????? 1 — ?????? ?????? ? Docker

- Compose — Управление сервисами
- Docker — Отладка контейнеров
- Docker Compose — Структура docker-compose.yml

Compose — ?????????? ?????????

Эта страница объясняет базовые команды `docker compose`, которые используются для запуска, остановки, пересборки и перезапуска сервисов в проекте. Материал рассчитан на начинающих: здесь разобрано, что делает каждая команда, в каких случаях её использовать и на что обратить внимание, чтобы случайно не сломать рабочее окружение.

??? ????? Docker Compose

`Docker Compose` — это инструмент для управления несколькими контейнерами как единым приложением. Обычно в проекте есть файл `docker-compose.yml`, где описано:

- какие сервисы нужно запускать
- какие образы использовать
- какие порты пробрасывать
- какие папки подключать как volumes
- какие переменные окружения передавать контейнерам

Например, один проект может состоять из:

- `nginx`
- `php`
- `mysql`
- `redis`

Вместо того чтобы запускать каждый контейнер отдельно длинными командами `docker run`, Compose позволяет поднять всё одной командой.

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

Обычно работа с Compose ведётся из директории проекта, где лежит `docker-compose.yml`.

Пример:

```
cd /opt/flamy_projects/project_name
```

Проверить, что файл действительно есть в текущей папке:

```
ls -la
```

Если нужно найти compose-файлы во всей папке проектов:

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

> **Примечание:** Большинство команд `docker compose` нужно выполнять именно в той директории, где находится `docker-compose.yml`. Иначе Docker не поймёт, с каким проектом работать.

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

Чаще всего цикл работы выглядит так:

1. перейти в папку проекта
2. запустить сервисы
3. проверить, что контейнеры работают
4. посмотреть логи при ошибках
5. при необходимости пересобрать или перезапустить сервисы

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

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

```
docker compose up -d
```

Эта команда:

- читает `docker-compose.yml`
- создаёт нужные контейнеры
- запускает их
- оставляет терминал свободным для дальнейшей работы

Флаг `-d` означает **detached mode**, то есть запуск в фоне.

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

Подходит почти для любого обычного старта проекта:

- после перезагрузки сервера

- после первого клонирования проекта
- после остановки контейнеров
- после мелких изменений конфигурации

??????

```
cd /opt/flamy_projects/site1
docker compose up -d
```

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

Если образа ещё нет локально, Docker попытается:

- скачать его из реестра
- либо собрать локально, если это описано в конфигурации

> **Для новичков:** `up` не всегда означает просто «включить». Эта команда может не только запускать, но и создавать контейнеры заново, если это требуется по конфигурации.

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

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

```
docker compose down
```

Команда останавливает и удаляет:

- контейнеры проекта
- созданные Compose-сети

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

Подходит, если нужно:

- полностью выключить проект
- освободить ресурсы
- заново поднять окружение
- сбросить сетевые проблемы внутри проекта

??? ??????? **?? ???????**

По умолчанию `docker compose down` **не удаляет volumes**. Это важно, потому что именно в volumes часто лежат данные:

- база данных
- загруженные файлы
- кэш приложения
- постоянные данные сервисов

> **Важно:** Для новичков это одна из самых полезных вещей: обычный `down` не стирает данные сам по себе. Но всё равно всегда нужно понимать, где именно проект хранит данные — в volumes, bind mounts или внутри контейнера.

??????

```
docker compose down
```

После этого проект можно снова поднять:

```
docker compose up -d
```

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

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

```
docker compose up -d --force-recreate
```

Эта команда заставляет Docker пересоздать контейнеры, даже если он считает, что ничего не поменялось.

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

Полезно, если:

- контейнер ведёт себя странно
- изменились переменные окружения
- конфигурация применена не так, как ожидалось
- нужен «чистый» перезапуск контейнера без удаления volumes

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

- контейнеры будут пересозданы

- volumes останутся на месте
- данные в volumes не пропадут

> **Пояснение:** Контейнер и volume — это не одно и то же. Контейнер — это «запущенная оболочка приложения». Volume — это место, где могут лежать постоянные данные. Поэтому пересоздание контейнера не обязательно означает потерю данных.

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

Пересборка нужна тогда, когда меняется не только запуск контейнера, а именно его образ. Обычно это происходит, если вы изменили:

- `Dockerfile`
- системные пакеты
- зависимости приложения
- шаги сборки

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

```
docker compose build --no-cache
```

Docker при сборке старается использовать кэш, чтобы не выполнять одинаковые шаги повторно. Это ускоряет работу, но иногда мешает, если нужен действительно «чистый» rebuild.

