

# Основи во Docker Compose

Оперативни системи

Аудиториска вежба 8



# Вовед во Docker Compose

- Алатка која овозможува повисока апстракција при дефинирање и стартување на повеќе-контејнерски Docker апликации.
- YAML датотека - конфигурацијата на апликациските сервиси.
- Compose може да се користи во сите околии: продукција, развој, тестирање, како и во работни текови на континуирана испорака.
- Со една команда, може да се креираат и стартуваат сите конфигурирани сервиси одеднаш.

# Docker Compose: Multi Container Applications

## Без Docker Compose

- Градење (build) и стартување еден по еден контејнер;
- Поврзувањето меѓу контејнерите е рачно;
- Претпазливост при менаџирање на зависности и редослед на стартување на контејнерите;
- Пишување на долги команди со многу опции;
- ...

## Со Docker Compose

- Поедноставена контрола на повеќе-контејнерски апликации;
- Ефикасна колаборација;
- Конфигурациски-ориентирано менаџирање на зависности и редослед на стартување;
- Брз развој на апликации;
- Зголемена портабилност;
- ...



# Docker Compose: Multi Container Applications

## Без Docker Compose

- `$ docker run -d -it --name redis redis`
- `$ docker run -d -it --name postgres  
linhmtran168/postgres`
- `$ docker run -d -it --name web \ -v  
~/Dev/gitlab.com/linhmtran168/test-  
project:/var/www/html \ --link postgres:db --link  
redis:redis linhmtran168/php-web`
- `$ docker run -d -it -p 80:80 --name nginx \ --link  
web:web --volumes-from web linhmtran168/php-  
nginx`
- `$ docker run -d -it --name node --link web:web \ --  
volumes-from web linhmtran168/gulp-bower`

## Co Docker Compose

```
web:
  build: .
  links:
    - redis:redis
    - postgres:db
  volumes:
    - ../var/www/html
nginx:
  build: ../docker-php-nginx
  ports:
    - "80:80"
  links:
    - web:web
  volumes_from:
    - web
```

# Docker Compose во три чекори

- Чекор 1: Дефинирај ја околината на секоја апликација посебно во Dockerfile.
- Чекор 2: Дефинирај ги сите сервиси во compose.yaml.
- Чекор 3: Изврши `docker compose up -d`.

# Структура на **compose.yaml** датотеката

- Version → version: '3'
  - The top-level version property is defined by the Compose Specification for backward compatibility. It is only informative.
- Services → services:  
# ...
  - A service is an abstract definition of a computing resource within an application which can be scaled or replaced independently from other components
- Networks → networks:  
# ...
  - The top-level networks element lets you configure named networks that can be reused across multiple services.
- Volumes → volumes:  
# ...
  - The top-level volumes declaration lets you configure named volumes that can be reused across multiple services. To use a volume across multiple services, you must explicitly grant each service access by using the volumes attribute within the services top-level element.

# Docker Compose команди

- `docker compose [OPTIONS] COMMAND`
- Команди:
  - `up` – ги креира, доколку е потребно, и стартува контејнерите.
  - `down` – ги стопира, доколку е потребно, и отстранува контејнерите и мрежите асоцирани со нив,
  - `logs` – преглед на унифицирана верзија на сите логови.
  - `restart` – рестарт на сите или специфичен сервис контејнер.
  - `ps` – листање на контејнерите

# Опкружувачки променливи

- Со користење на .env датотека.

```
$ cat .env  
TAG=v1.5
```

- Променливите се референцираат со \${} нотацијата:

```
$ cat docker-compose.yml  
version: '3'  
services:  
  web:  
    image: "webapp:${TAG}"
```



# Docker Compose Networking

- Docker контејнерите комуницираат помеѓу себе во своја мрежа, која се креира имплицитно или експлицитно преку конфигурација со помош на Docker Compose.

- Еден сервис може да комуницира со друг сервис на иста мрежа преку референцирање

- `<container_name>:<container_port>` (**пример**: `network-example-service:80`),

- **Портата се изложува преку expose**

```
services:  
  network-example-service:  
    image: karthequian/helloworld:latest  
    expose:  
      - "80"
```

- За да се пристапи контејнерот од домаќинот, портите мора да се изложени декларативно преку ports.

