# Модуль 1. Создание инфраструктуры для простого проекта машинного обучения.

## В этом модуле

Как вы уже знаете из предыдущих модулей в разных компаниях и разных проектах машинного обучения уровень использования методов MLOps сильно отличается, как по набору инструментов, так и по степени автоматизации процессов. Модель «зрелости» процессов MLOps хорошо описана в статье [1], в которой выделяются три уровня зрелости процессов MLOps:

* «0 уровень»: почти отсутствующая автоматизация, артефакты передаются вручную от процесса к процессу и от специалиста к специалисту,
* «1 уровень»: выполнена частичная автоматизация процесса машинного обучения и внедрены отдельные функции CI/CD для быстрого развертывания, но большинство функций эксплуатации выполняется по-прежнему вручную, что не дает возможности автоматизировать переход от обучения к эксплуатации и обратно,
* «2 уровень»: имеется очень тесная связь процессов машинного обучения и эксплуатации, что позволяет быстро разворачивать решение в промышленном окружении, проверять новые гипотезы и использовать полученный при эксплуатации опыт для формирования и проверки новых гипотез и улучшения качества работы модели машинного обучения.

Переход к высокому уровню автоматизации не происходит мгновенно, обычно компаниям приходится проводить испытания нескольких вариантов решения задачи. Для простых проектов и небольших команд достаточно самого начального уровня автоматизации. В этом модуле мы будем работать над простым проектом машинного обучения, для организации которого достаточно «нулевого», самого начального, по терминологии [1], уровня зрелости MLOps. Однако даже в простых проектах важно контролировать выполняемые задачи и процессы и управлять действиями и результатами на всех этапах проекта, чтобы минимизировать количество ошибок и уменьшить непроизводительные затраты времени. Например, важной является информация о компонентах проекта машинного обучения и их изменении в результате экспериментов:

* структура и предназначение датасетов,
* правила проверки и предобработки данных,
* состав и результаты работы конвейеров моделей, метрики,
* версии библиотек и параметры среды выполнения программ,
* наборы признаков и достигнутых на них результатах.

И это относится не только к промышленным проектам, но и к учебным, образовательным, соревновательным или проектам-хобби, так называемым “pet-project”.

Для простоты освоения материала и выполнения практического задания в этом модуле рассмотрен простой проект и использован минимум средств для автоматизации операций, основной акцент сделан на практическом изучении последовательности необходимых действий, возникающих артефактах, то есть результатах, получаемых на отдельных этапах, и типовых инструментах. Использование полученной информации для выполнения практических заданий модуля поможет вам научиться реализовывать минимальный набор операций в конвейере простого проекта машинного обучения. Этой информации достаточно для выполнения основных задач MLOps для небольших проектов машинного обучения. Сведения из этого модуля полезны для практического понимания задач MLOps и быстрого перехода к более сложным задачам и инструментам, которые будут рассматриваться в следующих модулях.

Общая задача, решаемая в модуле: создать инфраструктуру для небольшого проекта машинного обучения, включающую инструменты для работы с данными и проведения исследований, средства автоматизации, создание производственного окружения и запуск модели в эксплуатацию. Эта общая задача решается с помощью отдельных частных подзадач:

1. развертывание виртуальных машин,
2. установка инструментов для участников команды
   1. python как основной язык программирования
   2. инструменты для работы с виртуальными окружениями
   3. JupyterHub для командной работы исследователей в Jupyter
   4. VSCode для разработчиков
   5. Docker для вывода модели в эксплуатацию
3. создание инфраструктуры для хранения и контроля версия для данных и программного кода, настройка git и dvc
4. организация работы участников проекта с помощью dvc
   1. препроцессинг данных
   2. трекинг экспериментов
   3. контроль метрик
   4. версионирование моделей

Следующим логичным этапом жизни модели является вывод эксплуатацию. Мы создадим соответствующие элементы архитектуры для этой задачи в данном модуле, однако самому выводу в эксплуатацию будет посвящен следующий модуль.

Мы создадим три виртуальные машины:

* для хранения данных: исходных датасетов и их измененных версий,
* для проведения экспериментов и обучения моделей и создания программного кода системы,
* для эксплуатации модели.

Для управления наборами данных мы используем dvc. Также dvc будет использоваться для трекинга обучения моделей и контроля метрик. Контроль версий программного кода делается с использованием git. Исследователи проверяют гипотезы с использованием Jupyter Notebook. Наиболее удачные гипотезы оформляются в виде python кода для обучения модели и обучение выполняется на больших данных, результат сохраняется в виде обученных моделей в формате pickle. dvc используется для сохранения комбинаций «данные»-«модель»-«метрика». Разработчики системы используют VSCode для работы над кодом проекта, создают бэкэнд и фронтэнд системы и добавляют модель машинного обучения, полученную исследователем. Для эксплуатации система, включая модель машинного обучения, разворачивается в виде docker контейнеров, под управлением docker-compose.

## Содержание юнитов

Последовательность тем, изучаемых в отдельных юнитах модуля:

|  |
| --- |
| 1. Постановка задачи. |
| 2. Создание и настройка базовой инфраструктуры проекта. |
| 2.1. Создание виртуальных машин VirtualBox. |
| 2.2. Установка, настройка и администрирование linux ubuntu. |
| 2.3. Установка и настройка python и необходимых библиотек. |
| 2.4. Настройка виртуальных окружений. |
| 3. Установка специального программного обеспечения. |
| 3.1. Настройка взаимодействия с git и dvc. |
| 3.2. Установка и настройка программного обеспечения JupyterHub, настройка работы с различными виртуальными окружениями в jupyter ноутбук. |
| 3.3. Установка и настройка VSCode. |
| 3.4. Настройка docker и docker-compose. |
| 4. Настройка рабочего окружения для проекта. |
| 4.1. Формирование рабочей структуры директориев. |
| 4.2. Подготовка python скриптов для отдельных этапов проекта, подключение к github, создание репозитория проекта в git. Загрузка сырых данных в хранилище данных, подготовка датасетов, предобработка, сохранение и загрузка рабочего датасета. |
| 4.3. Проведение первичных исследований в Jupyter Notebook. Подготовка python кода для проведения экспериментов и обучения модели. |
| 4.4. Загрузка данных через dvc. Создание рабочего пространства в VSCode. Обучение модели, сохранение артефактов. |

# Модуль 2. Юнит 1. Постановка задачи.

*Введение*: В этом юните делается постановка решаемой в модуле задачи и даются рекомендации по действиям, которые необходимо выполнить для решения задачи.

*Содержание:*

В этом модуле разбирается практический пример создания инфраструктуры для простого проекта машинного обучения. Несмотря на то, что рассматриваемый проект имеет очень простую структуру и решает учебную, а не производственную, задачу, возникающие при прохождения модуля ситуации часто возникают в реальных производственных проектах.

Давайте рассмотрим достаточно типовой сценарий в простых проектах машинного обучения. Исследователь создает модель, обучает ее на имеющихся данных, выкладывает программный код в git репозиторий и с использованием flask создает web сервис с API или простым графическим интерфейсом для того, чтобы разработанной моделью могли воспользоваться пользователи. Даже если это учебный проект, или так называемый pet-проект, возникает множество практических задач:

* Собирать, обрабатывать и исследовать данные
* Проводить эксперименты и исследовать гипотезы
* Управлять аппаратным обеспечением для обучения модели
* Контролировать перечень и зависимости используемых библиотек
* Управлять процессом обучения модели, контролировать переобучение
* Обеспечивать версионирование
* Выводить модель в эксплуатацию и осуществлять мониторинг качества работы.

Мы будем стремиться к созданию простой, но полнофункциональной инфраструктуры для нашего проекта, которая включает основные сегменты и инструменты для всего конвейера проекта от сбора и обработки данных до вывода модели в промышленное окружение и мониторинга.

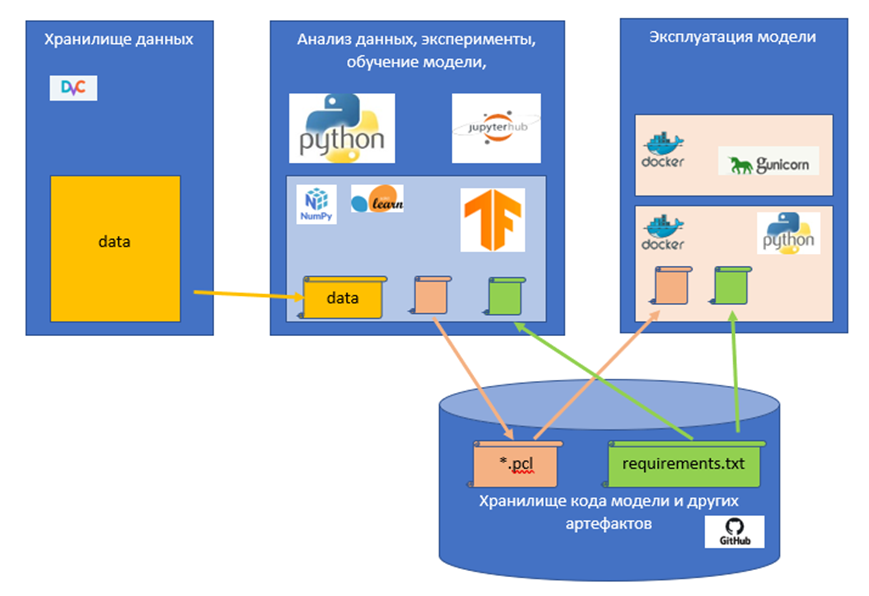


Рисунок «Целевая инфраструктура»

**«Хранилище данных»**

В проекте мы будем работать с “Titanic Disaster” с Kaggle.com.

В качестве операционной системы будем использовать Ubuntu.

Для организации доступа к данным мы настроим работу утилиты dvc. Один из вариантов хранилищ данных, поддерживаемых dvc, это локальный сервер с доступом по ssh, для реализации этой схемы мы установим openssh-server.

**«Анализ данных, эксперименты и обучение модели»**

Нам понадобится python. Будем устанавливать операционную систему Ubuntu 20, которая по умолчанию содержит python3.8. Для фиксации рабочего окружения будем использовать виртуальное окружение, создаваемое с помощью venv. В это окружение установим необходимые библиотеки для анализа данных и машинного обучения: numpy, pandas, matplotlib, scikit-learn, tensorflow. Для анализа данных и исследований эффективно использовать инструменты jupyter, для многопользовательской работы используется jupyterhub, который мы установим.

В результате обучения мы получим модель, которую сохраним в формате pickle на dvc хранилище. Также на github сохраним и другие артефакты, полученные на этом этапе: описание параметров виртуального окружения, метаинформацию по датасету.

**«Эксплуатация модели»**

Для запуска модели в эксплуатации нам понадобится собрать весь проект целиком, включая backend и frontend проекта. Для удобства проект будет представлять собой микросервис, который запускается в docker. Настройки среды внутри контейнера берутся из файла requirements.txt.

## Тест

1. Какая утилита будет использоваться для организации доступа к набору данных? (0.25)
   1. curl
   2. ftp
   3. tcp
   4. **dvc**
2. Какие инструменты мы установим на сервер обучения модели? (0.25)
   1. **numpy**
   2. **tensorflow**
   3. docker
   4. **jupyter notebook**
3. Какая утилита используется на сервере эксплуатации? (0.25)
   1. openssh-server
   2. **docker**
   3. dvc
   4. jupyter notebook
4. Сколько виртуальных машин планируется развернуть в проекте? (0.25)
   1. 1
   2. 2
   3. **3**
   4. 4

## Итоги/выводы

В этом юните сформулирована задача модуля, описаны инструменты для ее решения, описано что будем делать в следующих юнитах модуля.

# Модуль 2. Юнит 2. Создание и настройка базовой инфраструктуры проекта.

*Введение:* Для работы над проектом в данном модуле нам необходима инфраструктура: аппаратное обеспечение, общее и специальное программное обеспечение. В этом юните мы создадим базовую часть целевой инфраструктуры: виртуальные машины с операционной системой, интерпретатор python и основные библиотеки машинного обучения, средства для создания виртуальных окружений. Также мы осуществим базовую настройку операционных систем, добавим основные необходимые утилиты и создадим пользователей, с учетом ролей участников команды.

*Содержание:*

В предыдущем параграфе вы познакомились с постановкой задачи и целевой инфраструктурой, которую нам необходимо создать для решения задач MLOps в нашем проекте.

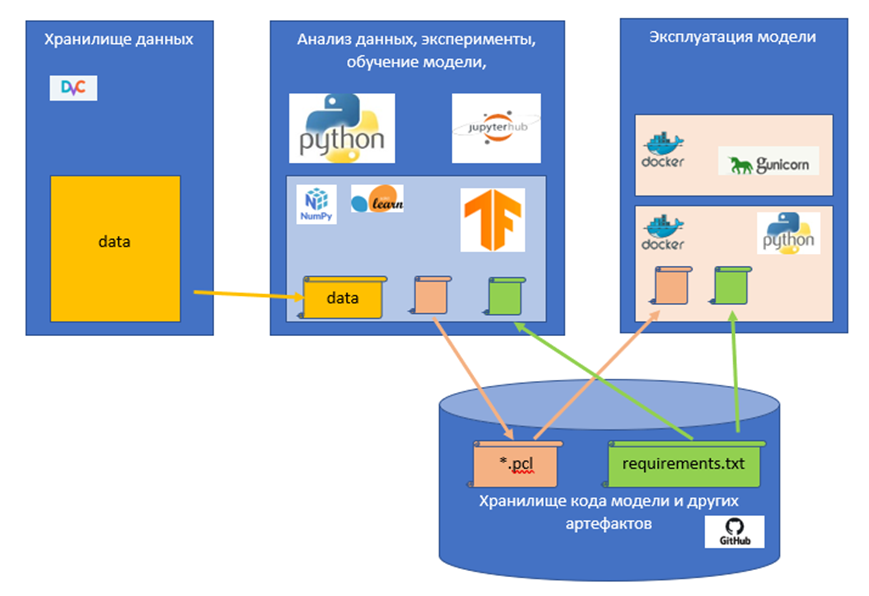


Рисунок «Целевая инфраструктура»

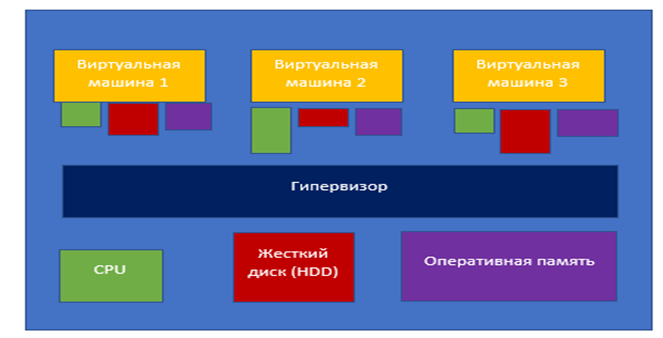
Давайте рассмотрим подробнее какие виртуальные машины нам понадобятся, какое программное обеспечение для них необходимо.

## Виртуальные машины.

В этой части мы рассмотрим процедуру создания виртуальных машин VirtualBox. Мы создадим три виртуальные машины:

|  |  |  |
| --- | --- | --- |
| hostname | Назначение | Содержание |
| data\_srv | Хранилище данных | OS Ubuntu, dvc, openssh-server |
| ml\_srv | Анализ данных, эксперименты, обучение модели | OS Ubuntu, python3.8, jupyterhub, venv, pip, numpy, sklearn |
| prod\_srv | Эксплуатация модели | OS Ubuntu, docker |

Создаваемые виртуальные машины будут совместно использовать аппаратные ресурсы компьютера-хоста.



Далее в проекте мы будем работать с уже известным нам свободно распространяемым программным обеспечением для виртуализации серверов VirtualBox. Официальная страница для скачивания продукта <https://www.virtualbox.org/wiki/Downloads>. В VirtualBox каждая виртуальная машина может иметь до четырех сетевых адаптеров, а каждый такой адаптер имеет определённый режим работы:

* NAT (Network Address Translation): 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,
* NAT Network: то же, что и NAT, но виртуальные машины могут взаимодействовать между собой через внутреннюю сеть,
* Bridged: виртуальная машина имеет свой собственный статический IP адрес и доступна из внешней сети напрямую,
* Host-only: 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.

Эти режимы отвечают за способ взаимодействия виртуальных машин с хостом и между собой, а также будет ли у них доступ в интернет и смогут ли другие устройства в сети хоста взаимодействовать с этими виртуальными машинами.

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

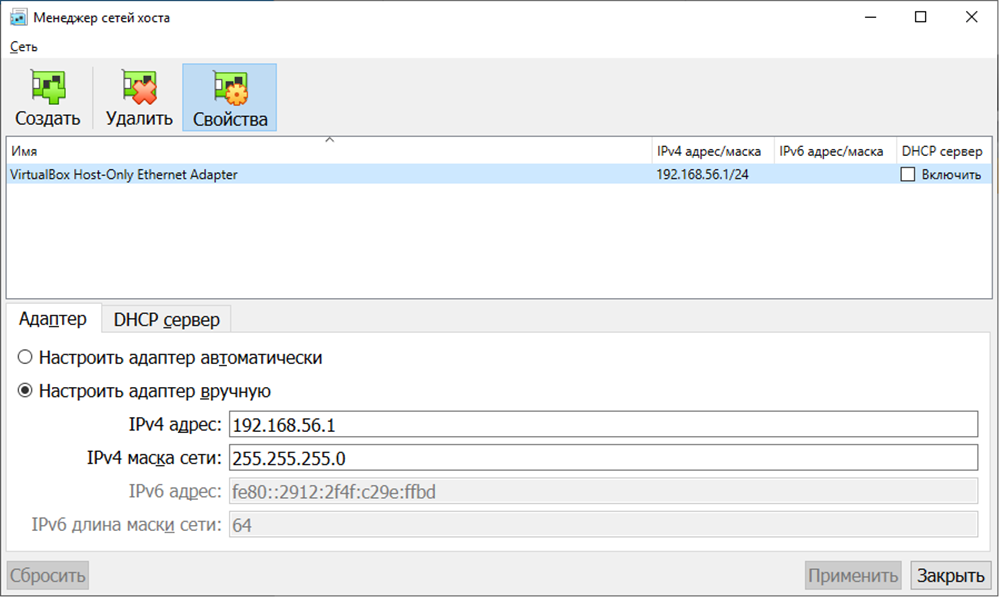


Рисунок “Менеджер сетей хоста VirtualBox”.

Если требуется отключить динамическое назначение IP адресов для виртуальных машин, то необходимо настроить эту опцию в параметре “DHCP сервер”. Поле “IPv4 адрес” задает IP адрес хоста. Мы можем назначать IP адреса виртуальным машинам из одной подсети с хостом, подсеть задается маской в поле “IPv4 маска сети”. После изменения адреса сетевого адаптера необходимо обязательно перезагрузить компьютер, т.к. без этого виртуальные машины не будут запускаться.

Для создания виртуальной машины в пункте меню “Машина” необходимо выбрать раздел “Создать”, после этого можно задать имя виртуальной машины, указать ее тип и версию. Также можно перейти в расширенные настройки параметров виртуальной машины и изменить такие параметры как объем памяти и способ организации виртуального жесткого диска.