Флаг `--no-cache` говорит: **не использовать ранее сохранённые слои**, а собрать всё заново.

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

Подходит, если:

- изменения в Dockerfile «не подхватываются»
- обновились зависимости
- есть подозрение на устаревший build cache
- нужно убедиться, что образ собран полностью с нуля

??????

```
docker compose build --no-cache
docker compose up -d
```

> **Примечание:** Эта операция может занять заметно больше времени, чем обычная сборка.

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

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

Команда делает два действия сразу:

1. при необходимости пересобирает образ
2. запускает контейнеры

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

Это один из самых частых сценариев после изменения:

- `Dockerfile`
- зависимостей
- конфигурации сборки

??????

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

??? ?????????? ?? `build --no-cache`

- `docker compose up -d --build` может использовать кэш
- `docker compose build --no-cache` собирает всё полностью заново

Если нужно просто обновить контейнер после обычных изменений — чаще хватает:

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

Если есть сомнения в корректности кэша — используйте:

```
docker compose build --no-cache
docker compose up -d
```

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

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

```
docker compose pull
```

Команда скачивает новые версии образов, которые указаны в `docker-compose.yml`.

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

Подходит, если проект использует готовые образы из registry, например:

- `nginx:latest`
- `redis:7`
- `mysql:8`

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

`docker compose pull` **только скачивает образы**, но **не перезапускает контейнеры автоматически**.

Чтобы применить обновление, обычно выполняют:

```
docker compose pull && docker compose up -d
```

??????

```
cd /opt/flamy_projects/site1
docker compose pull
docker compose up -d
```

> **Важно:** На production-серверах обновление образов нужно делать осторожно. Новая версия образа может изменить поведение приложения или сломать совместимость.

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

Иногда контейнер не нужно пересобирать или пересоздавать — достаточно просто перезапустить процесс.

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

```
docker compose restart <service>
```

Пример:

```
docker compose restart nginx
```

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

Подходит, если:

- сервис завис
- нужно перечитать конфигурацию
- произошёл временный сбой
- после некоторых изменений достаточно обычного рестарта

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

```
docker compose restart
```

Перезапускает все сервисы, описанные в compose-файле.

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

Подходит, если:

- нужно быстро перезапустить весь проект
- нет уверенности, какой именно сервис работает некорректно
- нужно «освежить» всё окружение без удаления контейнеров

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

```
docker compose stop <service> && docker compose start <service>
```

Пример:

```
docker compose stop php && docker compose start php
```

????? ??? ?????, ???? ???? `restart`

Иногда удобнее разделить действия на две части:

- сначала корректно остановить контейнер

- затем отдельно запустить снова

Это бывает полезно при диагностике или когда нужно убедиться, что сервис полностью остановился.

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

```
docker compose up -d <service>
```

Пример:

```
docker compose up -d nginx
```

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

Полезно, если:

- нужно поднять только один сервис
- вы тестируете конкретный контейнер
- не хотите трогать всё окружение

> **Примечание:** Если сервис зависит от других, Compose может поднять и их тоже.

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

?????????? ???????? compose-??????

```
docker compose config
```

Эта команда показывает итоговую конфигурацию после обработки:

- `docker-compose.yml`
- override-файлов
- переменных окружения
- подстановок из `.env`

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

Очень полезно перед запуском, если:

- не подставляются env-переменные
- есть сомнения в корректности конфигурации
- нужно понять, что Docker реально «видит»

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

Иногда в YAML всё выглядит правильно, но итоговая конфигурация получается другой. `docker compose config` помогает увидеть конечный результат до запуска.

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

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

```
docker compose up -d
```

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

```
docker compose pull && docker compose up -d
```

????? ????????? Dockerfile ??? ????????????

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

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

```
docker compose build --no-cache
docker compose up -d
```

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

```
docker compose restart nginx
```

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

```
docker compose down
```

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

????????: ????? ?????? ????????? ??????

Используйте:

```
docker compose up -d
```

????????: ????????? Dockerfile

Используйте:

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

????????: ????? ????? ???? ???????, ????? ?????? rebuild

Используйте:

```
docker compose build --no-cache
docker compose up -d
```

????????: ????? ??????? ?????? ?????? ?? registry

Используйте:

```
docker compose pull && docker compose up -d
```

????????: ?????? ?????? ????? ????????????

