

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности



С.Т. Князев

«24» декабря 2021 г.

Автоматизация машинного обучения

Учебно-методические материалы по направлению подготовки
09.04.01 Информатика и вычислительная техника
Образовательная программа «Инженерия искусственного интеллекта»

Екатеринбург

2021

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Директор ИШИАО, доцент,
канд.техн.наук

A handwritten signature in blue ink, appearing to read 'Андрей Владимирович', is written over a horizontal line.

Созькин Андрей
Владимирович

СОДЕРЖАНИЕ

Модуль 1. Введение.	4
Модуль 1. Юнит 1: Зачем надо изучать MLOps и что для этого требуется?	5
Модуль 1. Юнит 2: Из чего состоит MLOps?	14
Модуль 1. Юнит 3: Управление проектом и MLOps.	22
Модуль 1. Юнит 4: Команда ML проекта (роли)	29
Модуль 1. Юнит 5: Обзор инструментов ML/MLOps	35
Модуль 1. Юнит 6: Учет бизнес-требований и их влияние на MLOps	41
Модуль 1. Юнит 7: Обзор требований к специалистам MLOps	45
Модуль 1. Юнит №8	51
Модуль 1. Юнит №9: Практическое задание к модулю	52
Модуль 2. Непрерывная интеграция, доставка и обучение (CI/CD/CT) в проектах ML	53
Модуль 2. Юнит 1. Автоматизация. Непрерывная интеграция, развертывание и тестирование.	54
Модуль 2. Юнит 2. Задачи и инструменты CI/CD/CT в проектах машинного обучения	61
Модуль 2. Юнит 3. Знакомство с Jenkins. Установка.	67
Модуль 2. Юнит 4. Использование Jenkins	71
Модуль 2. Юнит 5. Пример практического применения Jenkins для проекта машинного обучения	83
Итоги модуля	90
Практическое задание модуля	90
Модуль 3. Виртуализация и контейнеризация.	92
Модуль 3. Юнит 1. Различные подходы к созданию изолированных программных сред.	93
Модуль 3. Юнит 2. Виртуализация с использованием VirtualBox.	99
Модуль 3. Юнит 3. Виртуальные окружения.	113
Модуль 3. Юнит 4. Основы контейнеризации с docker. Установка и настройка.	121
Модуль 3. Юнит 5. Базовые команды docker.	129
Модуль 3. Юнит 6. Практический пример использования docker.	135
Итоги/выводы по модулю	137
Практическое задание	137
Список терминов и сокращений	137
Дополнительные материалы	138
Модуль 4. Управление данными.	142
В этом модуле:	142
Модуль 4. Юнит 1. Задачи инженерии данных	143
Модуль 4. Юнит 2. Архитектура для работы с данными. Принципы работы ETL систем.	150

Модуль 4. Юнит 3. Инструменты для управления данными. Примеры использования dvc	156
Практическое задание на модуль	176
Модуль 5. Конвейеры операций	177
Модуль 5. Юнит 1. Конвейеры операций.	178
Модуль 5. Юнит 2. Apache Airflow	182
Модуль 5. Юнит 3. Создание конвейера операций в Airflow.	195
Итоги/выводы по модулю	203
Практическое задание	203
Список источников	203
Модуль 6. Автоматизация тестирования.	204
Модуль 6. Юнит 1. Виды тестирования.	205
Модуль 6. Юнит 2. Тестирование данных	211
Модуль 6. Юнит 3. Модульное и функциональное тестирование	217
Итоги/выводы по модулю	223
Практическое задание	223
Список источников	224
Модуль 7. Запуск проекта в промышленном окружении	226
Модуль 7. Юнит 1. Архитектуры программных проектов. Преимущества микросервисной архитектуры для проектов машинного обучения.	227
Модуль 7. Юнит 2. Сервисы и инструменты в проектах машинного обучения	231
Модуль 7. Юнит 3. Организация рабочего пространства проекта машинного обучения	235
Модуль 7. Юнит 4. Системы управления конфигурациями	241
Модуль 7. Юнит 5. Пример создания инфраструктуры для проекта машинного обучения	253
Итоги/выводы по модулю	259
Практическое задание по модулю	259
Список источников	260
Приложение 1. Бекэнд программных систем на примере фреймворка Django.	261
Приложение 2. Внутреннее взаимодействие программных систем. API для взаимодействия микросервисов.	277
Приложение 3. Внешнее взаимодействие программных систем. web-сервер, задачи и инструменты	290
Приложение 4. Базы данных	295
Приложение 5. Установка и настройка JupyterHub	299
Модуль 8. Эксплуатация моделей машинного обучения	301
Модуль 8. Юнит 1. Утилиты мониторинга и управления ресурсами в linux.	303
Модуль 8. Юнит 2. Утилита Tensorboard для мониторинга.	321

Модуль 8. Юнит 3. Визуализация и контроль с использованием Grafana.	326
Модуль 8. Юнит 4. Практический пример использования Grafana для проекта машинного обучения.	339
Итоги/выводы по модулю	343
Полезные ссылки	344

Модуль 1. Введение.

Содержание модуля:

В этом модуле вы познакомитесь с дисциплиной «Автоматизация машинного обучения», или, как чаще называют эту методологию на практике, MLOps. Информации в модуле будет достаточно для понимания большинства задач дисциплины и методов их решения, даже если вы раньше никогда не слышали о DevOps/MLOps. Поскольку это общее предварительное знакомство, многие сведения изложены обзорно, чтобы сформировать предварительное представление о дисциплине. Более подробно эти сведения изучаются в следующих модулях курса.

В этом модуле слушатели:

- узнают, как появилась методология MLOps и почему она так востребована,
- ознакомятся с терминологией MLOps, базовыми понятиями и методами,
- изучат важные аспекты управления проектами машинного обучения: жизненный цикл проекта, состав команды и роли участников проекта,
- познакомятся с наиболее популярными инструментами для решения задач MLOps,
- установят важную взаимосвязь между техническими требованиями к проекту машинного обучения и содержанием задач MLOps.

Изучение любой дисциплины не начинается “с нуля”, поэтому в модуле описывается перечень знаний, необходимых для быстрого и эффективного освоения излагаемого в курсе материала.

Знания, полученные при изучении этого модуля, создадут хорошую основу для понимания информации в следующих модулях. Удачи!

Модуль 1. Юнит 1: Зачем надо изучать MLOps и что для этого требуется?

В этом юните:

Слушатели узнают почему знание методологии MLOps очень востребовано, что необходимо изучать, чтобы стать специалистом MLOps, и какими навыками необходимо обладать, чтобы обучение было наиболее эффективным.

Содержание юнита:

Индустрия программного обеспечения, появившаяся в середине XX века, в настоящее время является важной производственной отраслью. Все чаще информационные системы и искусственный интеллект заменяют человеческий труд. Цифровые технологии успешно заменяют человека в таких задачах как обработка документов, управление производством и процессами, консультирование, перевод и даже создание произведений искусства. И дальше эта тенденция будет только усиливаться. “Did you know 2019 (shift happens)” <https://www.youtube.com/watch?v=TwTS6Jy3lI8>. Вот как выглядит, например, картина, “нарисованная” моделью машинного обучения ruDALL-E, нейронной сетью для генерации картинок (<https://developers.sber.ru/portal/news/rudall-e-03-11-2021>), по запросу “берег моря с парусником”:



<https://rudalle.ru/demo> (<https://creativecommons.org/licenses/by/4.0/deed.ru>)

Программирование и программная инженерия, разработка программного обеспечения, создание сложных комплексных информационных систем - без этого почти невозможно представить современное предприятие. Задачи и методы в разработке программных систем, постоянно развиваются.

Основная задача в проектах создания программных систем (как, впрочем, и в любом проекте) это скорейшая реализация проекта с соблюдением имеющихся ограничений на ресурсы.

Как оказалось, в разработке программного обеспечения есть особенности, которые сильно отличают такие проекты от проектов в промышленности или на производстве. *Например, если увеличить команду разработчиков вдвое, то это практически никогда не приведет к сокращению сроков, а даже часто наоборот - увеличит сроки.* Причина в том, что новую команду надо вводить в курс дела, обучать, настраивать среду для работы. Появляются дополнительные накладные расходы, связанные с ошибками новых участников. Традиционная шутка программистов: “один программист будет делать работу один месяц, а два программиста будут делать ту же работу два месяца”.

Необходимость ускорения вывода продукта на рынок привело к созданию специальных инструментов и методов разработки программного обеспечения:

- интегрированная среда разработки, Integrated Development Environment, IDE,
- стандартные библиотеки, реализующие типовые функции,
- микросервисная архитектура и типовые интерфейсы взаимодействия,
- использование продуктов с открытым программным кодом, которое разрабатывает большое сообщество,
- создание шаблонов (паттернов) для проектирования и разработки систем,
- специальные организационные методы для командной разработки (agile), предполагающие более ранний показ “незрелого” продукта заказчику (а иногда и запуск в промышленную среду) для получения более быстрой обратной связи и ее учет в следующих итерациях процесса разработки.

Эти и многие другие нововведения существенно ускорили вывод продукта разработки на рынок с сохранением приемлемого качества, однако также добавили накладные расходы проекта. Например,

1. скорость выхода продукта сопряжена с появлением так называемого “технического долга” — перечня функций, которые необходимо разработать для нормальной эксплуатации продукта, после его вывода в эксплуатацию,
2. использование готовых библиотек, веб-серверов, систем управления очередями, баз данных существенно ускоряет процесс разработки, однако создает проблему совместимости компонентов между собой,
3. микросервисная архитектура предполагает полную независимость отдельных компонентов общего решения и их разработчиков, вследствие этого “улучшения”, введенные разработчиками одного компонента, могут стать катастрофическими для всей системы целиком, если другие компоненты окажутся несовместимыми с новой версией.

Подробнее об этом можно прочитать в книге Ф.Брукса “Мифический человеко-месяц или Как создаются программные системы”, которая была написана в прошлом веке, но остается актуальной и в настоящий момент.



<https://www.ozon.ru/product/mificheskiy-cheloveko-mesyats-ili-kak-sozdayutsya-programmnye-sistemy-83760/?sh=ycWDzgAAAA>

Из всего этого следует, что любой улучшающий подход или инструмент необходимо применять аккуратно, постоянно анализируя его влияние на общий процесс. Мониторинг и автоматизация различных этапов разработки как раз и предназначены для того, чтобы избежать описанных выше проблем. Вот методы, которые помогают в борьбе с хаосом в разработке программного обеспечения:

- контроль изменений в программном обеспечении на уровне изменений кода,
- контроль версий (версионирование), управление разными ветками программного обеспечения в разработке,
- автоматизация сборки всей разрабатываемой системы из отдельных модулей, над которыми работают разные команды,
- автоматизация тестирования,
- автоматизация установки разработанной системы на оборудовании для промышленной эксплуатации, включая автоматическую настройку инфраструктуры для развертывания системы,
- непрерывный мониторинг качества на всех этапах.

Создание и использование этих методов в процессе разработки программного обеспечения и привело к появлению методологии DevOps, являющейся связующим элементом между разработкой (Development, Dev) и эксплуатацией (Operations, Ops).

Все вышеизложенное относится к любым проектам, в которых создается программный код, и, в том числе, к проектам машинного обучения. Работа с большими данными и машинное обучение (Machine Learning, ML) сравнительно недавно стали популярным инструментом

в программных системах для решения производственных задач. Причины, по которым методы, разработанные еще во второй половине XX века, начали внедряться в практику сейчас, следующие:

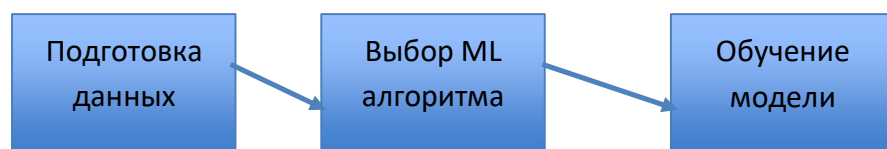
- увеличение производительности вычислительных систем,
- появление надежных и высокопроизводительных технологий для сбора, передачи, обработки и хранения данных,
- удешевление ресурсов для хранения данных и вычислений,
- повышение сложности информационных систем и их взаимодействия между собой, требующее автоматизации и использования интеллектуальных алгоритмов.

Действующий закон Мура косвенно описывает процесс постоянного уменьшения стоимости вычислительного оборудования и увеличения скорости вычислений. *Например, стоимость расшифровки генома человека (сложная вычислительная задача, позволяющая выявить риск наследственных заболеваний на основе генетики), снизилась на порядки за несколько десятков лет*
https://vademec.ru/news/2015/10/02/stoimost_rasshifrovki_genoma_snizilas_do_1_000.

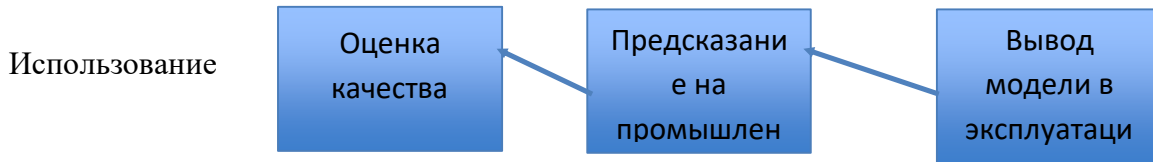
Появившиеся вычислительные возможности привели к взрывному (точнее, экспоненциальному) росту использования различных моделей машинного обучения, в том числе и глубокого обучения (Deep Learning, DL) на основе нейронных сетей. Разработанные алгоритмы машинного обучения и их использование в программных системах дали замечательные практические результаты, которые ранее невозможно было получить: алгоритмы начали распознавать изображения лучше человека, научились играть в шахматы и го, начали управлять производственными процессами и многое другое. Однако у решений на основе машинного обучения и больших данных есть и обратная сторона. Оказывается, что новые технологии и алгоритмы привели к новым проблемам, которые необходимо учитывать и решать при реализации проекта. Это связано с тем, что в машинном обучении есть отличия от обычных проектов разработки программного обеспечения, а именно:

- наборы данных (датасеты) оказывают сильное влияние на качество работы всей системы, при этом имеют свойство существенно меняться и на практике часто содержат ошибки,
- в проекте невозможно создать идеальное решение для продолжительного использования, практически сразу после вывода в продакшн модели машинного обучения начинают постепенно деградировать,
- модели машинного обучения очень чувствительны к окружению, в котором работают, небольшие различия в окружении для обучения и тестирования модели и в продакшн могут привести к кардинальным различиям в результатах работы модели,
- экспериментальный характер системы ведет к необходимости постоянного дообучения модели на основе новых данных или при изменении окружения.

Чтобы понять с какими сложностями может столкнуться специалист MLOps давайте посмотрим на этапы типового проекта машинного обучения, которые схематично приведены на рисунке ниже



Обучение



На этой схеме этапы проекта приведены укрупненно, более детально они будут описаны в последующих модулях. Но уже на этой схеме можно сформулировать и описать те дополнительные сложности, с которыми придется столкнуться при реализации проекта ML:

Этап	Задачи этапа	Возможные проблемы
Подготовка данных	Получение и анализ данных, устранение ошибок. Передача и хранение данных. Организация доступа к данным. Обезличивание данных. Обогащение данных. Формирование обучающего и проверочного датасетов.	Много разрозненных датасетов, на которых модель может показывать разные результаты. Изменение структуры и характеристик данных со временем.
Выбор ML алгоритма, решающего задачу	Изучение различных алгоритмов машинного обучения, поиск эффективного.	Много разных алгоритмов, у каждого свои параметры. Необходимо учитывать ограничения: аппаратные ресурсы, требования к качеству.
Обучение модели	Модель ML обучается с использованием набора данных для обучения (train dataset, обучающий датасет).	Результаты обучения модели могут сильно отличаться при незначительном изменении в данных или программном окружении (набор и версии библиотек)
Внедрение модели в продуктивную среду	Разработанная модель выводится в промышленную эксплуатацию (как отдельный микросервис или как часть более крупной системы)	Обеспечение совместимости среды, в которой эксплуатируется модель, с моделью. Взаимодействие модели с другими компонентами.

<p>Предсказания на промышленных данных</p>	<p>В среде эксплуатации модель получает новые данные, в том числе и те, которые она ранее не видела.</p>	<p>Новые данные могут отличаться от данных, на которых обучали модель.</p> <p>Неустойчивая работа модели даже при незначительных изменениях характеристик среды выполнения программы или в данных.</p> <p>В крайних случаях это может привести к краху модели (проблема повторяемости результатов).</p>
<p>Анализ качества</p>	<p>Контроль качества обучения. Принятие решения о переобучении модели.</p>	<p>Методы машинного обучения представляют собой «черный ящик», логику работы которого сложно объяснить (проблема интерпретируемости результатов).</p> <p>С течением времени данные изменяются, что приводит к ухудшению качества работы модели.</p>

В совокупности эти проблемы приводят к тому, что даже у самой хорошей идеи, реализованной на уровне гипотезы, возникают сложности в практической эксплуатации. Статистика проектов, использующих большие данные и машинное обучение, говорит о том, что большинство (более 80%) проектов не попадают в промышленную эксплуатацию (production).



Картинка из курса LEAN_DS в сообществе ODS.

Проблемы весьма серьезные, но если бы это останавливало команды разработчиков проектов машинного обучения, то мы никогда бы не увидели замечательные проекты с использованием технологий распознавания изображений, переводчики текстов, интеллектуальные транспортные системы и беспилотные автомобили. Зная о потенциальных опасностях, можно предпринять меры по их недопущению и одним из способов сделать это является правильная организация инфраструктуры и процесса работы над проектом машинного обучения. Собственно, этими задачами и занимается специалист MLOps. Итак, мы пришли к пониманию, что роль MLOps инженера крайне важна для общего успеха проекта.

Как вы увидите в последующих модулях курса, от MLOps инженера требуется большая универсальность в знаниях и навыках, но перечень необходимых знаний может сильно зависеть от проекта и команды. В ходе работы MLOps инженеру могут понадобиться навыки:

- настройки и администрирования аппаратного обеспечения,
- программирования python (и некоторых других языков), понимание принципов работы окружения этих языков (компиляторов, интерпретаторов, служебных библиотек, установщиков и пр.)
- администрирования операционных систем linux, написания bash скриптов,
- создания средств виртуализации: виртуальные машины, контейнеры docker/kubernetes,
- настройка баз данных и написание SQL запросов,
- понимание механизмов работы и форматов API, http, JSON,
- знание архитектуры и протоколов сетей передачи данных,
- понимание методов машинного обучения и работы с большими данными,

и многое другое...

Кроме технологических знаний (так называемые “hard-skills” навыки) от MLOps инженера требуются и “soft-skills” навыки, например,

- понимание agile методов организации работы команд разработки (scrum, kanban),
- умение коммуницировать и доносить свою позицию до других участников команды,
- активность в поиске и устранении “слабых мест” в проекте,
- желание изучать новые технологии, в том числе относящиеся к смежным подразделениям.

В крупнейшем русскоязычном сообществе Open Data Science (<https://ods.ai>) есть отдельная ветка LeanDS (<https://ods.ai/projects/leands>), в которой изучаются вопросы применения agile-методов для проектов машинного обучения.

Для быстрого освоения материала и выполнения практических заданий очень важно иметь эти требуемые навыки, поэтому слушателям, которые не чувствуют себя уверенно в данных темах, рекомендуется изучить их подробнее самостоятельно.

Итак, в данном юните вы узнали, что:

- методологии DevOps/MLOps призваны ускорить процесс разработки и повысить качество проектов,
- на разных этапах проекта машинного обучения возникают различные проблемы, в решении которых помогает MLOps, например

- различные результаты работы системы в тестовом и промышленно окружении
- большое количество модулей и библиотек и их несовместимость между собой
- необходимость выполнять итерации жизненного цикла проекта машинного обучения чаще, чем в обычных проектах разработки
- и многие другие
- от инженера MLOps требуется большой кругозор и знание проекта, причем не существует исчерпывающего универсального списка, перечень знаний и навыков существенно зависит от команды и проекта, что мы увидим в дальнейшем.

Теперь вы знаете почему появились методологии DevOps и ее адаптация для проектов машинного обучения MLOps. Конечно, для решения задач в MLOps уже разработаны различные эффективные инструменты, библиотеки, скрипты, организационные подходы. Их перечень очень большой, большинство имеют статус свободно распространяемого программного обеспечения, выбор того или иного инструмента зависит от множества факторов (тип проекта, количество человек в команде, ограничения по ресурсам, опыт MLOps инженера). По сути, сложность инфраструктуры большого проекта машинного обучения сопоставима со сложностью комплексной технологической системы, например автомобиля. Итак, самое время заглянуть “под капот” MLOps.

Проверим:

1. Что содержит аббревиатура MLOps
 - a. **Machine Learning + Operations**
 - b. Machine Learning + Opinions
 - c. Maximum Learning + Operations
 - d. Machine Lean + Operations
2. Отметьте причины, по которым в проектах разработки программного обеспечения добавились дополнительные риски, касающиеся машинного обучения
 - e. **чувствительность модели машинного обучения к любому шуму в данных**
 - f. малое количество статей и готовых решений
 - g. **невозможность долговременного сохранения окружения, в котором эксплуатируется модель машинного обучения**
 - h. использование сложных математических алгоритмов
3. Расположите этапы проекта машинного обучения в правильном хронологическом порядке
 - i. сбор данных
 - j. преобразование данных
 - k. тренировка модели машинного обучения
 - l. вывод в эксплуатацию
 - m. предсказание на промышленных данных

п. оценка качества работы

4. Поставьте в соответствие навыки, которыми должен обладать MLOps, и причины важности обладания этими навыками

Навык	Причина
Программирование на C++	Основной код разрабатываемой системы написан на этом языке
Администрирование linux	Большинство инструментов, которыми пользуется команда, реализованы этими средствами
SQL запросы	Для организации сбора данных
bash, python	Для разработки скриптов автоматизации процедур
Машинное обучение, метрики	Для своевременного отслеживания ухудшения качества работы системы

Модуль 1. Юнит 2: Из чего состоит MLOps?

В этом юните:

Слушатели узнают о задачах и инструментах MLOps более подробно.

Содержание юнита:

Идеология DevOps, на основе которой появился MLOps, предполагает непрерывность всех этапов процесса разработки от планирования системы до ее эксплуатации. И тут возникает самое интересное, современные так называемые гибкие (agile) подходы к управлению проектами разработки программного обеспечения предполагают не только то, что после эксплуатации сразу может следовать новое планирование, но и то, что таких повторений (итераций) может быть много, чем больше, тем лучше! Получается такой магический символ бесконечности (перевернутая “восьмерка” в математике является символом “бесконечности”, что символизирует постоянное повторение итераций цикла проекта), символизирующий постоянное улучшение процесса и повторение шагов: планирование (plan) - разработка кода (code) - сборка системы (build) - тестирование (test) - выпуск релиза (release) - развертывание (deploy) - эксплуатация (operate) - мониторинг (monitor)

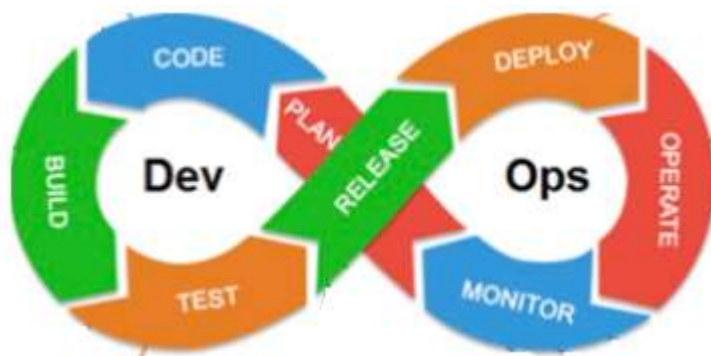


Схема жизненного цикла agile-проекта разработки программного обеспечения

При этом принимается тот факт, что на первых итерациях качество разработанного решения может быть явно недостаточным, однако **польза такого подхода состоит в том, что команда выводит решение как можно раньше и получает обратную связь от заказчиков, других подразделений, которую можно проанализировать и учесть в следующей итерации.** Продолжительность такой итерации и их число сильно зависят от конкретного проекта и отрасли, возможности испытывать предварительные версии на системе или заказчике, наличия ресурсов. Однако ясно, что раз эти действия являются повторяющимися и от скорости их выполнения зависит успех всего проекта, то это совершенно необходимо автоматизировать.

Рассмотрим это на примере: молодая команда разработчиков web-сайтов испытывает большие сложности с получением точного технического задания от заказчика, маркетолога производственного предприятия, который хочет, чтобы “было все красиво”, но совершенно не готов обсуждать “верстку CSS”, используемые стили и скорость работы отдельных функций сайта. Отчаявшись, ребята сделали прототип сайта как посчитали более правильным, максимально учитывая те крохи информации, которую они получили от маркетолога. На быстро последовавшем показе маркетолог сказал: “Ну, наконец то до вас дошло что я хотел, только надо еще вот это и вот это”, и в

последующих итерациях уже указанные конкретные проблемы были устранены. Для лучшего понимания этой ситуации наберите в youtube “семь перпендикулярных красных линий”.

И, как вы уже наверное догадываетесь, для проекта машинного обучения эта ситуация становится еще более сложной. Проект машинного обучения содержит стандартные этапы, которые представлены на следующей диаграмме



Этапы проекта машинного обучения

На каждом из этапов существуют отдельные задачи, результатом каждого этапа является так называемый **артефакт**, который передается для обработки на следующий уровень. Рассмотрим подробнее эти этапы, попутно рассуждая какие артефакты возникают при выполнении этих этапов и какие задачи при этом возникают для специалиста MLOps.

5. Сбор данных

Сбор данных может осуществляться из различных разнородных источников с использованием различных средств передачи информации, например, служебных технологических протоколов (OPC, modbus), стандартных сетевых протоколов и средств (SNMP, syslog), интерфейсов баз данных (SQL, JSON). Для получения данных может потребоваться авторизация. Устройств для опроса может быть огромное количество.

Задачи MLOps:

- автоматизировать сбор информации, организовать опросы устройству по списку и расписанию, с учетом их интерфейсов взаимодействия,
- организовать мониторинг доступности источников данных и каналов передачи информации, формирование сообщений об ошибке в случае проблем с получением данных,
- создать систему для надежного хранения полученных данных, включая, при необходимости, резервирование и авторизованный доступ к данным.

Артефакты этапа:

- информация, собранная из источников данных и сохраненная в структурированном виде
- данные системы мониторинга

6. Предобработка данных

В данных почти всегда содержатся ошибки. Это может быть связано с ошибками передачи информации, выходом из строя источника данных, аномальным поведением окружающей среды или ошибкой, связанной с человеческим фактором. Ошибки в данных могут дорого стоить на следующих этапах проекта, например при эксплуатации модели (иногда даже применяется термин “отравление модели”), а использование плохих датасетов на этапе тренировки модели вообще может сделать невозможным получение качественно

работающей системы. Поэтому обязательным этапом работы с данными перед передачей их в работу модели является обработка данных: очистка от аномальных значений, нормализация, удаление шумов, заполнение пропусков. Это рутинные типовые операции, которые целесообразно автоматизировать. Кроме того, существует предобработка, позволяющая существенно улучшить данные для их использования в проекте. Например, обогащение имеющихся данными с использованием информации из других источников. На этом этапе уже есть место эксперименту, поскольку только экспериментально можно уточнить какой набор данных даст наилучший результат. В здесь возникает необходимость работы с большим количеством различных датасетов, а, следовательно, и контроля их версионности.

Задачи MLOps:

- автоматизация рутинных процедур обработки сырых данных: выгрузка, преобразование типов, устранение ошибок или пропусков, обработка выбросов
- создание и использование системы версионирования различных конвейеров предобработки данных
- интеграция решений MLOps этого этапа со следующими этапами (корректная передача артефактов этого этапа на следующие уровни).

Артефакты этапа:

- обработанные наборы данных (датасеты) для использования на последующих этапах
- сохраненные конвейеры (пайплайны) обработки данных

7. Разведывательный анализ данных

На этом этапе в данных ищут скрытые закономерности, строят предварительные экспериментальные модели машинного обучения, проводят вычислительные эксперименты и формируют гипотезы для последующей проверки. Этот этап связан с большим количеством преобразований данных и испытаний различных моделей с разным набором параметров. Поэтому, кроме контроля версий датасетов на этом этапе необходимо контролировать версии кода для проверки гипотез (чаще всего это jupyter-ноутбуки).

Задачи MLOps:

- организация рабочего окружения для исследователей данных: автоматическое создание среды выполнения эксперимента с нужными версиями библиотек и настроек среды, выделение вычислительных мощностей для эксперимента, сохранение и версионирование результатов эксперимента
- автоматизация вычислительных экспериментов
- автоматизация подбора параметров моделей машинного обучения и сохранения наилучших результатов
- мониторинг экспериментов и обучения моделей машинного обучения

Артефакты этапа:

- программный код (например, jupyter-ноутбуки), являющийся результатом выполнения эксперимента
- сохранённые модели (например, гиперпараметры, веса нейронной сети), полученные в результате предварительных экспериментов и являющиеся

ориентировочными (так называемыми бэйзлайнами) для улучшения в последующих экспериментах

- данные мониторинга процесса обучения

8. Выделение важной информации

На этом этапе выделяют (или формируют новые) признаки для организации обучения модели машинного обучения. Возникает необходимость в хранилище признаков (так называемый feature store).

Задачи MLOps:

- автоматизация процесса подбора эффективных признаков (анализ существующих и подбор новых)
- сохранение и версионирование набора признаков

Артефакты этапа:

- сохраненный набор признаков и значения показателей качества работы системы, связанные с этими наборами

9. Построение и обучение модели машинного обучения

На этапе тренировки модель машинного обучения обучается с использованием тренировочного набора данных (train dataset), а после обучения модель проверяется на новых данных (test dataset). Предполагается, что у инженера машинного обучения (ML инженер) есть набор данных, описывающий необходимый процесс, и из этого набора выделяются тренировочный (train) и тестовый (test) датасеты. Важно, чтобы эти датасеты не пересекались, соотношение при разделении обычно 70% данных для тренировочной выборки и 30% данных для тестовой. Такой подход в машинном обучении называется кросс-валидация. Еще одним подходом является разделение имеющихся данных на три датасета: тренировочный (train), тестовый (test) и валидационный (validation). При этом тестовый и валидационный наборы данных отдельно используются для выбора модели и подбора их гиперпараметров. Обычно соотношение в объеме данных в этом случае составляет 60% для тренировочной выборки, 20% для тестовой и 20% для валидационной выборок. Для разных типов задач и разных моделей используются различные подходы для формирования датасетов, их подбор также может быть частью эксперимента, поэтому данный процесс необходимо автоматизировать и уметь запоминать различные варианты датасетов, на которых обучалась и проверялась модель, то есть необходимо версионирование.

Часто алгоритмы машинного обучения представляют собой «черный ящик», логику работы которого сложно объяснить. С практической точки зрения этого бывает достаточно, однако такая непредсказуемость может вести к неустойчивой работе при отклонениях в окружении работы модели. Поэтому необходимо контролировать версии как самих моделей, так и наборов гиперпараметров и окружения (библиотеки, переменные), в которых работают модели. Важно на этом этапе учитывать, что успешные модели на следующих этапах перейдут в промышленное окружение, поэтому надо создавать обучающее окружение как можно более похожее на промышленную среду. Это касается использования библиотек, ограничения на производительность процессора и расход оперативной памяти, требований к производительности модели.

Задачи MLOps:

- автоматизация вычислительных экспериментов и обучения модели машинного обучения
- автоматизация подбора обучающего и тренировочного набора данных, их версионирование и привязку к обученной модели и окружению, в котором обучалась модель
- автоматизация развертывания среды обучения и испытания модели

Артефакты этапа:

- сохранённые модели (например, гиперпараметры, веса нейронной сети), полученные в результате предварительных экспериментов и являющиеся ориентировочными (так называемыми бэйзлайнами) для улучшения в последующих экспериментах
- параметры работы модели (требования к памяти, CPU, планируемая скорость работы при инференсе)

10. Развертывание модели машинного обучения

При развертывании важно как можно ближе сохранить окружение для модели машинного обучения. Это достигается за счет использования технологий виртуализации (виртуальные машины, контейнеры) и автоматической настройки окружения (Ansible).

Задачи MLOps:

- автоматически создать окружение для работы модели, включая согласованные версии операционной системы, системных утилит и библиотек, переменных окружения,
- выполнить сборку и загрузку проекта в рабочее окружение,
- произвести автоматические настройки оборудования и программного обеспечения для запуска модели машинного обучения,
- обеспечить необходимый мониторинг работы системы.

Артефакты этапа:

- работающая модель машинного обучения в промышленной среде

11. Эксплуатация

На этом этапе построенная модель выводится в промышленное окружение и начинает работать на реальных данных. На этом этапе часто возникают проблемы, связанные с качеством данных на реальных объектах, поэтому в эксплуатации важны функции мониторинга работы системы в целом и качества работы модели машинного обучения. Наиболее часто для целей мониторинга применяются средства визуализации и контроля.

Задачи MLOps:

- автоматизированный мониторинг работы системы, реагирование на инциденты,
- обеспечение автоматизированного контроля данных, на которых работает модель машинного обучения.

Артефакты этапа:

- результаты эксплуатации модели машинного обучения в промышленной среде: данные мониторинга, сообщения об ошибках, информация об использовании ресурсов (оперативная память, CPU), данные о скорости выполнения операций

12. Контроль и оценка качества

Постоянный автоматизированный контроль качества и, при необходимости, переобучение модели являются обязательными в эксплуатации любой модели машинного обучения. Практически невозможно создать универсальную модель для долгого срока эксплуатации, так как промышленные данные и окружение имеют свойство изменяться, что приводит к ухудшению качества работы модели.

Задачи MLOps:

- постоянный автоматизированный анализ качества работы модели машинного обучения,
- анализ инцидентов, обнаружение причин,
- рекомендации по дальнейшей работе с моделью машинного обучения (дообучение на новых данных, создание другой модели).

Артефакты этапа:

- данные о соответствии результатов работы модели машинного обучения запланированным метрикам качества
- наборы данных (датасеты) для дообучения или переобучения модели машинного обучения.

Итак, мы теперь знаем какие задачи MLOps возникают на каждом из этапов проекта машинного обучения. Рассмотрим с практической точки зрения самые важные из них.

Задача 1: обеспечение повторяемости результатов проекта.

Данные в обучающем датасете и в реальных данных в эксплуатации модели могут отличаться. Кроме того, могут отличаться среды, в которых работают модели. Например, исследователи могут строить модели в операционной системе Linux Debian, а на объекте может быть установлена система Kali Linux. Большая практическая проблема, касающаяся всех участников проекта ML, это обеспечение повторяемости результатов работы модели в среде разработки и производственной среде. Поиск решения проблемы, связанной с использованием разных версий библиотек, может занимать большое количество времени.

Для этого необходимо контролировать множество параметров:

- наборы данных (количество признаков, качество данных, однородность данных)
- настройки окружения (характеристики аппаратного обеспечения, скорость передачи данных)
- гиперпараметры модели
- используемые служебные библиотеки

Задача 2: автоматизация.

Представьте, что ML инженер наконец-то обучил свою модель на мощных графических процессорах, сохранил веса этой модели и готов теперь применить полученную модель для анализа производственных данных. Не тут-то было! Его супермодель является лишь частью большого проекта и для запуска в промышленную эксплуатацию надо собрать всю систему. И протестировать. И обеспечить наличие всех необходимых библиотек, настроек. Конечно, это все можно делать руками и глазами, но можете себе представить насколько это эффективно, когда приходится одни и те же действия повторять бесчисленное количество

раз. Тут недалеко и от ошибки, связанной с “человеческим фактором”. И на помощь приходит MLOps, решающий задачу автоматизации. С использованием специальных утилит автоматизации (git, Jenkins и т.п.) или даже обычных скриптов (bash, python) можно автоматизировать рутинные операции по сборке всего пакета программного обеспечения целиком.

Задача 3: создание правильного рабочего окружения.

Развертывание рабочего окружения может стать головной болью для инженера MLOps, поскольку включает в себя множество узлов со специальными настройками. Поднимать сервера с нужной операционной системой, устанавливать библиотеки и делать настройки среды, выбирать правильные и совместимые версии для используемого программного обеспечения (базы данных, технологии контейнеризации, компиляторы и интерпретаторы) - выполнение этих задач может растянуться надолго, а проект требует быстрого вывода в продакшн. Поэтому на данном этапе активно используют инструменты для автоматизации рутинных повторяющихся действий:

- docker, docker-compose или kubernetes для виртуализации
- chef, ansible, terraform для создания необходимых конфигураций

В этом юните вы узнали:

- из каких этапов состоит процесс разработки программного обеспечения
- во чем отличия проекта машинного обучения от обычных проектов разработки программного обеспечения
- что является результатом работы на каждом этапе
- какие задачи возникают на различных этапах и как эти задачи решает MLOps:
 - автоматизация на всех этапах
 - контроль версий датасетов, моделей, окружений

Теперь изучим этот вопрос в привязке к жизненному циклу.

Проверим:

1. Что символизирует символ “бесконечность” на рисунке, описывающем процесс DevOps
 - a. то, что проблем в DevOps/MLOps бесконечно много
 - b. то, что итерации в процессе DevOps/MLOps должны быстро и много повторяться**
 - c. то, что в DevOps/MLOps используется бесконечно много разных инструментов
 - d. то, что расходы на DevOps/MLOps неограничены
2. Что такое артефакт этапа?
 - a. документ с перечнем ошибок
 - b. список задач этапа
 - c. результат выполнения этапа, который передается на следующий этап**
 - d. недокументированные возможности работы системы

3. Что можно автоматизировать в проекте машинного обучения?
- a. подготовку технического задания
 - b. сбор данных**
 - c. обучение модели машинного обучения**
 - d. вывод системы в продакшн**
4. Какие проблемы может решить MLOps?
- a. человеческие ошибки при повторении большого количества одинаковых рутинных операций**
 - b. высокая стоимость вычислительных ресурсов
 - c. ошибки, связанные с несовместимостью различных версий библиотек, использованных в решении**
 - d. некорректное техническое задание

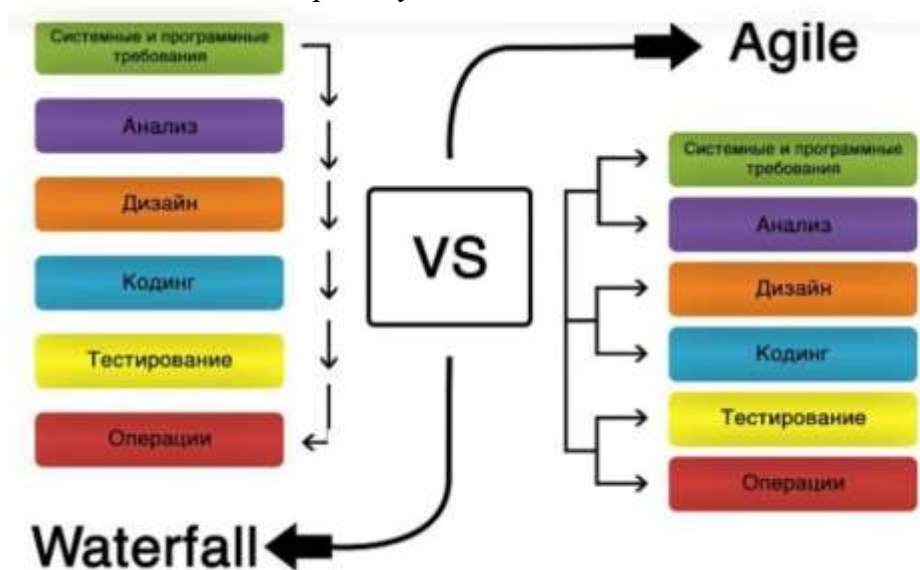
Модуль 1. Юнит 3: Управление проектом и MLOps.

Об этом юните:

Существуют различные методологии управления проектом, но для повышения эффективности проекта машинного обучения ключевыми факторами являются эффективная автоматизация и обеспечение непрерывности процесса. В этом юните мы узнаем о том, как MLOps помогает сделать проект эффективнее.

Содержание юнита:

Любой проект обладает своим жизненным циклом. **Жизненные циклы различаются для разных типов проектов, как по перечню этапов и задач, так и по их продолжительности.** В проектах типа “водяной мельницы” каждый этап может продолжаться очень долго, необходимым условием для перехода на следующий этап является выполнение всех задач предыдущего этапа.



Сравнение методов управления проектами, <https://worksection.com/blog/waterfall-vs-agile.html>.

В противоположность этому в гибких (agile) подходах акцент сделан на быстрое создание прототипа и большое количество повторений всех циклов проекта. Проекты машинного обучения практически всегда управляются с использованием agile-методологии, и вызвано это не столько желанием руководителя проекта поскорее показать первую версию проекта заказчику, сколько тем, что главным свойством объектов в проекте (датасеты, модели, конвейеры, окружение) является изменчивость. Вчерашние данные могут существенно отличаться от сегодняшних, разработчики программного обеспечения могут внести изменения в библиотеку, а заказчик может поменять настройки окружения. Поэтому команда проекта машинного обучения должна быть всегда готова к изменениям, постоянные и частые итерации проекта должны быть девизом участников команды.

В методологии управления проектами большое внимание уделяется соблюдению параметров проекта, которыми являются: время, ресурсы, ожидаемые функции (качество). Редкому руководителю проекта удастся сдать проект в оговоренное время, не превысив

имеющиеся ресурсы, при этом реализовать все, что от проекта ожидалось. Практически всегда один из параметров страдает.



Параметры проекта

Ограничения на ресурсы проекта могут быть следующие:

- размер команды,
- используемый технологический стек (в т.ч. возможность использования открытого ПО),
- возможность изменений технического задания при работе над проектом.

Искусство проектного управления состоит в том, чтобы провести проект по всем этапам с соблюдением всех существующих ограничений. Наиболее популярными agile-методами являются Scrum и Kanban. Эти методы позволили существенно ускорить выход продукта на рынок, повысить качество. Основная идея этих методов состоит в том, что команда работает над проектом короткими итерациями (спринтами), в ходе которых осуществляется сборка всего проекта целиком и показ заказчику, после которого уточняется содержание задач в “техническом долге” и формируется план работы на следующий спринт. Такой подход требует повторения одинаковых рутинных операций, которые эффективно объединить в группы и разработать для каждой группы свои правила и методы, в том числе автоматизацию процессов.

В любом проекте разработки программного обеспечения контролируемая автоматизация связывается с обеспечением так называемой **непрерывности** процесса. **Под непрерывностью понимается управляемый и контролируемый автоматически выполняемый набор стандартных операций, которые ведут к выполнению какой-либо крупной задачи в проекте.** Обычно в качестве таких операций в проекте машинного обучения рассматриваются:

- непрерывная сборка (Continuous Integration, CI),
 - проверка кода,

- выполнение задач,
- валидация кода,
- unit тестирование,
- объединение кода,
- непрерывное развертывание и доставка (Continuous Delivery, CD),
 - сборка кода в единую систему,
 - тестирование,
 - выпуск релиза,
- непрерывное обучение (Continuous Training, CT).
 - мониторинг работы модели,
 - измерения,
 - дополнительное обучение, тюнинг,
 - эксплуатация.

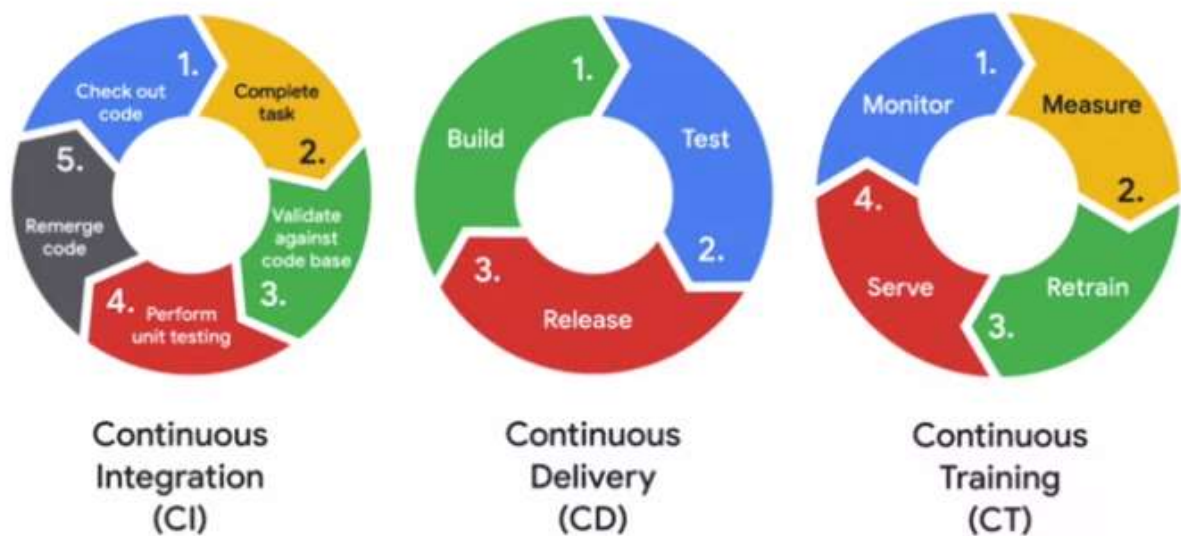


Рисунок “Автоматизация для разных операций проекта машинного обучения”

Практически все задачи на всех этапах в проекте машинного обучения можно выполнять вручную или автоматически. Понятно, что автоматизация позволяет быстрее и точнее решать задачи, без автоматизации немыслима реализация большого проекта, в котором большое количество участников, много отдельных модулей и систем, большой объем кода, тестов.

Важным параметром проекта является “технический долг”: накопленные задачи и замечания, появляющиеся после показов заказчику и эксплуатации проекта. Технический долг присутствует практически в любом проекте, управляемом по agile, поскольку идеология agile-методов предполагает как можно более быстрый вывод прототипа проекта в эксплуатацию и получение обратной, как правило негативной, связи от заказчика. Техническим долгом необходимо эффективно управлять, определяя наиболее важные для устранения замечания, устраняя причины появления замечаний, улучшая качество кода и архитектуры.

Пример: уже известная нам из предыдущих юнитов команда амбициозных web-разработчиков решила реализовать на свой страх и риск версию сайта без уточнения всех деталей у постановщика. Автоматически после первой демонстрации у команды образовался “технический долг”: функции и элементы сайта, которые необходимо доделать.

Технический долг в проектах машинного обучения имеет обыкновение увеличиваться быстрее, чем в классических проектах создания программного обеспечения, так как его могут создавать:

- многофункциональная команда, большое количество ролей и специализаций,
- экспериментальный характер процесса (на начальном этапе неясно будет ли та или иная гипотеза эффективной),
- сложность тестирования (отсутствие обучающих данных),
- сложность развертывания (множество разных библиотек),
- разрушение модели в эксплуатации (со временем данные изменяются, необходимо переучивать модель).

Наиболее системно эти факты изложены в известной статье Google “Hidden technical debts in machine learning systems”.



Если коротко сформулировать основную идею Google, то она выглядит так: “Разработка и развертывание систем машинного обучения происходит относительно быстро и дешево, но поддерживать их в течение долгого времени дорого и сложно”.

Идеология MLOps появилась для того, чтобы ускорять и улучшать качество в проекте машинного обучения, используя автоматизацию процессов, контроль версий компонентов решения, тестирование, сборку и вывод решения в производственную среду. Методы и прикладные инструменты MLOps, которые мы будем рассматривать в этом учебном курсе, как раз направлены на решение проблемы разрастания “технического долга”. Жизненный цикл проектов машинного обучения состоит из типовых этапов, изученных нами в предыдущем юните. На разных этапах используются разные методики и инструменты MLOps для борьбы с “техническим долгом”.

На этапах сбора и предобработки данных накапливается большое количество сырых и обработанных данных, которые необходимо сохранять, управлять изменениями в них,

обеспечивать доступ к нужным данным. На этом этапе применяются следующие методики и инструменты MLOps:

- контроль и версионирование данных (dvc),
- автоматизация обработки данных, конвейеры данных (python, sklearn, Apache)
- скрипты автоматизации и загрузки данных (python, Apache).

Разведывательный анализ данных и выделение важной информации, в том числе важнейших признаков и взаимосвязей в данных, предполагает активное исследование, эксперименты с данными, создание новых признаков. На этом этапе необходимо контролировать версионность экспериментальных данных, создаваемых конвейеров и пробных моделей машинного обучения.

Методики и инструменты MLOps:

- автоматизация процесса подбора эффективных признаков, анализ существующих и подбор новых, хранилище признаков (streamSQL)
- сохранение и версионирование набора моделей (git),
- сохранение и версионирование датасетов и пайплайнов (dvc)

На этапе **построения и обучения модели машинного обучения** проводится много исследований и проверок гипотез, поэтому часто возникают ситуации, когда модель начинает показывать результаты хуже, поэтому требуется вернуться к предыдущей версии.

Методики и инструменты MLOps:

- создание виртуальной среды выполнения модели (venv, virtualenv, poetry, docker)
- версионирование и контроль моделей машинного обучения (git)
- контроль процесса обучения модели (tensorboard)

При **развертывании модели машинного обучения** важно быстро создавать правильное окружение для работы модели, поэтому на этом этапе используются различные технологии виртуализации. Перед развертыванием как правила проводится сборка всего проекта целиком, с тестированием отдельных компонентов и всего решения.

Методики и инструменты MLOps:

- автоматическая сборка решения (Jenkins),
- тестирование (pytest),
- создание контейнеров микросервисов для организации работы модели (docker, docker-compose, kubernetes),
- настройка среды выполнения программы (Ansible)

При **эксплуатации** важны качественный мониторинг всех параметров работы системы. Важной частью процесса эксплуатации является этап **контроля и оценки качества работы** модели.

Методики и инструменты MLOps:

- мониторинг, реагирование на инциденты, визуализация (grafana, Kibana),
- обеспечение автоматизированного контроля данных, на которых работает модель машинного обучения.

Отдельно необходимо отметить важность визуализации на каждом из этапов проекта. Хорошая визуализация полезна на любом этапе жизненного цикла проекта.

- При анализе данных с помощью содержательной диаграммы можно легче найти характерные признаки в данных, например, наличие выбросов или статистические свойства.
- При обучении модели требуется постоянно анализировать графики качества работы модели, чтобы избежать переобучения.
- В эксплуатации от средств мониторинга зависит скорость реагирования на внештатные ситуации.
- Незаменимыми являются хорошие, содержательные, легко интерпретируемые графики при презентации проделанной работы руководству.

В этом юните мы узнали, что

- MLOps тесно связан с каждым этапом жизненного цикла модели машинного обучения
 - сбор, обработка и анализ данных
 - проверка гипотез, эксперименты и построение моделей
 - сборка и вывод в эксплуатацию проекта
 - эксплуатация и анализ качества
- для каждого этапа разработано множество инструментов, в том числе с открытым исходным кодом (open-source)
- визуализация очень важна на каждом из этапов

Теперь рассмотрим вопрос с точки зрения состава команды проекта, ролей и их функций.

Проверим:

1. что такое “технический долг”
 - a. задолженность исполнителя перед заказчиком
 - b. задолженность заказчика перед исполнителем
 - c. перечень функций, которые необходимо реализовать, чтобы получить приемлемое для заказчика качество продукта**
 - d. список ошибок, найденных при тестировании
2. что такое “версионирование”
 - a. контроль версий**
 - b. правила нумерации датасетов
 - c. правила нумерации всех объектов
 - d. формат номера
3. каких задач MLOps нет в обычном проекте разработки
 - a. версионирование датасетов**
 - b. версионирование пайплайнов**
 - c. управление изменениями в коде
 - d. настройка виртуального окружения
4. поставьте в соответствие этапам задачи, которые решаются с помощью визуализации

анализ данных	обнаружение ошибок в данных, выявление характерных статистических признаков
построение моделей	контроль переобучения
эксплуатация	контроль качества работы решения, обнаружение отклонений в работе
анализ результатов проекта	создание наглядных интерпретируемых графиков для презентации результатов заказчику

Модуль 1. Юнит 4: Команда ML проекта (роли)

В этом юните:

Мы поговорим о том какие существуют роли участников в проектах машинного обучения. Для каждой роли существует свой набор задач, и мы узнаем, как с этим связаны задачи MLOps.

Содержание юнита:

При реализации проектов машинного обучения часто используются стандартные подходы из классического проектного управления разработкой программного обеспечения, однако в проектах ML есть свои особенности

- бизнесовая составляющая: качество работы модели машинного обучения измеряется различными математическими метриками и большим искусством является превратить бизнес-требования и бизнес-метрики в задачи для команды проекта и технологические метрики,
- научная составляющая: важной частью проекта ML является работа с данными и исследовательский этап,
- инфраструктурная составляющая: при испытаниях и использовании модели на промышленных данных важно учитывать большое количество факторов, например, окружение модели, производительность, ожидаемую скорость работы, интеграцию с другими системами и т.п.

Особенности проектов ML находят отражение в определении необходимых ролей участников проекта и требований к ним. Этим ролей может быть много, в больших компаниях за каждую ролью может быть закреплен целый отдел, а, напротив, в небольших компаниях-стартапах один человек может выполнять задачи сразу нескольких ролей.



Рисунок “Различные роли участников проекта машинного обучения”.

Давайте разберемся с тем, что делают различные участники проекта, чтобы понять какие задачи MLOps от них могут появляться:

- Бизнес
 - Эксперт, разбирающийся в предметной области (финансы, медицина, государственное управление, образование и т.п.), обычно это представитель заказчика, однако для эффективной реализации проекта такой человек должен быть и в команде проекта. В сферу ответственности эксперта входят постановка задачи, уточнение бизнес-требований, консультирование по технологиям, итоговый вывод о соответствии созданного решения поставленной задаче и анализ эффективности (в том числе технико-экономический анализ).
 - Руководитель проекта: обеспечение выполнения проекта в срок с требуемым качеством в рамках установленного бюджета
 - Руководитель продукта: связь с рынком, CustDev
 - Бизнес-аналитик: постановка задачи разработчиком на основе полученных задач от заказчика
 - Заказчик (спонсор) проекта, может быть как внешний, так и внутренний: выделение ресурсов для проекта, приоритезация, политическая и организационная поддержка
- Работа с данными и моделями
 - Дата инженер, архитектор данных: сбор, передача, хранение данных, формирование всего конвейера работы с данными, понимание источников получения данных
 - Дата аналитик: разведочный анализ данных, поиск признаков, формирование гипотез, извлечение смысла из данных
 - Исследователь данных: интеллектуальный анализ данных, формулировка гипотез
 - ML инженер: организация работы инфраструктуры, контроль версий библиотек
 - ML исследователь: проверка гипотез, реализация прототипов, анализ SOTA
- Реализация проекта
 - Архитектор, разработчик: создание архитектуры решения, написание кода
 - Тестировщик: тестирование решения
 - DevOps/MLOps/DataOps: автоматизация задач, развертывание систем
 - Дизайнеры: создание графических интерфейсов, дашбордов
 - Технические писатели: написание документации
- Эксплуатация
 - специалисты технической поддержки: обеспечивают работоспособность решения, реагируют на инциденты.

Практически невозможно найти так называемого специалиста DataScience (на эту тему подробнее можно посмотреть выступление В.Бабушкина “Почему вы никогда не найдете датасайнтиста” <https://www.youtube.com/watch?v=Cs3ae65tmKA>). Главная причина состоит в том, что задач в проекте очень много и их перечень сильно зависит от типа проекта и команды. Как правило это означает, что в команде проекта машинного обучения много

ролей и все участники пользуются различными инструментами для решения конкретных задач.

Чтобы понять какие задачи возникают в MLOps давайте еще раз рассмотрим всю последовательность от постановки задачи до вывода модели в производственную среду и ее эксплуатации.



Рисунок «Последовательное выполнение задач проекта машинного обучения и некоторые инструменты для их решения»

Схема работы над проектом:

1. бизнес-аналитик совместно с заказчиком и экспертом описывает задачу, определяет требования к качеству, ограничения задачи

Задачи MLOps:

- участие в формировании и обсуждении технического задания с целью определить требования к системе, ограничения
- выбрать инструменты с учетом имеющихся ограничений на ресурсы
- определить целевую конфигурацию аппаратного обеспечения для
 - сбора и хранения данных и производных датасетов, признаков
 - проверки гипотез и хранения моделей
 - сборки и тестирования решения
 - развертывания решения у заказчика

На этом этапе никаких специальных инструментов MLOps специалист не использует, занимается выработкой требований и определением стека технологий для будущих задач.

2. дата аналитик и дата инженер исследуют источники получения данных, проводят их систематизацию, определяют способ получения и преобразования данных

Задачи MLOps:

- организовать сбор данных с использованием определенных в техническом задании технологий
- обеспечить надежное хранение данных,
- реализовать политики безопасного доступа к данным с учетом их специфики (коммерческая тайна, персональные данные и т.п.)
- организовать рабочее окружение для обработки данных

Инструменты MLOps:

- сетевые протоколы и утилиты для передачи информации: socket, API, ftp, syslog
- программное обеспечение для сбора и преобразования данных: Apache Flume, Nifi
- управление версиями датасетов и пайплайнов-обработчиков: dvc

3. с полученными данными начинает работать ML исследователь и ML инженер, которые должны построить модели

Задачи MLOps:

- организовать доступ к данным
- организовать рабочее окружение для ML инженера
 - виртуальная среда
 - интерактивное окружение для работы
 - рабочие библиотеки

Инструменты MLOps:

- библиотеки машинного обучения: python, numpy, pandas, matplotlib, sklearn, TensorFlow, PyTorch и др.
- интерактивные python-ноутбуки Jupyter, JupyterHub, binder для работы исследователей
- создание виртуальных окружений: venv, virtualenv, conda, poetry

4. Для моделей ML формируются технологические метрики оценки качества, увязанные с бизнес-метриками, необходим их постоянный контроль

Задачи MLOps:

- мониторинг метрик
- мониторинг процесса обучения модели
- сохранение наилучших параметров модели для переиспользования

Инструменты MLOps:

- визуализация мониторинга: grafana, Kibana
- визуализация контроля обучения: tensorboard

5. В соответствии с полученными метриками и построенными моделями инженер-тестировщик проводит тестирование модели

Задачи MLOps:

- автоматизация тестирования модели

Инструменты MLOps:

- автоматизация тестирования: pytest
- визуализация контроля обучения: tensorboard

6. На основе созданного и протестированного прототипа модели инженер-программист разрабатывает продакшн-вариант, либо интегрирует уже существующий модуль в общее решение, с учетом имеющихся ограничений на ресурсы, сформулированные заказчиком

Задачи MLOps:

- сборка решения
- общее тестирование решения

Инструменты MLOps:

- автоматизация сборки: Jenkins, git
- тестирование: pytest
- виртуализация: docker, docker-compose, kubernetes.
- различные решения для организации работы системы целиком
 - базы данных
 - веб-сервер
 - бэкенд
 - фронтэнд
 - очереди сообщений
 - API
 - и т.п.

7. Инженеры выводят созданную модель в продуктивное использование и обеспечивают контроль работоспособности модели в продуктивной среде.

Задачи MLOps:

- организация рабочего окружения для работы решения
- вывод в продакшн
- автоматизированный мониторинг работы системы
 - качество данных
 - качество модели
 - использование ресурсов

Инструменты MLOps:

- автоматизация сборки: Jenkins, git
- мониторинг: grafana, Kibana.
- виртуализация: docker, docker-compose, kubernetes.

Теперь вы знаете, что:

- для вывода проекта машинного обучения в промышленную эксплуатацию необходима большая команда разных специалистов, у каждого свои задачи,
- роль MLOps наладить технологическую одновременную работу всех специалистов, автоматизировать рутинные операции для ускорения получения результата и повышения качества.
- на каждом этапе существуют различные инструменты, которыми должен уметь пользоваться специалист MLOps
- также во многих проектах важно знать и уметь пользоваться инструментами, которыми пользуются другие участники команды

Теперь пришло время более подробно изучить инструментарий.

Проверим:

1. Распределите инструменты, которые применяются участниками команды проекта машинного обучения с учетом их роли
 - a. Data Engineer - SQL
 - b. ML Researcher - Jupyter Notebook
 - c. топ-менеджер, спонсор проекта - dashboard
 - d. MLOps Engineer - Jenkins
2. Установите соответствие между инструментом и его назначением

pytest	автоматизация тестирования
sklearn	библиотека машинного обучения
dvc	контроль версий датасетов
python	интерпретатор языка программирования

Модуль 1. Юнит 5: Обзор инструментов ML/MLOps

В этом юните:

Сделан обзор наиболее популярных инструментов, используемых в проектах машинного обучения и MLOps. Перечень инструментов может меняться. Знать плюсы и минусы, особенности применения, важно для эффективного решения задач MLOps.

Содержание:

На практике существует множество инструментов, используемых для автоматизации задач машинного обучения, о некоторых из них мы уже узнали в предыдущем юните. В этом юните мы изучим эти инструменты, что позволит наконец переходить к практическим задачам.

Состояние индустрии машинного обучения достигло такого уровня, что практически для каждой задачи сейчас существуют надежные и высокопроизводительные библиотеки и фреймворки, хорошо решающие эту задачу. Это объясняется активностью сообщества, большим количеством open-source проектов, доступностью технологий и вовлечением большого количества исследователей. Стандартные фреймворки (например, TensorFlow, PyTorch, sklearn и многие другие) уже содержат готовые, протестированные, хорошо документированные функции для решения практически всех возможных задач машинного обучения. **Поэтому сейчас чаще всего разработка модели машинного обучения связана с использованием и адаптацией готовых решений, создание модели «с нуля» зачастую является «изобретением велосипеда» и неэффективно с точки зрения ресурсов.** У такого подхода есть и минусы – поскольку большинство инструментов находятся в постоянной разработке, да еще и разработчиков огромное количество, то постоянно появляются новые версии библиотек, при этом часто возникает проблема совместимости разных версий библиотек между собой, так как большой проект включает в себя множество разных библиотек. Это может привести к тому, что при незначительном изменении версии одной из используемых библиотек модель начинает вести себя по-другому. Такого рода ошибки особенно сложно выявлять, **поэтому управление составом и версиями используемых компонентов, датасетов и конфигурациями модели позволяет существенно сократить время на создание, тестирование, оптимизацию, поиск ошибок в моделях машинного обучения.**

На разных этапах жизненного цикла проекта используется различное программное обеспечение, его выбор и конфигурация определяются условиями проекта, имеющимися ограничениями и иногда личным вкусом участников проекта. Однако перед использованием того или иного инструмента важно задавать следующие вопросы:

- Какова стоимость используемого решения и оправдано ли соотношение цены и качества? Можно ли использовать бесплатное решение?
- Какие временные затраты связаны с внедрением и эксплуатацией?
- Есть ли встроенные механизмы контроля качества?
- Насколько просто будет обучить разработчиков и пользователей, есть ли необходимая документация?
- Все ли ожидаемые функции выполняются?

- Какие потребуются ресурсы: вычислительная мощность, оперативная память? Список вопросов может быть шире для различных проектов.

Вот далеко не полный перечень некоторых инструментов, используемых для автоматизации проектов разработки программного обеспечения, с привязкой этих инструментам к этапам жизненного цикла проекта.

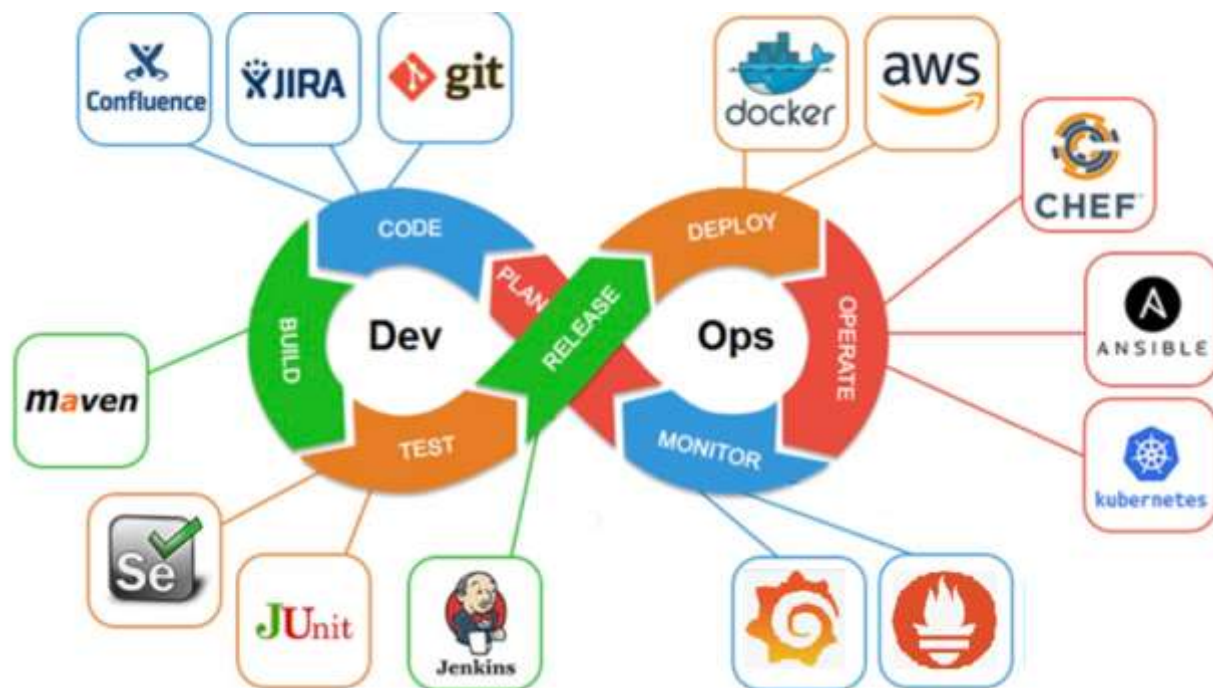


Рисунок «Этапы жизненного цикла проекта и некоторые инструменты автоматизации»

Далее рассмотрим эти инструменты более подробно и обсудим какие навыки они требуют от MLOps инженера.

1. Инструменты исследователей данных, ML исследователей и инженеров

Название	Источники	Описание	Что требуется от MLOps
tensorflow, tensorboard	tensorflow.org	Открытый фреймворк машинного обучения, созданных Google. Позволяет быстро создавать и обучать сложные модели (например, нейронные сети). Эффективен для проверки гипотез.	Установка, наладка, мониторинг, контроль версий и окружения, администрирование работы с GPU.

pytorch	pytorch.org	Открытый фреймворк машинного обучения, эффективен для продакшн	Установка, наладка, мониторинг, контроль версий и окружения, администрирование работы с GPU.
sklearn	sklearn.org	Множество различных алгоритмов машинного обучения. Удобен для проверки гипотез.	Установка, наладка, мониторинг, контроль версий и окружения.
jupyter, jupyterhub, binder	jupyter.org	Интерактивная среда разработки.	Установка, настройка, контроль ресурсов (память, CPU, HDD), поддержка.
python	python.org	Язык разработки, интерпретатор.	Установка, настройка, поддержка.
venv, virtualenv, poetry, conda	pip python.org anaconda.com	Виртуальные среды. Позволяют создавать индивидуальное окружение (версии библиотек, переменные окружения) для отдельного проекта.	Установка, настройка, поддержка.

2. Управление версионированием кода, датасетов, моделей, конвейеров

Название	Источники	Описание	Что требуется от MLOps
github	github.com	Система хранения и версионирования кода программного обеспечения	создание репозиторий, установка и настройка, организация работы, интеграция с другими системами

			(например, Jenkins)
dvc	dvc.org	Система хранения и версионирования датасетов	настройка, организация работы, интеграция с другими системами
Apache Airflow	airflow.apache.org	Автоматизация конвейеров машинного обучения	настройка, организация работы

3. Автоматизация тестирования и сборки

Название	Источники	Описание	Что требуется от MLOps
jenkins	jenkins.org	Автоматизация всех этапов от создания до эксплуатации	настройка, организация работы, написание скриптов автоматизации
pytest	pytest.org	Автоматизация тестирования	настройка, организация работы
pydoc	pydoc.org	Автоматизация подготовки документации	настройка, организация работы

4. Контейнеризация

Название	Источники	Описание	Что требуется от MLOps
docker	docker.org	Инструмент для создания виртуальных сред - контейнеров для выполнения программ	установка и настройка, организация работы, контроль ресурсов
docker-compose	docker.org	Управление различными контейнерами	установка и настройка, организация работы,

			контроль ресурсов
kubernetes	kubernetes.org	Управление различными контейнерами	установка и настройка, организация работы, контроль ресурсов

5. Мониторинг

Название	Источники	Описание	Что требуется от MLOps
grafana	grafana.org	Построение дашбордов	настройка, организация работы, интеграция с базами данных
kibana	elastic.org	Визуализация	настройка, организация работы
tensorboard	tensorflow.org	Визуализация и контроль работы модели машинного обучения	настройка, организация работы, интеграция со средой обучения модели

С этими инструментами более подробно вы познакомитесь в следующих модулях курса.

В изученном юните:

- рассмотрели наиболее популярные инструменты для проектов машинного обучения и MLOps:
 - версионирование: git, dvc,
 - инструменты исследователей данных и моделей машинного обучения: python, sklearn, tensorflow
 - автоматизация: jenkins, pytest, pydoc
 - контейнеризация: docker, docker-compose, kubernetes,
 - мониторинг: grafana, kibana
- узнали на что необходимо обращать внимание при выборе того или иного инструмента: требования к ресурсам, навыки команды, оборудование.

Теперь, когда мы вооружены знаниями об инструментах мы можем смело вступать во взаимодействие с внешней средой - заказчиком, создающим бизнес-требования, оказывающие существенное влияние на все аспекты процесса и, в том числе, на MLOps задачи.

Проверим:

1. Отметьте инструменты, которые используются для контроля версий различных артефактов
 - a. **git**
 - b. **dvc**
 - c. grafana
 - d. linux
2. Кто является разработчиком TensorFlow, осуществляющим поддержку этого решения на открытом исходном коде
 - a. Илон Маск
 - b. Facebook
 - c. **Google**
 - d. открытое сообщество
3. Какие инструменты используются для мониторинга
 - a. **grafana**
 - b. tensorflow
 - c. pytorch
 - d. **kibana**
4. Какие инструменты используются для организации виртуальной среды
 - a. **venv**
 - b. **docker**
 - c. **kubernetes**
 - d. http
5. Установите соответствие между участниками команды и используемыми ими инструментами

jupyter ноутбук	исследователь
jenkins	DevOps инженер
grafana	эксплуатация
pytest	тестировщик

Модуль 1. Юнит 6: Учет бизнес-требований и их влияние на MLOps

В этом юните вы узнаете:

- почему для реального практического проекта важно учитывать бизнес-требования,
- как правильно должны быть сформулированы бизнес-требования, чтобы это не стало головной болью участников проекта машинного обучения и, в том числе, MLOps,
- что может входить в постановку задачи,
- как постановка задачи влияет на MLOps.

Содержание юнита:

Выполнение любого проекта связано с достижением целей, сформулированных постановщиком задачи, как правило это заказчик. Эти требования связаны с достижением бизнес-целей. Бизнес-целью может быть, например, повышение конверсии посещений сайта в заключенные договоры, увеличение трафика в торговой точке, снижение издержек на транспортировку товара. Есть бизнес-цели и в проектах машинного обучения. То, кто уже имеет опыт работы в реальных промышленных проектах машинного обучения, уже знают, что реальный проект отличается от учебных и тренировочных (например, от соревнований на площадке kaggle.com). Самое главное отличие состоит в том, что в соревновательном проекте зачастую главное это показать наилучший результат работы модели, “побить бейзлайн”, а в реальном проекте необходимо очень хорошо понимать и выполнять бизнес-требования заказчика проекта, при этом возможна ситуация, когда построенная модель показывает не лучшее качество, однако укладывается в имеющиеся ограничения (оперативная память, скорость работы, CPU). С определением бизнес-требования вы ознакомились в рамках курса “Управление проектами”. В этом же юните мы посмотрим на влияние бизнес-требований на задачи MLOps, поскольку понимание бизнес-требований позволяет эффективно планировать инфраструктуру, что в последующем экономит силы и время и является залогом общего успеха проекта.

Одним из важных условий создания и запуска в эксплуатацию эффективной модели машинного обучения является наличие грамотно составленного технического задания. **Техническое задание это важнейший проектный документ, фиксирующий ожидания от проекта и возможные ограничения.** Обычные пункты такого технического задания включают:

- Сроки проекта.
- Требования к качеству, бизнес-метрики качества модели. На основе понятных и прозрачных бизнес-метрик аналитики команды проекта осуществляют их привязку к технологическим метрикам, понятным разработчикам и исследователям.
- Технологические ограничения, связанные с выводом модели в продакшн:
 - ограничения по памяти,
 - ресурсам процессора,
 - скорости работы модели,
 - требования к устойчивости модели к шуму в данных,
 - уровень надежности.

На практике внедрение новейших технологий, фреймворков, наилучших передовых алгоритмов (SOTA, State-of-the-Art) не является самоцелью. Главное в проекте, это решение практической производственной задачи, которое приносит ощутимую пользу, при этом укладывается в заданные ограничения по ресурсам. Если руководитель проекта не может сказать какой бизнес кейс решает проектная команда, причем в цифрах, то такой проект скорее всего будет бесполезен.

Для проектов машинного обучения сложно посчитать точно эффект, так как машинное моделирование не дает никогда 100% точности. Поэтому чаще оценивают – какой эффект для бизнеса принесет улучшение качества работы модели (например, на 0.1%, 1%, 10%). Часто бывает, что потраченные усилия на дообучение модели не дают желаемого коммерческого эффекта, поэтому можно было это дообучение не проводить.

Пример: если стоимость создания модели оценивается в миллионы рублей (с учетом аппаратного и программного обеспечения, стоимости оплаты работы команды, электроэнергии и остальных затрат), а задача могла бы быть выполнена с использованием ручного труда небольшого числа специалистов низкой квалификации, то создание такой модели должно быть признано неэффективным.

Модели машинного обучения эффективно применять для задач, на которые у человека потребовалось бы большое количество времени. Важно при этом понимать:

- какую задачу (проблему) мы хотим решить?
- какой ожидаем получить результат?
- какой уже имеется результат и можем ли мы его превзойти?
- как понять, что полученный результат лучше существующих решений, например, ручной обработки?
- какие дополнительные накладные расходы готов нести заказчик

Пример неэффективной постановки задачи: у нас есть большое количество разных данных, давайте попробуем что-то в них найти или применить алгоритмы “X” и “Y”, чтобы система стала работать лучше.

Пример эффективной постановки задачи: мы хотим повысить объем продаж на сайте на 10 процентов за счет использования новой рекомендательной системы для рекомендаций товаров посетителям.

Без четкой постановки задачи применение технологий машинного обучения будет скорее всего неуспешным. Важно понимать, что для разных участников проекта изложенные выше вопросы будут преобразованы в различные цели, связь между которыми не очевидна даже для самих участников проекта. Например, инженер, решающий задачу автоматизации процесса развертывания, реализует развертывание по расписанию с выполнением необходимых автотестов и выводом решения в продакшн. Такой специалист как правило не знает, что бизнес цель и развертывания этой модели это сократить процент оттока посетителей онлайн-сервиса на 10%. В командах, работающих по agile, обычно приветствуется обмен информацией, в том числе все участники проекта имеют возможность получить обратную связь от рынка и оценить важность бизнес-составляющей

проекта. Однако в больших проектах такое не всегда возможно, поэтому важна роль экспертов и бизнес-аналитиков, обеспечивающих связь бизнес-метрик (например, увеличение конверсии посетителей на 10%, повышение среднего чека на 5%, уменьшение оттока покупателей на 1% и т.п.) и технологических метрик (precision, recall, F1 в классификации, MAE, MSE в регрессии и т.п.).

Важным документом, фиксирующим условия выполнения проекта, необходимые требования и ресурсы, является техническое задание (ТЗ). Это типовой документ, часто имеющий юридическую силу (если является приложением к договору). От качественного составления и согласования технических требований в техническом задании очень сильно зависит успех проекта. Наиболее важными в техническом задании проекта машинного обучения являются следующие пункты, от точности описания которых зависит объем ресурсов, сроки, требуемые навыки команды:

- измеримые бизнес цели,
- скорость,
- точность модели,
- сроки реализации,
- ограничения в аппаратных ресурсах (производительность процессора, оперативная память, размер жесткого диска, количество серверов, скорость передачи данных),
- необходимость резервирования
- с какими системами необходимо будет интегрировать модель ML
- бизнес-процесс, включающий в себя использование ML (включая обработку FP, FN)
- план внедрения системы, как правило внедрение должно идти постепенно, с анализом качества работы

На практике специалиста MLOps часто привлекают к обсуждению технического задания, поскольку это сильно влияет на выбор инструментов и подходов к решению задач MLOps.

В этом юните вы узнали:

- на что необходимо обращать внимание при постановке задачи
 - конкретные и измеримые показатели, которых нужно достичь
 - ограничения ресурсов, которые нельзя нарушать
 - организационные требования: сроки, частота релизов
- что оказывает сильное влияние на задачи MLOps
 - какую задачу (проблему) мы хотим решить?
 - какой ожидаем получить результат?
 - как понять, что полученный результат лучше существующих решений, например, ручной обработки?
 - какие есть ресурсы?

Теперь вы знаете, что MLOps очень востребован в современных проектах машинного обучения в связи с множеством различных задач и технологий. Из примеров, изложенных в данном модуле, становится ясно, что специалистам MLOps необходимо очень много знать для эффективного выполнения своих задач. Пришло время убедиться в этом на конкретных примерах.

Проверим:

1. Отметьте примеры правильных бизнес-метрик
 - a. **уменьшить отток клиентов на 1%**
 - b. сделать сайт покруче
 - c. увеличить выручку
 - d. **увеличить процент посетителей сайта, совершивших покупку, на 5%**
2. Отметьте примеры правильных технологических метрик
 - a. быстрая скорость обучения модели
 - b. **F1 не ниже 0.9**
 - c. **скорость инференса (работы модели на реальных данных) не более 0.5 секунды**
 - d. использование GPU
3. Установите соответствие между бизнес задачей и технологической метрикой

Определить лояльных пользователей	accuracy, precision, recall, F1
Спрогнозировать уровень нагрузки сети	MSE, MAE
Разделить магазины на группы (кластеры), обладающие сходством элементов	метрика “силуэт”

4. Поставьте в соответствие ограничение в техническом задании и объект, на который оказывает влияние это ограничение

скорость инференса	модель машинного обучения
объем жесткого диска	количество и тип признаков для обучения
частота релизов	инструменты автоматизации
возможность появления шума в данных	инструменты мониторинга и резервирования

Модуль 1. Юнит 7: Обзор требований к специалистам MLOps

Об этом юните:

После изучения предыдущих юнитов данного модуля вы уже знаете, что к специалистам MLOps предъявляются очень высокие требования. Наилучший способ убедиться в этом - рассмотреть реальные вакансии компаний на позиции, связанные с MLOps.

Содержание юнита:

Каждый этап жизненного цикла проекта и каждый участник проекта создают определенные задачи для MLOps, при этом состав задач MLOps может сильно зависеть от конкретного предприятия, отрасли или проекта. Поэтому от специалистов MLOps требуется широкий кругозор. В качестве примера проанализированы требования к специалистам MLOps на основе открытых вакансий от промышленных компаний. Но даже в таких неопределенных условиях можно применять типовые сценарии и инструменты, которые будут изучаться в данном курсе. В данном юните на примере конкретных описаний требований к вакансиям MLOps разбираются требования.

Вакансия 1

Описание компании:

Небольшая, географически распределенная ИТ-компания реализует проект в сфере работы с медиа контентом.

Описание должности:

MLOps инженер, от 320 000 руб. до вычета налогов, требуемый опыт работы: 3–6 лет, полная занятость, удаленная работа

Задачи:

- участие в жизненном цикле проектов Data Science на всех этапах, включая дизайн и разработку сервисов, планирование ресурсов и обслуживание,
- участие в устранении и расследовании причин сбоев, постмортемы,
- улучшение процессов связанных с обслуживанием систем (отказоустойчивость, масштабирование, работа в режиме высокой нагрузки и т.п.),
- автоматизация процессов сборки, тестирования и доставки приложений;
- автоматизация построения пайплайнов обработки данных, поддержка инфраструктуры,
- взаимодействие с командами разработки и тестирования в части подготовки, запуска и поддержки сервисов.

Требования:

- опыт в области машинного обучения более трех лет,
- опыт использования инструментов для построения инфраструктуры работы с данными, платформ для управления жизненным циклом ML, оркестрацией и процессами (MLflow, Kubeflow, Airflow, DVC или аналоги),
- знакомство с процессами машинного обучения,
- опыт работы с JupyterHub/Lab, Spark, Kafka, Presto, Hive, Flink, MySQL,
- умение проектировать архитектуру приложения и хранилища,

- опыт работы с torch serving, tensorflow serving, triton serving,
- опыт конвертации моделей в ONNX, TensorRT,
- уверенное владение Linux,
- знание языков программирования (Python, Go желательно),
- понимание принципов построения отказоустойчивых решений,
- опыт работы с HighLoad системами,
- понимание работы и опыт организации CI/CD,
- аналитический склад ума,
- аккуратность, ответственность,
- желание разбираться в сложных задачах.

Как видите, кроме опыта работы с необходимым инструментарием MLOps от специалиста требуется широкий круг знаний и навыков: программирование (python, Go), проектирование, администрирование систем. Кроме того, подчеркивается важность так называемых soft-skills (аналитический склад ума, ответственность, готовность к сложным задачам), что обосновано важностью роли специалиста в команде - MLOps инженер в этом проекте активно участвует в развитии продукта и много взаимодействует с другими участниками. Это характерно для проектов небольших компаний-стартапов, в которых участники команды обычно выполняют множество обязанностей.

Вакансия 2

Описание компании:

Команда внутри большой компании (страховой бизнес) создает современную платформу для задач Data Science. В команду требуется MLOps (DevOps) инженер.

Задачи:

- развитие среды исполнения моделей машинного обучения,
- обучение сотрудников отдела практикам и владению инструментами MLOps,
- настройка CI/CD для проектов DS,
- помощь в настройке систем сбора признаков, исполнения моделей и мониторинга.

Требования:

- опыт настройки CI/CD в ML,
- уверенное знание Linux,
- знание и опыт применения контейнеризации (Docker, Kubernetes, Openshift),
- опыт настройки и использования Apache Airflow.

Это уже совсем другой проект, сильно отличающийся от первой вакансии. От специалиста MLOps ожидается уже меньшее количество навыков, так как команда работает внутри большой компании, в которой уже многие процессы налажены и информационные системы внедрены. Поэтому не требуется навыком проектирования систем и хранилищ.

Вакансия 3

Описание компании:

Команда проекта создает модели машинного обучения для беспилотного транспорта (self-driving).

Описание требований:

- MLOps Engineer (AI Driving Data), требуемый опыт работы: 3–6 лет, полная занятость, полный день,
- улучшение качества ML-решений,
- выбор и внедрение инструментов MLOps, поддержка инфраструктуры для обучения моделей, создание и поддержка ML-пайплайнов,
- организация обработки данных от беспилотников, участие в создании хранилища для этих данных,
- стек: MLFlow, Kubeflow, DVC, ETL, CI/CD/CT,
- опыт разработки на Python от трех лет,
- знание алгоритмов и методов машинного обучения,
- опыт построения инфраструктуры работы с данными,
- опыт организации CI/CD,
- опыт разработки на C++,
- знание лучших практик DevOps.

Перечень задач:

- внедрить и развивать платформу полного жизненного цикла ML-моделей,
- собирать и запускать ML-пайплайны,
- поддерживать модели, наборы данных и параметры в системах CI/CD/CT,
- поддерживать модели в проде, решать сложные проблемы.

