

## **Методических материалов по темам «Контейнеризации и оркестровки контейнеров» с использованием технологической платформы ПАО «Ростелеком»**

В результате проведения практических работ по темам **«Контейнеризации и оркестровки контейнеров»** осуществляется проверка и формирование нижеследующих умений и знаний.

*Результаты обучения: умения, знания*

Уметь:

- Управлять ресурсами технологической платформы ПАО «Ростелеком»
- устанавливать и сопровождать системы контейнеризации;
- выполнять оптимизацию системы в зависимости от поставленных задач;

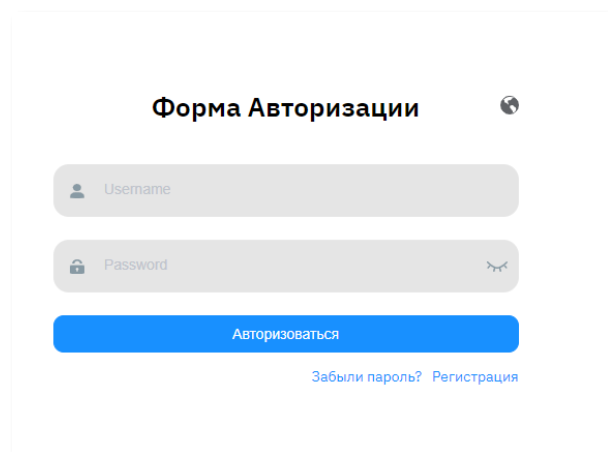
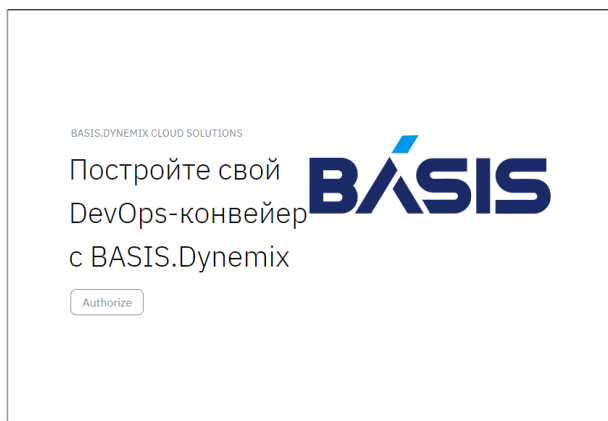
*Знать:*

- порядок настройки виртуальной машины через пользовательский интерфейс технологической платформы ПАО «Ростелеком»
- порядок установки Docker, Docker compose;
- последовательность запуска контейнера на основе Image;
- настройки Volumes;
- настройка Network drivers;
- сборка образов;
- безопасность контейнеров;
- способы подключения к кластеру Kubernetes;
- запуск контейнеров в Kubernetes.

*Критерии оценки*

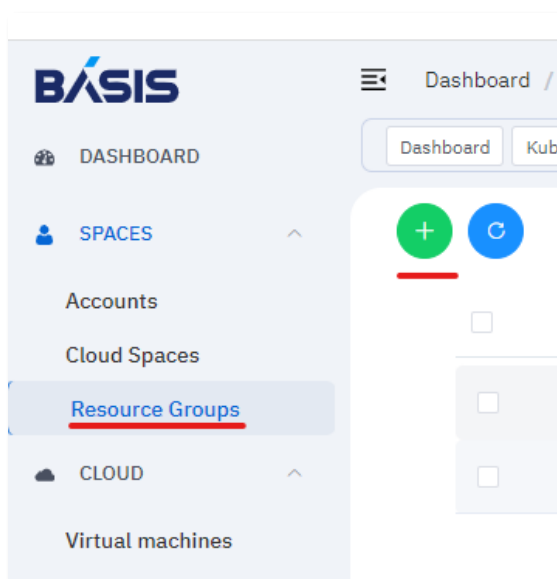
- полнота выполнения задания;
- качество оформления отчёта;
- правильность ответов на контрольные вопросы.

## 1. Авторизация в технологической платформе ПАО «Ростелеком» (ТП РТК)



The image shows a login form titled "Форма Авторизации" (Authorization Form) in a bold, black font. To the right of the title is a small globe icon. Below the title, there are two input fields: the first is labeled "Username" with a user icon on the left, and the second is labeled "Password" with a lock icon on the left and a toggle icon on the right. Below these fields is a large blue button with the text "Авторизоваться" (Authorize). At the bottom right, there are two links: "Забыли пароль?" (Forgot password?) and "Регистрация" (Registration).

## 2. Создание ресурсной группы



Create a new Resource Group

1 ————— 2

General QoS

\* Name

Description

\* Account

Owner

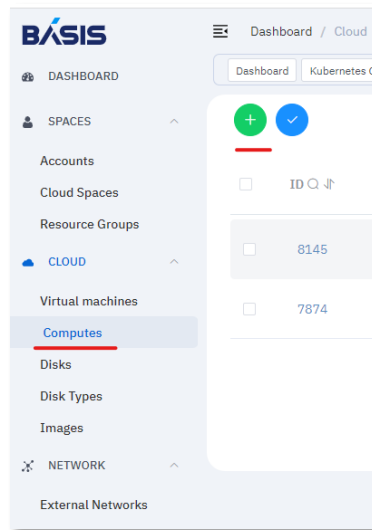
Register Computes ☐ Yes ☒ No

\* Default Network

\* External Network

Next Step

### 3. Создание виртуального сервера



Указываем ресурсную группу, имя, архитектуру и драйвер

This is the first step of a three-step wizard to create a new compute instance. The steps are labeled 1 (General), 2 (Sizes), and 3 (Network). The 'General' step is active. It contains several fields: 'Account' (Universiade\_education), 'Resource Group' (rg), 'Name' (Test), 'Description' (empty), 'Architecture' (X86\_64 selected, PPC64\_LE unselected), and 'Driver' (KVM\_X86). A 'Next Step' button is at the bottom right.

Указываем количество CPU, объем памяти, образ системы, объем жесткого диска

This is the second step of the wizard, 'Sizes'. Step 1 is marked as complete with a green checkmark. The 'Sizes' step is active. It contains four fields: 'CPU' (1), 'Memory in MB' (512), 'Image' (OS Ubuntu 18.04), and 'Vdisk in GB' (3). 'Previous step' and 'Next Step' buttons are at the bottom.

