????? 2 — ?????????? ? ??????????

- Docker — Мониторинг и использование ресурсов
- Docker — Работа с сетями (Networks)
- Docker — Logs (Работа с логами)
- Docker — Cleanup (Очистка и обслуживание)

Docker — ?????????? ? ????????????? ????????

Эта страница объясняет, как отслеживать использование ресурсов Docker и как управлять дисковым пространством, занятым контейнерами, образами и volumes. Здесь разобраны команды для просмотра нагрузки на CPU и память, анализа использования диска, а также безопасной очистки неиспользуемых ресурсов. Материал ориентирован на начинающих и помогает понять, **почему Docker может занимать много места на диске или ресурсов системы** и как это контролировать.

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

Docker активно использует ресурсы системы:

- CPU
- оперативную память (RAM)
- дисковое пространство
- сетевые ресурсы

Со временем на сервере могут накапливаться:

- старые контейнеры
- неиспользуемые образы
- volumes с данными
- временные слои сборки

Если их не очищать, Docker может занять **десятки или даже сотни гигабайт дискового пространства**.

Поэтому регулярный мониторинг и очистка — важная часть администрирования Docker.

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

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

```
docker stats
```

Эта команда показывает статистику всех работающих контейнеров.

Пример вывода:

```
CONTAINER ID   NAME            CPU %   MEM USAGE / LIMIT   MEM %   NET I/O
a12bc34d       project_php     2.5%    120MiB / 2GiB       5.8%    1.2MB / 500KB
b45de67f       project_nginx   0.2%    15MiB / 2GiB        0.7%    200KB / 100KB
```

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

Поле	Описание
CPU %	загрузка процессора контейнером
MEM USAGE	используемая память
MEM %	процент от доступной памяти
NET I/O	сетевой трафик

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

Полезно, если:

- сервер начал тормозить
- приложение потребляет слишком много ресурсов
- нужно найти контейнер с высокой нагрузкой

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

```
docker stats --no-stream <container>
```

Флаг `--no-stream` выводит статистику **один раз**, без постоянного обновления.

Пример:

```
docker stats --no-stream project_php
```

Это удобно, если нужно быстро проверить текущую нагрузку.

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

Иногда нагрузку создаёт конкретный процесс.

Можно посмотреть процессы внутри контейнера:

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

Пример:

```
docker compose top php
```

Вывод покажет процессы, которые выполняются внутри контейнера.

Это помогает определить, что именно создаёт нагрузку:

- веб-сервер
- PHP-процесс
- worker
- cron-задача

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

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

```
docker system df
```

Пример вывода:

```
TYPE            TOTAL     ACTIVE    SIZE
Images          12        4         3.2GB
Containers      8         4         120MB
Local Volumes   6         5         5.6GB
Build Cache     -         -         2.1GB
```

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

Тип	Описание
Images	Docker-образы
Containers	контейнеры
Volumes	постоянные данные
Build Cache	кэш сборки Docker

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

Эта команда помогает понять:

- почему Docker занимает много места
- какие ресурсы занимают больше всего диска

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

```
docker system df -v
```

Флаг `-v` (verbose) показывает более подробную информацию:

- какие именно образы занимают место
- какие контейнеры используют volumes
- какие слои можно удалить

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

Со временем Docker оставляет множество временных объектов:

- остановленные контейнеры
- старые образы
- временные сети
- build cache

Их можно безопасно удалить.

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

```
docker image prune -f
```

Удаляет **dangling images** — образы, которые больше не используются контейнерами.

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

Если после сборок осталось много старых образов.

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

```
docker system prune -f
```

Удаляет:

- остановленные контейнеры
- неиспользуемые сети
- dangling images
- build cache

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

Эта команда **не удаляет volumes**.

То есть:

- базы данных
- файлы проекта
- persistent storage

останутся на месте.

??????? volumes

```
docker volume prune -f
```

Удаляет **неиспользуемые volumes**.

?????????

