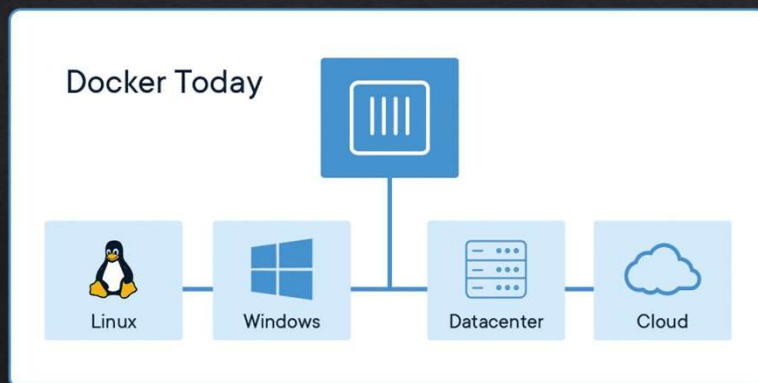


docker

Контейнерная виртуализация: Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.



Виртуализация

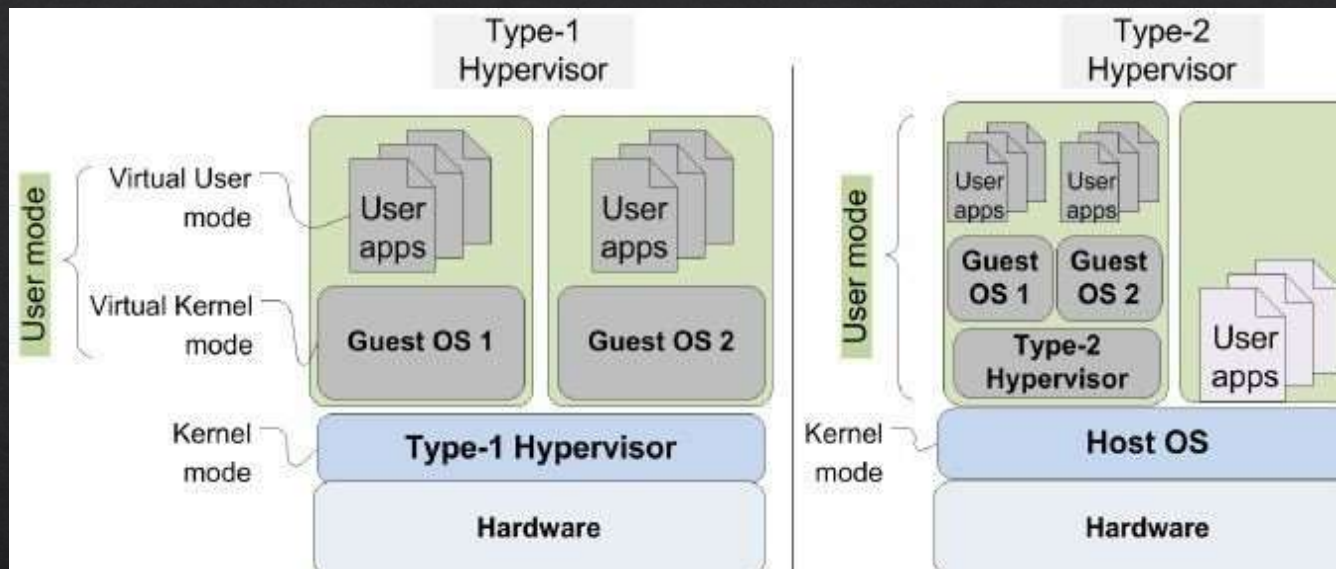
предоставление ресурсов, абстрагированное от их реализации

- Материальная выгода
- Безопасность
- Универсальность
- Гибкость

Virtualization						
Hardware <ul style="list-style-type: none">• Full<ul style="list-style-type: none">• Bare-Metal• Hosted• Partial• Para	Network <ul style="list-style-type: none">• Internal Network Virtualization• External Network Virtualization	Storage <ul style="list-style-type: none">• Block Virtualization• File Virtualization	Memory <ul style="list-style-type: none">• Application Level Integration• OS Level Integration	Software <ul style="list-style-type: none">• OS Level• Application• Service	Data <ul style="list-style-type: none">• Database	Desktop <ul style="list-style-type: none">• Virtual desktop infrastructure• Hosted Virtual Desktop

Виртуализация программная, аппаратная, контейнерная, паравиртуализация, виртуализация приложений, сетей, окружений...

