• В этом семестре — зачёт, в следующем — экзамен; занятия по практике проходят дистанционно, нужна стабильная связь и среда разработки (VS Code/PyCharm либо Google Colab).

• Базовый язык — Python (стандарт курса); альтернативные языки теоретически возможны, но сдача подразумевает Python.

• Docker: зачем нужен, архитектура (daemon/CLI/registry), базовые сущности (image/container), ключевые команды (pull/images/rmi/ps/run/stop/rm). Dockerfile — сборка образа; Compose — описание и запуск многоконтейнерных приложений. Для оптимизации — лёгкие образы (alpine/distroless), объединение RUN, чистка кеша.

• API: что это такое и как классифицируются — по доступу (открытые/внутренние/партнёрские) и по способу взаимодействия (REST, SOAP, GraphQL, WebSocket, gRPC). Примеры применений: Firebase (аутентификация/данные), платёжные API, IoT, TensorFlow Serving, биржевые API.

• FastAPI: быстрый Python‑фреймворк для API с автогенерацией документации (OpenAPI/Swagger), высокой производительностью и опорой на type hints. Практическая ценность — быстро «оборачивать» ваш код/модель в веб‑сервис для внутренних пользователей.

Полный переписанный текст лекции

Организационные моменты и формат курса

Занятия по практике проводятся дистанционно: вы получаете ссылку и подключаетесь в указанные дни. Для работы нужна стабильная связь и установленная среда разработки. Если локально установить VS Code или PyCharm по каким‑то причинам не получается, подойдёт Google Colab: там можно выполнять задания прямо в браузере.

Текущий учебный год разбит так: в этом семестре зачёт, в следующем — экзамен. Чтобы получить «автомат», важно вовремя закрывать лабораторные: первые работы несложные и рассчитаны на быстрое выполнение. Задания будут выложены в электронный университет; о сроках и деталях вы узнаете на ближайшей паре. Посещение практик желательно для всех — по ним деканат следит особо.

Почему Python

Образовательный стандарт предполагает старт именно с Python как наиболее простого языка для входа. Да, многие задачи можно решить и на других языках, но сдача и проверка завязаны на Python, поэтому в рамках дисциплины используем его. Дальше по программе будет теория по самому Python и базовые конструкции, необходимые для выполнения заданий.

Переход к Docker: зачем он нужен

Docker — это платформа для разработки, доставки и запуска контейнеризированных приложений. Она позволяет упаковать приложение и его зависимости в контейнер, который можно запустить практически в любом окружении без изменения кода. Главные выгоды: портативность, изоляция и, как следствие, более предсказуемое использование ресурсов и повышение отказоустойчивости. Современные высоконагруженные системы, где одновременно работают многие пользователи, практически везде опираются на контейнерный подход.

Архитектура и базовые сущности

Docker использует клиент‑серверную архитектуру: демон (docker daemon) управляет контейнерами, а Docker CLI — это клиентская утилита, через которую мы взаимодействуем с демоном. Registry (например, Docker Hub или GitHub Container Registry) — это хранилище образов, откуда мы «тянем» готовые сборки.

Образ (image) — неизменяемый шаблон, из которого запускаются контейнеры. Контейнер — это запущенный экземпляр образа. Установка Docker несложна как на рабочую станцию (Docker Desktop), так и на сервер/ВМ; подробные инструкции всегда есть в официальной документации.

Минимально необходимые команды

• docker pull — загрузить образ из реестра (например, базу данных, веб‑сервер и т. п.).

• docker images — показать локальные образы.

• docker rmi — удалить ненужный образ.

• docker ps — показать запущенные контейнеры.

• docker run — запустить контейнер из образа (например, NGINX с пробросом портов, если требуется).

• docker stop — остановить контейнер; docker rm — удалить контейнер.

Команд достаточно, чтобы увидеть, что запущено, получить новые образы, удалить лишнее и управлять жизненным циклом контейнеров.