Volumes могут содержать:

- базы данных
- пользовательские файлы
- кэш приложения

Поэтому перед удалением нужно убедиться, что volume действительно не используется.

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

```
docker network prune -f
```

Удаляет неиспользуемые Docker-сети.

Эта команда безопасна, потому что:

- сети удаляются только если они не используются контейнерами

???????? ???????? Docker Engine

Иногда полезно посмотреть информацию о самой Docker-системе.

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

```
docker info | grep -i memory
```

Показывает информацию о памяти, доступной Docker.

???????? CPU

```
docker info | grep -i cpu
```

Показывает количество доступных CPU.

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

Некоторые контейнеры могут иметь ограничения по памяти.

Проверить это можно так:

```
docker inspect <container> | grep -i memory
```

Пример:

```
docker inspect project_php | grep -i memory
```

Эта команда покажет:

- лимит памяти
- настройки использования RAM

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

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

Самая частая причина — накопившиеся:

- старые образы
- build cache
- остановленные контейнеры

Решение:

```
docker system prune -f
```

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

Проверить использование:

```
docker system df
```

Затем очистить ненужные данные.

?????????? ??????????? ??????? ????? CPU

Найти контейнер:

```
docker stats
```

После этого:

- проверить логи
- посмотреть процессы внутри контейнера

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

Возможные причины:

- утечка памяти
- неправильная конфигурация приложения
- слишком большие кэши

Проверить:

```
docker stats
```

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

На серверах разработки полезно периодически выполнять:

```
docker system prune -f
```

Если используется активная сборка образов:

```
docker builder prune -f
```

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

Если сервер начал тормозить:

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

```
docker stats
```

---

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

```
docker system df
```

---

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

```
docker system prune -f
```

---

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

```
docker system df
```

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

```
docker stats                          # нагрузка контейнеров в реальном времени
docker stats --no-stream <container> # разовая статистика

docker compose top <service>          # процессы внутри контейнера

docker system df                      # использование диска Docker
docker system df -v                   # подробная информация

docker image prune -f                 # удалить неиспользуемые образы
docker system prune -f                # очистка контейнеров, сетей и cache
docker volume prune -f                # удалить неиспользуемые volumes
docker network prune -f               # удалить неиспользуемые сети

docker info | grep -i memory          # информация о памяти
docker info | grep -i cpu             # информация о CPU
docker inspect <container> | grep -i memory   # лимиты контейнера
```

Docker — ?????? ? ?????? (Networks)

Эта страница объясняет, как Docker управляет сетями и как контейнеры взаимодействуют друг с другом. Здесь разобраны основные команды для просмотра сетей, диагностики сетевых проблем и проверки портов сервисов. Материал ориентирован на начинающих и помогает понять, **как контейнеры находят друг друга внутри Docker-проекта и как сервисы становятся доступными снаружи**.

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

По умолчанию каждый контейнер подключается к одной или нескольким Docker-сетям. Сеть позволяет контейнерам:

- общаться друг с другом по имени сервиса
- передавать данные между сервисами
- изолировать проекты друг от друга

Например, в одном проекте могут работать контейнеры:

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

Все они находятся в одной сети и могут обращаться друг к другу по имени сервиса:

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

Пример подключения к базе внутри контейнера:

```
mysql:3306
```

Где `mysql` — имя сервиса из `docker-compose.yml`.

?????? Docker-?????

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

```
docker network ls
```

Пример вывода:

```
NETWORK ID     NAME               DRIVER
9a8b7c6d5e     bridge             bridge
8f7e6d5c4b     host               host
7e6d5c4b3a     project_default    bridge
```

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

Сеть	Назначение
bridge	стандартная сеть Docker
host	контейнер использует сеть хоста
project_default	сеть Docker Compose проекта

???? Docker Compose

Когда запускается проект через:

```
docker compose up -d
```

Docker автоматически создаёт сеть:

```
projectname_default
```

Например:

```
site1_default
```

Все сервисы из `docker-compose.yml` автоматически подключаются к этой сети.