# Docker Compose Networking – Пример 1

services:

network-example-service:

image: karthequian/helloworld:latest

ports:

- "80:80"

...

my-custom-app:

image: myapp:latest

ports:

- "8080:3000"

...

my-custom-app-replica:

image: myapp:latest

ports:

- "8081:3000"

- Порта 80 на network-example-service е видлива од домаќинот.
- Портите 3000 на останатите сервиси ќе бидат достапни на портите 8080 (за my-custom-app) и 8081 (за my-custom-app-replica).

# Docker Compose Networking – Пример 2

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    networks:
      - my-shared-network
    ...
  another-service-in-the-same-network:
    image: alpine:latest
    networks:
      - my-shared-network
    ...
  another-service-in-its-own-network:
    image: alpine:latest
    networks:
      - my-private-network
    ...
networks:
  my-shared-network: {}
  my-private-network: {}
```

- another-service-in-the-same-network **ќе може да ја пристапи порта 80 на network-example-service,**
  - Се наоѓаат на иста мрежа (my-shared-network).
- another-service-in-its-own-network **нема да може да ја пристапи порта 80 на network-example-service**
  - Користи своја приватна мрежа.

# Менаџирање на Volumes

- Named Volumes
  - поседуваат кориснички-дефинирано име;
  - едноставни за идентификација, менаџирање и споделување помеѓу повеќе сервиси;
  - Docker има можност да креира и менаџира Named Volumes и ги чува на специфична локација на домаќинот.
- Anonymous Volumes
  - Не поседуваат кориснички-дефинирано име;
  - Docker автоматски ги креира при потреба и им доделува уникатно ID;
  - Тешки за менаџирање – најчесто се користат за привремено складирање на податоци.
- Host Volumes
  - Мапирање на фолдер од датотечниот систем на домаќинот со фолдер на сервисот.

# Менаџирање на Volumes - Пример

```
services:
  volumes-example-service:
    image: alpine:latest
    volumes:
      - my-named-global-volume:/my-volumes/named-global-volume
      - /tmp:/my-volumes/host-volume
      - /home:/my-volumes/readonly-host-volume:ro
    ...
  another-volumes-example-service:
    image: alpine:latest
    volumes:
      - my-named-global-volume:/another-path/the-same-named-global-volume
    ...
volumes:
  my-named-global-volume:
```

- И двата контејнери имаат read/write пристап до my-named-global-volume споделениот фолдер.
- Host volumes: /tmp и /home ќе бидат достапни само за volumes-example-service.
- /tmp фолдерот на домаќинот е мапиран со /my-volumes/host-volume фолдерот на контејнерот со write привилегии.
- /home фолдерот на домаќинот е мапиран со /my-volumes/readonly-host-volume фолдерот на контејнерот со read-only привилегии.

# Дефинирање на зависности (**dependencies**) меѓу сервисите

- Вериги од зависности се потребни за оркестрација на процесот на стартување и завршување на сервисите.
- Дефинирање преку `depends_on`.

```
services:
  kafka:
    image: wurstmeister/kafka:2.11-0.11.0.3
    depends_on:
      - zookeeper:
          condition: service_healthy
    ...
  zookeeper:
    image: wurstmeister/zookeeper
    ...
```

- Во примерот, Kafka побарува сервисот zookeeper да е комплетно стартуван (`condition: service_healthy`) пред да започне со стартување.

# Пример 1 - Services yml

Да се направат два Docker Compose сервиси. Едниот да користи nginx и да сервира html, а другиот да користи Java, да направи GET барање и да врати стринг.

## Решение:

project/

├─ docker-compose.yml

├─ frontend/

| └─ index.html

└─ backend/

├─ Dockerfile

└─ SimpleHttpServer.java

docker-compose.yml

```
version: '3'
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./frontend:/usr/share/nginx/html

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
```

# Пример 1 – Services code

index.html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">

    <title>Simple Docker Compose
Example</title>

</head>

<body>

    <h1>Hello from Docker Compose!</h1>

</body>

</html>
```