## Указываем тип сети (External Network) и тип IP Address

Create a new Compute

General

Sizes

3  
Network

\* Network Type

☐ None

☐ ViNS

☒ External Network

\* Select Network

Select External Network ^

\* IP Address Type

Previous

176.118.164.0/24 176.118.164.0/24

45.134.255.0/24 45.134.255.0/24

88.218.249.0/24 88.218.249.0/24

## Просмотр информации о созданной виртуальной машине: IP Address и Login/Password

Dashboard / Cloud / Compute

Dashboard Computes Compute 7874

General Console Users Access Disks Network Snapshots GPU Devices PCI Devices Logs

| Parameter       | Value                   |
|-----------------|-------------------------|
| ID              | 7874                    |
| Name            | srv-                    |
| Description     | -                       |
| Status          | ENABLED                 |
| Tech.Status     | STARTED                 |
| IP Addresses    | * 8.2 * .249.2 *        |
| Login           | user                    |
| Password        | *****                   |
| CPU             | 2                       |
| Memory (MB)     | 2048                    |
| Storage (GB)    | 20                      |
| Image ID        | 361                     |
| Image Name      | OS Ubuntu 18.04         |
| Architecture    | X86_64                  |
| Driver          | KVM_X86                 |
| Pinned to Stack | -                       |
| Creation Time   | 2022-05-02 11:11:57     |
| Created By      | alexey_eliseev_1@decs3o |
| Updated Time    | 2022-05-02 11:32:37     |
| Updated By      | *****_1@decs3o          |
| Deletion Time   | -                       |
| Deleted By      | -                       |

#### 4. Подключаемся к созданному ресурсу

```
$ ssh user@xxx.xxx.xxx.xxx
```

user и xxx.xxx.xxx.xxx – указать в соответствии с созданными параметрами виртуальной машины (см. выше)

#### 5. Настраиваем репозиторий

Обновите `apt` индекс пакетов и установите пакеты, чтобы разрешить `apt` использование репозитория через HTTPS:

```
$ sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Добавьте официальный GPG-ключ Docker:

```
$ curl -fsSL
https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /usr/share/keyrings/docker-archive-
keyring.gpg
```

#### 6. Установите Engine Docker

Обновите `apt` индекс пакета и установите последнюю версию `Docker Engine`, `containerd` и `Docker Compose`:

```
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli
containerd.io docker-compose-plugin
```

Убедитесь, что Docker Engine установлен правильно, запустив `hello-world` образ.

```
$ sudo docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
535020c3e8ad: Pull complete
af340544ed62: Pull complete
Digest:
sha256:a68868bfe696c00866942e8f5ca39e3e31b79c1e50feae
e4ce5e28df2f051d5c
Status: Downloaded newer image for hello-world:latest

Hello from Docker.

This message shows that your installation appears to
be working correctly.

To generate this message, Docker took the following
steps:

1. The Docker Engine CLI client contacted the Docker
Engine daemon.

2. The Docker Engine daemon pulled the "hello-world"
image from the Docker Hub.

3. The Docker Engine daemon created a new container
from that image which runs the
```



executable that produces the output you are currently reading.

4. The Docker Engine daemon streamed that output to the Docker Engine CLI client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker Hub account:

<https://hub.docker.com>

For more examples and ideas, visit:

<https://docs.docker.com/userguide/>

Эта команда загружает тестовый образ и запускает его в контейнере. Когда контейнер запускается, он печатает сообщение и завершает работу.

Теперь выполните `docker ps -a` что бы увидеть все контейнеры в системе.

```
$ docker ps -a
```

| CONTAINER ID | IMAGE       | COMMAND  | CREATED        | STATUS        |
|--------------|-------------|----------|----------------|---------------|
| PORTS        | NAMES       |          |                |               |
| 592376ff3eb8 | hello-world | "/hello" | 25 seconds ago | Exited (0) 24 |
| seconds ago  |             | no name  |                |               |

Вы увидите ваш `hello-world` контейнер в списке, выведенном командой `docker ps -a`.

Команда `docker ps` отображает только запущенные контейнеры.

Поскольку `hello-world` уже выполнен и завершен, то соответствующий контейнер не отображается по команде `docker ps`.

## 7. Сборка и оптимизация образов на основе Dockerfile

Для выполнения этого задания нужно установить `docker-compose`

Создаем директорию `hello-docker`

Создаем в ней два файла:

`main.go`:

```
package main

import (
    "fmt"
    "net/http"

    "github.com/sirupsen/logrus"
)

func handle(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello docker!")
}

func main() {
    logrus.Info("App run")
    if err := http.ListenAndServe(":8080", http.HandlerFunc(handle));
    err != nil {
        logrus.Error(err)
    }
}
```

### Dockerfile:

```
FROM golang:alpine AS builder

WORKDIR /src

COPY main.go .

RUN go mod init hello-docker

RUN go mod tidy

RUN go build

FROM alpine:3.15

WORKDIR /app

COPY --from=builder /src/hello-docker /app/hello-docker

EXPOSE 8080

ENTRYPOINT ["/app/hello-docker"]
```

### Производим сборку:

```
$ docker build . -t hwi
```

```
[+] Building 1.3s (15/15) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 297B 0.0s
=> [internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/alpine:3.15 1.0s
=> [internal] load metadata for docker.io/library/golang:alpine 1.0s
=> [internal] load build context 0.0s
=> => transferring context: 29B 0.0s
=> [stage-1 1/3] FROM docker.io/library/alpine:3.15@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aa 0.0s
=> [builder 1/6] FROM docker.io/library/golang:alpine@sha256:7cc62574fcf9c5fb87ad42a9789d5539a6a085971d58ee 0.0s
=> CACHED [stage-1 2/3] WORKDIR /app 0.0s
=> CACHED [builder 2/6] WORKDIR /src 0.0s
=> CACHED [builder 3/6] COPY main.go . 0.0s
=> CACHED [builder 4/6] RUN go mod init hello-docker 0.0s
=> CACHED [builder 5/6] RUN go mod tidy 0.0s
=> CACHED [builder 6/6] RUN go build 0.0s
=> CACHED [stage-1 3/3] COPY --from=builder /src/hello-docker /app/hello-docker 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:b0e8a28756eae119de8cb4b40e84da9558c479078bb40679eb67075a485cf6a7 0.0s
=> => naming to docker.io/library/hwi 0.0s
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

## Запускаем собранный образ

```
$ docker run hwi -p 8080:8080
time="2022-06-23T16:15:34Z" level=info msg="App run"
```

## Проверяем доступность запущенного приложения

```
$ curl localhost:8080
Hello docker!
```

Для сравнения размеров образов полученного приложения и образа со средой компиляции приложения go выполним следующую сборку:

```
$ docker build --target builder . -t hwi-intermediate
```

```
[+] Building 3.1s (11/11) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 32B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/golang:alpine 1.9s
=> [builder 1/6] FROM docker.io/library/golang:alpine@sha256:7cc62574fcf9c5fb87ad42a9789d5539a6a085971d58ee 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 346B 0.0s
=> CACHED [builder 2/6] WORKDIR /src 0.0s
=> CACHED [builder 3/6] COPY main.go . 0.0s
=> CACHED [builder 4/6] RUN go mod init hello-docker 0.0s
=> CACHED [builder 5/6] RUN go mod tidy 0.0s
=> CACHED [builder 6/6] RUN go build 0.0s
=> exporting to image 1.0s
=> => exporting layers 0.9s
=> => writing image sha256:ac4dac42fc7e97dc6befb6b2afb3d829d285935e5df9b178990e0681311704b4 0.0s
=> => naming to docker.io/library/hwi-intermediate 0.0s
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

Теперь сравним размеры образов

```
$ docker image ls | grep hwi
```

|                  |        |              |                |       |
|------------------|--------|--------------|----------------|-------|
| hwi              | latest | b0e8a28756ea | 42 minutes ago | 12MB  |
| hwi-intermediate | latest | ac4dac42fc7e | 2 hours ago    | 349MB |

## 8. Запуск docker-compose с применением различных Network drivers в docker

Создаем директорию `networks`

В директории создаем следующие директории:

```
code
docker
docker/nginx
docker/php-fpm
```

В директориях создаем следующие файлы:

Docker-compose.yml:

```
version: '3'
services:
  nginx:
    image: nginx:1.17.8
```

```

ports:
  - "8080:80"
volumes:
  - ./code:/code
  - ./docker/nginx/site.conf:/etc/nginx/conf.d/site.conf

php:
  build:
    context: docker/php-fpm
  volumes:
    - ./code:/code
    - ./docker/php-fpm/php.ini:/usr/local/etc/php/php.ini

db:
  image: mysql:8.0
  restart: always
  environment:
    MYSQL_DATABASE: 'base'
    MYSQL_USER: 'user'
    MYSQL_PASSWORD: '12345'
    MYSQL_ROOT_PASSWORD: 'root'
  volumes:
    - ./docker/db:/var/lib/mysql
  ports:
    - "3306:3306"

```

code/index.php:

```

<?php

$greetingWord = 'Hello!';

echo '<div>' . $greetingWord . '</div>';

```

docker/nginx/site.conf:

```

server {

```

```

index index.php index.html;

server_name base-dev.local;

error_log /var/log/nginx/error.log;
access_log /var/log/nginx/access.log;

root /code;


location ~ /\.php$ {
    try_files $uri =404;
    fastcgi_split_path_info ^(.+\.php) (/\.+)$;
    fastcgi_pass php:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
}
}

```

**docker/php-fpm/Dockerfile:**

```

FROM php:7.4-fpm

RUN pecl install xdebug-2.9.1 \
    && docker-php-ext-enable xdebug

RUN docker-php-ext-install pdo_mysql

```

**docker/php-fpm/php.ini:**

```

max_execution_time = 1000
max_input_time = 1000

```

**Запускаем сборку:**

```
$ docker-compose up
```

## Подождём этап сборки и запуска приложений в контейнерах:

```
[+] Building 100.3s (7/7) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 160B                                              0.0s
=> [internal] load .dockerignore                                                0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/php:7.4-fpm                 2.6s
=> [1/3] FROM docker.io/library/php:7.4-
fpm@sha256:4c6b8927a4757797e7e6d3e159cf5ec7f1c14171758b442aeac1f6ee3244d926 34.5s
=> => resolve docker.io/library/php:7.4-
fpm@sha256:4c6b8927a4757797e7e6d3e159cf5ec7f1c14171758b442aeac1f6ee3244d926 0.0s
=> => sha256:78fdfd2598e0ffdaa39011b909e2a79a75a60a0a87998f1072ec5d9256f19868 225B / 225B 0.4s
=> => sha256:26769c8659f467675c0f34948c16e051ea7aab69ec3198c65063c299b9771a05 91.60MB / 91.60MB 18.3s
=> => sha256:bdaf6fac8dceba28623cd6b0f4d8b8ea66c7fbbb234838f05a3eb6bfe083e69c 2.41kB / 2.41kB 0.0s
=> => sha256:2bd77e634ff6f1ca51fe0acbc5a57a231a5c35563c3cfd474e81eb160350d0f9 11.38kB / 11.38kB 0.0s
=> => sha256:4c6b8927a4757797e7e6d3e159cf5ec7f1c14171758b442aeac1f6ee3244d926 1.86kB / 1.86kB 0.0s
=> => sha256:b85a868b505ffd0342a37e6a3b1c49f7c71878afe569a807e6238ef08252fcb7 31.38MB / 31.38MB 5.1s
=> => sha256:0bd105fadbe34a7e12eb66c424bd0de4b9c2c5c2daf01063eef044ff10e5e437 271B / 271B 0.8s
=> => sha256:2fad39daff2b705ef90885be71ba6dace3eec5dac380662c17bff9c50714687 10.74MB / 10.74MB 5.4s
=> => sha256:acf0bff511002457219f797053d37f627e32240061fd4ecd4c18dd01901a8950 495B / 495B 5.6s
=> => extracting sha256:b85a868b505ffd0342a37e6a3b1c49f7c71878afe569a807e6238ef08252fcb7 6.9s
=> => sha256:402939ef1cc5a8c6bad9733a05f9ca1593f003e96bdf21c816d354153ea38c34 25.40MB / 25.40MB 14.9s
```



```

=> => sha256:6886d213a86f3c3fe28e21ccb352853d2b49bf867e4bb1c4ac56e0ca33da18b7 2.45kB / 2.45kB 6.0s
=> => sha256:89cb460f70a117898c9979657d357df8262a7d1e4ea0d56f08b97297b0f69eb6 244B / 244B 6.8s
=> => sha256:4fdf0d4fa32467559369089c9ca0e7249dc9e46b69d85c10ff6a65246e46e94f 8.45kB / 8.45kB 7.4s
=> => extracting sha256:78fdfd2598e0ffdaa39011b909e2a79a75a60a0a87998f1072ec5d9256f19868 0.0s
=> => extracting sha256:26769c8659f467675c0f34948c16e051ea7aab69ec3198c65063c299b9771a05 10.1s
=> => extracting sha256:0bd105fadbe34a7e12eb66c424bd0de4b9c2c5c2daf01063eef044ff10e5e437 0.0s
=> => extracting sha256:2fadb39daff2b705ef90885be71ba6dace3eec5dac380662c17bff9c50714687 0.2s
=> => extracting sha256:acf0bfff511002457219f797053d37f627e32240061fd4ecd4c18dd01901a8950 0.0s
=> => extracting sha256:402939ef1cc5a8c6bad9733a05f9ca1593f003e96bdf21c816d354153ea38c34 2.9s
=> => extracting sha256:6886d213a86f3c3fe28e21ccb352853d2b49bf867e4bb1c4ac56e0ca33da18b7 0.0s
=> => extracting sha256:89cb460f70a117898c9979657d357df8262a7d1e4ea0d56f08b97297b0f69eb6 0.0s
=> => extracting sha256:4fdf0d4fa32467559369089c9ca0e7249dc9e46b69d85c10ff6a65246e46e94f 0.0s
=> [2/3] RUN pecl install xdebug-2.9.1 && docker-php-ext-enable xdebug 42.8s
=> [3/3] RUN docker-php-ext-install pdo_mysql 19.3s
=> exporting to image 0.7s
=> => exporting layers 0.4s
=> => writing image sha256:c6efbcfafd1752a062cdda8871025301bce783fbcd947d6048dd25bc86aabbcb8 0.0s
=> => naming to docker.io/library/2-live-networks_php 0.0s

```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

[+] Running 4/4