Dockerfile: как описывается образ

Dockerfile — это сценарий сборки образа. Типичный пример для Python‑приложения: базовый образ python:3.9, установка рабочей директории, копирование кода, установка зависимостей из requirements.txt и команда запуска (CMD/ENTRYPOINT). Жизненный цикл привычный: сначала docker build, затем docker run. Такой подход делает сборку воспроизводимой и удобной для переноса между средами.

Docker Compose: многоконтейнерные приложения

Compose позволяет декларативно описывать несколько сервисов одного приложения в одном YAML‑файле: например, web, app и db. Можно определить зависимости (depends\_on), пробросить порты, задать environment‑переменные (вроде POSTGRES\_USER и POSTGRES\_PASSWORD) и сетевые связи между сервисами. Запуск и управление происходят через docker compose up/down. Прелесть подхода — устойчивость: если один из сервисов временно «падает», остальные продолжают работать, а вы чините проблему точечно.

Оптимизация контейнеров

Для уменьшения размера и ускорения поставки используют лёгкие базовые образы (alpine, distroless), объединяют команды RUN, чтобы уменьшить количество слоёв, и при необходимости чистят кеш и временные файлы на этапе сборки. Для учебных и небольших проектов это не всегда критично, но хорошая привычка формируется сразу.

API: что это и какие бывают

API — интерфейс взаимодействия программных систем: набор правил/протоколов, по которым одно приложение запрашивает данные или функции другого и получает ответы.

Классификация по доступу:

• Открытые (public): доступны всем разработчикам (пример: Google Maps, OpenWeather).

• Внутренние (private): используются только внутри компании (например, для доступа к данным клиентов).

• Партнёрские: по соглашениям для ограниченного круга партнёров и крупных клиентов.

Классификация по способу взаимодействия:

• REST — поверх HTTP с методами GET/POST/PATCH/DELETE; прост в использовании, доминирует в веб‑разработке.

• SOAP — обмен по XML и более строгим протоколам; часто встречается в корпоративных системах и интеграциях с «тяжёлыми» форматами.

• GraphQL — клиент запрашивает ровно те поля, что ему нужны; гибко, но сложнее в реализации.

• WebSocket — двусторонняя связь в реальном времени (чаты, онлайн‑игры, стриминг событий).

• gRPC — бинарный протокол, очень быстрый и экономичный, популярен для высоконагруженных бэкендов.

Где применяются API: веб‑приложения (соцсети, хранилища файлов), взаимодействие с ОС (WinAPI), библиотечные интерфейсы (модули/SDK), аппаратные интерфейсы (например, графические). На практике немало полезных API коммерческие; полностью бесплатные решения под конкретные кейсы встречаются реже.

Примеры использования API

• Мобильные приложения: Firebase для аутентификации и хранения данных — не нужно писать всё с нуля, используете готовый сервис.

• Веб‑платежи: интеграция платёжных систем через их API (например, PayPal) — безопасность и маршрутизация транзакций берёт на себя провайдер.

• IoT/умный дом: Google Home API и подобные позволяют управлять устройствами и получать телеметрию.

• Машинное обучение: TensorFlow Serving — развёртывание моделей как сервиса, чтобы к ним обращались по сети.

• Финтех: биржевые API — для алгоритмической торговли и аналитики рынка.

FastAPI: быстрые веб‑сервисы на Python

FastAPI — современный фреймворк для создания веб‑API на Python. Его плюсы: высокая производительность, аннотации типов (type hints), из которых автоматически формируется спецификация OpenAPI и документация (Swagger UI), и удобная разработка. Практический сценарий: «обернуть» модель или анализатор в простой веб‑сервис, чтобы «бизнес‑пользователь» мог загрузить данные и получить результат одной кнопкой. При выборе фреймворков полезно смотреть на популярность экосистемы: у массовых инструментов больше примеров, решений типовых проблем и активности сообщества.

Заключение

Дальше по курсу будет плотный блок по Python: разберём синтаксис и базовые конструкции, необходимые для выполнения лабораторных. Подготовьте среду, проверьте доступность интернета и следите за объявлениями о сроках выкладки заданий.