Основные типы виртуализации



*Типе 1: быстрее, защищенное, устойчивее к ошибкам, более оптимальная
утилизация ресурсов системы
Типе 2: менее зависимы от железа, проще в
использовании, шире возможности эмуляции*

Type-1 vs Type-2

Hypervisor/Attributes	Bare metal hypervisor	Hosted hypervisor
Alias	Also known as Type 1 or Native hypervisor	Also known as Type 2 hypervisor
Hardware access	Can access the hardware directly	Accesses and uses the hardware resources via the underlying operating system
Virtualization	Hardware-based virtualization	Operating System based virtualization
Operation	The hypervisors run the guest OS and its applications	The hypervisor runs on the underlying operating system
Scalability	Highly scalable since it is not dependent on an underlying OS	Its reliance on the underlying OS makes it less scalable
Speed	Faster as it can access hardware resources directly	Slower because of the dependence on the underlying OS
Performance	The absence of a middle layer makes its performance good	The presence of a middle layer (underlying OS) reduces performance as it runs with extra overhead
Suitability	Well suited for enterprise setups and large deployments	Well suited for personal/small deployments
Security	Run directly on physical hardware without underlying OS, and are hence secure from the vulnerabilities of the OS	Are subject to the vulnerabilities of the OS, and are hence less secure
Cost	Higher cost	Comparatively lower cost
Example Implementations	VMWare ESXi, Microsoft Hyper-V Server, and open-source KVM	Oracle VM VirtualBox, VMware Workstation Pro, VMware Fusion

<https://www.parallels.com/blogs/ras/bare-metal-hypervisor/>

Для сервера: KVM,
VMWare, Hyper-V
(если Windows)

- для домашнего:
Proxmox

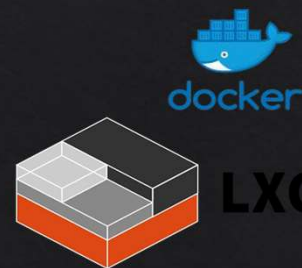
Для дома: Hyper-V,
VirtualBox, Parallels

Контейнерная виртуализация

- Виртуализация, которая работает на уровне операционной системы, в ядре.

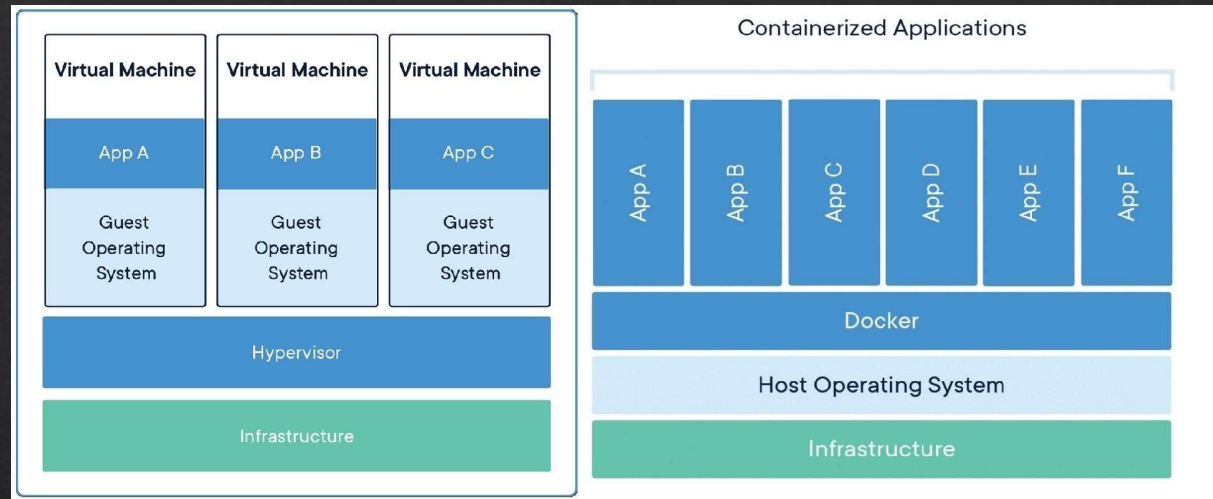
- При контейнеризации ядро операционной системы поддерживает несколько изолированных экземпляров пространства вместо одного. Эти экземпляры пространств называют контейнерами.

