DOCKER DOC

https://www.simpliaxis.com/resources/how-does-docker-and-kubernetes-work-together

In this age of rapid developments, more and more organizations are embarking on the cloud-native DevOps journey and migrating their infrastructure and architecture to keep pace with the cloud-native and data-driven era. This means an increasing number of digital transformation tools are being introduced frequently. As companies work toward matching this ongoing technology trend, words like cloud computing, containerization, and container orchestration have become commonplace, and their interest is increasing every day. But when we talk of cloud-native, it is very difficult to not mention Docker and Kubernetes. These are the two basic tools that facilitate a company's transformation strategy. So, you must learn to work with them comfortably. These are revolutionary tools and have changed the way we create and deploy software at scale. Let us first understand what these tools are.

Docker

Docker is primarily a platform for software development. It is a type of virtualization technology. With Docker, developing and deploying applications inside virtual containerized environments becomes easier. Docker is compatible with almost every kind of machine which means the software remains system-agnostic or you can say that the applications are independent of the Operating System (OS). With Docker, you have to do less work to develop the application and it becomes easy to deploy and maintain it thus making the use of the application simpler. With Docker, developers find it easier to run their applications in a similar environment without any dependencies because Docker containers have native operating system libraries. Earlier, the developer had no option but to send their code to the tester but because of many dependency issues, the code would not run on the tester's machine although it ran well on the developer's machine. This created a lot of confusion. But with Docker, this problem has been removed as both the developer and the tester now have the same system that runs on the Docker container. Being an open-source tool, Docker makes it possible for developers to make applications anywhere.

Developers can create Dockerfile with Docker. They use it to define the requirements of the application in it. It describes the process of building the software. When you give it to the "Docker build" command, an entrenched image is created. This image is usually the blueprint or template of the application and shows the dependencies. When the Docker image is run, it creates a Docker container which is just a run time instance of the Docker image. It can run on any environment like Mac, Windows PC, Linux, Cloud, etc. Docker also provides an online cloud repository called Docker Hub where you can store all the Docker images that you create. The Docker containerization consists of infrastructure and an operating system but lacks Hypervisor as in virtualization. The operating system on which the process runs is called Docker Daemon. This OS enables running containers on the system and manages the Docker images and all Docker commands. Therefore, we can say that the main aim of Docker is to package and containerize the applications and send them to and run them anywhere any number of times.

Kubernetes

There are certain specific tasks that need to be handled in the Docker cluster environment like scheduling, communications, and scalability. And an orchestration tool has to be used to handle such tasks. Kubernetes is one such platform that is more popular around the globe. Kubernetes is again an open-source container orchestration platform that is used to manage, automate, and scale containerized applications. Though Docker Swarm is also an orchestration tool, Kubernetes is taken as the standard because it is more flexible and has more scaling capacity. It takes the primary container responsibilities like container deployment, scaling, healing, and load balancing of the container. Kubernetes cluster is a working deployment of Kubernetes. This Kubernetes cluster can be seen as two different parts which are the control plane or the master node and the compute machines or the worker node. Further, each of these nodes could be a physical or virtual machine and is its own Linux environment. Pods, consisting of containers are run by each node.

Since each node can be a physical or virtual machine, you can manage thousands of containerized applications in different types of environments like physical, virtual, or even hybrid machines with Kubernetes. To put it simply, Kubernetes ensures that each container is in its right place and that all of them can work together. It is not the processes that define Kubernetes but it is defined by states. It makes sure that once it has defined a pod, it keeps running always. In the case of a container getting killed or going down, Kubernetes attempts to start a new one. Kubernetes is used by companies for automating the deployment and management of containerized applications.

Docker and Kubernetes together

Now, we have made it amply clear that Docker helps you in creating the containers while Kubernetes facilitates their management at run time. While Docker packages and ships the application, Kubernetes helps you to deploy and scale the application. So, it can be inferred that both go together. One is needed for the other to work smoothly. Smaller companies or even startups that have a lesser number of containers, can generally manage them without using Kubernetes. But wherever the number of containers is large, organizations need Kubernetes to manage them. This is because in larger companies the infrastructure is bigger and as the infrastructure becomes bigger the number of containers also rises and it becomes harder to manage them. So, the need to employ Kubernetes arises.

When Docker and Kubernetes are used together they act as facilitators for digital transformation and this transformation becomes a lot easier as they serve as tools for modern cloud architecture. Today, it has become normal for companies to use Docker and Kubernetes for faster development, deployment, and release of applications. Just running the containers in production is not enough, they have to be regulated. Kubernetes has some really strong features that enable easier regulation of the containers. These include auto-scaling, health checks, and load balancing. All these are critical for managing the container lifecycle. Kubernetes keeps checking if the deployment has been done as per the YAML definition. Although Docker and Kubernetes work better together, it has been strongly recommended that at the time of building stacks, you should clearly understand the differences between Docker and Kubernetes. Both of them have different scopes and can be used separately also.

If seen from the viewpoint of software development, Docker's strong point is development. This means its use is to configure, create, and distribute the containers and use Docker Hub as an image registry. Kubernetes, on the other hand, works really well in operations where you can use your existing Docker containers and handle its complexities related to deployment, scaling, networking, and monitoring.

In short, you can use Docker to isolate your containers and pack them but with dependencies. But Kubernetes will help you in the deployment and orchestration of these containers. But when you use both of these together, productivity increases and so does the confidence of everyone. Organizations use microservices to change their monolithic applications into smaller components so that they can be packaged and deployed separately. The purpose of these microservices is to help you in delivering more scalable, agile, and strong applications that you can update, change, and redeploy faster. And this is where tools such as Docker and Kubernetes help companies to do these activities in an easier and faster way. They provide the necessary flexibility to the companies to deploy and scale the applications as necessary. The use of Docker and Kubernetes is spreading rapidly and more and more companies are adopting them and using them in huge scaling of production. So, owing to their usefulness, companies all around the world are integrating these two platforms for their containers and microservices.

Docker:

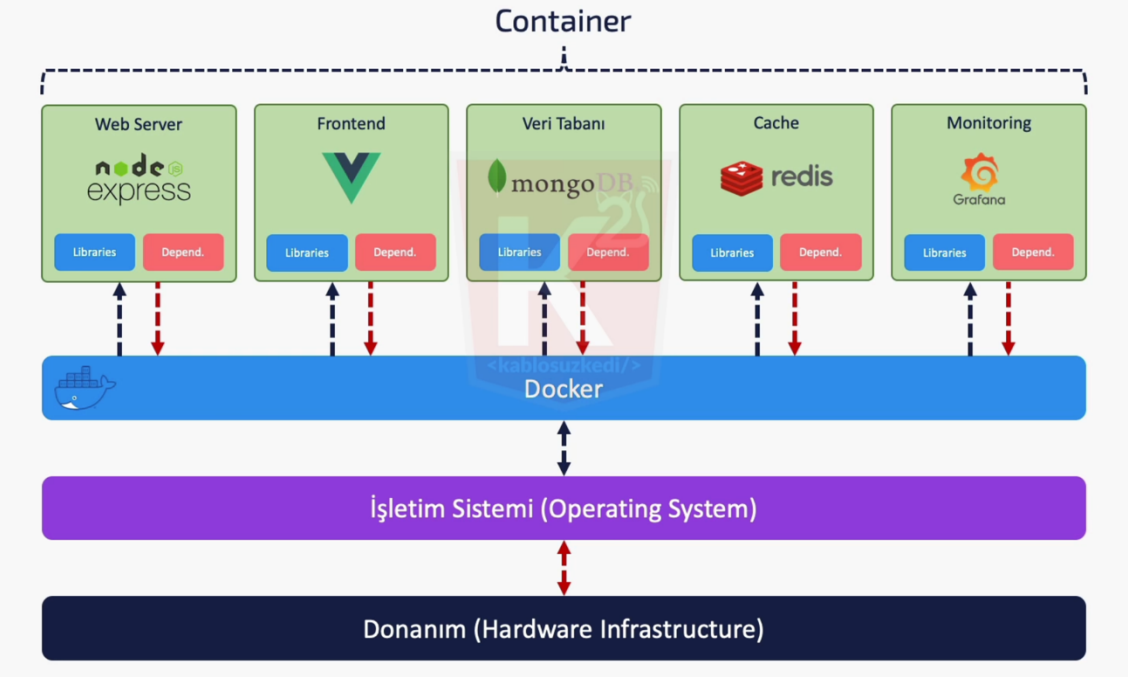
Makes environment the same so apps do not effected by different dependicies and libraries.
Like one OS divided for each services and works separately (containers shares Kernel).
Despite of VM Hypervisor docker shares OS for each container.
These two VM Hypervisor and Docker used together on Cloud techs a lot.
Provides exact same environment for development, test, and production
developer and devops can work seperatly (dockerfile defines all configurations so devops only need to run it, no complexity)

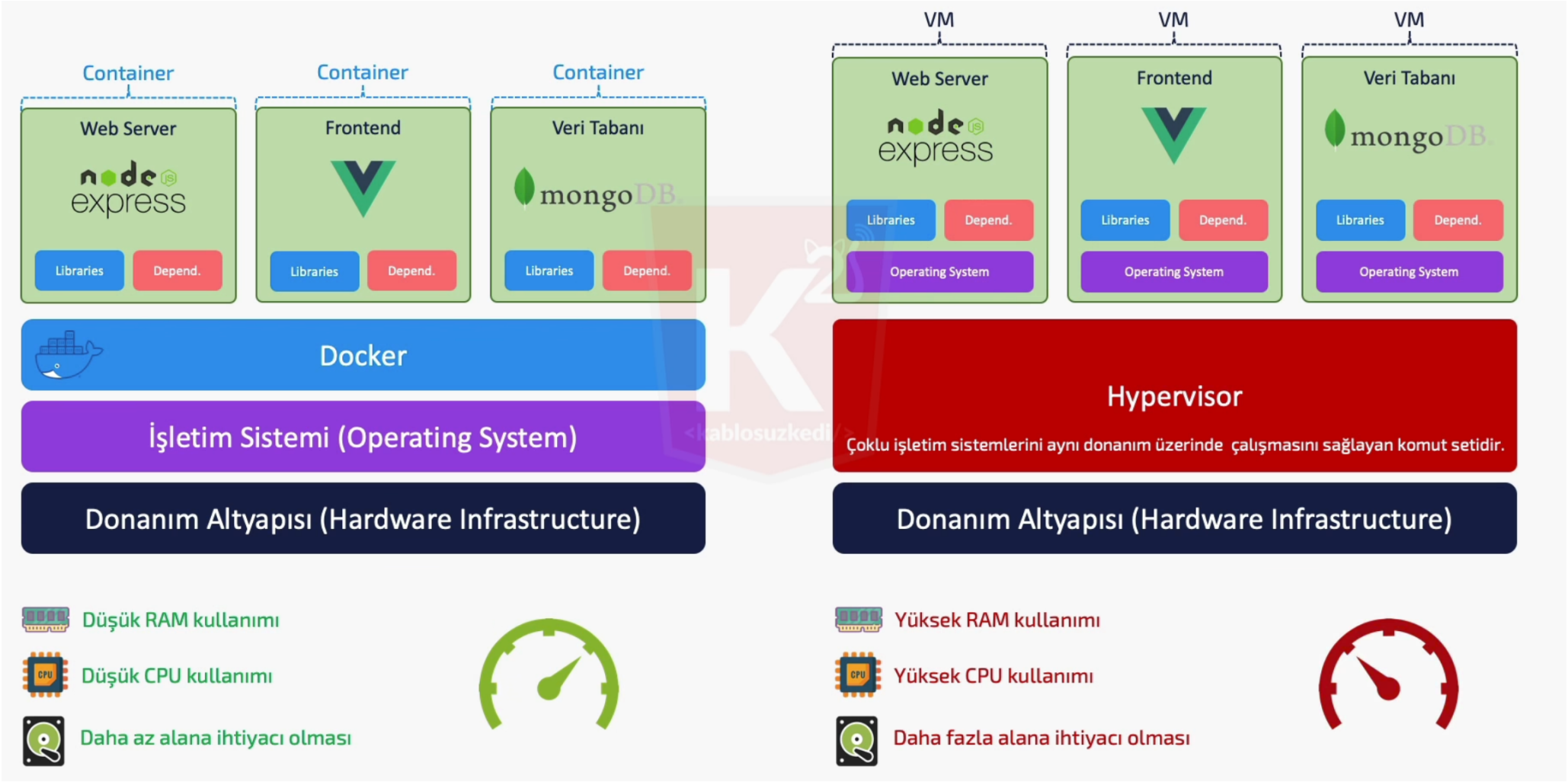
Container: (Docker uses LXC container type)

A package of code and dependencies, standardized unit of software.
Containers shares Kernel.
Containers have their own procesess, services and network.
Containers works on the same OS.
Containers are all isolated environments.
Procesess of running images. Each run are different but images the same.
we can run programs as standalone without any installation

Images:

Images are READ-ONLY
Template, plan or blueprints for containers that contains OS, Apps, others (its kind of exe for me like you install containers by using them).
Images are placed on Docker Hub (docker run ubuntu, docker run mongo).
We can create our own images of our applications by Dockerfiles.
When building images layer based architecture used so the same steps of Dockerfile instructions will be cached and will be so fast.




Container
Container
Container
Web Server
Frontend
Veri Tabanı
express
mongoDB.
Libraries
Depend.
Libraries
Depend.
Libraries
Depend.
Docker
İşletim Sistemi (Operating System)
Donanım Altyapısı (Hardware Infrastructure)
Düşük RAM kullanımı
Düşük CPU kullanımı
Daha az alana ihtiyacı olması
VM
VM
VM
Web Server
Frontend
Veri Tabanı
express
mongoDB.
Libraries
Depend.
Operating System
Libraries
Depend.
Operating System
Libraries
Depend.
Operating System
Hypervisor
Çoklu işletim sistemlerini aynı donanım üzerinde çalışmasını sağlayan komut setidir.
Donanım Altyapısı (Hardware Infrastructure)
Yüksek RAM kullanımı
Yüksek CPU kullanımı
Daha fazla alana ihtiyacı olması

Commands: (some important ones needed to known) https://docs.docker.com/engine/reference/commandline/logs/#examples

