Модуль 7. Запуск проекта в промышленном окружении.

# Образовательный результат:

После изучения этого модуля Вы научитесь планировать архитектуру и создавать инфраструктуру для всех этапов и задач проекта машинного обучения, начиная от сбора данных и заканчивая выводом решения в производственное окружение и его эксплуатацией. Вы узнаете об основных элементах инфраструктуры и инструментах для их создания. Полученные практические знания вы сможете применить для решения подобных задач в ваших проектах.

# В этом модуле:

В этом модуле описано планирование архитектуры и создание инфраструктуры, аппаратной и программной частей, проекта машинного обучения и его запуска в производственном окружении (production). При выводе модели машинного обучения в производственную среду необходимо учитывать множество деталей, относящихся к разработке программного обеспечения. От качественного планирования архитектуры и реализации инфраструктуры зависит скорость и качество выполнения задач проекта, успешная совместная работа отдельных участников команды, быстрая адаптируемость проекта под изменяющиеся требования. И, наоборот, неудачная архитектура приводит к ошибкам и неэффективному использованию инфраструктуры, нарастанию технического долга. Знания, полученные в этом модуле, позволят вам избежать неудачных решений при создании инфраструктуры проекта машинного обучения.

Темы, изучаемые в модуле:

1. Архитектуры программных проектов, преимущества микросервисной архитектуры для проектов машинного обучения.
2. Сервисы и инструменты в проектах машинного обучения.
3. Пример организации рабочего пространства проекта.
4. Использование средства автоматизации Ansible для автоматизации развертывания инфраструктуры. Системы управления конфигурациями.
5. Пример создания инфраструктуры проекта.
6. Приложения, содержащие дополнительную информацию о типовых компонентах программных систем, для факультативного изучения:
   1. Бэкенд проекта машинного обучения на примере Django.
   2. Внутреннее взаимодействие. API для взаимодействия отдельных компонентов.
   3. Внешнее взаимодействие. Web сервер, задачи, инструменты.
   4. Базы данных.
   5. Установка и настройка JupyterHub, инструмента для командной работы исследователей.

# Модуль 7. Юнит 1. Архитектуры программных проектов. Преимущества микросервисной архитектуры для проектов машинного обучения.

*Введение:* В этом юните мы разберем два варианта архитектур для программного обеспечения, которые применимы и для проектов машинного обучения. Типовые компоненты этих архитектур и инструменты для их реализации вы можете изучить в Приложениях 1-4. Выбор архитектуры зависит от особенностей конкретного проекта и вы узнаете почему микросервисная архитектура является более подходящей для проектов машинного обучения. После изучения этого юнита вы сможете подбирать подходящие для вашего проекта компоненты архитектуры, основываясь на технических требованиях и задачах проекта.

*Содержание юнита:*

Изначально программные системы имели небольшой набор функций, поэтому для их создания не требовалось каких-то специальных сложных архитектур. Появляющиеся новые функции выносились в отдельные модули и библиотеки. Отдельные компоненты взаимодействовали в рамках одной платформы как части одной программы. Такой подход сейчас называется “монолитный”.

Усложнение функциональных спецификаций создаваемых программных систем привело к появлению специализации в разработке разных компонентов: отдельные команды или даже разные организации специализируются на каком то одном компоненте или решении, которое работает независимо от других, взаимодействие происходит по стандартному специфицированному интерфейсу. Такая специализация позволила создавать качественные решения для решения конкретной задачи и их переиспользовать в различных проектах, где требуется эта функция. Большинство продуктов open-source создается в соответствии именно с такой парадигмой. Этот подход называется “микросервисная архитектура”, он имеет множество преимуществ по сравнению с “монолитным” и в настоящее время является основным в реализации сложных многофункциональных решений.

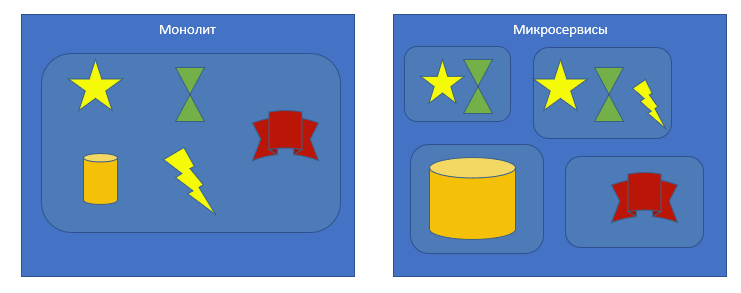
**

Рисунок “Монолитная и микросервисная архитектура”

Отдельным микросервисом, например, может быть сетевая база данных, к которой могут обращаться разные программы, или веб-сервер, обслуживающий входящие запросы от пользователей и распределяющий их по сервисам-исполнителям. Разработчики комплексных систем используют готовые базы данных (например, PostgreSQL) или web-сервера (например, nginx), разработанные другими командами, и включают их в свои решения как отдельные микросервисы в общей архитектуре.

Микросервисная архитектура лучше подходит функционально распределенным командам, когда разные команды решают различные задачи в проекте, при этом работают максимально независимо друг от друга, со своими планами графиками и техническими заданиями. Такие команды создают отдельные сервисы, взаимодействующие между собой с использованием специальных протоколов, например, API (Application Programming Interface). Другие команды не заботятся о внутренней организации используемых ими сторонних сервисов, обращаясь к этим сервисам как к “черным ящикам”, с использованием стандартных команд, ожидая специфицированные в документации API ответы.

При увеличении технических требований микросервисная архитектура позволяет масштабировать решение более гибко. Например, если мы столкнемся с ограничением системы по нагрузке в связи с увеличившимся количеством пользователей, то для монолитной архитектуры нам понадобится расширять всю систему, а для микросервисной архитектуры мы можем выявить отдельные сервисы, влияющие на производительность, и производить увеличение ресурсов отдельно для них. Кроме того, отдельные компоненты решения удобно менять, например, переходить от одной базы данных к другой. Также при выходе из строя одного компонента система в целом продолжит функционировать, а современные средства управления микросервисными архитектурами позволяют быстро перезапустить проблемный сервис и минимизировать время простоя.

Благодаря таким преимуществам микросервисная архитектура гораздо лучше подходит для создания и эксплуатации проектов машинного обучения, чем монолитная. Команда проекта состоит из различных участников или целых отделов, специализирующихся на своих задачах. Исследователь данных не должен понимать тонкости работы backend части системы, а инженер машинного обучения может обсуждать детали создания frontend-интерфейса с дизайнером только как потенциальный пользователь диаграмм и дашбордов. Вопросы производительности, безопасности, распределения нагрузки решают команды, специализирующиеся на этих инструментах. Увеличение производительности аппаратного обеспечения для ускорения работы модели машинного обучения не должно приводить к реинжинирингу всей системы целиком, бэкенд, фронтенд, база данных вообще не должны как-то изменяться после добавления аппаратного обеспечения, например, GPU для ускорения расчетов, эти изменения должны касаться только части машинного обучения, представляющего собой отдельный микросервис. Поэтому для большого проекта, включающего модуль машинного обучения, необходимо применять микросервисную архитектуру, в которой модель машинного обучения является одним из микросервисов в общей архитектуре, взаимодействующим с другими сервисами по стандартному интерфейсу, например, API.

В виде отдельных компонентов-микросервисов в общем решении могут быть представлены базы данных, функции backend, веб-сервер, системы управления очередями задач и многое другое. Популярные технологии для организации работы таких микросервисов это уже известные вам docker и kubernetes. Развертывание модели машинного обучения в docker контейнере в качестве микросервиса имеет ряд преимуществ, например:

* возможность автоматически развернуть нужную нам инфраструктуру с учетом всех требуемых зависимостей,
* контроль ресурсов, расходуемых контейнером,
* возможность масштабировать решение, например запустить несколько экземпляров контейнеров с моделью машинного обучения и распределить между ними нагрузку,
* контроль контейнера, в том числе перезапуск в случае аварии,
* быстрая пересборка контейнера при незначительных изменениях программного обеспечения.

В Приложениях 1-4 к данному модулю вы можете подробнее узнать о наиболее важных типовых микросервисах, используемых в разработке программного обеспечения и, в том числе, в проектах машинного обучения:

* бэкенд,
* API для организации внутреннего взаимодействия между сервисами,
* веб-сервер для организации внешнего взаимодействия,
* базы данных.

Там же для каждого из этих микросервисов разобраны практические аспекты их создания и использования. Рекомендуем вам самостоятельно попробовать создать свое приложение в микросервисной архитектуре, пользуясь инструкциями из Приложений 1-4.

*Тест*

1. Какие бывают виды архитектур программного обеспечения? (0.25)
   1. **монолитная**
   2. монотонная
   3. макросервисная
   4. **микросервисная**
2. Что такое API? (0.25)
   1. **Application Programming Interface**
   2. Activation Programming Interchange
   3. Ask Program Independently
   4. Any Program Interconnection
3. Какие преимущества имеет микросервисная архитектура по сравнению с монолитной? (0.25)
   1. **гибкое масштабирование**
   2. **технологическое функциональное распределение компонентов системы**
   3. возможность использовать open-source
   4. **оптимизация использования аппаратных средств**
4. Почему микросервисная архитектура более предпочтительна для проектов машинного обучения? (0.25)
   1. **в проекте ML работают команды, решающие независимые задачи**
   2. микросервисная архитектура позволяет создавать более простые приложения быстрее
   3. **легче масштабировать решение, например, при увеличении нагрузки на систему**
   4. удобнее версионировать модель машинного обучения.

*Итоги/выводы*

В этом юните вы узнали про два вида принципиально различных архитектур для систем программного обеспечения: монолитную и микросервисную. Каждая из этих архитектур имеет свои сильные и слабые стороны. Из-за специфики для проектов машинного обучения лучше подходит микросервисная архитектура.

# 

# Модуль 7. Юнит 2. Сервисы и инструменты в проектах машинного обучения.

*Введение:* В этом юните мы упорядочим полученные ранее сведения в части используемого в проектах машинного обучения инструментария, а также вспомним типовые инструменты, используемые в проектах разработки программного обеспечения. Эти знания нам понадобятся для того, чтобы научиться планировать правильную архитектуру проекта машинного обучения, которая нужна для создания инфраструктуры проекта, решающей задачи для всех участников проекта.

*Содержание юнита:*

В курсе “Программная инженерия” давалось такое понятие: "Архитектура программного обеспечения описывает способ декомпозиции крупной программной системы на отдельные модули, организации их взаимодействие между собой и с внешним миром.". Таким образом, важно уметь декомпозировать большую систему на отдельные компоненты, с учетом задач, выполняемых этими компонентами, и команд, работающих над ними. Для разных компонентов используются разные инструменты.

В разработке программного обеспечения сложились типовые приемы, помогающие эффективно решать отдельные задачи проекта. Эти приемы относятся как к технологическим, так и к организационным мерам. К технологическим мерам, повышающим эффективность разработки программного обеспечения, относятся:

* использование специализированных инструментов разработки, например, интегрированных программных сред, IDE, включающих автопроверку синтаксиса, подсказки по типовым функциям, графическую “подсветку” конструкций языка, и пр.,
* разделение проекта на микросервисы, независимые между собой компоненты программного обеспечения, для организации работы отдельных команд или предприятий по разным направлениям,
* переиспользование кода, в том числе open-source,
* использование типовых шаблонов проектирования и разработки,
* автоматизация операций проекта для скорейшего вывода в производственную среду с минимумом ошибок,
* и многое другое.

К организационным мерам, повышающим эффективность разработки программного обеспечения, относятся:

* применение новых способов управления проектами, например agile, scrum, kanban, применение специализированных программных средств для этих подходов, например Jira или Trello,
* разделение участников команды по сферам ответственности, организация кросс-функциональных команд,
* новые способы разработки программного обеспечения: например, экстремальное программирование.

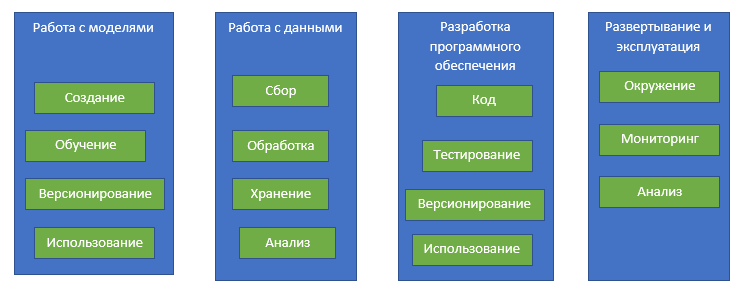
Реализация этих мер сказывается на архитектуре проекта и обязательно учитывается при планировании инфраструктуры и подборе необходимого инструментария для всех участников команды.

Из предыдущих модулей вы уже знаете, что в проектах машинного обучения существует множество специфических задач, которых нет в разработке обычного программного обеспечения. Например, обработка данных и работа с моделями, необходимость управлять конвейерами и более строгий мониторинг качества, как на этапе обучения, так и в производственном окружении. Также вы уже знаете, что команда проекта машинного обучения состоит из множества сотрудников, у каждого свои полномочия и задачи. При этом в команде проекта машинного обучения присутствует много ролей, которых нет в обычных проектах, например,

* инженер данных (Data Engineer),
* исследователь данных (Data Scientist, Data Researcher),
* исследователь моделей машинного обучения (ML Researcher),
* инженер качества данных ( Data Quality Engineer),
* MLOps инженер.

И, конечно, у вас уже не должно быть сомнений по поводу того, что главная цель проекта это вывести разрабатываемую систему как можно скорее и с минимумом ошибок в производственное окружение (production). При этом в проектах машинного обучения важно обеспечить повторяемость результатов и быструю воспроизводимость эксперимента, чтобы команда проекта могла получать качественные и содержательные сведения для улучшения качества работы системы.

Обобщая опыт, полученный в предыдущих модулях, вы сейчас можете собрать в единую схему все сведения о необходимых инструментах и задачах в проекте машинного обучения.



Ниже приведен перечень некоторых наиболее популярных инструментов для решения этих задач:

|  |  |  |
| --- | --- | --- |
| Этап | Инструменты | Описание, ссылка |
| **Разработка** |  |  |
| код | VSCode | <https://code.visualstudio.com>, IDE (Integrated Development Environment, интегрированная программная среда) |
| код | PyCharm | <https://www.jetbrains.com/ru-ru/pycharm>, IDE (Integrated Development Environment, интегрированная программная среда) |
| код | Jupyter | <https://jupyter.org>, IDE (Integrated Development Environment, интегрированная программная среда) |
| код | JupyterHub | <https://jupyter.org>, IDE для командной работы (Integrated Development Environment, интегрированная программная среда) |
| версионирование | git (gitlab, github) | gitlab.com, github.com. Хранение кода и метаданных, версионирование. |
| **Данные** |  |  |
| сбор, обработка | Kafka | <https://kafka.apache.org>. Инструмент для потокового сбора и обработки данных. |
| сбор, обработка | Airflow | <https://airflow.apache.org>. Автоматизация сбора данных и других операций. |
| хранение, обработка | PostgreSQL | <https://www.postgresql.org>. Высокопроизводительная реляционная база данных. |
| хранение, обработка | sqlite | <https://www.sqlite.org>. Небольшая реляционная база данных, невысокая производительность. |
| хранение, обработка | EdgeDB, MongoDB, TerminusDB | Нереляционные базы данных. |
| версионирование | dvc | dvc.org. Версионирование датасетов. |
| **Модели** |  |  |
| Создание, обучение, использование | snakemake | <https://snakemake.readthedocs.io>. Трекинг моделей машинного обучения. |
| Создание, обучение, использование | Dvc | dvc.org. Трекинг моделей машинного обучения. |
| Создание, обучение, использование | Airflow | Автоматизация операций. |
| Обучение | python, numpy, sklearn, Tensorflow, PyTorch | Множество различных библиотек и фреймворков машинного обучения. |
| **Развертывание и эксплуатация** |  |  |
| Организация микросервиса | Django, flask, FastAPI | Фреймворки для веб сервисов. |
| Настройка окружения | Terraform, Ansible | Инструменты для автоматизации настройки окружения для работы системы. |
| Мониторинг | Grafana, Prometeus, ELK | Сбор и мониторинг работы системы в производственном окружении. |
| Оркестрация микросервисов | docker, docker-compose, kubernetes | Запуск и мониторинг сервисов. |
| Организация окружения | nginx, gunicorn | web серверы. Организация доступа извне в сервису, графический интерфейс, распределение нагрузки, обратное проксирование - сокрытие топологии системы для пользователя. |

Конечно, перечень популярных инструментов постоянно изменяется, появляются новые сервисы и выходят из употребления устаревшие и неподдерживаемые. В вышеуказанный перечень не добавлены различные коммерческие инструменты, которых также большое множество. Задача инженера MLOps состоит в качественной и бесперебойной организации работы этих сервисов, поскольку это влияет на качество и скорость проекта. В следующих юнитах мы разберем отдельные сервисы и инструменты.

*Тест:*

1. Какие полезные функции содержат интегрированные программные среды (IDE), позволяющие облегчить и ускорить процесс разработки? (0.25)
   1. **поиск синтаксических и структурных ошибок в коде, связанных с конкретным языком программирования**
   2. **подсказки по синтаксису функций**
   3. **графическая подсветка синтаксиса (ключевые слова, операторы)**
   4. ускоренная компиляция
2. Какие инструменты используются для мониторинга работы модели? (0.25)
   1. **grafana**
   2. pip
   3. **kibana**
   4. sqlite
3. Какие инструменты используются для автоматизированного сбора данных? (0.25)
   1. **kafka**
   2. linux
   3. pycharm
   4. **airflow**
4. Что относится к базам данных? (0.25)
   1. **sqlite**
   2. **postgresql**
   3. kafka
   4. **terminusdb**

*Итоги/выводы:*

В данном юните сделан обзор инструментов и сервисов, используемых различными участниками команды проекта машинного обучения на различных этапах проекта. Назначение этих инструментов необходимо знать для создания и эксплуатации эффективной инфраструктуры проекта, что относится к задачам MLOps.

