МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра автоматизированные системы управления

ЛАБОРАТОРНАЯ РАБОТА № 6

по дисциплине: «ДПО Интаро Софт»
на тему: «Контейнеризация».

Студент	АС-23		Кирнос Б. В.
	группа	подпись, дата	фамилия, инициалы

Руководитель

к.т.н., доцент кафедры АСУ		Кургасов В.В.
ученая степень, ученое звание	подпись, дата	фамилия, инициалы

Липецк 2025

Задание:

1. С помощью Docker Compose на своем компьютере поднять сборку nginx+phpfpm+postgres, продемонстрировать ее работоспособность, запустив внутри контейнера демо-проект на symfony (Исходники взять отсюда https://github.com/symfony/demo /ссылка на github/. По умолчанию проект работает с sqlite-базой. Нужно заменить ее на postgres. (Для этого: 1. Создать новую БД в postgres;

2. Заменить DATABASE_URL в .env на строку подключения к postgres;

3. Создать схему БД и заполнить ее данными из фикстур, выполнив в консоли (php bin/console doctrine:schema:create php bin/console doctrine:fixtures:load)). Проект должен открываться по адресу http://demo-symfony.local/ (Код проекта должен располагаться в папке на локальном хосте) контейнеры с fpm и nginx должны его подхватывать. Для компонентов nginx, fpm есть готовые docker-образы, их можно и нужно использовать.

Нужно расшарить папки с локального хоста, настроить подключение к БД. В .env переменных для постгреса нужно указать путь к папке, где будет лежать база, чтобы она не удалялась при остановке контейнера.

На выходе должен получиться файл конфигурации docker-compose.yml и env файл с настройками переменных окружения.

Дополнительные требования: Postgres также должен работать внутри контейнера. В .env переменных нужно указать путь к папке на локальном хосте, где будут лежать файлы БД, чтобы она не удалялась при остановке контейнера.

Ход работы

Подготовка окружения

Сначала устанавливаем нужные утилиты для работы с репозиториями и HTTPS (сертификаты, curl, GPG и утилиты определения версии дистрибутива):

sudo apt install ca-certificates curl gnupg lsb-release -y

Создаём каталог для хранения ключей apt и сохраняем в него GPG-ключ Docker (мы скачиваем ключ и конвертируем его в бинарный формат gpg):

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

Регистрируем официальный репозиторий Docker в списках apt. Команда подставляет текущую архитектуру машины и кодовое имя версии Ubuntu, а также указывает использовать сохранённый ключ для проверки пакетов:

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Обновляем индекс пакетов и устанавливаем Docker Engine, клиент, контейнерную систему и плагин Compose:

sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y

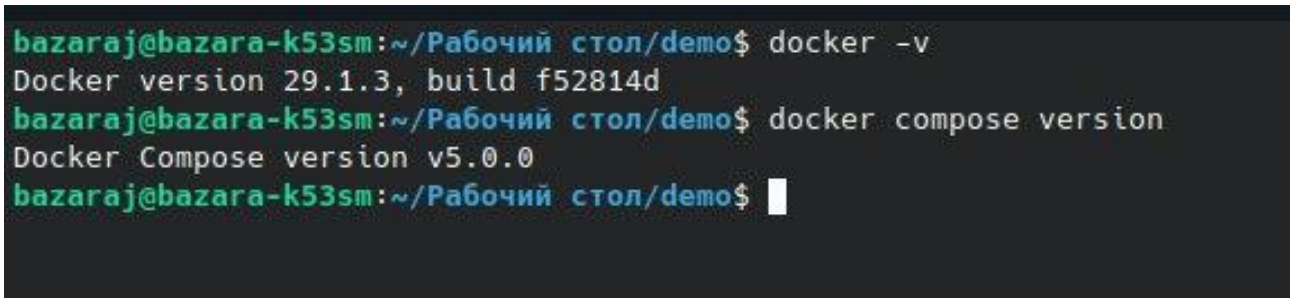

Рис. 1 – Установка docker и docker compose

Установка symphony

Для этого клонирую себе на машину репозиторий демо-приложения

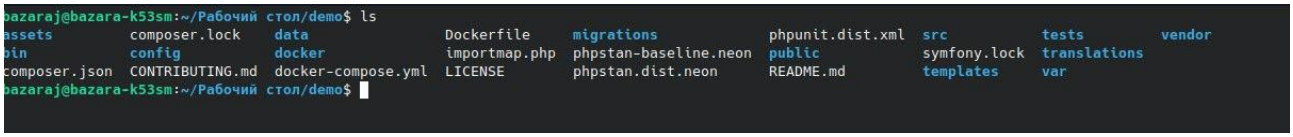

Рис. 2 – Каталог приложения

Далее с помощью composer install установим необходимые зависимости. Запускать приложение будем через symphony cli, установим его с помощью curl -sS https://get.symfony.com/cli/installer | bash.
Стоит отметить, что по умолчанию тут используется sqlite, на этапе работы с докером перенесем на postgres.

