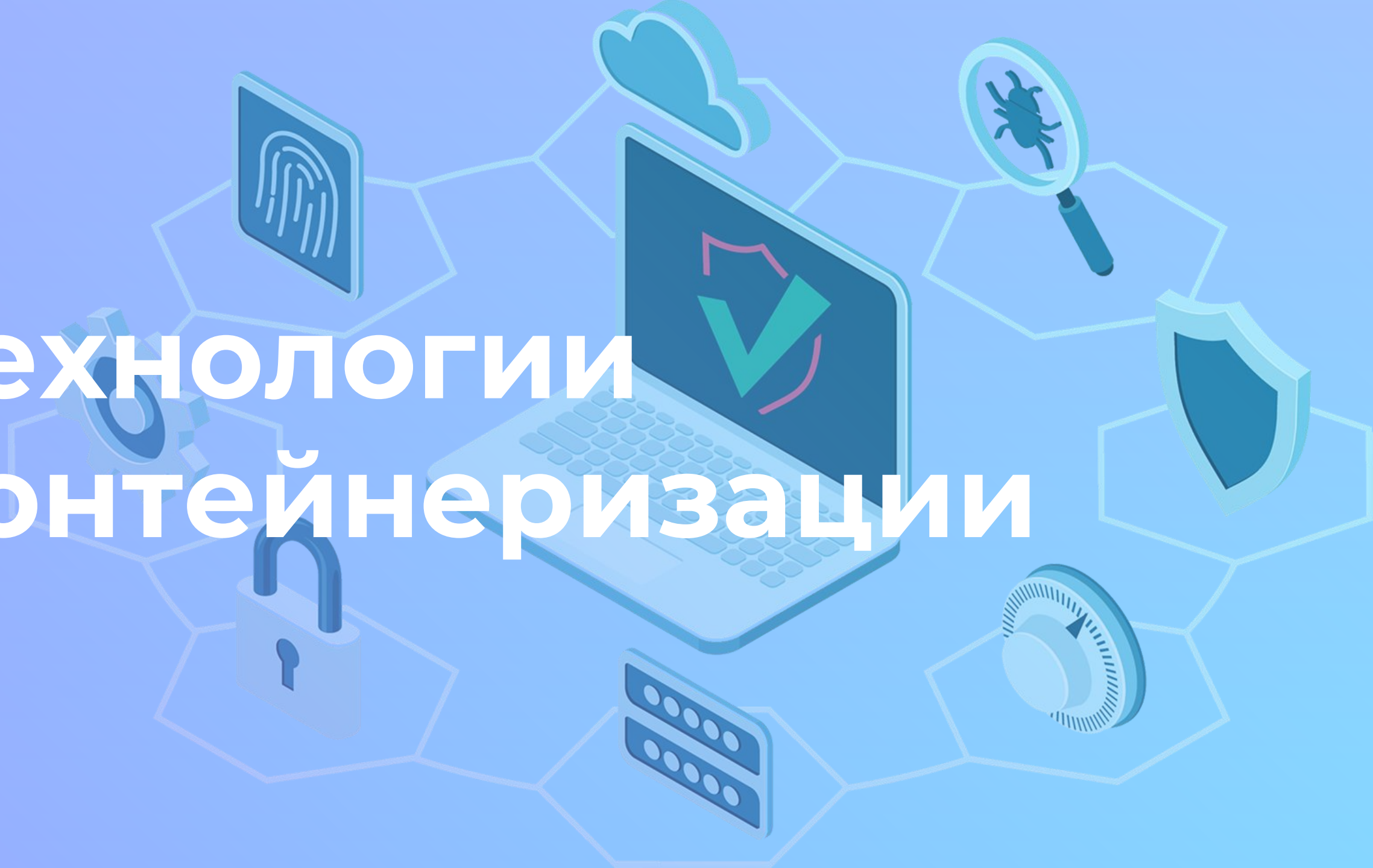


# Технологии контейнеризации



# Содержание

1. Основы Linux. Зачем нужна контейнеризация? Начало работы с docker.
2. Ключевые концепции контейнеризации, такие как образы, контейнеры и реестры
3. Практикум. Образы и контейнеры. выполнение заданий. Основы управления контейнерами (Основные команды docker)
4. Тест
5. Создание образов. Продвинутое управление контейнерами.
6. Тест
7. Практические задания (tier0 tier1)

# Зачем нужна контейнеризация?

контейнер делит с хостом:

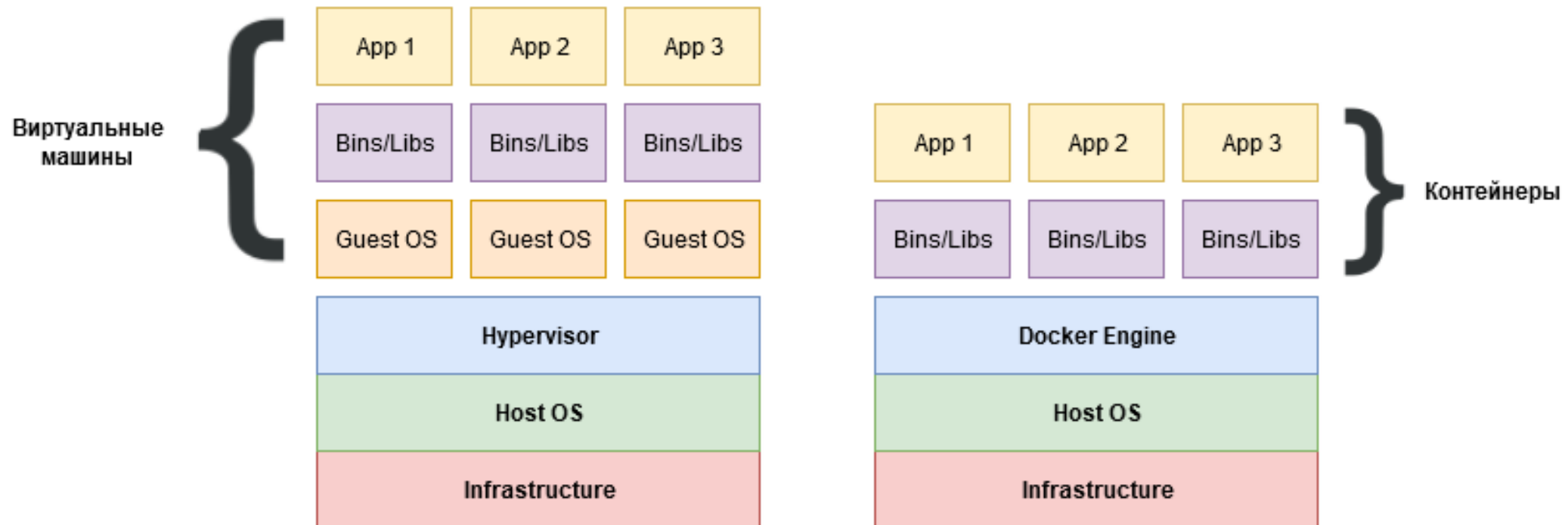
- ядро
- пространство памяти ядра

контейнер изолирует:

- пользовательское окружение

контейнер использует для изоляции возможности операционной системы (пространство имен). Применительно к Docker используются так называемые `cgroups` в ядре Linux.

# Зачем нужна контейнеризация?



# Зачем нужна контейнеризация?

	Docker	Виртуальные машины (VM)
<b>Время загрузки</b>	Загрузка через несколько секунд.	Загрузка виртуальных машин занимает несколько минут.
<b>Работает на</b>	Docker используют механизм исполнения.	ВМ используют гипервизор.
<b>Эффективность памяти</b>	Для виртуализации не требуется места, а значит, и меньше памяти.	Требуется загрузка всей ОС перед запуском поверхности, поэтому она менее эффективна.
<b>Изоляция</b>	Нет условий для систем изоляции.	Возможность вмешательства минимальна из-за эффективного механизма изоляции. Изолируется средствами процессора.
<b>Развертывание</b>	Развертывание легко, так как только одно изображение в контейнере может использоваться на всех платформах.	Развертывание сравнительно длительное, поскольку за выполнение отвечают отдельные экземпляры.