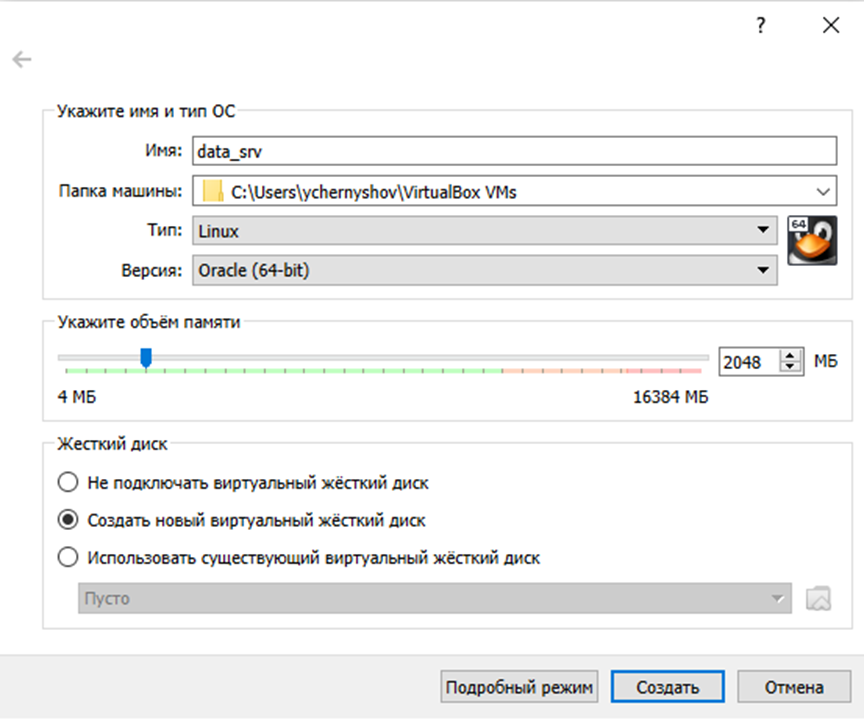


Рисунок “Настройки виртуальной машины data\_srv”.

Наш проект будет распределен по нескольким серверам, взаимодействующим между собой и с внешними ресурсами, например, github.com или python.org. Поэтому мы должны для каждого виртуального сервера сделать IP адрес, через который они будут друг с другом общаться.

После этого создаем для виртуальной машины новый виртуальный жёсткий диск в формате VDI (VirtualBox Disk Image) с форматом хранения “Динамический виртуальный жесткий диск” и размером 16Гб (чтобы у нас была возможность устанавливать нужные библиотеки и работать с большими данными). После этого наша виртуальная машина готова.

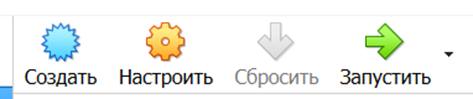


Рисунок “Параметры созданной виртуальной машины VirtualBox”.

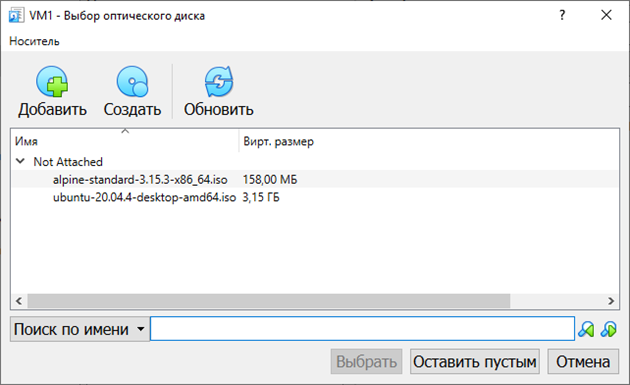
У созданной виртуальной машины можно теперь менять параметры через пункт меню “Настройки”, например, изменить количество процессоров.

## Операционная система, пользователи и группы.

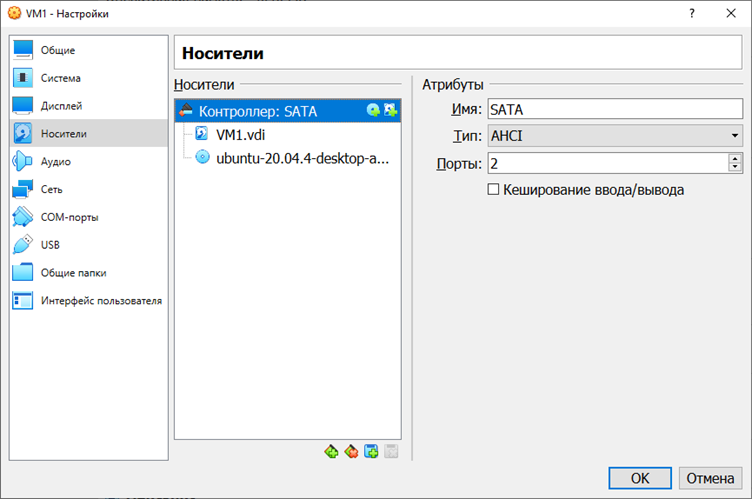
В этой части мы рассмотрим установку, настройку и администрирование операционной системы linux ubuntu и создадим некоторых пользователей. Виртуальная машина не может эксплуатироваться без операционной системы, хотя и обладает после создания виртуальными ресурсами - процессором, оперативной памятью, жестким 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”



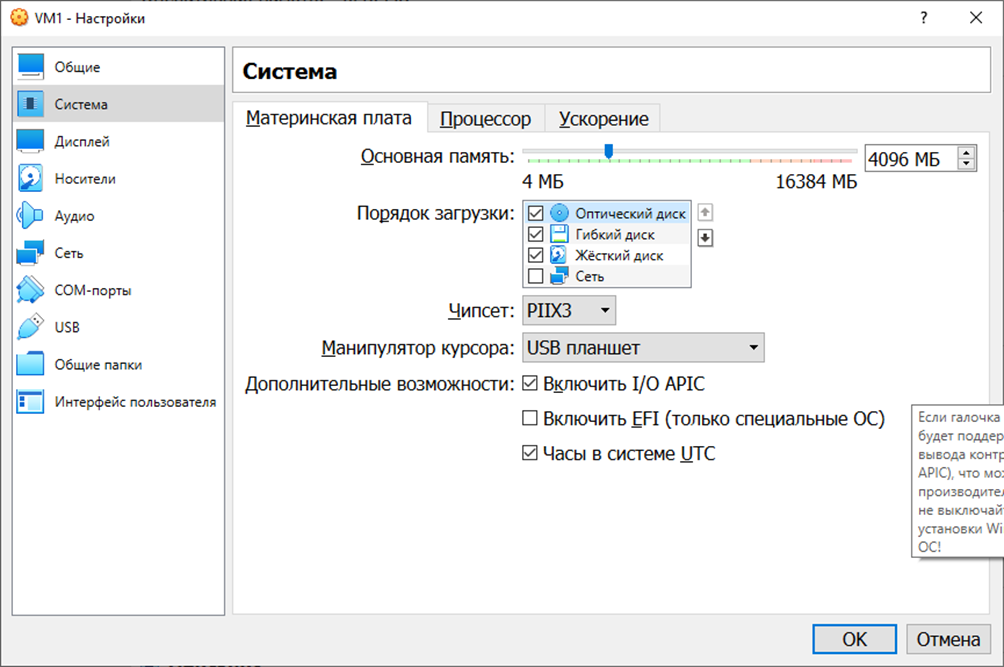
после чего появится окно для настроек, в котором в разделе “Носители” надо добавить оптический диск



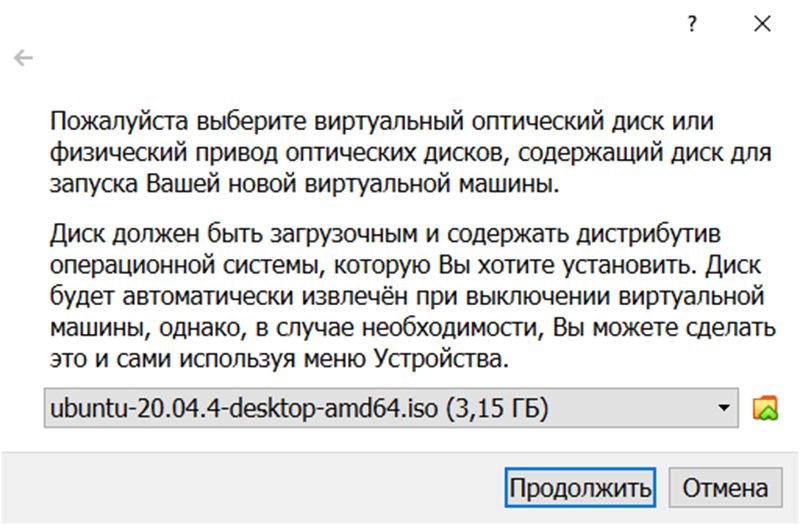
Он должен появиться в перечне дисков



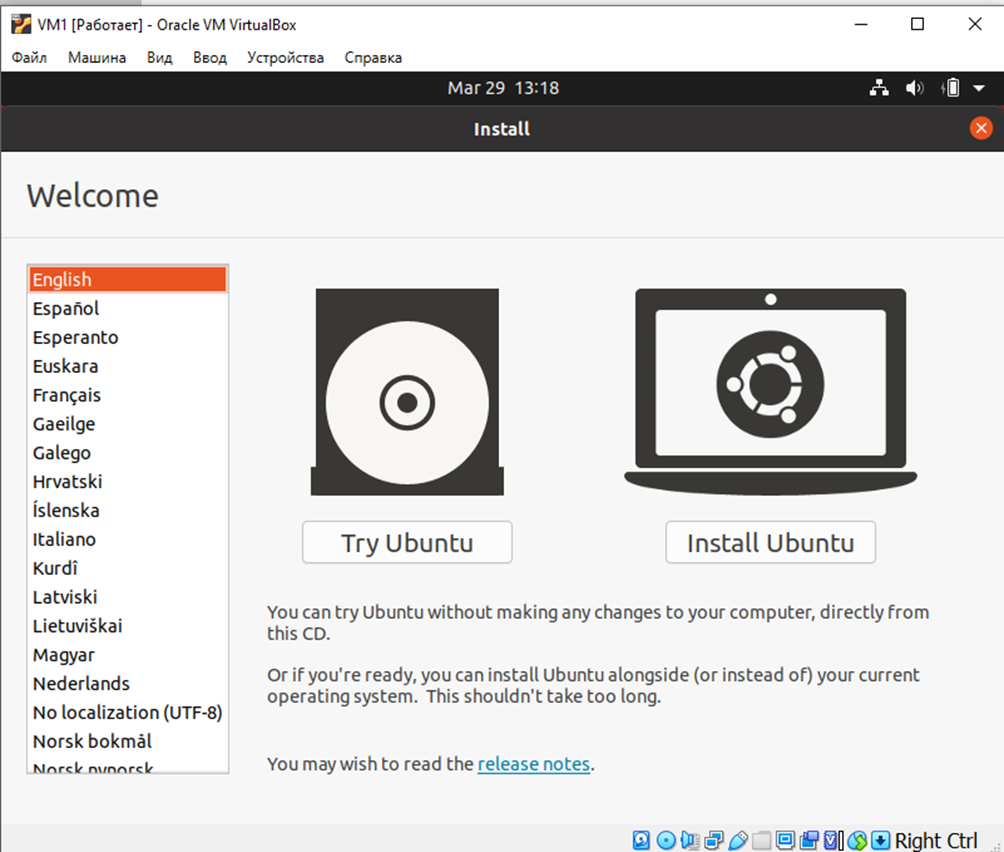
После этого в разделе “Система” меняем порядок загрузки, делаем оптический диск первым



После этого нажимаем кнопку “Запустить” и в появившемся окне надо выбрать виртуальный загрузочный диск



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.

В каждой системе могут работать несколько пользователей, которые объединяются в группы, в соответствие с задачами, которые выполняют участники групп. Для этих целей в операционной системе Linux существуют объекты «пользователь» (user) и «группа» (group).

Для работы с JupyterHub и другими программами используются соответствующие пользовательские аккаунты, то есть администратор системы должен создать пользователя, например, с помощью команды

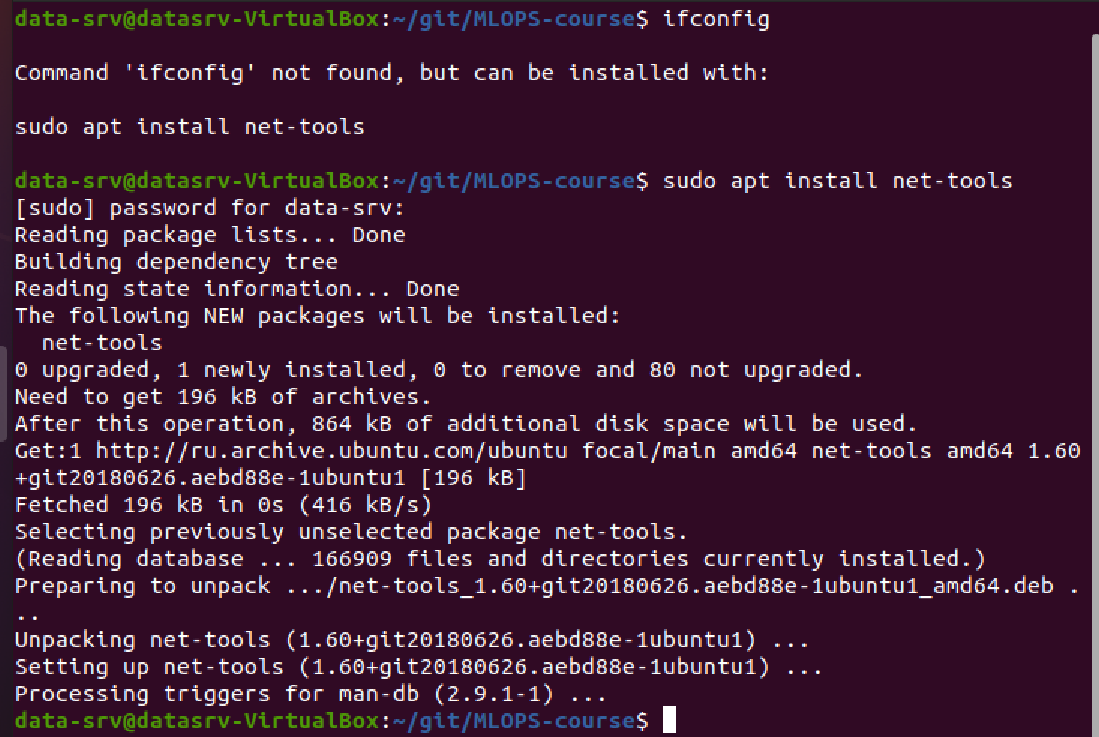
**sudo adduser --home “рабочий директорий пользователя” “имя пользователя”**

На каждом из трех серверов data\_srv, ml\_srv и prod\_srv нами уже созданы аккаунты с правами суперпользователя, являющиеся членами группы sudo, это делается при создании виртуальной машины. Например, вы могли им присвоить имена data-srv-admin, ml-srv-admin и prod-srv-admin, как сделано далее, либо просто admin для всех серверов, либо придумать свое имя. Обычно такими расширенными привилегиями обладает системный администратор, DevOps или MLOps инженер, однако в команде проекта существуют другие участники, различающиеся выполняемыми задачами и используемыми инструментами. Давайте создадим двух пользователей на сервере ml\_srv: data-eng и ml-eng. Эти два пользователя в проекте будут иметь возможность использовать JupyterHub. Эти пользователи создаются следующими командами

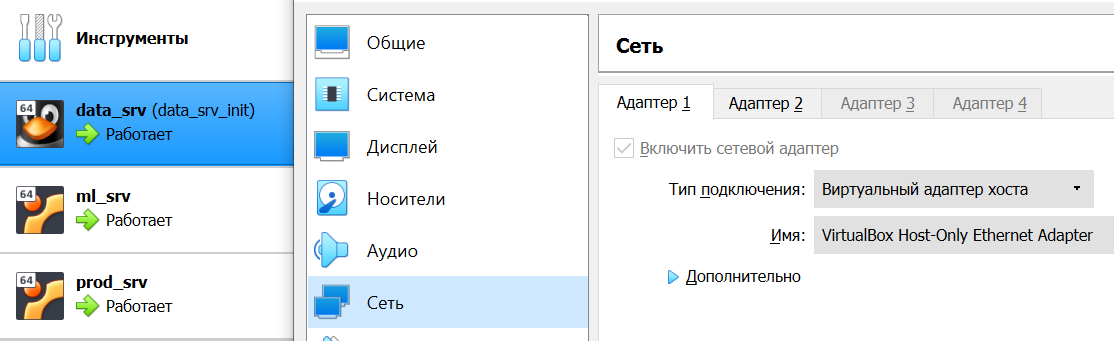
**sudo adduser - -home=/home/ml-eng ml-eng**

**sudo adduser - -home=/home/data-eng data-eng**

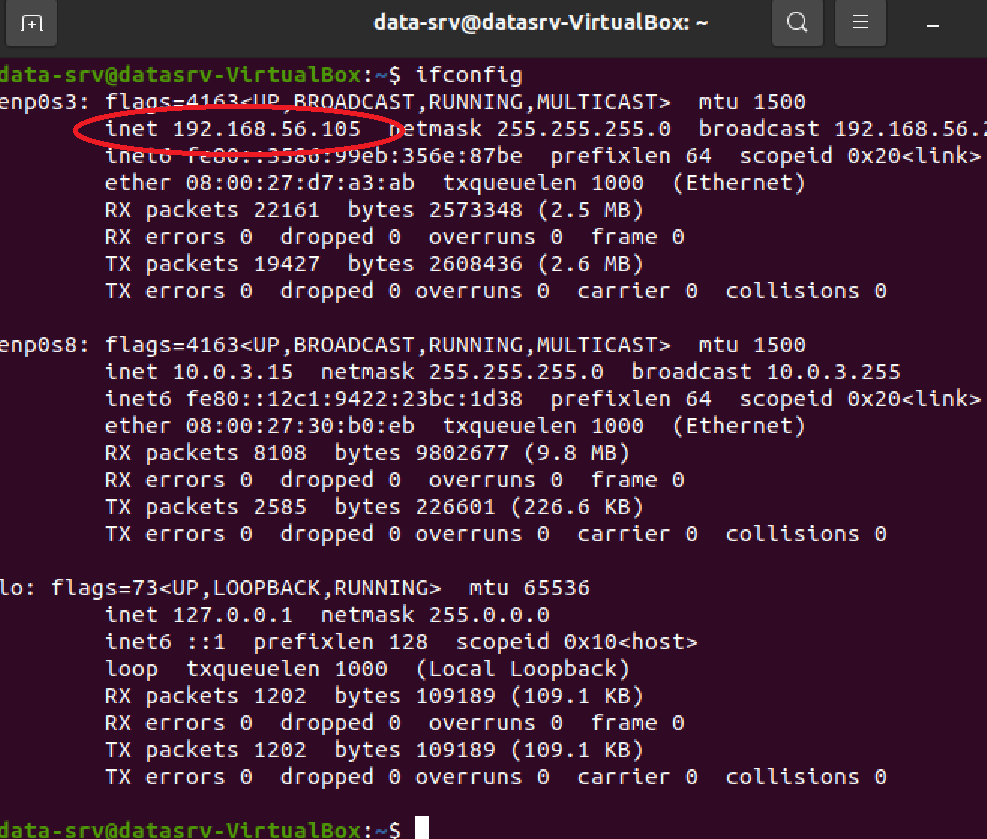
Многие из используемых утилит являются сетевыми сервисами, поэтому важно иметь возможность их мониторинга. Удобным инструментом для этого является утилиты netstat, ifconfig и другие, которые входят в пакет программ net-tools



Для нашей целевой конфигурация с использованием виртуальных машин необходимо использовать соответствующую настройку виртуальных сетевых адаптеров, а именно, обязательно выбрать в «Типе подключения» значение «Виртуальный адаптер хоста»

****

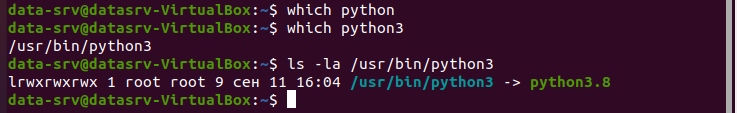
Иначе вы не сможете получать доступ до этого локального хранилища dvc с других виртуальных машин, например, ml\_srv. При этом будет использоваться тот IP адрес, который «виден» с других виртуальных машин



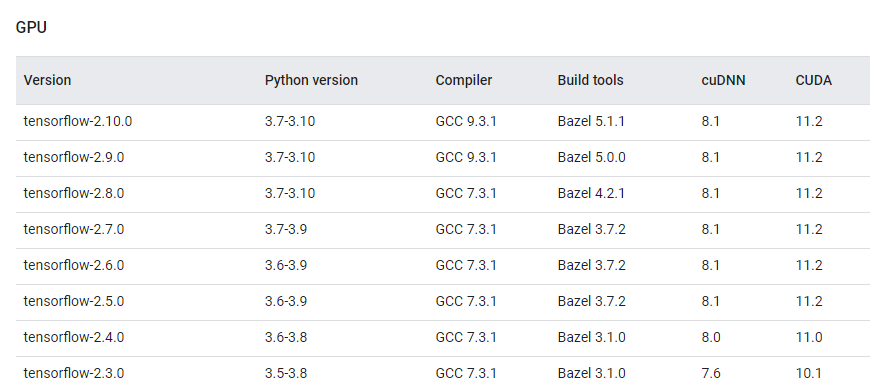
Вы можете использовать до четырех виртуальных типов подключений на одной виртуальной машине, например, на одной выбрать «NAT», удобный для скачивания программного обеспечения из открытой сети Интернет, например с использованием pip, а на другом «Виртуальный адаптер хоста», который нужен для взаимодействия виртуальных машин между собой.