`docker pull ubuntu –>` downloads ubuntu from dockerhub to local docker engine
`docker pull redis`
`docker run imageName|containerName|containerID`
`docker run redis –>` downloads images and runs as a container instance of redis (we can run multiple instances)
`docker images –>` lists docker images
`docker run –it ubuntu –>` runs ubuntu with interactive terminal (so we can interact with ubuntu)
`docker ps –>` lists running containers
`docker ps –a –>` lists all containers
`docker run –it ––name named_ubuntu ubuntu –>` gives a name to container
`docker start named_ubuntu –>` starts in detached mode as default
`docker stop named_ubuntu –>` stops
`docker rm named_ubuntu –>` deletes stopped containers
`docker rmi ubuntu –>` deletes images
`docker ps –aq –>` lists all container ids
`docker rm $(docker ps –aq) –>` deletes all stopped containers
`docker run redis:5 –>` downloads tagged 5 redis and runs as container
`docker container prune –>` deletes all stopped containers

`docker image tag ubuntu tagged-ubuntu:v1 ->` tags an image

```
PS /Users/zoothii> docker image tag ubuntu tagged-ubuntu:v1
Error response from daemon: No such image: ubuntu:latest
PS /Users/zoothii> docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
005e2837585d: Pull complete
Digest: sha256:6042500cf4b44023ea1894effe7890666b0c5c7871ed83a97c36c76ae560bb9b
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
PS /Users/zoothii> docker images
REPOSITORY                           TAG                          IMAGE ID       CREATED         SIZE
ubuntu                               latest                       da935f064913   11 days ago     69.3MB
k8s.gcr.io/kube-apiserver            v1.25.2                      0b1fb9b45fa3   15 months ago   123MB
k8s.gcr.io/kube-proxy                v1.25.2                      68348500321c   15 months ago   58MB
k8s.gcr.io/kube-controller-manager   v1.25.2                      c6c296d44024   15 months ago   113MB
k8s.gcr.io/kube-scheduler            v1.25.2                      873dc124ec69   15 months ago   49.3MB
registry.k8s.io/pause                3.8                          4e42fb3c9d90   18 months ago   514kB
k8s.gcr.io/etcd                      3.5.4-0                      8e041a3b0ba8   19 months ago   179MB
registry.k8s.io/coredns/coredns      v1.9.3                       b19406328e70   19 months ago   47.7MB
docker/desktop-vpnkit-controller     v2.0                         2edf9c994f19   2 years ago     19.2MB
docker/desktop-storage-provisioner   v2.0                         c027a58fa0bb   2 years ago     39.8MB
amigoscode/kubernetes                springboot-react-fullstack-v1   778d5ed2c7f9   54 years ago    509MB
PS /Users/zoothii> docker image tag ubuntu tagged-ubuntu:v1
PS /Users/zoothii> docker images
REPOSITORY                           TAG                          IMAGE ID       CREATED         SIZE
tagged-ubuntu                        v1                           da935f064913   11 days ago     69.3MB
ubuntu                               latest                       da935f064913   11 days ago     69.3MB
k8s.gcr.io/kube-apiserver            v1.25.2                      0b1fb9b45fa3   15 months ago   123MB
k8s.gcr.io/kube-proxy                v1.25.2                      68348500321c   15 months ago   58MB
k8s.gcr.io/kube-controller-manager   v1.25.2                      c6c296d44024   15 months ago   113MB
k8s.gcr.io/kube-scheduler            v1.25.2                      873dc124ec69   15 months ago   49.3MB
registry.k8s.io/pause                3.8                          4e42fb3c9d90   18 months ago   514kB
k8s.gcr.io/etcd                      3.5.4-0                      8e041a3b0ba8   19 months ago   179MB
registry.k8s.io/coredns/coredns      v1.9.3                       b19406328e70   19 months ago   47.7MB
docker/desktop-vpnkit-controller     v2.0                         2edf9c994f19   2 years ago     19.2MB
docker/desktop-storage-provisioner   v2.0                         c027a58fa0bb   2 years ago     39.8MB
amigoscode/kubernetes                springboot-react-fullstack-v1   778d5ed2c7f9   54 years ago    509MB
PS /Users/zoothii>
```

`docker run -d --myredis redis ->` runs in detached mode (runs in background)
`docker attach myredis ->` puts attached mode

```
PS /Users/zoothii> docker run -d --name myredis redis
66d5268cfaee6b2018a7c5293004cdce4cff6ed4d1777a7f9611311521e4f126
PS /Users/zoothii> docker attach myredis
```

`docker logs myredis` -> shows logs

```
PS /Users/zoothii> docker logs myredis
1:C 23 Dec 2023 14:18:25.228 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 23 Dec 2023 14:18:25.228 * Redis version=7.2.3, bits=64, commit=00000000, modified=0, p
1:C 23 Dec 2023 14:18:25.228 # Warning: no config file specified, using the default config.
1:M 23 Dec 2023 14:18:25.228 * monotonic clock: POSIX clock_gettime
1:M 23 Dec 2023 14:18:25.229 * Running mode=standalone, port=6379.
1:M 23 Dec 2023 14:18:25.229 * Server initialized
1:M 23 Dec 2023 14:18:25.229 * Ready to accept connections tcp
1:signal-handler (1703341589) Received SIGINT scheduling shutdown...
1:signal-handler (1703341589) You insist... exiting now.
PS /Users/zoothii>
```

`docker inspect myredis` -> shows the details for example volume mapping details. (can be used for images too.)

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "myredis-data",
        "Source": "/var/lib/docker/volumes/myredis-data/_data",
        "Destination": "/data",
        "Driver": "local",
        "Mode": "z",
        "RW": true,
        "Propagation": ""
    }
],
```