- Ядро – центральная часть ОС, которая координирует ресурсы компьютера (процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации) для приложений. Также ядро предоставляет файловую систему и сетевые протоколы



Docker

```
# Installation
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker
$USER docker run hello-world
```

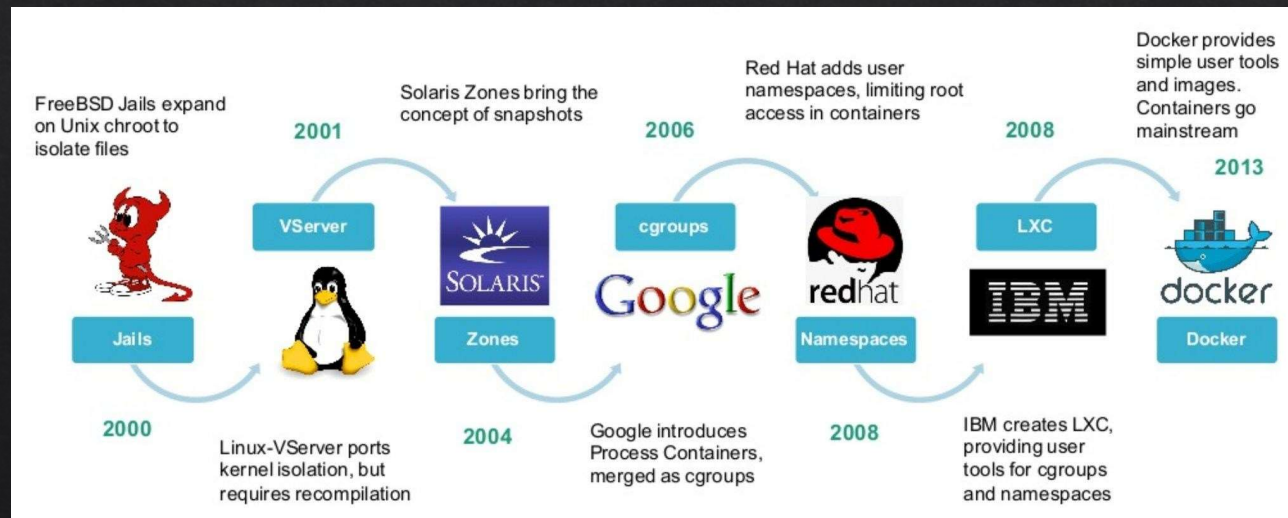
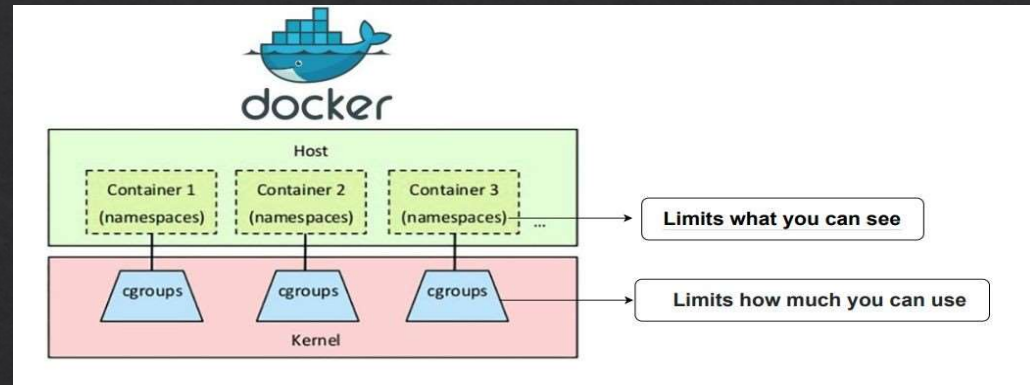