# 

# Модуль 7. Юнит 3. Организация рабочего пространства проекта машинного обучения.

*Введение:* Любой проект разработки программного обеспечения содержит множество объектов, которые используются в процессе разработки и эксплуатации, а также множество артефактов, являющихся результатом выполнения отдельных задач. Объектами в проекте являются исполняемые файлы, код программного обеспечения, значения переменных среды, библиотеки. Артефактами являются результаты вычислений, динамически генерируемые html страницы для отображения в пользовательском интерфейсе, сгенерированные графики и изображения. В проектах машинного обучения, как вы уже знаете из предыдущих модулей, объектов и артефактов еще больше, чем в обычных проектах. Для проектов разработки комплексного программного обеспечения, включая и проекты машинного обучения, важно создать удобное, непротиворечивое, эффективное рабочее пространство, в котором все сущности проекта будут на своем месте, а разные участники большой команды проекта не будут иметь проблем с доступом к нужным им объектам и артефактам. Хорошо продуманная файловая структура позволяет быстро найти нужный датасет, скрипт или программный код. Правильное распределение инструментария по отдельным серверам, позволяет поддерживать совместимость библиотек и окружений на разных этапах работы с моделью машинного обучения: анализ данных, обучение модели, тестирование, эксплуатация. В этом юните вы узнаете как можно решить эту задачу. Предложенные методы не являются универсальным средством на все случаи жизни, поскольку каждый проект имеет свои особенности. Однако, полученные в этом юните сведения и навыки вы сможете применить к своему проекту, учитывая его особенности.

*Содержание юнита:*

Каждый современный проект разработки многофункционального программного обеспечения имеет сложную структуру, поскольку состоит из множества отдельных функций, модулей, библиотек, сервисов. Это относится и к проектам машинного обучения. Сущностями проекта машинного обучения являются

* наборы данных (датасеты)
  + исходные сырые данные
  + обработанные данные
    - удаленные пропуски (NaN)
    - заполненные пропуски (NaN)
    - добавленные новые признаки
    - измененные типы полей
  + данные для выбора модели
  + данные для обучения модели
  + данные для тестирования модели
  + эталонные данные для проверок
* модели машинного обучения
  + веса моделей
  + гиперпараметры моделей
  + метрики, полученные в результате обучения и оценки моделей
* окружения
  + версии библиотек
  + значения переменных среды окружения
* конвейеры операций

При проходе цикла проекта от работы с данными к эксплуатации модели, эти сущности используются и порождают новые сущности (артефакты). Поскольку в проектах машинного обучения целью является как можно более частое выполнение циклов проекта в различных сочетаниях параметров, количество таких сущностей начинает расти очень быстро и при неправильной организации структуры проекта команда быстро начнет путаться в версиях файлов, датасетов, моделей. Это приводит к ошибкам и непроизводительной потере времени.

Давайте рассмотрим пример правильной структуры данных проекта машинного обучения, а именно, организации структуры хранения артефактов проекта: файлов, данных, моделей, скриптов. Например, ваш проект может иметь следующую структуру папок

* /data - директорий для хранения наборов данных (датасетов):
  + /raw – “сырые” необработанные данные,
  + /external – различные данные из внешних источников (API, базы знаний, экспертные отчеты, данные от других компаний, датасеты из соревнований или научно-исследовательские данные из статей),
  + /preprocessed - “промежуточные” обработанные данные (очистка, добавление, слияние данных из разных датасетов),
  + /baselines - данные, на которых были получены отдельные бейзлайны работы моделей машинного обучения,
  + /articles - данные проекта, которые использовались для написания статей,
  + /competitions - данные для проведения соревнований или хакатонов, которые вы можете организовать для поиска идей решения задачи,
  + /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)
* /venv - папка виртуального окружения
* /settings - папка настроечных файлов
* /docker - папка для организации работы docker
* /git - папка для организации взаимодействия с серверами версионирования программного кода
* /dvc - папка для организации взаимодействия с серверами версионирования наборов данных (датасетов).

Вы уже знакомы с инструментами git для версионирования и управления программным кодом и служебными файлами, а также с dvc для версионирования датасетов. git и dvc используются для быстрого развертывания правильного окружения для обучения или эксплуатации модели, что важно для повторяемости результатов и исключения различных ошибок. При настройке взаимодействия с git и dvc репозиториями устанавливаются связи между этими репозиториями и вышеописанными директориями проекта.

Во многих случаях неупорядоченное хранение данных в виде xml, json, csv или других подобных форматов является недостаточным. Используя этот вид хранения вам самостоятельно придется решать вопрос резервирования, защиты от случайного удаления, хранения истории изменений в файлах. Гораздо эффективнее использовать специальный инструмент для работы с данными: базы данных, которые являются важным элементом любой программной системы, обеспечивающей надежное хранение и управление данными. В проектах машинного обучения базы данных могут быть использованы для хранения:

* служебных настроек,
* результатов работы модели,
* гиперпараметров модели,
* информации о пользователях.

Общую информацию о системах управления базами данных (СУБД) вы можете получить в Приложении 4, либо в соответствующей литературе, статьях и других информационных источниках.

Кроме эффективной организации хранения сущностей проекта также необходимо правильно организовывать различные элементы, используемые в проекте:

* аппаратное обеспечение
  + серверы
    - разработка (DEV)
    - тестирование (TEST, STAGE)
    - производственная эксплуатация (PROD)
    - хранилище данных (DATA\_STORAGE)
    - хранилище программного кода, моделей, метаинформации (GIT)
  + CPU/GPU
  + оперативная память
  + дисковое пространство
* общее программное обеспечение
  + рабочие операционные системы
  + управление пользователями, авторизация
  + системы резервирования
  + контейнеризация docker, kubernetes
  + низкоуровневое программное обеспечение, драйверы
  + системы виртуализации
  + системы мониторинга
* специализированное программное обеспечение
  + IDE (интегрированные среды разработки)
  + рабочие библиотеки для различных участников команды
  + инструменты для командной работы
    - jira
    - confluence
    - git

Состав этих элементов зависит от решаемых в проекте задач, масштаба проекта, объема данных для хранения, вычислительной нагрузки. Немаловажное значение имеет бюджет проекта, потому что разнесение отдельных функций на отдельные сервера связано с серьезными затратами. Поэтому для небольших проектов отдельные функции могут быть объединены на одном вычислительном ресурсе, с учетом решаемых задач. Обычно в проекте машинного обучения решаются следующие практические задачи:

1. развертывание рабочего окружения для исследований и обучения модели
   1. создание рабочего окружения для работы исследователей данных (Data Researcher) и моделей машинного обучения (ML Researcher) с использованием JupyterHub,
   2. создание виртуального окружения, установка необходимых библиотек, сохранение параметров виртуального окружения для повторяемости результатов,
   3. получение и сохранение данных, создание файловой структуры для хранения данных, создание dvc репозитория,
   4. создание различных моделей машинного обучения для решения задачи,
   5. анализ данных, формирование обучающего датасета с использованием наиболее важных признаков, выполнение необходимой предобработки признаков (например, заполнение пропущенных значений, преобразование текстовых признаков в числовые),
   6. выполнение обучения моделей, подбор гиперпараметров,
   7. сохранение результатов обучения моделей, метрик, весов,
   8. сохранение кода лучшей модели и данных виртуального окружения в git,
   9. сохранение использованного датасета в dvc хранилище.
2. улучшение качества работы модели
   1. загрузка данных п.1.h в новое окружение,
   2. изменение датасета, добавление новых признаков,
   3. выполнение обучения на новом датасете,
   4. сохранение результатов обучения моделей, метрик, весов,
   5. сохранение кода модели и данных виртуального окружения в git,
   6. сохранение использованного датасета в dvc хранилище.
3. вывод модели в prod
   1. выполнение скриптов для организации производственного окружения
   2. непрерывное развертывание всего решения
   3. загрузка модели
   4. эксплуатация модели на тестовых данных
   5. вывод модели из эксплуатации (остановка сервиса)

Давайте рассмотрим очень простой пример организации инфраструктуры проекта машинного обучения, который включает в себя наиболее важные элементы

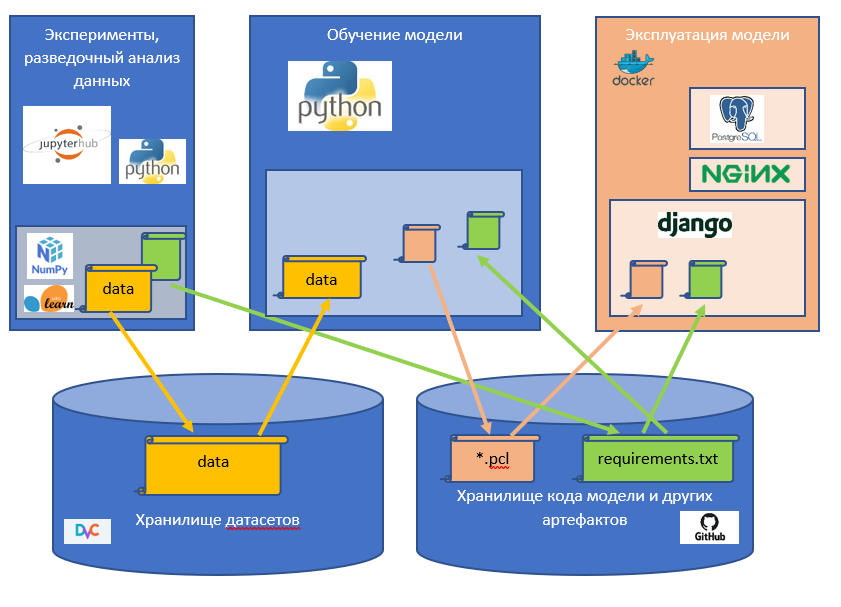


Рисунок “Пример инфраструктуры модели машинного обучения”.

Эта инфраструктура состоит из нескольких важных элементов:

* сервер для проведения разведочного анализа данных, экспериментов и проверки первичных гипотез,
* сервер для обучения модели, обычно имеет улучшенные аппаратные характеристики,
* сервер для эксплуатации модели,
* хранилище датасетов,
* хранилище кода моделей, параметров сред окружения, метаданных,

Для организации такой инфраструктуры можно использовать физически выделенные аппаратные серверы, либо пользоваться сервисами виртуальных серверов от провайдеров облачных сервисов. В юните 5 этого модуля рассмотрен практический пример создания такой инфраструктуры с использованием виртуальных машин и гипервизора VirtualBox, который вы изучили в Модуле 3 курса.

*Тестовое задание:*

1. Какие типовые сервера существуют в проекте разработки программного обеспечения? (0.25)
   1. **сервер разработки (DEV)**
   2. сервер подготовки дашбордов (SHOW)
   3. **сервер тестирования (TEST)**
   4. **сервер эксплуатации (PROD)**
2. Какое программное обеспечение может использоваться на сервере DEV? (0.25)
   1. **jupyter**
   2. **IDE**
   3. nginx
   4. **tensorflow**
3. каким образом можно организовать эффективную и безопасную работу ключевых элементов инфраструктуры проекта машинного обучения: серверов для разработки, тестирования и эксплуатации? (0.25)
   1. **использовать отдельные физические серверы**
   2. **использовать виртуальные машины на одном сервере под управлением гипервизора**
   3. **использовать виртуальные среды в инфраструктуре провайдера облачных услуг**
   4. запустить все на одном сервере
4. Чем отличаются серверы для обучения модели и для эксплуатации? (0.25)
   1. **для сервера обучения модели требуется больше GPU, чем для эксплуатации**
   2. для обучения модели можно использовать только сервера определенных марок, для эксплуатации подходят любые
   3. для сервера эксплуатации модели требуется больше GPU, чем для обучения
   4. обучение модели можно выполнять только в облаке

*Итоги:*

Организация правильной структуры проекта позволяет эффективно управлять множеством сущностей проекта

* программным кодом
* параметрами окружений
* версиями используемых библиотек
* моделями и конвейерами машинного обучения
* наборами данных
* метриками.

В этом юните вы узнали об одном из вариантов организации структуры проекта. Понимание особенностей формирования структуры проекта позволит вам создавать эффективную именно для вашего проекта структуру.

# 

# Модуль 7. Юнит 4. Системы управления конфигурациями.

*Введение:*

Итак, вы уже знаете, что в инфраструктуре проекта машинного обучения как правило используется несколько рабочих узлов - серверов для выполнения различных задач. Настройка таких серверов предполагает выполнение большого количества операций в определенной последовательности. Незначительное отклонение от правильной последовательности операций или ошибка в параметрах отдельной операции может привести к сбою, после которого всю процедуру установки надо начинать заново. Для решения этой проблемы можно было бы использовать заранее подготовленные образы виртуальных машин, которые мы изучали в Модуле 3. Например, один раз создав требуемую конфигурацию в виртуальной машине можно сохранить образ этой виртуальной машины и использовать его при создании новой виртуальной машины. Однако в этом случае пришлось бы даже при незначительном изменении параметров конфигурации каждый раз заново создавать и сохранять образ виртуальной машины целиком, что привело бы к непроизводительному расходованию пространства жесткого диска и не дало бы никакого выигрыша в скорости. Гораздо эффективнее описывать необходимую инфраструктуру скриптами, как например в системах контейнеризации docker или kubernetes. Однако системы контейнеризации вносят дополнительные накладные расходы, поскольку используют общие ресурсы для выполнения своих задач. Также docker или kubernetes могут не подойти для проекта потому, что для их использования необходима дополнительная квалификация участников. И, наконец, инструменты контейнеризации могут содержать ошибки или уязвимости в коде. Если по одной из вышеперечисленных причин вам не подходят виртуальные машины или контейнеры для решения задачи автоматизированного развертывания инфраструктуры, то вы можете применить специальный инструмент, который называется **система управления конфигурациями**.

Системы управления конфигурациями незаменимы для системных администраторов, инженеров DevOps или MLOps, поскольку позволяют автоматизировать рутинные операции по развертыванию сложных систем, или простых систем, которых очень много. Этот подход укладывается в идеологию Infrastructure-As-A-Code (IAAC), “инфраструктура-как-код”, которая лежит в основе DevOps.

Примеры систем управления конфигурациями: chef, ansible, terraform. Мы в данном юните подробно разберем инструмент ansible.

*Содержание юнита:*

Настройка оборудования и программного обеспечения под конкретную задачу связана с установкой необходимых значений параметров окружения, установкой программного обеспечения и его настройкой, созданием пользовательских ролей и пользователей. Часто это одинаковые повторяющиеся операции. Например, настройка пользовательского компьютера связана с установкой одних и тех же программ и настройкой одинаковых параметров, прописыванием одинаковых пользовательских прав и настроек доступа. Если таких пользователей сотни, то придется вручную повторять одни и те же действия по настройке каждого пользовательского компьютера сотни раз, если нет систем автоматизации. Есть и другой сценарий, когда объектов для настройки не очень много, но их конфигурирование само по себе сложный процесс, требующий высокой квалификации и точности в выполнении действий. Например, это уникальный производственный сервер со специальным аппаратным обеспечением, шагов по его настройке много, их последовательность важна, любая ошибка в выполнении отдельного шага настройки может привести к проблемам в запуске и эксплуатации системы. Чтобы решать описанные выше проблемы автоматизации и унификации используют **системы управления конфигурациями**. Системы управления конфигурациями позволяют производить настройку оборудования по заранее подготовленным сценариям-скриптам. Примеры таких систем:

|  |  |  |
| --- | --- | --- |
| Chef | https://www.chef.io | Система управления конфигурациями, написанная на Ruby (клиентская часть) и Erlang (серверная часть). Для описания необходимый действий используются «рецепты», описывающие управление приложениями, включая версии пакетов, службы, создаваемые файлы. |
| Puppet | https://puppet.com | Кроссплатформенное клиент-серверное приложение, которое позволяет централизованно управлять конфигурацией операционных систем и программ, установленных на нескольких компьютерах. |
| Ansible | https://www.ansible.com | Система управления конфигурациями, написанная на языке программирования Python, с использованием yaml разметки для описания конфигураций. Используется для автоматизации настройки и развертывания программного обеспечения. |
| Terraform | https://www.terraform.io | Программное обеспечение с открытым исходным кодом. Позволяет описывать инфраструктуру центра обработки данных с помощью декларативного языка конфигурации, известного как HashiCorp Configuration Language или JSON. |

Выбор конкретного инструмента зависит от проекта и навыков команды проекта. Подробно про каждый инструмент можно узнать на приведенных выше официальных сайтах, либо из соответствующих книг или статей. Мы же более подробно рассмотрим Ansible.

Ansible это простой инструмент, не требует специальных настроек и установки клиентских приложений, максимально использует элементы стандартной сетевой инфраструктуры и типовое программное обеспечение, например:

* python3 для выполнения скриптов,
* существующую инфраструктуру ssh для взаимодействия узлов,
* существующие пользовательские роли и пользователей для выполнения операций на узлах,
* файлы формата yaml для хранения настроечных данных.

При использовании ansible системные администраторы или инженеры DevOps/MLOps с одного или нескольких управляющий серверов подключаются через сеть передачи данных по защищенному протоколу ssh к удаленным скриптам и выполняют действия по настройке удаленных серверов на основе команд, описанных в плейбуках.

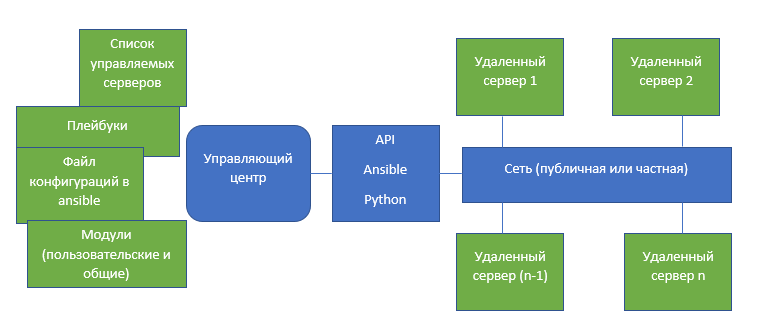


Рисунок “Схема работы Ansible”

Основные задачи, которые решает ansible:

* управление конфигурациями серверов, быстрая и точная настройка,
* управление процессом развертывания новых серверов, установка и обновление приложений,
* оркестрация, координация компонентов всей инфраструктуры при выполнении сложного развертывания с множеством зависимостей между отдельными компонентами и узлами,
* мониторинг и логирование информации.

