

Основи во Docker

Оперативни системи

Аудиториска вежба 7

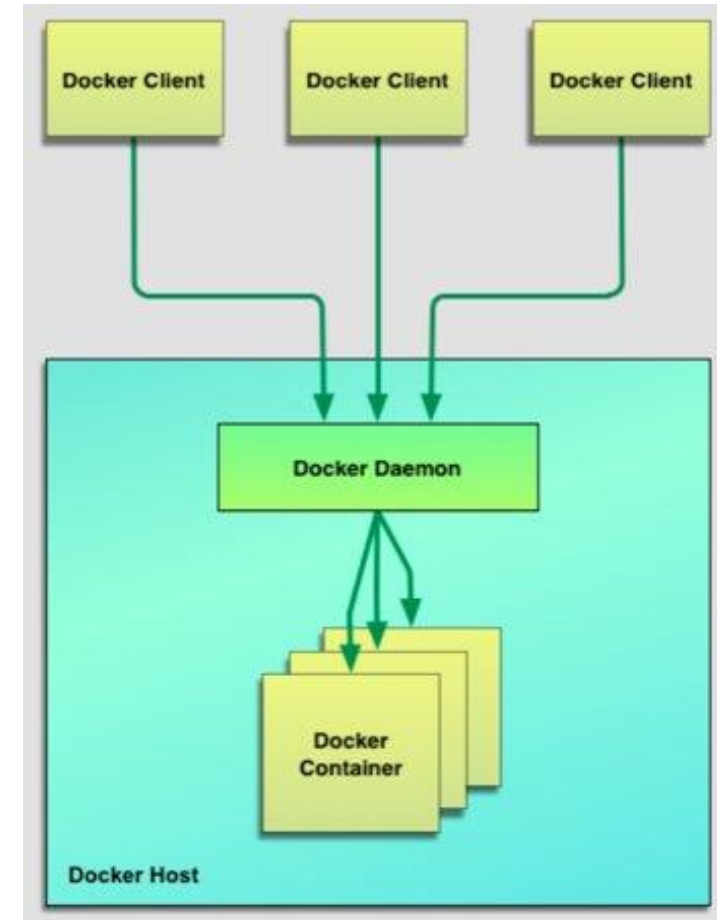


Виртуелна Машина vs Контејнер

VMs	Containers
Heavyweight	Lightweight
Limited Performances	Native Performances
Each VM runs its own OS	All containers share the host OS
Hardware-level virtualization	OS Virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Архитектура на Docker (1)

- Docker компоненти:
 - Docker Engine (Docker client and server);
 - Docker Images;
 - Registries;
 - Docker Containers;
- Docker Engine
 - Docker **клиентот** зборува со Docker **серверот** (client-server app);
 - Docker **серверот** а.к.а. Docker Engine, или Docker daemon;
 - CLI binary: **docker**;



Архитектура на Docker (2)

- Docker Images

Градбени блокови во Docker светот;

Слоевит формат, градени чекор-по-чекор:

- Додади датотека;
- Изврши команда;
- Отвори порта;

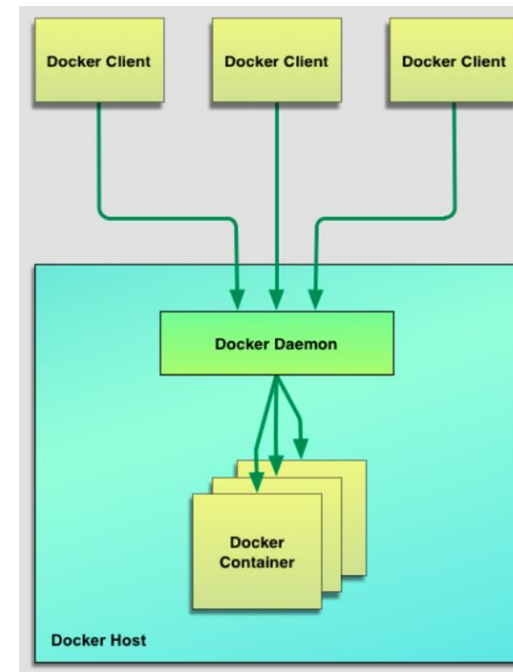
Претставуваат „изворен код“ за Docker контејнерите;

- Регистри

Репозиториуми за чување на Docker Images;

Јавни и приватни;

Docker Hub: <https://hub.docker.com>;



Архитектура на Docker (3)

- Docker контејнери

контејнер= пакет за една апликација / сервис;

Контејнер се стартува од Docker Image;

- Image = градбен блок;
- Container = извршувачки блок;

Може да се креира, стартува, стопира, рестартира и уништува;

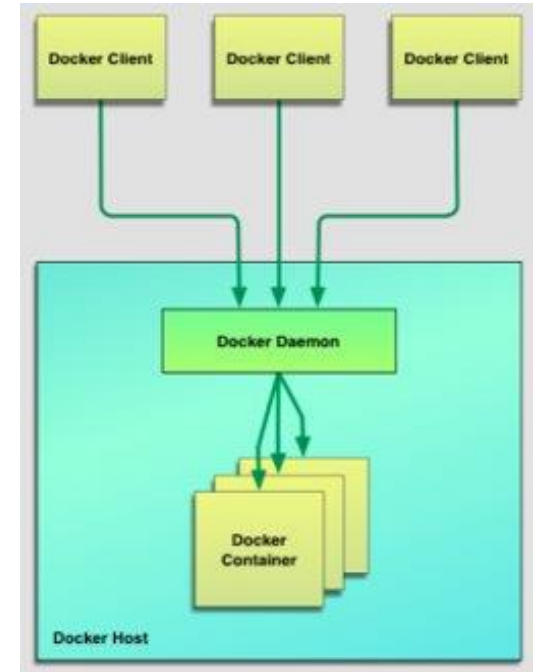
Docker не се грижи за што се извршува во контејнерот

