ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ

Государственное бюджетное профессиональное образовательное   
учреждение города Москвы

«Колледж автоматизации и информационных технологий №20»

**ИНДИВИДУАЛЬНЫЙ ПРОЕКТ**

на тему

**«Docker контейнеризация»**

по дисциплине

«Индивидуальный проект»

Обучающийся:

Хаксар М.Н

УО ДАТАХАБ курс 1 группа ИСП131

Руководитель ИП:

Ломаченкова А.И.

**МОСКВА**

**2024**

**Структура проекта:**

**ГЛАВА 1. Теория**

**ВВЕДЕНИЕ**

1. Определение. Актуальность……………………….

2. Цель проекта.

3. Задачи проекта.

4. Гипотеза. Продукт.

5.Целевая аудитория.

6.Объект исследования.

7.Предмет исследования.

**ОСНОВНАЯ ЧАСТЬ**

1.Терминология

2.Теоритическая основа

3.Причины использования

4.Сферы эксплуатации

5.

6.

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ЛИТЕРАТУРЫ**

**ВВЕДЕНИЕ**

**Актуальность**

Docker – это платформа, позволяющая автоматизировать развертывание, масштабирование и управление приложениями с использованием контейнеров.

Контейнеры представляют собой легковесные, изолированные среды (операционные системы), в которых можно запускать приложения и их зависимости. Это значительно упрощает процесс разработки и развертывания, обеспечивая согласованность на разных стадиях жизненного цикла программного обеспечения.

Платформа позволяет быстро тестировать и выкладывать приложения, запускать на одной машине требуемое количество контейнеров, что делает её лидером среди малочисленных аналогов.

По данным опросов, таких как Stack Overflow Developer Survey, около 30-40% разработчиков используют контейнерные технологии в своих проектах, а Docker занимает около 83% на рынке контейнеризации в целом.

По статистике сайта TheirStack.com более 100 000 организаций уже активно используют Docker контейнеризацию в своих сервисах.Среди них: Yandex, eBay, Spotify, Netflix, PayPal. Число этих компаний растёт с каждым днём. Сумма дохода первых десяти организаций в подборке составила более 135 миллиардов долларов.

Сама же организация настолько разрослалсь, что к 2022 году смогла привлечь финансирование в размере более 100 миллионов долларов.

По отчетам: рынок контейнеризации ожидает роста на 25-30% в год; к 2025 году его объем может достигнуть более 10 миллиардов долларов.

Благодаря своей эффективности и отсутствию большой конкуренции на рынке, Docker-контейнеризация является ведущим инструментом на рынке цифровых технологий, что, несомненно, говорит о его высокой актуальности.

**Цель проекта**

Разработать практическое руководство по контейнеризации Python-приложений с использованием Docker.

**Задачи проекта**

1. **Изучить теоретические основы Docker:** ознакомиться с основными понятиями и технологиями Docker, такими как образы (images), контейнеры (containers), Dockerfile, Docker Compose, а также репозитории Docker Hub.

2. **Разработать файл с зависимостями, Dockerignore:** изучить зависимости, нужные для работы приложения и создать файл с этими зависимостями. Создать Dockerignore.

3. **Разработать Dockerfile: создать** Dockerfile для python-приложения, включающий все необходимые зависимости и настройки для его работы в контейнере.

4. **Собрать Docker-образ:** собрать Docker-контейнер с помощью Docker Hub и проверить его работоспособность локально.

5. **Развернуть приложение на веб-сервере:** развернуть собранный Docker-образ на выбранный веб-сервер. Описать процесс развертывания.

6. **Протестировать развернутое приложение:** проверить работоспособность развернутого приложения на веб-сервере, проверить доступность и функциональность.

7. **Вывод: подвести итоги, сделать вывод.**

**Гипотеза, продукт.**

**Гипотеза:**

Проект должен раскрыть достоинства контейнеризации и то, насколько удобным делает этот процесс платформа Docker.

