



Уральский
федеральный
университет

имени первого Президента
России Б.Н.Ельцина

Институт радиоэлектроники
и информационных
технологий — РТФ

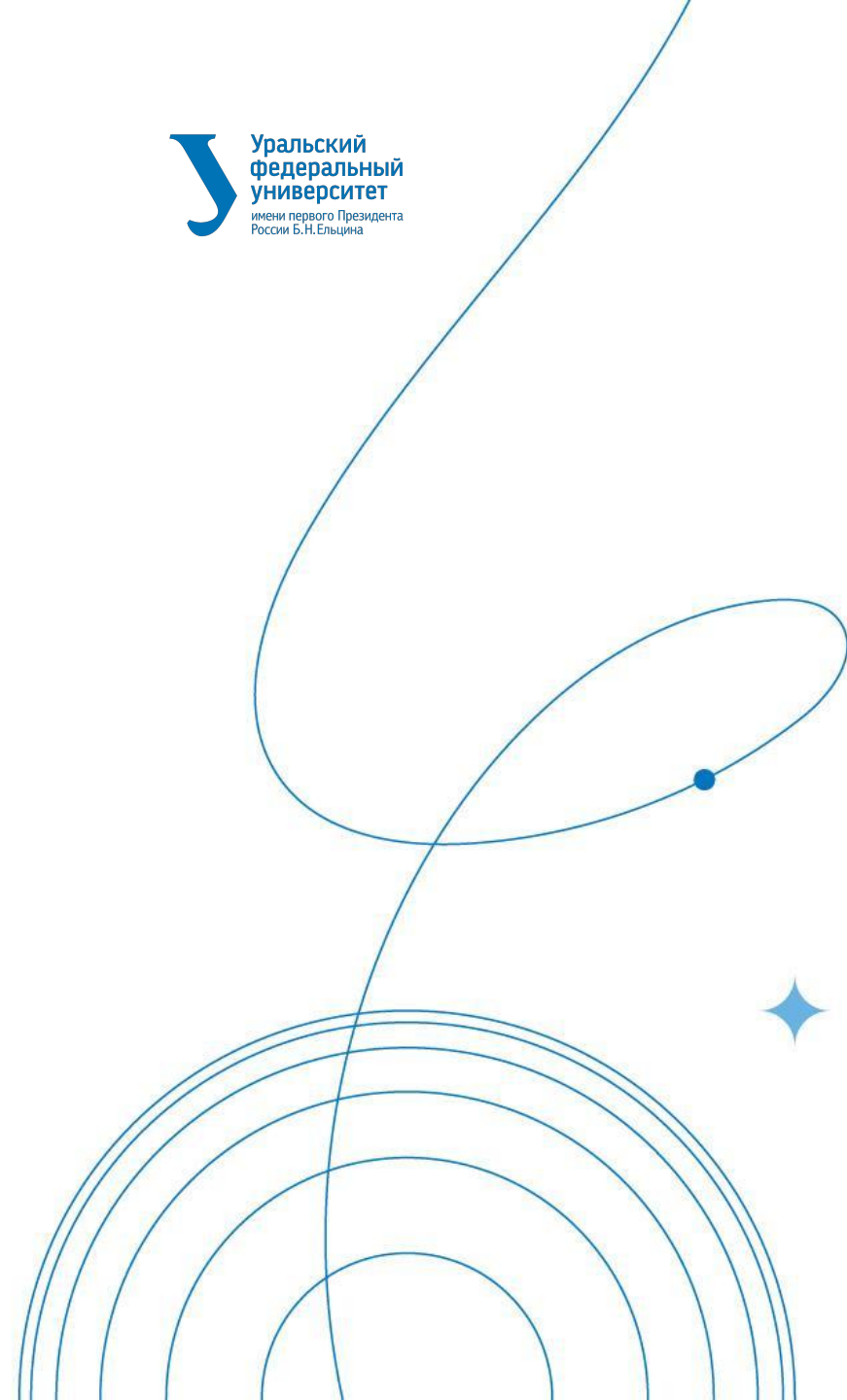


Уральский
федеральный
университет

имени первого Президента
России Б.Н.Ельцина

Виртуализация и контейнеризация

Докладчик
Корелин Иван



В этой лекции мы рассмотрим следующие темы:

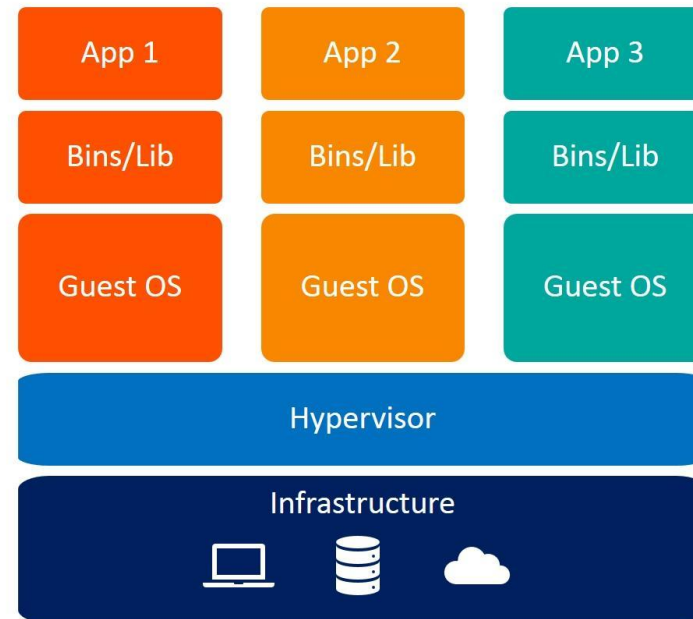
- Зачем необходимы изолированные среды выполнения программ и какие есть подходы для ее реализации.
- Виртуальные машины: подходы и программное обеспечение.
- Виртуальные среды: подходы и программное обеспечение.
- Контейнеризация и практические аспекты работы с docker и docker-compose.

Подход №1 — Виртуализация

Такой подход эффективен тем, что позволяет оптимально использовать имеющиеся аппаратные ресурсы:

- вычислительные процессоры:
 - центральный процессор (CPU, Central Processing Unit),
 - графический процессор (GPU, Graphics Processing Unit),
 - тензорный процессор (TPU, Tensor Processing Unit),
- оперативную память,
- средства сетевого обмена информацией,
- средства хранения информации, жесткие и оптические диски.

Изолированный процесс, пользующийся виртуальными ресурсами, называется **виртуальной машиной**.

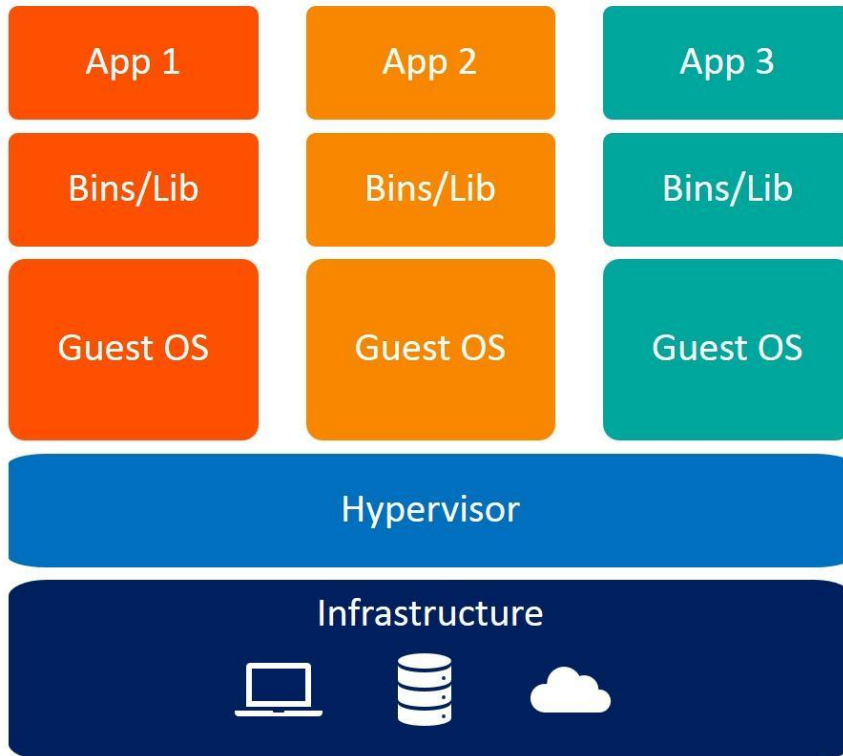


Virtual Machines

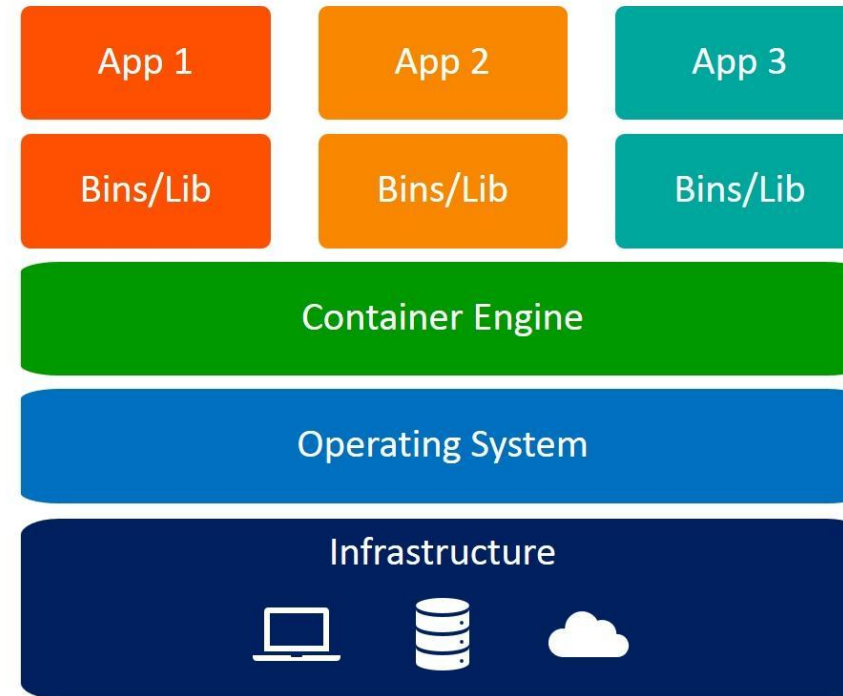
Виртуальная машина имеет свою операционную систему и работает как независимый компьютер. Кроме того, у нее есть свои уникальные характеристики (мощность процессора и объем памяти). В одних системах нужен большой объем оперативной памяти, а в других — повышенные требования к скорости вычислений. Ясно что суммарный объем виртуальных мощностей не может превышать мощности физического аппаратного обеспечения.

Система, управляющая виртуальными машинами, называется **гипервизор**.

Оборудование, на котором работает виртуализация, называется **хост**, его операционная система — **хостовой**.

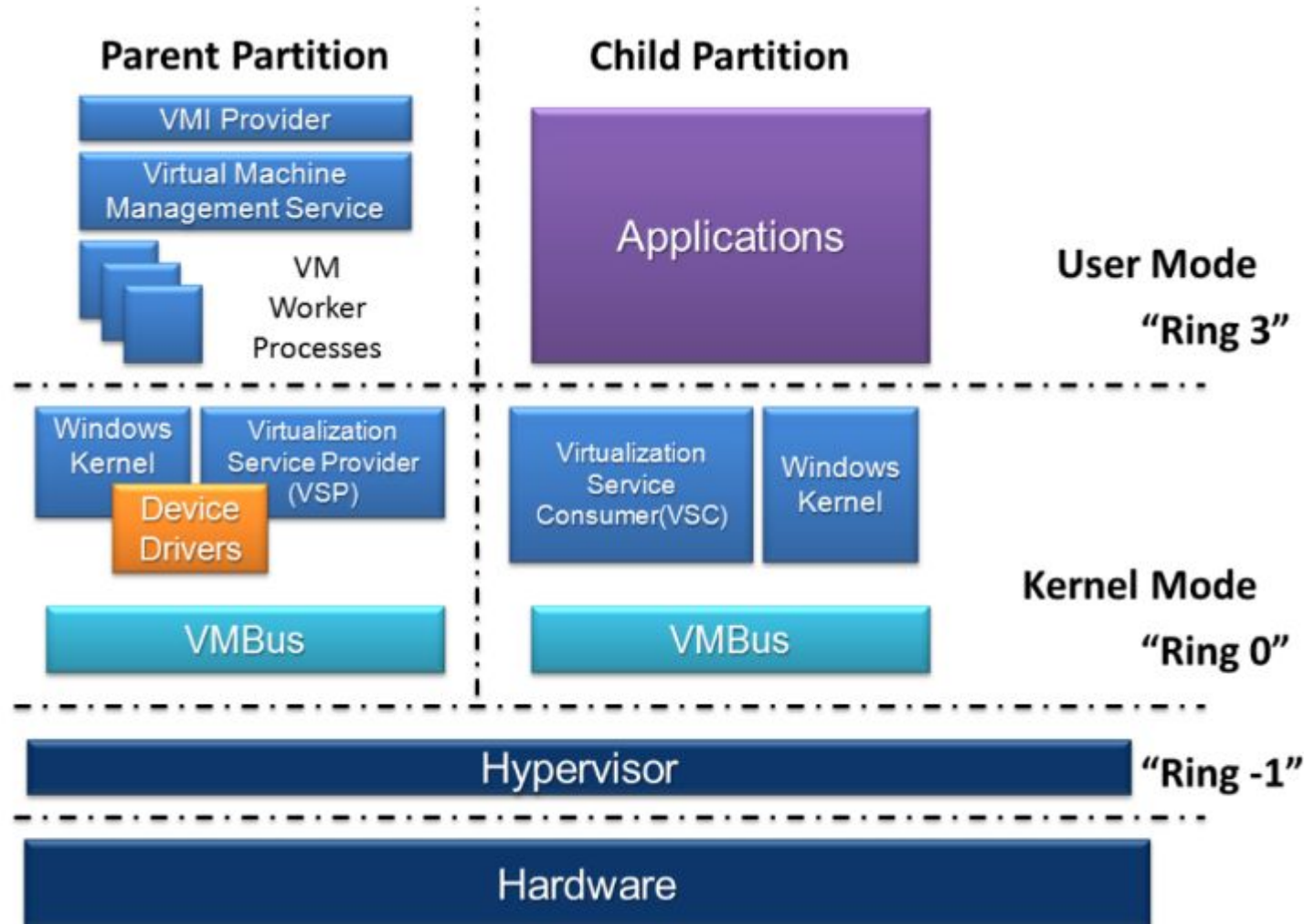


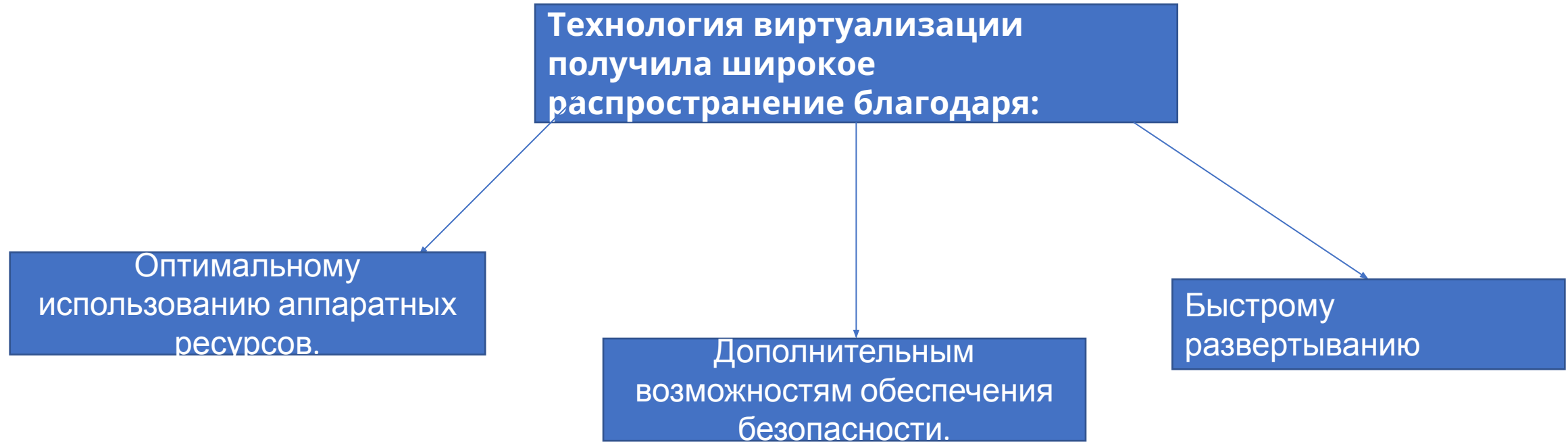
Virtual Machines



Containers

По такой схеме работают все системы виртуализации, отличаясь между собой в технических деталях и реализации. Вот, например, схема работы гипервизора Hyper-V, который входит в состав Microsoft Windows.