What's Diff?	VMs	Containers
size	Heavyweight (GB)	Lightweight(MB)
BootTime	Startup time in minutes	Startup time in seconds
Performance	Limited performance	Native performance
OS	Each VM runs in its own OS	All containers share the host OS
Runs on	Hardware-level virtualization(Type1)	OS virtualization
Memory	Allocates required memory	Requires less memory space
Isolation	Fully isolated and hence more secure	Process-level isolation, possibly less secure

VMs vs



Before Docker



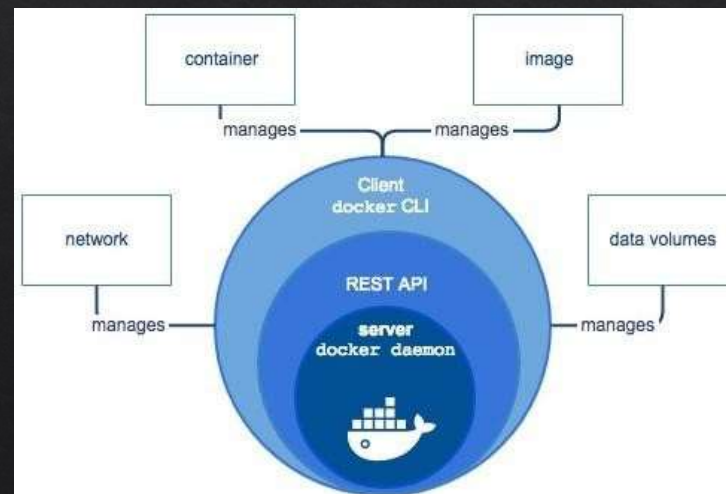
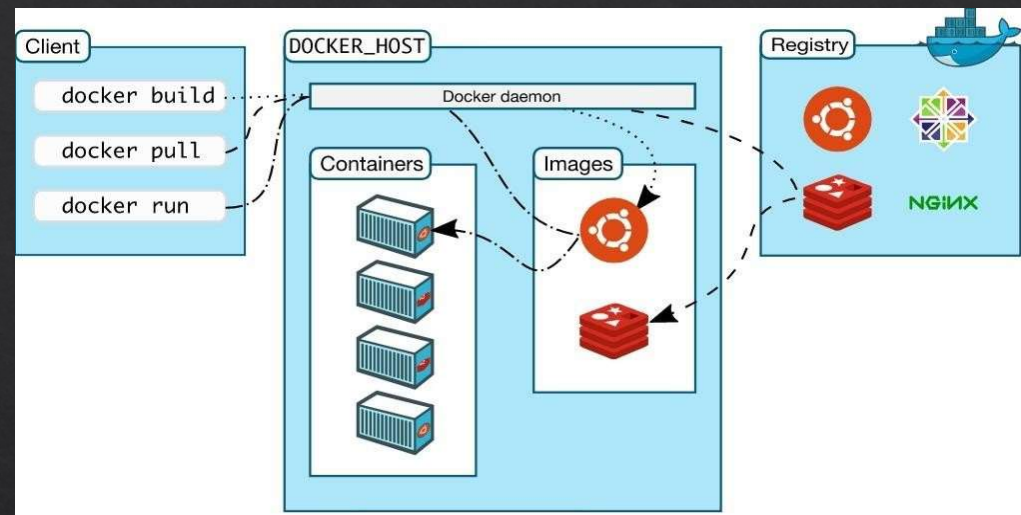
Docker architecture

Docker Client: утилита Docker

Objects: контейнеры и образы

Docker Daemon: демон (бэкграунд сервис), выполняющий сборку образов, запуск, управление контейнерами (может располагаться удаленно)