# Зачем нужна контейнеризация?

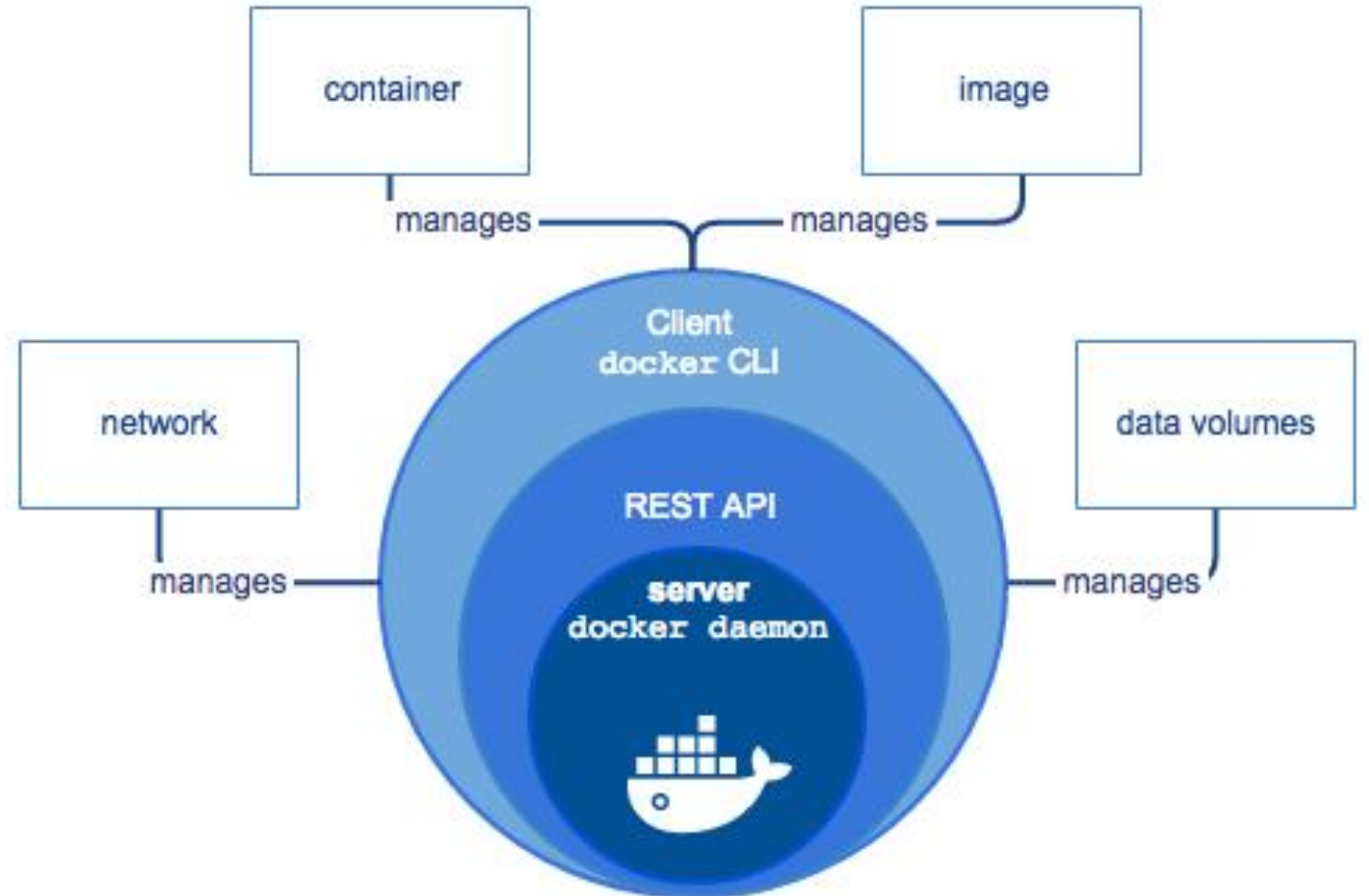


Источник: IHS Markit, 2019

Технологии контейнеризации приложений помогают сделать приложения более безопасными, облегчают их развертывание и улучшают возможности по их масштабированию.

# Начало работы с Docker

Docker (Docker Engine) - это одна из систем, позволяющих контейнеризировать приложения, предназначена для разработки, развертывания и запуска приложений в контейнерах



# Установка Docker и Docker-compose на Windows 10

## Системные требования

- Windows 10 64-bit: Pro, Enterprise, Education (Build 16299 или выше).

Для успешного запуска Client Hyper-V в Windows 10 требуются следующие предварительные требования к оборудованию:

- 64 bit процессор с поддержкой Second Level Address Translation (SLAT).
- 4GB системной памяти.
- Поддержка аппаратной виртуализации на уровне BIOS должна быть включена в настройках BIOS.



# Установка Docker и Docker-compose на Windows 10

## Подготовка

### Включаем функции Hyper-V Containers Window

Панель управления -> установка и удаление программ -> включение или отключение компонентов Windows.

