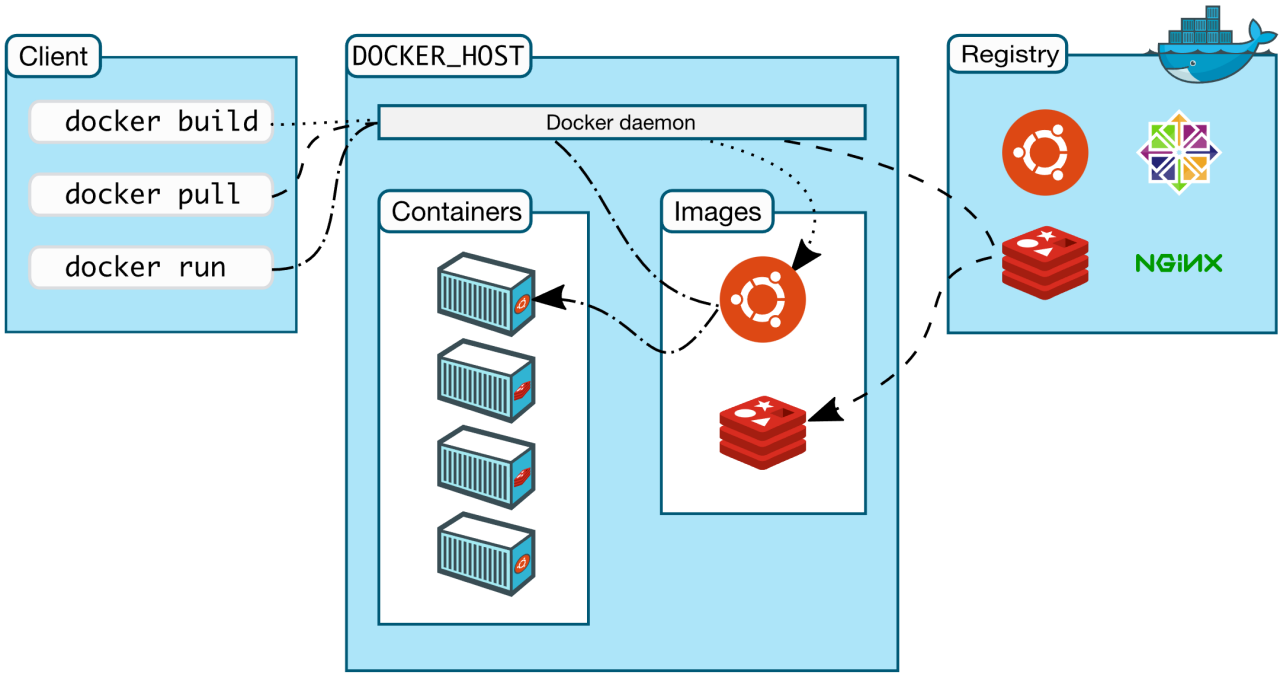
Docker (PHP Chapter #1)

What is a Docker?