Используйте:

```
docker compose restart <service>
```

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

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

Если выполнить `docker compose up -d` не там, где лежит `docker-compose.yml`, можно получить ошибку о том, что конфигурационный файл не найден.

???????? ????? `restart`, `up`, `build` ? `down`

Кратко:

- `restart` — просто перезапуск уже существующих контейнеров
- `up` — поднять проект
- `up --build` — поднять проект с пересборкой
- `down` — остановить и удалить контейнеры проекта

????????, ??? `pull` ??? ??????? ?????????? ??????????

`docker compose pull` только скачивает новый образ. Чтобы контейнеры начали работать на новой версии, нужен ещё запуск:

```
docker compose up -d
```

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

```
docker compose up -d                    # Запуск проекта в фоне
docker compose down                     # Остановка и удаление контейнеров проекта
docker compose up -d --force-recreate   # Пересоздать контейнеры принудительно
docker compose build --no-cache         # Полная пересборка без кэша
docker compose up -d --build            # Пересборка и запуск
docker compose pull                     # Скачать свежие образы
docker compose restart <service>        # Перезапуск одного сервиса
docker compose restart                  # Перезапуск всех сервисов
docker compose stop <service> && docker compose start <service>  # Остановить и запустить
сервис отдельно
docker compose up -d <service>          # Запуск одного сервиса
docker compose config                   # Проверка итоговой конфигурации
```

Docker — ??????? ???????????

Эта страница объясняет основные команды Docker, которые используются для диагностики и отладки контейнеров. Здесь разобраны способы посмотреть запущенные контейнеры, проверить их состояние, изучить логи, подключиться внутрь контейнера и выполнить команды для анализа проблемы. Материал рассчитан на начинающих и помогает быстро понять, **что происходит внутри контейнера**, если приложение работает некорректно.

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

На практике команды из этой страницы используются в следующих ситуациях:

- контейнер **не запускается**
- сервис **падает после старта**
- приложение **возвращает ошибки**
- сервис **не отвечает**
- контейнер **постоянно перезапускается**
- нужно проверить **что происходит внутри контейнера**

Типичный порядок диагностики:

1. проверить список контейнеров
2. проверить статус контейнера
3. посмотреть логи
4. зайти внутрь контейнера
5. проверить процессы или выполнить команды

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

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

```
docker compose ps
```

Команда показывает контейнеры, которые относятся к текущему `docker-compose` проекту.

Пример вывода:

```
NAME                IMAGE          STATUS        PORTS
project_nginx       nginx:latest   Up 2 hours    0.0.0.0:80->80/tcp
project_php         php:8.2-fpm    Up 2 hours
```

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

- имя контейнера
- используемый образ
- статус контейнера
- проброшенные порты

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

Это первая команда, которую стоит выполнить, если нужно проверить:

- запущен ли сервис
- работает ли контейнер
- не упал ли он

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

```
docker ps
```

Эта команда показывает **все запущенные контейнеры в системе**, независимо от проекта.

Пример:

```
CONTAINER ID   IMAGE         STATUS         PORTS
ab12cd34       nginx         Up 2 hours     80/tcp
cd34ef56       redis         Up 3 hours     6379/tcp
```

??? ??????????? ?? `docker compose ps`

Команда	Показывает
`docker compose ps`	контейнеры текущего проекта
`docker ps`	все контейнеры на сервере

????????? ??? ??????????? (??????? ?????????????)

```
docker ps -a
```

Показывает:

- работающие контейнеры
- остановленные контейнеры
- контейнеры с ошибками

Пример статуса:

```
Exited (1) 5 minutes ago
```

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

Контейнер **запустился, но приложение внутри завершилось с ошибкой**.

В таком случае следующим шагом нужно посмотреть логи.

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

```
docker ps -a --format "table {{.ID}}\t{{.Names}}\t{{.Status}}\t{{.Ports}}"
```

Этот формат удобен, если нужно быстро увидеть:

- ID контейнера
- имя
- статус
- порты

Пример вывода:

```
CONTAINER ID   NAMES         STATUS          PORTS
3acb1234       project_php   Up 10 minutes
7d8e5678       project_nginx Up 10 minutes   80/tcp
```

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

Логи — это основной источник информации при отладке.