## Интерпретатор python

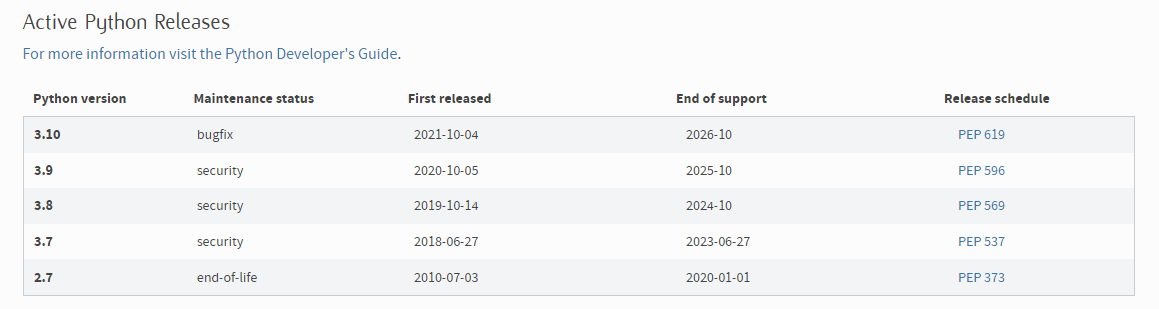
В этой части мы рассмотрим установку и настройку интерпретатора python и необходимых библиотек. По умолчанию в Ubuntu 20 уже установлен интерпретатор python 3.8.



При необходимости можно скачать и настроить другую версию python. Такая необходимость может возникнуть, например, при использовании определенной версии библиотеки Tensorflow, популярной при создании нейронных сетей глубокого машинного обучения. Вот, например, данные о совместимости различных версий python, Tensorflow и CUDA драйверов, необходимых для корректной работы с графическим процессором (GPU), данные взяты с официальной страницы <https://www.tensorflow.org/install/source#gpu>

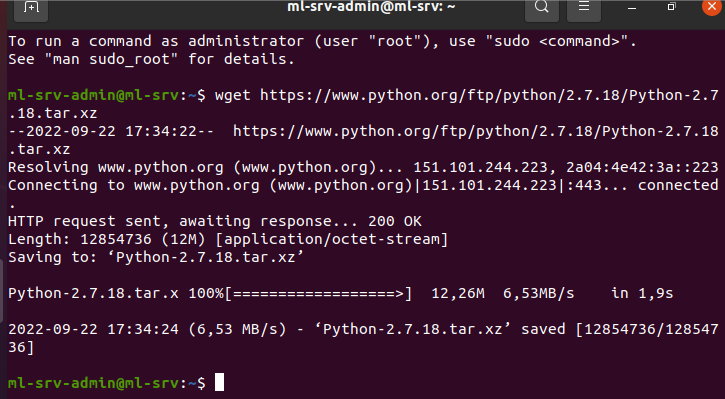


Для использования конкретной версии python можно установить требуемую версию с использованием официальной страницы проекта <https://www.python.org/downloads>. В частности, здесь можно найти информацию о текущем актуальном состоянии версий интерпретатора python.



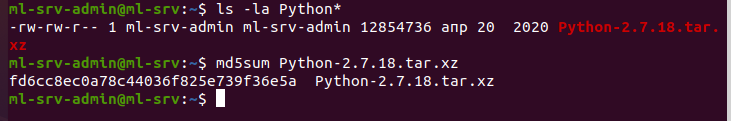
Несмотря на то, что версия python2 находится в состоянии “end-of-life”, вы еще можете встретиться с программным обеспечением, которое требует данной версии интерпретатора. В этом случае в системе одновременно существует Python версии 2 и 3, поэтому команда, запускающая интерпретатор, имеет вид python2 или python3.

Давайте установим python2 на виртуальном сервере ml\_srv.



Контрольная сумма вычисляется путем суммирования всех цифр нужных данных. Ее можно использовать в дальнейшем для обнаружения ошибок в проверяемых данных при хранении или передаче. Тогда контрольная сумма пересчитывается еще раз и полученное значение сверяется с предыдущим. Контрольные суммы Linux с вычисляемые по алгоритму MD5 (Message Digest 5) могут быть использованы для проверки целостности строк или файлов. MD5 сумма - это 128 битная строка, которая состоит из букв и цифр. Суть алгоритма MD5 в том, что для конкретного файла или строки будет генерироваться 128 битный хэш, и он будет одинаковым на всех машинах, если файлы идентичны. Трудно найти два разных файла, которые бы выдали одинаковые хэши. Это важно для повышения уровни безопасности и доверия с программному обеспечению, скачанному из открытого доступа в Интернет.

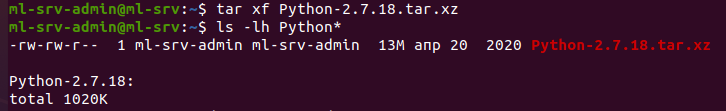
В Linux для подсчета контрольных сумм по алгоритму md5 используется утилита md5sum.



Сравниваем с информацией, которая приведена на странице <https://www.python.org/downloads/release/python-2718/> и убеждаемся в правильности контрольной суммы для скачанного файла



Теперь скачанный архив можно разархивировать



После этого потребуется перейти в папку Python-2.7.18 и выполнить команды, которые создадут и откомпилируют все необходимые файлы и подготовят установку python. Последовательность шагов, необходимых для установки, а также другая полезная информация описана в файле Python-2.7.18/README. Для выполнения процедуры установки потребуется компилятор gcc для языка C, который устанавливается с помощью команды

**sudo apt install build-essential**

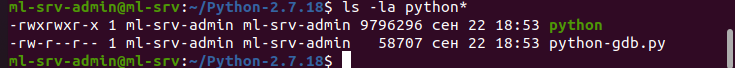
После успешной установки компилятора можно выполнить команду

**./configure**

и затем

**make**

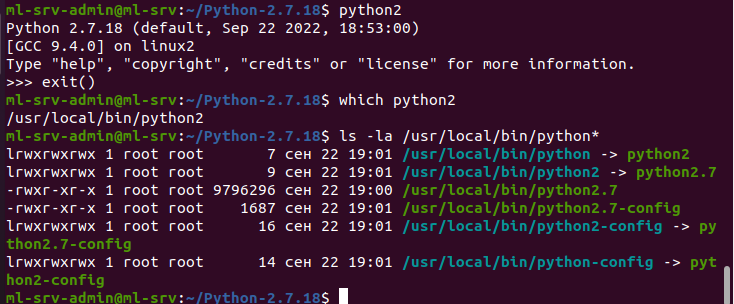
Команда make создает исполняемый файл для программы, написанной на языке C, в соответствии с инструкциями, описанными в файле Makefile. В нашем случае результатом выполнения программы make будет исполняемый файл ./python.



Теперь необходимо с использованием прав суперпользователя осуществить установку нужной нам версии интерпретатора в служебную папку /usr/local. Для этого надо выполнить команду

**sudo make install**

После этого будет установлен необходимый софт, добавлены нужные пути до исполняемых файлов в переменную окружения PATH и создана символьная ссылка python2.



Теперь на сервере ml\_srv у нас установлены две версии интерпретатора python: версия 2 и версия 3. Часто требуется иметь возможность пользоваться разными версиями python3, для этого можно установить соответствующий пакет программного обеспечения, используя инструкцию, описанную выше. Далее, изменяя символьную ссылку python3 можно установить нужную версию интерпретатора. Однако более аккуратным способом является создание виртуальных окружений, включающих в себя нужную версию интерпретатора python, а не использующих глобально установленный в системе интерпретатор.

Как вы заметили, мы добавили программное обеспечение компилятора gcc, без которого нельзя было выполнить процедуру make, для этого использовали установку build-essentials. Кроме этого пакета есть и другие пакеты, связанные с организацией среды разработки, например

* build-essential – пакеты, необходимые для компиляции программного обеспечения
* libssl-dev – необходим для разработки с криптографическими протоколами SSL и TLS
* libffi-dev
* python3-dev

Все пакеты можно установить командой

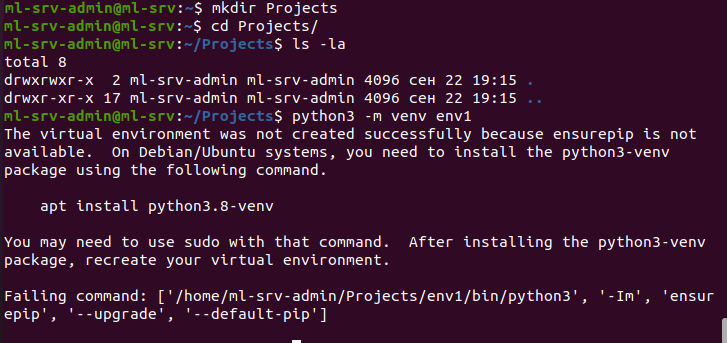
**sudo apt install -y build-essential libssl-dev libffi-dev python3-dev**

## Виртуальные окружения.

В этой части мы рассмотрим установку и настройку виртуальных окружений. Для создания виртуального окружения в python можно использовать различные инструменты, например, virtualenv, venv, conda, poetry. Далее в проекте мы будем использовать venv. Создать виртуальное окружение можно следующим образом:

**python3 –m venv <имя папки>**

Если предварительно утилита venv не была установлена, то вы получите соответствующее предупреждение

****

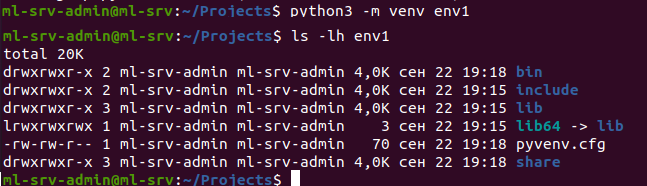
из которого ясно, что необходимо выполнить установку с помощью команды

**sudo apt install python3.8-venv**

После этого можно создать виртуальное окружение командой

**python3 -m venv env1**

после чего поcмотреть содержимое созданной папки проекта env1.



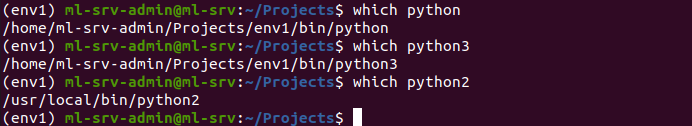
Установленное виртуальное окружение можно активировать следующим образом

**source bin/activate**

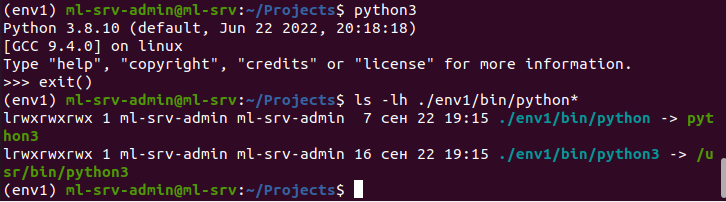
****

Рисунок “Активация виртуального окружения с помощью venv”

Интересно теперь посмотреть какой используется интерпретатор python в созданном виртуальном окружении



откуда мы видим, что интерпретатор python2 используется глобальный. Более того, более подробное изучение содержимого venv1 показывает, что и интерпретатор python3 также глобальный.



Теперь можно устанавливать требуемые версии библиотек программного обеспечения с помощью стандартного установщика pip.

Деактивировать виртуальное окружение можно командой

**deactivate**

## Тест

1. Какая версия python установлена в ОС Ubuntu 20 по умолчанию? (0.25)
   1. 1
   2. 2
   3. **3**
   4. 4
2. Какая команда дает возможность выполнять скрипты в linux с привилегиями суперпользователя? (0.25)
   1. **sudo**
   2. superuser
   3. main
   4. super
3. На каком сайте можно найти официальные и надежные установочные пакеты python? (0.25)
   1. python.com
   2. **python.org**
   3. get-python.org
   4. free-python.org
4. Какая команда деактивирует виртуальное окружение? (0.25)
   1. close
   2. **deactivate**
   3. disconnect
   4. venv

## Итоги/выводы

В изученном юните описана процедура создания и настройки базовой инфраструктуры для любого проекта, не только для машинного обучения. Вы научились создавать виртуальные сервера и устанавливать на них необходимое базовое программное обеспечение для разработки: операционную систему, компилятор языка Си, интерпретатор python, виртуальные окружения. Процедуры установки необходимых компонентов бывают разные, как выполнение одной команды, так и более тонкая сборка пакета на рабочем сервере из исходных файлов, вы познакомились с обоими вариантами. Далее на базовую инфраструктуру необходимо добавить специализированное программное обеспечение, с учетом роли оборудования и задач сотрудников. Этому посвящен следующий юнит.

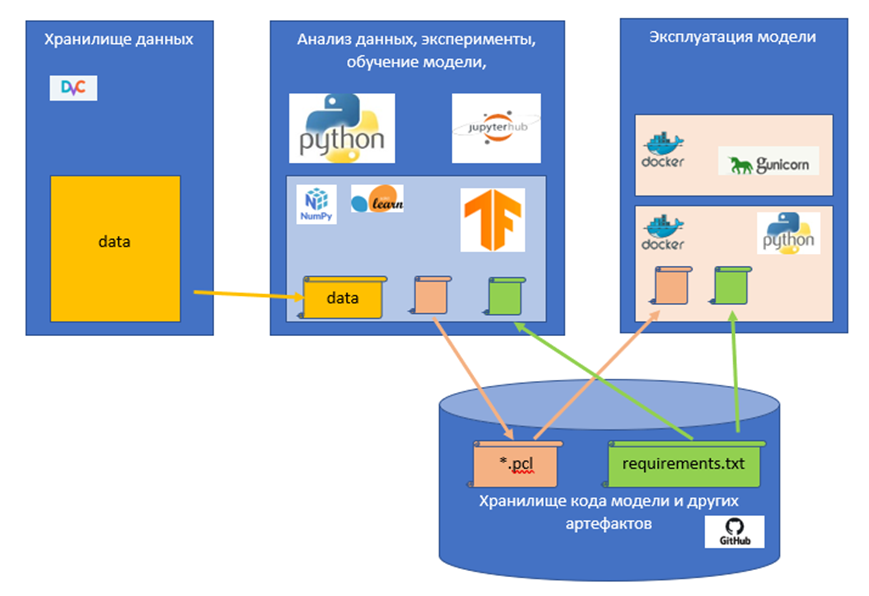
# Модуль 2. Юнит 3. Установка специального программного обеспечения.

*Введение:* Вы создали три виртуальные машины и установили базовое программное обеспечение для них. В этом юните мы осуществим установку специального программного обеспечения на каждую из этих виртуальных машин, соответствующее тем функциям, которые они выполняют:

* хранилище данных,
* эксперименты и обучение модели,
* эксплуатация.

*Содержание:*

Давайте вспомним нашу целевую архитектуру, которую мы хотим создать для нашего проекта:



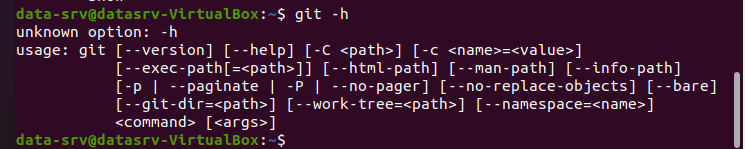
На каждом из трех серверов должны быть установлены свои компоненты для решения конкретных задач.