Для настройки и понимания принципов работы ansible вам надо уметь работать с ssh. ssh (secure shell) это сетевой протокол прикладного уровня, который позволяет в защищенном режиме взаимодействовать с удаленным сервером, например, управлять операционной системой. В работе ssh использует два компонента: ssh-сервер и ssh-клиент. Работающий ssh-сервер ожидает входящие запросы на установление соединения на порте 22 (по умолчанию), а также обеспечивает выполнение требования аутентификации сторон. Есть несколько вариантов проверки аутентификации при соединении:

* Пароль. Используется чаще всего. При таком типе аутентификации между клиентом и сервером создается общий секретный ключ: он шифрует трафик.
* С помощью пары ключей, открытого публичного и закрытого приватного. Предварительно генерируется открытый ключ (private key) и закрытый ключ (public key). На устройстве, с которого нужно подключиться, хранится закрытый ключ, а на сервере, к которому выполняется удаленное подключение, хранится открытый ключ. При подключении файлы ключей не передаются, система только проверяет, что устройство имеет доступ не только к открытому, но и к закрытому ключу,
* IP адрес. При подключении система идентифицирует устройство по IP адресу. Такой тип аутентификации небезопасен и используется редко.

Более подробно вы можете прочитать про ssh в специальных книгах и статьях, некоторые приведены в списке источников к этому модулю.

Лучше всего изучать инструменты на практике, поэтому давайте совместно проделаем шаги для запуска и использования ansible:

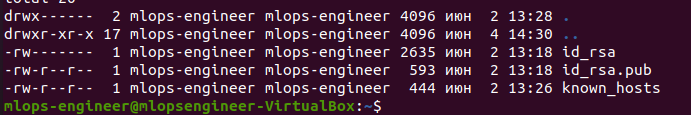
1. Запустить серверы, которые мы хотим администрировать. В примерах ниже у нас два сервера: EDA\_server (sudo пользователь data-engineer) и ML\_server (sudo пользователь ml-engineer). Также отдельно запустим сервер admin\_server, на котором работает mlops инженер (пользователь mlops-engineer), выполняющий роль системного администратора или MLOps инженера. На серверах должен быть установлен python3. Также на серверах необходимо установить и настроить безопасный протокол доступа ssh, это описано в следующих шагах.

2. Настроить протокол ssh для взаимодействия. Для этого на сервере mlops инженера, с которого в дальнейшем будет осуществляться вход на удаленные сервера для администрирования, понадобится создать открытый и закрытый ключи, это делается с помощью команды linux

**ssh-keygen**

или с использованием исполняемого файла puttygen.exe в Windows.

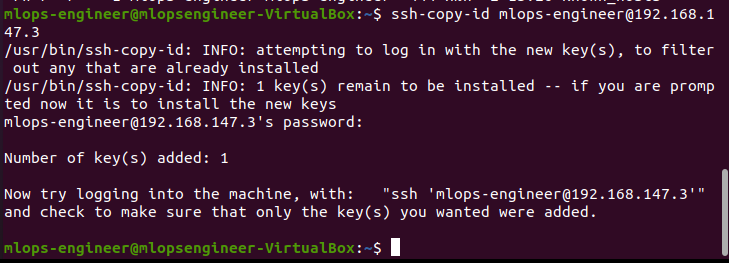
При этом создаются файлы закрытого и публичного ключа в директории .ssh



Если вы знаете пароль пользователя, то процесс можно упростить. Теперь надо передать на удаленный сервер, который вы хотите администрировать удаленно, файл открытого ключа id\_rsa.pub. Это можно сделать вручную, создав новый файл и скопировав в него содержимое файла id\_rsa.pub, или перенеся файл с помощью ftp. Кроме этого, есть удобная возможность, команда

**ssh-copy-id user@server**

позволяет скопировать ключ на нужный вам сервер, не редактируя файлы вручную.



3. Установить ansible на сервере mlops инженера.

**sudo apt update**

**sudo apt install ansible**

4. Отредактировать файл инвентаризации /etc/ansible/hosts. В этом файле необходимо прописать правильно сервера, которые будем удаленно администрировать, а также прописать интерпретатор python. Проверить файл инвентаризации можно командами

**ansible-inventory --list -y**

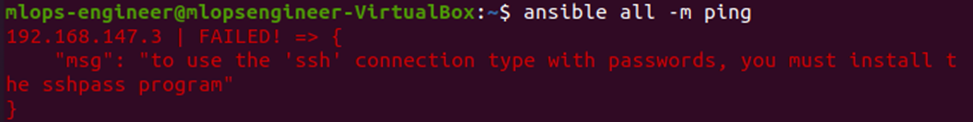
или

**ansible all --list-hosts**

5. Проверка связности и доступности серверов осуществляется с использованием команды

**ansible -m ping all -u “имя пользователя”**

При выполнении этой команды может случиться вот такая ошибка:



В этом случае надо установить утилиту sshpass на сервере, на котором работает mlops инженер и откуда он осуществляет подключения на удаленные сервера для администрирования

**sudo apt-get install sshpass**

В качестве имени пользователя надо использовать то имя, которое есть на удаленных серверах, к которым хочет подключаться mlops инженер. Из соображений безопасности не стоит использовать пользователя root, хотя он и присутствует на всех серверах. Вместо этого на всех серверах, которые мы хотим администрировать удаленно с помощью ansible, надо создать пользователя, например mlops-engineer, и наделить его правами sudo.

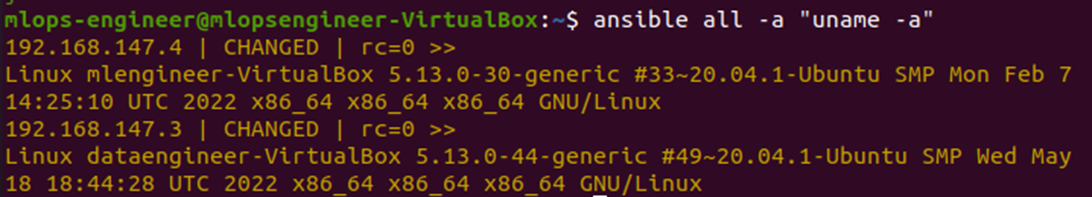
В случае проблем можно отлаживать ситуацию с использованием опции -vvv

**ansible -m ping all -vvv**

6. Выполнение команд на удаленных серверах осуществляется с помощью команды

**ansible all -a "команда"**

Вот результат выполнения команды “uname -a”, которая дает развернутую информацию о системе.



Для организации работы ansible использует следующие сущности:

* файл конфигурации ansible,
* переменные, групповые (общие) или частные (отдельных хостов),
* плейбуки (playbook), сценарии, по которым работает Ansible, пишутся в формате yaml,
* роли, это структурированные плейбуки,
* списки групп хостов, удаленных серверов, которые управляются с помощью ansible.

Файл конфигурации ansible описывается в формате INI, имеет расширение cfg. Можно переопределить часть или всю конфигурацию в параметрах playbook или переменных окружения. При исполнении команд Ansible проверяет наличие файла конфигурации в следующих местах:

* в переменной окружения ANSIBLE\_CONFIG,
* файл в текущей директории: ./ansible.cfg,
* файл в домашней директории: ~/.ansible.cfg,
* файл в директории, созданной при установке ansible: ./etc/ansible/ansible.cfg

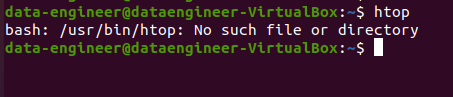
Все действия, которые мы хотим выполнить при удаленном администрировании серверов, необходимо записывать в плейбуки. В плейбуке указываются:

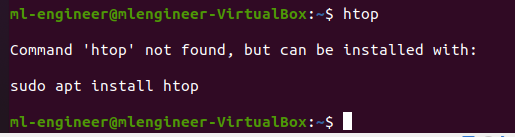
* целевая группа хостов, к которым применяются действия (hosts),
* действия (tasks) или роли (roles),
* пользователь для ssh-connect (remote\_user),
* пользователь sudo (become\_user),
* необходимость использования повышенных привилегий, прав суперпользователя (become: True/False),
* переменные (vars),
* файлы переменных (vars\_files),
* количество одновременных коннектов (serial)

Вот пример ansible скрипта, который устанавливает утилиту htop на удаленные сервера:

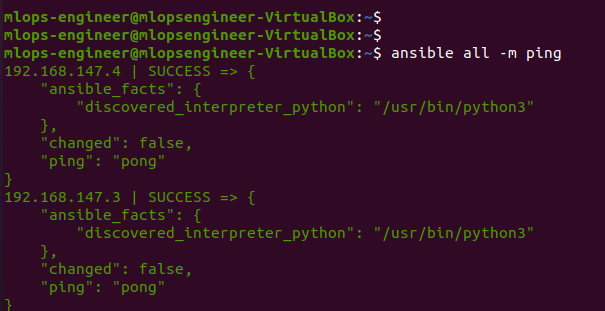


Утилита htop используется для расширенного мониторинга ресурсов системы. На удаленных серверах этой утилиты нет:

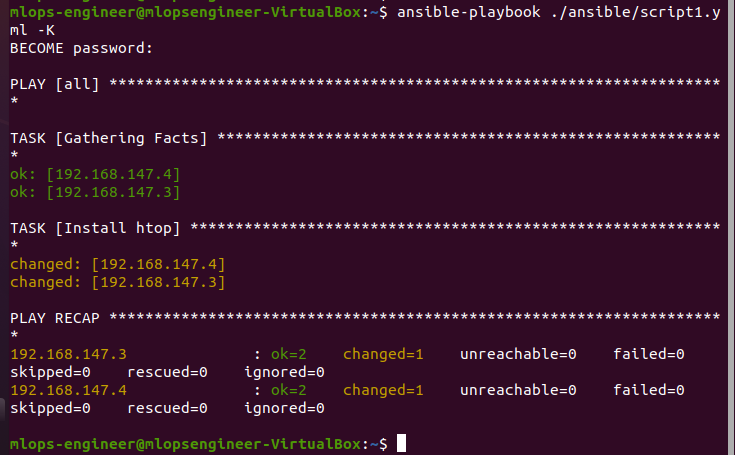




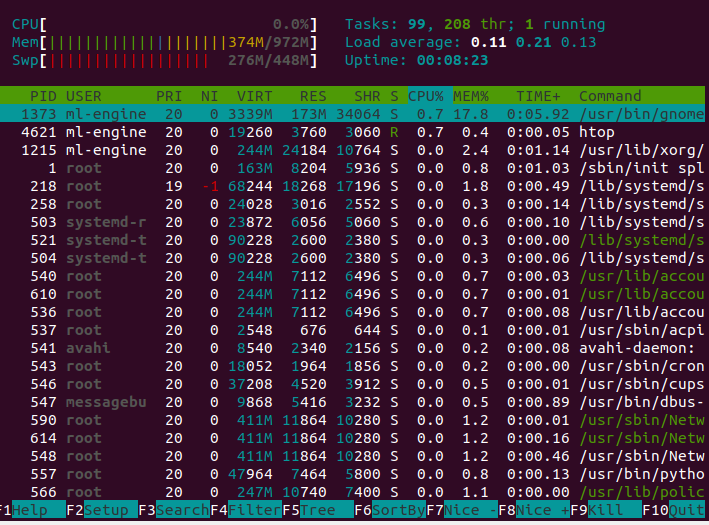
Проверим доступность серверов на которых хотим установить htop



Запускаем скрипт ansible



В результате мы установили утилиту htop на удаленные серверы, теперь ей можно пользоваться

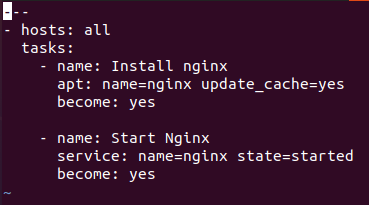


С помощью вышеописанного подхода можно устанавливать необходимое программное обеспечение на удаленные серверы в автоматическом режиме, с использованием заранее подготовленного скрипта. Вы можете определять нужные версии программного обеспечения и другие параметры, устанавливать переменные среды окружения.

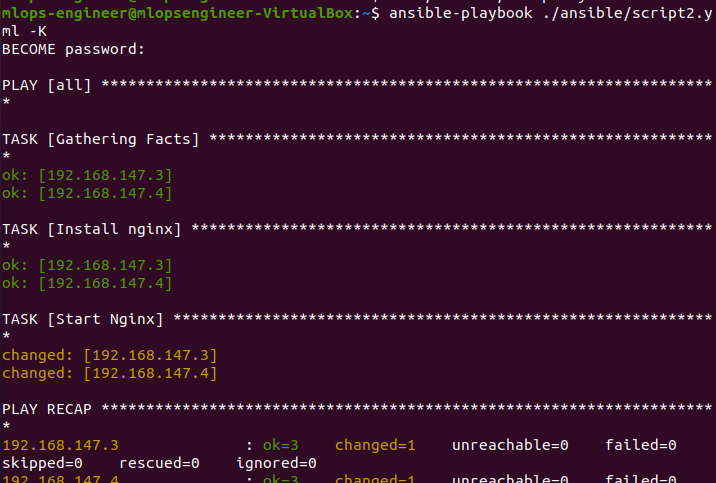
Вот важные сведения о написании плейбуков Ansible

* все YAML файлы должны начинаться с "---". Это часть формата YAML и означает начало документа,
* все члены списка должны находится с одинаковым отступом от начала строки, и должны начинаться с пробела или "-",
* комментарии начинаются с "#",
* словарь представлен в виде «ключ:» (двоеточие и пробел) «значение»,
* для переменных Ansible использует "{{ var }}" из формата jinja.

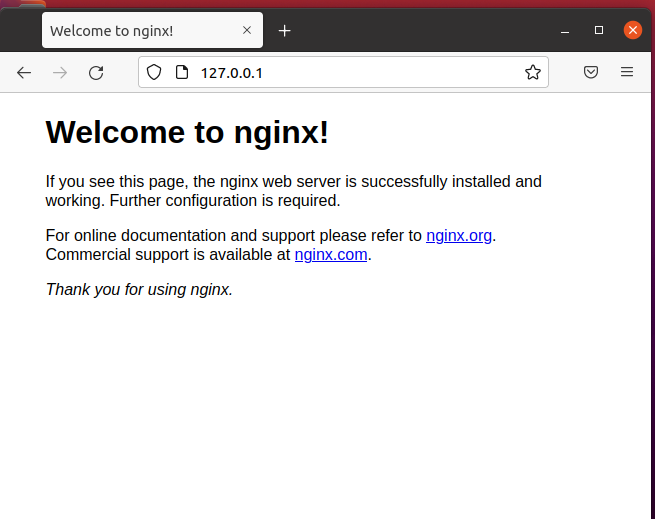
Пример плейбука, который устанавливает и запускает на удаленном сервере web-сервер nginx



После выполнения этого скрипта на управляющем сервере вы увидите следующий результат:



И на удаленном сервере запущен web-сервер nginx



И в заключении этого юнита, в котором вы знакомитесь с ansible, давайте изучим полезный инструмент, модули ansible. Модули это набор команд, которые вы можете использовать в скриптах-плейбуках для решения определенных задач. Например, наиболее часто используемые модули:

|  |  |  |
| --- | --- | --- |
| Имя модуля | Описание | Пример скрипта |
| command | Принимает команду и аргументы и обеспечивает выполнение этой команды на удаленном сервере. | Перезагрузка серверов |
| copy | Модуль для копирования файлов |  |
| raw | Используется тогда, когда другие модули использовать не получается. Обеспечивает запуск команд на удаленных управляемых серверуах по ssh. Может работать без python3. | Записать данные в файл |

Также в состав ansibl входят и другие полезные модули, например git, template. Используя иже известные вам команды можно поставить скрипты для любых задач, требующих выполнения на удаленных серверах. Эти знания понадобятся вам в дальнейшем при выполнении практических заданий данного модуля.

*Тест:*

1. Какие программы используются для работы ansible? (0.25)
   1. java
   2. **python3**
   3. **ssh**
   4. sql
2. Как называется файл, содержащий перечень команд, используемый ansible для настройки удаленных серверов? (0.25)
   1. инструкция
   2. **плейбук**
   3. commands
   4. actions
3. Какая команда опрашивает все удаленные сервера? (0.25)
   1. ansible echo all
   2. ansible get reply
   3. **ansible all -m ping**
   4. ansible all -m ask servers
4. Какой формат используеься для ansible плейбука? (0.25)
   1. **yaml**
   2. txt
   3. py
   4. c

*Выводы:*

Процедуры настройки серверов, установки и настройки программного обеспечения требуют глубоких знаний, внимательности, точности в соблюдении последовательности действий, поэтому занимают много времени. Системы управления конфигурациями позволяют автоматизировать процесс настройки, что сокращает время настройки, уменьшает количество ошибок. В этом юните вы познакомились с одной из таких систем ansible.

# Модуль 7. Юнит 5. Пример создания инфраструктуры для проекта машинного обучения.

*Введение:*

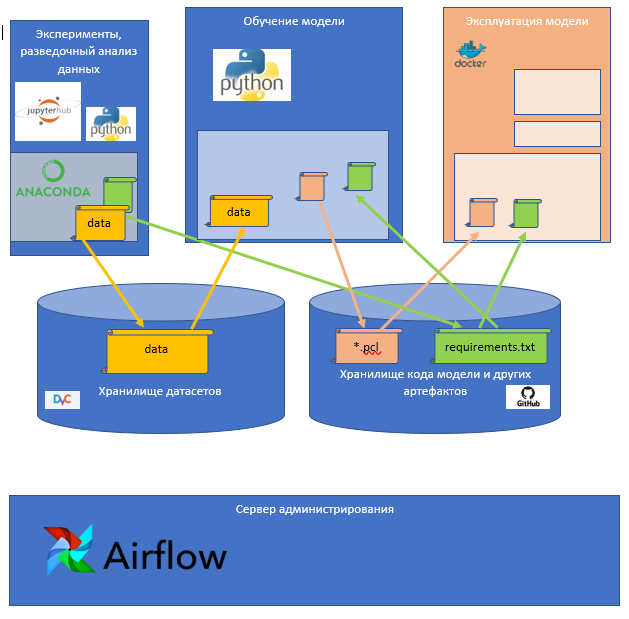
В этом юните мы обобщим полученные в предыдущих юнитах этого модуля, а также некоторых других предыдущих модулях, знания о том, каким образом можно создать эффективную инфраструктуру для обучения и вывода модели машинного обучения в производственную среду (production). В практической части юнита вы будете применять эти знания для создания инфраструктуры учебного проекта с использованием автоматизации ansible.

*Содержание:*

Из курса “Программная инженерия” или аналогичных вы уже наверняка знаете, что современный многофункциональный комплекс включает в себя не только модель машинного обучения, но и важные компоненты для организации бекэнда, фронтенда, базы данных и других функций промышленных программных систем. Эти элементы являются типовыми в архитектуре программного обеспечения, если вы недостаточно знакомы с этими понятиями и инструментами для их реализации, вы можете кратко ознакомиться с ними в Приложениях 1-4 данного модуля. Создавать каждый раз такую инфраструктуру с “нуля” крайне неэффективно и долго, поэтому в DevOps и MLOps применяются различные инструменты для автоматизации, например, виртуальные машины, контейнеры или системы управления конфигурациями.