Примеры виртуальных машин:



бесплатный инструмент для виртуализации от корпорации Oracle



vmware

обеспечивает виртуализацию серверов, рабочих столов (desktops), хранилищ данных



входит в состав ОС Microsoft Windows, предоставляет решения по виртуализации серверов и рабочих компьютеров



виртуализация для серверов, рабочих компьютеров и приложений (недоступен в России).

Подход №2 —

Контейнеризация

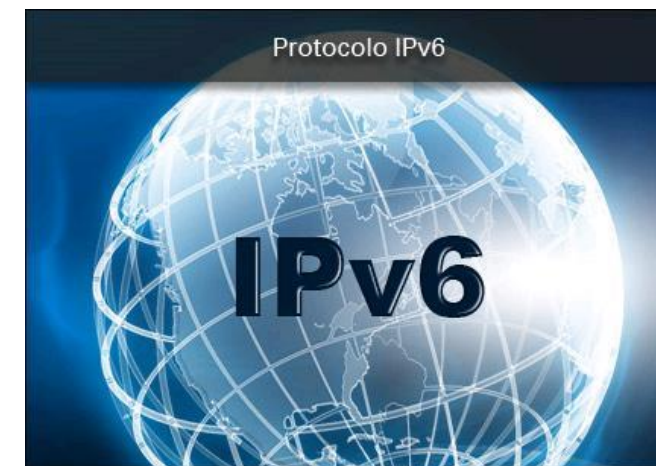
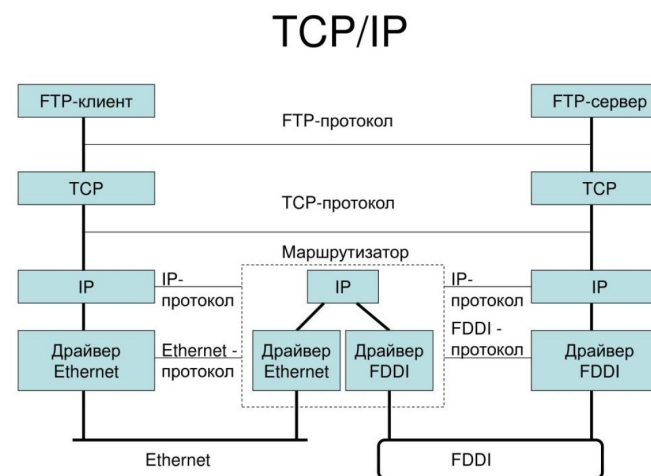
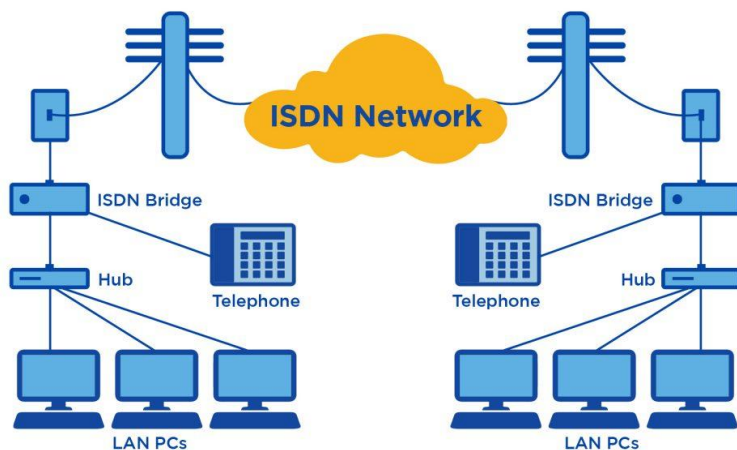
контейнерная виртуализация — это виртуализация на уровне хостовой операционной системы, которая позволяет запускать изолированные виртуальные системы на одном физическом узле, но не позволяет запускать операционные системы с ядрами, отличными от типа ядра базовой операционной системы.

Для управления группой контейнеров используются инструменты



Инструментов для виртуализации и контейнеризации достаточно много. В этой лекции мы ограничимся рассмотрением инструмента **VirtualBox** для виртуализации и **Docker** и **Docker-compose** для контейнеризации.

Подход №2 — Контейнеризация



Итоги

- Вы изучили ключевые технологии для создания виртуальных изолированных сред выполнения программы: **виртуализацию и контейнеризацию**.
- Эти технологии соответствуют общему тренду использования микросервисной архитектуры при создании программного обеспечения, ускоряют операции и процессы в проекте.
- Наиболее популярные инструменты для виртуализации: VirtualBox Oracle, VMWare, KVM, Hyper-V.
- Практически без конкурентов в настоящее время инструменты контейнеризации: Docker, Docker-compose и Kubernetes.
- В следующих юнитах мы подробнее рассмотрим создание виртуальных машин с VirtualBox и контейнеризацию с Docker, Docker-compose.

Нам понадобится

- Инструмент для виртуализации **VirtualBox** разработан компанией Oracle и распространяется бесплатно
- Также для работы вам понадобится образ для операционной системы, которая будет работать в виртуальной машине. В этой лекции мы будем использовать **Ubuntu**

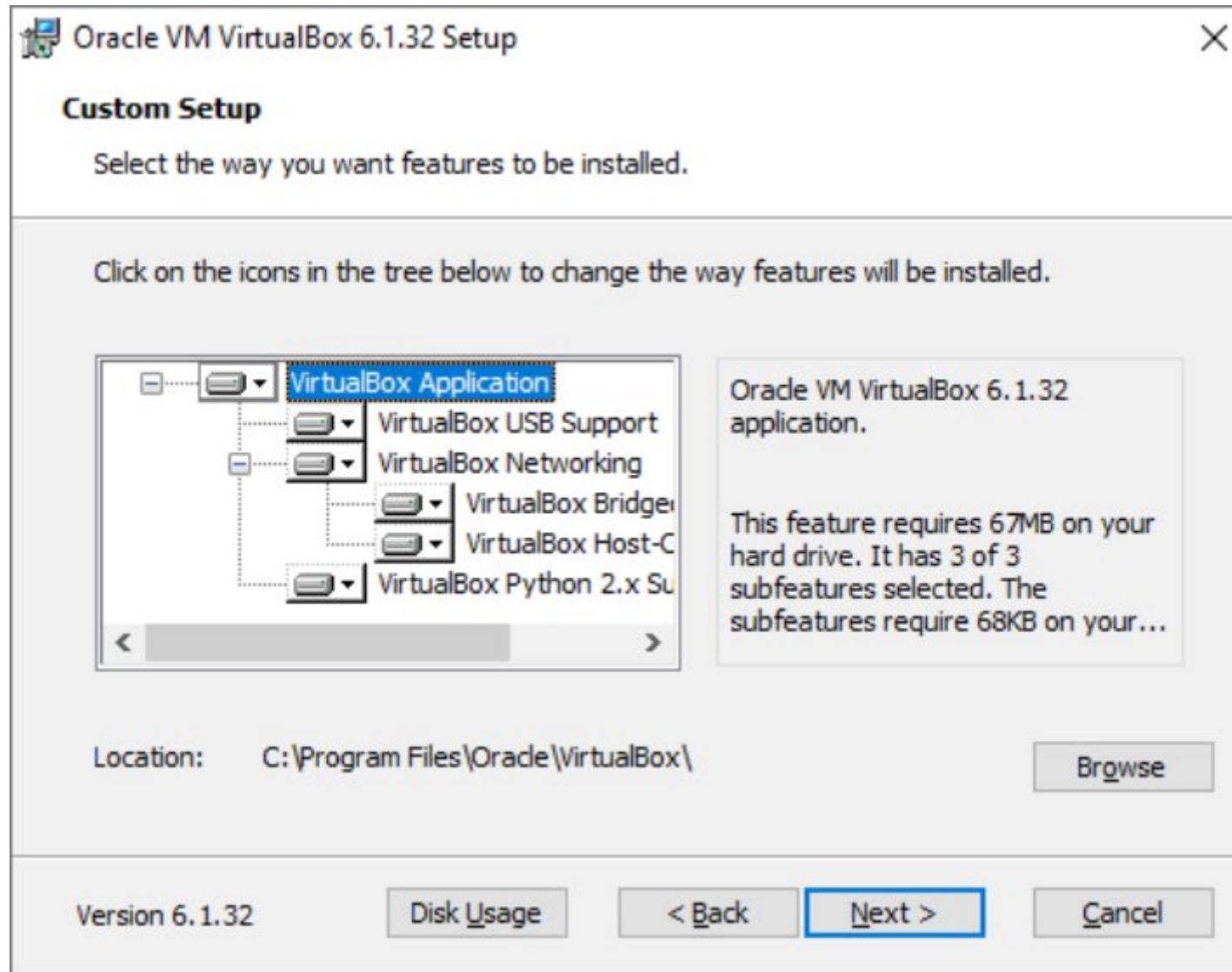
Для разнообразия мы изучим практическое применение инструментов в операционной системе Windows 64-bit. Внимательно выберите подходящую программу для установки VirtualBox на сайте. Кроме установщика самой программы понадобится еще установщик для пакета расширений VirtualBox, его можно скачать там же.

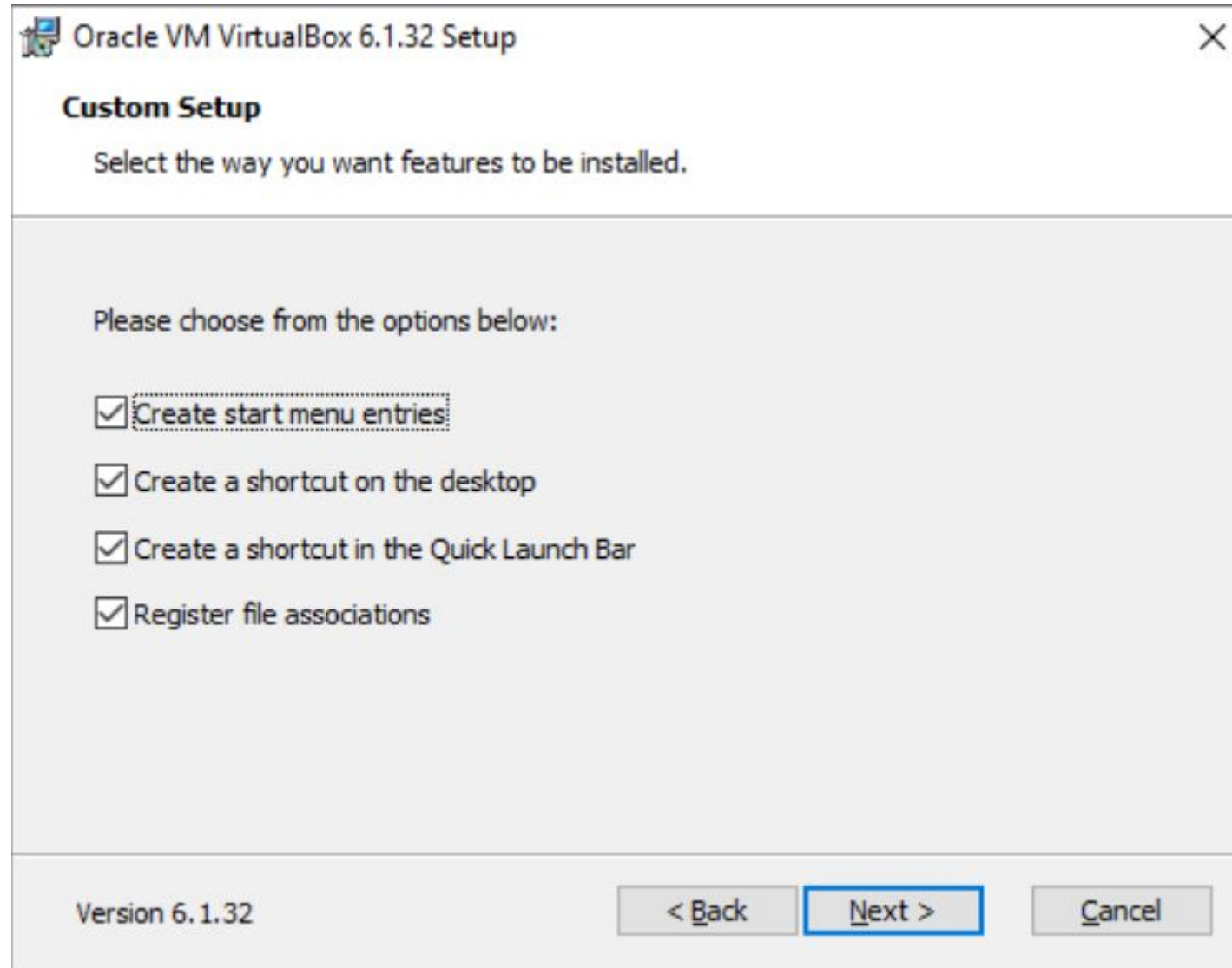
Виртуализация с использованием VirtualBox

После запуска исполняемого файла «*VirtualBox...-Win.exe*» следуйте обычному сценарию установки ПО в Windows.