Запустим наше приложение с помощью symfony server:start.

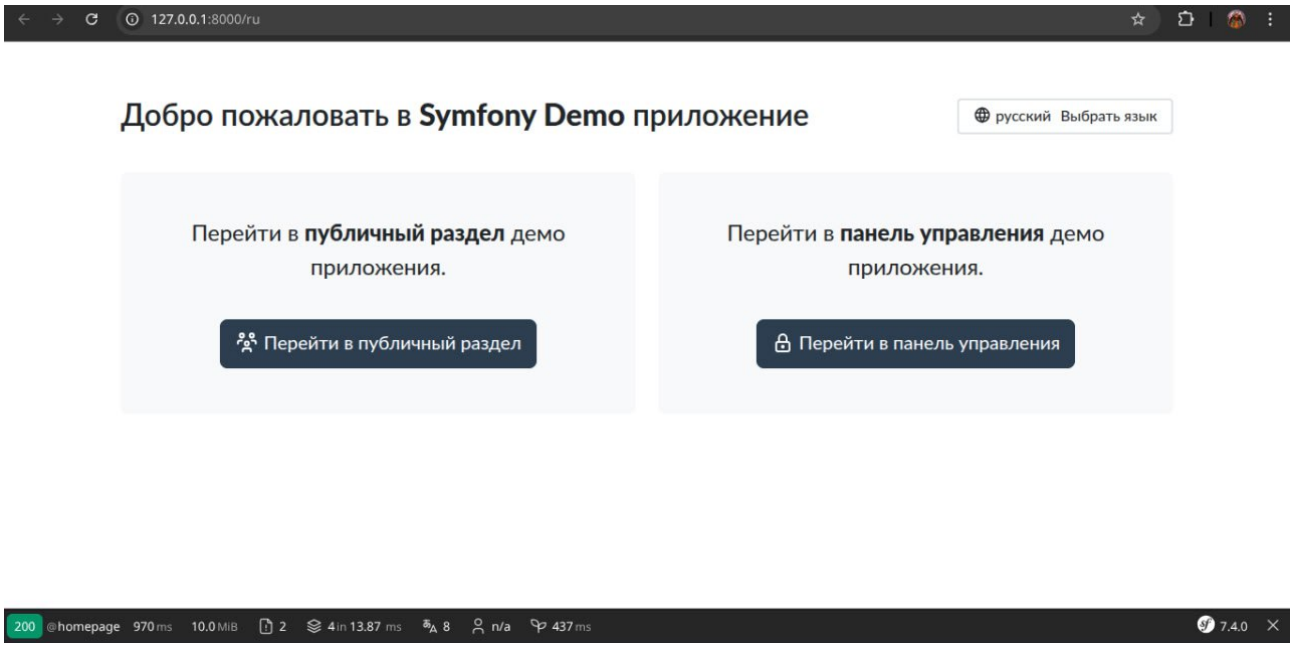

Рис. 3 – Приложение (Не в контейнере)

Успешный запуск приложения говорит о правильной установке и настройке symphony.

Работа с docker

Сперва создадим Dockerfile

```
FROM webdevops/php-nginx:8.2-alpine

# Устанавливаем расширения PostgreSQL
RUN apk add --no-cache postgresql-dev \
    && docker-php-ext-install pdo pdo_pgsql pgsql

# Устанавливаем дополнительные расширения PHP
RUN apk add --no-cache \
    libpng-dev \
    libjpeg-turbo-dev \
    freetype-dev \
    libzip-dev \
    libxml2-dev \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install \
        gd \
        zip \
        xml \
        intl \
        bcmath

# Устанавливаем Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /app

# Копируем зависимости и устанавливаем
COPY composer.json composer.lock ./
RUN composer install --prefer-dist --no-scripts --no-dev --no-autoloader

# Копируем весь проект
COPY . .

# Генерируем автозагрузчик
RUN composer dump-autoload --optimize

# Настраиваем права
RUN chown -R application:application /app/var

# Создаем симлинк для public директории
RUN ln -sf /app/public /app/html
```

```
# Настройка окружения
ENV WEB_DOCUMENT_ROOT=/app/public
ENV PHP_DATE_TIMEZONE=Europe/Moscow
ENV PHP_DISPLAY_ERRORS=1

EXPOSE 80
```

Используется готовый образ на базе Alpine Linux с уже установленными PHP 8.2, Nginx и базовой конфигурацией для их совместной работы. Это упрощает настройку веб-сервера и экономит время.