Именно здесь можно увидеть:

- ошибки приложения
- ошибки запуска
- проблемы подключения к базе
- ошибки конфигурации

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

```
docker compose logs <service>
```

Пример:

```
docker compose logs nginx
```

Вывод покажет весь лог контейнера.

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

```
docker compose logs -f <service>
```

Флаг `-f` означает **follow** — следить за логом в реальном времени.

Пример:

```
docker compose logs -f php
```

Это удобно, если:

- вы только что перезапустили сервис
- нужно увидеть новые ошибки

Выход из режима просмотра:

```
CTRL + C
```

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

```
docker compose logs --tail=100 <service>
```

Показывает только последние 100 строк.

Пример:

```
docker compose logs --tail=100 nginx
```

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

Если контейнер работает давно и лог очень большой.

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

```
docker compose logs --timestamps <service>
```

Пример:

```
docker compose logs --timestamps php
```

Вывод будет выглядеть примерно так:

```
2026-03-11T12:32:10 app started
2026-03-11T12:32:11 connected to database
```

Это удобно для анализа событий во времени.

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

Иногда для диагностики нужно **зайти внутрь контейнера**, чтобы:

- посмотреть файлы
- проверить конфигурацию
- выполнить команды
- проверить сетевые подключения

????????????? ????? shell

```
docker compose exec <service> sh
```

Пример:

```
docker compose exec php sh
```

После выполнения команды вы окажетесь внутри контейнера.

????????????? bash

Если в контейнере установлен `bash`, можно использовать:

```
docker compose exec <service> bash
```

Пример:

```
docker compose exec php bash
```

> **Важно:** Не во всех контейнерах есть `bash`. Многие минимальные образы используют только `sh`.

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

После входа можно выполнять обычные команды Linux:

```
ls
cd
cat
ps
top
```

Например:

```
ls /var/www/html
```

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

```
docker compose top <service>
```

Пример:

```
docker compose top php
```

Команда показывает список процессов внутри контейнера.

Пример вывода:

```
UID     PID     CMD
root    123     php-fpm
```

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

Полезно, если:

- приложение зависло
- нужно проверить, запущен ли процесс
- нужно убедиться, что сервис действительно работает

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

Можно выполнить команду **без входа в shell**.

Формат:

```
docker compose exec <service> <command>
```

Пример:

```
docker compose exec php php -v
```

Вывод:

```
PHP 8.2.4 (cli)
```

??? ???????

Проверить установленные пакеты:

```
docker compose exec php composer --version
```

Проверить Node.js:

```
docker compose exec node node -v
```

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

Иногда нужно забрать файл из контейнера.

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

```
docker cp container:/path/file .
```

Пример:

```
docker cp project_php:/var/log/php.log .
```

Файл будет скопирован в текущую папку.

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

```
docker cp file container:/path/
```

Пример:

```
docker cp config.php project_php:/var/www/config.php
```

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

Если сервис не работает, обычно делают следующее:

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

```
docker compose ps
```

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

```
docker compose logs --tail=100
```

3 ???? ?????? ?? ???????? — ????????????? ??????

```
docker compose exec php sh
```

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

```
docker compose top php
```

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

Например:

```
curl localhost
```

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

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

В большинстве случаев ошибка уже записана в лог.

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

В Compose:

- **service** — описание контейнера в `docker-compose.yml`
- **container** — фактически запущенный экземпляр

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

Если статус:

```
Exited
```

контейнер уже остановился, и нужно смотреть логи.

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

```
docker compose ps                          # контейнеры текущего проекта
docker ps                                  # все запущенные контейнеры
docker ps -a                               # все контейнеры включая остановленные

docker compose logs <service>              # логи сервиса
docker compose logs -f <service>           # логи в реальном времени
docker compose logs --tail=100 <service>   # последние строки логов
docker compose logs --timestamps <service> # логи с временными метками

docker compose exec <service> sh           # войти в контейнер
docker compose exec <service> bash         # войти через bash
docker compose exec <service> <command>    # выполнить команду

docker compose top <service>               # процессы контейнера

docker cp container:/path/file .           # копировать из контейнера
docker cp file container:/path/            # копировать в контейнер
```

Docker Compose — ????????? docker-compose.yml

Эта страница объясняет структуру файла `docker-compose.yml` и основные параметры, используемые при описании сервисов Docker. Здесь разобраны ключевые разделы конфигурации, такие как `services`, `volumes`, `networks`, `environment`, `ports` и другие. Материал рассчитан на начинающих и помогает понять, **как правильно описывать инфраструктуру приложения в Docker Compose**.