Давайте рассмотрим конкретную практическую, хотя и учебную, задачу. Пусть нам требуется организовать инфраструктуру проекта машинного обучения для создания продукта, классифицирующего пассажиров “Титаника”, уже известная нам по kaggle.com задача “Titanic Disaster”. Эта задача потребует создания сервера для анализа данных и проверки гипотез, сервера для обучения модели и “производственного” сервера, на котором модель будет эксплуатироваться. Конечно, модели из соревнований kaggle.com редко попадают в промышленное окружение, но мы поставим перед собой такую задачу в учебных целях. В итоге нам нужна инфраструктура, которая содержит следующие серверы:

|  |  |  |
| --- | --- | --- |
| Название | Назначение | Программное обеспечение |
| EDA\_server | для организации работы исследователей данных и моделей машинного обучения, проведения разведочного анализа данных (EDA, Exploratory Data Analyze), проверки первичных гипотез, экспериментов | jupyterhub, python3, conda, dvc  в виртуальной среде conda устанавливливаются нужные версии python библиотек для исследований, файл requirements.txt сохраняется в git репозиторий |
| ML\_server | для обучения моделей, обычно содержит GPU, так как выполнение задач связано с большой вычислительной нагрузкой | python3, conda  В виртуальной среде conda устанавливливаются нужные версии python библиотек для исследований, файл requirements.txt сохраняется в git репозиторий |
| PROD\_server | для эксплуатации моделей | python3, docker  В docker контейнерах поднимаются необходимые сервисы: web-сервер, фреймворк для бэкенда, база данных, система мониторинга и пр. |
| DATA\_server | для хранения данных | python3, ssh |
| admin\_server | для работы MLOps инженера | ssh, ansible, Airflow  Также на этот сервер может быть установлено и другое программное обеспечение, например, Jenkins, MLFlow. |



Этапы создания инфраструктуры проекта:

|  |  |  |
| --- | --- | --- |
| 1. Настройка сервера администратора (инженера MLOps), admin\_server | | |
| Создание необходимых пользователей на всех серверах | admin\_server, EDA\_server, ML\_server, DATA\_server, PROD\_server | На каждом сервере должен быть прописан пользователь, являющийся администратором MLOps, обладающий правами sudo (см. Юнит 4 текущего Модуля). |
| установка и настройка ssh | ssh | создать приватный и публичный ключи, скопировать публичные ключи на удаленные сервера (см. Юнит 4 текущего модуля) |
| установка и настройка ansible | ansible | установить, настроить доступ до удаленных серверов по ssh, написать плейбуки (см. Юнит 4 текущего модуля) |
| установка и настройка AirFlow | Airflow | Установить, настроить, прописать пользователей. (см. Юнит 2 Модуля 5). |
| 1. Развертывание рабочего окружения для исследований данных и проверки гипотез, EDA\_server | | |
| создание рабочего окружения для работы исследователей данных (Data Researcher) и моделей машинного обучения (ML Researcher) | JupyterHub | Установить с помощью скрипта ansible, добавить пользователей.  (создать скрипты установки jupyterhub с использованием сведений из курса “Программная инженерия”, Приложения 5 текущего модуля и Юнита 4 текущего модуля). |
| создание виртуального окружения, установка необходимых библиотек, сохранение параметров виртуального окружения для повторяемости результатов | conda | Установить с помощью скрипта ansible.  (создать скрипты установки conda с использованием сведений из курса “Программная инженерия” и Юнита 4 текущего модуля). |
| установка инструментов для версионирования | git, dvc | Установить с помощью скрипта ansible.  (создать скрипты установки dvc с использованием сведений из курса “Программная инженерия”, Юнита 3 Модуля 4 и Юнита 4 текущего модуля). |
| 1. Развертывание рабочего окружения для обучения моделей, ML\_server | | |
| создание виртуального окружения, установка необходимых библиотек, сохранение параметров виртуального окружения для повторяемости результатов | conda | Установить с помощью скрипта ansible.  (создать скрипты установки conda с использованием сведений из курса “Программная инженерия” и Юнита 4 текущего модуля). |
| установка инструментов для версионирования | git, dvc | Установить с помощью скрипта ansible.  (создать скрипты установки dvc с использованием сведений из курса “Программная инженерия”, Юнита 3 Модуля 4 и Юнита 4 текущего модуля). |
| Установка необходимых библиотек | Загрузка в виртуальном окружении conda |  |
| 1. Создание сервера для производственной эксплуатации модели, PROD\_server | | |
| Установка docker | docker | Установить с помощью скрипта ansible.  (создать скрипты установки docker с использованием сведений из курса “Программная инженерия”, Юнита 4 Модуля 3 и Юнита 4 текущего модуля). |
| 1. Создание хранилища данных, DATA\_server | | |
| Установка ssh | ssh | Установить взаимодействие с dvc с использованием сведений из Модуля 4. |

Теперь в созданной инфраструктуре команда может выполнять типовые действия проекта машинного обучения.

1. Анализ данных, проверка гипотез и создание предварительных моделей машинного обучения на сервере EDA\_server
   1. получение и сохранение данных о пассажирах “Титаника” из задачи “Titanic Disaster”, создание файловой структуры для хранения данных, создание dvc репозитория (см. Модуль 4),
   2. создание нескольких моделей классификации (линейная регрессия, деревья решений, случайный лес, XGBoost),
   3. формирование обучающего датасета с использованием признаков: “Age” (возраст), “Sex” (пол), “Parch” (количество близких родственников), “SibSp” (количество дальних родственников), выполнение необходимой предобработки признаков (например, заполнение пропущенных значений, преобразование текстовых признаков в числовые),
   4. выполнение обучения моделей с подбором гиперпараметров, названия и сетка значений гиперпараметров берется из json файла,
   5. сохранение результатов обучения моделей, метрик, весов,
   6. сохранение кода лучшей модели и данных виртуального окружения в github,
   7. сохранение использованного датасета в dvc хранилище (см. Модуль 4).
2. улучшение качества работы модели, обучение на сервере ML\_server
   1. загрузка данных п.1.f в новое окружение на сервере ML\_server,
   2. изменение датасета, добавление новых признаков (например, добавить новый признак “Title” (титул), который берется из Name),
   3. выполнение обучения на новом датасете,
   4. сохранение результатов обучения моделей, метрик, весов,
   5. сохранение кода модели и данных виртуального окружения в github,
   6. сохранение использованного датасета в dvc хранилище (см. Модуль 4).
3. вывод модели в промышленную эксплуатацию на сервер PROD\_server
   1. выполнение скриптов для организации производственного окружения
   2. непрерывное развертывание всего решения
   3. загрузка модели
   4. эксплуатация модели на тестовых данных
   5. вывод модели из эксплуатации (остановка сервиса)

*Практическое задание:*

В практическом задании данного юнита, подытоживающем полученные в модуле знания, вам будет необходимо создать инфраструктуру проекта машинного обучения и настроить ее с использованием скриптов ansible.

Этап 1: создание серверов.

В целях упрошения мы не будем создавать всю целевую инфраструктуру, описанную в модуле и включающую в себя сервера для разработки, тестирования, хранения данных, продакшн. В этом практическом задании нам понадобится три сервера:

* сервер управления (MLOps), примечание: в качестве этого сервера вы можете использовать свой рабочий компьютер,
* сервер разработки (DEV),
* сервер промышленной эксплуатации (PROD).

Для организации серверов вы можете использовать виртуальные машины одного из провайдеров облачных сервисов, либо систему виртуализации VirtualBox или аналогичные для создания виртуальных машин на вашем рабочем компьютере. Требования к серверам можно взять самые минимальные, так как мы не предполагаем выполнение практических задач на них, а только настройку.

На серверах должен быть установлен python3 и ssh. Закрытый файл ключа ssh необходимо разместить на управляющем сервере MLOps.

Этап 2:

Необходимо подготовить скрипты ansible для настройки серверов. На серверах необходимо настроить и запустить следующее программное обеспечение

|  |  |
| --- | --- |
| Название | Программное обеспечение |
| DEV | Предустановлено: python3, ssh  Установить: jupyterhub, conda, dvc, git |
| PROD | Предустановлено: python3, ssh  Установить: docker |

Установку “вручную” этого программного обеспечения вы изучали в предыдущих модулях этого курса, либо в других курсах, например “Программная инженерия”. В этом задании вам необходимо этот процесс автоматизировать с помощью плейбуков ansible.

Результат выполнения задания:

1. скриншот успешного выполнения команды ansible all -m ping
2. два ansible плейбука для настройки серверов DEV и PROD.

Итоги/выводы

В этом юните мы разобрали простой учебный пример решения задач MLOps для конкретного проекта. Мы определили состав серверов и необходимое программное обеспечение, создали ansible скрипты для автоматизации настройки этих серверов. Установили взаимодействие между этими серверами. Описанный метод решения учебной задачи не является универсальным методом на все случаи жизни, а призван продемонстрировать основные концепции, задачи и методы их решения. Применяя полученную информацию к вашим проектам вы сможете организовать собственную инфраструктуру с настройками, эффективными для вашего проекта.

# Итоги/выводы по модулю

В этом модуле были рассмотрены вопросы, связанные с созданием инфраструктуры проекта машинного обучения для вывода в производственную среду (production). Это важнейшая задача любого проекта разработки программного обеспечения, так как если проект не приносит пользу в реальной эксплуатации, то работа разработчиков была выполнена впустую. В проектах машинного обучения задачи эксплуатации тесно связаны с задачами разработки и тестирования. Весь конвейер машинного обучения проекта, начиная от анализа данных и заканчивая эксплуатацией модели, должен работать как единое целое. В команде проекта машинного обучения существует множество ролей, для каждой роли свои инструменты. Однако, несмотря на кажущуюся обособленность специалиста по анализу данных от инженера эксплуатации, эта обособленности мнимая. Данные, полученные при эксплуатации, важны для инженера данных, поскольку позволяют лучше понять закономерности в данных и улучшить рекомендации по обучению модели. Поэтому важнейшая роль MLOps инженера заключается в создании эффективной инфраструктуры для всех участников проекта, направленной на создание полезного продукта и выводу его в промышленную эксплуатацию. В данном модуле описаны этапы решения этой задачи и инструменты для их выполнения.

# Практическое задание по модулю.

В итоговом задании по модулю вам предлагается создать полноценную программную систему, включающую модель машинного обучения, и с использованием изученных вами средств автоматизации вывести эту модель в производственное окружение. То есть необходимо разработать проект, в котором обученная модель машинного обучения принимает через API данные и отдает на их основе прогноз. Выбор задачи, датасета и модели остается на ваше усмотрение, вы можете использовать задачу “Титаник”, либо любую задачу из предыдущих модулей или курсов.

Задача может быть выплонена двумя способами:

Способ 1: использовать проект, подготовленный вами в курсе “Программная инженерия”, в котором вы применяли flask для создания микросервиса модели машинного обучения.

Способ 2: создать проект с использованием фреймворка django на основе информации из Приложений 1-4. Понадобится запустить проект django с использованием nginx в качестве прокси-сервера. Настроить в nginx работу со статическими файлами.

Весь процесс должен быть полностью автоматизирован в ansible скрипте:

* установка необходимого программного обеспечения на сервер
* копирование файлов проекта на сервер (например из git)
* сборка и запуск проекта

Результатом является ansible плейбук, выполнение которого приводит к запуску сервиса, отвечающего на API запросы.

# Список источников

1. web серверы

<https://nginx.org/ru/docs/beginners_guide.html>

<https://gunicorn.org/>

https://httpd.apache.org/

1. ansible

https://www.ansible.com/

1. django

https://www.djangoproject.com/

1. SSH

<https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-20-04-ru>

<https://www.digitalocean.com/community/conceptual_articles/an-introduction-to-configuration-management-with-ansible>

# Приложение 1. Бекэнд программных систем на примере фреймворка Django.

*Введение:* В этом приложении вы узнаете о фреймворке Django, который очень популярен для реализации программных проектов, особенно web-сервисов. Django позволяет эффективно решить основные задачи бэкенд уровня, предоставляет удобные средства для быстрой разработки и мониторинга решения. Альтернативами Django являются flask, fastapi и другие подобные сервисы. Поняв основные задачи и приемы использования фреймворка Django вы сможете самостоятельно разобраться с его аналогами.

*Содержание:*

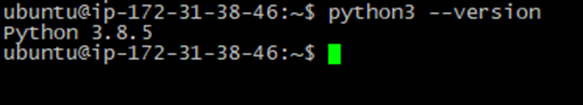
Django это фреймворк для разработки программных систем, написанный на python и использующий python. Django берет на себя решение основных задач, возникающих перед разработчиком программных систем, например

* организация одновременной работы и взаимодействия многих компонентов,
* высокоуровневое описание логики работы web интерфейса, правил обработки url, привязка html информации,
* взаимодействие с базами данных,
* авторизация пользователей, функции безопасности приложения,
* мониторинг и контроль логики работы приложения через панель администратора,
* обработка временных зон,
* локализация, использование различных языков в приложении,
* и многое другое.

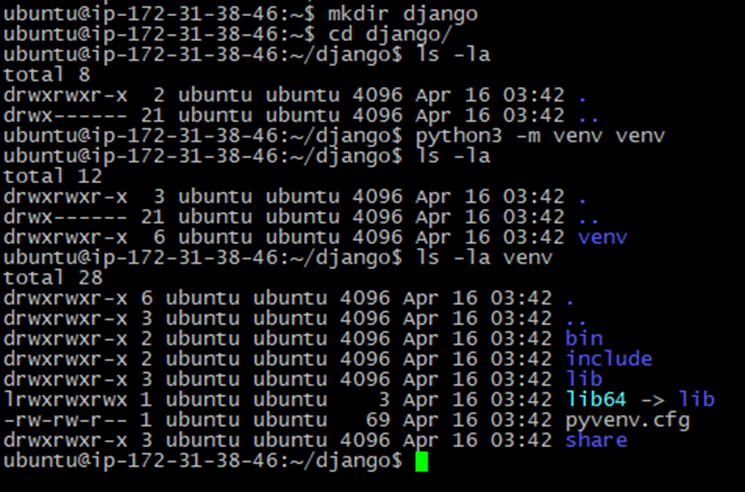
Мощь Django заключается в том, что разработчик получает уже готовые удобные инструменты для решения этих задач, а не решает их “с нуля”. Поэтому Django очень популярен для проектов, в которых надо быстро развернуть “минимально жизнеспособное решение” (MVP, Minimal Viable Product).

Установить Django очень просто. Сначала необходимо проверить установлен ли python (обычно это версия 2) или python3 (обычно это версия 3). Рекомендуемая для работы версия python3. Как вы уже знаете из Модуля 3, хорошей практикой является создание изолированного окружения c использованием virtualenv, venv или аналогичных инструментов. При этом создается изолированная виртуальная среда

**python3 –m venv venv**

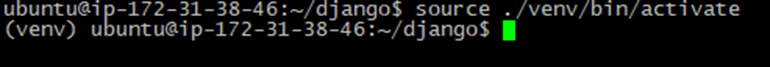
****

Любая активированная библиотека в изолированном окружении будет устанавливаться в папку «имя окружения»/lib/site-packages.



Активировать виртуальное окружение можно с использованием команды

**source ./”имя папки окружения”/bin/activate**

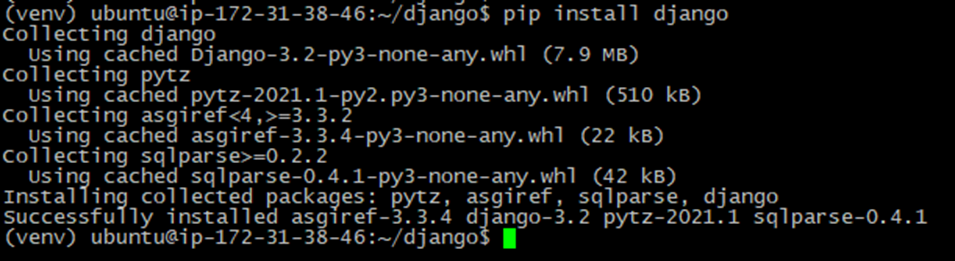
****

При этом меняется промптер. Деактивация производится скриптом **deactivate**.



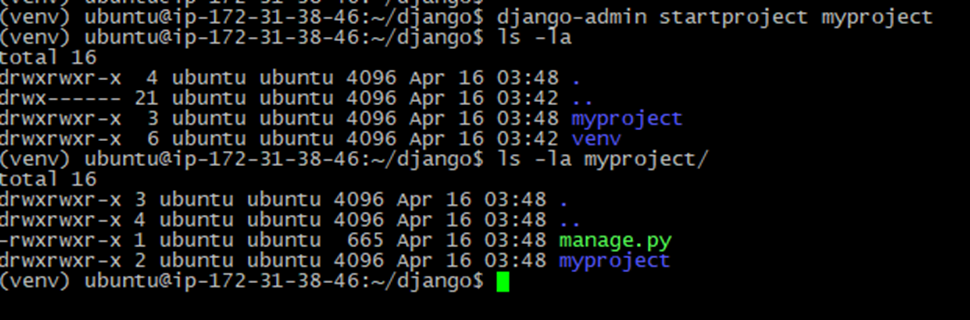
После этого можно устанавливать Django и другие библиотеки внутри виртуального окружения, без опасения, что устанавливаемые вами библиотеки вступят в конфликт с другим программным обеспечением.

**python3 -m pip install django**

****

Итак, вы настроили виртуальное окружение и установили в него программного пакета Django. Давайте теперь создадим программную систему с использованием Django, последовательно выполняя главные этапы создания таких систем.