Это позволяет контейнерам обращаться друг к другу **по имени сервиса**.

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

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

```
docker network inspect <network_name>
```

Пример:

```
docker network inspect site1_default
```

Эта команда показывает:

- подключенные контейнеры
- IP-адреса контейнеров
- настройки сети

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

Внутри результата можно увидеть блок:

```
"Containers": {
  "php_container": {
    "Name": "site1_php",
    "IPv4Address": "172.20.0.3"
  },
  "mysql_container": {
    "Name": "site1_mysql",
    "IPv4Address": "172.20.0.4"
  }
}
```

Это означает, что контейнеры подключены к сети и могут взаимодействовать.

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

Контейнер может быть доступен:

- **внутри Docker-сети**
- **снаружи сервера**

Для доступа извне используются **проброшенные порты**.

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

```
docker compose port <service> <port>
```

Пример:

```
docker compose port nginx 80
```

Результат:

```
0.0.0.0:8080
```

Это означает, что контейнерный порт `80` доступен на сервере через порт `8080`.

???????? ?????? ????? docker ps

Можно также проверить порты так:

```
docker ps
```

Пример вывода:

```
CONTAINER ID   IMAGE      PORTS
ab12cd34       nginx      0.0.0.0:8080->80/tcp
```

Это означает:

```
сервер:8080 -> контейнер:80
```

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

Иногда контейнер нужно подключить к существующей сети.

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

```
docker network connect <network> <container>
```

Пример:

```
docker network connect site1_default redis_container
```

После этого контейнер сможет взаимодействовать с сервисами сети.

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

```
docker network disconnect <network> <container>
```

Пример:

```
docker network disconnect site1_default redis_container
```

Контейнер больше не сможет общаться с сервисами этой сети.

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

Если сервисы не могут взаимодействовать, нужно проверить соединение.

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

```
docker compose exec php sh
```

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

Например, проверить базу данных:

```
ping mysql
```

или

```
nc -zv mysql 3306
```

Если соединение работает — сеть настроена правильно.

???????? DNS ?????? Docker

Docker автоматически создаёт DNS внутри сети.

Это позволяет обращаться к сервисам **по имени**.

Например:

```
mysql
redis
```

```
php
nginx
```

????????? DNS

Внутри контейнера:

```
nslookup mysql
```

или

```
ping mysql
```

Если DNS работает, контейнер получит IP.

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

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

Возможные причины:

- контейнеры находятся в разных сетях
- неправильное имя сервиса
- сервис не запущен

Проверить:

```
docker network inspect <network>
```

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

В Docker Compose нужно использовать **имя сервиса**, а не имя контейнера.

Пример:

```
mysql
```

а не:

```
project_mysql_1
```

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

Проверить `docker-compose.yml`.

Пример правильного проброса:

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

Это означает:

```
сервер:8080 -> контейнер:80
```

?????????? ???????, ?? ???? ?? ???????????

Проверить:

```
docker ps
```

Если порт не указан — он не проброшен наружу.

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

Если сервис не работает:

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

```
docker compose ps
```

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

```
docker network ls
```

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

```
docker network inspect <network>
```

4 ????????? DNS ?????? ???????????

```
docker compose exec php ping mysql
```

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

```
docker network ls                       # список сетей
docker network inspect <network>        # информация о сети

docker compose port <service> <port>    # проверить проброс порта

docker network connect <network> <container>     # подключить контейнер
docker network disconnect <network> <container>  # отключить контейнер

docker ps                               # посмотреть проброшенные порты
```

Docker — Logs (?????? ? ??????)

Эта страница объясняет, как Docker хранит и обрабатывает логи контейнеров. Здесь разобраны команды для просмотра логов, способы ограничения их размера, настройка log rotation и рекомендации по работе с логами в production. Материал рассчитан на начинающих и помогает понять, **где хранятся Docker-логи и как предотвратить переполнение диска из-за их роста**.

