Курс DevOps

Лабораторная работа №3

Docker

1 апреля 2026 г.

Общие инструкции

Рекомендации к выполнению

- Настоятельно рекомендуется выполнять команды из листинга отдельно, одну за другой.
- Рекомендуется копировать команды через промежуточную вставку в текстовый редактор во избежание лишних символов.
- Перед началом работы ознакомтесь с инструкцией по сдаче лабораторных работ.

Требования к выполняемой работе

- Добавьте в репозиторий все необходимые файлы: код приложения, файлы конфигурации и скрипты и т.д.

Внимание: Данная лабораторная работа является продолжением лабораторной работы №2. Для выполнения данной работы необходимо иметь подготовленный репозиторий из второй лабораторной работы.

1 Цель работы

Получить базовые навыки работы с Docker, научиться упаковывать приложение в контейнер и интегрировать его сборку в процесс CI/CD.

2 Теоретические сведения

2.1 Что такое Docker

Docker — это платформа для разработки, доставки и запуска приложений в изолированных средах, называемых контейнерами. В отличие от виртуальных машин, контейнеры не эмулируют целую операционную систему, а используют ядро хостовой ОС, что делает их лёгкими и быстрыми в запуске.

2.2 Основные понятия

- **Образ (Image)** — шаблон для запуска контейнера. Содержит файловую систему и инструкции для запуска приложения.
- **Контейнер (Container)** — экземпляр запущенного образа. Является изолированной средой с приложением внутри.
- **Dockerfile** — текстовый файл с инструкциями для сборки образа.
- **Registry** — хранилище образов (например, DockerHub, GitHub Container Registry, GitLab Container Registry).

2.3 Жизненный цикл контейнера

Основные шаги при работе с Docker:

1. Создание Dockerfile с описанием зависимостей и шагов сборки.
2. Сборка образа с помощью команды docker build. Как правило, это происходит на CI сервере.
3. Публикация образа в registry. Registry в данном случае играет роль посредника между сборкой и развертыванием: в одном окружении образ создаётся и отправляется в хранилище, а в другом окружении он загружается для запуска.[1]
4. Запуск контейнера из образа с помощью команды docker run на целевой машине (например, в тестовой или продуктивной среде).

[1] В ряде случаев сборка и развертывание могут выполняться на одной и той же машине, однако это не является рекомендуемой практикой по ряду причин.

2.4 Основные инструкции Dockerfile

Ниже приведены основные инструкции, используемые при создании Dockerfile. Эти команды достаточно знать для выполнения лабораторной работы:

- FROM — задаёт базовый образ.
- RUN — выполняет команды при сборке образа.
- WORKDIR — устанавливает рабочую директорию внутри контейнера.
- COPY, ADD — копируют файлы в образ.
- EXPOSE — указывает порты, которые контейнер открывает для связи снаружи.
- CMD, ENTRYPOINT — определяют команду, выполняемую при запуске контейнера.

Для более подробного описания всех инструкций и их опций можно обратиться к официальной документации Docker.

3 Содержание работы

В данной работе необходимо контейнеризировать приложение из предыдущей лабораторной работы с помощью Docker. Также необходимо встроить сборку контейнера в CI пайплайн, а в CD пайплайне перейти к развертыванию приложения через Docker контейнер. Опционально, в CI пайплайн необходимо встроить линтер для Dockerfile — hadolint.

3.1 Задание 1

Необходимо установить Docker на виртуальную машину. Официальный туториал от docker.

3.2 Задание 2

Создать Dockerfile для приложения из предыдущей лабораторной работы. Поместите полученный Dockerfile в репозиторий с исходным кодом.

1. Определите базовый образ с помощью инструкции FROM. Используйте произвольный базовый образ.
2. Установите необходимые зависимости с помощью RUN.
3. Установите рабочую директорию (WORKDIR) и скопируйте файлы приложения (COPY).
4. Настройте команду запуска приложения с помощью CMD или ENTRYPOINT.
5. Определите порты, которые будут использоваться приложением (EXPOSE).