**Продукт:**

* 2 python-приложения.
* Dockerfile, dockerignore, 2 requirements (файлы с базовыми зависимостями, подходящие почти под всех ботов.)
* Документация, включающая всё, что нужно для ознакомления с платформой и работой с ней.
* Краткое руководство по работе с git и github, некоторыми модулями python..

**Целевая аудитория**

* Студенты IT направлений
* Профессиональные разработчики
* Dev-Ops инженеры
* Работники узконаправленных специальностей в IT
* Организации и группы разработчиков

**Объект исследования**

Dev-Ops инженерия.

**Предмет исследования**

Контейнеризация python приложений.

**ГЛАВА 1**

**1.1. Терминология**

Общие термины

* **Развертывание (Деплой)**: Процесс переноса готового приложения или системы из среды разработки в среду, где оно будет доступно конечным пользователям (в основном серверы).
* **Автоматизация**: Использование технологий для выполнения задач без ручного вмешательства.
* **Масштабируемость**: Способность системы справляться с увеличением нагрузки путем добавления ресурсов (масштабирование "вверх") или дополнительных экземпляров системы (масштабирование по горизонтали).
* **Воспроизводимость**: Возможность многократно создавать одинаковое окружение и результаты, что важно для тестирования и развертывания.
* **Конфигурация** **(Configuration)**: Настройка параметров и окружения приложения для его правильной работы.
* **Окружение:** обозначение конкретного набора переменных, настроек, путей, зависимостей и других параметров, которые необходимы приложению для корректной работы в конкретной среде.
* **Репозиторий:** Хранилище, которое может быть удалённым или локальным (GitHub, DockerHub)
* **Среда (Environment):** конкретные условия и ресурсы, необходимые для запуска, тестирования и работы приложения.
* **Веб-сервер:** Программное обеспечение, которое принимает запросы от клиентов (браузеров) и отправляет им веб-страницы или другие данные.
* **Непрерывное развертывание:** автоматизированный подход, при котором каждый коммит автоматически развертывается в производственную среду.
* **Переменные окружения (Environment variables):** Настройки, которые хранятся в операционной системе и могут быть доступны приложениям во время выполнения. Часто используются для хранения конфиденциальных данных (паролей, ключей API и т.д.) и настроек приложения, которые могут меняться в разных средах.
* **Зависимости:** конфигурации, файлы, модули, которые нужны для работы приложения.
* **Продакшн (Production):** конечный продукт.

Docker термины

* **Контейнер:** Изолированная среда, в которой приложение может выполняться.
* **Docker**: Платформа для автоматизации развертывания приложений в контейнерах.
* **Dockerfile**: Текстовый файл, содержащий все команды, которые нужно выполнить для сборки образа Docker.
* **Образ (Image)**: Статический файл, содержащий все необходимые зависимости и конфигурации для запуска контейнера. Образ создается на основании Dockerfile.
* **.dockerignore**: Файл, где перечисляются ненужные для работы контейнера файлы, папки и. т. д.

**Git термины**

* **GitHub**: Платформа для хостинга репозиториев git, которая позволяет управлять версиями кода и сотрудничать с другими разработчиками.
* **Коммит(commit):** Способ сохранения изменений в коде.
* **Push:** Команда, позволяющая отправить коммит в репозиторий.
* **.gitignore:** Файл, где перечисляются ненужные для репозитория файлы, папки и. т. д.

Python термины

* **pip**: Пакетный менеджер для Python, используемый для установки и управления библиотеками.
* **requirements.txt**: Файл, содержащий список зависимостей Python-проекта.

**1.2. Историческая основа**

Виртуальная контейнеризация возникла как ответ на потребности в более эффективном использовании ресурсов вычислительных систем. Её основой стали технологии виртуализации, появившиеся в 1960-х годах. Первые серверы позволяли запускать несколько операционных систем на одном физическом сервере, что значительно увеличивало степень загрузки оборудования.