Вакансия 4

Описание команды и проекта:

Команда внутри большой компании создает набор продуктов и сервисов, предназначенных для решения задач кибербезопасности: противодействие внутреннему и внешнему мошенничеству, защита инфраструктуры организации (банковский сектор), анализ рисков и угроз, аудит кибербезопасности и безопасной разработки.

Ядро платформы - кластер Hadoop (Cloudera), хранит более 10Пб данных, которые обрабатываются на более чем 5000 процессорных ядер и 50Тб оперативной памяти в поточном и пакетном режимах. Состав разрабатываемой системы:

- среда исследования и моделирования - продукт, позволяющий дата-сайнтистам проводить исследования на промышленных данных. Продукт представляет из себя кластер Hadoop (Cloudera), поднимающийся on-demand, а так же набор сервисов для работы с данными на кластере - Spark, Hive, Apache Kafka, Apache Ni-fi, Apache Flink, Clickhouse, Jupyterhub, Apache Airflow,
- Model Execution Framework (MEF) - фреймворк, в основе которого лежит Opehshift, позволяющий типовым образом деплоить обученные AI-модели на тестовые и промышленные среды.

Обязанности:

- развитие среды исследования и моделирования,
- деплой моделей в MEF,

- автоматизация процесса сопровождения,
- устранение инцидентов,
- обеспечение надежности сервисов,
- ведение документации,
- мониторинг,
- наставничество.

Требования:

- релевантный опыт работы в качестве DevOps инженера,
- понимание как работает Nadoor и сервисы его экосистемы,
- уверенное владение Linux (желательно RHEL),
- умение писать роли и плейбуки Ansible,
- умение создавать пайплайны в Jenkins, либо аналогичных системах,
- Git на уровне создать ветку, выставить PR, пройти code review и “вмержить” в мастер,
- желательно знакомство с Zabbix,
- опыт работы с Kubernetes/OpenShift,
- владение Python для дебага моделей.

Из описания видно, что компания уже обладает развитой мощной экосистемой для работы с большими данными, имеет отлаженные процессы и настроенные инструменты, а также уже созданный продукт, который необходимо развивать. Поэтому основной акцент на задачах, близких к эксплуатации и развитию системы, а также обучению (наставничество).

Вакансия 5

Описание компании и проекта:

We at Samsung AI Center are looking for a MLOps engineer for several computer vision projects. We are doing R&D projects in generative computer vision and 3D computer vision, which require a lot of data processing, storing and experiments and we want to improve our main research and production pipelines. Samsung AI Center Moscow has been founded in 2018. Project topics include:

- Generation of photorealistic human avatars
- Point clouds-based 3D image and video synthesis
- Techniques for image manipulation using generative networks
- Training neural networks using Bayesian methods
- Creation of multi-agent systems
- Single-view and multi-view 3D reconstruction
- Localization and navigation using 3D vision

Responsibilities:

- Benchmarking our hardware and software setups on clusters
- Collecting and serving different docker images and repository
- Creating new tools to improve our experiments tracking and deployment pipelines

- Continuously support our engineers in best code practices and automation of all development pipelines
- Introduce CI/CD tools in our working process
- Probably take part in NAS and mobile networks papers
- Maintain our local clusters, monitor jobs and support software updates
- Write and run scripts in python and bash to process raw data, collect statistics, clean, store dataset and show real time previews

Requirements:

- Professional experience with
- Maintaining linux-based servers and clusters, including CUDA
- Python, Pytorch, Tensorflow, Bash scripts
- Any frontend development framework (e.g. Angular, React, Vue.js)
- Docker, Kubernetes, KubeFlow
- Nvidia NGC docker repo, Nvidia Triton Inference server
- Basic data exchange protocols, e.g. GRPC, JSON, Mosquitto Buffers
- Experience with data storage formats, such as SQL, LMDB, HDFS and parallel/distributed files systems, e.g. GPFS
- As a plus:
 - Got ML and DL algorithms knowledge
 - Experience with writing/reimplementing ML/DL papers
 - Contributed to open-source github projects
 - Have a track of contributions to IT products and services
 - Experience with TFLite and Mediapipe
 - Professional C++/CUDA, OpenCL or OpenGL experience
 - Experience with MLFlow/ Neptune.ai/ Weights and Biases and other experiments tracking software
 - Experience with FairScale, Torch DDP, Pytorch Lightning, Tensorflow Extended, GPipe

Это самая “богатая” по требованиям вакансия из приведенных в данном юните, от одной из ведущих исследовательских лабораторий в сфере искусственного интеллекта Samsung AI Center. Компания использует передовой стек технологий, в том числе такие инструменты, которые не входят в данный курс.

Итак, из рассмотренных вакансий ясно, что требования к специалисту MLOps могут быть самые различные, это очень зависит от компании и проекта. Но зачастую эти требования гораздо шире, чем просто автоматизация процесса развертывания модели, поэтому специалисту MLOps для эффективного выполнения своих обязанностей необходимо вникать в предметную область (финансы, медицина, телекоммуникации, страховое дело, ...), а также понимать цели и задачи смежных подразделений.

Из этого юнита вы узнали, что

- требования, предъявляемые к специалистам MLOps, обеспечивающим автоматизацию в проектах машинного обучения, очень высокие, что мы увидели на примере рассмотренных вакансий,
- часто от специалиста MLOps требуется погружение в смежные задачи (команды, заказчиков) для лучшего понимания проекта и более эффективного решения задач
 - понимание программного кода,
 - умение работать с инструментами, используемыми командой, понимание правил их настройки,
 - умение вникнуть в техническое задание и сформулировать требования к инфраструктуре на его основе.
- сложно составить “универсальный” перечень знаний, который подойдет для любого проекта, однако есть “необходимый” уровень знаний, востребованных в каждом проекте
 - знание основных инструментов автоматизации (python, bash, Jenkins или аналоги)
 - программирование скриптов
 - умение создавать виртуальные среды
 - настройка оборудования и программного обеспечения.

Модуль 1. Юнит №8

В этом модуле были изучены следующие вопросы:

- Зачем изучать MLOps?
 - использование этой идеологии позволяет сократить время и издержки, повысить качество проекта машинного обучения
- Какие знания нужны на старте?
 - разработка программного обеспечения (подходы, архитектура, технологии)
 - операционные системы, чаще всего linux
 - машинное обучение
 - скрипты автоматизации (python, bash)
- Из чего состоит MLOps?
 - автоматизация процессов
 - версионирование объектов проекта
 - датасеты
 - пайплайны обработки данных
 - модели машинного обучения
 - виртуальные среды
- Как архитектура и жизненный цикл проекта машинного обучения влияют на задачи MLOps?
 - на каждом этапе жизненного цикла проекта машинного обучения возникают свои специфические задачи MLOps
 - архитектура решения и имеющиеся ограничения по ресурсам оказывают влияние на задачи и инструментарий MLOps
- Какие инструменты используются в MLOps?
 - версионирование: git, dvc
 - виртуализация: venv, virtualenv, poetry
 - контейнеризация: docker, docker-compose, kubernetes
- Как постановка задачи и бизнес-требования влияют на MLOps?
 - бизнес-требования трансформируются в технические требования к инфраструктуре и технические метрики проекта.
- Какие требования предъявляют компании к специалистам MLOps (на примере вакансий hh.ru)?

Дополнительные материалы

LEAN_DS <https://ods.ai/tracks/lean-ds-df2020>

<https://www.atlassian.com/software/jira>

<https://www.atlassian.com/ru/software/confluence>

https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#devops_versus_mlops

Модуль 1. Юнит №9: Практическое задание к модулю

Цель задания:

В этом задании вы решите простую и во многом учебную задачу по созданию автоматического конвейера проекта машинного обучения. Подобный подход с применением простых скриптов автоматизации для “склейки” отдельных частей конвейера используется в небольших проектах. Чаще же для автоматизации используется специализированное программное обеспечение, например, Jenkins.

Содержание задания:

Необходимо из “подручных средств” создать простейший конвейер для автоматизации работы с моделью машинного обучения. Отдельные этапы конвейера машинного обучения описываются в разных python-скриптах, которые потом соединяются (иногда используют термин “склеиваются”) с помощью bash-скрипта.

Этапы:

1. Создайте python-скрипт (`data_creation.py`), который создает различные наборы данных, описывающие некий процесс (например, изменение дневной температуры). Таких наборов должно быть несколько, в некоторые данные можно включить аномалии или шумы. Часть наборов данных должны быть сохранены в папке “train”, другая часть в папке “test”.
2. создайте python-скрипт (`model_preprocessing.py`), который выполняет предобработку данных, например, с помощью `sklearn.preprocessing.StandardScaler`.
3. создайте python-скрипт (`model_preparation.py`), который создает и обучает модель машинного обучения на построенных данных из папки “train”.
4. создайте python-скрипт (`model_testing.py`), проверяющий модель машинного обучения на построенных данных из папки “test”.
5. Напишите bash-скрипт (`pipeline.sh`), последовательно запускающий все python-скрипты.

Подготовленные скрипты необходимо опубликовать в git репозитории, ссылку на который необходимо предоставить как результат выполнения задания.

Творческое задание на вебинар:

Предположим, что заказчик поставил перед командой задачи: разработать прогнозную модель для зарплат специалистов MLOps на два года вперед. Заказчик хочет Определять уровень зарплат в среднем по рынку, чтобы нанимать в течении двух лет 10 специалистов и не переплатить более 500.000 т.р. Решение заказчик хочет эксплуатировать на своих серверах (поскольку не хочет раскрывать персональные данных своих сотрудников). Однако сервера у заказчика ненадежные, технические описания он не предоставляет.

1. Предложите технические метрики машинного обучения, которые могли бы соответствовать этой постановке задачи и инструменты для ее решения.
2. С какими сложностями может столкнуться специалист MLOps при реализации задач в такой постановке
3. Какой перечень мероприятий можно предложить для минимизации рисков.

Модуль 2. Непрерывная интеграция, доставка и обучение (CI/CD/CT) в проектах ML

В этом модуле:

Вы уже знаете из предыдущего модуля, что в программной инженерии придумано много способов повысить эффективность проекта. Одним из действенных способов повышения эффективности проекта является автоматизация отдельных рутинных операций, связанных с созданием, сборкой и тестированием сложных проектов программного обеспечения, а также установкой этого программного обеспечения в промышленную среду и организацией эксплуатации. Методы, связанные с созданием, тестированием и сборкой проекта называются «непрерывная интеграция» (Continuous Integration, CI). Все, что связано с выводом решения в производственную среду эксплуатации (production, «прод») называется «непрерывная доставка» (Continuous Delivery, CD). Кроме того, для проектов актуальна автоматизация тестирования (Continuous Testing, CT), при этом, как будет показано далее в модуле, этот набор методов приобретает другое, более важное значение для проектов машинного обучения, «непрерывное обучение» (Continuous Training).

В этом модуле обсуждается то, что необходимо автоматизировать в проекте машинного обучения, а также описаны задачи и инструменты для CI/CD/CT для проектов машинного обучения. В качестве примера более подробно рассматривается инструмент Jenkins.

Темы, изучаемые в модуле:

1. Автоматизация. Непрерывная интеграция, развертывание и тестирование.
2. Задачи и инструменты CI/CD/CT в проектах машинного обучения.
3. Пример организации CI/CD/CT в проектах машинного обучения с использованием Jenkins.

Модуль 2. Юнит 1. Автоматизация. Непрерывная интеграция, развертывание и тестирование.

Введение: В этом юните вы узнаете о задачах CI/CD/CT. Отдельно более подробно разобрана наиболее важная задача: автоматизация.

Содержание юнита:

Современные программные системы состоят из множества отдельных взаимодействующих между собой модулей, над которыми могут работать различные команды проекта или даже разные компании. Вот пример такой системы, имеющей микросервисную архитектуру, в которой отдельные части решения реализованы в виде отдельных независимых друг от друга сервисов, общение происходит через служебные интерфейсы и протоколы, например API, telnet, FTP, ssh, сокет.

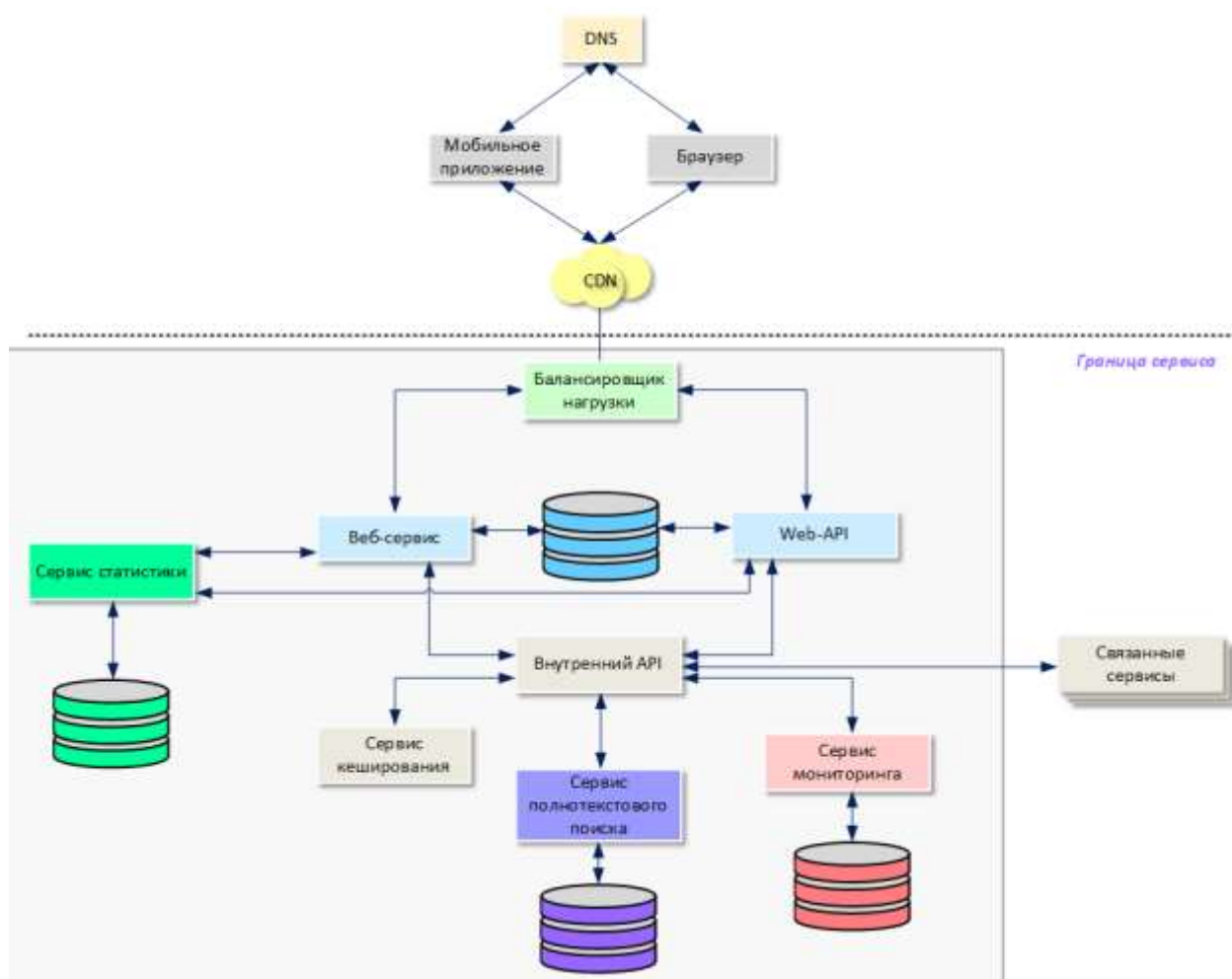


Рисунок из статьи «Архитектура современных веб-сервисов и способы их защиты»
https://www.anti-malware.ru/analytics/Technology_Analysis/Architecture-of-modern-web-services

Количество микросервисов в большом приложении может достигать сотен экземпляров, взаимодействие становится все более сложным. Управление созданием, тестированием, развертыванием и эксплуатацией подобных систем стало невозможным без специальных средств автоматизации, что привело к созданию методологии DevOps.

Давайте рассмотрим небольшой, часто встречающийся на практике пример.

Команда проекта машинного обучения состоит из следующих специалистов:

- инженер данных: пишет SQL запросы к базе данных, создает и обрабатывает датасет для построения моделей машинного обучения, отвечает за целостность и своевременность предоставления данных, не имеет доступа в production из-за ограничений в сети заказчика, использует в работе базу данных postgres,
- разработчик ML: получает данные от инженера данных, на их основе создает и обучает модель машинного обучения в виде python файла, вручную добавляет разработанную модель в в виде функции в существующий web-сервис, использует в работе окружение jupyter для экспериментов и библиотеки машинного обучения tensorflow, scikit-learn,
- специалист эксплуатации: отвечает за установку решения на сети заказчика, качество ее работы.

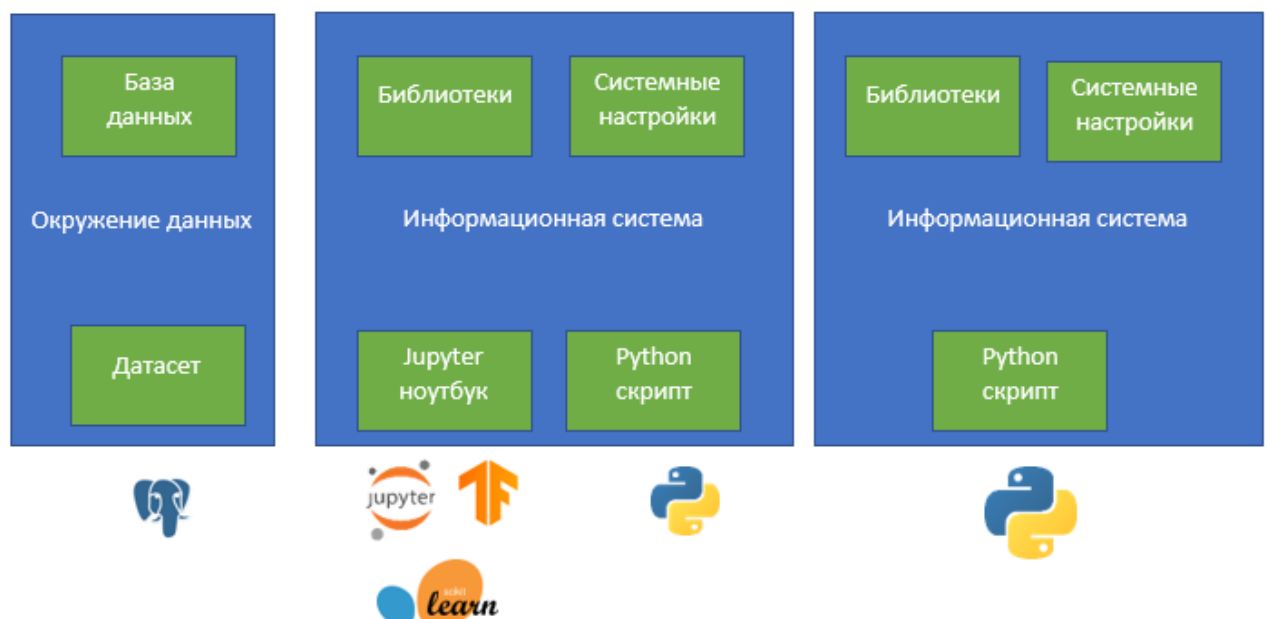


Рисунок “Задачи и инструменты команды проекта ML”

Технически возможно организовать работу в проекте без автоматизации вообще:

1. инженер данных руками делает выгрузку из базы данных с использованием заранее подготовленных SQL запросов, сохраняет результат в csv файл на жесткий диск, сообщает разработчику ML имя сохраненного файла,
2. разработчик ML формирует pandas датафрейм из csv файла, обучает на нем модель ML, подбирает эффективные значения гиперпараметров модели, сохраняет модель в формате pickle, либо сохраняет веса модели, либо python скрипт с найденными оптимальными значениями гиперпараметров модели,
3. специалист эксплуатации добавляет сохраненную модель ML в общий код проекта. При этом необходимо обеспечить идентичность окружений в которых модель обучалась и эксплуатировалась. Если не использовать автоматизацию, то это означает скрупулезную настройку всех элементов окружения - оборудования, операционной системы, библиотек.

Давайте проанализируем проблемы, которые будут “съедать” уйму времени у участников данного проекта и, в итоге, почти наверняка приведут к неуспеху проекта:

1. разработчик ML может перепутать имя csv файла и обучить модель на неправильных данных, потерять время и непроизводительно использовать аппаратные ресурсы,
2. разработчик ML может столкнуться с тем, что официальная версия одной из используемых им библиотек перестала поддерживаться (а это часто случается в open-source проектах), поэтому необходимо использовать другую, более новую версию, это может повлечь проблемы с совместимостью с другими библиотеками. Это ведет к перенастройке всей инфраструктуры для промышленной эксплуатации, поскольку для повторяемости результатов в промышленной среде необходимо стремиться к одинаковым настройкам.
3. после начала эксплуатации прошло некоторое время и специалист эксплуатации заметил, что модель стала работать плохо. Уведомление об этом инциденте получил разработчик ML, проанализировал свою модель и пришел к выводу, что модель не должна ошибаться. После длительного анализа выяснилось, что в промышленных данных появились расхождения с теми данными, на которых обучалась модель, поэтому эти расхождения необходимо добавить в базу данных, из которой формируется csv файл для обучения модели. После того как команда в авральном режиме изменила базу данных и переобучила модель выяснилось, что изменения в данных бали ошибкой прибора, и теперь все необходимо возвращать обратно.
4. инженер данных узнал, что в реальных промышленные данные стали отличаться от тех, которые сохранены в качестве тренировочных для обучения и проверки модели. Получив новые данные он загрузил их в базу, однако модель после обучения на новых данных стала работать хуже. Выяснилось, что изменились параметры работы моделируемой производственной системы и теперь старые данные более не актуально использовать.

Тот, кто хотя бы раз решал описанные выше проблемы, или подобные им, уже не откажется от возможности автоматизировать процесс, особенно есть возможность контролировать качество на каждом этапе и своевременно обнаруживать проблемы. Например, в описанном выше примере мы могли бы автоматизировать передачу данных от инженера данных разработчику ML с использованием специального конвейера по анализу и обработке информации. Специалист эксплуатации мог бы использовать специальные скрипты, которые разворачивают необходимые окружения с уже установленными параметрами. **Автоматизация мониторинга качества данных позволит своевременно увидеть изменения а данных, на которых работает модель. Каждая из этих мер автоматизации позволяет экономить большое количество времени, которое уходит на анализ проблемы и ее устранение.**

К счастью, такие методики автоматизации уже разработаны. **Автоматизация позволяет сделать процесс “непрерывным”, то есть продолжающимся постоянно, без участия человека.** И методы обеспечения непрерывности были разделены на несколько групп, с учетом того, какие задачи эти методы решают - **непрерывная интеграция, непрерывная доставка и непрерывное тестирование.** Рассмотрим их подробнее.

Уже знакомые нам из предыдущего модуля гибкие agile-методологии управления проектами включают в себя несколько стандартных шагов, в том числе относящиеся к DevOps:

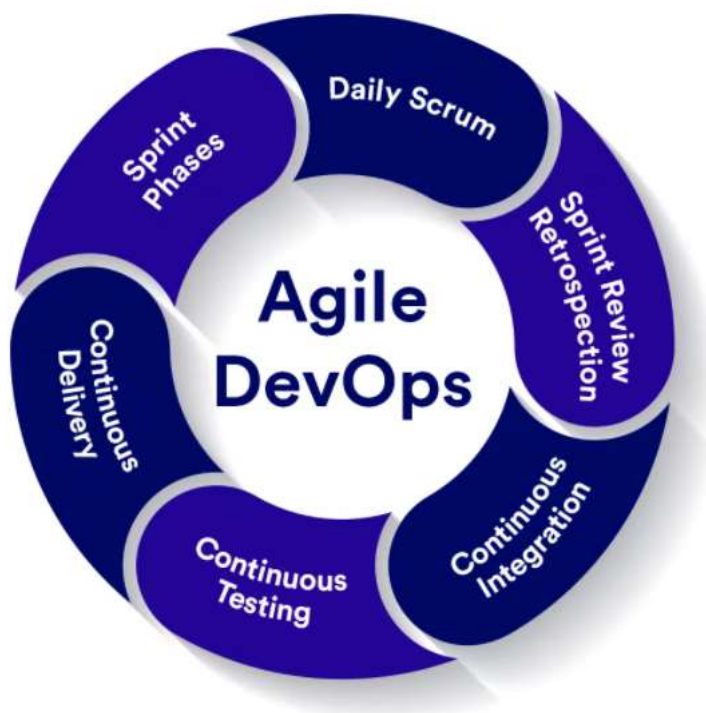


Рисунок “Место и роль CI/CD/CT в общем процессе проекта разработки программного обеспечения, управляемого по agile”

Мы помним, что цель любого проекта: предоставить заказчику (пользователю) решение, содержащее как можно меньше проблем, как можно быстрее. Рассмотрим отдельные фазы автоматизации, которые позволяют существенно упростить и ускорить процессы сборки и развертывания программного продукта.

1. Непрерывная интеграция (Continuous Integration, CI)

CI позволяет участникам проекта писать новый и обновлять существующий код разрабатываемой системы и сохранять (commit, «коммитить») эти изменения в едином пространстве для управления кодом. Такое пространство называют репозиторией проекта. На практике участники проекта могут сохранять множество изменений в разных файлах в течении дня. Для обеспечения качества проекта изменения в коде должны быть проверены. Не каждое внесенное изменение удастся проверить человеку, особенно если возможные ошибки касаются взаимодействия нескольких модулей. На этом этапе необходимым шагом является автоматизация нескольких задач, которые могут запускаться автоматически при обнаружении факта обновления кода, либо по расписанию:

- проверка правильности написания кода (автоматизированный код ревью)
 - выполнение правил именования функций и переменных,
 - наличие комментариев,
 - выполнение требования типизации параметров функции

- проверка корректности зависимостей между используемыми библиотеками, поиск противоречий или несовместимости
- запуск сценариев тестирования для проверки надежности кода.

Не следует думать, что технологии CI являются средством от всех “болезней”. Например, CI точно не принесет пользы в следующих случаях:

- команда состоит из одного человека, проект небольшой, изменений мало (например, pet-проект),
- разработчики редко «коммитят» в общий репозиторий проекта,
- команды разработчиков отдельных модулей работают в своих отдельных ветках, которые редко интегрируются с главной веткой.

Таким образом **CI непрерывно интегрирует написанный код в общее хранилище кода проекта, своевременно обнаруживая конфликты между отдельными частями кода.**

2. Непрерывная доставка (Continuous Delivery, CD)

CD содержит средства автоматизации для развертывания проекта в среде, в которой эта разработанная система будет использоваться (production, prod, “прод”). Иногда этот процесс называют «установка и настройка программного обеспечения», также используют жаргонный термин «выкатить в прод». Процесс развертывания программного обеспечения для промышленной эксплуатации состоит из отдельных шагов, кот некоторые из них:

- сконфигурировать и настроить оборудование, либо виртуальную облачную среду,
- установить и настроить операционную систему,
- установить переменные окружения,
- установить и настроить программы, утилиты, служебные библиотеки, обеспечивающие работу основной системы,
 - базы данных,
 - технологии контейнеризации,
 - компиляторы, интерпретаторы,
 - системы управления очередями,
 - веб-сервер,
 - система аутентификации,
 - и многое другое, что зависит от конкретного проекта.
- выполнить установку и настройку основной системы,
- создать необходимые пользовательские роли,
- обеспечить мониторинг.

CD обеспечивает непрерывную доставку разработанного программного обеспечения в среду эксплуатации, то есть позволяет автоматизировать настройку и установку с использованием специальных скриптов автоматизации, собираемых в единый конвейер.

3. Непрерывное тестирование (Continuous Testing, CT)

В отличие от рассмотренных CI/CD, главная цель которых это ускорение процессов за счет автоматизации рутинных процедур, назначение CT заключается в обеспечении необходимого качества за счет автоматизации различных тестов на различных этапах. Часто CT не рассматривают отдельно при изучении DevOps, предполагая, что достаточно описать автоматизацию тестов как часть процессов интеграции или доставки. Однако роль

тестирования становится все значительнее. И, как мы увидим в дальнейшем, тестирование в проектах машинного обучения является ключевым фактором успеха проекта. В проекте возможны следующие виды тестов:

- системное тестирование (system testing),
- юнит тестирование (unit testing),
- нагрузочное тестирование (load testing),
- тестирование соответствия стандартам (conformance тестирование),
- тестирование взаимодействия (interoperability testing)

Кроме того, с учетом специфики конкретного проекта могут понадобиться дополнительные виды тестов. **СТ обеспечивает непрерывное тестирование проекта на всех этапах.**

На каждом этапе жизненного цикла проекта высока вероятность человеческой ошибки, например

- программист может допустить ошибку в коде,
- инженер по работе с данными может написать неверный SQL запрос,
- при сборке в проект может быть включена ошибочная версия библиотеки,
- разработчик машинного обучения может выбрать неправильные метрики качества для обучения модели,
- настройщик системы может установить неправильные значения переменных среды окружения.

CI/CD/СТ совместно минимизируют возможность появления ошибок в коде, ускоряют процесс разработки и развертывания решения, эффективно дополняя друг друга.

Тест

1. Что такое непрерывная интеграция (0.25)
 - a. Непрерывно работающий сервис по сбору данных от микросервисов
 - b. Автоматизированный процесс сборки общего программного решения из отдельных элементов**
 - c. Постоянный мониторинг всех процессов
 - d. Ручная сборка программного пакета
2. Как расшифровывается СТ (0.25)
 - a. Common tools
 - b. Continuous training
 - c. Continuous testing**
 - d. Cost transfer
3. Установите соответствие между CI/CD/СТ и задачами (0.25)

Автоматический запуск тестов	CI, СТ
Настройка окружения для запуска программы	CD
Проверка изменений в программном коде	CI
Создание ролей пользователя	CD

4. Что может быть триггером для запуска процесса сборки программного пакета в CI (0.25)
 - a. Публикация изменений кода в репозитории проекта**
 - b. Расписание**

с. Ручной запуск

d. Письмо от заказчика с негативным отзывом о работе системы

Итоги

В этом юните вы узнали:

Автоматизация мониторинга качества данных позволить своевременно увидеть изменения а данных, на которых работает модель. Каждая из этих мер автоматизации позволяет экономить большое количество времени, которое уходит на анализ проблемы и ее устранение. Методы обеспечения непрерывности были разделены на несколько групп, с учетом того, какие задачи эти методы решают - непрерывная интеграция (CI), непрерывная доставка (CD) и непрерывное тестирование (CT). Для каждого из этих этапов существует характерный для них перечень задач, а именно:

1. Непрерывная интеграция
 1. Проверка правильности сделанных изменений
 2. Публикация всех изменений программного кода, служебных скриптов, настроек в главном репозитории проекта
 3. Сборка всех изменений в единый проект
 4. Тестирование совместимости измененных частей проекта
 5. Проверка функциональности
2. Непрерывная доставка
 1. Настройка оборудования
 2. Настройка среды выполнения программы
 3. Установка и настройка системы
 4. Создание пользовательских ролей
 5. Организация мониторинга
3. Непрерывное тестирование
 1. Тестирование отдельных модулей (юнит тесты)
 2. Тестирование совместимости
 3. Тестирование производительности
 4. Тестирование функций

В этом юните речь шла о DevOps для проектов разработки программного обеспечения, а в следующем юните вы узнаете об отличиях в CI/CD/CT в проектах машинного обучения.

Модуль 2. Юнит 2. Задачи и инструменты CI/CD/CT в проектах машинного обучения

Введение: В этом юните мы рассмотрим особенности CI/CD/CT в проектах машинного обучения, отличающие их от обычных проектов разработки программного обеспечения.

Содержание юнита:

Давайте теперь посмотрим чем отличается CI/CD/CT в проектах машинного обучения от обычных проектов. То, что такие отличия есть, мы уже знаем из первого модуля: в проектах машинного обучения больше задач, которые надо автоматизировать (обработка данных, обучение модели), больше объектов и артефактов на каждом этапе (датасеты, конвейеры обработки данных, модели машинного обучения, среды выполнения), которыми необходимо управлять, есть другие роли в команде (инженеры и исследователи данных, инженеры и разработчики ML). Поэтому неудивительно, что CI/CD/CT в MLOps сложнее и многограннее, чем в обычных проектах.

Рассмотрим отдельные этапы, как, например, изложено в статье Google “MLOps continuous delivery and automation pipelines in ML”

https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#ml_pipeline_triggers

На этапе **разработки дизайна** модели машинного обучения решаются задачи:

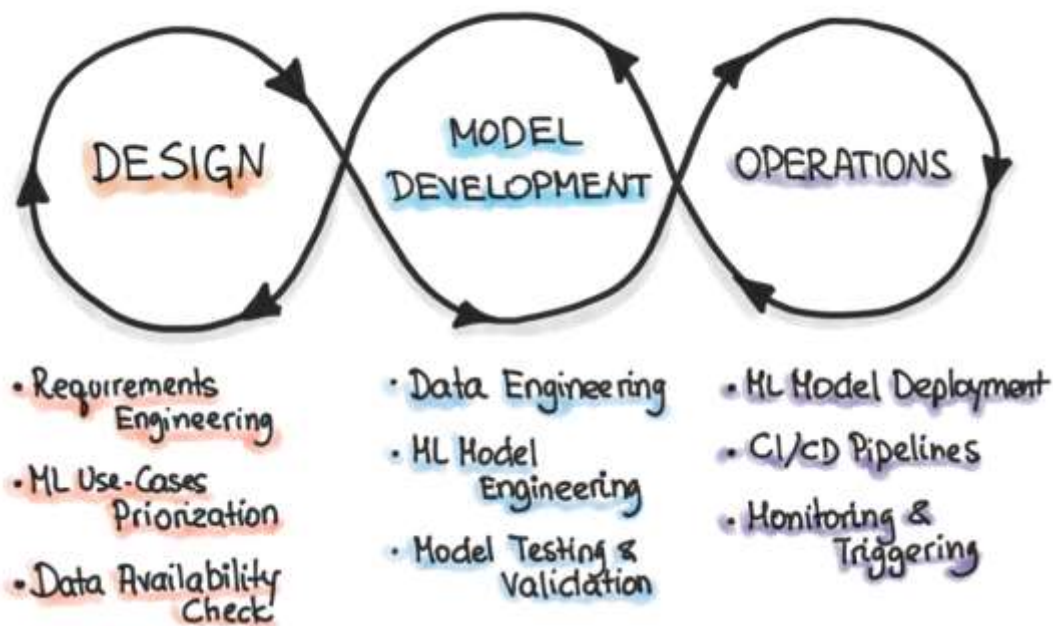
- определение требований
- анализ сценариев использования и их выгод для заказчика
- определение источников данных, их качества и доступности

На этапе **разработки модели** машинного обучения решаются задачи:

- работа с данными, обработка, выделение признаков
- разработка модели
- тестирование и валидация модели

На этапе **эксплуатации модели** машинного обучения решаются задачи:

- развертывание модели
- организация конвейера CI/CD
- мониторинг



https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#ml_pipeline_triggers

Таким образом, так же как и в любом проекте разработки программного обеспечения, проект машинного обучения начинается с фазы системного дизайна (**system design**), на котором внимательно анализируются требования заказчика, полученные в техническом задании, проверяются имеющиеся данные, делаются предположения о том, с помощью каких моделей машинного обучения можно было бы решать задачу.

Следующая фаза, разработка модели (**model development**) состоит из анализа данных, создания и обучения моделей и их валидации для возможности использования в промышленной среде.

И следующий, заключительный этап, связан с эксплуатацией модели (**operations**). На этом этапе происходит развертывание модели, выполнение логики, заложенной в пайплайны CI/CD, организация мониторинга работы модели и качества данных. **Важно, что эти фазы проходятся в проекте в цикле и на каждой фазе необходима автоматизация с использованием CI/CD/CT.**

Давайте обратим внимание на ряд отличий, которые возникают в проекте машинного обучения по сравнению с «обычным» проектом разработки программного обеспечения. **Основная причина отличий CI/CD/CT MLOps от DevOps состоит в том, что основой проекта машинного обучения является эксперимент для построения модели: с использованием имеющегося датасета для обучения создается система, умеющая делать выводы на основе изучения этого датасета.** Построенная модель должна хорошо работать не только на учебном датасете, но и в реальной среде эксплуатации, на реальных данных. Эти данные имеют свойство меняться со временем, поскольку как в природе, так и в технологических процессах нет ничего постоянного. Как следствие, качество самой хорошей модели машинного обучения начинает ухудшаться практически сразу после ее вывода в эксплуатацию. И для проекта машинного обучения быстрая повторяемость жизненного цикла проекта становится еще более актуальной, чем для разработки обычного программного обеспечения, так как на каждой итерации цикла система машинного

обучения получает новую важную информацию, которую использует для улучшения качества работы. **Следовательно, построение конвейера, обеспечивающего обучение, интеграцию, развертывание, мониторинг проекта машинного обучения, является ключевым.**

Давайте рассмотрим особенности проектов машинного обучения и то, как они влияют на CI/CD/CT:

Факт №1: Участники команды проекта ML могут не быть высоко квалифицированными программистами, поскольку сосредоточены в основном на изучении данных и эксперименте для построения эффективной модели. Это может приводить к некачественному и неэффективному коду, неправильному оформлению результатов. Такой код в будущем будет сложно поддерживать, интегрировать в другие системы, изменять и развивать.

Как это влияет на CI:

- усиленная проверка кода (pycodestyle, pylint, flake, black),
- проверка документации (pydocstyle),
- контроль версий библиотек при сборке (conda, pip, git, bitbucket),
- унификация, автоматизация и сокрытие от участников команды процесса интеграции решения (Jenkins).

Как это влияет на CD:

- обеспечить идентичность среды разработки и среды выполнения (conda, pip, docker, git, bitbucket)
- унификация, автоматизация и сокрытие от участников команды процесса интеграции решения (docker, docker-compose, kubernetes, Jenkins, ansible).

Как это влияет на CT:

- функциональное тестирование моделей, проверка качества работы на тестовых датасетах (pytest),
- тестирование устойчивости модели к шуму в данных (pytest),
- мониторинг процесса обучения, проверка переобучения модели, выбор оптимальных результатов обучения (tqdm, tensorboard)
- проверка производительности модели, скорости инференса (pytest).

Факт №2: Разработка модели машинного обучения носит экспериментальный характер, поэтому для получения наиболее эффективной модели приходится пробовать разные методы, конфигурации, алгоритмы, пытаться найти их оптимальное сочетание.

Как это влияет на CI:

- обеспечить управление версиями изменяющихся сущностей для того, чтобы была возможность определить наиболее подходящий вариант, обеспечив возможность его быстрого воспроизведения, максимально используя уже созданное (git, bitbucket, dvc)
 - написанный код,
 - вычисленные параметры модели,
 - найденные оптимальные сочетания версий библиотек.

Как это влияет на CD:

- быстрая доставка измененной модели в промышленную эксплуатацию для получения быстрой обратной связи (Jenkins, ansible, docker, docker-compose),
- быстрый возврат к предыдущей рабочей версии (Jenkins, ansible, docker, docker-compose)
- автоматизация развертывания, настройка среды выполнения, установка библиотек для обеспечения идентичности среды исполнения модели (Jenkins, ansible, docker, docker-compose)

Как это влияет на СТ:

- автоматизация обучения модели и тестирования для ускорения процесса и реализации как можно большего числа повторений цикла обучения модели (pytest).

Факт №3: К набору тестов необходимо добавить проверку данных, оценку и проверку моделей, тестирование среды выполнения программы, проверку совместимости библиотек.

Как это влияет на СІ:

- тестирование данных (скрипты автоматизации на python, bash),
- проверка запросов к данным (встроенные средства СУБД, скрипты автоматизации),
- данные являются неотъемлемой частью текущей версии (ветки) проекта, необходимо обеспечить взаимосвязь протестированных данных, модели, версий библиотек и настроек среды (git, bitbucket, dvc)

Как это влияет на СD:

- в промышленном окружении устанавливать средства для мониторинга поступающих данных (скрипты автоматизации, Grafana)

Как это влияет на СТ:

- постоянно контролировать качество работы модели и качество данных (скрипты автоматизации, Grafana).

Факт №4: Развертывание модели машинного обучения может потребовать больше шагов от MLOps специалиста, поскольку для обеспечения повторяемости работы модели в промышленной среде необходимо скрупулезно по шагам настроить эту среду.

Как это влияет на СІ:

- При сохранении ветки системы требуется запоминать информацию о настройке среды для развертывания (git, bitbucket, docker, pip, conda).

Как это влияет на СD:

- Автоматизация шагов по доставке решения, обеспечивающая идентичность сред для обучения и промышленной эксплуатации (git, butbucket, docker, pip, conda).

Как это влияет на СТ:

- Тестирование правильности настройки среды промышленной эксплуатации (скрипты автоматизации, pip, conda).

Факт №5: При эксплуатации системы машинного обучения необходимо иметь возможность постоянно мониторить качество работы как самой модели, так и данных, на которых она работает.

Как это влияет на СІ:

- Добавить возможность быстрого внесения изменений и сборки, либо отката к предыдущей рабочей версии, при обнаружении каких-либо проблем в промышленной среде (git, bitbucket, Jenkins)

Как это влияет на СD:

- установка средств мониторинга и контроля работы модели, данных, ресурсов (prometheus, Grafana, Kibana),
- добавить возможность быстрых изменений и сборки, либо отката к предыдущей рабочей версии, при обнаружении каких-либо проблем в промышленной среде (git, Jenkins)

Как это влияет на СТ:

- постоянный мониторинг и тестирование работы модели в промышленной среде (prometheus, Grafana, Kibana)
 - скорость работы модели
 - устойчивость результатов
 - отслеживание аномальных значений (выбросов)

- занятие аппаратных ресурсов
- отклонения в поступающих данных
- мониторинг аппаратного обеспечения (htop, syslog, Grafana, zabbix):
 - загрузка CPU/GPU,
 - расходование оперативной памяти,
 - объем свободного пространства на жестком диске,
 - наличие системных уведомлений об ошибках

Итак, CI/CD/CT MLOps весьма схожи с аналогичными задачами в DevOps, однако более многогранные и сложные, ввиду особенностей проекта машинного обучения.

Тест

1. Что отличает MLOps CI от обычного проекта (0.25)
 - a. Наличие датасетов**
 - b. Наличие моделей машинного обучения**
 - c. Наличие комментариев в коде
 - d. Повышенные требования к производительности системы
2. Что отличает MLOps CD от обычного проекта (0.25)
 - a. Необходимость более тщательно настраивать среду выполнения программы для более качественной работы модели (инференса)**
 - b. Необходимость проверять качество данных, поступающих при эксплуатации системы, на которых работает модель**
 - c. Повышенные требования к работе аппаратного обеспечения
 - d. Большее количество ролей участников в проекте
3. Какой еще смысл возникает у термина CT, который означает Continuous Testing, применительно к проектам машинного обучения (0.25)
 - a. Common Thinking
 - b. Collaborative Testing
 - c. Continuous Training**
 - d. Нет никаких изменений у термина CT
4. Какие параметры важно поставить под мониторинг при развертывании и эксплуатации модели машинного обучения
 - a. Оперативную память системы**
 - b. Скорость работы модели**
 - c. Качество поступающих данных**
 - d. Температуру окружающей среды

Итоги

Мы изучили особенности CI/CD/CT для проектов машинного обучения. Вот наиболее важные отличия CI/CD/CT MLOps от обычного проекта:

- В MLOps CI задачи интеграции применяются и к новым объектам и артефактам, относящимся к проектам машинного обучения: данным, моделям, пайплайнам, средам исполнения.
- MLOps CD это не просто задача установки пакета программного обеспечения или сервиса, а система последовательных операций (конвейер машинного обучения, ML

training pipeline), являющаяся неотъемлемой частью проекта ML, обеспечивающая его успех,

- в MLOps CT появляется новая сущность, уникальная для ML, суть ее в автоматическом переобучении в процессе работы моделей машинного обучения на новых данных.

В следующем юните мы познакомимся с популярным инструментом Jenkins для решения задач CI/CD/CT.

Модуль 2. Юнит 3. Знакомство с Jenkins. Установка.

Введение: В этом юните мы познакомимся с Jenkins, позволяющим решать задачи CI/CD/CT, это позволит быстрее переходить к практическим задачам. Подробно описана процедура установки Jenkins.

Содержание юнита:

Итак, приступим к изучению Jenkins, очень популярного инструмента для автоматизации различных операций, возникающих в CI/CD/CT. Как обычно, у любого популярного инструмента есть объяснения этой популярности. Jenkins написан на широко распространенном языке программирования Java, является бесплатным и имеет большое сообщество, которое занимается его развитием, созданием лучших практик использования. Jenkins можно применять практически на всех этапах жизненного цикла проекта.

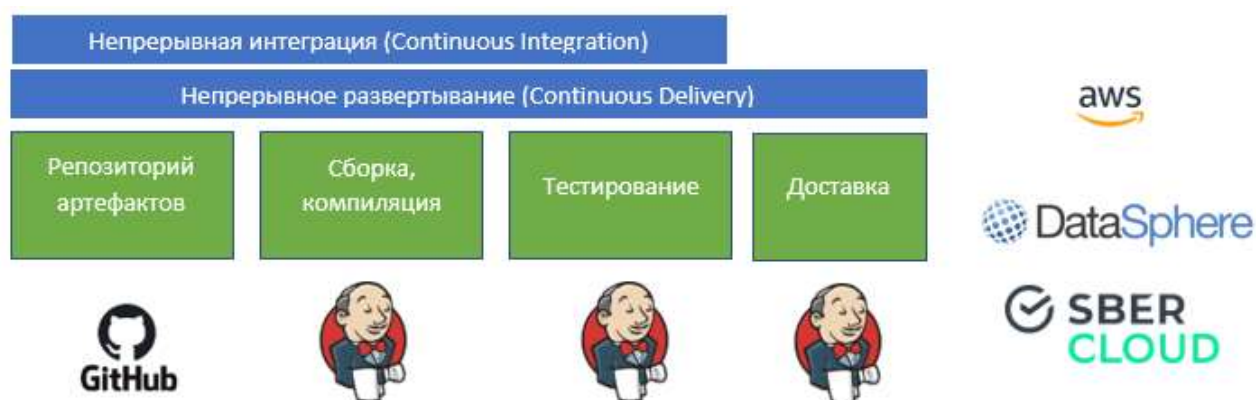


Рисунок “Место Jenkins в этапах проекта разработки программного обеспечения”.

Справедливости ради необходимо отметить, что у Jenkins есть альтернативы: Bamboo (Atlassian), Circle CI, Team City, GitLab CI/CD, Travis, Harness. Jenkins не является стандартом “де-факто”, имеющим огромные преимущества перед другими системами.

Установка Jenkins выполняется очень просто и подробно описана на сайте <https://www.jenkins.io/>. На этом сайте можно найти последнюю версию LTS и обновления weekly. Там же можно найти требования к системе, на которой устанавливается Jenkins <https://www.jenkins.io/doc/book/installing/linux/>. Рекомендуется перед началом работы внимательно ознакомиться с этими требованиями. Есть особенности установки для различных операционных систем. Также важно знать, что работа Jenkins поддерживается только с Java 8.

Далее мы будем рассматривать практическую работу с linux системой. Рекомендуем вам, параллельно изучению модуля, повторять описанные шаги на вашем устройстве.

Выяснить какая операционная система используется можно с помощью команды `cat /etc/os-release`

```
ubuntu@ip-172-31-95-38:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
ubuntu@ip-172-31-95-38:~$ █
```

Проверить текущую версию Java можно так:

java -version

Если Java не установлена, то выполнить установку можно следующим образом (для выполнения *sudo* нужны права суперпользователя):

sudo apt-get install openjdk-8-jre

Если установщик *apt-get* не может найти соответствующий пакет, то рекомендуется выполнить его обновление

sudo apt-get update

Перед установкой необходимо добавить ключ в систему

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Затем добавить информацию в установщик apt

```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

Обновить установщик

sudo apt-get update

И, наконец, установить Jenkins

sudo apt-get install jenkins

После успешного окончания установки можно проверить статус процесса с использованием команды

sudo service jenkins status

```
ubuntu@ip-172-31-95-38:~$ sudo service jenkins status
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: active (running) since wed 2022-03-16 10:17:33 UTC; 1min 6s ago
     Main PID: 5263 (java)
        Tasks: 33 (limit: 1147)
       Memory: 285.4M
```

Полезная информация: для обновления версии Jenkins в linux системе достаточно просто обновить war файл и перезапустить сервис.

```
ubuntu@ip-172-31-0-20:~$ sudo ls -la /usr/share/jenkins/
total 70564
drwxr-xr-x  2 root root    4096 Dec 19 06:37 .
drwxr-xr-x 123 root root    4096 Dec 19 06:47 ..
-rw-r--r--  1 root root 72247484 Dec  1 17:18 jenkins.war
ubuntu@ip-172-31-0-20:~$ █
```

По умолчанию Jenkins начнет работать как web сервер на порте 8080.

Для начала работы с Jenkins в графическом web-интерфейсе необходимо через браузер зайти на сервер, на котором запущен Jenkins, на номер порта 8080. После этого в окне

браузера появится следующее сообщение о необходимости разблокировать (unlock) Jenkins для начала работы.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

В соответствии с этой информацией необходимо скопировать текст с сервера из файла **`sudo vi /var/lib/jenkins/secrets/initialAdminPassword`** и подставить его в поле для ввода текста **“Administrator password”**. Это действие выполняется только один раз перед началом работы.

Тест

1. Для автоматизации каких задач предназначен Jenkins? (0.25)
 - a. публикация изменений кода в репозиторий проекта
 - b. разработка программного обеспечения
 - c. **тестирование изменений**
 - d. доставка системы в среду выполнения
2. Какая библиотека необходима для работы Jenkins? (0.25)
 - a. python
 - b. Java 1
 - c. **Java 8**
 - d. jenkins-lib
3. Какой командой в linux можно проверить статус работы сервиса Jenkins? (0.25)
 - a. get status jenkins
 - b. **sudo service jenkins status**
 - c. select jenkins status
 - d. which status for jenkins
4. На каком номере порта запускается web-интерфейс Jenkins? (0.25)
 - a. 1
 - b. 80
 - c. **8080**
 - d. 80808

Итоги

Вы узнали для чего используется популярный инструмент Jenkins и каким образом осуществить его установку в операционной системе linux. Далее мы рассмотрим практическое применение Jenkins для автоматизации задач проекта.

Модуль 2. Юнит 4. Использование Jenkins

Введение: В этом юните вы узнаете, как использовать Jenkins для задач CI/CD/CT.

Содержание юнита:

Итак, вы успешно осуществили процедуру установки Jenkins и открыли web-интерфейс в браузере для работы с Jenkins через графический интерфейс. Сначала вы увидите вот такую форму, через которую Jenkins сразу же предложит себя усовершенствовать (кастомизировать)

Getting Started

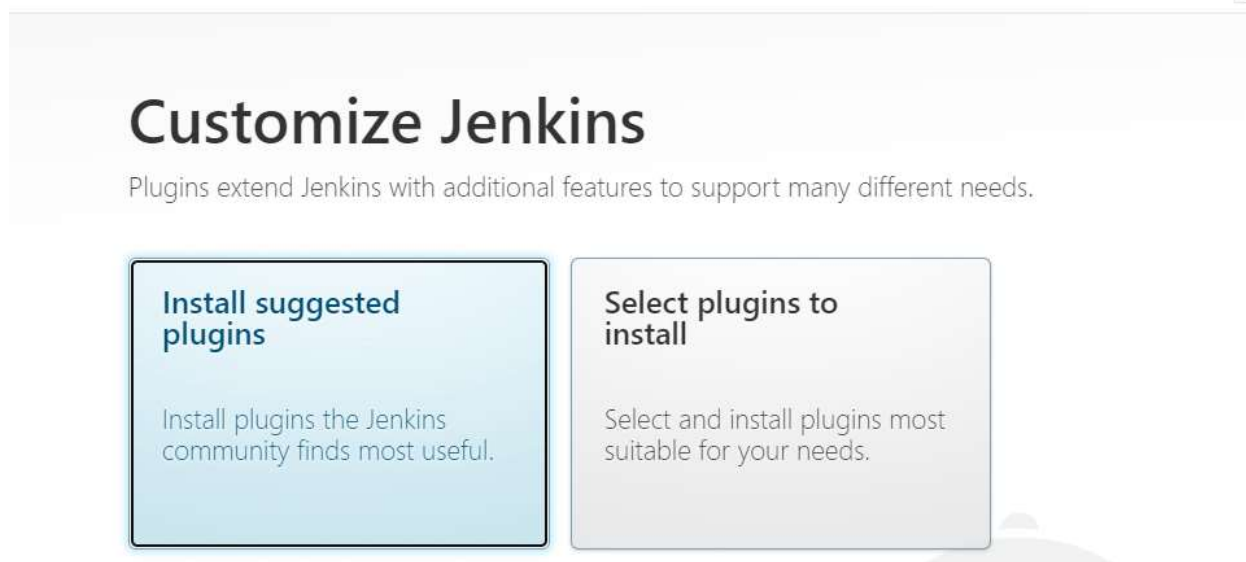


Рисунок “Стартовое окно для начала работы с Jenkins”.

Настройка Jenkins осуществляется через так называемые “плагины” (plugins). **Плагины Jenkins это своего рода «конструктор lego», внутри Jenkins по умолчанию мало функций, поэтому нужные для конкретного проекта или задачи функции добавляются через специальные плагины.** Пользователи могут создавать свои плагины, однако следует помнить, что уже созданных плагинов очень много. Для решения почти любой задачи уже, как правило, существует несколько плагинов, созданных разными авторами. На практике пользуются рекомендованными, наиболее популярными, либо привычными и проверенными плагинами.

Рассмотрим процесс установки плагинов. Например, вы можете выбрать установку рекомендованных плагинов (Install suggested plugins), либо самостоятельно выбирать те плагины, которые вам нужны (Select plugins to install). Если вы выберете установку рекомендованных плагинов, то автоматически запустится установка.

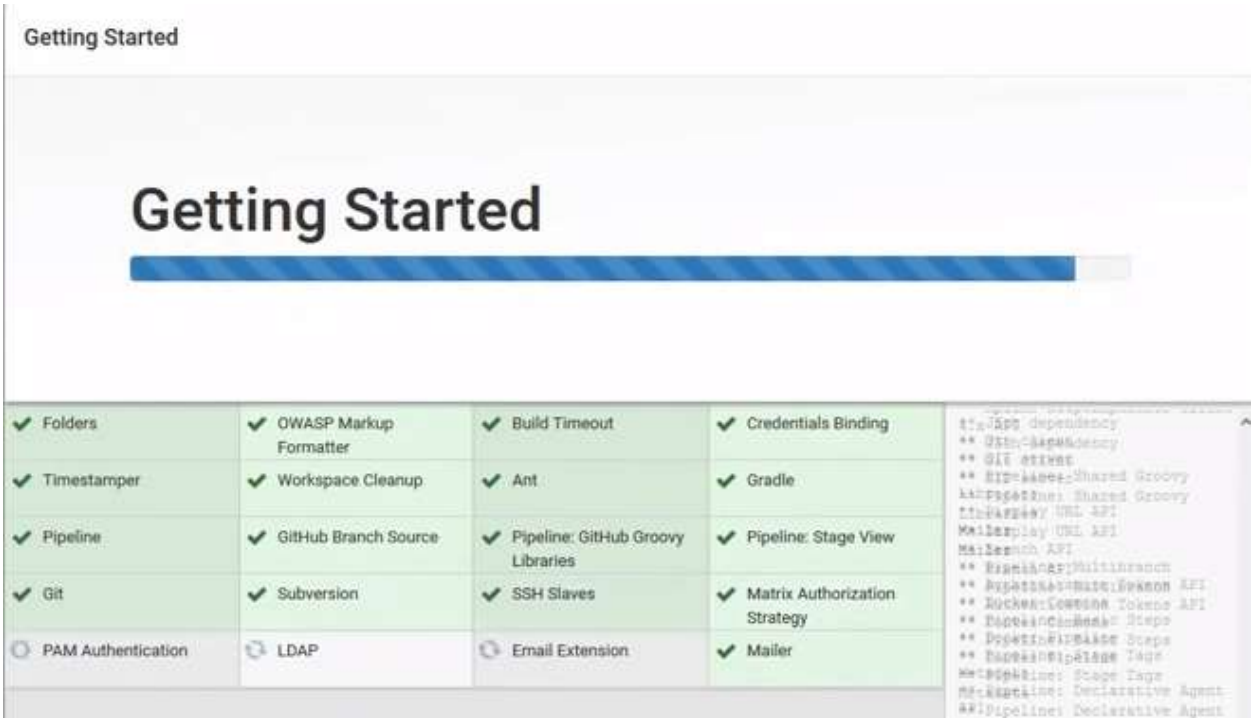


Рисунок “Процесс установки рекомендованных плагинов, установленные плагины отмечены зеленым цветом”.

После этого Jenkins предложит вам создать пользователя для работы в системе.



Рисунок “Окно создания пользователя Jenkins”

После того как пользователь создан можно переходить к работе в Jenkins.

Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins

Рисунок “Приглашение к работе”

После нажатия кнопки запуска консоли управления Jenkins “Start using Jenkins” открывается рабочее пространство для работы в Jenkins.

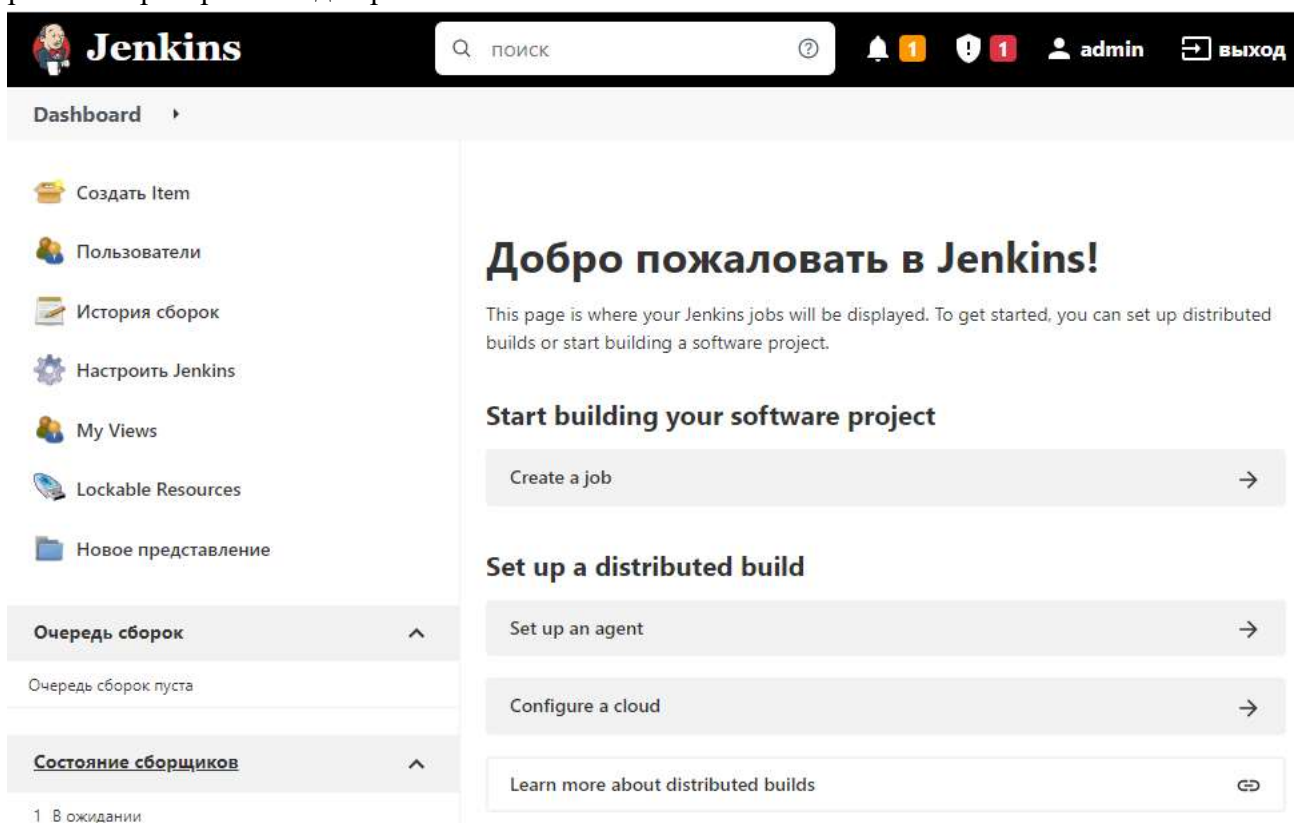
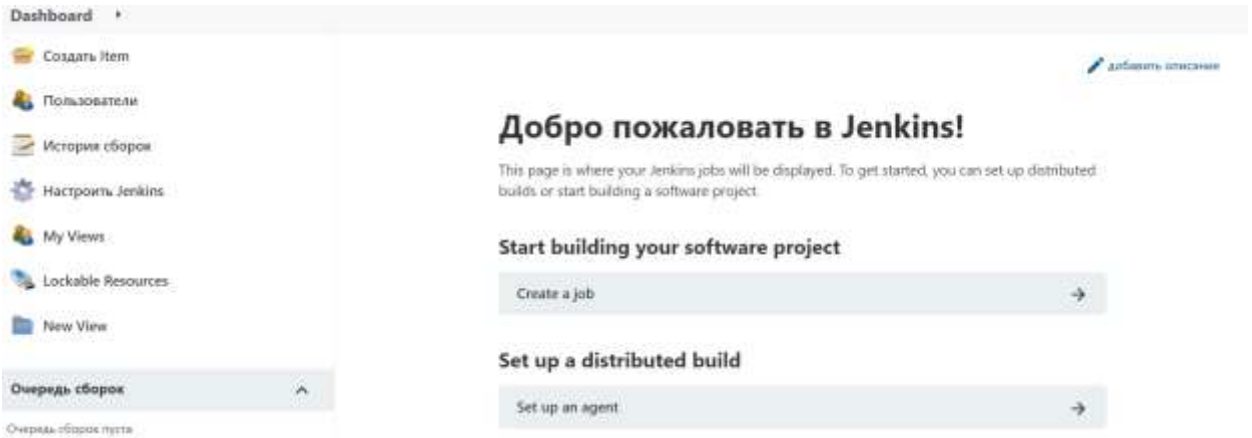
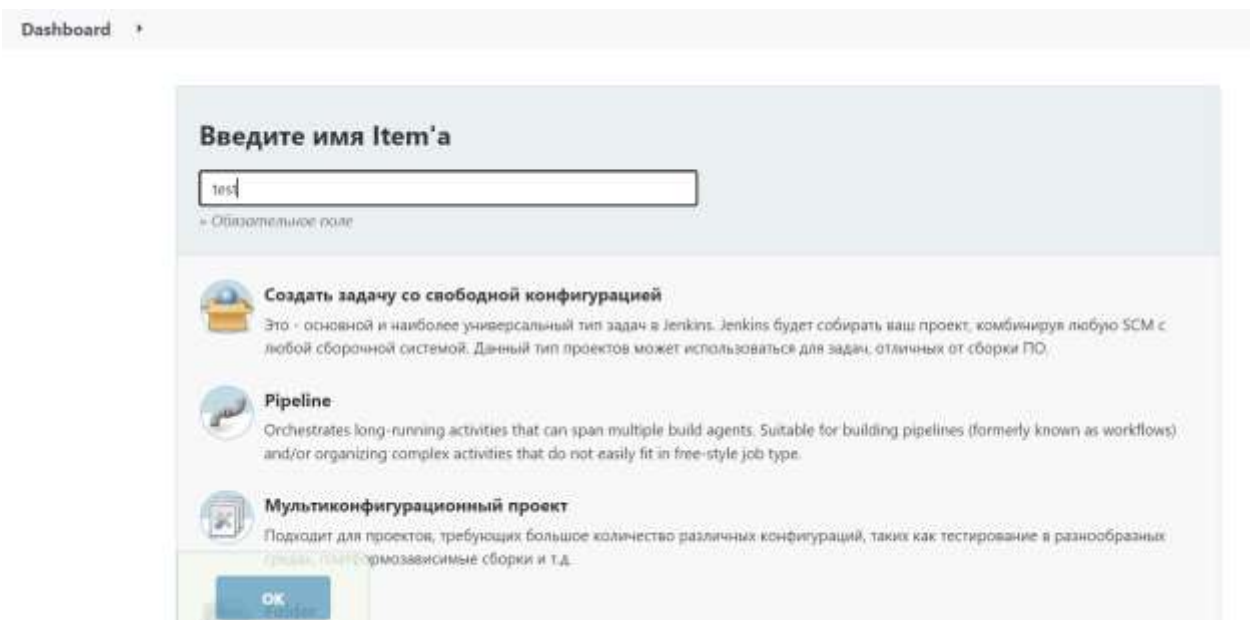


Рисунок “Рабочая консоль Jenkins”.

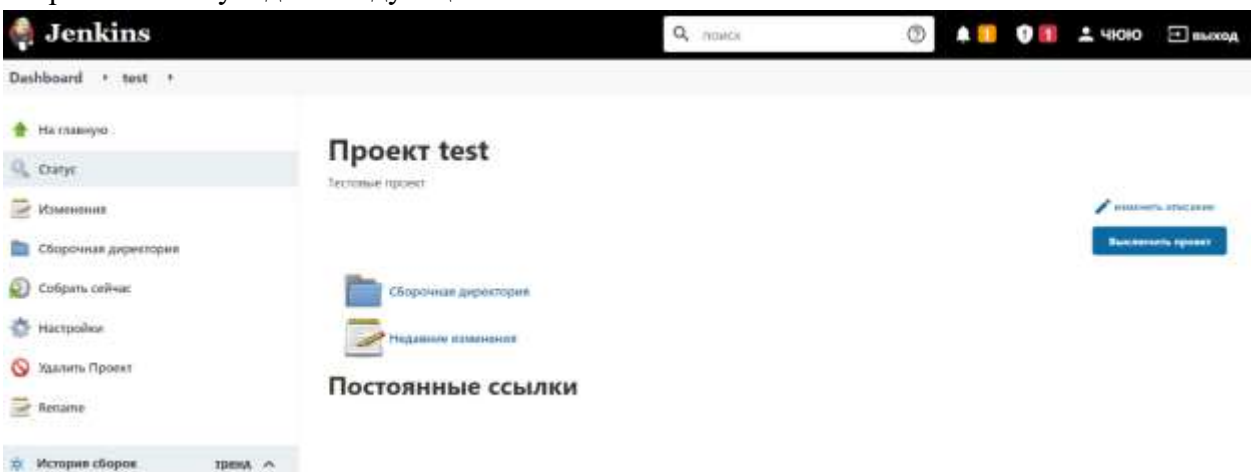
Давайте создадим первую **тестовую сборку (job)**. Термин job очень распространен в документации Jenkins, поэтому даже в русскоязычной документации используют термин “джоб”, хотя его использование носит неформальный характер. Также иногда для объекта job используют перевод “задача” или “сборка”. Далее мы будем применять термин “сборка”. Для создания сборки нажимаем на пункт “Создать Item”



и в открывшейся форме задаем имя и выбираем опцию “Создать задачу со свободной конфигурацией” после чего нажимаем кнопку “ОК”.



После этого выбираем опции управления проектом на разных этапах (чем больше плагинов, тем больше опций) и сохраняем проект. На этом этапе будут предложены различные опции для проекта, их можно пропустить, необходимые опции можно добавить потом. После сохранения мы увидим следующее окно.



Итак, у нас создан первый проект в Jenkins.

Теперь давайте посмотрим на те возможности, которые у нас есть в главном окне управления Jenkins.

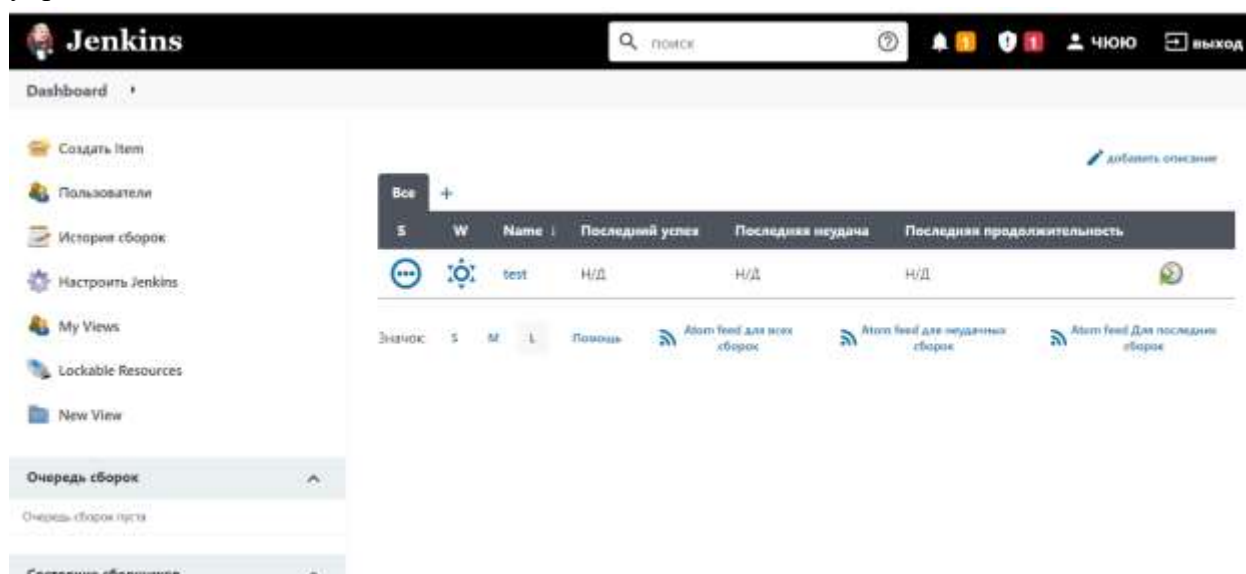


Рисунок “Главная панель для работы с Jenkins”.

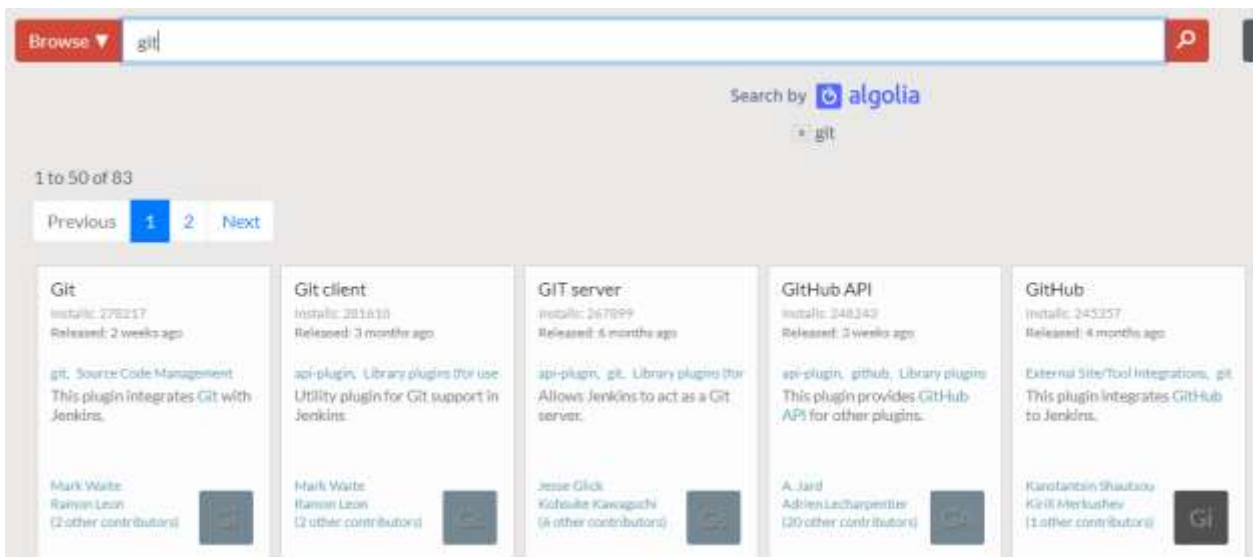
Для нас доступны следующие важные действия:

1. Создать Item - создание новых сущностей, например, задач (jobs)
2. Пользователи - управление пользователями
3. История сборок - работа со сборками
4. Настроить Jenkins - системные настройки Jenkins
5. MyViews - управление визуализациями.

В окне “Очередь сборок” показывается состояние сборок (jobs). На основе этой информации можно определить количество одновременно выполняемых сборок (jobs), в соответствии с имеющимися ресурсами.

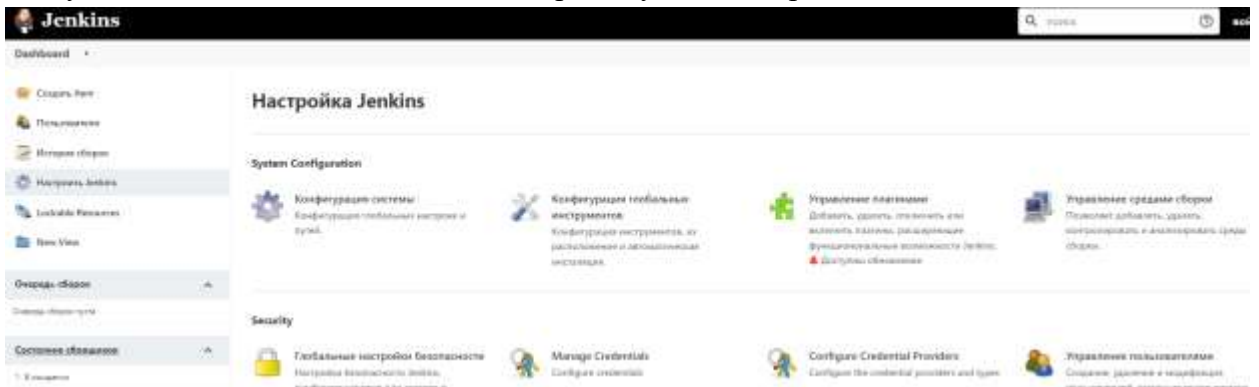
Можно к главному серверу Jenkins (master) добавить вспомогательные серверы (slaves, agents), на которых будут выполняться сборки (jobs)

Для выполнения различных практических задач нам понадобятся плагины. Все дополнительные функции в Jenkins реализуются через плагины. Например, есть специальный плагин для выгрузки кода проекта из git репозитория. Плагины можно найти на сайте <https://plugins.jenkins.io/>



На практике лучше выбирать самый популярный плагин, статистика его использования показывается на карточке плагина.

Для установки плагина необходимо выбрать пункт “Настроить Jenkins”



После этого необходимо выбрать нужный плагин и установить его.

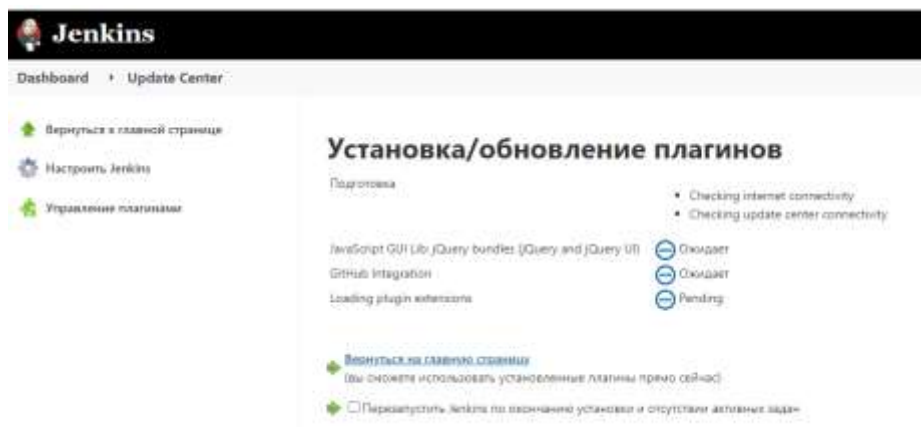


Рисунок “Установка и обновление плагинов”.

Теперь давайте рассмотрим более подробно создание тренировочной, “игрушечной” сборки (job). Перейдя на главную панель Jenkins мы увидим перечень всех сборок, которые мы подготовили. У нас пока только одна сборка с именем “test”.

S	W	Name	Последний успех	Последняя неудача	Последняя продолжительность
		test	55 секунд #1	Н/Д	74 ms

Значок: S M L Помощь Atom feed для всех сборок

Рисунок “Перечень доступных сборок Jenkins”

Нажав на выбранную нами сборку мы получим возможность ее редактирования, в том числе описания различных действий по автоматизации, для этого надо выбрать пункт “Настройки”.

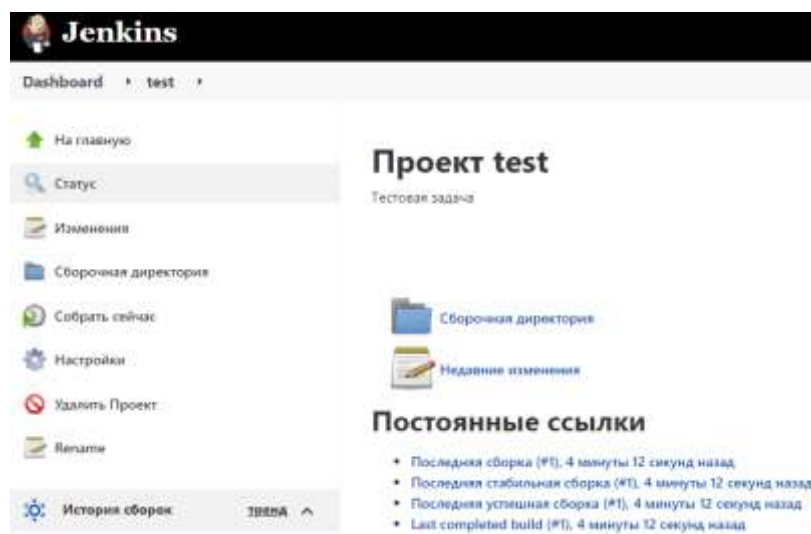


Рисунок “Настройка сборки (job)”

После этого мы увидим множество различных опций

- GitHub project
- This build requires lockable resources
- Throttle builds ?
- Удалять устаревшие сборки ?
- Это - параметризованная сборка ?
- Приостановить сборки ?
- Разрешить параллельный запуск задачи ?

Управление исходным кодом

Нет
 Git ?

Триггеры сборки

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?
- Запускать периодически ?
- GitHub hook trigger for GITScm polling ?
- Опрашивать SCM об изменениях ?

Среда сборки

- Delete workspace before build starts
- Use secret text(s) or file(s) ?
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- With Ant ?

Сборка

Добавить шаг сборки ▾

Послесборочные операции

Добавить шаг после сборки ▾

Можно пропустить все пункты, добавить только выполнение shell команд на этапе «Сборка».

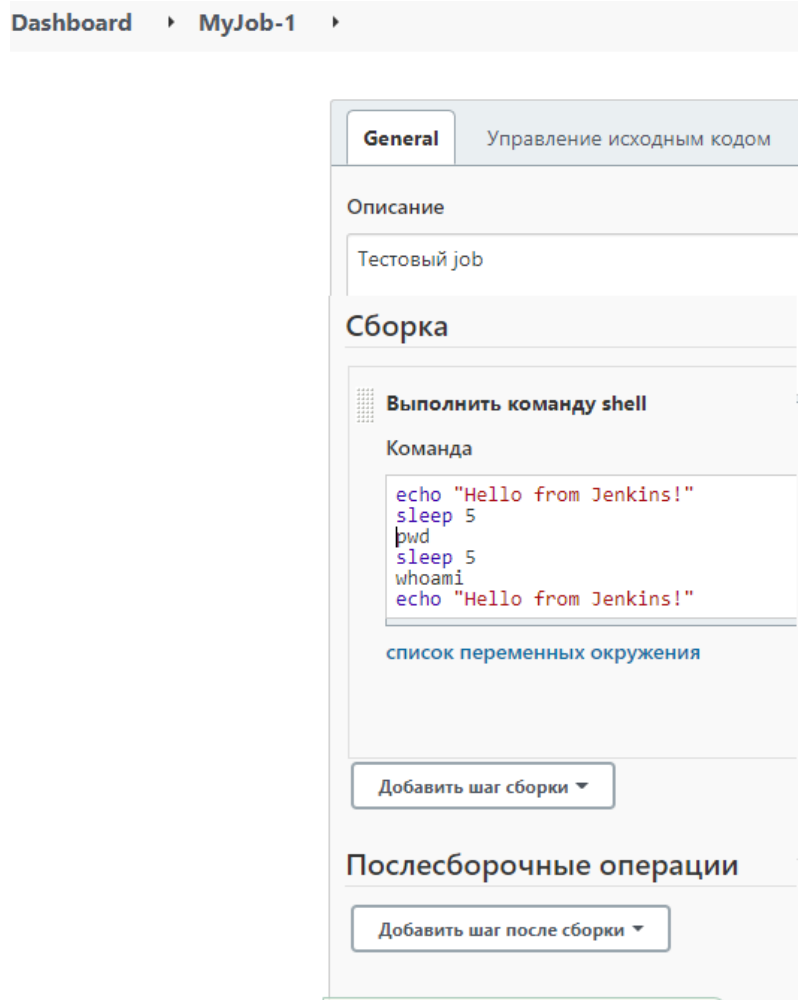
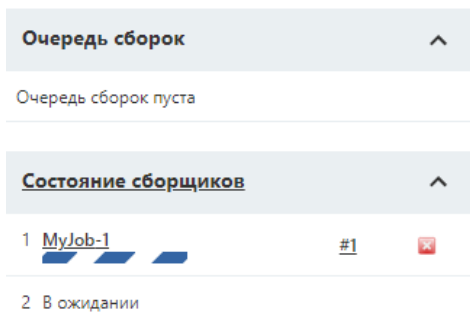


Рисунок “Создание простейшей автоматизации в сборке Jenkins”

После этого новый job появится в списке и мы можем запустить его.



В окне “Очередь сборок” мы можем видеть, что сборка начала выполняться.



Для того, чтобы увидеть результаты работы shell скрипта, который пишет информацию на стандартный вывод системы, нам необходимо открыть “Вывод консоли”:



Рисунок “Вывод консоли”

После этого мы можем увидеть результаты выполнения скрипта автоматизации

Вывод на консоль