3.3 Задание 3

Собрать и проверить локально Docker образ.

1. Соберите образ и запустите контейнер с пробросом порта на хост машину.
2. Попробуйте отправить полученный контейнер в GitHub Docker Registry с вашей виртуальной машины
3. Убедитесь, что приложение работает и доступно на localhost на указанном порту.

3.4 Задание 4

Встроить сборку Docker образа в CI пайплайн.

1. Добавьте шаг сборки Docker образа в существующий CI workflow.
2. Настройте тегирование образа и его загрузку в GitHub Docker Registry
3. Убедитесь, что при push в репозиторий CI workflow собирает образ и загружает его
4. Опционально: интегрируйте линтер Hadolint для проверки Dockerfile. Не завершать пайплайн в случае найденных ошибок с помощью continue-on-error.

Полезная информация:

- Туториал по сборке контейнера в GitHub Actions от GitHub
- Работа с GitHub Docker Registry из GitHub Actions
- Ещё один гайд по работе с Docker Registry из GitHub Actions
- Полезные GitHub Actions:
  - docker/login-action
  - docker/build-push-action
  - docker/metadata-action

3.5 Задание 5

Настроить CD пайплайн для развертывания приложения через Docker контейнер.

1. Добавьте шаг развертывания контейнера в CD workflow.
2. Обеспечьте автоматический деплой, используя Docker контейнер собранный в CI пайплайне.
3. Настройте перезапуск контейнера при обновлении образа и протестируйте развертывание.

4. Убедитесь, что приложение обновляется и доступно по тому же адресу: http://app.$ID.$PROXY

Примечание:

- Учтите, что все необходимые зависимости приложения должны добавлены в image. На виртуальной машине допускается только установка Docker.
- Если ваше приложение имеет персистентный слой (например, базу данных), то при пересоздании контейнера вы потеряете все данные. В текущей работе это допустимое упрощение, в следующей работе мы исправим это поведение.
- В данной работе в контейнер допустимо складывать все зависимости тестов. В промышленной разработке это не является рекомендуемой практикой.
- Если ваше приложение имеет интеграционные тесты (тесты, которые требуют запущенное приложение — E2E, UI, API), в данной работе допустимо не запускать их. Для желающих можно настроить их запуск через service containers или через запуск тестов после делпоя новой версии на работающем приложении.

4 Контрольные вопросы

1. Что такое Docker и в чём отличие контейнера от виртуальной машины?
2. Что такое Dockerfile? Какие основные инструкции вы знаете?
3. Объясните жизненный цикл контейнера: от написания Dockerfile до запуска контейнера.
4. Зачем использовать registry при работе с Docker и CI/CD?
5. Как можно получить сетевой доступ к приложению, которое запущено в контейнере, снаружи контейнера?
6. В каких случаях сборка и развертывание на одном окружении может быть допустима, и почему это не является рекомендуемой практикой?
7. Почему хранение тестов и их зависимостей внутри контейнера не является рекомендуемой практикой?
8. Как скачать имедж контейнера из GitHub Container Registry и отправить его в DockerHub?
9. Как можно переместить имедж с одного сервера (например, CI сервера, где происходит сборка) на другой сервер, где происходит развертывание, без использования registry?
10. Опишите ситуацию, когда кеширование сборки имеджа может оказаться нежелательным?

11. Вашему приложению необходимо использовать секрет, например, для подключения к внешнему API. Какие есть подходы к решению этой задачи при запуске приложения в контейнере?

12. В чем разница между инструкциями CMD и ENTRYPOINT в Dockerfile?

13. Как посмотреть сколько оперативной памяти и процессорного времени использует контейнер?

14. Как ограничить контейнер в потреблении оперативной памяти и процессорно времени? Что произойдет с контейнером, при достижении указанных лимитов?