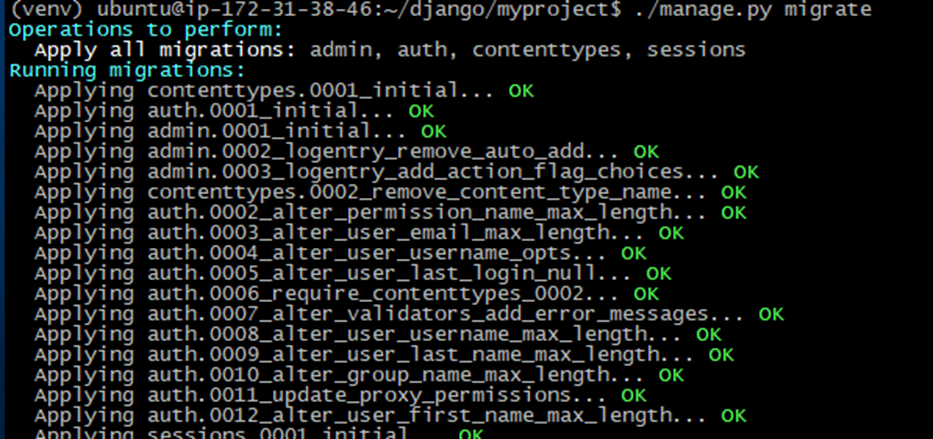
Этап 1. Создание проекта. Для работы с Django необходимо создать проект.



Создана папка проекта и скрипт manage.py с помощью которого будут осуществляться дальнейшие настройки и управление проектом.

Этап 2. Создание служебной базы данных.

В django используется база данных для хранения служебных системных настроек, даже если пока нет еще проекта, таблицы django уже необходимы.



Сначала создаются служебные таблицы, относящиеся к работе Django приложения. Далее в базу данных можно добавлять пользовательские таблицы. В версию python3 встроена поддержка SQLite – легкая база данных, удобная для небольших проектов. Django поддерживает и другие базы данных – PostgreSQL, MySQL, MariaDB, Oracle – база данных выбирается в зависимости от потребностей разрабатываемой системы.

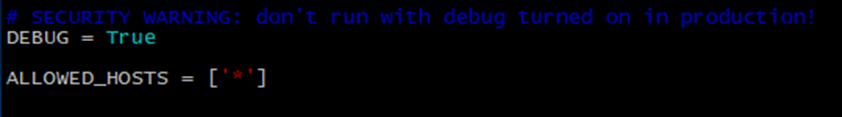
После миграции в рабочем директории появился файл базы данных sqlite.

Этап 3. Запуск проекта

Выполненных действий достаточно, чтобы запустить веб сервер с настройками «из коробки» с помощью команды

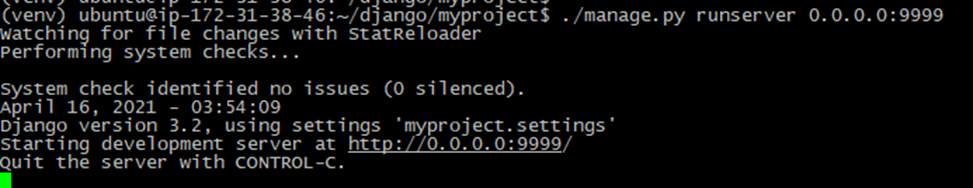
**./manage.py runserver**

Однако если пользовательское подключение к серверу планируется с другого хоста, то необходимо прописать правила доступа в файле **“имя папки проекта”/settings.py**.

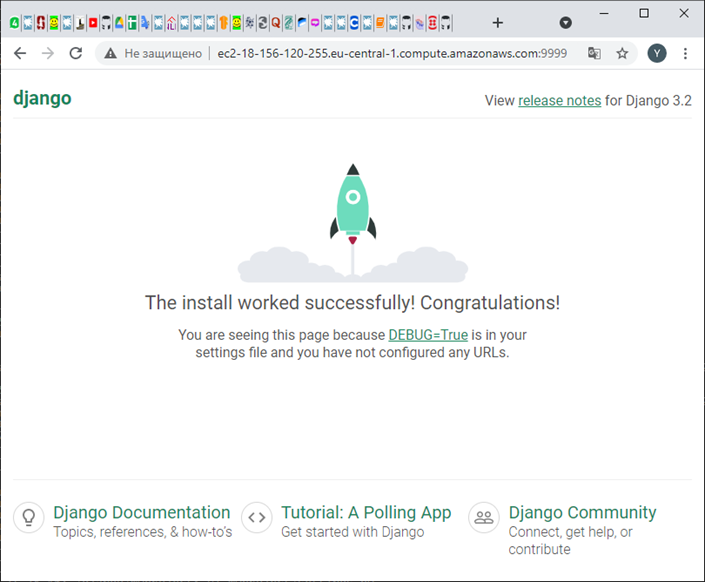


Для целей отладки для переменной DEBUG установлено значение True, а для ALLOWED\_HOSTS значение [‘\*’]. При выводе сервиса в «боевое» использование (production) рекомендуется отключать режим отладки для безопасности проекта.

Django имеет встроенный web сервер, который можно использовать для разработки, однако для продуктивной среды рекомендуется использовать стандартные веб сервера (например, gunicorn, apache, nginx).

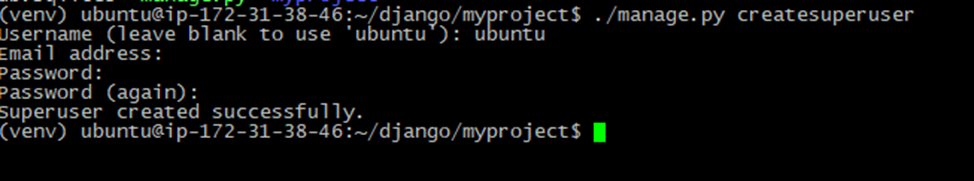


Теперь можно зайти через браузер на порт приложения и появится страница Django.

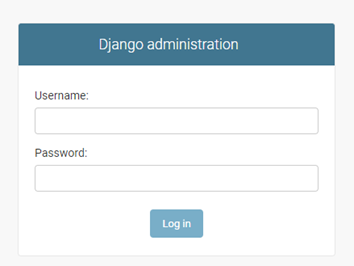


Этап 4. Создание суперпользователя

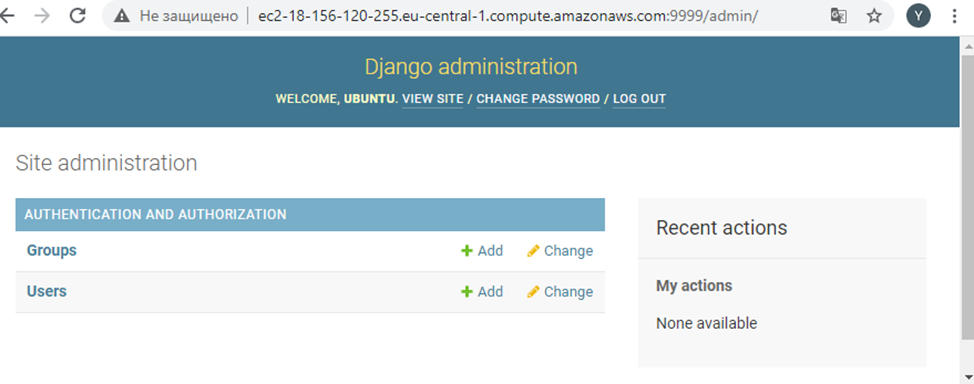
В Django «из коробки» доступны многие опции, например, панель администратора, через которую можно управлять многими настройками приложения. Для доступа необходимо создать суперпользователя (логин и пароль).



После этого admin панель доступна по адресу **«hostname (или IP адрес)»/admin**

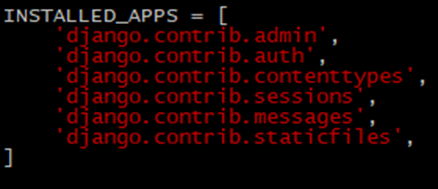
****

Надо ввести логин и пароль, заданные на предыдущем шаге, после этого откроется панель администратора.



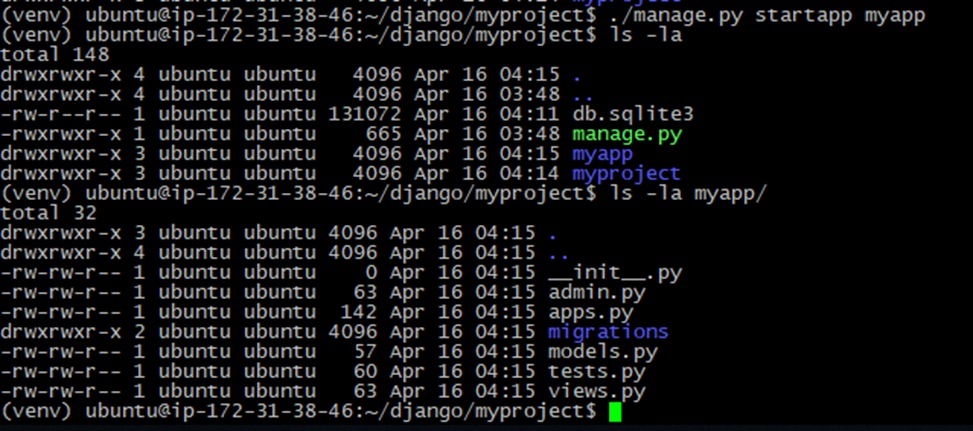
Этап 5. Создание приложения

Прикладная логика реализуется в Django приложениях. По умолчанию уже существуют служебные приложения Django, которые регистрируются в файле настроек проекта settings.py



Пользовательское приложение необходимо создать отдельно с помощью команды

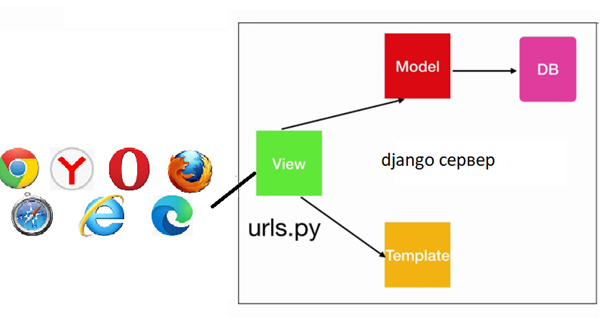
**./manage.py startapp “имя приложения”**

****

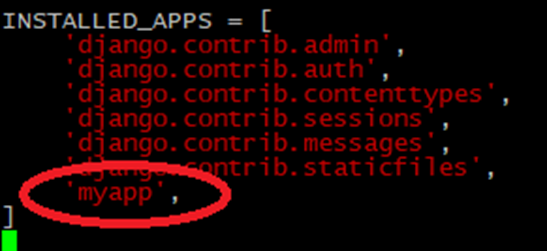
После этого появляется папка приложения, содержащая служебные файлы приложения с дефолтными настройками.

* admin.py – в этом файле регистрируются модели для добавления их в систему администрирования Django (использование сайта администрирования Django не является обязательным);
* apps.py – файл, содержащий основную конфигурацию приложения;
* migrations – папка, содержащая миграции базы данных приложения. Миграции позволяют Django отслеживать изменения моделей и синхронизировать их со схемой данных базы;
* models.py – модели данных приложения. В любом Django-приложении должен быть этот файл, но он может оставаться пустым;
* tests.py – этот файл предназначен для создания тестов для приложения;
* views.py – вся логика приложения описывается здесь. Каждый обработчик получает HTTP-запрос, обрабатывает его и возвращает ответ.

В Django используется модель для работы MVT (Model-View-Template).

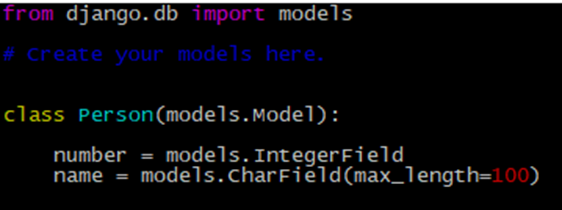


Созданное приложение необходимо записать в переменную INSTALLED\_APPS в файле settings.py.



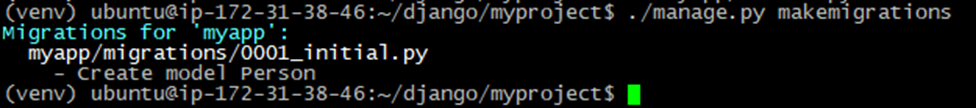
Этап 6. Создание модели.

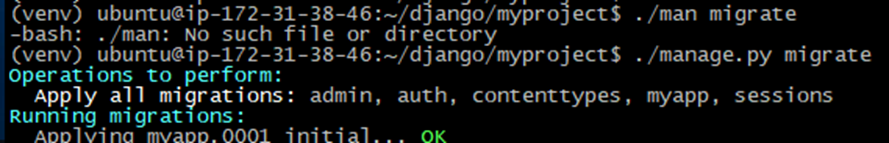
В файле models.py описывается модель



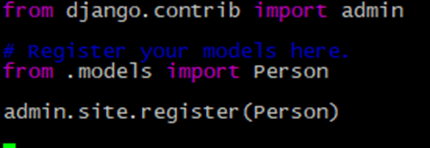
В данном примере модель представляет собой класс, содержащий поля двух типов – поле “number”, имеющее тип Integer, и поле “name”, имеющее тип Charstring. Для создания модели типы импортируются из файла models. Далее для модели будет создана таблица в базе данных приложения, имеющая ту же структуру (поля number и name соответствующих типов).

Для создания таблицы базы данных, соответствующей модели, необходимо выполнить миграцию

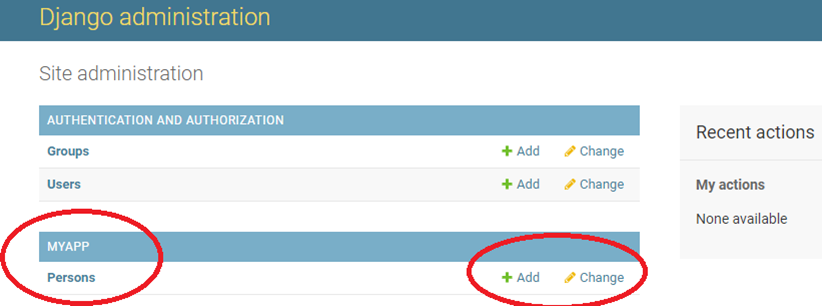




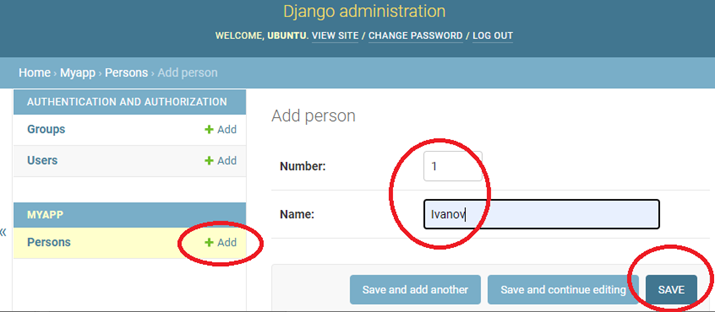
Данную модель необходимо зарегистрировать в файле admin.py для корректного отображения в панели администратора



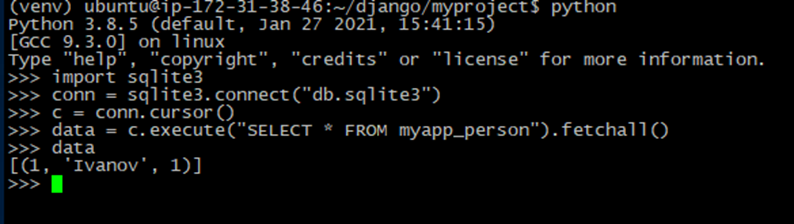
После этого модель Person приложения myapp появляется в панели администратора, можно добавлять новые экземпляры модели и редактировать существующие (это будет соответствовать операциям добавления и изменения записей в базе данных).



Этап 7. Создание новой модели.

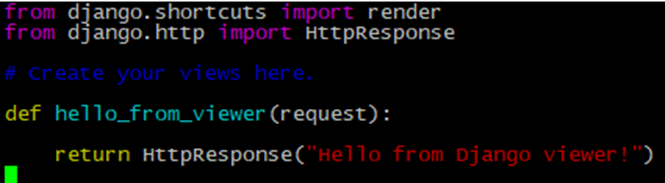


После нажатия кнопки Save происходит внесение новой записи в таблицу базы данных. Для модели Person приложения myapp создается таблица myapp\_person. Можно это проверить, выполнив SQL запрос.

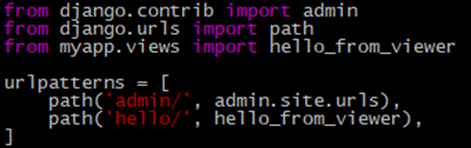


Этап 8. Создание контроллера (view).

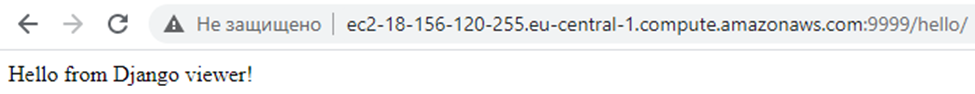
Контроллеры (view) описываются в файле views.py и предназначены для описания логики отображения информации пользователю.



В файле папки проекта url.py необходимо настроить маршрутизацию приложения по URL адресам, например, можно привязать URL /hello к выполнению созданного viewer.

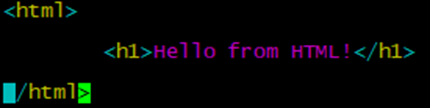


После этого можно через URL получить доступ к странице, создаваемой соответствующим viewer-ом.

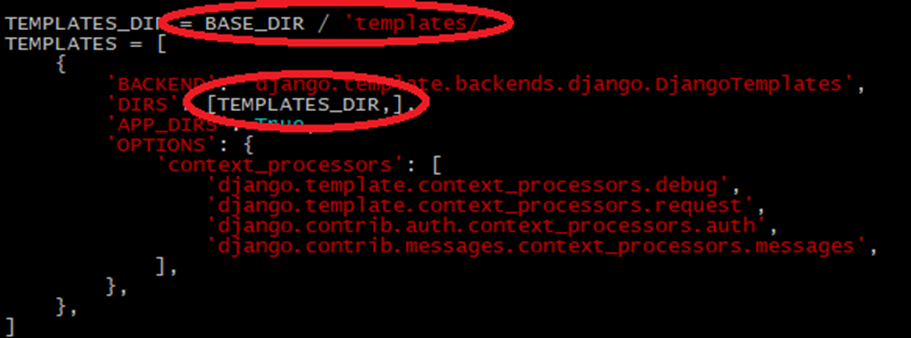


Этап 9. Создание контроллера (view) для отображения html страницы.

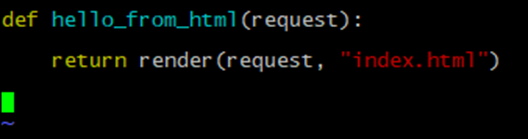
Необходимо создать html файл и разместить его в специальной директории (стандартным вариантом является папка templates).