```

Started by user unknown or anonymous
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/MyJob-1
[MyJob-1] $ /bin/sh -xe /tmp/jenkins1954891194487289511.sh
+ echo Hello from Jenkins!
Hello from Jenkins!
+ sleep 5
+ pwd
/var/lib/jenkins/workspace/MyJob-1
+ sleep 5
+ whoami
jenkins
+ echo Hello from Jenkins!
Hello from Jenkins!
Finished: SUCCESS

```

Можно запустить несколько раз и увидим несколько в очереди выполнения. Количество сборок (jobs) в очереди и в текущей обработке можно настраивать.



Рисунок “Несколько сборок (jobs) в ожидании в очереди выполнения”
Общая история сборок находится здесь:

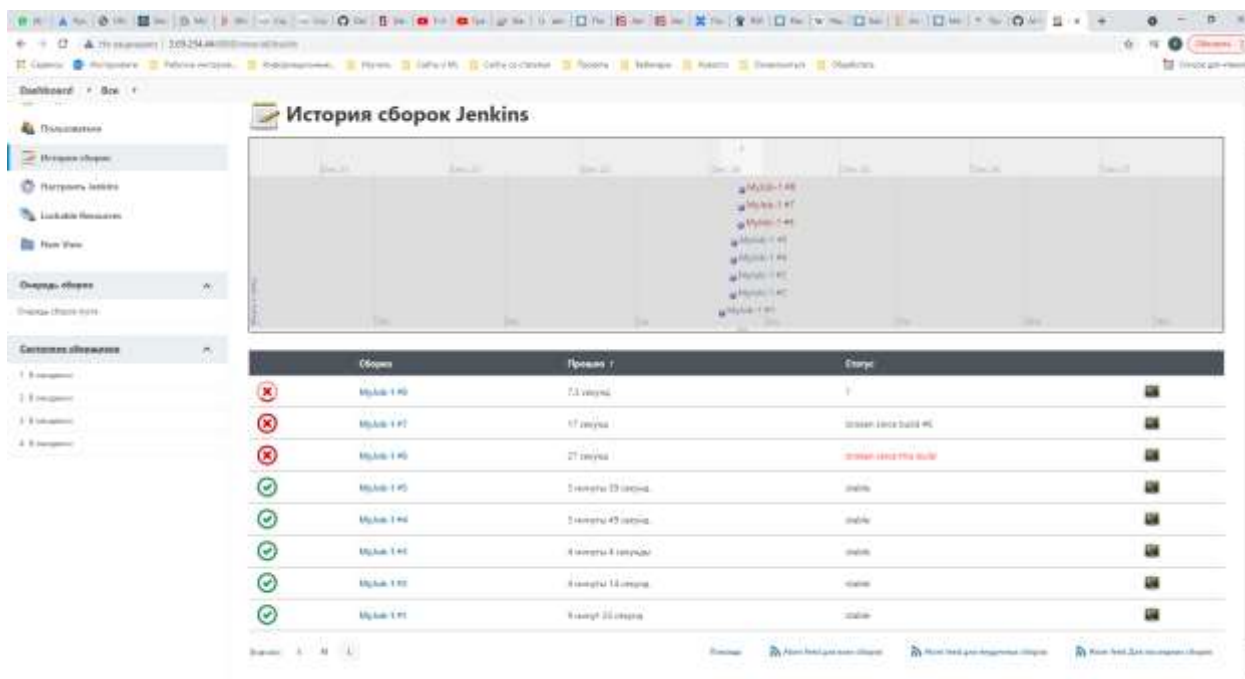


Рисунок “Общая история сборок Jenkins”

Итак, мы смогли создать сборку (job) в Jenkins.

В заключении этого юнита важный практический совет который, к сожалению, на первых порах будет очень полезен. Если вы вдруг забыли пользовательский пароль от Jenkins, то вы можете исправить ситуацию, выполнив следующие действия:

остановить службу

sudo service jenkins stop

скорректировать параметр useSecurity в файле /var/lib/jenkins/config.xml

запустить службу Jenkins

sudo service jenkins start

После этого можно зайти в систему без пароля и установить для будущего использования новый пароль.

Тест

1. Как называется инструмент для настройки отдельных функций в Jenkins? (0.25)
 - a. модуль
 - b. плагин**
 - c. библиотека
 - d. пакет
2. Какой термин используется для job в русскоязычном переводе? (0.25)
 - a. работа
 - b. рабочая функция
 - c. сборка**
 - d. проект
3. Какие лучше использовать плагины в Jenkins? (0.25)
 - a. коммерческие, чем дороже, тем лучше
 - b. самые популярные**

- c. **рекомендованные по умолчанию**
 - d. **любые**
4. Как можно увидеть статус выполнения сборки? (0.25)
- a. ps -a
 - b. sudo ps -a
 - c. в диспетчере задач
 - d. **в окне “Очередь сборок”**

Итоги

Мы научились запускать и настраивать Jenkins.

Дальше мы используем эти знания для автоматизации задач в проекте машинного обучения.

Модуль 2. Юнит 5. Пример практического применения Jenkins для проекта машинного обучения

Введение: В этом юните мы применим Jenkins для автоматизации простого проекта машинного обучения, включая: получение данных, обучение модели, использование модели для предсказания.

Содержание юнита:

Создание сборки (job) для проекта машинного обучения аналогично уже описанному в предыдущем юните процессу, за исключением того, что отдельными блоками автоматизации будут типовые задачи проекта машинного обучения.

Для того, чтобы практически разобраться в этом вопросе давайте рассмотрим автоматизацию с помощью Jenkins простых операций, которые входят в конвейер проекта машинного обучения:

- Начинается процесс с получения данных, в нашем случае мы будем использовать данные из библиотеки `catboost.datasets.titanic` с информацией о пассажирах “Титаника” (известная учебная задача с `kaggle.com` “Titanic Disaster”), для данных мы проведем необходимую предобработку.

Артефакт этапа: подготовленные тренировочный и тестовый датасеты в формате `csv`, записанные на диск.

- Следующим шагом является построение и обучение классификатора пассажиров, который будет предсказывать выжил пассажир или нет. Для этого мы используем простой классификатор логистической регрессии из модуля `sklearn.linear_model.LogisticRegression`, который обучается на данных предыдущего этапа.

Артефакт этапа: обученный классификатор, выгруженный в формате `pickle`.

- Для предсказания на новых данных мы загрузим обученный классификатор из внешнего файла в формате `pickle` и применим его для тестовых данных.

Артефакт этапа: предсказание для первого пассажира из тестового набора.

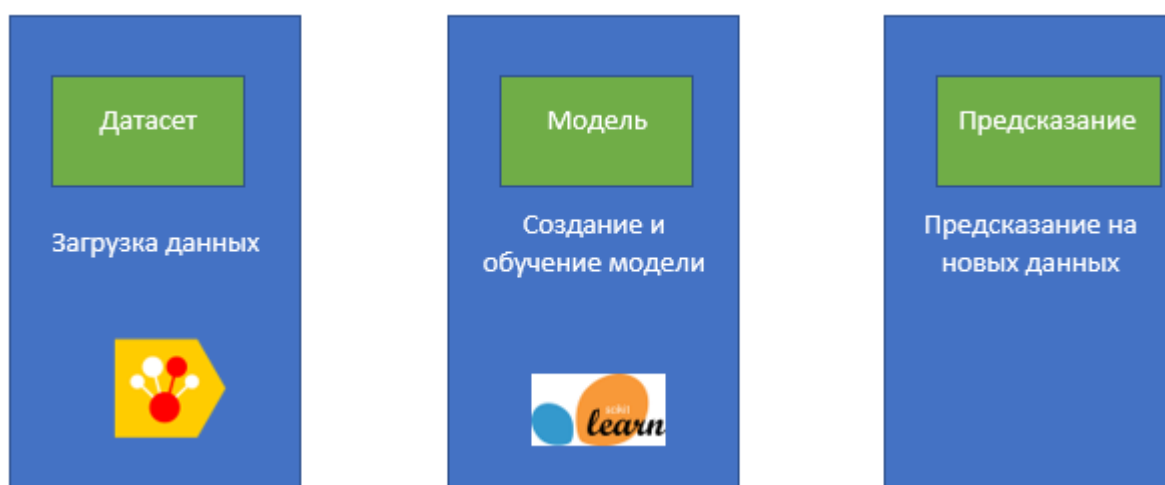


Рисунок “Описание автоматизируемых операций”

Каждый из этих отдельных этапов мы оформим в виде отдельного файла *.ру:

- create_dataset.py,
- train_model.py,
- make_prediction.py.

Содержимое файла create_dataset.py

```

from catboost.datasets import titanic
import pandas as pd

# Загрузка данных
train, test = titanic()

# обработка данных
# заполним данные о поле пассажира числовыми данными (0 или 1) вместо текстовых ('male' или 'female')
train['Sex'] = train['Sex'].apply(lambda x: 0 if 'male' else 1)
test['Sex'] = test['Sex'].apply(lambda x: 0 if 'male' else 1)

# в признаке "Возраст" много пропущенных (NaN) значений, заполним их средним значением возраста
train['Age'] = train['Age'].fillna(train.Age.mean())
test['Age'] = test['Age'].fillna(train.Age.mean())

# запишем созданные датасеты во внешние csv файлы
train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Survived']].to_csv('data_train.csv', index=False)
test[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].to_csv('data_test.csv', index=False)

```

Содержимое файла train_model.py

```

import pandas as pd

# прочитаем из csv файла подготовленный датасет для обучения
data_train = pd.read_csv('data_train.csv')
X_train = data_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].values
y_train = data_train['Survived'].values

```

```
# загрузим модель машинного обучения
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=100_000).fit(X_train, y_train)

# сохраним обученную модель
import pickle

pickle.dump(model, open('model.pkl', 'wb'))
```

Содержимое файла `make_prediction.py`

```
import pickle
import pandas as pd

loaded_model = pickle.load(open('model.pkl', 'rb'))

# прочитаем из csv файла подготовленный датасет для обучения
data_test = pd.read_csv('data_test.csv')
X_test = data_test[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].values

# сделаем предсказание для первого пассажира из тестовой выборки
print(loaded_model.predict(X_test[0:1]))
```

Итак, нам надо последовательно запускать три python скрипта, каждый будет сохранять артефакты (результаты работы) на диск для использования следующим скриптом.

Давайте автоматизируем это в Jenkins. Договоримся, что файлы скриптов будут расположены в установленной нами папке проекта, например `/home/ubuntu/project/`. В этом случае мы можем выполнять python скрипты с помощью следующей команды

```
python3 /home/ubuntu/project/script.py
```

Если вы вдруг забыли установить интерпретатор python на рабочий сервер, на котором мы планируем выполнять скрипты, то сделать это можно командой

```
sudo apt install python3.9
```

Также нам понадобятся различные библиотеки python, которые можно установить с помощью установщика пакетов python pip. Если установщик pip отсутствует на вашем сервере, то его надо установить командой

sudo apt-get install python3-pip

После этого с помощью pip можно легко устанавливать новые библиотеки, например, catboost и scikit-learn

pip install catboost

pip install scikit-learn

Перейдем к автоматизации процесса с помощью простой сборки Jenkins. Для этого создадим новый Item со свободной конфигурацией, дадим ему имя, например “Simple ML pipeline”, и добавим выполнение shell скриптов на этапе Сборки (Build). В окне для написания shell команд пока напишем простой набор команд как на рисунке ниже и нажмем кнопку “Сохранить”.

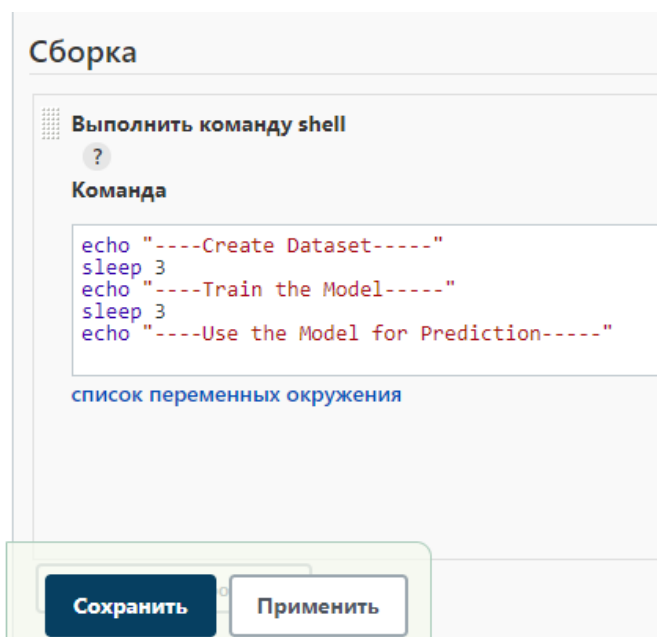


Рисунок “shell команды для выполнения на этапе сборки”


У нас появился новый проект “Simple ML pipeline” в котором выполняются заданные shell команды. Давайте попробуем запустить и посмотреть результат. Для запуска необходимо перейти на главную рабочую панель и нажать зеленый треугольник справа для нужной сборки

S	W	Name	Последний успех	Последняя неудача	Последняя продолжительность	
✓	⚙	MyJob-1	1 час 19 минут #6	Н/Д	10 секунд	▶
⚙	⚙	Simple ML pipeline	Н/Д	Н/Д	Н/Д	▶
✓	⚙	test	3 дня 22 часов #1	Н/Д	74 мс	▶

Экран: 3 M L

Панель: [Узел test1 для всех сборок](#) [Узел test1 для ежедневной сборки](#) [Узел test1 для последней сборки](#)

После этого в окне “Состояние сборщиков” мы увидим статус выполнения программы

<u>Состояние сборщиков</u>		
1	Simple ML pipeline	#1 
2	В ожидании	

и в консоли можно увидеть то, что выполнял описанный нами shell скрипт.

Вывод на консоль

```
Started by user unknown or anonymous
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Simple ML pipeline
[Simple ML pipeline] $ /bin/sh -xe /tmp/jenkins9220191282695584849.sh
+ echo ----Create Dataset-----
----Create Dataset-----
+ sleep 3
+ echo ----Train the Model-----
----Train the Model-----
+ sleep 3
+ echo ----Use the Model for Prediction-----
----Use the Model for Prediction-----
Finished: SUCCESS
```

Теперь давайте добавим вызов python-скриптов, которые выполняют описанные нами задачи по созданию датасета, обучению модели и ее применению для предсказания на тестовых данных. Для этого изменим содержание команд shell следующим образом

```
echo "----Create Dataset (begin)-----"
python3 /home/ubuntu/project/create_dataset.py
echo "----Create Dataset (end)-----"
echo "----Train the Model (begin)-----"
python3 /home/ubuntu/project/train_model.py
echo "----Train the Model (end)-----"
echo "----Use the Model for Prediction (begin)-----"
python3 /home/ubuntu/project/prediction.py
echo "----Use the Model for Prediction (begin)-----"
```

В графическом интерфейсе Jenkins это будет выглядеть вот так:

Сборка

Выполнить команду shell

?

Команда

```
echo "----Create Dataset (begin)-----"
python3 /home/ubuntu/project/create_dataset.py
echo "----Create Dataset (end)-----"
echo "----Train the Model (begin)-----"
python3 /home/ubuntu/project/train_model.py
echo "----Train the Model (end)-----"
echo "----Use the Model for Prediction (begin)-----"
python3 /home/ubuntu/project/prediction.py
echo "----Use the Model for Prediction (begin)-----"
```

Обратите внимание, что здесь указаны полные пути до python скриптов. В вашем проекте могут быть другие пути до файлов, в этом случае их необходимо скорректировать. После изменений в командах shell мы сохраняем сборку и запускаем ее на выполнение. Все должно быть хорошо и вы должны увидеть вот такое сообщение в консоли выполнения сборки:

Вывод на консоль

```
Started by user unknown or anonymous
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Simple ML pipeline
[Simple ML pipeline] $ /bin/sh -xe /tmp/jenkins8473367985961387693.sh
+ echo ----Create Dataset (begin)-----
----Create Dataset (begin)-----
+ python3 /home/ubuntu/project/create_dataset.py
+ echo ----Create Dataset (end)-----
----Create Dataset (end)-----
+ echo ----Train the Model (begin)-----
----Train the Model (begin)-----
+ python3 /home/ubuntu/project/train_model.py
+ echo ----Train the Model (end)-----
----Train the Model (end)-----
+ echo ----Use the Model for Prediction (begin)-----
----Use the Model for Prediction (begin)-----
+ python3 /home/ubuntu/project/prediction.py
[0]
+ echo ----Use the Model for Prediction (begin)-----
----Use the Model for Prediction (begin)-----
Finished: SUCCESS
```

Рисунок “Итоговый результат автоматизации”

Итак, мы построили пайплайн, который выполняет отдельные задачи проекта, запуская специальные скрипты, контролирует правильность выполнения этих скриптов, показывая

статус выполнения, дает возможность в “Консоли вывода” проанализировать выполнение и увидеть возможные ошибки.

Конечно, этот сильно упрощенный пайплайн еще очень далек от настоящих промышленных конвейеров проектов машинного обучения, которые можно создавать в Jenkins. **В промышленный конвейерах используется получение кода из git, автоматическая загрузка на тестовые (test, stage) и производственные (production) сервера, автоматическая настройка эти серверов и многое другое.** Всему этому мы научимся в следующих модулях. А сейчас вы можете себя поздравить с тем, что создали свой первый конвейер проекта машинного обучения, включающий важнейшие этапы: сбор данных, обучение модели и использование модели для предсказаний.

Проверим:

1. С помощью какой команды shell можно сделать паузу в выполнении скрипта (0.25)
 - a. **sleep**
 - b. delay
 - c. wait
 - d. pause
2. какая python библиотека дает возможность сохранять обученные модели в специальном бинарном формате (0.25)
 - a. rar
 - b. zip
 - c. **pickle**
 - d. tar
3. как можно получить информацию об ошибке в ходе выполнения сборки Jenkins (0.25)
 - a. по электронной почте
 - b. по SNMP
 - c. **в окне “Вывод консоли”**
 - d. никак, это закрытая информация
4. Что означает #3 для сборки Jenkins (0.25)
 - a. **третья по счету запущенная версия сборки**
 - b. ошибка номер три при запуске сборки
 - c. количество процессоров, необходимое для корректной работы сборки
 - d. третий приоритет сборки

Итоги

В этом юните мы применили Jenkins для автоматизации задач очень простого проекта машинного обучения. Тем не менее этот проект включал важные этапы, входящие в любой проект машинного обучения: получение данных, обучение модели и применение этой модели. Уже на этом простом примере мы увидели преимущества автоматизации и мониторинга хода выполнения задач. Более сложные инструменты мы будем изучать в следующих модулях.

Итоги модуля

В этом модуле были рассмотрены технологии непрерывной интеграции, доставки и тестирования.

Непрерывная интеграция (CI) решает задачи:

1. Проверка правильности сделанных изменений
2. Публикация всех изменений программного кода, служебных скриптов, настроек в главном репозитории проекта
3. Сборка всех изменений в единый проект
4. Тестирование совместимости измененных частей проекта
5. Проверка функциональности

Непрерывная доставка (CD) решает задачи:

1. Настройка оборудования
2. Настройка среды выполнения программы
3. Установка и настройка системы
4. Создание пользовательских ролей
5. Организация мониторинга

Непрерывное тестирование (CT) решает задачи:

1. Тестирование отдельных модулей (юнит тесты)
2. Тестирование совместимости
3. Тестирование производительности
4. Тестирование функций

Мы изучили особенности CI/CD/CT для проектов машинного обучения и наиболее важные отличия CI/CD/CT MLOps от обычного проекта:

- в MLOps CI задачи интеграции применяются и к новым объектам и артефактам, относящимся к проектам машинного обучения: данным, моделям, пайплайнам, средам исполнения,
- MLOps CD это не просто набор действий по установке пакета программного обеспечения или сервиса, а система последовательных операций (конвейер машинного обучения, ML training pipeline), являющаяся неотъемлемой частью проекта ML, обеспечивающая его успех,
- в MLOps CT появляется новая сущность, уникальная для ML, суть ее в автоматическом переобучении моделей машинного обучения на новых данных в процессе работы .

Также мы изучили популярный инструмент Jenkins и на простом примере рассмотрели его применение для автоматизации.

Практическое задание модуля

В практическом задании по этому модулю вам нужно разработать собственный конвейер автоматизации для проекта машинного обучения, аналогичный тому, который мы рассмотрели в последнем юните этого модуля. Для этого вам понадобится виртуальная машина с установленным Jenkins, python и необходимыми библиотеками. В ходе выполнения практического задания вам необходимо автоматизировать сбор данных, подготовку датасета, обучение модели и работу модели.

Этапы задания

1. Развернуть сервер с Jenkins, установить необходимое программное обеспечение для работы над созданием модели машинного обучения.
2. Выбрать способ получения данных (скачать из github, из Интернета, wget, SQL запрос, ...).
3. Провести обработку данных, выделить важные признаки, сформировать датасеты для тренировки и тестирования модели, сохранить.
4. Создать и обучить на тренировочном датасете модель машинного обучения, сохранить в pickle или аналогичном формате.
5. Загрузить сохраненную модель и проанализировать ее качество на тестовых данных.

Критерии выполнения задания:

1. установлен и настроен Jenkins
2. написан корректно работающий job в Jenkins, решающий задачу
3. получены данные о качестве работы модели машинного обучения.

Форма сдачи: на синхронном занятии с преподавателем.

Модуль 3. Виртуализация и контейнеризация.

Одной из самых серьезных технологических проблем в проектах разработки программного обеспечения является обеспечение совместимости различных библиотек, используемых в проекте. Для машинного обучения разработано множество open-source инструментов (библиотек, фреймворков), что является несомненным плюсом. Однако каждый из этих инструментов разрабатывается по собственной стратегии, разными командами разработчиков. Разные библиотеки развиваются и меняют версии с разной скоростью, что приводит к несовместимости различных библиотек между собой. Такие проблемы очень тяжело диагностировать, **поэтому важная часть в работе с моделями машинного обучения на всех этапах это обеспечение повторяемости результата за счет строгого контроля версий всех компонентов используемого программного обеспечения и системных настроек среды, в которой выполняется программа. Это достигается с помощью создания выделенных сред выполнения программ, среди которых наиболее популярными являются виртуализация ресурсов и контейнеризация приложений.** Архитектура программного обеспечения подстраивается под эти подходы, отдавая предпочтение все более популярной идеологии микросервисной архитектуры программного обеспечения, соответствующей разделению общего монолитного решения на отдельные независимые компоненты.

Этот модуль необходим для того, чтобы вы познакомились с основными технологиями и инструментами создания изолированных сред выполнения программ и научились применять эти знания для решения задач MLOps.

В результате изучения данного модуля вы сможете объяснить для чего нужны виртуализация и контейнеризация в проектах машинного обучения и использовать типовые инструменты для решения важных задач в проектах машинного обучения: обеспечения повторяемости результата в разработке и эксплуатации, обеспечения совместимости библиотек, быстрого развертывания сред разработки и эксплуатации, управления конфигурациями сред.

Темы, изучаемые в модуле:

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

Модуль 3. Юнит 1. Различные подходы к созданию изолированных программных сред.

Введение: В этом юните рассматриваются различные подходы к созданию виртуальных изолированных сред для выполнения программ.

Содержание юнита:

Подход №1 “Виртуализация”

Термин “виртуализация” стал очень популярным благодаря развивающимся технологиям облачных вычислений. Появились облачные сервисы, предоставляющие услуги доступа к виртуальным аппаратным и программным ресурсам таким образом, что для пользователя это выглядит как работа с локальным компьютером или сервером. Такой подход эффективен тем, что позволяет оптимально использовать имеющиеся аппаратные ресурсы:

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

При виртуализации имеющиеся аппаратные ресурсы разделяются на логические виртуальные сущности для того, чтобы создать несколько изолированных друг от друга процессов, пользующихся виртуальными ресурсами независимо друг от друга. Это позволяет оптимально распределить между изолированными процессами нагрузку на имеющиеся “физические” аппаратные мощности, избежать простоя и непроизводительного использования ресурсов, повышает рентабельность вложений в аппаратное обеспечение. **Изолированный процесс, пользующийся виртуальными ресурсами, называется виртуальная машина.**



Рисунок “Разделение аппаратных ресурсов между виртуальными машинами”

Виртуальная машина имеет свою операционную систему и работает как независимый компьютер. Кроме того, виртуальная машина имеет свои уникальные характеристики мощности процессора и объема памяти, в одних системах нужен большой объем оперативной памяти, а в других - повышенные требования к скорости вычислений. Ясно

что суммарный объем “виртуальных мощностей” не может превышать мощности “физического” аппаратного обеспечения. В современных кластерах одновременно может работать много виртуальных машин, поэтому актуален вопрос их управления и мониторинга. Система, управляющая виртуальными машинами, называется **гипервизор**. Оборудование, на котором работает виртуализация, называется **хост**, его операционная система - **хостовой**. В противоположность этому, **гостевые** операционные системы это операционные системы, работающие в виртуальных машинах, пользующиеся ресурсами хоста. Гостевые операционные системы работают с эмуляцией аппаратных ресурсов.

Основная роль гипервизора состоит в координации работы виртуальных машин и физического оборудования хоста при предоставлении виртуальным машинам доступа к физическим ресурсам. Также гипервизор изолирует виртуальные машины друг от друга, чтобы они не конфликтовали за память или вычислительный ресурс.



Рисунок “Роль гипервизора в координации доступа виртуальных машин к ресурсам”.

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

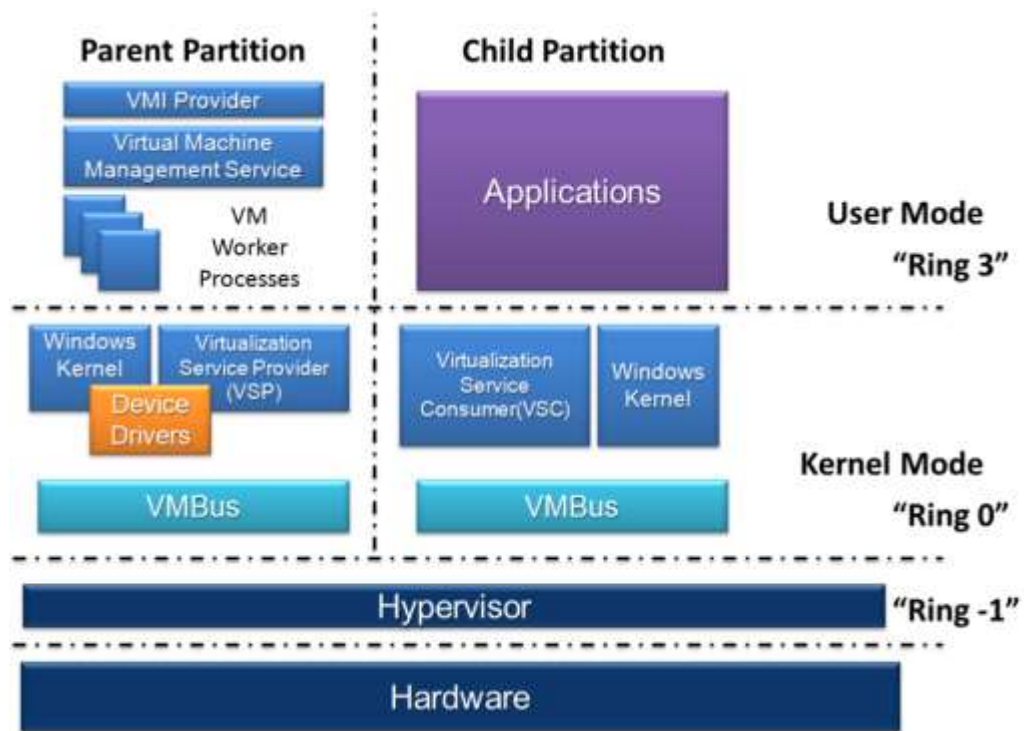


Рисунок “Архитектура работы гипервизора Hyper-V, схема с официального сайта <https://docs.microsoft.com/ru-ru/virtualization/hyper-v-on-windows/about/>”.

Технология виртуализации получила широкое распространение благодаря своим несомненным преимуществам:

- оптимальное использование аппаратных ресурсов
 - Один аппаратный ресурс может быть задействован в нескольких виртуальных машинах, которые будут разделять его использование между собой, что снижает время непроизводительного простоя.
 - *Пример для MLOps: как вы уже наверное знаете, популярный вычислительный ресурс GPU в машинном обучении стоит очень дорого, однако применяется главным образом для обучения моделей. В то время как одна модель уже обучена и проходит тестирование, ресурс GPU может быть использован для обучения другой модели.*
- дополнительные возможности обеспечения безопасности
 - Виртуализация позволяет создать копию рабочей системы, чтобы быстро переключиться на вторую систему при выходе из строя первой. Кроме того, виртуальная машина выглядит идентично обыкновенной системе, однако злоумышленник, получивший доступ к виртуальной машине, не сможет вывести из строя основную, хостовую систему.
 - *Пример для MLOps: в случае возникновения неполадок при работе можно переключиться на резервную систему, сократив время простоя.*
- быстрое развертывание:
 - Можно создать типовые шаблоны и политики для создания виртуальных машин и их настройки, единообразный подход позволяет упростить и ускорить процедуру развертывания решения. Для унифицированной конфигурации проще проанализировать технологические угрозы и есть

возможность создать единые политики безопасности и контролировать их выполнение.

- *Пример для MLOps: для рабочей конфигурации основных элементов можно создать правила развертывания и настройки, сведя к минимуму время на развертывание системы.*

Как видите, технологии виртуализации поддерживают тренд в разработке и использовании программного обеспечения, который мы обсуждали в предыдущих модулях, обеспечивая скорость выполнения операций и надежность.

Примеры виртуальных машин:

- VirtualBox компании Oracle (<https://www.virtualbox.org/wiki/Downloads>), бесплатный инструмент для виртуализации серверов,
- VMware (<https://www.vmware.com>), обеспечивает виртуализацию серверов, рабочих столов (desktops), хранилищ данных,
- Hyper-V (<https://docs.microsoft.com/ru-ru/virtualization/hyper-v-on-windows/about/>), входит в состав ОС Microsoft Windows, предоставляет решения по виртуализации серверов и рабочих компьютеров,
- Citrix, виртуализация для серверов, рабочих компьютеров и приложений.

Список инструментов для виртуализации достаточно широкий, например, в него также входят решения IBM LPAR, Xen, KVM, Bhyve. В конечном счете специалисты DevOps выбирают наиболее оптимальные инструменты, руководствуясь необходимыми функциями, производительностью, стоимостью решения.

С другой стороны, как и любая новая полезная технология, виртуализация порождает новые проблемы с безопасностью. Например, если злоумышленник взломает гипервизор, то он может получить контроль над всеми виртуальными машинами и их операционными системами. Эту проблему будет сложно диагностировать с помощью анализа сетевого трафика, так как гипервизоры обеспечивают взаимодействие виртуальных машин друг с другом без участия физической сети. Поэтому необходимо применять продукты для безопасности виртуализации, способные сканировать и исправлять виртуальные машины, зараженные вредоносным кодом, шифровать все виртуальные диски виртуальных машин и контролировать доступ к ним.

Необходимо отметить, что кроме рассмотренной виртуализации серверов, рабочих компьютеров, процессоров и систем хранения данных еще существуют технологии виртуализации сетей, приложений, отдельных компонентов программной системы или даже больших комплексных решений, например ЦОДов. Основные преимущества подхода точно такие же, как мы рассмотрели, быстрая настройка и конфигурирование, простота в эксплуатации, гибкое управление и надежность.

Подход №2 Контейнеризация

Теперь давайте рассмотрим другой популярный подход для создания изолированной среды выполнения программы, который называется **контейнеризация**. Виртуальные машины и контейнеры принципиально отличаются друг от друга, хотя и решают похожие задачи. При виртуализации необходимо полностью создавать систему, повторяющую полноценный

рабочий компьютер или сервер, включая операционную систему и системные службы, даже если мы создаем среду для выполнения всего одной программы. Конечно же это не производительно, с точки зрения ресурсов, создает избыточность и дополнительные накладные расходы. **Контейнеризация**, ярким представителем которой является docker, **решает эту проблему с использованием контейнеров для упаковки прикладного программного обеспечения, которые обращаются к единому ядру хостовой операционной системы, запуская для себя только те библиотеки или функции, которые нужны для контейнера и приложения в нем.** Это позволяет существенно уменьшить накладные расходы на создание и использование контейнера и повысить общую производительность. То есть, **контейнерная виртуализация — это виртуализация на уровне хостовой операционной системы, которая позволяет запускать изолированные виртуальные системы на одном физическом узле, но не позволяет запускать операционные системы с ядрами, отличными от типа ядра базовой операционной системы.** При таком подходе не существует отдельного слоя гипервизора, вместо этого сама хостовая операционная система отвечает за разделение аппаратных ресурсов между несколькими гостевыми системами (контейнерами) и обеспечивает их независимость.

Идея контейнеризации появилась достаточно давно и у современных популярных инструментов есть достаточно много предшественников: FreeBSD Jail (2000), Virtuozzo Containers (2000), Solaris Containers (2005), OpenVZ (2005), LXC (2008), iCore Virtual Accounts (2008). Появившиеся и развивавшиеся в виде стартапов продукты Docker (2013) и Kubernetes (2014) быстро завоевали большую популярность и в настоящее время являются основными средствами контейнеризации. Например, docker, ставшим стандартом “де-факто”, настолько популярен, что появился даже специальный термин “**докеризация**”. Мы будем рассматривать этот инструмент в следующих юнитах данного модуля. Когда контейнеров много появляются дополнительные издержки на их синхронизацию между собой, актуальным становится вопрос резервирования. Для управления группой контейнеров используются инструменты docker-compose и kubernetes.

Для полноты картины необходимо упомянуть еще инструмент OpenStack (<https://www.openstack.org>), представляющий собой набор open-source проектов программного обеспечения. OpenStack может быть использован для создания инфраструктурных облачных сервисов и облачных хранилищ. Все проекты OpenStack распространяются под лицензией Apache License. Это очень большой проект, включающий множество инструментов, разрабатываемых большим количеством компаний-разработчиков. Перечень отдельных модулей OpenStack очень обширный. OpenStack не является популярным инструментом в MLOps, поскольку задачи в проектах машинного обучения эффективнее решаются другими инструментами. Однако OpenStack набирает популярность в индустрии разработки программного обеспечения, поэтому рекомендуем вам познакомиться с этим инструментом самостоятельно.

В следующих юнитах рассмотрим инструменты для создания виртуальных машин, программных виртуальных сред и контейнеров. Инструментов для виртуализации и контейнеризации достаточно много. Мы в данном модуле ограничимся рассмотрением

инструмента VirtualBox для виртуализации и docker и docker-compose для контейнеризации.

Тест

1. Отметьте гипервизоры (0.25)
 - a. **Hyper-V**
 - b. docker
 - c. **KVM**
 - d. **VMWare**
2. Отметьте инструменты для контейнеризации (0.25)
 - a. **docker**
 - b. **kubernetes**
 - c. containers
 - d. VMWare
3. Отметьте задачи, которые решает виртуализация в проектах машинного обучения (0.25)
 - a. дает возможность делать качественные визуализации
 - b. **ускоряет развертывание проекта с заданными характеристиками**
 - c. **обеспечивает повторяемость результатов работы модели**
 - d. позволяет обрабатывать большие объемы информации
4. Отметьте положительные стороны использования виртуальных машин (0.25)
 - a. **возможность создания и использования унифицированных типовых конфигураций и политик для виртуальных машин**
 - b. существенное ускорение производительности оборудования
 - c. **оптимизация расходов на аппаратное обеспечение**
 - d. гарантия качества работы виртуальной машины компанией-разработчиком

Итоги/выводы

В этом юните вы изучили ключевые технологии для создания виртуальных изолированных сред выполнения программы: виртуализацию и контейнеризацию. Эти технологии соответствуют общему тренду использования микросервисной архитектуры при создании программного обеспечения, ускоряют операции и процессы в проекте.

Наиболее популярные инструменты для виртуализации: VirtualBox Oracle, VMWare, KVM, Hyper-V.

Практически без конкурентов в настоящее время инструменты контейнеризации: docker/docker-compose и kubernetes.

В следующих юнитах мы подробнее рассмотрим создание виртуальных машин с VirtualBox и контейнеризацию с docker/docker-compose.

Модуль 3. Юнит 2. Виртуализация с использованием VirtualBox.

Введение: В этом юните рассматривается задача виртуализации с использованием популярного бесплатного инструмента VirtualBox компании Oracle. С использованием простых шагов, подробно описанных в этом юните, вы сможете самостоятельно создавать виртуальные машины.

Содержание юнита:

Инструмент для виртуализации VirtualBox разработан компанией Oracle и распространяется бесплатно. Инструкции по установке и дистрибутивы для скачивания можно найти здесь:

<https://www.virtualbox.org/wiki/Downloads>

Также для работы вам понадобится образ для операционной системы, которая будет работать в виртуальной машине. В этом юните мы будем использовать Ubuntu, образ которой можно скачать здесь: <https://ubuntu.com/download/server>. Необходимо учитывать, что это очень объемный образ, поэтому для экспериментов с созданием виртуальных машин можно также воспользоваться образом более “легкой” операционной linux системы alpine, скачать которые можно здесь: <https://alpinelinux.org/downloads>.

Для разнообразия в этом юните мы будем изучать практическое применение инструментов в операционной системе Windows 64-бит, надо внимательно выбрать подходящую программу для установки VirtualBox на сайте. Кроме установщика самой программы понадобится еще установщик для пакета расширений VirtualBox, его можно скачать там же.

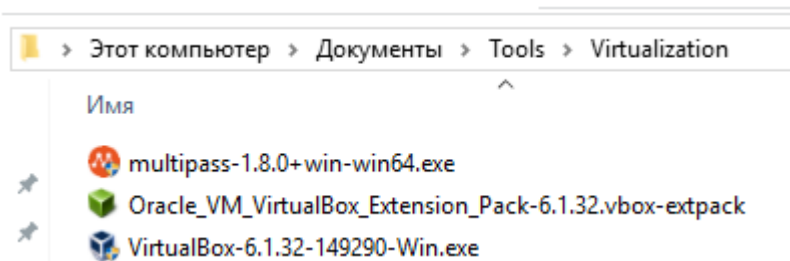
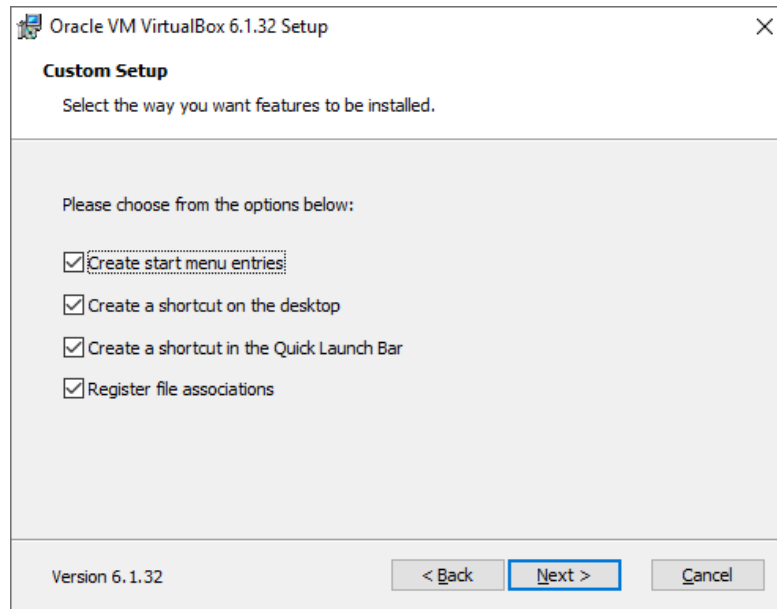
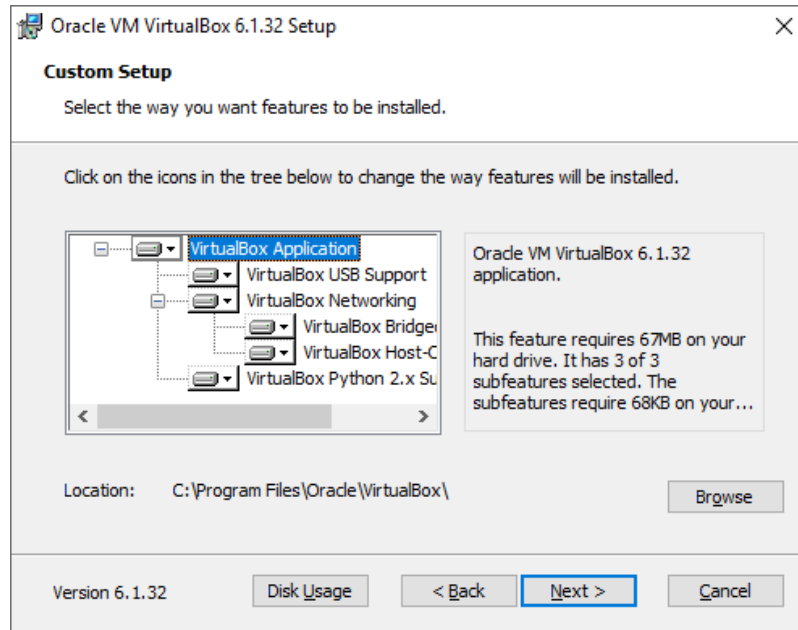
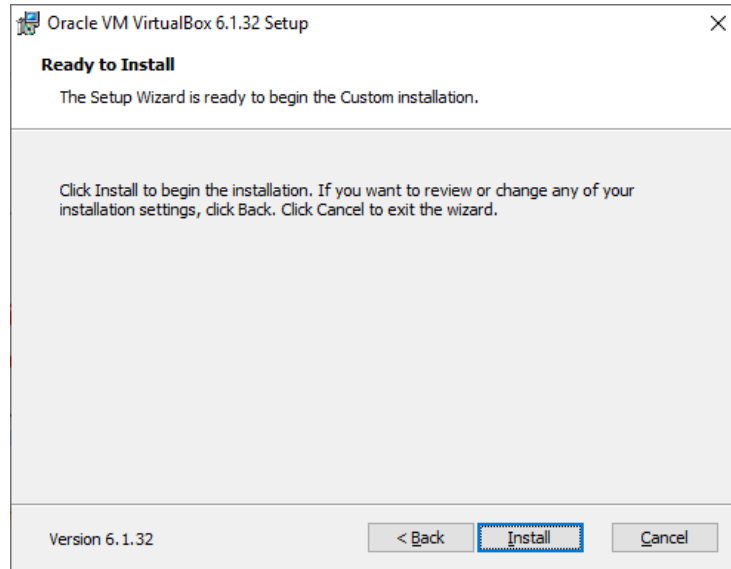


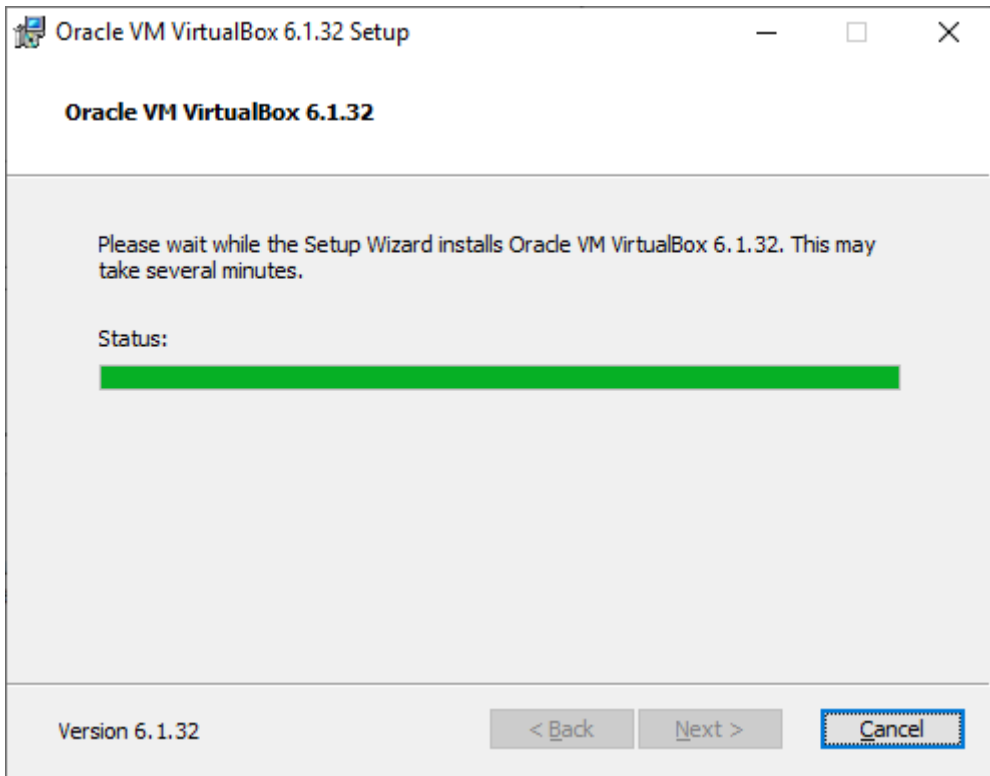
Рисунок “Скачанные дистрибутивы для установки VirtualBox”

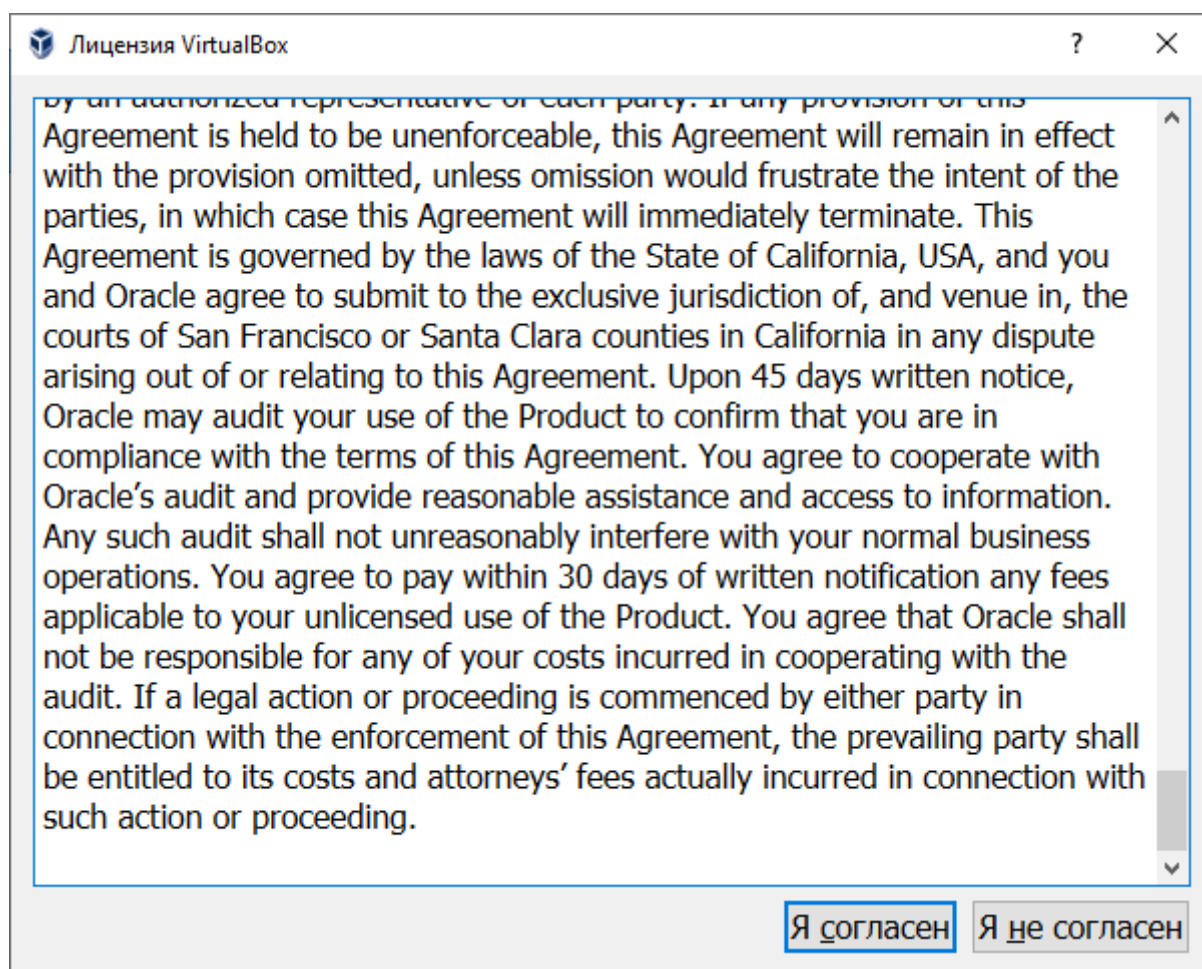
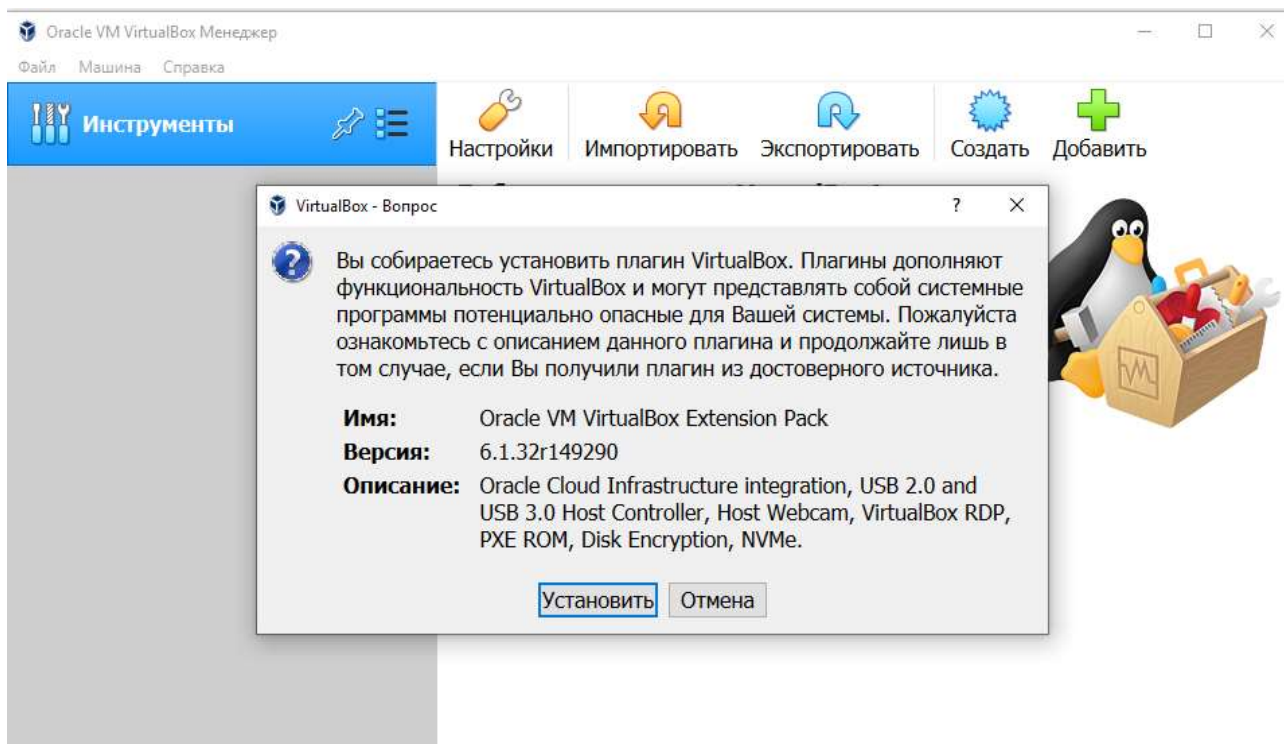
После запуска исполняемого файла “VirtualBox...-Win.exe” вы будете следовать обычному сценарию установки программного обеспечения в Windows. Далее приведены скриншоты отдельных этапов установки.

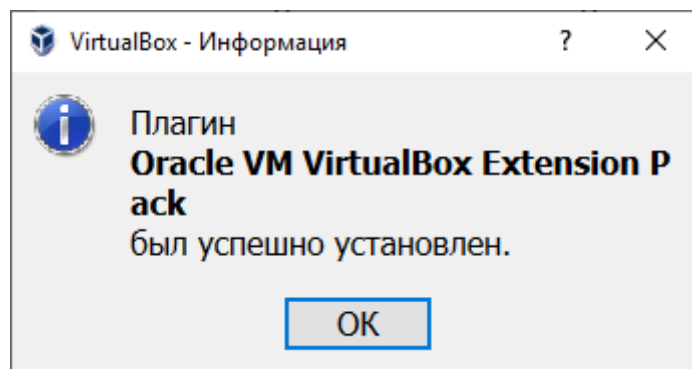
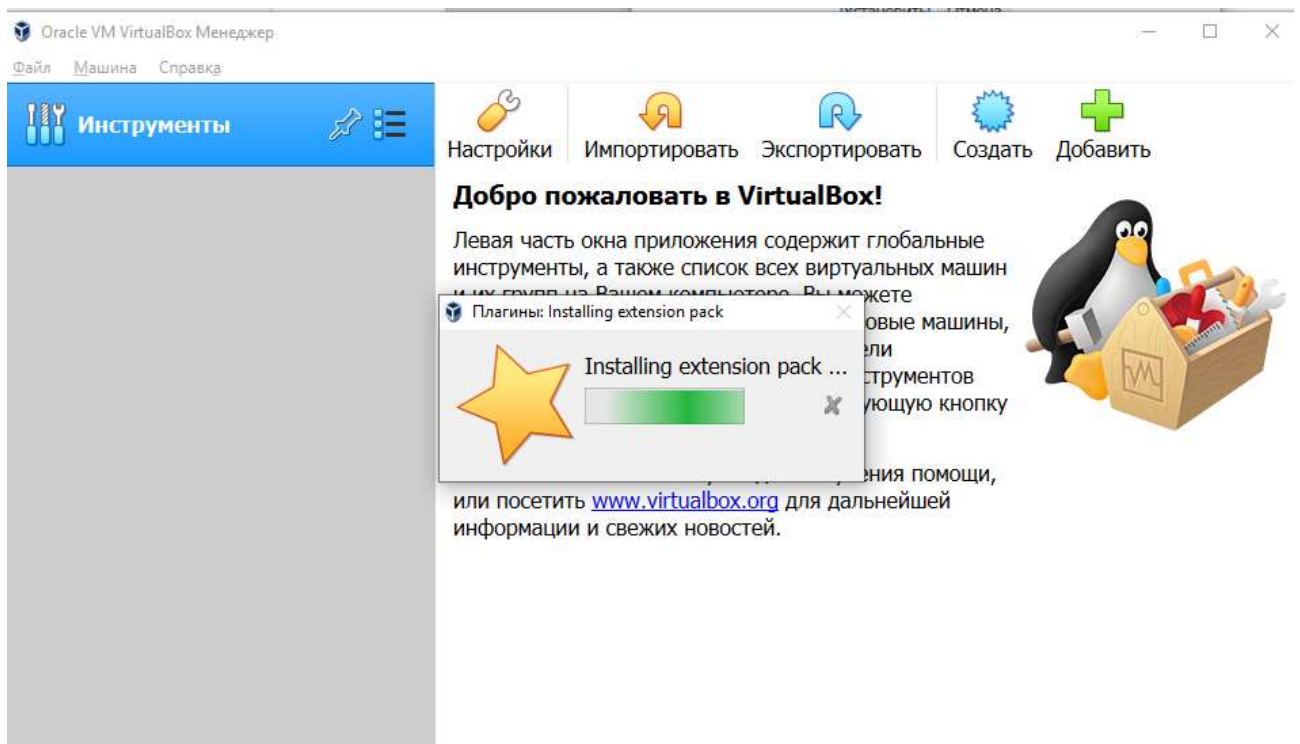




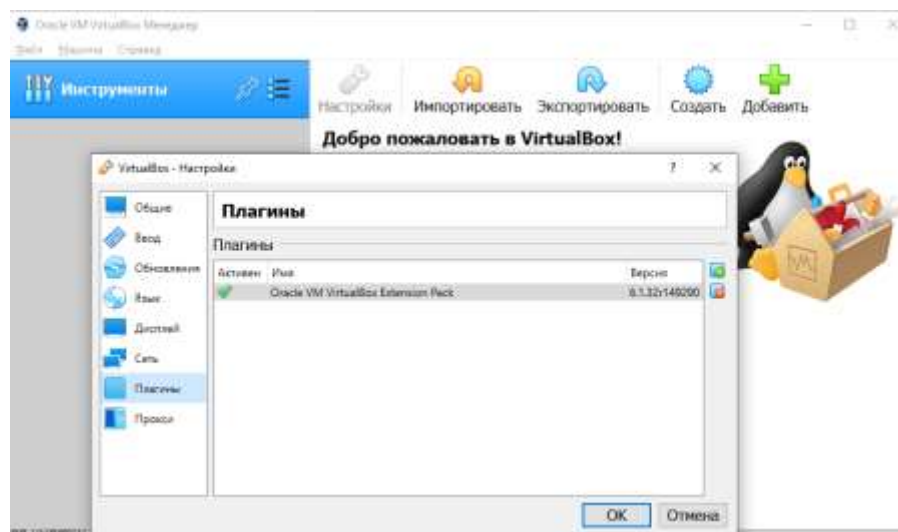








После установки в меню **Файл -> Настройки -> Плагины** (или, в англоязычном варианте, **File -> Preferences -> Extensions**) можно перейти в настройки и убедиться, что Extension Pack установлен.



В VirtualBox каждая виртуальная машина может иметь до четырех сетевых адаптеров, а каждый такой адаптер имеет определённый режим работы:

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

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

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

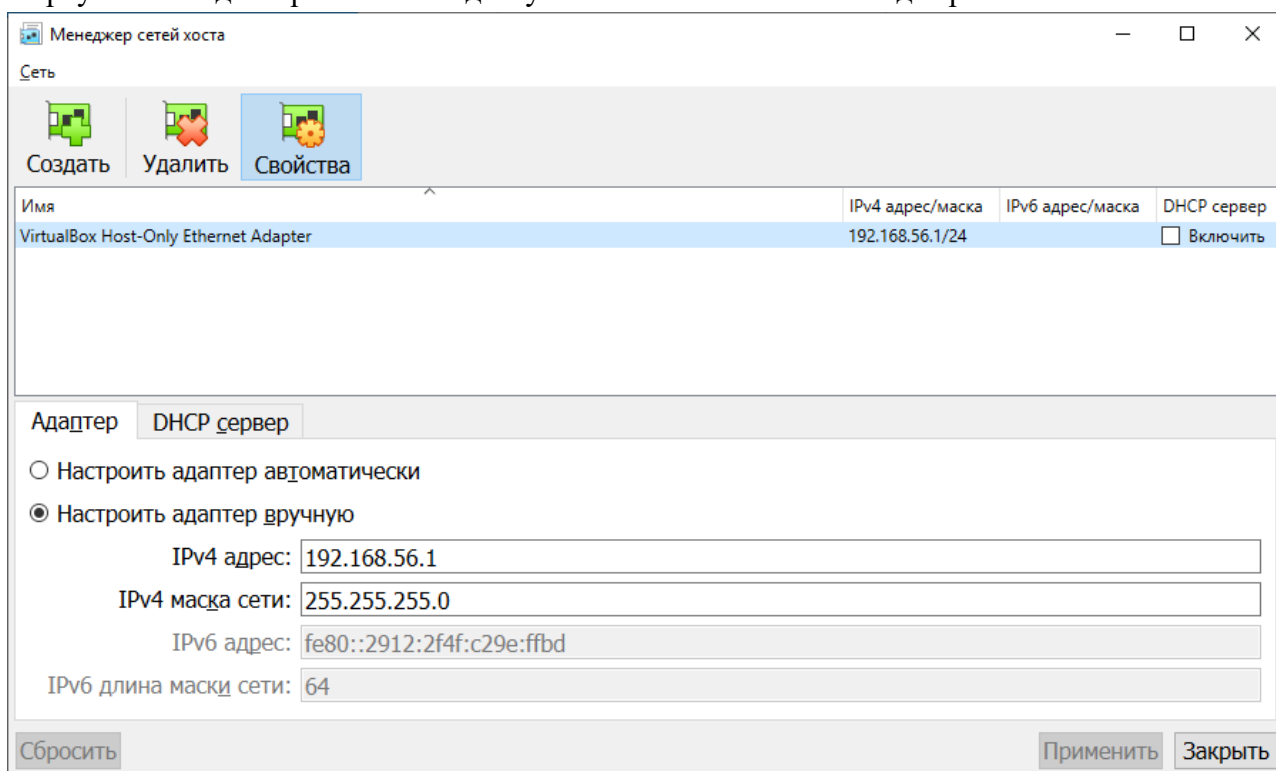


Рисунок “Менеджер сетей хоста VirtualBox”.

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

Давайте создадим шаблонную виртуальную машину. Для этого в пункте меню “Машина” необходимо выбрать раздел “Создать”.

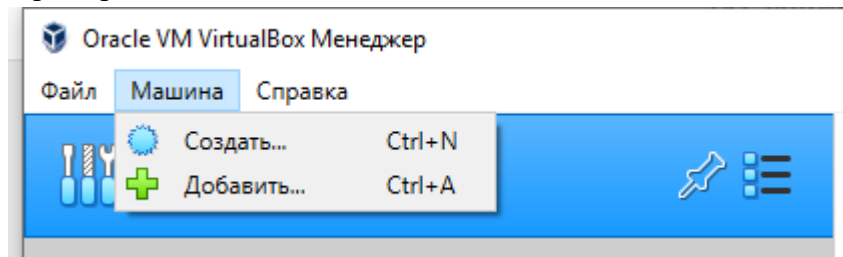


Рисунок “Создание виртуальной машины”

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

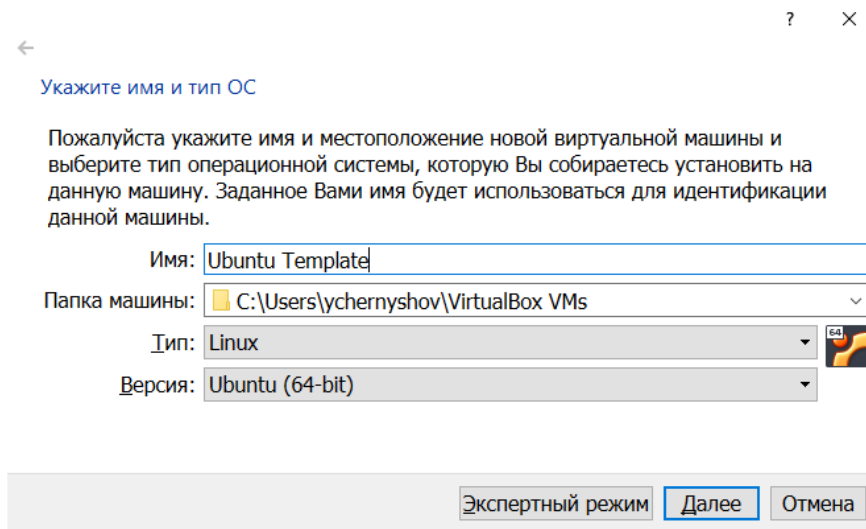


Рисунок “Указание параметров виртуальной машины”

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

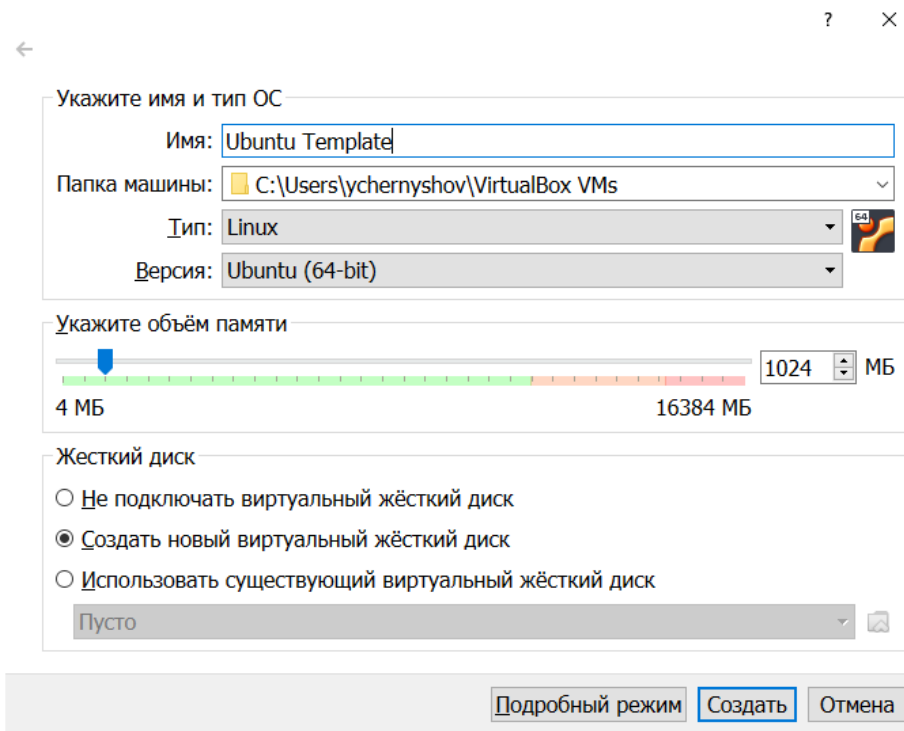


Рисунок “Расширенные настройки виртуальной машины”.

После этого создаем для виртуальной машины новый виртуальный жёсткий диск в формате VDI (VirtualBox Disk Image) с форматом хранения “Динамический виртуальный жесткий диск” и размером 10Гб.

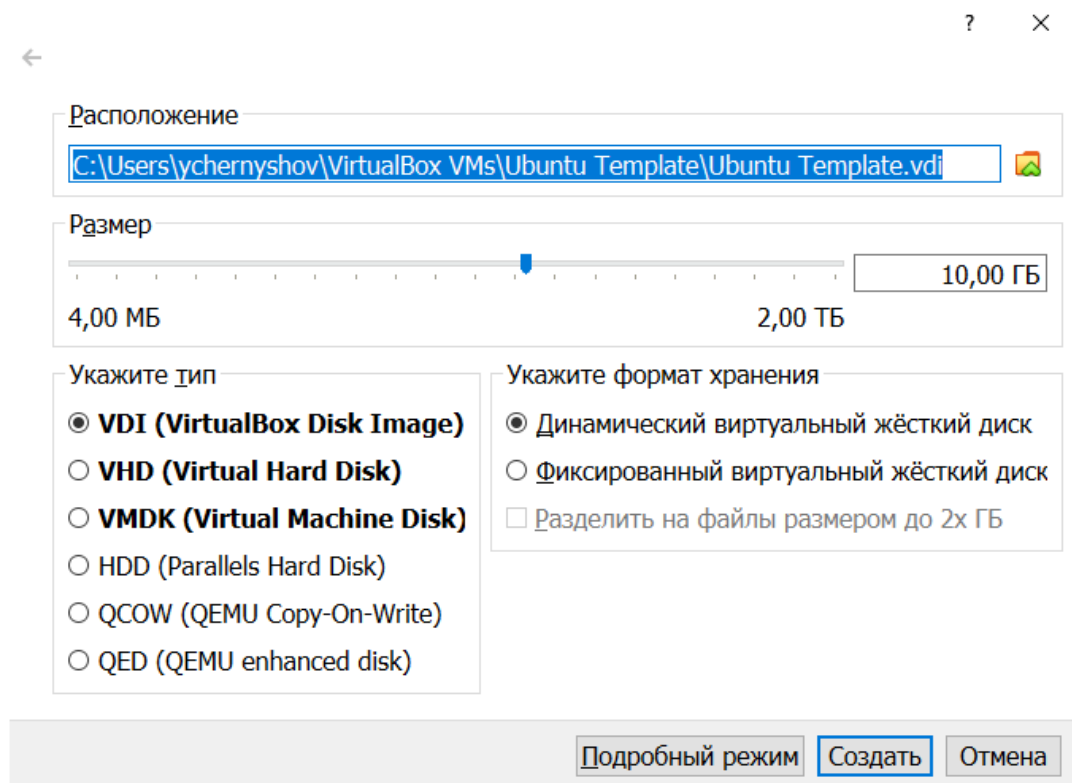


Рисунок “Параметры виртуального жесткого диска”

После этого наша виртуальная машина готова.

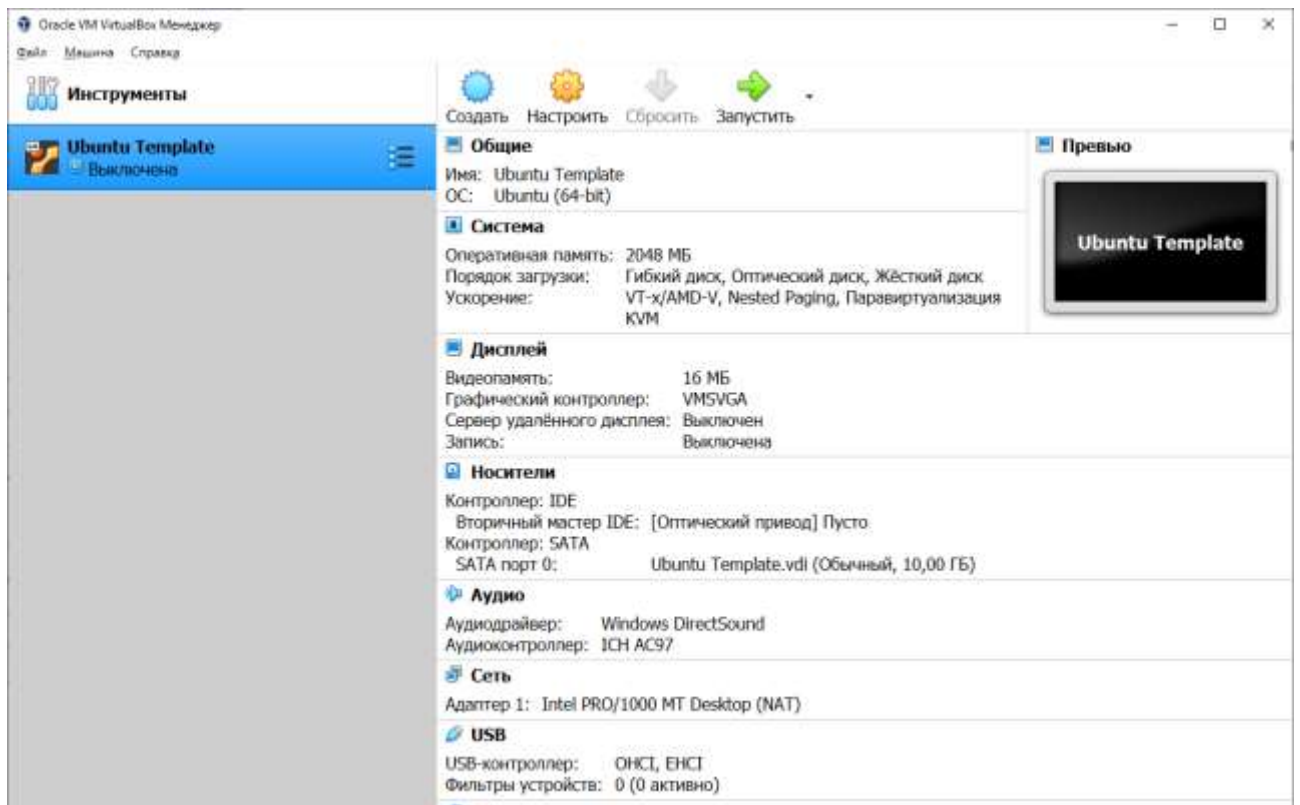


Рисунок “Переметры созданной виртуальной машины VirtualBox”.

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

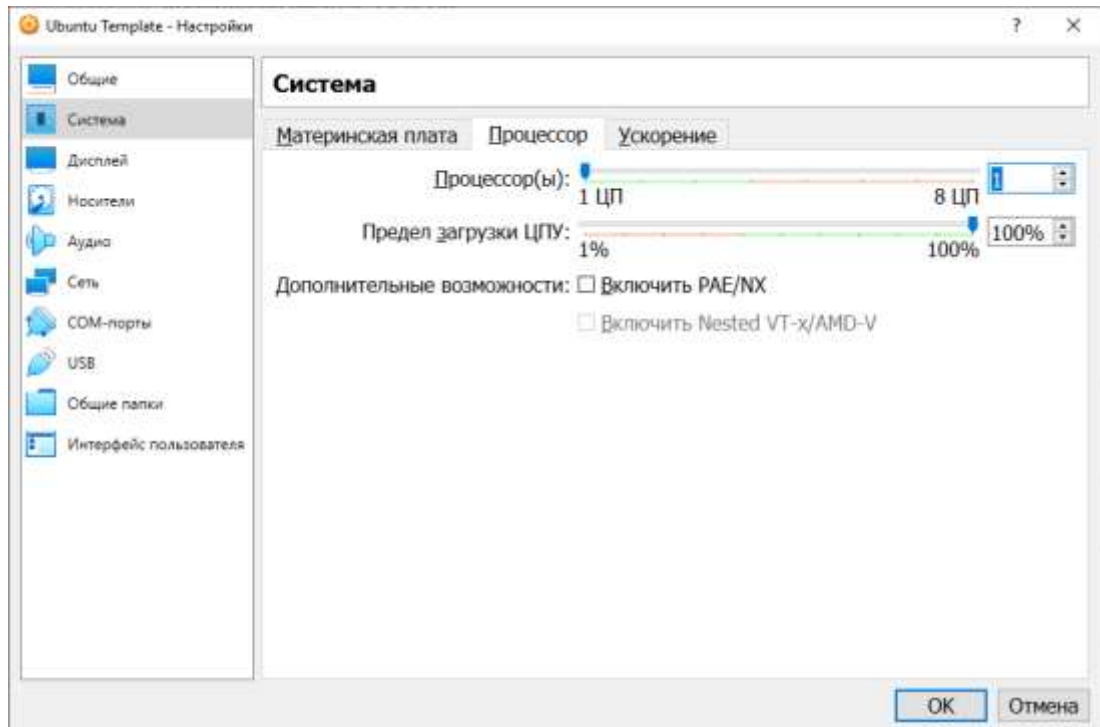
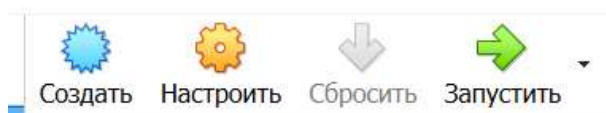


Рисунок “Настройки параметров виртуальной машины, изменение количества процессоров”.

Конечно же, виртуальная машина пока не может эксплуатироваться, хотя и обладает уже виртуальными ресурсами - процессором, оперативной памятью, жестким диском, сетевым

адаптером. Для работы необходимо использовать установочный образ, с которого в виртуальную машину будет загружена гостевая операционная система. *Один из способов указать виртуальной машине откуда брать загрузочные данные, это создать виртуальный оптический диск и прикрепить к нему образ операционной системы, скачанный с официального сайта (например, в формате ISO).* Для этого в панели управления виртуальными машинами надо для рабочей виртуальной машины выбрать пункт “Настроить”



после чего появится следующее окно для настроек

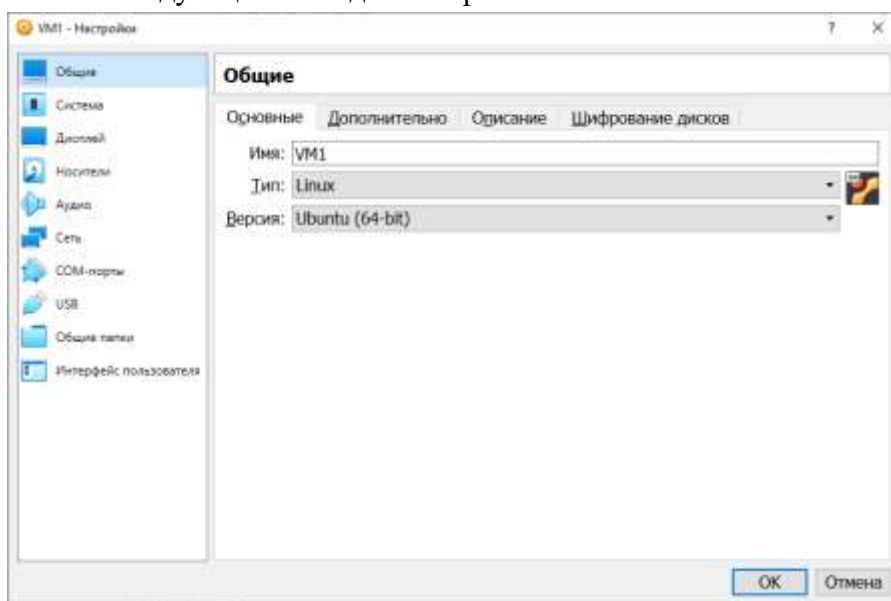
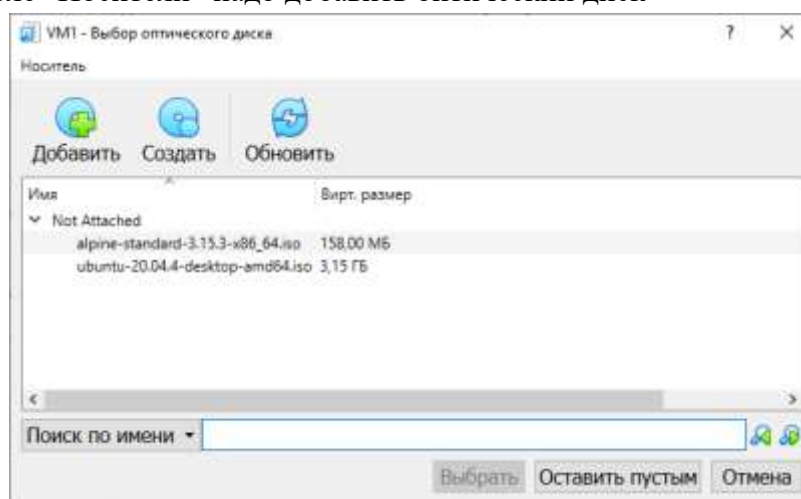
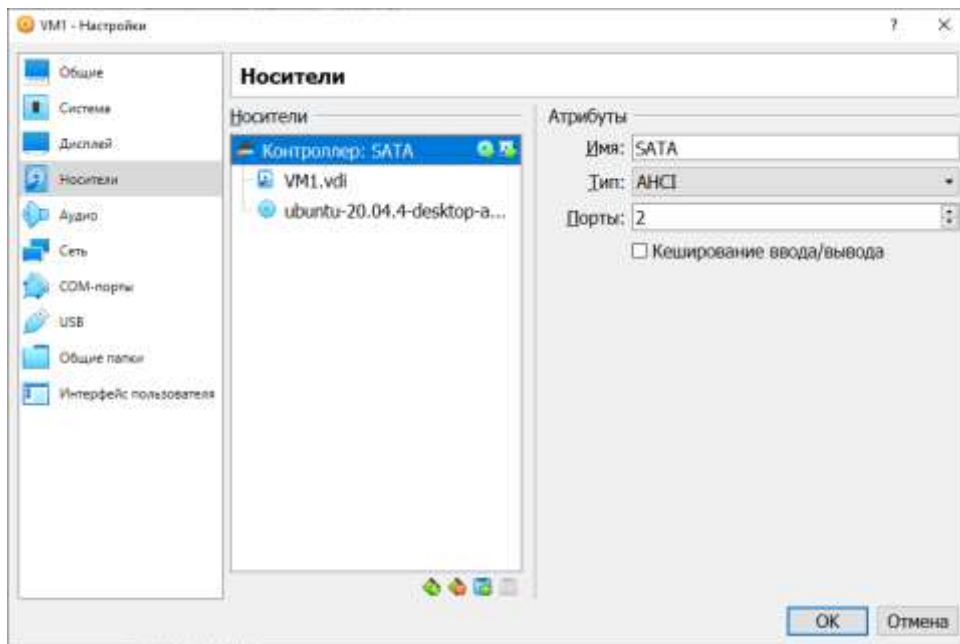


Рисунок “Настройки виртуальной машины”

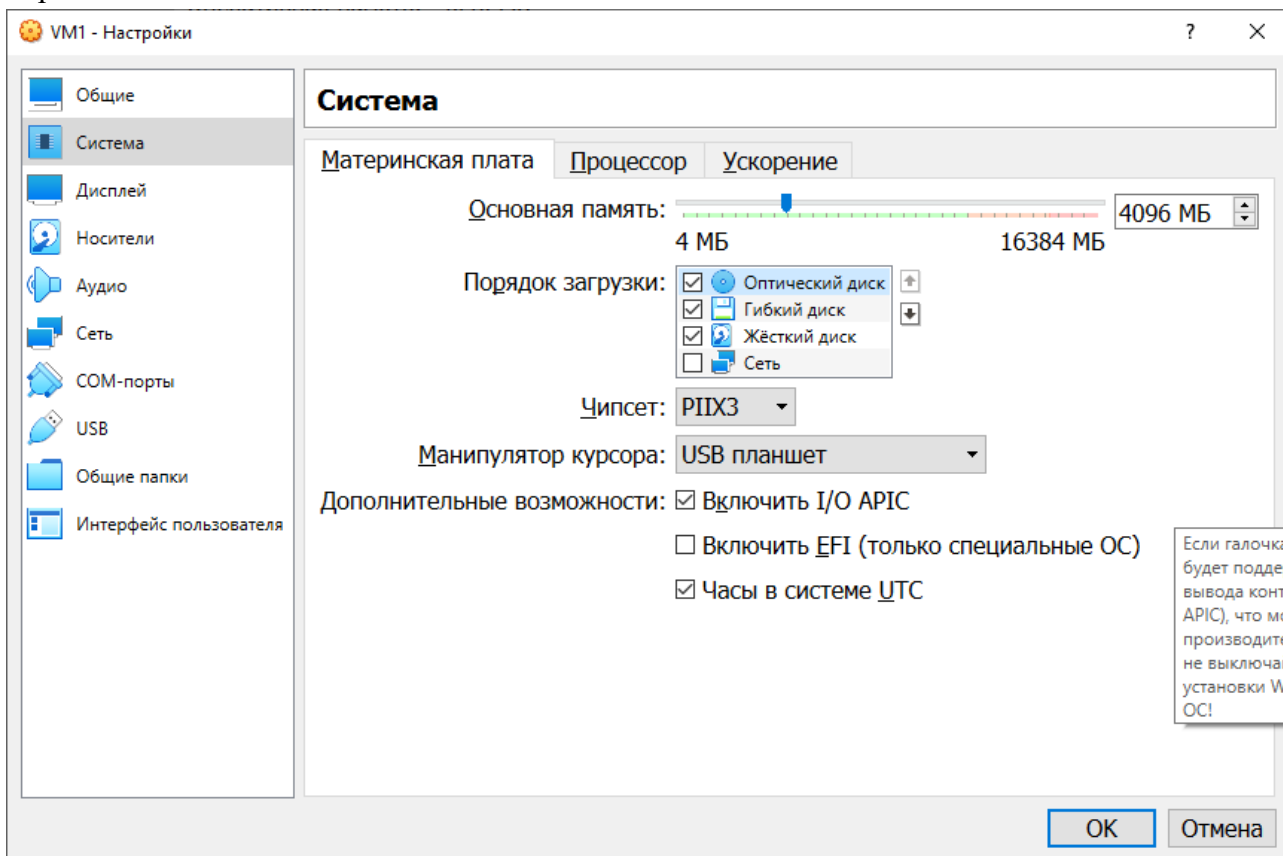
Сначала в разделе “Носители” надо добавить оптический диск



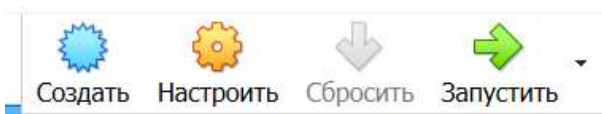
Он должен появиться в перечне дисков



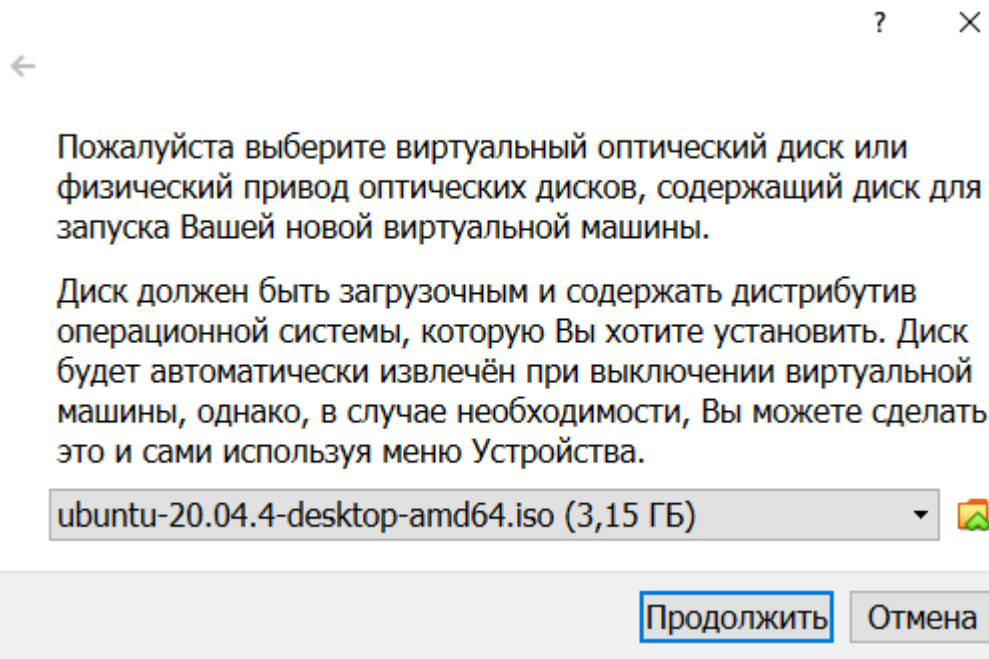
После этого в разделе “Система” меняем порядок загрузки, делаем оптический диск первым



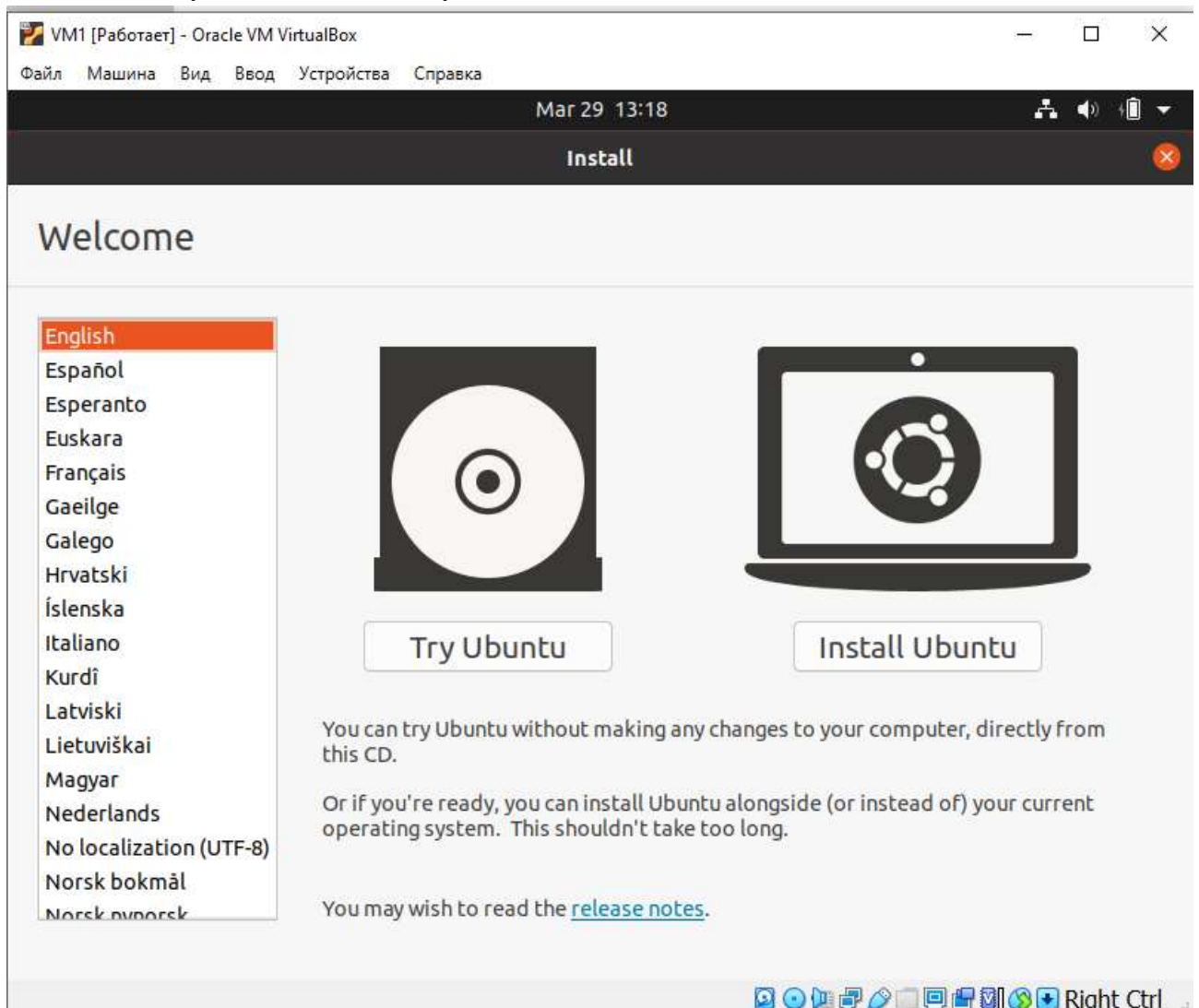
После этого нажимаем кнопку “Запустить”



и в появившемся окне надо выбрать виртуальный загрузочный диск



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



Полезная информация: если вы нажмете курсором мыши на окно виртуальной машины, то произойдет перехват управления и передача всех управляющих сигналов от мыши и клавиатуры к виртуальной машине. То есть мышь и клавиатура не будут работать в домашней операционной системе. Поначалу это может удивлять и пугать. Вернуть управление можно нажатием специальной клавиши, в большинстве случаев это правая клавиша Ctrl.

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

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

```
sudo apt install virtualbox  
sudo apt install vagrant  
vagrant --version
```

Тест

1. Как можно получить официальный дистрибутив VirtualBox? (0.25)
 - a. скачать установщик с сайта производителя
 - b. использовать утилиту pip install
 - c. использовать консольную команду virtualbox install
 - d. эта утилита входит по умолчанию во все стандартные операционные системы, ее не надо скачивать и устанавливать
2. сколько виртуальных сетевых адаптеров может быть связано с виртуальной машиной в VirtualBox? (0.25)
 - a. 1
 - b. 2
 - c. 3
 - d. 4
3. сколько операционных систем может работать в виртуальной машине? (0.25)
 - a. 1
 - b. 2
 - c. 3
 - d. 4
4. как вернуть перехваченное управление клавиатурой и мышью из гостевой операционной системы обратно в операционную систему хоста? (0.25)
 - a. написать письмо в техническую поддержку производителя
 - b. нажать правую клавишу ctrl
 - c. перезагрузить компьютер
 - d. нажать ctrl+alt+del.

Итоги/Выводы

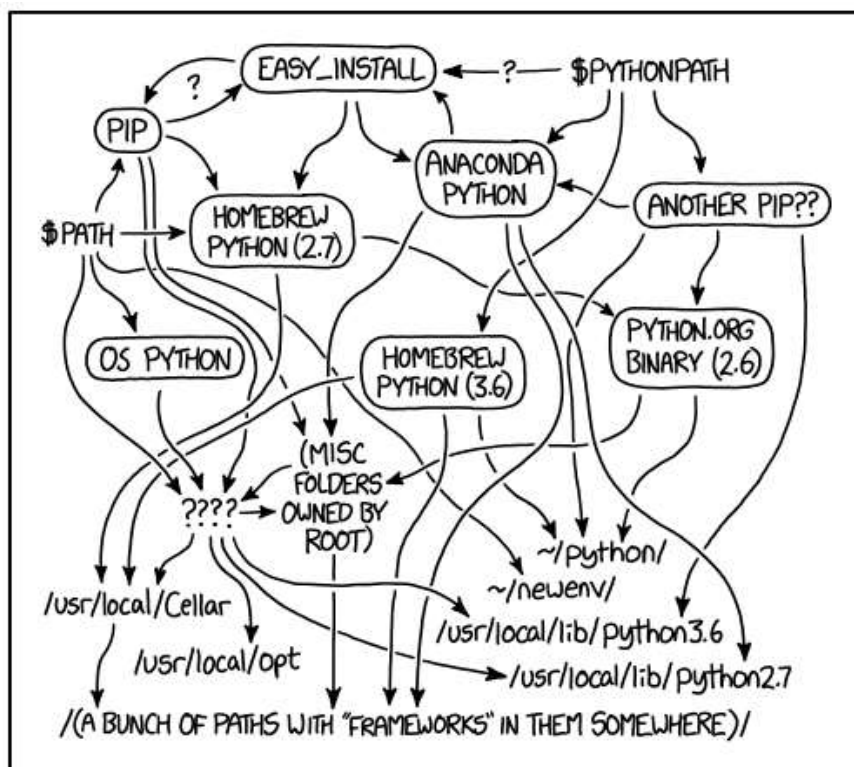
В этом юните вы научились устанавливать программное обеспечение VirtualBox Oracle и использовать его для создания виртуальных машин. В качестве примера мы разобрали создание виртуальной машины с операционной системой Ubuntu.

Модуль 3. Юнит 3. Виртуальные окружения.

Введение: Виртуализация и контейнеризация решают задачу создания среды выполнения программы на уровне аппаратного обеспечения или ядра операционной системы хоста, что требует достаточно много ресурсов. На практике для отдельных задач в проекте разработки программного обеспечения бывает достаточно в операционной системе создать виртуальное окружение, без создания новой виртуальной машины или контейнера. В этом случае изоляция будет не очень сильной, однако это позволяет изолировать среду для уникального набора используемых утилит, настроек, библиотек например, python. Такой механизм называется **виртуальное окружение** (virtual environment). Виртуальные окружения активно применяются, в том числе в проектах машинного обучения, в сочетании с виртуализацией и контейнеризацией. В этом юните мы более подробно рассмотрим вопросы создания и использования виртуальных окружений.

Содержание юнита:

Виртуальное программное окружение отличается от рассмотренных виртуальных машин и контейнеров тем, что не содержит операционную систему. **Основное назначение виртуального окружения состоит в создании изолированной конфигурации программного обеспечения с определенным зафиксированным набором библиотек определенных версий.** Это позволяет быстро повторить нужную конфигурацию и избежать проблем актуальности версий и совместимости библиотек между собой. Вот иллюстрация того какие боли могут возникнуть в проекте с использованием python



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

<https://xkcd.com/1987/>

Виртуальное окружение позволяет зафиксировать работающие конфигурации используемых библиотек. При необходимости можно быстро переключиться в необходимое виртуальное окружение и запустить прикладную программу в этом

уникальном сочетании библиотек, настроек, конфигураций. Это позволяет быстрее повторить и локализовать проблему. При работе с python проектами, а в машинном обучении python является наиболее распространенным инструментом, подход к созданию целевой конфигурации виртуального окружения состоит в создании и запуске этого виртуального окружения и установки необходимых версий библиотек в это окружение, например, с помощью популярной утилиты **pip** (<https://pypi.org/project/pip/>).

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

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

Если виртуальные окружения не используются, то во время установки пакета полезно использовать дополнительно ключ `--user`, устанавливая пакет локально только для текущего пользователя.

Существует множество утилит для создания виртуального окружения в python, вот наиболее популярные из них: `virtualenv`, `venv`, `conda`, `poetry`. Давайте разберем работу с ними подробнее. *Кстати, для экспериментов вы можете использовать виртуальную машину с гостевой операционной системой Ubuntu, создание которой мы разобрали в предыдущем юните.*

1. virtualenv

По умолчанию в Ubuntu утилита `virtualenv` отсутствует, необходимо устанавливать.

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

Установка осуществляется командой

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

После этого можно пользоваться virtualenv

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

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

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

В рабочей папке виртуального окружения после создания формируется следующая структура каталогов

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

3 directories, 20 files
```

2. venv

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

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

```

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

```

Рисунок “Активация виртуального окружения с помощью venv”

После выполнения скрипта activate при успешной активации виртуальной среды вы увидите соответствующий промптер перед курсором.