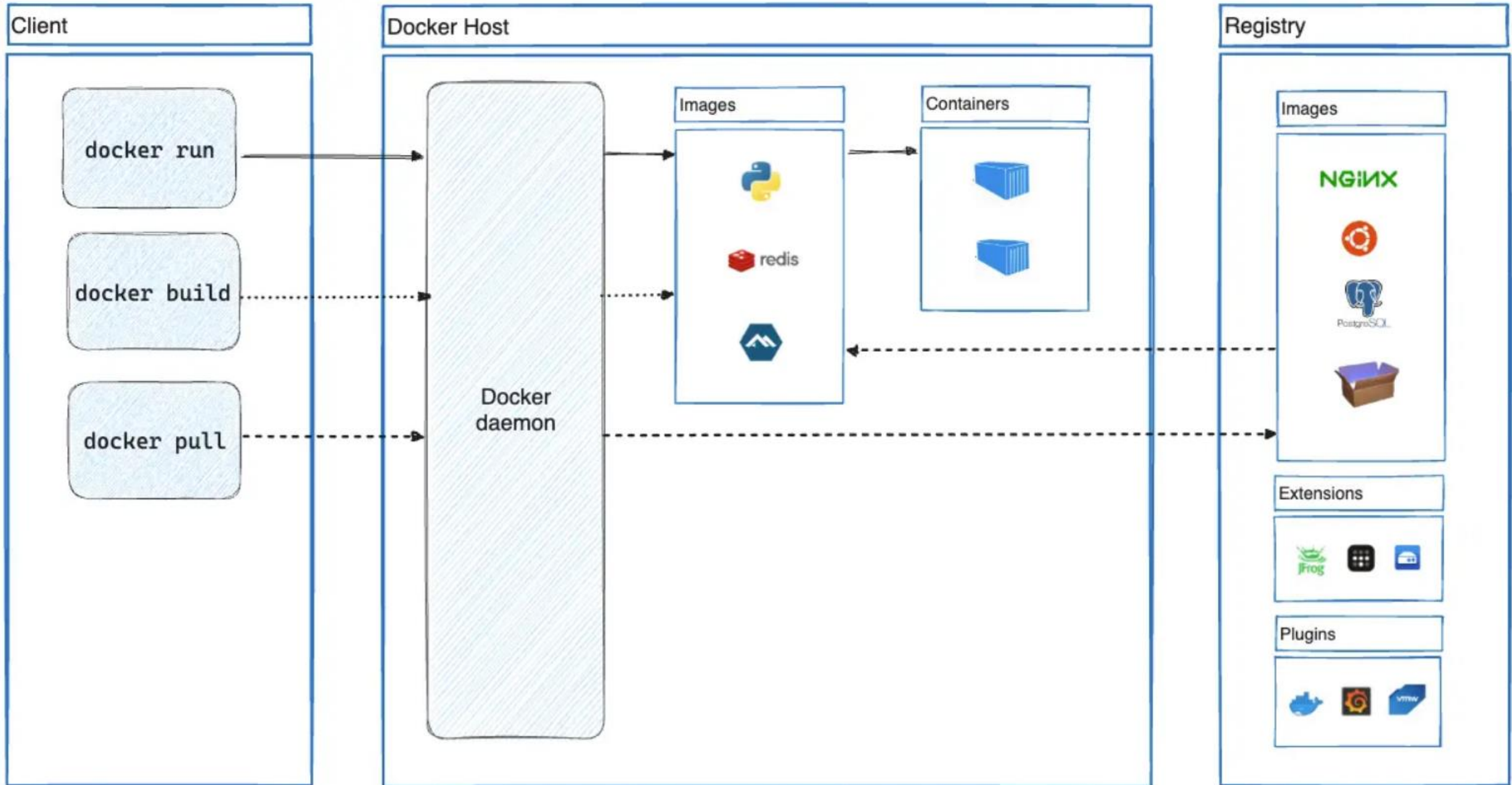
Docker не се грижи каде ќе биде испорачан контејнерот;

- Docker мрежа

Docker мрежите се виртуелни мрежи кои се користат за да се поврзат повеќе контејнери.

Овозможуваат контејнерите да соработуваат помеѓу себе и со ОС-домаќин.





Docker Архитектура (4)

- Dockerfile
 - Dockerfile претставува скрипта со инструкции за како да се создаде Docker Image.
 - Содржи информации за оперативниот систем, јазиците, системските променливи, локациите на датотеките, мрежни порти и други детали кои се потребни за да се стартува сликата.
 - Командите во датотеката се поставуваат во групи, и овие групи се извршуваат автоматски.
- Docker Compose
 - Docker Compose е алатка која се користи за дефинирање и стартување на Docker апликации кои работат во повеќе контејнери.
 - Овозможува корисничко дефинирање на сервисите кои ги користи апликацијата во една единствена датотека.
 - Docker Compose овозможува едноставна конструкција, стартување и менаџирање на апликациите кои се дел од поголемо софтверско решение, без оглед на нивниот број.
 - Корисникот може да ги стартува одеднаш, да ги стопира, рестартира, гледа нивните логови, менаџира нивната приватна мрежа итн.
- Docker Swarm
 - Docker Swarm е сервис кој овозможува оркестрација помеѓу контејнерите, нивна распределба и извршување низ кластер на сервери.
 - Kubernetes - позната алтернатива на Docker Swarm