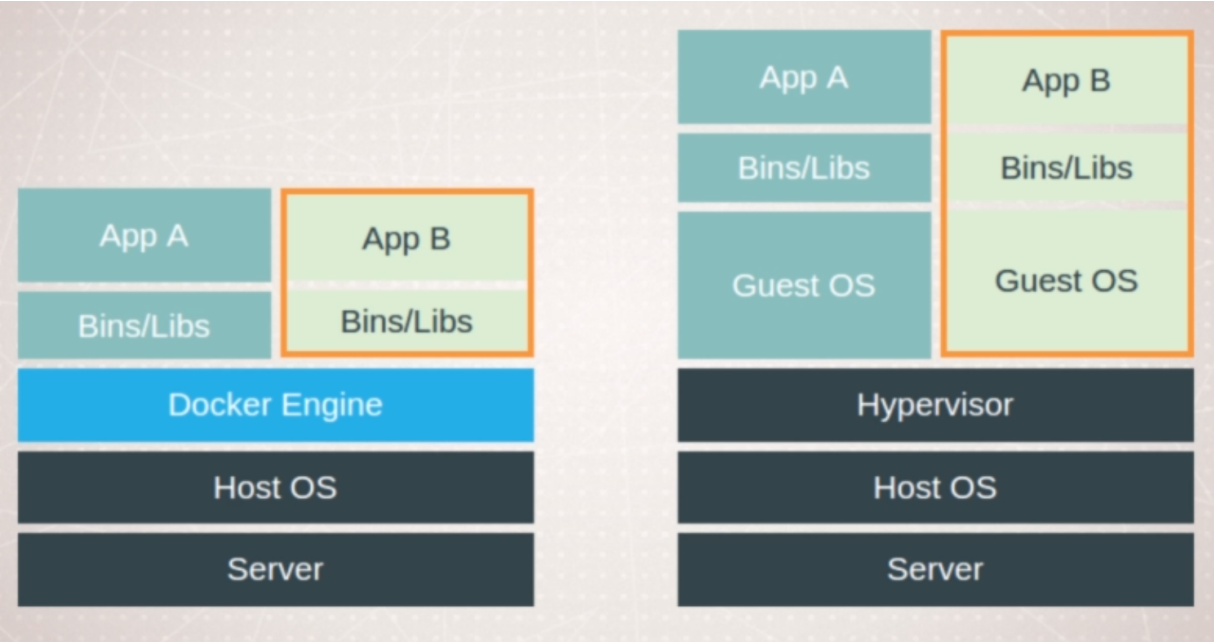
- https://docs.docker.com/get-started/overview/
- Docker is an open platform for **developing**, **shipping**, and **running applications**
- Docker provides the ability to **package** and run an application in a loosely isolated environment called a **container**
- Containers are great for continuous integration and continuous delivery (CI/CD) workflows
- Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments
- Docker is **lightweight** and **fast**. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources



- Docker uses a **client-server architecture**. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, which lets you work with applications consisting of a set of containers.
- A Docker ***registry*** stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. https://hub.docker.com/
- `docker version`
- `docker info` shows most configuration values for the engine
- `docker` shows a list of commands
- `/var/lib/docker`
  - ```
    vavetic@47:/var/lib$ cd docker
    vavetic@47:/var/lib/docker$ ls
    buildkit  containers  image  network  overlay2  plugins  runtimes  swarm  tmp  trust  volumes
    ```

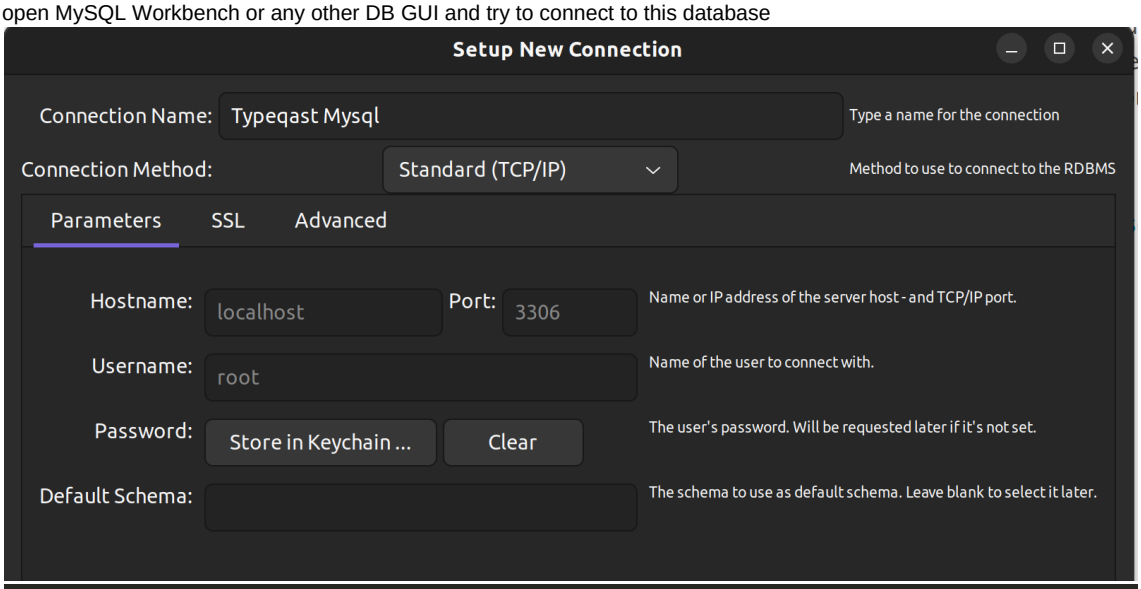
Containers

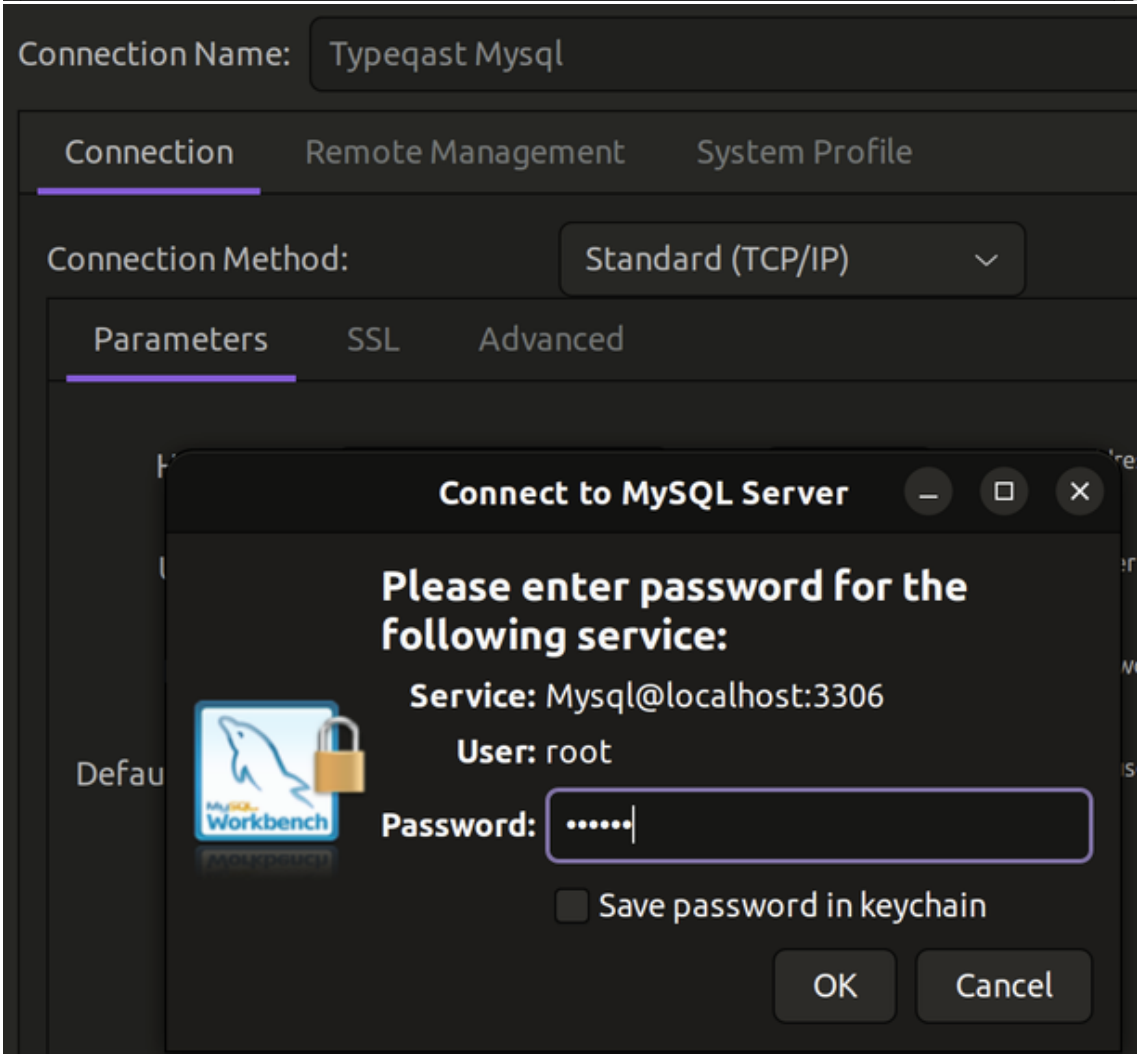
- A container is a **runnable** instance of an image. You can **create**, **start**, **stop**, **move**, or **delete** a container using the Docker API or CLI. You can **connect** a container to one or more **networks**, **attach storage** to it, or even create a new image based on its current state.
- By default, a container is **relatively well isolated** from other containers and its host machine. You can control how isolated a container's **network**, **storage**, or other **underlying subsystems** are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, **any changes to its state that are not stored in persistent storage disappear.**
- Containers vs VMs
    ```

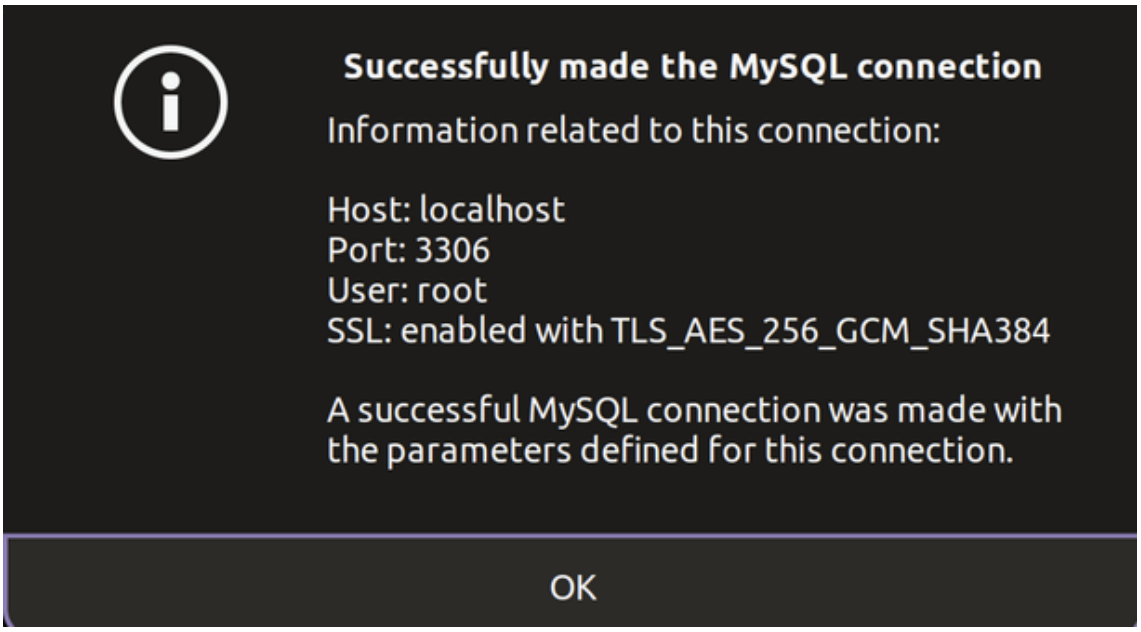


- https://www.youtube.com/watch?v=cjXI-yxqGTI
- https://www.youtube.com/watch?v=0qotVMX-J5s
- https://www.youtube.com/watch?v=LMAEbB2a50M
- software vs hardware virtualization
- isolation of process vs isolation of machine
- namespaces & cgroups
    - Docker uses namespaces of various kinds **to provide the isolation that containers need in order to remain portable and refrain from affecting the remainder of the host system**. Each aspect of a container runs in a separate namespace and its access is limited to that namespace. https://medium.com/@kasunmaduraeng/docker-namespace-and-cgroups-dece27c209c7
    - Docker uses a technology called `namespaces` to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container
    - **namespaces** https://en.wikipedia.org/wiki/Linux_namespaces
    - **cgroups** https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/