??? ????? Docker Logs

Каждый контейнер записывает стандартные потоки вывода:

- **stdout** — обычные сообщения
- **stderr** — ошибки

Docker автоматически сохраняет эти данные как **логи контейнера**.

Пример логов:

```
[INFO] Server started
[INFO] Connection established
[ERROR] Database connection failed
```

Логи помогают:

- диагностировать ошибки
- отслеживать работу приложения
- анализировать поведение сервисов

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

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

```
docker logs <container>
```

Пример:

```
docker logs nginx_container
```

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

```
docker logs --tail=100 <container>
```

Пример:

```
docker logs --tail=100 nginx_container
```

Это покажет последние 100 строк логов.

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

```
docker logs -f <container>
```

Пример:

```
docker logs -f nginx_container
```

Флаг `-f` означает **follow** — поток логов в реальном времени.

Выход из режима просмотра:

```
CTRL + C
```

???? Docker Compose

Если используется Docker Compose, можно смотреть логи сервисов.

???? ???????

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

Пример:

```
docker compose logs nginx
```

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

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

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

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

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

По умолчанию Docker использует драйвер логирования:

```
json-file
```

Файлы логов находятся в директории:

```
/var/lib/docker/containers/
```

Пример пути:

```
/var/lib/docker/containers/<container_id>/<container_id>-json.log
```

Эти файлы могут со временем **значительно увеличиваться**.

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

Если диск сервера заполняется, одной из причин могут быть логи контейнеров.

Проверить размер можно так:

```
du -sh /var/lib/docker/containers/*
```

Это покажет размер логов каждого контейнера.

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

Чтобы логи не занимали весь диск, можно настроить **log rotation**.

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

Пример конфигурации:

```
services:
  app:
    image: my_app
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

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

Параметр	Назначение

max-size	максимальный размер файла логов
max-file	количество файлов

Пример выше означает:

```
максимум 3 файла по 10 MB
```

То есть максимум:

```
30 MB логов
```

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

Ограничение логов можно задать для всего Docker.

Файл конфигурации:

```
/etc/docker/daemon.json
```

Пример:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

После изменения нужно перезапустить Docker:

```
sudo systemctl restart docker
```

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

Иногда логи могут стать очень большими.

Можно очистить файл логов:

```
truncate -s 0 /var/lib/docker/containers/<container_id>/<container_id>-json.log
```

> **Важно:** Очищать логи нужно осторожно и только при необходимости.

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

Можно посмотреть настройки логирования контейнера:

```
docker inspect <container>
```

В выводе нужно найти раздел:

```
LogConfig
```

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

Docker поддерживает разные драйверы логов.

Драйвер	Назначение
json-file	стандартный
syslog	системный лог
journald	systemd
fluentd	централизованный лог
gelf	Graylog

Для production часто используют централизованные системы логирования.

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

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

Если log rotation не настроен, файлы логов могут вырасти до нескольких гигабайт.

Решение:

- настроить `max-size`
- настроить `max-file`

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

Некоторые приложения пишут слишком подробные логи.

Решение:

- уменьшить уровень логирования
- использовать `INFO` вместо `DEBUG`

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

Иногда полезно посмотреть логи нескольких сервисов:

```
docker compose logs
```

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

Если сервис не работает:

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

```
docker compose ps
```

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

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

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

```
docker compose logs -f
```

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

```
docker logs <container>                # логи контейнера
docker logs -f <container>             # логи в реальном времени
docker logs --tail=100 <container>     # последние строки

docker compose logs                    # логи всех сервисов
docker compose logs -f <service>       # логи сервиса

du -sh /var/lib/docker/containers/*    # размер логов

truncate -s 0 logfile                  # очистить лог
```

Docker — Cleanup (??????? ? ????????????)

Эта страница описывает команды и практики очистки Docker-окружения. Со временем на сервере накапливаются остановленные контейнеры, старые образы, build-cache и неиспользуемые volumes. Здесь разобраны способы безопасной очистки этих ресурсов и рекомендации по регулярному обслуживанию Docker-системы.

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

При работе Docker постоянно создаёт временные и устаревшие ресурсы:

- остановленные контейнеры
- старые образы
- build cache после сборок
- неиспользуемые сети
- volumes с данными

Если их не удалять, они могут занять **десятки гигабайт диска**.

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

Перед очисткой полезно проверить, сколько места занимает Docker.