Теперь можно устанавливать требуемые версии библиотек программного обеспечения.

```

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

```

Деактивировать виртуальное окружение можно командой

deactivate

3. conda

Anaconda является еще одним популярным инструментом для организации работы над python проектам. В том числе поддерживается возможность создания виртуальных сред. Скачать дистрибутив можно здесь <https://www.anaconda.com/products/distribution>.

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

conda create --name “имя окружения” python=3.6

Активировать виртуальное окружение можно с помощью команды с консоли

conda activate “имя окружения”

Далее можно управлять содержанием виртуального окружения, добавлять нужные библиотеки

conda install numpy pandas

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

```
ychernyshov@bruteforce:~$ conda info
active environment : None
user config file : /home/ychernyshov/.condarc
populated config files :
conda version : 4.10.1
conda-build version : 3.21.4
python version : 3.8.8.final.0
virtual packages :
  __cuda__=10.2=0
  __linux__=3.2.0=0
  __glibc__=2.31=0
  __unix__=0=0
  __archspec__=1=x86_64
base environment : /etc/anaconda3 (read only)
conda av data dir : /etc/anaconda3/etc/conda
conda av metadata url : https://repo.anaconda.com/pkgs/main
channel urls : https://repo.anaconda.com/pkgs/main/linux-64
               https://repo.anaconda.com/pkgs/main/noarch
               https://repo.anaconda.com/pkgs/r/linux-64
               https://repo.anaconda.com/pkgs/r/noarch
package cache : /etc/anaconda3/pkgs
envs directories : /home/ychernyshov/.conda/envs
                  /etc/anaconda3/envs
platform : linux-64
user-agent : conda/4.10.1 requests/2.25.1 CPython/3.8.8 Linux/3.2.0-kali2-amd64 kali/2020.4 glibc/2.31
uid:gid : 1001:1004
netrc file : None
offline mode : False
```

4. poetry

Poetry это хорошая альтернатива pip, которая позволяет отказаться от requirements.txt в пользу более гибкой настройки проекта. Благодаря poetry можно в любой момент посмотреть информацию о зависимостях любого пакета, гибко настраивать версии и обмениваться poetry.lock файлами с уже заготовленным списком версий пакетов. Установщик pip хранит данные о зависимостях в файле requirements.txt, а poetry хранит информацию в файле pyproject.toml, однако, в случае с pip, в его файле хранится только список зависимостей с описанием версий, а в .toml хранится вся основная информация о проекте, что очень удобно, так как все данные собраны в одном месте. Кроме того poetry также может управлять окружением проекта.

Установка осуществляется с помощью команды

pip install poetry

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

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

- name - имя проекта
- version - версия проекта
- description - описание проекта
- license - лицензия проекта
- authors - список авторов проекта в формате name <email>

- maintainers - список менторов проекта формате name <email>
- readme - readme файл проекта в формате README.rst или README.md
- homepage - URL сайта проекта
- repository - URL репозитория проекта
- documentation- URL документации проекта
- keywords - список ключевых слов проекта (макс: 5)
- classifier - список PyPI классификаторов

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

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

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

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

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

[tool.poetry.dependencies]

- pycopg2 = { version = "^2.7", optional = true }
- pymysql = { version = "1.0.2", optional = true }

[tool.poetry.extras]

- mysql = ["pymysql"]
- pgsq1 = ["psycopg2"]

Далее зависимости можно установить двумя способами:

poetry install --extras "mysql pgsq1"

poetry install -E mysql -E pgsq1

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

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

Чтобы создать новый проект с помощью poetry, достаточно выполнить

poetry new <название папки с проектом>

После чего создастся папка с названием вашего проекта, в этой папке будет лежать файл pyproject.toml.

```
ychernyshov@ychernyshov-VirtualBox:~$ python3 -m poetry new env3
Created package env3 in env3
ychernyshov@ychernyshov-VirtualBox:~$ tree env3 -L 2
env3
├── env3
│   ├── __init__.py
│   ├── pyproject.toml
│   ├── README.rst
│   └── tests
│       ├── __init__.py
│       └── test_env3.py
└── 2 directories, 5 files
ychernyshov@ychernyshov-VirtualBox:~$
```

Чтобы установить зависимости проекта достаточно выполнить команду:

poetry install

Чтобы добавить новую библиотеку достаточно выполнить:

poetry add numpy

Чтобы удалить зависимость достаточно выполнить:

poetry remove numpy

Чтобы посмотреть зависимости проекта достаточно выполнить:

poetry show

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

Итак, мы рассмотрели различные инструменты для создания виртуальных окружений. Этих инструментов достаточно много и вопрос выбора конкретного инструмента зависит от навыков и квалификации специалиста DevOps/MLOps. Более надежными считаются инструменты venv и virtualenv, поскольку они появились раньше и авторы успели устранить многие замечания, которые заявлялись пользователями. Однако и poetry набирает сейчас популярность, авторы устраняют найденные замечания и повышают надежность.

Тест

1. Отметьте утилиты для создания виртуальных окружений (0.25)
 - a. virtual-environment
 - b. venv**
 - c. windows
 - d. poetry**
2. Какой командой запускается виртуальное окружение? (0.25)
 - a. start env
 - b. locate /bin/activate**
 - c. run env
 - d. activate env
3. Какой программой остановить виртуальное окружение? (0.25)
 - a. stop

- b. release
 - c. deactivate**
 - d. exit
4. Какая команда poetry добавить библиотеку numpy в проект? (0.25)
- a. poetry add numpy**
 - b. poetry install numpy
 - c. poetry load numpy
 - d. poetry get numpy

Итоги/выводы

В этом юните мы рассмотрели виртуальные программные окружения. Ознакомились с задачами, которые они решают. Изучили инструменты для их реализации.

Модуль 3. Юнит 4. Основы контейнеризации с docker. Установка и настройка.

Введение: В этом юните вы познакомитесь с docker, изучите основные понятия, связанные с docker, научитесь устанавливать и настраивать docker.

Содержание юнита:

Этот юнит начинает ваше знакомство с одним из самых популярных инструментов для контейнеризации, docker. В этом юните вы познакомитесь с основными понятиями docker, его архитектурой. Также вы научитесь устанавливать и настраивать docker для работы.

Контейнеризация приложения — это упаковка приложения в отдельный контейнер, специальную среду с операционной системой и всеми необходимыми библиотеками, связями, зависимостями. Контейнеризация приложений очень популярна в разработке программного обеспечения и практических задачах: организации разработки и тестирования, развертывании инфраструктуры распределенных систем, эксплуатации. Контейнеризация помогает сделать приложения более безопасными, облегчает их простое и быстрое развёртывание в продуктивной среде, улучшает масштабирование. Технология контейнеризации переживает фазу бурного роста и считается одним из основных направлений развития в индустрии разработки программного обеспечения.

Технология Docker предназначена для разработки, установки и запуска в специальных сущностях - контейнерах. С использованием инструментов Docker контейнеризуются программные продукты, информационные системы, отдельные приложения или масштабные системы со сложной архитектурой, состоящей из множества сервисов, в соответствии с концепцией микросервисной архитектуры.

Технология Docker настолько распространена, что стала фактически стандартом де-факто в контейнеризации, часто используется термин «**докеризация**».

Технология контейнеризации идеологически похожа на виртуализацию, но есть существенные различия. Виртуальные машины обеспечивают полный уровень изоляции гостевых операционных систем, однако на это расходуется много ресурсов. Принцип работы Docker отличается от принципов работы виртуальных машин тем, что виртуальная машина взаимодействует напрямую с аппаратным обеспечением, а Docker работает с низкоуровневыми инструментами основной операционной системы, позволяя экономить ресурсы и выполнять задачи быстрее. На практике распространено применение docker - контейнеров в виртуальных машинах.

Давайте познакомимся с ключевыми понятиями и терминами docker.

Docker Платформа (Docker Platform) — это программный комплекс, который упаковывает приложения в контейнеры, запускает контейнеры в аппаратных средах (серверах), управляет логикой работы контейнеров, обеспечивает работу пользователя в системе. Платформа Docker позволяет помещать в контейнеры код и его зависимости (используемые внешние библиотеки, переменные среды окружения, служебные файлы,

параметры). При таком подходе упрощается запуск, перенос, воспроизведение, масштабирование систем.

Docker «Движок» (Docker Engine) — это клиент-серверное приложение, обеспечивающее весь цикл работы с технологией Docker. Docker Engine может использоваться в одном из двух вариантов:

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

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

Docker Образ (Docker Image) это набор данных, содержащий:

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

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

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

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

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

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

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

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

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

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

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

alpine	легкая ОС linux, оптимальна для простых задач или обучения, для большинства задач требует дополнительной установки пакетов	https://hub.docker.com/_/alpine
ubuntu	ОС linux с большим набором библиотек и утилит	https://hub.docker.com/_/ubuntu
nginx	самый популярный и очень функциональный web сервер	https://hub.docker.com/_/nginx
python	ОС linux с установленным программным обеспечением для работы с Python	https://hub.docker.com/_/python/

В образ контейнера поверх базового образа можно добавлять дополнительные слои в соответствии с инструкциями из Dockerfile. Если Dockerfile описывает образ, который планируется использовать для решения задач машинного обучения, то в нём могут быть инструкции для включения библиотек машинного обучения NumPy, Pandas и Scikit-learn. В образе может содержаться, поверх всех остальных, ещё один «тонкий» слой, содержащий программу, которую планируется запускать в контейнере.

Пример Dockerfile

```
FROM python # определяет базовый образ
COPY test.py /apps/ # копирует test.py в директорию /apps образа
CMD ["python", "/apps/test.py"] # запускает выполнение файла test.py
```

Docker Том (Docker Volume) это специальный уровень Docker контейнера, который позволяет передавать данные в Docker контейнер. Docker Том это наиболее

предпочтительный механизм постоянного хранения данных, используемых или создаваемых приложениями. Здесь могут храниться таблицы базы данных, конфигурационные файлы, изображения, html файлы, сохраненные модели в формате pickle и т.п.

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

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

Несмотря на то, что для Docker созданы инструменты для разных операционных систем, использование Windows в качестве хостовой операционной системы не рекомендуется, работать с такими образами сложнее. Изначально docker разрабатывался для linux операционных систем, например, Ubuntu, Red Hat, Debian. Поэтому работа с docker в linux операционных системах рекомендована как основной вариант использования. Давайте рассмотрим процедуру установки docker для операционной системы Ubuntu. Установка docker для различных операционных систем подробно описана здесь <https://docs.docker.com/engine/install/>.

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



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

Server

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

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

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

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

Надо обратить внимание на необходимые параметры операционной системы, в которую мы будем устанавливать docker. На момент подготовки этого материала для установки Docker Engine требовалась одна из 64-битовых версий Ubuntu:

- Ubuntu Impish 21.10
- Ubuntu Hirsute 21.04
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)

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

```
sudo uname -a
```

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

Сначала необходимо удалить старые версии docker, установленные в системе

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

Затем обновить установщик apt-get

```
sudo apt-get update
```

и установить необходимые пакеты для возможности использования HTTPS для установки

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

После этого добавляется ключ для взаимодействия с репозиторием docker по ssh

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

и устанавливаются необходимые для этого настройки

```
$echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
```

```
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

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

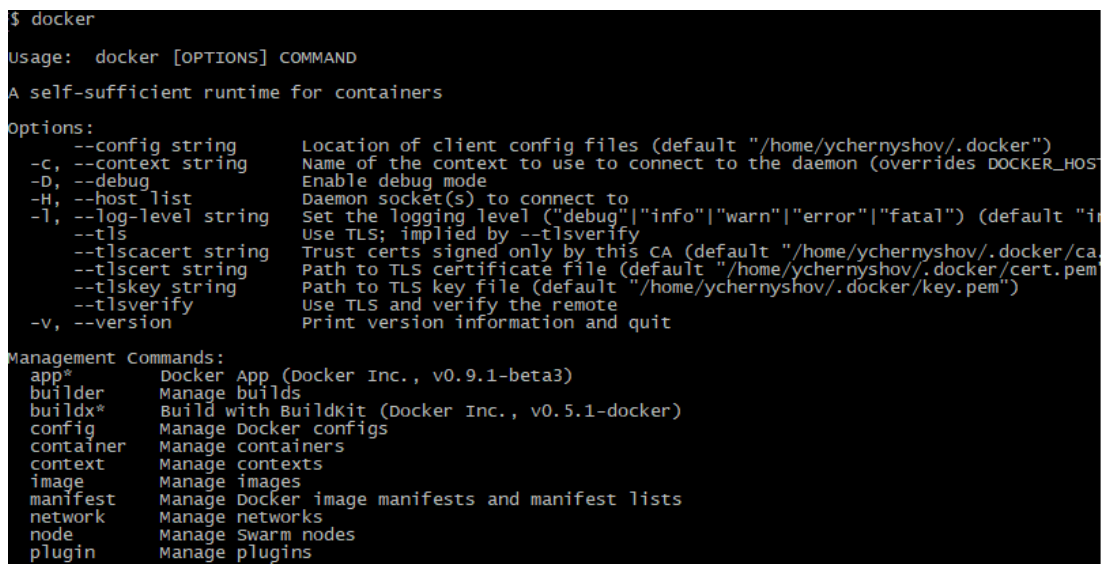
Теперь у нас все готово для установки, для этого еще раз обновим установщик apt-get и запустим установку с помощью команды apt-get install

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

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

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

```
$docker
```



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

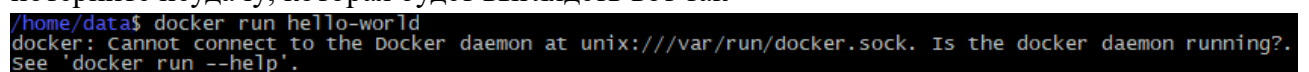
A self-sufficient runtime for containers

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

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

Рисунок “Основное окно docker”

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



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

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

умолчанию запуск команд `docker` требует прав суперпользователя и команды должны запускаться в формате “`sudo docker ...`”. Для корректной работы необходимо настроить `linux` группы, имеющие специальные права доступа к служебным ресурсам `Docker`.

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

```
sudo groupadd docker
```

Добавить пользователя в группу

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

Проверить какие пользователи входят в группу `docker`

```
sudo members docker
```

Удалить пользователя из группы

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

```
sudo deluser user group
```

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

```
$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

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

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

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

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

Рисунок “Результат выполнения команды `docker run hello-docker`”

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

Hello World: What Happened?

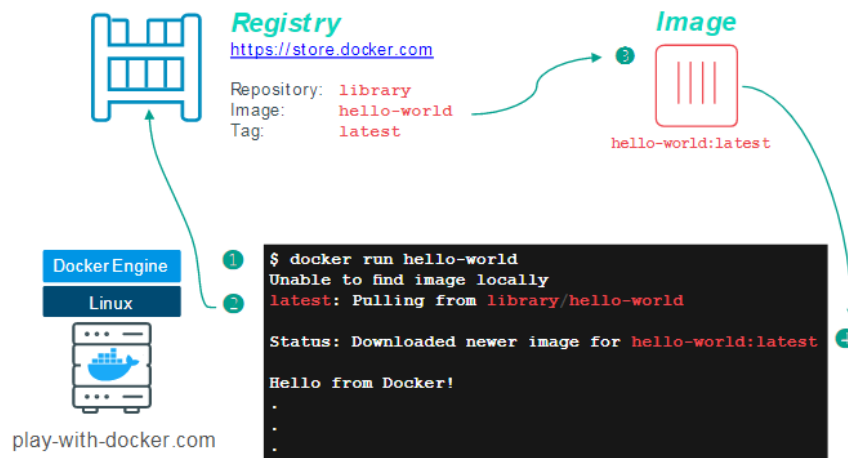


Рисунок “Hello World для Docker, пример с официального сайта docker.com”

Тест

1. Что такое docker образ (0.25)
 - a. виртуальная машина
 - b. набор данных, содержащий операционную систему, служебные файлы, библиотеки, прикладное программное обеспечение**
 - c. набор команд docker для создания контейнера
 - d. исходный код docker
2. Что необходимо для запуска docker контейнера (0.25)
 - a. программист
 - b. docker образ**
 - c. доступ к сайту docker.org
 - d. linux
3. В каком файле находятся команды для создания образа docker (0.25)
 - a. commands-docker-file
 - b. Dockerfile**
 - c. docker-instructions
 - d. docker_image_data
4. Как избежать постоянного применения sudo при использовании docker (0.25)
 - a. sudo docker unset
 - b. добавить пользователя в группу docker, обладающую соответствующими правами**
 - c. sudo docker set primary mode
 - d. внести соответствующие корректировки в Dockerfile

Итоги/выводы

В этом юните вы рассмотрели ключевые понятия и терминологию docker. В следующем юните вы изучите базовые команды docker для выполнения типовых операций.

Модуль 3. Юнит 5. Базовые команды docker.

Введение: В этом юните вы узнаете базовые команды docker для выполнения типичных операций: создание и изменение образа, запуск контейнера, мониторинг работы.

Содержание юнита:

Этот юнит продолжит ваше знакомство с одним из самых популярных инструментов для контейнеризации, docker. В этом юните вы познакомитесь с основными командами docker и примерами применения docker для решения задач.

Базовые команды docker:

docker --version	Используемая версия docker в сжатом формате
docker version	Расширенная информация о версиях различных компонентов docker
docker info	Информация о текущем состоянии docker, служебная информация, статистика, настройки
docker container run hello-world	Проверка работы docker («hello, world» для docker)
docker ps -a	Посмотреть информацию о контейнерах
docker rm <Container ID>	Удалить контейнер
docker stop <Container ID>	Остановить docker контейнер
docker network	Работа с docker сетями
docker image pull alpine	Загрузить docker образ (docker image) операционной системы alpine
docker image ls	Посмотреть имеющиеся в локальной системе пользователя docker образы (docker image)
docker container ls -a	Посмотреть все имеющиеся docker контейнеры (docker container)
docker container run alpine ls -l	Запустить команду «ls -l» в docker контейнере с docker образом операционной системы alpine
docker container run -it alpine sh	Запустить docker контейнер с docker образом с операционной системой alpine в интерактивном режиме и открыть терминал sh
docker container start <ContainerId>	Запустить docker контейнер с идентификатором <ContainerId>
docker container exec <ContainerId> <Command>	Выполнить команду <Command> в docker контейнере с идентификатором <ContainerId>
docker container diff <ContainerId>	Посмотреть изменения, сделанные в docker контейнере с идентификатором <ContainerId>
docker run -d -P --name <имя контейнера> <имя образа>	Запустить образ в контейнере в фоновом режиме (флаг -d, detached mode), все внутренние порты контейнера сделать внешними открытыми и случайными (флаг -

	Р), внутренние (т.е. относящиеся к приложению в контейнере) и внешние (т.е. те, через которые контейнер общается в внешней среде) порты связываются.
<code>docker ps -a -q</code>	Вывести идентификаторы всех существующих контейнеров
<code>docker rm \$(docker ps -a -q -f status=exited)</code>	Удалить все контейнеры, находящиеся в статусе Exited
<code>docker port <Container ID></code>	Посмотреть порты, относящиеся к контейнеру
<code>docker image build -t image_name:v1 .</code>	Создать новый образ с тэгом <code>image_name:v1</code> . Последний параметр (точка, «.») указывает на то, что действия происходят в текущей директории (в которой должен находиться Dockerfile)
<code>docker container commit <ContainerId></code>	Создать новый образ на основе измененного контейнера
<code>docker image tag <ContainerId> <Image Name></code>	Создать тег для образа
<code>docker image history <ContainerId></code>	Посмотреть историю образа
<code>docker image inspect <ImageName></code>	Просмотр детализированной информации об образе
<code>docker image inspect --format "{{ json .Os }}" alpine</code>	Просмотр детализированной информации об образе
<code>docker image inspect --format "{{ json .RootFS.Layers }}" alpine</code>	Просмотр детализированной информации об образе
<code>docker push <Image name></code>	Загрузить образ в docker репозиторий
<code>docker network create <имя сети></code>	Создать сеть docker для обмена сообщениями между контейнерами
<code>docker run -n <имя сети> <имя контейнера></code>	Запустить контейнер с привязкой к сети

Основное полезное содержание работы с docker состоит в создании docker образов, включая операционную систему, необходимые библиотеки и утилиты и в последующем запуске контейнера на основе этого образа. Рассмотрим далее эти две важные операции: создание образа и запуск контейнера.

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

Команда	Назначение	Пример использования
FROM	задаёт базовый (родительский) образ	FROM ubuntu
LABEL	описывает метаданные, например сведения о том, кто создал и поддерживает образ	LABEL maintainer="researcher1"
ENV	устанавливает постоянные переменные среды	ENV PATH="/home/user"
RUN	выполняет команду и создаёт слой образа, используется для установки в контейнер пакетов	RUN pip install numpy
COPY	копирует в контейнер файлы и папки	COPY ./apps
ADD	копирует файлы и папки в контейнер, может распаковывать tar-файлы (архивы)	ADD test.py /apps
CMD	описывает команду с аргументами, которая должна выполняться при запуске контейнера, может быть лишь одна инструкция CMD	CMD ["python", "test.py"]
WORKDIR	задаёт рабочую директорию для следующей за ней инструкции	WORKDIR /apps
ARG	задаёт переменные для передачи Docker во время сборки образа	ARG my_var=1
ENTRYPOINT	предоставляет команду с аргументами для вызова во время выполнения контейнера, аргументы не переопределяются.	ENTRYPOINT ["python", "./script.py"]
EXPOSE	указывает на необходимость открыть порт	EXPOSE 8000
VOLUME	создаёт точку монтирования для работы с постоянным хранилищем	VOLUME /my_volume

В том случае, когда контейнеров много, управлять ими становится сложно. Представьте, что вам придется запустить десять контейнеров командой `docker container run` и к концу запуска десятого вы поймете, что первый уже не работает... Можно, конечно, использовать shell скрипт, автоматизирующий эту ручную работу, но более универсальным инструментом является применение `docker-compose`. **Docker Compose** это

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

Установка docker-compose описана здесь: <https://docs.docker.com/compose/install/>

Команды docker-compose

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

При вызове команды docker-compose ищется файл docker-compose.yml, содержащий необходимые инструкции для docker-compose. Пример такого файла:


```

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

```

Руководствуясь командами из этого файла docker-compose запустит три контейнера, nginx, db и web, используя соответствующие параметры.

Тест

1. Какая команда запускает docker контейнер с использованием образа Image1? (0.25)
 - a. docker container exec Image1
 - b. docker container create Image1
 - c. docker container run Image1**
 - d. docker container start Image1
2. Что произойдет при выполнении команды “docker container run alpine ls -l”? (0.25)
 - a. ошибка выполнения команды
 - b. запустятся два контейнера: alpine и ls с опцией -l

- c. **запустится контейнер с операционной системой alpine и в нем будет выполнена команда просмотра текущей директории**
 - d. будет создан контейнер run
3. Для чего нужна директива FROM в Dockerfile (0.25)
- a. чтобы указать папку, из которой требуется перенести файлы в контейнер
 - b. для указания местонахождения главного исполняемого файла проекта
 - c. **для определения основного образа, на основе которого будет создаваться контейнер**
 - d. для применения интернет-ссылок
4. Какая команда docker-compose запустит все контейнеры, описанные в docker-compose.yml? (0.25)
- a. docker-compose begin
 - b. **docker-compose up**
 - c. docker-compose start
 - d. docker-compose run

Итоги/Выводы

В этом юните вы изучили базовые команды docker, а также познакомились с утилитой docker-compose, которая позволяет запускать несколько docker контейнеров. В следующем юните вы выполните практическое задание с использованием docker, чтобы на практике применить уже имеющиеся у вас знания.

Модуль 3. Юнит 6. Практический пример использования docker.

Введение: В этом юните вы примените на практике имеющиеся знания по docker. Вы научитесь собирать образ и запускать контейнер docker, а также анализировать статистику.

Содержание юнита:

Напомним, что если у пользователя недостаточно прав для выполнения инструкций docker, то эти инструкции необходимо выполнять с правами суперпользователя (superuser), например

sudo docker ps -a

Такой подход не является рекомендованным, так как связан с повышенными рисками информационной безопасности, ведь пользователь может выполнять и другие действия с использованием sudo. Более правильная альтернатива использованию sudo это создание группы пользователей docker и наделение пользователей этой группы соответствующими правами. Далее команда sudo опускается.

Давайте создадим образ и запустим контейнер с этим образом. Для этого надо выполнить следующие шаги.

1. Сначала попробуем запустить docker-контейнер с “чистой” операционной системой alpine, запускающего простой python скрипт.

docker container run alpine python

```
1 docker container run alpine python
unable to find image 'alpine:latest' locally
latest: pulling from library/alpine
486959232610: pull complete
digest: sha256:F22945d1ee2eb4d443e07a431d9f94fcd0ca768bb1acf898d92ce51f7bf04
status: downloaded newer image for alpine:latest
docker: Error response from daemon: OCI runtime create failed: container_linux.go:367: starting container process caused: exec: "python": executable file not found in $PATH: unknown.
[5009] error waiting for container: context canceled
```

По сообщениям системы видим, что python в образе alpine по умолчанию отсутствует. Это было предсказуемо, так как операционная система alpine по умолчанию содержит мало предустановленных утилит и библиотек, благодаря чему занимает очень мало места, но требует дополнительной настройки. По этой причине нам недостаточно контейнера с “чистой” операционной системой alpine и требуется дополнительно устанавливать необходимые нам служебные библиотеки и прикладное программное обеспечение.

2. Создаем файл Dockerfile с содержимым

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

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

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

4. Создаем Docker образ

docker image build -t test_python:0.1 .

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

5. Запускаем контейнер с использованием созданного Docker образа

docker container run test_python:0.1

6. Проверяем статус запущенного контейнера

docker container ls -a

Вы реализовали очень простой пример, не имеющий практической пользы. Дальше вы можете попробовать самостоятельно создать образ docker, содержащий необходимые версии библиотек программного обеспечения для проектов машинного обучения (например, numpy, pandas, scikit-learn) и выполнить в docker образе python код с обучением и использованием модели машинного обучения.

Практическое задание

Используя пример из юнита в качестве образца создайте docker образ для модели машинного обучения, в котором обучается модель логистической регрессии для предсказания типа цветка ириса из датасета sklearn.datasets.iris и возвращает предсказание класса на каком-то тестовом примере, например, [1, 1, 1, 1].

1. Подготовить python код для модели (0.25)
2. Создать Docker file (0.25)
3. Создать docker образ (0.25)
4. Запустить docker контейнер (0.25)

Итоги/Выводы

В этом юните вы выполнили практическое задание с использованием docker.

Итоги/выводы по модулю

В этом модуле вы познакомились с технологиями создания изолированных сред, в которых могут выполняться программы: виртуализацией и контейнеризацией. Преимущества использования этих технологий:

- быстрое развертывание необходимой конфигурации
- удобное управление и мониторинг
- повышение уровня информационной безопасности, возможность локализовать вредоносные действия внутри изолированной среды, возможность быстро вернуться к работающей конфигурации
- единая техническая политика, упрощающая настройку и использование
- эффективное использование аппаратного обеспечения в режиме разделения времени, что повышает эффективность инвестиций в оборудование

Практическое задание

В практическом задании по модулю вам необходимо применить полученные знания по работе с docker и docker-compose. Вам необходимо использовать полученные ранее знания по созданию микросервисов. В этом задании необходимо развернуть микросервис в контейнере докер на облачной платформе Яндекс. Например, это может быть модель машинного обучения, принимающая запрос по API и возвращающая ответ.

1. Получение доступа к инфраструктуре Яндекс.cloud.
 - a. Создать аккаунт на [yandex.ru](https://yandex.ru/cloud/).
 - b. Зайти на [cloud.yandex.ru](https://console.cloud.yandex.ru/) <https://console.cloud.yandex.ru/>
2. Настройка окружения
 - a. Создать виртуальную машину с публичным образом linux.
 - b. Установить docker и docker-compose
3. Создать микросервисы с моделью машинного обучения, принимающей данные и отвечающей по API
4. Запуск через docker-compose

Список терминов и сокращений

ATA	Advanced Technology Attachment. Параллельный интерфейс подключения накопителей к компьютеру.
IDE	Integrated Drive Electronics. Параллельный интерфейс подключения накопителя (жесткого или оптического диска) к компьютеру или серверу. Маркетинговое название для ATA.
SATA	Serial ATA. Последовательный интерфейс обмена данными с накопителями информации. SATA является развитием параллельного интерфейса ATA (IDE).

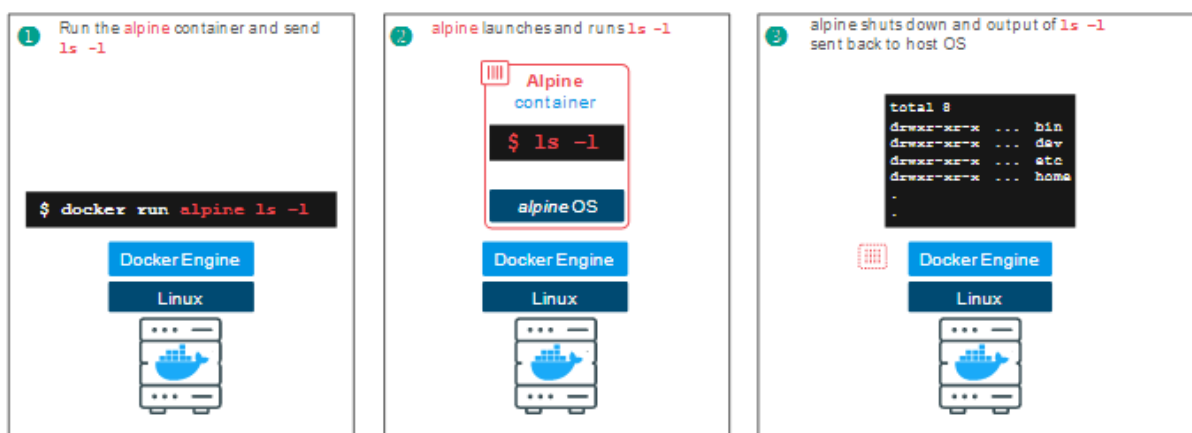
Дополнительные материалы

В этом разделе приведены иллюстрации и описания основных команд docker, взятые с официальных ресурсов о docker в Интернет: <https://training.play-with-docker.com>, [docker.com](https://docs.docker.com) и другие.

1. Выполнение команды docker run

При выполнении команды **docker run <параметры> <имя образа> Docker Engine** запускает контейнер с указанным в команде образом и параметрами.

docker run Details

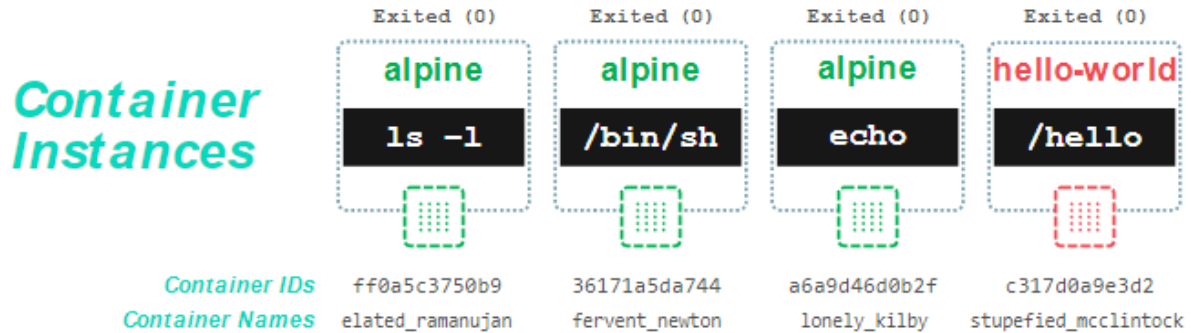


2. Docker контейнеры

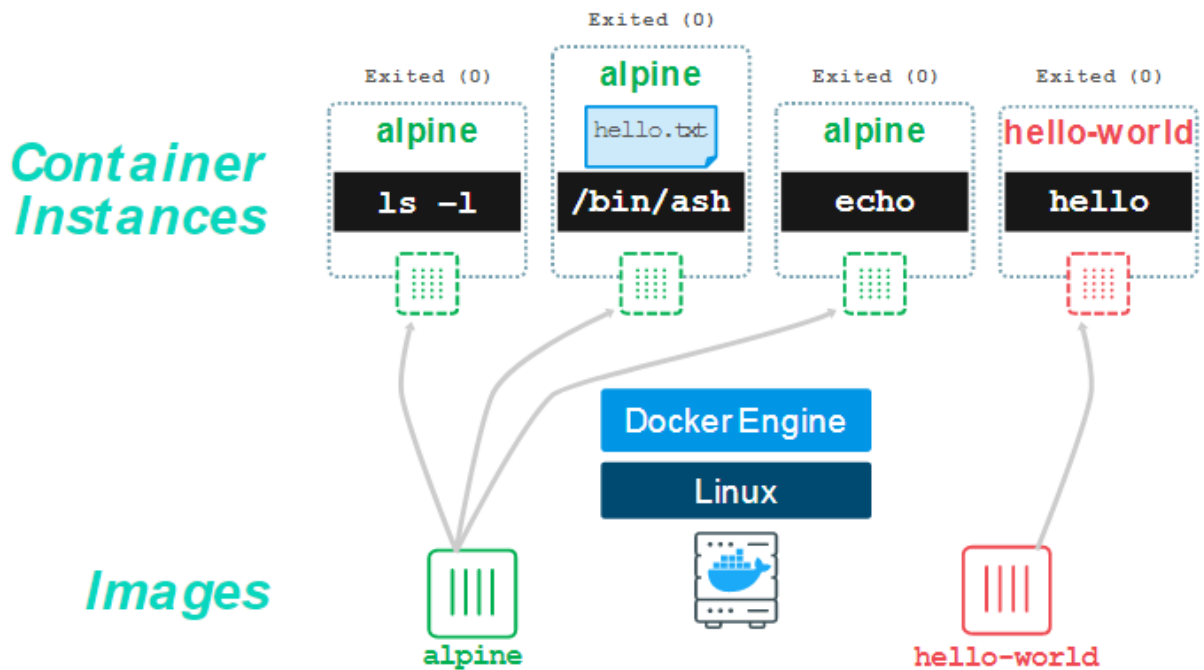
Docker контейнеры, запущенные на выполнение командой `docker run`, представляют собой отдельные изолированные сущности. У каждого контейнера есть свой идентификатор. Информацию о имеющихся контейнерах можно посмотреть командой **docker container ls -a** (флаг `-a` указывает на то, что надо посмотреть все контейнеры, а не только выполняющиеся)

Docker Container Instances

Output of `docker container ls -a`



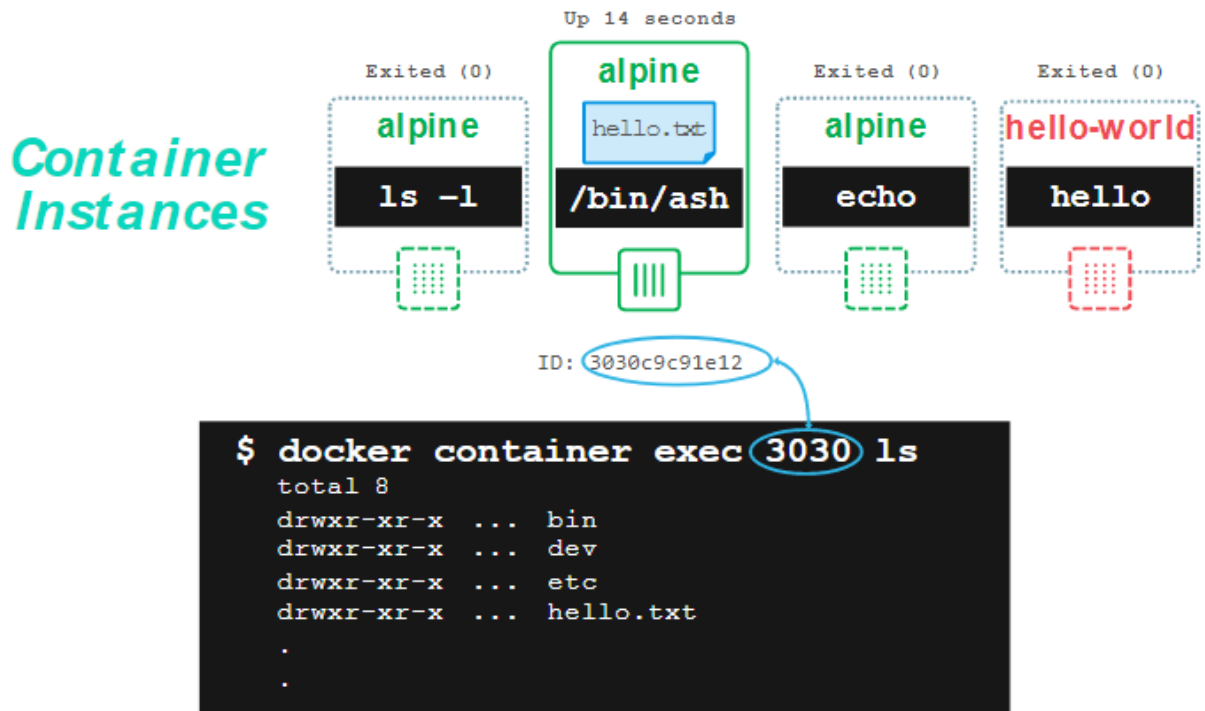
Docker Container Isolation



3. Выполнение команд в контейнере

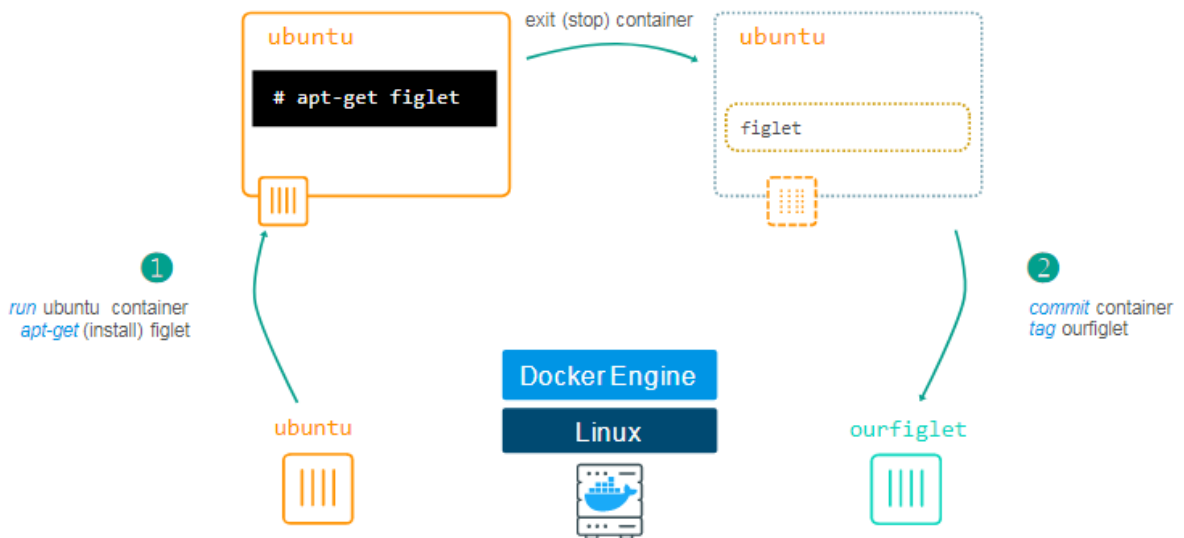
В отдельном контейнере можно выполнять команды, контейнеры идентифицируются номером.

docker container exec



4. Запуск и остановка docker контейнера

Image Creation: Instance Promotion

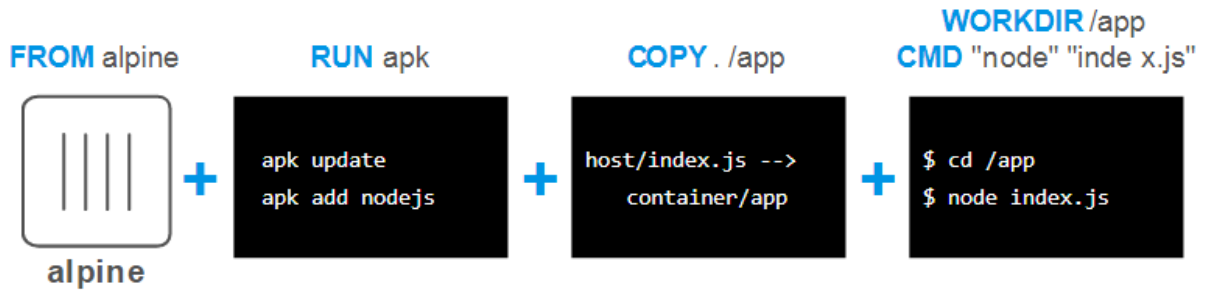


5. Dockerfile

Dockerfiles

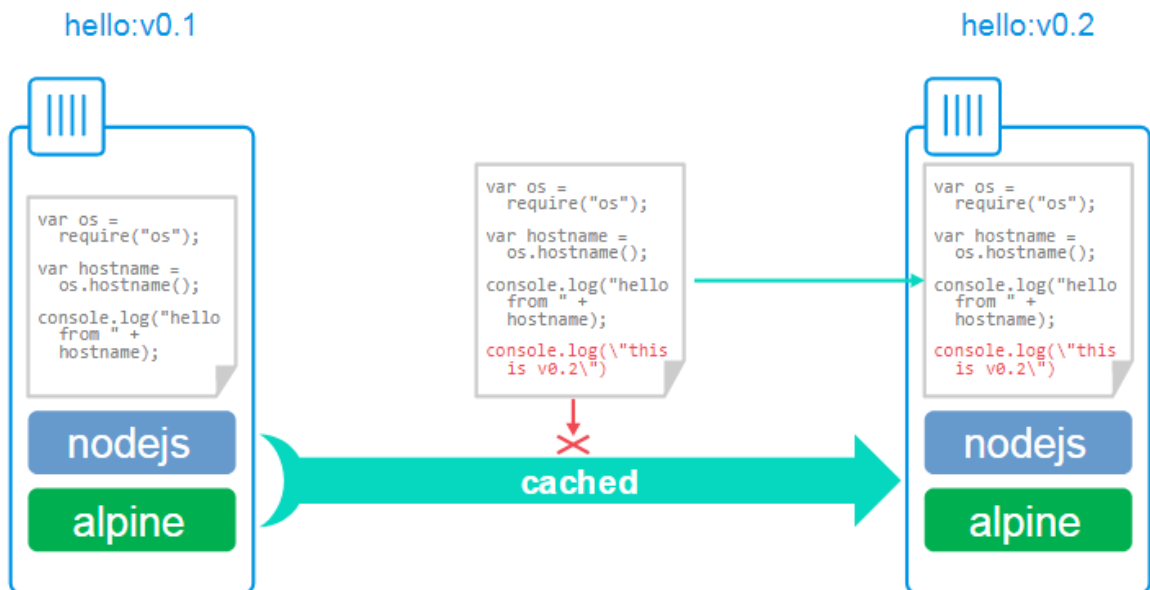
Dockerfile:

```
FROM alpine
RUN apk update && apk add nodejs
COPY . /app
WORKDIR /app
CMD ["node","index.js"]
```

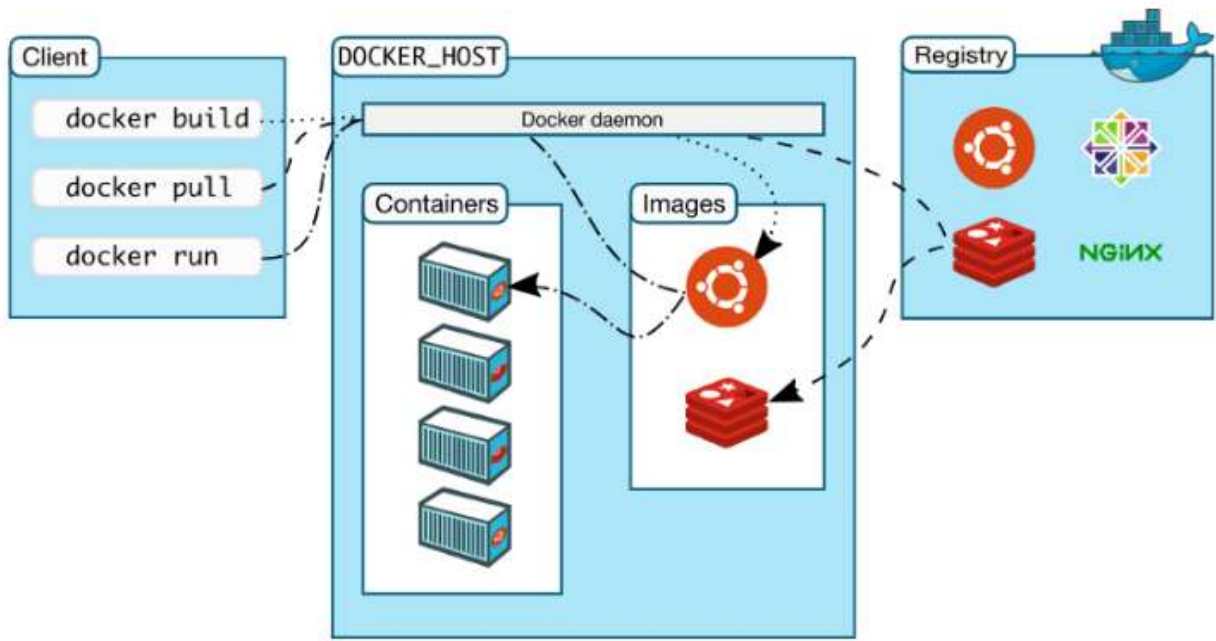


6. Слои docker образа

Layers & Cache



7. Создание нового docker образа



Модуль 4. Управление данными.

Образовательный результат: в результате прохождения данного модуля слушатели научатся:

1. формулировать задачи для каждого этапа обработки данных в проекте машинного обучения, в том числе:
 - a. сбор,
 - b. препроцессинг (предварительная обработка данных),
 - c. передача,
 - d. сохранение,
 - e. разведочный анализ данных,
 - f. выбор признаков для обучения модели и задачи.
2. выбирать инструменты для решения этих задач,
3. применять инструмент dvc для решения практических задач для управления данными.

В этом модуле:

Данные являются важной частью проекта машинного обучения. С использованием наборов данных (датасетов) обучается модель машинного обучения, эксплуатация решения напрямую связана с данными, на которые применяется обученная модель. Наличие данных создает дополнительные требования к проекту, так как необходимо предусмотреть место для хранения данных, запланировать вычислительные ресурсы для обработки данных, организовать мониторинг качества данных. Задачи инженера данных (Data Engineer, DataOps) в проекте машинного обучения могут быть очень разнообразными, например: обеспечение доступности (демократизация) данных, контроль и обеспечение качества данных, управление версиями датасетов, организация конвейера обработки данных. В этом модуле вы узнаете о задачах инженера данных подробнее, изучите необходимый инструментарий и научитесь его применять для решения задач MLOps.

Темы, изучаемые в модуле:

1. Задачи инженерии данных
2. Этапы обработки данных
3. Управление данными на примере dvc

Модуль 4. Юнит 1. Задачи инженерии данных

Введение: В этом юните вы познакомитесь с задачами, которые решает инженер данных (Data Engineer, DataOps), наиболее актуальными проблемами при работе с данными, методами предупреждения или решения этих проблем, базовым инструментарием.

Содержание юнита:

Данных становится все больше. Все чаще появляются новые области знаний, о появлении которых люди даже не задумывались еще несколько лет назад. В любой предметной области, в любой отрасли генерируется огромное количество данных. Эмпирическим путем подмечено, что количество данных сейчас увеличивается по экспоненте, то есть скорость изменения количества данных линейно зависит от текущего объема данных, чем больше становится данных, тем быстрее скорость их увеличения.

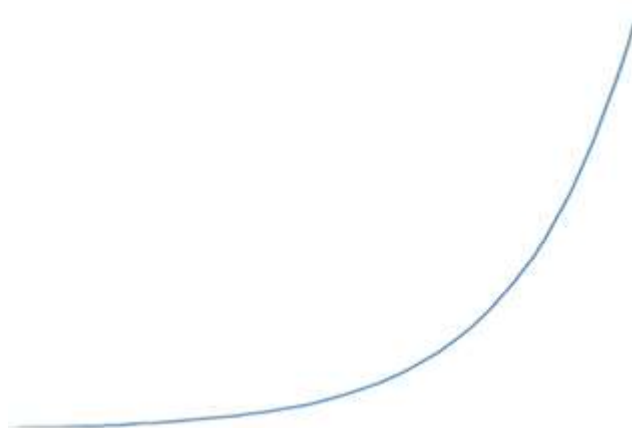


Рисунок “График экспоненты”.

Это очень серьезный тренд, и психологически, и технологически. Такой большой объем информации надо эффективно собирать, передавать, хранить. Однако самая большая проблема, связанная с данными, заключается не в их объеме, а в их неструктурированности. Данные появляются из различных не связанных между собой источников, в разных форматах, в разное время и по разному расписанию. *Поэтому перед использованием в практических задачах данные упорядочивают, преобразуют, приводят в форму, эффективную для хранения и использования. Все эти задачи решает инженер данных.*

Наличие данных отличает проекты машинного обучения от всех остальных проектов разработки программного обеспечения. Конечно, существуют информационные системы без машинного обучения, которые используют данные, ведь почти в каждом программном продукте есть своя база данных. **Однако особенность проектов машинного обучения состоит в том, что результат работы системы очень сильно зависит от качества и содержания данных на всех этапах жизненного цикла проекта: от обучения до эксплуатации.** Поэтому инженер данных является важным участником команды проекта машинного обучения. На практике часто получается, что задачи инженерии данных тесно пересекаются с задачами MLOps. Часто случается, что это один и тот же человек. Поэтому специалисту MLOps необходимо уметь разбираться в задачах, связанных с данными, и инструментах для их решения.

Основные задачи инженерии данных:

1. сбор и передача данных
 - а . организация сбора данных (по расписанию, по триггеру, в пассивном режиме, настройка доступа к данным),
 - б . создание каналов связи и настройка прикладного ПО для передачи данных, при необходимости создаются защищенные каналы связи с шифрованием,
 - с . мониторинг и анализ работы источников данных, выявление проблем в сборе и передаче данных,
2. анализ данных
 - а . обнаружение аномалий, пропусков, ошибок,
 - б . статистические характеристики для числовых данных, сравнение с предыдущими характеристиками, выявление отклонений в поведении данных,
3. сохранение полученных данных
 - а . проектирование базы данных
 - б . организация хранилища, настройка оборудования и программного обеспечения, организация доступа
 - с . контроль изменений в данных
 - д . защита информации, шифрование, резервирование
4. предобработка данных
 - а . приведение типов,
 - б . устранение ошибок,
 - с . обработка выбросов и аномалий,
 - д . преобразование имеющихся признаков,
 - е . создание новых признаков,
5. использование при обучении модели

Согласно исследованию Forbes, 80% времени инженеры данных тратят на подготовку данных (<https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says>).

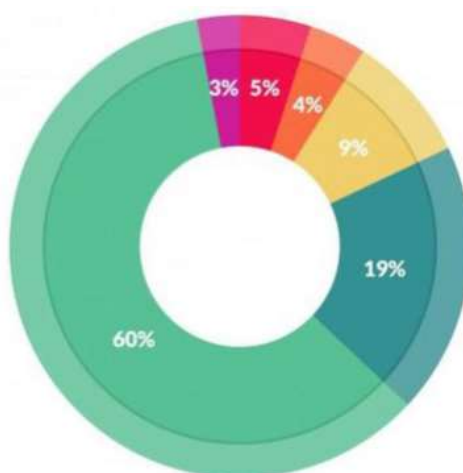


Рисунок “Распределение задач инженеров данных, согласно исследования Forbes”

Распределение объемов работ, согласно данным исследования, следующее:

- 3% создание тренировочных датасетов,

- 60% очистка и форматирование данных, организация структуры данных,
- 19% сбор датасетов,
- 9% поиск паттернов в данных,
- 4% улучшение алгоритмов,
- 5% остальные задачи.

Давайте разберем эти задачи подробнее.

1. Сбор и передача данных.

Источников для получения данных может быть очень много, у них различные форматы и регламенты получения данных.

Источник	Форматы	Объем наборов данных
Текстовый файл	txt, xlsx, csv/tsv	небольшой
IoT устройства (датчики, счетчики)	API, специализированные технологические протоколы (modbus)	небольшой
Информационные системы	API	средний
Базы данных	SQL, NoSQL,	большой

На этом этапе важно контролировать исполнение так называемого «контракта данных», который описывает структуру, тип данных, регламент получения данных, расписание.

Собранные данные необходимо передавать дальше для обработки и хранения. Для систем получения и передачи данных важным является правильная оценка пропускной способности канала передачи данных. Для этого необходимо правильно определять точки концентрации данных для обработки и хранения, чтобы своевременно обнаруживать слабые места.

2. Анализ данных.

Наибольшая польза в использовании данных заключается в возможности проведения анализа для выявления полезных свойств данных, закономерностей, скрытой информации, имеющейся в данных. С этой целью дата инженеры и инженеры машинного обучения анализируют структур имеющихся данных, исследуют имеющиеся признаки и конструируют новые признаки для обучения моделей.

Важно понимать, что дата аналитики не являются классическими программистами. Обычный язык программирования, которым владеют аналитики данных, это Python, который предназначен не для разработки, а для экспериментов. Результат работы исследователя данных это Jupyter-ноутбук, который не является программным кодом в классическом смысле, а является законченным исследованием, включающим таблицы,

графики, вычисления. Поэтому для исследователя данных надо очень точно формулировать - что является артефактом выполнения его части работы, чтобы не случился кризис контроля версий. Наиболее логичным артефактом на этапе анализа данных является

- набор данных (dataset), представляющий собой обработанные, структурированные, сохраненные в определенном формате данных, предназначенные для обучения модели машинного обучения,
- набор признаков, эффективный для обучения модели,
- статистическая информация о данных в целом и отдельных признаках.

3. Сохранение данных

Отдельно надо хранить данные для обучения, тестирования, валидации. Отдельно хранится разметка. Для проектов машинного обучения как правило требуются большие датасеты, которые занимают много места. Поэтому необходимо качественно планировать структуру датасета для хранения и необходимый объем памяти.

4. Обработка данных.

Сырые, необработанные данные всегда имеют скрытые проблемы: пропуски, разная кодировка, изменчивый характер, разные структуры таблиц, разные единицы измерения. Вручную такие ситуации обрабатывать и “чистить” тяжело, поэтому нужен предварительный анализ и автоматизация.

Из всего набора данных полезно выделить наиболее эффективные для модели признаки, либо сконструировать новые, этот процесс называется конструирование признаков (feature engineering).

Сырые данные, как правило, являются незамеченными, поэтому если модель предполагает “обучение с учителем”, то организуется процесс разметки данных, ручной или автоматический.

5. Использование данных при обучении и эксплуатации модели

Проблема в датасете то, что он эволюционирует. При эксплуатации модели машинного обучения происходит ухудшение качества работы модели и это связано прежде всего с тем, что данные в промышленных системах изменяются со временем: меняется их структура, статистические характеристики. Информационные системы изменяются в процессе эксплуатации, поэтому при проектировании, разработке и эксплуатации моделей машинного обучения необходимо учитывать, что датасеты могут изменяться, следовательно, надо контролировать все эти изменения.

Общая концепция, описывающая подходы к сбору, обработке и загрузке данных, называется ETL (Extract, Transfer, Load), подробно она описана в следующем юните.

Причина важности контроля потока обработки данных состоит в том, что в данных могут быть ошибки, а цена таких ошибок может быть очень большой. В целом, для проектов машинного обучения выполняется правило, что лучше предотвратить повреждение данных, чем потом исправлять ошибки во всей системе. *Есть проекты, затраты на обеспечение надежности и качество в которых могут достигать больших объемов в совокупной стоимости проекта (более 50%), однако стоимость устранения неполадок, в которых*

несоизмеримо больше, поэтому на такие затраты идут. К таким проектам относят и работу с данными. Поиск ошибок в проектах с данными в проектах ML может быть очень затратным по времени.

Причины «поломок» в данных:

1. Изменения в данных:
 - a. изменяются правила срабатывания триггеров событий (ивент-трекеры, технологические уставки),
 - b. не соблюдается «контракт данных» (меняется структура, типы данных)
 - c. сбои в инфраструктуре (отключение датчика, поломки в сети передачи данных)
2. Неправильная обработка данных:
 - a. изменение бизнес-логики обработки данных,
 - b. изменение схемы данных.

Если плохие данные уже поступили в модель, то она уже отравлена. Можно отлавливать проблему анализируя появление аномалий и проводить анализ корневых причин неисправности (Root Cause Analysis, RCA) с изучением ретроспективной корреляции, а можно заранее озаботиться тем, чтобы плохие данные не попали в систему.

Способы контроля качества данных в проекте машинного обучения:

1. версионирование всех элементов решения, затрагивающих данные: код, схемы и процедуры обработки (от трекинга до визуализации). Версионирование становится полезным тогда, когда данных и моделей становится много и информация о той или иной версии хранится только в головах инженеров.
2. оценка влияния изменений на данные
 - a. Assertions tests (тестирование правильности результатов на отдельных дтапах) позволяют устанавливать контрольные точки и правила
 - b. Data Diff (отслеживание изменений) для определения того, в каких именно данных произошли изменения и какой процесс мог их поменять.
 - c. Lineage (отслеживание взаимосвязей) позволяет проанализировать взаимосвязь в данных. Например, из каких таблиц берутся финальные данные, как появляются данные на предыдущем шаге и так далее.
3. Разработка регламента для каждого процесса, который делает изменения в данных, внедрение и строгий контроль выполнения требований этого регламента.

Применяя эти методы можно быстро изолировать проблему, откатить проблемное изменение назад, отслеживать историю изменений.

Инструменты для решения этих задач:

Инструмент	Ссылка	Описание
AirFlow	https://airflow.apache.org	Создание и контроль потоков автоматизации, например, обработки данных

		или обучения модели. Набор последовательно выполняемых операций представляется в виде направленного ациклического графа (DAG, Directed Acyclic Graph).
MLFlow	https://mlflow.org	Платформа для управления жизненным циклом моделей машинного обучения. Умеет версионировать данные.
Git-lfs	https://git-lfs.github.com	git large file storage, система git для управления большими файлами
dvc	dvc.org	data version control, система управления датасетами, моделями и конвейерами.

Идеальная организация проекта машинного обучения позволяет любому участнику команды проекта запускать весь проект «по щелчку» с минимальными накладными расходами. Возможность использовать при этом правильный датасет является ключевой, так как повторяемость эксперимента невозможно обеспечить на разных наборах данных.

Тест

1. Отметьте причины, по которым в процессе эксплуатации модели машинного обучения происходит незначительное ухудшение качества работы модели (0.25)
 - a. изменяется структура данных, например меняется набор параметров для наблюдения,
 - b. выход из строя источников данных (датчики, счетчики),
 - c. кратковременный сбой в работе каналов передачи данных,**
 - d. шум в данных, обусловленный ошибкой датчика**
2. Какие изменения в данных приводят к необходимости изменения модели машинного обучения? (0.25)
 - a. изменение бизнес-логики процесса,**
 - b. шум в данных,
 - c. сбой в каналах передачи данных,
 - d. изменение структуры данных**
3. Какие задачи относятся к инженерии данных? (0.25)
 - a. сбор данных,**
 - b. анализ и преобразование данных,**
 - c. подбор гиперпараметров модели машинного обучения,
 - d. организация хранения данных**
4. Какие инструменты используются для версионирования (0.25)
 - a. **dvc,**
 - b. git,**
 - c. jupyterhub,
 - d. python.

Итоги/выводы

Данные являются важнейшей частью проекта машинного обучения, поэтому роли инженера данных (data engineer, DataOps) и инженера машинного обучения (MLOps) очень тесно связаны. В этом юните вы узнали о задачах инженерии данных, к которым относятся:

- сбор,
- препроцессинг (предварительная обработка данных),
- передача,
- сохранение,
- разведочный анализ данных,
- выбор признаков для обучения модели и задачи.

Также вы кратко ознакомились с инструментами, которые использует в своей работе инженер данных. Эти темы более подробно будут рассмотрены в следующих юнитах.

Модуль 4. Юнит 2. Архитектура для работы с данными. Принципы работы ETL систем.

Введение: Правильная промышленная архитектура для работы с данными состоит из множества различных компонентов, решающих множество задач. В этом юните вы познакомитесь с основными понятиями ETL для работы с данными, узнаете из каких элементов обычно состоят системы для работы с данными и как правильно организовывать их работу.

Содержание юнита:

Данных стало гораздо больше, чем раньше, и их количество стремительно увеличивается. Типичное хранилище данных (DWH, Data Warehouse) государственной или коммерческой организации, использующей в своей деятельности аналитику данных, состоит из десятков тысяч связанных таблиц, в которых информация распределяется по этим таблицам, образуя сложные взаимосвязи между отдельными частями данных. Эти данные используются для создания моделей машинного обучения или бизнес-отчетов, на базе которых принимаются стратегические решения. Ясно, что неверные или неполные данные на входе приведут к неверным решениям на выходе.

Широко используемый термин ETL является сокращением от английских терминов Extract (достать, извлечь), Transform (преобразовать), Load (загрузить). Назначение ETL систем заключается в приведении к единой форме и обеспечении возможности использования данных из множества различных информационных систем. Необходимость применения принципов ETL обусловлена большим разнообразием источников, форматов представления данных и способов их получения. Например, на крупном предприятии данные берутся из множества используемых информационных систем, каждая система имеет свой формат данных и регламент их получения. Поэтому основные задачи ETL это:

1. сбор данных от различных источников,
2. обработка данных для последующего хранения и использования,
3. сохранение данных,
4. журналирование операций, связанных с обработкой данных, чтобы в любой момент была возможность установить происхождение конкретных величин,
5. применение данных, поиск полезных взаимосвязей, применение для автоматизации, отчетности, машинного обучения.

Логика работы ETL очень простая, состоит из простых операций. На каждом из этапов существуют типовые сложности и проблемы, о которых необходимо знать при создании систем обработки данных



1. **Загрузка данных** состоит в получении данных из разных источников произвольного качества для дальнейшей обработки.

Типичные проблемы: на этом этапе возможны различные ошибки, связанные с получением и передачей информации, например, сбой в формировании ответа на SQL запрос или перебои в канале передачи данных могут привести к тому, что фактически будет получено меньше данных, чем должно быть.

Для организации процесса загрузки важно уметь оценить продолжительность загрузки данных и увязать ее с существующими бизнес и технологическими требованиями. Например, если данных окажется очень много и процесс загрузки займет неделю, то это может не соответствовать бизнес-модели, которую описал заказчик проекта в техническом задании. В таком случае возможно придется уточнять технические требования или искать дополнительные возможности оптимизации загрузки.

2. Проверка качества данных и очистка от ошибок.

Типичные проблемы: ошибки могут быть самые разные, связанные как с человеческим фактором, так и со сбоями в хранении и передачи информации. Например, у данных могут оказаться разные форматы, когда часть информации представлена в числовом виде, а другая часть в текстовом. Возможны разные формы предоставления информации о датах, геолокации. Большую проблему представляют собой групповые ошибки в данных, когда каждое значение является легитимным, а группа значений содержит ошибку. Также плохо влияют на качество имеющиеся пропуски в данных, которые необходимо обрабатывать.

Создавать алгоритмы для проверки данных необходимо, чтобы исключить дальнейшее использование данных, способных повредить систему или нарушить процесс.

3. Приведение данных в соответствие с целевой моделью. На этом этапе обеспечивается единство структуры таблиц и форматов данных. Делается преобразование форматов и добавление необходимых таблиц. Имеющиеся данные могут быть преобразованы в другие данные, чтобы соответствовать целевой таблице.

Типичные проблемы: поскольку данные собираются из разрозненных источников, их использование в общей системе становится затруднительным, поскольку в разных системах разная структура таблиц, которые сложно соединить в общую структуру. Перед использованием данных проводится анализ их структуры и проверка соответствия “контракту данных”, который определяет формат, структуру, регламент получения данных от разных источников. Отклонение от “контракта данных” должно быть соответствующим образом обработано.

4. Процесс агрегации данных позволяет реализовать промежуточную логику между слоем сбора данных OLTP (Online Transaction Process, обработка транзакций в реальном времени) и слоем применения методов аналитики OLAP (Online Analytical Processing, онлайн обработка данных).

Типичные проблемы: на этом этапе обрабатываются огромные массивы информации, небольшие ошибки в логике обработки данных на этом этапе ведут к серьезным накладным расходам, как временным, так и вычислительным.

5. Выгрузка в систему, в которой будут использоваться данные.

6. **Использование данных.** Иногда применяется термин *drill-down*. На этом этапе применяются разные методы формирования отчетов, аналитики и машинного обучения.

Рассмотрим подробнее последний этап, на котором используется результат работы всего конвейера ETL. **Углубление в данные (drill-down data)** широко используется в различных контекстах. **В анализе данных обычно имеется в виду сосредоточение, погружение в структуру данных, чтобы получить информацию, полезную для поддержки принятия решений.** Проникновение в данные, как правило, начинается с верхних уровней, уровней наибольшего обобщения. Например, информация о количестве инцидентов компьютерной безопасности это всего одно число, которое само по себе не позволит проанализировать динамику компьютерных атак в течение года. Для получения информации о такой динамике необходимо получить более детализированные данные. Углубление (drill-down) в данные является одной из важнейших операций с так называемыми OLAP-кубами, она позволяет осуществлять навигацию между различными уровнями агрегирования данных. **OLAP-куб это многомерный массив данных, как правило, разреженный и долговременно хранимый, используемый в OLAP.** Индексам массива соответствуют измерения или оси куба, а значениям элементов массива меры куба.

Давайте рассмотрим подробнее несколько важных элементов из ETL архитектуры:

- **Озеро данных (data lake)** это большой репозиторий необработанных исходных данных, как неструктурированных, так и частично структурированных. Данные собираются из различных источников и хранятся в исходном виде, без каких-либо преобразований. Поэтому для дальнейшего использования этих данных требуется предварительная подготовка, например, очистка, форматирование, устранение выбросов. В озерах данных могут храниться как объекты структурированной информации (строки и таблицы из реляционных баз данных), так и полуструктурированной (например, xml или json файлы) и даже совсем неструктурированной (текстовые файлы, изображения и видео). Для данных в озере данных как правило фиксируются источник этих данных, владелец, регулярность получения, время хранения. Также могут присутствовать метаданные (то есть данные о данных). Несмотря на большие накладные расходы, связанные с хранением неструктурированной информации, озера данных дают выигрыш в том, что для их создания не применяются никакие аналитические алгоритмы, а современные средства хранения позволяют эффективно сохранять все большие объемы неструктурированной информации.
- **Хранилище данных (data warehouse)** представляет собой данные, агрегированные из разных источников в единый центральный репозиторий, который унифицирует их по качеству и формату. Далее эти данные могут быть использованы в таких сферах, как data mining, искусственный интеллект, машинное обучение и в бизнес-аналитике.
- **Витрина данных (data mart)** это хранилище данных, предназначенное для определенного круга пользователей в организации. Витрина данных может использоваться руководством компании для анализа производительности и

контроля выполнения планов. Наборы данных в витрине данных часто используются в режиме реального времени для аналитики и получения практических результатов.

Некоторые важные инструменты, о которых необходимо знать:

- **Kafka** получает неотфильтрованные и необработанные сообщения и функционирует как принимающий узел в озере данных. Kafka обеспечивает надежный и высокопроизводительный сбор сообщений. Кластер Kafka обычно содержит несколько разделов для сырых данных, обработанных (для потоковой обработки) и недоставленных или искаженных данных.
- **Flink** принимает сообщение из узла с необработанными данными от Kafka, фильтрует данные и делает, при необходимости, предварительное обогащение. Затем данные передаются обратно в Kafka (в отдельный раздел для отфильтрованных и обогащенных данных). В случае сбоя, или при изменении бизнес-логики, эти сообщения можно будет вызвать повторно, т.к. что они сохраняются в Kafka. Это распространенное решение для в потоковых процессов. Между тем, Flink записывает все неправильно сформированные сообщения в другой раздел для дальнейшего анализа.
- Для управления потоками данных (data-flows) и ETL можно использовать **Apache Airflow**, который позволяет запускать конвейеры (pipeline) обработки данных с использованием Python и объектов типа Directed Acyclic Graph (DAG). Airflow позволяет гибко настраивать конвейеры и отслеживать выполнение задач через графический интерфейс. Также Airflow можно использовать для обработки внешних данных.
- **Spark** может использоваться для обогащения сырых отфильтрованных данных в соответствии с бизнес-задачами, после чего эти данные могут использоваться дата-аналитиками и бизнес-аналитиками. Для большой команды проекта Spark можно использовать для настройки многопользовательских интерфейсов для работы с данными, их сбором и анализом.
- **Hive** это система управления базами данных на основе системы Hadoop, позволяет реализовать стандартный интерфейс баз данных (например, SQL) для получения данных, хранящихся в Hadoop.
- **ElasticSearch** инструмент полнотекстового поиска в наборах данных, позволяет за счет индексации данных сделать такой поиск эффективнее, является важной частью ELK тека (ElasticSearch, Logstash, Kibana), в котором Logstash используется для приема и обработки log файлов, а Kibana для визуализации.
- HDFS файловая система **hadoop** для хранения больших объемов информации.

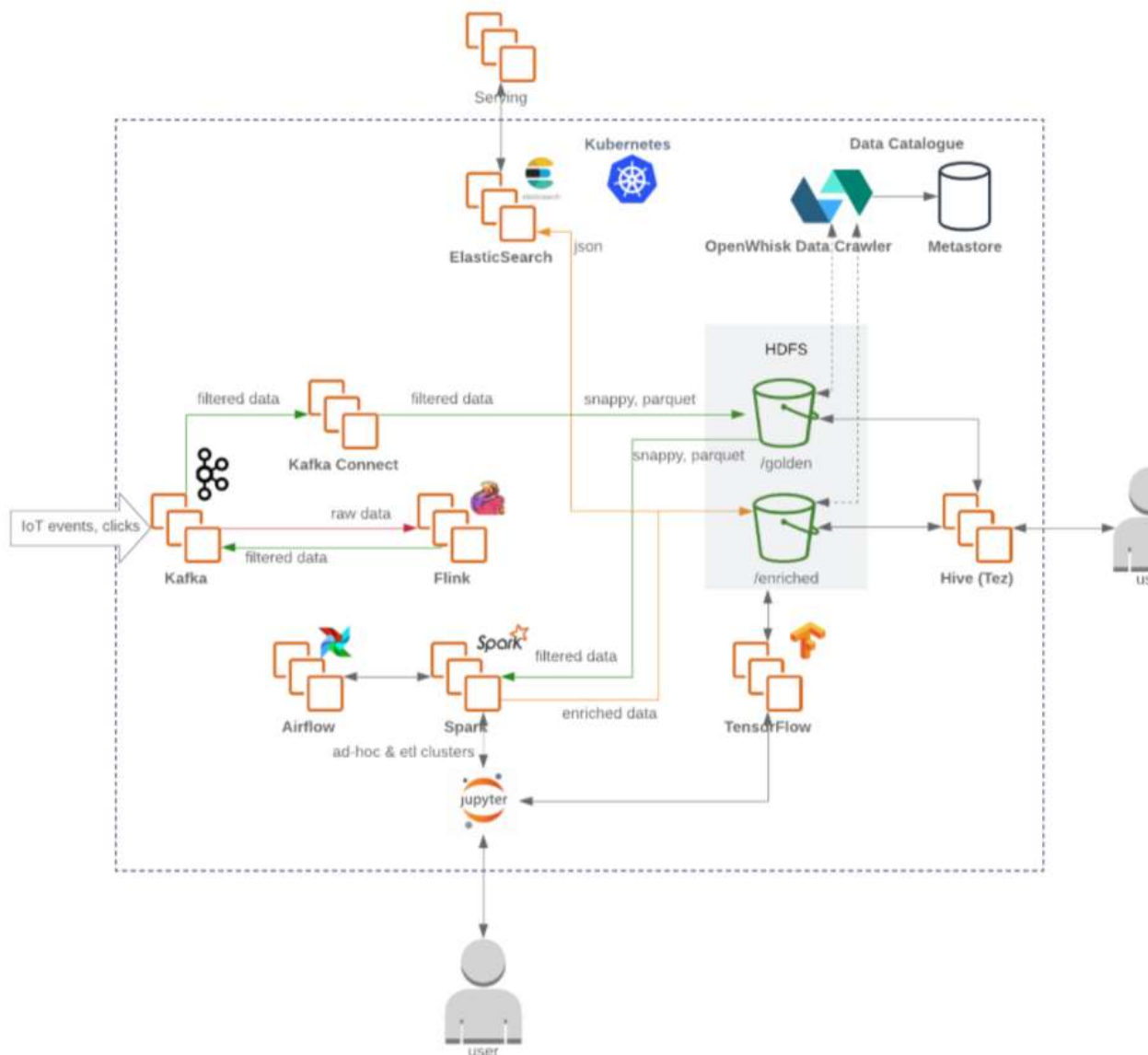


Рисунок “Вариант целевой архитектуры для организации ETL процессов, источник <https://habr.com/ru/company/opsguru/blog/514322/>”

Тест

1. Из каких операций состоит ETL? (0.25)
 - a. **Extract, Transform, Load**
 - b. Extract, Transfer, Load
 - c. Enhance, Transit, List
 - d. Enter, Test, Leave
2. Что такое витрина данных? (0.25)
 - a. веб-интерфейс для просмотра данных
 - b. специальный формат графика
 - c. **специальная форма хранилища данных**
 - d. элемент онлайн-площадки для торговли и обмена данными
3. Какие объекты являются видами корпоративных хранилищ данных? (0.25)
 - a. **data lake**
 - b. data river

- c. **data warehouse**
 - d. data office
4. Что может быть причиной ошибок в данных? (0.25)
- a. **несовпадение форматов**
 - b. **человеческий фактор**
 - c. **сбои в передаче информации**
 - d. **ошибки в преобразовании данных**

Итоги/выводы

В этом юните вы изучили архитектуру ETL, ее основные задачи и этапы, используемые в организациях, применяющих аналитику данных. На практике в проектах машинного обучения данные могут поступать из различных источников. В этом смысле полезно знать архитектуру ETL как один из возможных источников для получения данных, а также понимать ее методы, чтобы по аналогии создавать похожие механизмы для работы с данными в своих проектах.