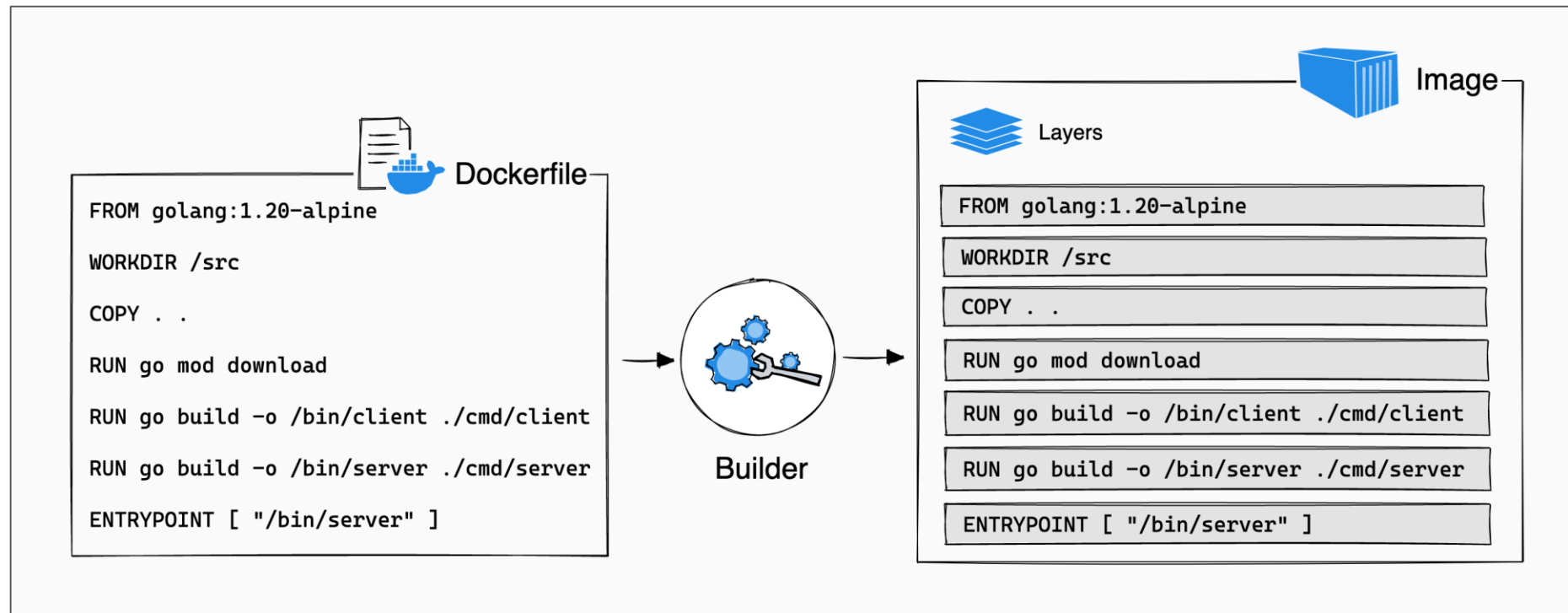
Docker Image

- Docker Image претставува read-only шаблон кој содржи множество на инструкции за креирање на контејнер кој може да се стартува на Docker платформата.
- Обезбедува пригоден начин за пакување апликации и претходно конфигурирани серверски околина, кои можете да ги користите за ваша приватна употреба или јавно да ги споделувате со други корисници на Docker.
- Составен од колекција на датотеки, како што се инсталации, код на апликација и зависности.
- Docker Image може да се креира со еден од следните два методи:
 - Интерактивно
 - Dockerfile

Docker слоеви

- Секоја од датотеките што ја сочинуваат Docker Image е позната како слој. ,
- Овие слоеви формираат серија на меѓу-слики, изградени една врз друга во фази, каде што секој слој зависи од слојот веднаш под него.
- Хиерархијата на слоевите е клучна за ефикасно управување со животниот циклус на Docker Image.
 - Слоевите кои најчесто се менуваат треба да бидат колку што е можно повисоко во стекот. На овој начин ќе се заштеди време при rebuild.
- Слој на контејнерот
 - При стартување на контејнер од слика, Docker додава тенок слој на кој може да се менува, познат како слој на контејнер, кој ги зачувува сите промени во контејнерот во текот на неговото извршување.
- Родителска слика (Parent Image)
 - Првиот слој на Docker image е познат како „parent image“.

Docker слоеви




Кеширани слоеви


- При build на Docker Image од Dockerfile, Docker се обидува да ги реискористи слоевите од претходните извршувања.
- Ако слојот на сликата не е променет, Docker ќе ја реискористи старата кеширана слика.
- Ако слојот се променил од претходниот build, Docker одново ќе го изгради тој слој.

Layers	Cache?
FROM golang:1.20-alpine	✓
WORKDIR /src	✓
COPY . .	✗
RUN go mod download	✗
RUN go build -o /bin/client ./cmd/client	✗
RUN go build -o /bin/server ./cmd/server	✗
ENTRYPOINT ["/bin/server"]	✗

Кеширани слоеви - Пример за оптимизација

 Layers	Cache?
FROM golang:1.20-alpine	✓
WORKDIR /src	✓
COPY . .	✗
RUN go mod download	✗
RUN go build -o /bin/client ./cmd/client	✗
RUN go build -o /bin/server ./cmd/server	✗
ENTRYPOINT ["/bin/server"]	✗



 Layers	Cache?
FROM golang:1.20-alpine	✓
WORKDIR /src	✓
* COPY go.mod go.sum .	✓
RUN go mod download	⊙
COPY . .	✗
RUN go build -o /bin/client ./cmd/client	✗
RUN go build -o /bin/server ./cmd/server	✗
ENTRYPOINT ["/bin/server"]	✗

Docker команди – Менаџмент на контејнери

<code>docker ps</code>	List the running containers
<code>docker ps -a</code>	List all the containers, both running and stopped.
<code>docker create [image]</code>	Create a container without starting it.
<code>docker create -it [image]</code>	Create an interactive container with pseudo-TTY
<code>docker rename [container] [new-name]</code>	Rename a container
<code>docker rm [container]</code>	Remove a stopped container.
<code>docker rm -f [container]</code>	Force remove a container, even if it is running
<code>docker logs [container]</code>	View logs for a running container