```
docker system df
```

Пример вывода:

```
TYPE            TOTAL     ACTIVE    SIZE
Images          12        5         3.4GB
Containers      8         4         150MB
Local Volumes   6         4         8.2GB
Build Cache     -         -         2.1GB
```

Это показывает:

- сколько образов хранится
- сколько контейнеров существует
- сколько места занимают volumes
- размер build cache

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

Контейнеры, которые были остановлены, продолжают существовать в системе.

Удалить их можно так:

```
docker container prune
```

Docker попросит подтверждение.

Чтобы выполнить без подтверждения:

```
docker container prune -f
```

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

Если контейнеры пересобираются, старые образы остаются в системе.

Удалить их можно так:

```
docker image prune
```

Удаляются **dangling images** — образы без тегов.

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

```
docker image prune -a
```

Эта команда удаляет все образы, которые не используются контейнерами.

> **Важно:** После удаления Docker при следующем запуске контейнера может заново скачать образ.

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

Volumes могут занимать много места, особенно если в них хранятся базы данных.

Удалить неиспользуемые volumes:

```
docker volume prune
```

Без подтверждения:

```
docker volume prune -f
```

> **Важно:** Удаляются только volumes, не подключённые к контейнерам.

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

Иногда остаются неиспользуемые Docker-сети.

Удалить их можно так:

```
docker network prune
```

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

Docker предоставляет команду для очистки большинства временных ресурсов.

```
docker system prune
```

Удаляются:

- остановленные контейнеры
- неиспользуемые сети
- dangling images
- build cache

Без подтверждения:

```
docker system prune -f
```

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

```
docker system prune -a
```

Удаляются:

- все остановленные контейнеры
- все неиспользуемые образы
- сети

- build cache

??????? build cache

Build cache создаётся при сборке Docker-образов.

Удалить cache можно так:

```
docker builder prune
```

Без подтверждения:

```
docker builder prune -f
```

????????? ???????? Docker-??????????

Иногда полезно посмотреть, какие директории занимают больше всего места.

```
du -sh /var/lib/docker/*
```

Это поможет понять, где именно находятся большие файлы.

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

На серверах разработки рекомендуется периодически выполнять:

```
docker system prune -f
```

Если активно собираются образы:

```
docker builder prune -f
```

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

Пример набора команд:

```
docker container prune -f
docker image prune -a -f
docker volume prune -f
docker network prune -f
docker builder prune -f
```

Эти команды удаляют почти все неиспользуемые ресурсы.

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

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

Если выполнить:

```
docker compose down -v
```

Docker удалит **все volumes проекта**, включая базы данных.

Поэтому эту команду нужно использовать осторожно.

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

Если удалить все образы, Docker будет вынужден заново скачивать их из registry.

Это может занять время.

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

Очистку стоит выполнять, если:

- диск сервера заполняется
- Docker занимает слишком много места
- накопилось много старых образов
- часто выполняется сборка контейнеров

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

```
docker system df                # использование диска Docker

docker container prune          # удалить остановленные контейнеры
docker image prune              # удалить dangling images
docker image prune -a           # удалить неиспользуемые образы

docker volume prune             # удалить неиспользуемые volumes
docker network prune            # удалить неиспользуемые сети

docker builder prune            # очистить build cache

docker system prune             # очистка системы Docker
docker system prune -a          # полная очистка
```

????

Со временем Docker накапливает большое количество неиспользуемых ресурсов:

- контейнеров
- образов
- volumes
- build cache

Регулярная очистка системы помогает:

- освободить дисковое пространство
- поддерживать стабильную работу сервера
- избежать проблем с переполнением диска.