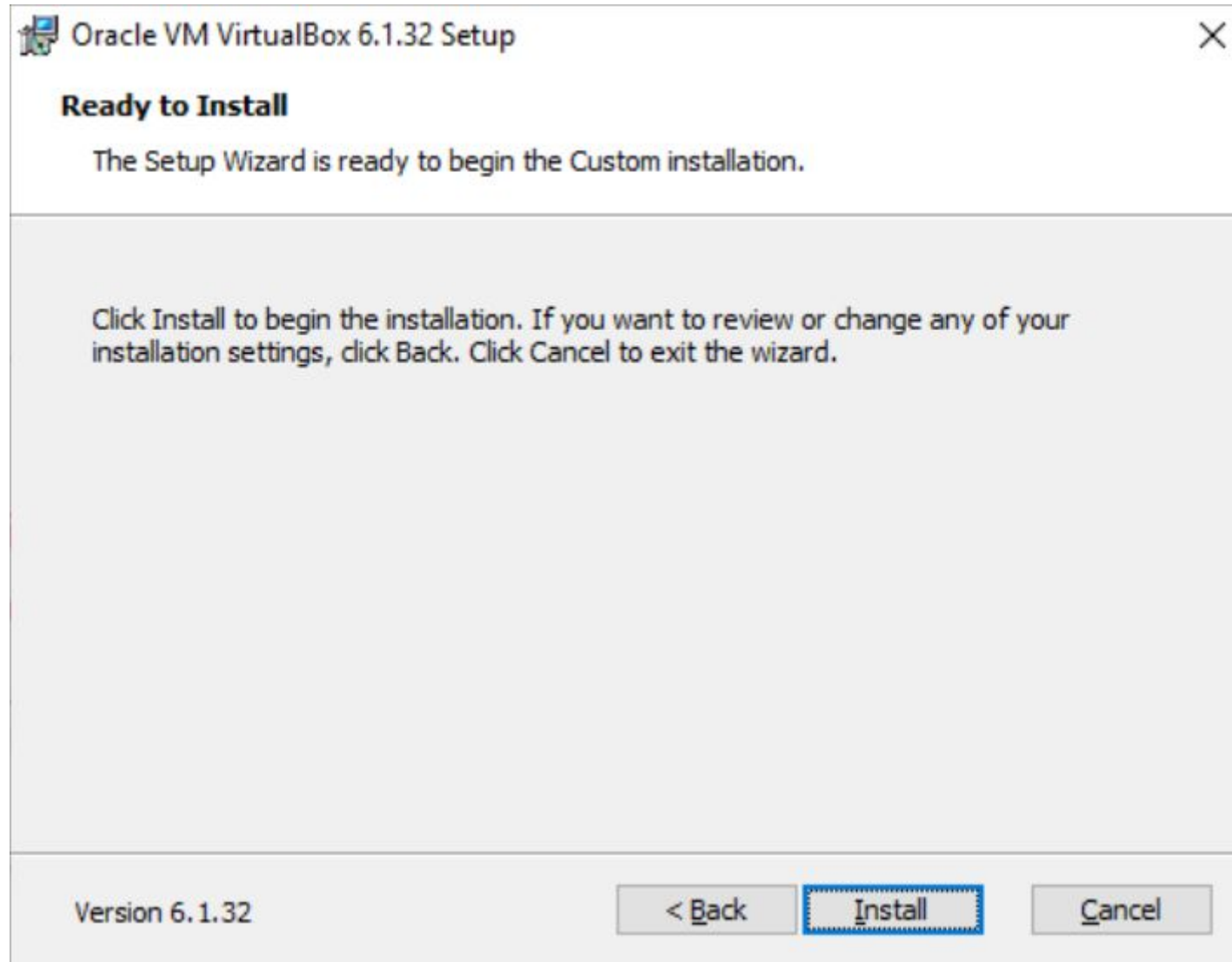


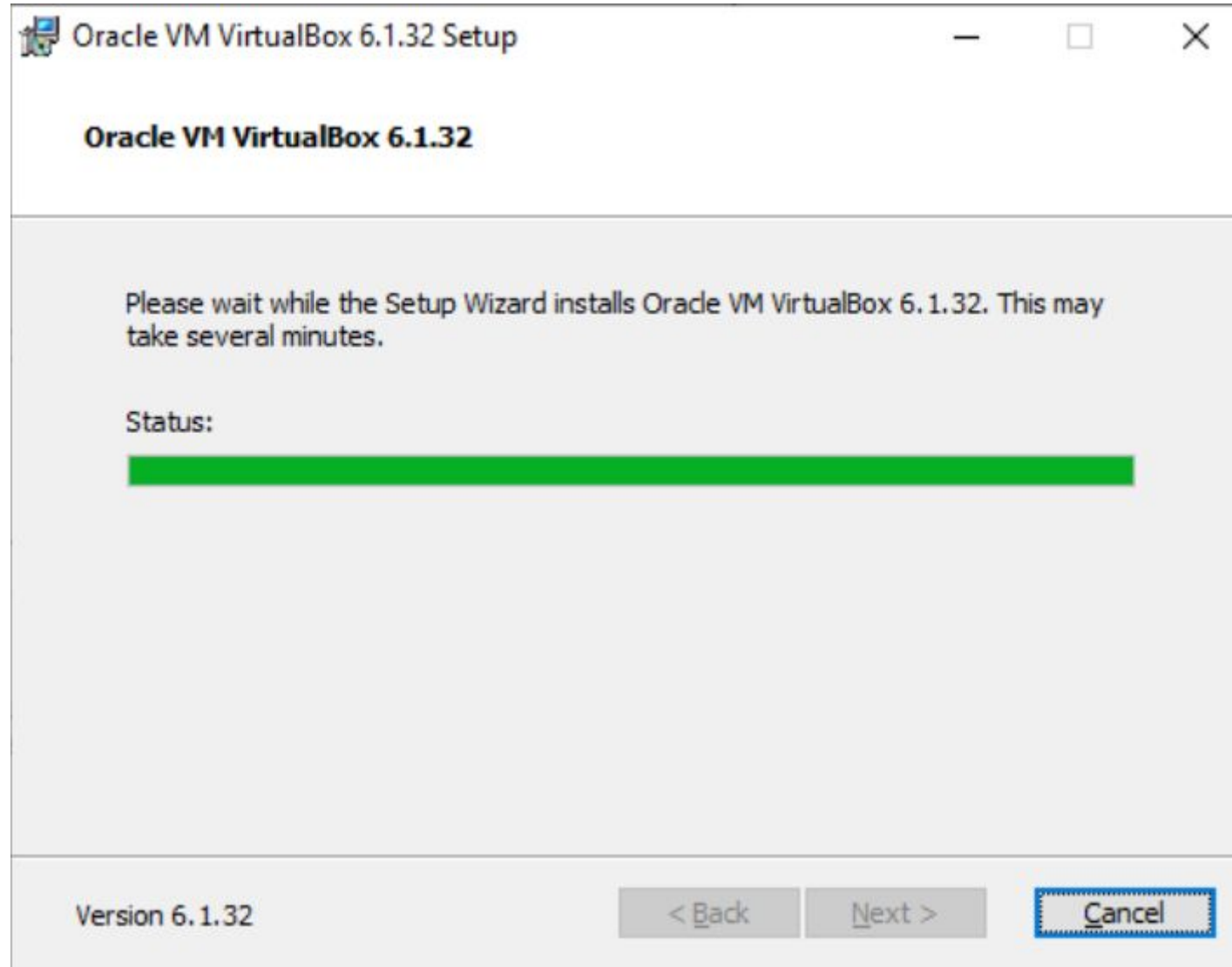
Виртуализация с использованием VirtualBox