Для организации исследований в небольших проектах часто достаточно более простых и в основном open-source инструментов, некоторые из которых мы рассмотрим в следующих юнитах.

Модуль 4. Юнит 3. Инструменты для управления данными. Примеры использования dvc

Введение: Управление данными важным процессом в проектах машинного обучения. Наборы данных меняются в процессе работы над проектом. Разные участники команды используют разные версии датасетов для своих задач. Без системы контроля версий этим процессом невозможно управлять. **Одним из широко распространенных инструментов контроля версий датасетов, а также моделей машинного обучения, является dvc, работа с которым подробно рассмотрена в этом юните.**

Содержание юнита:

В ходе работы над проектом машинного обучения наборы данных постоянно меняются. Причины этих изменений могут быть как внутренними, обусловленные экспериментальным характером процесса построения модели, так и внешними, связанными с изменениями структуры и характеристик данных.

Внутренние причины изменений в данных:

- в результате обработки имеющихся данные для улучшения качества работы модели, проведена очистка данных,
- удаление или обработка выбросов и аномалий,
- шкалирование данных,
- приведение к наиболее эффективному формату для хранения,
- создание новых признаков, позволяющие лучше обучать модель,
- удаление неэффективных признаков.

Внешние причины изменения данных:

- обогащение данных за счет использования новых дополнительных источников информации,
- обновление данных, получение новых данных с объекта эксплуатации,
- изменения в структуре данных, отклонение от “контракта данных”.

Все это приводит к появлению новых наборов данных, которые необходимо сохранять, чтобы обеспечить повторяемость результата работы модели. *Например, если новый участник команды попытается использовать одну из обученных моделей, но при этом использует неправильный набор данных, то модель может аварийно завершить работу или, что еще хуже, выполнить работу с ухудшением качества, причину которого обнаружить будет очень трудно.*

Для решения этой проблемы в разработке программного обеспечения традиционно используют **системы контроля версий**. Система контроля версий записывает изменения в файле, происходившие в течение всего периода работы над проектом, запоминает дату изменения и автора, позволяет в любой момент вернуться обратно к нужной версии. Централизация системы контроля версий позволяет многим разработчикам иметь доступ к версиям, синхронизировать работу больших команд, управлять изменениями. Важно, чтобы эта система была распределенной, а не хранила всю информацию в одном месте, иначе выход центрального узла из строя может привести к потерям всей информации. Одна из самых популярных систем контроля версий это **git**. В распределенных git системах

каждая копия репозитория является полной копией (бэкапом) всех данных проекта, опубликованных в репозитории.

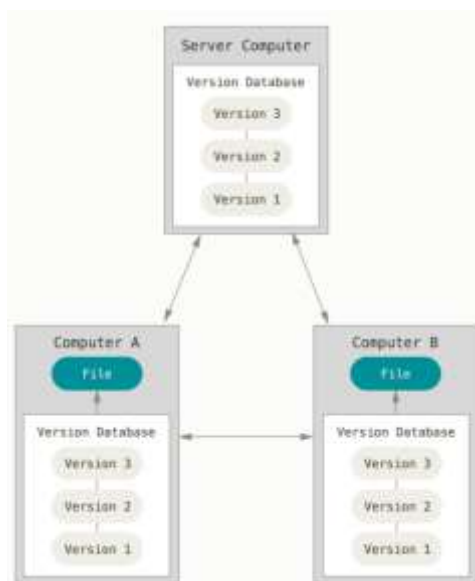


Рисунок “Схема работы распределенной системы версионирования”.

Такой подход позволяет делать разные ветки изменений в файлах проекта и управлять этой системой веток, обеспечивая “нелинейность” процесса разработки. Наиболее популярными инструментами для контроля изменений в процессе разработки программного обеспечения являются github (<https://github.com>) и gitlab (<https://gitlab.com>). Эти инструменты вы подробнее рассматривали в предыдущих курсах. Также легко можно найти информацию о применении этих инструментов в Интернет.

Для наборов данных в проектах машинного обучения контроль изменений так же важен, поэтому появились аналоги git для управления данными в проектах машинного обучения. Одним из наиболее популярных инструментов в настоящее время является **утилита dvc** (сокращение от data version control). Разработкой и поддержкой dvc занимается компания Iterative.ai (<https://iterative.ai/>). Проект разрабатывается в формате open-source, git репозиторий проекта находится здесь: <https://github.com/iterative/dvc>. Первая версия dvc появилась в 2017 году, однако, несмотря на относительно небольшой возраст, у решения dvc сформировалось большое сообщество пользователей и разработчиков.

dvc предоставляет следующие возможности:

- контроль изменений в наборах данных,
- версионирование моделей,
- контроль и версионирование экспериментов, благодаря которому можно быстро восстановить любой эксперимент,
- создание потоков операций для автоматизации (workflow), включая перевод обученной модели в эксплуатацию,
- хранение артефактов в локальном кэше или на локальном сервере с доступом по ssh,
- загрузка артефактов в удаленное хранилище, например,
 - Amazon S3,

- Google Drive,
- MS Azure,
- Yandex DataSphere (<https://cloud.yandex.ru/docs/datasphere/concepts/dvc-and-git>),
- специализированный набор инструментов для воспроизводимости экспериментов.

Таким образом, кроме контроля версий датасетов `dvc` практически может быть полезен и для автоматизации и контроля проведения экспериментов, и для перевода модели машинного обучения в эксплуатацию.

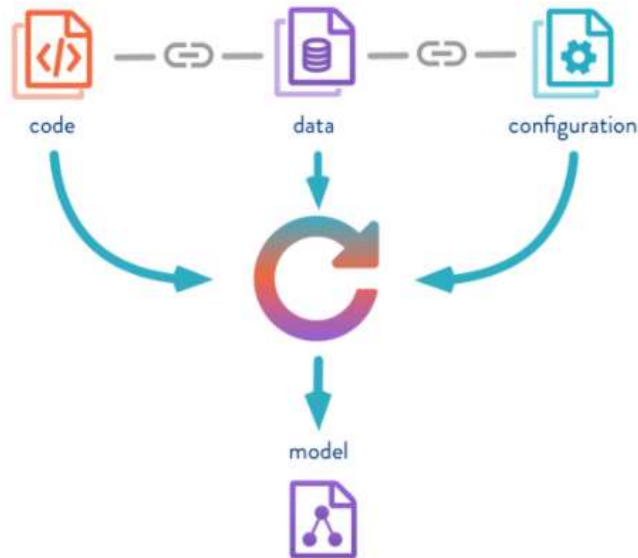


Рисунок “Использование `dvc` (с сайта `dvc.org`)”

Утилита `dvc` работает только совместно с `git`. Также многие стандартные команды `git` используются в `dvc`, например, `add`, `commit`, `status`, `pull`, `push`.

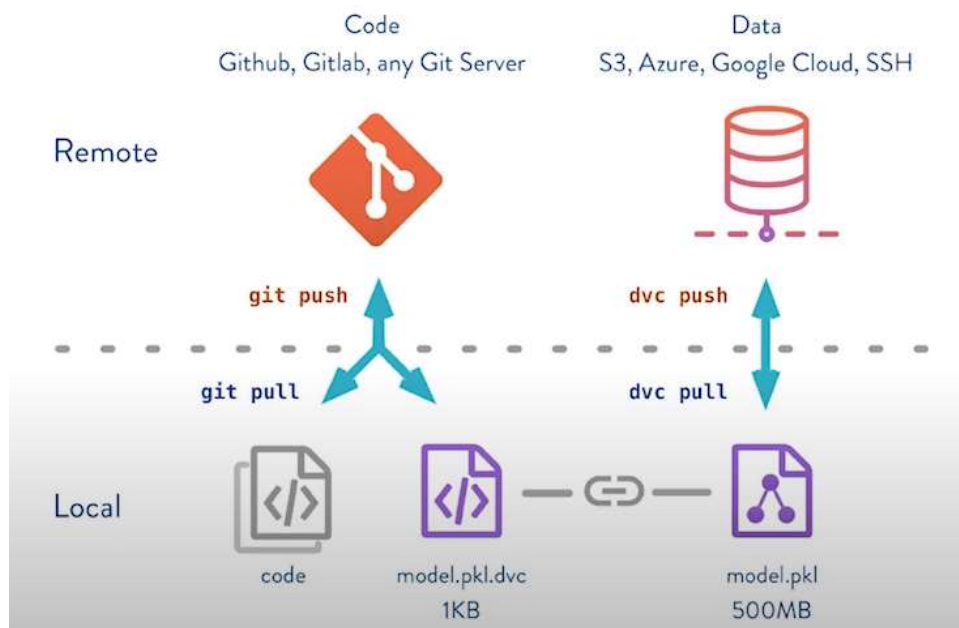


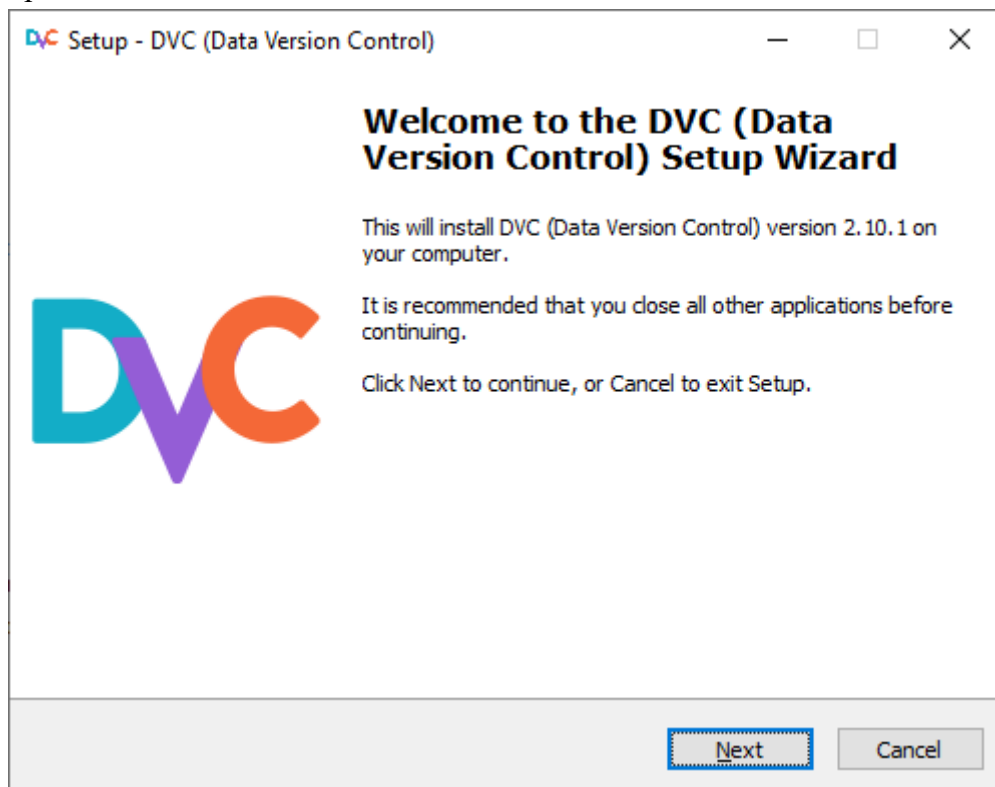
Рисунок “Совместная работа `git` и `dvc` (с сайта `dvc.org`)”

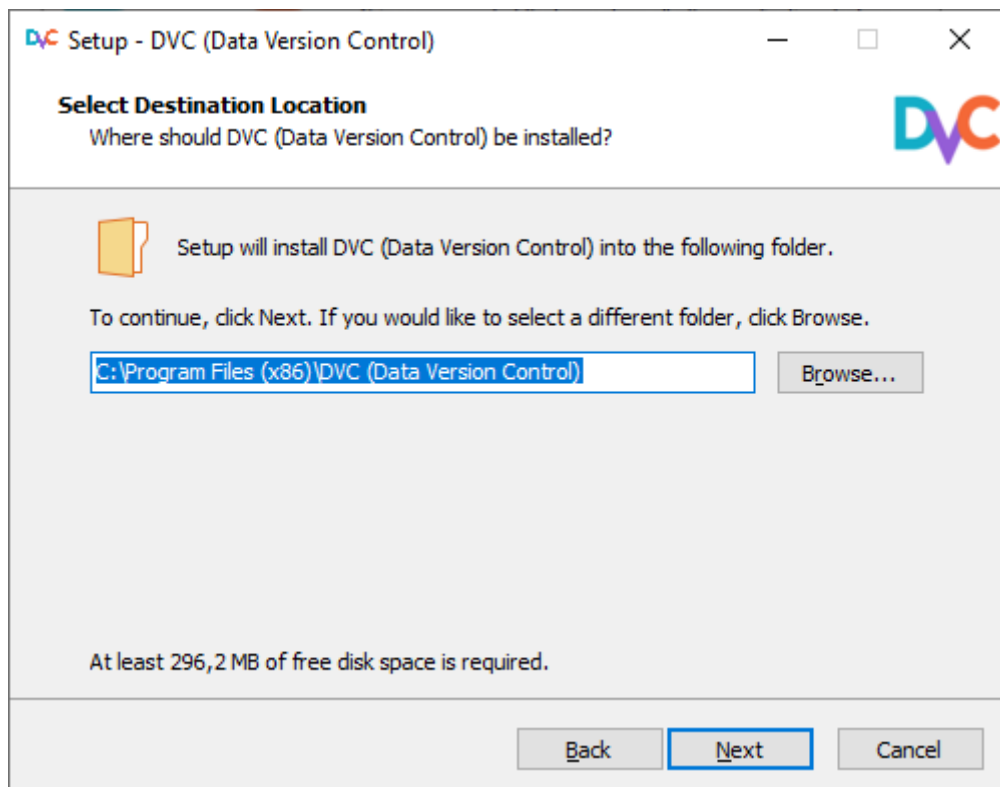
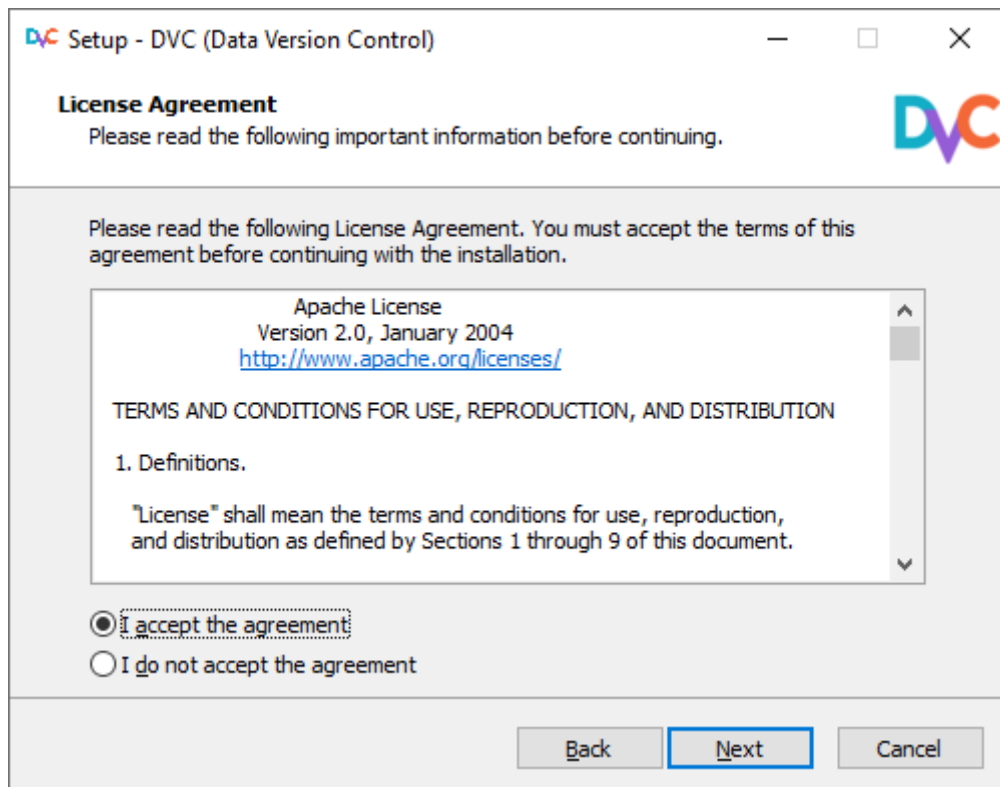
Установка dvc очень проста, поддерживается установка и использование для различных операционных систем. Например, в linux системе установка может быть выполнена с использованием установщика pip.

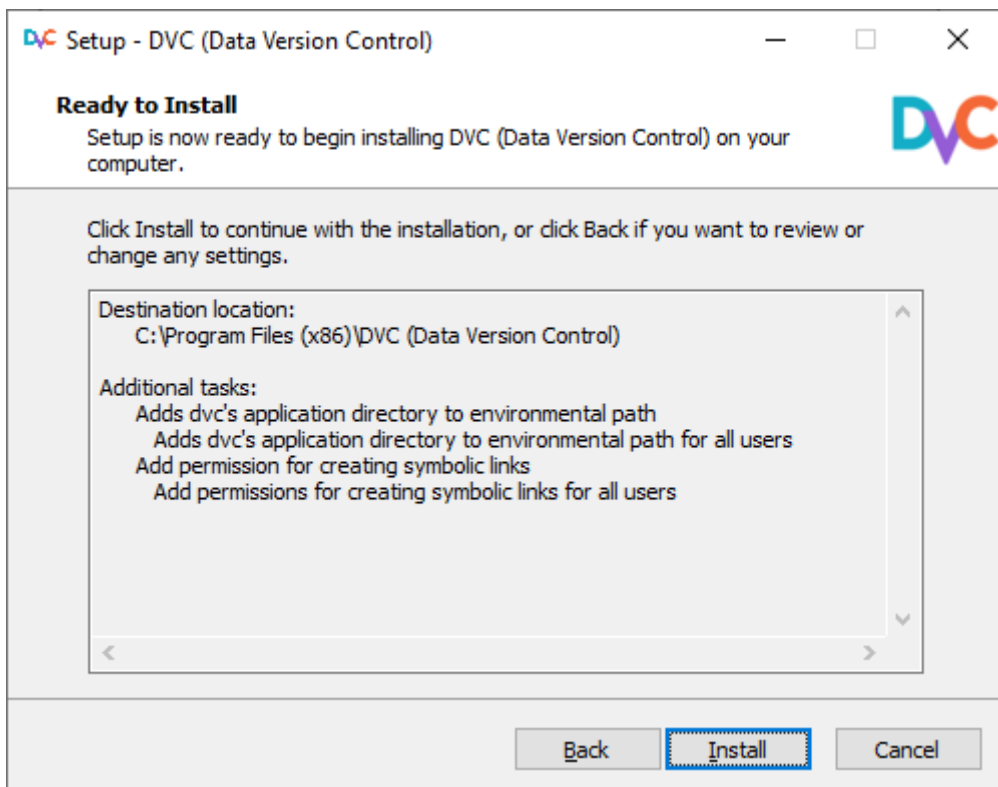
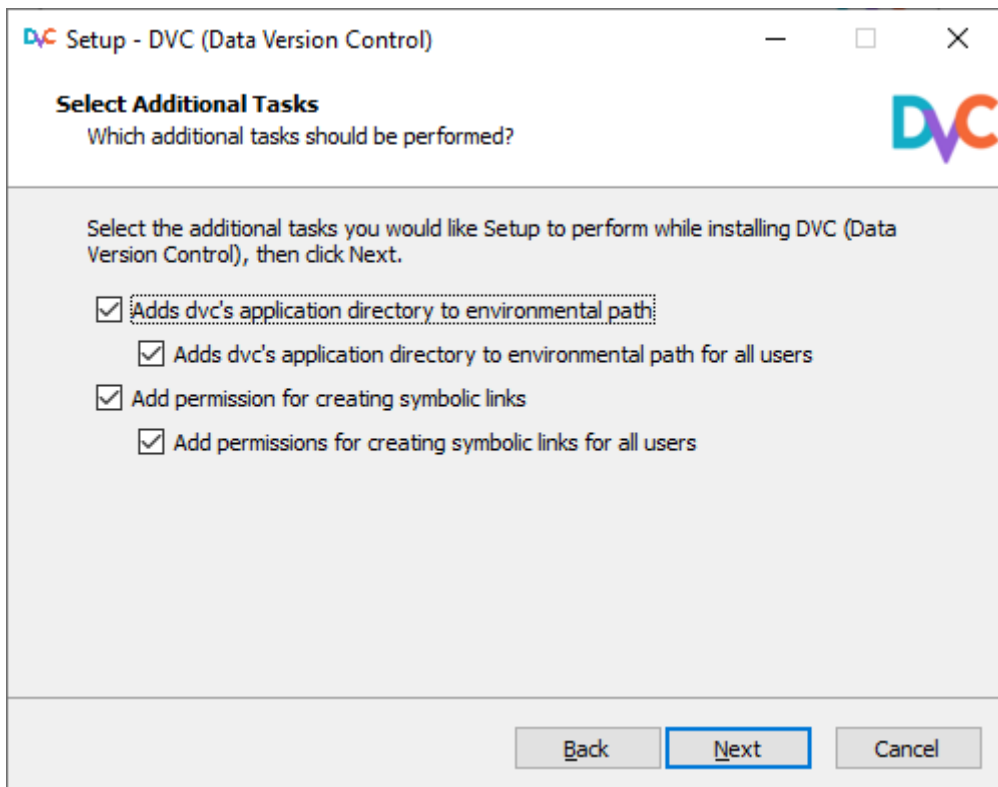
```
ubuntu@ip-172-31-95-38:~$  
ubuntu@ip-172-31-95-38:~$ pip freeze | grep dvc  
ubuntu@ip-172-31-95-38:~$  
ubuntu@ip-172-31-95-38:~$ pip install dvc
```

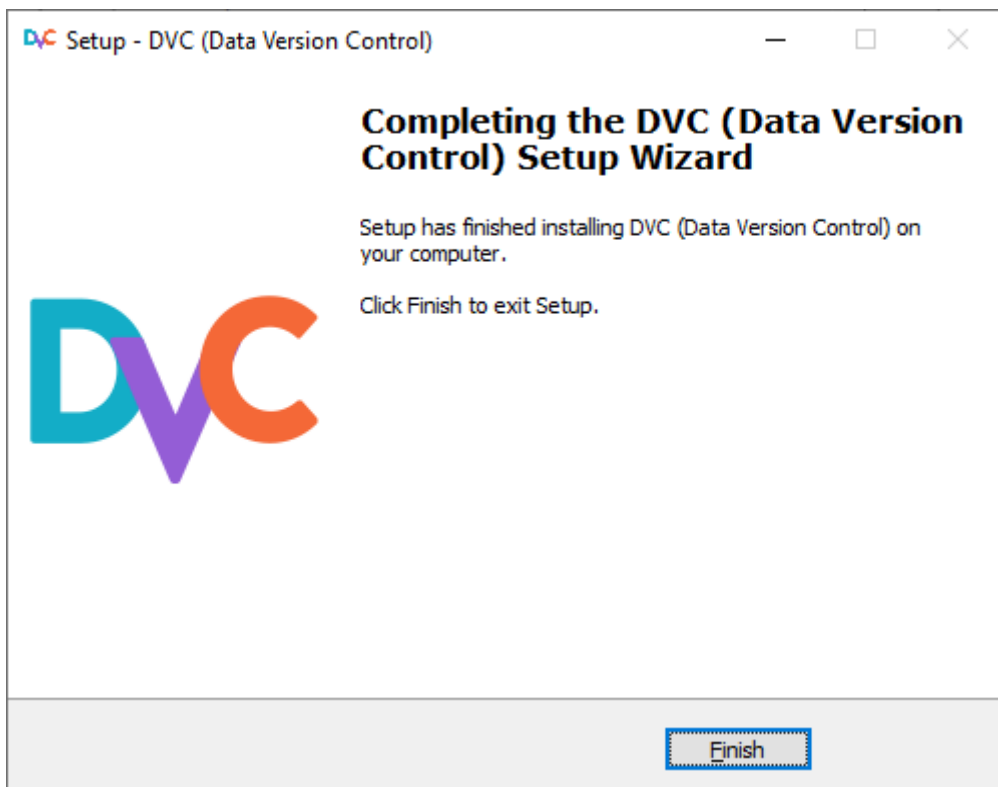
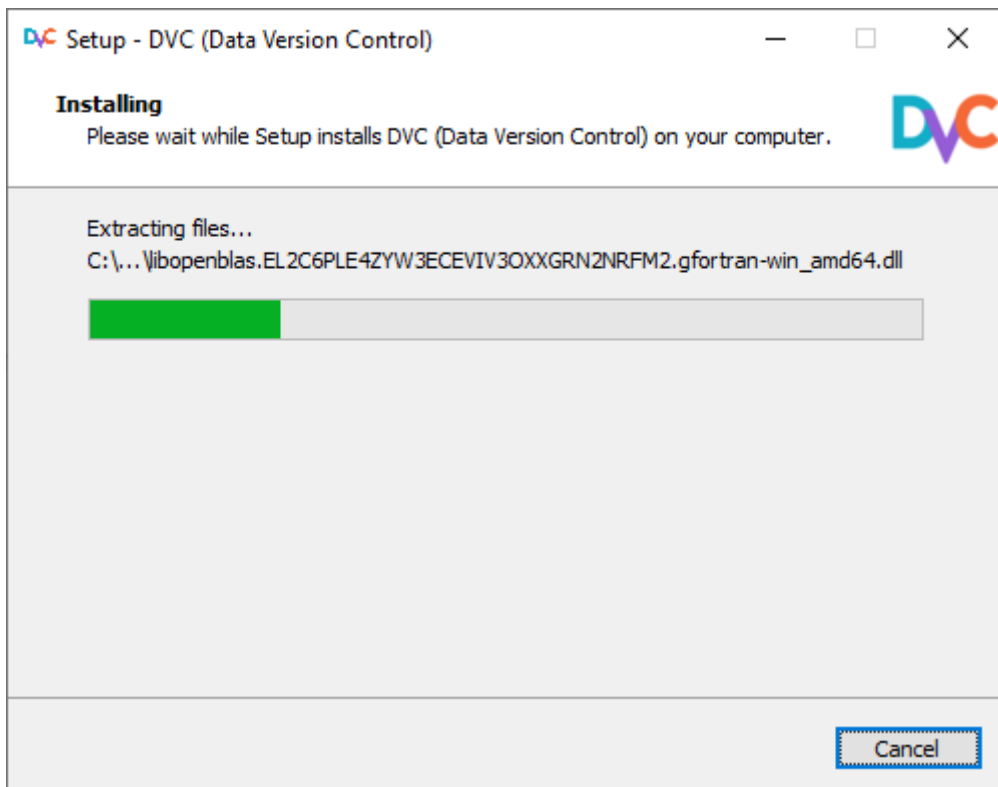
```
ubuntu@ip-172-31-95-38:~$ pip freeze | grep dvc  
dvc==2.10.1  
dvc-render==0.0.4  
ubuntu@ip-172-31-95-38:~$ █
```

Для работы с dvc в операционной системе Windows необходимо скачать установочный exe файл с официального сайта dvc.org. После этого процедура установка стандартная для windows приложений.









Теперь можно пользоваться dvc, например в PowerShell

```
PS C:\Users\dvc --version  
2.10.1
```

Далее мы рассматриваем примеры работы в linux операционных системах. Если после установки dvc сразу попробовать начать работу, то появится сообщение об ошибке, так как необходимо инициализировать git.

```
ubuntu@ip-172-31-95-38:~$ dvc init
ERROR: failed to initiate DVC - /home/ubuntu is not tracked by any supported SCM tool (e.g. Git).
ubuntu@ip-172-31-95-38:~$
```

Для того, чтобы устранить эту проблему и начать пользоваться dvc необходимо сделать рабочую папку git репозиторием с использованием команды

git init

после этого инициализация dvc проходит успешно.

```
ubuntu@ip-172-31-95-38:~$ dvc init
ERROR: failed to initiate DVC - /home/ubuntu is not tracked by any supported
ubuntu@ip-172-31-95-38:~$
ubuntu@ip-172-31-95-38:~$ mkdir git
ubuntu@ip-172-31-95-38:~$ mkdir git/project
ubuntu@ip-172-31-95-38:~$ cd git/project/
ubuntu@ip-172-31-95-38:~/git/project$ git init
Initialized empty Git repository in /home/ubuntu/git/project/.git/
ubuntu@ip-172-31-95-38:~/git/project$ ls -la
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr  7 13:11 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr  7 13:10 ..
drwxrwxr-x 7 ubuntu ubuntu 4096 Apr  7 13:11 .git
ubuntu@ip-172-31-95-38:~/git/project$ dvc init
Initialized DVC repository.

You can now commit the changes to git.

-----
DVC has enabled anonymous aggregate usage analytics.
Read the analytics documentation (and how to opt-out) here:
<https://dvc.org/doc/user-guide/analytics>
-----

what's next?
-----
- Check out the documentation: <https://dvc.org/doc>
- Get help and share ideas: <https://dvc.org/chat>
- Star us on GitHub: <https://github.com/iterative/dvc>
ubuntu@ip-172-31-95-38:~/git/project$ ls -la
total 20
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr  7 13:11 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr  7 13:10 ..
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr  7 13:11 .dvc
-rw-rw-r-- 1 ubuntu ubuntu 139 Apr  7 13:11 .dvcignore
drwxrwxr-x 7 ubuntu ubuntu 4096 Apr  7 13:11 .git
ubuntu@ip-172-31-95-38:~/git/project$
```

Для более подробного практического ознакомления с инструментом dvc для контроля версий наборов данных давайте на простом “игрушечном” примере сделаем стандартный набор операций с данными: создание, изменение, сохранение, загрузка нужной версии данных. Для этого создадим рабочую папку проекта в этой директории папку для датасетов.

```
ubuntu@ip-172-31-95-38:~$
ubuntu@ip-172-31-95-38:~$
ubuntu@ip-172-31-95-38:~$ mkdir project
ubuntu@ip-172-31-95-38:~$ mkdir project/datasets
ubuntu@ip-172-31-95-38:~$ cd project/
ubuntu@ip-172-31-95-38:~/project$ ls -la
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 16 11:07 .
drwxr-xr-x 7 ubuntu ubuntu 4096 Apr 16 11:06 ..
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 16 11:07 datasets
ubuntu@ip-172-31-95-38:~/project$
```

Сделаем эту папку git репозиторием с помощью команды

git init

при этом появится служебная папка .git. После этого можно выполнить команду

Вы видите подсказку системы, что для трекинга датасетов необходимо в git добавить файл .gitignore, который содержит информацию о том, чего не должно быть в git репозитории, а также файл datasets.dvc следующего содержания

```
ubuntu@ip-172-31-95-38:~/project$ cat datasets.dvc
outs:
- md5: 8fcbe9c1d38df6d0e3fb8d4ac4de310a.dir
  size: 12
  nfiles: 1
  path: datasets
ubuntu@ip-172-31-95-38:~/project$ █
```

Это служебный файл, который содержит хеш наблюдаемого объекта. А если заглянуть в файл .gitignore, который содержит перечень правил, запрещающих публикацию в git, то вы увидите там репозиторий /datasets, в котором хранятся наши данные. И это правильно, так как в git публиковать эти данные не требуется. Давайте добавим эти файлы в набор для commit и сделаем commit в git

```
ubuntu@ip-172-31-95-38:~/project$ git status
on branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore
  datasets.dvc

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ip-172-31-95-38:~/project$ git add .gitignore datasets.dvc
ubuntu@ip-172-31-95-38:~/project$ git commit -m "Put datasets under control"
[master 898fc0d] Put datasets under control
Committer: Ubuntu <ubuntu@ip-172-31-95-38.ec2.internal>

2 files changed, 6 insertions(+)
create mode 100644 .gitignore
create mode 100644 datasets.dvc
ubuntu@ip-172-31-95-38:~/project$ █
```

Теперь мы опубликовали в git данные для того, чтобы работать с датасетами, но не сами датасеты. Для хранения датасетов нам надо определить хранилище. dvc поддерживает стандартные хранилища: Amazon S3, MS Azure, Google Drive, а также хранилище на сервере с доступом по ssh. Мы рассмотрим пример организации хранилища на Google Drive. Для этого понадобится аккаунт в Google, с использованием которого можно воспользоваться сервисом “Google Disc”. Создадим в этом сервисе папку для хранения версий наборов данных и установим права доступа, для возможности записи в папку необходимо установить роль “Редактор”. После этого можно получить ее идентификатор папки для использования в dvc.

Google

Введите поисковый запрос или URL

Почта Картинки

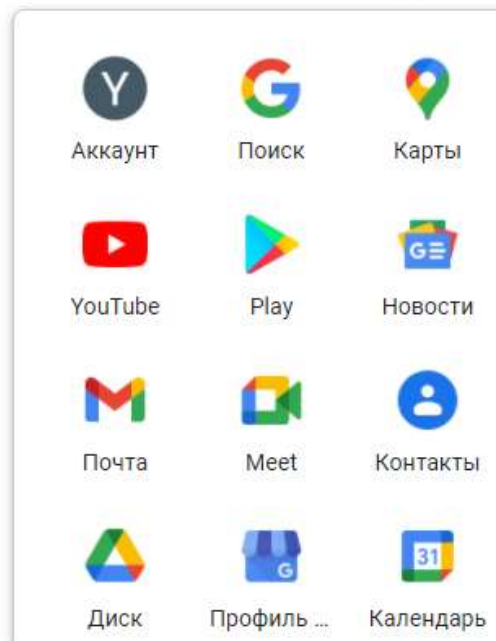


Рисунок “Запуск сервиса Google Disk в google.com”

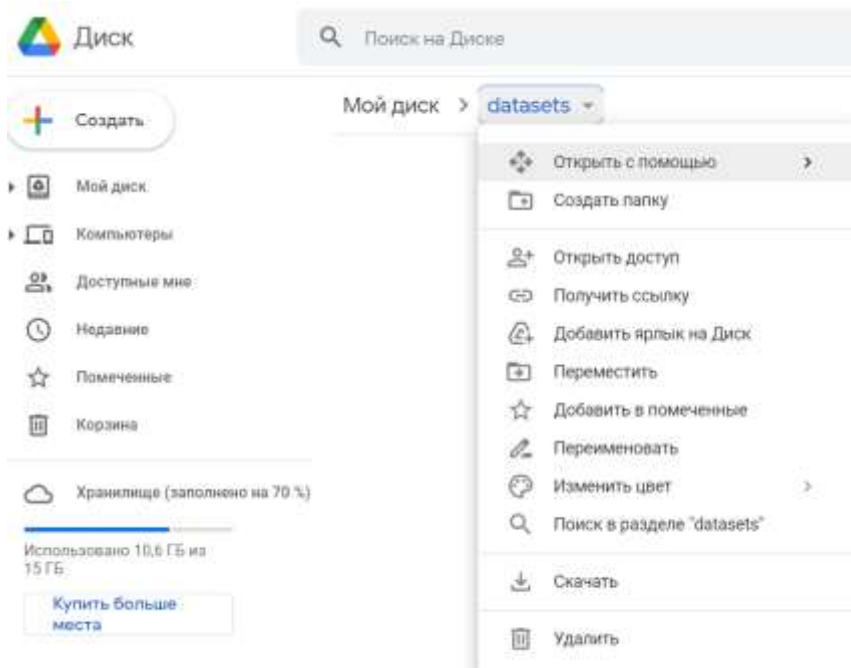


Рисунок “Создание папки в Google Disk”

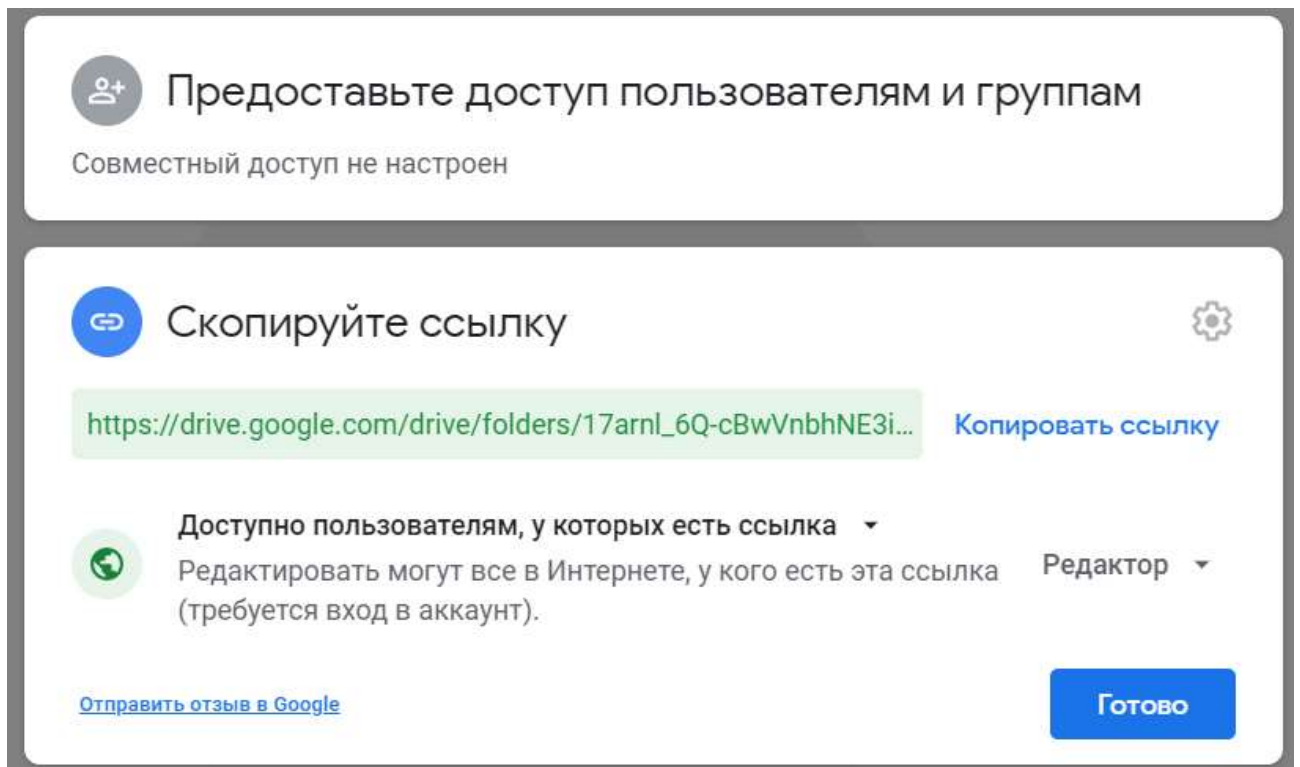


Рисунок “Редактирование прав доступа к папке в Google Disk”

Нам понадобится уникальный идентификатор этой папки для использования в dvc (у каждого пользователя и каждой папки он свой)

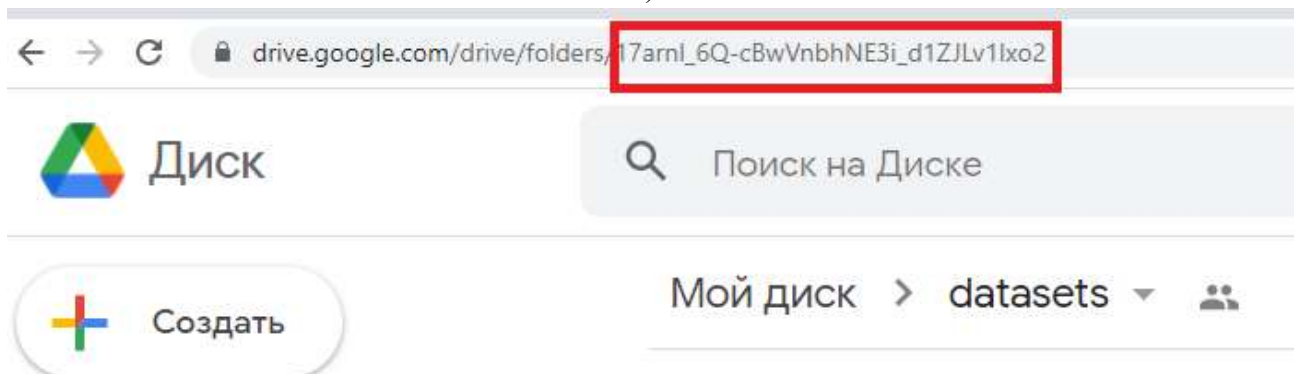


Рисунок “Уникальный идентификатор папки в Google Disk для хранения датасетов”

Это идентификатор используется при вызове команды dvc для того, чтобы добавить этот диск в перечень удаленных репозиториев для хранения датасетов

dvc add remote “внутреннее имя репозитория” “уникальный идентификатор”

```
ubuntu@ip-172-31-95-38:~/project$ dvc remote add "mydisk" gdrive://17arnl_6Q-cBwVnbhNE3i_d1ZJLv1Ixo2
ubuntu@ip-172-31-95-38:~/project$ dvc remote list
mydisk gdrive://17arnl_6Q-cBwVnbhNE3i_d1ZJLv1Ixo2
ubuntu@ip-172-31-95-38:~/project$ git status
On branch master
changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   .dvc/config

no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-95-38:~/project$ git add .dvc/config
ubuntu@ip-172-31-95-38:~/project$ git commit -m "Google Disk was added as remote dataset folder"
[master 580b053] Google Disk was added as remote dataset folder
 1 file changed, 2 insertions(+)
ubuntu@ip-172-31-95-38:~/project$
```

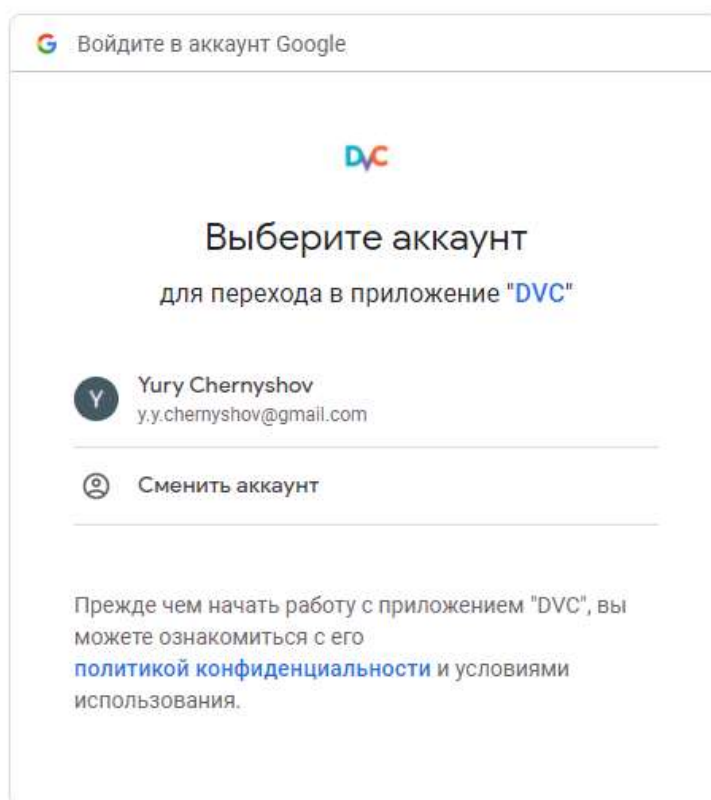
Необходимо учитывать, что для доступа к ресурсам Google требуется авторизация, про которую подробно можно почитать тут: <https://dvc.org/doc/user-guide/privacy>. Мы для упрощения выполним самый простой способ авторизации, когда проверка прав доступа выполняется вручную. Выполнив команду `dvc push` для публикации датасета в удаленном репозитории, который является аналогом команды `git push`, вы увидите следующее сообщение

```
ubuntu@ip-172-31-95-38:~/project$ dvc push -r mydisk
0% Querying remote cache|
Go to the following link in your browser:

  https://accounts.google.com/o/oauth2/auth?client_id
%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.appdata&acce

Enter verification code:
```


в котором предлагается пройти по указанной ссылке, получить код верификации и ввести его для продолжения авторизованной работы. После перехода по ссылке вы увидите следующее окно (каждый пользователь будет видеть свои персональные данные)



После входа в аккаунт необходимо разрешить приложению `dvc` взаимодействовать с Google Disk



Приложению "DVC" нужны права доступа к вашему аккаунту Google

 y.y.chernyshov@gmail.com

Выберите, какие права доступа вы хотите
предоставить приложению "DVC"



Просмотр, создание, изменение и
удаление всех файлов на Google Диске
[Подробнее...](#)



Просмотр, создание и удаление
информации о своих параметрах на вашем
Google Диске [Подробнее...](#)

Убедитесь в надежности сервиса "DVC"

Этот сайт или приложение сможет получить доступ к
конфиденциальной информации. Посмотреть или
удалить приложения и сайты с доступом к вашему
аккаунту можно на странице [Аккаунт Google](#).

Подробнее о том,
[как Google помогает безопасно делиться данными...](#)


Ознакомьтесь с [политикой конфиденциальности](#) и
условиями использования приложения "DVC".

Отмена

Продолжить

После этого вы увидите верификационный код, который необходимо скопировать и использовать для авторизации в утилите dvc.

Убедитесь, что приложение "DVC" надежно

 Приложение "DVC" запрашивает доступ к вашему аккаунту Google. В целях безопасности сначала убедитесь, что знаете это приложение и оно надежно.


Выполните вход или предоставьте приложению "DVC" доступ к аккаунту

Чтобы выполнить вход или предоставить доступ:

1. Скопируйте код авторизации из раздела с кодом.
2. Перейдите в приложение "DVC".
3. Вставьте код авторизации в "DVC".

Код авторизации

Скопируйте код, перейдите в приложение и вставьте его в нужное поле:

4/1AX4XfwI5Es9dHjpbnzBYdBEHf1hWAFRycFjdNHCKYb0GY
v1z4cRH09M3k3c 

Не выполняйте вход и не предоставляйте приложению "DVC" доступ к аккаунту

Если вы не хотите продолжать, закройте это окно.


Из этого окна необходимо скопировать код авторизации и подставить его в консоли

```
ubuntu@ip-172-31-95-38:~/project$ dvc push -r mydisk
0% Querying remote cache|
Go to the following link in your browser:
  https://accounts.google.com/o/oauth2/auth?client_id=710796635688-iivsqbgbsb6uv1fap6635dhvvei09o66c.apps.google
%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.appdata&access_type=offline&response_type=code&approval_prompt=force
Enter verification code: 4/1AX4XfwI5Es9dHjpbnzBYdBEHf1hWAFRycFjdNHCKYb0GYv1z4cRH09M3k3c
Authentication successful.
2 files pushed
ubuntu@ip-172-31-95-38:~/project$ █
```

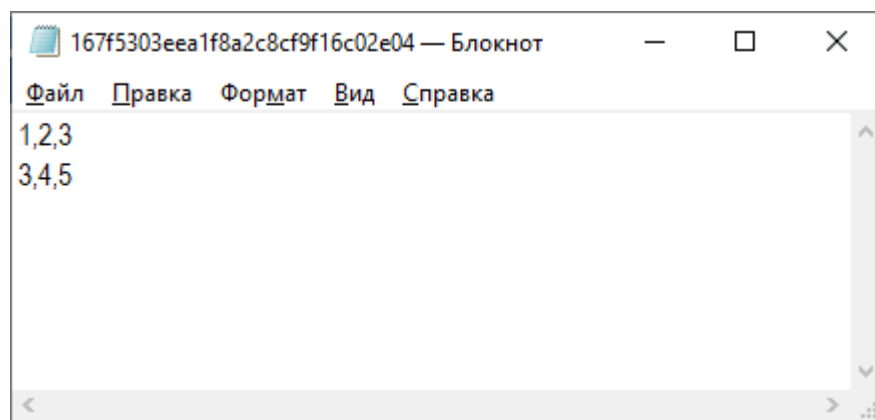
На этом процедура авторизации завершена, теперь вы можете пользоваться удаленным репозиторием для хранения датасетов, расположенным на Google Disk. В частности, мы видим сообщение "2 files pushed". Файлы датасетов хранятся в удаленном репозитории.

Поиск на Диске

Мой диск > datasets > b9 ▾ 👤

Название ↑	Владелец
 167f5303eee1f8a2c8cf9f16c02e04 👤	я

Если этот файл открыть в текстовом редакторе, то мы увидим содержание нашего датасета.



Теперь попробуем внести изменения в данные, сохранить их и выполнить переключение между различными версиями в данных. Для этого в наш датасет ./datasets/a.csv добавим еще одну строчку.

```
ubuntu@ip-172-31-95-38:~/project$ cat datasets/a.csv
1,2,3
3,4,5
ubuntu@ip-172-31-95-38:~/project$ echo "7,8,9" >> datasets/a.csv
ubuntu@ip-172-31-95-38:~/project$ git status
On branch master
nothing to commit, working tree clean
ubuntu@ip-172-31-95-38:~/project$ dvc add datasets
100% Adding... |
To track the changes with git, run:
    git add datasets.dvc
To enable auto staging, run:
    dvc config core.autostage true
ubuntu@ip-172-31-95-38:~/project$ git status
on branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   datasets.dvc
no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-95-38:~/project$
```

После изменения файла a.csv команда git status не видит изменений в файлах, так как папка датасетов ./datasets указана в .gitignore как запрещенная для коммита в git, что правильно,

так как мы не публикуем датасеты в git. Для фиксации изменений в датасете необходимо выполнить команду `dvc add`, после чего будут сделаны изменения в файле `datasets.dvc`, которые уже необходимо опубликовать в git.

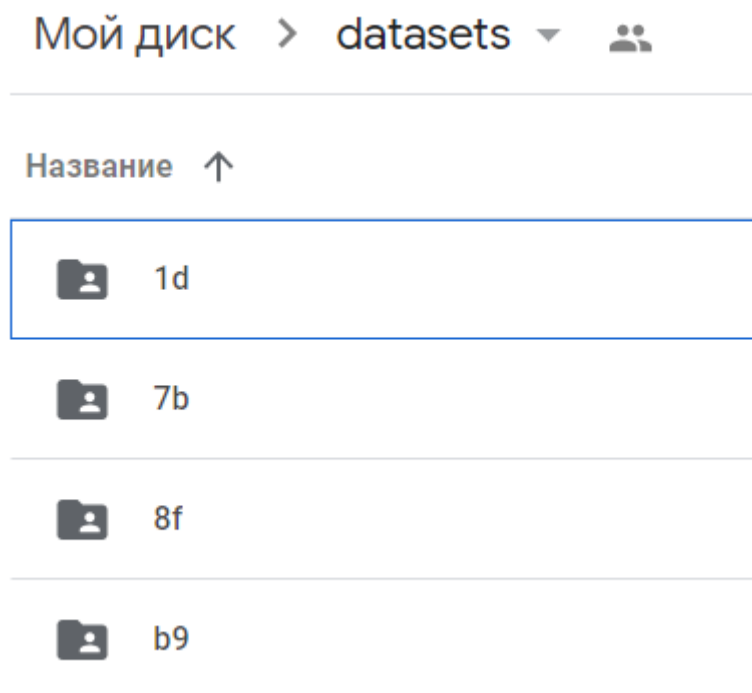
```
ubuntu@ip-172-31-95-38:~/project$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   datasets.dvc

no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-95-38:~/project$
ubuntu@ip-172-31-95-38:~/project$ git commit -a -m "First change in dataset"
[master 0cbd539] First change in dataset
  Committer: Ubuntu <ubuntu@ip-172-31-95-38.ec2.internal>
  1 file changed, 2 insertions(+), 2 deletions(-)
ubuntu@ip-172-31-95-38:~/project$
```

Выполнив команду `dvc push` мы можем опубликовать новую версию датасета в репозиторий

```
ubuntu@ip-172-31-95-38:~/project$ dvc push -r mydisk
2 files pushed
ubuntu@ip-172-31-95-38:~/project$
```

после чего в папке `datasets` на Google Disk появится еще папки.



Папок появляется каждый раз по две, так как мы сохраняем информацию об изменениях в файле `a.out` и папке `datasets`.

Теперь наш датасет и изменение в нем сохранены в удаленном репозитории. Давайте удалим датасет и загрузим его последнюю версию из удаленного репозитория.

```
ubuntu@ip-172-31-95-38:~/project$ ls -lh
total 8.0K
drwxrwxr-x 2 ubuntu ubuntu 4.0K Apr 16 12:06 datasets
-rw-rw-r-- 1 ubuntu ubuntu 90 Apr 16 11:55 datasets.dvc
ubuntu@ip-172-31-95-38:~/project$
ubuntu@ip-172-31-95-38:~/project$ rm -rf datasets
ubuntu@ip-172-31-95-38:~/project$
ubuntu@ip-172-31-95-38:~/project$ ls -lh
total 4.0K
-rw-rw-r-- 1 ubuntu ubuntu 90 Apr 16 11:55 datasets.dvc
ubuntu@ip-172-31-95-38:~/project$
ubuntu@ip-172-31-95-38:~/project$ dvc pull -r mydisk
A      datasets/
1 file added
ubuntu@ip-172-31-95-38:~/project$ cat datasets/a.csv
1,2,3
3,4,5
7,8,9
ubuntu@ip-172-31-95-38:~/project$ █
```

Как видите, мы нормально восстановили последнюю версию датасета из репозитория. Давайте теперь попробуем выбрать старую версию датасета. Все наши публикации изменений (коммиты) можно увидеть с помощью команды `git log`

```
ubuntu@ip-172-31-95-38:~/project$ git log --oneline
0cbd539 (HEAD -> master) First change in dataset
580b053 Google Disk was added as remote dataset folder
898fc0d Put datasets under control
c9ef683 Init dvc
ubuntu@ip-172-31-95-38:~/project$ █
```

Вы можете перевести указатель HEAD на один шаг назад с помощью `HEAD^1`, либо в явном виде указать хэш коммита, на который хотите перейти

```
ubuntu@ip-172-31-95-38:~/project$ git checkout 0cbd539
Previous HEAD position was 898fc0d Put datasets under control
HEAD is now at 0cbd539 First change in dataset
ubuntu@ip-172-31-95-38:~/project$ git log --oneline
0cbd539 (HEAD, master) First change in dataset
580b053 Google Disk was added as remote dataset folder
898fc0d Put datasets under control
c9ef683 Init dvc
ubuntu@ip-172-31-95-38:~/project$ git checkout 580b053
Previous HEAD position was 0cbd539 First change in dataset
HEAD is now at 580b053 Google Disk was added as remote dataset folder
ubuntu@ip-172-31-95-38:~/project$ █
```

Теперь мы можем загрузить предыдущую версию датасета

```
ubuntu@ip-172-31-95-38:~/project$ git pull -r mydisk
fatal: 'mydisk' does not appear to be a git repository
fatal: could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
ubuntu@ip-172-31-95-38:~/project$ dvc pull -r mydisk
A      datasets/
1 file added and 1 file fetched
ubuntu@ip-172-31-95-38:~/project$ cat datasets/a.csv
1,2,3
3,4,5
ubuntu@ip-172-31-95-38:~/project$ █
```

Необходимо отметить, что также в `dvc` можно создавать пайплайны работы с моделями машинного обучения с использованием команды `dvc run`, при этом описание пайплайна делается обычными ключами, в том числе результат работы модели сохранять в определенный выбранный файл:

```
> dvc run -f catboost_dev/.dvc \  
>         -d constants.csv \  
>         -d loader.py \  
>         -d preparator.py \  
>         -d data/raw_feats.csv \  
>         -d catboost_fitter.py \  
>         -m metrics/ctb_binary.txt  
> python preparator.py
```

В следующих модулях мы также рассмотрим другие инструменты для создания пайплайнов.

Тест

1. без какой утилиты не будет работать `dvc` (0.25)
 - a. java
 - b. git**
 - c. ntp
 - d. netstat
2. какие объекты можно версионировать в `dvc`? (0.25)
 - a. датасеты
 - b. модели машинного обучения**
 - c. конвейеры машинного обучения**
 - d. метрики
3. какая команда создает удаленный репозиторий? (0.25)
 - a. dvc remote add**
 - b. `dvc remote create`
 - c. `dvc remote connect`
 - d. `dvc remote setup`
4. Что может быть использовано в качестве хранилища для `dvc` (0.25)
 - a. Amazon S3**
 - b. Google Drive**
 - c. сервер с доступом по `ssh`
 - d. локальный жесткий диск

Итоги/выводы

В юните вы узнали как использовать утилиту `dvc` для организации работы с наборами данных (датасетами). Вы научились:

- создавать необходимую структуру папок
- версионировать и публиковать изменения в датасетах
- менять рабочие версии датасетов
- публиковать метаданные в `git`
- быстро разворачивать необходимую структуру с использованием клонирования `git` репозитория и вытягивания (`pull`) датасетов.

Практическое задание на модуль

В практическом задании данного модуля вам необходимо продемонстрировать навыки практического использования утилиты `dvc` для работы с данными. В результате выполнения этих заданий вы выполните все основные операции с `dvc` и закрепите полученные теоретические знания практическими действиями.

Этапы задания:

1. Установите `git` и `dvc`.
2. Создайте папку проекта.
3. Настройте папку проекта для работы с `git` и `dvc`.
4. Настройте `git` репозиторий.
5. Настройте удаленное хранилище файлов, например на Google Disk.
6. Создайте датасет о пассажирах “Титаника”, например, `catboost.titanic()`.
7. Создайте датасет, в котором содержится информация о классе (“Pclass”), поле (“Sex”) и возрасте (“Age”) пассажира. Сделайте коммит в `git` и `push` в `dvc`.
8. Создайте новую версию датасета, в котором пропущенные (`nan`) значения в поле “Age” будут заполнены средним значением. Сделайте коммит в `git` и `push` в `dvc`.
9. Создайте новый признак с использованием `one-hot-encoding` для строкового признака “Пол” (“Sex”). Сделайте коммит в `git` и `push` в `dvc`.
10. Выполните переключение между всеми созданными версиями датасета.

При правильном выполнении задания и вас появится `git` репозиторий с опубликованной метаинформацией и папка на Google Disk, в которой хранятся различные версии датасетов.

В постановке задачи используется датасет из конкурса “Titanic Disaster”, однако вы можете использовать свои наборы данных, в этом случае в п.п.6-9 необходимо использовать информацию и признаки из вашего датасета.

Рекомендация: вы можете пользоваться описанием соответствующих процедур, которые были в юните 3 данного модуля.

В качестве результата необходимо прислать ссылку на репозиторий `git`.

Модуль 5. Конвейеры операций

Образовательный результат:

В результате прохождения этого модуля слушатели научатся

1. Декомпонировать процесс работы с моделью машинного обучения на отдельные операции
2. Создавать конвейеры операций с использованием Apache Airflow
3. Применять в эксплуатации моделей машинного обучения конвейеры операций, осуществлять запуск и мониторинг выполнения этих операций.

В этом модуле:

Вы познакомитесь с понятием конвейера операций и инструментами для его создания. Конвейер операций позволяет описывать отдельные процессы и обеспечивать их взаимосвязанное последовательное выполнение. Конвейеры операций часто применяются в проектах машинного обучения для автоматизации отдельных этапов, от сбора данных до вывода модели машинного обучения в эксплуатацию.

Сначала вы узнаете о том какие бывают инструменты для создания и использования конвейеров, а также ключевые понятия и термины.

Затем мы сосредоточимся на одном из самых популярных инструментов Apache Airflow. Вы научитесь создавать и запускать конвейеры и отслеживать их состояние в Airflow.

В качестве практического примера использования Airflow мы рассмотрим одну из уже изученных нам задач машинного обучения “Titanic Disaster”.

Темы, изучаемые в модуле:

1. Теория конвейеров операций.
2. Создание конвейера операций в Apache Airflow.
3. Решение практической задачи с Apache Airflow.

Модуль 5. Юнит 1. Конвейеры операций.

Введение: В этом юните описаны конвейеры операций и сделан обзор инструментов для их создания.

Содержание юнита:

Как вы уже знаете (например, из Модулей 1 и 2 данного курса) создание модели машинного обучения состоит из различных этапов, наиболее важные из которых:

- сбор и проверка данных,
- подбор и обучение модели,
- тестирование,
- подготовка к переводу модели в производственную эксплуатацию.

Каждый из этих этапов содержит множество рутинных операций, перечень и содержание которых зависит от конкретного проекта.

1. Сбор и проверка данных
 - a. опрос источников данных по расписанию или триггеру,
 - b. прием информации от источников в режиме прослушивания,
 - c. формирование наборов данных от разнородных источников для использования в обучении модели или формировании аналитического отчета,
 - d. проверка качества данных,
 - e. предобработка данных, преобразование форматов, разархивирование,
 - f. сохранение сформированных наборов данных, датасетов,
 - g. передача данных в другие модули проекта.
2. Подбор и обучение модели
 - a. настройка окружения для проведения эксперимента, установка необходимых библиотек,
 - b. загрузка данных,
 - c. проведение эксперимента для выбора оптимальной модели, подбор гиперпараметров,
 - d. сохранение артефактов отдельных этапов: набора данных, кода, весов модели, метрик, параметров окружения,
 - e. загрузка и запуск нужной версии модели.
3. Тестирование
 - a. проверка правильности работы модели,
 - b. тестирование устойчивости модели к шумам в данных,
 - c. проверка окружения (библиотек, параметров),
 - d. проверка быстродействия,
 - e. интеграционное тестирование.
4. Подготовка к переводу модели в промышленную эксплуатацию
 - a. настройка серверов для развертывания,
 - b. сборка всего пакета для установки,
 - c. установка и запуск.

Все эти операции взаимосвязаны между собой, некоторые могут выполняться параллельно, выполнение других же может начаться только после того, как выполнен ряд других операций. Удобнее всего такие взаимосвязанные операции представлять в виде

ориентированного ациклического графа (DAG, Directed Acyclic Graph). С использованием DAG задачи объединяются в единый конвейер данных. Благодаря наглядности графового представления связи между отдельными узлами, представляющими операции, хорошо видны, что позволяет удобно проследить цепочку задач.

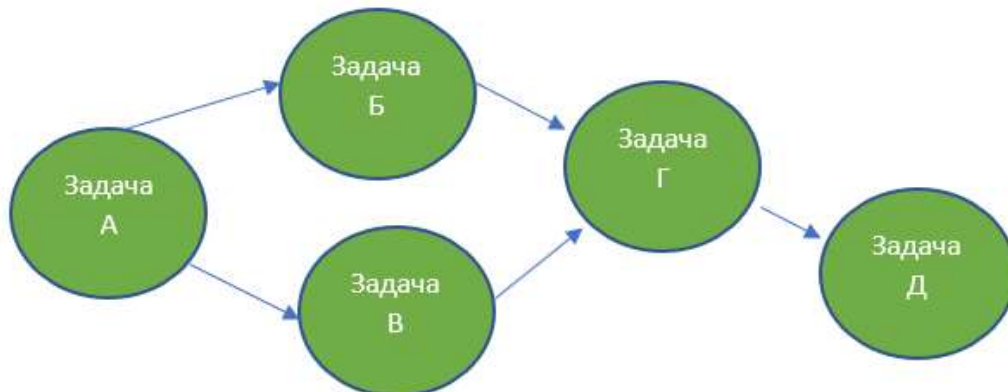


Рисунок «Пример ациклического направленного графа».

На этом рисунке «Задача Д» является конечной задачей в цепочке, она зависит от выполнения всех остальных предшествующих ей задач. «Задача Б» и «Задача В» не зависят друг от друга, а только от «Задачи А», поэтому могут выполняться одновременно, что сокращает продолжительность работы всего конвейера.

Анализ задач, выполняющихся последовательно, позволяет определить те из них, которые не зависят друг от друга. Также важно определить продолжительные по времени задачи, сбой выполнения которых может повлиять на другие задачи. Все это позволяет разместить задачи на графе наиболее эффективным образом. Для такого эффективного и надежного управления последовательностями операций необходимо выполнение ряда условий и наличие специальных функций:

- планировщик задач с удобным инструментом для запуска и контроля,
- централизованный мониторинг и обработка ошибок и внештатных ситуаций,
- управление зависимостями, например между функциями/классами,
- возможность восстановления в точке останова, которая произошла вследствие ошибки или внештатной ситуации,
- удобный графический интерфейс,
- гибкий инструмент контроля, например, возможность использования командной строки.

Инструменты, реализующие набор вышеуказанных методов, активно разрабатываются. Краткий обзор open-source инструментов, которые используются для создания конвейеров операций:

Название	Разработчик, ссылка	Описание
Luigi	Spotify, https://github.com/spotify/luigi	Инструмент для запуска и контроля операций. Не может запускать операции по расписанию, но для этих целей можно использовать crontab. Также неэффективно работает с распределением задач, для этих целей применяется Celery. До сих пор достаточно широко распространен в проектах, хотя чаще

		используется Airflow.
Apache Airflow	<p>Разработан Airbnb, позже перешел в open-source Apache Foundation</p> <p>https://airflow.apache.org/</p>	<p>Эффективная обработка сложных потоков операций (workflow), запуск по расписанию, поддержка различных операций, например, bash, запросы в базы данных, работа с хранилищами данных.</p> <p>Плюсы: удобный функциональный интерфейс, масштабируемость, большое сообщество и хорошая поддержка.</p> <p>Минусы: негибкий инструмент.</p> <p>Архитектура: flask, планировщик, воркер.</p>
MLFlow	<p>Изначально разработан Databricks, с 2020 года входит в Linux Foundation.</p> <p>https://mlflow.org</p>	<p>Платформа для реализации полного цикла машинного обучения, упрощает создание и развертывание моделей машинного обучения, а также обмен ими. Инструмент предполагает набор API, которые работают с любой библиотекой, в том числе с PyTorch, TensorFlow, и XGBoost, а также в любой среде, включая облачные сервисы.</p> <p>Есть встроенные интеграции с Docker, TensorFlow, PyTorch, Kubernetes, Java, Spark и другими открытыми проектами.</p> <p>Позволяет сохранять модель, метрики, перезапускать, следить за экспериментом, сохранять код, артефакты.</p> <p>Можно получить любую версию модели и развернуть ее.</p>
Prefect	https://www.prefect.io/	Фреймворк с открытым исходным кодом для описания и выполнения цепочек процессов обработки данных с использованием языка Python. Позволяет создавать, запускать и контролировать конвейеры данных различного масштаба.
Dagster	https://dagster.io/	Оркестратор данных для машинного обучения, аналитики и ETL.

Oozie	Apache Foundation. https://oozie.apache.org/	Система планирования рабочих процессов для управления заданиями в экосистеме для работы с большими данными с Hadoop.
-------	---	--

Кроме того, существует множество других систем создания и контроля конвейеров операций, часто распространена ситуация, когда такая система разрабатывается самостоятельно под свои нужды. Также много коммерческих решений, как правило являющихся частью какого-то глобального решения по работе с данными. Создание системы управления операциями “с нуля” затратно по времени, а использование коммерческих решений не всегда оправдано экономически, поэтому частый сценарий развертывания конвейера операций это использование одного из open-source решений. Наиболее функциональный и популярный в настоящее время Apache Airflow, его рассмотрению и будут посвящены следующие юниты этого модуля.

Тест, практическое задание

1. Что такое DAG? (0.25)
 - a. Directory Access Gateway
 - b. Dynamic Array Gain
 - c. Directed Acyclic Graph**
 - d. Digital Alignment Growth
2. Какая компания разработала Luigi? (0.25)
 - a. Microsoft
 - b. Yandex
 - c. Spotify**
 - d. Airbnb
3. Что можно отнести к полезным функциям системы управления цепочками операций? (0.25)
 - a. удобный графический интерфейс**
 - b. возможность просмотра логов**
 - c. возможность использования GPU
 - d. платформонезависимость
4. С использованием каких инструментов можно описать конвейер операций? (0.25)
 - a. Apache Airflow**
 - b. MLFlow**
 - c. pandas
 - d. prefect**

Итоги/выводы

В этом юните вы узнали о том как работа с моделью машинного обучения может быть разбита на отдельные операции и об инструментах создания и контроля таких операций. В следующих юнитах мы подробно рассмотрим один из таких инструментов, Apache Airflow.

Модуль 5. Юнит 2. Apache Airflow

Введение: В этом юните вы подробнее узнаете об одном из самых популярных инструментов для организации и контроля выполнения конвейера операций, Apache Airflow. Вы научитесь устанавливать этот инструмент и настраивать его для работы. Эти знания понадобятся для выполнения практического задания в данном модуле.

Содержание юнита:

Apache Airflow это программное обеспечение для создания и управления потоками операций, многофункциональный workflow менеджер. Apache Airflow часто используется инженерами данных и специалистами DevOps/MLOps для автоматизации и контроля выполнения цепочек задач в проектах.

Airflow был создан в 2014 году компанией Airbnb. Через некоторое время Airflow был передан в организацию Apache Foundation (open-source проект), в которой в январе 2019 года получил статус Top-Level проекта. В настоящее время это один из самых популярных open-source инструментов для автоматизации и контроля выполнения операций.

Далее в этом модуле вы узнаете про установку, настройку и запуск конвейера операций в Airflow и создадите свой первый конвейер. Но сначала давайте познакомимся с важными понятиями и определениями Airflow, многие из которых используются и в других подобных инструментах. Понимание этой терминологии необходимо для правильной и эффективной работы с Airflow.

DAG (Directed Acyclic Graph). DAG это ориентированный ациклический граф, т.е. граф у которого отсутствуют циклы, но могут быть параллельные пути, выходящие из одного и того же узла. С использованием DAG задачи объединяются в единый конвейер (pipeline). Благодаря наглядности графового представления связи между отдельными узлами, представляющими операции, хорошо видны, что позволяет удобно проследить цепочку задач и, при необходимости, оптимизировать.

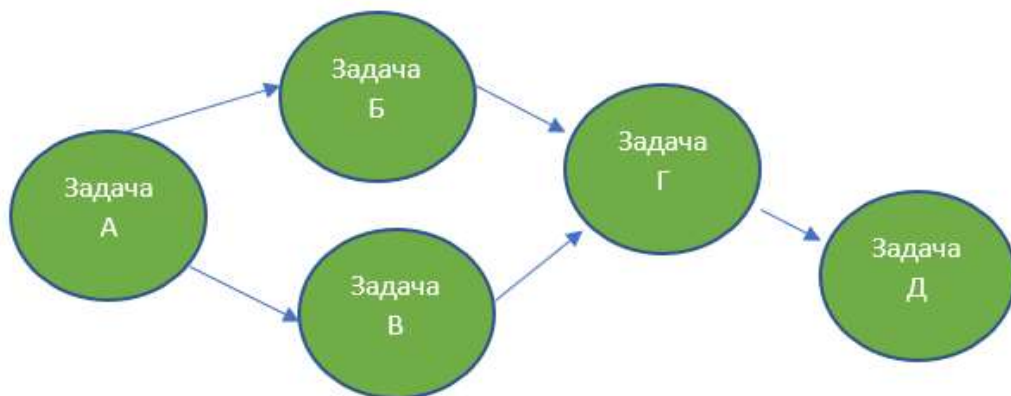


Рисунок «Пример цепочки задач, представленной в виде графа».

Оператор (Operator). Оператор в Airflow это звено в цепочке задач. С использованием операторов в Airflow описывается какую задачу необходимо выполнить. В Airflow есть набор готовых операторов для типовых задач, например:

- PythonOperator для исполнения python кода,

- BashOperator для запуска bash скриптов/команд,
- PostgresOperator для вызова SQL запросов в базе данных PostgreSQL,
- EmailOperator для отправки электронных писем.

Полный список стандартных операторов Airflow можно найти [в документации в Apache Airflow](https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html) (https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html).

Граф операций DAG графически показывает операторы и их связи между собой, на рисунке выше «Задача А», «Задача Б» и далее это отдельные операторы. Операторы полностью независимы друг от друга, разные операторы могут выполняться на разных аппаратных платформах, в разных программных средах, с разными параметрами. Связь двух операторов в DAG не означает передачу результата работы первого оператора для обработки второму оператору. В этом отличие DAG в Airflow от аналогичной конструкции DAG в Tensorflow, который связывает между собой отдельные функции и результат выполнения предыдущей функции передается в следующую. *В Airflow операции не передают в другие операции результат своей работы.*

Хук (Hook). Хуки это интерфейсы для работы с различными внешними сервисами:

- базы данных, в т.ч. redis, memcached,
- распределенные хранилища, например Amazon S3, Google Disk,
- внешние ресурсы с доступом через API интерфейс.

Хуки являются частями операторов и реализуют логику взаимодействия с хранилищем конфигурационных файлов и доступов. Использование хуков позволяет решить проблему хранения секретной информации в коде (например, пароли к доступам), делая инфраструктуру более безопасной.

Сенсор (Sensor). Сенсоры являются разновидностью операторов, их удобно использовать при реализации конвейеров, в которых необходимо учитывать возникновение определенных событий. Также как и для операторов, для сенсоров существуют типовые варианты, например:

- PythonSensor ожидает, когда функция вернёт True,
- S3Sensor проверяет наличие объекта по ключу в S3-бакете,
- RedisPubSubSensor проверяет наличие сообщения в pub-sub очереди redis,
- RedisKeySensor: проверяет существует ли переданный ключ в redis хранилище.

Полный перечень доступных типовых сенсоров можно найти [в официальной документации Airflow](https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/sensors/index.html?highlight=sensors#module-airflow.sensors) (https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/sensors/index.html?highlight=sensors#module-airflow.sensors).

Также есть возможность создать собственный сенсор, для этого можно использовать класс BaseSensorOperator, в котором требуется переопределить метод poke.

Исполнители (Airflow Executors). Исполнители (Executors) в Airflow отвечают за выполнение отдельных задач. В Airflow есть несколько видов типовых исполнителей:

- SequentialExecutor
- LocalExecutor
- CeleryExecutor
- DaskExecutor

- **KubernetesExecutor**

На практике чаще всего встречается **CeleryExecutor**, предназначенный для работы с планировщиком задач Celery. Рассмотрим подробнее отдельные типы исполнителей.

Исполнитель **SequentialExecutor** установлен в качестве значения по умолчанию в `airflow.cfg` (параметр `executor`), представляет из себя простой вид воркера, который не умеет запускать параллельные задачи. В этом случае в конкретный момент времени может выполняться только одна задача. Используется в основном в учебных целях или для простых экспериментов, для продуктивной среды он не подходит.

Исполнитель **LocalExecutor** наиболее похож на продуктивную среду в тестовом окружении или окружении разработки. С его помощью можно выполнять задачи параллельно, например, исполнять несколько DAGов одновременно, путём порождения дочерних процессов. Этот тип исполнителя также не предназначен для производственного окружения из-за ограничений:

- ограничение при масштабировании ресурсами аппаратного обеспечения, на котором он запущен,
- отсутствие отказоустойчивости, если система с этим типом воркера аварийно завершает работу, то задачи перестают исполняться до момента восстановления работоспособности.

Исполнитель **LocalExecutor** можно эффективно использовать при небольшом количестве задач, т.к. это проще, быстрее и не требует настройки дополнительных сервисов.

CeleryExecutor это наиболее часто встречающийся на практике исполнитель, использующий функции Celery для управления потоками выполнения операций. Так же, как и в Celery, в **CeleryExecutor** необходимо дополнительно настроить брокер сообщений, например Redis или RabbitMQ.

Исполнитель **DaskExecutor** аналогичен **CeleryExecutor**, но вместо Celery использует Dask, в частности `dask-distributed`.

Исполнитель **KubernetesExecutor** предназначен для выполнения задач на кластере `kubernetes`. Многие системы сейчас имеют микросервисную архитектуру, поэтому этот новый вид исполнителя является актуальным для использования, но и одновременно сложным, так как для использования контейнеров и этого исполнителя требуется настроить `kubernetes` кластер, что является непростой задачей.

В стандартной конфигурации `Airflow` по умолчанию используется **SequentialExecutor**, для использования исполнителей других типов необходимо в файле конфигураций `airflow.cfg` изменить значение параметра `executor`, например установить значение в **LocalExecutor**.

Итак, вы теперь знаете основные сущности `Airflow`:

- направленный граф операций (DAG),
- операторы (Operators),
- исполнители (Executors),

- хуки (Hooks),
- сенсоры (Sensors).

В этом юните мы рассмотрим использование графов операций, операторов и исполнителей. Примеры использования хуков и сенсоров вы можете найти самостоятельно в документации к Airflow.

Давайте рассмотрим процесс установки и запуска Apache Airflow, который описан [в документации на официальном сайте](https://airflow.apache.org/docs/apache-airflow/stable/installation/index.html) <https://airflow.apache.org/docs/apache-airflow/stable/installation/index.html>. Существуют различные способы развернуть Airflow, например с использованием docker и helm. Далее рассмотрен самый простой вариант это установка с использованием python установщика pip.

Сначала создадим рабочую папку и виртуальное окружение, в которое будем устанавливать необходимое программное обеспечение

```
ubuntu@ip-172-31-95-38:~$ mkdir airflow
ubuntu@ip-172-31-95-38:~$
ubuntu@ip-172-31-95-38:~$ cd airflow/
ubuntu@ip-172-31-95-38:~/airflow$
ubuntu@ip-172-31-95-38:~/airflow$ python3 -m venv venv
ubuntu@ip-172-31-95-38:~/airflow$
ubuntu@ip-172-31-95-38:~/airflow$ ls -la
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 25 10:23 .
drwxr-xr-x 8 ubuntu ubuntu 4096 Apr 25 10:23 ..
drwxrwxr-x 6 ubuntu ubuntu 4096 Apr 25 10:23 venv
ubuntu@ip-172-31-95-38:~/airflow$
ubuntu@ip-172-31-95-38:~/airflow$ source ./venv/bin/activate
(venv) ubuntu@ip-172-31-95-38:~/airflow$
(venv) ubuntu@ip-172-31-95-38:~/airflow$
```

Процесс установки выполним с использованием установщика pip

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$
(venv) ubuntu@ip-172-31-95-38:~/airflow$ pip install apache-airflow
collecting apache-airflow
```

Мы видим установленные пакеты:

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$ pip freeze | grep airflow
apache-airflow==2.2.5
apache-airflow-providers-ftp==2.1.2
apache-airflow-providers-http==2.1.2
apache-airflow-providers-imap==2.2.3
apache-airflow-providers-sqlite==2.1.3
(venv) ubuntu@ip-172-31-95-38:~/airflow$
```

Кроме того, сейчас доступна команда **airflow**:

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$ airflow version
2.2.5
(venv) ubuntu@ip-172-31-95-38:~/airflow$
```

Apache Airflow свои настройки хранит в файле `airflow.cfg`, который по умолчанию будет создан в домашней директории пользователя с использованием следующего пути `~/airflow/airflow.cfg`.

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$ pwd
/home/ubuntu/airflow
(venv) ubuntu@ip-172-31-95-38:~/airflow$ ls -la
total 68
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 25 10:28 .
drwxr-xr-x 8 ubuntu ubuntu 4096 Apr 25 10:23 ..
-rw-rw-r-- 1 ubuntu ubuntu 44536 Apr 25 10:28 airflow.cfg
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 25 10:28 logs
drwxrwxr-x 6 ubuntu ubuntu 4096 Apr 25 10:23 venv
-rw-rw-r-- 1 ubuntu ubuntu 4695 Apr 25 10:28 webserver_config.py
(venv) ubuntu@ip-172-31-95-38:~/airflow$
```

Для изменения этого пути необходимо переменной окружения AIRFLOW_PATH присвоить другое значение.

Для начала работы с Airflow требуется выполнить процедуру инициализации базы данных. При этом создаются служебные таблицы в базе данных, необходимые для работы в Airflow. По умолчанию это база данных sqlite. Если вы имеете опыт работы с фреймворком Django, то увидите сходство в этой процедуре с инициализацией служебной базы данных Django. В более ранних версиях эта операция делалась с использованием команды **airflow initdb**, сейчас она заменена на команду **airflow db init**. Эта команда создаст необходимые таблицы в базе данных, в рабочей папке появится файл **airflow.db**.