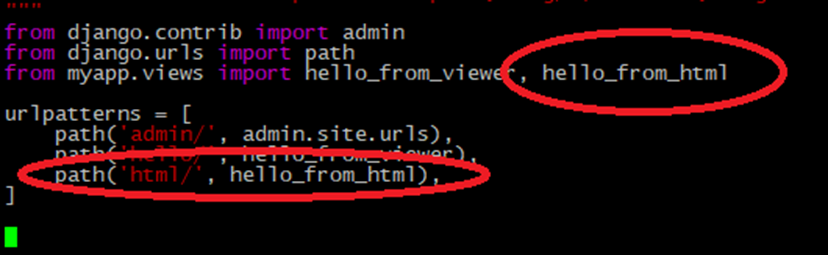
Путь до темплейтов (файлов html) необходимо прописать в settings.py



Необходимо создать контроллер (viewer), отображающее данный темплейт



И прописать маршрутизацию до данного представления на основе URL в файле urls.py



После этого можно открыть web страницу и увидеть содержание html (Django сервер должен быть запущен)



Давайте рассмотрим подробнее содержимое очень важного файла django проекта, settings.py. В этом файле устанавливаются значения ключевых переменных django проекта:

* DEBUG – булевое значение, которое включает и отключает режим отладки проекта. Если оно равно True, Django будет отображать подробные страницы с ошибками при возникновении исключений (exceptions) в приложении. При установке приложения в production нужно установить эту настройку в False в целях безопасности (чтобы скрыть от стороннего пользователя внутренние данные приложения).
* ALLOWED\_HOSTS – необходимо добавить домен сайта в эту настройку, для того чтобы Django мог с ним работать, в отладочных целях можно добавить значение [‘\*’] (разрешить доступ по всем доменам).
* INSTALLED\_APPS – описывает подключенные приложения (в том числе, созданные пользователем), по умолчанию подключаются
  + django.contrib.admin – сайт администрирования,
  + django.contrib.auth – подсистема аутентификации,
  + django.contrib.contenttypes – подсистема для работы с типами объектов системы,
  + django.contrib.sessions – подсистема сессий,
  + django.contrib.messages – подсистема сообщений,
  + django.contrib.staticfiles – подсистема для управления статическим содержимым сайта.
* MIDDLEWARE – список подключенных промежуточных слоев.
* ROOT\_URLCONF – указывает на Python-модуль, который содержит корневые шаблоны url ссылок для приложения.
* DATABASES – представляет собой словарь, содержащий настройки для всех баз данных проекта. Должна быть указана хотя бы одна база данных (по умолчанию подключена СУБД SQLite3).
* LANGUAGE\_CODE – определяет код языка по умолчанию для Django-сайта.
* USE\_TZ – указывает Django на необходимость поддержки временных зон. В Django включена возможность использовать объекты дат, учитывающие временные зоны. Эта настройка устанавливается в True, когда мы создаем проект с помощью команды startproject.

В Django есть важная полезная концепция Object Relation Model (ORM). ORM это концепция, в соответствии с которой объектам в языке программирования верхнего уровня ставятся в соответствие записи в базе данных, с которой связан фреймворк. Например, созданные на когда-либо ранее записи можно посмотреть не через SQL запросы, а с использованием ORM. Метод Person.object.all() получает все записи таблицы myapp\_persons базы данных, соответствующие модели Persons.

В Django необходимо настроить базу данных для хранения служебной и прикладной информации. Настройки, используемые для интеграции с базой данных, находятся в файле settings.py проекта и выглядят следующим образом (для примера sqlite)

**DATABASES = {**

**'default': {**

**'ENGINE': 'django.db.backends.sqlite3',**

**'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),**

**}**

**}**

Можно проверить правильность настроек с использованием командной строки в django

**python manage.py shell**

**import sqlite3**

**conn = sqllite3.connect(‘имя базы данных’)**

**c = conn.cursor()**

**c.execute("select \* from sqlite\_master")**

**c.fetchall()**

**c.execute("select \* from myapp\_modelapp")**

**c.fetchall()**

**c.execute(“INSERT INTO myapp\_mymodel VALUES (2, 2, \’Second name\’, \’2020-01-01 00:00:01\’)”)**

**conn.commit()**

Реляционная модель Django ORM основана на объектах запросов QuerySet. QuerySet это коллекция объектов, полученных из базы данных. К ней могут быть применены фильтрация и сортировка. Каждая модель Django имеет как минимум один менеджер модели, по умолчанию называемый objects. С его помощью мы получаем объект запроса QuerySet. Для того чтобы получить все объекты из таблицы, мы можем использовать метод all() стандартного менеджера:

**>>> all\_posts = Post.objects.all()**

Создание объекта

**>>> Post.objects.create(i1=’’, f2=’’)**

Чтение объекта из базы данных

**>>> Post.objects.get(f1=’’)**

**>>> p.save()**

html – формы

**{{p}}**

**{{f.as\_p}} показать форму с использованием формы Django**

**<html>**

**<body>**

**<form action=”.” method=”POST”>**

**{{form.as\_p}}**

**<input typr=”submit”>**

**</form>**

**</body>**

**</html>**

**{% for %} {% endfor %}**

**{% if %}{% else %}{% endif %}**

**{% block%} {% endblock %}**

**{% url %}**

Теперь у вас есть все, чтобы попробовать создать свой первый web сервис с использованием django. Конечно, за кадром осталось еще множество нюансов. Более подробно про фреймворк Django вы можете узнать из большого количества специальных книг и статей. Разобранной в этом приложении информации, в сочетании со сведениями о API, web сервере и базе данных, которые разбираются в следующих приложениях, достаточно для того, чтобы начать использовать Django в простой конфигурации для развертывания моделей машинного обучения в производственной среде.

*Тест:*

1. Какой скрипт используется для работы с проектом Django? (0.25)
   1. control.py
   2. **manage.py**
   3. django.py
   4. command.py
2. Что такое ORM? (0.25)
   1. Object Restricted Mode
   2. **Object Relation Model**
   3. Object Recognition Model
   4. Object Resolution Mode
3. Что такое миграции в Django? (0.25)
   1. копирование кода из одного проекта в другой
   2. **перенос информации из python кода django в соответствующие структуры базы данных**
   3. замена программиста в проекте
   4. изменение аппаратного обеспечения, смена провайдера облачных услуг для проекта.
4. Какие файлы Django проекта необходимы для описания основных функций? (0.25)
   1. **models.py**
   2. main.py
   3. **views.py**
   4. **urls.py**

*Итоги/выводы:*

Вы познакомились с фреймворком Django для быстрого создания приложений. Этот фреймворк организует работу различных сервисов, один из которых может быть моделью машинного обучения. Django берет на себя функцию организатора взаимодействия между элементами бэкенда (базы данных, вычисления, API) и фронтэнда, позволяя разработчику сосредоточится на развитии необходимых функций наиболее эффективным способом.

# Приложение 2. Внутреннее взаимодействие программных систем. API для взаимодействия микросервисов.

*Введение:* В этом приложении вы узнаете об инструменте взаимодействия между микросервисами, API, Application Programming Interface.

*Содержание:*

Многие современные программные системы имеют сложную структуру, состоят из различных компонентов (сервисов), взаимодействуют с другими информационными системами и устройствами. В связи с этим важным является механизм эффективного взаимодействия между разными информационными системами, либо между сервисами внутри одной системы. При этом такой механизм взаимодействия должен быть унифицированным, расширяемым, функциональным, удобным в использовании. Для решения такой задачи при разработке и эксплуатации информационных систем используют прикладной программный интерфейс (Application Programming Interface, API).

Существует несколько способов организации API, но наиболее часто применяют REST (REpresentational State Transfer). В терминологии REST API каждый URL (Uniform Resource Locator) называется ресурсом, с ресурсами осуществляются следующие действия:

* GET - возвращает описание ресурса,
* POST - добавляет новый ресурс,
* PUT - изменяет ресурс,
* DELETE - удаляет ресурс.

Для этой группы действий существует общее название CRUD (Create, Read, Update, Delete) и совместно действия GET, POST, PUT и DELETE предоставляют простой CRUD интерфейс для других приложений или сервисов, взаимодействие с которым происходит через протокол HTTP. Соответствие CRUD действий и HTTP методов:

* CREATE – POST,
* READ – GET,
* UPDATE – PUT,
* DELETE – DELETE.

Во взаимодействии через REST API применяются коды HTTP ответов (например, статусы вида «2xx» означают успешную операцию, «4xx» – ошибки при обработке запроса и т.п.). Общепринятые форматы взаимодействия REST – JSON и XML. При использовании JSON необходимо помнить, что для передаваемых объектов требуется сериализация (преобразование сложных данных, таких как наборы запросов queryset или объекты моделей Django, в типы данных, которые затем можно легко преобразовать в JSON, XML).

REST API интерфейс очень удобен для межпрограммного взаимодействия. Например, мобильное приложение может выступать в роли клиента, который манипулирует данными посредством REST.

Для веб-служб, созданных с использованием идеологии REST, применяют термин «RESTful».

В отличие от веб-сервисов (веб-служб) на основе SOAP, для RESTful веб-API не существует «официального» стандарта. REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом, большинство RESTful-реализаций используют такие стандарты, как HTTP, URL, JSON и XML.

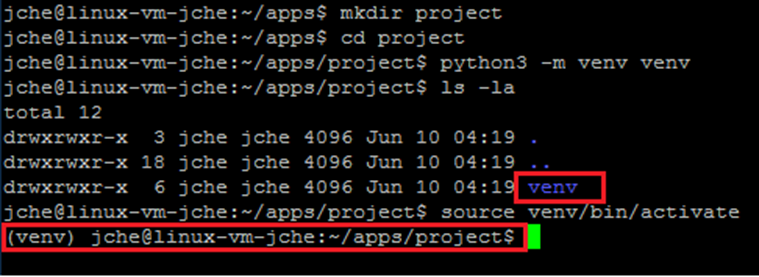
Рассмотрим пример создания API на базе библиотеки Django Rest Framework (DFR). Пример для операционной системы linux. Поэтапный порядок действий:

Этап 1. Создать и активировать виртуальное окружение, установить необходимые библиотеки.

**mkdir project**

**cd project**

**python3 –m env env**

****

После выполненных действий в текущей рабочей директории создастся директорий venv в которой находятся скрипты для работы с виртуальным окружением. В директорию venv будут устанавливаться все библиотеки.

Активировать виртуальное окружение:

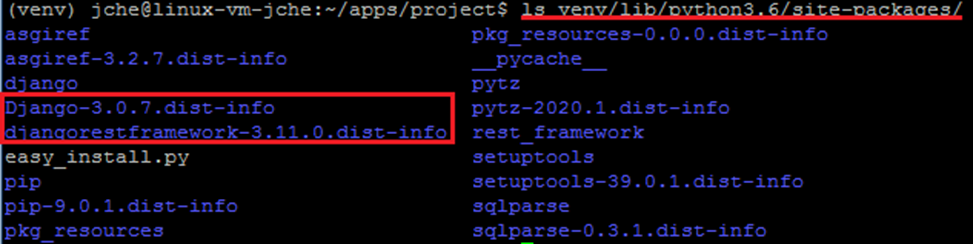
**source venv/bin/activate**

Установить django и djangorestframework:

**pip install django**

**pip install djangorestframework**

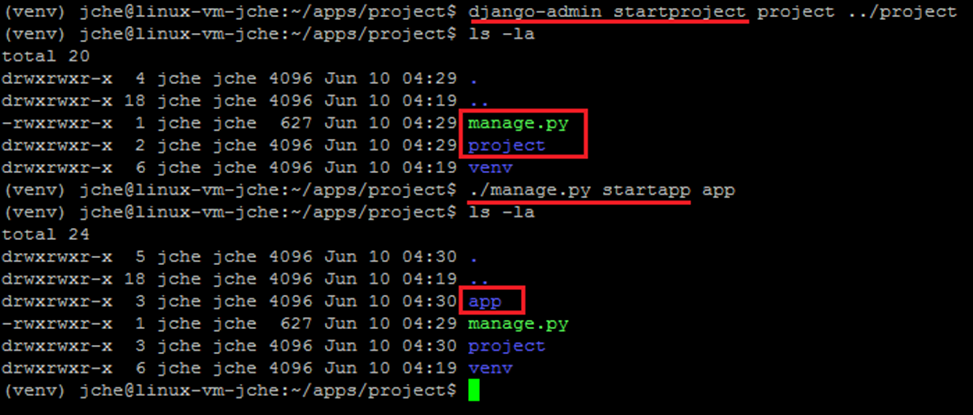
После этого в папке venv установятся необходимые файлы для работы с django, а также установятся необходимые значения переменных окружения.



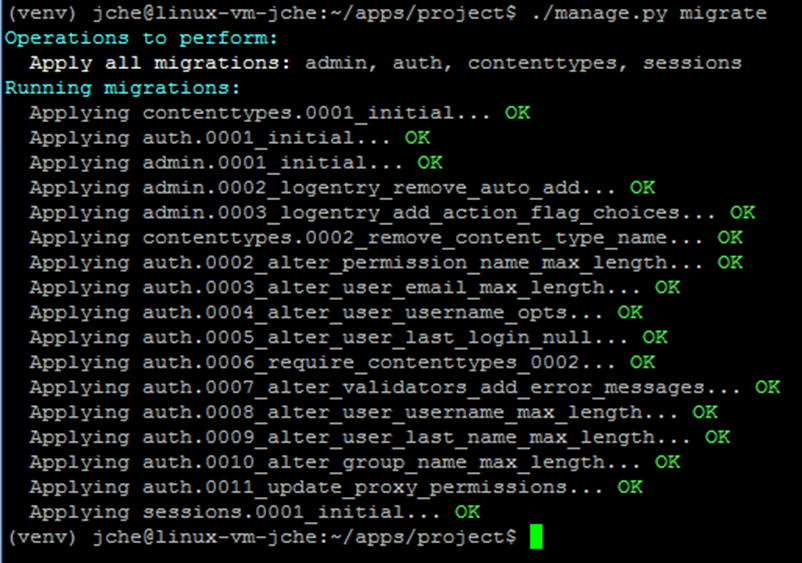
Этап 2. Создать django-проект и django-приложение, базы данных

**django-admin startproject project ../project**

**./manage.py startapp app**

****

**./manage.py migrate**

****

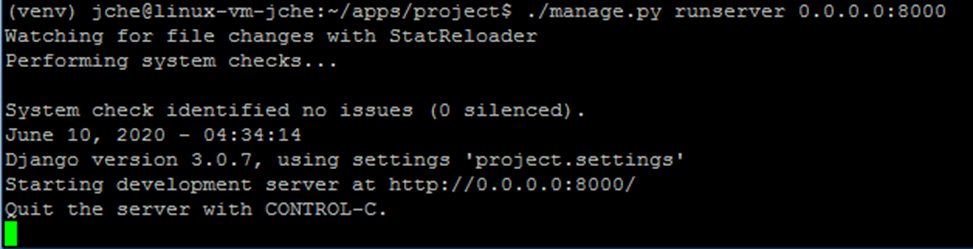
При выполнении этой команды создаются необходимые базы данных.

Редактирование файла project/settings.py. Необходимо отредактировать переменные: ALLOWED\_HOSTS, SECRET\_KEY, DEBUG.



Примечание: такие значения переменных SECRET\_KEY и DEBUG используются только в отладочных целях, не рекомендуется при запуске в боевом окружении задавать SECRET\_KEY в явном виде (лучше задавать через переменную окружения или из служебного файла) и устанавливать вывод внутренней информации django на экран (DEBUG=True) из соображений безопасности.

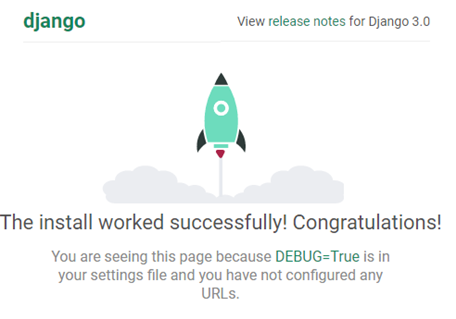
**./manage.py runserver 0.0.0.0:8000**

****

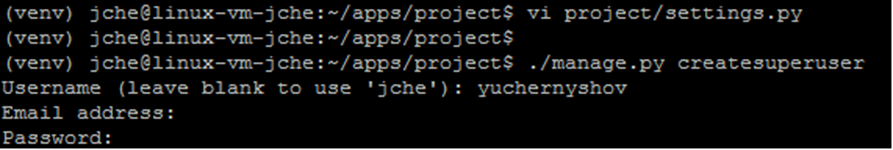
При выполнении этой команды запускается встроенный web-сервер на порте 8000.

После этого можно в браузере открыть страницу django

**http://<ваш IP адрес>:8000**

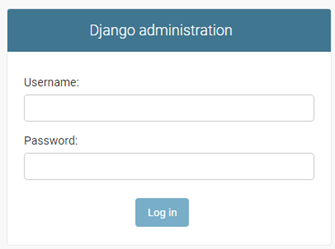


**./manage.py createsuperuser**



Эта команда создает суперпользователя (superuser) с помощью которого можно через панель администратора получать доступ к внутренней информации проекта django. Требуется задать имя пользователя и пароль (электронная почта - опционально). После этого можно открыть панель администратора проекта django (web-сервер должен быть по прежнему запущен).

**http://<ваш IP адрес>:8000/admin**

****

После ввода имени администратора и пароля открывается панель администратора

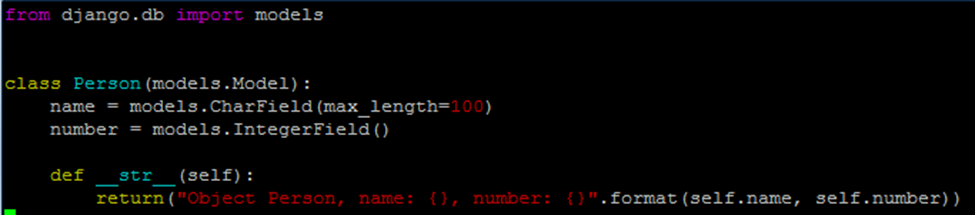


Этап 3. Добавить приложения app и rest\_framework в переменную INSTALLED\_APPS в файле project/settings.py

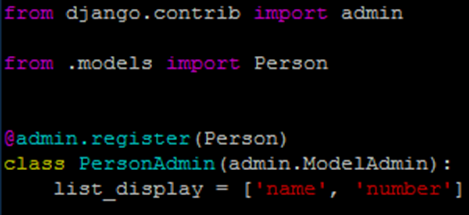


4. Добавление django модели Person в приложение app.