Позже возникла потребность в более легковесных решениях, и в 6 августа 2008 года на свет появились LXC (Linux Containers) и схожие аналоги.

Docker задумывался как способ работы с контейнерами, вдохновлённый LXC, но гораздо проще и удобнее, дружелюбнее к неподготовленному пользователю. В 2013 году Docker был запущен как open-source проект, что привело к значительному росту сообщества и появлению экосистемы инструментов.

**1.3. Причины использования**

Главная цель Docker — обеспечение универсальности и согласованности работы программного обеспечения, независимо от используемой платформы и специфики окружения. Это достигается посредством технологии контейнеризации, которая изолирует приложение и все его зависимости в единый, самодостаточный блок. Традиционно, разработчики сталкиваются с проблемой "работает на моей машине", когда приложение, успешно функционирующее в среде разработки, может вызывать сбои и конфликты при переносе в тестовую или, что критичнее, в рабочую среду. Эти проблемы обусловлены различиями в операционных системах, версиях установленных библиотек, конфигурациях сетевого окружения и других факторах, которые сложно отследить и воспроизвести в разных окружениях. Docker устраняет этот разрыв, предоставляя стандартизированный способ упаковки приложений, что значительно снижает вероятность возникновения таких проблем. Благодаря контейнеризации можно запускать свои программы практически на любом устройстве, где можно установить Docker Engine, будь то сервер, персональный компьютер, ноутбук или даже роутер. Это открывает новые возможности для развертывания приложений в самых разнообразных условиях, от локальной разработки до облачных вычислений.

Кроме того, использование Docker способствует более эффективному использованию ресурсов, так как контейнеры значительно легче виртуальных машин (ВМ). Контейнеры используют ядро хостовой операционной системы, в то время как ВМ эмулируют целую операционную систему. Это приводит к тому, что контейнеры занимают меньше дискового пространства, потребляют меньше памяти и быстрее запускаются. Контейнеры могут быть запущены на одном сервере в большом количестве, что позволяет повысить плотность размещения приложений и снизить затраты на инфраструктуру. В итоге, Docker становится ключевым инструментом для современных подходов к разработке и развёртыванию приложений, обеспечивая высокую гибкость, масштабируемость, эффективность использования ресурсов и ускоряя процессы разработки, тестирования и эксплуатации.

Из-за вышеперечисленных плюсов, Docker стал незаменимым инструментом в контексте микросервисной архитектуры, где приложение состоит из множества независимых сервисов. Каждый сервис может быть упакован в отдельный контейнер, что позволяет разрабатывать, тестировать и развертывать сервисы независимо друг от друга. Это обеспечивает большую гибкость и устойчивость системы в целом, позволяя командам быстро реагировать на изменения и сбои. Использование Docker Compose и других инструментов управления контейнерами позволяет эффективно управлять множеством взаимосвязанных сервисов и автоматизировать их развертывание и масштабирование.

**1.4. Сферы эксплуатации**

Docker нашел широкое применение в множестве сфер, включая веб-разработку, автоматизацию тестирования, DevOps, микросервисную архитектуру, научные вычисления и машинное обучение (машинное обучение и анализ данных требуют множества зависимостей и библиотек, что делает Docker идеальным вариантом).

В веб-разработке Docker особенно ценен за возможность создания сред, максимально приближенных к боевым условиям. Разработчики могут использовать Docker для создания контейнеров, в которых будут установлены те же самые версии операционной системы, библиотек и других зависимостей, что и на Продакшн-сервере. Это исключает конфликты версий и гарантирует, что приложение будет работать предсказуемо в любой среде. Docker также способствует созданию более согласованной среды разработки, когда все участники команды используют одинаковые конфигурации.

