

МЕТОДИЧЕСКАЯ РАБОТА

Контейнеризованные базы данных в Docker

MySQL • сети • volumes • Dockerfile • Docker Compose • резервное копирование

Группа	CA-24, 4 курс
Дисциплина / модуль	Системное администрирование. Контейнеризация
Формат	Лабораторная работа с пояснениями и заданиями
Ориентировочное время	90-120 минут
Основа	Docker Docs: Use containerized databases; материал дополнен

Об учебной среде: Учебные пароли в работе намеренно простые. Их нельзя использовать в реальных системах. Цель лабораторной - понять механизмы Docker, а не настроить production-сервер MySQL.

Версия методички: сентябрь 2026

1. Цель работы

В предыдущих лабораторных работах вы уже запускали контейнеры, использовали Docker Compose и собирали собственные образы. В этой работе эти навыки объединяются на примере базы данных MySQL. База данных - хороший пример stateful-сервиса: нам важно не только запустить процесс, но и сохранить данные, обеспечить сетевой доступ, выполнить начальную настройку, а затем корректно остановить и пересоздать контейнер без потери информации.

После выполнения работы студент должен уметь:

- запускать MySQL из Docker Official Image и читать журналы запуска;
- подключаться к СУБД из контейнера, с хостовой системы и из другого контейнера;
- объяснять разницу между портом контейнера и опубликованным портом хоста;
- создавать пользовательскую Docker-сеть и использовать встроенное разрешение имен;
- хранить файлы базы данных в named volume и проверять сохранность данных после пересоздания контейнера;
- создавать собственный образ MySQL с SQL-скриптом инициализации;
- описывать MySQL и phpMyAdmin в compose.yaml;
- использовать healthcheck и понимать, почему depends_on без проверки готовности недостаточно;
- создавать резервную копию базы данных и выполнять базовое восстановление;
- называть основные риски при контейнеризации СУБД.

2. Что потребуется

Компонент	Требование
Docker	Docker Engine + Compose plugin или Docker Desktop
Доступ в Интернет	Для загрузки образов mysql и phpmyadmin
Терминал	Bash, PowerShell или терминал Docker Desktop
Браузер	Для проверки phpMyAdmin
Свободные порты	3307 и 8080 на хостовой системе

```
docker version
docker compose version
```

Пояснение: Если обе команды выводят версии без ошибок, среда готова. Если команда `docker` работает только через `sudo`, добавляйте `sudo` перед командами этой методички либо настройте доступ пользователя к Docker отдельно.

3. Теория перед началом

3.1. Почему база данных отличается от обычного веб-контейнера

Контейнеры удобно пересоздавать. Для stateless-приложения это обычно не проблема: новый контейнер стартует из того же образа и получает состояние из внешних сервисов. База данных, наоборот, хранит изменяемое состояние - таблицы, индексы, журналы транзакций и служебные файлы. Если сохранить эти файлы только в writable layer контейнера, удаление контейнера удалит и его writable layer. Поэтому для СУБД почти всегда требуется отдельное постоянное хранилище.

Важно: Перезапуск контейнера и удаление контейнера - разные операции. После `docker restart` данные в `writable layer` остаются, потому что контейнер тот же. После `docker rm writable layer` удаляется. В лабораторной мы специально удалим контейнер, чтобы проверить работу `volume`.

3.2. Образ, контейнер и каталог данных

Объект	Назначение	Пример в работе
Image	Неизменяемый шаблон для создания контейнера	mysql:8.4
Container	Запущенный экземпляр образа	my-mysql
Writable layer	Изменяемый слой конкретного контейнера	временные файлы контейнера
Named volume	Хранилище, жизненный цикл которого отделен от контейнера	sa24-mysql-data
Docker network	Изолированная виртуальная сеть контейнеров	sa24-db-net

3.3. Почему в работе используется mysql:8.4, а не mysql:latest

Тег `latest` удобен для знакомства, но он может начать указывать на новую основную версию. Для лабораторной работы важна воспроизводимость: одинаковые команды должны давать одинаковый результат у всей группы. Поэтому далее используется ветка `mysql:8.4`. Это не означает, что тег `latest` запрещен; это означает, что в учебных и особенно в производственных конфигурациях версию образа лучше выбирать осознанно.

3.4. Переменные окружения официального образа MySQL

Переменная	Что делает
MYSQL_ROOT_PASSWORD	Задаёт пароль встроенной учётной записи <code>root</code> при первичной инициализации.
MYSQL_DATABASE	Создаёт указанную базу данных при первичной инициализации.
MYSQL_USER	Создаёт дополнительного пользователя. Используется вместе с <code>MYSQL_PASSWORD</code> .
MYSQL_PASSWORD	Пароль дополнительного пользователя.

Важно: Эти параметры инициализации не предназначены для изменения уже существующей БД. Если `/var/lib/mysql` содержит ранее созданную базу, `entrypoint` сохраняет существующие данные и не переинициализирует их.

3.5. Порты и сети

MySQL внутри контейнера слушает TCP-порт 3306. Этот порт существует в сетевом пространстве контейнера и сам по себе не обязан быть доступен с хоста. Публикация порта создается отдельно параметром `-p` или `ports` в `Compose`.

```
127.0.0.1:3307:3306
^^^^^^^^ ^^^^ ^^^^
IP хоста   |   |
порт хоста |   |
           |   |
           |   | порт контейнера
```

В этой методичке для локального доступа используется привязка к 127.0.0.1. Это значит, что опубликованный порт предназначен только для локальной машины. Если приложение работает в другом контейнере той же Docker-сети, публиковать порт базы данных на хост вообще не требуется: контейнеры обращаются друг к другу по имени и внутреннему порту.

4. Этап 1. Запуск контейнера MySQL

Сначала загрузите выбранный образ:

```
docker pull mysql:8.4
```

Запустите MySQL в фоновом режиме:

```
docker run --name my-mysql \
-e MYSQL_ROOT_PASSWORD=sa24-root-pass \
-e MYSQL_DATABASE=sa24db \
-e MYSQL_USER=sa24user \
-e MYSQL_PASSWORD=sa24-user-pass \
-d mysql:8.4
```

Примечание: В PowerShell символ переноса строки отличается от Bash. Если многострочная команда не выполняется, введите ее одной строкой. Смысл параметров от этого не меняется.

4.1. Разбор команды

Фрагмент	Назначение
--name my-mysql	Читаемое имя контейнера.
-e ...	Передает переменную окружения внутрь контейнера.
-d	Detached mode: контейнер работает в фоне.
mysql:8.4	Образ и выбранная ветка версии.

Проверьте состояние:

```
docker ps
```

Если контейнер не отображается, покажите в том числе остановленные контейнеры:

```
docker ps -a
```

Посмотрите журнал запуска MySQL:

```
docker logs my-mysql
```

Пояснение: На первом запуске MySQL требуется время на создание системных таблиц и настройку каталога данных. Строка о запуске контейнера в `docker ps` не всегда означает, что СУБД уже готова принимать соединения. Для диагностики сначала смотрите `docker logs`.

5. Этап 2. Работа внутри контейнера

Откройте интерактивную оболочку контейнера:

```
docker exec -it my-mysql bash
```

Обратите внимание: вы не запускаете новый контейнер. `docker exec` запускает дополнительный процесс внутри уже работающего контейнера `my-mysql`.

```
hostname
pwd
mysql --version
cat /etc/os-release
```

Запустите клиент MySQL:

```
mysql -u root -p
```

Введите пароль: sa24-root-pass. Затем выполните:

```
SHOW DATABASES;
SELECT VERSION();
SELECT CURRENT_USER();
USE sa24db;
SHOW TABLES;
```

Создадим небольшую таблицу, которая понадобится далее:

```
CREATE TABLE students (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  group_name VARCHAR(20) NOT NULL
);

INSERT INTO students (name, group_name) VALUES
('Ivan Ivanov', 'SA-24'),
('Petr Petrov', 'SA-24');

SELECT * FROM students;
```

Примечание: SQL здесь нужен только на базовом уровне. Главное в лабораторной - не синтаксис СУБД, а связь между процессом MySQL, контейнером, сетью и постоянным хранилищем.

Для выхода из клиента выполните `exit`;, затем еще раз `exit` для выхода из `bash` контейнера.

6. Этап 3. Подключение к базе данных с хоста

Текущий контейнер не имеет опубликованного порта. Убедитесь в этом:

```
docker port my-mysql
```

Команда не должна показывать сопоставление 3306 с портом хоста. Пересоздадим контейнер с публикацией порта. Данные этого первого контейнера можно удалить - постоянное хранилище мы добавим на следующем этапе.

```
docker rm -f my-mysql

docker run --name my-mysql \
  -p 127.0.0.1:3307:3306 \
  -e MYSQL_ROOT_PASSWORD=sa24-root-pass \
  -e MYSQL_DATABASE=sa24db \
  -e MYSQL_USER=sa24user \
  -e MYSQL_PASSWORD=sa24-user-pass \
  -d mysql:8.4

docker port my-mysql
docker ps
```

6.1. Что означает 3307:3306

Клиент на хостовой системе подключается к 127.0.0.1:3307. Docker перенаправляет соединение на порт 3306 внутри контейнера. Мы используем 3307 на хосте, чтобы не конфликтовать с MySQL, который может быть установлен локально и использовать 3306.

6.2. Проверка клиентом MySQL

Если на хосте установлен клиент mysql, выполните:

```
mysql -h 127.0.0.1 -P 3307 -u sa24user -p
```

Пароль: sa24-user-pass. После входа:

```
USE sa24db;
SHOW TABLES;
```

Примечание: После пересоздания без *volume* предыдущая таблица *students* отсутствует. Это ожидаемый результат: первый контейнер был удален вместе со своим *writable layer*.

6.3. Проверка через графический клиент

В DBeaver, MySQL Workbench или другом клиенте используйте следующие параметры:

Параметр	Значение
Host	127.0.0.1
Port	3307
Database	sa24db
User	sa24user
Password	sa24-user-pass

7. Этап 4. Связь двух контейнеров через Docker network

В реальном приложении база данных часто должна быть доступна веб-приложению, но не обязана быть опубликована на сетевых интерфейсах хоста. Для этого сервисы помещают в одну пользовательскую Docker-сеть.

```
docker rm -f my-mysql
docker network create sa24-db-net
```

Запустите MySQL без -p, но подключите его к созданной сети:

```
docker run --name my-mysql \
  --network sa24-db-net \
  -e MYSQL_ROOT_PASSWORD=sa24-root-pass \
  -e MYSQL_DATABASE=sa24db \
  -e MYSQL_USER=sa24user \
  -e MYSQL_PASSWORD=sa24-user-pass \
  -d mysql:8.4
```

Теперь запустите phpMyAdmin в той же сети:

```
docker run --name my-phpmyadmin \
  --network sa24-db-net \
  -p 127.0.0.1:8080:80 \
  -e PMA_HOST=my-mysql \
  -e PMA_PORT=3306 \
  -d phpmyadmin:latest
```

7.1. Почему PMA_HOST равен my-mysql

В пользовательской Docker-сети контейнеры получают встроенное разрешение имен. Имя контейнера my-mysql становится сетевым именем, которое другой контейнер этой сети может разрешить в IP-адрес. phpMyAdmin обращается к my-mysql:3306 напрямую внутри Docker-сети. Порт 3306 при этом не опубликован на хост.

```
docker network inspect sa24-db-net
```

7.2. Проверка через браузер

Откройте в браузере <http://localhost:8080>. Для входа используйте sa24user и sa24-user-pass. Убедитесь, что база sa24db отображается.

Важно: Не используйте адрес localhost:3306 между двумя контейнерами. Для процесса внутри phpMyAdmin localhost означает сам контейнер phpMyAdmin, а не контейнер MySQL.

8. Этап 5. Постоянное хранение данных в volume

Теперь решим главную проблему stateful-сервиса - отделим данные от жизненного цикла контейнера.

```
docker rm -f my-phpmyadmin my-mysql
docker volume create sa24-mysql-data
docker volume ls
```

Запустите MySQL с named volume. В официальном образе каталог данных MySQL - /var/lib/mysql.

```
docker run --name my-mysql \
  --network sa24-db-net \
  -e MYSQL_ROOT_PASSWORD=sa24-root-pass \
  -e MYSQL_DATABASE=sa24db \
  -e MYSQL_USER=sa24user \
  -e MYSQL_PASSWORD=sa24-user-pass \
  --mount type=volume,src=sa24-mysql-data,dst=/var/lib/mysql \
  -d mysql:8.4
```

Пояснение: В Docker существуют две популярные формы записи mounts: -v и --mount. Здесь используется --mount, потому что она более явная: видно тип, источник и точку назначения.

Проверьте подключение volume:

```
docker inspect my-mysql
```

В разделе Mounts найдите Type: volume, Name: sa24-mysql-data и Destination: /var/lib/mysql.

8.1. Создаем данные

```
docker exec my-mysql mysql \
  -u sa24user -psa24-user-pass \
  -D sa24db \
  -e "CREATE TABLE IF NOT EXISTS inventory (id INT PRIMARY KEY AUTO_INCREMENT, item VARCHAR(100)); INSERT INTO inventory (item) VALUES ('router'), ('switch'); SELECT * FROM inventory;"
```

Важно: Передача пароля прямо в командной строке удобна только для лабораторной. MySQL может вывести предупреждение о небезопасном использовании пароля в CLI. В production секреты не следует хранить в истории shell и compose.yaml в открытом виде.

8.2. Удаляем контейнер, но сохраняем volume

```
docker rm -f my-mysql
docker volume ls
```

Контейнера больше нет, но sa24-mysql-data должен остаться. Это и есть отделение данных от контейнера.

8.3. Создаем новый контейнер с тем же volume

```
docker run --name my-mysql \
  --network sa24-db-net \
  -e MYSQL_ROOT_PASSWORD=sa24-root-pass \
  -e MYSQL_DATABASE=sa24db \
  -e MYSQL_USER=sa24user \
  -e MYSQL_PASSWORD=sa24-user-pass \
  --mount type=volume,src=sa24-mysql-data,dst=/var/lib/mysql \
  -d mysql:8.4

docker exec my-mysql mysql -u sa24user -psa24-user-pass -D sa24db -e "SELECT * FROM inventory;"
```

Если вы видите router и switch, опыт выполнен правильно. Таблица хранится не в пересозданном контейнере, а в volume.

Пояснение: Переменные `MYSQL_DATABASE`, `MYSQL_USER` и другие переданы повторно для читаемости команды. Поскольку `volume` уже содержит инициализированную БД, официальный `entrypoint` не создает ее заново и не меняет существующие учетные данные.

9. Дополнение. Резервное копирование и восстановление

Volume повышает живучесть данных при пересоздании контейнера, но volume не является резервной копией. Ошибочное удаление таблицы, повреждение данных или удаление самого volume также приведет к потере информации. Поэтому администратор должен уметь создавать логические дампы.

9.1. Создание SQL-дампа

```
docker exec my-mysql sh -c 'exec mysqldump -uroot -p"$MYSQL_ROOT_PASSWORD" --databases sa24db'
> sa24db-backup.sql
```

Проверьте, что файл появился на хосте и не пустой:

```
ls -lh sa24db-backup.sql
head -n 20 sa24db-backup.sql
```

Примечание: Оператор `>` выполняется оболочкой хоста. Поэтому результат `mysqldump`, запущенного внутри контейнера, сохраняется в файл на хостовой машине.

9.2. Восстановление в отдельный контейнер

```
docker volume create sa24-mysql-restore

docker run --name my-mysql-restore \
  -e MYSQL_ROOT_PASSWORD=restore-root-pass \
  --mount type=volume,src=sa24-mysql-restore,dst=/var/lib/mysql \
  -d mysql:8.4
```

Дождитесь готовности MySQL. Для проверки можно несколько раз выполнить `docker logs my-mysql-restore`. Затем восстановите дамп:

```
docker exec -i my-mysql-restore sh -c 'exec mysql -uroot -p"$MYSQL_ROOT_PASSWORD" < sa24db-backup.sql'
```

```
docker exec my-mysql-restore mysql -uroot -prestore-root-pass -D sa24db -e "SELECT * FROM inventory;"
```

Пояснение: Восстановление выполнено в отдельный контейнер специально, чтобы не рисковать исходной базой. Такой подход удобен и для проверки резервных копий.

10. Этап 6. Собственный образ MySQL с инициализацией

Официальный образ MySQL умеет выполнять скрипты из каталога /docker-entrypoint-initdb.d при первичной инициализации пустого каталога данных. Используем это, чтобы получить образ, который автоматически создает таблицу.

10.1. Структура проекта

```
mysql-custom/
├── Dockerfile
├── scripts/
│   └── 01-init.sql
```

10.2. Dockerfile

```
# syntax=docker/dockerfile:1
FROM mysql:8.4

# Это не секрет: задаем только имя БД.
ENV MYSQL_DATABASE=sa24db

COPY ./scripts/ /docker-entrypoint-initdb.d/
```

Важно: Не записывайте `MYSQL_ROOT_PASSWORD` или `MYSQL_PASSWORD` в `Dockerfile` через `ENV`. Значение станет частью метаданных образа и будет распространяться вместе с ним.

10.3. SQL-скрипт 01-init.sql

```
CREATE TABLE IF NOT EXISTS sa24db.devices (
  id INT PRIMARY KEY AUTO_INCREMENT,
  hostname VARCHAR(100) NOT NULL,
  ip_address VARCHAR(45) NOT NULL
);

INSERT INTO sa24db.devices (hostname, ip_address) VALUES
('srv-docker-01', '10.10.10.10'),
('gw-lab-01', '10.10.10.1');
```

10.4. Сборка и запуск

```
cd mysql-custom
docker build -t sa24-mysql:1.0 .
docker volume create sa24-custom-data

docker run --name my-custom-mysql \
  -e MYSQL_ROOT_PASSWORD=sa24-root-pass \
  -e MYSQL_USER=sa24user \
  -e MYSQL_PASSWORD=sa24-user-pass \
  --mount type=volume,src=sa24-custom-data,dst=/var/lib/mysql \
  -d sa24-mysql:1.0

docker exec my-custom-mysql mysql -u sa24user -psa24-user-pass -D sa24db -e "SELECT * FROM devices;"
```

10.5. Критически важная особенность init-скриптов

Скрипты из `/docker-entrypoint-initdb.d` выполняются при первичной инициализации новой базы. Если `sa24-custom-data` уже содержит MySQL, удаление контейнера, пересборка образа и создание нового контейнера с тем же `volume` не заставят `01-init.sql` выполняться повторно. Это защищает существующую базу от случайной переинициализации.

```
docker rm -f my-custom-mysql
# Volume sa24-custom-data НЕ удаляем - база уже инициализирована.
```

Пояснение: Если во время лабораторной вы изменили `01-init.sql` и хотите проверить скрипт с нуля, используйте новый `volume` или удалите учебный `volume`. Никогда не переносите эту привычку на `production` без резервной копии.

11. Этап 7. MySQL и phpMyAdmin через Docker Compose

Вручную запускать два контейнера удобно для понимания механики. Для повторяемого проекта лучше описать сервисы декларативно в `compose.yaml`. Вы уже знакомы с Compose, поэтому здесь внимание будет на базе данных, `volume` и готовности сервиса.

11.1. Создайте каталог проекта

```
mkdir docker-db-compose
cd docker-db-compose
```

Создайте файл `.env`:

```
MYSQL_ROOT_PASSWORD=sa24-root-pass
MYSQL_DATABASE=sa24db
MYSQL_USER=sa24user
MYSQL_PASSWORD=sa24-user-pass
```

Важно: Файл `.env` удобен для лабораторной, но сам по себе не является защищенным хранилищем секретов. Его нельзя бездумно помещать в публичный `Git`-репозиторий. Для реальных систем применяют секрет-хранилища, `Docker secrets` или иные механизмы управления секретами.

11.2. Файл compose.yaml

```
services:
  db:
    image: mysql:8.4
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    volumes:
      - mysql-data:/var/lib/mysql
    healthcheck:
      test: ["CMD-SHELL", "mysqladmin ping -h 127.0.0.1 -uroot -p${MYSQL_ROOT_PASSWORD} --silent"]
      interval: 5s
      timeout: 3s
      retries: 10
      start_period: 20s

  phpmyadmin:
    image: phpmyadmin:latest
    restart: unless-stopped
    depends_on:
      db:
        condition: service_healthy
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
    ports:
      - "127.0.0.1:8080:80"

volumes:
  mysql-data:
```

11.3. Что здесь нового

- У db нет секции ports. База доступна phpMyAdmin по внутренней сети Compose, но не опубликована на хост.
- Compose автоматически создает общую сеть проекта. Имя сервиса db используется как DNS-имя.
- Named volume mysql-data хранит /var/lib/mysql вне жизненного цикла контейнера.
- healthcheck проверяет не только факт запуска процесса, но и способность MySQL отвечать.
- depends_on с condition: service_healthy задерживает запуск зависимого сервиса до успешной проверки здоровья базы.
- Знак \$\$ нужен, чтобы передать символ \$ внутрь команды контейнера, а не подставить переменную на этапе обработки Compose.

11.4. Проверка конфигурации и запуск

```
docker compose config
docker compose up -d
docker compose ps
```

В выводе docker compose ps дождитесь состояния healthy для db. Затем откройте <http://localhost:8080> и войдите под sa24user.

```
docker compose logs -f db
```

Примечание: Для выхода из режима просмотра логов используйте Ctrl+C. Контейнеры при этом не останавливаются, потому что они запущены с -d.

11.5. Создаем данные и проверяем сохранность

```
docker compose exec db mysql -u sa24user -psa24-user-pass -D sa24db -e "CREATE TABLE
compose_test (id INT PRIMARY KEY, value_text VARCHAR(50)); INSERT INTO compose_test VALUES (1,
'persistent data'); SELECT * FROM compose_test;"

docker compose down
docker compose up -d
```

После повторного запуска проверьте таблицу:

```
docker compose exec db mysql -u sa24user -psa24-user-pass -D sa24db -e "SELECT * FROM
compose_test;"
```

Пояснение: *docker compose down* удаляет контейнеры и сеть проекта, но *named volume* по умолчанию сохраняет. Поэтому данные переживают *down/up*.

11.6. Чем опасен параметр -v у docker compose down

Следующую команду сейчас выполнять не нужно - она приведена для сравнения:

```
docker compose down -v
```

Эта команда дополнительно удаляет *named volumes*, объявленные проектом. Для учебной среды это удобный способ начать с нуля. Для базы данных с ценными данными такая команда потенциально разрушительна.

Важно: *Перед удалением volume всегда задайте себе два вопроса: что именно в нем хранится и существует ли проверенная резервная копия?*

12. Небольшое задание на самостоятельное изменение

Не копируя готовое решение из предыдущего раздела, измените Compose-проект так, чтобы выполнить следующие требования:

- phpMyAdmin открывался на локальном порту 8081 вместо 8080;
- имя базы данных стало `admin_lab`;
- обычный пользователь назывался `adminuser`;
- после первого запуска в базе появилась таблица `hosts` с полями `id`, `hostname` и `role`;
- в `hosts` было не менее трех записей: `router`, `web-server`, `db-server`;
- после `docker compose down` и `docker compose up -d` данные сохранялись;
- порт MySQL не был опубликован на хост.

Примечание: Для автоматического создания таблицы можно использовать *init SQL*-скрипт, подключив его в `/docker-entrpoint-initdb.d`, либо собственный образ. Помните: *init* выполняется только для пустого каталога данных.

13. Что проверить перед сдачей

№	Проверка	Команда / способ
1	Контейнеры запущены	<code>docker ps</code> или <code>docker compose ps</code>
2	MySQL принимает соединения	<code>mysql</code> / <code>phpMyAdmin</code> / <code>healthcheck</code>
3	Контейнеры видят друг друга по имени	<code>phpMyAdmin</code> подключается к <code>db</code> или <code>my-mysql</code>
4	Данные хранятся в <i>volume</i>	<code>docker volume ls</code> + пересоздание контейнера
5	Init-скрипт создал таблицу	<code>SELECT</code> из созданной таблицы

6	Резервная копия создана	файл sa24db-backup.sql существует и не пуст
7	После down/up данные остаются	SELECT после повторного запуска Compose

14. Что приложить к отчету

В отчет не нужно вставлять скриншот каждой команды. Нужны доказательства ключевых результатов и краткие выводы.

- Титульная часть: ФИО, группа, название лабораторной работы.
- Скриншот или текстовый вывод `docker ps / docker compose ps` с работающими сервисами.
- Результат `SELECT` из таблицы, созданной вручную.
- Результат `SELECT` после удаления и пересоздания контейнера с тем же volume.
- Фрагмент `docker network inspect` или объяснение, почему phpMyAdmin подключается к `my-mysql/db` по имени.
- Содержимое собственного Dockerfile и SQL-скрипта инициализации.
- Содержимое итогового `compose.yaml`.
- Доказательство создания резервной копии и успешного восстановления либо проверка содержимого дампа.
- Ответы на контрольные вопросы раздела 16.
- Краткий вывод: 5-8 предложений о том, чем контейнеризация базы данных отличается от запуска stateless-контейнера.

15. Типичные ошибки и диагностика

Симптом	Вероятная причина	Что проверить
Container name is already in use	Контейнер с таким именем уже существует	<code>docker ps -a</code> ; удалить или переименовать контейнер
Bind for 127.0.0.1:3307 failed	Порт 3307 занят	выбрать другой хостовый порт, например 3308
phpMyAdmin не подключается сразу после запуска	MySQL еще инициализируется	<code>docker logs db</code> ; <code>healthcheck</code> ; дождаться <code>healthy</code>
Access denied for user	Неверный пароль/пользователь либо volume создан со старыми учетными данными	проверить переменные и понять, был ли volume пустым
Изменил <code>MYSQL_PASSWORD</code> , но старый пароль продолжает работать	База уже была инициализирована в volume	переменные <code>init</code> не меняют существующую БД
Изменил <code>01-init.sql</code> , но новая таблица не появилась	Используется старый volume	<code>init</code> -скрипты выполняются только при первичной инициализации
После <code>docker compose down</code> данные остались	Это нормальное поведение <code>named volume</code>	<code>docker volume ls</code> ; <code>down</code> без <code>-v</code> volume не удаляет
После <code>docker compose down -v</code> данные исчезли	Volume был удален вместе с проектом	восстанавливать из backup; не использовать <code>-v</code> без понимания последствий
localhost не соединяет два контейнера	localhost внутри контейнера указывает на него самого	использовать имя сервиса/контейнера в общей Docker-сети

16. Контрольные вопросы

1. Почему хранить файлы MySQL только в writable layer контейнера рискованно?
2. Чем docker restart отличается от удаления и повторного создания контейнера с точки зрения данных?
3. Что означает запись 127.0.0.1:3307:3306?
4. Почему контейнер phpMyAdmin может подключиться к MySQL без публикации порта 3306 на хост?
5. Что означает имя db в PMA_HOST: db при использовании Compose?
6. Чем named volume отличается от каталога внутри writable layer контейнера?
7. Сохраняется ли named volume после docker rm контейнера?
8. Что делает каталог /docker-entrypoint-initdb.d в официальном MySQL image?
9. Почему init SQL-скрипт может не выполниться после пересборки образа?
10. Почему хранить пароль в Dockerfile хуже, чем передавать его при запуске?
11. Зачем базе данных healthcheck?
12. Почему обычный depends_on не всегда означает, что СУБД уже готова принимать подключения?
13. Что произойдет с данными после docker compose down? А после docker compose down -v?
14. Почему volume не заменяет резервное копирование?
15. Для чего используется mysqldump в этой работе?
16. В каком случае имеет смысл публиковать порт MySQL на хост, а когда лучше этого не делать?
17. Почему для учебной работы выбран фиксированный тег mysql:8.4 вместо latest?

17. Краткий конспект

Понятие	Главная мысль
Stateful container	Сервис хранит изменяемое состояние, поэтому жизненный цикл данных нужно отделить от контейнера.
MySQL port 3306	Внутренний порт СУБД. Для связи контейнер-контейнер публикация на хост не обязательна.
Published port	Сопоставляет IP/порт хоста с портом контейнера.
User-defined network	Дает контейнерам изоляцию и разрешение имен.
Named volume	Постоянное хранилище, управляемое Docker и существующее отдельно от контейнера.
/var/lib/mysql	Каталог, в котором официальный MySQL image хранит данные.
/docker-entrypoint-initdb.d	Каталог init-скриптов, выполняемых при первичной инициализации пустой БД.
healthcheck	Проверяет готовность/здоровье сервиса, а не только факт существования процесса.
docker compose down	Удаляет контейнеры и сеть проекта, но обычно сохраняет named volumes.
docker compose down -v	Дополнительно удаляет named volumes проекта.

mysqldump	Создает логическую резервную копию SQL.
-----------	---

18. Дополнительные замечания по безопасности и эксплуатации

Лабораторная работа показывает базовые механизмы, но production-развертывание базы данных требует большего. Минимальный набор правил, который следует унести из этой работы:

- не публиковать порт СУБД наружу, если доступ нужен только другим контейнерам;
- не использовать root как учетную запись приложения;
- не хранить реальные пароли в Dockerfile, публичном Git и общедоступных compose-файлах;
- закреплять версии образов и планировать обновления, а не обновляться случайно через latest;
- делать резервные копии и периодически проверять восстановление;
- контролировать свободное место на хосте и размер volume;
- следить за логами, healthcheck и рестартами контейнера;
- перед удалением volume понимать последствия;
- для критичных систем отдельно проектировать отказоустойчивость, репликацию, мониторинг и управление секретами.

Примечание: Docker решает задачу упаковки и запуска процесса, но не отменяет обязанности администратора по резервному копированию, обновлениям, мониторингу и защите СУБД.

19. Очистка учебной среды

Выполняйте очистку только после того, как сохранили отчет и нужные файлы.

```
docker rm -f my-mysql my-mysql-restore my-custom-mysql my-phpmyadmin
docker network rm sa24-db-net
```

Учебные volumes удаляйте отдельно, осознанно:

```
docker volume rm sa24-mysql-data sa24-mysql-restore sa24-custom-data
```

Для Compose-проекта:

```
docker compose down -v
```

Важно: Если методика выполняется в общей или рабочей Docker-среде, не используйте массовые команды `docker system prune` / `docker volume prune` без разрешения преподавателя: они могут удалить ресурсы других проектов.

20. Источники и справочные материалы

- **Docker Docs - Use containerized databases:** <https://docs.docker.com/guides/databases/>
- **Docker Docs - Volumes:** <https://docs.docker.com/engine/storage/volumes/>
- **Docker Docs - Networking overview:** <https://docs.docker.com/engine/network/>
- **Docker Docs - Control startup and shutdown order in Compose:** <https://docs.docker.com/compose/how-tos/startup-order/>
- **Docker Hub - MySQL Official Image:** https://hub.docker.com/_/mysql

- **Docker Compose file reference - services:** <https://docs.docker.com/reference/compose-file/services/>

Методическая работа основана на официальной лабораторной Docker по контейнеризованным базам данных, но дополнена пояснениями по сетям, постоянному хранению, особенностям первичной инициализации MySQL, healthcheck, резервному копированию, диагностике и безопасному учебному использованию.