## git и dvc

Часто в конфигурации операционной системы по умолчанию не установлена утилита для работы с git, которая позволяет пользователям взаимодействовать с сервером git, например github.com. В linux системах установить утилиту для работы с git можно командой

**sudo apt install git**

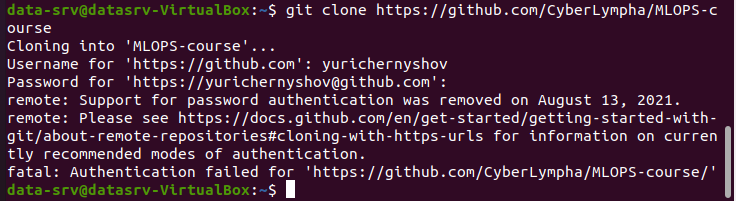
после чего можно этой утилитой можно пользоваться:



С помощью git можно выполнять стандартные действия, например, клонировать рабочий репозиторий нашего учебного проекта с помощью команды

**git clone https://github.com/CyberLympha/MLOPS-course**

Поскольку у нас не настроено подключение по ssh, а github запрещает взаимодействие без ssh, то вы увидите такое сообщение

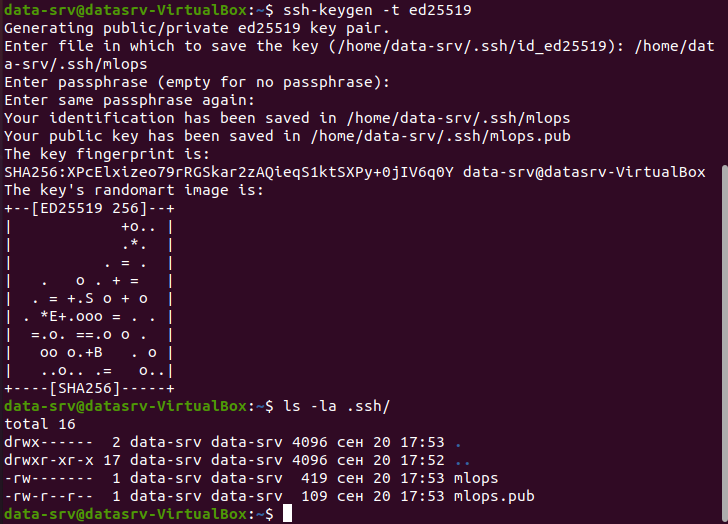


Давайте настроим доступ по ssh к учебному git репозиторию. Дальнейшие действия, которые вы будете выполнять, подробно описаны на официальной странице github:

<https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh>

Сначала необходимо создать открытый (public) и закрытый (private) ключи ssh, это делается командой

**ssh-keygen -t ed25519 -C "your\_email@example.com"**

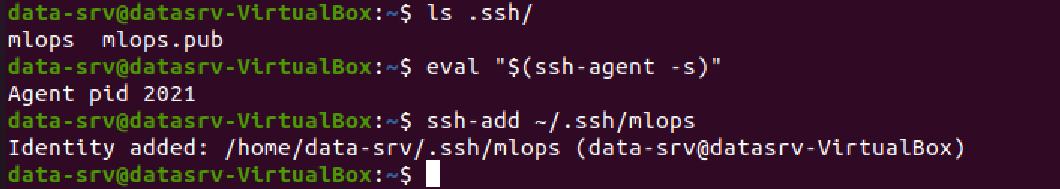


Далее необходимо запустить ssh агента

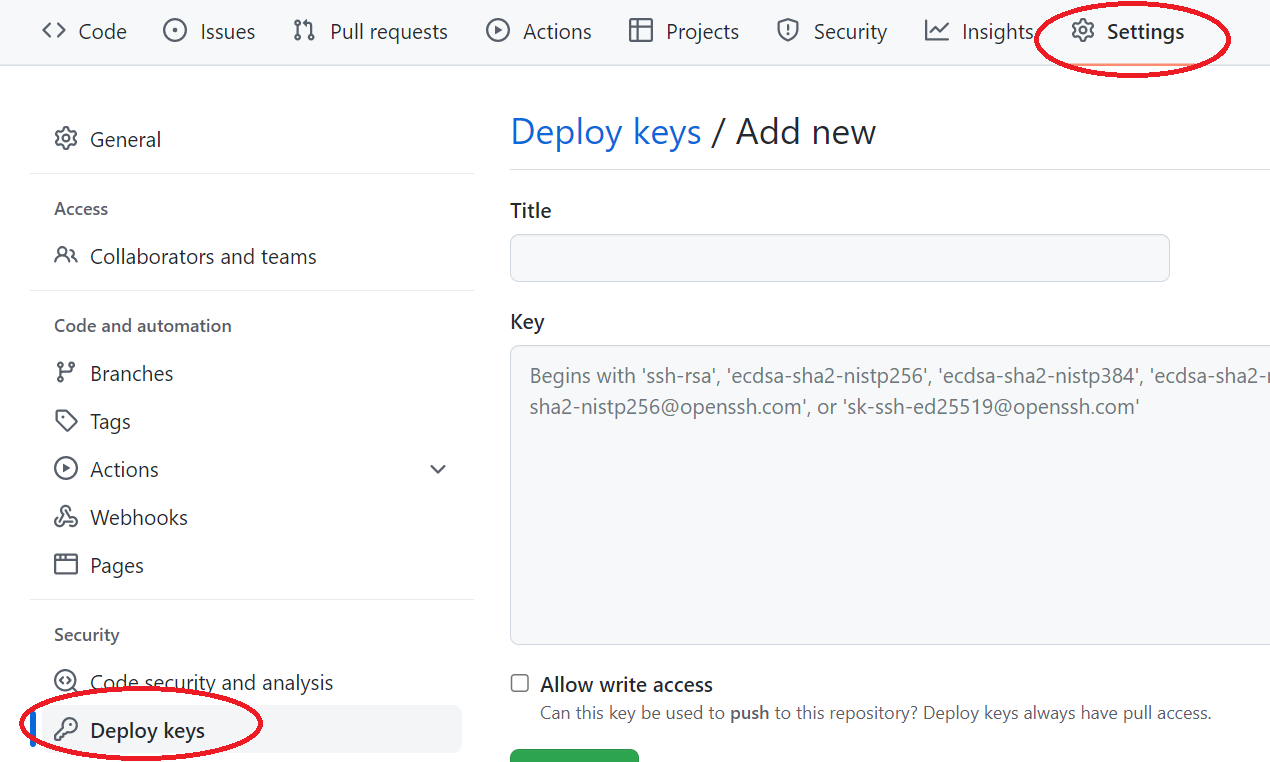
**eval "$(ssh-agent -s)"**

После этого можно добавить ключ

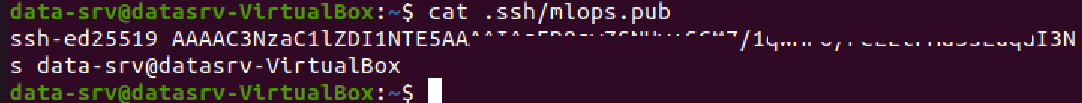
**ssh-add ~/.ssh/mlops**

****

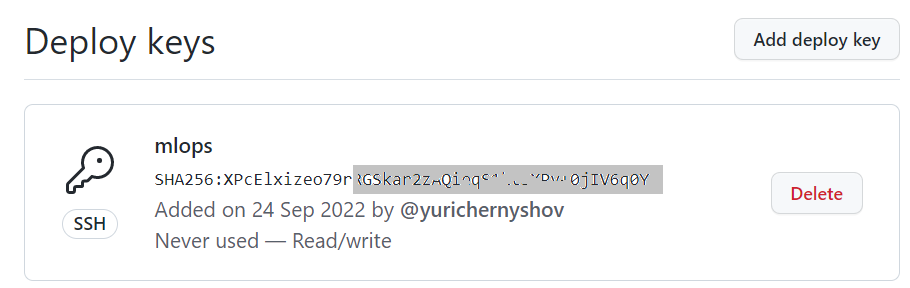
И теперь содержание публичного (public) ключа надо добавить на страницу github.com. Для этого можно просто скопировать содержимое текстового файла.



В разделе «Settings»-«Deploy keys» надо написать имя для ключа в поле Title и скопировать содержимое pub файла в поле Key.

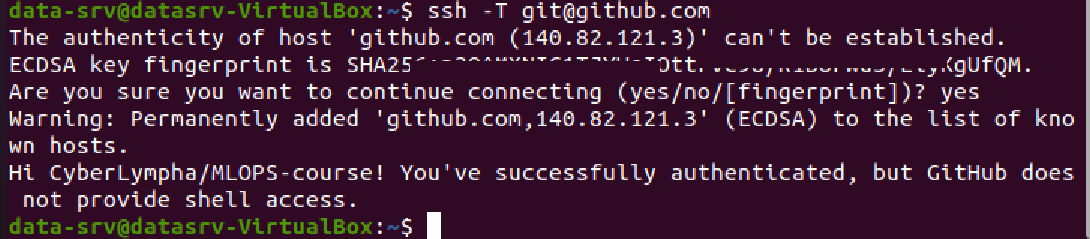


Если требуется разрешить владельцу ключа осуществлять операции записи в репозиторий, например push, то необходимо отметить пункт «Allow write access». После выполнения этих действия появится соответствующий ключ в репозитории в пункте “Deploy keys”.

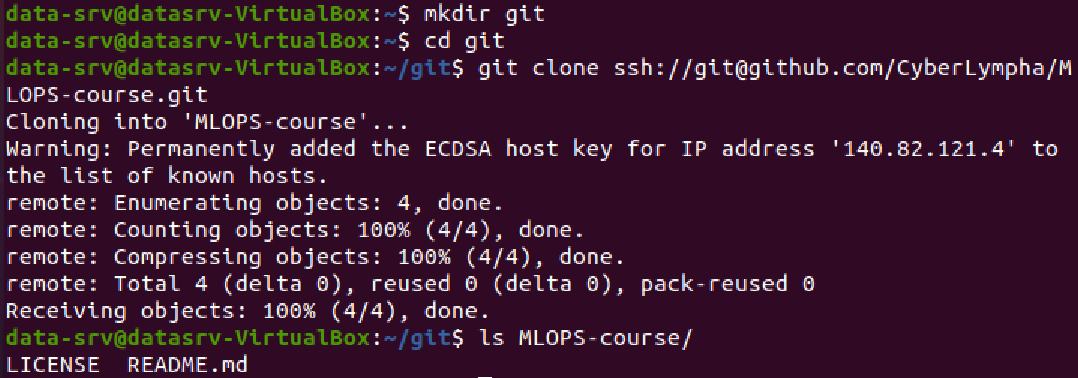


Протестировать связь можно с помощью команды

**ssh -T** [**git@github.com**](mailto:git@github.com)

****

После установки и настройки git и параметров подключения можно выполнять стандартные операции, например, клонировать репозиторий

**git clone ssh://git@github.com/<user>/<repository name>.git**

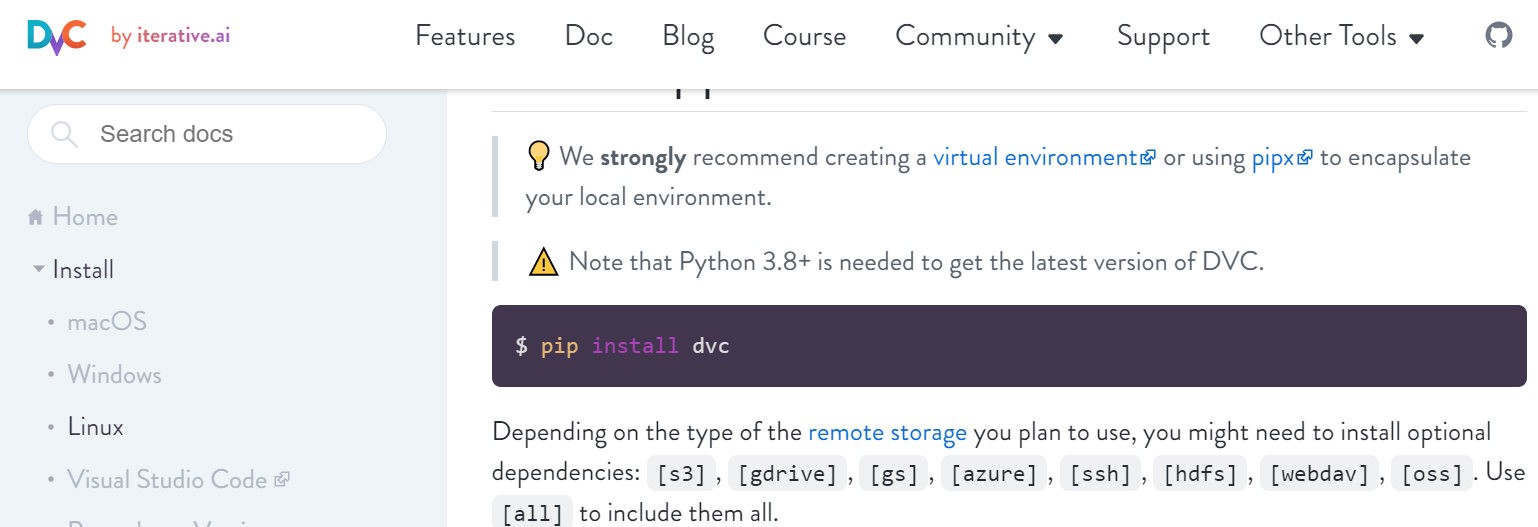
или опубликовать изменения

**git push ssh://< ACCESS\_TOKEN>@github.com/<GIT\_USERNAME>/<REPOSITORY\_NAME>.git**

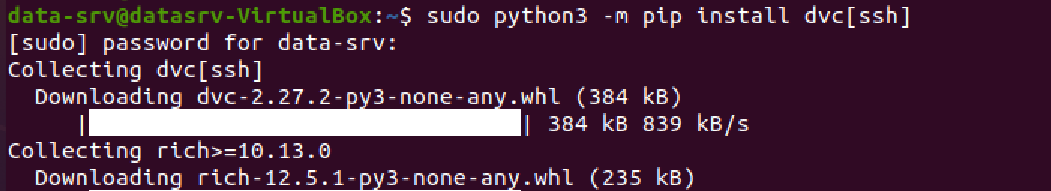
Например

**git push ssh://git@github.com/CyberLympha/MLOPS-course.git**

Теперь нам потребуется настроить dvc для работы с датасетами. Официальная страница dvc находится по адресу <https://dvc.org>, там же можно найти инструкции и необходимое программное обеспечение для установки для соответствующей операционной системы. Также можно осуществить установку с использованием стандартного установщика pip. Рекомендации по такой установке приведены на <https://dvc.org/doc/install/linux>



Мы будем использовать в качестве хранилища локальный сервер с доступом по ssh, поэтому при установке укажем соответствующую опцию dvc[ssh].

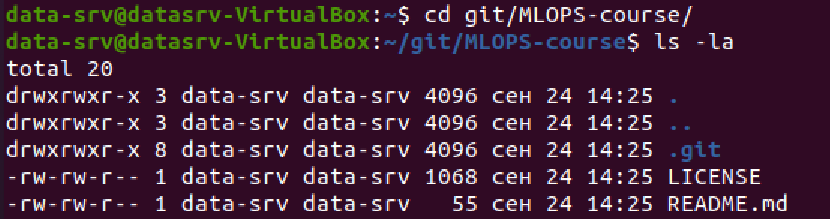


Инициализация проекта осуществляется командами

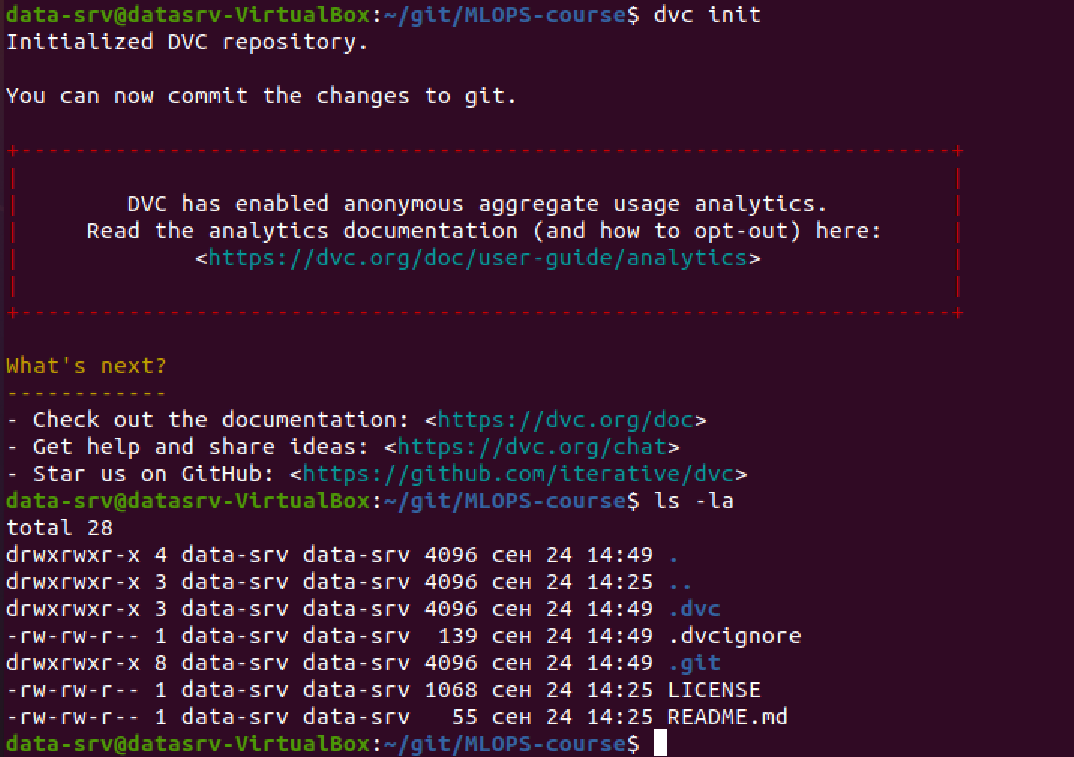
**git init**

**dvc init**

Однако мы уже клонировали git репозиторий и в нем находится скрытая служебная папка .git, поэтому необходимости выполнять команду git init нет.



Выполним команду dvc init и посмотрим на изменения в структуре репозитория



Видим, что в репозитории теперь созданы директорий .dvc и файл .dvcignore, являющиеся аналогами .git и .gitignore.

Возможно, что для корректной работы вам потребуется добавить конфигурационные данные пользователя, это можно сделать следующим образом

**git config user.email "\*\*\*\*\*\*\*\*\*\*\*\*@\*\*\*\*.\*\*"**

**git config** [**user.name**](http://user.name/) **"\*\*\*\*\*\*\*\*\*\*"**

Вы можете создать набор данных и проиндексировать его командой

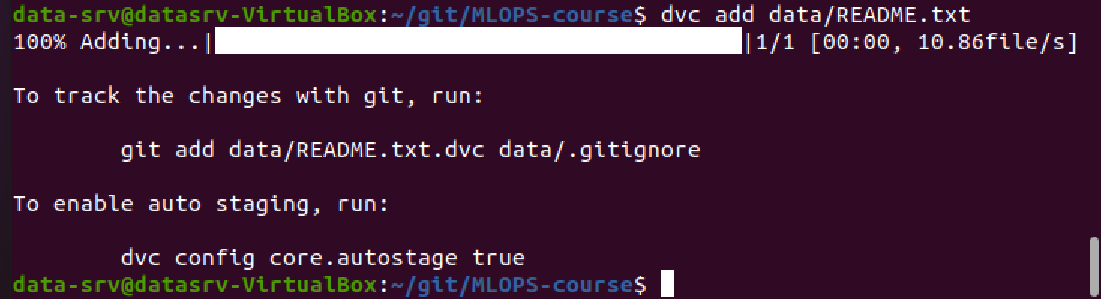
**mkdir data**

**cd data**

Для примера создайте с помощью любого текстового редактора файл README.txt и добавьте его в список для коммита.

