



# Знакомство с Docker

# Наша цель – портативность приложений!



**Python**

Запуск приложений в среде интерпретатора



**Django Framework**

Запуск приложений на сервере



REST API

**Django REST**

Доступ к приложениям через универсальные интерфейсы

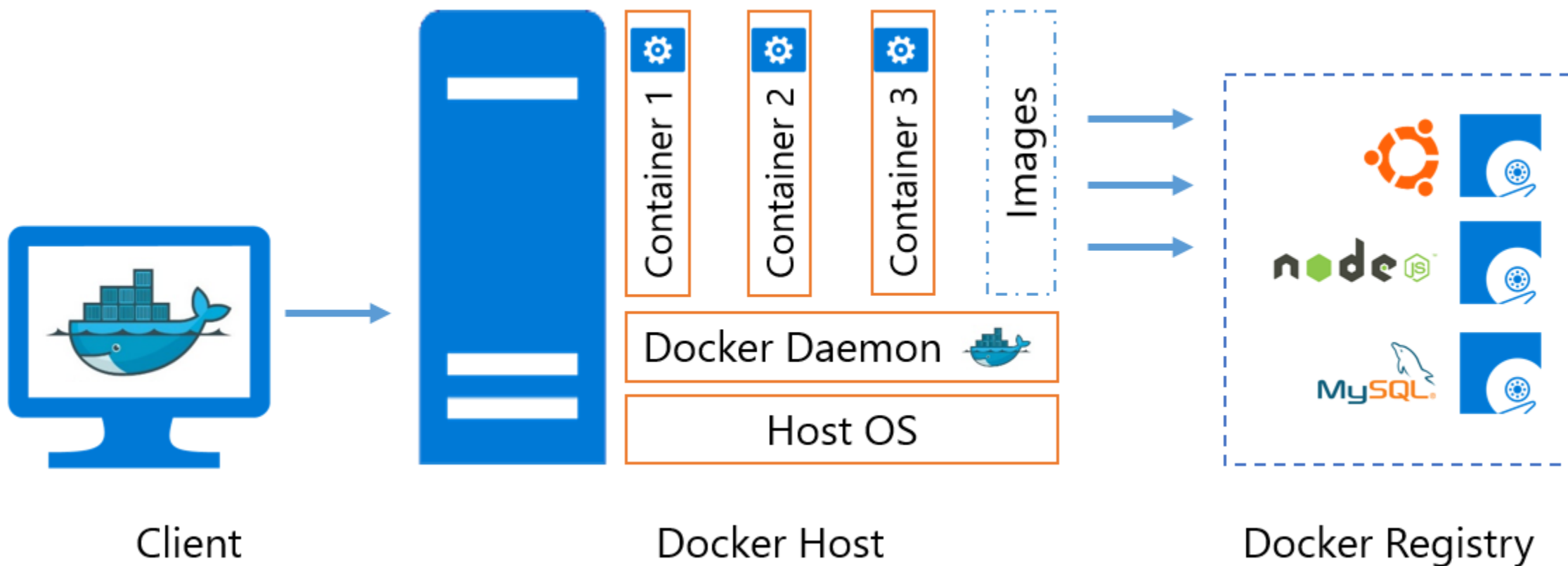


**Docker Compose**

Запуск приложений в независимых облачных контейнерах

# Что такое Docker

Docker – фреймворк для виртуализации на уровне ОС. Упаковывает приложения в контейнеры.