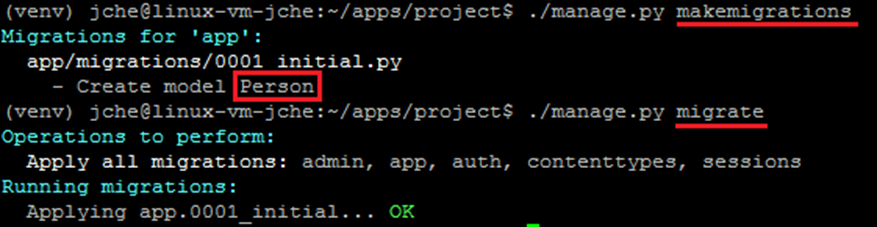
Описать класс модели в файле app/models.py



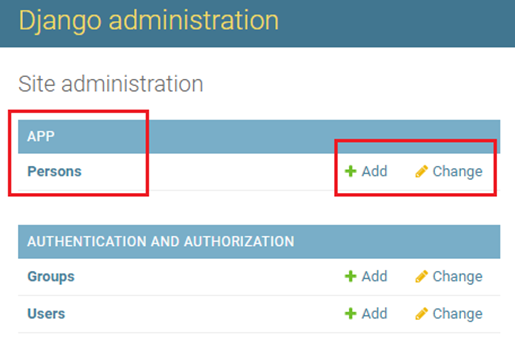
Зарегистрировать модель в admin.py



Сделать необходимые изменения в базе данных.



Теперь на панели администратора http://<ваш IP адрес>:8000/admin появился раздел моделей и модель Person. В панели администратора можно добавлять, редактировать и удалять объекты Person.



Этап 5. Реализация представления API (API view).

Вы наверняка знаете, что язык python является объектно-ориентированным языком программирования и все сущности в python являются классами. Для решения конкретных задач эффективнее всего воспользоваться уже существующими классами, унаследовав от них необходимые методы. Этот же подход применим и к нашей задаче создания API. В файле app/view.py давайте опишем класс person\_api\_view, который является наследником класса rest\_framework.views.APIView. Для класса person\_api\_view опишем методы

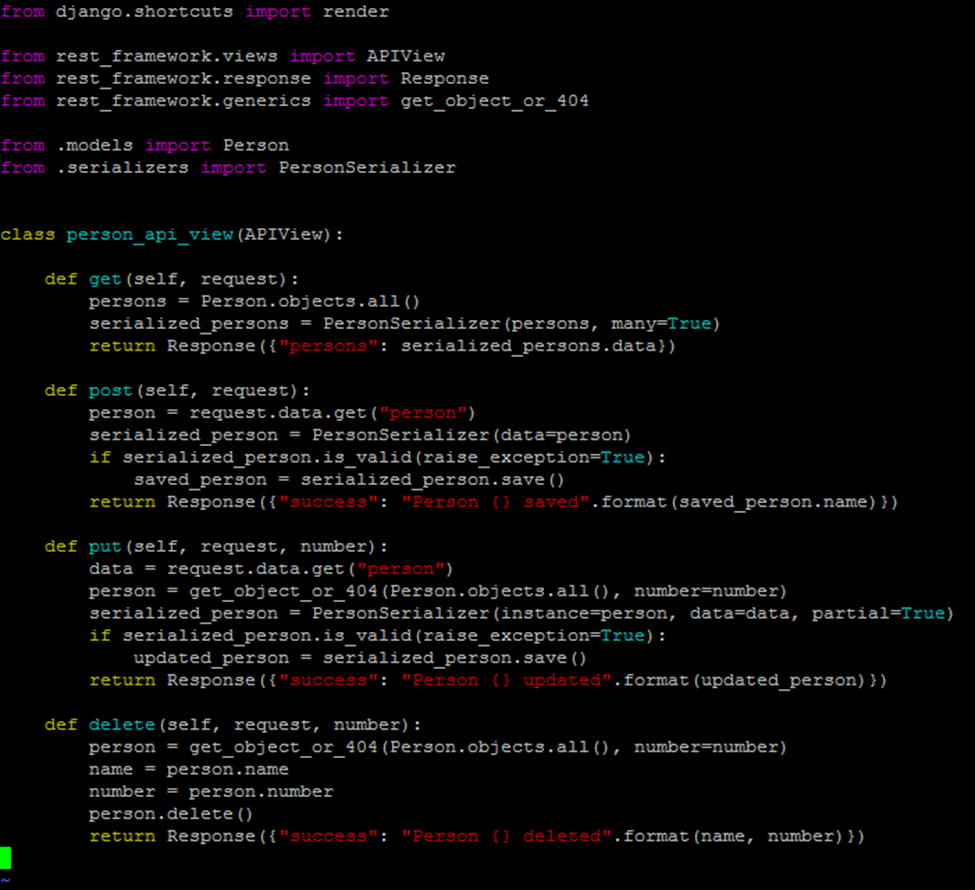
* get - получение информации об объектах Person в базе данных,
* post – публикация нового объекта Person,
* put – редактирование существующего объекта Person (идентификация по Person.number),
* delete – удаление существующего объекта Person (идентификация по Person.number).

Перед передачей объекта по API в другую систему необходимо этот объект сериализовать, т.е. привести к виду, удобному для эффективной передачи объекта. Для серилизации объектов Person создается класс PersonSerializer, который является наследником класса rest\_framework.serializers. Этот класс может быть описан в произвольном файле, но обычно используют serializers.py. В классе PersonSerializer описываются методы:

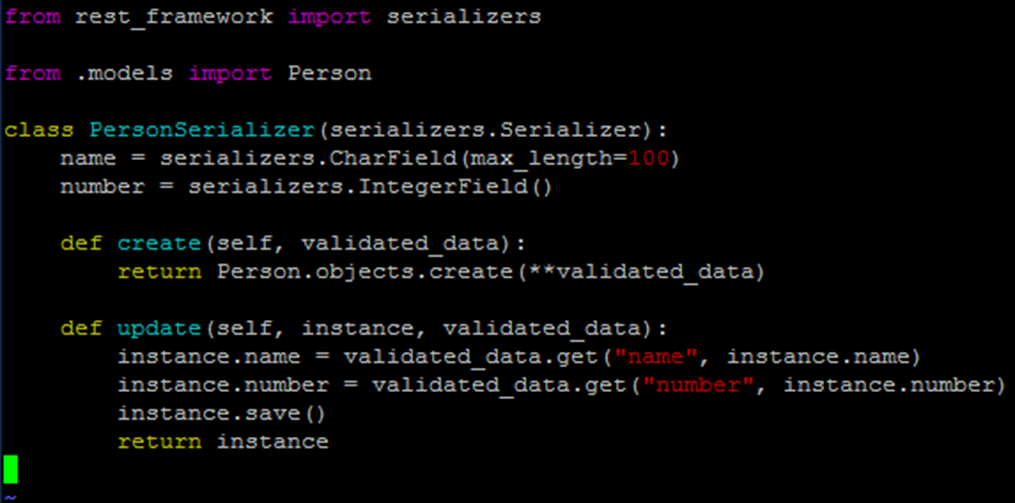
* create – используется при создании нового объекта (POST)
* update – используется при редактировании существующего объекта (PUT).

В файлах project/urls.py и app/urls.py должны быть прописаны правила обработки URL, в том числе обработка передаваемого параметра number в методах PUT и DELETE. Листинг для представления person\_api\_view и PersonSerializer приведен ниже. Рекомендуется реализовывать последовательно части, относящиеся к GET, POST, PUT, DELETE и проверять работу с использованием приложения chrome ARC (REST API клиент).

Описание представления person\_api\_view в файле app/views.pu

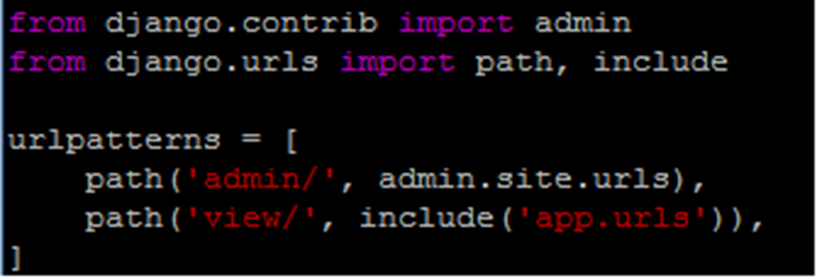


Описание сериализатора PersonSerializer в файле app/serializers.py

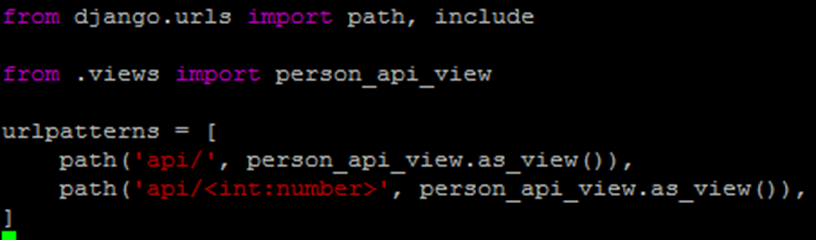


Этап 6. Описать правила направлений по url в файлах urls.py проекта и приложения.

Установить правило для обработки http://<ваш IP адрес>:8000/view (перенаправлять на правила, описанные в app/urls.py)



Установить правила в app/urls.py

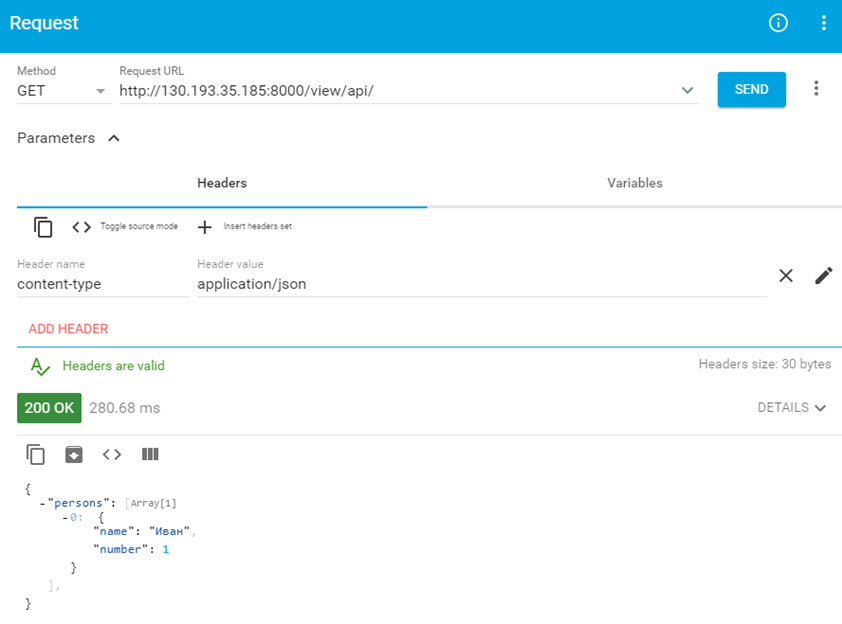


Этап 7. Проверка работы.

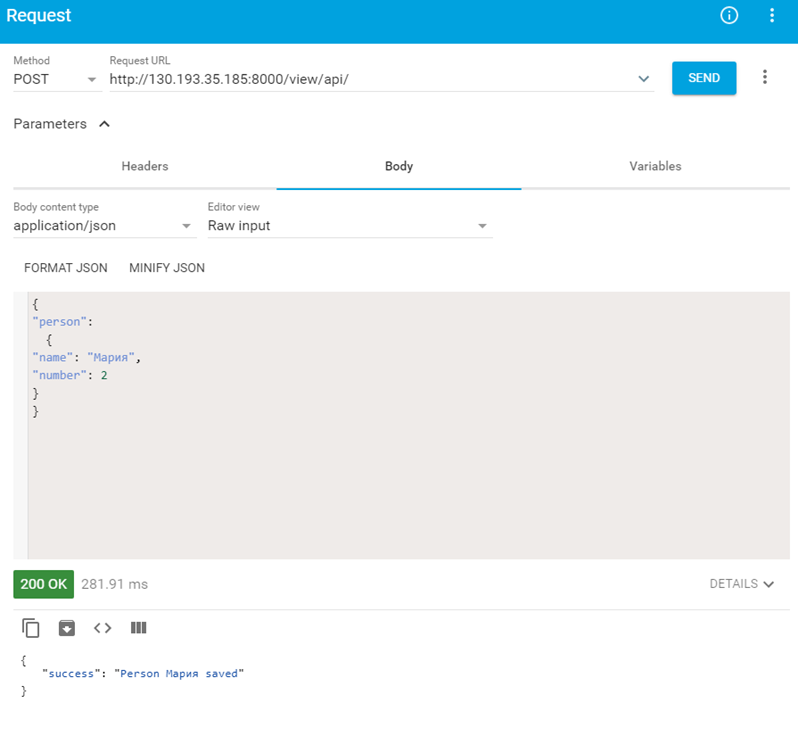
Для тестирования работы API можно использовать chrome, но удобнее работать со специальным приложением chrome ARC (является бесплатным, легко устанавливается). В запросах ниже вам надо использовать IP адреса ваших систем.

Создадим объект Person в панели администратора django (также далее можно использовать панель администратора для проверки выполнения команд API)

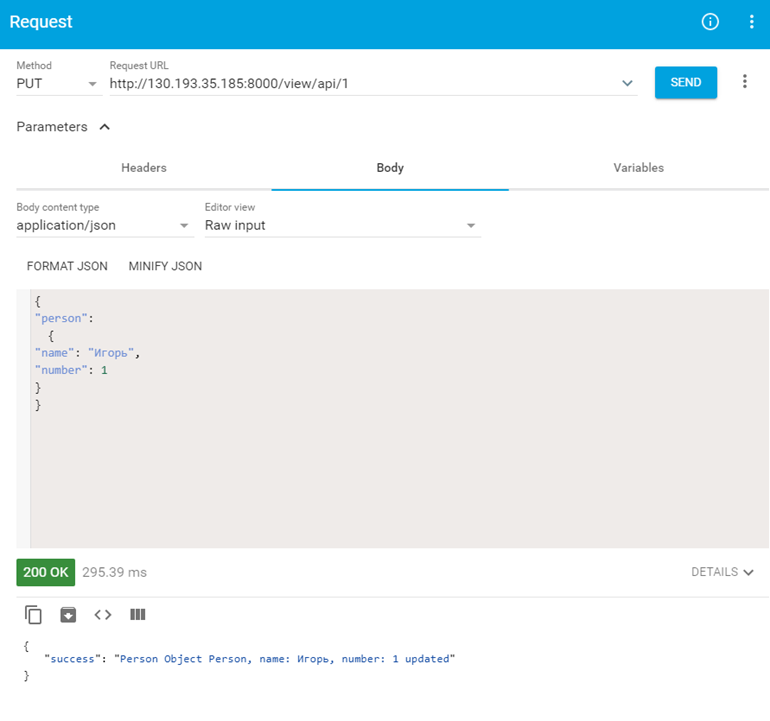
Запрос GET



Запрос POST



Запрос PUT



Запрос DELETE



Итак, вы узнали и практически поработали с мощным средством взаимодействия программных систем и сервисов между собой. Проектированию, созданию и тестированию API посвящено много литературы и статей.

API очень популярен, многие современные приложения предоставляют открытые API интерфейсы для использования другими системами. Социальные сети (Google, Twitter, Facebook), глобальные технологические платформы (Amazon, Microsoft Azure, Yandex), узкоспециальные прикладные системы (gismeteo.ru) – дают инструменты использования своих данных (информация о пользователях), либо использования встроенных функций для обработки данных (распознавание и синтез речи, распознавание изображений). Ниже перечислены некоторые из имеющихся в распоряжении ресурсов.

* Facebook API предоставляет интерфейс доступа к объектам социальной сети Facebook: посты, комментарии, лайки, перепосты. Эта обширная информация дает возможность для анализа данных для социальной инженерии. Есть удобный инструмент Facebook Graph API для извлечения данных с помощью R и python.

Ссылки:<https://developers.facebook.com/?locale=ru_RU>

Описание:<https://developers.facebook.com/docs/graph-api>

* Google Map API – одно из наиболее часто используемых API (области применения – от сервисов заказа такси до игры Pokemon Go). Дает возможность получать информацию: координаты, расстояния между объектами, маршруты и т.п. Интересной возможностью является создание свойств расстояний, учитывающих данных.

Ссылки:<https://console.developers.google.com/apis/dashboard?pli=1>

Описание:<https://developers.google.com/maps/documentation/maps-static/intro>

* Twitter API предоставляет доступ к данным: твиты, сделанные любым пользователем, твиты, содержащие отдельные термины или комбинации терминов, твиты, сделанные в определенный временной промежуток и т.п. Являются мощным инструментом в решении задач исследования социального мнения, настроения.

Описание:<https://developer.twitter.com/en/docs>

* IBM Watson предлагает набор из API для выполнения сложных задач, таких как анализ тона, преобразование документов, идентификация личности, визуальное распознавание, преобразование текста в голос и голоса в текст, и многие другие, с использованием нескольких строчек кода. Отличается от предыдущих тем, что предоставляет не инструменты доступа к данным, а обработки данных.

Описание:<https://www.ibm.com/watson>

* Quandl позволяет работать с временными рядами при анализе акций. Установка Quandl API очень проста и предоставляет ресурсы для решения задач анализа рынка акций.

Описание:<https://www.quandl.com/>

* Яндекс является одной из ведущих российских ИТ-компаний. В перечень разработок Яндекс входит множество сервисов для физических лиц и коммерческих компаний: заказ услуг (такси, еда), работа с картами, погода, распознавание и синтез речи и т.п.

Ссылка:<https://yandex.ru/dev/>

*Практическое задание:*

В данном приложении подробно описан процесс создания django проекта с поддержкой внешнего взаимодействия по API. Реализуйте этот проект самостоятельно и проверьте, что вы можете читать, писать, редактировать и удалять записи в базе данных с использованием API.

В качестве результата необходимо предъявить код проекта в git.