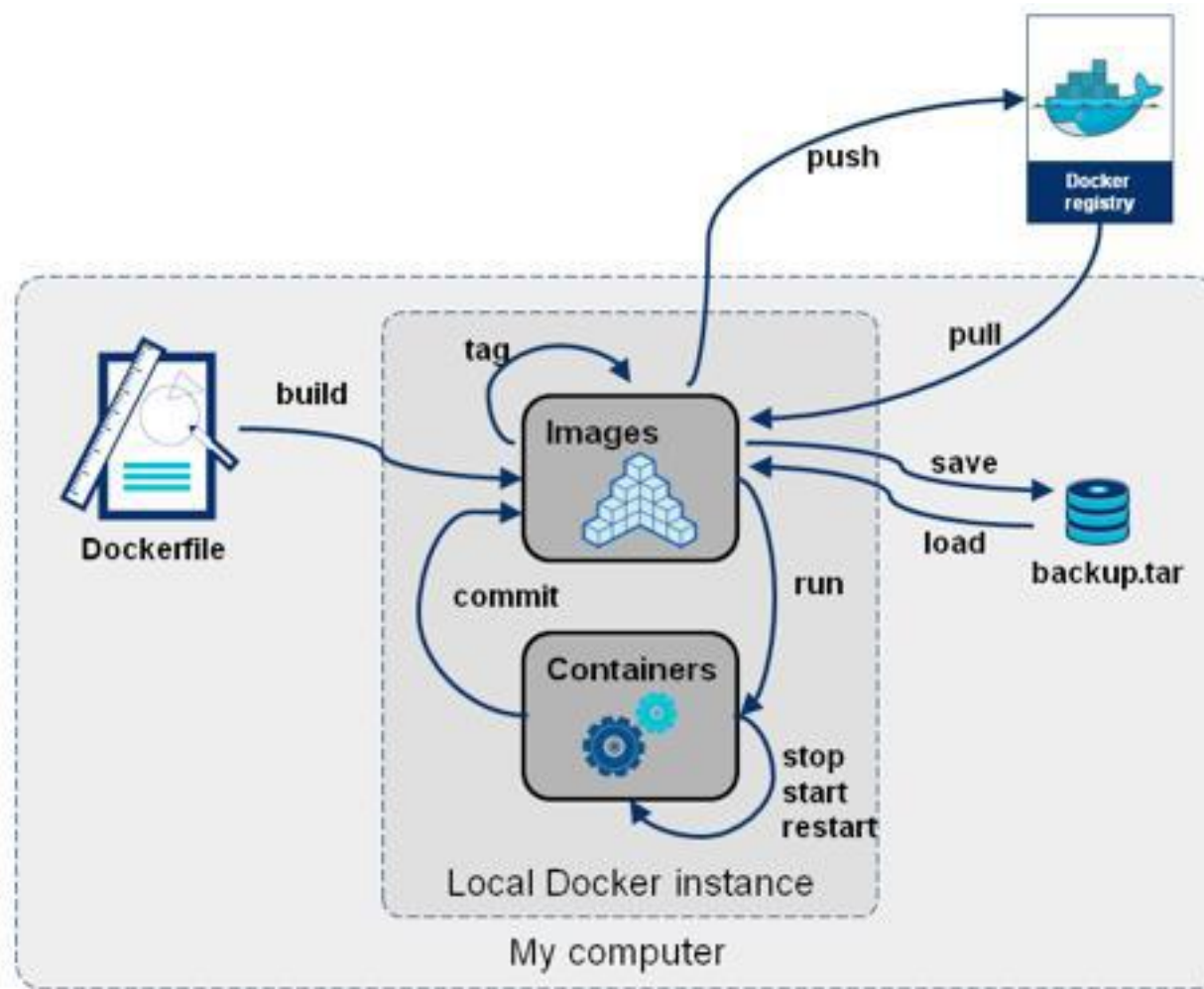
Docker команди – Стартување на контејнер

<code>docker run [image] [command]</code>	Run a command in a container based on an image.
<code>docker run -name [container-name] [image]</code>	Create, start and name a container.
<code>docker run -p [host]:[container-port] [image]</code>	Map a host port to a container port.
<code>docker run -rm [image]</code>	Run a container and remove it after it stops.
<code>docker run -d [image]</code>	Run a detached (background) container.
<code>docker run -it [image]</code>	Run an interactive process, e.g. a shell in a container.
<code>docker start [container]</code>	Start a container.
<code>docker stop [container]</code>	Stop a container
<code>Docker restart [container]</code>	Restart a container.
<code>docker exec -it [container] [shell]</code>	Run a shell inside a running container.

Docker команди– Менаџмент на Docker Images

<code>docker build [dockerfile-path]</code>	Create an image from a Dockerfile
<code>docker build .</code>	Build an image using the files from the current path.
<code>docker build -t [name]:[tag] [location]</code>	Create an image from a Dockerfile and tag it.
<code>docker build -f [file]</code>	Specify a file to build from.
<code>docker pull [image]</code>	Pull an image from registry.
<code>docker push [image]</code>	Push an image to a registry.
<code>docker commit [container] [new-image]</code>	Create an image from a container.
<code>docker images</code>	Show all locally stored top level images.
<code>docker rmi [image]</code>	Remove an image.
<code>docker images prune</code>	Remove unused images.

Docker команди– дијаграм на состојби



Docker мрежи

- Вмрежувањето на контејнерите (Container networking) се однесува на способноста контејнерите да се поврзуваат и да комуницираат едни со други, како и со оптоварувања што не се Docker контејнери.
- Контејнерот нема информации за тоа на каква мрежа е прикачен, или дали останатите во мрежата се Docker контејнери или не.
- Контејнерот гледа само мрежен интерфејс со IP адреса, порта, рутирачка табела, DNS услуги и други мрежни детали.

```
#list networks
```

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
c35ebcb20e5e	bridge	bridge	local
77bf26c77932	host	host	local
d5eb7bcca8af	none	null	local

```
#inspect network
```

```
docker network inspect bridge
```

```
# list ports for container
```

```
docker port nginx
```

Креирање контејнер мрежи

Кориснички дефинирани мрежи

- Откако ќе се поврзат на мрежа дефинирана од корисникот, контејнерите можат да комуницираат едни со други користејќи IP адреси на контејнери или имиња на контејнери.

```
docker network create -d bridge my-net
```

```
docker network ls
```

```
docker run --network=my-net -itd --name=container3 busybox
```

Можете директно да прикачите контејнер на мрежниот stack на друг контејнер, користејќи го:

```
--network container:<name|id>
```

```
docker run -d --name redis example/redis --bind 127.0.0.1
```

```
docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1
```

Бришење на мрежата

```
docker network rm my-net
```

Објавување на порти

- Кога креирате или стартувате контејнер користејќи `docker create` или `docker run`, контејнерот не изложува (`expose`) ниту една од неговите порти на надворешниот свет. Користете го знамето `--publish` или `-p` за да направите порта достапна за услуги надвор од Docker.
- Ова создава firewall rule на домаќинот, мапирајќи ја контејнерската порта до портата на домаќинот со надворешниот свет.
- Објавувањето порти од контејнерот е небезбедно. Што значи, кога ги објавувате портите на контејнерот, тој станува достапен не само за домаќинот на Docker, туку и за надворешниот свет.
- За да се направи контејнер достапен за други контејнери, не е неопходно да се објавуваат портите на контејнерот. Можете да овозможите интерконтејнерска комуникација со поврзување на контејнерите на истата мрежа.

Flag value	Description
<code>-p 8080:80</code>	Map port 8080 on the Docker host to TCP port 80 in the container .
<code>-p 192.168.1.100:8080:80</code>	Map port 8080 on the Docker host IP 192.168.1.100 to TCP port 80 in the container .
<code>-p 8080:80/udp</code>	Map port 8080 on the Docker host to UDP port 80 in the container .
<code>-p 8080:80/tcp</code> <code>-p 8080:80/udp</code>	Map TCP port 8080 on the Docker host to TCP port 80 in the container , and map UDP port 8080 on the Docker host to UDP port 80 in the container .

