

```
# Версия схемы, которую мы используем.
# Зависит от установленной версии docker
# https://docs.docker.com/compose/compose-file/
version: "3"
# Определяем список сервисов — services
# Эти сервисы будут частью нашего приложения
```

```
services:
  app: # Имя сервиса
    build:
      # Контекст для сборки образа,
      # в данном случае, текущая директория
      context: .
      # Имя докерфайла из которого будет собран образ
      dockerfile: Dockerfile
      # Команда, которая будет выполнена после старта сервиса
    command: make start
    ports: # Проброс портов
      - "3000:8000"
    # Перечисляем тома (volumes)
    # Они будут подключены к файловой системе сервиса
    # Например, всё что находится в . мы увидим в директории /app
    volumes:
      # Текущая директория пробрасывается в директорию /app внутри контейнера
      - ".:/app"
      - "/tmp:/tmp"
    # Сервис будет запущен, только после старта db
    depends_on:
      - db
  db:
    # Имя образа. Здесь мы используем базу данных Postgres
    image: postgres:latest
    environment:
      # А так задаются переменные окружения
      POSTGRES_PASSWORD: password
    volumes:
      - pgdata:/var/lib/postgresql/data


DOCKER-COMPOSE COMMANS:

# Собирает сервисы, описанные в конфигурационных файлах
docker-compose build

# Запускает собранные сервисы
docker-compose up

# Запуск контейнеров на фоне с флагом -d
docker-compose up -d

# Если какой-то из сервисов завершит работу,
# то остальные будут остановлены автоматически
docker-compose up --abort-on-container-exit

# Запустит сервис application и выполнит внутри команду make install
docker-compose run application make install

# А так мы можем запустить сервис и подключиться к нему с помощью bash
docker-compose run application bash
```

```
# Со флагом --rm запускаемые контейнеры будут автоматически удаляться
docker-compose run --rm application bash

# Останавливает и удаляет все сервисы,
# которые были запущены с помощью up
docker-compose down

# Останавливает но не удаляет сервисы, запущенные с помощью up
# Их можно запустить снова с помощью docker-compose start
docker-compose stop

# Перезапускает все остановленные и запущенные сервисы
docker-compose restart

# Запустить/остановить/сбилдить определенный docker-compose.yaml файл
docker-compose -f docker-compose.yaml up -d
docker-compose -f docker-compose-eme.yaml stop
docker-compose -f docker-compose-eme.yaml build
```

Work with volume

- Volumes are the preferred mechanism |for persisting data| generated by and used by Docker containers.

Named volumes

- Think of a named volume as simply a bucket of data.
- Docker maintains the physical location on the disk and you only need to remember it`s name

`docker volume create <volume name>`

- Create a volume by using the docker volume create command.
- To find out where the volume is stored

`docker volume inspect <volume name>`

- Mountpoint is the actual location on the disk where the data is stored.

Bind volumes

- With bind mounts, we control the exact mountpoint on the host.
- We can use this to persist data, but it`s often used to provide additional data into containers.

```
docker run -dp 3000:3000 \
-w /app -v "$(pwd):/app" \
node:12-alpine \
sh -c "yarn install && yarn run dev"
```

`-dp 3000:3000` - same as before. Run in detached (background) mode and create a port mapping

`-w /app` - sets the “working directory” or the current directory that the command will run from

`-v "$(pwd):/app"` - bind mount the current directory from the host in the container into the /app directory

`node:12-alpine` - the image to use. Note that this is the base image for our app from the Dockerfile

`sh -c "yarn install && yarn run dev"` - the command. We`re starting a shell using sh (alpine doesn`t have bash) and running yarn install to install all dependencies and then running yarn run dev. If we look in the package.json, we`ll see that the dev script is starting nodemon.

```
version: "3.7"
services:
  nginx-reverse-proxy:
    build:
      context: .
    container_name: nginx-reverse-proxy
    restart: always
    ports:
      - '80:80'
      - '443:443'
# Прокидываем папку конфигурации nginx из текущей директории в контейнер (волюм)
# Прокидываем файл из текущей директории в контейнер (волюм)
    volumes:
      - ./conf:/etc/nginx/conf.d
      - ./dhparam.pem:/etc/nginx/dhparam.pem
```

Docker hub

Login

`docker login -u YOUR-USER-NAME`

- Login to the Docker Hub using the command

Tag image

`docker tag getting-started YOUR-USER-NAME/getting-started`

- Use the docker tag command to give the getting-started image a new name.
- Be sure to swap out YOUR-USER-NAME with your `Docker ID`.

Push image

`docker push YOUR-USER-NAME/getting-started`

- If you do not specify a tag, Docker will use a tag called latest.

Tips and Tricks

`docker rm $(docker ps -a -q)`

- Remove all images

`docker run --rm ubuntu`

- Flag [--rm] removes a container after it became unused

`docker ps --filter 'label=app=web1' --format "table {{.ID}\t{{.Status}}}"`

- Filter by label

`docker run --privileged=true`

- Give process all capabilities that your root has

Links:

- https://docs.docker.com/get-started/
- https://docs.docker.com/get-started/07_multi_container/
- https://www.youtube.com/watch?v=-YnMr1lj4Z8

https://labs.play-with-docker.com/ | Docker playground