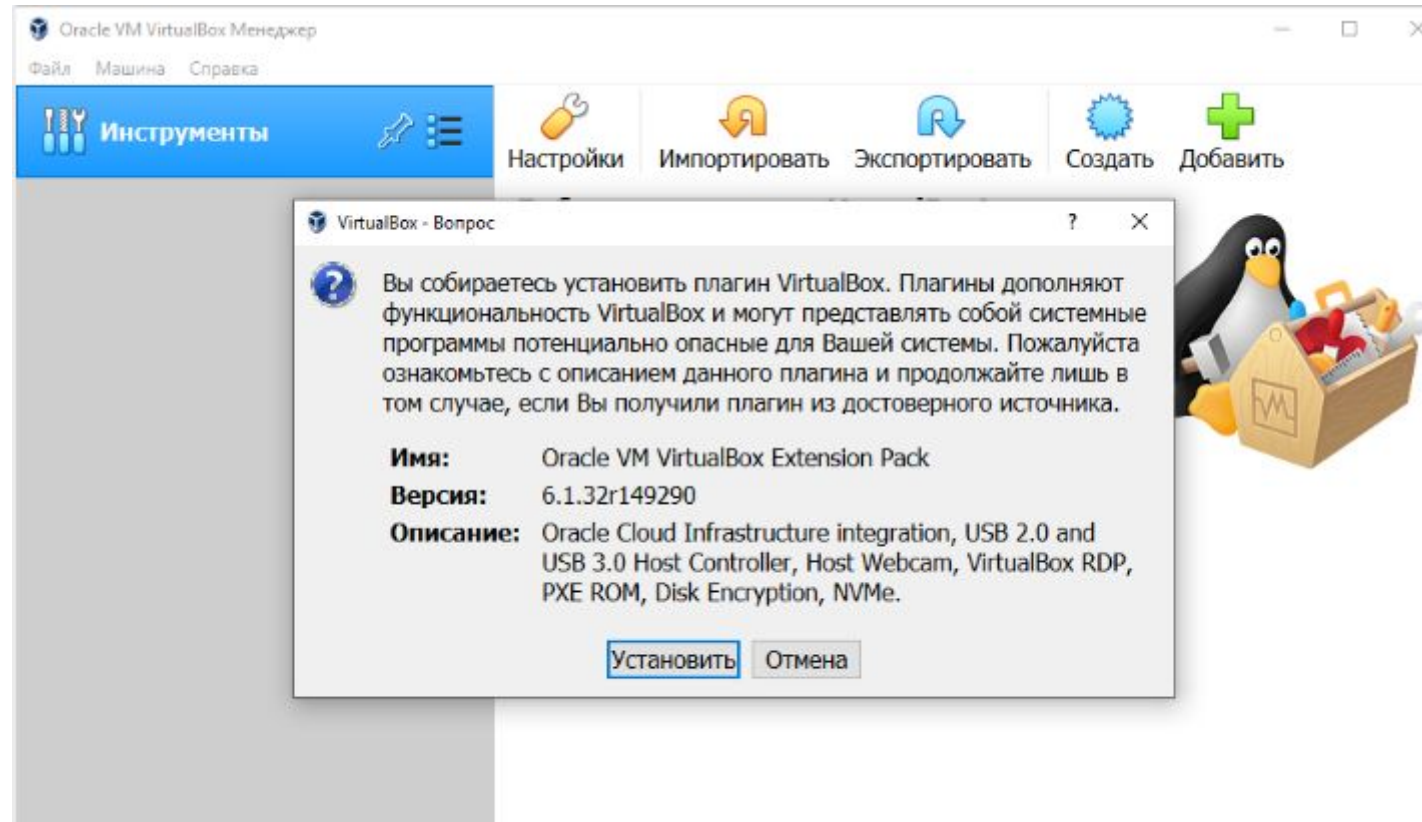


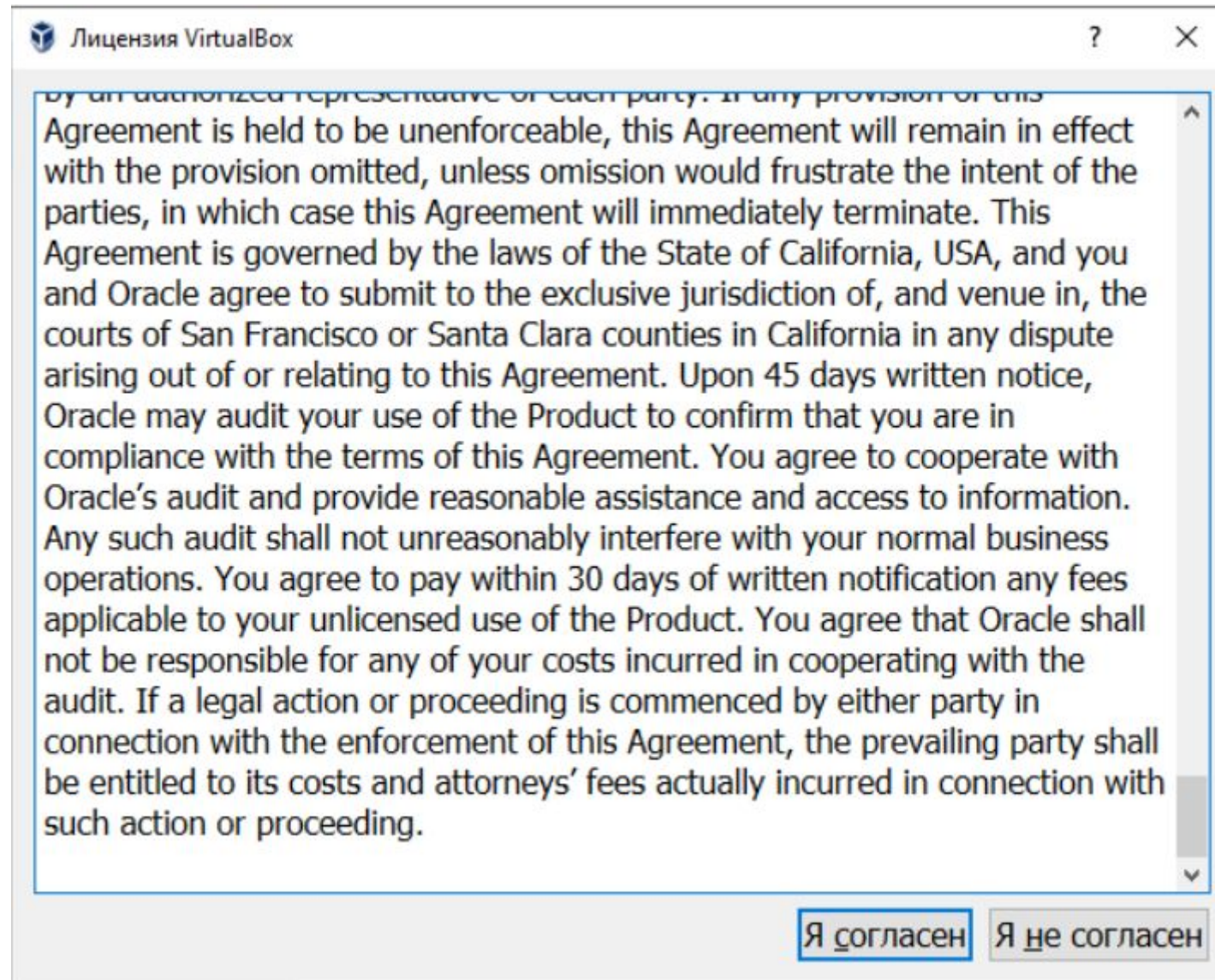


Виртуализация с использованием VirtualBox

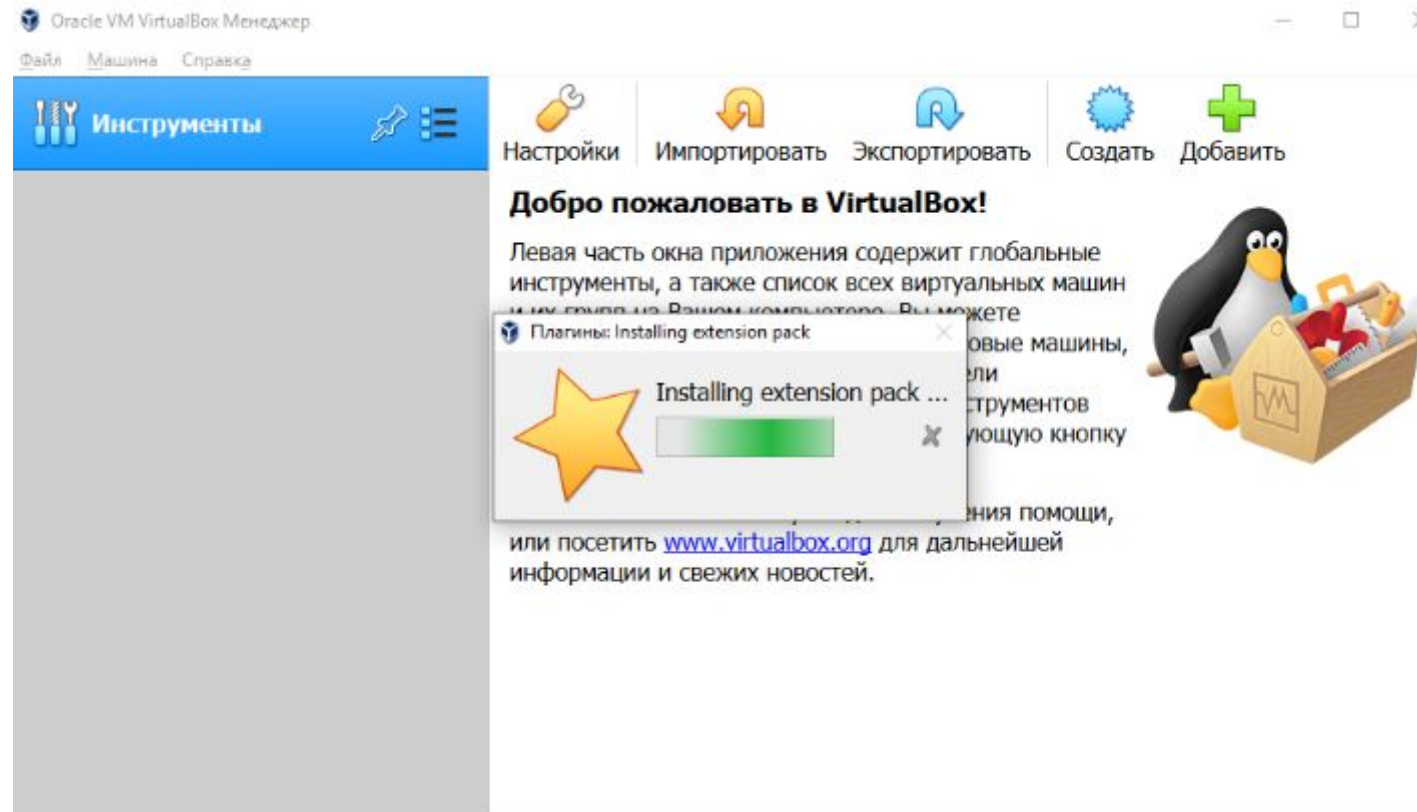


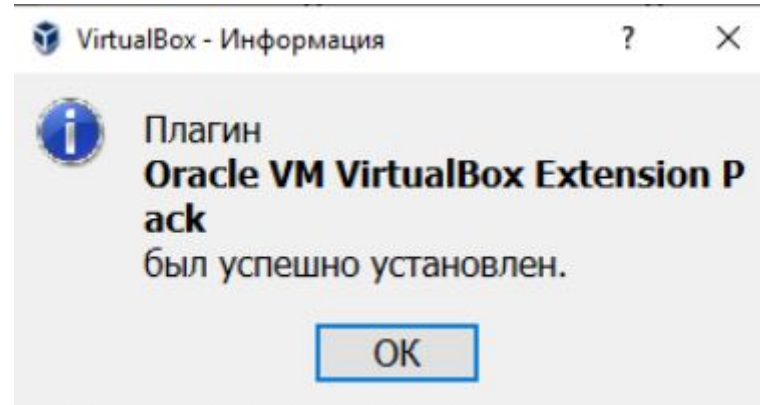
Виртуализация с использованием VirtualBox



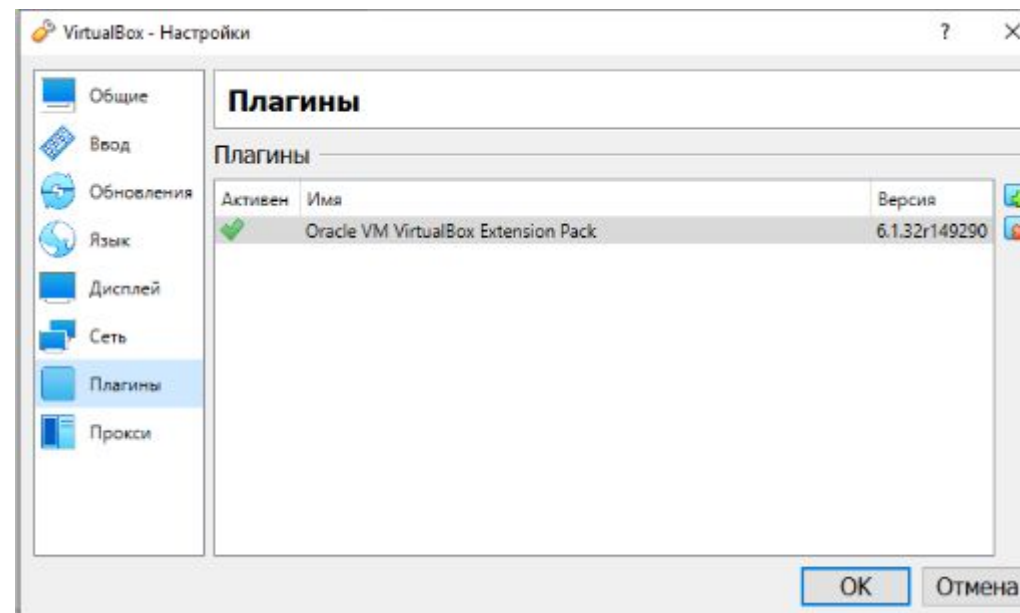


Виртуализация с использованием VirtualBox





После установки в меню **Файл -> Настройки -> Плагины** (или в англоязычном варианте **File -> Preferences -> Extensions**) можно перейти в настройки и убедиться, что **Extension Pack** установлен.

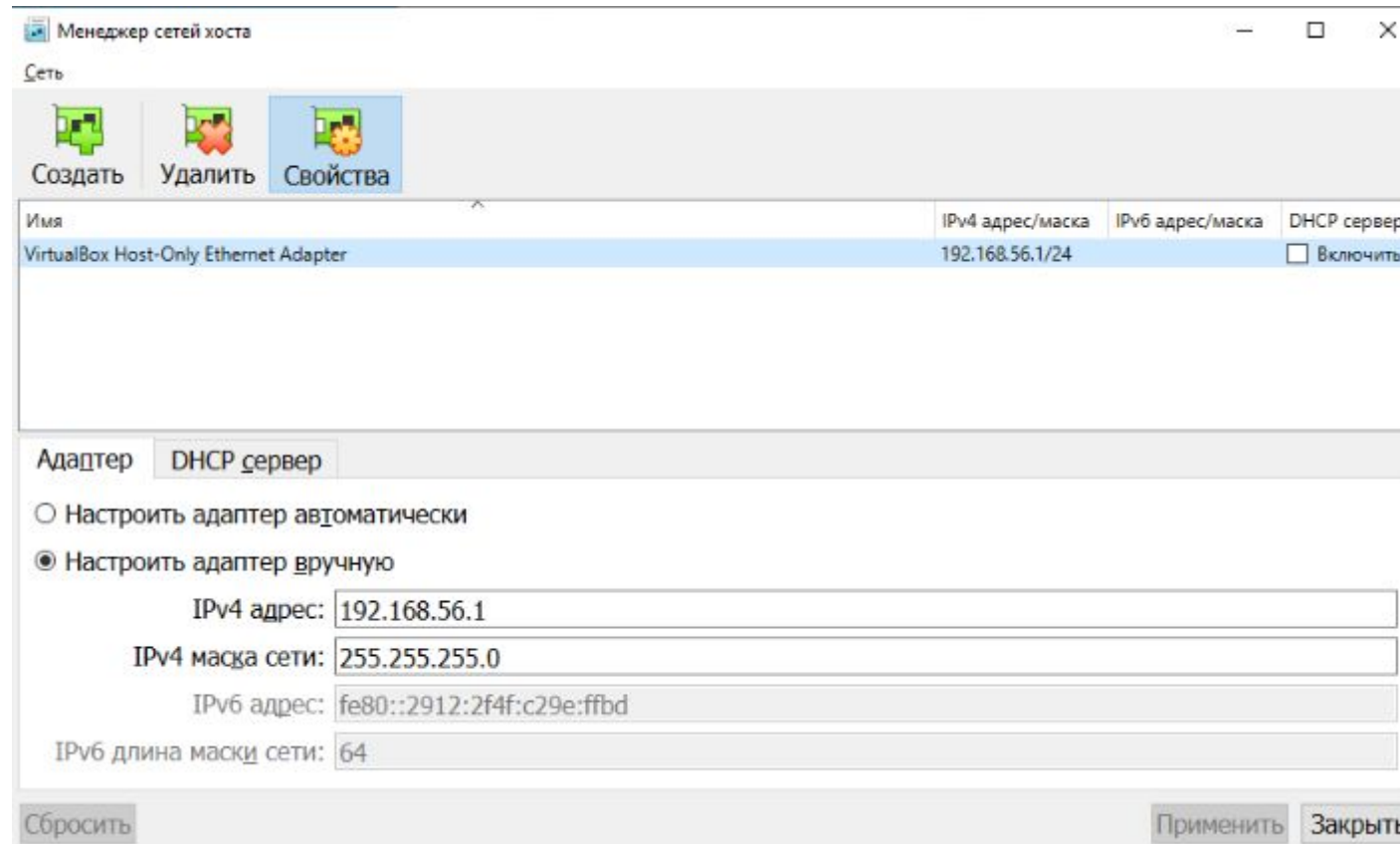


Режим работы сетевого адаптера

NAT (Network Address Translation)	NAT Network	Bridged	Host-only:
позволяет виртуальным машинам получать доступ в интернет через хост, но не позволяет им взаимодействовать друг с другом. IP в этом случае назначается динамически, виртуальные машины доступны из внешней сети, но внешняя сеть видит только хост машину.	то же, что и NAT, но виртуальные машины могут взаимодействовать между собой через внутреннюю сеть.	виртуальная машина имеет свой собственный статический IP адрес и доступна из внешней сети напрямую.	создается одна общая сеть между хостом и всеми виртуальными машинами, каждой из которых можно назначить статический IP адрес, требует создания виртуального сетевого адаптера на хосте.

Виртуализация с использованием VirtualBox

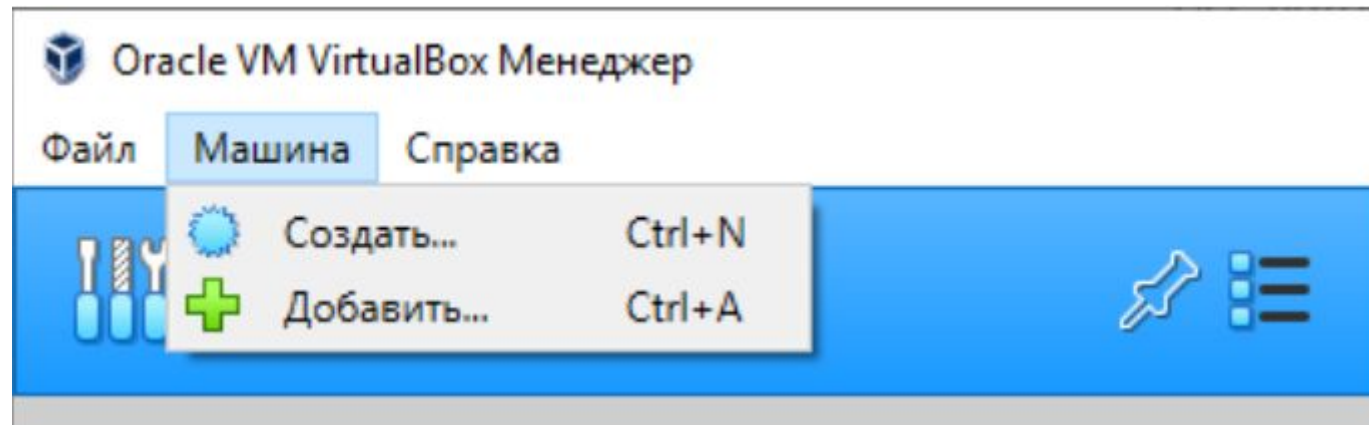
Виртуальный адаптер сети хоста доступен в меню **Файл -> Менеджер сетей хоста**.



Виртуализация с использованием VirtualBox

Давайте создадим шаблонную виртуальную машину

Для этого в пункте меню «Машина» необходимо выбрать раздел «Создать».




После этого можно задать имя виртуальной машины, указать ее тип и версию.

Укажите имя и тип ОС

Пожалуйста укажите имя и местоположение новой виртуальной машины и выберите тип операционной системы, которую Вы собираетесь установить на данную машину. Заданное Вами имя будет использоваться для идентификации данной машины.

Имя:

Папка машины:

Тип: 

Версия:


Экспертный режим

Также можно перейти в расширенные настройки параметров виртуальной машины и изменить такие параметры как объем памяти и способ организации виртуального жесткого диска.

Укажите имя и тип ОС

Имя:

Папка машины:

Тип: 

Версия:

Укажите объем памяти

1024 МБ

4 МБ 16384 МБ

Жесткий диск

Не подключать виртуальный жёсткий диск

Создать новый виртуальный жёсткий диск

Использовать существующий виртуальный жёсткий диск

Виртуализация с использованием VirtualBox

После этого создаем для виртуальной машины новый виртуальный жёсткий диск в формате **VDI (VirtualBox Disk Image)** с форматом хранения «*Динамический виртуальный жесткий диск*» объемом 10 ГБ.