Docker Registry: хранилище Docker-образов



Начинаем работу с Docker

```
docker pull hello-world # Скачивает образ "hello-world" из Docker Hub (можно из Google Container Registry, Azure Container Registry, AWS ECR, self-hosted репозитория)
```

```
docker images # Список скачанных докер-образов в системе
```

```
docker ps # Список запущенных контейнеров
```

```
docker ps -a # Список всех контейнеров (в т.ч. остановленных - их можно снова запустить)
```

```
docker rm <cont_id> # Удалить контейнер
```

https://docs.docker.com/get-started/docker_cheatsheet.pdf

Запускаем контейнер

`docker run hello-world` # Пуллит (`docker pull hello-world`) образ из Docker Hub, создает контейнер на его основе (`docker create hello-world`) и запускает его (`docker start ...`)

`docker run -it ubuntu /bin/bash` # Запускаем Ubuntu с командой `/bin/bash` в интерактивном режиме

`docker run -d --rm -p 8080:80 nginx` # Запускает nginx в бекграунде (`-d`), перенаправляет 8080 порт хоста в 80ый порт в контейнере (`-p 8080:80`), после остановки контейнера он автоматически будет удален (`--rm`)

`docker exec -it <cont_id> <command>` # Выполнить команду внутри работающего контейнера в интерактивном режиме (`-it`)

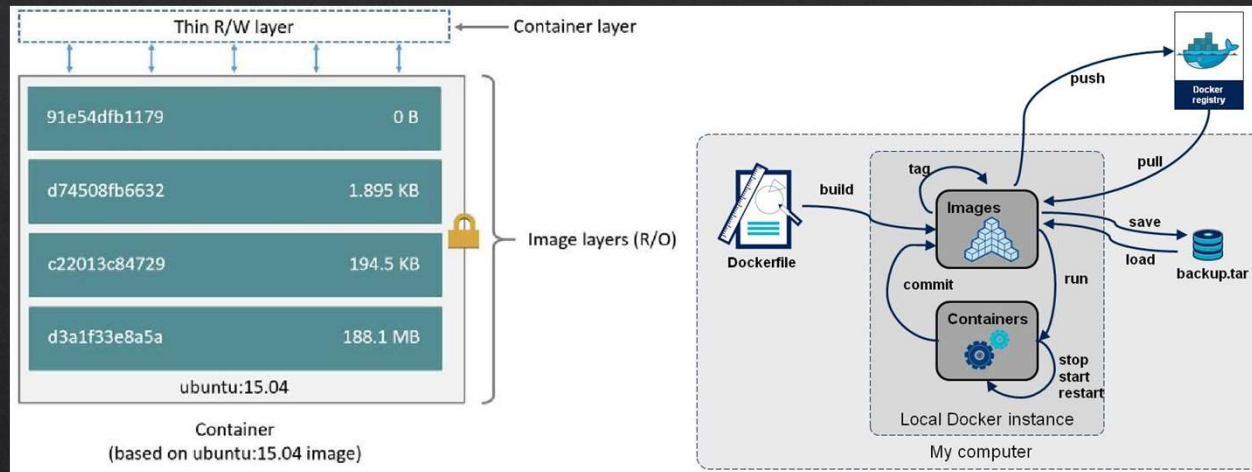
Флаг `-v` позволяет mount'ить директорию
хоста внутрь контейнера
`docker run \`
 `-it \`
 `--rm \`
 `-p 0.0.0.0:8888:8888 \`
 `-e NOTEBOOK_ARGS="--`
 `NotebookApp.token="" \`
 `-v $(pwd):/home/jovyan/work \`
 `jupyter/scipy-notebook`

`docker logs <cont_id>` # Логи контейнера

`docker stop <cont_id>` # Остановить контейнер

https://docs.docker.com/get-started/docker_cheatsheet.pdf

Создаем свой Docker-образ



Образы и контейнеры хранятся тут:

Ubuntu: /var/lib/docker/

Fedora: /var/lib/docker/

Debian: /var/lib/docker/

Windows: C:\ProgramData\DockerDesktop

MacOS: ~/Library/Containers/com.docker.docker/Data/vms/0/

```
# Создаем докер-образ из текущей  
директории docker build -t  
image_name:tag_name .
```

```
# Логинимся на Docker Hub  
docker login
```

```
# Загружаем собранный образ на Docker  
Hub docker push image_name:tag_name
```


Dockerfile

FROM: defines the base image; the FROM instruction must be the first instruction in Dockerfile.