SimpleHttpServer.java

```
import java.io.IOException;

import java.io.OutputStream;

import com.sun.net.httpserver.HttpServer;

import com.sun.net.httpserver.HttpHandler;

import com.sun.net.httpserver.HttpExchange;

public class SimpleHttpServer {

    public static void main(String[] args) throws IOException {

        HttpServer server = HttpServer.create(new java.net.InetSocketAddress(8080), 0);

        server.createContext("/", new MyHandler());

        server.setExecutor(null); // creates a default executor

        server.start();

        System.out.println("Server is running on port 8080...");

    }

    static class MyHandler implements HttpHandler {

        @Override

        public void handle(HttpExchange t) throws IOException {

            String response = "Hello from Java HTTP server!";

            t.sendResponseHeaders(200, response.length());

            OutputStream os = t.getResponseBody();

            os.write(response.getBytes());

            os.close();

        }

    }

}
```



# Пример 1 – Services Dockerfile

## Dockerfile

```
FROM openjdk:11-jdk
```

```
WORKDIR /app
```

```
COPY SimpleHttpServer.java /app
```

```
RUN javac SimpleHttpServer.java
```

```
CMD ["java", "SimpleHttpServer"]
```

# Пример 1 - Docker Compose commands

```
docker compose up -d --build
```

...

```
[+] Running 3/3
```

```
✓ Network project_default Created 0.2s
```

```
✓ Container project-web-1 Started 0.4s
```

```
✓ Container project-backend-1 Started 0.4s
```

frontend: localhost:80  
backend: localhost:8080

```
docker compose stop
```

```
[+] Stopping 2/2
```

```
✓ Container project-web-1 Stopped 0.5s
```

```
✓ Container project-backend-1 Stopped
```

```
docker compose start
```

```
[+] Running 2/2
```

```
✓ Container project-backend-1 Started  
0.4s
```

```
✓ Container project-web-1 Started
```

```
docker compose down
```

```
[+] Running 3/3
```

```
✓ Container project-backend-1 Removed  
0.7s
```

```
✓ Container project-web-1 Removed  
0.6s
```

```
✓ Network project_default Removed
```

```
docker compose start
```

```
service "web" has no container to start
```



# Пример 2 – Network, Volumes and Environment variables

Да се надгради пример 1.

Барања:

1. Кодот од сервисите се чува во volumes кои што се викаат (frontend-code, backend-code)
2. Креирајте една мрежа со име: frontend-backend-network, која го има како network driver: bridge.
3. Креирајте опкружувачки променливи: NGINX\_HOST, NGINX\_PORT за сервисот кој користи NGINX и API\_KEY=your\_api\_key во Java сервисот.
4. Стартувајте го Docker compose.
5. Во командна линија излистај ја содржината на frontend-code, backend-code volumes.
6. Во командна линија погледнете ги IPv4Address на двата контерјнери кои се во frontend-backend-network.
7. Во командна линија испечатете ги опкружувачки променливи на двата сервиси.
8. Променете ги вредностите на опкружувачки променливи.

## Пример 2 - yml

docker-compose.yml

```
version: '3.8'

services:
  frontend:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - frontend-code:/usr/share/nginx/html
    networks:
      - frontend-backend-network
    environment:
      - NGINX_HOST=frontend.example.com
      - NGINX_PORT=80
```

```
backend:
  build:
    context: ./backend
    dockerfile: Dockerfile
  ports:
    - "8080:8080"
  volumes:
    - backend-code:/app
  networks:
    - frontend-backend-network
  environment:
    - API_KEY=your_api_key
```

```
networks:
  frontend-backend-network:
    driver: bridge

volumes:
  frontend-code:
  backend-code:
```

# Пример 2

## Команди:

Барање 4:

```
docker compose up -d
```

Барање 5:

```
docker ps
```

```
docker container list
```

```
docker exec <frontend_container_id> ls /usr/share/nginx/html
```

```
docker exec <backend_container_id> ls /app
```

Барање 6:

```
docker inspect <frontend_container_id>
```

```
docker inspect <backend_container_id>
```

Барање 7:

```
docker exec <container_id> env
```

Барање 8:

За да ги промниме вредностите на опкружувачките променливи може да го променме docker-compose.yml

```
docker compose stop
```

```
docker compose up -d
```

```
docker exec <container_id> env
```