Објавување на порти: примери

```
docker pull hello-world
```

```
docker run -d -p 8080:80 hello-world
```

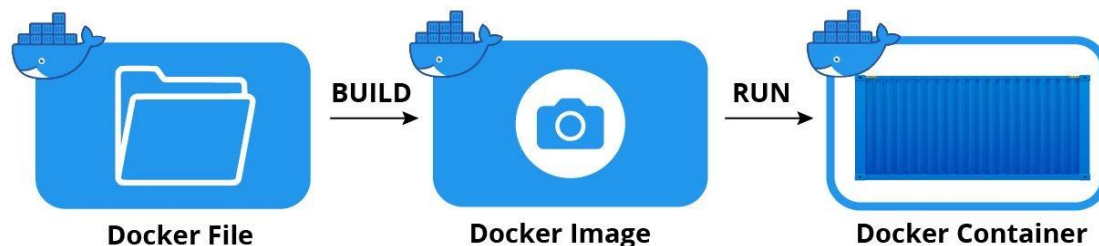
```
docker run -d -p 192.168.1.100:8080:80 hello-world
```

```
docker run -d -p 8080:80/udp hello-world
```

```
docker run -d -p 8080:80/tcp -p 8080:80/udp hello-world
```

Методи за градење на Docker Image

- Интерактивно
 - **Предности:** Најбрзиот и наједноставниот начин за креирање на Docker слики. Идеален за тестирање, решавање проблеми, одредување зависности и валидација на процеси.
 - **Недостатоци:** Тешко управување со животниот циклус, побарува постојана рачна реконфигурација.
- Dockerfile
 - Овозможува градење на чисти, компактни и повторливи слики базирани на точно дефинирани рецепти.
 - Полесно управување со животниот циклус и полесна интеграција во процесите на континуирана интеграција (CI) и континуирана испорака (CD).



Интерактивен метод - Пример

- Стартувај интерактивна shell сесија на контејнер. Контејнерот се заснова на Docker image со Debian OS преземен од Docker Hub:

```
$ docker run -it debian:11-slim bash
```

- Инсталирај nginx сервер:

```
$ apt-get update && apt-get install -y nginx
```

- Излези од интерактивен режим:

```
Ctrl + D
```

- Излистај ги сите docker контејнери (вклучувајќи ги и стопираните):

```
$ docker ps -a
```

output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
61d961dfdf7b	debian:11-slim	"bash"	About a minute ago	Exited (0) 6 seconds ago		

- Зачувај ја состојбата на контејнерот во слика со docker commit наредбата:

```
$ docker commit 61d961dfdf7b debian_nginx
```

- Излистај ги docker images за проверка дали операцијата е успешна:

```
$ docker images
```

output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian_nginx	latest	538a64303a41	8 seconds ago	164MB

Dockerfile команди

Команда	Опис
FROM	To specify the parent (base) image.
WORKDIR	To set the working directory for any commands that follow in the Dockerfile.
RUN	To install any applications and packages required for your container.
COPY	To copy files or directories from a specific location on the host to a location in the container
ADD	As COPY, but also able to handle remote URLs and unpack compressed files.
ENTRYPOINT	Command that will always be executed when the container starts. If not specified, the default is /bin/sh -c
CMD	Arguments passed to the entrypoint. If ENTRYPOINT is not set (defaults to /bin/sh -c), the CMD will be the commands the container executes.
EXPOSE	To define which port through which to access your container application.
LABEL	To add metadata to the image.

Dockerfile - Пример

Содржина на Dockerfile датотеката

```
# Use the official debian:11-slim as base
FROM debian:11-slim

# Install nginx and curl
RUN apt-get update && apt-get upgrade -y && apt-get install -y nginx curl && rm -rf /var/lib/apt/lists/*

# Set the entrypoint
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Користи ја `docker build` командата за креирање на слика од Dockerfile. Со `-t` знаменцето се сетира име на сликата и таг:

```
$ docker build -t my-nginx:0.1 .
```

Изврши ја `docker images` командата за преглед на ново креираната слика:

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-nginx	0.1	f95ae2e1344b	10 seconds ago	164MB

Dockerfile - Пример (2)

- Стартување на контејнер од Docker Image
 - `docker run --name <docker_container_name> <docker_image_name>`
 - `-d` за да стартува во background (detached) режим на работа
- `docker run -d --name nginx_hello_world my-nginx`
- Доколку сакаме да споделиме порта на контејнерот со домаќинот потребно е да ја искористиме опцијата `-p`
 - `docker run -d p 8080:80 --name nginx_hello_world my-nginx`

Пример 1 – Контејнеризација на Java апликација

- Направете контејнеризација на следната Java класа во Docker Image со име hello-world-java. Од сликата, направете контејнер со име hello-world-java-container. Користете Dockerfile метод за градење на Docker Image.

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World from Docker!");  
    }  
}
```

Решение: Изглед на Dockerfile

```
FROM eclipse-temurin
```

```
WORKDIR /usr/src/myapp
```

```
COPY ..
```

```
RUN javac Main.java
```

```
CMD ["java", "Main"]
```

Решение

`docker build -t hello-world-java .`

```
[+] Building 1.1s (9/9) FINISHED          docker:default
=> [internal] load build definition from Dockerfile          0.0s
=> => transferring dockerfile: 161B          0.0s
=> [internal] load metadata for docker.io/library/eclipse-temurin:latest 1.1s
=> [internal] load .dockerignore          0.0s
=> => transferring context: 2B          0.0s
=> [internal] load build context          0.0s
=> => transferring context: 60B          0.0s
=> [1/4] FROM docker.io/library/eclipse-temurin:latest@sha256:3e7d577a0977d784 0.0s
=> CACHED [2/4] COPY . /usr/src/myapp          0.0s
=> CACHED [3/4] WORKDIR /usr/src/myapp          0.0s
=> CACHED [4/4] RUN javac Main.java          0.0s
=> exporting to image          0.0s
=> => exporting layers          0.0s
=> => writing image sha256:d2c1c110c3f5a34bbf72682d5c85a2864bb07aca5fe70a626aa 0.0s
=> => naming to docker.io/library/hello-world-java          0.0s
```

`docker images`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world-java	latest	d2c1c110c3f5	15 minutes ago	436MB

Решение (2)

- `docker run` командата стартува команда во нов контејнер врз база на `hello-world-java` Docker Image-от.
- `docker run -it --rm --name hello-world-java-container hello-world-java`
 - it – стартува во интерактивен режим
 - rm – го пребришува контејнерот по неговото завршување.
 - name – поставува име на контејнерот по кој подоцна ќе биде препознатлив за одново стартување.