**vi data/README.txt**

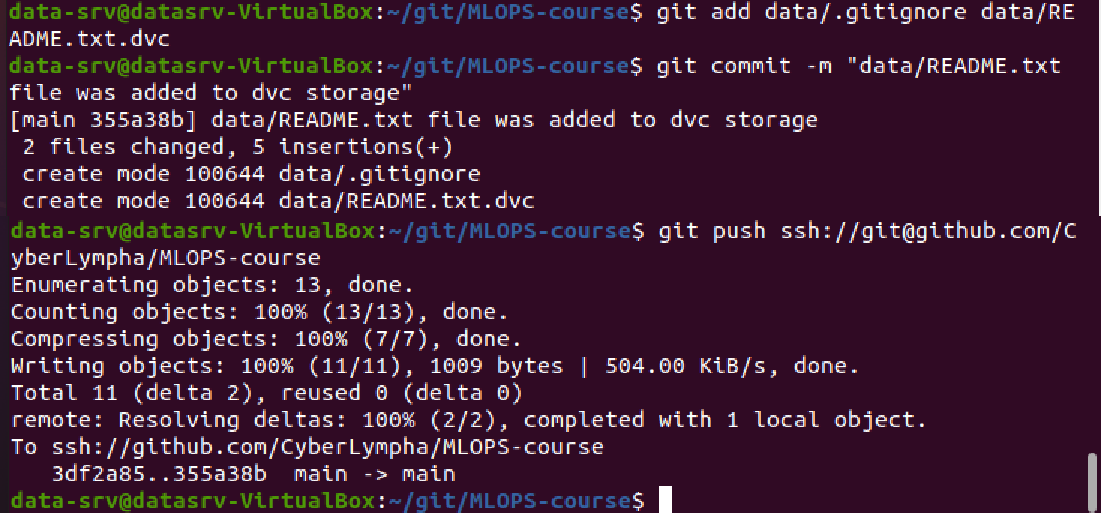
**dvc add data/README.txt**

****

Теперь можно проверить, что утилита dvc создала служебные файлы /data/.gitignore, который содержит список того, что не надо публиковать в git, в данном случае это файл README.txt. Также был создан README.txt.dvc файл с макроинформацией, касающейся публикации файла README.txt в dvc хранилище. Именно этот \*.dvc файл сохраняется в репозиторий git.

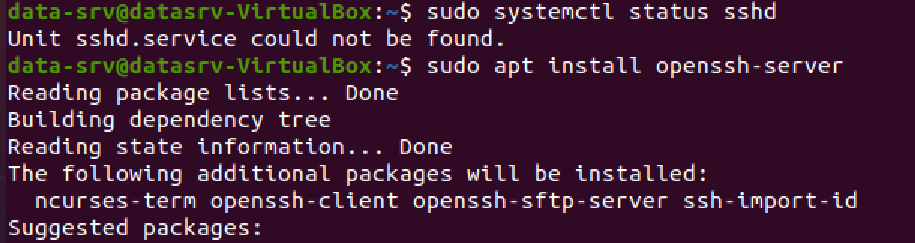
****

Файл .gitignore и \*.dvc файлы нам надо опубликовать в git

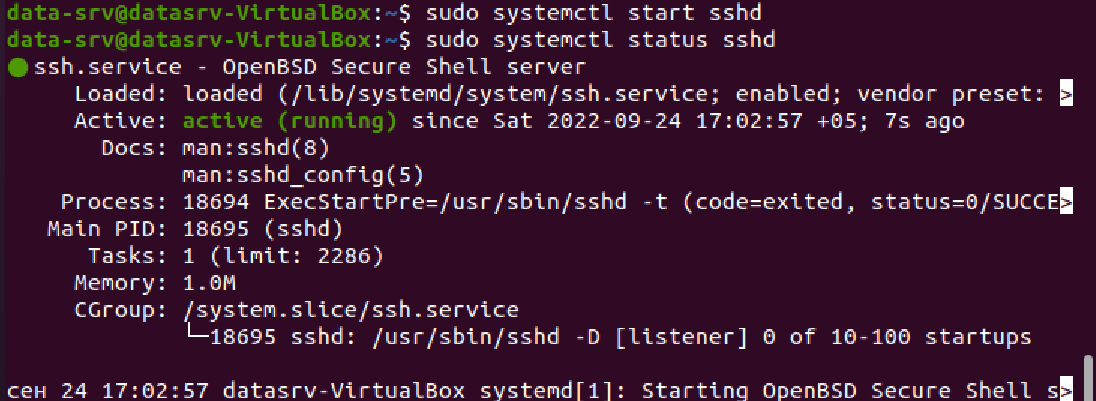


Теперь нам необходимо выполнить важную часть настройки инфраструктуры, а именно, организовать работу хранилища данных, которое будет представлять собой локальный сервер data\_srv на котором необходимо запустить ssh сервер для организации удаленного доступа с других серверов. Кроме локальных серверов с доступом по ssh, dvc поддерживает интеграцию с другими видами хранилищ, например Amazon S3, Google Drive, Microsoft Azure и другие.

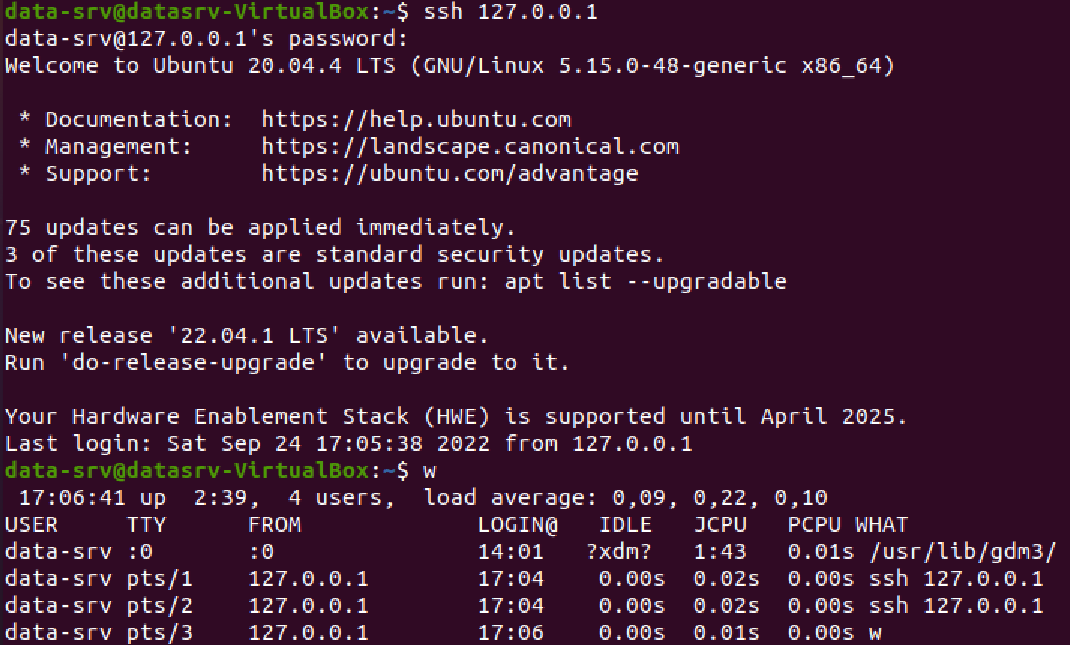
Для организации работы сервера sshd на data\_srv необходимо установить программное обеспечение OpenSSH



После чего мы можем запустить сервер



Чтобы проверить корректную работу сервера sshd на data\_srv вы можете, находясь локально на data\_srv, осуществить подключение к сервису sshd по адресу 127.0.0.1, который является служебным, зарезервированным, и означает «этот сервер».



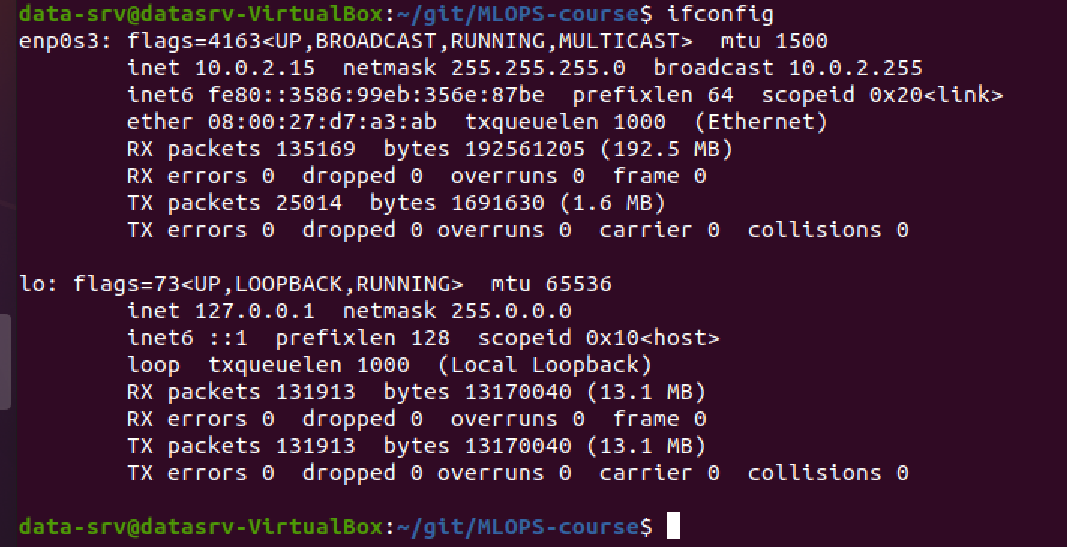
Теперь можно закрыть сессию ssh, набрав команду exit



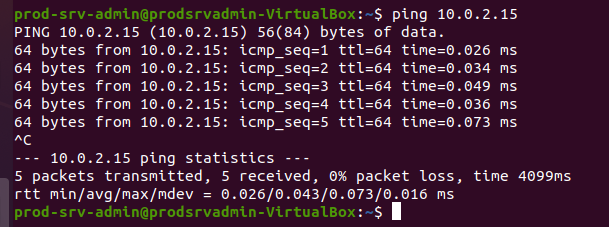
Вы создали локальный ssh сервер на data\_srv для того, чтобы организовать на нем хранилище данных с доступом через dvc.

Давайте добавим это удаленное хранилище в наши данные dvc. Конечно, теперь нам уже придется использовать глобальный IP адрес, так как мы предполагаем, что доступ к хранилищу должен будет осуществляться с различных адресов.

Чтобы проверить доступные интерфейсы на сервере надо воспользоваться установленной нами в юните 2 командой ifconfig, с ее помощью мы видим открытые интерфейсы

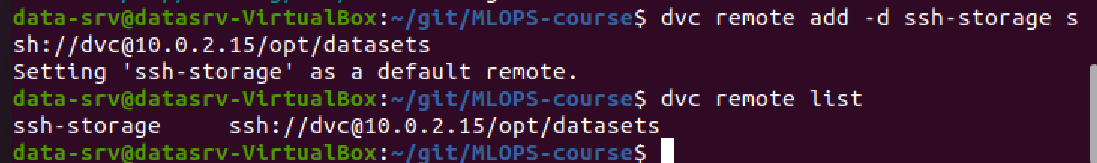


Поскольку мы сразу настраивали виртуальный сетевой адаптер таким образом, чтобы виртуальные машины могли общаться друг с другом по IP адресам, мы можем выполнить команду ping для проверки доступа с сервера prod\_srv на сервер data\_srv и убедиться в наличии сетевой связности



У нас все готово для того, чтобы организовать хранилище dvc на локальном сервере data\_srv с доступом по ssh, для этого надо выполнить команду

**dvc remote add -d ssh-storage ssh://dvc@10.0.2.15/opt/datasets**

****

**git config user.email "\*\*\*\*\*\*\*\*\*\*\*\*@\*\*\*\*.\*\*"  
git config**[**user.name**](http://user.name/)**"\*\*\*\*\*\*\*\*\*\*"**

**git commit .dvc/config -m "Configure remote storage"**

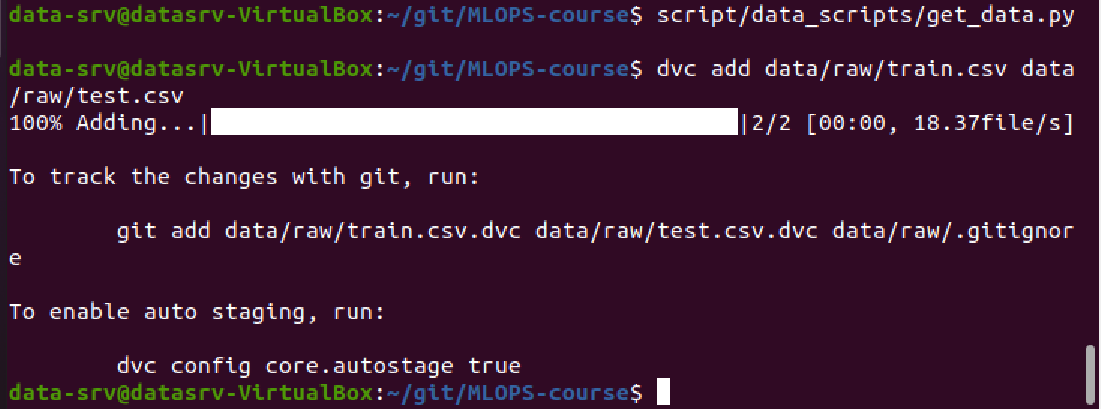
****

**dvc remote modify ssh-storage user “тут надо указать пользователя на data-srv”**

**dvc remote modify ssh-storage port 22**

**dvc remote modify --local ssh-storage password “тут надо указать пароль на data-srv”**

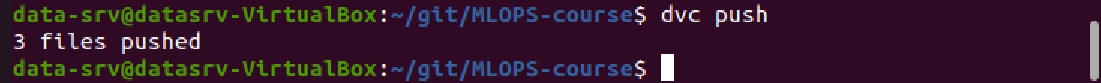
Теперь создадим файлы датасета для публикации и добавим их под контроль dvc

****

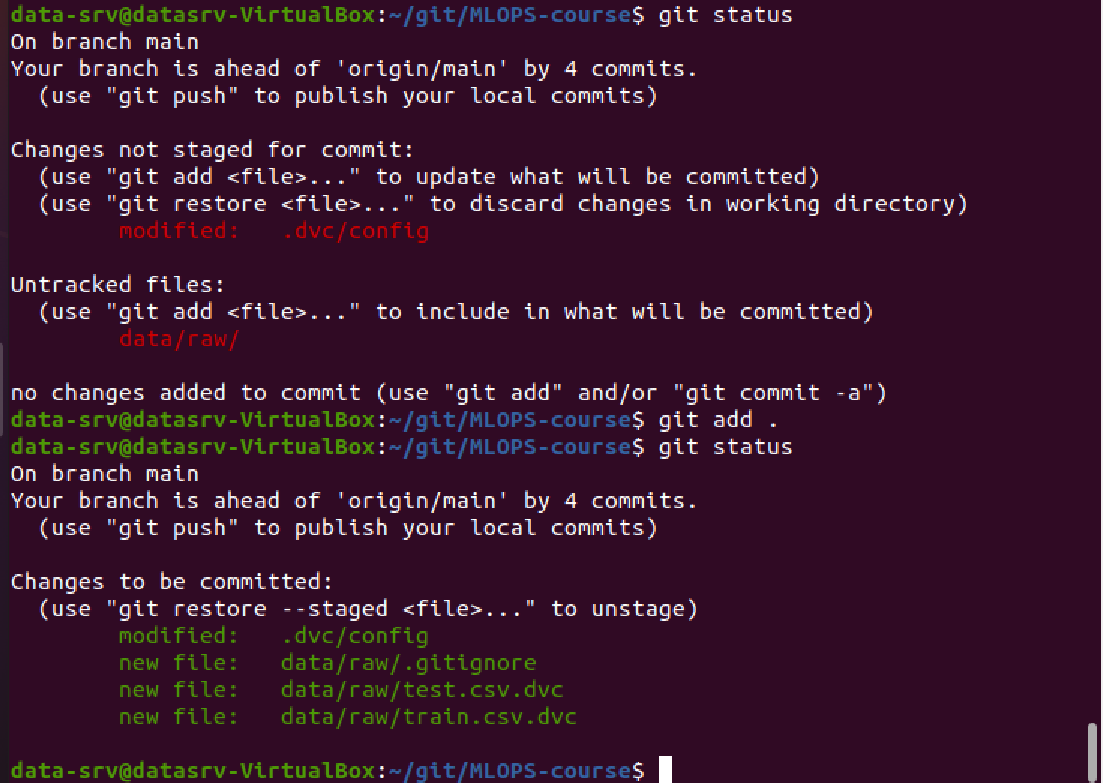
Первая попытка закончится неудачей, так как мы пытаемся писать файлы в директорий /opt/datasets, владелец которого root

****

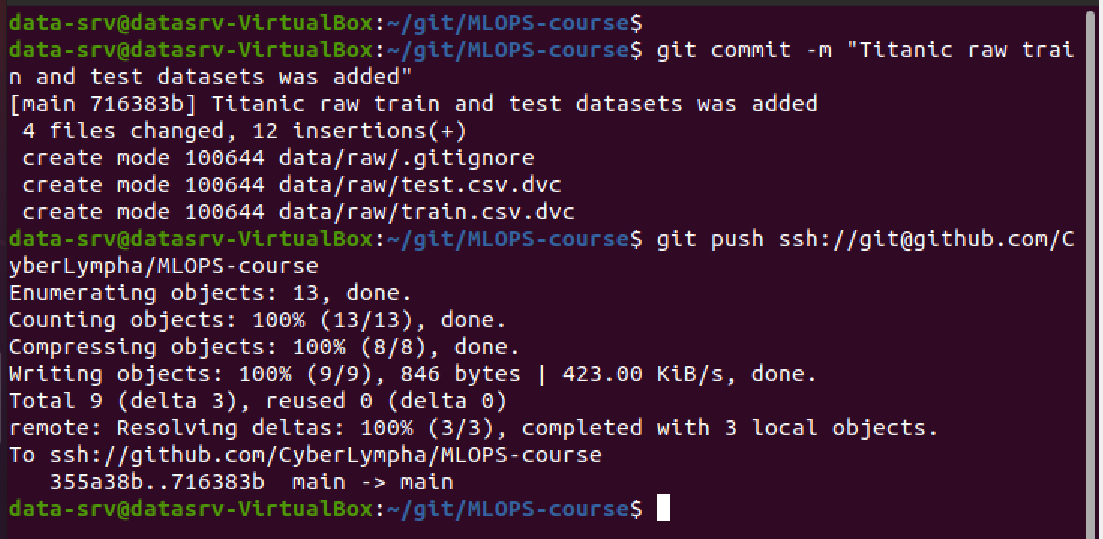
После этого команда dvc push отрабатывает успешно



Также надо не забыть сделать commit в git, чтобы сохранить файлы \*.dvc с макроинформацией о публикации файлов в dvc.

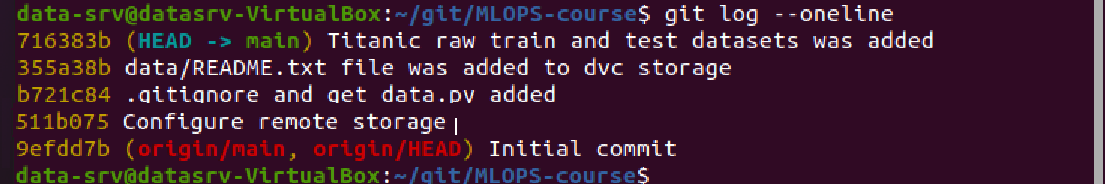


Видим, что нужные нам файлы готовы к публикации в репозитории.



