Министерство науки и высшего образования
Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования «Рыбинский государственный
авиационный технический университет имени П. А. Соловьева»

ИНСТИТУТ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ
УПРАВЛЕНИЯ»

Кафедра вычислительных систем

МЕТОДИЧЕСКОЕ ПОСОБИЕ
по дисциплине:
«Вычислительные системы»
для лабораторной работы на тему:
«Основы работы с *Docker*»

Составили: студенты группы ИВМ-24
Морозов А. А.
Смирнов Д. И.

Рыбинск 2024

1. Введение

Современные вычислительные системы становятся все более сложными. Рост объемов данных, необходимость быстрого масштабирования и требования к надежности и отказоустойчивости вынуждают разработчиков и администраторов искать новые подходы к разработке и эксплуатации программного обеспечения. Одним из таких подходов является контейнеризация, которая позволяет создавать изолированные среды для приложений, что облегчает их переносимость, развертывание и управление.

Контейнеризация приобрела популярность благодаря своей легковесности и эффективности по сравнению с виртуализацией. Контейнеры используют общий системный ресурс (операционную систему), что делает их более производительными, чем виртуальные машины. Одним из самых известных инструментов для контейнеризации является *Docker*, который упрощает создание, упаковку и развертывание контейнеров.

2. Цель работы

1. Изучить основы работы с *Docker*.
2. Освоить процесс создания и управления *Docker*-контейнерами.
3. Научиться создавать образы с использованием *Dockerfile* или скачать готовый с *Docker Hub*.
4. Изучить основные команды для взаимодействия с контейнерами и образами в *Docker*.

3. Подготовка к работе

1. Скачайте и установите *Docker* с официального сайта (*https://www.docker.com/*).
2. Запустите *Docker*.

4. Теоретические сведения

В работе с *Docker* самым важным является работа с контейнером. Контейнер представляет собой изолированную среду, в которой запускается приложение, полностью отделённое от системы хоста и других контейнеров. Контейнеры используют общие ресурсы операционной системы, что делает их лёгкими и производительными по сравнению с виртуальными машинами.

Контейнер – это работающий экземпляр образа *Docker*, который включает в себя всё необходимое для выполнения приложения: код, зависимости, библиотеки и системные настройки. Однако контейнеры отличаются от виртуальных машин тем, что не включают отдельную операционную систему. Вместо этого они используют ядро хостовой операционной системы, минимизируя накладные расходы. Если ядро системы на сервере и ядро в контейнере будут различаться, то *Docker* будет использовать технологию *Windows Subsystem for Linux* (*WSL*) для успешной работы контейнера. Данная технология интерпретирует системные вызовы *Linux* и преобразует их в эквиваленты *Windows*.

Образ *Docker* – это неизменяемый файл, содержащий всё необходимое для запуска приложения внутри контейнера. Образ можно рассматривать как шаблон или основу, из которой создаются контейнеры. Он включает операционную систему, приложения, библиотеки, зависимости и любые конфигурации, необходимые для корректной работы программного обеспечения (рисунок 1). В качестве операционной системы можно использовать *alpine*, объём которой составляет лишь 5 МБ. После создания образа его можно загрузить в репозиторий для хранения и обмена. Наиболее популярным является *Docker Hub* – центральный репозиторий *Docker*, где можно найти образы для различных операционных систем, баз данных, веб-серверов и других инструментов. Репозитории могут быть как и публичными, так и приватными.

Рисунок 1 – Образ *Docker*

Для создания образа *Docker* программисту необходимо написать так называемый *Dockerfile*. В нём используется концепция слоёв, каждый из которых начинается со специфичных инструкций (*FROM, RUN, COPY, ADD*). Каждый новый слой содержит только новую информацию и *Docker* кэширует каждый слой, что позволяет экономить место, время для сборки образов и развёртывание контейнеров.