Пример 2. ExecutionCounter.java

- Потребно е да ја контейнеризирате дадената Java апликација која во дадена датотека ќе чува број на извршувања на Docker контейнерот. Доколку датотеката не постои, истата се креира и се запишува иницијална вредност 1.

```
import java.io.EOFException;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;

public class ExecutionCounter {

    public static void main(String[] args) throws IOException {
        RandomAccessFile raf = null;
        Integer counter = 0;
        try {
            raf = new RandomAccessFile("./data//raf.out", "rw");
            counter = raf.readInt();
            counter++;
            System.out.println(counter);
            raf.seek(0);
            raf.writeInt(counter);
            raf.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (EOFException e) {
            counter = 1;
            System.out.println(counter);
            raf.writeInt(counter);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (raf != null) {
                    raf.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Решение: Dockerfile

```
FROM eclipse-temurin
```

```
WORKDIR /usr/src/myapp
```

```
COPY ..
```

```
RUN mkdir data
```

```
RUN javac ExecutionCounter.java
```

```
CMD ["java", "ExecutionCounter"]
```

Решение:

- `docker build -t counter-java .`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
counter-java	latest	2abdd4ee0aff	18 minutes ago	436MB

- Execution 1: `docker run -it --rm --name counter-java-counter counter-java`

Output: 1

- Execution 2: `docker run -it --rm --name counter-java-counter counter-java`

Output: 1

- Execution 3: `docker run -it --rm --name counter-java-counter counter-java`

Output: 1

Проблем!



Решение: (so volume)

- Mount a volume in a container:

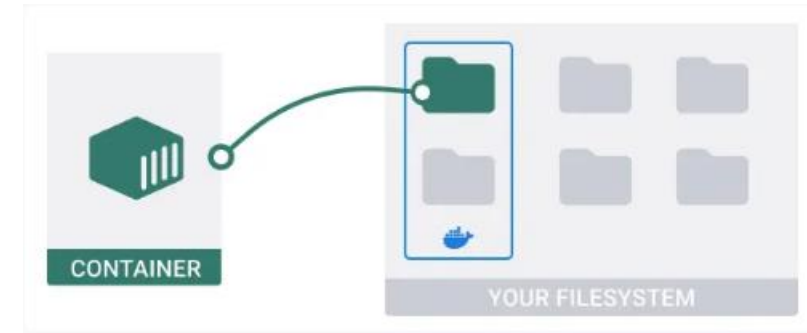
Use the `-v` parameter

```
docker run -it -v ./data:/usr/src/myapp/data/ --rm --name counter-java-counter counter-java
```

`./data/` - **патека до именик на домаќинот .**

`/usr/src/myapp/data/` - **апсолутна патека до мапираниот именик кај контејнерот.**

- Execution 1: `docker run -it -v ./data:/usr/src/myapp/data/ --rm --name counter-java-counter counter-java`
- Output: 1
- Execution 2: `docker run -it -v ./data:/usr/src/myapp/data/ --rm --name counter-java-counter counter-java`
- Output: 2
- Execution 3: `docker run -it -v ./data:/usr/src/myapp/data/ --rm --name counter-java-counter counter-java`
- Output: 3



Екстернизација на конфигурација со опкружувачки променливи (**environment variables**)

- Опкружувачките променливи се променливи чија вредност се поставува надвор од кодот на програмата, најчесто преку функционалност вградена во оперативниот систем.
 - **PATH** променливата;
- Претставуваат клуч/вредност парови:
REACT_APP_URL=http://react.example.com
- Опкружувачките променливи овозможуваат чување на конфигурациски параметри за локалните и продукциските околин при развојот на одредена апликација.

**Пример 3. Направете
измени на Пример 2, со
тоа што датотеката во
која ќе се чува бројачот,
ќе се вчитува од
опкружувачка
променлива.**

```
docker build -t counter-java-env .
```

```
docker run -it -v ./data:/usr/src/myapp/data/  
-e  
PERSISTING_FILE_PATH=/usr/src/myapp/da  
ta/data.out --rm --name counter-java-  
counter-env counter-java-env
```

```
import java.io.*;  
  
public class ExecutionCounter2 {  
  
    public static void main(String[] args) throws IOException {  
        RandomAccessFile raf = null;  
        Integer counter = 0;  
        try {  
            String envPath = System.getenv("PERSISTING_FILE_PATH");  
            if (envPath == null) {  
                throw new RuntimeException("Environment variable does not exists!");  
            }  
            File path = new File(envPath);  
            raf = new RandomAccessFile(path, "rw");  
            counter = raf.readInt();  
            counter++;  
            System.out.println(counter);  
            raf.seek(0);  
            raf.writeInt(counter);  
            raf.close();  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        } catch (EOFException e) {  
            counter = 1;  
            System.out.println(counter);  
            raf.writeInt(counter);  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally {  
            try {  
                if (raf != null) {  
                    raf.close();  
                }  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Пример 4: Споделување на **Docker image**

- Откако ги решивте претходните задачи, потребно е да ги споделите со колегите за и тие да можат да ги стартуваат како контејнер на нивната машина.
- Потребно е да им пратите Docker image од задачите.

Решение: Docker save (1)

Вне:

```
docker save counter-java > counter-java.tar
```

```
ls -sh counter-java.tar
```

```
436MB counter-java.tar
```

```
docker save --output counter-java.tar counter-java
```

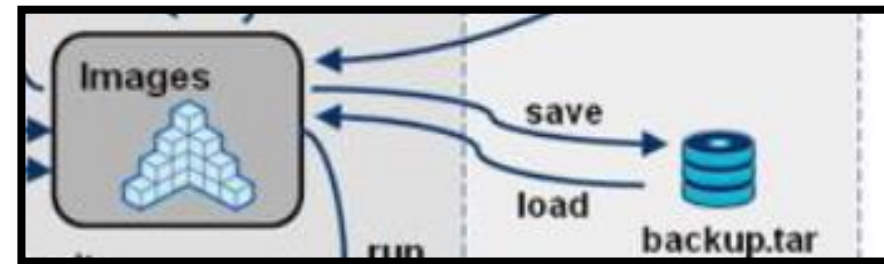
```
ls -sh counter-java.tar
```

```
436MB counter-java.tar
```

```
docker save -o counter-java.tar counter-java
```

```
docker save -o counter-java.tar counter-java :latest
```

Usage	docker image save [OPTIONS] IMAGE [IMAGE...]	
Aliases	docker save	
Option	Default	Description
-o, --output		Write to a file, instead of STDOUT



Решение: Docker load (2)

Колегите:

```
docker load < counter-java.tar.gz
```

Loaded image: counter-java:latest

docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
counter-java	latest	2abdd4ee0aff	7 weeks ago	436MB

```
docker load --input counter-java.tar
```

Loaded image: counter-java:latest

Description	Load an image from a tar archive or STDIN	
Usage	docker image load [OPTIONS]	
Aliases	docker load	
	Option	Description
	-i, --input	Read from tar archive file, instead of STDIN

Решение: Docker push и Docker pull (3)

Објаснување:

Docker Hub е местото каде што се складираат отворените Docker слики.