Активируем пункт Hyper-V, который включает Hyper-V Managment Tools, Hyper-V Platform.

это можно выполнить через powershell или dism (необходимы права администратора).

#### **Powershell:**

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

#### **DISM:**

```
DISM /Online /Enable-Feature /All /FeatureName:Microsoft-Hyper-V
```

# Установка Docker и Docker-compose на Windows 10

## Установка

Скачиваем установщик Docker (Docker Desktop Installer) с Docker Hub:

<https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>

Установка Docker Desktop включает Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes и Credential Helper.

Запускаем установщик Docker Desktop Installer.exe и ожидаем пока он скачает все необходимые компоненты. После установки система потребует перезагрузки. Перезагружаемся и входим в систему.

После входа может возникнуть запрос на установку дополнительного компонента WSL2. Переходим по ссылке и скачиваем необходимый пакет с официального сайта Microsoft.

После скачивания выполняем установку WSL2, после которой снова потребуется перезагрузка.

# Установка Docker и Docker-compose на Windows 10

## Установка

Скачиваем установщик Docker (Docker Desktop Installer) с Docker Hub:

<https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>

Установка Docker Desktop включает Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes и Credential Helper.

Запускаем установщик Docker Desktop Installer.exe и ожидаем пока он скачает все необходимые компоненты. После установки система потребует перезагрузки. Перезагружаемся и входим в систему.

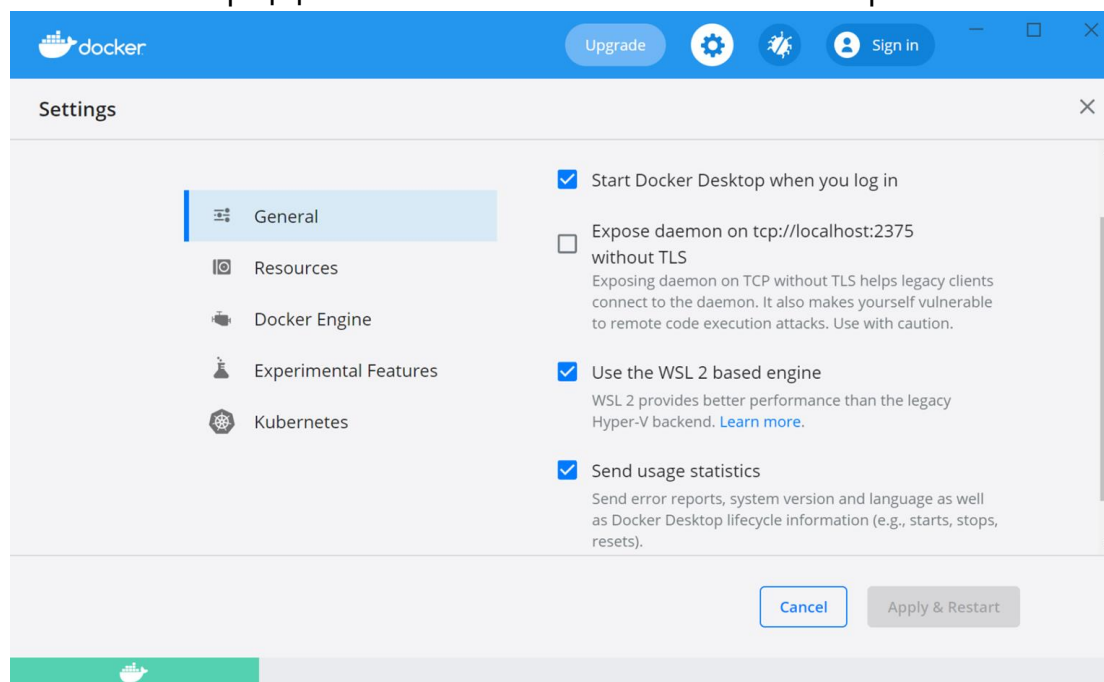
После входа может возникнуть запрос на установку дополнительного компонента WSL2. Переходим по ссылке и скачиваем необходимый пакет с официального сайта Microsoft.

После скачивания выполняем установку WSL2, после которой снова потребуется перезагрузка.