LABEL: it's a Description about anything you want to define about this image,

ADD: copies a file into the image but supports tar and remote URL

COPY: copy files into the image, preferred over ADD.

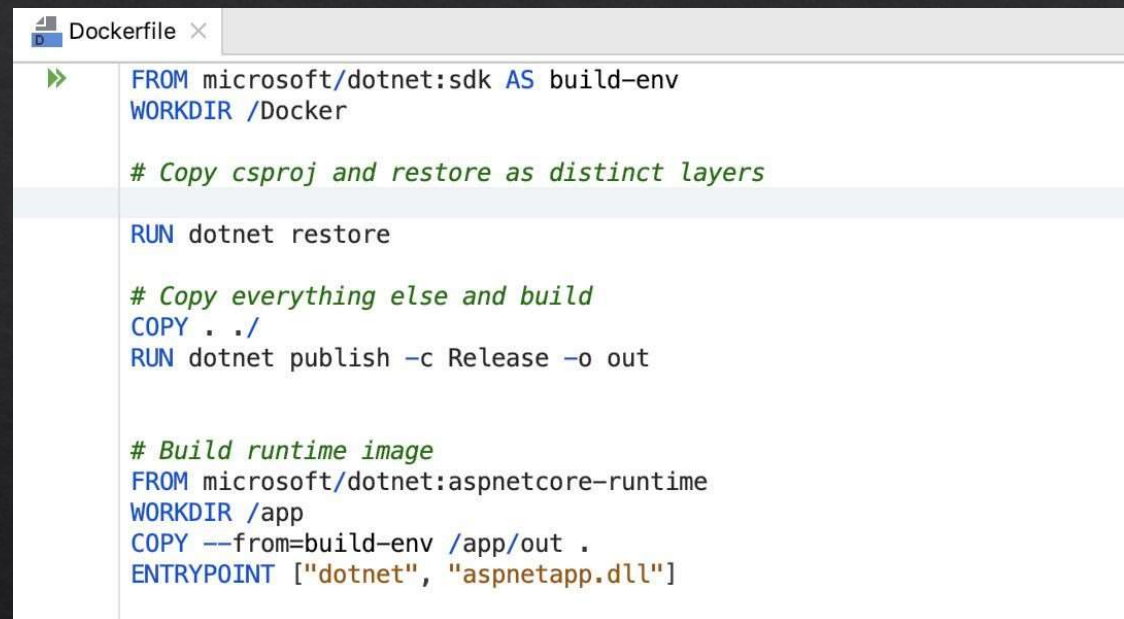
VOLUME: creates a mount point as defined when the container is run.

ENTRYPOINT: the executable runs when the container is run.

EXPOSE: documents the ports that should be published.

CMD ["executable","param1","param2"] There can only be one **CMD** instruction in a Dockerfile.

ENV: used to define environmental variables in the container.



```
Dockerfile x
>> FROM microsoft/dotnet:sdk AS build-env
WORKDIR /Docker

# Copy csproj and restore as distinct layers

RUN dotnet restore


# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Docker Compose

- указываем путь к build context'у
- имена сервисов
- параметры запуска
- Volume'ы
- порты
- виртуальные подсети
- ...

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



```
1 version: '3'
2
3 services:
4   product-service:
5     build: ./product
6     volumes:
7       - ./product:/usr/src/app
8     ports:
9       - 5001:80
10
11   website:
12     image: php:apache
13     volumes:
14       - ./website:/var/www/html
15     ports:
16       - 5000:80
17     depends_on:
18       - product-service
19
```

Continuous Integration / Continuous Delivery

Github Actions

Docker: build & push

<https://github.com/marketplace/actions/build-and-push-docker-images>

```
name: ci

on:
  push:
    branches:
      - 'main'

jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      -
        name: Set up QEMU
        uses: docker/setup-qemu-action@v2
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Login to DockerHub
        uses: docker/login-action@v2
        with:
          username: ${ secrets.DOCKERHUB_USERNAME }
          password: ${ secrets.DOCKERHUB_TOKEN }
      -
        name: Build and push
        uses: docker/build-push-action@v3
        with:
          push: true
          tags: user/app:latest
```