```
"HostConfig": {
    "Binds": [
        "myredis-data:/data"
    ],
    "ContainerIDFile": "",
    "LogConfig": {
        "Type": "json-file",
        "Config": {}
    },
    "NetworkMode": "default",
    "PortBindings": {
        "6379/tcp": [
            {
                "HostIp": "",
                "HostPort": "6379"
            }
        ]
    },
```

PORT MAPPING

`docker run -p OUT_PORT:IN_PORT -d --name myredis redis` -> out_port (port we will use in local) in_port(port used in docker engine for redis default 6379)

`docker run -p 6379:6379 -d --name myredis redis` -> maps docker engine redis 6379 port to 6379 for local machine

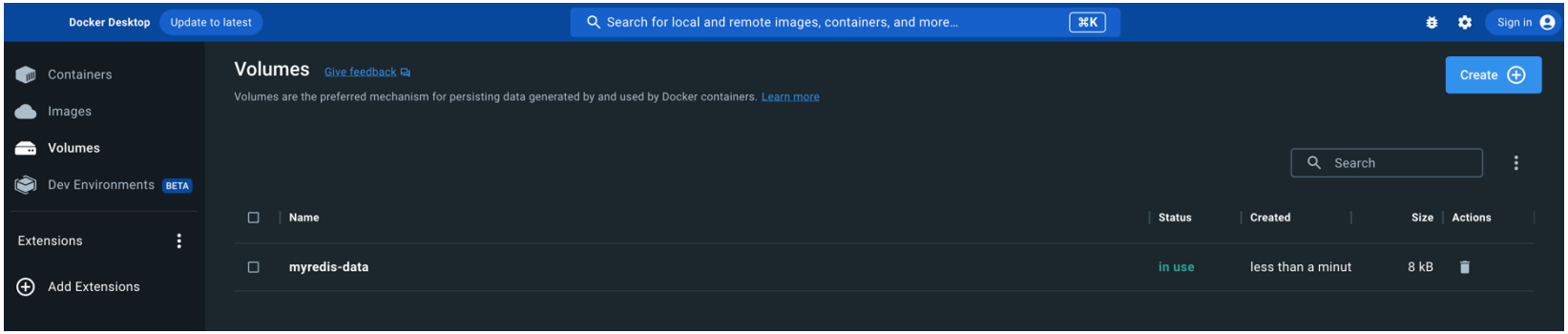
`docker run -p 5858:6379 -d --name myredis redis` -> maps docker engine redis 6379 port to 5858 for local machine

VOLUME MAPPING

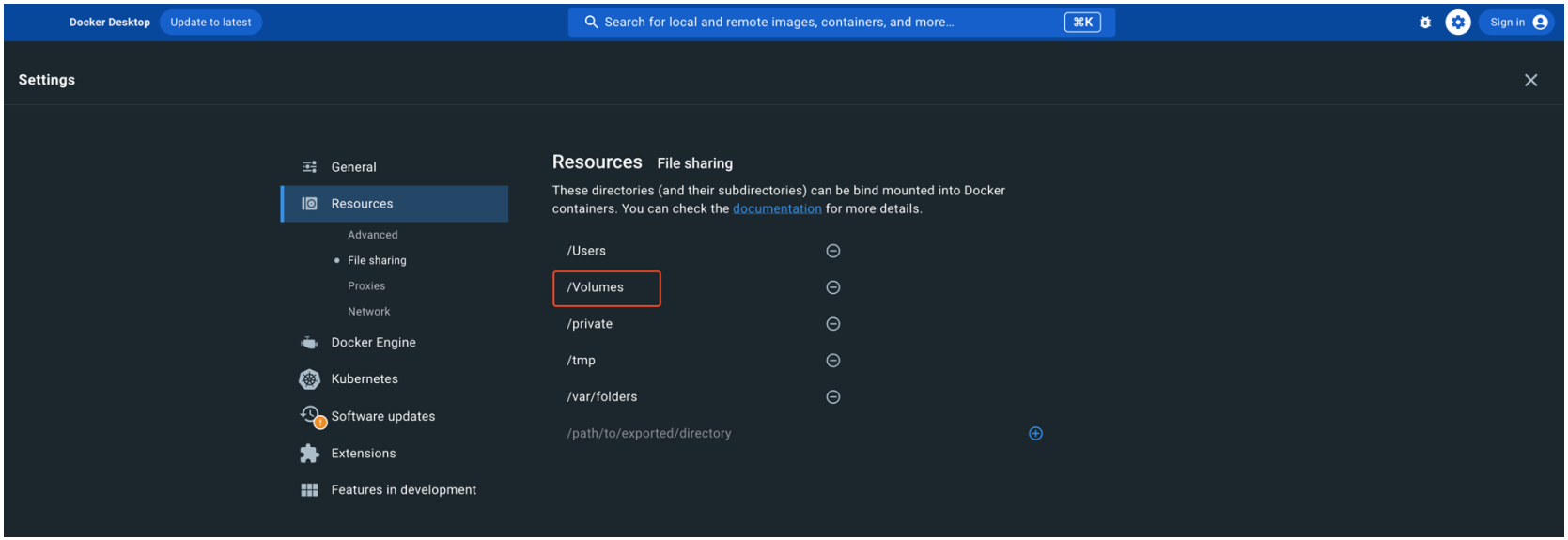
docker engine runs containers as stateless (means there is no data stored, when we stop container its data will be removed). Volumes used to solved this problem. To do that when we run a container we should show the folder path which data will be stored on docker engine.

`docker run -v myredis-data:/data -p 6379:6379 -d --name myredis redis` -> redis data volume mapping into docker engine myredis-data volume

```
PS /Users/zoothii/Workspace/devops/docker> docker run -v myredis-data:/data -p 6379:6379 -d --name myredis redis
d2ef1feb9f198de373d19fea48faf9f557cde24e8498e78e91b56b7408571a9a
PS /Users/zoothii/Workspace/devops/docker>
```



The path should be shared from the host (Docker Engine), if we don’t define a path (such as myredis-data it creates this folder under Volumes (/var/lib/docker/volumes/).).



`docker run -e MYSQL_ROOT_PASSWORD=zoothii -d --name mymysql mysql ->` -e environment variable

```
PS /Users/zoothii/Workspace/devops/docker> docker run mysql
2023-12-26 06:34:28+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.2.0-1.el8 started.
2023-12-26 06:34:28+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-12-26 06:34:28+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.2.0-1.el8 started.
2023-12-26 06:34:28+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
    You need to specify one of the following as an environment variable:
    - MYSQL_ROOT_PASSWORD
    - MYSQL_ALLOW_EMPTY_PASSWORD
    - MYSQL_RANDOM_ROOT_PASSWORD
PS /Users/zoothii/Workspace/devops/docker> docker run -e MYSQL_ROOT_PASSWORD=zoothii -d --name mymysql mysql
71190b067d53350215802e5a6884a6c422eb79995632e92a03dc3f64bf215ebd
PS /Users/zoothii/Workspace/devops/docker>
```

CONTAINER LINKS

Connecting MySQL and PhpMyAdmin using links.
First run the container which will be connected to (MySQL for this case).

```
docker run -e MYSQL_ROOT_PASSWORD=zoothii -p 3306:3306 -v mymysql-data: /etc/mysql/conf.d --name mymysql -d mysql
```

then run the container which will connect (PhpMyAdmin for this case).

```
docker run --link mymysql:db --name myphpmyadmin -p 8000:80 -d phpmyadmin
```

```
PS /Users/zoothii/Workspace/devops/docker> docker run -e MYSQL_ROOT_PASSWORD=zoothii -p 3306:3306 -d --name mymysql mysql
b4e0f74213ef7c28971fa3938c405c4690845599f9b8c356954c89a9d45cbe5d
PS /Users/zoothii/Workspace/devops/docker> docker run --link mymysql:db --name myphpmyadmin -p 8000:80 -d phpmyadmin
ffcf165f14b979f0ad641e6580dd40d3bf2603d68c2ca00f028ad9276730114f
PS /Users/zoothii/Workspace/devops/docker>
```

for **--link mymysql:db** (db alias comes from here they defined it as db for PhpMyAdmin, instead of saying connect to **localhost** in local machine we say connect to **db** in docker engine).

Example `docker-compose.yml` for `phpmyadmin`:

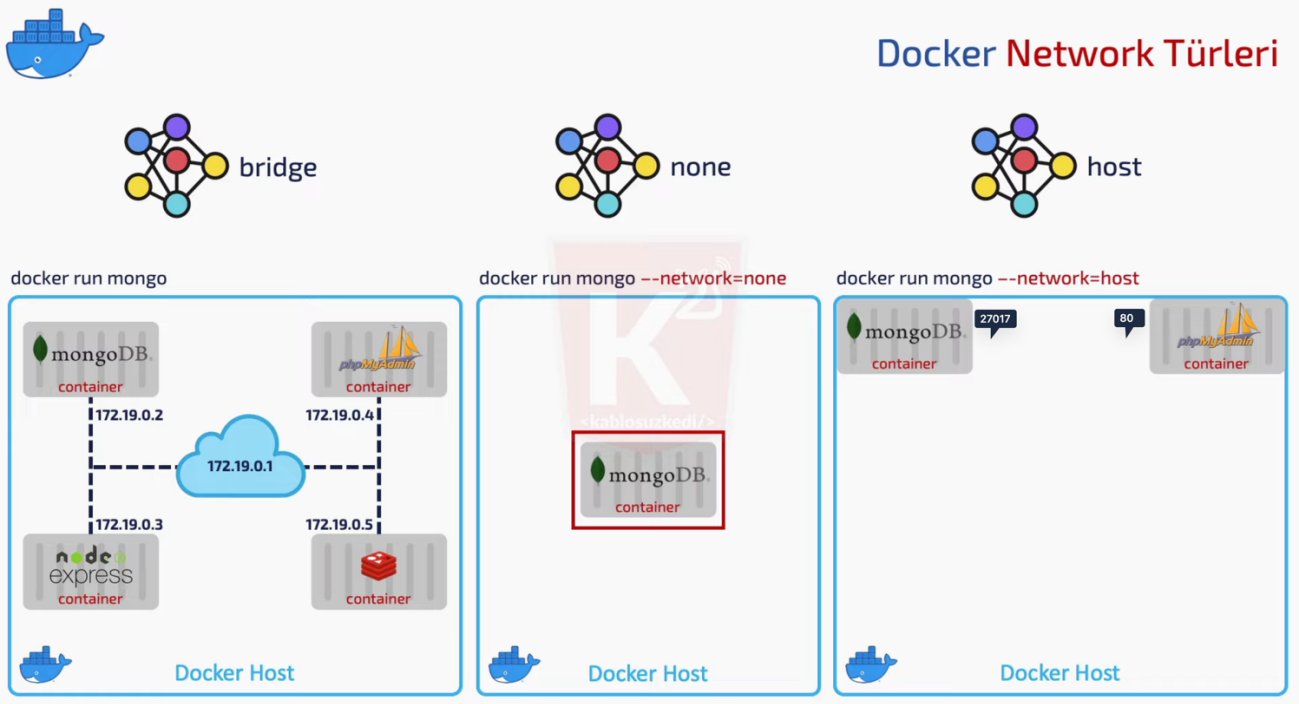
```
version: '3.1'