    - https://www.linux.com/news/understanding-and-securing-linux-namespaces/

- **Exercise #1:**
    - run offical `nginx` image as docker container: https://hub.docker.com/_/nginx
    - https://github.com/nginxinc/docker-nginx/blob/04226fe92cc11bed68dae464eb60fd5399daf3b1/mainline/debian/Dockerfile
    - `docker container run --name typeqast-nginx -p 80:80 -d nginx`
        - docker gives the container virtual IP address inside the docker network
        - route request from host's port 80 to container's port 80
        - docker pulls the image from the Docker hub (which is the default option if any other is not defined)
        - `-d` flag means detach - container is running in the background. if -d is not passed, the container will run in the foreground
        - at the bottom of the nginx dockerfile CMD instruction is defined
            - `CMD ["nginx", "-g", "daemon off;"]`
            - this is default process that starts to run once the container is up
            - process can be changed at run time
                - `docker container run -it --name typeqast-nginx nginx sh`
                - in this case, `sh` replaces `nginx` as the default container process
    - `docker container ls` check if container is up
    - `docker container exec -it typeqast-nginx sh` ssh into a running container
        - `apt-get update && apt-get install iputils-ping`
        - `ping google.com`
    - `docker container top typeqast-nginx` list running processes inside the container
        - `ps aux | grep nginx`
        - `docker stop typeqast-nginx`
        - `ps aux | grep nginx`
    - http://localhost/ - nginx home page should be visible on `localhost:80`
    - `docker container logs typeqast-nginx` displays container logs
    - `docker container stop typeqast-nginx`
    - `docker start typeqast-nginx`- start again stopped container
    - `docker container rm typeqast-nginx`
    - `docker container run --name tq-nginx -p 80:80 -d nginx`
        - start a new container from the existing image

    - the image exists on the host (locally in cache) so the docker engine will not pull the image from the Docker hub
- pass env vars to the container
    - `docker container run -it --name typeqast-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=typeqast -e MYSQL_DATABASE=typeqast-db mysql`