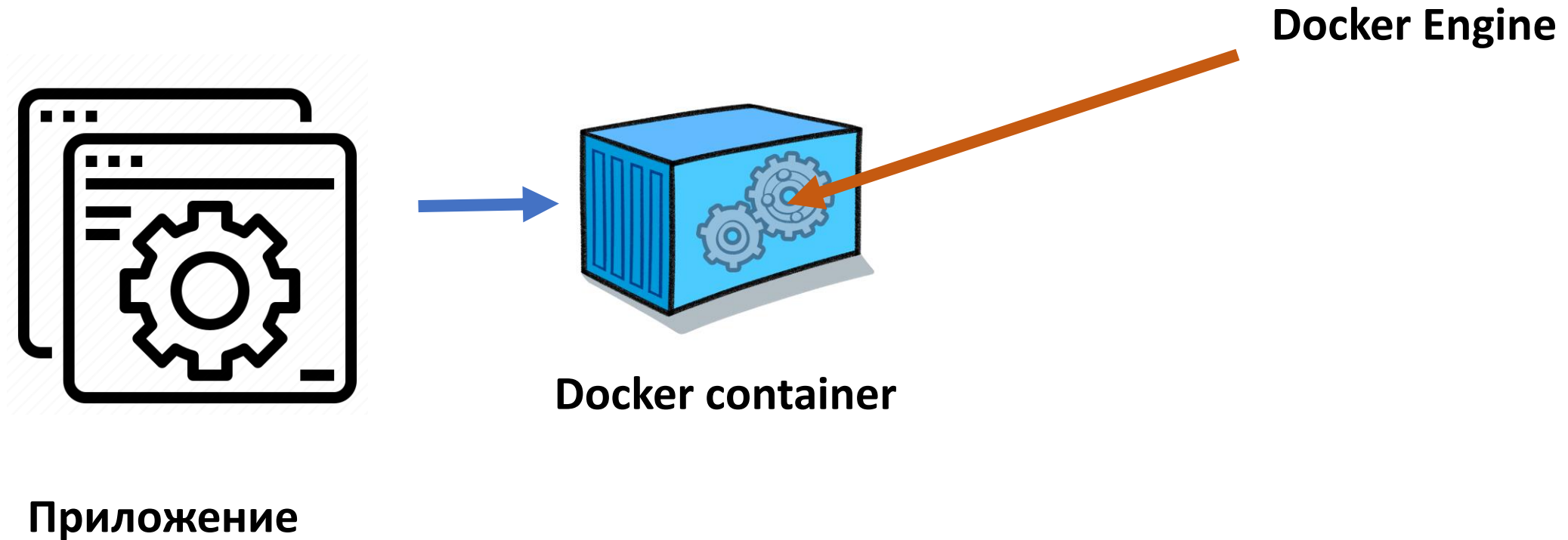
# Глоссарий

**Контейнер** – изолированная среда для запуска приложений. Представляет собой запускаемый экземпляр Docker-образа исходного приложения.

**Registry (реестр)** – центральный репозиторий, где распространяются контейнерные образы Docker. Официальный реестр – DockerHub. Можно создать private Docker Registry.

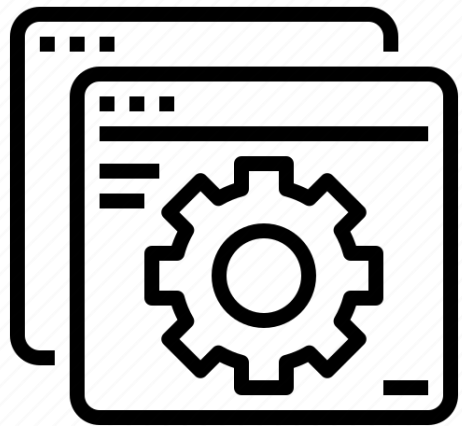
**Микросервис** – независимый компонент приложения, реализующий небольшую часть общего функционала.

# Контейнеры и контейнеризация

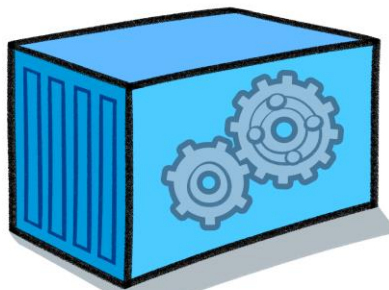


# Зачем всё это надо?

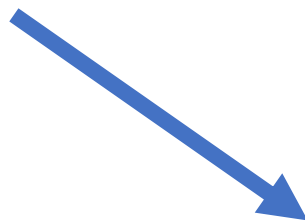
Docker container



Приложение



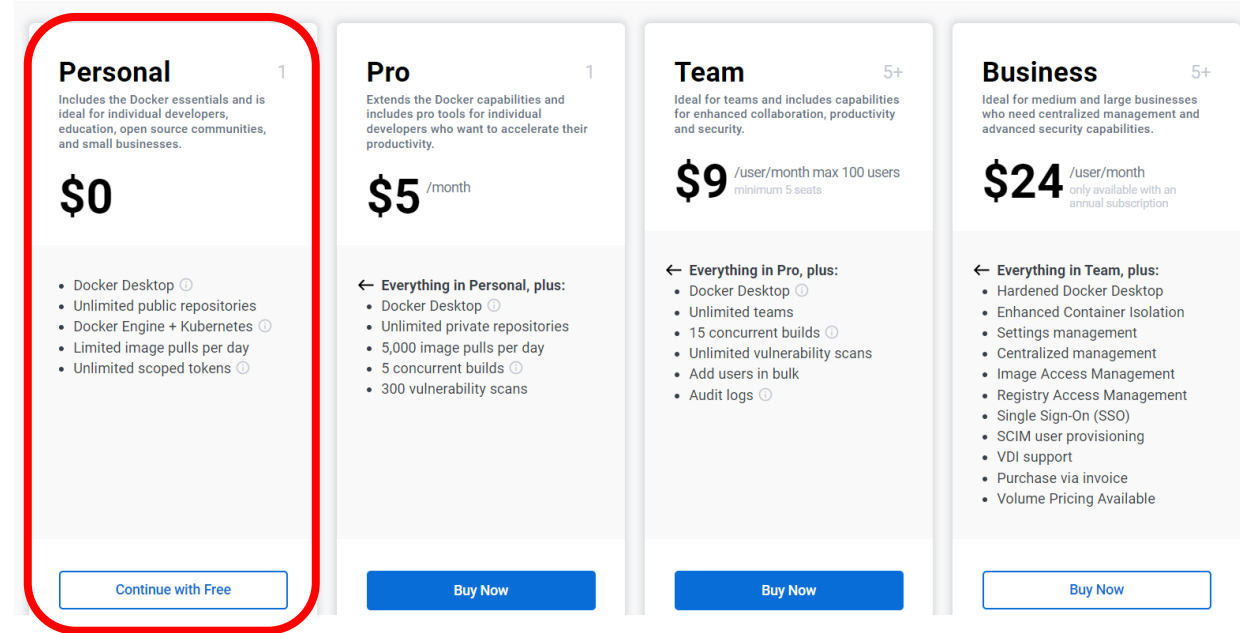
Маленький сервер  
\$ 5 в месяц



Мега сервер  
\$ 100 в месяц

# Подготовка к работе

1. Аккаунт в репозитории <https://hub.docker.com/>  
После регистрации можно и нужно скачать приложение для своей ОС.