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$ ls -la
total 684
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 25 10:34 .
drwxr-xr-x 8 ubuntu ubuntu 4096 Apr 25 10:23 ..
-rw-rw-r-- 1 ubuntu ubuntu 44536 Apr 25 10:28 airflow.cfg
-rw-r--r-- 1 ubuntu ubuntu 626688 Apr 25 10:34 airflow.db
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 25 10:28 logs
drwxrwxr-x 6 ubuntu ubuntu 4096 Apr 25 10:23 venv
-rw-rw-r-- 1 ubuntu ubuntu 4695 Apr 25 10:28 webserver_config.py
(venv) ubuntu@ip-172-31-95-38:~/airflow$
```

Файл базы данных *.db является файлом sqlite, которая используется в Airflow по умолчанию. *Sqlite не подходит для производственной среды, поскольку не обладает достаточной производительностью, поэтому при развертывании в производственной среде рекомендуется использовать PostgreSQL или mysql*. Здесь опять же уместна аналогия с Django, где также sqlite используется в качестве базы данных по умолчанию для учебных целей или простых проектов, который заменяется на postgresql или другие более производительные базы данных при выводе проекта в производственное окружение.

Для запуска графического web интерфейса для работы с Airflow необходимо запустить веб сервер с помощью команды

airflow webserver -p “номер порта”

```
(venv) ubuntu@ip-172-31-95-38:~/airflow$ airflow webserver -p 8081

500} INFO - filling up the DagBag from /dev/null
779} WARNING - No user yet created, use flask fab command to do it.
496} INFO - Created Permission View: menu access on List Users
558} INFO - Added Permission menu access on List Users to role Admin
496} INFO - Created Permission View: menu access on Security
558} INFO - Added Permission menu access on Security to role Admin
496} INFO - Created Permission View: menu access on List Roles
558} INFO - Added Permission menu access on List Roles to role Admin
496} INFO - Created Permission View: can read on User Stats chart
558} INFO - Added Permission can read on User Stats chart to role Admin
496} INFO - Created Permission View: menu access on User's Statistics
558} INFO - Added Permission menu access on User's Statistics to role Admin
496} INFO - Created Permission View: menu access on Base Permissions
558} INFO - Added Permission menu access on Base Permissions to role Admin
496} INFO - Created Permission View: can read on View Menus
558} INFO - Added Permission can read on View Menus to role Admin
496} INFO - Created Permission View: menu access on Views/Menus
558} INFO - Added Permission menu access on Views/Menus to role Admin
496} INFO - Created Permission View: can read on Permission Views
558} INFO - Added Permission can read on Permission Views to role Admin
496} INFO - Created Permission View: menu access on Permission on Views/Menus
558} INFO - Added Permission menu access on Permission on Views/Menus to role Admin
496} INFO - Created Permission View: can get on MenuApi
558} INFO - Added Permission can get on MenuApi to role Admin
512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs,can_create Admin
496} INFO - Created Permission View: menu access on Providers
558} INFO - Added Permission menu access on Providers to role Admin
496} INFO - Created Permission View: can create on XCOMs
558} INFO - Added Permission can create on XCOMs to role Admin

Running the Gunicorn Server with:
workers: 4 sync
Host: 0.0.0.0:8081
Timeout: 120
Logfiles: -
Access Logformat:

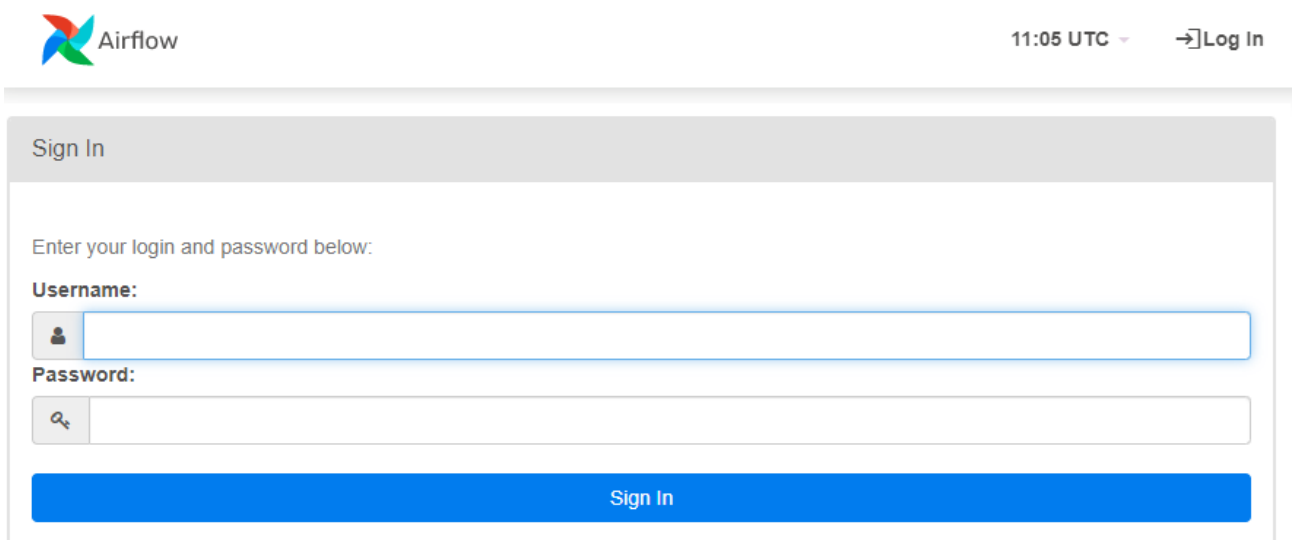
[2022-04-25 11:03:31 +0000] [122280] [INFO] Starting gunicorn 20.1.0
[2022-04-25 11:03:31 +0000] [122280] [INFO] Listening at: http://0.0.0.0:8081 (122280)
[2022-04-25 11:03:31 +0000] [122280] [INFO] Using worker: sync
[2022-04-25 11:03:31 +0000] [122282] [INFO] Booting worker with pid: 122282
[2022-04-25 11:03:31 +0000] [122283] [INFO] Booting worker with pid: 122283
[2022-04-25 11:03:31 +0000] [122284] [INFO] Booting worker with pid: 122284
[2022-04-25 11:03:32 +0000] [122285] [INFO] Booting worker with pid: 122285
```

Для работы в системе понадобится пользователь и пароль, который создается с использованием команды `airflow users create`

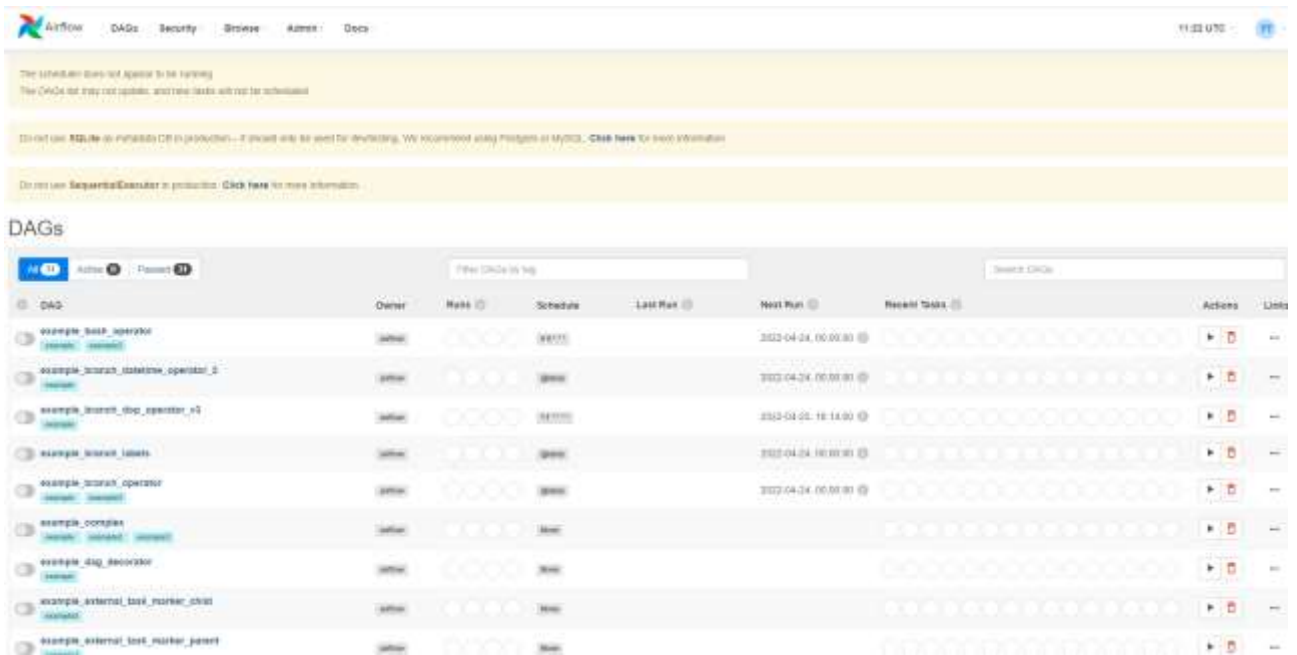
```
$ airflow users create \
--username admin \
--firstname FIRST_NAME \
--lastname LAST_NAME \
--role Admin \
--email admin@example.org

airflow users create command error: the following arguments are required: -e/--email, -f/--firstname, -l/--lastname, see help above.
(venv) ychernyshov@bruteForce:~/airflow$ airflow users create --username admin --role Admin -f test -l test -e test@test.com
[2022-04-25 11:03:31 +0000] [122280] [INFO] Starting gunicorn 20.1.0
[2022-04-25 11:03:31 +0000] [122280] [INFO] Listening at: http://0.0.0.0:8081 (122280)
[2022-04-25 11:03:31 +0000] [122280] [INFO] Using worker: sync
[2022-04-25 11:03:31 +0000] [122282] [INFO] Booting worker with pid: 122282
[2022-04-25 11:03:31 +0000] [122283] [INFO] Booting worker with pid: 122283
[2022-04-25 11:03:31 +0000] [122284] [INFO] Booting worker with pid: 122284
[2022-04-25 11:03:32 +0000] [122285] [INFO] Booting worker with pid: 122285
[2022-04-25 11:03:32 +0000] [122280] [INFO] Password:
[2022-04-25 11:03:32 +0000] [122280] [INFO] repeat for confirmation:
[2022-04-25 11:03:32 +0000] [122280] [INFO] User "admin" created with role "Admin"
(venv) ychernyshov@bruteForce:~/airflow$
```

Теперь можно по адресу `http://имя или IP адрес сервера:номер порта` открыть графический web интерфейс для работы (в котором предсказуемо вас попросят сначала авторизоваться)



Теперь необходимо ввести регистрационные данные, после чего вы увидите главное меню для работы с Airflow.



Сразу же можно увидеть несколько уведомлений, например, что не запущен планировщик задач (“The scheduler does not appear to be running”). Также можно увидеть две полезные рекомендации: не использовать базу данных SQLite и исполнитель SequentialExecutor в продуктивной среде, однако на этапе обучения для наших простых проектов можно с этим смириться. Тем не менее нужно помнить, что установка и настройка Airflow сильно зависят от характеристик проекта. При использовании Airflow в производственном окружении необходимо использовать более производительные и безопасные компоненты.

На главной странице веб сервера Airflow можно увидеть различные готовые примеры, например, `example_bash_operator`, в котором демонстрируются возможности работы с `bash` в Airflow.

DAGs

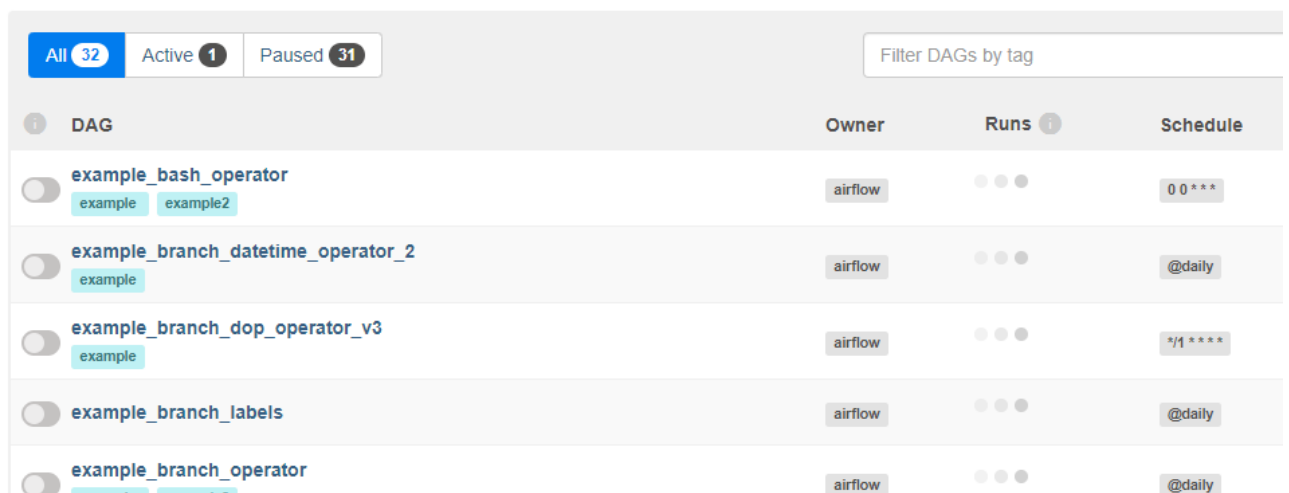


Рисунок “Различные примеры Airflow”

Использование этих примеров позволяет быстро создавать свои конвейеры операций. Например, кликнув на `example_bash_operator`, можно увидеть его структуру

DAG: example_bash_operator

Tree Graph Calendar Task Duration

2022-05-04T19:03:02Z Runs 25 Up

BashOperator DummyOperator

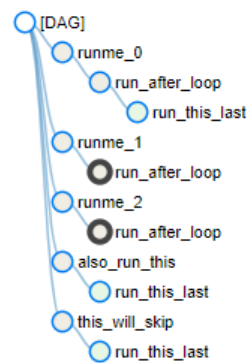


Рисунок “Древовидный вид конвейера операций example_bash_operator”

Также для удобства можно открыть этот конвейер в виде графа, выбрав вкладку “Graph”

DAG: example_bash_operator

Tree Graph Calendar Task Duration

2022-05-04T19:05:05Z Runs 25 Run

BashOperator DummyOperator

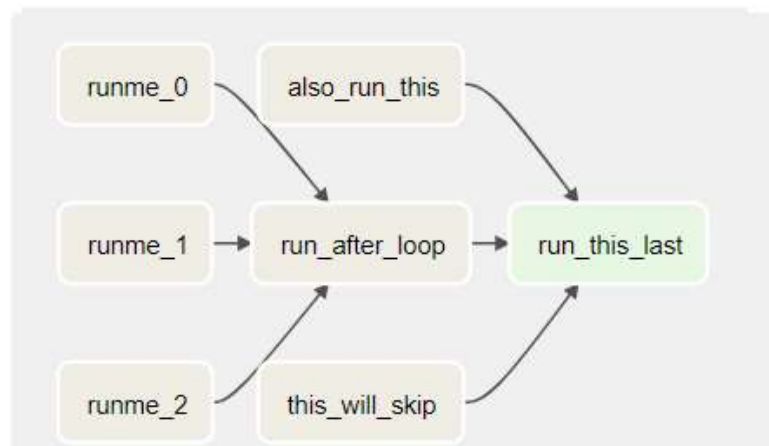


Рисунок “Конвейер example_bash_operator в виде графа”.

И, наконец, можно взглянуть на программный код операций, выбрав вкладку “Code”

Содержание программного кода, позволяющего автоматизировать цепочку выполнения bash операций, кода следующее:

```
# Licensed to the Apache Software Foundation (ASF) under one  
# or more contributor license agreements. See the NOTICE file  
# distributed with this work for additional information  
# regarding copyright ownership. The ASF licenses this file  
# to you under the Apache License, Version 2.0 (the  
# "License"); you may not use this file except in compliance  
# with the License. You may obtain a copy of the License at  
#  
# http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing,  
# software distributed under the License is distributed on an  
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  
# KIND, either express or implied. See the License for the  
# specific language governing permissions and limitations  
# under the License.
```

```
"""Example DAG demonstrating the usage of the BashOperator."""
```

```
import datetime
```

```
import pendulum
```

```
from airflow import DAG
```

```
from airflow.operators.bash import BashOperator
```

```
from airflow.operators.dummy import DummyOperator
```

```
with DAG(  
    dag_id='example_bash_operator',  
    schedule_interval='0 0 * * *',  
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),  
    catchup=False,  
    dagrun_timeout=datetime.timedelta(minutes=60),  
    tags=['example', 'example2'],  
    params={"example_key": "example_value"},  
) as dag:
```

```
    run_this_last = DummyOperator(  
        task_id='run_this_last',  
    )
```

```
    run_this = BashOperator(  
        task_id='run_after_loop',  
        bash_command='echo 1',  
    )
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this_last.set_downstream(run_this)
```

```
    run_this_last.set_downstream(run_this)
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this.set_downstream(run_this_last)
```

```
    run_this.set_downstream(run_this_last)
```

```
# [START howto_operator_bash]
```

```
run_this = BashOperator(  
    task_id='run_after_loop',  
    bash_command='echo 1',  
)
```

```
run_this.set_downstream(run_this_last)
```

```
run_this_last.set_downstream(run_this)
```

```
)
```

```
# [END howto_operator_bash]
```

```
run_this >> run_this_last
```

```
for i in range(3):
```

```

task = BashOperator(
    task_id='runme_' + str(i),
    bash_command='echo "{{ task_instance_key_str }}" && sleep 1',
)
task >> run_this

#[START howto_operator_bash_template]
also_run_this = BashOperator(
    task_id='also_run_this',
    bash_command='echo "run_id={{ run_id }} | dag_run={{ dag_run }}"',
)
#[END howto_operator_bash_template]
also_run_this >> run_this_last

#[START howto_operator_bash_skip]
this_will_skip = BashOperator(
    task_id='this_will_skip',
    bash_command='echo "hello world"; exit 99;',
    dag=dag,
)
#[END howto_operator_bash_skip]
this_will_skip >> run_this_last

if __name__ == "__main__":
    dag.cli()

```

Разберем этот код подробнее.

Сначала устанавливаются настройки DAG

```

with DAG(
    dag_id='example_bash_operator',
    schedule_interval='0 0 * * *',
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
    tags=['example', 'example2'],
    params={"example_key": "example_value"},
) as dag

```

DAG имеет следующие параметры:

- `dag_id` - идентификатор DAG,
- `shedule_interval` - расписание запуска в формате crontab,
- `start_date` - время старта,
- `catchup` - позволяет рассматривать формальную дату начала работы (`start_date`) как текущую дату,
- `dagrun_timeout` - таймаут между запусками DAG,
- `tags` - теги, по которым можно найти DAG, например, показываются в графическом интерфейсе,
- `params` - параметры DAG.

Далее в DAG добавляются операторы. Например, добавляется “пустой” оператор `DummyOperator`, который завершает всю цепочку операторов. Этот оператор получает идентификатор `run_this_last`.

```
run_this_last = DummyOperator(
    task_id='run_this_last',
)
```

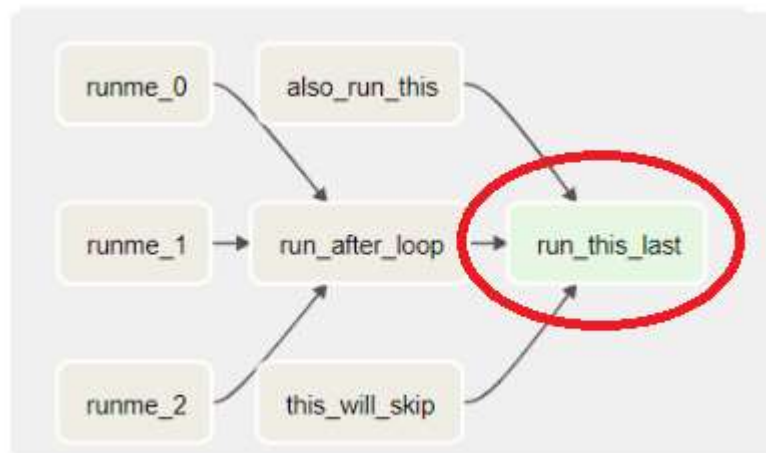


Рисунок “Пустой оператор run_this_last”

Затем создается bash оператор, который выполняет команду “echo 1”, этому оператору присваивается идентификатор run_after_loop, оператор сохраняется под именем run_this, которая используется ниже для задания последовательности выполнения операторов

```
# [START howto_operator_bash]
run_this = BashOperator(
    task_id='run_after_loop',
    bash_command='echo 1',
)
# [END howto_operator_bash]
```

После этого мы можем установить последовательность выполнения операторов с использованием оператора побитового сдвига >>

```
run_this >> run_this_last
```

Поясним алгоритм работы этого оператора. Например, при выполнении команды

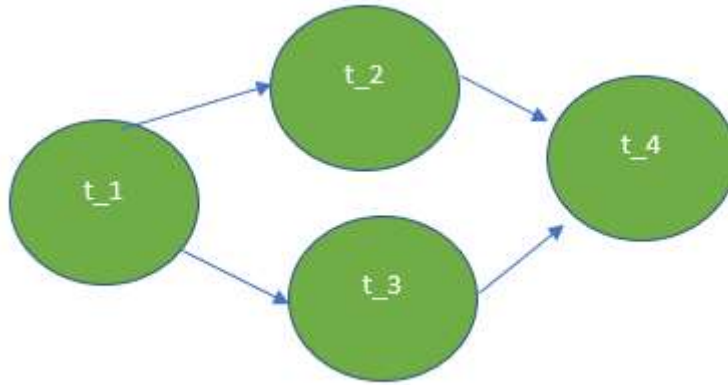
```
t_3 >> t_2 >> t_1
```

сначала выполнится задача t_3, затем t_2 и после нее t_1. Параллельное выполнение отдельных задач задается следующим образом:

```
t_1 >> t_2 >> t_4
```

```
t_1 >> t_3 >> t_4
```

В результате получается следующий граф операций



Таким образом, задачи t_2 и t_3 будут выполняться параллельно. Задача в DAG начнет выполнение только в том случае, если предшествующие ей задачи были успешно выполнены. То есть t_4 начнет выполнение, после успешного завершения t_2 и t_3.

Далее везде используется аналогичная нотация. Например можно добавить сразу три оператора, выполняющихся одновременно:

```
for i in range(3):
    task = BashOperator(
        task_id='runme_' + str(i),
        bash_command='echo "{{ task_instance_key_str }}" && sleep 1',
    )
    task >> run_this
```

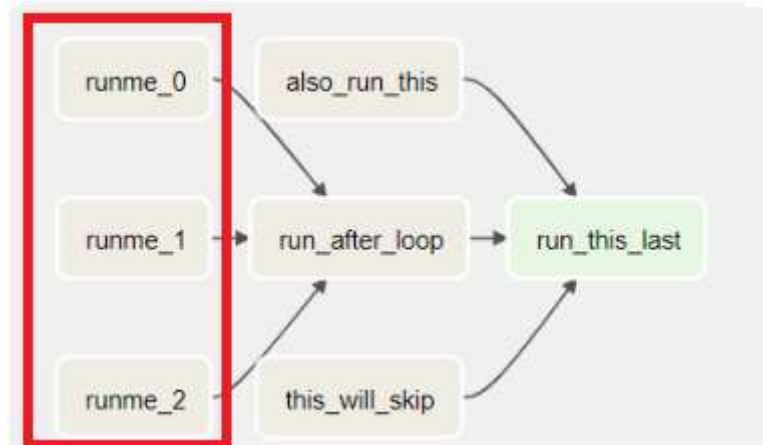


Рисунок “”

Из этого и остальных примерах работы с DAG Airflow легко понять основные правила и синтаксис скрипта python, описывающего DAG. В следующем юните мы рассмотрим создание собственного DAG.

Тест.

1. Какие существуют способы установки Airflow? (0.25)
 - a. pip
 - b. docker

- c. установочный exe-файл для windows
 - d. apt get
2. Что является сущностями Airflow? (0.25)
- a. **Исполнители**
 - b. Постановщики
 - c. **Операторы**
 - d. Клиенты
3. С помощью какого оператора задается последовательность выполнения операций в Airflow? (0.25)
- a. >
 - b. ->
 - c. >>
 - d. from...to...
4. Что не рекомендуется делать при запуске Airflow в производственной среде? (0.25)
- a. **использовать sqlite в качестве служебной базы данных**
 - b. использовать Postgresql в качестве служебной базы данных
 - c. **использовать SequentialExecutor**
 - d. разворачивать Airflow в docker контейнере

Итоги/выводы

В этом юните вы подробно изучили Apache Airflow, позволяющий описывать и контролировать выполнение операций. Теперь вы знаете основные сущности Airflow: граф операций DAG, операторы, хуки, сенсоры, исполнители. Также вы знаете как установить и настроить Airflow для работы. В следующем юните вы создадите свой первый конвейер операций в Airflow.

Модуль 5. Юнит 3. Создание конвейера операций в Airflow.

Введение: В этом юните вы создадите простой конвейер операций в Airflow. В качестве примера используем уже хорошо знакомый нам набор данных из задачи «Titanic Disaster».

Содержание юнита:

Теперь давайте построим наш первый конвейер данных в Apache Airflow. Для работы с Airflow необходимо знать python, так как код, описывающий DAG, пишется именно на python.

В файле настроек airflow.cfg есть параметр dags_folder, он указывает на путь, где лежат файлы с описаниями DAG. По умолчанию значение переменной dags_folder равно \$AIRFLOW_HOME/dags. В эту папку необходимо скопировать код с описанием конвейера операций, который вы создадите далее.

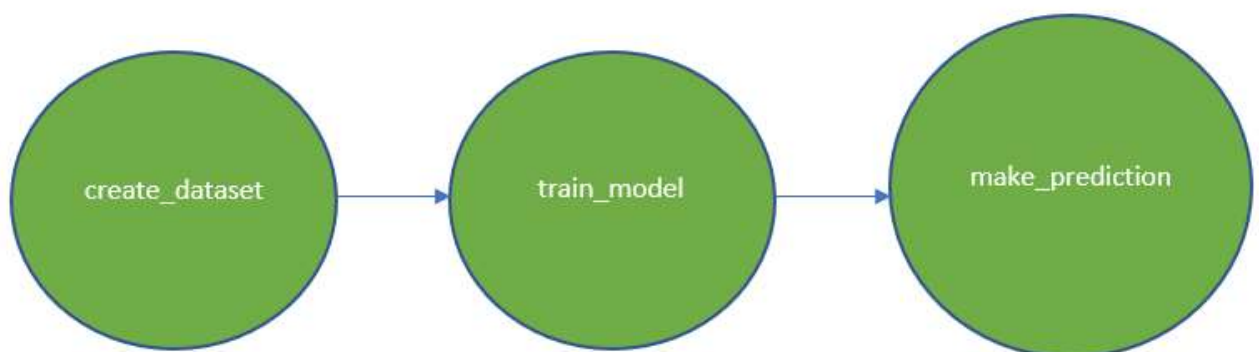
В качестве примера будем решать ту же задачу, что и в Юните 5 Модуля 2, в котором вы автоматизировали решение задачи «Titanic Disaster» с использованием Jenkins. Для создания конвейера операций нам понадобятся три функции:

- create_dataset() – создает набор данных
- train_model() – обучает модель
- make_prediction() – выполняет предсказание

Кроме того, необходимо установить настройки DAG в Airflow, к которым относятся

- Время начала выполнения конвейера(start_date)
- Периодичность запуска (schedule_interval)
- Информация о владельце DAG (owner)
- Количество повторений в случае неудач (retries)
- Пауза между повторами (retry_delay)

Схематично конвейер операций DAG, который мы планируем построить, выглядит так:



Код python файла, который необходимо загрузить в папку dags, приведен в листинге ниже (код имеет простую и понятную структуру, рекомендуем вам его изучить и попробовать написать самостоятельно, обращаясь к листингу в случае затруднений):


```

from catboost.datasets import titanic
import pandas as pd
from sklearn.linear_model import LogisticRegression
import pickle
import datetime as dt

from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator

args = {'owner': 'airflow',
        'start_date': dt.datetime(2020, 2, 11),
        'retries': 1,
        'retry_delay': dt.timedelta(minutes=1),
        'depends_on_past': False,
        }

def create_dataset():
    # Загрузка данных
    train, test = titanic()

    # обработка данных
    # заполним данные о поле пассажира числовыми данными (0 или 1) вместо текстовых ('male' или 'female')
    train['Sex'] = train['Sex'].apply(lambda x: 0 if 'male' else 1)
    test['Sex'] = test['Sex'].apply(lambda x: 0 if 'male' else 1)

    # в признаке "возраст" много пропущенных (NaN) значений, заполним их средним значением возраста
    train['Age'] = train['Age'].fillna(train.Age.mean())
    test['Age'] = test['Age'].fillna(train.Age.mean())

    # запишем созданные датасеты во внешние csv файлы
    train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Survived']].to_csv('data_train.csv', index=False)
    test[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].to_csv('data_test.csv', index=False)

def train_model():
    # прочитаем из csv файла подготовленный датасет для обучения
    data_train = pd.read_csv('data_train.csv')
    X_train = data_train[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].values
    y_train = data_train['Survived'].values

    # загрузим модель машинного обучения
    model = LogisticRegression(max_iter=100_000).fit(X_train, y_train)

    # сохраним обученную модель
    pickle.dump(model, open('model.pkl', 'wb'))

def make_prediction():
    loaded_model = pickle.load(open('model.pkl', 'rb'))

    # прочитаем из csv файла подготовленный датасет для обучения
    data_test = pd.read_csv('data_test.csv')
    X_test = data_test[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].values

    # сделаем предсказание для первого пассажира из тестовой выборки
    print(loaded_model.predict(X_test[0:1]))

with DAG(dag_id='test_airflow', default_args=args, schedule_interval=None) as dag:
    create_dataset = PythonOperator(task_id='create_dataset',
                                   python_callable=create_dataset,
                                   dag=dag)

    train_model = PythonOperator(task_id='train_model',
                                 python_callable=train_model,
                                 dag=dag)

    make_prediction = PythonOperator(task_id='make_prediction',
                                     python_callable=make_prediction,
                                     dag=dag)

    create_dataset >> train_model >> make_prediction

```

После того как мы выполнили перечисленные выше действия в web интерфейсе панели управления Airflow появится информация об описанном нами DAG, либо сообщение об ошибке, если мы что-то сделали неправильно. Например, мы не установили библиотеку catboost в нашем виртуальном окружении, поэтому нам Airflow напоминает об этом:

❗ DAG Import Errors (1)

```
Broken DAG: [/home/ychernyshov/airflow/dags/test_airflow.py] Traceback (most recent call last):
  File "<frozen importlib._bootstrap>", line 228, in _call_with_frames_removed
  File "/home/ychernyshov/airflow/dags/test_airflow.py", line 1, in <module>
    from catboost.datasets import titanic
ModuleNotFoundError: No module named 'catboost'
```

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres

Do not use **SequentialExecutor** in production. [Click here](#) for more information.

DAGs

All **31**
Active **0**
Paused **31**

Filter DAGs by tag

DAG	Owner	Runs	Schedule
No results			

« < > »

То же самое относится и к библиотеке sklearn, хотя и Airflow не сразу пишет об этом. Поэтому надо установить эти две библиотеки с помощью команд

```
pip install catboost
```

```
pip install scikit-learn
```

После этого вы увидите созданный DAG в списке Airflow

DAGs

All **1**
Active **0**
Paused **1**

Filter DAGs by tag

DAG	Owner	Runs	Schedule
<input type="checkbox"/> test_airflow	airflow	○ ○ ○ ○	None

« < 1 > »

Вы можете запустить выполнение этого конвейера, нажав на треугольник в части «Actions» справа. После этого зелеными кругами будут отмечены успешные запуски:

DAGs

The screenshot shows the Airflow DAGs list. At the top, there are filters for 'All', 'Active', and 'Paused'. A search bar contains 'test_airflow'. Below the filters is a table with columns: DAG, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. The first row shows 'test_airflow' with a status of 'Active', a last run time of '2022-04-25 19:50:43', and a series of task status icons. The bottom right corner indicates 'Showing 1-1 of 1 DAGs'.

Теперь можно посмотреть детализацию нашего конвейера, например, увидеть цепочку успешно выполненных операций.

The screenshot shows the detailed view of the DAG 'test_airflow'. The top navigation bar includes 'Airflow', 'DAGs', 'Security', 'Browse', 'Admin', and 'Docs'. Below the title 'DAG: test_airflow', there are tabs for 'Tree', 'Graph', 'Calendar', 'Task Duration', 'Task Times', 'Landing Times', 'Gantt', 'Details', and 'Code'. The 'Graph' tab is selected, showing a flowchart with three tasks: 'create_dataset', 'train_model', and 'make_prediction', connected by arrows. Above the graph, there are filters for 'Runs' (25), 'Run' (manual_2022-04-25T19:50:43.782067+00:00), and 'Layout' (Left > Right). A legend below the graph shows various task statuses: PythonOperator, queued, running, success, failed, up_for_retry, up_for_reschedule, upstream_failed, skipped, scheduled, deferred, and no_status.

Например, выбрав отдельную операцию и нажав на кнопку «Log» можно увидеть log-файл выполнения этой операции.

DAG: test_airflow

Tree Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code

Task Instance: make_prediction at 2022-04-25, 19:50:43

Task Instance Details Rendered Template Log XCom

Log by attempts

1

```
*** Reading local file: /home/ychernyshov/airflow/logs/test_airflow/make_prediction/2022-04-25T19:50:43.782067+00:00/1.log
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1043} INFO - Dependencies all met for <TaskInstance: test_airflow.make_prediction manual__2
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1043} INFO - Dependencies all met for <TaskInstance: test_airflow.make_prediction manual__2
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1249} INFO -
-----
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1250} INFO - Starting attempt 1 of 2
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1251} INFO -
-----
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1270} INFO - Executing <Task(PythonOperator): make_prediction> on 2022-04-25 19:50:43.78206
[2022-04-25, 15:50:48 UTC] {standard_task_runner.py:52} INFO - Started process 31107 to run task
[2022-04-25, 15:50:48 UTC] {standard_task_runner.py:79} INFO - Running: ['airflow', 'tasks', 'run', 'test_airflow', 'make_prediction',
[2022-04-25, 15:50:48 UTC] {standard_task_runner.py:80} INFO - Job 4: Subtask make_prediction
[2022-04-25, 15:50:48 UTC] {logging_mixin.py:109} INFO - Running <TaskInstance: test_airflow.make_prediction manual__2022-04-25T19:50:4
[2022-04-25, 15:50:48 UTC] {taskinstance.py:1446} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=test_airflow
AIRFLOW_CTX_TASK_ID=make_prediction
```

Давайте теперь намеренно “испортим” операцию model_predict(), чтобы увидеть в Airflow проблемы с ее выполнением. Например, укажем имя несуществующего pickle-файла, из которого функция make_prediction() пытается загрузить веса модели:

```
def make_prediction():
    loaded_model = pickle.load(open('model_wrong.pkl', 'rb'))
    # прочитаем из csv файла подготовленный датасет для обучения
    data_test = pd.read_csv('data_test.csv')
    X_test = data_test[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch']].values
    # сделаем предсказание для первого пассажира из тестовой выборки
    print(loaded_model.predict(X_test[0:1]))
```

Осуществив запуск этого конвейера мы увидим, что система пытается запускать операцию, однако при этом последняя функция возвращает ошибку

DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run
test_airflow	airflow	2	None	2022-04-25, 20:02:59	

Search DAGs

Recent Tasks i Actions Links

○ ○ ○ ○ ○ 4 ○ ○ ○ 2 ○ ○ ○ ○ ○ ▶ 🗑️ ⋮

Showing 1-1 of 1 DAGs

Посмотрев детализацию в log-файле мы увидим проблему:

List Dag Run

Search

Actions

<input type="checkbox"/>	State	Dag Id	Logical Date	Run Id	Run Type	Queued At
<input type="checkbox"/> 🔍 🗑️	failed	test_airflow	2022-04-25, 20:02:59	manual__2022-04-25T20:02:59.738012+00:00	manual	2022-04-25, 20:02:59
<input type="checkbox"/> 🔍 🗑️	failed	test_airflow	2022-04-25, 20:02:29	manual__2022-04-25T20:02:29.541203+00:00	manual	2022-04-25, 20:02:29

Airflow DAGs Security Browse Admin Docs

DAG: test_airflow failed Schedule: None Next Run: None

Tree **Graph** Calendar Task Duration Task Titles Landing Times ▶ 🗑️

2022-04-25T20:03:00Z Runs 25 Run manual__2022-04-25T20:02:59.738012+00:00 Layout Left > Right Update

PythonOperator queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred no_status

```

graph LR
    A[create_dataset] --> B[train_model]
    B --> C[make_prediction]
  
```

Task Instance: make_prediction at 2022-04-25, 20:02:59

[Task Instance Details](#) [Rendered Template](#) [Log](#) [XCom](#)

Log by attempts

1

2

```
*** Reading local file: /home/ychernyshov/airflow/logs/test_airflow/make_prediction/2022-04-25T20:02:59.738012+00:00/2.log
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1043} INFO - Dependencies all met for <TaskInstance: test_airflow.make_prediction
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1043} INFO - Dependencies all met for <TaskInstance: test_airflow.make_prediction
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1249} INFO -
-----
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1250} INFO - Starting attempt 2 of 2
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1251} INFO -
-----
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1270} INFO - Executing <Task(PythonOperator): make_prediction> on 2022-04-25 20:02:59.738012+00:00
[2022-04-25, 16:04:06 UTC] {standard_task_runner.py:52} INFO - Started process 32656 to run task
[2022-04-25, 16:04:06 UTC] {standard_task_runner.py:79} INFO - Running: ['airflow', 'tasks', 'run', 'test_airflow', 'make_prediction']
[2022-04-25, 16:04:06 UTC] {standard_task_runner.py:80} INFO - Job 15: Subtask make_prediction
[2022-04-25, 16:04:06 UTC] {logging_mixin.py:109} INFO - Running <TaskInstance: test_airflow.make_prediction manual__2022-04-25T20:02:59.738012+00:00>
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1446} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=test_airflow
AIRFLOW_CTX_TASK_ID=make_prediction
AIRFLOW_CTX_EXECUTION_DATE=2022-04-25T20:02:59.738012+00:00
AIRFLOW_CTX_DAG_RUN_ID=manual__2022-04-25T20:02:59.738012+00:00
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1774} ERROR - Task failed with exception
Traceback (most recent call last):
  File "/home/ychernyshov/airflow/venv/lib/python3.9/site-packages/airflow/operators/python.py", line 174, in execute
    return_value = self.execute_callable()
  File "/home/ychernyshov/airflow/venv/lib/python3.9/site-packages/airflow/operators/python.py", line 188, in execute_callable
    return self.python_callable(*self.op_args, **self.op_kwargs)
  File "/home/ychernyshov/airflow/dags/test_airflow.py", line 52, in make_prediction
    loaded_model = pickle.load(open('model_wrong.pkl', 'rb'))
FileNotFoundError: [Errno 2] No such file or directory: 'model_wrong.pkl'
[2022-04-25, 16:04:06 UTC] {taskinstance.py:1278} INFO - Marking task as FAILED. dag_id=test_airflow, task_id=make_prediction,
[2022-04-25, 16:04:06 UTC] {standard_task_runner.py:93} ERROR - Failed to execute job 15 for task make_prediction ([Errno 2] No such file or directory: 'model_wrong.pkl')
[2022-04-25, 16:04:06 UTC] {local_task_job.py:154} INFO - Task exited with return code 1
[2022-04-25, 16:04:06 UTC] {local_task_job.py:264} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Итак, вы создали конвейер в Airflow, автоматизирующий операции подготовки датасета, обучению модели и выполнению инференса.

Тест

1. В каком параметре `airflow.cfg` хранится информация о папке с файлами, описывающими DAG? (0.25)
 - a. `dag_path`
 - b. `dag_folder`**
 - c. `dag_files`
 - d. `dag_info`
2. Какой параметр DAG устанавливает интервал периодического запуска задач? (0.25)
 - a. `pause`
 - b. `time_delay`
 - c. `schedule_interval`**
 - d. `task_delay`
3. Какая переменная окружения определяет путь до домашней директории Airflow? (0.25)
 - a. `AIRFLOW_PATH`**

- b. AIRFLOW_PLACE
 - c. AIRFLOW_DIR
 - d. AIRFLOW_HOME**
4. Какая команда запускает графическое веб-приложение для работы с airflow? (0.25)
- a. airflow start
 - b. airflow webserver**
 - c. airflow run
 - d. airflow go

Итоги/выводы

В юните вы научились создавать и запускать свой собственный конвейер операций для работы с типовым набором данных “Titanic Disaster”. Конечно, применение Airflow для решения задачи “Titanic Disaster” явно избыточно. Основная цель данного юнита заключалась в том, чтобы на конкретной задаче продемонстрировать работу с Airflow. Практический учебный опыт применения Airflow на простых задачах позволяет лучше разобраться с инструментом и быстрее перейти к использованию Airflow в больших проектах со сложной структурой и большим количеством операций, для которых он и предназначен.

Итоги/выводы по модулю

В этом модуле были рассмотрены конвейеры операций и различные инструменты для их создания. Более подробно был рассмотрен самый популярный инструмент Apache Airflow. Конвейеры операций позволяют решать самые разные задачи в проектах разработки программного обеспечения. Основное их назначение: контролировать выполнение большого числа взаимосвязанных операций.

Практическое задание

Цель задания: закрепить практические навыки работы с Apache Airflow.

Этапы выполнения задания:

1. создать виртуальную среду выполнения, например, с использованием `venv`
2. установить Airflow
3. создайте конвейер операций из следующих шагов:
 - a. создайте искусственно 100 текстовых файлов, состоящих из символов латинского алфавита с помощью скрипта предоставленного преподавателем
 - b. создание 100 параллельных процессов для подсчета количества вхождений символа 'a', по одному процессу на файл, результат необходимо записать в текстовый файл `n.res`, где `n` - номер исходного файла
 - c. создать итоговый процесс, который выполняется после п.3.б, суммирующий данные во всех файлах `*.res`.

Что необходимо получить в итоге: сумму вхождений символа 'a' во все исходные файлы. В качестве результата опубликовать ссылку на git репозиторий с соответствующим кодом.

Список источников

<https://airflow.apache.org/docs/>

<https://www.youtube.com/watch?v=cVDIbEsCTow>

<https://www.bigdataschool.ru/blog/airflow-use-cases.html>

<https://www.bigdataschool.ru/wiki/airflow>

Модуль 6. Автоматизация тестирования.

В этом модуле:

Проверка качества работы разработанного программного обеспечения является важным этапом разработки, позволяющим сократить расходы на устранение замечаний, снизить объем технического долга. В машинном обучении это особенно важно, так как расходы ресурсов на поиск и устранение замечаний в этом случае обычно гораздо выше, чем в обычных проектах.

Различные средства автоматизации позволяют тестировать как отдельные функции и компоненты системы, так и всю систему целиком, в том числе с учетом окружения системы. Прежде всего проверке подвергаются параметры, связанные с бизнес-метриками, оговоренными в техническом задании на разработку системы. Это может быть: объем оперативной памяти, производительность процессора, скорость выполнения операций, точность работы модели, поддержка необходимых протоколов взаимодействия и многое другое. Любое изменение в программном обеспечении или данных может изменить поведение модели машинного обучения, сделать его непредсказуемым.

В данном модуле слушатели познакомятся с основными понятиями из области тестирования программного обеспечения, а также с практическими инструментами для автоматизации тестирования.

Темы, изучаемые в модуле:

1. Роль процесса тестирования в проекте разработки программного обеспечения и какие бывают виды тестирования. Обзор средств автоматизации тестирования. Особенности тестирования для проектов машинного обучения.
2. Тестирование данных.
3. Автоматизация тестирования с unittest и pytest.

Модуль 6. Юнит 1. Виды тестирования.

Введение: В этом юните вы познакомитесь с процессом тестирования, являющимся важным этапом любого проекта разработки программного обеспечения. Вы узнаете какие бывают тесты, на что необходимо обращать внимание при планировании и выполнении тестов. В проектах машинного обучения тестирование тоже необходимо. При этом, из-за особенностей проектов машинного обучения, тестирование в таких проектах имеет свои специфические нюансы. Информация, полученная при изучении этого юнита, позволит лучше понять идеологию тестирования и подготовиться к следующим юнитам, в которых будут рассматриваться инструменты для тестирования их применение для решения практических задач.

Содержание юнита:

Назначение тестирования в любом проекте разработки программного обеспечения одинаковое: с помощью тестов проверяют, что после внесенных изменений в программном коде и настройках разрабатываемая система продолжает работать в соответствии с функциональными и техническими требованиями, по которым эта система разрабатывается. Требования бывают самые разнообразные, затрагивающие как небольшие части программного обеспечения, так и всю систему целиком, в том числе во взаимодействии с окружающими системами. Кроме того, проектные команды разработки как правило создают собственные тесты, не относящиеся к бизнес-метрикам, чтобы контролировать качество разработки на внутренних этапах и уменьшить вероятность попадания ошибки в следующий этап разработки. Это позволяет упреждающе обнаруживать проблему и сокращает расходы на поиск и устранение неисправности в сложных проектах, состоящих из множества этапов и различных компонентов.

Тестирование программного обеспечения можно классифицировать разными способами. Например, можно сделать разделение по способу выполнения тестов:

- **Ручное тестирование** выполняется специалистом, взаимодействующим с системой через интерфейсы, например, GUI, API, устройства ввода-вывода. Этот вид тестирования является очень затратным, так как объем выполняемой работы ограничен человеческими возможностями. Скорость такого тестирования невелика. Высока вероятность ошибки из-за «человеческого фактора». На больших объемах этот вид тестирования менее предпочтителен. Лучше всего это тестирование подходит для задач, где при тестировании необходимо проявить смекалку, придумать нестандартную ситуацию, которую нельзя описать шаблонным сценарием и автоматизировать.
- **Автоматическое тестирование** применяется для проверки с использованием стандартных повторяемых операций, которые можно описать в виде шаблонов тестирования. С использованием алгоритмов можно выполнить большой объем тестирования в короткое время. Можно запустить тестирование в течении продолжительного времени, чтобы оценить стабильности системы под длительной нагрузкой. Автоматизация позволяет выполнять тесты в неурочное время, например, ночью. В большинстве проектов тесты как раз проводятся по ночам, когда разработка не ведется. Автоматическое тестирование является важной частью

непрерывной интеграции (CI) и непрерывного развертывания (CD), которые мы обсуждали в предыдущих модулях.

Также виды тестирования отличаются по выполняемым задачам и месту в общей архитектуре программного обеспечения:

- модульное тестирование расположено ближе всего к исходному программному коду, проверяет работу отдельных функций и модулей,
- функциональное тестирование проверяет соответствие бизнес требованиям, изложенным в техническом задании, то есть проверяет правильность функционирования разработанной системы,
- интеграционное тестирование проверяет совместную работу нескольких компонентов системы и их взаимодействие между собой,
- системное тестирование проверяет работу системы в целом,
- приемочные тесты предполагают проверку выполнения требований пользователя, описанных в приемо-сдаточной документации, часто пишутся на основе технического задания, в их выполнении как правило принимают участие специалисты заказчика системы,
- регрессионное тестирование проверяет, что внесенные в код или настройки изменения не приводят к найденным ранее и уже известным ошибкам, позволяют убедиться, что изменения не вызывают ошибки повторно,
- тестирование стабильности и производительности проверяет работу системы под нагрузкой, в течении длительного времени, с использованием больших объемов данных, при значительном увеличении параметров, позволяет проверить защитные механизмы системы при перегрузках,
- тестирование безопасности проверяет возможные уязвимости в коде и архитектуре, часто требует привлечения профильных специалистов, пентестеров, однако распространено и применение методов машинного обучения или правил, по которым обнаруживаются уязвимости, в частности, github может подсказывать разработчикам о тех или иных угрозах, которые содержатся в коде программ, опубликованных в репозитории,
- тестирование пользовательского интерфейса (UX/UI, User eXperience User Interface) проверяет правильность работы пользовательских интерфейсов, непротиворечивость, доступность, работоспособность.

Кроме перечисленных существует еще множество других видов тестирования для отдельных специфических задач: тестирование правильности установки программного обеспечения, тестирование удобства использования, тестирование на отказ и восстановление и пр.

Ручное тестирование обладает бесспорным преимуществом в ситуациях, когда важно проверять работу не по сценарию, искать неизвестные уязвимости в работе системы, создавать непредвиденные сложности. Однако разрабатываемые в настоящее время информационные системы как правило имеют сложную архитектуру из множества компонентов, в их разработке участвуют большие команды разработчиков, такие системы работают в сложных окружениях во взаимодействии с другими системами, на различных платформах. Поэтому тестирование сложных систем невозможно без автоматизации. Для разработки скриптов автоматизации тестирования используют разные языки

программирования, например, Java, Python, или bash скрипты. Также существуют специальные библиотеки и фреймворки для создания и выполнения тестов.

В таблице ниже приведены некоторые инструменты автоматического тестирования.

unittest	https://docs.python.org/3/library/unittest.html	Python библиотека для написания unit тестов.
pytest	https://docs.pytest.org	Python фреймворк для написания небольших удобно читаемых тестов, а также для их масштабирования на большие проекты для функционального тестирования приложений или библиотек.
Robot Framework	https://robotframework.org/	Фреймворк для разработки приемочных автотестов, позволяющий автоматизировать действия пользователей.
Mocha	https://mochajs.org/	Специализированная среда для тестов для javascript
JUnit	https://junit.org/junit5/	Библиотека для модульного тестирования программного обеспечения, написанного на языке Java.
TestProject	https://testproject.io/	Бесплатный фреймворк для автоматизации тестирования. Используется для тестирования мобильных, web и других видов приложений с помощью открытого SDK. Поддерживаются html отчеты.

Конечно же перечень библиотек и фреймворков для автоматизации гораздо шире и постоянно пополняется.

Давайте теперь познакомимся с важными терминами тестирования.

- испытательный стенд (test fixture), это аппаратная и программная инфраструктура для выполнения тестов, как правило создание и удаление этой инфраструктуры унифицируется и автоматизируется, испытательный стенд может иметь сложную топологию и включать множество компонентов, требующих специальной настройки,
- тестовый случай, тест кейс (test case) это минимальная единица или блок процедуры тестирования, проверяет конкретную рабочую ситуацию или сценарий,
- набор тестов (test suite) это наборов тестов и/или тест кейсов, используется для объединения тестов, которые должны быть выполнены вместе, обычно по принципу группировки тестов для модуля, функциональности, интерфейса,
- исполнитель тестов (test runner) это компонент, который управляет выполнением тестов и предоставляет пользователю результат, может содержать графический интерфейс,
- отчет о тестировании (test report) это документ, который появляется в результате выполнения тестов, содержит результаты.

Простые действия, выполняемые при тестировании, соответствуют так называемому паттерну Arrange-Act-Assert, или “3А”, который организует и упорядочивает входные данные, действия и результаты при выполнении тестовых действий:

- arrange – настроить различные входы для тестирования,
- act – применить входные данные к тестируемому компоненту,
- assert – подтвердить получение ожидаемого результата.

Вышеописанные фреймворки, в том числе unittest и pytest, которые мы будем рассматривать подробнее в дальнейшем, позволяют выполнять эти действия.

В разных командах тестирование организуется по-разному. Иногда за написание тестов несут ответственность сами разработчики. В других случаях существуют специальные отделы тестирования с делением на junior, middle и senior специалистов и руководителем тестирования. Поэтому и степень погружения специалиста по тестированию в детали проекта бывает разная: от самого поверхностного уровня до глубокого погружения в архитектуру продукта, наравне с ведущими разработчиками и архитекторами. Конечно же второй вариант означает более высокую и востребованную квалификацию, которая позволяет тестировщику быть эффективным и уважаемым участником команды, демонстрируя свою полезность в достижении общего успеха.

В проектах машинного обучения есть свои особенности тестирования, связанные с тем, что в машинном обучении используются специальные объекты: данные, модели, среды, конвейеры. Необходимо проверять не только то, что процесс доработал до конца, но и то, что соблюдаются необходимые характеристики в данных и моделях, например, отсутствие аномалий. Поскольку проекты машинного обучения связаны с большими объемами информации, то невозможно обойтись без автоматизации тестов. **Специфику создания, тестирования, вывод в производственное окружение и эксплуатацию учитывает концепция MLOps, используя лучшие методы и инструменты из инженерии программного обеспечения DevOps, а также собственные, уникальные для машинного обучения.**

Для решения практических задач тестирования в проектах машинного обучения достаточно средств, используемых в обычных проектах разработки программного обеспечения, например unittest или pytest. Автоматизированные тесты с использованием этих библиотек делают часть пайплайна проекта разработки, обучения и вывода в производственную среду модели машинного обучения, позволяя осуществлять проверки качества на отдельных этапах и локализовать проблему как можно раньше, что позволяет существенно снизить расходы на поиск и устранение ошибок.

Однако в проектах машинного обучения есть то, что отличает эти проекты от других, а именно: наборы данных или датасеты. Наличие ошибок в датасетах, как тренировочных, так и реальных, появляющихся в практической эксплуатации модели, может приводить к ошибкам, которые очень сложно диагностировать. Поэтому в проектах машинного обучения большое внимание уделяют проверке данных.

В основных этапах жизненного цикла модели машинного обучения актуальны обычные для проектов разработки виды тестирования:

- на этапе создания кода для проектов актуальны модульные тесты, проверяющие корректность работы отдельных,
- при включении модели машинного обучения в общую систему или при интеграции модели с другими системами актуально выполнение интеграционных тестов,
- в процессе эксплуатации модели в производственной среде важна организация мониторинга, тестирующего параметры работы модели под реальной нагрузкой и на реальных производственных данных.



Рисунок «Задачи и объекты тестирования в проекте машинного обучения».

Кроме тестирования кода и интеграционных тестов системы в машинном обучении можно выделить три важных категории тестов, специфических для машинного обучения:

- тестирование признаков (features, фичей) применяется, чтобы проверить важность признака с точки зрения бизнес-цели, например, улучшает ли данный признак точность модели машинного обучения, может быть частью процесса конструирования признаков (feature-engineering),
- тестирование данных на различных этапах, в том числе при сборе данных, конструировании признаков, обучении модели, эксплуатации модели,
- тестирование моделей, что является по сути главной частью машинного обучения, проверяется качество работы модели, ее устойчивость к шумам в данных, способность давать надежные результаты, соблюдение расходов аппаратных и программных ресурсов в рамках технических требований,
- тестирование инфраструктуры ML, в ходе которого проверяется правильность конфигурации элементов инфраструктуры, наличие необходимых ресурсов, правильность зависимостей библиотек. Может выполняться как перед развертыванием системы, так и регулярно в процессе эксплуатации, для контроля параметров системы.

При практическом тестировании модели машинного обучения используют тестовые данные - наборы параметров и датасетов, подготовленные и упорядоченные для последующего использования в тестах, с заранее известным результатом, получение которого проверяется. Тестовые данные бывают валидные и невалидные. На валидных

данных проверяется то насколько правильно программа их обрабатывает. На невалидных данных проверяется устойчивость алгоритма к разным шумам, аномалиям, нестандартным ситуациям. Создание тестовых наборов данных может выполняться:

- вручную, с использованием специальных скриптов,
- с использованием программ автоматизации, в том числе специальных генераторов данных,
- с использованием основных средств и функций разработки проекта, например работа напрямую с базой данных на уровне back-end для вноса тестовых данных. Требуется внимательность, полезно иметь UI или специальные утилиты, осуществляющие контроль и проверку таких действий и дающие возможность быстро вернуть назад сделанные изменения.

Тест

1. Укажите виды тестирования (0.25)
 - a. модульное
 - b. корпусное
 - c. интеграционное
 - d. дифференциальное
2. Что проверяет регрессионное тестирование? (0.25)
 - a. Возможность решения задачи с помощью линейной регрессии
 - b. Корреляцию между величинами
 - c. **Отсутствие ранее известных ошибок после внесения изменений**
 - d. Отсутствие новых ошибок после внесения изменений
3. Отметьте фреймворки для автоматизации тестов (0.25)
 - a. **pytest**
 - b. windows
 - c. **unittest**
 - d. assembler
4. Что необходимо тестировать в проекте машинного обучения? (0.25)
 - a. модели
 - b. данные
 - c. признаки
 - d. **инфраструктуру**

Итоги/выводы

В этом юните вы узнали о важном этапе любого проекта разработки программного обеспечения и, в том числе, в проектах машинного обучения: о тестировании. Теперь вы знаете каких видов бывает тестирование и чем отличаются друг от друга разные виды: модульное, функциональное, системное, интеграционное, нагрузочное и другие виды тестирования. Также вы знаете плюсы и минусы ручного и автоматического видов тестирования, после чего не должно остаться сомнений в необходимости использования автоматизированного тестирования в проектах машинного обучения, поскольку в таких проектах много данных и параметров для обучения.

В проектах машинного обучения важную роль играют данные, поэтому в следующем юните мы подробнее рассмотрим вопрос тестирования данных.

В юните мы перечислили и кратко описали некоторые фреймворки и библиотеки для автоматизации тестирования. В третьем юните мы подробнее рассмотрим unittests и pytest и используем их для решения практической задачи.

Модуль 6. Юнит 2. Тестирование данных

Введение: Данные составляют важную часть проекта машинного обучения. Без данных невозможно обучить модель и не имеет смысла эксплуатация модели. От ошибок в данных проблемы могут быть как в обучении, так и в эксплуатации. После того как такая ошибка случилась ее сложно обнаружить и понять причину ее появления, так как такие ошибки нельзя обнаружить стандартными средствами разработки, например, пошаговой отладкой программы. Зачастую, для понимания причины ошибки и ее влияния на процесс необходимо заново повторить эксперимент, что затратно или невозможно. Поэтому целесообразно предпринять все меры для того, чтобы ошибки в данных не появлялись вообще, в том числе с использованием тестирования данных. В этом юните вы познакомитесь с практическими аспектами тестирования данных в проектах машинного обучения.

Содержание юнита:

Использование данных в реальных практических проектах машинного обучения связано с серьезными сложностями:

- данных не хватает,
- формат и способ получения данных могут меняться,
- источники данных работают ненадежно,
- нужные данные принадлежат другой компании,
- данные низкого качества,
- данные занимают слишком много места,
- ...

Специалист в области машинного обучения как правило сталкивается именно с этими сложностями при переходе от теории и решении соревновательных задач к практическим производственным проектам машинного обучения. Поэтому, в проектах машинного обучения кроме Data Science и Machine Learning специалистов очень значимой является роль специалистов Data Quality, работающих с данными и отвечающих за их качество. Data Quality инженер следит за источниками данных, конвейерами получения данных, отслеживает ошибки в получении и обработке данных, проверяет качество данных. На этом этапе делаются выводы о качестве данных на входе и выходе, характеристиках данных, достаточности данных, соблюдении «контракта данных». Data Quality является частью более важной парадигмы Data Management, которая выходит за рамки этого учебного курса и с которой вы можете при желании ознакомиться по ссылкам, приведенным в конце этого модуля.

При проверке качества данных может проверяться множество параметров, их перечень и требования к ним зависят от специфики конкретного проекта, однако можно выделить наиболее распространенные и часто проверяемые параметры данных:

- уникальность: если в наборе данных много повторяющихся элементов, то такие данные имеют более низкий приоритет для работы с моделью, поскольку не дают полезной информации для обучения модели,
- консистентность, непротиворечивость: в данных не должно быть противоречий, влияющих на обучение модели, например поле «Дата отправки» должна

предшествовать полю «Дата получения», если нам важно запомнить такой порядок как важное свойство процесса,

- интегральность: наличие признака, по которому можно связать данные из разных источников, если в модели используются данные из разных источников, отсутствие такого признака делает невозможным их использование,
- целостность: отсутствие пропусков в данных,
- своевременность: данные должны быть за актуальный временной период, имеющий смысл для обучения модели, содержащий характерные для процесса свойства, которые мы хотим выявить с помощью модели машинного обучения,
- соответствие синтаксису: наиболее часто возникают проблемы с датами, суммами, адресами. Пример такой проверки синтаксиса: правило, что идентификатор должен состоять из 10 цифр,
- соответствие статистическим характеристикам,
- корректность данных: при этой проверке в данных надо проверять:
 - соответствие типам,
 - отсутствие поврежденных, искаженных данных,
 - количественные характеристики, например, совпадение общего количества записей до и после обработки,
 - правильности названий признаков, столбцов в датасетах, при изменении формата исходных данных, например, csv файла или json в API, можно столкнуться с тем, что изменились названия признаков.

Полезным инструментом, который позволяет проверять данные по заранее определенным шаблонам, являются регулярные выражения. Это очень мощный инструмент, изучение которого выходит за рамки данного курса, если вы с ним незнакомы, то рекомендуем обратиться к статьям и книгам, описывающим этот инструмент. Здесь же мы ознакомим только с некоторыми опциями, которые можно применить при тестировании данных. Регулярные выражения это инструмент, который можно применить для извлечения данных по определенному шаблону, который можно задавать очень гибко. В python есть соответствующий модуль `re` для работы с регулярными выражениями, который позволяет удобно создавать и применять шаблоны регулярных выражений. Вот пример такого шаблона, который проверяет формат даты:

```

Ввод [1]: import re

Ввод [2]: data = "01/02/2013"

Ввод [3]: pattern = re.compile(".")
          print(pattern.findall(data))

          ['0', '1', '/', '0', '2', '/', '2', '0', '1', '3']

Ввод [4]: pattern = re.compile("[0-9][0-9]/[0-9][0-9]/[0-9]{4}")
          print(pattern.findall(data))

          ['01/02/2013']

Ввод [5]: wrong_data = "01/фев/2013"

Ввод [6]: pattern = re.compile("[0-9][0-9]/[0-9][0-9]/[0-9]{4}")
          print(pattern.findall(wrong_data))

          []

```

Видно, что во втором случае шаблон обнаружил ошибку, не смог определить данные по заданному шаблону.

Давайте рассмотрим наиболее распространенные варианты синтаксиса шаблоны регулярных выражений, которые можно проверять при проверке данных:

1. Основной синтаксис

.	один символ кроме новой строки
\.	точка, обратный слеш \ убирает специальные символы
\d	одна цифра
\D	один символ, кроме цифры
\w	один буквенный символ, включая цифры
\W	один символ, кроме буквы и цифры
\s	один пробельный символ, включая табуляцию и перенос строки
\S	один непробельный символ
\b	границы слова
\n	новая строка
\t	табуляция

2. Модификаторы

\$	конец строки
----	--------------

^	начало строки
ab cd	соответствует ab или de.
[ab-d]	один символ: a, b, c, d
[^ab-d]	любой символ, кроме: a, b, c, d
()	извлечение элементов в скобках
(a(bc))	извлечение элементов в скобках второго уровня

3. Повторы

[ab]{2}	2 непрерывных появления a или b
[ab]{2,5}	от 2 до 5 непрерывных появления a или b
[ab]{2,}	2 и больше непрерывных появления a или b
+	одно или больше
*	0 или больше
?	0 или 1

С использованием таких конструкций удобно составлять правила проверки соответствия исследуемых данных определенному шаблону и автоматизировать такую проверку с помощью скриптов.

Важен контроль качества данных на всей цепочке – «data chain». Если данные получаются в результате цепочки операций, например, в ETL-процессе, то необходимо проверять правильность всех изменений, произведенных ETL-процессом. Если данные хранятся в SQL базе, то проверка может быть реализована с помощью тестирующих SQL запросов, проверяющих вышеуказанные характеристики на небольших объемах считываемых данных.

Практическая реализация тестирования такой цепочки тестирования может быть выполнена с использованием уже рассмотренного в Модуле 5 инструмента Apache Airflow, где операции тестирования можно добавлять в общий конвейер проекта, описываемый с помощью графа операций DAG. Airflow поддерживает операторы для выполнения таких задач, например, оператор `airflow.contrib.operators.bigquery_check_operator.BigQueryCheckOperator`, который проверяет данные с использованием SQL запроса к базе данных или другому источнику информации (например, JSON).

Например, мы хотим проверить JSON на отсутствие в `event_id` пропущенных или null значений. JSON имеет следующую структуру:

```
{ "timestamp":1500233640,"user_id":1234,"event_id":"view",...}
{"timestamp":1500233641,"user_id":4321,"event_id":"post",...}
```

Вы можете добавить в файл с описанием графа операций Airflow DAG оператор, осуществляющий такую проверку следующим образом

```
# Проверяем, что результат равен 0.  
expected = 0
```

```
sql = """  
SELECT COUNT(*) AS event_id_null_count  
FROM event_log.event_log_{{ id }}  
WHERE JSON_EXTRACT(event_id) IS NULL  
"""
```

```
checker = BigQueryValueCheckOperator(  
    dag=dag,  
    task_id='bq_checker',  
    bigquery_conn_id='bq_connection_id',  
    sql=sql,  
    pass_value=expected)
```

Далее этот оператор следует встроить в общий граф с использованием стандартных средств Airflow, которые вы уже изучили.

Полезной опцией системы тестирования данных, особенно применяемой в реальной эксплуатации, является возможность направлять уведомления в мессенджеры сотруднику, осуществляющему оперативную поддержку или отвечающему за качество данных. Такая опция поддерживается, например, в Apache Airflow. Для подключения этой возможности необходимо создать и добавить в Airflow DAG следующий оператор:

```
slack = SlackAPIPostOperator(  
    dag=dag,  
    task_id='post_error_message_to_slack',  
    token=YOUR_SLACK_TOKEN,  
    channel='#data-quality',  
    username='airflow',  
    text='event_log on {{ yesterday_ds_nodash }} has record(s) whose event_id is null.',  
    trigger_rule=TriggerRule.ALL_FAILED)
```

Тест

1. Что необходимо тестировать в данных в проекте машинного обучения? (0.25)
 - a. Соответствие новых данных заданным типам
 - b. Изменение структуры, например, названий признаков
 - c. Сохранение статистических характеристик
 - d. Надежность источника
2. Что проверяет Data Quality инженер (0.25)
 - a. Уникальность данных
 - b. Целостность данных
 - c. Соответствие синтаксису
 - d. Своевременность данных
3. Что такое невалидные тестовые данные? (0.25)
 - a. Данные, непригодные для проверки модели
 - b. Данные с ошибками для проверки работы во внештатных ситуациях

- c. Данные из непроверенных источников
 - d. Данные, ссылка на которые потеряла актуальность
4. Что такое регулярные выражения (0.25)
- a. Корректные математические выражения
 - b. Формальный язык поиска данных и манипуляций с ними**
 - c. Повторяющиеся регулярно операторы, например, в цикле
 - d. Функции, запускаемые по расписанию

Итоги/выводы

В этом юните отдельно рассмотрен процесс тестирования данных в проектах машинного обучения. Теперь вы знаете с какими проблемами можно столкнуться при использовании некачественных данных в проекте машинного обучения и о важной роли Data Quality инженера. Кроме того, теперь вы знаете о регулярных выражениях и о том как их можно применять для проверки правильности данных, а также о более сложном примере проверки с использованием типовых операторов графа операций DAG Airflow.

Модуль 6. Юнит 3. Модульное и функциональное тестирование

Введение: В этом юните вы узнаете о стандартных python библиотеках для создания модульных и функциональных тестов, а также выполните практическое задание с применением этих библиотек.

Содержание юнита:

Наиболее популярными для создания функциональных тестов являются python библиотеки unittest и pytest. Большинство задач тестирования в проектах разработки программного обеспечения вполне успешно решается с использованием этих библиотек. В этом юните мы их рассмотрим подробнее.

Создание unittest было вдохновлено другим популярным инструментом, JUnit, библиотекой для тестирования программного обеспечения на языке Java. Ранее эта библиотека называлась pyunit. Версия для python2 называется unittest2. Unittest поддерживает важные функции:

- автоматизация тестирования,
- переиспользование кода для выполнения одинаковых операций в разных тестах, например, подготовка к тесту и завершение теста,
- агрегация тестов в коллекции,
- независимая работа тестов от контролирующей системы.

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Пример использования unittest, с сайта <https://docs.python.org/3/library/unittest.html#>

Создаваемый в этом примере тестовой кейс наследуется от стандартного класса unittest.TestCase и содержит методы, выполняющие тестовые действия с помощью стандартных функций unittest. Каждый тест содержит метод **assertEqual()** для проверки выполнения равенства, **assertTrue()** или **assertFalse()** для проверки выполнения условия и **assertRaises()** для проверки появления исключительной ситуации. Метод **setUp()** позволяет описать действия, выполняемые в начале каждого теста, а метод **tearDown()** содержит действия, выполняемые при завершении теста.

После запуска данного скрипта вы увидите в консоли результат выполнения тестов

```
...
-----
Ran 3 tests in 0.000s

OK
```

Результат выполнения unittest, с сайта <https://docs.python.org/3/library/unittest.html#>

unittest можно запускать из консоли, например, с помощью команды

python -m unittest test_module1 test_module2

При этом test_module1 и test_module2 должны быть python модулями, то есть директориями, содержащими файл `__init__.py`.

Опции unittest:

- `-b (--buffer)` - вывод программы на консоль при неудачном тесте будет показан,
- `-c (--catch)` - после нажатия комбинации Ctrl+C во время выполнения теста ожидается завершение текущего теста и затем сообщаются результаты на данный момент. Второе нажатие комбинации Ctrl+C вызывает обычное исключение KeyboardInterrupt,
- `-f (--failfast)` - выход сразу после первого неудачного теста,
- `--locals` (начиная с Python 3.5) - показывает локальные переменные для неуспешных тестов,

Для обнаружения тестов используется опция `discover` при этом тесты должны быть модулями

python3 -m unittest discover

Эту команду можно выполнять со следующими параметрами:

- `-v (--verbose)` - подробный вывод,
- `-s (--start-directory) directory_name` - директория начала обнаружения тестов (текущая по умолчанию),
- `-p (--pattern) pattern` - шаблон названия файлов с тестами (по умолчанию `test*.py`),
- `-t (--top-level-directory) directory_name` - директория верхнего уровня проекта (по умолчанию равна `start-directory`).

Из других полезных вещей – unittest содержит функции-декораторы, например, позволяющие пропустить отдельный тест (`@skip`, `@skipIf`).

В итоге, unittest хорошо подходит для модульного тестирования, позволяет писать и выполнять тесты, поддерживает генерацию отчетов. К недостаткам, которые не соответствуют парадигме python и поэтому могут оказаться неудобными, можно отнести:

- недостаточную понятность кода в виду использования абстракций при наследовании классов,
- большое количество шаблонного кода,
- использование принятой в JUnit camel-нотации (например, CamelNotation) вместо принятой в python snake-нотации (`snake_notation`).

Более гибкая и функциональная альтернатива для unittest это библиотека `pytest`. `pytest` часто используется для написания тест кейсов для API. В отличие от unittest библиотека `pytest` не входит в стандартный набор программ, устанавливаемый вместе с python, поэтому ее необходимо устанавливать отдельно.


```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest test.py
Command 'pytest' not found, but can be installed with:
sudo apt install python-pytest
ychernyshov@ychernyshov-VirtualBox:~/tests$
```

Знакомство с pytest проще всего начать с примера. Создадим папку tests

```
ychernyshov@ychernyshov-VirtualBox:~$ mkdir tests
ychernyshov@ychernyshov-VirtualBox:~$ cd tests
ychernyshov@ychernyshov-VirtualBox:~/tests$
ychernyshov@ychernyshov-VirtualBox:~/tests$
```

и в ней создадим файл test.py, содержащий тест. Вот так выглядит тест

```
def test1():
    assert 2+2 == 4
```

Этот тест легко запустить с помощью pytest и увидеть результаты выполнения теста

```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest test.py
===== test session starts =====
platform linux2 -- Python 2.7.18, pytest-4.6.9, py-1.8.1, pluggy-0.13.0
rootdir: /home/ychernyshov/tests
collected 1 item

test.py . [100%]

===== 1 passed in 0.02 seconds =====
ychernyshov@ychernyshov-VirtualBox:~/tests$
```

Давайте внесем в файл намеренную ошибку, чтобы увидеть как в интерфейсе pytest будет отображен неправильно выполненный тест, для этого поправим файл test.py

```
def test1():
    assert 2+2 == 5
```

и после запуска pytest увидим результат

```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest test.py
===== test session starts =====
platform linux2 -- Python 2.7.18, pytest-4.6.9, py-1.8.1, pluggy-0.13.0
rootdir: /home/ychernyshov/tests
collected 1 item

test.py F [100%]

===== FAILURES =====
_____ test1 _____
def test1():
> assert 2+2 == 5
E       assert (2 + 2) == 5

test.py:2: AssertionError
===== 1 failed in 0.04 seconds =====
```

Более того, если выполнить `pytest` с флагом `-v`, то можно увидеть еще более подробное описание, указывающее на причину ошибки, например, для нашего случая будет в явном виде указано правильное значение 4 вместо неправильного 5:

```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest -v test.py
===== test session starts =====
platform linux2 -- Python 2.7.18, pytest-4.6.9, py-1.8.1, pluggy-0.13.0 -- /usr
/bin/python2
cachedir: .pytest_cache
rootdir: /home/ychernyshov/tests
collected 1 item

test.py::test1 FAILED [100%]

===== FAILURES =====
test1

  def test1():
>     assert 2+2 == 5
E       assert 4 == 5
E         -4
E         +5

test.py:2: AssertionError
===== 1 failed in 0.02 seconds =====
ychernyshov@ychernyshov-VirtualBox:~/tests$
```

А если для указания ошибки необходимо выделить позицию ошибки в тексте, то `pytest` использует для этого символ “^”. Например, для неправильного символа в строке

```
def test1():
    assert "hello" == "Hello"
```

будет указана его позиция:

```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest -v test.py
===== test session starts =====
platform linux2 -- Python 2.7.18, pytest-4.6.9, py-1.8.1, pluggy-0.13.0 -- /usr
/bin/python2
cachedir: .pytest_cache
rootdir: /home/ychernyshov/tests
collected 1 item

test.py::test1 FAILED [100%]

===== FAILURES =====
test1

  def test1():
>     assert "hello" == "Hello"
E       AssertionError: assert 'hello' == 'Hello'
E         - hello
E         ? ^
E         + Hello
E         ? ^

test.py:2: AssertionError
===== 1 failed in 0.04 seconds =====
```

При планировании тестирования вы можете заставить pytest пропустить тест, используя функции- декораторы `@pytest.mark.skip()` или `pytest.mark.skipif()`.

И, в заключение знакомства с pytest давайте познакомимся с понятием фикстур (fixture). Фикстуры необходимы для структурирования тестирующего кода, давая возможность добавлять стандартные части кода, необходимые для конкретной программной среды. Это функции, выполняемые pytest до или после тестовых функций для инициализации или завершения теста. Код в фикстуре может выполнять любые операции, например, вы можете использовать фикстуры, чтобы загрузить датасет. Это позволяет правильно инициализировать тестовое окружение, а также корректно освободить ресурсы после завершения работы.

Вот простой пример фикстуры:

```
import pytest

@pytest.fixture()
def init_data():
    return(111)

def test1(init_data):
    assert init_data == 111
```

И результат функции с ее выполнением

```
ychernyshov@ychernyshov-VirtualBox:~/tests$ pytest -v test.py
===== test session starts =====
platform linux2 -- Python 2.7.18, pytest-4.6.9, py-1.8.1, pluggy-0.13.0 -- /usr
/bin/python2
cachedir: .pytest_cache
rootdir: /home/ychernyshov/tests
collected 1 item

test.py::test1 PASSED [100%]

===== 1 passed in 0.01 seconds =====
ychernyshov@ychernyshov-VirtualBox:~/tests$
```

Больше опций можно получить в документации pytest на официальной странице модуля <https://docs.pytest.org/>, также удобно можно получить пользовательскую справку командой `pytest -h`.

Тест

1. От какого фреймворка унаследовал идеологию и часть функций unittest? (0.25)
 - a. Java
 - b. pytest
 - c. JUnit**
 - d. Jenkins
2. Какие методы в unittest проверяют выполнение условия? (0.25)
 - a. `checkCondition`
 - b. `assertTrue`**

- c. **assertFalse**
 - d. `assert`
3. Что такое фикстуры в `pytest`? (0.25)
- a. Специальные константы
 - b. **Специальные функции-декораторы**
 - c. Специальные функции агрегаторы
 - d. **Функции для выполнения типовых операций в начале или конце теста**
4. Какой стандартный метод в `python` и `pytest` используется для проверки утверждений? (0.25)
- a. `check`
 - b. **`assert`**
 - c. `case`
 - d. `statement`

Итоги/выводы

В этом юните вы познакомились с `python` библиотеками для автоматизации модульного тестирования `unittest` и `pytest`.

Итоги/выводы по модулю

В этом модуле были рассмотрены задачи, связанные с тестированием в проектах машинного обучения. Тестирование играет важную роль в любом проекте разработки программного обеспечения, а машинное обучение вносит свои особенности в этот процесс.

Теперь вы знаете основные виды тестирования, например,

- модульное - применяется для тестирования отдельных модулей,
- интеграционное - для тестирования взаимодействия компонентов системы между собой,
- системное - для тестирования системы в целом,
- нагрузочное - для проверки работы системы в течение долгого времени под нагрузкой.

Вы узнали, что ручное тестирование является более затратным и подходит больше для нестандартных ситуаций, в которых необходимо проявить творческое мышление, и которые невозможно описать типовыми кейсами. Для тестирования больших проектов, содержащего большое количество стандартных повторяемых операций, лучше подходит автоматическое тестирование, по этой причине в проектах машинного обучения предпочтительнее применять автоматизацию тестирования.

Вы знаете основные библиотеки и фреймворки для автоматизации тестирования и умеете применять две типовых python библиотеки для автоматизации тестирования: unittest и pytest.

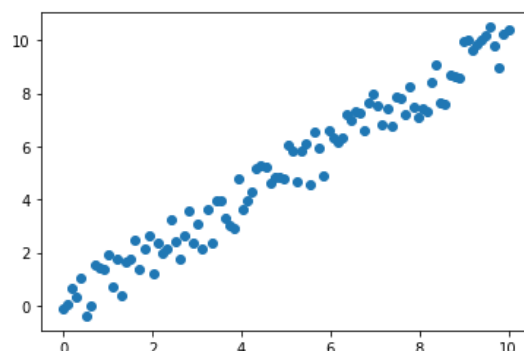
Практическое задание

- Цель задания: применить средства автоматизации тестирования python для автоматического тестирования качества работы модели машинного обучения на различных датасетах.
- Содержание задания:
 - Создать три датасета с «качественными» данными, на которых можно обучить простую модель линейной регрессии, например

```
import matplotlib.pyplot as plt
import numpy as np
```

```
xs = np.linspace(0, 10, 100)
ys = xs + np.random.random(100)*2-1
```

```
plt.scatter(xs, ys)
plt.show()
```

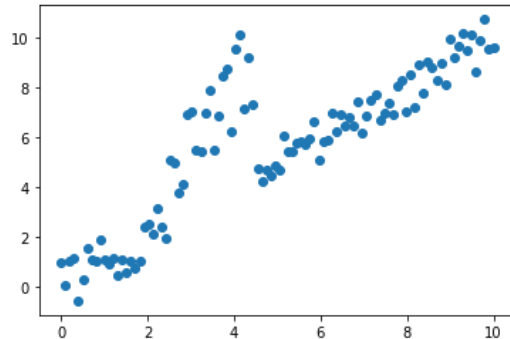


- На одном из этих датасетов обучить модель линейной регрессии

- Создать датасет с шумом в данных, например

```
xs = np.linspace(0, 10, 100)
ys = xs + np.random.random(100)*2-1
ys[25:45] *= 2

plt.scatter(xs, ys)
plt.show()
```



- Провести тестирование работы модели на разных датасетах с использованием pytest, анализируя качество предсказания, обнаружить проблему на датасете с шумами.
 - Критерии: данное задание необходимо полностью выполнить в виде jupyter ноутбука и предоставить его на проверку.
 - Подсказка: вы можете записать содержимое ячейки jupyter ноутбука в отдельный файл с помощью команды
`%%writefile"имя файла"`
А также можете выполнить любую linux команду прямо из ячейки jupyter ноутбука, с помощью синтаксиса
`! "имя команды"`

Список источников

Стандарты Data Management

<https://www.iso.org/committee/45342/x/catalogue/>

DAMA-DMBOK: Data Management Body of Knowledge: 2nd Edition

<https://www.amazon.com/DAMA-DMBOK-Data-Management-Body-Knowledge/dp/1634622340>

Описание задач и типов тестирования при непрерывном развертывании от Jira

<https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing>

Описание основ юнит тестирования

<https://docs.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2022>

Обзорная статья о различных фреймворках тестирования

<https://habr.com/ru/company/otus/blog/576760/>

Официальная страница pytest

<https://docs.pytest.org/>

Официальная страница unittest

<https://docs.python.org/3/library/unittest.html>

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:
 - a. Бэкенд проекта машинного обучения на примере Django.
 - b. Внутреннее взаимодействие. API для взаимодействия отдельных компонентов.
 - c. Внешнее взаимодействие. Web сервер, задачи, инструменты.
 - d. Базы данных.
 - e. 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 “монокотный”.

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 “монокотный” и в настоящее время является основным в реализации сложных многофункциональных решений.

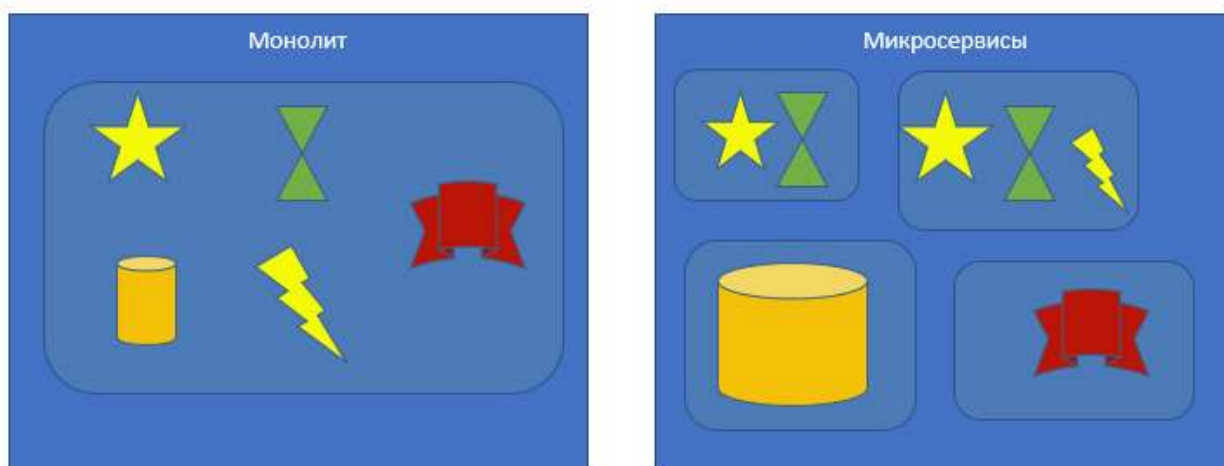


Рисунок “Монокотная и микросервисная архитектура”

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)
 - a. **монолитная**
 - b. монотонная
 - c. макросервисная
 - d. **микросервисная**
2. Что такое API? (0.25)
 - a. **Application Programming Interface**
 - b. Activation Programming Interchange
 - c. Ask Program Independently
 - d. Any Program Interconnection
3. Какие преимущества имеет микросервисная архитектура по сравнению с монолитной? (0.25)
 - a. **гибкое масштабирование**
 - b. **технологическое функциональное распределение компонентов системы**
 - c. возможность использовать open-source
 - d. **оптимизация использования аппаратных средств**
4. Почему микросервисная архитектура более предпочтительна для проектов машинного обучения? (0.25)
 - a. **в проекте ML работают команды, решающие независимые задачи**
 - b. микросервисная архитектура позволяет создавать более простые приложения быстрее

- c. **легче масштабировать решение, например, при увеличении нагрузки на систему**
- d. 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, Prometheus, 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)
 - a. поиск синтаксических и структурных ошибок в коде, связанных с конкретным языком программирования
 - b. подсказки по синтаксису функций
 - c. графическая подсветка синтаксиса (ключевые слова, операторы)
 - d. ускоренная компиляция
2. Какие инструменты используются для мониторинга работы модели? (0.25)
 - a. grafana
 - b. pip
 - c. kibana
 - d. sqlite
3. Какие инструменты используются для автоматизированного сбора данных? (0.25)
 - a. kafka
 - b. linux
 - c. pycharm
 - d. airflow
4. Что относится к базам данных? (0.25)
 - a. sqlite
 - b. postgresql
 - c. kafka
 - d. 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 прохождении 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, метаданных (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
 - a. создание рабочего окружения для работы исследователей данных (Data Researcher) и моделей машинного обучения (ML Researcher) с использованием JupyterHub,
 - b. создание виртуального окружения, установка необходимых библиотек, сохранение параметров виртуального окружения для повторяемости результатов,
 - c. получение и сохранение данных, создание файловой структуры для хранения данных, создание dvc репозитория,
 - d. создание различных моделей машинного обучения для решения задачи,
 - e. анализ данных, формирование обучающего датасета с использованием наиболее важных признаков, выполнение необходимой предобработки признаков (например, заполнение пропущенных значений, преобразование текстовых признаков в числовые),
 - f. выполнение обучения моделей, подбор гиперпараметров,
 - g. сохранение результатов обучения моделей, метрик, весов,
 - h. сохранение кода лучшей модели и данных виртуального окружения в git,
 - i. сохранение использованного датасета в dvc хранилище.
2. улучшение качества работы модели
 - a. загрузка данных p.l.h в новое окружение,
 - b. изменение датасета, добавление новых признаков,
 - c. выполнение обучения на новом датасете,
 - d. сохранение результатов обучения моделей, метрик, весов,
 - e. сохранение кода модели и данных виртуального окружения в git,
 - f. сохранение использованного датасета в dvc хранилище.
3. вывод модели в prod
 - a. выполнение скриптов для организации производственного окружения
 - b. непрерывное развертывание всего решения

- c. загрузка модели
- d. эксплуатация модели на тестовых данных
- e. вывод модели из эксплуатации (остановка сервиса)

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

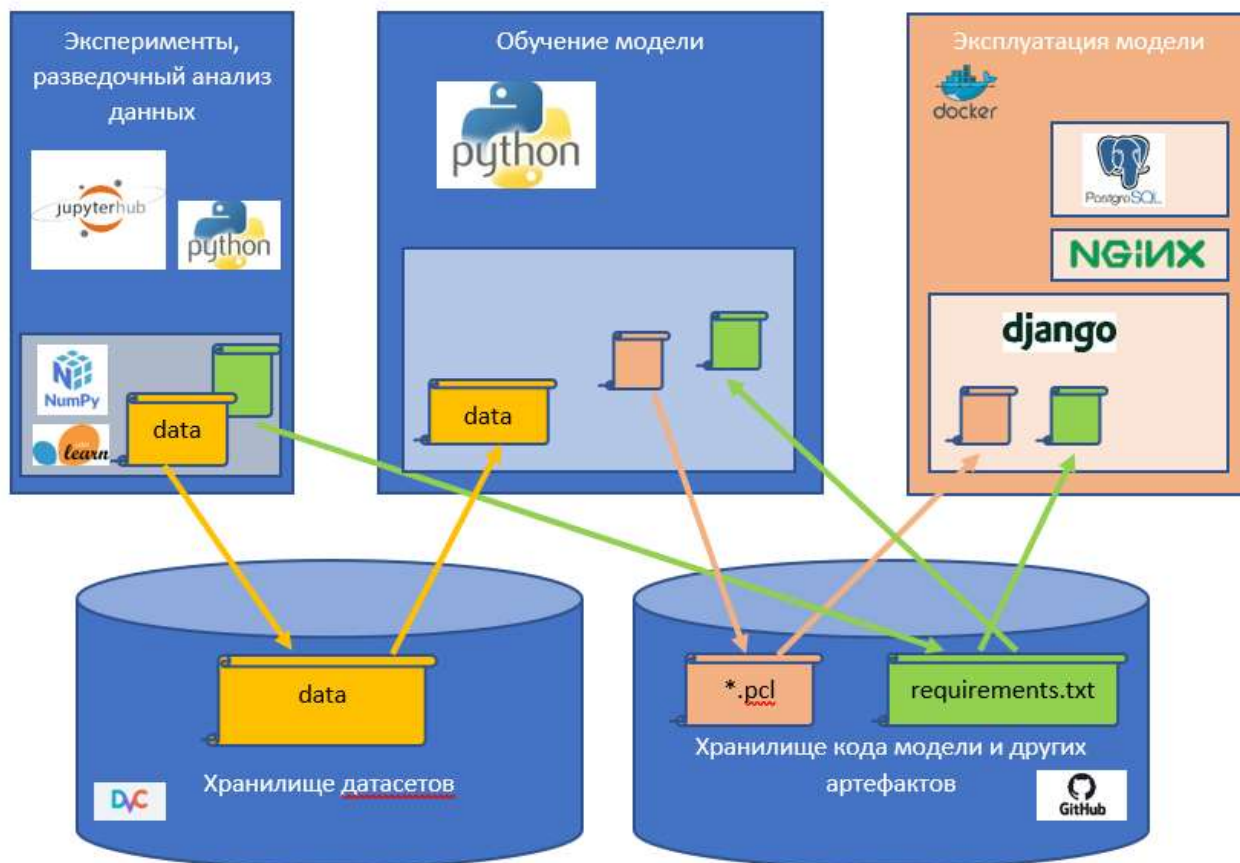


Рисунок “Пример инфраструктуры модели машинного обучения”.