services:
  db:
    image: mariadb:10.6
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: notSecureChangeMe

  phpmyadmin:
    image: phpmyadmin
    restart: always
    ports:
      - 8080:80
    environment:
      - PMA_ARBITRARY=1
```

DOCKER NETWORKS

Bridge: Used as default, containers can have connections using gateway

None: Non accessable (in and out)

Host: Can be accessed using docker host ip address and default port



User Defined -> we will make an example using gkandemir todo-app and mongo db

```
docker network create --driver bridge --subnet 182.18.0.1/24 --gateway 182.18.0.1 mynetwork
```

```
PS /Users/zoothii/Workspace/devops/docker> docker network create --driver bridge --subnet 182.18.0.1/24 --gateway 182.18.0.1 mynetwork
d1532c6ee6f310f7cceab0f028251bc59cdc164a5bf9eb00bf8d0fefdc47fe80
PS /Users/zoothii/Workspace/devops/docker> docker network ls
NETWORK ID     NAME        DRIVER    SCOPE
ea8fedecb49d   bridge      bridge    local
c66a505a84b6   host        host      local
d1532c6ee6f3   mynetwork   bridge    local
bde048a89d02   none        null      local
PS /Users/zoothii/Workspace/devops/docker>
```

Note: For example to connect database (PostgreSQL) from API (a Spring Java Web APP) we defined services as **api** and **postgresqldb** in **compose.yaml** file so to connect to db from api we used **postgresqldb:5432** instead of **localhost:5432** in datasource.url in application.properties because in docker engine (docker host) its defined as postgresqldb because **docker does not guarantee the ip addresses of the containers will be the same in its network** so pushes us to use container names instead. (This is my project I created and used it to understand better)

Code Blame 32 lines (31 loc) · 696 Bytes Code 55% faster with GitHub Copilot

```yaml
version: '3.9'

services:
  api:
    image: '[redacted]api:latest'
    container_name: [redacted]-api
    build:
      context: ./api[redacted]
      dockerfile: Dockerfile
    restart: unless-stopped
    ports:
      - "4024:4024"
    depends_on:
      - postgresqldb
  postgresqldb:
    image: postgres:14.2-alpine
    container_name: posgresqldb
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=[redacted]
      - POSTGRES_PASSWORD=[redacted]
      - POSTGRES_DB=[redacted]
```

Code | Blame | 40 lines (40 loc) · 1.95 KB | Code 55% faster with GitHub Copilot