```

- Network 2-live-networks_default Created 0.6s

```

```

- Container 2-live-networks-php-1      Created                               2.1s
- Container 2-live-networks-nginx-1    Created                               1.8s
- Container 2-live-networks-db-1       Created                               2.1s
Attaching to 2-live-networks-db-1, 2-live-networks-nginx-1, 2-live-networks-php-1
2-live-networks-db-1      | 2022-06-23 17:27:43+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server
2-live-networks-db-1      | 8.0.29-1debian10 started.
2-live-networks-php-1     | [23-Jun-2022 17:27:44] NOTICE: fpm is running, pid 1
2-live-networks-php-1     | [23-Jun-2022 17:27:44] NOTICE: ready to handle connections
2-live-networks-db-1      | 2022-06-23 17:27:45+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2-live-networks-db-1      | 2022-06-23 17:27:45+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server
2-live-networks-db-1      | 8.0.29-1debian10 started.
2-live-networks-db-1      | 2022-06-23 17:27:45+00:00 [Note] [Entrypoint]: Initializing database files
2-live-networks-db-1      | 2022-06-23T17:27:45.646119Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld
2-live-networks-db-1      | (mysqld 8.0.29) initializing of server in progress as process 44
2-live-networks-db-1      | 2022-06-23T17:27:45.703261Z 0 [Warning] [MY-010159] [Server] Setting
2-live-networks-db-1      | lower_case_table_names=2 because file system for /var/lib/mysql/ is case insensitive
2-live-networks-db-1      | 2022-06-23T17:27:45.817354Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization
2-live-networks-db-1      | has started.
2-live-networks-db-1      | 2022-06-23T17:27:52.307662Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization
2-live-networks-db-1      | has ended.
2-live-networks-db-1      | 2022-06-23T17:28:02.416249Z 6 [Warning] [MY-010453] [Server] root@localhost is
2-live-networks-db-1      | created with an empty password ! Please consider switching off the
2-live-networks-db-1      | --initialize-insecure option.
2-live-networks-db-1      | 2022-06-23 17:28:20+00:00 [Note] [Entrypoint]: Database files initialized
2-live-networks-db-1      | 2022-06-23 17:28:20+00:00 [Note] [Entrypoint]: Starting temporary server

```

```

2-live-networks-db-1      | 2022-06-23T17:28:20.784151Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld
(mysql 8.0.29) starting as process 93

2-live-networks-db-1      | 2022-06-23T17:28:20.799480Z 0 [Warning] [MY-010159] [Server] Setting
lower_case_table_names=2 because file system for /var/lib/mysql/ is case insensitive

2-live-networks-db-1      | 2022-06-23T17:28:20.853460Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization
has started.

2-live-networks-db-1      | 2022-06-23T17:28:22.495678Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization
has ended.

2-live-networks-db-1      | 2022-06-23T17:28:23.987921Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem
is self signed.

2-live-networks-db-1      | 2022-06-23T17:28:23.988803Z 0 [System] [MY-013602] [Server] Channel mysql_main
configured to support TLS. Encrypted connections are now supported
for this channel.

2-live-networks-db-1      | 2022-06-23T17:28:24.048946Z 0 [Warning] [MY-011810] [Server] Insecure configuration
for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a
different directory.

2-live-networks-db-1      | 2022-06-23T17:28:24.220678Z 0 [System] [MY-011323] [Server] X Plugin ready for
connections. Socket: /var/run/mysqld/mysqlx.sock

2-live-networks-db-1      | 2022-06-23T17:28:24.224287Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready
for connections. Version: '8.0.29' socket: '/var/run/mysqld/mysqld.sock' port: 0 MySQL Community Server -
GPL.

2-live-networks-db-1      | 2022-06-23 17:28:24+00:00 [Note] [Entrypoint]: Temporary server started.

2-live-networks-db-1      | Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping
it.

2-live-networks-db-1      | Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone.
Skipping it.

2-live-networks-db-1      | Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.

```

```

2-live-networks-db-1      | Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping
it.
2-live-networks-db-1      | 2022-06-23 17:28:34+00:00 [Note] [Entrypoint]: Creating database base
2-live-networks-db-1      | 2022-06-23 17:28:34+00:00 [Note] [Entrypoint]: Creating user user
2-live-networks-db-1      | 2022-06-23 17:28:34+00:00 [Note] [Entrypoint]: Giving user user access to schema
base
2-live-networks-db-1      |
2-live-networks-db-1      | 2022-06-23 17:28:34+00:00 [Note] [Entrypoint]: Stopping temporary server
2-live-networks-db-1      | 2022-06-23T17:28:34.642435Z 13 [System] [MY-013172] [Server] Received SHUTDOWN from
user root. Shutting down mysqld (Version: 8.0.29).
2-live-networks-db-1      | 2022-06-23T17:28:37.759863Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld:
Shutdown complete (mysqld 8.0.29) MySQL Community Server - GPL.
2-live-networks-db-1      | 2022-06-23 17:28:38+00:00 [Note] [Entrypoint]: Temporary server stopped
2-live-networks-db-1      |
2-live-networks-db-1      | 2022-06-23 17:28:38+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for
start up.
2-live-networks-db-1      |
2-live-networks-db-1      | 2022-06-23T17:28:39.070794Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld
(mysqld 8.0.29) starting as process 1
2-live-networks-db-1      | 2022-06-23T17:28:39.104767Z 0 [Warning] [MY-010159] [Server] Setting
lower_case_table_names=2 because file system for /var/lib/mysql/ is case insensitive
2-live-networks-db-1      | 2022-06-23T17:28:39.204213Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization
has started.
2-live-networks-db-1      | 2022-06-23T17:28:45.766135Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization
has ended.

```

```
2-live-networks-db-1      | 2022-06-23T17:28:48.968577Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem
is self signed.

2-live-networks-db-1      | 2022-06-23T17:28:48.971560Z 0 [System] [MY-013602] [Server] Channel mysql_main
configured to support TLS. Encrypted connections are now supported
for this channel.

2-live-networks-db-1      | 2022-06-23T17:28:49.269435Z 0 [Warning] [MY-011810] [Server] Insecure configuration
for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a
different directory.

2-live-networks-db-1      | 2022-06-23T17:28:49.684209Z 0 [System] [MY-011323] [Server] X Plugin ready for
connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock

2-live-networks-db-1      | 2022-06-23T17:28:49.684301Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready
for connections. Version: '8.0.29' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server
- GPL.
```

Проверяем запуск наших контейнеров:

```
$ docker ps
```

| CONTAINER ID | IMAGE               | NAMES                   | COMMAND                  | CREATED       | STATUS       | PORTS                  |
|--------------|---------------------|-------------------------|--------------------------|---------------|--------------|------------------------|
| 74c954f154d5 | mysql:8.0           |                         | "docker-entrypoint.s..." | 7 minutes ago | Up 7 minutes | 0.0.0.0:3306->3306/tcp |
| 06->3306/tcp | 33060/tcp           | 2-live-networks-db-1    |                          |               |              |                        |
| 93c127f53b65 | nginx:1.17.8        |                         | "nginx -g 'daemon of..." | 7 minutes ago | Up 7 minutes | 0.0.0.0:80->80/tcp     |
| 80->80/tcp   |                     | 2-live-networks-nginx-1 |                          |               |              |                        |
| 65418eb1d89d | 2-live-networks_php |                         | "docker-php-entrypoi..." | 7 minutes ago | Up 7 minutes | 9000/tcp               |
|              |                     | 2-live-networks-php-1   |                          |               |              |                        |

Проинспектируем созданные сети:

```
$ docker network ls
```

| NETWORK ID   | NAME                    | DRIVER | SCOPE |
|--------------|-------------------------|--------|-------|
| bb0141995383 | 2-live-networks_default | bridge | local |
| 0507ad46874b | bridge                  | bridge | local |
| 3ceb0f16d293 | host                    | host   | local |
| 2a3f49dae137 | none                    | null   | local |

Остановим наши контейнеры `Ctrl+C`

Модифицируем файл

Docker-compose.yml:

```
version: '3'
services:
  nginx:
    image: nginx:1.17.8
    ports:
      - "8080:80"
    volumes:
      - ./code:/code
      -
      ./docker/nginx/site.conf:/etc/nginx/conf.d/site.conf
    network_mode: none
```

```
php:
  build:
    context: docker/php-fpm
  volumes:
    - ./code:/code
    - ./docker/php-fpm/php.ini:/usr/local/etc/php/php.ini
  network_mode: none

db:
  image: mysql:8.0
  restart: always
  environment:
    MYSQL_DATABASE: 'base'
    MYSQL_USER: 'user'
    MYSQL_PASSWORD: '12345'
    MYSQL_ROOT_PASSWORD: 'root'
  volumes:
    - ./docker/db:/var/lib/mysql
  ports:
    - "3306:3306"
  network_mode: none
```

Добавили в каждый блок описания сервиса `network_mode: none`

Запускаем наши контейнеры `$ docker-compose up`

```
[+] Running 3/0
- Container 2-live-networks-php-1    Created                                0.0s
- Container 2-live-networks-nginx-1  Created                                0.0s
- Container 2-live-networks-db-1     Created                                0.0s
Attaching to 2-live-networks-db-1, 2-live-networks-nginx-1, 2-live-networks-php-1
2-live-networks-nginx-1 | 2022/06/23 17:46:34 [emerg] 1#1: host not found in upstream "php" in /etc/nginx/conf.d/site.conf:11
2-live-networks-nginx-1 | nginx: [emerg] host not found in upstream "php" in /etc/nginx/conf.d/site.conf:11
2-live-networks-db-1    | 2022-06-23 17:46:34+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.29-1debian10
started.
2-live-networks-php-1   | [23-Jun-2022 17:46:34] NOTICE: fpm is running, pid 1
2-live-networks-php-1   | [23-Jun-2022 17:46:34] NOTICE: ready to handle connections
2-live-networks-nginx-1 exited with code 1
2-live-networks-db-1    | 2022-06-23 17:46:35+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2-live-networks-db-1    | 2022-06-23 17:46:35+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.29-1debian10
started.
2-live-networks-db-1    | 2022-06-23T17:46:35.593621Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.29) starti
ng as process 1
2-live-networks-db-1    | 2022-06-23T17:46:35.606592Z 0 [Warning] [MY-010159] [Server] Setting lower_case_table_names=2 becau
se file system for /var/lib/mysql/ is case insensitive
2-live-networks-db-1    | 2022-06-23T17:46:35.637817Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2-live-networks-db-1    | 2022-06-23T17:46:36.919392Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2-live-networks-db-1    | 2022-06-23T17:46:37.281079Z 0 [System] [MY-010229] [Server] Starting XA crash recovery...
```

Видим, что контейнер `nginx` не запустился.

Проверяем какие контейнеры запущены и обращаем внимание на то, что в этот раз порты никакие не используются:

```
$ docker ps
```

| CONTAINER ID | IMAGE               | COMMAND                  | CREATED       | STATUS       | PORTS | NAMES                 |
|--------------|---------------------|--------------------------|---------------|--------------|-------|-----------------------|
| 5dea4983319a | mysql:8.0           | "docker-entrypoint.s..." | 2 minutes ago | Up 2 minutes |       | 2-live-networks-db-1  |
| a76759d35789 | 2-live-networks_php | "docker-php-entrypoi..." | 2 minutes ago | Up 2 minutes |       | 2-live-networks-php-1 |

Проинспектируем network drivers:

```
$ docker network inspect none
[
  {
    "Name": "none",
    "Id": "2a3f49dae13719941a5eeb45ee8b037d3fc9d331fa94307356880c5e24a8e445",
    "Created": "2021-11-03T17:12:40.0687775Z",
    "Scope": "local",
    "Driver": "null",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": []
    },
    "Internal": false,
```



```

    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "5dea4983319a30c0ee6d60a787b0f14e00ad55c112e8934676eebe0ed9f9cfe9": {
            "Name": "2-live-networks-db-1",
            "EndpointID":
"4c686ea32adcb78fe939165027fbd46df4f255109fecc0513dd18ac578b38f4a",
            "MacAddress": "",
            "IPv4Address": "",
            "IPv6Address": ""
        },
        "a76759d35789864d8de8cf2b2f76b43968284fd18f676b293dc1f9b6c05ca473": {
            "Name": "2-live-networks-php-1",
            "EndpointID":
"d815d5f548eb9a0ea1feacfbe2e03331f0852c92d5d7bcf14126eb9e273fc29e",
            "MacAddress": "",
            "IPv4Address": "",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {}
}
]

```

Видим, что в такой конфигурации, когда контейнеры изолированы `nginx` не может запуститься.

Модифицируем наш файл

Docker-compose.yml:

```
version: '3'
```

```

services:
  nginx:
    image: nginx:1.17.8
    ports:
      - "8080:80"
    volumes:
      - ./code:/code
      -
./docker/nginx/site.conf:/etc/nginx/conf.d/site.conf
    networks:
      - test-network

  php:
    build:
      context: docker/php-fpm
    volumes:
      - ./code:/code
      - ./docker/php-
fpm/php.ini:/usr/local/etc/php/php.ini
    networks:
      - test-network

  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: 'base'
      MYSQL_USER: 'user'
      MYSQL_PASSWORD: '12345'
      MYSQL_ROOT_PASSWORD: 'root'
    volumes:

```

```

- ./docker/db:/var/lib/mysql
ports:
- "3306:3306"
networks:
- test-network

networks:
  test-network:
    driver: bridge

```

Добавляем в него описание network drivers `bridge`

Запускаем контейнеры `$ docker-compose up`

Проверяем количество запущенных контейнеров:

```
$ docker ps
```

| CONTAINER ID | IMAGE               | COMMAND                  | CREATED       | STATUS       | PORTS                   |
|--------------|---------------------|--------------------------|---------------|--------------|-------------------------|
| ad9def65581a | 2-live-networks_php | "docker-php-entrypoi..." | 2 minutes ago | Up 2 minutes | 9000/tcp                |
| 71d01f006bab | nginx:1.17.8        | "nginx -g 'daemon of..." | 2 minutes ago | Up 2 minutes | 0.0.0.0:8080->80/tcp    |
| 9f2da1d68ba7 | mysql:8.0           | "docker-entrypoint.s..." | 2 minutes ago | Up 2 minutes | 0.0.0.0:3306->3306/tcp, |

Должно отобразиться три запущенных контейнера.

Посмотрим наши сети:

```
$ docker network ls
```

| NETWORK ID   | NAME                         | DRIVER | SCOPE |
|--------------|------------------------------|--------|-------|
| bb0141995383 | 2-live-networks_default      | bridge | local |
| 79a9d5184f94 | 2-live-networks_test-network | bridge | local |
| 0507ad46874b | bridge                       | bridge | local |
| 3ceb0f16d293 | host                         | host   | local |
| 2a3f49dae137 | none                         | null   | local |

Видим нашу сеть `2-live-networks_test-network` с драйвером `bridge`.

Теперь проинспектируем нашу сеть:

```
$ docker network inspect 2-live-networks_test-network
[
  {
    "Name": "2-live-networks_test-network",
    "Id": "79a9d5184f94c77b8bd49362176cf7a0b902744753d988b9198c195591d84e69",
    "Created": "2022-06-23T17:59:02.9531594Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "71d01f006bab7dd6f86290d6a65b7a31e81a4e32d2098a0834b1de977910db24": {
        "Name": "2-live-networks-nginx-1",
        "EndpointID": "dc6b2535cc65eb37049295fb0453c25326c1f0c75db7727d8ad84e9b78d5dc25",
        "MacAddress": "02:42:ac:13:00:04",
```

```

        "IPv4Address": "172.19.0.4/16",
        "IPv6Address": ""
    },

    "9f2da1d68ba706498632cfba6feede3bbd26b381a68c0eae6d723a60e9c360bd": {
        "Name": "2-live-networks-db-1",
        "EndpointID":
        "b721e6b85cb4f1a26c6ff71b17b6ad43fe96e4b71560b2ede563aca625dc7300",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
    },

    "ad9def65581a24ab9dd6d95167de3861e2972d39d7f6e60016a57a09c116afb3": {
        "Name": "2-live-networks-php-1",
        "EndpointID":
        "f8aca9ca00d3715a599910d886f0b7c2c6944bca1427a50c657ffc59dcb196b6",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
    }
},
"Options": {},
"Labels": {
    "com.docker.compose.network": "test-network",
    "com.docker.compose.project": "2-live-networks",
    "com.docker.compose.version": "2.5.1"
}
}
]

```

Обратим внимание, что каждому контейнеру присвоены IP адреса.

Теперь добавим в секции `networks` описание IP адресов

```

networks:
  test-network:
    driver: bridge
    ipam:

```

```
config:
  - subnet: 192.168.110.0/24
```

Запуск контейнеров с новой конфигурацией нужно производить после выполнения команды `docker-compose down`

Мы же воспользуемся «агрессивном» способом с описанием последствий и способа решения возникшей проблемы.

Останавливаем наши контейнеры `Ctrl+C`.

```
$ docker network ls
```

| NETWORK ID   | NAME                         | DRIVER | SCOPE |
|--------------|------------------------------|--------|-------|
| bb0141995383 | 2-live-networks_default      | bridge | local |
| 113b81a2c1a9 | 2-live-networks_test-network | bridge | local |
| 0507ad46874b | bridge                       | bridge | local |
| 3ceb0f16d293 | host                         | host   | local |
| 2a3f49dae137 | none                         | null   | local |

Из списка сетей копируем имя нашей сети и вставляем ее в команду:

```
$ docker network rm 2-live-networks_test-network
2-live-networks_test-network
```

Проверяем, что сеть удалена:

```
$ docker network ls
```

| NETWORK ID   | NAME                    | DRIVER | SCOPE |
|--------------|-------------------------|--------|-------|
| bb0141995383 | 2-live-networks_default | bridge | local |
| 0507ad46874b | bridge                  | bridge | local |
| 3ceb0f16d293 | host                    | host   | local |
| 2a3f49dae137 | none                    | null   | local |

Попробуем запустить наши контейнеры:

```
$ docker-compose up
```

И видим ошибку:

```
[+] Running 4/1
- Network 2-live-networks_test-network Created 0.1s
- Container 2-live-networks-nginx-1 Created 0.0s
- Container 2-live-networks-db-1 Created 0.0s
- Container 2-live-networks-php-1 Created 0.0s
Attaching to 2-live-networks-db-1, 2-live-networks-nginx-1, 2-live-networks-php-1
Error response from daemon: network 113b81a2c1a9ba8fc41c78d50060e61ebc5f4fe3b9334f606091c774932f3f97 not found
```

Чтобы ее исправить запускаем ниши контейнеры с флагом

```
--force-recreate
```

Проверим список сетей и проинспектируем нашу сеть и убедимся, что применилась наша конфигурация.

```
$ docker network ls
```

| NETWORK ID   | NAME                         | DRIVER | SCOPE |
|--------------|------------------------------|--------|-------|
| bb0141995383 | 2-live-networks_default      | bridge | local |
| 2bc38ef75678 | 2-live-networks_test-network | bridge | local |
| 0507ad46874b | bridge                       | bridge | local |
| 3ceb0f16d293 | host                         | host   | local |
| 2a3f49dae137 | none                         | null   | local |

```
$ docker network inspect 2-live-networks_test-network
```

```
[
  {
    "Name": "2-live-networks_test-network",
    "Id": "2bc38ef756785c7c0b090677d39c628c8eb5efb37f50046e7cbf024284bc0cd1",
    "Created": "2022-06-23T18:28:35.1209514Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {

```

```

        "Subnet": "192.168.110.0/24"
    }
]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {
    "b40a0e59fcb6888628b85346eee609aae533782f70295118cca52232e87e9345": {
        "Name": "2-live-networks-php-1",
        "EndpointID":
        "b32865407c8d1bfcc89a3dd6474f65a022ffa08a74d44990b289545984864a01",
        "MacAddress": "02:42:c0:a8:6e:02",
        "IPv4Address": "192.168.110.2/24",
        "IPv6Address": ""
    },
    "b492b1c207120819e84c42a72316cc8e6c7897d80650a118716371b5a64c20a9": {
        "Name": "2-live-networks-db-1",
        "EndpointID":
        "00d6777bdf89f7f3726bf154cf1408e26497e30226be4d277e4829f6544c80c8",
        "MacAddress": "02:42:c0:a8:6e:03",
        "IPv4Address": "192.168.110.3/24",
        "IPv6Address": ""
    },
    "e67535d50a330ebc9c6aa7d313c852bb7185530a2cddf78edc3aad5ff2032c00": {
        "Name": "2-live-networks-nginx-1",
        "EndpointID":
        "6c86108a7bbd557f34d1e2c077db002a11c160dc8e543d42226686868a7679c5",
        "MacAddress": "02:42:c0:a8:6e:04",
        "IPv4Address": "192.168.110.4/24",
        "IPv6Address": ""
    }
}

```



```

    }
  },
  "Options": {},
  "Labels": {
    "com.docker.compose.network": "test-network",
    "com.docker.compose.project": "2-live-networks",
    "com.docker.compose.version": "2.5.1"
  }
}
]

```

## 9. Построение информационной безопасности контейнеров

Запускаем контейнеры командой:

```
docker run -p 127.0.0.1:3306:3306 --name mariadb -e
MARIADB_ROOT_PASSWORD=superpass -d mariadb
```

Дождемся загрузки и разворачивания контейнеров.

Проверяем, что контейнер запущен:

```
$ docker ps
```

| CONTAINER ID      | IMAGE                    | COMMAND                  | CREATED            |
|-------------------|--------------------------|--------------------------|--------------------|
| STATUS            | PORTS                    |                          | NAMES              |
| cd398b4dfe26      | mariadb                  | "docker-entrypoint.s..." | About a minute ago |
| Up About a minute | 127.0.0.1:3306->3306/tcp |                          | mariadb            |

Зайдем «внутрь» контейнера:

```
docker exec -ti mariadb sh
```

Смотрим из-под какого пользователя мы зашли:

```
# whoami
root
```

Выведем список всех пользователей:

```
# cat etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
mysql:x:999:999::/home/mysql:/bin/sh
```

Видим, что среди пользователей есть пользователь `mysql 999`

Выходим из `shell` командой `exit`.

Останавливаем контейнер и удаляем его

```
$ docker stop mariadb
mariadb
$ docker rm mariadb
mariadb
```

## Запускаем контейнер с флагом -u(user) 999

```
docker run -u 999 -p 127.0.0.1:3306:3306 --name mariadb -e  
MARIADB_ROOT_PASSWORD=superpass -d mariadb
```

Смотрим под каким пользователем теперь зашли и сможем ли мы перейти в каталог `root`?