    - open MySQL Workbench or any other DB GUI and try to connect to this database
    - 

    - 



- inspect docker container
  - `docker container inspect typeqast-nginx`
    - shows metadata about the container
  - `docker container stats typeqast-nginx`
    - shows performance data
- run a container from `ubuntu` image, install `curl`, and ping `google.com`
  - `docker container run -it --name typeqast-ubuntu ubuntu bash`
  - `root@2581242eab2e:/# apt-get update && apt-get install curl`
  - `curl google.com`

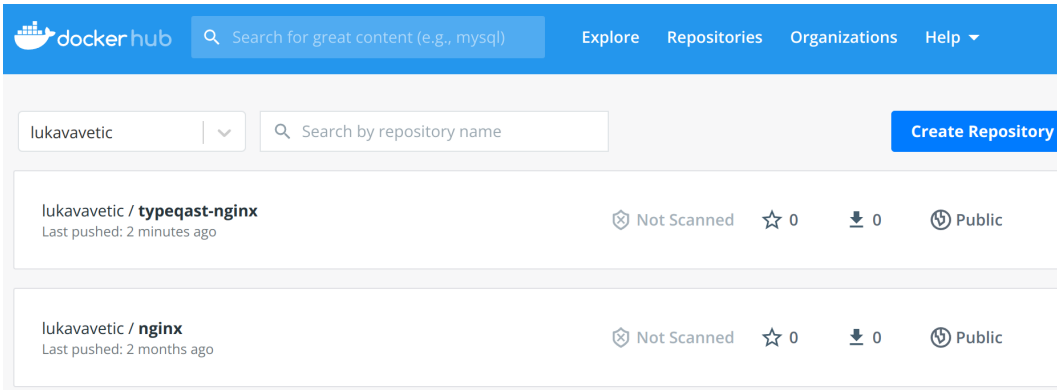
Image

- An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization.
- To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.
- Since the image contains the container's filesystem, it must contain everything needed to run an application - all dependencies, configuration, scripts, binaries, etc. The image also contains other configurations for the container, such as environment variables, a default command to run, and other metadata.
- **Exercise #2:**
  - https://hub.docker.com/search?q=nginx
  - pull the image with the tag
    - `docker image pull nginx:1.11.9`
    - `docker image pull nginx`
    - if a tag is not defined, `latest` is a default tag
  - listing all images on your host
    - `docker image ls`
    - ```
      vavetic@47:~$ docker image ls
      REPOSITORY                                          TAG                 IMAGE ID            CREATED             SIZE
      mysql                                               latest              0ef9083d9892        2 days ago          524MB
      nginx                                               latest              55f4b40fe486        7 days ago          142MB
      ```
  - difference between regular and alpine image
    - `alpine` images are smaller in size
    - `docker image pull nginx`
    - `docker image pull nginx:alpine`
    - ```
      nginx                                               latest              55f4b40fe486        7 days ago          142MB
      nginx                                               alpine              f246e6f9d0b2        7 days ago          23.5MB
      ```
    - notice the difference in size of the images 142 MB comparing to 23.5 MB (`alpine`)
  - an image consists of one or multiple layers
    - display layers history
      - `docker image history nginx:latest`
      -