Расположение
C:\Users\ychernyshov\VirtualBox VMs\Ubuntu Template\Ubuntu Template.vdi

Размер
4,00 МБ 10,00 ГБ 2,00 ТБ

Укажите тип

- VDI (VirtualBox Disk Image)**
- VHD (Virtual Hard Disk)**
- VMDK (Virtual Machine Disk)**
- HDD (Parallels Hard Disk)
- QCOW (QEMU Copy-On-Write)
- QED (QEMU enhanced disk)

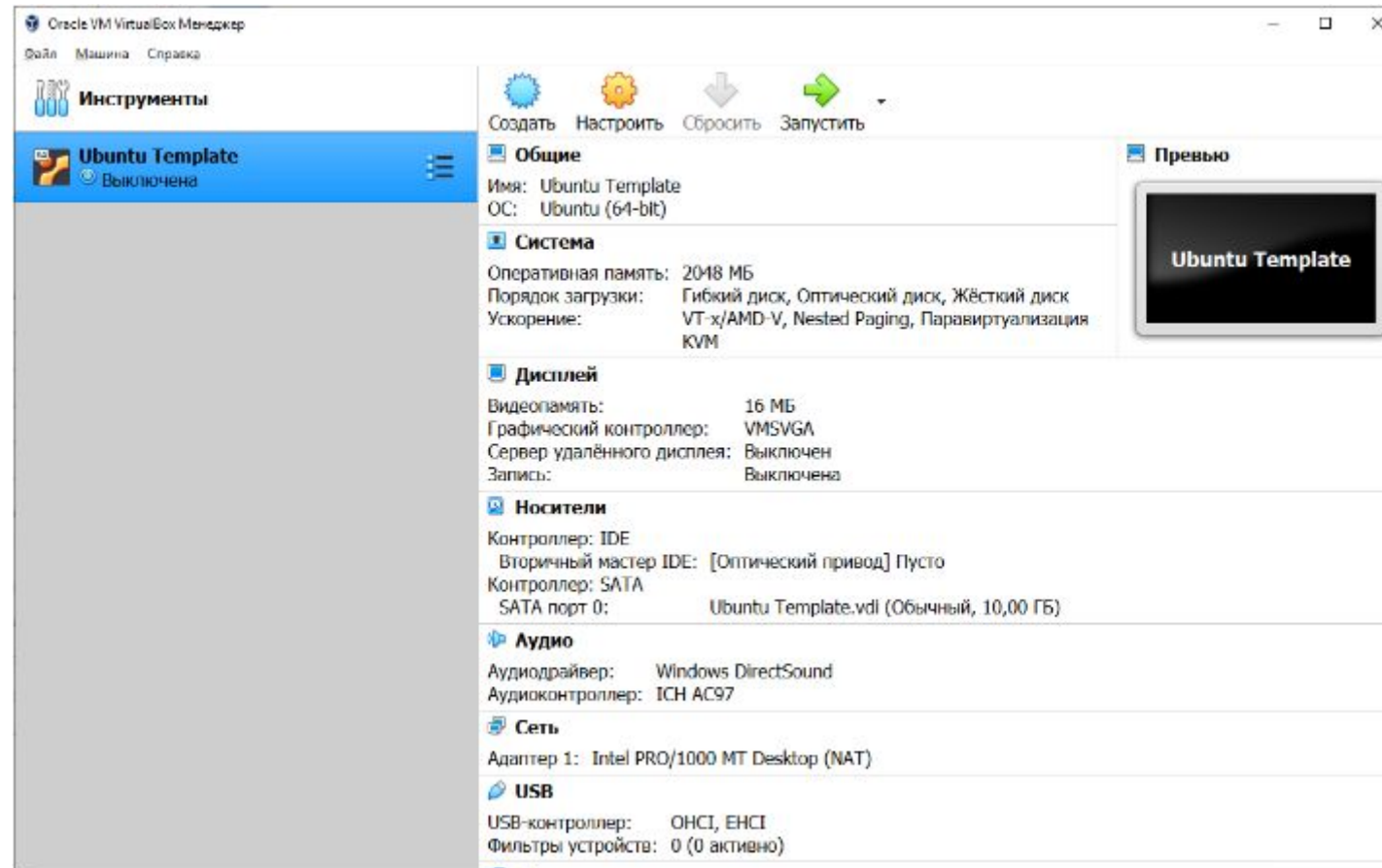
Укажите формат хранения

- Динамический виртуальный жёсткий диск
- Фиксированный виртуальный жёсткий диск
- Разделить на файлы размером до 2х ГБ

Подробный режим **Создать** Отмена

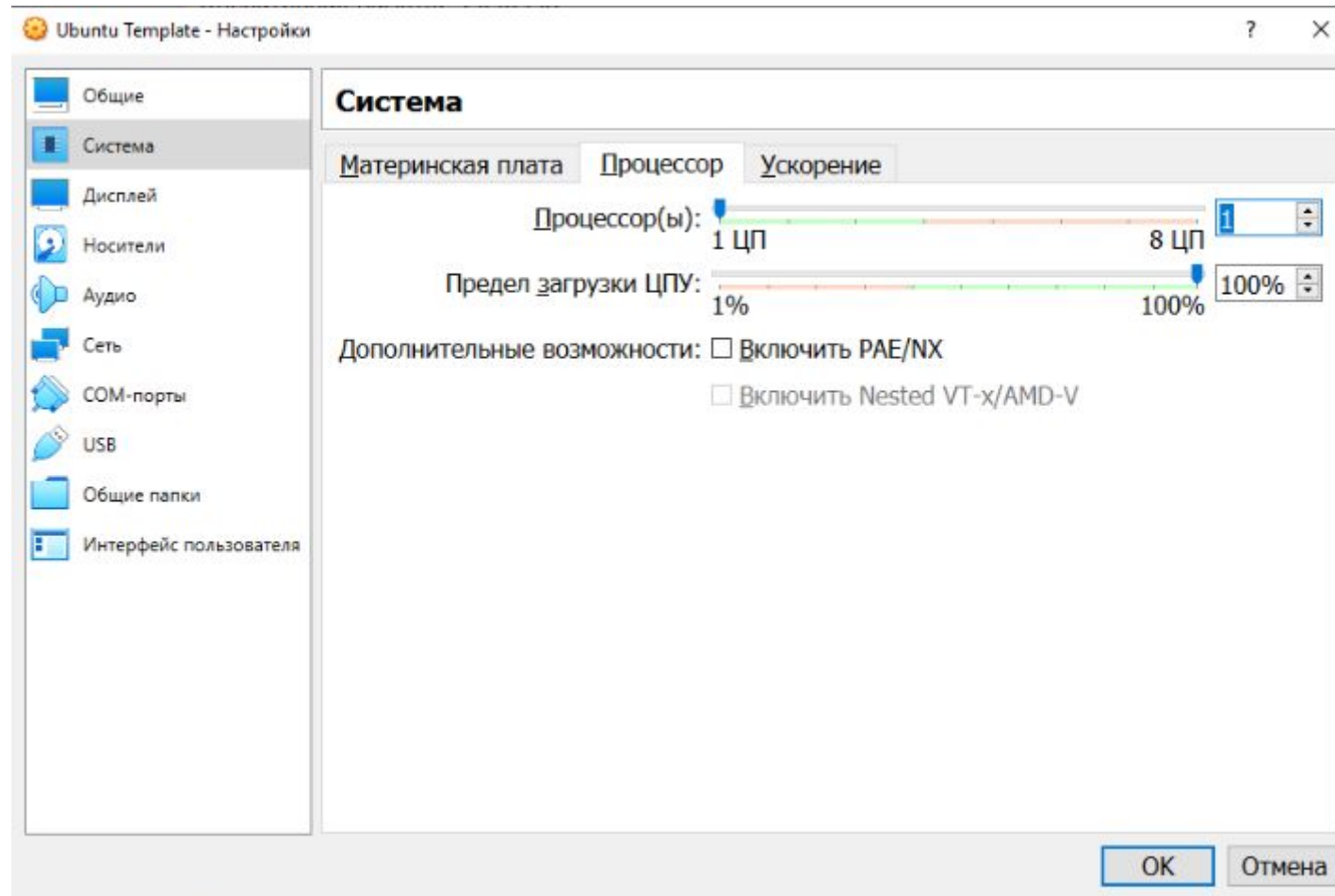
Виртуализация с использованием VirtualBox

После этого наша виртуальная машина
готова.



Виртуализация с использованием VirtualBox

У созданной виртуальной машины можно теперь менять параметры, например, изменить количество процессоров.



Конечно же, виртуальная машина пока не может эксплуатироваться, хотя и обладает уже виртуальными ресурсами — **процессором, оперативной памятью, жестким диском, сетевым адаптером**. Для работы необходимо использовать установочный образ, с которого в виртуальную машину будет загружена гостевая операционная система.

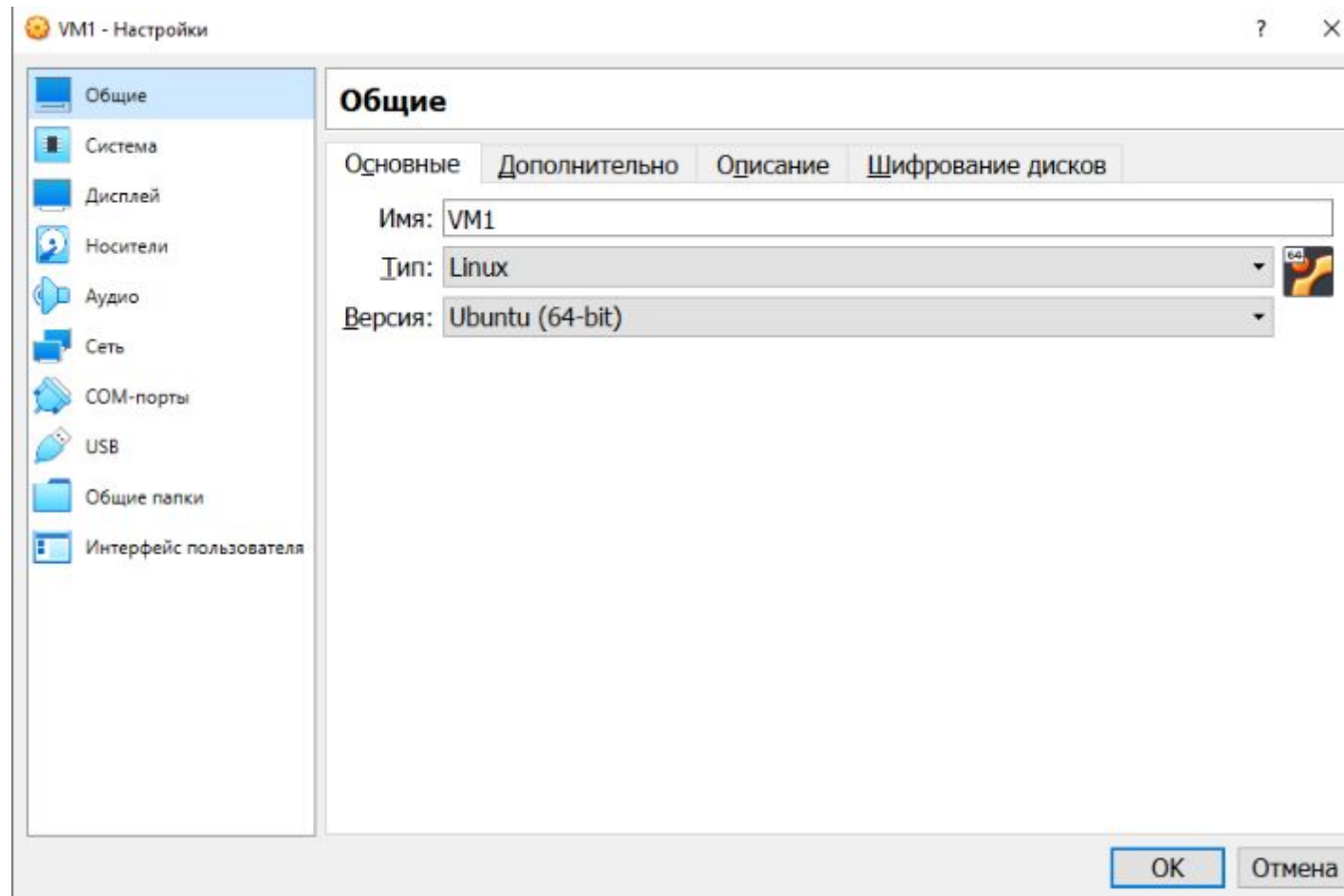
Один из способов указать виртуальной машине откуда брать загрузочные данные — это создать **виртуальный оптический диск** и прикрепить к нему образ операционной системы, скачанный с официального сайта (например, в формате ISO).

Для этого в панели управления виртуальными машинами надо для рабочей виртуальной машины выбрать пункт *«Настроить»*.



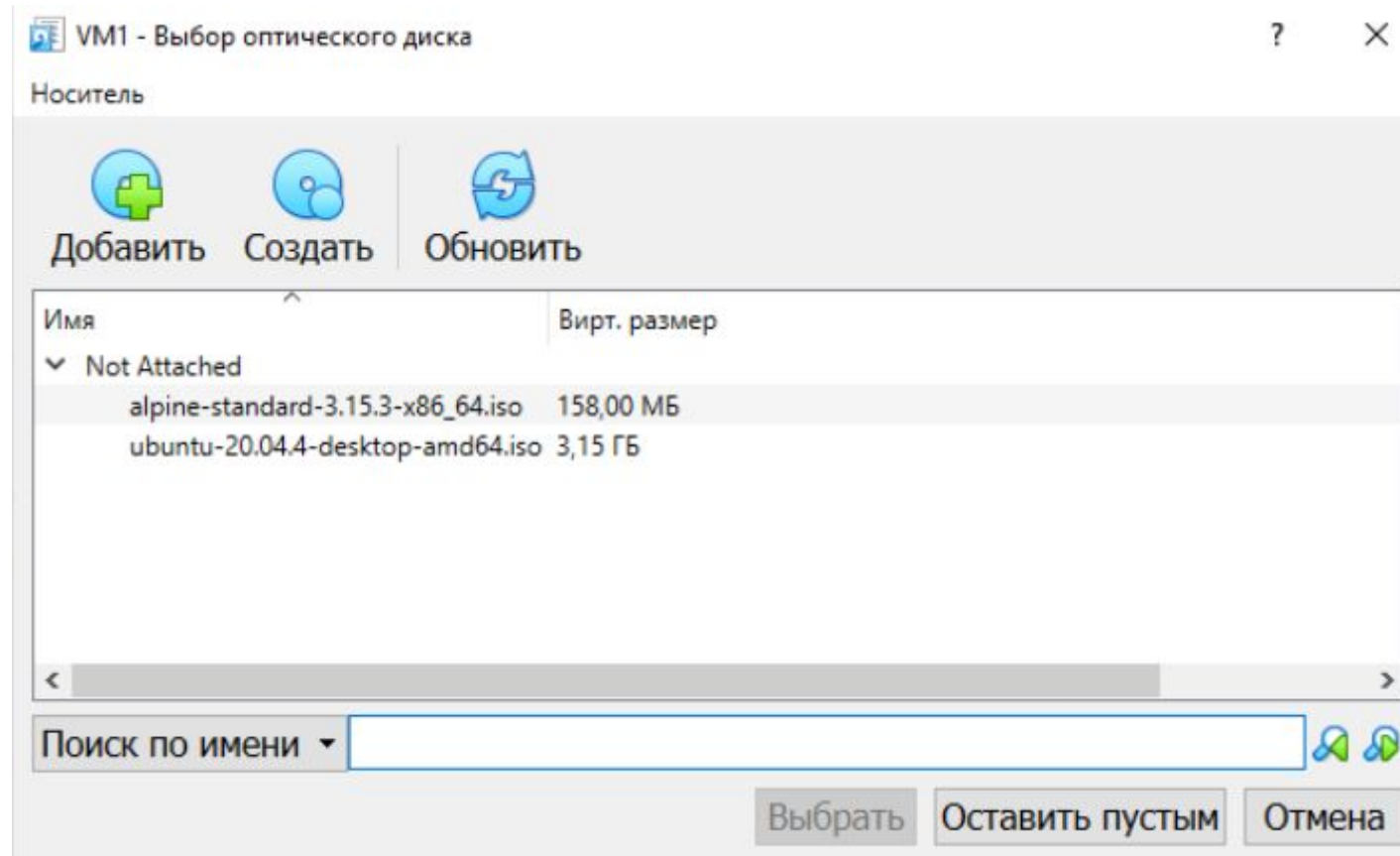
Виртуализация с использованием VirtualBox

После чего появится следующее окно для настроек.