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

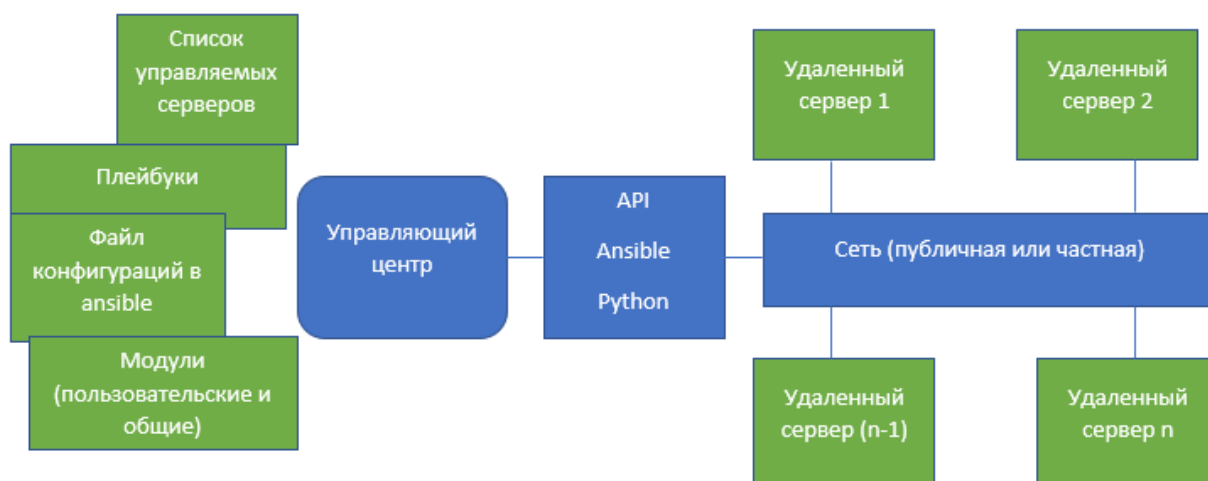


Рисунок “Схема работы Ansible”

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

```
total 20
drwx----- 2 mlops-engineer mlops-engineer 4096 июн  2 13:28 .
drwxr-xr-x 17 mlops-engineer mlops-engineer 4096 июн  4 14:30 ..
-rw----- 1 mlops-engineer mlops-engineer 2635 июн  2 13:18 id_rsa
-rw-r--r-- 1 mlops-engineer mlops-engineer  593 июн  2 13:18 id_rsa.pub
-rw-r--r-- 1 mlops-engineer mlops-engineer  444 июн  2 13:26 known_hosts
mlops-engineer@mlopsengineer-VirtualBox:~$
```

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.

```
mlops-engineer@mlopsengineer-VirtualBox:~$ ssh-copy-id mlops-engineer@192.168.147.3
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
mlops-engineer@192.168.147.3's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'mlops-engineer@192.168.147.3'"
and check to make sure that only the key(s) you wanted were added.

mlops-engineer@mlopsengineer-VirtualBox:~$
```

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:

```
mlops-engineer@mlopsengineer-VirtualBox:~$ ansible all -m ping
192.168.147.3 | FAILED! => {
  "msg": "to use the 'ssh' connection type with passwords, you must install the sshpass program"
}
```

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`

б. 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.

```
mlops-engineer@mlopsengineer-VirtualBox:~$ ansible all -a "uname -a"
192.168.147.4 | CHANGED | rc=0 >>
Linux mlopsengineer-VirtualBox 5.13.0-30-generic #33-20.04.1-Ubuntu SMP Mon Feb 7
14:25:10 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
192.168.147.3 | CHANGED | rc=0 >>
Linux dataengineer-VirtualBox 5.13.0-44-generic #49-20.04.1-Ubuntu SMP Wed May
18 18:44:28 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
```

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.cf`,
- 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:

```
--  
- hosts: all  
  become: true  
  tasks:  
    - name: Install htop  
      apt: name=htop
```

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

```
data-engineer@dataengineer-VirtualBox:~$ htop  
bash: /usr/bin/htop: No such file or directory  
data-engineer@dataengineer-VirtualBox:~$
```

```
ml-engineer@mlengineer-VirtualBox:~$ htop  
  
Command 'htop' not found, but can be installed with:  
sudo apt install htop  
  
ml-engineer@mlengineer-VirtualBox:~$
```

Проверим доступность серверов на которых хотим установить htop

```

mlops-engineer@mlopsengineer-VirtualBox:~$
mlops-engineer@mlopsengineer-VirtualBox:~$
mlops-engineer@mlopsengineer-VirtualBox:~$ ansible all -m ping
192.168.147.4 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.168.147.3 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}

```

Запускаем скрипт ansible

```

mlops-engineer@mlopsengineer-VirtualBox:~$ ansible-playbook ./ansible/script1.y
ml -K
BECOME password:

PLAY [all] *****
*

TASK [Gathering Facts] *****
*
ok: [192.168.147.4]
ok: [192.168.147.3]

TASK [Install htop] *****
*
changed: [192.168.147.4]
changed: [192.168.147.3]

PLAY RECAP *****
*
192.168.147.3      : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.147.4      : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

mlops-engineer@mlopsengineer-VirtualBox:~$ █

```

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

```

CPU[          0.0%]   Tasks: 99, 208 thr; 1 running
Mem[|||||||374M/972M] Load average: 0.11 0.21 0.13
Swp[|||||||276M/448M] Uptime: 00:08:23

```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1373	ml-engine	20	0	3339M	173M	34064	S	0.7	17.8	0:05.92	/usr/bin/gnome
4621	ml-engine	20	0	19260	3760	3060	R	0.7	0.4	0:00.05	htop
1215	ml-engine	20	0	244M	24184	10764	S	0.0	2.4	0:01.14	/usr/lib/xorg/
1	root	20	0	163M	8204	5936	S	0.0	0.8	0:01.03	/sbin/init spl
218	root	19	-1	68244	18268	17196	S	0.0	1.8	0:00.49	/lib/systemd/s
258	root	20	0	24028	3016	2552	S	0.0	0.3	0:00.14	/lib/systemd/s
503	systemd-r	20	0	23872	6056	5060	S	0.0	0.6	0:00.10	/lib/systemd/s
521	systemd-t	20	0	90228	2600	2380	S	0.0	0.3	0:00.00	/lib/systemd/s
504	systemd-t	20	0	90228	2600	2380	S	0.0	0.3	0:00.06	/lib/systemd/s
540	root	20	0	244M	7112	6496	S	0.0	0.7	0:00.03	/usr/lib/accou
610	root	20	0	244M	7112	6496	S	0.0	0.7	0:00.01	/usr/lib/accou
536	root	20	0	244M	7112	6496	S	0.0	0.7	0:00.08	/usr/lib/accou
537	root	20	0	2548	676	644	S	0.0	0.1	0:00.01	/usr/sbin/acpi
541	avahi	20	0	8540	2340	2156	S	0.0	0.2	0:00.08	avahi-daemon:
543	root	20	0	18052	1964	1856	S	0.0	0.2	0:00.00	/usr/sbin/cron
546	root	20	0	37208	4520	3912	S	0.0	0.5	0:00.01	/usr/sbin/cups
547	messagebu	20	0	9868	5416	3232	S	0.0	0.5	0:00.89	/usr/bin/dbus-
590	root	20	0	411M	11864	10280	S	0.0	1.2	0:00.01	/usr/sbin/Netw
614	root	20	0	411M	11864	10280	S	0.0	1.2	0:00.16	/usr/sbin/Netw
548	root	20	0	411M	11864	10280	S	0.0	1.2	0:00.46	/usr/sbin/Netw
557	root	20	0	47964	7464	5800	S	0.0	0.8	0:00.13	/usr/bin/pytho
566	root	20	0	247M	10740	7400	S	0.0	1.1	0:00.00	/usr/lib/police

```

1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 SortBy F7 Nice - F8 Nice + F9 Kill F10 Quit

```

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 находиться 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

```

--
- hosts: all
  tasks:
    - name: Install nginx
      apt: name=nginx update_cache=yes
      become: yes

    - name: Start Nginx
      service: name=nginx state=started
      become: yes

```


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

```
mlops-engineer@mlopsengineer-VirtualBox:~$ ansible-playbook ./ansible/script2.y
ml -K
BECOME password:

PLAY [all] *****
*

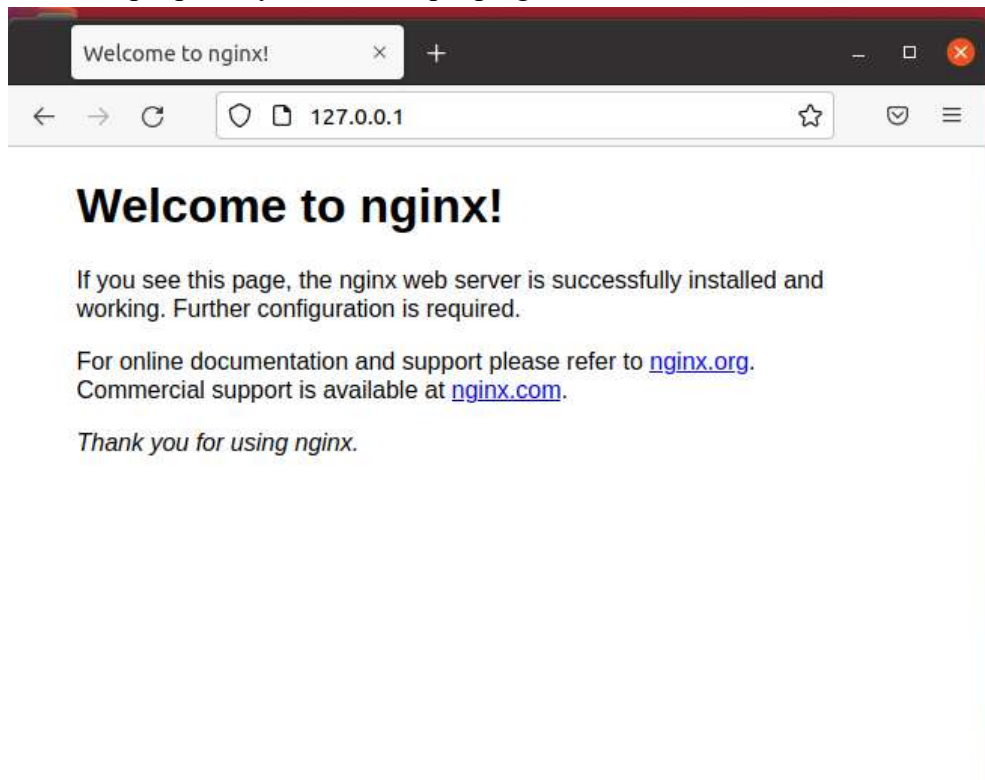
TASK [Gathering Facts] *****
*
ok: [192.168.147.3]
ok: [192.168.147.4]

TASK [Install nginx] *****
*
ok: [192.168.147.3]
ok: [192.168.147.4]

TASK [Start Nginx] *****
*
changed: [192.168.147.3]
changed: [192.168.147.4]

PLAY RECAP *****
*
192.168.147.3 : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
192.168.147.4 : ok=3    changed=1    unreachable=0    failed=0
```

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.	<pre> -- - hosts: all become: yes tasks: - name: Reboot command: /sbin/shutdown -r now </pre>
copy	Модуль для копирования файлов	<pre> -- - hosts: all become: yes tasks: - name: Copy file copy: src=data.txt dest=/home/ml owner=ml </pre>
raw	Используется тогда, когда другие модули использовать не получается. Обеспечивает запуск команд на удаленных управляемых серверах по ssh. Может работать без python3.	<pre> -- - hosts: all become: yes tasks: - name: Write data to file raw: ls -la > a.txt </pre>

Также в состав ansible 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. Без каких технологий не будет работать ansible? (0.25)
 - a. java
 - b. python3**
 - c. ssh
 - d. sql
2. Как называется файл, содержащий перечень команд, используемый ansible для настройки удаленных серверов? (0.25)
 - a. инструкция
 - b. плейбук**
 - c. commands
 - d. actions

3. Какая команда опрашивает все удаленные сервера? (0.25)
- a. `ansible echo all`
 - b. `ansible get reply`
 - c. `ansible all -m ping`**
 - d. `ansible all -m ask servers`
4. фыва? (0.25)
- a. фыва
 - b. фыва
 - c. фыва
 - d. фыва

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.

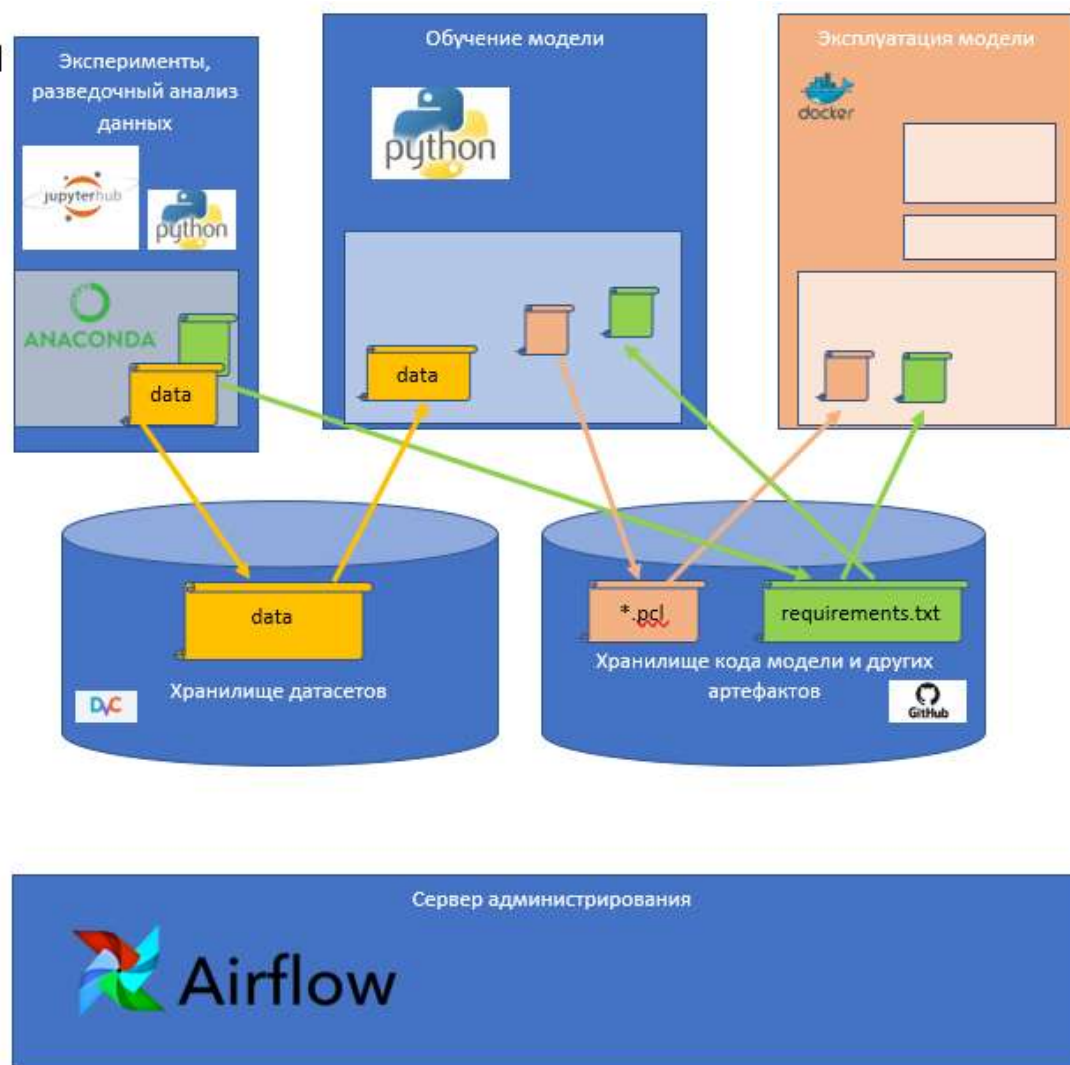
Содержание:

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 устанавливаются 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	python3, conda В виртуальной среде conda

	большой вычислительной нагрузкой	устанавливаются 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).
2. 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).
3. 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	

4. 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).
5. 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
 - a. получение и сохранение данных о пассажирах “Титаника” из задачи “Titanic Disaster”, создание файловой структуры для хранения данных, создание dvc репозитория (см. Модуль 4),
 - b. создание нескольких моделей классификации (линейная регрессия, деревья решений, случайный лес, XGBoost),
 - c. 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),
 - d. выполнение обучения моделей с подбором гиперпараметров, названия и сетка значений гиперпараметров берется из json файла,
 - e. сохранение результатов обучения моделей, метрик, весов,
 - f. сохранение кода лучшей модели и данных виртуального окружения в github,
 - g. сохранение использованного датасета в dvc хранилище (см. Модуль 4).
2. улучшение качества работы модели, обучение на сервере ML_server
 - a. загрузка данных п.1.f в новое окружение на сервере ML_server,
 - b. изменение датасета, добавление новых признаков (например, добавить новый признак “Title” (титул), который берется из Name),
 - c. выполнение обучения на новом датасете,
 - d. сохранение результатов обучения моделей, метрик, весов,
 - e. сохранение кода модели и данных виртуального окружения в github,
 - f. сохранение использованного датасета в dvc хранилище (см. Модуль 4).
3. вывод модели в промышленную эксплуатацию на сервер PROD_server
 - a. выполнение скриптов для организации производственного окружения
 - b. непрерывное развертывание всего решения
 - c. загрузка модели
 - d. эксплуатация модели на тестовых данных

е. 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 упрощения 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: <code>python3</code> , <code>ssh</code> Установить: <code>jupyterhub</code> , <code>conda</code> , <code>dvc</code> , <code>git</code>
PROD	Предустановлено: <code>python3</code> , <code>ssh</code> Установить: <code>docker</code>

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 важная 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 важная 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 выполнена 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/>

2. ansible

<https://www.ansible.com/>

3. django

<https://www.djangoproject.com/>

4. 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 бэкэнд 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 с использованием virtualenv, venv или аналогичных инструментов. При этом создается изолированная виртуальная среда

```
python3 -m venv venv
```

```
ubuntu@ip-172-31-38-46:~$ python3 --version
Python 3.8.5
ubuntu@ip-172-31-38-46:~$ █
```

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

```

ubuntu@ip-172-31-38-46:~$ mkdir django
ubuntu@ip-172-31-38-46:~$ cd django/
ubuntu@ip-172-31-38-46:~/django$ ls -la
total 8
drwxrwxr-x  2 ubuntu ubuntu 4096 Apr 16 03:42 .
drwx----- 21 ubuntu ubuntu 4096 Apr 16 03:42 ..
ubuntu@ip-172-31-38-46:~/django$ python3 -m venv venv
ubuntu@ip-172-31-38-46:~/django$ ls -la
total 12
drwxrwxr-x  3 ubuntu ubuntu 4096 Apr 16 03:42 .
drwx----- 21 ubuntu ubuntu 4096 Apr 16 03:42 ..
drwxrwxr-x  6 ubuntu ubuntu 4096 Apr 16 03:42 venv
ubuntu@ip-172-31-38-46:~/django$ ls -la venv
total 28
drwxrwxr-x  6 ubuntu ubuntu 4096 Apr 16 03:42 .
drwxrwxr-x  3 ubuntu ubuntu 4096 Apr 16 03:42 ..
drwxrwxr-x  2 ubuntu ubuntu 4096 Apr 16 03:42 bin
drwxrwxr-x  2 ubuntu ubuntu 4096 Apr 16 03:42 include
drwxrwxr-x  3 ubuntu ubuntu 4096 Apr 16 03:42 lib
lrwxrwxrwx  1 ubuntu ubuntu   3 Apr 16 03:42 lib64 -> lib
-rw-rw-r--  1 ubuntu ubuntu   69 Apr 16 03:42 pyvenv.cfg
drwxrwxr-x  3 ubuntu ubuntu 4096 Apr 16 03:42 share
ubuntu@ip-172-31-38-46:~/django$ █

```

Активировать виртуальное окружение можно с использованием команды

source ./имя папки окружения/bin/activate

```

ubuntu@ip-172-31-38-46:~/django$ source ./venv/bin/activate
(venv) ubuntu@ip-172-31-38-46:~/django$ █

```

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

```

(venv) ubuntu@ip-172-31-38-46:~/django$ deactivate
ubuntu@ip-172-31-38-46:~/django$ █

```

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

```

(venv) ubuntu@ip-172-31-38-46:~/django$ pip install django
Collecting django
  using cached django-3.2-py3-none-any.whl (7.9 MB)
Collecting pytz
  using cached pytz-2021.1-py2.py3-none-any.whl (510 kB)
Collecting asgiref<4,>=3.3.2
  using cached asgiref-3.3.4-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Installing collected packages: pytz, asgiref, sqlparse, django
Successfully installed asgiref-3.3.4 django-3.2 pytz-2021.1 sqlparse-0.4.1
(venv) ubuntu@ip-172-31-38-46:~/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.

```

(venv) ubuntu@ip-172-31-38-46:~/django$ django-admin startproject myproject
(venv) ubuntu@ip-172-31-38-46:~/django$ ls -la
total 16
drwxrwxr-x  4 ubuntu ubuntu 4096 Apr 16 03:48 .
drwx----- 21 ubuntu ubuntu 4096 Apr 16 03:42 ..
drwxrwxr-x  3 ubuntu ubuntu 4096 Apr 16 03:48 myproject
drwxrwxr-x  6 ubuntu ubuntu 4096 Apr 16 03:42 venv
(venv) ubuntu@ip-172-31-38-46:~/django$ ls -la myproject/
total 16
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 16 03:48 .
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 16 03:48 ..
-rwxrwxr-x 1 ubuntu ubuntu  665 Apr 16 03:48 manage.py
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 16 03:48 myproject
(venv) ubuntu@ip-172-31-38-46:~/django$ █

```

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.

```

(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

```

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.

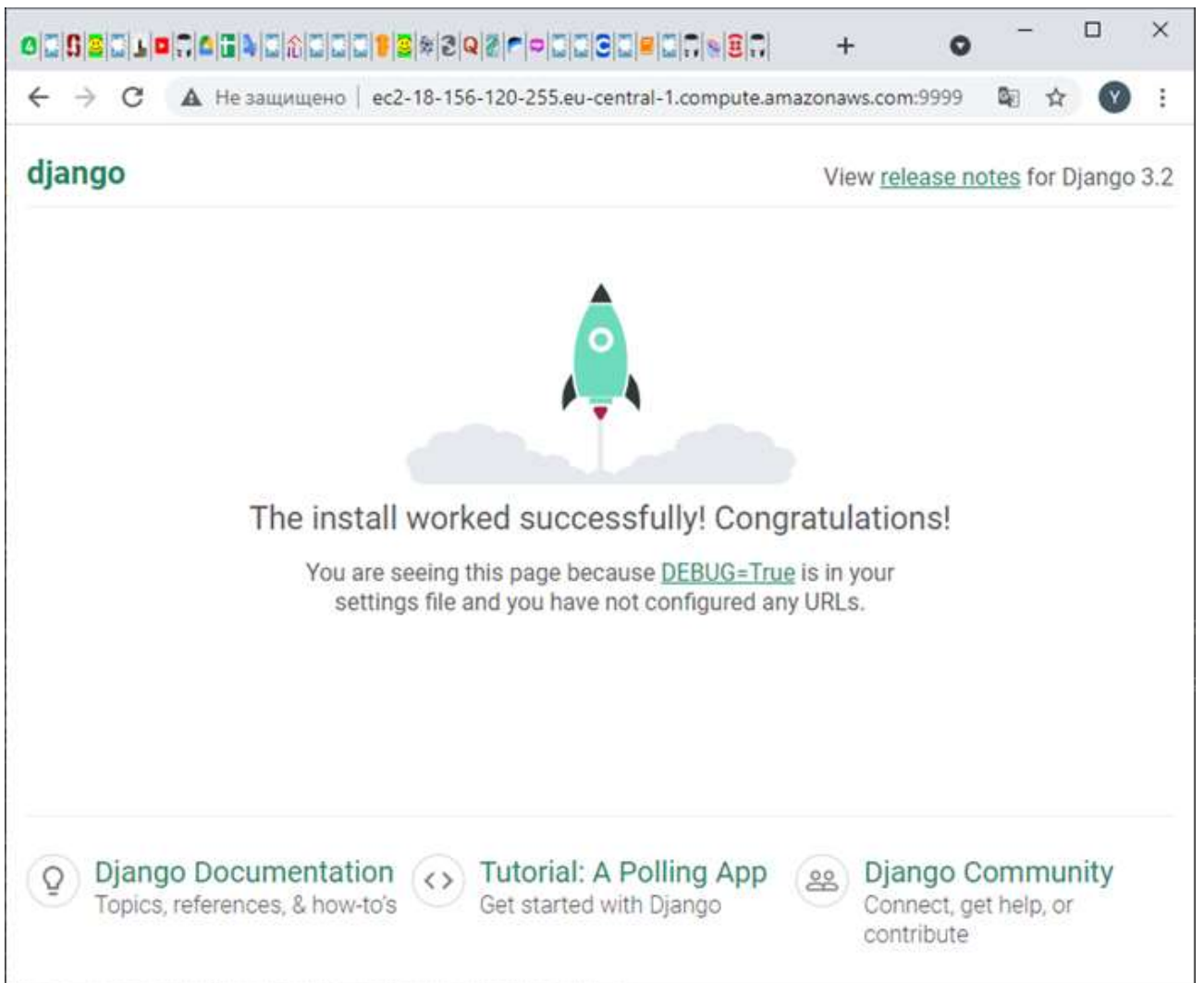
```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED_HOSTS = ['*']
```

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).

```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py runserver 0.0.0.0:9999  
watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
April 16, 2021 - 03:54:09  
Django version 3.2, using settings 'myproject.settings'  
Starting development server at http://0.0.0.0:9999/  
quit the server with CONTROL-C.
```

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



Этап 4. Создание суперпользователя

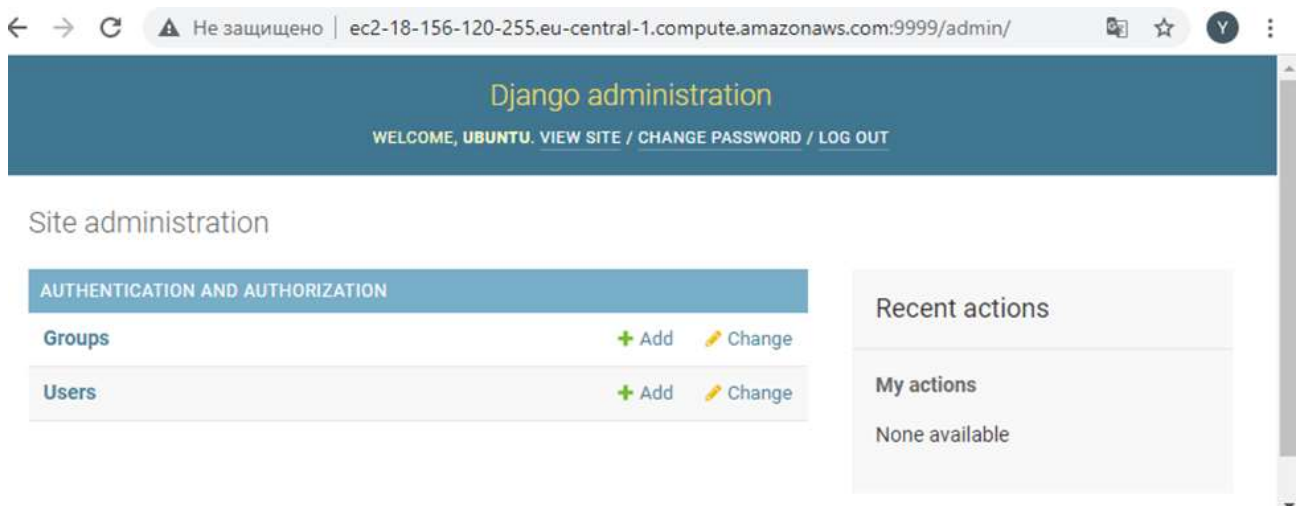
В Django «из коробки» доступны многие опции, например, панель администратора, через которую можно управлять многими настройками приложения. Для доступа необходимо создать суперпользователя (логин и пароль).

```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py createsuperuser
Username (leave blank to use 'ubuntu'): ubuntu
Email address:
Password:
Password (again):
superuser created successfully.
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ █
```

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

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Пользовательское приложение необходимо создать отдельно с помощью команды

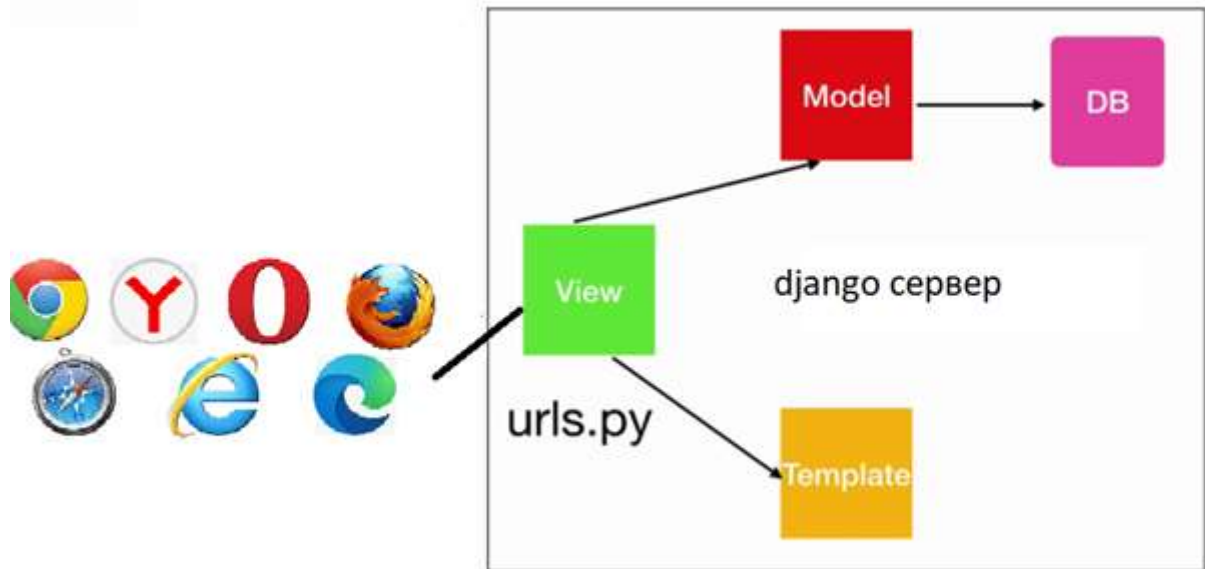
`./manage.py startapp "имя приложения"`

```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py startapp myapp
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ls -la
total 148
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 16 04:15 .
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 16 03:48 ..
-rw-r--r-- 1 ubuntu ubuntu 131072 Apr 16 04:11 db.sqlite3
-rwxrwxr-x 1 ubuntu ubuntu 665 Apr 16 03:48 manage.py
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 16 04:15 myapp
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 16 04:14 myproject
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ls -la myapp/
total 32
drwxrwxr-x 3 ubuntu ubuntu 4096 Apr 16 04:15 .
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 16 04:15 ..
-rw-rw-r-- 1 ubuntu ubuntu 0 Apr 16 04:15 __init__.py
-rw-rw-r-- 1 ubuntu ubuntu 63 Apr 16 04:15 admin.py
-rw-rw-r-- 1 ubuntu ubuntu 142 Apr 16 04:15 apps.py
drwxrwxr-x 2 ubuntu ubuntu 4096 Apr 16 04:15 migrations
-rw-rw-r-- 1 ubuntu ubuntu 57 Apr 16 04:15 models.py
-rw-rw-r-- 1 ubuntu ubuntu 60 Apr 16 04:15 tests.py
-rw-rw-r-- 1 ubuntu ubuntu 63 Apr 16 04:15 views.py
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$
```

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`.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
]
```

Этап 6. Создание модели.

В файле `models.py` описывается модель

```
from django.db import models
# Create your models here.

class Person(models.Model):
    number = models.IntegerField
    name = models.CharField(max_length=100)
```

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

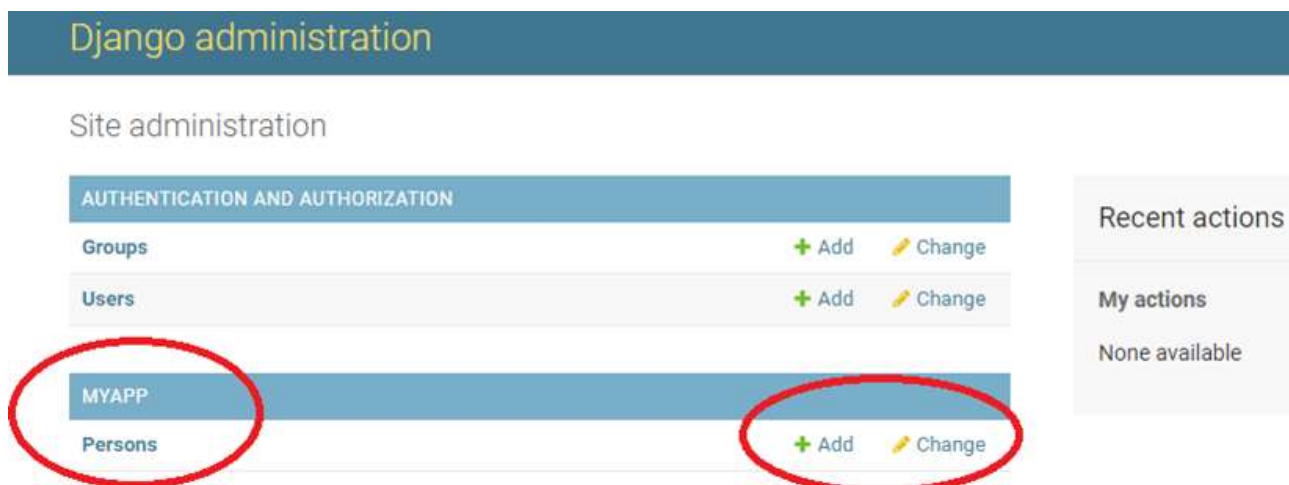
```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py makemigrations
Migrations for 'myapp':
  myapp/migrations/0001_initial.py
  - Create model Person
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ █
```

```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./man migrate
-bash: ./man: No such file or directory
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp, sessions
Running migrations:
  Applying myapp.0001_initial... OK
```

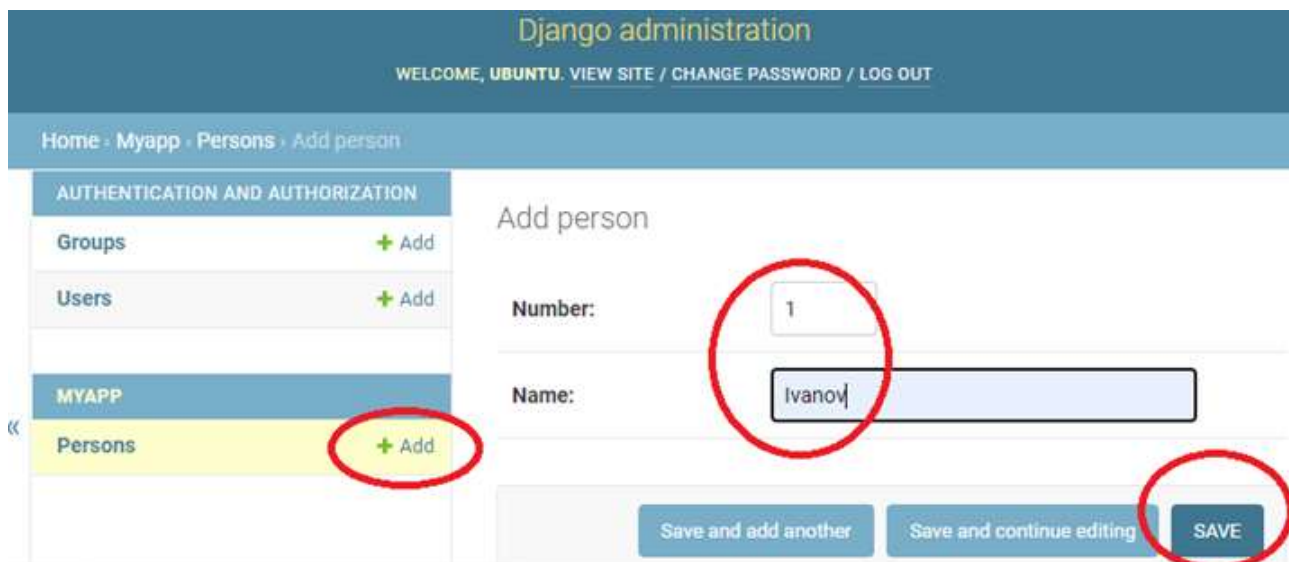
Данную модель необходимо зарегистрировать в файле admin.py для корректного отображения в панели администратора

```
from django.contrib import admin
# Register your models here.
from .models import Person
admin.site.register(Person)
```

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 туарр создается таблица туарр_person. Можно это проверить, выполнив SQL запрос.

```
(venv) ubuntu@ip-172-31-38-46:~/django/myproject$ python
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>> conn = sqlite3.connect("db.sqlite3")
>>> c = conn.cursor()
>>> data = c.execute("SELECT * FROM myapp_person").fetchall()
>>> data
[(1, 'Ivanov', 1)]
>>>
```

Этап 8. Создание контроллера (view).

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

```
from django.shortcuts import render
from django.http import HttpResponse

# create your views here.

def hello_from_viewer(request):
    return HttpResponse("hello from Django viewer!")
```

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

```
from django.contrib import admin
from django.urls import path
from myapp.views import hello_from_viewer

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hello/', hello_from_viewer),
]
```

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



← → ↻ Не защищено | ec2-18-156-120-255.eu-central-1.compute.amazonaws.com:9999/hello/
Hello from Django viewer!

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

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

```
<html>
    <h1>hello from HTML!</h1>
</html>
```

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

```
TEMPLATES_DIRS = BASE_DIR / 'templates/'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATES_DIRS],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Необходимо создать контроллер (viewer), отображающее данный темплейт

```
def hello_from_html(request):  
    return render(request, "index.html")
```

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

```
from django.contrib import admin  
from django.urls import path  
from myapp.views import hello_from_viewer, hello_from_html  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('viewer/', hello_from_viewer),  
    path('html/', hello_from_html),  
]
```

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

| ec2-18-156-120-255.eu-central-1.compute.amazonaws.com:8000/html/

Hello from HTML!

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

- `DEBUG` – булево 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 = sqlite3.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)
 - a. control.py
 - b. manage.py**
 - c. django.py
 - d. command.py
2. Что такое ORM? (0.25)
 - a. Object Restricted Mode
 - b. Object Relation Model**
 - c. Object Recognition Model
 - d. Object Resolution Mode
3. Что такое миграции в Django? (0.25)
 - a. копирование кода из одного проекта в другой
 - b. перенос информации из python кода django в соответствующие структуры базы данных**
 - c. замена программиста в проекте
 - d. изменение аппаратного обеспечения, смена провайдера облачных услуг для проекта.
4. Какие файлы Django проекта необходимы для описания основных функций? (0.25)
 - a. models.py**
 - b. main.py
 - c. views.py**
 - d. 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 сосредоточиться 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 (DRF). 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

```
jche@linux-vm-jche:~/apps$ mkdir project
jche@linux-vm-jche:~/apps$ cd project
jche@linux-vm-jche:~/apps/project$ python3 -m venv venv
jche@linux-vm-jche:~/apps/project$ ls -la
total 12
drwxrwxr-x  3 jche jche 4096 Jun 10 04:19 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
drwxrwxr-x  6 jche jche 4096 Jun 10 04:19 venv
jche@linux-vm-jche:~/apps/project$ source venv/bin/activate
(venv) jche@linux-vm-jche:~/apps/project$
```

После выполненных действий в текущей рабочей директории создается директория 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.

```
(venv) jche@linux-vm-jche:~/apps/project$ ls venv/lib/python3.6/site-packages/
asgiref                               pkg_resources-0.0.0.dist-info
asgiref-3.2.7.dist-info                __pycache__
django                                 pytz
Django-3.0.7.dist-info                 pytz-2020.1.dist-info
djangorestframework-3.11.0.dist-info  rest_framework
easy_install.py                       setuptools
pip                                     setuptools-39.0.1.dist-info
pip-9.0.1.dist-info                   sqlparse
pkg_resources                          sqlparse-0.3.1.dist-info
```

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

django-admin startproject project ../project

./manage.py startapp app

```
(venv) jche@linux-vm-jche:~/apps/project$ django-admin startproject project ../project
(venv) jche@linux-vm-jche:~/apps/project$ ls -la
total 20
drwxrwxr-x 4 jche jche 4096 Jun 10 04:29 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
-rwxrwxr-x 1 jche jche 627 Jun 10 04:29 manage.py
drwxrwxr-x 2 jche jche 4096 Jun 10 04:29 project
drwxrwxr-x 6 jche jche 4096 Jun 10 04:19 venv
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py startapp app
(venv) jche@linux-vm-jche:~/apps/project$ ls -la
total 24
drwxrwxr-x 5 jche jche 4096 Jun 10 04:30 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
drwxrwxr-x 3 jche jche 4096 Jun 10 04:30 app
-rwxrwxr-x 1 jche jche 627 Jun 10 04:29 manage.py
drwxrwxr-x 3 jche jche 4096 Jun 10 04:30 project
drwxrwxr-x 6 jche jche 4096 Jun 10 04:19 venv
(venv) jche@linux-vm-jche:~/apps/project$ █
```

./manage.py migrate

```
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
(venv) jche@linux-vm-jche:~/apps/project$ █
```

При выполнении этой команды создаются необходимые базы данных.

Редактирование файла `project/settings.py`. Необходимо отредактировать переменные: `ALLOWED_HOSTS`, `SECRET_KEY`, `DEBUG`.

```
SECRET_KEY = 'h=(n$vlb^@ls@b9ncspdpo*j2x272cz^jb@*wx+b!#*+~w*d@9'  
DEBUG = True  
ALLOWED_HOSTS = ['*']
```

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.

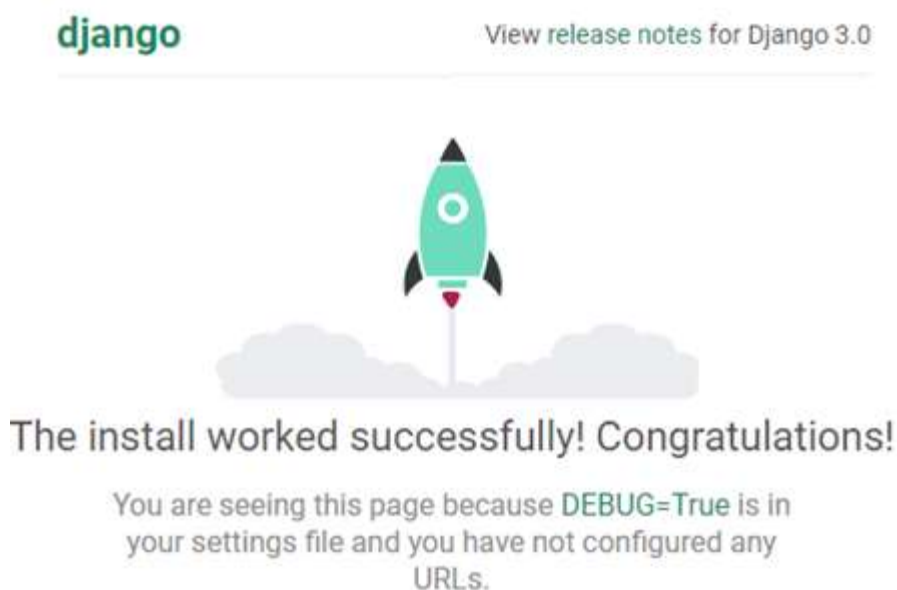
`./manage.py runserver 0.0.0.0:8000`

```
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py runserver 0.0.0.0:8000  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
June 10, 2020 - 04:34:14  
Django version 3.0.7, using settings 'project.settings'  
Starting development server at http://0.0.0.0:8000/  
Quit the server with CONTROL-C.
```

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`


```
(venv) jche@linux-vm-jche:~/apps/project$ vi project/settings.py
(venv) jche@linux-vm-jche:~/apps/project$
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py createsuperuser
Username (leave blank to use 'jche'): yuchernyshov
Email address:
Password:
```

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`

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
    'rest_framework',
]

```

4. Добавление django модели Person в приложение app.

Описать класс модели в файле app/models.py

```

from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=100)
    number = models.IntegerField()

    def __str__(self):
        return("Object Person, name: {}, number: {}".format(self.name, self.number))

```

Зарегистрировать модель в admin.py

```

from django.contrib import admin

from .models import Person

@admin.register(Person)
class PersonAdmin(admin.ModelAdmin):
    list_display = ['name', 'number']

```

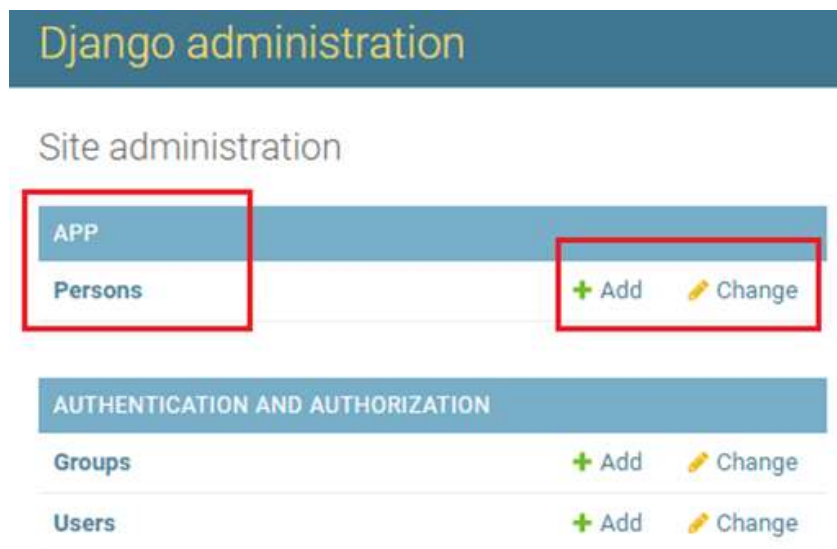
Сделать необходимые изменения в базе данных.

```

(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py makemigrations
Migrations for 'app':
  app/migrations/0001_initial.py
    - Create model Person
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, app, auth, contenttypes, sessions
Running migrations:
  Applying app.0001_initial... OK

```

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 сериализации 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.py

```
from django.shortcuts import render

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.generics import get_object_or_404

from .models import Person
from .serializers import PersonSerializer

class person_api_view(APIView):

    def get(self, request):
        persons = Person.objects.all()
        serialized_persons = PersonSerializer(persons, many=True)
        return Response({"persons": serialized_persons.data})

    def post(self, request):
        person = request.data.get("person")
        serialized_person = PersonSerializer(data=person)
        if serialized_person.is_valid(raise_exception=True):
            saved_person = serialized_person.save()
            return Response({"success": "Person {} saved".format(saved_person.name)})

    def put(self, request, number):
        data = request.data.get("person")
        person = get_object_or_404(Person.objects.all(), number=number)
        serialized_person = PersonSerializer(instance=person, data=data, partial=True)
        if serialized_person.is_valid(raise_exception=True):
            updated_person = serialized_person.save()
            return Response({"success": "Person {} updated".format(updated_person)})

    def delete(self, request, number):
        person = get_object_or_404(Person.objects.all(), number=number)
        name = person.name
        number = person.number
        person.delete()
        return Response({"success": "Person {} deleted".format(name, number)})
```

Описание сериализатора PersonSerializer в файле app/serializers.py

```

from rest_framework import serializers

from .models import Person

class PersonSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
    number = serializers.IntegerField()

    def create(self, validated_data):
        return Person.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.name = validated_data.get("name", instance.name)
        instance.number = validated_data.get("number", instance.number)
        instance.save()
        return instance

```

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`)

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('view/', include('app.urls')),
]

```

Установить правила в `app/urls.py`

```

from django.urls import path, include

from .views import person_api_view

urlpatterns = [
    path('api/', person_api_view.as_view()),
    path('api/<int:number>', person_api_view.as_view()),
]

```

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

Request

Method: GET Request URL: http://130.193.35.185:8000/view/api/ **SEND**

Parameters

Headers

Header name	Header value
content-type	application/json

ADD HEADER

Headers are valid Headers size: 30 bytes

200 OK 280.69 ms **DETAILS**

```
{
  "persons": [Array(1)]
  - 0: {
    "name": "Иван",
    "number": 1
  }
}
```

Запрос POST

Request

Method: POST Request URL: http://130.193.35.185:8000/view/api/ **SEND**

Parameters

Body

Body content type: application/json Editor view: Raw input

FORMAT JSON MINIFY JSON

```
{
  "person": {
    "name": "Иван",
    "number": 2
  }
}
```

200 OK 281.91 ms **DETAILS**

```
{
  "success": "Person Иван saved"
}
```

Запрос PUT

Request ⓘ ⋮

Method: PUT Request URL: `http://130.193.35.185:8000/view/api/1` **SEND** ⋮

Parameters ^

Headers Body Variables

Body content type: application/json Editor view: Raw input

FORMAT JSON MINIFY JSON

```
{
  "person": {
    "name": "Игорь",
    "number": 1
  }
}
```

200 OK 295.39 ms DETAILS ▾

📄 ⚙️ ⏪ ⏩

```
{
  "success": "Person Object Person, name: Игорь, number: 1 updated"
}
```

Запрос DELETE

Request ⓘ ⋮

Method: DELETE Request URL: `http://130.193.35.185:8000/view/api/2` **SEND** ⋮

Parameters ^

Headers Body Variables

📄 ⏪ ⏩ Toggle source mode + Insert headers set

Header name	Header value	✕	✎
content-type	application/json		

ADD HEADER

🟢 Headers are valid Headers size: 30 bytes

200 OK 284.27 ms DETAILS ▾

📄 ⚙️ ⏪ ⏩

```
{
  "success": "Person Мария deleted"
}
```

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 данные.

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-потoki, 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/
lighttpd	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 веб-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 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`. 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 = sqlite3.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 закомментированные настройки (то есть данные, перед которыми стоит знак “#”, что означает комментарий в python) и их описания на английском языке. Можно задавать

при запуске jupyterhub полное имя конфигурационного файла, но лучше хранить файл конфигурации в /etc/jupyterhub/.

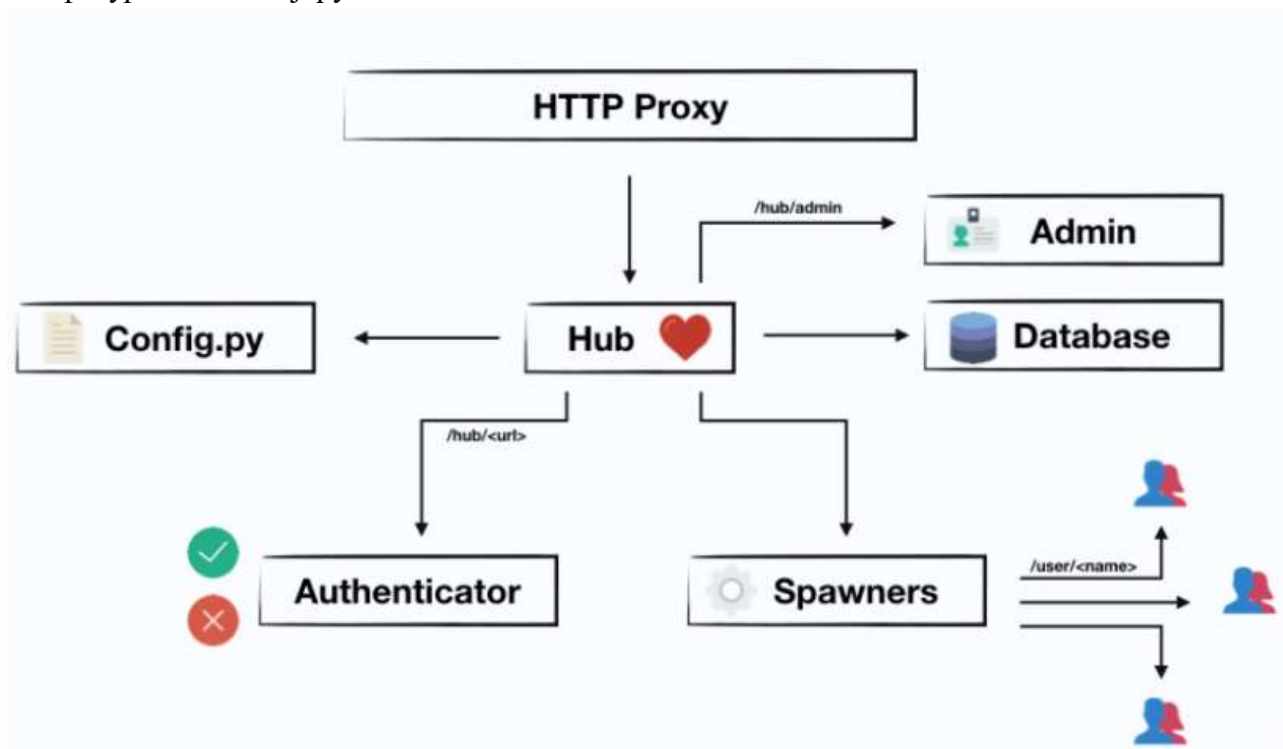


Рисунок “Схема работы JupyterHub”

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

sudo adduser --home “рабочий директориий 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
- 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».cert

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.

Модуль 8. Эксплуатация моделей машинного обучения

Образовательный результат:

Термин “эксплуатация” в технологических проектах означает “использование”, “применение”. Процесс эксплуатации любой программной системы является не только закономерным итогом всего проекта разработки программного обеспечения, но и его наиболее практически полезной частью. Программное обеспечение разрабатывается для того, чтобы кто-то его использовал. Однако после вывода решения в эксплуатацию работа в проекте не завершается, поскольку процесс эксплуатации выявляет ошибки в системе, выдвигает новые задачи перед разработчиками, дает новые идеи. И если процесс эксплуатации выстроен неэффективно, то будут нарастать проблемы с качеством, увеличиваться технический долг по проекту, ресурсы проекта будут расходоваться неоптимально. В итоге неэффективная эксплуатация может привести к неуспешному завершению даже самый качественный проект. Для повышения эффективности эксплуатации создают специальные инструменты, решающие основные эксплуатационные задачи: тестирование, мониторинг, управление, анализ, эксперименты, обнаружение инцидентов. Знание и эффективное владение инструментами для эксплуатации, умение их правильно и быстро установить и настроить для работы, являются востребованными навыками специалистов DevOps/MLOps.

После изучения этого модуля Вы научитесь пользоваться инструментами для эксплуатации программных систем, которые применимы и для проектов машинного обучения.

В этом модуле:

Успешная эксплуатация программных систем зависит от навыков специалистов команды, обеспечивающей работу программной системы в производственной среде, а также от используемого инструментария, помогающего решать эксплуатационные задачи.

Организации и управлению команд специалистов, занимающихся эксплуатацией программных систем, посвящено большое количество специализированной литературы. Есть специальные системы, помогающие организовать этот процесс эффективно, например “Help Desk”, “Service Desk”, CRM и другие подобные. Это важная часть проекта, требующая отдельного изучения и важная для практического применения, но в данном курсе эта тема не рассматривается, поскольку основная цель курса это познакомить вас с задачами и инструментами DevOps/MLOps. Поэтому далее в этом модуле мы рассмотрим инструменты для:

- мониторинга ресурсов
 - оперативная память,
 - процессорная мощность,
 - жесткий диск,
 - сетевые интерфейсы
- мониторинга процессов
 - проверка качества данных,

- обработка данных,
- подбор гиперпараметров,
- обучение модели,
- инференс (использование в производственной среде) модели
- контроля системы
 - управление пользователями и группами,
 - управление процессами,
 - управление сетевыми интерфейсами,
- организация взаимодействия отдельных элементов
 - передача файлов,
 - удаленное подключение,
 - настройка защищенного доступа.

Важно понимать, что для задач эксплуатации актуально все, что мы уже обсуждали в предыдущих модулях: автоматизация, унификация, управление версиями используемых объектов, всесторонний контроль и аналитика. Владение инструментами для решения этих задач позволяет сделать процесс эксплуатации более управляемым и эффективным, а работу модели машинного обучения более качественной и надежной. Также это позволяет своевременно выявлять проблемы в работе модели машинного обучения и системы в целом, и принимать меры для устранения неисправностей.

В этом модуле описаны различные инструменты для решения описанных выше задач. Темы, изучаемые в модуле:

1. Средства для мониторинга и управления ресурсами в linux.
2. Утилита Tensorboard для мониторинга.
3. Визуализация и контроль с использованием Grafana.
4. Практический пример использования Grafana для мониторинга проекта машинного обучения.

Модуль 8. Юнит 1. Утилиты мониторинга и управления ресурсами в linux.

Введение:

Linux является очень популярной операционной системой, в том числе для разработки и эксплуатации моделей машинного обучения. Это связано с тем, что большинство библиотек и утилит, используемых в проектах машинного обучения, эффективнее работают в linux системах, чем в других. С момента своего создания linux был бесплатным программным обеспечением с открытым программным кодом, что предопределило его огромный успех и массовое использование в различных проектах. Любой специалист в любой точке мира мог без технических проблем и финансовых вложений начать использовать linux в своих проектах. В том числе это сделало возможным привлечение большого сообщества разработчиков и инженеров для разработки и эксплуатации программного обеспечения для linux. Все это в полной мере относится и к машинному обучению. Появление большого количества библиотек и фреймворков машинного обучения с открытым программным кодом (jupyter, numpy, sklearn, Tensorflow, PyTorch и многих других) стало возможным с использованием идеологии open-source. Оптимизация выполнения вычислительных операций, благодаря которой библиотеки машинного обучения стали такими эффективными, оказалась возможна благодаря открытости операционной системы linux и использованию “коллективного разума” большого сообщества разработчиков, которые занимались разработкой и отладкой кода, тестированием и применением разработок. **Это позволило создать высокопроизводительные и надежные инструменты машинного обучения, которые активно используются специалистами во всем мире, в том числе и в высоконагруженных системах и в системах с повышенными требованиями к надежности, скорости работы.**

Другим существенным преимуществом использования linux в проектах машинного обучения является простота установки и настройки инструментария. Большинство программ можно установить с использованием всего нескольких команд. Почти все инструменты готовы к использованию сразу после установки с настройками по умолчанию. Простота процедуры начала работы с инструментом и удобство его использования, наличие широкой информационной базы примеров и сообщества специалистов, помогающих в поиске ответов на возникающие вопросы, предопределяют популярность любого инструмента. **Все это привело к тому, что операционные системы linux и программное обеспечение на их основе стали стандартом “де-факто” в проектах машинного обучения.**

В связи с такой популярностью linux от специалистов MLOps требуется хорошее знание этой операционной системы, умение ее администрировать. Без этих навыков невозможно установить, оптимально настроить и использовать большинство инструментов MLOps, которые вы изучили в предыдущих модулях. Вы изучали linux в курсе «Операционная система linux» (и в рамках мастер-класса во втором семестре), при необходимости рекомендуется вспомнить сведения, полученные в этом курсе. Не будет лишним, если вы прочитаете дополнительно одну из многих книг про администрирование linux и подкрепите полученные сведения практической работой в командной строке linux системы, поскольку

в основном для работы используется командная строка. Некоторые из книг вы можете найти [по ссылкам в конце модуля](#).

Основное содержание юнита составляют описания утилит для мониторинга и управления, входящие в состав linux систем. Вы узнаете как получить информацию об используемых системных ресурсах: оперативной памяти, жестком диске, процессорах, сетевых интерфейсах. Также полезными являются сведения о пользователях, группах и сервисах. И, наконец, в юните рассказывается о различных типовых инструментах, используемых системными администраторами и инженерами DevOps/MLOps в повседневной работе: текстовых редакторах, управлении сервисами, запуске по расписанию, удаленной работе.

Содержание юнита:

Большинство служебных утилит, которые используются для администрирования операционной системы linux и решения практических задач, выглядят одинаково для различных версий linux (RedHat, Fedora, Ubuntu, Kali, CentOS и др.). Тем не менее иногда имеются различия в синтаксисе, например, могут различаться штатные установщики пакетов программного обеспечения в версиях linux:

- rpm в RedHat,
- apt, snap или dpkg в Ubuntu и Kali,
- apk в Alpine,
- dnf в Fedora
- yum в CentOS.

Если необходимый вам инструмент не входит в состав операционной системы, то он как правило легко устанавливается. В большинстве случаев достаточно выполнить единственную команду с использованием штатного установщика, либо точно следовать инструкции по установке на официальном сайте разработчика утилиты.

Например, утилита для мониторинга работы сетевого оборудования netstat по умолчанию не входит в набор утилит операционной системы, поэтому необходимо ее установить отдельно.

```
data-engineer@dataengineer-VirtualBox:~$ netstat
bash: /usr/bin/netstat: No such file or directory
data-engineer@dataengineer-VirtualBox:~$ sudo apt install netstat
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package netstat
data-engineer@dataengineer-VirtualBox:~$
```

Команда установки неуспешна, так как netstat не устанавливается отдельно, а входит в пакет программ net-tools, именно его и надо установить

```

data-engineer@dataengineer-VirtualBox:~$ sudo apt install net-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
 net-tools
0 upgraded, 1 newly installed, 0 to remove and 55 not upgraded.
Need to get 196 kB of archives.
After this operation, 864 kB of additional disk space will be used.
Get:1 http://ru.archive.ubuntu.com/ubuntu focal/main amd64 net-tools amd64 1.60
+git20180626.aebd88e-1ubuntu1 [196 kB]
Fetched 196 kB in 0s (499 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 203965 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20180626.aebd88e-1ubuntu1_amd64.deb .
..
Unpacking net-tools (1.60+git20180626.aebd88e-1ubuntu1) ...
Setting up net-tools (1.60+git20180626.aebd88e-1ubuntu1) ...
Processing triggers for man-db (2.9.1-1) ...
data-engineer@dataengineer-VirtualBox:~$ whereis netstat
netstat: /usr/bin/netstat /usr/share/man/man8/netstat.8.gz
data-engineer@dataengineer-VirtualBox:~$ █

```

С помощью утилиты `whereis` мы проверили, что установка успешно осуществилась и утилита `netstat` доступна по адресу `/usr/bin/netstat`. Утилит в `linux` очень много, для каждой задачи уже создана специальная утилита. Удобство их использования заключается еще и в том, что в `linux` для инструментов существует подробная документация, это было заложено в идеологии `linux` и поддерживается с развитием новых инструментов. Для доступа к технологической документации можно использовать ключ `-h` для любой команды или воспользоваться утилитой `man`. Например, вот так можно получить справку по команде `yes`

```

data-engineer@dataengineer-VirtualBox:~$ man yes

```

в результате вы увидите документацию, описывающую применение команды `yes`.

```

YES(1)                                     User Commands                               YES(1)

NAME
  yes - output a string repeatedly until killed

SYNOPSIS
  yes [STRING]...
  yes OPTION

DESCRIPTION
  Repeatedly output a line with all specified STRING(s), or 'y'.

  --help display this help and exit

  --version
        output version information and exit

AUTHOR
  Written by David MacKenzie.

REPORTING BUGS
  GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
  Report yes translation bugs to <https://translationproject.org/team/>

COPYRIGHT
  Copyright © 2018 Free Software Foundation, Inc. License GPLv3+: GNU
  GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
  This is free software: you are free to change and redistribute it.
Manual page yes(1) line 1 (press h for help or q to quit)

```

Документация man содержит описания всех ключей и опций использования для команд linux, ей удобно пользоваться.

Давайте рассмотрим некоторые полезные утилиты, которые могут пригодиться вам при развертывании и эксплуатации сложных программных систем в операционной системе linux. Также рассмотрим некоторые примеры задач DevOps/MLOps, в которых применяются эти утилиты.

1. Утилиты для мониторинга ресурсов

В проектах машинного обучения активно используются системные аппаратные ресурсы: память на жестких дисках, оперативную память, вычислительную мощность процессоров. При отсутствии должного контроля или при ошибках в программном коде можно получить ситуацию, когда ресурсы заканчиваются. Обычно одними и теми же ресурсами пользуются многие участники команды и сбой одной программы может привести к невозможности работы всей команды. По этой причине необходимо уметь диагностировать задействованные ресурсы, находить опасные тенденции и причины их возникновения.

Давайте рассмотрим утилиты для оценки задействования памяти на жестких дисках.

df

```

data-engineer@dataengineer-VirtualBox:~$ df
Filesystem      1K-blocks      Used Available Use% Mounted on
udev            463932          0    463932   0% /dev
tmpfs           99484          1384    98100   2% /run
/dev/sda5       9736500 8591800    630396  94% /
tmpfs           497404          0    497404   0% /dev/shm
tmpfs           5120            4     5116   1% /run/lock
tmpfs           497404          0    497404   0% /sys/fs/cgroup
/dev/loop0      128            128         0 100% /snap/bare/5

```

При выполнении этой команды видим объем свободной и задействованной памяти.

du

```

data-engineer@dataengineer-VirtualBox:~$ du -Bk -a jupyterhub/
4K      jupyterhub/jupyterhub_cookie_secret
152K    jupyterhub/jupyterhub.sqlite
160K    jupyterhub/
data-engineer@dataengineer-VirtualBox:~$ █

```

Более продвинутой версией утилиты du является утилита **ncdu**, которую надо специально установить, поскольку она не входит в стандартный пакет программ. ncdu обладает простым графическим интерфейсом, в котором можно перемещаться по папкам с использованием клавиш курсора, быстро видеть информацию по размеру папок и даже удалять ненужные папки прямо из этого графического интерфейса.

Давайте рассмотрим практическую задачу. Пусть у нас есть папка /test, имеющая структуру для использования виртуального окружения venv следующего вида

```

data-engineer@dataengineer-VirtualBox:~$ tree test -L 2
test
├── requirements.txt
└── venv
    ├── bin
    ├── include
    ├── lib
    ├── lib64 -> lib
    ├── pyvenv.cfg
    └── share

6 directories, 2 files
data-engineer@dataengineer-VirtualBox:~$ █

```

Вызов ncdu для просмотра папки /test позволит оценить размер использованного пространства жесткого диска как для папок, так и для отдельных файлов

```

ncdu 1.14.1 ~ Use the arrow keys to navigate, press ? for help
--- /home/data-engineer/test -----
74,1 MiB [#####] /venv
4,0 KiB [ ] requirements.txt

```

При необходимости можно удобно перемещаться по папкам, детализируя информацию о задействовании пространства жесткого диска внутри папок. Например, вы можете зайти в папку `venv` и оценить размер ее подпапок.

```
ncdu 1.14.1 ~ Use the arrow keys to navigate, press ? for help
--- /home/data-engineer/test/venv -----
|
| 71,8 MiB [#####] /lib
|  2,2 MiB [          ] /share
| 60,0 KiB [          ] /bin
|e  4,0 KiB [          ] /include
|  4,0 KiB [          ] pyvenv.cfg
|@  0,0  B [          ] lib64
```

Одно из полезных практических применений этой утилиты: обнаружение разросшихся файлов логов. По умолчанию системы пишут информацию о своей работе в `log` файлы, обычное место хранения этой информации `/var/log`. При отсутствии контроля, несмотря на то что `log` файлы имеют текстовую структуру, размер этой папки может быстро увеличиваться и в итоге привести к нехватке рабочего пространства на жестких дисках. Продиагностировать ситуацию может команда

```
data-engineer@dataengineer-VirtualBox:~$ ncdu /var/log
data-engineer@dataengineer-VirtualBox:~$
```

в результате выполнения которой вы увидите информацию о размерах папок внутри `/var/log`

```
--- /var/log -----
| 216,1 MiB [#####] /journal
|  1,3 MiB [          ] syslog.1
|  1,3 MiB [          ] dpkg.log
| 788,0 KiB [          ] /installer
| 764,0 KiB [          ] syslog
| 572,0 KiB [          ] kern.log.1
| 472,0 KiB [          ] /apt
| 388,0 KiB [          ] kern.log
| 192,0 KiB [          ] jupyterhub.log
| 132,0 KiB [          ] /unattended-upgrades
| 128,0 KiB [          ] auth.log.1
| 128,0 KiB [          ] auth.log
| 104,0 KiB [          ] syslog.2.gz
| 104,0 KiB [          ] bootstrap.log
| 100,0 KiB [          ] syslog.5.gz
|  92,0 KiB [          ] wtmp
|  52,0 KiB [          ] syslog.4.gz
|  48,0 KiB [          ] boot.log.7
|  48,0 KiB [          ] syslog.7.gz
|  48,0 KiB [          ] dmesg.0
|  48,0 KiB [          ] dmesg
|  48,0 KiB [          ] /cups
|  44,0 KiB [          ] lastlog
```

Видим, что наибольшее пространство занимает `/var/log/journal`, можем переместиться в него и посмотреть более детально что за файлы занимают это пространство, при необходимости некоторые из них можно удалить.

```
ncdu 1.14.1 ~ Use the arrow keys to navigate, press ? for help
--- /var/log/journal/c9afc4df69354155908527a93a87ab87 -----
/..
24,0 MiB [#####] system@0005e116d4bfad8f-68f08156d7fbc3c7.journal~
24,0 MiB [#####] system@0005e0b41e0a71f0-9faef60fb9c115df.journal~
16,0 MiB [#####] user-1000.journal
16,0 MiB [#####] system@0005e0edd5747f2e-a37effc3853eb2c8.journal~
16,0 MiB [#####] system@0005e0700c05f419-cc1c1c16837be6aa.journal~
16,0 MiB [#####] system.journal
8,0 MiB [###] user-1002@0005e0b7d5d7e448-5fa9ee7167b2cae5.journal~
8,0 MiB [###] user-1002.journal
8,0 MiB [###] user-1000@0005e0715182adda-b74670a2763ae5fb.journal~
8,0 MiB [###] user-1000@0005e04c941ea2e9-402b516c1422291a.journal~
8,0 MiB [###] user-1000@0005e04a44d0480c-7f30323435ea933f.journal~
8,0 MiB [###] system@0005e0fccbb9398d-d674bf7bf83a709c.journal~
8,0 MiB [###] system@0005e0b7cfeca99c-d1d2303448b1d3de.journal~
8,0 MiB [###] system@0005e074f289c3f7-89a1f4c092836a95.journal~
8,0 MiB [###] system@0005e071507ffb5d-150c62782254fc61.journal~
8,0 MiB [###] system@0005e04c91b4b6d4-7cfe27f2e57e5f39.journal~
8,0 MiB [###] system@0005e04b7013bc19-7819d792d93fef57.journal~
8,0 MiB [###] system@0005e04a43729993-8ac72b4c86ce75da.journal~
8,0 MiB [###] user-1001.journal
```

Еще одной полезной утилитой, позволяющей получить информацию о задействовании пространства жесткого диска, является **free**, вот как например можно посмотреть с ее помощью свободную и задействованную память в мегабайтах

```
data-engineer@dataengineer-VirtualBox:~$ free -h --mega
              total        used        free      shared  buff/cache   available
Mem:           971M          384M          117M           4,0M          469M          437M
Swap:           448M          332M           116M
```

Другим важнейшим ресурсом является вычислительная мощность центрального процессора (CPU, Central Processing Unit) или графического процессора (GPU, Graphical Processing Unit). Получить информацию о CPU можно следующим образом

cat /proc/cpuinfo

в результате вы увидите описательную информацию об установленном центральном процессоре и его характеристиках


```

data-engineer@dataengineer-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model        : 126
model name    : Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz
stepping     : 5
cpu MHz      : 1497.600
cache size   : 6144 KB
physical id  : 0
siblings     : 1
core id      : 0
cpu cores    : 1
apicid      : 0
initial apicid : 0
fpu         : yes
fpu_exception : yes
cpuid level  : 22
wp          : yes

```

Получить же информацию об объеме задействования вычислительной процессорной мощности можно с помощью удобной утилиты для мониторинга ресурсов `top`, либо ее более удобной версии `htop`. Вот пример графического интерфейса `htop`

The screenshot shows the `htop` interface. At the top, it displays system statistics: CPU usage at 0.7%, Memory usage at 561M/971M, and Swap usage at 160M/448M. It also shows 107 tasks, 216 threads, 1 running, a load average of 0.02, and an uptime of 00:37:13. Below this is a table of processes with columns for PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The process list includes `htop` (PID 2324) and various system services like `gnome`, `systemd`, `cron`, and `dbus`. At the bottom, there is a legend for function keys: F1 Help, F2 Setup, F3 Search, F4 Filter, F5 Tree, F6 SortBy, F7 Nice, F8 Nice, F9 Kill, F10 Quit.

Эта утилита дает нам очень много информации о системе, поэтому давайте подробно разберем содержание этого интерфейса.

CPU	Загрузка центрального процессора, 0.7% от имеющейся производительности
Mem	Задействованная оперативная память, 561 Мб из 971 Мб свободных

Swp	swp это механизм виртуальной памяти, при котором часть данных из RAM сохраняется на жесткий диск. Позволяет экономить RAM. Задействованная swp память, 160Мб из 448Мб свободных
Tasks	Информация о запущенных задачах (Tasks), являющихся в linux синонимом процесса. Также здесь можно видеть информацию о выполняющихся потоках (thr, threads). Поток отличается от процесса тем, что у процесса собственные, изолированные от других процессов ресурсы, а потоки могут обращаться к одним и тем же ресурсам, например, одной области памяти. 107, 216 thr; 1 running 107 - активных задач (Tasks), которые занимают ресурсы, но могут находиться в пассивном режиме, 216 thr - запущенных потоков (Threads), 1 running - одна задача выполняется (в активной фазе)
Load average	Средняя загрузка системы за последние 1, 5 и 15 минут 0.02 0.02 0.00
Uptime	Продолжительность работы системы после последнего перезапуска

Далее идет таблица, в которой перечислены процессы и их параметры: PID, имя, пользователь, время запуска, задействованные процессом ресурсы, приоритет в системе. Эту таблицу удобно использовать при анализе проблем нехватки ресурсов.

Для проектов машинного обучения использования htop может оказаться недостаточным, так как эта утилита не умеет работать с графическими процессорами GPU, в htop нельзя увидеть статистику по задействованию GPU. Для мониторинга и управления GPU необходимо использовать специализированные утилиты, например, для работы с GPU производства NVidia используется утилита nvidia-smi

```
$ nvidia-smi
Fri Jun 10 15:50:50 2022
+-----+
| NVIDIA-SMI 440.44          | Driver Version: 440.44          | CUDA Version: 10.2          |
+-----+-----+-----+
| GPU  Name                | Persistence-M | Bus-Id        | Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          | Pwr:Usage/Cap|      /          |   /     | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+
|  0   GeForce RTX 207...   | off           | 00000000:02:00:0 | off    |      0%      N/A   |
| 0%   26C    P8           | 12w / 215w   |      11MiB /   7982MiB |         |             Default |
+-----+-----+-----+-----+-----+
|  1   GeForce RTX 207...   | off           | 00000000:0A:00:0 | off    |      0%      N/A   |
| 0%   26C    P8           | 17w / 215w   |      11MiB /   7982MiB |         |             Default |
+-----+-----+-----+-----+-----+
+-----+
| Processes:                 |                                     | GPU Memory Usage             |
| GPU       PID    Type   Process name                      |                                     |                               |
+-----+-----+-----+-----+-----+
| No running processes found |                                     |                               |
+-----+-----+-----+-----+-----+
```

В консольном выводе утилиты nvidia-smi вы видите характеристики оборудования и информацию о задействовании GPU.

И, в заключении этого раздела, отметим полезную python библиотеку psutil, которую можно использовать прямо в python коде, например в jupyter ноутбуке при проверке имеющихся ресурсов перед началом эксперимента.

Практическая задача: Выяснить какая папка имеет наибольший размер в директории /etc. Подсказка: воспользоваться утилитой ncdu.

2. Утилиты для работы с сетью

Сети передачи данных часто играют важную роль в проектах машинного обучения, поскольку большинство проектов машинного обучения имеют распределенную инфраструктуру, в которой отдельные узлы связываются друг с другом через сетевые интерфейсы с использованием сетевых протоколов.

Например:

- инженеры данных получают данные по сетевым протоколам от удаленных устройств,
- датасеты после преобразования сохраняются в удаленное сетевое хранилище данных,
- модель машинного обучения обучается на отдельном сервере с мощными GPU, после чего обученная модель сохраняется в git репозитории,
- служебные сервисы MLOps Airflow, MLFlow, Jenkins являются сетевыми и часто работают на отдельных служебных серверах,
- высокопроизводительные базы данных, например PostgreSQL, являются сетевыми и работают на выделенных серверах с возможностью доступа через сеть передачи данных.

Иметь возможность мониторить состояние сетей передачи данных и отдельных сетевых сервисов очень важно для контроля работы системы и диагностики неисправностей. Давайте рассмотрим основные, наиболее часто используемые утилиты.

Посмотреть состояние сетевых интерфейсов можно с помощью команды ifconfig

```
data-engineer@dataengineer-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.147.3 netmask 255.255.255.0 broadcast 192.168.147.255
    inet6 fe80::26ad:5174:d30:2262 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:97:41:71 txqueuelen 1000 (Ethernet)
    RX packets 80 bytes 10296 (10.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 101 bytes 12780 (12.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Для доступных сетевых интерфейсов ifconfig показывает

- параметры
 - MAC и IP адреса,
 - маску сети,
 - адрес шлюза,
 - размер максимальной единицы передачи (MTU, Maximum Transfer Unit)
- статистику
 - полученные и принятые пакеты
 - количество ошибок

Более подробную информацию о состоянии сетевых интерфейсов можно получить с помощью команды `netstat` с различными флагами, более подробную информацию об использовании того или иного флага можно прочитать в `man netstat`.

```
data-engineer@dataengineer-VirtualBox:~$ netstat -ap
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 localhost:domain       0.0.0.0:*               LISTEN
-
tcp        0      0 0.0.0.0:ssh            0.0.0.0:*               LISTEN
-
tcp        0      0 localhost:ipp          0.0.0.0:*               LISTEN
-
tcp        0      0 0.0.0.0:http           0.0.0.0:*               LISTEN
-
tcp6       0      0 [::]:ssh              [::]:*                  LISTEN
-
tcp6       0      0 ip6-localhost:ipp     [::]:*                  LISTEN
-
tcp6       0      0 [::]:http              [::]:*                  LISTEN
-
udp        0      0 localhost:domain       0.0.0.0:*               *
-
udp        0      0 dataengineer-Vir:bootpc 192.168.147.2:bootps    ESTABLISHED
-
udp        0      0 dataengineer-Vir:bootpc _gateway:bootps        ESTABLISHED
```

В зависимости от используемых флагов с помощью команды `netstat` можно посмотреть статистику по сетевым сервисам, задействованные порты, типы протоколов, состояния соединений, адреса удаленных подключений и многое другое. Командой `netstat` удобно пользоваться для диагностики инструментов машинного обучения, использующих сетевые подключения, например Jupyter, Airflow, Jenkins. Например, можно легко выяснить занят ли порт 22 (этот порт зарезервирован за службой `ssh` в файле `/etc/services`) следующим образом.

```
data-engineer@dataengineer-VirtualBox:~$ sudo netstat -ap | grep ssh
tcp        0      0 0.0.0.0:ssh            0.0.0.0:*               LISTEN
639/sshd: /usr/sbin
tcp6       0      0 [::]:ssh              [::]:*                  LISTEN
639/sshd: /usr/sbin
unix 2      [ ACC ]     STREAM  LISTENING  31982    1215/systemd
/run/user/1000/gnupg/S.gpg-agent.ssh
unix 2      [ ACC ]     STREAM  LISTENING  34155    1230/gnome-keyring-
/run/user/1000/keyring/ssh
unix 2      [ ACC ]     STREAM  LISTENING  33487    1390/ssh-agent
/tmp/ssh-rXkvhy797Dn4/agent.1323
unix 3      [ ]       STREAM  CONNECTED  24385    639/sshd: /usr/sbin
```

Из этой информации видим, что кроме стандартного демона `sshd`, который слушает в фоновом режиме входящие соединения на порте 22, в операционной системе есть и другие сетевые процессы `ssh`.