```
vavetic@47:~$ docker history nginx:latest
IMAGE          CREATED      CREATED BY                                      SIZE
55f4b40fe486   7 days ago   /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>      7 days ago   /bin/sh -c #(nop)  STOPSIGNAL SIGQUIT           0B
<missing>      7 days ago   /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>      7 days ago   /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>      7 days ago   /bin/sh -c #(nop) COPY file:09a214a3e07c919a…   4.61kB
<missing>      7 days ago   /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>      7 days ago   /bin/sh -c #(nop) COPY file:0b866ff3fc1ef5b0…   1.96kB
<missing>      7 days ago   /bin/sh -c #(nop) COPY file:65504f71f5855ca0…   1.2kB
<missing>      7 days ago   /bin/sh -c set -x     && addgroup --system -…   61.1MB
<missing>      7 days ago   /bin/sh -c #(nop)  ENV PKG_RELEASE=1~bullseye   0B
<missing>      7 days ago   /bin/sh -c #(nop)  ENV NJS_VERSION=0.7.5        0B
<missing>      7 days ago   /bin/sh -c #(nop)  ENV NGINX_VERSION=1.23.0     0B
<missing>      7 days ago   /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>      7 days ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      7 days ago   /bin/sh -c #(nop) ADD file:8adbbab04d6f84cd8…   80.4MB
```

- each layer has its own unique SHA hash which enables Docker to efficiently cache layers and stays performant
  - `docker image inspect nginx:latest`
  - ```
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:08249ce7456a1c0613eafe868aed936a284ed9f1d6144f7d2d08c514974a2af9",
            "sha256:d5b40e80384bb94d01a8d2d8fb2db1328990e7088640132c33d3f691dd8a88ee",
            "sha256:b2f82de68e0d9246de01fa8283876427af5d6f3fe21c4bb04785892d5d071aef",
            "sha256:41451f050aa883f9102df03821485fc2e27611da05689c0ba25f69dcda308988",
            "sha256:44193d3f4ea2bae7a5ae5983f2562f551618b787751a6abfb732b6d17393bb88",
            "sha256:e7344f8a29a34b4861faf6adcf072afb26fadf6096756f0e3fc4c289cdefb7c2"
        ]
    },
    ```
- tag an image
  - `docker image tag nginx lukavavetic/typeqast-nginx`
  - `docker image ls`
  - ```
    lukavavetic/typeqast-nginx                    latest          55f4b40fe486   7 days ago     142MB
    nginx                                         alpine          f246e6f9d0b2   7 days ago     23.5MB
    ```
  - different image tag but pointing to the same image ID
- log in and push image to Docker Hub
  - `docker login`
  - ```
    vavetic@47:~$ docker login
    Login with your Docker ID to push and pull images from Docker Hub. If you
    Username: lukavavetic
    Password:
    WARNING! Your password will be stored unencrypted in /home/vavetic/.docker/
    Configure a credential helper to remove this warning. See
    https://docs.docker.com/engine/reference/commandline/login/#credentials-st