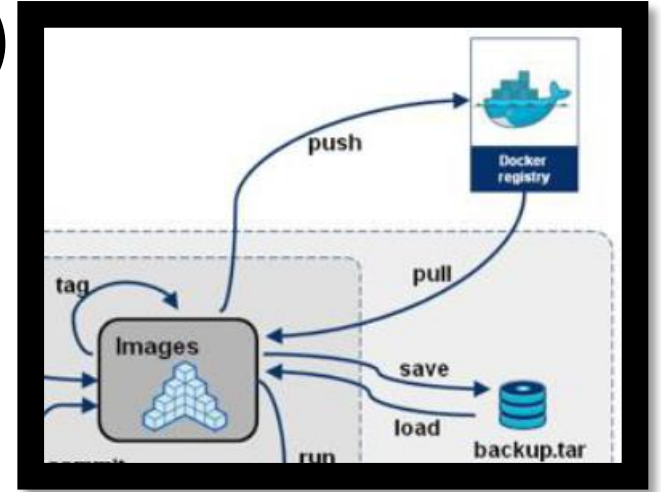
```
sudo docker run -it --rm ubuntu /bin/bash
```

софтверот прво провери дали оваа слика е достапна на вашиот компјутер и доколку не е, ќе ја презеде сликата од Docker Hub.

Така, добивањето слика од Docker Hub функционира автоматски.

Ако сакате само да ја повлечете сликата, но не да ја извршите, можете исто така да го направите тоа:

```
docker pull ubuntu
```



Решение: Docker Hub (4)

- Најавете се на <https://hub.docker.com/>
- Кликнете *Create Repository*.
- Одберете име (пример: counter-java) и опис за вашиот репозиториум и кликнете *Create*.
- Најавете се на Docker Hub од командна линија

```
docker login --username=yourhubusername --email=youremail@company.com
```


Решение: Push/Pull (5)

Вие:

```
docker tag image_id yourhubusername/counter-java:latest
```

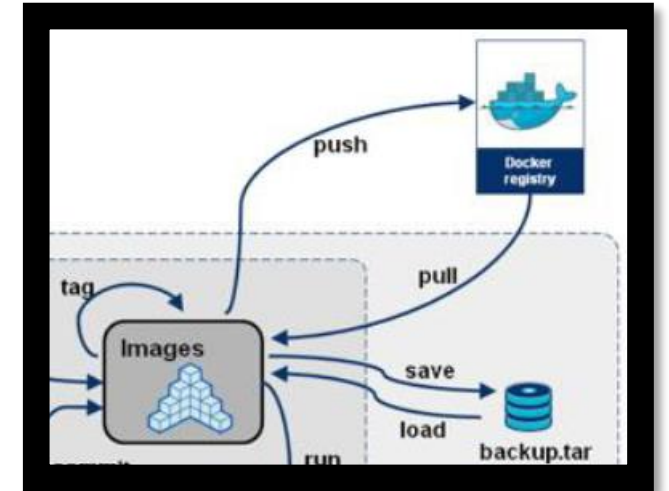
```
docker push yourhubusername/counter-java:latest
```

← tag

Вашата docker слика сега е достапна за секого.

Колегите:

```
docker pull yourhubusername/counter-java:latest
```



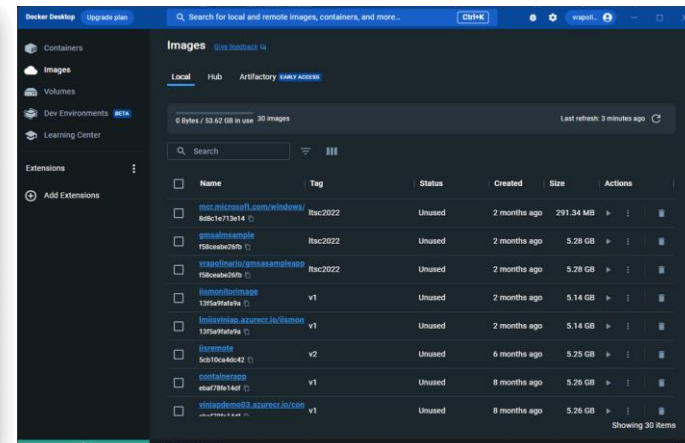
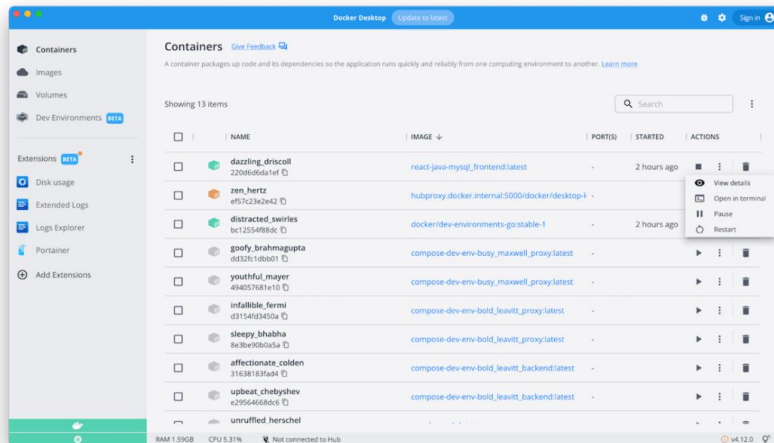
Решение: **Save/Load vs Push/Pull (6)**

Docker Hub недостатоци:

- Bandwidth – многу ISPs имаат многу помал upload bandwidth одколку download bandwidth.
- Се додека не' плаќате дополнително за приватни репозиториуми, push е еднакво на јавно објавување на docker сликите.
- Кога се работи со кластери и многу слики, секогаш кога ќе се стартува job кој користи Docker container прави pull the на сликата од Docker Hub, и ако пуштиме многу такви jobs, може да е многу споро.