DevOps-инженеры (специалисты, которые объединяют разработку программного обеспечения и администрирование с целью повышения эффективности и качества разработки, развертывания и эксплуатации приложений) используют Docker для упрощения развертывания приложений и управления их жизненным циклом. Docker автоматизирует процесс упаковки, распространения и запуска приложений, что позволяет командам быстро и эффективно вносить изменения и релизы. Docker также упрощает процесс управления инфраструктурой, предоставляя стандартный способ определения и запуска приложений, что позволяет быстрее масштабировать ресурсы в зависимости от текущей нагрузки.

Тестировщики могут использовать Docker для создания воспроизводимых сред, в которых они могут проводить автоматическое и ручное тестирование, зная, что приложение будет работать в предсказуемых условиях, а разработчики могут использовать Docker для создания индивидуальных контейнеров для разработки новых функций, исправления ошибок или проведения экспериментов.

**Теоретические основы. Компоненты.**

\*\*\*\*Для начала изучения Docker, важно рассмотреть основы контейнеризации.

1. Контейнеры представляют собой изолированные среды, где каждое приложение работает в своей ОС со всеми необходимыми зависимостями.
2. Изоляция обеспечивает отсутствие конфликтов даже при запуске несовместимых систем на одном хосте, при условии правильной настройки\*\*\*\* Возможно вырезать

Экосистема Docker состоит из системных и переменных типов компонентов:

Системные компоненты включают в себя: Docker host (сервер Docker), Docker daemon (демон Docker), Docker client (клиент Docker) и Docker-compose (менеджер запуска кластера контейнеров).

Docker host - это просто компьютер или виртуальный сервер, на котором установлен Docker. Docker daemon - это центральный системный компонент, который управляет всеми процессами Docker: создание образов, запуск и остановка контейнеров, скачивание образов. Docker client - это утилита, предоставляющая API к демону Docker.

Основные переменные компоненты включают в себя: Dockerfile (файл Docker), Docker image (образ Docker), Docker container (контейнер Docker).

Файл Dockerfile состоит из серии инструкций, которые выполняются в определенном порядке. Некоторые из наиболее распространенных инструкций включают FROM (используется для указания базового образа), RUN (для выполнения команд в контейнере), COPY (для копирования файлов в контейнер), EXPOSE (для указания портов, которые должны быть открыты для внешнего доступа) и ENTRYPOINT/CMD (для определения команды, которая будет запущена при старте контейнера).  
При создании нового образа Docker, Dockerfile используется для создания нового контейнера на основе базового образа, который затем может быть дополнительно настроен и оптимизирован для конкретного приложения или задачи. После создания образа Docker, он может быть сохранен в репозитории Docker Hub или другом частном репозитории для последующего использования.  
Пример DockerFile

**СПИСОК ЛИТЕРАТУРЫ**

**Книги**

### *Парминдер Сингх Кочер*, Микросервисы и контейнеры Docker, ДМК Пресс, год, 237

### *Моут Эдриен*, Использование Docker, ДМК Пресс, год, 346

### *Иан Милл,Эйдан Хобсон Сейерс*; Docker на практике, ДМК Пресс, год, 511

### ~~ ОСТАЛЬНЫЕ КНИГИ НА АНГЛИЙСКОМ!!!

### Сайты

### atlex.ru, <https://www.atlex.ru/baza-znanij/tematicheskij-glossarij/kontejnery-v-it/> датаобр

### datadoghq.com, <https://www.datadoghq.com/container-report/>

### yandex.cloud.ru, <https://yandex.cloud/ru/blog/posts/2022/03/docker-containers>

### habr.com, <https://habr.com/ru/companies/flant/articles/326784/>

### habr.com, <https://habr.com/ru/articles/659049/>

### habr.com, <https://habr.com/ru/companies/yandex_praktikum/articles/760076/>

### Нумерация страниц, многоточие(необ), поменять название ссылок на статийные интервалы, размер шрифта? Метки типа 1.1, 1.2