```
1  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
2  spring.jpa.hibernate.ddl-auto=create
3  spring.jpa.generate-ddl=true
4  spring.datasource.url=jdbc:postgresql://postgresqldb:5432/[redacted]
5  spring.datasource.username=[redacted]
6  spring.datasource.password=[redacted]
```

`docker run --name mongo-server --net mynetwork -v mymango-data:/etc/mongo -d mongo ->` we don't need port binding because we will not used this mongo on our local mongodb and todo-app will connect theirself on docker engine using our own custom network mynetwork.

`docker run --name todo-app --net mynetwork -p 3000:3000 -d gkandemir/todo-app`

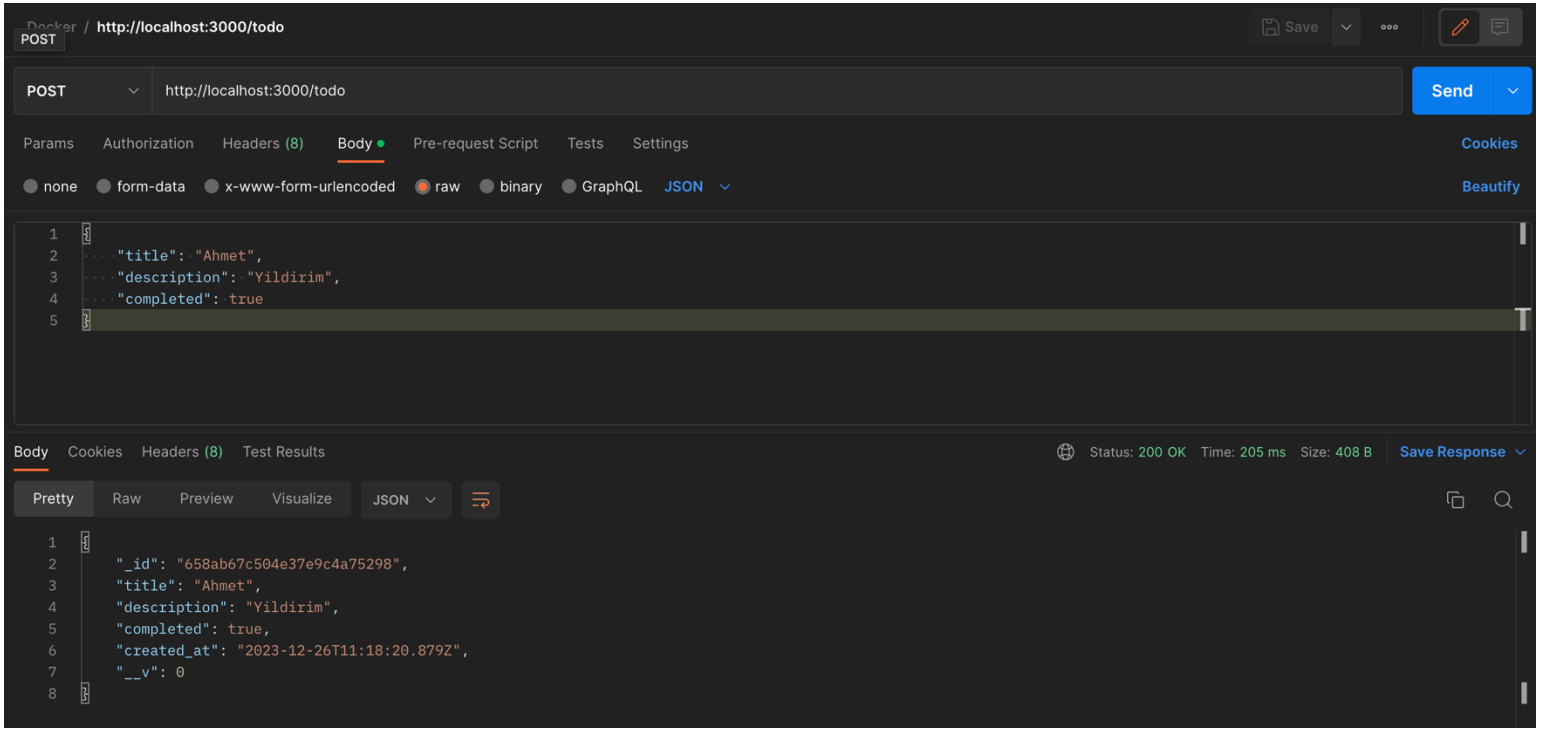
```
PS /Users/zoothii/Workspace/devops/docker> docker run --name mongo-server --net mynetwork -v mymango-data:/etc/mongo -d mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
005e2837585d: Already exists
e60b3ed21100: Pull complete
81fcf60fea85: Pull complete
05da3aee34af: Pull complete
c0f2bf503f32: Pull complete
e7e4d672dd05: Pull complete
f1bc5c150557: Pull complete
3c174931be51: Pull complete
500987cab84a: Pull complete
Digest: sha256:d14158139a0bbc1741136d3eded7bef018a5980760a57f0014a1d4ac7677e4b1
Status: Downloaded newer image for mongo:latest
9144013a4ff3e74c460ecf5354d1b3a7d3ee3afee197931670bc43a333a9f6a3
PS /Users/zoothii/Workspace/devops/docker> docker run --name todo-app --net mynetwork -p 3000:3000 gkandemir/todo-app
Unable to find image 'gkandemir/todo-app:latest' locally
latest: Pulling from gkandemir/todo-app
cae7303ade7f: Pull complete
8dd505804d99: Pull complete
2760faff36de: Pull complete
a4244e00f89b: Pull complete
b5b2d4487bed: Pull complete
08bb2fd22d85: Pull complete
f775e912435f: Pull complete
8cf8f1291e57: Pull complete
Digest: sha256:093ed7cd92401f6b2a7eff42043b923332fe5c5bc380227c5dc148b43dd29f6b
Status: Downloaded newer image for gkandemir/todo-app:latest
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was re
quested
Sunucu çalışıyor... MongoDB'ye bağlanılacak..
MongoDB'ye bağlantı başarılı!
```

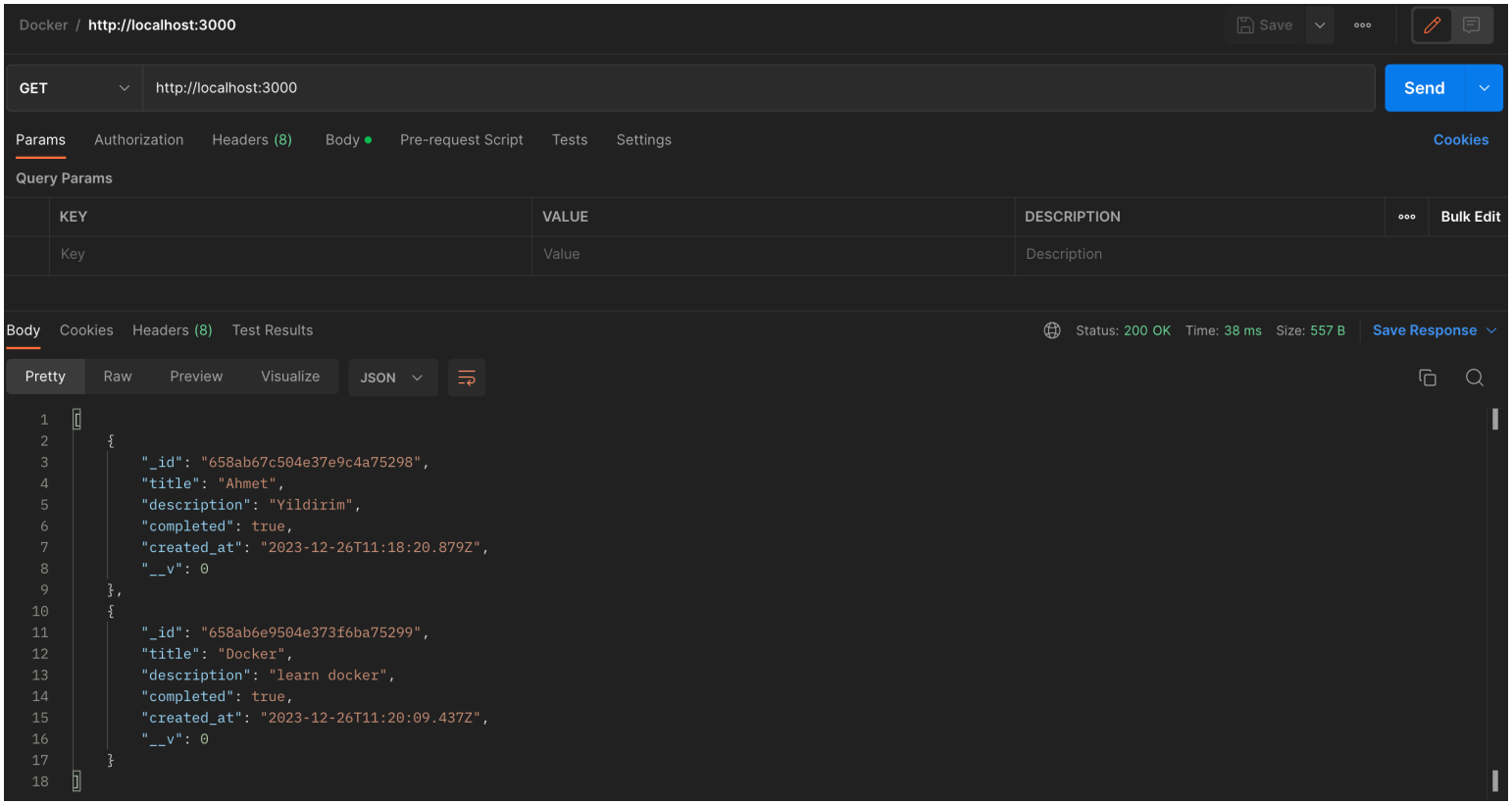
`docker inspect mongo-server/todo-app ->` here we can see mongo-server and todo-app uses our custom network


```
            "Networks": {
                "mynetwork": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": [
                        "9144013a4ff3"
                    ],
                    "NetworkID": "d1532c6ee6f310f7cceab0f028251bc59cdc164a5bf9eb
00bf8d0fefdc47fe80",
                    "EndpointID": "3a8b17c48ea8e1c306940ecb09a78ab3f872edb3bb7b7
fb832d5b17666434154",
                    "Gateway": "182.18.0.1",
                    "IPAddress": "182.18.0.2",
                    "IPPrefixLen": 24,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:b6:12:00:02",
                    "DriverOpts": null
                }
            }
```