??? ????? docker-compose.yml

`docker-compose.yml` — это файл конфигурации, который описывает:

- какие контейнеры нужно запустить
- какие образы использовать
- какие порты открыть
- какие volumes подключить
- какие сети создать

Compose позволяет запускать всё приложение **одной командой**:

```
docker compose up -d
```

??????? ????????? docker-compose.yml

Пример минимального файла:

```
version: "3.9"

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

```
    ports:
      - "8080:80"
```

Этот файл создаёт контейнер nginx и открывает порт `8080`.

???????? ???????? docker-compose.yml

Типичная структура файла выглядит так:

```
services:
volumes:
networks:
```

?????? services

Раздел `services` описывает контейнеры проекта.

Пример:

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

  php:
    image: php:8.2-fpm
```

В этом примере создаются два сервиса:

- nginx
- php

Каждый сервис становится отдельным контейнером.

???????? image

Определяет Docker-образ, который будет использоваться.

Пример:

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

Docker скачает образ `nginx:1.25`, если он отсутствует.

???????? build

Используется, если образ нужно **собрать локально из Dockerfile**.

Пример:

```
services:
  app:
    build: .
```

Это означает:

```
собрать образ из Dockerfile текущей директории
```

Можно указать путь:

```
services:
  app:
    build: ./app
```

???????? ports

Используется для проброса портов.

Пример:

```
ports:
  - "8080:80"
```

Это означает:

```
сервер:8080 → контейнер:80
```

После запуска приложение будет доступно:

```
http://server:8080
```

???????? volumes

Подключает volumes или папки сервера.

Пример:

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

Это означает:

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

????????????? Docker Volumes

Пример с volume:

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

```
volumes:
  mysql_data:
```

Volume будет хранить данные базы.

???????? environment

Передаёт переменные окружения контейнеру.

Пример:

```
environment:
  MYSQL_ROOT_PASSWORD: secret
  MYSQL_DATABASE: app_db
```

Контейнер получит переменные:

```
MYSQL_ROOT_PASSWORD
MYSQL_DATABASE
```

????????????? .env ?????

Переменные можно хранить в `.env`.

Пример:

```
DB_PASSWORD=secret
APP_ENV=production
```

И использовать в Compose:

```
environment:
  DB_PASSWORD: ${DB_PASSWORD}
```

???????? depends_on

Определяет зависимости между сервисами.

Пример:

```
services:
  php:
    depends_on:
      - mysql
```

Это означает:

```
сначала запускается mysql
затем php
```

> **Важно:** `depends_on` не гарантирует, что сервис полностью готов к работе.

???????? restart

Определяет политику перезапуска контейнера.

Пример:

```
restart: always
```

Возможные значения:

Значение	Поведение
no	не перезапускать
always	всегда перезапускать
on-failure	только при ошибке
unless-stopped	перезапускать, пока не остановлен вручную

???????? networks

Определяет сети Docker.

Пример:

```
services:
  app:
    networks:
      - backend

networks:
  backend:
```

Контейнер будет подключён к сети `backend`.

?????? ???????? docker-compose.yml

Пример простого веб-приложения:

```
services:

  nginx:
    image: nginx:1.25
    ports:
      - "80:80"
    volumes:
      - ./src:/var/www/html

  php:
    image: php:8.2-fpm
    volumes:
      - ./src:/var/www/html
    depends_on:
      - mysql

  mysql:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: root
```

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

volumes:
  mysql_data:
```

Этот файл запускает:

- nginx
- php
- mysql

???????? docker-compose.yml

Перед запуском полезно проверить конфигурацию:

```
docker compose config
```

Команда покажет итоговую конфигурацию после обработки переменных.

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

Запуск всех сервисов:

```
docker compose up -d
```

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

```
docker compose down
```

???????? ?????? ? docker-compose.yml

?????? YAML

Файл YAML чувствителен к отступам.

Неправильно:

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

Правильно:

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

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

Если порт уже используется:

```
Bind for 0.0.0.0:80 failed
```

Нужно изменить порт.

?????????? volumes

Если данные не вынесены в volumes, они могут потеряться при пересоздании контейнера.

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

```
services     → контейнеры проекта
image        → используемый образ
build        → сборка образа
ports        → проброс портов
volumes      → подключение хранилища
environment  → переменные окружения
depends_on   → зависимости сервисов
restart      → политика перезапуска
networks     → сети контейнеров
```