Теперь, если вам понадобится вернуться к предыдущим версиям наборов данных, это можно сделать с помощью команд

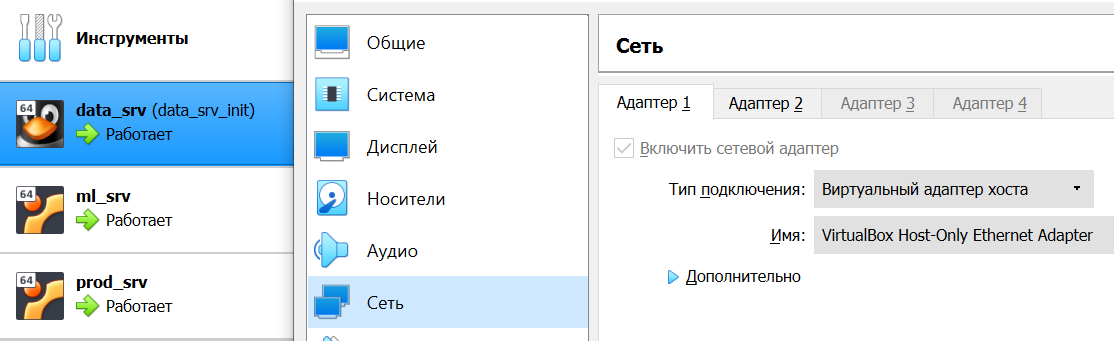
**git log –oneline**

****

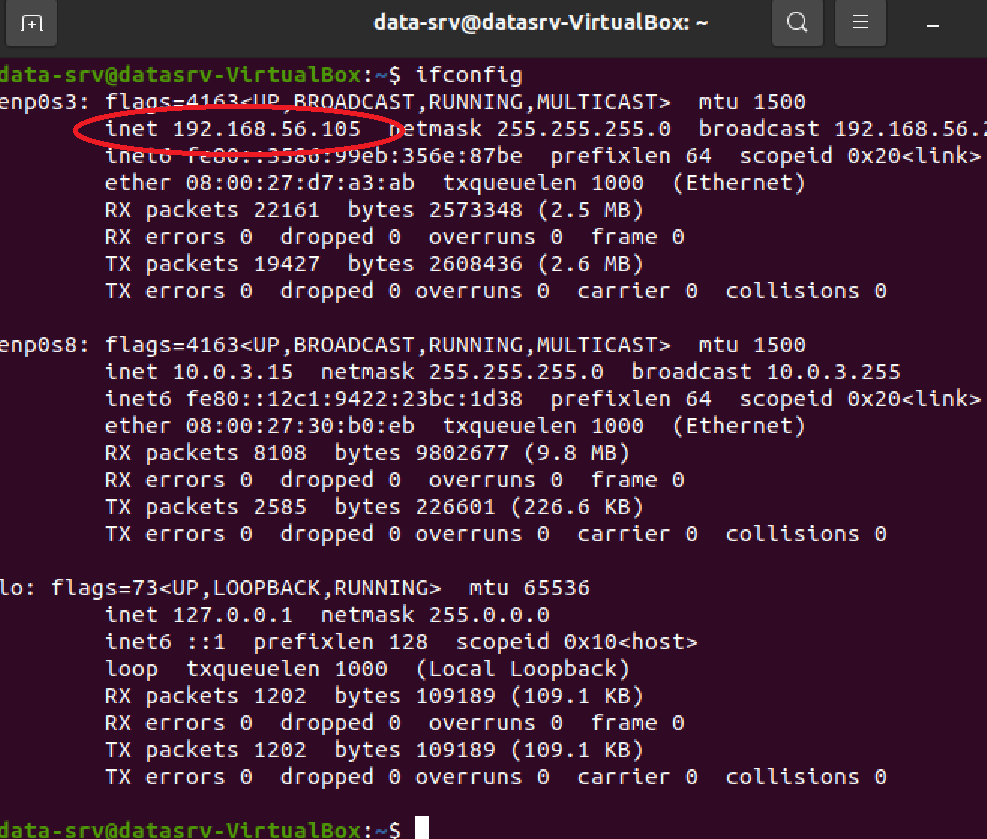
**git checkout HEAD^1 data/README.txt.dvc**

**dvc checkout**

В заключение отмечу, что наша конфигурация с использованием виртуальных машин имеет ряд особенностей, в частности, для связи виртуальных машин необходимо использовать соответствующую настройку виртуальных сетевых адаптеров, а именно, обязательно выбрать в «Типе подключения» значение «Виртуальный адаптер хоста»

****

Иначе вы не сможете получать доступ до этого локального хранилища dvc с других виртуальных машин, например, ml\_srv. При этом будет использоваться тот IP адрес, который «виден» с других виртуальных машин



Итак, теперь у нас есть инструмент dvc для того, чтобы управлять изменениями в данных, по аналогии с тем, как используется git для контроля изменений в разрабатываемом программном коде. Однако этим полезные возможности dvc не ограничиваются, в следующих юнитах мы будем использовать dvc также для управления экспериментами и контроля метрик.

## JupyterHub для исследователей

Программные продукты, входящие в экосистему Jupyter, очень популярны среди исследователей данных и инженеров машинного обучения. Интерактивные jupyter ноутбуки являются удобным инструментом для изучения данных, проверки гипотез, поскольку позволяют в интерактивном режиме направлять команды интерпретатору и видеть результат их выполнения, в том числе удобные графики, таблицы. По сути, jupyter ноутбук является цифровым аналогом “рабочей тетради ученого”, которая дает возможность вести интерактивные исследования, включая вычисления, эксперименты, графики. JupyterHub является многопользовательским сервером для пользователей Jupyter ноутбуков, удобным для командной работы.

На сервере ml\_srv предполагается активная работа исследователей данных и инженеров ML, поэтому для организации работы нам понадобится установить на сервер ml\_srv программное обеспечение JupyterHub, установка и настройка которого описана далее.

Для установки JupyterHub необходимо выполнить команды:

**sudo python3 -m pip install jupyterhub==1.3.0**

**sudo python3 -m pip install jupyter**

**sudo apt-get install npm nodejs**

**sudo npm install -g configurable-http-proxy**

Примечания:

* Если в текущей конфигурации у вас не установлен установщик python пакетов, его можно установить командой

**sudo apt install python3-pip**

* Если при установке jupyterhub не указать версию 1.3.0, то установщик автоматически выберет самую свежую версию, что может привести к проблемам совместимости библиотек, рекомендуем применять указываемые в тексте модуля версии программного обеспечения
* nodejs, Node.js это программная платформа, позволяющая использовать язык программирования javascript для создания серверных приложений, изначально javascript создавался только для использования в браузерах для реализации динамических функций сайтов, Node.js используется в jupyterhub,
* npm это менеджер пакетов, входящий в состав Node.js
* в инструкциях можно встретить **sudo apt-get install npm nodejs-legacy**, однако nodejs-legacy был замещен с nodejs.

После этого уже можно запускать JupyterHub “из коробки”, так как предварительных настроек достаточно для того, чтобы система начала работать. Например, можно вызвать справочник по параметрам:

**jupyterhub --help-all**

Для тонкой настройки конфигурации JupyterHub необходимо редактировать файл jupyterhub\_config.py, который создается в результате выполнения команды

**jupyterhub --generate-config**

В результате создается текстовый файл конфигурации jupyterhub\_config.py, который содержит закомментированные настройки (то есть данные, перед которыми стоит знак “#”, что означает комментарий в python) и их описания на английском языке. Можно задавать при запуске jupyterhub полное имя конфигурационного файла, но лучше хранить файл конфигурации в /etc/jupyterhub/.

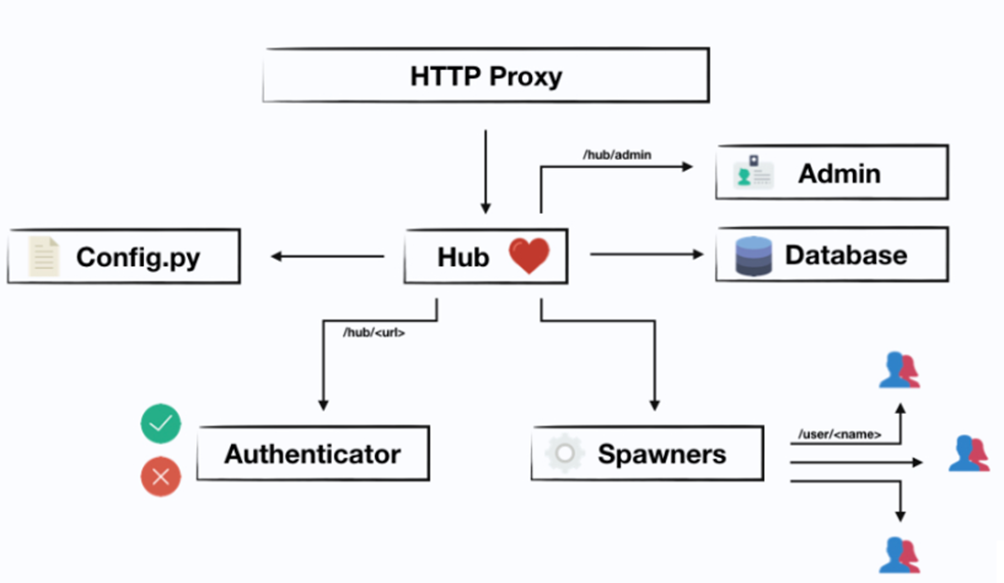
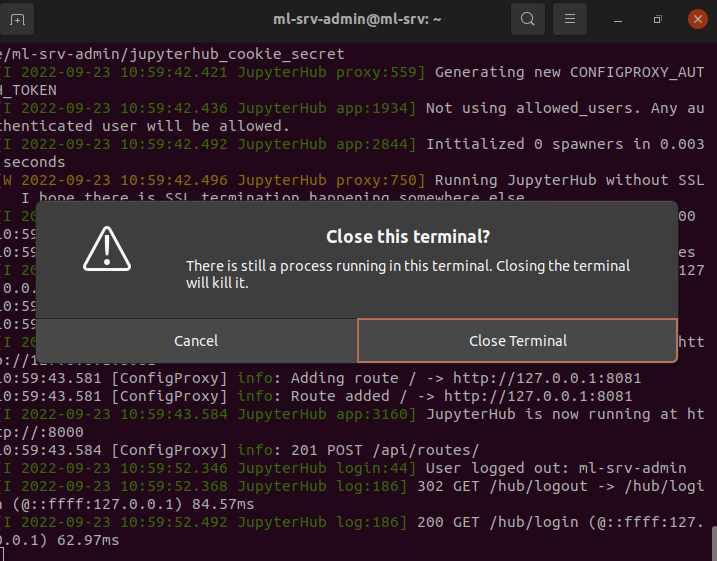


Рисунок “Схема работы JupyterHub”

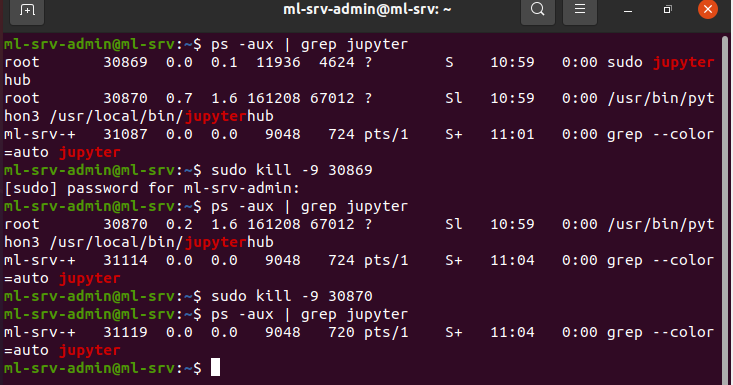
При запуске системы и подключении пользователей возможны проблемы, поэтому необходимо внимательно читать log файлы, в которые пишется вся информация о работе всей системы и конкретного приложения. Конечно, если вы запустите программу в консольном окне и потом закроете это окно – сеанс связи прервется.



Либо может случиться еще интереснее – процесс jupyterhub не остановится после закрытия консольного окна и его придется искать отдельно



Команда ps нам дает информацию о запущенных процессах, из которых мы оставляем только те, которые содержать “jupyter” в описании. После этого мы знаем, например, что запущены два процесса, связанные с jupyterhub, и их идентификаторы процессов PID имеют значения 30869 и 30870. Зная эти идентификаторы можно вручную остановить процессы, послам им сигнал kill -9, для этого нужны необходимые пользовательские привилегии, использовать эту возможность надо осторожно.



Поскольку jupyterhub является сетевым сервисом, он использует различные сокеты для своей работы. Сокет – это комбинация сетевого адреса и порта, различные сервисы запускаются в системе на своих уникальных портах, для большинства сервисов номера портов зарезервированы, эту информацию можно посмотреть в файле /etc/services. Чтобы проанализировать какие процессы занимают какие порты удобно использовать утилиту netstat, входящую в пакет программ net-tools, установку которого мы рассмотрели в юните 2. С помощью netstat можно проанализировать какие порты заняты какими процессами. Например, может так случиться, что процесс node останется работать в фоновом режиме и занимать порт 8001, даже если jupyterhub будет остановлен. В этом случае повторный запуск jupyterhub приведет к ошибке, так как необходимый порт уже занят другим процессом. Обнаружить это можно следующим образом



Теперь из этой информации мы понимаем, что порт 8001 занят процессом node, который имеет идентификатор 32069 и, при необходимости, также можем остановить этот процесс.

Самым удобным средством для администрирования программ через терминальные окна, позволяющим избежать вышеуказанных проблем, является мультиплексор терминалов tmux. Для установки tmux надо выполнить команду

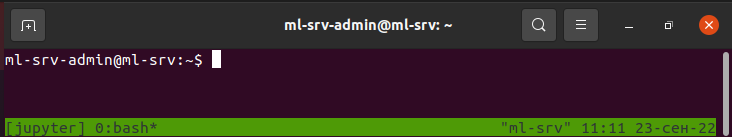
**sudo apt install tmux**

tmux позволяет управлять пользовательскими сессиями, при этом запоминает их состояние, даже если пользователь закрывает консольное окно, при повторном открытии окна вы снова увидите ту сессию, которая была.

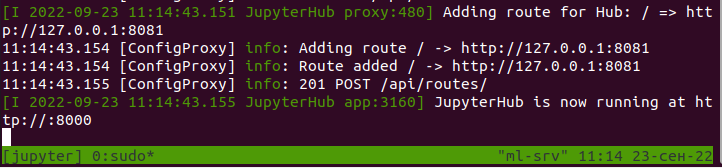
Например, создать новую сессию с именем “jupyter”, в которой мы будем запускать jupyterhub, можно следующей командой tmux new -s “имя сессии”, например

**tmux new -s jupyter**

после чего вы увидите следующее окно:



В этом окне можно запустить jupyterhub в фоновом режиме, не забудьте для этого поставить знак «&» в конце команды в консоли, после этого в консоли начнут появляться «логи», то есть сообщения о работе программы. Эти сообщения, как правило, очень информативны и позволяют сделать вывод о текущем состоянии программы и имеющихся ошибках, а также их причинах.



Теперь вы можете отключиться от терминала с помощью команды

**tmux detach**

После этого терминальные сессии останутся активными, их можно посмотреть с помощью команды

**tmux ls**



и для подключения к сессии снова надо выполнить команду

**tmux attach -t “имя нужной сессии”**

Запускать без SSL JupyterHub в открытой сети небезопасно, поскольку пароли передаются в открытом виде. Для проектов можно использовать коммерческий сертификат SSL, но есть и альтернативные варианты:

* можно сделать доменный https сертификат бесплатно здесь: [www.letsencrypt.org/getting-started](http://www.letsencrypt.org/getting-started)
* сделать самоподписанный сертификат, это подходит для ресурсов без доменного имени, это делается следующей командой:

**sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout «имя ключа».key -out «имя сертификата».crt**

после этого можно запустить jupyterhub

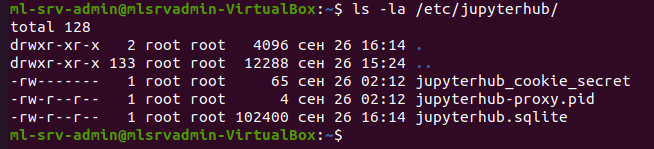
**jupyterhub --ip 0.0.0.0 --port 443 --ssl-key «имя ключа».key --ssl-cert «имя сертификата».cert**

например

**jupyterhub --ip 0.0.0.0 --port 443 --ssl-key /etc/ssl/private/jupyterhub-selfsigned.key --ssl-cert /etc/ssl/certs/jupyterhub-selfsigned.crt &**

Можно использовать разные механизмы авторизации, например авторизацию Github или LDAP.

Для хранения данных о работе JupyterHub, например пользовательских настроек, используется служебная база данных sqlite. Этот файл создается автоматически при первом запуске jupyterhub, находится в директории из которой пользователь запустил jupyterhub, если иное не указано в явном виде в конфигурационном файле.



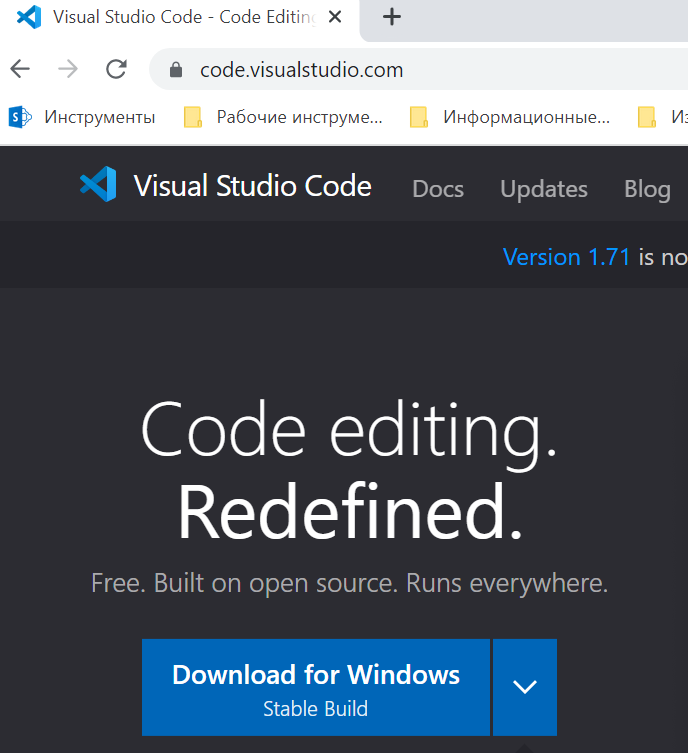
Важно знать, что при существенной переконфигурации системы использование старого jupyterhub.sqlite файла может привести к сбоям.