Dockerfile – это текстовый файл, содержащий последовательность инструкций, которые *Docker* использует для создания образа. Он позволяет автоматизировать процесс сборки образа *Docker*, задавая необходимые параметры, такие как базовый образ, команды для установки программ, копирование файлов и настройка окружения.

5. Практическая часть

Задание: создать и запустить контейнер *Docker*.

6. Ход работы

1. Для создания контейнера понадобится образ *Docker*. Образ можно либо создать, либо скачать с *Docker Hub*:

- для создания образа *Docker* необходимо создать *Dockerfile*. В нём прописывается всё необходимое для успешного создания образа.

- с *Docker Hub* можно скачать готовый образ для создания контейнера.

2. Запустите созданный из *Dockerfile* или скачанный с *Docker Hub* контейнер.

В качестве вариантов для создания контейнера можно рассмотреть следующие задачи:

1) Простой скрипт на *Python*. Написать *Dockerfile* для программы на Python, которая выводит текст на экран. При запуске контейнера программа должна выполнить скрипт и завершиться;

2) Веб-сервер на базе *Flask*

Создать веб-приложение с использованием Python и библиотеки *Flask*, которое отвечает "Hello, World!" при обращении к серверу. *Dockerfile* должен устанавливаться все зависимости и запускать приложение внутри контейнера.

3) Сайт на *HTML*

Разместить статический *HTML*-сайт и собрать образ, который развертывает веб-сервер (например, *Nginx*) для обслуживания этого сайта. Это подойдёт для демонстрации использования контейнеров с веб-серверами.

4) Программа на *C++* или другом языке. Написать консольную программу на *C++* (или другом языке, например *Java*), собрать её внутри контейнера, а затем запустить. Это продемонстрирует, как *Docker* может использоваться для компиляции программ в изолированном окружении.

5) База данных с сохранением данных в том. Создать и запустить контейнер, разворачивающий базу данных (например, *MySQL* или *PostgreSQL*). Подключить том для сохранения данных и показать, как данные сохраняются при удалении контейнера.

6) Оптимизация размера образа. Создать два *Dockerfile* для одного и того же приложения. Один образ использовать с полноценным базовым образом (например, *python*:3.9), а другой – с лёгким (например, *python*:3.9-*alpine*). Сравнить размеры образов и время их сборки.

7) Взаимодействие между контейнерами. Создать *Docker Compose*-конфигурацию, запускающую два контейнера: один с бекендом (например, *Flask*-приложением), а другой с фронтендом (например, веб-сервером *Nginx*). Реализовать взаимодействие между ними, чтобы фронтенд отправлял запросы на бекенд.

7. Отчёт по лабораторной работе.

Отчет по лабораторной работе содержит:

Цели работы: перечислите цели, которые были поставлены перед началом работы.

Ход работы: опишите, что было сделано в каждом задании.

Результаты и выводы: Приведите результаты вашей работы по созданию и запуску контейнера *Docker* и сделайте выводы о различии между контейнером и виртуальной машиной. Добавьте скриншоты.

7. Контрольные вопросы

1. Что такое *Docker*?
2. Какие задачи решает технология контейнеризации?
3. Для чего используется файл *Dockerfile*?
3. Что такое *Docker*-образ?
4. Чем отличается *Docker*-образ от *Docker*-контейнера?
5. Какой командой можно создать *Docker*-образ?
6. Какой командой можно запустить контейнер?
7. Какой командой можно запустить контейнер в фоновом режиме?
8. Как просмотреть список запущенных контейнеров?

9. Что происходит при выполнении команды *docker build -t* имя_образа .?

10. Как остановить работающий контейнер?

11. Как удалить Docker-контейнер?

12. Как проверить, создан ли образ после выполнения команды *docker build*?

13. Какой командой можно удалить образ *Docker*?

14. Что делает команда *docker ps -a*?

15. Можно ли использовать *Docker* для запуска приложения, написанного на языке, отличном от *Python*?