```
$ whoami  
Mysql
```

```
$ cd /root/  
sh: 2: cd: can't cd to /root/
```

```
$ exit
```

## Останавливаем контейнер и удаляем его

```
$ docker stop mariadb  
mariadb  
$ docker rm mariadb  
mariadb
```

Теперь убедимся, что при стандартном запуске контейнера мы не сможем, например, удалить сетевой интерфейс.

```
docker run -p 127.0.0.1:3306:3306 --name mariadb -e  
MARIADB_ROOT_PASSWORD=superpass -d mariadb
```

```
docker exec -ti mariadb sh
```

```
# whoami  
root
```

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
115: eth0@if116: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
# ip link delete eth0
RTNETLINK answers: Operation not permitted
# exit
```

Запустим контейнер с ключом `--privileged` и убедимся, что теперь сможем удалить сетевой интерфейс:

```
docker run --privileged -p 127.0.0.1:3306:3306 --name
mariadb -e MARIADB_ROOT_PASSWORD=superpass -d mariadb
```

```
# whoami
root
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
117: eth0@if118: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

```
# ip link delete eth0

# ip link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000

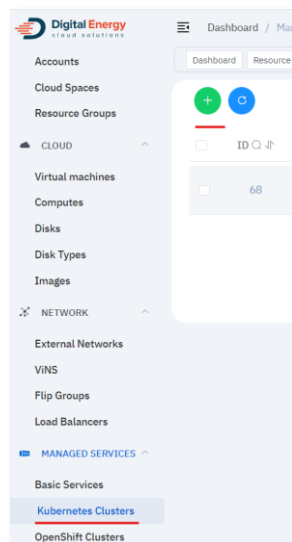
    link/ipip 0.0.0.0 brd 0.0.0.0

3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT
group default qlen 1000

    link/sit 0.0.0.0 brd 0.0.0.0

# exit
```

## 10.Создание кластера Kubernetes



Create a new K8s

① General    ② Master Node    ③ Worker Node    ④ Network

\* Name

\* Account

\* Resource Group

\* K8s Instance

\* Worker Group Name

Description

Next Step

Create a new K8s

General Master Node Worker Node Network

Master Nums

\* CPU

\* Memory in MB

\* Vdisk in GB

Previous step Next Step

Create a new K8s

General Master Node Worker Node Network

Worker Nums

\* CPU

\* Memory in MB

\* Vdisk in GB

Labels

Taints

Annotations

Previous step Next Step

Create a new K8s




General Master Node Worker Node Network

Internal Net./Vins

Ext. Networks

Previous step Confirm Creating

## 11.Подключение к кластеру

| General Master Workers |   |
|------------------------|---|
| Parameter              | Value   |
| Kuber Config           |    |
| ID                     | 68  |
| Name                   | k8s   |
| Account ID             | 97  |
| RG ID                  | 1411  |
| Status                 | ENABLED   |
| Tech.Status            | STARTED   |

Сохраняем скаченную конфигурацию в \$HOME/.kube/config

```
$ kubectl get services
```

| NAME       | TYPE      | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------|-----------|------------|-------------|---------|-----|
| kubernetes | ClusterIP | 10.96.0.1  | <none>      | 443/TCP | 12s |

## 12. Запуск контейнеров mariadb+adminer в kubernetes

Запускаем mariadb

```
$ kubectl run mariadb --image=mariadb --port 3306 --env  
MARIADB_ROOT_PASSWORD=superpass  
pod/mariadb created
```

Запускаем adminer

```
$ kubectl run adminer --image=adminer  
pod/adminer created
```

Открываем «межподовое» взаимодействие

```
$ kubectl expose pod mariadb  
service/mariadb exposed
```

```
$ kubectl expose pod adminer --port 8080 --  
type=NodePort  
service/adminer exposed
```

Проверим список запущенных подов

```
$ kubectl get pods
```

| NAME    | READY | STATUS  | RESTARTS | AGE   |
|---------|-------|---------|----------|-------|
| adminer | 1/1   | Running | 0        | 5m11s |

|         |     |         |   |       |
|---------|-----|---------|---|-------|
| mariadb | 1/1 | Running | 0 | 5m24s |
|---------|-----|---------|---|-------|

## Проверим список запущенных сервисов

```
$ kubectl get services
```

| NAME           | TYPE      | CLUSTER-IP    | EXTERNAL-IP | PORT(S)  |
|----------------|-----------|---------------|-------------|----------|
| AGE            |           |               |             |          |
| adminer        | NodePort  | 10.102.229.28 | <none>      |          |
| 8080:32225/TCP | 96s       |               |             |          |
| kubernetes     | ClusterIP | 10.96.0.1     | <none>      | 443/TCP  |
| 7m23s          |           |               |             |          |
| mariadb        | ClusterIP | 10.102.61.247 | <none>      | 3306/TCP |
| 2m6s           |           |               |             |          |

Воспользуемся возможностью локального проброса портов до нужного пода.

Выведем список подов с указанием пространства имен:

```
$ kubectl get pods --all-namespaces
```

| NAMESPACE   | NAME                                 | READY | STATUS  |
|-------------|--------------------------------------|-------|---------|
| RESTARTS    | AGE                                  |       |         |
| default     | adminer                              | 1/1   | Running |
| 0           | 26m                                  |       |         |
| default     | mariadb                              | 1/1   | Running |
| 0           | 26m                                  |       |         |
| kube-system | coredns-f9fd979d6-fq9js              | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | coredns-f9fd979d6-tmfpv              | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | etcd-s122-g264-c1                    | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | kube-apiserver-s122-g264-c1          | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | kube-controller-manager-s122-g264-c1 | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | kube-flannel-ds-8xmzp                | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | kube-flannel-ds-crrzw                | 1/1   | Running |
| 0           | 27m                                  |       |         |
| kube-system | kube-proxy-4vjlw                     | 1/1   | Running |
| 0           | 27m                                  |       |         |



|             |                             |     |         |
|-------------|-----------------------------|-----|---------|
| kube-system | kube-proxy-xcrgj            | 1/1 | Running |
| 0           | 27m                         |     |         |
| kube-system | kube-scheduler-s122-g264-c1 | 1/1 | Running |
| 0           | 27m                         |     |         |

NAMESPACE, в котором запущен под `adminer` имеет имя `default`.

Команда проброса портов будет выглядеть следующим образом:

```
$ kubectl -n default port-forward adminer 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Зайдем в браузере по адресу `http://localhost:8080/`