    Login Succeeded
    vavetic@47:~$ cat /home/vavetic/.docker/config.json
    {
            "auths": {
                    "https://index.docker.io/v1/": {
                            "auth": "bHVrYXZhdmV0aWM6RG9ja2VyMDk4OTU4ODU4Ng=="
                    },
    ```
  - `docker image push lukavavetic/typeqast-nginx`
    - ```
      vavetic@47:~$ docker image push lukavavetic/typeqast-nginx
      Using default tag: latest
      The push refers to repository [docker.io/lukavavetic/typeqast-nginx]
      e7344f8a29a3: Pushed
      44193d3f4ea2: Pushed
      41451f050aa8: Pushed
      b2f82de68e0d: Pushed
      d5b40e80384b: Pushed
      08249ce7456a: Pushed
      latest: digest: sha256:3536d368b898eef291fb1f6d184a95f8bc1a6f863c48457395aab859fda354d1 size: 1570
      ```
    -



- **Exercise #3:**
  - create index.html

```
<h1>Hello world!</h1>
```

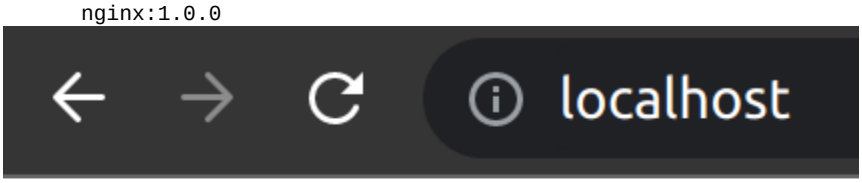
  - create Dockerfile

```
FROM nginx:1.22.0

RUN apt-get upgrade && apt-get install curl iputils-ping -y

WORKDIR /usr/share/nginx/html

COPY index.html index.html
```

  - build a new image from Dockerfile
    - `docker image build -t lukavavetic/typeqast-nginx:1.0.0 .`
  - run container from `lukavavetic/typeqast-nginx:1.0.0` image
    - `docker container run -d --name lukavavetic-typeqast-nginx -p 80:80 lukavavetic/typeqast-nginx:1.0.0`
  - 

Hello world!

  - `docker image inspect lukavavetic/typeqast-nginx1.0.0`

- "Layers": [
    "sha256:
  08249ce7456a1c0613eafe868aed936a284ed9f1d6144f7d2d08c514974a2af9",
    "sha256:
  675a292a738a0826b2e5e435f3700b7901aee89a58701cee7f9123bd1106d034",
    "sha256:
  7cde5b916efcbe6847240b499e209f500180e948fc3f94c21d18625e0e4d9238",
    "sha256:
  b390cd0ded64971ec2b5e9d802dfbae7ae55111bc7138528fbd0ed5af7ac5d35",
    "sha256:
  17ab32f939a6dff8be87755363c5d418aab66a189d7e91492a87e7cc04e455da",
    "sha256:
  fff6fff137a37ee2935a68007336223a68d65afd1babfbbb0a0a25933fe9f40a",
    "sha256:
  6994c27835f9eacfeb3ae709ba2e3f6635ddb06e5b25cbf22e2b2eae6554f4f5",
    "sha256:
  54bd31654ea4f5eb15dc0bac0270924e90d5268937486b833bf82aca617af4d0"
  ]

- let's change name of the index.html file to test-index.html , rewrite our Dockerfile and rebuild the image
  - `COPY index.html test-index.html`
  - `docker image inspect lukavavetic/typeqast-nginx1.0.0`

```
"Layers": [
    "sha256:
08249ce7456a1c0613eafe868aed936a284ed9f1d6144f7d2d08c51497
4a2af9",
    "sha256:
675a292a738a0826b2e5e435f3700b7901aee89a58701cee7f9123bd11
06d034",
    "sha256:
7cde5b916efcbe6847240b499e209f500180e948fc3f94c21d18625e0e
4d9238",
    "sha256:
b390cd0ded64971ec2b5e9d802dfbae7ae55111bc7138528fbd0ed5af7
ac5d35",
    "sha256:
17ab32f939a6dff8be87755363c5d418aab66a189d7e91492a87e7cc04
e455da",
    "sha256:
fff6fff137a37ee2935a68007336223a68d65afd1babfbbb0a0a25933f
e9f40a",
    "sha256:
6994c27835f9eacfeb3ae709ba2e3f6635ddb06e5b25cbf22e2b2eae65
54f4f5",
    "sha256:
3aef041ed9e1ba21de981fd564dfee69c78d4bf54a87d317ae4bfd369f
ec23e7"
]
```