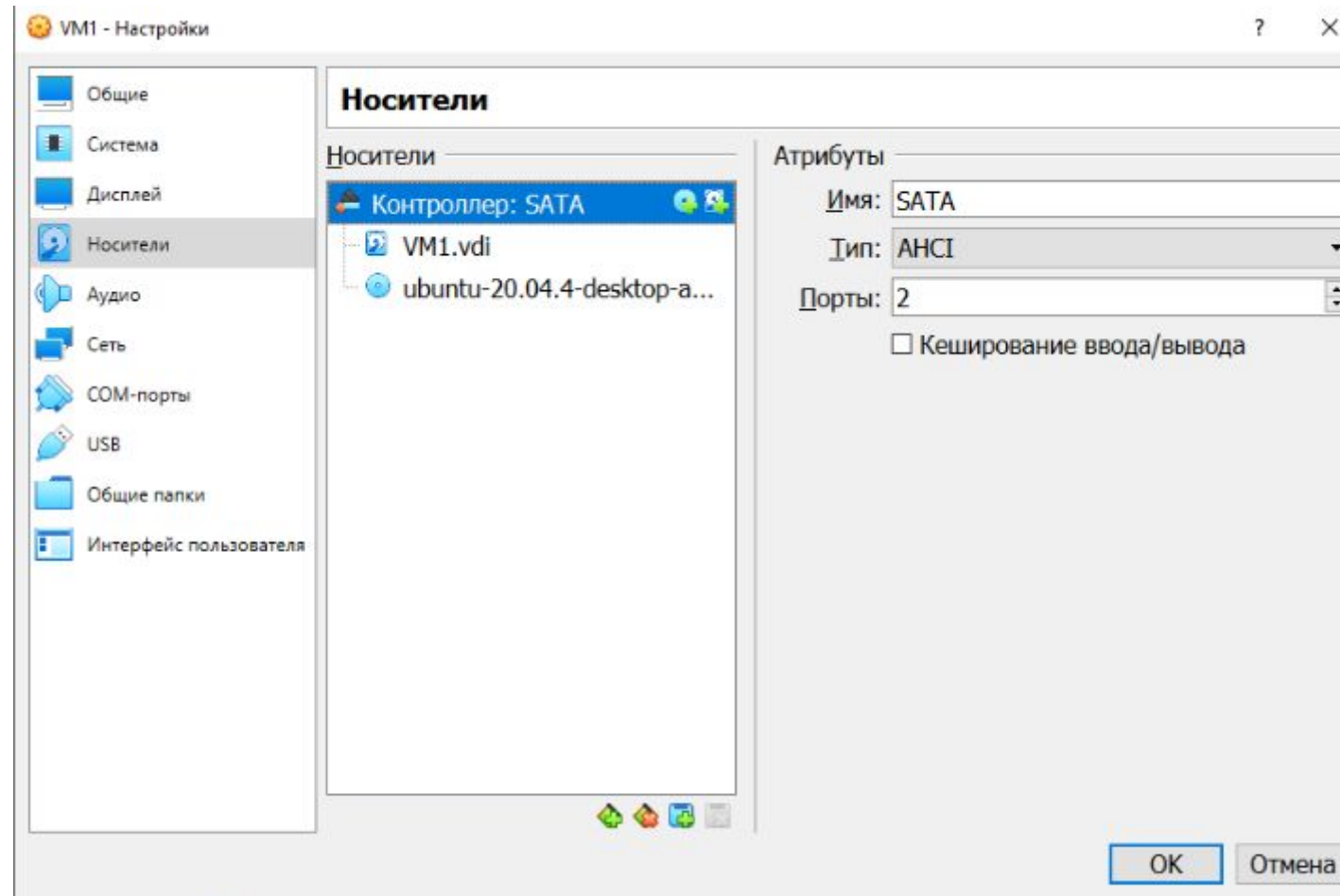
Виртуализация с использованием VirtualBox

Сначала в разделе «Носители» надо добавить оптический диск.



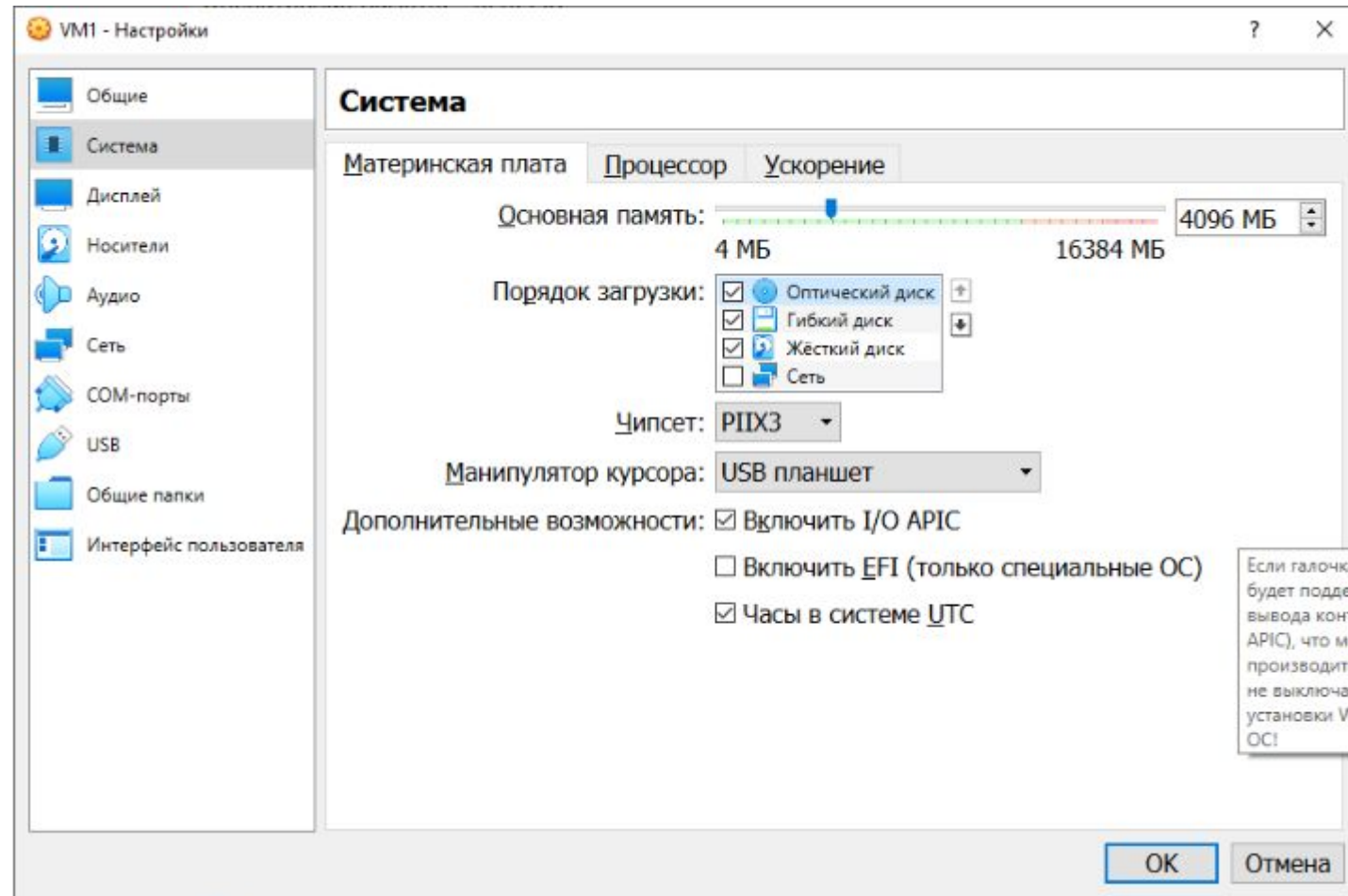
Виртуализация с использованием VirtualBox

Он должен появиться в перечне
ДИСКОВ.



Виртуализация с использованием VirtualBox

После этого в разделе «Система» меняем порядок загрузки, делаем оптический диск первым.



После этого нажимаем
кнопку «Запустить».



и в появившемся окне надо выбрать виртуальный загрузочный
ДИСК.

Пожалуйста выберите виртуальный оптический диск или
физический привод оптических дисков, содержащий диск для
запуска Вашей новой виртуальной машины.

Диск должен быть загрузочным и содержать дистрибутив
операционной системы, которую Вы хотите установить. Диск
будет автоматически извлечён при выключении виртуальной
машины, однако, в случае необходимости, Вы можете сделать
это и сами используя меню Устройства.

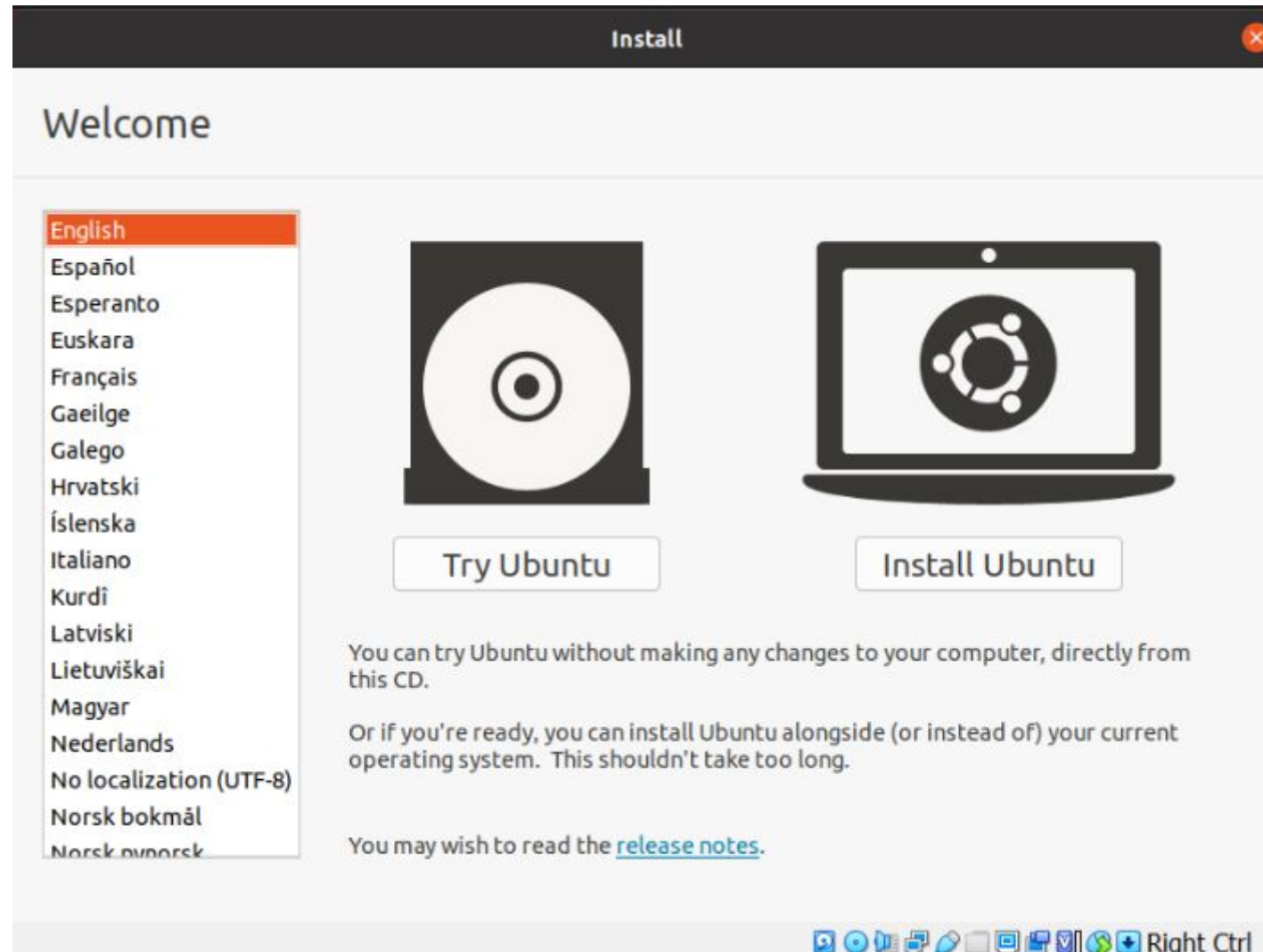
ubuntu-20.04.4-desktop-amd64.iso (3,15 ГБ) 

Продолжить

Отмена

Виртуализация с использованием VirtualBox

После этого система начнет использовать диск с загрузчиком операционной системы, в частности, вы увидите диалог для установки **Ubuntu**.



Теперь у вас на одном компьютере может быть много различных операционных систем, работающих независимо и изолированно друг от друга.

Дополнительно к информации о VirtualBox еще отметим инструмент **Vagrant**, свободное и открытое программное обеспечение для создания и конфигурирования виртуальной среды разработки. Vagrant является оберткой для программного обеспечения виртуализации, например, **VirtualBox**, и средств управления конфигурациями, таких как **Chef**, **Salt** и **Puppet**.

```
sudo apt install virtualbox
```

```
sudo apt install vagrant
```

```
vagrant --version
```

Виртуальное программное окружение отличается от рассмотренных виртуальных машин и контейнеров тем, что не содержит операционную систему

Основное назначение виртуального окружения состоит в **создании изолированной конфигурации программного обеспечения с определенным зафиксированным набором библиотек определенных версий.**

Виртуальное окружение позволяет зафиксировать работающие конфигурации используемых библиотек. При необходимости можно быстро переключиться в необходимое виртуальное окружение и запустить прикладную программу в этом уникальном сочетании библиотек, настроек, конфигураций. Это позволяет быстрее повторить и локализовать проблему.

Вот основные команды при работе с pip в командной строке Windows или в терминале Ubuntu:

Команда	Описание
pip help	Справка по командам
pip search “имя пакета”	Поиск пакета
pip show “имя пакета”	Информация об пакете
pip install “имя пакета”	Установка пакета
pip uninstall “имя пакета”	Удаление пакета
pip list	Список установленных пакетов
pip install -U	Обновление пакета

virtualenv

У
По умолчанию в Ubuntu утилита virtualenv отсутствует, ее необходимо установить.

```
ychernyshov@ychernyshov-VirtualBox:~$ virtualenv  
Command 'virtualenv' not found, but can be installed with:  
sudo apt install python3-virtualenv
```

Установка осуществляется

```
sudo pip install python3-virtualenv
```

После этого можно пользоваться

virtualenv

```
ychernyshov@ychernyshov-VirtualBox:~$ virtualenv --version  
virtualenv 20.0.17 from /usr/lib/python3/dist-packages/virtualenv/__init__.py
```

Виртуальные окружения

Вот основные команды при работе с VirtualEnv в командной строке Windows и в терминале Ubuntu:

<code>mkvirtualenv "имя окружения"</code>	Создать новое виртуальное окружение. Создается папка с именем "имя окружения", содержащая всю необходимую для работы виртуального окружения информацию.
<code>workon</code>	Получить список окружений
<code>workon "имя окружения"</code>	Изменить используемое виртуальное окружение
<code>deactivate</code>	Выйти из виртуального окружения
<code>rmvirtualenv "имя окружения"</code>	Удалить виртуальное окружение

В рабочей папке виртуальной среды при создании формируется следующая структура каталогов

```
ychernyshov@ychernyshov-VirtualBox:~$ tree env1 -L 2
env1
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── activate.ps1
│   ├── activate_this.py
│   ├── activate.xsh
│   ├── easy_install
│   ├── easy_install3
│   ├── easy_install-3.8
│   ├── pip
│   ├── pip3
│   ├── pip-3.8
│   ├── pip3.8
│   ├── python -> /usr/bin/python3
│   ├── python3 -> python
│   ├── python3.8 -> python
│   ├── wheel
│   ├── wheel3
│   └── wheel-3.8
├── lib
│   └── python3.8
└── pyvenv.cfg

3 directories, 20 files
```