```
            "Networks": {
                "mynetwork": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": [
                        "d1eda5b4660e"
                    ],
                    "NetworkID": "d1532c6ee6f310f7cceab0f028251bc59cdc164a5bf9eb0
0bf8d0fefdc47fe80",
                    "EndpointID": "e349dc44b686438dfda60274bbeea20567966e27d7bbbd
16c1fc0357558c315a",
                    "Gateway": "182.18.0.1",
                    "IPAddress": "182.18.0.3",
                    "IPPrefixLen": 24,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:b6:12:00:03",
                    "DriverOpts": null
                }
            }
```


now we can use todo-app




Docker / http://localhost:3000
Save
GET
http://localhost:3000
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 38 ms
Size: 557 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "658ab67c504e37e9c4a75298",
"title": "Ahmet",
"description": "Yildirim",
"completed": true,
"created_at": "2023-12-26T11:18:20.879Z",
"__v": 0
},
{
"_id": "658ab6e9504e373f6ba75299",
"title": "Docker",
"description": "learn docker",
"completed": true,
"created_at": "2023-12-26T11:20:09.437Z",
"__v": 0
}
]

CREATING DOCKER IMAGE (DOCKERFILE)

node example using ubuntu container (we will create simple nodejs app under ubuntu under docker) (ubuntu 22.04.3 / nodejs 21.5.0)

```
docker run --name myubuntu -it ubuntu
```

inside ubuntu do these steps

```
1  apt-get install
2  apt-get update
3  apt-get install curl -y
4  curl -SLO https://deb.nodesource.com/nsolid_setup_deb.sh
5  chmod 500 nsolid_setup_deb.sh
6  ./nsolid_setup_deb.sh 21
7  apt-get install nodejs -y
8  cd opt/
9  mkdir node-app
10 cd node-app/
11 node index.js
```

```
root@795fe879b7db:/opt/node-app# history
    1  apt-get install
    2  apt-get update
    3  apt-get install curl -y
    4  curl -SLO https://deb.nodesource.com/nsolid_setup_deb.sh
    5  chmod 500 nsolid_setup_deb.sh
    6  ./nsolid_setup_deb.sh 21
    7  apt-get install nodejs -y
    8  clear
    9  node -v
   10  cd opt/
   11  mkdir node-app
   12  cd node-app/
   13  echo 'console.log("node-app from ubuntu...");' > index.js
   14  node index.js
   15  history
root@795fe879b7db:/opt/node-app#
```

when we run index.js with node inside our ubuntu this is the result

```
root@795fe879b7db:/# node -v
v21.5.0
root@795fe879b7db:/# cd opt/
root@795fe879b7db:/opt# mkdir node-app
root@795fe879b7db:/opt# cd node-app/
root@795fe879b7db:/opt/node-app# echo 'console.log("node-app from ubuntu...");' > index.js
root@795fe879b7db:/opt/node-app# node index.js
node-app from ubuntu...
root@795fe879b7db:/opt/node-app#
```

now we will see how to do these steps with **dockerfile** (you can find it in node-app-ubuntu folder)

```
# docker pull ubuntu
FROM ubuntu
RUN apt-get update
RUN apt-get install curl -y
RUN curl -SLO https://deb.nodesource.com/nsolid_setup_deb.sh
RUN chmod 500 nsolid_setup_deb.sh
RUN ./nsolid_setup_deb.sh 21
RUN apt-get install nodejs -y

# // copy . ( . means everything (index.js will be copied)) from this folder to /opt/node-app/
# COPY . /opt/node-app/

# // command = CMD (this step can work but we can do better)
# CMD ["node", "/opt/node-app/index.js"]
```

```
# // or change workdir
# // workdir command changes working directory
WORKDIR /opt/node-app/
# // copy . (everything in local path which is have index.js) from this local path to ubuntu
current workdir (we have changed it above /opt/node-app/)
COPY . .
ENV whoami=zoothii
CMD ["node", "index.js"]
```

our dockerfile for the same scenerio we did inside our ubuntu is ready, now we can make this dockerfile an image to do that

```
docker build . -t node-app-ubuntu -f /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu/Dockerfile
```

or change directory to where dockerfile is in.

```
cd /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu
docker build . -t node-app-ubuntu
```

```
PS /Users/zoothii/Workspace/devops/docker> docker build . -t node-app-ubuntu -f /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu/Dockerfile
 [+] Building 21.5s (14/14) FINISHED                                                                                                                  docker:desktop-linux
  => [internal] load build definition from Dockerfile                                                                                                                 0.0s
  => => transferring dockerfile: 863B                                                                                                                                 0.0s
  => [internal] load .dockerignore                                                                                                                                    0.0s
  => => transferring context: 2B                                                                                                                                      0.0s
  => [internal] load metadata for docker.io/library/ubuntu:latest                                                                                                     0.0s
  => [1/9] FROM docker.io/library/ubuntu                                                                                                                              0.0s
  => [internal] load build context                                                                                                                                    0.1s
  => => transferring context: 16.61MB                                                                                                                                 0.1s
  => [2/9] RUN apt-get update                                                                                                                                         4.6s
  => [3/9] RUN apt-get install curl -y                                                                                                                                3.1s
  => [4/9] RUN curl -SLO https://deb.nodesource.com/nsolid_setup_deb.sh                                                                                               0.6s
  => [5/9] RUN chmod 500 nsolid_setup_deb.sh                                                                                                                          0.2s
  => [6/9] RUN ./nsolid_setup_deb.sh 21                                                                                                                               4.5s
  => [7/9] RUN apt-get install nodejs -y                                                                                                                              8.0s
  => [8/9] WORKDIR /opt/node-app/                                                                                                                                     0.0s
  => [9/9] COPY . .                                                                                                                                                   0.0s
  => exporting to image                                                                                                                                               0.5s
  => => exporting layers                                                                                                                                              0.5s
  => => writing image sha256:29e5759b0874949fe35ac5ad4ef110c68fe9092d9060223f50e2ddb0198c5c8a                                                                         0.0s
  => => naming to docker.io/library/node-app-ubuntu                                                                                                                   0.0s

 View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/zcca4yyw42eznrrg27fah12v4

 What's Next?
   View a summary of image vulnerabilities and recommendations → docker scout quickview
PS /Users/zoothii/Workspace/devops/docker>
```

docker build image with dockerfile layer by layer (Layered Structure) and save it to cache and for the next run it uses these layers from cache if there is no change and that makes **it build so fast this is so important feature of docker**.