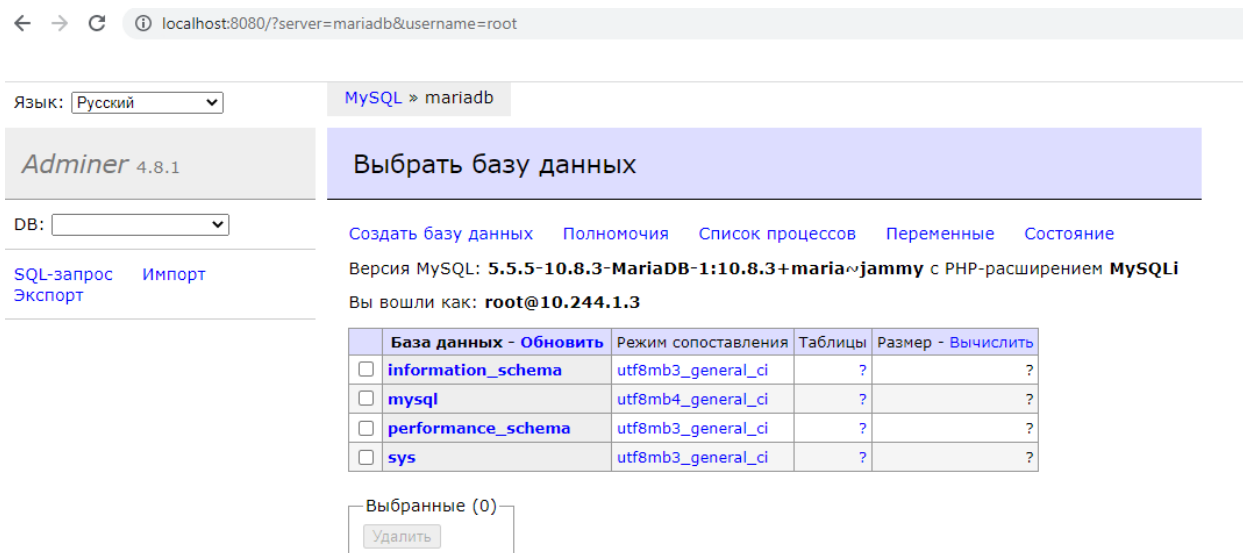
Проверяем работу, указывая параметры подключения

Сервер: `mariadb`

Имя пользователя: `root`

Пароль: `supperpass`

|                  |         |
|------------------|---------|
| Движок           | MySQL   |
| Сервер           | mariadb |
| Имя пользователя | root    |
| Пароль           | .....   |
| База данных      |         |



### 13. Развертывание инструментов DevOps/DevSecOps в DE k8s-managed-service

Используя заранее созданный кластер на платформе DE, скачайте kubeconfig. При выполнении задания используйте утилиту kubectl. При отправке команд, указывайте флаг kubeconfig для передачи токена и адреса подключения к кластеру DE k8s-managed-service

```
kubectl get nodes —kubeconfig='<путь-до-kubeconfig>'
```

#### I. Инсталляция инструмента MobSF

1) Создайте неймспейс для инструмента MobSF

```
kubectl create ns mobsf
```

2) Создайте deployment Mobsf

```
kubectl create deployment mobsf --image=opensecurity/mobile-security-framework-mobsf:v3.1.1 -n mobsf
```

(деплоймент позволяет отслеживать количество подов и сервисов, выполнять их обновление и модификацию)

3) Создайте службу для пода MobSF

```
kubectl expose deployment mobsf --port=8000 --type=NodePort -n mobsf
```

4) Если вы используете инетгресс-контроллер для обработки входящих соединений в кластере, узнайте его external-ip

```
kubectl get svc -n kube-system
```

5) Узнайте, какой порт на хосте был выделен службе MobSF

```
kubectl get svc -n mobsf
```

6) Используя браузер, перейдите по адресу (ip:port) и откройте веб-интерфейс MobSF

<http://172.26.0.3:31278>

## **II. Инсталляция инструмента Istio, плагинов, применение модифицирующего вебхука к инструменту MobSF для развертывания сервисной сетки.**

1) Создайте собственный неймспейс для Istio

```
kubectl create ns istio-system
```

2) Проинсталируйте helm (<https://helm.sh/docs/intro/install/>)

3) Добавьте репозиторий Istio в Helm

```
helm repo add istio https://istio-release.storage.googleapis.com/charts
```

4) Выполните апдейт репозиториев

```
helm repo update
```

5) Проинсталируйте базовые компоненты сервисной сетки Istio с помощью HELM

```
helm install istio-base istio/base -n istio-system
```

```
helm install istiod istio/istiod -n istio-system --wait
```

6) Проинсталируйте интегрированный в Istio дашборд Kiali через плагин

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.13/samples/addons/kiali.yaml
```

7) Модифицируйте манифест проинсталированной службы Kiali

```
kubectl edit svc kiali -n istio-system
```

учимся модифицировать манифест:

ищем в спецификации

type: ClusterIP

и меняем его на

type: NodePort

тем самым используя сеть хоста

сохраняем изменения, получаем уведомление, что service/kiali edited

7) Проинсталируйте интегрированный в Istio интегрированный инструмент по сбору метрик Prometheus через плагин

kubectl apply -f <https://raw.githubusercontent.com/istio/istio/release-1.13/samples/addons/prometheus.yaml>

8) Применяем модифицирующий вебхук к неймспесу с инструментом Mobsf

kubectl label namespace mobsf istio-injection=enabled

получаем уведомление, что namespace/mobsf labeled

9) Получаем информацию о поде с инструментом MobSF

kubectl get pods -n mobsf

10) Удаляем под с MobSf

kubectl delete pod mobsf-5db69649fc-5nl7t -n mobsf

(обратите внимание, что в вашем случае будет другое наименование пода, подставьте свое значения из вывода команды kubectl get pods -n mobsf)

поскольку мы с вами создавали deployment для mobsf, удаление пода вызовет ее переразвертывание (но уже с примененным модифицирующим вебхуком от Istio)

вызовите снова команду

kubectl get pods -n mobsf

и обратите внимание на вывод, в графе READY вы теперь видите "2/2", это означает, что в поде теперь находятся 2 контейнера - один это контейнер с mobsf, второй - это sidecar который внедрился Istio. Sidecar - это контейнер с прокси-сервером Envoy, который будет сообщать Istio о всех действиях контейнера)

11) Выполните команду и определите, по какому порту запущен дашборд kiali

kubectl get svc -n istio-system

Перейдите по адресу хоста, подставив обнаруженный порт службы kiali. Откроется интерфейс дашборда.

Открыв интерфейс дашборда kiali, перейдите в раздел "Graph", выберите namespace Mobssf в выпадающем списке "Select namespaces"

Во временном фильтре в правой верхней части экрана установите вместо "Last 1m" значение "Last 1d"

Перейдите в интерфейс MobSF и выполните имитацию полезной нагрузки (попереключайтесь между вкладками RECENT SCANS и главной страницей несколько раз)

Перейдите обратно в kiali и немного подождите, вы получите визуализацию активности внутри построенной вами сервисной сетки (см. приложенный скриншот "service-mesh-in-action.jpg").

### Результат успешного выполнения задания:

