????? 3 — ?????????? ??????????

- Docker — Операции с проектами
- Docker — Volumes (Хранение данных)
- Docker — Images (Управление образами)

Docker — ???????? ? ?????????

Эта страница описывает базовые операции по управлению Docker-проектами на сервере: организацию директорий, работу с файлами проектов, изменение прав доступа и создание резервных копий. Материал рассчитан на начинающих и помогает понять, **как правильно хранить и обслуживать несколько Docker-проектов на одном сервере**, что особенно актуально для веб-студий и хостинга клиентских проектов.

???????? ????????? ????????? ?? ???????

На серверах с несколькими сайтами или сервисами Docker-проекты обычно размещаются в одной общей директории.

Пример структуры:

```
/opt/flamy_projects
```

Внутри неё располагаются отдельные проекты:

```
/opt/flamy_projects
│
├─ site1
│  ├─ docker-compose.yml
│  ├─ .env
│  ├─ Dockerfile
│  └─ src
│
├─ site2
│  ├─ docker-compose.yml
│  └─ src
│
└─ api_service
```

```
    ├─ docker-compose.yml
    └─ app
```

Каждый каталог — это **отдельный Docker Compose проект**.

??????? ? ??????????? ???????

Перед выполнением большинства Docker-команд необходимо перейти в директорию проекта.

```
cd /opt/flamy_projects
```

Просмотреть содержимое папки:

```
ls -la
```

????? ????????????

Эта команда помогает:

- убедиться, что вы работаете в нужной директории
- увидеть список проектов
- проверить наличие файлов конфигурации

????? docker-compose ??????

Иногда на сервере много проектов, и нужно найти все compose-файлы.

```
find /opt/flamy_projects -type f -name "docker-compose*.yml"
```

Эта команда покажет:

- все `docker-compose.yml`
- дополнительные файлы вроде `docker-compose.prod.yml`

Пример результата:

```
/opt/flamy_projects/site1/docker-compose.yml
/opt/flamy_projects/site2/docker-compose.yml
/opt/flamy_projects/api_service/docker-compose.yml
```

????? ???????

- аудит серверов
- поиск старых проектов
- диагностика конфигураций

???????? ?????? ???????

Перед запуском проекта полезно проверить содержимое директории.

```
ls -la
```

Пример:

```
docker-compose.yml
.env
Dockerfile
src/
logs/
```

??? ????? ?????????

- наличие `docker-compose.yml`
- наличие `.env` файла
- наличие исходного кода

Если файл `docker-compose.yml` отсутствует — проект запустить не получится.

????????? ????????? ??????

Иногда файлы проекта принадлежат неправильному пользователю. Это может происходить, если:

- проект копировали через `root`
- архив был распакован другим пользователем
- файлы были перенесены с другого сервера

Исправить владельца можно так:

```
sudo chown -R Crimson:Crimson /opt/flamy_projects
```

??? ???????? ???????

Часть	Значение
chown	изменить владельца
-R	рекурсивно для всех файлов
Crimson:Crimson	пользователь и группа

?????????? ????????? ????????????? ??????????

Иногда проще назначить владельцем текущего пользователя.

```
sudo chown -R $(id -u):$(id -g) /opt/flamy_projects
```

??? ?????? ?????????

Команда	Значение
`id -u`	ID текущего пользователя
`id -g`	ID текущей группы

Это позволяет быстро исправить проблемы с правами.

????????? ???? ???????

Иногда Docker или приложение не может читать файлы из-за неправильных прав.

Исправить можно так:

```
sudo chmod -R 755 /opt/flamy_projects
```

??? ???????? 755

Право	Значение
7	чтение, запись, выполнение
5	чтение и выполнение
5	чтение и выполнение

В итоге:

- владелец может изменять файлы
- остальные пользователи могут читать и выполнять

> **Важно:** Не стоит использовать `777`, так как это даёт полный доступ всем пользователям и может быть небезопасно.

?????????? ???????????? ????????

Регулярное резервное копирование помогает защититься от:

- ошибок обновления
- повреждения данных
- случайного удаления файлов

????????????? ???????

Создать архив можно с помощью `tar`.

```
tar -czf flamy_projects_backup_$(date +%Y%m%d).tar.gz /opt/flamy_projects
```

??? ?????? ???????

Часть	Значение
tar	инструмент архивирования
-c	создать архив
-z	сжать через gzip
-f	имя файла

Файл будет иметь имя вроде:

```
flamy_projects_backup_20260311.tar.gz
```

????????? ???????????? ????? rsync

`rsync` позволяет копировать файлы между серверами или папками.

```
rsync -avz /opt/flamy_projects/ backup_location/
```

??? ???????? ?????????

Параметр	Назначение
-a	архивный режим
-v	подробный вывод
-z	сжатие данных

?????? ???????????? ?? ?????? ??????

```
rsync -avz /opt/flamy_projects user@backup-server:/backup/docker_projects
```

Это часто используется для:

- ежедневных backup
- синхронизации серверов
- миграции проектов

???????? ??????? ???????

Иногда полезно узнать размер проекта.

```
du -sh /opt/flamy_projects/*
```

Пример вывода:

```
1.2G site1
850M site2
430M api_service
```

Это помогает определить:

- какие проекты занимают больше всего места
- где могут находиться большие логи или кэш

???????? ????? ???????

Некоторые проекты сохраняют логи внутри директории.

Например:

```
logs/
storage/logs
var/log
```

Посмотреть последние строки:

```
tail -n 100 logs/app.log
```

???????? ??????? ???????

Обычно работа с проектом выглядит так:

1 ??????? ? ?????????? ???????

```
cd /opt/flamy_projects/site1
```

2 ????????? ?????

```
ls -la
```

3 ????????? ???????????

```
docker compose up -d
```

4 ????????? ???????????

```
docker compose ps
```

5 ????????? ????

```
docker compose logs --tail=50
```

?????? ? ??????????? ?????????

На сервере может быть десятки проектов.

Чтобы понять, какие контейнеры запущены:

```
docker ps
```

Можно увидеть:

```
site1_nginx
site2_php
api_service_db
```

Это помогает быстро определить, какие проекты работают.

???????? ??????

??????? ??????? ? ????? ???????????

Это упрощает:

- резервное копирование
- администрирование
- поиск конфигураций

???????????? `.env` ?????

В `.env` удобно хранить:

- пароли
- порты
- настройки окружения

??????? ??????????? backup

Минимум раз в день для production-серверов.

??????? ???????

```
cd /opt/flamy_projects                    # перейти в директорию проектов
ls -la                                    # список файлов

find /opt/flamy_projects -name "docker-compose*.yml"  # найти compose файлы

sudo chown -R user:user /opt/flamy_projects  # изменить владельца
sudo chmod -R 755 /opt/flamy_projects        # изменить права

tar -czf backup_$(date +%Y%m%d).tar.gz /opt/flamy_projects  # архив проекта

rsync -avz /opt/flamy_projects/ backup/      # резервная копия

du -sh /opt/flamy_projects/*                 # размер проектов
```

Docker — Volumes (???????? ??????)

Эта страница объясняет, как Docker хранит постоянные данные контейнеров с помощью **Volumes**. Здесь разобраны основные типы хранения данных, команды управления volumes, а также практические примеры резервного копирования и восстановления данных. Материал рассчитан на начинающих и помогает понять, **где Docker хранит данные и как не потерять их при пересоздании контейнеров**.

??? ????? Docker Volume

По умолчанию контейнеры Docker **не сохраняют данные после удаления**. Если контейнер удалить и создать заново, все файлы внутри него исчезнут.

Чтобы сохранить данные между перезапусками контейнеров, используются **Volumes**.

Volume — это специальное хранилище данных Docker, которое:

- существует независимо от контейнера
- сохраняет данные между перезапусками
- может использоваться несколькими контейнерами

?????? volumes ?????

Volumes используются для хранения **постоянных данных**, например:

- базы данных
- загруженные пользователями файлы
- кэш приложений
- логи

Пример:

```
mysql container
|
```

```
└── /var/lib/mysql  → volume
```

Если контейнер удалить и создать заново, данные в volume **сохранятся**.

??? Docker ?????? volumes

По умолчанию volumes находятся в системе Docker:

```
/var/lib/docker/volumes/
```

Каждый volume имеет свою директорию.

Пример структуры:

```
/var/lib/docker/volumes/mysql_data/_data
```

???? ???????? ?????? ? Docker

Docker поддерживает несколько способов хранения данных.