```
PS /Users/zoothii/Workspace/devops/docker> docker build . -t node-app-ubuntu -f /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu/Dockerfile
[+] Building 0.2s (14/14) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                  0.0s
 => => transferring dockerfile: 863B                                                                                                  0.0s
 => [internal] load .dockerignore                                                                                                     0.0s
 => => transferring context: 2B                                                                                                       0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                                                      0.0s
 => [1/9] FROM docker.io/library/ubuntu                                                                                               0.0s
 => [internal] load build context                                                                                                     0.0s
 => => transferring context: 5.78MB                                                                                                   0.0s
 => CACHED [2/9] RUN apt-get update                                                                                                   0.0s
 => CACHED [3/9] RUN apt-get install curl -y                                                                                          0.0s
 => CACHED [4/9] RUN curl -SLO https://deb.nodesource.com/nsolid_setup_deb.sh                                                         0.0s
 => CACHED [5/9] RUN chmod 500 nsolid_setup_deb.sh                                                                                    0.0s
 => CACHED [6/9] RUN ./nsolid_setup_deb.sh 21                                                                                         0.0s
 => CACHED [7/9] RUN apt-get install nodejs -y                                                                                        0.0s
 => CACHED [8/9] WORKDIR /opt/node-app/                                                                                               0.0s
 => [9/9] COPY . .                                                                                                                    0.1s
 => exporting to image                                                                                                                0.0s
 => => exporting layers                                                                                                               0.0s
 => => writing image sha256:5a52fc64269c06ed1963b5831dfcaaf45feb4e946da529c45bfb573d05c732b7                                          0.0s
 => => naming to docker.io/library/node-app-ubuntu                                                                                    0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/j2dhyqfk6ywtncx9avm8zs2y3

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS /Users/zoothii/Workspace/devops/docker>
```

now we can create a container using that images and see the result of it

```
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu> docker images
REPOSITORY                          TAG                          IMAGE ID       CREATED          SIZE
node-app-ubuntu                     latest                       21c9f7e8d269   15 seconds ago   319MB
mongo                               latest                       b1c7466a395d   8 days ago       721MB
phpmyadmin                          latest                       2b97d838c20f   8 days ago       572MB
mysql                               latest                       8e409b83ace6   9 days ago       641MB
tagged-ubuntu                       v1                           da935f064913   2 weeks ago      69.3MB
ubuntu                              latest                       da935f064913   2 weeks ago      69.3MB
redis                               latest                       748b81d2d695   3 weeks ago      158MB
k8s.gcr.io/kube-apiserver           v1.25.2                      0b1fb9b45fa3   15 months ago    123MB
k8s.gcr.io/kube-controller-manager  v1.25.2                      c6c296d44024   15 months ago    113MB
k8s.gcr.io/kube-proxy               v1.25.2                      68348500321c   15 months ago    58MB
k8s.gcr.io/kube-scheduler           v1.25.2                      873dc124ec69   15 months ago    49.3MB
registry.k8s.io/pause               3.8                          4e42fb3c9d90   18 months ago    514kB
k8s.gcr.io/etcd                     3.5.4-0                      8e041a3b0ba8   19 months ago    179MB
registry.k8s.io/coredns/coredns     v1.9.3                       b19406328e70   19 months ago    47.7MB
docker/desktop-vpnkit-controller    v2.0                         2edf9c994f19   2 years ago      19.2MB
docker/desktop-storage-provisioner  v2.0                         c027a58fa0bb   2 years ago      39.8MB
gkandemir/todo-app                  latest                       ec5c47421d25   2 years ago      181MB
amigoscode/kubernetes               springboot-react-fullstack-v1   778d5ed2c7f9   54 years ago     509MB
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu> docker run node-app-ubuntu
node-app from ubuntu...
ENV -> zoothii
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu>
```

`EXPOSE 8080` -> defines which port the container will work on (corresponds IN_PORT (docker engine port) for example redis works 6379 on docker host.).

`CMD` -> when container run this command will run too we can override this for example (`docker run ubuntu sleep 5` (sleep 5 will override cmd command of default ubuntu dockerfile)) also it can take parameter when we create dockerfile for example (`CMD ["sleep", "2"]`)

```
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu> docker run ubuntu sleep 5
PS /Users/zoothii/Workspace/devops/docker/examples/node-app-ubuntu> docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS                      PORTS                  NAMES
56623c64f6ff   ubuntu             "sleep 5"                12 seconds ago   Exited (0) 3 seconds ago                           stupefied_mah
avira
```

`ENTRYPOINT` -> when container run this command will run too but this can not be overridden and can't take paramaters

```
FROM ubuntu
ENTRYPOINT ["sleep"]
CMD ["2"]
```

`docker build . -t ubuntu-sleep-2`
`docker run ubuntu-sleep-2` -> it will sleep 2 sec for default
`docker run ubuntu-sleep-2 5` -> it will sleep 5 sec by overriding default 2 sec

```
FROM ubuntu
ENTRYPOINT ["sleep"]
```

`docker build . -t ubuntu-sleep`
`docker run ubuntu-sleep` -> it will ask for sleep: missing operand because we cant define parameter with entrypoint
`docker run ubuntu-sleep 5` -> it will sleep 5 sec

NOTE: By default `docker run ubuntu` -> executes `CMD ["/bin/bash"]` so it executes bash then exit.

DOCKER COMPOSE https://docs.docker.com/get-started/08_using_compose/#use-docker-compose

using `compose.yml` file we can define our multi-containers (services) configurations such as service name, container name, commands, volumes, networks, working_dir, builds, ports, environment variables.. and run them easily

can have multiple isolated environments

can preserve volume data by copying volumes from the old containers to new containers

uses caches (layerad structure) if there is no change, it uses existing containers

you can see an example here too https://docs.docker.com/compose/compose-file/compose-file-v3/#compose-file-structure-and-examples

```yaml
version: '3.9'

services:
  api:
    image: '[redacted]api:latest'
    container_name: [redacted]api
    build:
      context: ./api[redacted]
      dockerfile: Dockerfile
    restart: unless-stopped
    ports:
      - "4024:4024"
    depends_on:
      - postgresqldb
  postgresqldb:
    image: postgres:14.2-alpine
    container_name: posgresqldb
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=[redacted]
      - POSTGRES_PASSWORD=[redacted]
      - POSTGRES_DB=[redacted]
  ui:
    image: '[redacted]ui:latest'
    container_name: [redacted]ui
    build:
      context: ./ui[redacted]
      dockerfile: Dockerfile
    restart: unless-stopped
    ports:
      - "3000:3000"
```

for this example we can create our containers all at once using `docker compose up` command and close them with `docker compose down`, database will be created befor api (backend) because of depends_on parameter for our backend to connect database, database should be created first

the important thing is here api uses `postgresqldb` as host name (`postgresqldb:5432`)
we dont't need `version` parameter anymore

```yaml
services:
  api:
    image: 'api:latest' ## if there is an image on dockerhub it will use that
    container_name: api
    build:
      context: ./api ## tells us where is the dockerfile and app
      dockerfile: Dockerfile
    restart: unless-stopped ## if there will be anyproblem it will be restarted
    ports:
      - "4024:4024"
    depends_on:
      - postgresqldb ## api depends on database so database will be created first
  postgresqldb:
    image: postgres:14.2-alpine
    container_name: postgresqldb
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=x
      - POSTGRES_PASSWORD=x
      - POSTGRES_DB=x
    volumes:
```

```
      - mypostgresql-data:/var/lib/postgresql/data

volumes: ## for volume mapping to work its should be defined like that
  mypostgresql-data:
```

`docker compose up` –> creates images or pulls, create networks, create volumes and starts containers
`docker compose down` –> deletes containers and networks
`docker compose stop` –> stops containers
`docker compose start` –> start containers
`docker compose -f {path to compose.yml} up`

AHMET YILDIRIM

REFERENCES

https://www.youtube.com/watch?v=4XVfmGE1F_w&list=PL_f2F0Oyaj4_xkCDqnRWp4p5ypjDeC0kO
https://docs.docker.com/reference/
https://www.simpliaxis.com/resources/how-does-docker-and-kubernetes-work-together
https://www.udemy.com/course/docker-kubernetes-the-practical-guide/