Практическая задача: выяснить в каком состоянии находится порт 8000 (свободен или занят), и, если он занят каким-то процессом, определить что это за процесс и какой у него PID.

Подсказка: воспользоваться утилитой netstat

3. Утилиты для работы с пользователями и группами

Вы уже сталкивались в предыдущих модулях с необходимостью создания пользователя с необходимыми привилегиями. Например, для работы с Ansible необходимо было создать пользователей с правами суперпользователя, то есть участника группы sudo, для выполнения команд на удаленных управляемых серверах. Пользователи и группы создаются, модифицируются и удаляются не очень часто, обычно это связано с появлением нового участника в команде проекта, либо с его уходом или изменениями в функциональных обязанностях. Каждый участник команды должен иметь свой аккаунт на рабочих серверах с соответствующими правами и полномочиями. Это позволяет лучше контролировать использование общих системных ресурсов, планировать и контролировать работу, диагностировать и устранять неисправности. Например, вы можете диагностировать непроизводительное расходование оперативной памяти на рабочем сервере, после чего определить, что один из исследователей данных загрузил в pandas.DataFrame большой датасет и забыл закрыть свой jupyter ноутбук перед отпуском. Linux позволяет быстро диагностировать причину такой ситуации и принять необходимые меры.

Создать пользователя в linux можно так:

adduser --ingroup “имя группы” --home ”имя директория” “имя пользователя”

С помощью флага --ingroup пользователь сразу добавляется в нужную группу, а флаг --home позволяет создать для нового пользователя домашнюю директорию, обычно в папке /home. При этом группа “имя группы” должна существовать. Группы пользователей в linux используются для удобства администрирования большого количества пользователей. Пользователей с похожими задачами объединяют в группы и к группам системный администратор может применять групповые политики, что существенно сокращает время администрирования пользователей. Например, можно создать группы data-researchers, ml-researchers, developers, в которые объединить исследователей данных, исследователей машинного обучения и разработчиков, соответственно. Создать группу можно с помощью следующей команды

addgroup “имя группы”

Удалить группу тоже очень просто

delgroup --only-if-empty “имя группы”

Посмотреть состав групп можно непосредственно в файле /etc/group, либо с помощью команды

members

Посмотреть информацию о пользователе

id username

Добавить пользователя в группу (-а обязательно add, иначе при выполнении команды будет удалена информация о текущих группах пользователя)

usermod -a -G docker “имя пользователя”

В этом примере пользователь добавляется в группу `docker`.

Удалить пользователя из группы

```
gpasswd -d user group
```

либо

```
userdel user group
```

В процессе работы может возникнуть необходимость изменений в пользовательских параметрах. Например, пользователь может перейти в другой отдел или ему понадобятся расширенные права доступа. Такие изменения делаются с помощью утилиты `usermod`, например вот так можно изменить рабочий каталог пользователя:

```
usermod -d “новый каталог” “имя пользователя”
```

Поменять пользователя и группу для файла

```
chown “имя пользователя”：“имя группы” “имя файла”
```

Иногда системному администратору необходимо выполнить определенные действия от имени конкретного пользователя. Поменять пользователя можно следующей командой

```
su “имя пользователя”
```

или

```
sudo su “имя пользователя” (поменять без пароля, воспользовавшись правами суперпользователя).
```

При расследовании причин неисправностей бывает полезно знать последние действия пользователей в системе, например, с помощью команды

```
last -a
```

Дата последнего лога каждого пользователя выводится с помощью команды

```
lastlog
```

Увидеть кто из пользователей в настоящее время находится в системе можно с помощью команды

```
who
```

Перечисленные утилиты позволяют управлять пользователями и контролировать их работу.

Практическая задача: установить в систему `docker`, добавить пользователя и добавить этого пользователя в группу `docker`, чтобы он мог выполнять команды `docker` без `sudo`

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

4. Утилиты для работы с процессами.

Кроме пользователей в `linux` важную роль играют процессы, выполняющие задачи. Например, для работы `JupyterHub` запускается процесс, обеспечивающий работу пользовательского интерфейса, авторизацию, выполнение скриптов. Каждый процесс использует общие системные ресурсы. Иметь инструмент для обнаружения “зависших” процессов, либо несанкционированно запущенных, очень удобно.

Посмотреть процессы, работающие в системе, можно следующей командой:

```
ps
```

или, для более подробного вывода

```
ps -aux
```

Например, вот так можно найти все процессы с названием jupyter

```
ps -aux | grep jupyter
```

Вертикальная черта “|” передает результат выполнения команды “ps -aux” на вход команды “grep jupyter”. Такая конструкция в linux называется конвейер. При отладке пайплайнов машинного обучения, состоящих из отдельных скриптов *.py вы можете использовать такой конвейер, например в виде

```
python get_data.py | python data_proc.py | train_model.py
```

Но такой вид скорее подходит для упрощенных экспериментов на самых начальных этапах по исследованию данных и созданию моделей.

С любым процессом в linux связано несколько важных параметров, которые вы видите при результате выполнения команды ps

```
data-engineer@dataengineer-VirtualBox:~$ ps
  PID TTY          TIME CMD
 1834 pts/0    00:00:00 bash
 7682 pts/0    00:00:00 ps
data-engineer@dataengineer-VirtualBox:~$
```

Например, PID это идентификатор процесса (Process IDentificator), уникальный номер процесса в системе. Если вам понадобится срочно остановить процесс, то вы можете это сделать с помощью команды

```
kill -9 “PID процесса”
```

Несмотря на устрашающее название команда kill используется всего лишь для отправки различных сигналов процессам, сигналы это сообщения, которыми процессы обмениваются между собой, сигнал “-9” означает “SIGKILL”, то есть “остановить процесс”, полный список возможных сигналов:

```
data-engineer@dataengineer-VirtualBox:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT    7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
11) SIGSEGV   12) SIGUSR2   13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD   18) SIGCONT    19) SIGSTOP    20) SIGSTP
21) SIGTTIN   22) SIGTTOU   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF   28) SIGWINCH   29) SIGIO      30) SIGPWR
31) SIGSYS    34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
data-engineer@dataengineer-VirtualBox:~$
```

Давайте разберем практическую задачу: в одном терминале запустим утилиту yes, которая будет постоянно выводить символ “y” на экран. Кстати, это используется для автоматизации установки программного обеспечения, в которой требуется ввести символ “y” (yes). А в другом терминальном окне мы найдем этот процесс, узнаем его уникальный PID и остановим с помощью команды kill.

Запустим yes и в терминал начнут выводиться символы “y” (либо те, которые мы укажем)

```
y
y
y
y
y
y
y
y
y
```

В другом окне сделаем следующее

```
data-engineer@dataengineer-VirtualBox:~$ ps -aux | grep yes
data-en+  7787 28.4  0.0  8088  516 pts/1  S+  13:34  0:01 yes
data-en+  7789  0.0  0.0  9044  724 pts/0  S+  13:34  0:00 grep --color
=auto yes
data-engineer@dataengineer-VirtualBox:~$ kill -9 7787
```

После этого процесс выполнения утилиты “yes” будет остановлен и в первом терминальном окне мы увидим следующее

```
y
y
y
Killed
data-engineer@dataengineer-VirtualBox:~$
```

Рекомендуем пользоваться утилитой kill очень осторожно, чтобы случайно не остановить какой-то нужный процесс по ошибке.

Практическая задача: узнать потребление ресурсов (оперативная память и процессор) процессов, участвующих в обучении модели ML.

Подсказка: воспользоваться ps и htop.

5. Штатные текстовые редакторы

Часто в процессе эксплуатации возникает необходимость быстро вручную исправить конфигурационные файлы или посмотреть содержимое файла. Для этих целей удобно использовать типовые текстовые редакторы linux. Например vi/vim, nano, emacs, nano и другие. Хороший обзор редакторов и некоторых правил работы с ними можно найти здесь: <https://losst.ru/luchshie-tekstovye-redaktory-linux>. Многие редакторы имеют не очень понятную логику использования, особенно для пользователей Windows, привыкших работать с MS Word. Однако после ознакомления с основными приемами работы в текстовых редакторах linux пользователи понимают их удобство и преимущество для решения эксплуатационных задач. Рекомендуем вам ознакомиться с одним из этих редакторов самостоятельно, чтобы при выполнении практических задач вы могли быстро вносить исправления в конфигурационные файлы или искать в них нужную информацию.

6. Управление окружением

Информация о глобальном окружении операционной системы находится в файле /etc/environment, который можно посмотреть с помощью текстового редактора, например vi

```
vi /etc/environment
```

В /etc/environment сохраняются значения переменных окружения, в частности PATH.

7. Управление работой сервисов

С помощью утилиты `systemctl` можно управлять работой сервисов, например, запускать, останавливать, перезапускать, узнавать текущий статус. Выполнение этой команды требует наличие прав суперпользователя (`sudo`). Вот как можно узнать текущий статус сервера `nginx`.

```
data-engineer@dataengineer-VirtualBox:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset:
   Active: active (running) since Fri 2022-06-10 17:11:02 +05; 1h 21min ago
     Docs: man:nginx(8)
  Process: 620 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_proce
  Process: 658 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (c
 Main PID: 670 (nginx)
    Tasks: 2 (limit: 1087)
   Memory: 3.1M
    CGroup: /system.slice/nginx.service
            └─670 nginx: master process /usr/sbin/nginx -g daemon on; master_
              └─673 nginx: worker process

июн 10 17:11:02 dataengineer-VirtualBox systemd[1]: Starting A high performanc
июн 10 17:11:02 dataengineer-VirtualBox systemd[1]: Started A high performanc
lines 1-15/15 (END)
```

Из информации видно, что web-сервер `nginx` успешно запущен, находится в состоянии `active (running)`. Этот сервис можно остановить с помощью команды

```
sudo systemctl stop nginx
```

или перезапустить с помощью команды

```
sudo systemctl restart nginx
```

Запуск задач по расписанию осуществляется с помощью `crontab`.

Например, увидеть актуальное расписание для задач можно с помощью команды `crontab -l`

```
data-engineer@dataengineer-VirtualBox:~$ crontab -l
no crontab for data-engineer
data-engineer@dataengineer-VirtualBox:~$
```

Чтобы перейти к созданию расписания для выполнения задачи надо использовать команду `crontab -e`. Для создания расписания используется специфический `crontab` синтаксис

```
# строки-комментарии
#
# задача, выполняемая ежеминутно
* * * * * "команда"
# задача, выполняемая на 10-й минуте каждого часа
10 * * * * "команда"
# задача, выполняемая на 10-й минуте второго часа каждого дня и использующая перенаправление
стандартного потока вывода
10 2 * * * "команда" > /tmp/logfile.log
```

Вот полезный ресурс для того, чтобы осуществлять преобразование форматов в `crontab`
<https://crontab.guru/>

8. Средства для удаленной работы

- rsync синхронизация файлов и папок на разных хостах
- wget скачать файл
- nmap проверить порты
- telnet незащищенное консольное подключение
- ftp передача файлов
- ssh защищенное консольное подключение
- curl отправить информацию на удаленный сервер

Практическая задача: скачать удаленно файл из Интернет.

Подсказка: воспользоваться wget

Конечно, в linux существует еще масса полезных инструментов, как консольных, так и графических. Как и для любых других инструментов, уверенность в использовании приходит с опытом, поэтому рекомендуем вам практиковаться в использовании команд linux для решения типовых задач.

Тест

1. Какие команды используются для проверки доступного места на жестком диске? (0.25)
 - a. **df**
 - b. netstat
 - c. grep
 - d. free
2. Какая команда позволяет определить процесс, занимающий порт 8000? (0.25)
 - a. sudo get port 8000
 - b. **sudo netstat -ap | grep 8000**
 - c. sudo what port 8000
 - d. sudo ifconfig port 8000
3. Какая утилита позволяет выполнять команды от имени суперпользователя? (0.25)
 - a. su
 - b. superuser
 - c. **sudo**
 - d. do
4. Какая утилита позволяет администрировать GPU? (0.25)
 - a. gpu-util
 - b. **nvidia-smi**
 - c. hw-util
 - d. gpu-admin

Итоги/выводы

Большинство проектов машинного обучения используют в качестве операционной системы linux. Поэтому инженеры DevOps/MLOps должны уметь пользоваться необходимыми утилитами linux для эффективного мониторинга и управления ресурсами проекта. В этом юните вы познакомились с утилитами для системного администрирования операционной

системы Linux, которые вам обязательно пригодятся при создании и настройке инфраструктуры, мониторинге и поиске неисправностей.

Использование этих инструментов предполагает уверенное владение консолью для выполнения команд, что бывает неудобно для визуализации сложных процессов. Следующие юниты посвящены инструментам с более удобным графическим интерфейсом, что позволяет осуществлять мониторинг и управление программными системами без выполнения команд в консоли.

Модуль 8. Юнит 2. Утилита Tensorboard для мониторинга.

Введение: В предыдущих модулях вы уже узнали о различных инструментах автоматизации процесса машинного обучения, которые имеют средства визуального контроля выполнения операций, например Airflow, Jenkins. Имеющиеся в Airflow и Jenkins штатные средства визуализации не позволяют осуществлять подробный мониторинг процессов в системе. Например, в процессе обучения модели машинного обучения встроенные средства визуализации Jenkins или Airflow сигнализируют только о том, что тот или иной процесс завершился аварийно, а при необходимости узнать детали требуется изучение сообщений в консольном выводе приложений. Для оперативного мониторинга важных процессов в проектах машинного обучения используется специальный подход, когда выполняющийся скрипт или программа пишут в специальный log файл информацию о параметрах своей работы в реальном времени, а специальные средства визуализации читают информацию из этого log файла и отображают прочитанные параметры в графическом виде. Такой подход позволяет своевременно увидеть негативные тенденции во внутренних выполняющихся процессах. **Кроме рассмотренных ранее общесистемных параметров (оперативная память, процессор, жесткий диск, сетевые интерфейсы), к важным параметрам, которые нужно наблюдать и контролировать, относятся:**

- метрики качества модели машинного обучения в процессе обучения и инференса,
- количество ошибок в поступающих данных,
- процент выполнения задачи.

В этом юните мы изучим инструмент Tensorboard, а в следующем юните вы узнаете про его более универсальный и широко распространенный аналог, Grafana. Встроенный во фреймворк Tensorflow инструмент Tensorboard позволяет осуществлять мониторинг процесса обучения модели в Tensorflow, например, контролировать метрики, предупреждать переобучение. Благодаря наглядности и интерактивности Tensorboard является удобным инструментом контроля процесса машинного обучения и эксплуатации модели.

Содержание юнита:

Одна из важных задач в процессе машинного обучения это контроль переобучения модели. С этой целью обычно анализируют в процессе обучения качество работы модели на тренировочных данных, на которых модель учится, и тестовых данных, которые модель ранее не видела. Если в процессе обучения при росте качества работы модели на тренировочных данных стало ухудшаться качество работы на тестовых данных, то это означает, что модель переобучилась и дальнейшее обучение не имеет смысла.

Этот процесс можно контролировать штатными средствами python, например с помощью графиков, построенных в библиотеках для визуализации: matplotlib, seaborn или plotly. При этом сначала выполняется обучение модели, анализируется качество работы модели на обучающей и тестовой выборках на каждой эпохе, а после обучения модели эти результаты для каждой эпохи представляются на графиках. Вот, например, как можно получить информацию о ходе обучения модели в Tensorflow. Сначала результаты обучения на каждой из эпох сохраняются в специальных словарях `history.history['loss']` и `history.history['val_loss']`

```

[1] from tensorflow.keras.datasets.mnist import load_data
    from tensorflow.keras import layers as L
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.utils import to_categorical
    import matplotlib.pyplot as plt

[2] (X_train, y_train), (X_test, y_test) = load_data()
    X_train = X_train.reshape(X_train.shape[0], -1)
    X_test = X_test.reshape(X_test.shape[0], -1)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

[3] model = Sequential()
    model.add(L.Input(shape=784))
    model.add(L.Dense(100))
    model.add(L.Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

[4] history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5)

Epoch 1/5
1875/1875 [=====] - 10s 5ms/step - loss: 14.8551 - accuracy: 0.8524 - val_loss: 6.6528 - val_accuracy: 0.8829
Epoch 2/5
1875/1875 [=====] - 5s 2ms/step - loss: 3.3825 - accuracy: 0.8778 - val_loss: 1.6172 - val_accuracy: 0.8797
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 1.1448 - accuracy: 0.8675 - val_loss: 0.9458 - val_accuracy: 0.8536
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.9316 - accuracy: 0.8574 - val_loss: 0.8935 - val_accuracy: 0.8594
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.9743 - accuracy: 0.8537 - val_loss: 0.8220 - val_accuracy: 0.8796

[5] plt.plot(history.history['loss'], label='train loss')
    plt.plot(history.history['val_loss'], label='validation loss')
    plt.legend()
    plt.show()

```

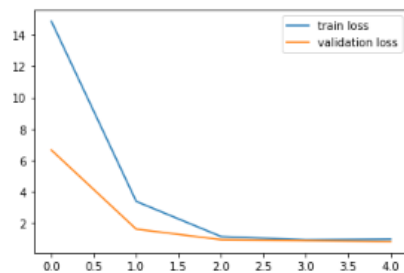


Рисунок “Пример обучения модели и визуализации потерь на разных эпохах”

То есть мы визуализировали информацию о динамике изменения loss (потери на тренировочной выборке) и val_loss (потери на валидационной выборке) штатными средствами matplotlib. И если в процессе обучения имеет место переобучение модели, то есть ситуация, при которой качество модели на тренировочных данных растет, а на валидационных постоянно снижается, это можно будет легко увидеть на графике. Такой подход позволяет проанализировать только уже завершённый процесс обучения. Есть инструменты, позволяющие мониторить процесс обучения в реальном времени. Один из таких инструментов TensorBoard, который является штатным средством TensorFlow. Использование TensorBoard предполагает запись в log-файлы на каждом этапе выполнения обучения, TensorBoard считывает информацию из этих log-файлов и показывает данные в удобном графическом формате.

Давайте сначала рассмотрим пример использования TensorBoard в среде google colab (<https://colab.research.google.com>), который позволяет обращаться к Tensorboard с использованием “магических команд” Jupyter ноутбука в google colab, %load_ext tensorboard и %tensorboard. Вне google colab эти магические команды не работают, для этих ситуаций использование tensorboard описано далее в юните. При использовании tensorboard

в google colab сначала надо загрузить tensorboard с использованием “магической” команды `%load_ext tensorboard`. Далее создается и обучается модель и при обучении идет сохранение информации о параметрах модели с использованием механизма функций “обратного вызова” (“функций-колбаков”, callbacks). Функции “обратного вызова” срабатывают при наступлении определенного события, в нашем случае они выполняются при завершении каждой эпохи обучения.

```

▶ %load_ext tensorboard
  from tensorflow.keras.callbacks import TensorBoard

↳ The tensorboard extension is already loaded. To reload it, use:
   %reload_ext tensorboard

[12] logdir = "logs"
     tensorboard_callback = TensorBoard(logdir, histogram_freq=1)

     history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, callbacks=[tensorboard_callback])

Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 1.0011 - accuracy: 0.8562 - val_loss: 0.9891 - val_accuracy: 0.8542
Epoch 2/5
1875/1875 [=====] - 5s 2ms/step - loss: 1.0028 - accuracy: 0.8625 - val_loss: 0.9312 - val_accuracy: 0.8862
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.9859 - accuracy: 0.8695 - val_loss: 0.8983 - val_accuracy: 0.8806
Epoch 4/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.9741 - accuracy: 0.8684 - val_loss: 0.8530 - val_accuracy: 0.8873
Epoch 5/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.9801 - accuracy: 0.8706 - val_loss: 1.2246 - val_accuracy: 0.8464

```

В процессе обучения можно вызвать TensorBoard с помощью магической команды `%tensorboard` и увидеть состояние параметров модели в режиме реального времени

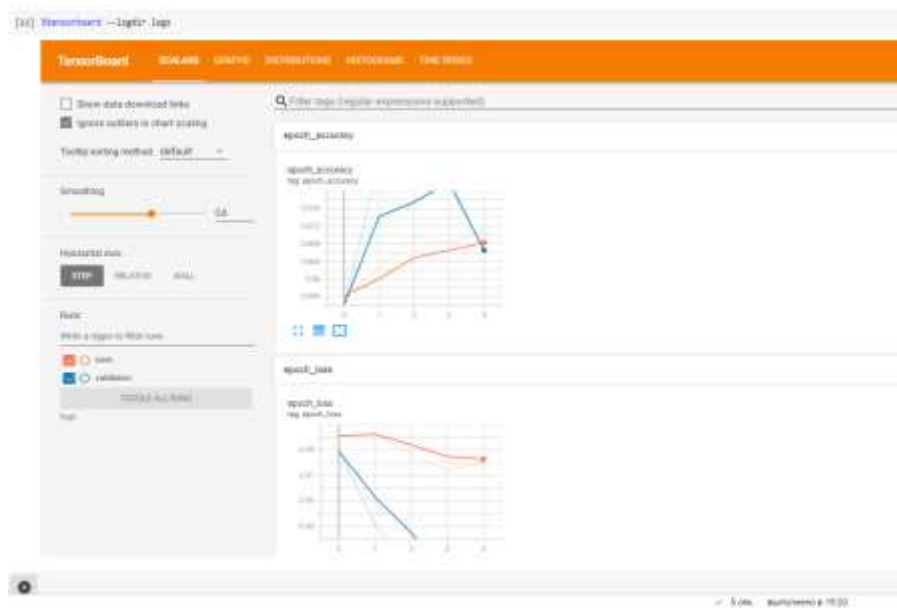


Рисунок “Графический интерфейс tensorboard”

При этом tensorboard является сетевым приложением и к нему можно подключиться с удаленного компьютера или системы, чтобы получить актуальную информацию о ходе процесса обучения.

```

$ tree log_dir/ -L 2
log_dir/
├── train
│   ├── events.out.tfevents.1655235625.bruteforce.20353.0.v2
│   ├── events.out.tfevents.1655235625.bruteforce.profile-empty
│   └── plugins
└── 2 directories, 2 files

```

Запустим продолжительное обучение модели с указанием функции обратного вызова, записывающей информацию в log файл.

```
Ввод [*]: model_conv.fit(X_train_proc, X_train_proc, epochs=10, verbose=1, callbacks=[callback_tensorboard])
Epoch 1/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.1587
Epoch 2/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.0691
Epoch 3/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.0548
Epoch 4/10
1875/1875 [=====] - 20s 11ms/step - loss: 0.0471
Epoch 5/10
231/1875 [==>.....] - ETA: 17s - loss: 0.0442
```

Одновременно запустим сервис tensorboard на том же сервере, где сохраняется log файл
tensorboard --logdir="имя лог файла"



Видим информацию о состоянии loss в графическом интерфейсе.

Конечно, с помощью функций обратного вызова можно выводить только информацию, касающуюся обучения, а для конкретных производственных задач может потребоваться большая гибкость в возможностях вывода. Например, мы хотели бы отслеживать в Tensorboard использование оперативной памяти или значение какой-то внутренней переменной программы. В Tensorboard поддерживается такая операция записи с использованием специального объекта writer, создаваемого с помощью функции tensorflow.summary.file_writer().

```
import tensorflow as tf

summary_writer = tf.summary.create_file_writer("./logs")
for i in range(5):
    with summary_writer.as_default():
        tf.summary.scalar('a', i**2, step=i)
```

В приведенном коде происходит запись квадрата величины i в лог файл. При этом в графическом интерфейсе Tensorboard мы увидим следующий график изменения этой величины. Используя этот механизм можно записывать в log файл для последующего отображения в графическом интерфейсе Tensorboard любую полезную информацию, которую необходимо контролировать.

Практическое задание:

Создайте свою произвольную модель машинного обучения в Tensorflow и визуализируйте процесс ее обучения с TensorBoard.

Результат выполнения задания: jupyter ноутбук, который можно выполнить в google colab.

Итоги/выводы

В связи с тем, что процесс обучения модели машинного обучения является очень продолжительным по времени, необходимы инструменты для мониторинга этого процесса в реальном времени, чтобы своевременно принимать меры по устранению возможных неполадок. Это позволит сэкономить значительное количество времени, поскольку аппаратное обеспечение не будет загружено бесполезными вычислениями. В этом юните вами был изучен инструмент Tensorboard для визуализации процесса машинного обучения в реальном времени. В качестве практической задачи рассмотрено применение TensorBoard в среде google colab и утилита tensorboard в linux. В следующем юните вы узнаете о более распространенном инструменте Grafana, который применяется не только в проектах машинного обучения.

Модуль 8. Юнит 3. Визуализация и контроль с использованием Grafana.

Введение:

Любая сложная технологическая система содержит множество параметров, влияющих на ее работу. Мониторинг этих параметров позволяет своевременно заметить опасные тенденции в работе системы и принять меры по их устранению. Как известно, проблему гораздо проще предотвратить, чем устранять последствия аварии. Анализировать большое количество параметров человеку очень непросто. Даже самый опытный и внимательный инженер имеет предел своей производительности, связанный с физическими ограничениями. Чрезмерная нагрузка на специалиста, занимающегося обслуживанием сложных технологических систем может привести к ошибкам, связанным с “человеческим фактором”, поскольку человек не может работать двадцать четыре часа в сутки продолжительное время и не может постоянно анализировать данные, поступающие от сотен различных систем. Обслуживание сложных технологических систем невозможно без специальных инструментов мониторинга и автоматизированного контроля, являющихся незаменимыми помощниками в процессе эксплуатации. Такие инструменты умеют собирать данные из различных источников, визуализировать эти данные, уведомлять специалистов при возникновении какого-то нарушения и многое другое. В этом юните вы узнаете об одном из таких инструментов, фреймворке Grafana. Полученные сведения и практические навыки вы сможете применить для добавления качественной визуализации процессов мониторинга и контроля с использованием Grafana в любой из этапов проекта машинного обучения, начиная от сбора и анализа данных и заканчивая процессом эксплуатации модели. Не умаляя важности всех этапов проекта машинного обучения следует отметить, что без обеспечения качественной процедуры эксплуатации модели в производственной среде все другие усилия и достижения команды проекта могут оказаться бесполезными, так как для заказчика польза системы будет сомнительной из-за плохих результатов в эксплуатации. Поэтому в изложении возможностей Grafana в этом юните сделан акцент именно на эксплуатационных задачах.

В результате изучения этого юнита вы узнаете:

- историю создания и развития фреймворка Grafana, его основные функциональные возможности,
- процедуру установки Grafana,
- варианты применения Grafana в проекте машинного обучения.

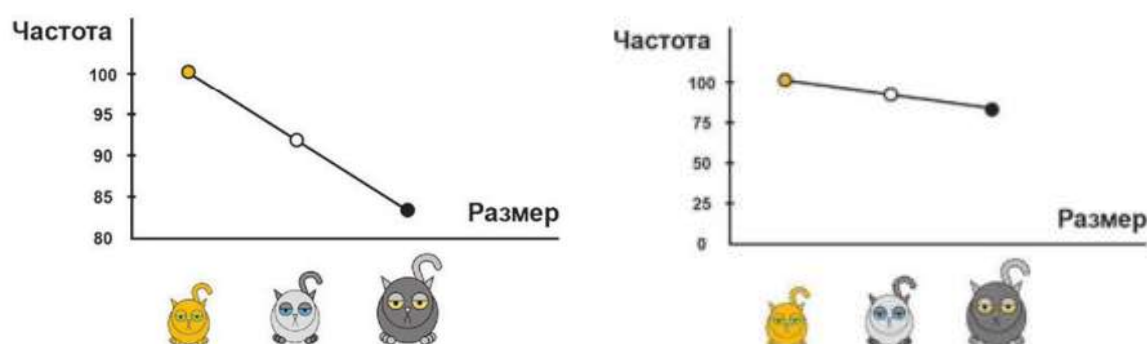
Содержание юнита:

В проектах машинного обучения часто встречаются участники, не владеющие навыками работы с данными, не умеющие обучать модели машинного обучения, не разбирающиеся в математике и статистике. Большинство людей и, в том числе, наших коллег, не обладают теми знаниями, которыми обладают исследователи данных или инженеры машинного обучения. Тем не менее такие специалисты могут играть очень важную для проекта роль: заниматься поддержанием работоспособности проекта на производственной площадке, работать с результатами работы модели или даже принимать решение о финансировании такого проекта машинного обучения. Во всех перечисленных случаях для инженера эксплуатации (обслуживающего систему), оператора (запрашивающего у модели машинного обучения результаты прогноза) или для финансового директора

характеристики, особенности и результаты работы проекта машинного обучения должны быть переведены на понятный им язык и, чаще всего, это содержательные и интерпретируемые графики. Чем более понятным и содержательным будет график, тем правильнее будет принято решение, основанное на данных, представленных на этом графике, даже если специалист, принявший это решение, не обладает глубоким пониманием нюансов работы модели.

Визуализация является важнейшей частью современных программных систем, которая относится к так называемому UX/UI (User eXperience / User Interface, пользовательский опыт / пользовательский интерфейс) и означает специальные подходы к созданию удобного, непротиворечивого, эргономичного, эффективного пользовательского интерфейса для работы с приложениями. UX/UI посвящено много тематической литературы, но в данном юните мы будем рассматривать важность качественного пользовательского интерфейса только в части использования эффективных графиков и дашбордов, содержащих визуальную информацию, необходимую для качественной эксплуатации программных систем и, в том числе, проектов машинного обучения.

То, что правильная визуализация влияет на восприятие, можно увидеть в интересном и показательном примере из книги Владимира Соловьева “Статистика и котики”, в которой в игровой форме излагаются сложные понятия и инструменты статистического анализа. Благодаря простоте и оригинальности изложения эта книга стала бестселлером. Ниже приведены рисунки из этой книги. На левом рисунке график, из которого следует, что при изменении размера происходит существенное изменение частоты появления особи такого размера. А вот на правом графике это изменение уже не такое радикальное, при увеличении размера график частоты меняется незначительно. Курьезность ситуации заключается в том, что на левом и правом графиках описан один и тот же процесс! Приглядитесь внимательно и вы поймете насколько может визуализация поменять представление об описываемом процессе или демонстрируемых результатах.



Таким образом, качественная визуализация позволяет лучше понять процесс, который эта визуализация описывает.

Современные средства, используемые в процессе эксплуатации сложных систем, используют специальные инструменты и подходы, которые позволяют эффективно решать задачи эксплуатации:

- интерактивные дашборды, позволяющие визуализировать данные с изменением параметров, увидеть изменение процесса в динамике, при меняющихся условиях,

здать фильтры или дополнительные условия для того, чтобы получить более информативную и понятную визуализацию,

- работа с многомерными данными и временными рядами, возможности для анализа их взаимосвязи,
- автоматизированный мониторинг значений параметров в заданных диапазонах, при превышении порога срабатывания (так называемой уставки), система может уведомить персонал эксплуатации о превышении,
- различные системы аналитики
 - описательная (дескриптивная) аналитика описывает уже произошедшие события
 - предиктивная аналитика позволяет предсказывать события
 - предписывающая аналитика дает рекомендации для принятия оптимальных действий
- типовые интерфейсы для автоматизации и взаимодействия с другими системами (SQL, JSON)
- подсистему авторизации пользователей, позволяющую предоставлять информацию только тем пользователям, которые авторизованы для получения этой информации.

В настоящее время создано большое количество библиотек, фреймворков, open-source и коммерческих продуктов, которые решают перечисленные выше задачи. Grafana, один из самых популярных open-source проектов, предназначен для визуализации данных, алертов – независимо от того где находятся данные. Мы рассмотрим этот инструмент в данном юните.

Вся необходимая для работы информация о Grafana находится на официальном сайте этого проекта <https://grafana.com>. Здесь можно найти все необходимые инструкции по работе, примеры, программное обеспечение. Grafana был создан в 2014 году, с тех пор этот инструмент приобрел огромную популярность. Grafana это платформа с открытым программным кодом (open-source), которую используют для визуализации, мониторинга и анализа данных. Grafana может взаимодействовать с множеством систем, которые являются для нее “источниками” данных для визуализации. В соответствии с поддерживаемой архитектурой «подключаемых» источников данных, Grafana имеет модули интеграции для большинства типовых инструментов работы с данными:

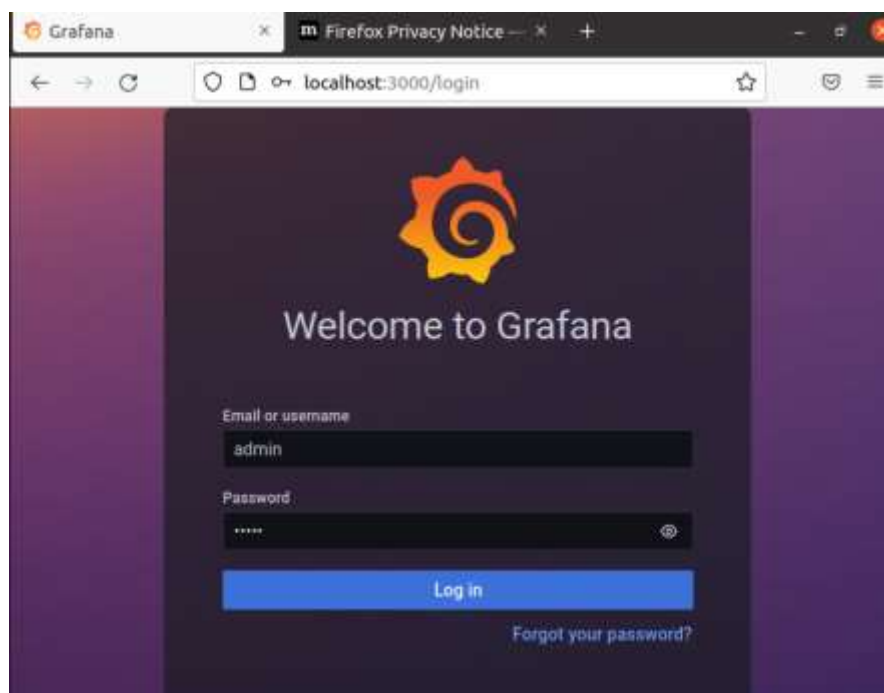
- реляционные базы данных: MySQL, PostgreSQL и др.
- нереляционные базы данных: Graphite, Prometheus, Elasticsearch, Open TSDB, InfluxDB и др.
- служебные системы мониторинга облачных сервисов: Google StackDriver, MS Azure, Amazon CloudWatch и др.

Для каждого источника, с которым взаимодействует Grafana, поддерживается синтаксис этого источника, реализуемый в специальном редакторе запросов.

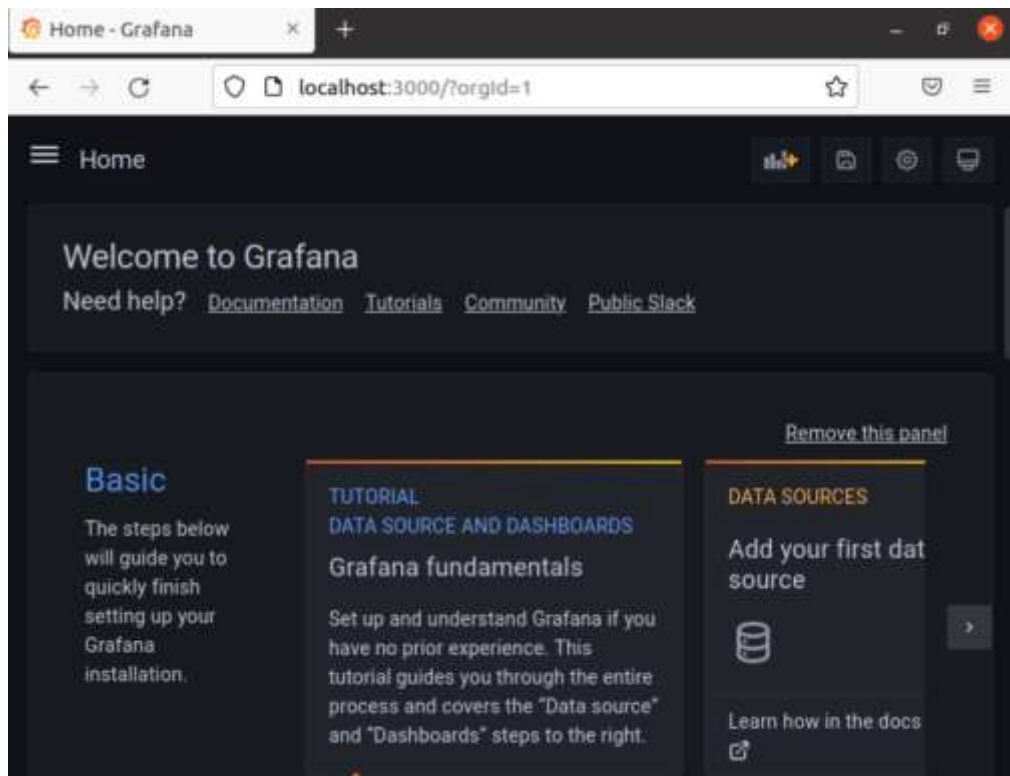
Grafana имеет возможность быстрого создания учебного макета, на котором любой может ознакомиться с возможностями Grafana, поэкспериментировать с настройками и выполнением типовых действий по администрированию и использованию. Это учебное

окружение для изучения grafana создается очень просто с использованием следующих действий:

- изучить описание по адресу <https://grafana.com/tutorials/grafana-fundamentals/?pg=docs>, установить необходимое, описанное в этой документации, программное обеспечение, например, docker, docker-compose и git,
- клонировать к себе официальный репозиторий проекта с помощью команды **git clone https://github.com/grafana/tutorial-environment.git**
- перейти в рабочую папку с помощью команды **cd tutorial-environment/**
- запустить контейнеры **docker-compose up**
- далее можно открыть с помощью браузера приложение на локальном хосте по адресу: hostname:3000 (по умолчанию имя пользователя admin, пароль admin, конечно же после логирования в системе рекомендуется эти параметры изменить)
- теперь можно использовать систему, например добавлять источники данных во вкладке Configuration -> Add Data->Prometheus.



При первом входе в систему вам будет предложено изменить реквизиты входа, рекомендуется это сделать, так как реквизиты admin:admin являются небезопасными, их использование делает систему уязвимой. После этого вы попадаете в основной интерфейс Grafana, который имеет следующий вид



Для дальнейшей работы необходимо понимать основные определения, которые используются в визуализациях Grafana:

- **Панель** Grafana это базовый элемент визуализации выбранных данных. Поддерживается работа с различными видами панелей: с графиками, статусами, таблицами, тепловыми картами и произвольным текстом, а также интеграцию с полезными плагинами (например, карта мира или часы) и приложениями, которые также можно визуализировать. Можно настроить стиль и формат каждой панели; все панели можно переносить на другое место, перестраивать и изменять их размер.
- **Дашборд** это набор отдельных панелей, сопровождающийся набором переменных, которые можно менять. При изменении переменных меняется и содержание дашборда. Все дашборды можно настраивать с учетом конкретных технических требований проекта. Благодаря поддержке большого сообщества в Grafana есть большой выбор готовых дашбордов для разных типов данных и источников.
- В дашбордах можно использовать **аннотации** для отображения определенных событий на разных панелях. Аннотации добавляются настраиваемыми запросами, на графике аннотация отображается вертикальной красной линией. При наведении курсора на аннотацию можно получить описание события и теги. Благодаря этому можно легко сопоставить время, конкретное событие и его последствия в приложении и исследовать поведение системы.

Вот примеры дашбордов Grafana

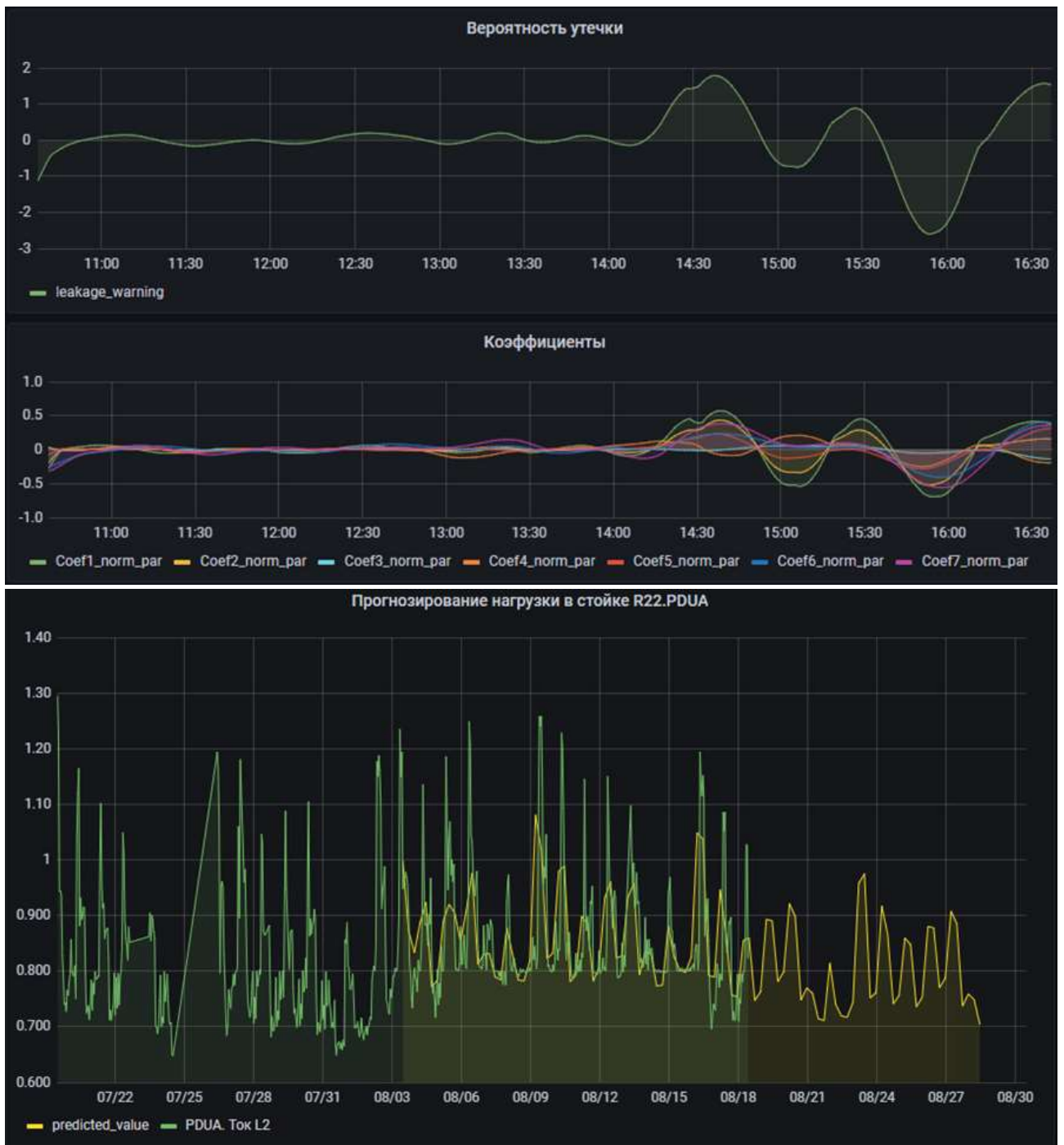
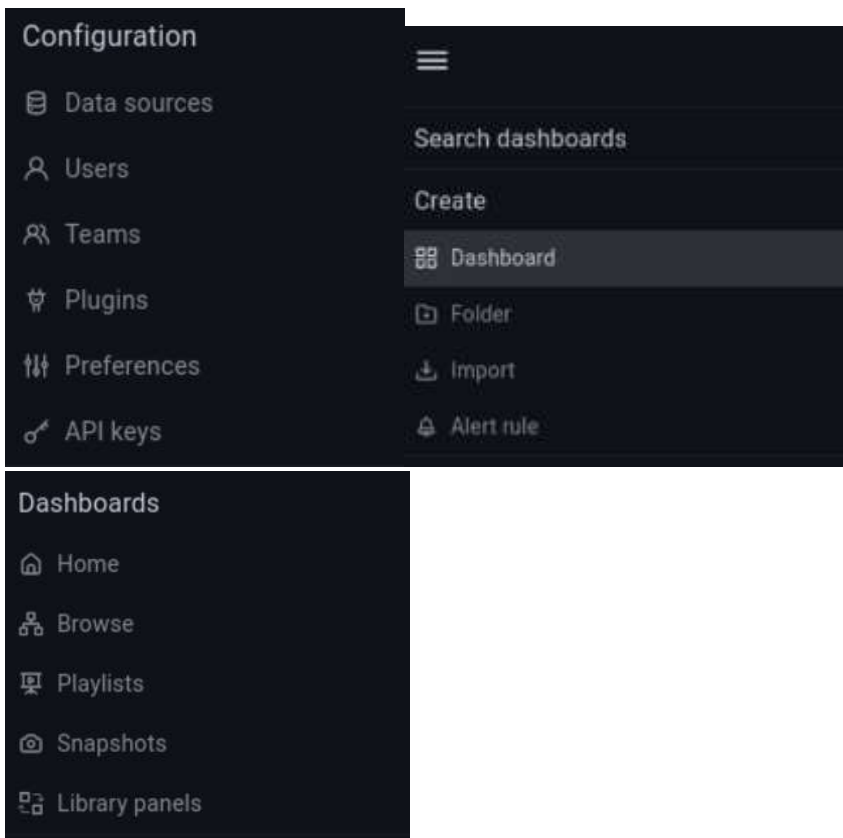


Рисунок “Примеры дашбордов (интерактивных панелей) в Grafana”

Для начала работы наиболее полезными являются разделы “Подключиться к источнику данных”, “Создать дашборд”, “Посмотреть готовые дашборды”



Давайте рассмотрим пример подключения Grafana к базе данных PostgreSQL, расположенной на другом сервере, и отображения на дашборде данных из этой базы данных. В операционной системе Ubuntu база данных PostgreSQL устанавливается очень просто, с использованием команд

```
sudo apt update
```

```
sudo apt install postgresql postgresql-contrib
```

После этого на вашем сервере будет установлена база данных PostgreSQL и служебные утилиты для этой базы. База данных postgresql поддерживает различные механизмы авторизации пользователей, по умолчанию используется концепция пользователей и ролей linux, при установке postgresql создается пользователь postgres, к которому можно перейти с использованием команды

```
sudo -i -u postgres
```

после этого можно вызвать утилиту **psql** и перейти к командной строке для взаимодействия с postgresql.

```
data-engineer@dataengineer-VirtualBox:~$ sudo -i -u postgres
postgres@dataengineer-VirtualBox:~$ psql
psql (12.11 (Ubuntu 12.11-0ubuntu0.20.04.1))
Type "help" for help.

postgres=#
```

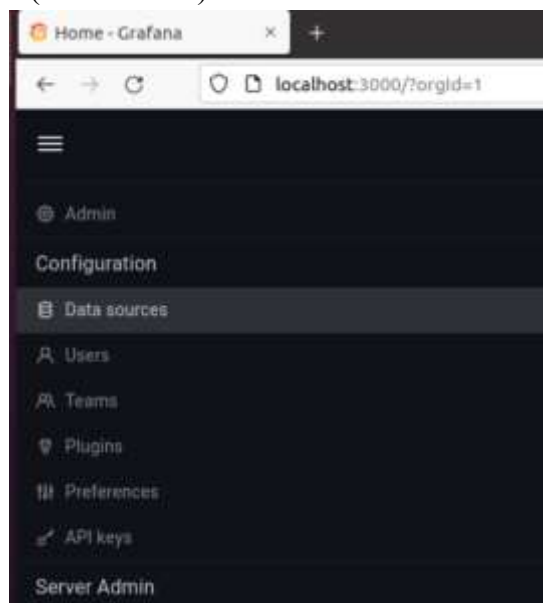
Теперь, после подключения в базе данных postgresql пользователем postgres мы можем создавать базы данных, таблицы и вносить в них данные с помощью SQL запросов. Команды, с помощью которых можно выполнить эти действия, приведены ниже. Рекомендуем вам самостоятельно изучить взаимодействие с базой данных postgresql и SQL запросы, в частности, можно воспользоваться информацией по ссылкам в конце модуля.

```

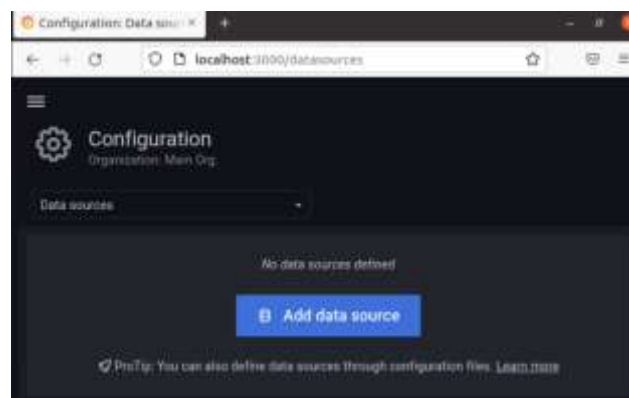
postgres=# CREATE DATABASE grafana;
CREATE DATABASE
postgres=# \c grafana
You are now connected to database "grafana" as user "postgres".
grafana=# CREATE TABLE grafana (ts real, acc real, val_acc real);
CREATE TABLE
grafana=# INSERT INTO grafana (ts, acc, val_acc) VALUES (1, 0.1, 0.1);
INSERT 0 1
grafana=# INSERT INTO grafana (ts, acc, val_acc) VALUES (2, 0.2, 0.2);
INSERT 0 1
grafana=# SELECT * FROM grafana;
 ts | acc | val_acc
-----+-----+-----
  1 | 0.1 |      0.1
  2 | 0.2 |      0.2
(2 rows)
grafana=#

```

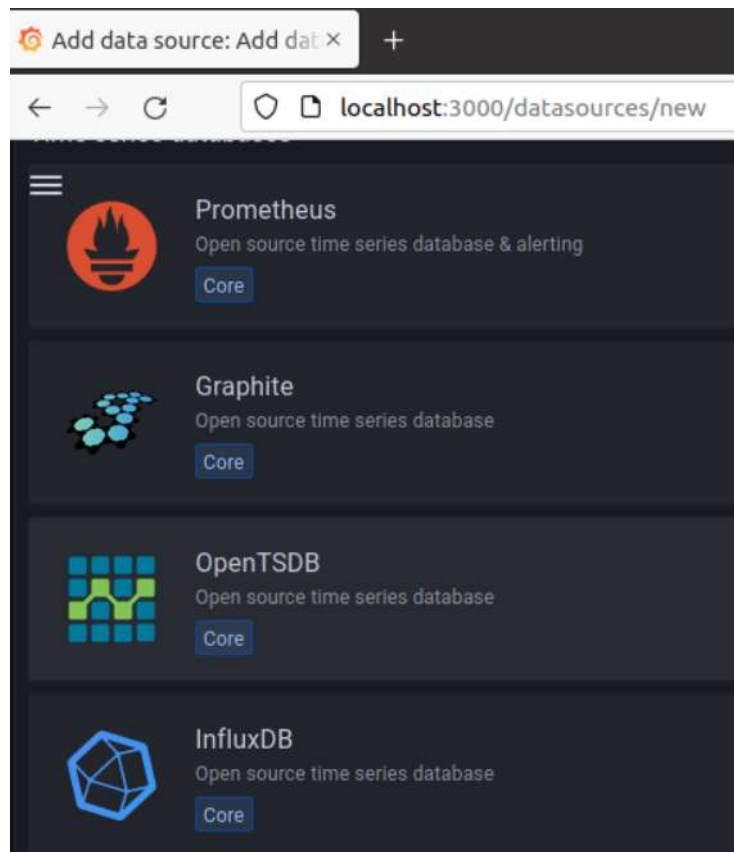
Теперь у вас в базе данных находятся два измерения для accuracy (поле acc) и validation accuracy (поле val_acc). Давайте отобразим их в grafana. Для этого сначала надо настроить в grafana источник данных (Data Source)



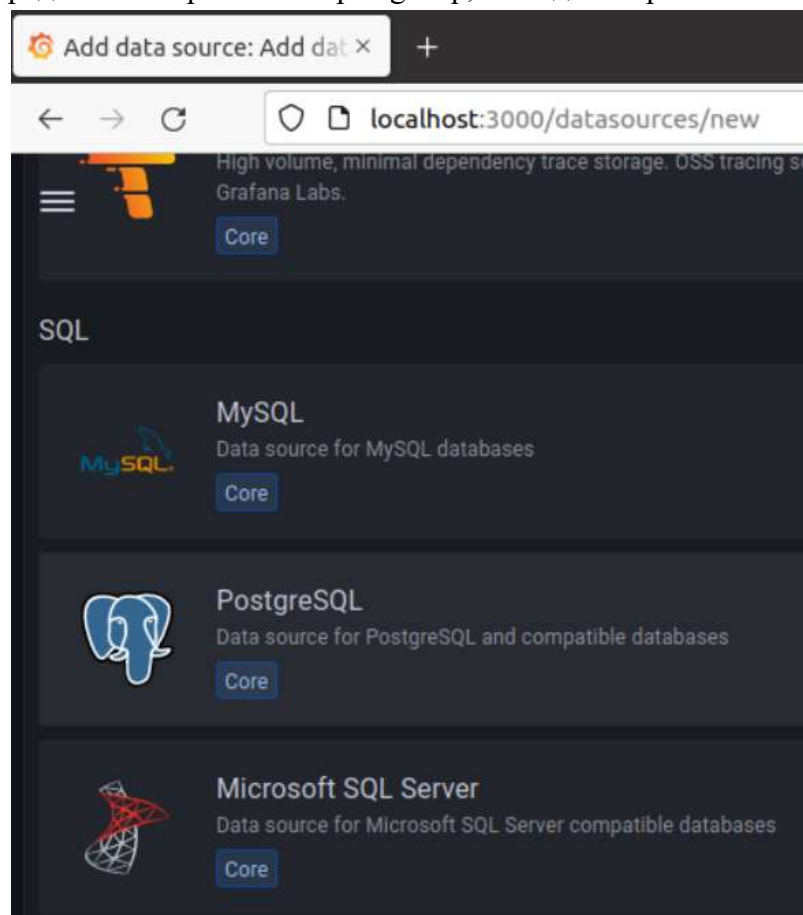
В появившемся диалоговом окне нажимаем клавишу “Add data source” (добавить источник данных).



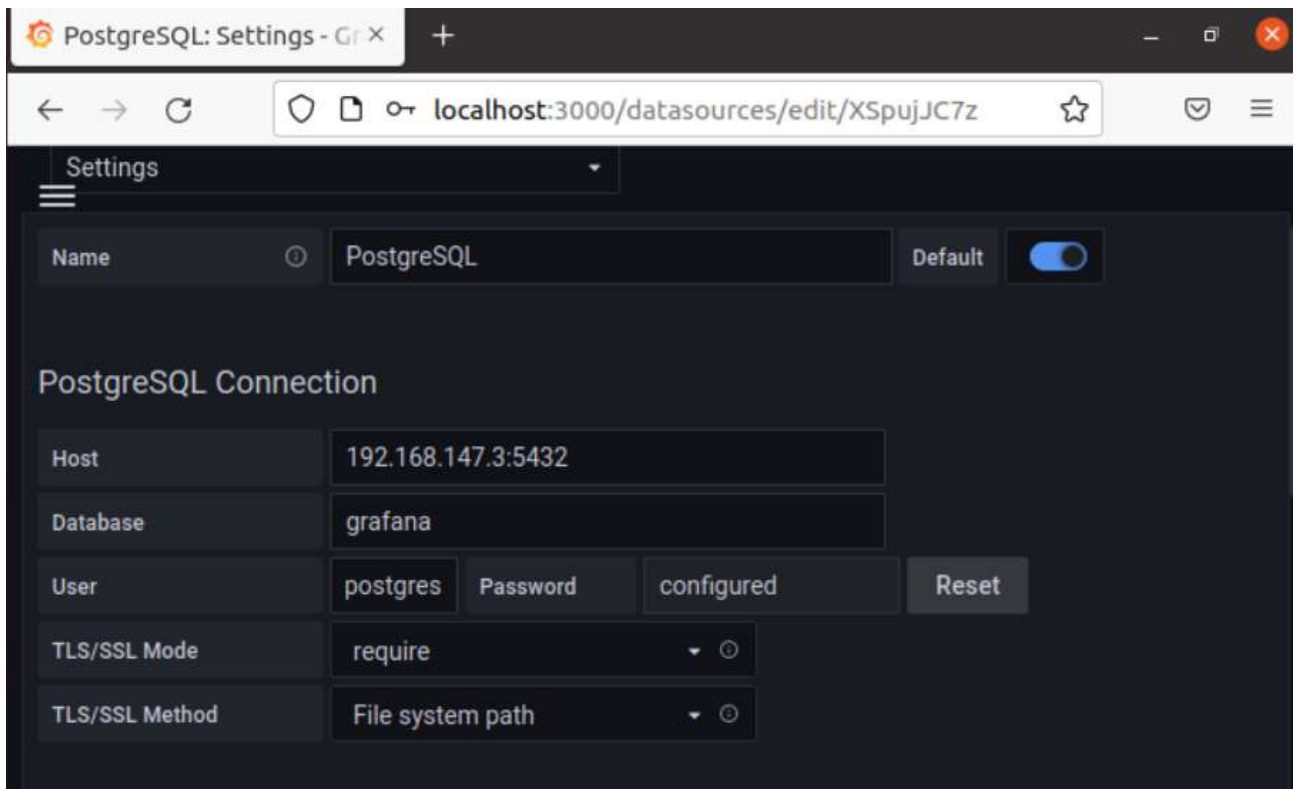
Вы увидите большое количество различных вариантов источников данных для grafana



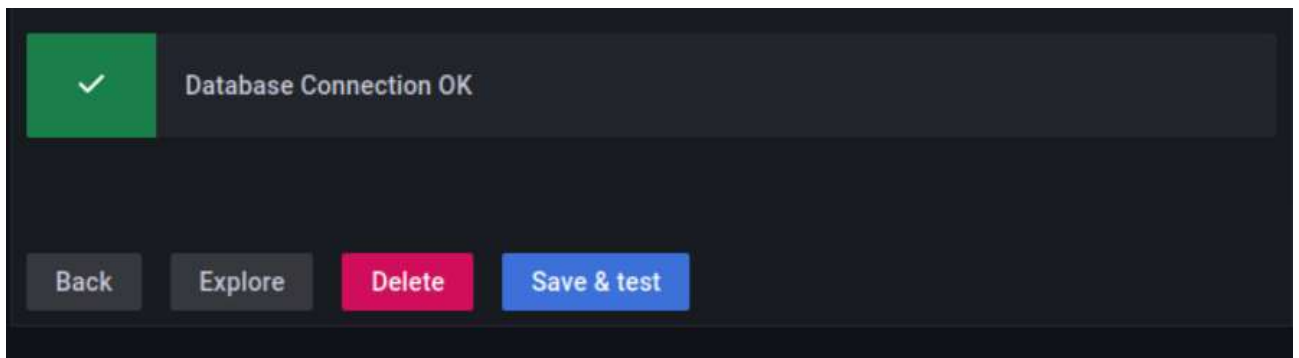
Поскольку мы предполагаем работать с postgresql, то надо выбрать этот источник



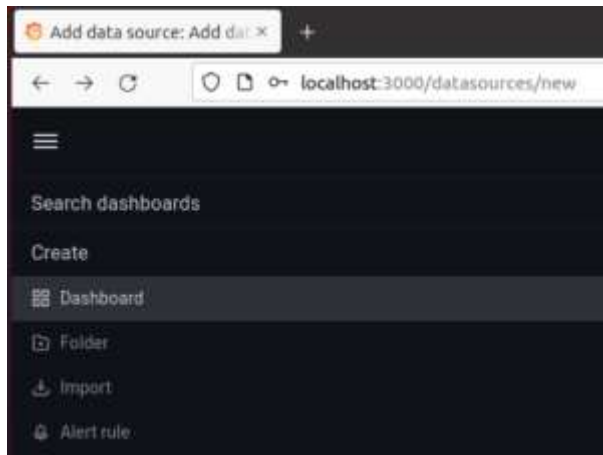
Необходимо вписать правильные реквизиты доступа к удаленному источнику данных для Grafana. В случае с postgresql вам необходимо внести следующие данные:



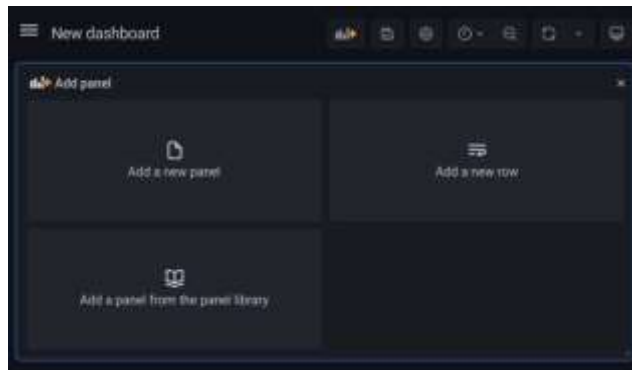
Конечно же вы должны использовать тот IP адрес, на котором установили postgresql, а вот номер порта для подключения к postgresql как правило стандартный, зарезервированный за postgresql, он имеет значение 5432. Также надо ввести правильное имя пользователя и его пароль и название базы данных, из которой вы хотите читать информацию. После этого можно проверить соединение, нажав клавишу “Save & test” (сохранить и протестировать), и если все в порядке, то вы увидите сообщение “Database Connection OK”.



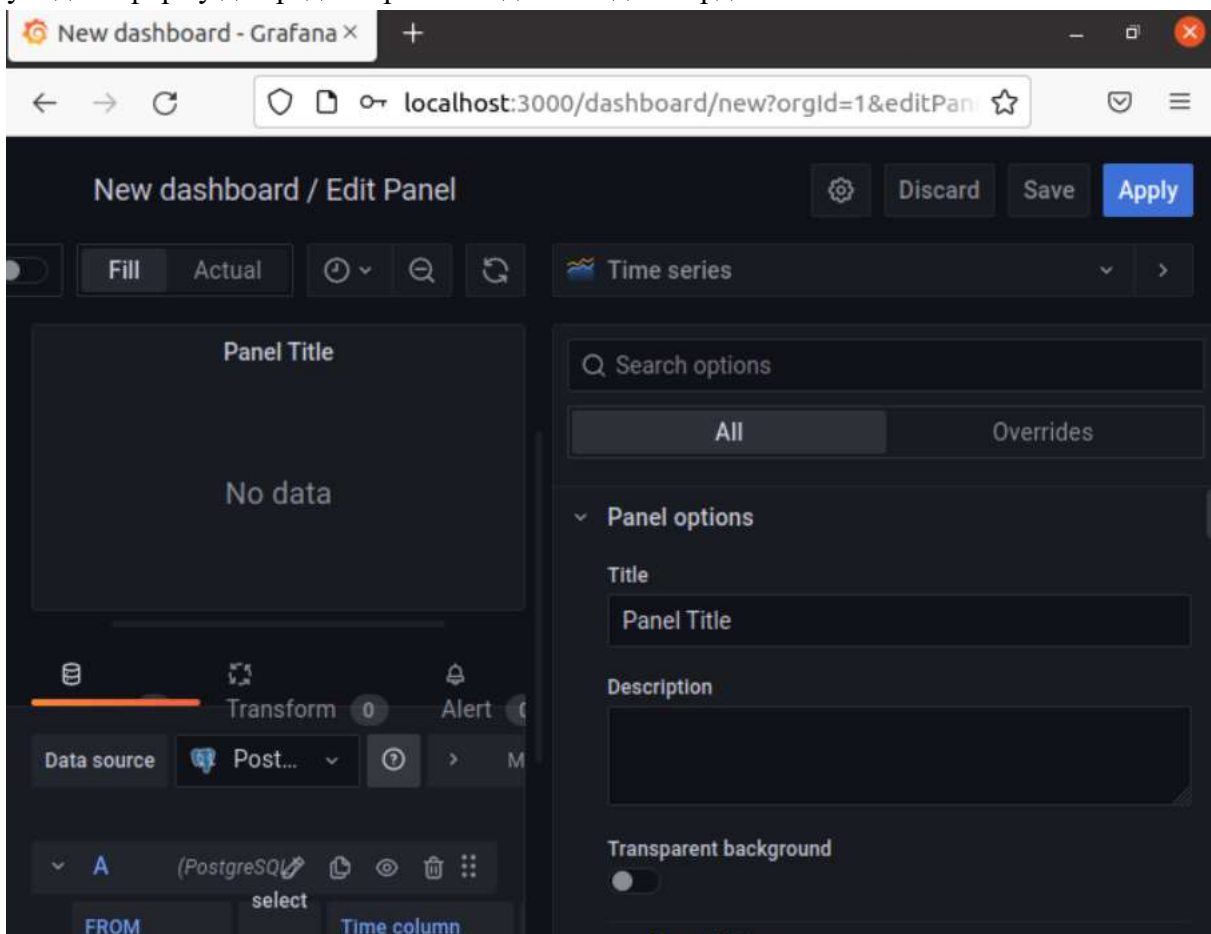
Теперь можно заняться созданием дашборда, который будет отображать информацию, полученную из базы данных. Для этого в Grafana надо воспользоваться вкладкой “Create Dashboards” (создать дашборд)



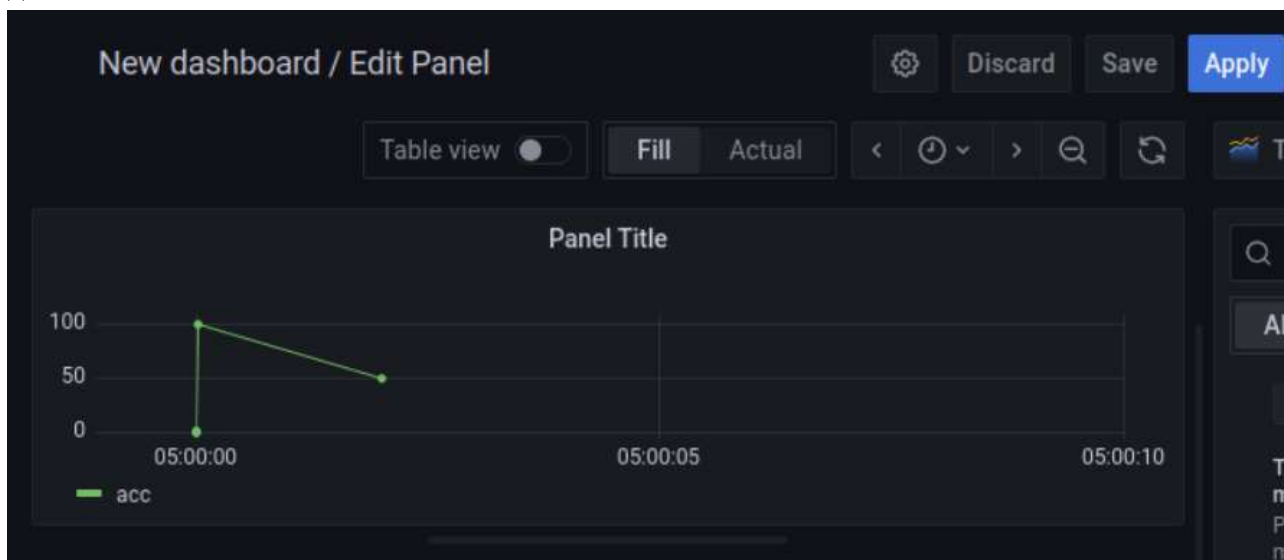
в предложенном интерфейсе будут предложены несколько опций для создания графиков



давайте создадим новую панель, нажав на клавишу “Add a new panel”, после чего вы увидите форму для редактирования данных дашборда.



Наиболее важным разделом является “Data source” (источник данных), в котором вы задаете способ получения данных для отображения. Вам необходимо указать столбец базы данных, который играет роль временной метки, это будет абсцисса графиков. Это обязательно, так как grafana отображает информацию в привязке ко времени. Кроме этого необходимо указать столбец который будет отображаться. После этого необходимо нажать на кнопку “Apply” (применить) и вы увидите на дашборде значения величины, сохраненной в базу данных



У этого дашборда можно менять названия, подписи, цвет и тип линии и многое другое. Это делается через простой и понятный интерфейс, предлагаем вам самостоятельно поэкспериментировать с параметрами. Также вы можете установить период обновления запросов в базу данных, из которой берутся данные для отображения. Установив этот таймер, например, на 5 секунд, и внеся изменения в базу данных, вы увидите эти изменения на дашборде через 5 секунд.

Тест:

1. Для решения каких задач может использоваться Grafana? (0.25)
 - a. обучение моделей машинного обучения
 - b. визуализация процессов**
 - c. мониторинг и контроль процессов
 - d. отправка оповещения о срабатывании триггера**
2. Какой порт по умолчанию используется в web-интерфейсе grafana? (0.25)
 - a. 1000
 - b. 2000
 - c. 3000**
 - d. 4000
3. С какими источниками данных может быть состыкована Grafana? (0.25)
 - a. реляционная база postgresql**
 - b. база данных временных рядов terminusDB**
 - c. реляционная база данных mysql
 - d. prometheus**
4. Какие сущности используются в grafana для визуализации? (0.25)
 - a. панель**

- b. линейка
- c. дашборд
- d. аннотация**

Итоги/выводы

В этом юните вы узнали про инструмент Grafana и для организации мониторинга и контроля. Вы научились его устанавливать, проводить настройку и создавать простые дашборды для визуализации информации. В следующем юните мы применим полученные знания для проекта машинного обучения.

Модуль 8. Юнит 4. Практический пример использования Grafana для проекта машинного обучения.

Введение:

В предыдущем юните вы познакомились с установкой и настройкой Grafana, а также с созданием простейших интерактивных дашбордов. В этом юните вы примените полученные знания для использования в проекте машинного обучения.

Содержание юнита:

Как видите, настройка Grafana для работы очень проста. Применение для мониторинга работы системы заключается в том, что исполняемый скрипт или программа пишет в специальную базу данных информацию о важных параметрах процесса, а специалист эксплуатации наблюдает за этими параметрами на дашборде, либо может установить правило срабатывания аварийного сигнала (Alert) при превышении некоторого порогового значения. Этот же подход используется в проектах машинного обучения. В различных процессах, которые входят в конвейер машинного обучения, исполняемый скрипт или главная программа записывают в базу данных важную для мониторинга и контроля информацию, а Grafana читает эту информацию и отображает на специально подготовленных дашбордах.

В комплексных проектах, состоящих из множества отдельных компонентов, Grafana может запускаться в виде микросервиса в общей инфраструктуре программной системы. Вот один из примеров рабочей архитектуры проекта машинного обучения, включающего модуль Grafana для мониторинга и визуализации

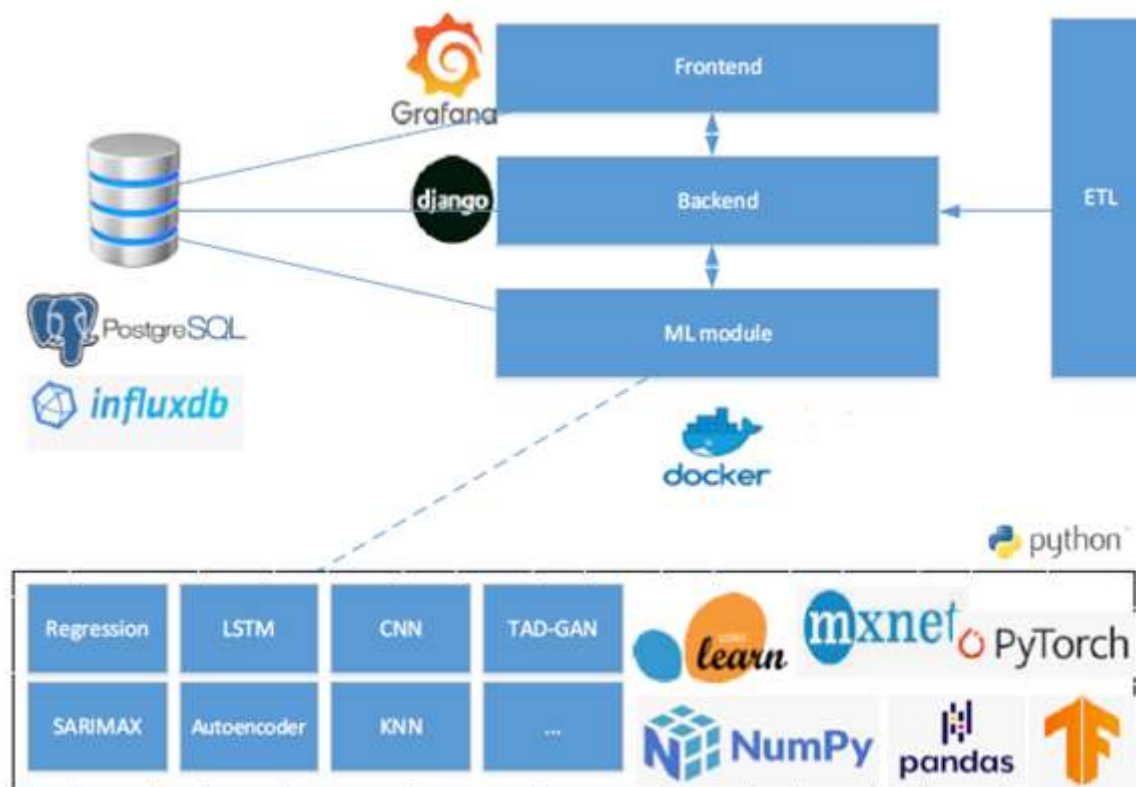


Рисунок “Архитектура проекта машинного обучения с использованием Grafana”

В этом проекте в виде отдельных микросервисов с помощью контейнеров docker реализованы:

- подсистема сбора и обработки данных (ETL)
- backend подсистема на django
- базы данных (например, PostgreSQL и InfluxDB)
- микросервис модели машинного обучения с необходимыми библиотеками
- подсистема frontend, включая визуализацию с Grafana

backend на django организует работу всех остальных микросервисов, обеспечивает выполнение модели машинного обучения, сохранение в базу данных для последующего отображения этих данных в Grafana. Для управления такими многоконтейнерными приложениями обычно используют docker-compose или kubernetes.

С созданием многоконтейнерных приложений вы познакомитесь в следующих модулях курса, а сейчас давайте реализуем запись параметров обучения модели на простом примере в базу данных PostgreSQL и ее отображение в Grafana.

Для подключения к PostgreSQL в python используется библиотека psycopg2. С ее помощью вы можете присоединиться к базе данных и записывать в нее необходимую информацию. Давайте используем базу данных, созданную в предыдущем юните, чтобы для каждого этапа обучения модели записать timestamp, accuracy и validation_accuracy

```
Ввод [1]: import psycopg2
from tensorflow.keras.datasets.mnist import load_data
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import datetime

Ввод [2]: (X_train, y_train), (X_test, y_test) = load_data()

X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

Ввод [3]: conn = psycopg2.connect(dbname='grafana', user='postgres', password='postgres', host='192.168.147.3')
cursor = conn.cursor()

model = MLPClassifier()

for _ in range(10):
    idxs = np.random.choice(len(X_train), size=10000)
    model.fit(X_train[idxs,:], y_train[idxs])
    ts = datetime.datetime.now().timestamp()
    acc = model.score(X_train[idxs,:], y_train[idxs])
    val_acc = model.score(X_test, y_test)
    cursor.execute('INSERT INTO grafana (ts, acc, val_acc) VALUES ({}., {}, {})'.format(ts, acc, val_acc))
    conn.commit()

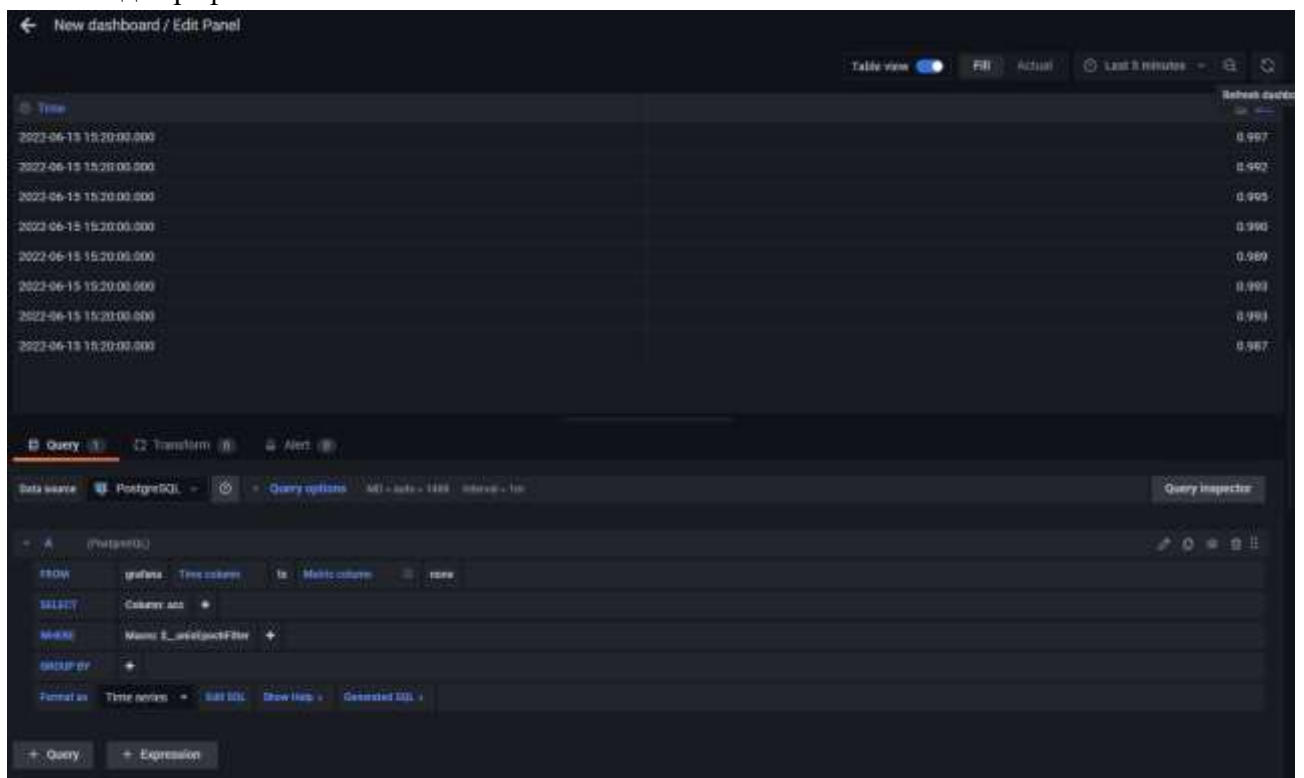
cursor.close()
conn.close()
```

Ввод []:

Выполнение этого кода приводит к тому, что в базе данных появляются строки в таблице

```
grafana=# SELECT * FROM grafana;
      ts      | acc  | val_acc
-----+-----+-----
 1.6552884e+09 | 0.9969 | 0.9229
 1.6552884e+09 | 0.9924 | 0.9238
 1.6552884e+09 | 0.9952 | 0.9286
 1.6552884e+09 | 0.9899 | 0.9181
 1.6552884e+09 | 0.9887 | 0.9237
 1.6552884e+09 | 0.9929 | 0.9272
 1.6552884e+09 | 0.9927 | 0.9293
 1.6552884e+09 | 0.9872 | 0.9143
 1.6552884e+09 | 0.99   | 0.9109
 1.6552884e+09 | 0.9914 | 0.9166
(10 rows)
```

Соответственно эта информация считывается в Grafana из базы данных в табличном виде или в виде графика



Практические задание:

Выполните описанные в юните шаги самостоятельно.

Шаг 1: создайте виртуальную машину для PostgreSQL, создайте базу данных grafana, с использованием сведений из предыдущего юнита.

Шаг 2: создайте модель машинного обучения в jupyter ноутбуке

Шаг 3: создайте привязку к базе данных способом, изложенным в этом юните

Шаг 4: Запустите сервер Grafana с использованием информации из предыдущего юнита, сделайте привязку к таблице grafana базы данных PostgreSQL, созданной на шаге 1.

Шаг 5: Проведите обучение модели с записью accuracy и validation_accuracy в базу данных.

Шаг 6: Проверьте, что в дашборде Grafana сохраненная информация нормально визуализируется.

В качестве результата задания приложите скриншот с дашбордом с визуализацией данных PostgreSQL.

Итоги/выводы:

В этом юните мы создали простейшую модель машинного обучения, провели ее обучение с записью параметров в базу данных и последующей их визуализацией с использованием Grafana.

В качестве результата задания приложите скриншот основного меню дженкинс с визуализацией успешного выполнения jobs (задачи).

Итоги/выводы по модулю

В этом модуле, который завершает первый семестр обучения по данному курсу, были рассмотрены задачи и инструменты эксплуатации. Выход программной системы в эксплуатацию является важным закономерным этапом разработки и развития продукта. На этапе эксплуатации проверяется качество работы всей команды, правильность реализации алгоритмов и точность выполнения технического задания. От успешной эксплуатации зависит успех проекта целиком. На этом этапе выявляются ошибки, новые идеи.

Для проекта машинного обучения эксплуатация имеет особенное значение, поскольку это своего рода самое главное тестирование качества работы модели, на реальных данных, в реальных производственных условиях. Нередко самые перспективные гипотезы и удачные реализации моделей машинного обучения оказываются несостоятельными перед реальными данными или производственными условиями. Поэтому к процессу эксплуатации модели машинного обучения необходим такой же подход, что и в любом другом проекте разработки программного обеспечения.

Для эффективной эксплуатации придуманы и используются на практике различные организационные и технологические инструменты. В этом модуле мы сосредоточились на средствах мониторинга и контроля процессов в проекте машинного обучения. Сначала мы изучили консольные команды `linux`, поскольку это самая популярная операционная система для реализации проектов ML. Затем рассмотрели средство мониторинга `Tensorboard`, который позволяет осуществлять мониторинг и контроль параметров модели во время обучения. И, в последнем юните, мы рассмотрели более универсальный инструмент, фреймворк `Grafana`, который используется не только в проектах ML, но и вообще является популярным инструментом визуализации в проектах разработки программного обеспечения. `Grafana` обладает полезными функциями:

- имеет множество коннекторов для взаимодействия с различными источниками данных
- позволяет создавать наглядные дашборды, управлять параметрами визуализации
- имеет инструменты для контроля превышения пороговых значений
- позволяет отправлять уведомления при возникновении внештатных ситуаций.

Кроме рассмотренных в модуле инструментов существуют и другие средства для мониторинга и контроля. Обычно конкретное средство подбирается под требования проекта. Для небольших проектов достаточно консольных команд и небольших скриптов для автоматизации. В более сложных проектах не обойтись без таких инструментов как `Grafana`. Но и при использовании `Grafana` умение напрямую работать с системой через командную строку оказывается очень эффективным во многих ситуациях.

Полученные знания позволят вам самостоятельно спланировать и настроить систему эксплуатации для вашего проекта.

Полезные ссылки

Ссылки на полезные книги по linux

<https://tproger.ru/books/linux/>

Описание инструмента Tensorboard на официальном сайте

<https://www.tensorflow.org/tensorboard>

Работа с SQL в PostgreSQL

<https://www.digitalocean.com/community/tutorials/introduction-to-queries-postgresql>