*Итоги/выводы:*

Для отдельных компонентов системы необходим протокол взаимодействия, хорошо описанный, непротиворечивый, легко расширяемый. API является хорошим средством для решения этой задачи. Вы научились создавать API интерфейс в популярном фреймворке Django.

# 

# Приложение 3. Внешнее взаимодействие программных систем. web-сервер, задачи и инструменты.

*Введение:* В этом приложении вы узнаете о том как программные системы взаимодействуют с внешним миром и о web-сервере, важном компоненте для организации такого взаимодействия. Любая самая крутая модель машинного обучения почти бесполезна, если она не может продемонстрировать результаты своей работы. Пользователи могут обращаться к веб-сервису для получения отчета, дашборда. Необходим сервис, который организует работу по обработке пользовательских запросов, передает запросы ответственным за их обработку сервисам и обеспечивает предоставление пользователям информации. web-сервер как раз и является таким сервисом.

*Содержание:*

**Веб-сервер** является важным компонентом в программных комплексах, работающих с web-технологиями, обеспечивает обработку запросов от веб-клиентов (веб-браузеров, информационных систем, служебных утилит и т.п.). Веб-клиент отправляет веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы это HTML-страницы, изображения, файлы, медиа-потоки, результаты работы модели машинного обучения или другие данные, которые необходимы клиенту. В ответ на запрос веб-сервер передаёт клиенту данные по протоколу HTTP. Терминологически веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно аппаратное обеспечение (компьютер, сервер), на котором работает программное обеспечение.

**HTTP** (Hypertext Transfer Protocol, протокол передачи гипертекстовых сообщений) является основным протоколом сети Интернет. Этот протокол хорошо стандартизован, стандарт описан в документе RFC2616 (https://tools.ietf.org/html/rfc2616). RFC (Request For Comment) – группа стандартов, описывающих правила работы сети Интернет, а также некоторых смежных технологий. Стандарты RFC разрабатываются рабочей группой IETF (Internet Engineering Task Force). HTTP описывает взаимодействие между клиентом и сервером с использованием сообщений: запрос (Request) и ответ (Response). Каждое сообщение состоит из трех частей: стартовая строка (Request/Response Line), заголовки (Header fields) и тело сообщения (Message Body).

Примеры стартовой строки сообщения:

Запрос:

**GET /hello.htm HTTP/1.1**

Ответ:

**HTTP/1.1 200 OK**

Пример заголовка:

**User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3**

**Host: www.usurt.ru**

**Accept-Language: en, ru**

**Date: Mon, 4 May 2020 12:28:53 GMT**

**Server: Apache**

**Content-Type: text/plain**

Пример тела сообщения:

**<html>**

**<body>**

**Hello!**

**</body>**

**</html>**

Стартовая строка является обязательным полем (mandatory), заголовки и тело сообщения – необязательными (optional). Стартовые строки для запроса и ответа имеют различный формат. Стартовая строка запроса, имеет вид:

**METHOD URI HTTP/VERSION**,

где

* METHOD - метод HTTP-запроса,
* URI - идентификатор ресурса,
* VERSION - версия протокола (на данный момент актуальна версия 1.1).

Заголовки это набор пар «имя-значение», разделенных двоеточием. В заголовках передается служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент и другие данные.

Тело сообщения это передаваемые данные. В ответе передаваемыми данными, как правило, является html-страница, которую запросил клиент, а в запросе в теле сообщения может передаваться содержимое файлов, загружаемых на сервер. Как правило, тело сообщения в запросе отсутствует.

В стартовой строке запроса есть параметр URI (Uniform Resource Identifier, единообразный идентификатор ресурса). Обычно ресурсом является файл на сервере (в этом случае URI может выглядеть так: '/styles.css'), но вообще ресурсом может являться и какой-то абстрактный объект, например файловая директория (в этом случае URI имеет вид'/blogs/webdev/').

Тип HTTP-запроса (также называемый HTTP-метод) указывает серверу на то, какое действие требуется выполнить с ресурсом. В первых версиях протокола HTTP действие было только одно – получение ресурса (метод GET). В дальнейшем в протоколе HTTP появилась возможность создавать ресурсы (метод POST), редактировать (метод PUT), удалять (метод DELETE) и многое другое.

Спецификация HTTP не обязывает сервер понимать все методы, обязателен только GET.

Список других методов можно найти, например, здесь:

<https://www.tutorialspoint.com/http/http_methods.htm>

Кроме основной функции (обработка запросов клиентов) веб-серверы могут выполнять различные дополнительные функции, например:

* автоматизация работы веб-страниц,
* ведение журналов (логов), например, регистрация обращений пользователей к ресурсам,
* аутентификация и авторизация пользователей,
* поддержка динамически генерируемых страниц,
* поддержка защищенного протокола HTTPS для защищенных соединений с клиентами,
* функции почтового сервера,
* выполнение функций прокси-сервера.

Прокси-сервер это промежуточный сервер (или комплекс программ) в компьютерных сетях, пользователем и целевым сервером. Прокси-сервер позволяет клиентам выполнять запросы (принимая и передавая их через прокси-сервер) к другим сетевым службам и получать ответы. Алгоритм работы: 1) клиент подключается к прокси-серверу и запрашивает ресурс, расположенный на целевом сервере; 2) прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного хранилища данных (кэша).

В некоторых случаях запрос клиента или ответ сервера может быть изменён прокси-сервером в определённых целях. Прокси-сервер позволяет защищать компьютер клиента от некоторых сетевых атак и помогает сохранять анонимность клиента.

Примеры использования прокси-серверов:

* обеспечение защищенного доступа компьютеров локальной сети к сети Интернет,
* кэширование данных: сохранение ресурсов для наиболее частых запросов для ускорения доступа,
* сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде для экономии сетевого трафика,
* ограничение доступа,
* злонамеренные цели: сокрытие параметров доступа, анонимизация действий.

Вот некоторые популярные веб-серверы:

|  |  |
| --- | --- |
| nginx | <https://nginx.org/ru/> |
| apache | <https://www.apache.org/> |
| gunicorn | <https://gunicorn.org/> |
| uwsgi | <https://uwsgi-docs.readthedocs.io/en/latest/> |
| lighttp | <https://www.lighttpd.net/> |

nginx является наиболее популярным из перечисленных веб-серверов, обладая высокой производительностью, надежностью, богатым набором функций.

Администрирование nginx осуществляется через командную строку и конфигурационные файлы. Вот некоторые команды, которые используются для nginx

|  |  |
| --- | --- |
| service nginx status | Текущий статус |
| service nginx start | Запуск сервера |
| Service nginx stop | Остановка сервера |

Настройки nginx хранятся в директории /etc/nginx/service-enabled

Лог-файлы можно задавать опционально в конфигурационном файле, обычно для хранения логов указывают директорию /var/logs/nginx

Пример конфигурационного файла

**server {**

**listen 80; # слушать порт 80**

**server\_name 84.201.170.215; # имя или IP адрес сервера**

**location /static/ { # для запросов с /static/ использовать папку из alias**

**autoindex on; # включить автопросмотр содержимого папки**

**alias /home/jche/apps/test\_api/static/;**

**}**

**location / { # перенаправлять запросы на другое приложение, которое слушает 8000 порт**

**proxy\_pass http://127.0.0.1:8000;**

**}**

**}**

Веб-сервер nginx обычно запускается в отдельном контейнере в виде отдельного микросервиса, к которому обращаются другие микросервисы через сетевые соединения. Это самый распространенный вариант для организации работы системы в производственной среде. Однако для стендов разработки (Dev) и тестирования (Test, Stage) возможности nginx как правило избыточны, а ресурсов требуется много. По этой причине на серверах разработки в окружении используются более легкие веб-серверы, например, gunicorn.

Вот пример запуска gunicorn и django

**gunicorn –bind 0.0.0.0:8000 <имя django приложения>.wsgi**

После выполнения этой команды запустится django приложение на номере порта 8000, при этом будет использоваться не маломощный встроенный веб-сервер django, а сервер gunicorn. В промышленном варианте django приложение, включающее модель машинного обучения и веб-сервер gunicorn, запускается в одном контейнере, а основной высокопроизводительный веб-сервер nginx в другом. При этом пользователь общается только с nginx и не может обратиться к модели машинного обучения напрямую, что дает приложению дополнительную надежность и возможность распределять нагрузку.

*Практическое задание:*

Запустите django приложение, разработанное в Приложении 2, с web-сервером gunicorn вместо встроенного django web-сервера, использующегося только для локальной разработки приложений.

*Итоги/выводы:*

Веб-сервер является важнейшей частью клиент-серверных приложений. Большинство современных систем имеет клиент-серверную архитектуру, в том числе и в проектах машинного обучения. Это позволяет обращаться к системе как к сервису, что делает использование более универсальным. Веб сервер может выполнять множество полезных функций: эффективная и быстрая обработка клиентских запросов, защита, распределение нагрузки. Наиболее эффективным для больших проектов является nginx.

# Приложение 4. Базы данных.

*Введение:* Данные являются важнейшей частью проекта машинного обучения. Одним из распространенных инструментов для хранения данных являются базы данных. Кроме данных в базах данных можно хранить системные настройки, информацию о модели и многое другое. Сервис базы данных является важной частью проекта машинного обучения, а иногда и основным инструментом для работы, поскольку многие системы управления базами данных уже содержат в себе элементы аналитики и даже отдельные функции машинного обучения. В этом приложении вы узнаете об использовании баз данных для хранения, обработки и анализа информации.

*Содержание:*

Базы данных – это программы, которые позволяют сохранять, обрабатывать и получать большие объемы связанной информации. Реляционные базы данных состоят из таблиц, которые содержат информацию. Когда вы создаете реляционную базу данных необходимо подумать о том, какие таблицы вам нужно создать и какие связи существуют между информацией в таблицах. Иначе говоря, вам нужно подумать о проекте вашей базы данных. Хороший проект базы данных обеспечивает целостность данных и простоту работы с ними. Общепринятый язык для запросов в реляционные базы данных это SQL, Structured Query Language. Примеры реляционных баз данных: sqlite, mysql, postgresql.

Существуют также нереляционные базы данных, еще их называют NoSQL базами данных. Это базы данных, оптимизированные для хранения элементов определенной структуры, например, временных рядов, изображений, текстов, логов и т.п. Примеры таких баз: MongoDB, TerminusDB, EdgeDB и т.п.

Для практического ознакомления с базами данных давайте рассмотрим модуль sqlite. sqlite – это автономный, работающий без сервера транзакционный механизм базы данных SQL. В python модуль sqlite3 появился в версии 2.5, с помощью этого модуля можно создавать базу данных SQLite в любой версии Python, без необходимости скачивания дополнительных инструментов.

Создать базу данных в SQLite очень просто, но для понимания процесса необходимо знать синтаксис и логику работы языка запросов SQL. Для SQL существует большое количество книг и статей, с которыми вы можете ознакомиться самостоятельно. Давайте разберемся как с помощью python кода можно создать базу данных для хранения информации:

**import sqlite3**

**conn = sqlite3.connect("mydatabase.db")**

**cursor = conn.cursor()**

**# Создание таблицы**

**cursor.execute("""CREATE TABLE albums**

**(title text, artist text, release\_date text,**

**publisher text, media\_type text)**

**""")**

Сначала нам нужно [импортировать](https://python-scripts.com/import-modules-python) модуль sqlite3 и создать связь с базой данных. Вы можете передать название файла или просто использовать специальную строку “:memory:” для создания базы данных в памяти. В нашем случае, мы создаем его на диске в файле под названием mydatabase.db.

Далее мы создаем объект cursor, который позволяет нам взаимодействовать с базой данных и добавлять записи, помимо всего прочего. При этом используется SQL для создания таблицы под названием “albums” со следующими полями: title, artist, release\_date, publisher и media\_type. SQLite поддерживает только пять типов данных: null, integer, real, text и blob. Давайте напишем этот код, который добавит данные в нашей новой таблице. Если вы запускаете команду CREATE TABLE, при этом база данных уже существует, вы получите сообщение об ошибке.

**cursor.execute("""INSERT INTO albums**

**VALUES ('Glow', 'Andy Hunter', '7/24/2012',**

**'Xplore Records', 'MP3')"""**

**)**

**# Сохраняем изменения**

**conn.commit()**

**# Вставляем множество данных в таблицу используя безопасный метод "?"**

**albums = [('Exodus', 'Andy Hunter', '7/9/2002', 'Sparrow Records', 'CD'),**

**('Until We Have Faces', 'Red', '2/1/2011', 'Essential Records', 'CD'),**

**('The End is Where We Begin', 'Thousand Foot Krutch', '4/17/2012', 'TFKmusic', 'CD'),**

**('The Good Life', 'Trip Lee', '4/10/2012', 'Reach Records', 'CD')]**

**cursor.executemany("INSERT INTO albums VALUES (?,?,?,?,?)", albums)**

**conn.commit()**

Вы использовали команду “INSERT INTO …”, чтобы вставить запись в базу данных. Обратите внимание на то, что каждый объект находится в одинарных кавычках. Это может усложнить работу, если вам нужно вставить строки, которые содержат одинарные кавычки. В любом случае, чтобы сохранить запись в базе данных, нам нужно создать её. Следующая часть кода показывает, как добавить несколько записей за раз при помощи метода курсора [executemany](https://python-scripts.com/sqlite). Обратите внимание на то, что мы используем знаки вопроса (?), вместо строк замещения (%) чтобы вставить значения. Обратите внимание, что использование строки замещения не безопасно, так как может стать причиной появления атаки инъекций SQL . Использование знака вопроса намного лучше, а использование SQLAlchemy тем более, так как он делаете все необходимое, чтобы уберечь вас от обработки встроенных одинарных кавычек на то, что SQLite в состоянии принимать.

Возможность обновлять записи в базе данных позволяет поддерживать существующие данные в актуальном состоянии, своевременно внося изменения. Если редактировать данные сложно или невозможно, то база станет быстро станет неактуальной и бесполезной. Иногда вам, в том числе, нужно будет удалять и строки.

**import sqlite3**

**conn = sqlite3.connect("mydatabase.db")**

**cursor = conn.cursor()**

**sql = """**

**UPDATE albums**

**SET artist = 'John Doe'**

**WHERE artist = 'Andy Hunter'**

**"""**

**cursor.execute(sql)**

**conn.commit()**

Здесь мы использовали команду SQL UPDATE, чтобы обновить таблицу альбомов. Здесь вы можете использовать команду SET, чтобы изменить поле, так что в нашем случае мы изменим имя исполнителя на John Doe в каждой записи, где поле исполнителя указано для Andy Hunter. Весьма просто, не так ли? Обратите внимание на то, что если вы не подтвердите изменения, то они не будут внесены в базу данных. Команда DELETE настолько же проста. Давайте посмотрим.

**import sqlite3**

**conn = sqlite3.connect("mydatabase.db")**

**cursor = conn.cursor()**

**sql = "DELETE FROM albums WHERE artist = 'John Doe'"**

**cursor.execute(sql)**

**conn.commit()**

Удаление еще проще, чем обновление. У SQL это занимает всего две строчки. В данном случае, все, что нам нужно сделать, это указать SQLite, из какой таблицы удалить (albums), и какую именно запись при помощи пункта WHERE. Таким образом, был выполнен поиск записи, в которой присутствует имя “John Doe” в поле исполнителей, после чего эти данные были удалены.

*Практическое задание:*

Создайте базу данных для хранения информации о данных о пассажирах “Титаника” тренировочной части датасета. Используйте базу данных SQLite, для которой есть существующая python библиотека. Пример работы с SQLite в python, который поможет вам справиться с заданием:

**import sqlite3**

**conn = sqllite3.connect(‘имя базы данных’)**

**c = conn.cursor()**

**c.execute("select \* from sqlite\_master")**

**c.fetchall()**

**c.execute("select \* from myapp\_modelapp")**

**c.fetchall()**

**c.execute(“INSERT INTO myapp\_mymodel VALUES (2, 2, \’Second name\’, \’2020-01-01 00:00:01\’)”)**

**conn.commit()**

В качестве результата вам необходимо в jupyter ноутбуке описать создание такой базы и чтение из нее с использованием SQL запроса информации о пассажирах, в титуле которых присутствует ‘Mrs’.

*Итоги/выводы:*

Работа с данными это важнейшая часть проекта машинного обучения. Управлению данными в проекте машинного обучения был посвящен отдельный Модулю 4. В этом юните мы обсудили какой может быть эффективная файловая структура проекта, а также назначение и практические аспекты использования баз данных в проектах машинного обучения.

# Приложение 5. Установка и настройка JupyterHub.

*Введение:*

Продукты, входящие в экосистему Jupyter, являются очень популярными среди исследователей данных и инженеров машинного обучения. Интерактивные 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 ноутбуки является цифровым аналогом “рабочей тетради ученого”, которая позволяет ему вести интерактивные исследования, включая вычисления, эксперименты, графики.

В этом приложении описывается процедура установки и настройки системы JupyterHub, являющейся многопользовательским сервером для пользователей Jupyter-ноутбуков, удобным для командной работы.

*Содержание:*

Установка JupyterHub выполняется следующим образом:

**python3 -m pip install jupyterhub**

**sudo apt-get install npm nodejs**

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

Примечания:

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

**sudo apt install python3-pip**

* 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 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 закоментированные 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/.

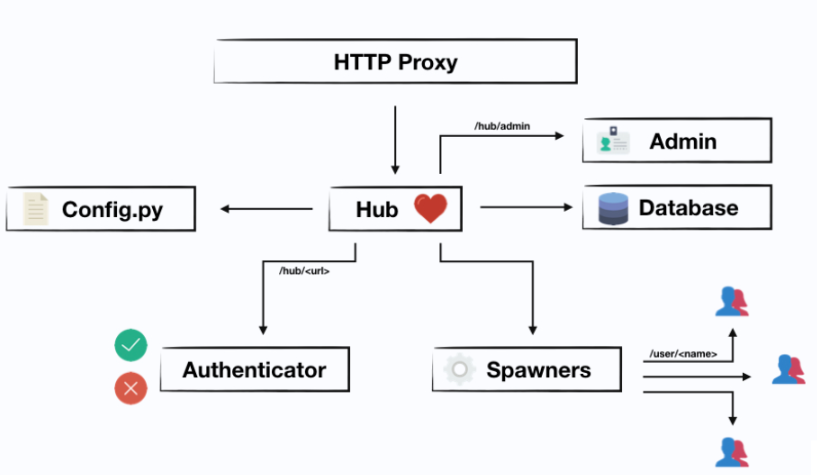


Рисунок “Схема работы JupyterHub”

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

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

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.