---

1. Docker Volume

Это основной и рекомендуемый способ хранения данных.

Пример в `docker-compose.yml`:

```
services:
  mysql:
    image: mysql:8
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
```

```
  mysql_data:
```

Преимущества:

- безопасное хранение данных
- управляется Docker
- легко делать backup

---

2. Bind Mount

Bind mount подключает **обычную папку сервера** внутрь контейнера.

Пример:

```
services:
  app:
    volumes:
      - ./src:/var/www/html
```

Это означает:

```
сервер ./src  → контейнер /var/www/html
```

Используется для:

- разработки
- подключения исходного кода
- конфигурационных файлов

??????? ????? Volume ? Bind Mount

Тип	Описание
Volume	управляется Docker
Bind mount	обычная папка сервера

Обычно используется:

- **Volumes → данные приложений**
- **Bind mounts → код проекта**

???????? volumes

?????? volumes

```
docker volume ls
```

Пример вывода:

```
DRIVER    VOLUME NAME
local     site1_mysql_data
local     redis_cache
```

?????????? ? volume

```
docker volume inspect <volume_name>
```

Пример:

```
docker volume inspect site1_mysql_data
```

Результат покажет:

- путь к volume
- драйвер
- настройки

???????? volume

Создать volume можно вручную:

```
docker volume create mysql_data
```

После этого volume можно использовать в контейнерах.

???????? volumes

??????? ???? volume

```
docker volume rm <volume>
```

Пример:

```
docker volume rm mysql_data
```

??????? ????????????? volumes

```
docker volume prune -f
```

Команда удаляет volumes, которые **не используются контейнерами**.

> **Важно:** Перед удалением нужно убедиться, что volume не содержит важных данных.

????????????? volumes ? Docker Compose

Пример конфигурации:

```
services:
  mysql:
    image: mysql:8
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
```

```
  mysql_data:
```

Это создаёт volume `mysql_data`, который будет хранить данные базы.

???????? volumes ??????????

Можно посмотреть volumes конкретного контейнера:

```
docker inspect <container>
```

Пример:

```
docker inspect site1_mysql
```

В выводе нужно найти блок:

```
Mounts
```

Он показывает подключённые volumes.

????????? ???????????? volume

Очень важная операция — backup данных.

Backup volume

```
docker run --rm \
  -v mysql_data:/data \
  -v $(pwd):/backup \
  alpine tar czf /backup/mysql_backup.tar.gz /data
```

Эта команда:

1. подключает volume
2. создаёт архив
3. сохраняет backup в текущей директории

?????????????? volume

Чтобы восстановить данные:

```
docker run --rm \
  -v mysql_data:/data \
  -v $(pwd):/backup \
  alpine tar xzf /backup/mysql_backup.tar.gz -C /
```

После этого данные снова появятся в volume.

???????? ?????? ??? ?????? ? volumes

?????? ?????? ????? ???????? ???????????

Если данные хранятся **внутри контейнера**, а не в volume, они будут потеряны.

Правильная практика:

```
volumes:
  - mysql_data:/var/lib/mysql
```

???????? volume ?????? ? ????????

Команда:

```
docker compose down -v
```

удаляет:

- контейнеры
- сети
- volumes

Это приведёт к **потере данных**.

?????????????? bind mount ??? ???? ??????

Не рекомендуется хранить базы данных через bind mount.

Лучше использовать:

```
volumes:
  - mysql_data:/var/lib/mysql
```

???????? volumes ???????

Показать volumes Compose проекта:

```
docker compose config --volumes
```

Это помогает понять, какие volumes использует проект.

??????? ???????

```
docker volume ls                    # список volumes
docker volume inspect <volume>      # информация о volume
docker volume create <volume>       # создать volume
docker volume rm <volume>           # удалить volume
```

```
docker volume prune -f             # удалить неиспользуемые volumes

docker inspect <container>         # посмотреть volumes контейнера
```

Docker — Images (?????????? ????????)

Эта страница объясняет, что такое Docker-образы (Images), как они создаются, хранятся и управляются. Здесь разобраны основные команды для просмотра, загрузки, удаления и очистки образов. Материал рассчитан на начинающих и помогает понять, **как работают Docker-образы и как управлять ими на сервере**.