## Излез:

```
[+] Running 2/2
✓ Container project-frontend-1 Started
✓ Container project-backend-1 Started
```

CONTAINER ID	IMAGE
e26e82fdec1d	nginx:alpine
ec4404754bc6	project-backend

```
50x.html
index.html
```

```
SimpleHttpServer$MyHandler.class
SimpleHttpServer.class
SimpleHttpServer.java
```

```
"Networks": {
  ...
  "IPAddress": "172.18.0.3",
  ...
}
```

```
HOSTNAME=e26e82fdec1d
NGINX_PORT=80
NGINX_HOST=frontend.example.com
NGINX_VERSION=1.25.5
PKG_RELEASE=1
NJS_VERSION=0.8.4
HOME=/root
```

```
HOSTNAME=e26e82fdec1d
NGINX_PORT=80
NGINX_HOST=service.frontend.example.com
NGINX_VERSION=1.25.5
PKG_RELEASE=1
NJS_VERSION=0.8.4
HOME=/root
```

## Пример 3 – Scaling and Replicas

Направете два Docker Compose сервиси. Првиот сервис треба да е nginx а другиот сервис да е ubuntu контејнер во кој е инсталирана командата curl. Ubuntu сервисот треба да прави бесконечно повикување на nginx со помош на curl командата.

3.1 Направете 1 реплика од ubuntu а 5 реплики од nginx.

3.2 Направете 5 реплики од ubuntu и 5 реплики од nginx.

3.3 Направете 100 реплики од ubuntu и 1 реплика од nginx.

Анализирај го користењето на ресурсите на контејнерите со користење на docker stats.

# Пример 3 - yml

docker-compose.yml

- replicas во docker-compose.yml или
- docker-compose up -d --scale nginx=5 --scale ubuntu=1

```
version: '3.8'

services:
  nginx:
    image: nginx
    deploy:
      replicas: 5

  ubuntu:
    image: ubuntu
    command: ["bash", "-c", "apt-get update && apt-get install -y curl && while true; do curl -s -o /dev/null http://nginx; sleep 1; done"]
    deploy:
      replicas: 1
    depends_on:
      - nginx
```

# Пример 3 – containers

`docker compose up`

[+] Running 6/0

✓ Container scale-nginx-1 Created 0.0s

✓ Container scale-nginx-2 Created 0.0s

✓ Container scale-nginx-3 Created 0.0s

✓ Container scale-nginx-4 Created 0.0s

✓ Container scale-nginx-5 Created 0.0s

✓ Container scale-ubuntu-1 Created

...

nginx-2 | 172.27.0.7 - - [21/Apr/2024:01:48:03 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "-"

nginx-2 | 172.27.0.7 - - [21/Apr/2024:01:48:04 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "-"

nginx-2 | 172.27.0.7 - - [21/Apr/2024:01:48:05 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "-"

nginx-2 | 172.27.0.7 - - [21/Apr/2024:01:48:06 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "-"

...

INF



# Пример 3 – docker stats (nginx - 5, ubuntu 1)



docker stats

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
6392950ac8fd	scale-ubuntu-1	1.68%	1008KiB / 15.54GiB	0.01%	26.8kB / 12.1kB	0B / 0B	2
9417ac5ffbcd	scale-nginx-5	0.00%	4.312MiB / 15.54GiB	0.03%	4.9kB / 656B	0B / 0B	5
dbd96f8c99b4	scale-nginx-1	0.00%	4.359MiB / 15.54GiB	0.03%	4.61kB / 656B	0B / 0B	5
1477f810f308	scale-nginx-4	0.00%	4.316MiB / 15.54GiB	0.03%	4.32kB / 656B	0B / 0B	5
6ba3e0347edb	scale-nginx-3	0.00%	4.332MiB / 15.54GiB	0.03%	4.23kB / 656B	0B / 0B	5
17af9994325f	scale-nginx-2	0.04%	4.406MiB / 15.54GiB	0.03%	13kB / 20.9kB	0B / 0B	5