- `docker image build -t lukavavetic/typeqast-nginx:1.0.0 .`
- ```
  Sending build context to Docker daemon  3.072kB
  Step 1/4 : FROM nginx:1.22.0
   ---> b3c5c59017fb
  Step 2/4 : RUN apt-get update && apt-get install curl iputils-ping -y
   ---> Using cache
   ---> 0ef618321981
  Step 3/4 : WORKDIR /usr/share/nginx/html
   ---> Using cache
   ---> c7c7308472e1
  Step 4/4 : COPY index.html test-index.html
   ---> 4e599397cb95
  Successfully built 4e599397cb95
  Successfully tagged lukavavetic/typeqast-nginx:1.0.0
  ```
- all layers which were not changed are being pulled from the cache

- study some docker images:
  - https://github.com/nginxinc/docker-nginx/blob/d4a47bc6602d3a1412dad48a8513b83805605ef3/mainline/alpine/Dockerfile
  - https://github.com/docker-library/mysql/blob/b22945da0ed9f152485cc68ff7565204e8d37db4/8.0/Dockerfile.debian
  - https://github.com/bitnami/bitnami-docker-php-fpm/blob/8.1.7-debian-11-r8/8.1/debian-11/Dockerfile

---

CI/CD
  ```

Exercise #4:

- https://github.com/lukavavetic/php-chapter-1
- let's create a basic Dockerfile for our application
  - https://github.com/lukavavetic/php-chapter-1/blob/main/Dockerfile
  - build the docker image locally
    - `docker image build -t lukavavetic/php-chapter-1 .`
  - push image to Docker Hub
    - `docker image push lukavavetic/php-chapter-1`
- now, let's use our docker image to build a container for running checks (lint, tests, static analysis) via GitHub actions
  - https://github.com/lukavavetic/php-chapter-1/blob/main/.github/workflows/main.yml
  - `checks` is the first job whose role is to run different checks for our application such as lint, PHPUnit, phpstan
  -

```yaml
on:
  push:
    branches: [ "main" ]
jobs:
    checks:
      runs-on: ubuntu-latest
      container: lukavavetic/php-chapter-1
      steps:
      - name: Obtain Latest Git ONLY within container for checkout
        run: |
          apt-get update
          apt-get install -y git

      - name: Checkout Repo Action
        uses: actions/checkout@v2

      - name: Run Composer update
        run: composer update

      - name: Run PHP Lint
        run: vendor/bin/phplint

      - name: Run PHP tests
        run: vendor/bin/phpunit

      - name: Run PHP Stan
        run: vendor/bin/phpstan analyse
```

- If the first job has run successfully then start the second job `docker-image`
  - https://github.com/lukavavetic/php-chapter-1/blob/main/.github/workflows/main.yml
  - first, let's create Github secrets so we can enable Github action to log in our Docker Hub
    - go to repository's `settings-> secrets -> actions -> new repository secret`
      - create `DOCKER_USER` and `DOCKER_SECRET`
  - the goal of this job is to log in to Docker Hub as your registry, build the docker image and push it to the registry
  -

```yaml
docker-image:
  needs: checks
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2

    - name: Docker Login
      env:
        DOCKER_USER: ${{ secrets.DOCKER_USER }}
        DOCKER_SECRET: ${{ secrets.DOCKER_SECRET }}
      run: docker login -u $DOCKER_USER -p $DOCKER_SECRET

    - name: Docker Image Build
      run: docker image build -t lukavavetic/php-chapter-1

    - name: Docker Image Push
      run: docker image push lukavavetic/php-chapter-1
```