# Установка Docker и Docker-compose на Windows 10

## Настройка и запуск

Входим в систему и ждем запуска всех служб Docker. Когда все службы будут запущены, мы увидим в трее классический значок Docker — это значит что служба установлена и запущена. Далее можно запустить приложение Docker desktop. Далее можно изменить настройки Docker при необходимости:



# Установка Docker и Docker-compose на Ubuntu и Debian

1. Проведите пакетное обновление сервера:

```
$ sudo apt update
```

1. Установите зависимости:

```
$ sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common
```

1. Добавьте ключ GPG от официального репозитория в систему управления пакетами APT :

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -  
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable"
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

1. Обновите базы данных индекс-пакетов APT системы и переключитесь в репозиторий Docker:

```
$ sudo apt update  
$ apt-cache policy docker-ce
```

1. Установить Docker:

```
$ sudo apt install docker-ce
```

# Проверка корректного завершения установки

```
$ docker run hello-world
```

>hello  
world

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

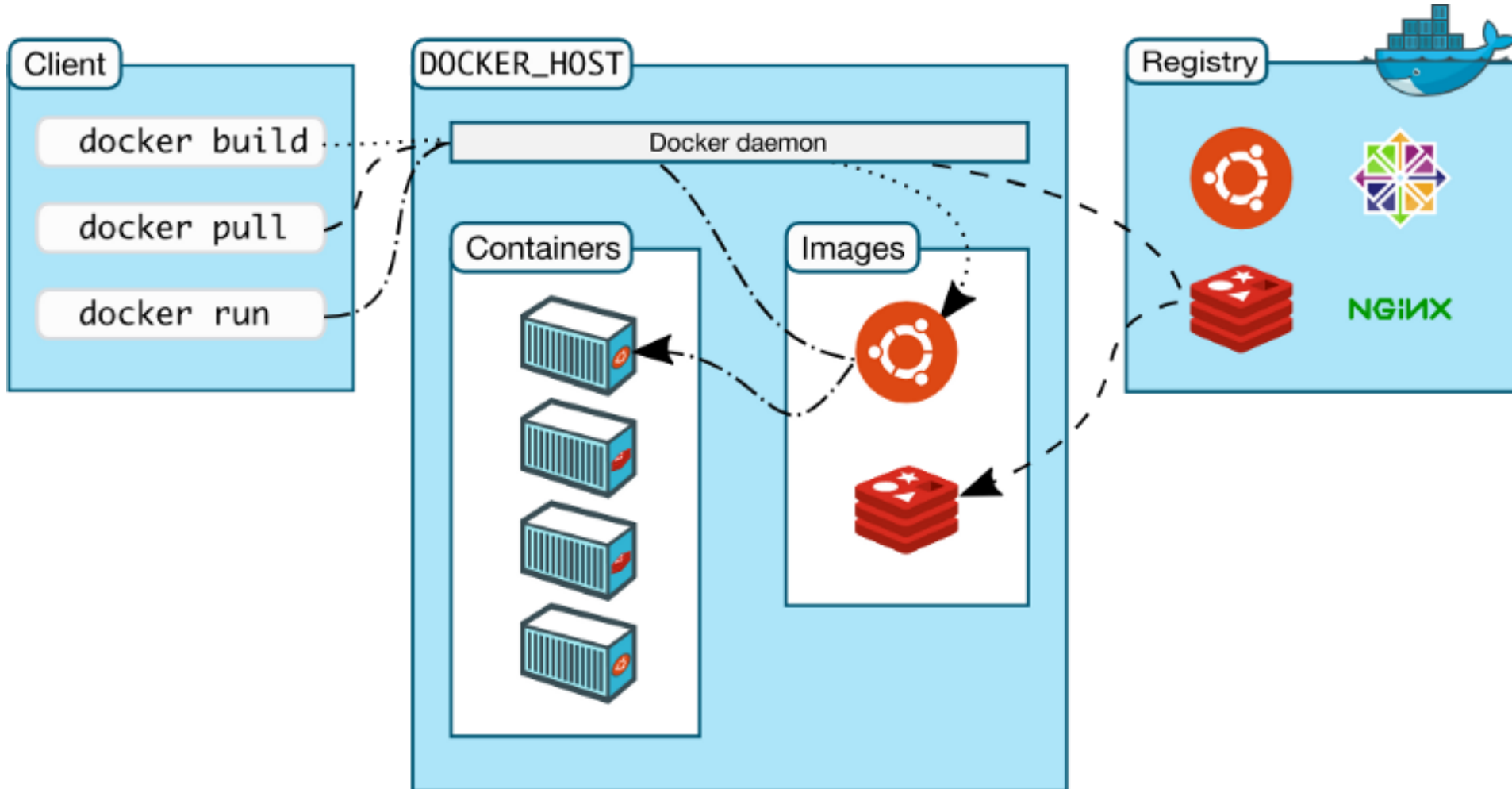
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

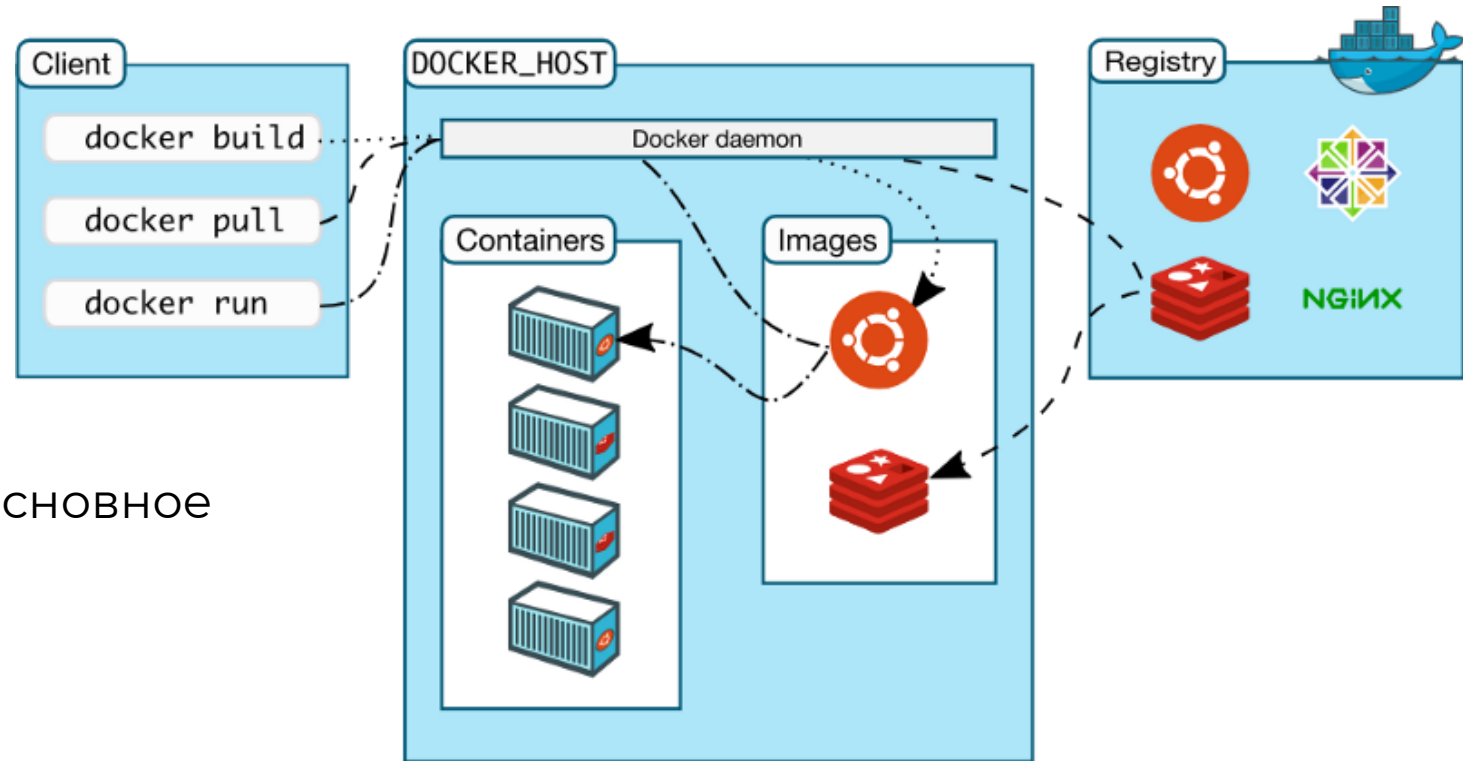
$ docker images hello-world
REPOSITORY   TAG       IMAGE ID   SIZE
hello-world  latest    feb5d9fea6a5  13.26kB
```

>hello  
world

# Ключевые концепции



# Ключевые концепции

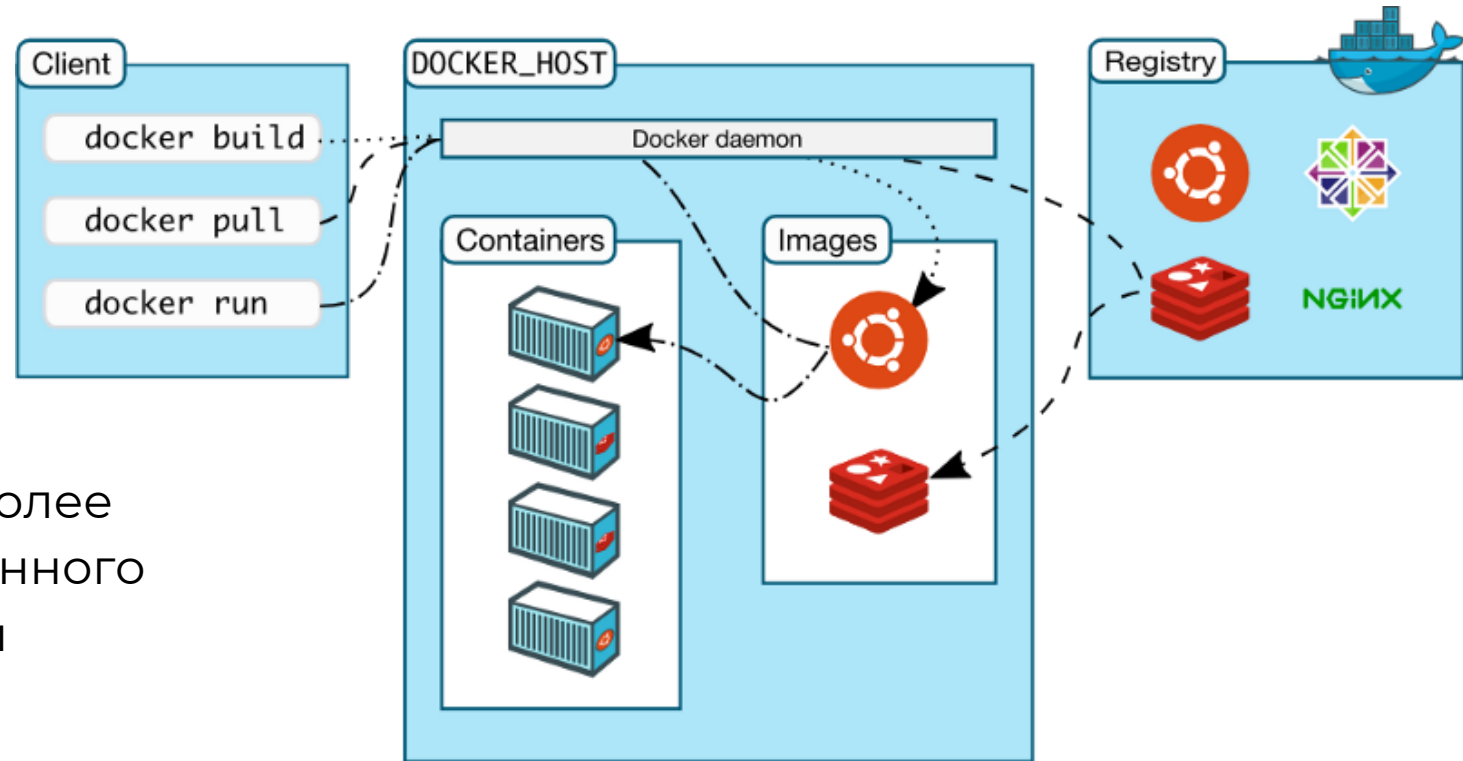


Клиент Docker (Docker Client) — это основное средство, которое используют для взаимодействия с Docker

Демон Docker (Docker Daemon) — это сервер Docker, который ожидает запросов к API Docker. Демон Docker управляет образами, контейнерами, сетями и томами



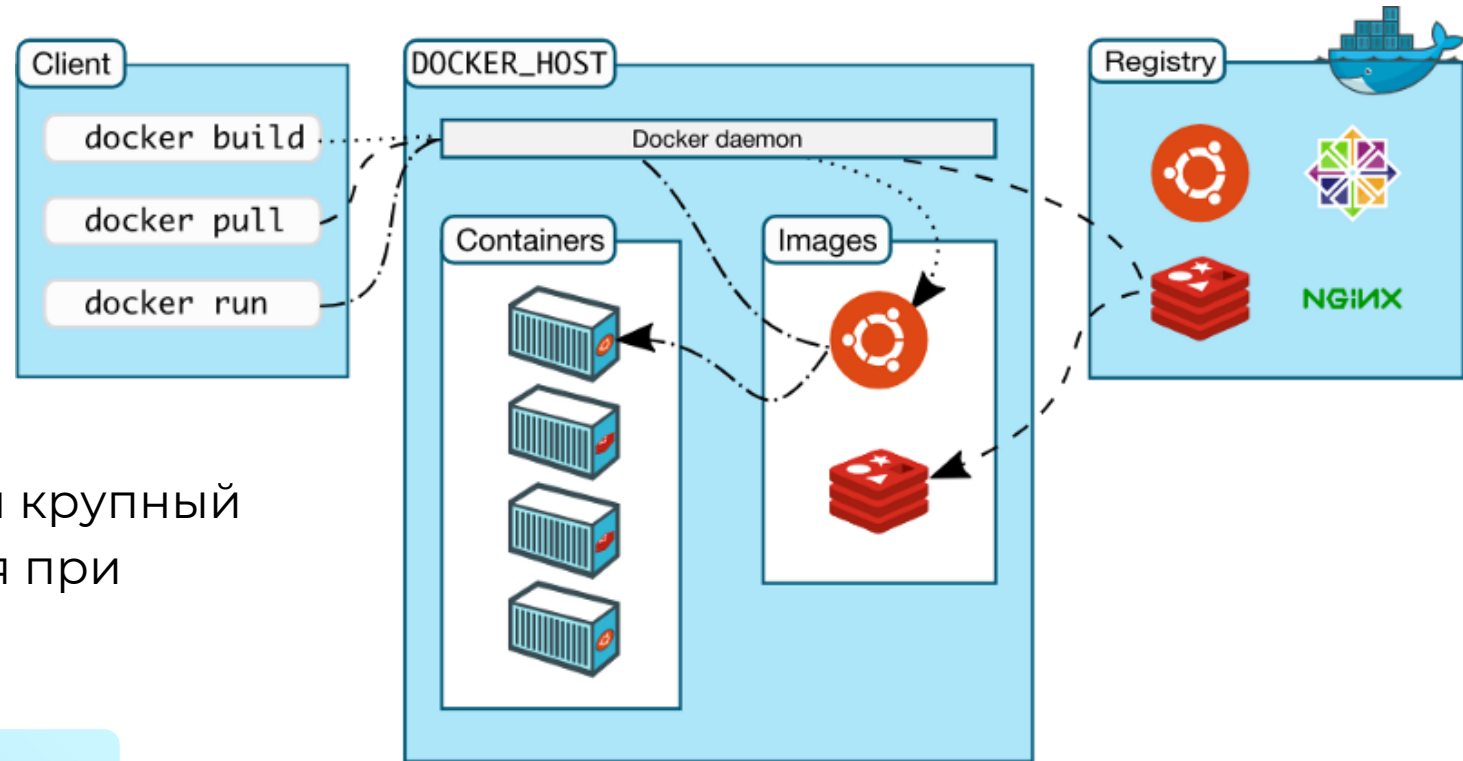
# Ключевые концепции



Тома Docker (Docker Volumes) – наиболее предпочтительный механизм постоянного хранения данных, потребляемых или производимых приложениями.

Реестр Docker (Docker Registry) – удаленная платформа, используемая для хранения образов Docker. В ходе работы с Docker образы отправляют в реестр и загружают из него.

# Ключевые концепции



**Docker Хаб (Docker Hub)** — это самый крупный реестр образов Docker, используется при работе с Docker по умолчанию.

**Репозиторий Docker (Docker Repository)** – набор образов Docker, обладающих одинаковыми именами и разными тегами.

**Теги** — идентификаторы образов.

# Ключевые концепции

**Контейнер** — среда, где разворачивается и запускается сохраненный заранее образ с отдельным ПО \ программой \ сервисом.

**Образ** — полностью работоспособный исполняемый пакет определенного ПО для запуска в контейнере Docker.

**Dockerfile** — файл инструкций для демона Docker под соответствующий образ. Для нового образа, сначала готовится Dockerfile после чего сам пакет образа.

# DEMO:

## Основные команды docker





СПАСИБО ЗА ВНИМАНИЕ!