## Пример 3 – docker stats (nginx - 5, ubuntu - 5)

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
d9af0c90ab4c	scale-ubuntu-3	0.59%	49.79MiB / 15.54GiB	0.31%	35.9MB / 459kB	0B / 0B	2
ec1fdecdaa9b	scale-ubuntu-5	1.06%	49.73MiB / 15.54GiB	0.31%	35.9MB / 477kB	0B / 0B	2
e6184f9ac9ac	scale-ubuntu-4	1.03%	49.71MiB / 15.54GiB	0.31%	35.9MB / 539kB	0B / 0B	2
f642ed52b75c	scale-ubuntu-2	1.05%	49.73MiB / 15.54GiB	0.31%	35.9MB / 563kB	0B / 0B	2
6392950ac8fd	scale-ubuntu-1	1.27%	1008KiB / 15.54GiB	0.01%	303kB / 35.3kB	0B / 0B	2
9417ac5ffbcd	scale-nginx-5	0.03%	4.41MiB / 15.54GiB	0.03%	23.4kB / 31.9kB	0B / 0B	5
dbd96f8c99b4	scale-nginx-1	0.17%	4.426MiB / 15.54GiB	0.03%	18.4kB / 21.3kB	0B / 0B	5
1477f810f308	scale-nginx-4	0.00%	4.387MiB / 15.54GiB	0.03%	22.9kB / 31.9kB	0B / 0B	5
6ba3e0347edb	scale-nginx-3	0.07%	4.41MiB / 15.54GiB	0.03%	20.4kB / 27.2kB	0B / 0B	5
17af9994325f	scale-nginx-2	0.02%	4.395MiB / 15.54GiB	0.03%	37.3kB / 65.3kB	0B / 0B	5

# Пример 3 - docker stats (nginx - 5, ubuntu - 5)

<  **scale-nginx-5**  
nginx  
074cb074651c 

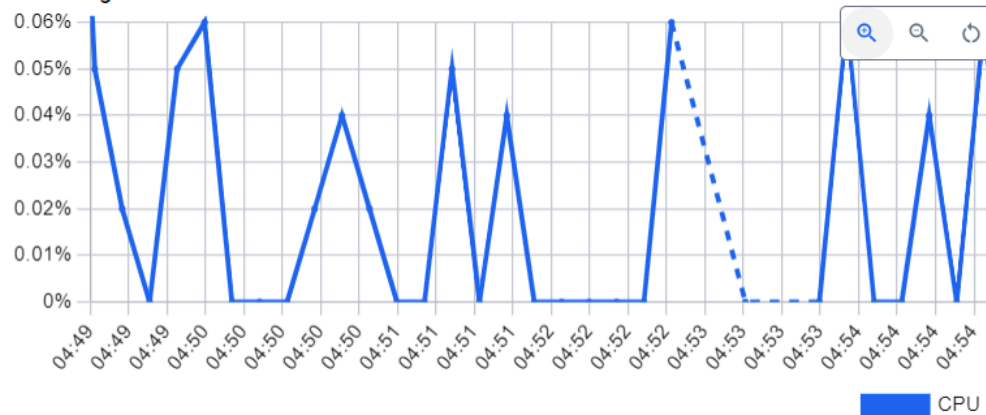
STATUS

Running (5 minutes ago)