Виртуальные окружения

venv

Использование утилиты venv аналогично использованию virtualenv. Создать виртуальное окружение можно следующим образом:

```
python3 -m venv <ИМЯ папки>  
source bin/activate
```

```
ychernyshov@bruteforce:~$ venv env1  
-bash: venv: command not found  
ychernyshov@bruteforce:~$  
ychernyshov@bruteforce:~$  
ychernyshov@bruteforce:~$ python3 -m venv env1  
ychernyshov@bruteforce:~$  
ychernyshov@bruteforce:~$ tree env1 -L 2  
env1  
├── bin  
│   ├── activate  
│   ├── activate.csh  
│   ├── activate.fish  
│   ├── Activate.ps1  
│   ├── easy_install  
│   ├── easy_install-3.9  
│   ├── pip  
│   ├── pip3  
│   ├── pip3.9  
│   ├── python -> python3  
│   ├── python3 -> /usr/bin/python3  
│   └── python3.9 -> python3  
├── include  
├── lib  
│   └── python3.9  
├── lib64 -> lib  
├── pyvenv.cfg  
└── share  
    └── python-wheels  
  
7 directories, 13 files  
ychernyshov@bruteforce:~$ source env1/bin/activate  
(env1) ychernyshov@bruteforce:~$ █
```

Виртуальные окружения

После выполнения скрипта `activate` при успешной активации виртуальной среды вы увидите соответствующий промптер перед курсором. Теперь можно устанавливать требуемые версии библиотек программного обеспечения.

```
ychernyshov@bruteforce:~$ source env1/bin/activate
(env1) ychernyshov@bruteforce:~$
(env1) ychernyshov@bruteforce:~$ pip freeze | grep numpy
(env1) ychernyshov@bruteforce:~$
(env1) ychernyshov@bruteforce:~$ pip install numpy
Collecting numpy
  Downloading numpy-1.22.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.8 MB)
    |-----| 16.8 MB 4.7 MB/s
Installing collected packages: numpy
Successfully installed numpy-1.22.3
(env1) ychernyshov@bruteforce:~$
(env1) ychernyshov@bruteforce:~$ pip freeze | grep numpy
numpy==1.22.3
(env1) ychernyshov@bruteforce:~$ █
```

Деактивировать виртуальное окружение можно командой

deactivate

conda

Виртуальная среда создается аналогично предыдущим инструментам, при этом создается папка со всем необходимым содержимым.

```
conda create --name "имя окружения"  
python=3.6
```

Активировать виртуальное окружение можно с помощью команды с консоли:

```
conda activate "имя  
окружения"
```

Далее можно управлять содержанием виртуального окружения, добавлять нужные библиотеки:

```
conda install numpy pandas
```

Виртуальные окружения

Утилиты conda есть много опций, их можно увидеть через справку. Например, можно воспользоваться командой `conda info` для получения подробной информации о текущей конфигурации.

```
ychernyshov@bruteforce:~$ conda info

  active environment : None
  user config file   : /home/ychernyshov/.condarc
  populated config files :
  conda version      : 4.10.1
  conda-build version : 3.21.4
  python version     : 3.8.8.final.0
  virtual packages   : __cuda=10.2=0
                     __linux=5.2.0=0
                     __glibc=2.31=0
                     __unix=0=0
                     __archspec=1-x86_64
  base environment   : /etc/anaconda3 (read only)
  conda av data dir  : /etc/anaconda3/etc/conda
  conda av metadata url : https://repo.anaconda.com/pkgs/main
  channel URLs       : https://repo.anaconda.com/pkgs/main/linux-64
                     https://repo.anaconda.com/pkgs/main/noarch
                     https://repo.anaconda.com/pkgs/r/linux-64
                     https://repo.anaconda.com/pkgs/r/noarch
  package cache      : /etc/anaconda3/pkgs
                     /home/ychernyshov/.conda/pkgs
  envs directories   : /home/ychernyshov/.conda/envs
                     /etc/anaconda3/envs
  platform           : linux-64
  user-agent          : conda/4.10.1 requests/2.25.1 CPython/3.8.8 Linux/5.2.0-kali2-amd64 kali/2020.4 glibc/2.31
  UID:GID            : 1001:1004
  netrc file         : None
  offline mode       : False
```


poetry у

Poetry это хорошая альтернатива **pip**, которая позволяет отказаться от **requirements.txt** в пользу более гибкой настройки проекта. Благодаря **poetry** можно в любой момент посмотреть информацию о зависимостях любого пакета, гибко настраивать версии и обмениваться **poetry.lock** файлами с уже заготовленным списком версий пакетов.

Установка осуществляется с помощью
команды:

```
pip install poetry
```

Главный файл для **poetry** — это **pyproject.toml**. Все данные о проекте должны быть записаны в нём. При установке пакетов **poetry** берёт данные из этого файла и формирует файл с зависимостями **poetry.lock** (если уже есть готовый файл **poetry.lock**, то данные будут браться из него). **Toml-файл** состоит из нескольких блоков, каждый из которых имеет свои особенности, рассмотрим их:

[tool.poetry] — содержит основную информацию о проекте, такую как:

- **name** — имя проекта
- **version** — версия проекта
- **description** — описание проекта
- **license** — лицензия проекта
- **authors** — список авторов проекта в формате name <email>
- **maintainers** — список менторов проекта формате name <email>
- **readme** — readme файл проекта в формате README.rst или README.md
- **homepage** — URL сайта проекта
- **repository** — URL репозитория проекта
- **documentation** — URL документации проекта
- **keywords** — список ключевых слов проекта (макс: 5)
- **classifier** — список PyPI классификаторов

Окружения

[tool.poetry.dependencies] — содержит описание всех зависимостей проекта. Каждая зависимость должна иметь название с указанием версии, также присутствует возможность скачать проект с github с указанием ветки/версии/тэга, например:

- requests = "^2.26.0"
- requests = { git = "https://github.com/requests/requests.git" }
- requests = { git = "https://github.com/kennethreitz/requests.git", branch = "next" }
- numpy = { git = "https://github.com/numpy/numpy.git", tag = "v0.13.2" }

[tool.poetry.scripts] — В данном разделе можно описать различные сценарии или скрипты, которые будут выполняться при установке пакетов или при запуске приложения. Например:

- poetry = 'poetry.console:run'
- main-run = 'new_proj.main:run' (после чего достаточно запустить poetry main-run и будет выполнен запуск функции run в файле new_prof/main.py)

[tool.poetry.extras] — В данном блоке описываются группы зависимостей, которые можно устанавливать отдельно:

[tool.poetry.dependencies]

- psycopg2 = { version = "^2.7", optional = true }
- pymysql = { version = "1.0.2", optional = true }

[tool.poetry.extras]

- mysql = ["pymysql"]
- pgsql = ["psycopg2"]

Далее зависимости можно установить двумя способами:

- poetry install --extras "mysql psycopg2"
- poetry install -E mysql -E psycopg2

[tool.poetry.urls] — помимо основных URL, указанных в [tool.poetry], можно указывать свои URL:

- "Bug Tracker" = "https://github.com/python-poetry/poetry/issues"

Виртуальные окружения

Чтобы создать новый проект с помощью poetry, достаточно выполнить

```
poetry new <название папки с  
проектом>
```

в каталоге, являющемся корнем вашего проекта, в которой будет лежать файл `pyproject.toml`.

```
ychernyshov@ychernyshov-VirtualBox:~$ python3 -m poetry new env3
Created package env3 in env3
ychernyshov@ychernyshov-VirtualBox:~$ tree env3 -L 2
env3
├── env3
│   └── __init__.py
├── pyproject.toml
├── README.rst
└── tests
    ├── __init__.py
    └── test_env3.py

2 directories, 5 files
ychernyshov@ychernyshov-VirtualBox:~$
```

Чтобы установить зависимости проекта достаточно выполнить команду:

```
poetry install
```

Чтобы добавить новую библиотеку достаточно выполнить:

```
poetry add numpy
```

Чтобы удалить зависимость достаточно
выполнить:

```
poetry remove numpy
```

Чтобы посмотреть зависимости проекта достаточно
выполнить:

```
poetry show
```

Также poetry содержит другие команды, с которыми можно ознакомиться в документации.

Контейнеризация приложения — это упаковка приложения в отдельный контейнер, специальную среду с операционной системой и всеми необходимыми библиотеками, связями, зависимостями.

Контейнеризация приложений очень популярна в разработке программного обеспечения и практических задачах:

- организация разработки и тестирования,
- развертывание инфраструктуры распределенных систем,
- эксплуатация.

Технология **Docker** предназначена для разработки, установки и запуска в специальных сущностях — **контейнерах**. С использованием инструментов Docker контейнеризуются программные продукты, информационные системы, отдельные приложения или масштабные системы со сложной архитектурой, состоящие из множества сервисов. Так реализуется **концепция микросервисной архитектуры**.

Различия контейнеризации и виртуализации

контейнеризация	виртуализация
экономит ресурсы и выполнять задачи быстрее	обеспечивают полный уровень изоляции гостевых операционных систем, однако на это расходуется много ресурсов
Docker работает с низкоуровневыми инструментами основной операционной системы	виртуальная машина взаимодействует напрямую с аппаратным обеспечением

Давайте познакомимся с ключевыми понятиями и терминами docker

Docker Платформа (Docker Platform)

Это программный комплекс, который упаковывает приложения в контейнеры, запускает контейнеры в аппаратных средах (серверах), управляет логикой работы контейнеров, обеспечивает работу пользователя в системе. Платформа Docker позволяет помещать в контейнеры код и его зависимости (используемые внешние библиотеки, переменные среды окружения, служебные файлы, параметры). При таком подходе упрощается запуск, перенос, воспроизведение, масштабирование систем.

Docker «Движок» (Docker Engine)

Это клиент-серверное приложение, обеспечивающее весь цикл работы с технологией Docker. Docker Engine может использоваться в одном из двух вариантов:

- 1. Docker Community Edition** — это бесплатное ПО, основанное на инструментах open-source,
- 2. Docker Enterprise** — платное программное обеспечение, предназначенное для использования производственными компаниями в больших коммерческих проектах.

Docker Клиент (Docker Client)

Это основной инструмент пользователя при работе с Docker. Взаимодействие осуществляется с использованием командной строки **Docker CLI (Docker Command Line Interface)**. В Docker CLI пользователь вводит команды, начинающиеся с ключевого слова «docker», эти команды обрабатываются Docker Клиентом и с использованием API Docker отправляются Docker Демону (Docker Daemon).

Docker Образ (Docker Image)

Это набор данных, содержащий:

- образ базовой операционной системы (файловая система, системные настройки, драйверы устройств),
- прикладное программное обеспечение для развертывания в базовой операционной системе, с настройками,
- библиотеки, служебные файлы.

Docker образ используется для создания **Docker Контейнера (Docker Container)**. Различают базовые и дочерние образы:

- *Base images (базовые образы)* не имеют родительского образа. Обычно это образы с операционной системой, такие как ubuntu, busybox или debian.
- *Child images (дочерние образы)* построены на базовых образах и обладают дополнительной функциональностью.

Существуют официальные и пользовательские образы (любые из них могут быть базовыми и дочерними):

- *Официальные образы* официально поддерживаются компанией Docker. Обычно в их названии одно слово (например, python, ubuntu).
- *Пользовательские образы* создаются пользователями и построены на базовых образах. Формат имени пользовательского образа «имя пользователя»/«имя образа».

Docker Контейнер (Docker Container)

Это запускаемый экземпляр Docker Образа. В контейнере запускаются приложения со всеми требуемыми настройками и зависимостями. Контейнер разделяет имеющиеся ресурсы на уровне ядра операционной системы с другими контейнерами, работает изолированно от других контейнеров в своей операционной системе (hosted OS).

Docker Демон (Docker Daemon)

Это сервис, запущенный в фоновом режиме, предназначенный для управления образами, контейнерами, сетями и томами. Взаимодействие с Docker Демоном осуществляется через запросы к API Docker.

Docker Реестр, Docker хаб (Docker Hub)

Это место хранения образов Docker, облачное хранилище. Многие провайдеры услуг хостинга предоставляют возможность хранить образы Docker в своих репозиториях (например, Amazon, Yandex). Самым популярным и наиболее используемым хранилищем является официальный репозиторий [Docker Hub](https://hub.docker.com/), используемый при работе с Docker по умолчанию. Обычно в репозиториях хранятся разные версии одних и тех же образов, обладающих одинаковыми именами и разными тегами (идентификаторами образов), разделенные двоеточием. Например,

- «python» — официальный репозиторий Python на Docker Hub,
- «python:3.7-slim» — версия образа с тегом «3.7-slim» в репозитории Python.

В реестр можно отправить как целый репозиторий, так и отдельный образ

Файл Dockerfile

Это текстовый файл, содержащий упорядоченный перечень команд, необходимых при создании (building) Docker образа (Docker image). Этот файл содержит описание базового образа, который будет представлять собой исходный слой образа. Популярные официальные базовые образы:

- [alpine](#) — легкая ОС linux, оптимальна для простых задач или обучения, для большинства задач требует дополнительной установки пакетов;
- [ubuntu](#) — ОС linux с большим набором библиотек и утилит;
- [nginx](#) — самый популярный и очень функциональный web сервер;
- [python](#) — ОС linux с установленным программным обеспечением для работы с Python.

Docker Том (Docker Volume)

Это специальный уровень Docker контейнера, который позволяет передавать данные в Docker контейнер. Docker Том это наиболее предпочтительный механизм постоянного хранения данных, используемых или создаваемых приложениями. Здесь могут храниться таблицы базы данных, конфигурационные файлы, изображения, html файлы, сохраненные модели в формате pickle и т.п.

Сетевые механизмы Docker (Docker Networking)

Организуют связь между контейнерами Docker. Соединённые с помощью Docker Сети (Docker Network) контейнеры могут выполняться на одном и том же хосте или на разных хостах, взаимодействуя между собой как отдельные независимые сервисы.

Docker Compose

Это инструмент экосистемы Docker, упрощающий работу с многоконтейнерными приложениями. Docker Compose выполняет инструкции, описанные в файле docker-compose.yml.

Установк

Давайте рассмотрим процедуру установки docker для операционной системы Ubuntu. Установка docker для других операционных систем [подробно описана здесь](#).

Для установки docker для операционной системы Ubuntu необходимо выбрать [на этой странице](#) вариант «*Docker for Linux*». После этого переходим к разделу «*Server*». Здесь описаны варианты установки для разных версий операционных систем.

The screenshot shows the Docker documentation website. The main heading is "Get Docker". Below the heading, there is a yellow banner with a warning icon and the text "Update to the Docker Desktop terms". Below this, there is a paragraph explaining 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." Below this paragraph, there is a sentence: "You can download and install Docker on multiple platforms. Refer to the following section and choose the best installation path for you." At the bottom, there are three cards for different operating systems: "Docker Desktop for Mac", "Docker Desktop for Windows", and "Docker for Linux". Each card has an icon (Apple, Windows, and Linux penguin respectively) and a brief description of the installation method.

Основы контейнеризации с docker. Установка и настройка

Server

Docker provides `.deb` and `.rpm` packages from the following Linux distributions and architectures:

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	s390x
CentOS	✓	✓		
Debian	✓	✓	✓	
Fedora	✓	✓		
Raspbian			✓	
RHEL				✓
SLES				✓
Ubuntu	✓	✓	✓	✓
Binaries	✓	✓	✓	

Основы контейнеризации с docker. Установка и настройка

Поскольку мы решили устанавливать docker для операционной системы Ubuntu, необходимо выбрать соответствующий пункт таблицы после чего появится инструкция по установке.