Опции за вежбање Docker

1. os.finki.ukim.mk (CAS login)
2. Локална инсталација
 1. [Linux](#)
 2. [Windows \(Docker Desktop\)](#)
 3. [MacOS](#)
3. PWD (Play with docker – Docker Lab)
4. <https://killercoda.com/playgrounds/scenario/ubuntu>



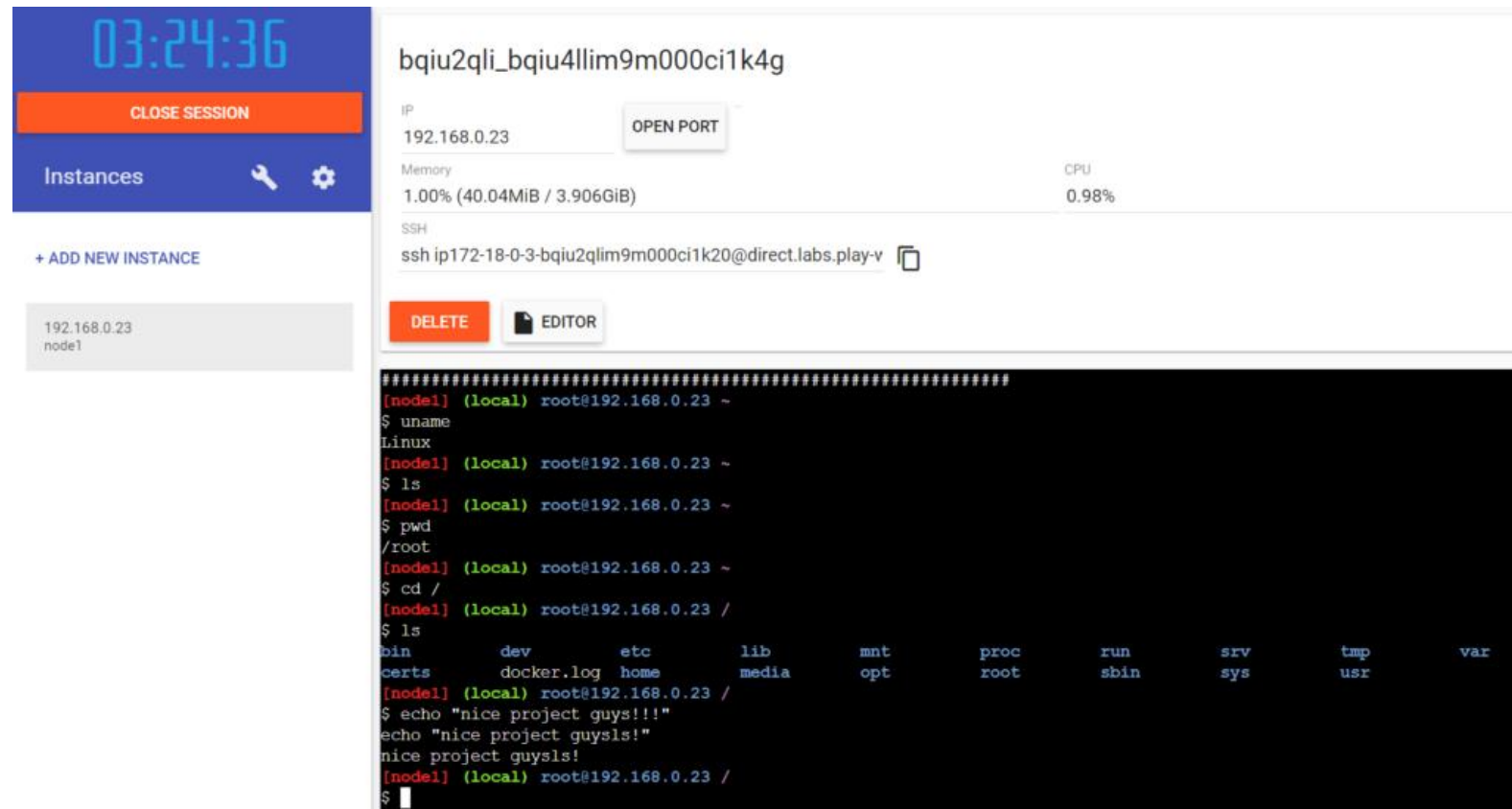
PWD : Linux инстанци со инсталиран Docker

[Play with Docker \(play-with-docker.com\)](https://play-with-docker.com)

PWD е околина за Docker кое им овозможува на корисниците да ги извршуваат командите на Docker за неколку секунди.

Тоа дава искуство дека имате бесплатна Linux виртуелна машина во прелистувачот, каде што можете да изградите и стартувате Docker контејнери, па дури и да креирате кластери со Docker Swarm.

PWD е официјален проект спонзориран од Docker Inc.



The screenshot displays the Play with Docker web interface. On the left, a sidebar shows a digital clock at 03:24:36, a 'CLOSE SESSION' button, and an 'Instances' section with a '+ ADD NEW INSTANCE' button and a list of instances including '192.168.0.23 node1'. The main panel shows details for instance 'bqiu2qli_bqiu4llim9m000ci1k4g' with IP '192.168.0.23', memory usage of 1.00% (40.04MiB / 3.906GiB), and CPU usage of 0.98%. It includes an 'OPEN PORT' button, an SSH command 'ssh ip172-18-0-3-bqiu2qlim9m000ci1k20@direct.labs.play-v', and 'DELETE' and 'EDITOR' buttons. The terminal window shows the following commands and output:

```
#####
[node1] (local) root@192.168.0.23 ~
$ uname
Linux
[node1] (local) root@192.168.0.23 ~
$ ls
[node1] (local) root@192.168.0.23 ~
$ pwd
/root
[node1] (local) root@192.168.0.23 ~
$ cd /
[node1] (local) root@192.168.0.23 /
$ ls
bin          dev          etc          lib          mnt          proc        run          srv          tmp          var
certs        docker.log  home         media        opt          root        sbin        sys         usr
[node1] (local) root@192.168.0.23 /
$ echo "nice project guys!!!"
echo "nice project guys!!!"
nice project guys!!!"
[node1] (local) root@192.168.0.23 /
$
```

killercoda



- <https://killercoda.com/playgrounds/scenario/ubuntu>
- Овозможува брз пристап до Linux или Kubernetes околина спремна за користење преку веб прелистувач.

ПРАШАЊА