2. Аккаунт на облачном сервисе для запуска контейнерных приложений. Например, Amazon Web Services <https://aws.amazon.com/>

# Установка docker

1. Установить Docker Desktop, соглашаясь с рекомендациями.
2. После запуска необходим перезапуск компьютера.
3. При возникновении ошибки WSL скачайте и установите обновление для ядра Linux: <https://learn.microsoft.com/ru-ru/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>



# Подготовка к работе

Автонастройка  
пробного проекта



What is a container?

Hands-on guide (5 mins)

```
1 FROM node
2 RUN mkdir -p
3 WORKDIR /app
4 COPY packa|
```

How do I run a container?

Hands-on guide (6 mins)



Run Docker Hub images

Hands-on guide (5 mins)


 Push to Hub





Publish your image

Hands-on guide (5 mins)

# «What is container»


Containers [Give feedback](#)

Search   ☐ Only show running containers

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	 <a href="#">welcome-to-docker</a> 6202cc2c1a53	<a href="#">docker/welcome-t</a>	Running	<a href="#">8088:80</a>	1 second ago	  

**3 Explore your container**

Containers are an isolated environment to run any code. Select the container, and go to the **Files** tab to see what's in it.

Name	Image
 <a href="#">welcome-to-docker</a> 0051bc7b19e	<a href="#">docker/wel</a>

3 шаг стартовой инструкции -> предложение установить расширение Telepresence (инструменты локальной разработки для Docker)

Install the Telepresence extension and connect to a shared cluster.

[Install](#)

[Dismiss](#)

1 шаг стартовой инструкции - можно запустить автоматически созданный контейнер. Он уже работает на localhost через порт 8088 (порт протокола TCP).

# «How do I run a container»

1. Создайте новый проект PyCharm и клонируйте проект
2. Зайдите в файл Docker, PyCharm предложит установить расширение Docker.
3. В Docker Desktop удалите контейнер, созданный по умолчанию (слайд 11) во избежание конфликта доступа к порту 8088.
4. В терминале:  
`docker run -d -p 8088:80 --name welcome-to-docker  
docker/welcome-to-docker`
5. В терминале убедитесь, что контейнер создан и запущен:  
`docker ps`

# Создание своего контейнера из того же приложения

1. Создайте свой собственный контейнер:  
**`docker build -t welcome-to-docker .`**
2. В Docker Desktop нажмите Run. В появившемся окне выберите любой свободный порт (не 8088). Можно взять, например, 8089.
3. Запускаем: `http://localhost:8089/`

# Основные команды docker

Команда	Описание	Пример
ps	Список запущенных контейнеров	<code>docker ps</code>
images	Список доступных образов	<code>docker images</code>
build	Сборка образа	<code>docker build -t my-dj-image .</code>
run	Запуск контейнера	<code>docker run -d --name my-cont my-dj-image</code> (-d - фоновый режим)
start / stop	Запуск / остановка контейнера	<code>docker start my-cont</code> <code>docker stop my-cont</code>
rm	Удаление контейнера	<code>docker rm my-cont</code>
rmi	Удаление образа	<code>docker rmi my-cont</code>
exec	Выполнить команду внутри контейнера	<code>docker run -d -p 8089:8089 --name my-cont my-dj-image</code>  <code>docker exec my-cont python manage.py migrate</code>

# Действия при загрузке приложения на Django в контейнер Docker

1. Создать Dockerfile в корне проекта
2. Создать requirements.txt, где описать конкретные зависимости  
Django==4.1  
ИЛИ  
install:  
    pip install -r requirements.txt
3. Собрать образ Docker  
docker build -t myapp:latest .
4. Запустить контейнер  
docker run -d -p 8089:8089 myapp:latest

# Используем базовый образ Python  
FROM python:3.11

# Копируем зависимости проекта  
COPY requirements.txt  
/app/requirements.txt

# Устанавливаем зависимости  
RUN pip install -r /app/requirements.txt

# Копируем код приложения в  
контейнер  
COPY . /app

# Устанавливаем рабочую директорию  
WORKDIR /app

# Запускаем приложение  
CMD ["python", "app.py"]

# Что почитать

Руководство по docker: <https://habr.com/ru/articles/310460/>

Docker для начинающих: <https://badtry.net/docker-tutorial-dlia-novichkov-rassmatrivaem-docker-tak-iesli-by-on-byl-ighrovoi-pristavkoi/>