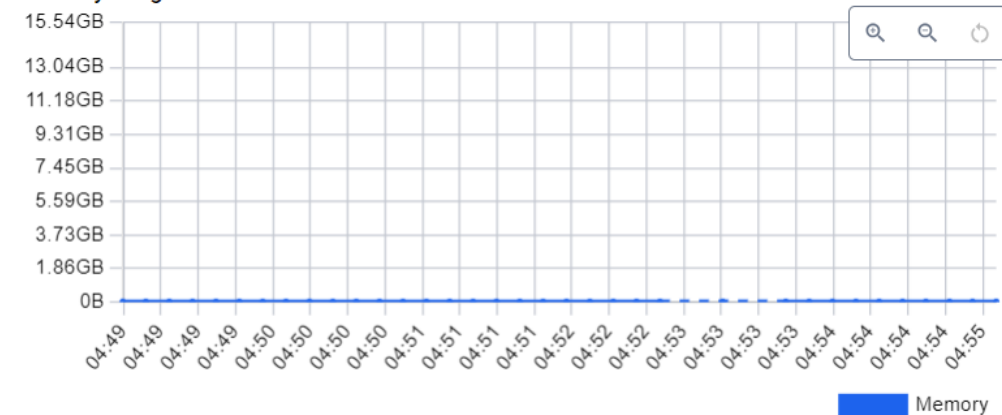


Logs Inspect Bind mounts Exec Files **Stats**

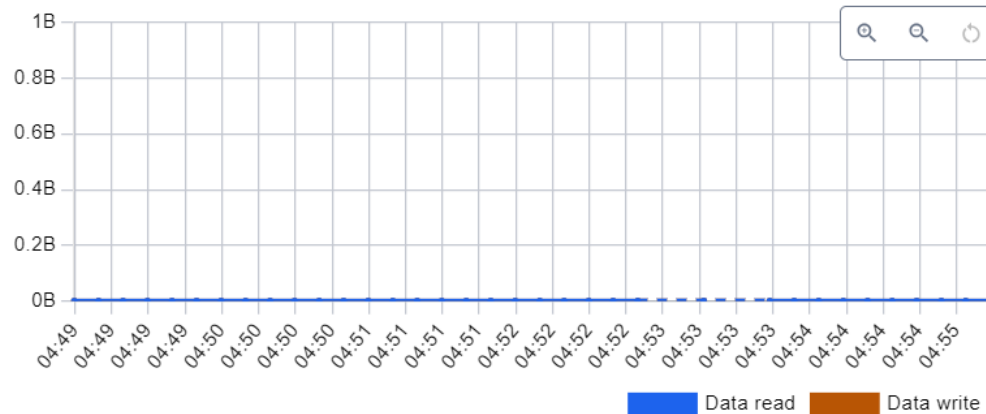
CPU usage: 0.06%



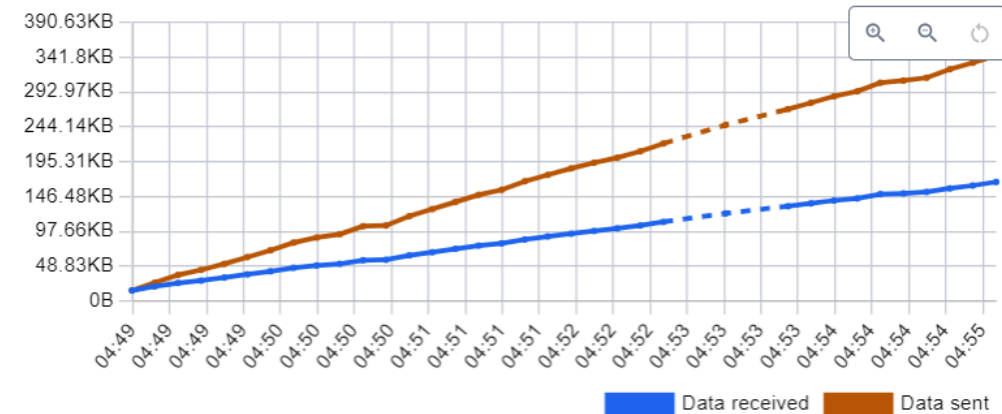
Memory usage: 4.46MB / 15.54GB



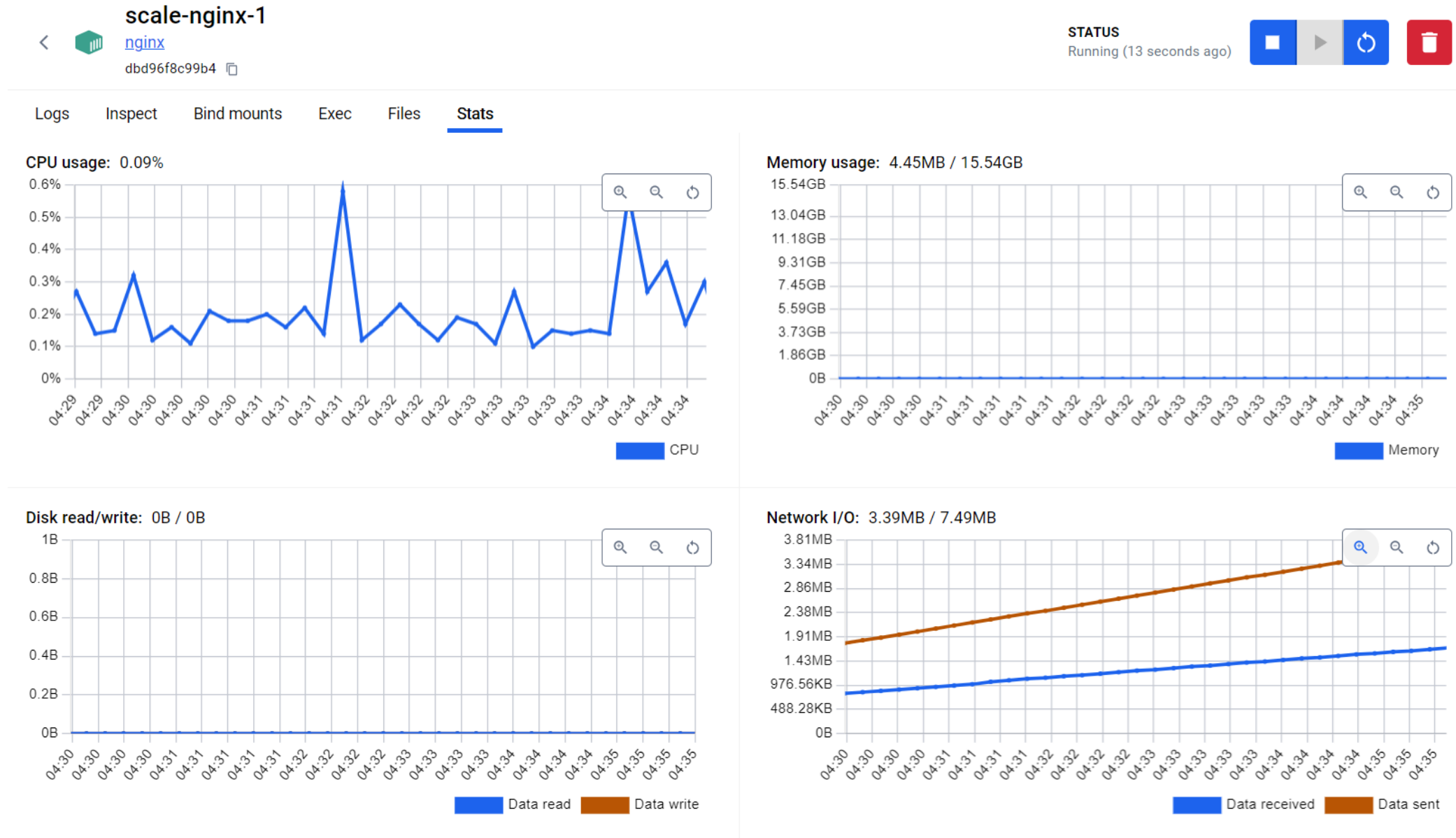
Disk read/write: 0B / 0B



Network I/O: 167KB / 344KB



# Пример 3 - docker stats (nginx - 1, ubuntu - 5)



# Пример 3 - docker stats (nginx - 1, ubuntu - 100)

## CPU CORES USAGE ⓘ

Allocated: 4

310.02% / 400%

## MEMORY USAGE

2.55GB / 15.54GB

## CONTAINERS

Application containers: 102

System containers: 3 ⓘ

101 / 105 running

[Go to containers](#)

Table view

Chart view

Columns

Filters

Name	Status	CPU (%)	Memory Usage...	MEM (%)	Disk Read/Write	Network I/O	PIDS	
scale (101)	running	310.02%	2.55GB / 15.54GB	16.43%	0B / 0B	2.86GB / 45.9MB	6,6,6,6,6,6,2,2,...	■ ⋮
scale-nginx-1	running	0.10%	4.43MB / 15.54GB	0.03%	0B / 0B	265KB / 383KB	5	■ ⋮
scale-ubuntu-1	running	0.30%	7.88MB / 15.54GB	0.05%	0B / 0B	22.1MB / 399KB	6	■ ⋮
scale-ubuntu-40	running	1.15%	7.98MB / 15.54GB	0.05%	0B / 0B	22.4MB / 339KB	6	■ ⋮
scale-ubuntu-19	running	0.11%	8.02MB /	0.05%	0B / 0B	23MB / 298KB	6	■ ⋮

**ПРАШАЊА**