## VSCode для разработчиков

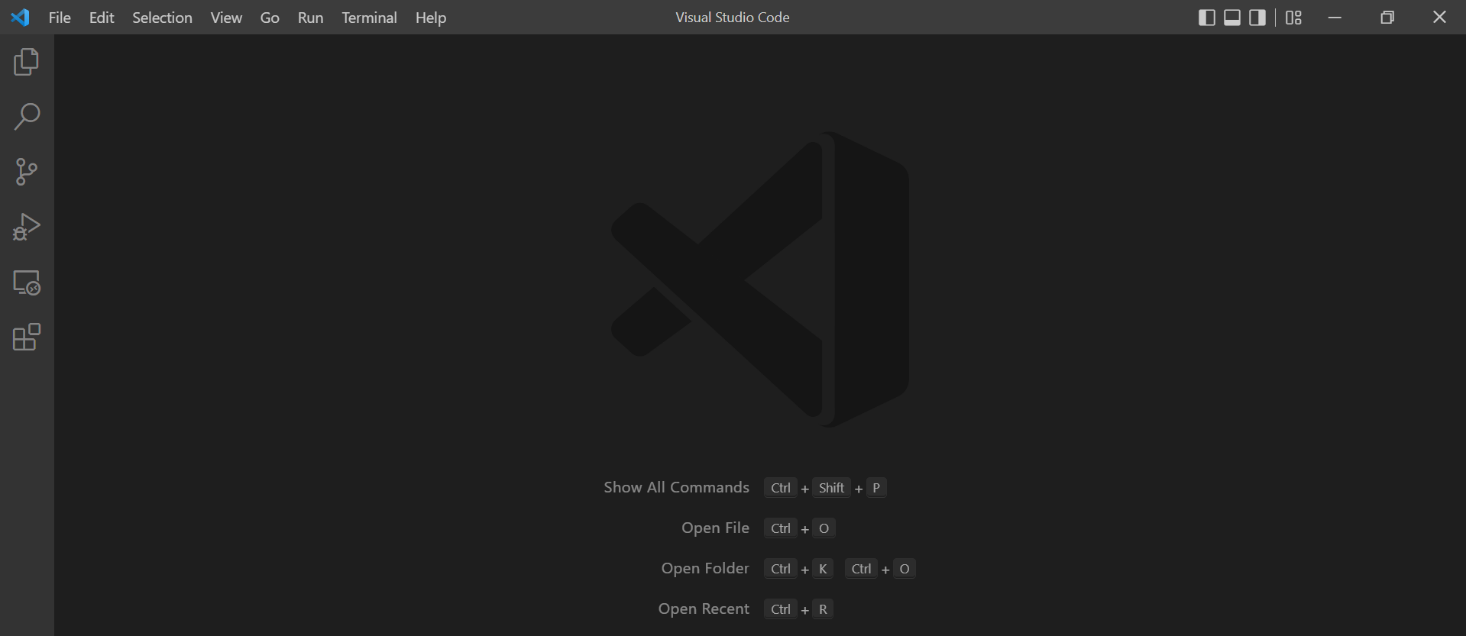
Бесплатное программное обеспечение Visual Studio Code, является удобным и многофункциональным средством разработки, поддерживающим многие полезные для продуктивной разработки функции: автоподстановку, подсказки, проверку синтаксиса и структуры кода, пошаговую отладку и многое другое. Официальной страницей проекта является <https://code.visualstudio.com>.

Установка VSCode осуществляется бесплатно с официального сайта. С помощью VSCode вы можете как разрабатывать программы локально на компьютере, так и подключаться к удаленному серверу, осуществляя редактирование и компиляцию кода непосредственно на самом удаленном сервере. Хорошей альтернативой VSCode является PyCharm, рекомендуем вам ознакомиться также с этим продуктом самостоятельно, официальная страница <https://www.jetbrains.com/ru-ru/pycharm>.

Например, установка и настройка VSCode для Windows осуществляется с использованием дистрибутива с официального сайта



Установка на ОС Windows осуществляется обычным способом. После установки можно добавить ярлык для быстрого запуска на «Рабочий стол» и «Панель программ». После запуска VSCode вы увидите рабочее окно программы

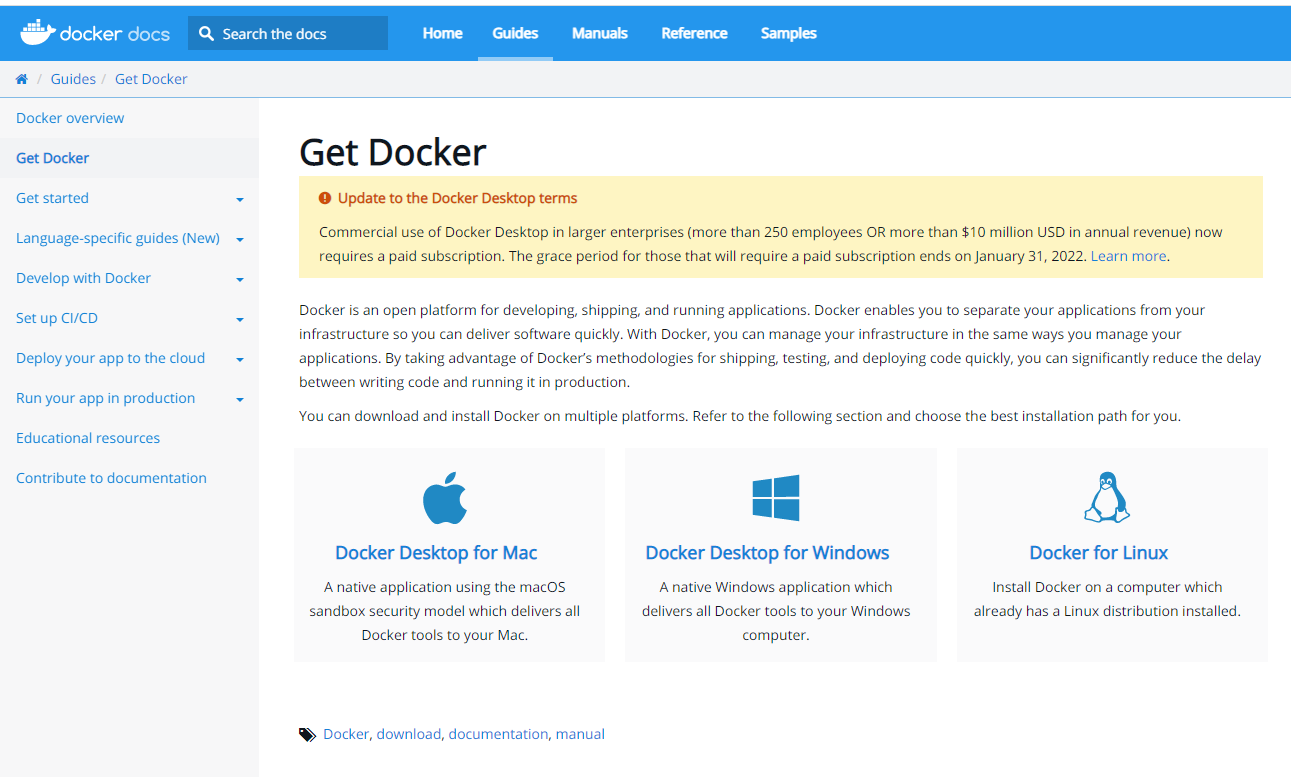


В этом рабочем пространстве можно редактировать файлы проекта локально на компьютере, а также подключать директории, находящиеся на удаленном сервере, осуществляя редактирование и компиляцию файлов удаленно, что является удобным для распределенных команд, работающих на различных серверах.

## docker

На виртуальной машине prod\_srv, на которой предполагается эксплуатация системы, необходимо установить и настроить систему контейнеризации docker. Давайте выполним процедуру установки docker для операционной системы Ubuntu. Установка docker для различных операционных систем подробно описана здесь <https://docs.docker.com/engine/install/>.

Для установки docker для операционной системы Ubuntu необходимо на странице <https://docs.docker.com/get-docker/>



выбрать вариант “Docker for Linux”. После этого переходим к разделу “Server” в котором описаны варианты установки для разных версий операционных систем.

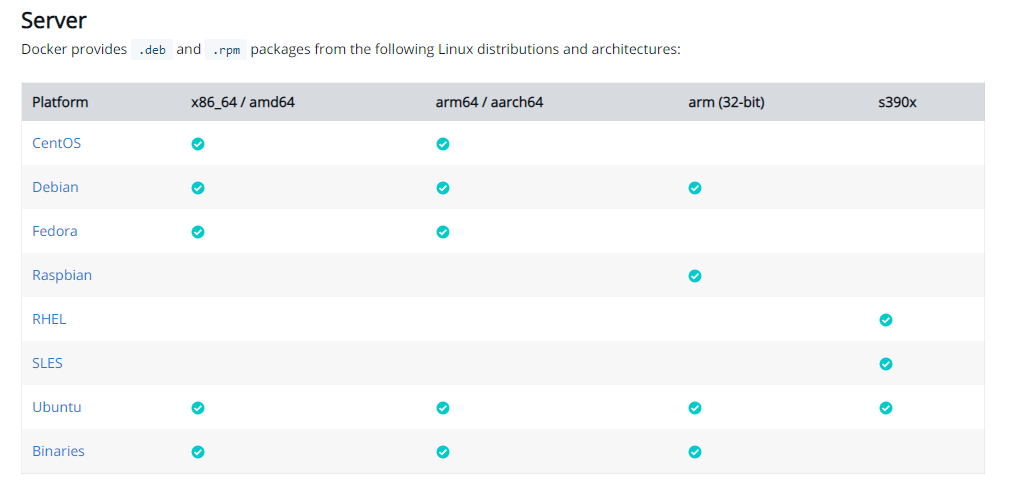
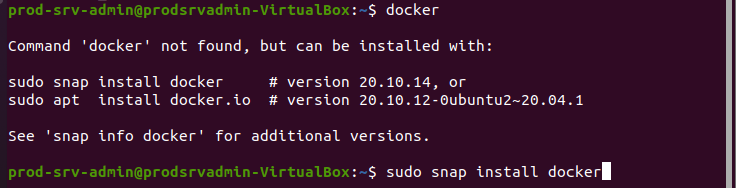


Рисунок “Техническая информация о возможностях использования docker для разных linux операционных систем”.

Поскольку мы решили устанавливать docker для операционной системы Ubuntu, необходимо выбрать соответствующий пункт таблицы после чего появится инструкция для установки. Надо обратить внимание на необходимые параметры операционной системы, в которую мы будем устанавливать docker.

Можно использовать быструю настройку с помощью snap



Другой, более сложный вариант, это процесс установки, описанный в официальной документации на сайте docker.org. Сначала необходимо удалить старые версии docker, установленные в системе

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

Затем обновить установщик apt-get

**sudo apt-get update**

и установить необходимые пакеты для возможности использования HTTPS для установки

**sudo apt-get install ca-certificates curl gnupg lsb-release**

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**

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 команду в командной строке

**docker**

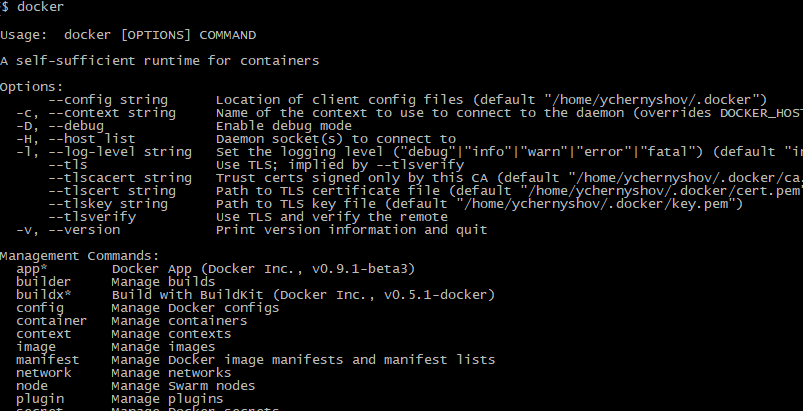
****

Рисунок “Основное окно docker”

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



Проблема заключается в том, что обычный пользователь по умолчанию не имеет доступа к служебному сокету unix:///var/run/docker.sock через который идет 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.

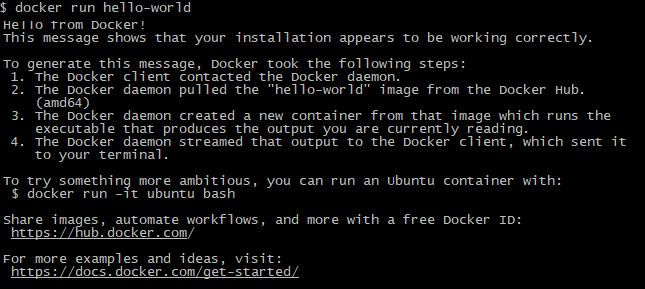


Рисунок “Результат выполнения команды docker run hello-docker”

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!»

## Тест

1. Без какой утилиты не будет функционировать dvc? (0.25)
   1. tensorflow
   2. netstat
   3. **git**
   4. java
2. Какая база данных по умолчанию используется для хранения настроек JupyterHub? (0.25)
   1. postgresql
   2. MongoDB
   3. **sqlite**
   4. hadoop
3. Какие риски возникают, если использовать доступ к JupyterHub по http, а не по https? (0.25)
   1. Такой доступ не будет работать
   2. **Пароли передаются в открытом виде, поэтому могут быть перехвачены**
   3. Доступ по http не поддерживается большинством браузеров
   4. При использовании https JupyterHub работает медленнее
4. Какая более правильная альтернатива существует для использования sudo при работе с docker? (0.25)
   1. Сделать всех пользователей участниками группы sudo
   2. **Создать группу docker и дать ее участникам соответствующие права на чтение и запись для служебных файлов docker**
   3. Использовать python скрипт
   4. Использовать kubernetes

## Итоги/выводы

В этом юните мы подготовили виртуальную инфраструктуру для того, чтобы решать с ее помощью задачи MLOps для нашего проекта. Мы создали три виртуальных машины: для хранения файлов, для обучения модели и для эксплуатации модели. В зависимости от назначения сервера мы установили специальное программное обеспечение, которое понадобится для выполнения задач MLOps. Теперь мы можем переходить к задачам, связанным непосредственно с проектом.

# Модуль 2. Юнит 4. Настройка рабочего окружения проекта.

*Введение:* от того насколько структурированы отдельные элементы проекта зависит удобство работы с ним, что в итоге сказывается на скорости и качестве выполнения отдельных задач. Как правило, проект ML состоит из данных, функций для обработки данных, модели машинного обучения, тестов. В этом юните мы создадим рабочее пространство проекта, стандартное для выполнения задач машинного обучения: обработка данных, проверка гипотез, эксперименты с моделью, анализ результатов, тестирование и вывод модели в промышленную эксплуатацию.

*Содержание:*

## Подготовка структуры директориев

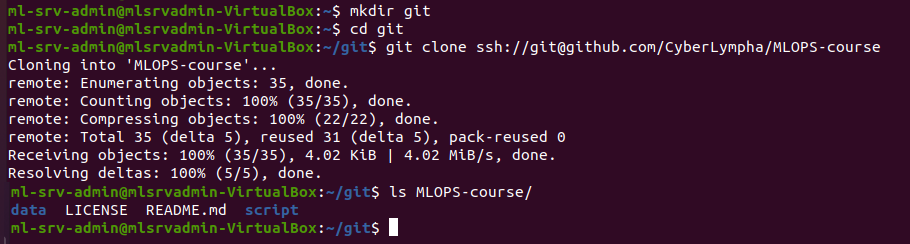
Для проекта машинного обучения создадим файловую структуру, учитывающую особенности проекта, в том числе важнейших артефактов: файлов, данных, моделей, скриптов. Для нашего рабочего проекта предлагается следующая структура папок

* /data - директорий для хранения наборов данных (датасетов):
  + /raw – “сырые” необработанные данные,
  + /baselines - данные, на которых были получены отдельные бейзлайны работы моделей машинного обучения,
  + /final – итоговые данные для использования в обучении модели, не подлежащие изменению.
* /scripts - скрипты для работы с данными, признаками и моделями
  + /data\_scripts
    - получение
    - преобразование типов
    - очистка
    - нормализация
  + /feature\_engineering\_scripts
    - добавление признаков
    - создание новых признаков
  + /model\_scripts
    - подготовка датасета для обучения модели
    - выбор модели
    - обучение модели
    - тестирование, оценка качества работы модели, оценка производительности
    - сохранение результатов экспериментов во внешний файл (база данных, xml, csv или json)
* /settings - папка настроечных файлов
* /docker - папка для организации работы docker

Кроме того, в нашем рабочем директории проекта будет еще две служебные папки, созданные автоматически для взаимодействия с git и dvc.

* .git - папка для организации взаимодействия с серверами версионирования программного кода
* .dvc - папка для организации взаимодействия с серверами версионирования наборов данных (датасетов).

В предыдущем юните мы уже создали частично эту структуру и опубликовали ее в рабочем репозитории на github.com. Давайте теперь клонируем рабочий репозиторий на сервер ml\_srv для того, чтобы организовать на нем работу исследователей данных и инженеров машинного обучения. Для этого нам потребуется установить git и dvc, а также настроить доступ по ssh аналогично тому, как вы это уже сделали для сервера data\_srv. После этого можно осуществить клонирование



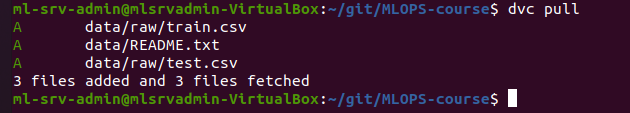
Теперь для работы нужны датасеты, которые находятся в dvc репозитории. Чтобы их достать надо сначала установить dvc на сервер ml\_srv командой

**sudo python3 -m pip install dvc**

затем необходимо осущетсвить локальную настройку пароля, так как пароль для доступа к dvc хранилищу намеренно не хранится в файле .dvc/config из соображений безопасности, делается это командой

**dvc remote modify --local ssh-storage password “тут надо указать пароль на data-srv”**

а после этого выполнить команду dvc pull.

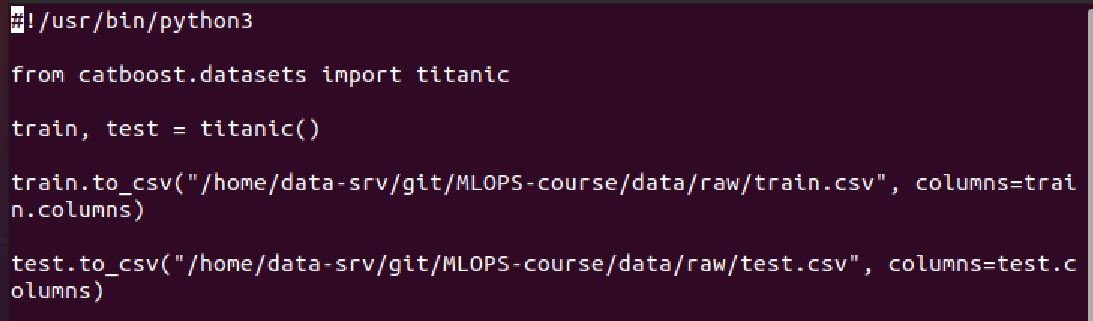


Теперь мы получили инфраструктуру для работы исследователей данных и инженеров машинного обучения на сервере ml\_srv.

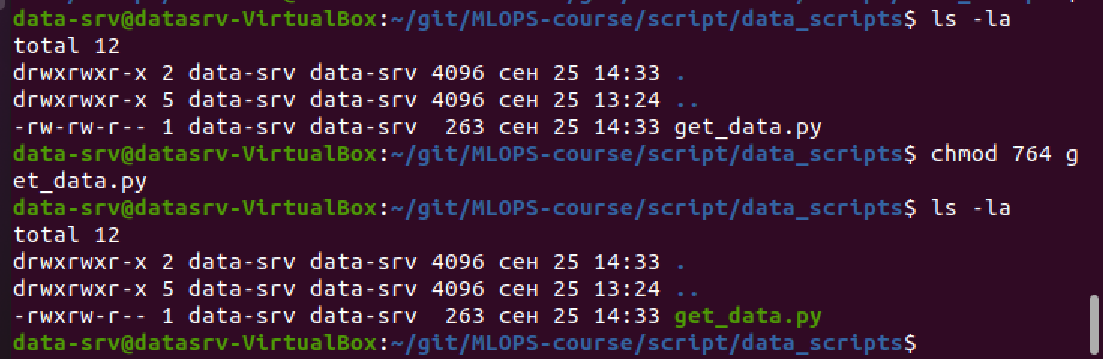
## Подготовка скриптов, создание структуры проекта, загрузка сырых данных

В этой части мы подготовим полезные python скрипты для отдельных этапов проекта, выполним подключение к github и создадим репозиторий проекта в git с необходимой структурой.