??? ????? Docker Image

Docker Image — это **шаблон контейнера**, из которого создаются запущенные контейнеры.

Проще говоря:

```
Docker Image → контейнер
```

Пример:

```
nginx:1.25  → контейнер nginx
mysql:8     → контейнер mysql
node:18     → контейнер node
```

Image содержит:

- операционную систему
- зависимости
- приложение
- конфигурацию запуска

Контейнер создаётся на основе образа и запускается как отдельный процесс.

??? ???????? Docker Images

Docker Images состоят из **слоёв (layers)**.

Пример:

```
Base OS
↓
System packages
↓
Application dependencies
↓
Application code
```

Каждый слой кэшируется. Это позволяет Docker **ускорять сборку образов**.

???????? ???????

???????? ?????? ???????

```
docker images
```

Пример вывода:

```
REPOSITORY   TAG      IMAGE ID       SIZE
nginx        1.25     a1b2c3d4e5     142MB
mysql        8        f6g7h8i9j0     520MB
node         18       k1l2m3n4o5     1.2GB
```

??? ???????? ?????? ???????

Поле	Описание
REPOSITORY	имя образа
TAG	версия образа
IMAGE ID	уникальный ID
SIZE	размер образа

???????? ?????? ?? registry

Docker может скачивать готовые образы из Docker Hub или другого registry.

```
docker pull <image>
```

Пример:

```
docker pull nginx:1.25
```

После этого образ появится в списке `docker images`.

???? ???????

Docker использует **теги** для обозначения версии образа.

Пример:

```
nginx:latest
nginx:1.25
node:18
```

?????? ?? ????? ???????????? `latest`

Тег `latest` может измениться в любой момент.

Лучше использовать фиксированную версию:

```
nginx:1.25
```

Это делает окружение более предсказуемым.

???????? Docker Images

??????? ?????

```
docker rmi <image>
```

Пример:

```
docker rmi nginx:1.25
```

??????? ????? ?? ID

```
docker rmi <image_id>
```

Пример:

```
docker rmi a1b2c3d4e5
```

??????? ?????????????? ??????

```
docker image prune -f
```

Удаляются **dangling images** — образы, которые больше не используются.

?????? ??????? ?????? ???????

Если на сервере накопилось много старых образов:

```
docker system prune -a
```

Эта команда удалит:

- остановленные контейнеры
- неиспользуемые образы

- сети
- build cache

> **Важно:** Использовать эту команду нужно осторожно, особенно на production.

?????? ??????

Образы можно создавать самостоятельно через `Dockerfile`.

Пример сборки:

```
docker build -t my_app .
```

Где:

Параметр	Значение
-t	тег образа
my_app	имя образа
.	текущая директория

???????? ???????? ??????

Можно посмотреть, из каких слоёв состоит образ:

```
docker history <image>
```

Пример:

```
docker history nginx
```

Это помогает понять:

- какие команды выполнялись при сборке
- размер каждого слоя

?????????????? ?????? ??? ??????? ???????????

Контейнер можно запустить напрямую:

```
docker run nginx
```

Но чаще используется Docker Compose.

Пример:

```
services:
  nginx:
    image: nginx:1.25
```

????? ???????

Можно искать образы на Docker Hub:

```
docker search nginx
```

Пример результата:

```
NAME              DESCRIPTION
nginx             Official build of Nginx
nginx-unit        Nginx Unit runtime
```

Лучше использовать **официальные образы**.

???????? ???????? ? ????????

??????? ????? ??????? ?? ???????

Проверить:

```
docker images
```

Удалить старые:

```
docker image prune -f
```

????? ?? ???????????

Если используется:

```
nginx:latest
```

Docker может использовать старую версию.

Решение:

```
docker pull nginx
```

??????? ??????? ?????? ??????

Некоторые образы могут занимать несколько гигабайт.

Проверить:

```
docker images
```

При необходимости удалить неиспользуемые.

??????? ???????

```
docker images                    # список образов
docker pull <image>              # скачать образ
```

```
docker search <image>             # поиск образа

docker rmi <image>                # удалить образ
docker image prune -f             # удалить неиспользуемые образы

docker build -t <name> .          # собрать образ
docker history <image>            # история слоев образа
```