В linux вы можете получить информацию о характеристиках операционной системы с помощью команды:

```
sudo uname -a
```

После того как мы убедились в правильности версии операционной системы можно переходить к следующим шагам установки.

Сначала необходимо удалить старые версии docker, установленные в системе:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Затем обновить установщик

```
sudo apt-get update
```

и установить необходимые пакеты для возможности использования HTTPS для установки:

```
$ sudo apt-get install ca-certificates curl gnupg lsb-release
```


Основы контейнеризации с docker. Установка и настройка

После этого добавляется ключ для взаимодействия с репозиторием docker по ssh:

```
$curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

и устанавливаются необходимые для этого настройки:

```
$echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Для непосвященных это может казаться магией, но эти действия описаны в официальной инструкции docker по установке. При желании вы можете глубже погрузиться в этот синтаксис и разобрать действия описанных выше команд.

Основы контейнеризации с docker. Установка и настройка

Теперь у нас все готово для установки, для этого еще раз обновим установщик **apt-get** и запустим установку с помощью команды **apt-get install**:

```
$sudo apt-get update
```

```
$sudo apt-get install docker-ce docker-ce-cli containerd.io
```

После этого у нас docker установлен, в этом можно убедиться, выполнив

```
$docker
```

```
$ docker
Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

options:
  --config string      Location of client config files (default "/home/ychernyshov/.docker")
  -c, --context string  Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and global settings)
  -D, --debug           Enable debug mode
  -H, --host list      Daemon socket(s) to connect to
  -l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string   Trust certs signed only by this CA (default "/home/ychernyshov/.docker/ca.pem")
  --tlscert string     Path to TLS certificate file (default "/home/ychernyshov/.docker/cert.pem")
  --tlskey string      Path to TLS key file (default "/home/ychernyshov/.docker/key.pem")
  --tlsverify          Use TLS and verify the remote
  -v, --version        Print version information and quit

Management Commands:
  app*      Docker App (Docker Inc., v0.9.1-beta3)
  builder   Manage builds
  buildx*   Build with BuildKit (Docker Inc., v0.5.1-docker)
  config    Manage Docker configs
  container Manage containers
  context   Manage contexts
  image     Manage images
  manifest  Manage Docker image manifests and manifest lists
  network   Manage networks
  node      Manage Swarm nodes
  plugin    Manage plugins
  secret    Manage Docker secrets
```

Основы контейнеризации с docker. Установка и настройка

Однако при попытке воспользоваться возможностями docker на данном этапе вы потерпите неудачу, которая будет выглядеть вот так:

```
/home/data$ docker run hello-world
docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?.
See 'docker run --help'.
```

Проблема заключается в том, что обычный пользователь по умолчанию не имеет доступа к служебному сокету **unix:///var/run/docker.sock** через который происходит взаимодействие. По умолчанию запуск команд docker требует прав суперпользователя, и команды должны запускаться в формате **“sudo docker ...”**. Для корректной работы необходимо настроить linux-группы, имеющие специальные права доступа к служебным ресурсам Docker.

После установки docker в операционной системе уже существует специальная группа docker, в которую можно добавлять пользователей. Проверить наличие группы можно в файле **/etc/group**. Если группы нет, то ее нужно создать:

```
sudo groupadd docker
```

Добавить пользователя в группу:

```
sudo usermod -aG docker user
```

Проверить какие пользователи входят в группу docker:

```
sudo members docker
```

Удалить пользователя из группы:

```
sudo gpasswd -d user group  
sudo deluser user group
```

Для применения настроек пользователя необходимо зайти в систему. После того как группа создана, и необходимый пользователь в нее добавлен, от имени этого пользователя можно выполнять команды docker.

Основы контейнеризации с docker. Установка и настройка

```
$ 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.

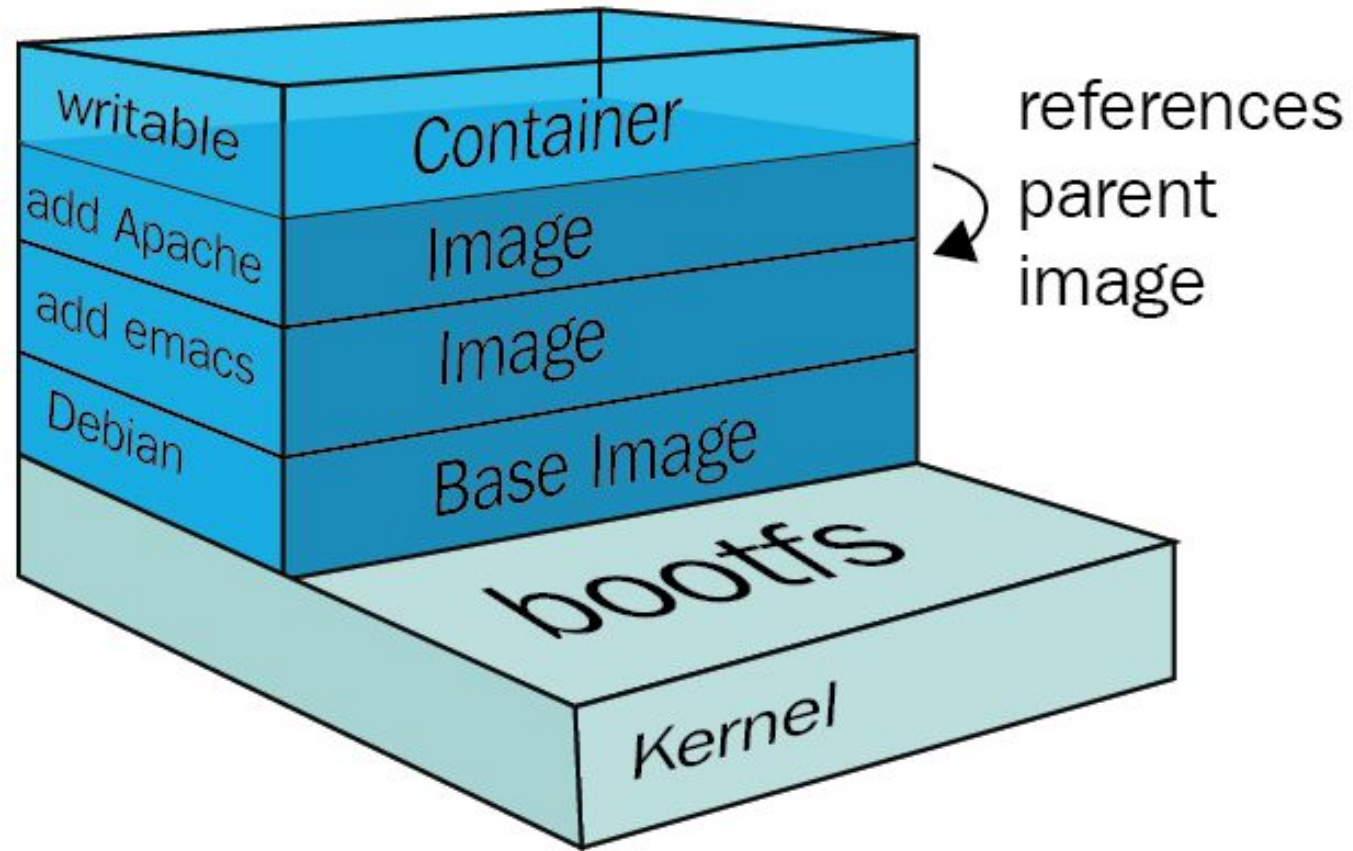
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

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/
```

При выполнении команды `docker run hello-world` выполняется запрос docker-образа `hello-world:latest` в реестре Docker. Этот образ загружается на локальный Docker Engine, запускается в контейнере, в результате выполнения выводится сообщение «Hello from Docker!».

Основы контейнеризации с docker. Установка и настройка



Hello World: What Happened?



Базовые команды docker

<code>docker --version</code>	Используемая версия docker в сжатом формате
<code>docker version</code>	Расширенная информация о версиях различных компонентов docker
<code>docker info</code>	Информация о текущем состоянии docker, служебная информация, статистика, настройки
<code>docker container run hello-world</code>	Проверка работы docker («hello, world» для docker)
<code>docker ps -a</code>	Посмотреть информацию о контейнерах
<code>docker rm <Container ID></code>	Удалить контейнер
<code>docker stop <Container ID></code>	Остановить docker контейнер
<code>docker network</code>	Работа с docker сетями
<code>docker image pull alpine</code>	Загрузить docker образ (docker image) операционной системы alpine

Базовые команды docker

Базовые команды docker

<code>docker image ls</code>	Посмотреть имеющиеся в локальной системе пользователя docker образы (docker image)
<code>docker container ls -a</code>	Посмотреть все имеющиеся docker контейнеры (docker container)
<code>docker container run alpine ls -l</code>	Запустить команду «ls -l» в docker контейнере с docker образом операционной системы alpine
<code>docker container run -it alpine sh</code>	Запустить docker контейнер с docker образом с операционной системой alpine в интерактивном режиме и открыть терминал sh
<code>docker container start <ContainerId></code>	Запустить docker контейнер с идентификатором <ContainerId>
<code>docker container exec <ContainerId> <Command></code>	Выполнить команду <Command> в docker контейнере с идентификатором <ContainerId>
<code>docker container diff <ContainerId></code>	Посмотреть изменения, сделанные в docker контейнере с идентификатором <ContainerId>

Базовые команды docker

```
docker run -d -P --name <имя контейнера> <имя образа>
```

Запустить образ в контейнере в фоновом режиме (флаг `-d`, `detached mode`), все внутренние порты контейнера сделать внешними открытыми и случайными (флаг `-P`), внутренние (т.е. относящиеся к приложению в контейнере) и внешние (т.е. те, через которые контейнер общается в внешней среде) порты связываются.

```
docker ps -a -q
```

Вывести идентификаторы всех существующих контейнеров

```
docker rm $(docker ps -a -q -f status=exited)
```

Удалить все контейнеры, находящиеся в статусе `Exited`

```
docker port <Container ID>
```

Посмотреть порты, относящиеся к контейнеру

```
docker image build -t image_name:v1 .
```

Создать новый образ с тэгом `image_name:v1`. Последний параметр (точка, «.») указывает на то, что действия происходят в текущей директории (в которой должен находиться

Базовые команды docker

Базовые команды docker

<code>docker container commit <ContainerId></code>	Создать новый образ на основе измененного контейнера
<code>docker image tag <ContainerId> <Image Name></code>	Создать тег для образа
<code>docker image history <ContainerId></code>	Посмотреть историю образа
<code>docker image inspect <ImageName></code>	Просмотр детализированной информации об образе
<code>docker image inspect --format "{{ json .Os }}" alpine</code>	Просмотр детализированной информации об образе
<code>docker image inspect --format "{{ json .RootFS.Layers }}" alpine</code>	Просмотр детализированной информации об образе
<code>docker push <Image name></code>	Загрузить образ в docker репозиторий
<code>docker network create <имя сети></code>	Создать сеть docker для обмена сообщениями между контейнерами
<code>docker run -n <имя сети> <имя контейнера></code>	Запустить контейнер с привязкой к сети

Создание образа

Новый Docker образ (Docker Image) создается командой **docker build** с использованием инструкций в **Dockerfile**, при этом подразумевается, что Dockerfile находится в текущей рабочей директории. Если Dockerfile находится в другом месте, то его на расположение нужно указать с использованием флага **-f**. В файлах Dockerfile содержатся следующие инструкции по созданию образа:

Команда	Назначение	Пример использования
FROM	задаёт базовый (родительский) образ	FROM ubuntu
LABEL	описывает метаданные, например сведения о том, кто создал и поддерживает образ	LABEL maintainer="researcher1"
ENV	устанавливает постоянные переменные среды	ENV PATH="/home/user"
RUN	выполняет команду и создаёт слой образа, используется для установки в контейнер пакетов	RUN pip install numpy

Базовые команды docker

Команда	Назначение	Пример использования
COPY	копирует в контейнер файлы и папки	COPY ./apps
ADD	копирует файлы и папки в контейнер, может распаковывать tar-файлы (архивы)	ADD test.py /apps
CMD	описывает команду с аргументами, которая должна выполняться при запуске контейнера, может быть лишь одна инструкция CMD	CMD ["python", "test.py"]
WORKDIR	задаёт рабочую директорию для следующей за ней инструкции	WORKDIR /apps
ARG	задаёт переменные для передачи Docker во время сборки образа	ARG my_var=1

Базовые команды docker

Команда	Назначение	Пример использования
ENTRYPOINT	предоставляет команду с аргументами для вызова во время выполнения контейнера, аргументы не переопределяются.	ENTRYPOINT ["python", "./script.py"]
EXPOSE	указывает на необходимость открыть порт	EXPOSE 8000
VOLUME	создаёт точку монтирования для работы с постоянным хранилищем	VOLUME /my_volume

Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развертыванием проектов, состоящих из нескольких независимых совместно работающих приложений. **docker-compose** позволяет запускать и контролировать работу многих контейнеров, описывать их взаимодействие между собой, перезапускать при необходимости аварийно завершившиеся контейнеры.

Команды docker-compose

<code>docker-compose build</code>	создать все необходимые docker-образы
<code>docker-compose up</code>	запустить все docker-контейнеры
<code>docker-compose down</code>	остановить все docker-контейнеры
<code>docker-compose logs -f [service name]</code>	посмотреть log-файлы
<code>docker-compose ps</code>	посмотреть все работающие контейнеры
<code>docker-compose exec [service name] [command]</code>	выполнить команду в определенном контейнере-сервисе
<code>docker-compose images</code>	посмотреть все доступные docker-образы

docker

При вызове команды `docker-compose` ищется файл **`docker-compose.yml`**, содержащий необходимые инструкции для `docker-compose`. Пример такого фай

```
1  version: '3'
2
3  services:
4      nginx:
5          build:
6              context: .
7              dockerfile: ./docker/nginx/Dockerfile
8          ports:
9              - 80:80
10         depends_on:
11             - web
12         volumes:
13             - static_volume:/apps/simbank/static
14     web:
15         build:
16             context: .
17             dockerfile: ./docker/python/Dockerfile
18         command: gunicorn simbank.wsgi:application --bind 0.0.0.0:8000
19         expose:
20             - 8000
21         env_file:
22             - ../.env.prod
23         depends_on:
24             - db
25         volumes:
26             - static_volume:/apps/simbank/static
27     db:
28         image: postgres:12.0-alpine
29         volumes:
30             - postgres_data:/var/lib/postgresql/data/
31         env_file:
32             - ../.env.prod.db
33
```


docker

Давайте создадим образ и запустим контейнер с этим образом.

Для этого надо выполнить следующие шаги.

1. Сначала попробуем запустить docker-контейнер с «чистой» операционной системой alpine простым python скриптом:

```
docker container run alpine python
```

По сообщениям системы видим, что python в образе alpine по умолчанию отсутствует.

2. Создаем файл Dockerfile с содержимым:

```
FROM alpine  
RUN apk add python  
COPY . /apps  
WORKDIR /apps  
CMD ["python", "test.py"]
```

Практический пример использования docker

3. Создаем файл test.py, который будет выполняться в контейнере, например:

```
a = [i**2 for i in range(1,11)]  
print(a)
```

4. Создаем Docker

```
docker image build -t test_python:0.1 .
```

Важно не забыть точку в конце этой конструкции. Она имеет важное значение, обозначает текущую директорию.

5. Запускаем контейнер с использованием созданного Docker образа:

```
docker container run test_python:0.1
```

6. Проверяем статус запущенного

```
docker container ls -a
```