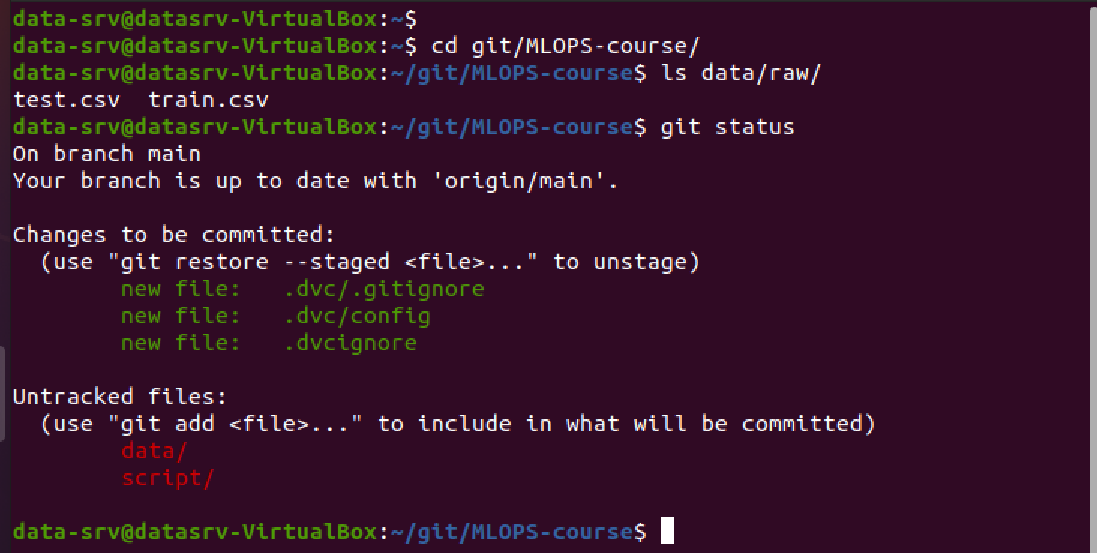
Одним из важных начальных шагов является загрузка сырых данных в хранилище данных на сервере data\_srv, подготовка датасетов, предобработка, сохранение и загрузка рабочего датасета в репозиторий dvc. Поэтому сначала подготовим скрипт, который обеспечивает нам получение сырых данных. Для нашего учебного проекта возьмем данные с соревнования “Titanic Disaster”. Их можно выгрузить с сайта соревнований Kaggle.com, либо создать с помощью библиотеки catboost. Для этой цели подготовим скрипт get\_data.py в директории /scripts/data\_scripts, который будет читать данные с помощью функции catboost.datasets.titanic() и записывать их в директорию /data/raw



Этот скрипт необходимо сделать исполняемым



После выполнения этого скрипта в папке data/raw появляются файлы train.csv и test.csv.



Более того, сейчас мы видим обновления в состоянии git репозитория, к уже готовым к commit служебным файлам dvc (Changes to be commited) добавились папки, в которых мы создали файлы: /data и /script, пока в состоянии «Untracked files».

## Проведение исследований

В этом разделе мы воспользуемся подготовленной инфраструктурой данных и установленным программным обеспечением на сервере ml\_srv для организации работы исследователя данных и программиста. Обычно в задачи этих специалистов входит изучение и обработка данных, извлечение признаков, проведение первичных исследований в Jupyter Notebook, подготовка python кода для проведения экспериментов и обучения модели.

Пользователи data-eng и ml-eng были созданы нами ранее, во втором юните. Пользователь data-eng выполняет роль датаинженера, который исследует данные, устраняет в них ошибки, создает новые признаки и новые датасеты. Пользователь ml-eng это инженер машинного обучения, который, на основе данных, создает модели машинного обучения. Обе эти роли предполагают возможность делать commit в git, поэтому для них необходимо настроить доступ к github.com по ssh. После настройки этой части можно клонировать репозиторий командой

**git clone ssh://git@github.com/CyberLympha/MLOPS-course**

И после этого можно увидеть папки репозитория в Jupyter Notebook.



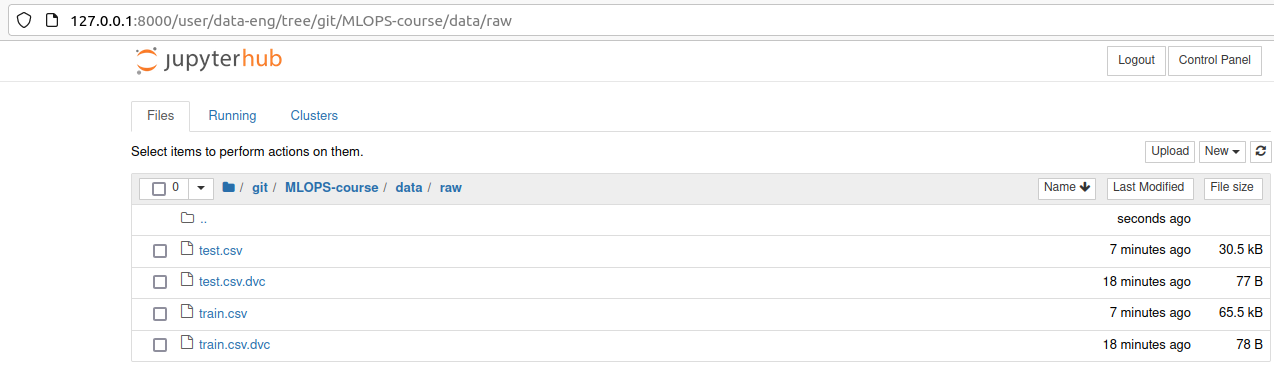
Однако для того, чтобы в нашем репозитории появились файлы датасетов train.csv и test.csv нам необходимо выполнить команду

**dvc remote modify --local ssh-storage password “тут надо указать пароль на data-srv”**

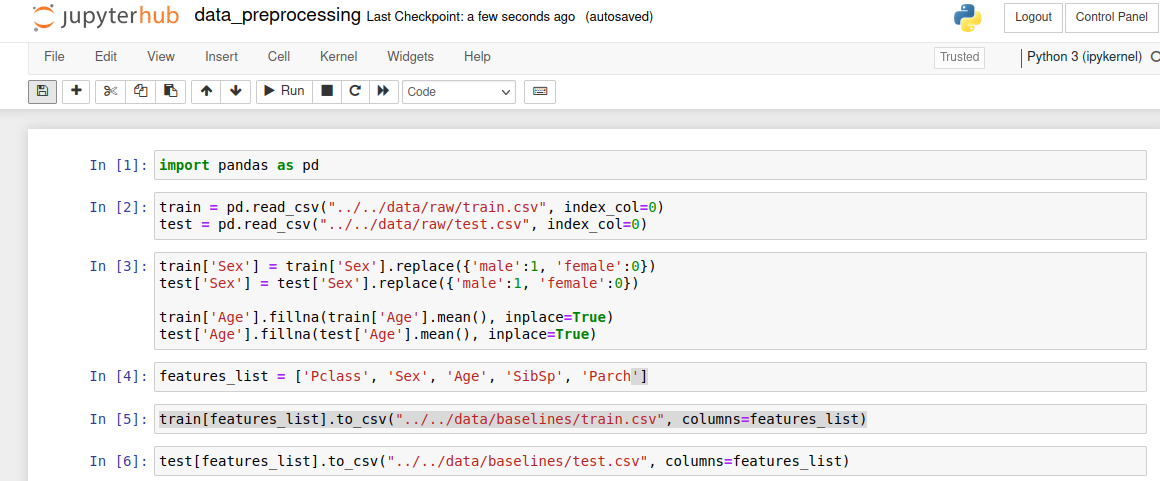
**dvc pull**



Теперь датасеты доступны для экспериментов

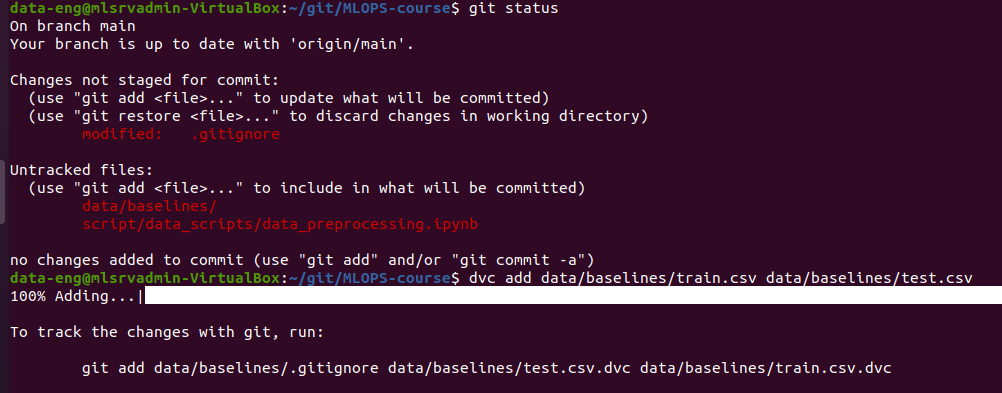


Давайте создадим исследовательский ноутбук для инженера данных, задача которого состоит в изучении исходного сырого датасета и его подготовки для обучения модели



В этом ноутбуке исследователь данных решил заменить текстовые описания пола (признак “Sex”) на числа 0 и 1, заполнил недостающие поля с информацией о возрасте средним значением возраста и оставил далее для исследования только пять числовых признаков, из которых сформировал датасеты train.csv и test.csv, сохранив их в директории /data/baselines/.

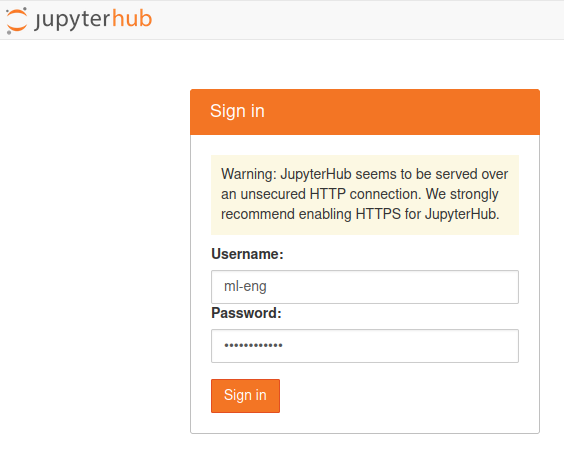
Теперь эти датасеты надо опубликовать в dvc



Мы опубликовали всю необходимую информацию, файлы .gitignore и \*.dvc, в git репозитории, теперь можем осуществить публикацию датасетов в хранилище данных



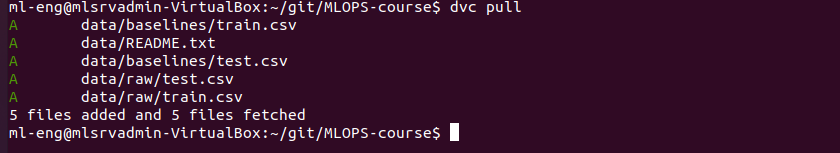
Теперь эти датасеты может использовать инженер машинного обучения, чтобы построить какие-то простые модели и проверить гипотезы перед тем, как запускать объемные вычисления на больших данных. Для этого создадим пользователя ml-eng и наделим его такими же правами, что и data-eng. Для работы в JupyterHub пользователю ml-eng необходимо залогироваться под своим паролем



После этого пользователь имеет доступ к файлам проекта и может создавать jupyter ноутбук для экспериментов с моделями. Однако перед этим ему необходимо взять файлы датасетов из dvc репозитория. При этом, ему нужны последние файлы, обработанные дата инженером.



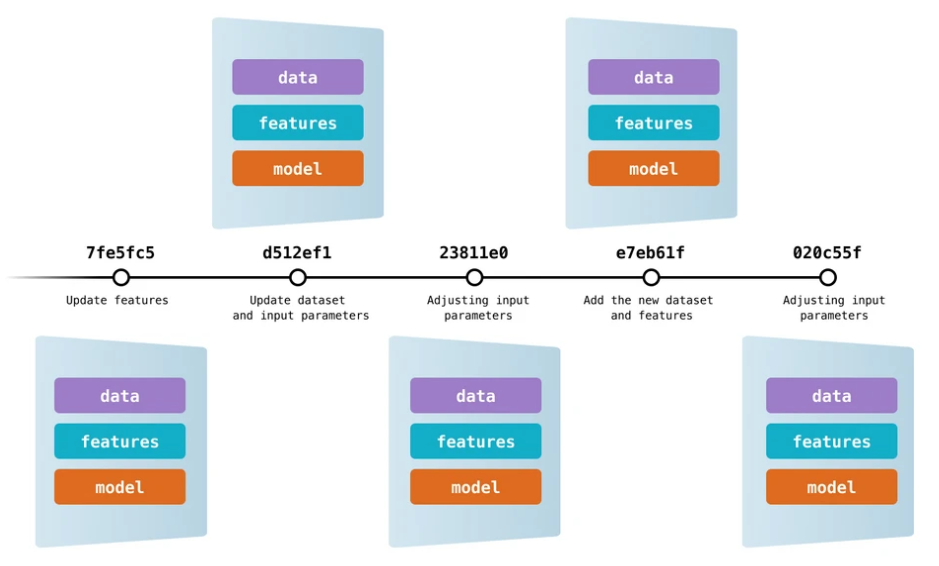
После выполнения команды dvc pull нужные файлы появляются и с ними можно работать



Исследователь на этом этапе обычно проверяет несколько распространенных моделей машинного обучения и получает начальные бейзлайны для продолжения работы.

## Обучение модели

В этом разделе мы разберем заключительную часть разработки модели машинного обучения. Последовательно мы выполним загрузку данных через dvc, обучим модель и сохраним артефакты обучения - метрики. Утилита dvc позволяет эффективно работать с сочетаниями «данные»-«модель»-«метрики».



c официального сайта dvc.org

Для запуска обучения с контролем выходных артефактов (флаг -o) и данных на входе (флаг -d) необходимо запустить скрипт

**dvc run -n train -d train.py -d data \**

**-o model.h5 -o bottleneck\_features\_train.npy \**

**-o bottleneck\_features\_validation.npy -M metrics.csv \**

**python train.py**

Эта команда создаст пайплайн машинного обучения с именем name.

## Тест

1. Какие роли участников команды были задействованы в юните? (0.25)
   1. **ML инженер**
   2. SCRUM мастер
   3. **инженер данных**
   4. постановщик задачи
2. Какие задачи могут быть решены в проекте с помощью JupyterHub? (0.25)
   1. **Первичный анализ и обработка данных**
   2. Автоматизация машинного обучения
   3. **Проверка гипотез**
   4. Вывод в промышленную эксплуатацию
3. Какой флаг используется в dvc для указания файла метрик качества работы модели в dvc run? (0.25)
   1. -metrics
   2. **-M**
   3. -Q
   4. -quality-report-file
4. Какие преимущества использования PyCharm? (0.25)
   1. **Удобная пошаговая отладка**
   2. **Проверка синтаксиса**
   3. **Автоподстановка**
   4. **Работа с удаленными серверами**

## Итоги/выводы

В этом юните с использованием созданной инфраструктуры мы приступили к реализации типовых задач участников проекта машинного обучения: сбор и преобразование данных, проверка гипотез, обучение модели. Эти действия были продемонстрированы на простом примере, однако являются типовыми и применимы и к более сложным задачам.

# Итоги/выводы по модулю

В этом модуле мы решали задачу: создать инфраструктуру для небольшого проекта машинного обучения, включающую инструменты для работы с данными и проведения исследований, средства автоматизации, создание производственного окружения и запуск модели в эксплуатацию. Эта общая задача была решена с помощью отдельных частных подзадач:

1. развертывание виртуальных машин,
2. установка инструментов для участников команды
   1. python как основной язык программирования
   2. инструменты для работы с виртуальными окружениями
   3. JupyterHub для командной работы исследователей в Jupyter
   4. VSCode для разработчиков
   5. Docker для вывода модели в эксплуатацию
3. создание инфраструктуры для хранения и контроля версия для данных и программного кода, настройка git и dvc
4. организация работы участников проекта с помощью dvc
   1. препроцессинг данных
   2. трекинг экспериментов
   3. контроль метрик
   4. версионирование моделей

# Термины

|  |  |
| --- | --- |
| CI/CD/CT | Continuous Integration/Continuous Delivery/Continuous Testing – непрерывная интеграция (добавление новых функций в разрабатываемое программное обеспечение), непрерывная доставка (вывод разработанного программного обеспечения в производственную среду) и непрерывное тестирование, методы автоматизации в процессе разработки программного обеспечения |
| MLOps | Machine Learning Operations: архитектура и набор правил и инструментов, позволяющих автоматизировать задачи проекта машинного обучения |
| DevOps | Development Operations: архитектура и набор правил и инструментов, позволяющих автоматизировать задачи проекта разработки программного обеспечения |
| git | Инструмент для версионирования данных |
| dvc | Data version control |
| ssh | Security Shell: протокол безопасного доступа с одного компьютера или сервера на другой. Представляет собой техническую архитектуру, включающую специальные данные, называемые ключами безопасности, протокол и специальное программное обеспечение. |
| CPU, GPU | Central Processor Unit, Graphics Processor Unit |
| API | Application Programming Interface |

# Список источников

Образовательные и обзорные статьи, справочники, FAQ

1. Статья компании Google об уровнях MLOps в компании, занимающейся проектом ML <https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>.
2. Большое количество учебных статей по linux [https://losst.ru](https://losst.ru/)
3. Инструкция по установке python3 и виртуального окружения <https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server-ru>
4. «Шпаргалка по tmux» <https://habr.com/ru/post/327630>
5. «Установка ssh в ubuntu» <https://losst.ru/ustanovka-ssh-ubuntu-16-04>
6. «Введение в bash» <https://habr.com/ru/post/471242>
7. <https://git-scm.com>
8. https://www.digitalocean.com/community/tutorials/how-to-use-visual-studio-code-for-remote-development-via-the-remote-ssh-plugin-ru

Официальные страницы программных продуктов, инструментов, утилит

1. [https://www.virtualbox.org](https://www.virtualbox.org/wiki/Downloads)
2. [https://jupyter.org](https://jupyter.org/)
3. <https://code.visualstudio.com>
4. https://www.jetbrains.com/ru-ru/pycharm
5. [https://dvc.org](https://dvc.org/)
6. <https://python.org>
7. <https://ubuntu.com>
8. <https://www.docker.com>

Репозитории и источники, использованные в модуле для практической работы

1. https://catboost.ai/en/docs/concepts/python-reference\_datasets\_titanic
2. пример репозитория проекта <https://github.com/CyberLympha>
3. источник датасета MNIST https://www.cs.toronto.edu/~kriz/cifar.html