Устанавливаются заголовочные файлы PostgreSQL (postgresql-dev), необходимые для сборки расширений.
Подключаются PHP-расширения:

- pdo — универсальный слой доступа к БД,
- pdo_pgsql — поддержка PostgreSQL через PDO,
- pgsql — нативное расширение PostgreSQL.

Дополнительные расширения PHP:

- Устанавливаются системные библиотеки, необходимые для сборки PHP-расширений.
- Настраивается и устанавливается gd с поддержкой JPEG и FreeType (работа с изображениями).
- Также добавляются:
    - zip — работа с архивами,
    - xml — обработка XML,
    - intl — интернационализация,
    - bcmath — высокоточные математические вычисления.

В рамках данной работы такие расширения избыточны, но я установил их с задумкой возможного расширения приложения.

Далее создадим docker-compose.yml

```
services:
  app:
    build: .
    container_name: symfony-app
    ports:
      - "8000:80"
    volumes:
      - .:/app
      - ./docker/nginx.conf:/etc/nginx/conf.d/default.conf
    environment:
      APP_ENV: dev
      APP_SECRET: ${APP_SECRET:-ThisTokenIsNotSoSecretChangeIt}
      DATABASE_URL:
postgresql://symfony:ChangeMe@db:5432/symfony?serverVersion=15&charset=utf
8
    depends_on:
      - db
    networks:
      - symfony-network
    restart: unless-stopped

  db:
    image: postgres:15-alpine
    container_name: symfony-postgres
    environment:
      POSTGRES_DB: symfony
      POSTGRES_USER: symfony
      POSTGRES_PASSWORD: ChangeMe
      POSTGRES_HOST_AUTH_METHOD: trust  # для разработки
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./docker/postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5433:5432"
    networks:
      - symfony-network
    restart: unless-stopped

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: symfony-phpmyadmin
    depends_on:
      - db
```

```
    environment:
      PMA_HOST: db
      PMA_PORT: 5432
      PMA_ARBITRARY: 1
    ports:
      - "8081:80"
    networks:
      - symfony-network
    restart: unless-stopped

networks:
  symfony-network:
    driver: bridge

volumes:
  postgres_data:
```

Установка phpMyAdmin не обязательна и на работоспособность приложения не влияет.

Чтобы удостовериться что всё корректно установлено и написано выполним команду с рисунка 4

Рис. 4 – Контейнеры docker`a

Далее последовала сборка и запуск контейнеров:

docker compose up -d --build

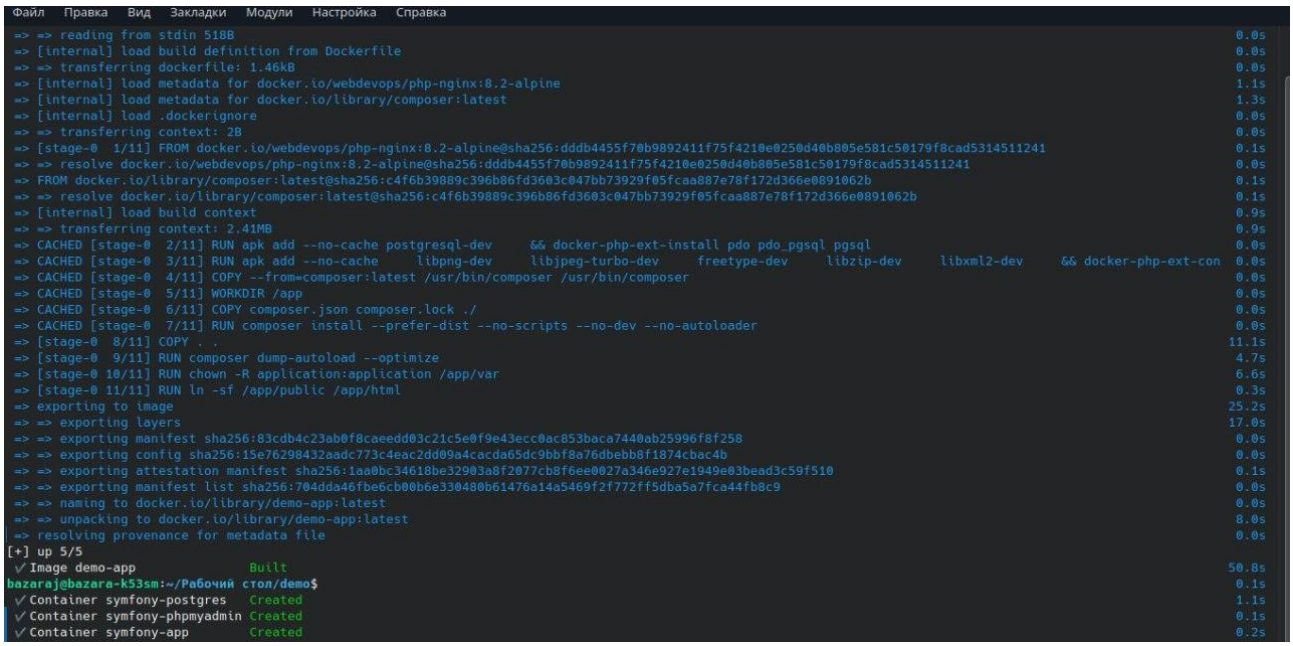

Рис. 5 – Результат сборки контейнеров

Наше приложение уже запущено по адресу, указанному в yml файле (localhost:8000), однако, при попытке взаимодействия с интерфейсом получаем такую ошибку:

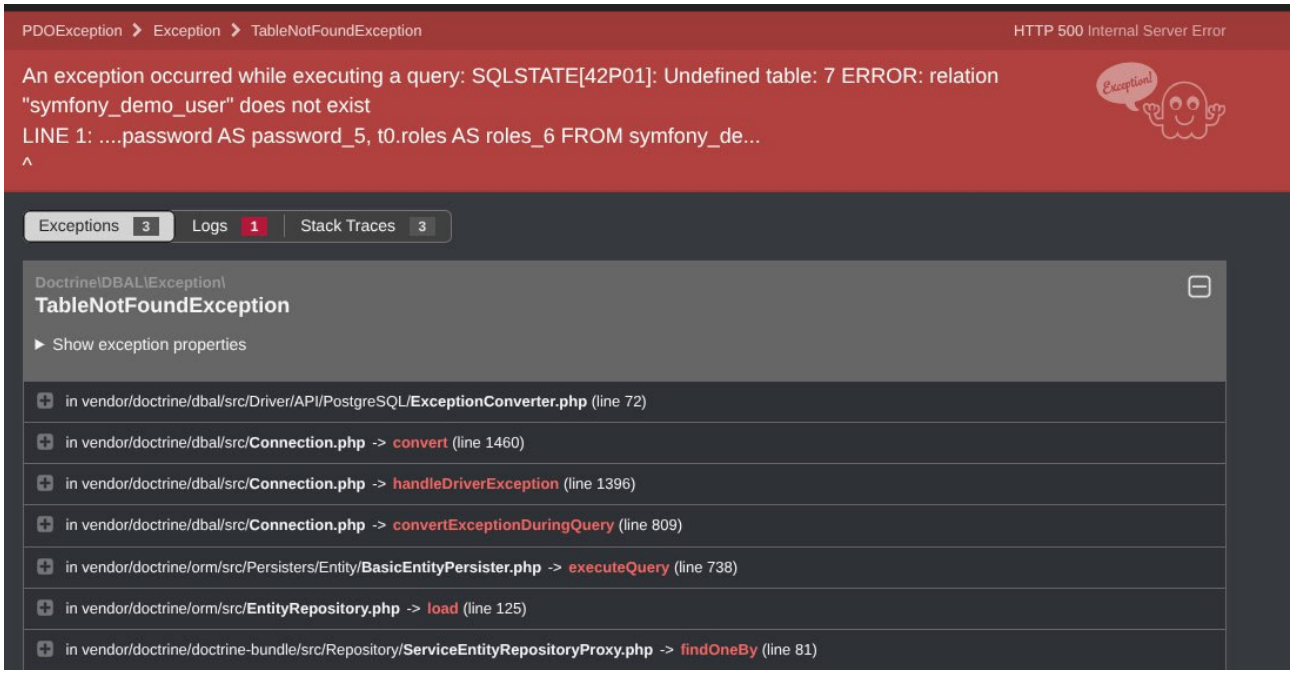

Рис. 6 – Ошибка БД

Для исправления нам необходимо установить и инициализировать зависимости Symfony непосредственно в PHP-контейнере:

1. Подключаемся к запущенному PHP-контейнеру:

sudo docker exec -it symfony_php bash

2. Устанавливаем зависимости проекта с помощью Composer:
composer install

3. Создаём структуру базы данных на основе описанных сущностей Doctrine:
php bin/console doctrine:schema:create

4. Загружаем тестовые (начальные) данные в базу:
php bin/console doctrine:fixtures:load

В результате контейнер получает все необходимые PHP-библиотеки, база данных инициализируется, а приложение становится готовым к работе.
Остановим контейнеры с помощью docker composer down, а затем запустим заново.

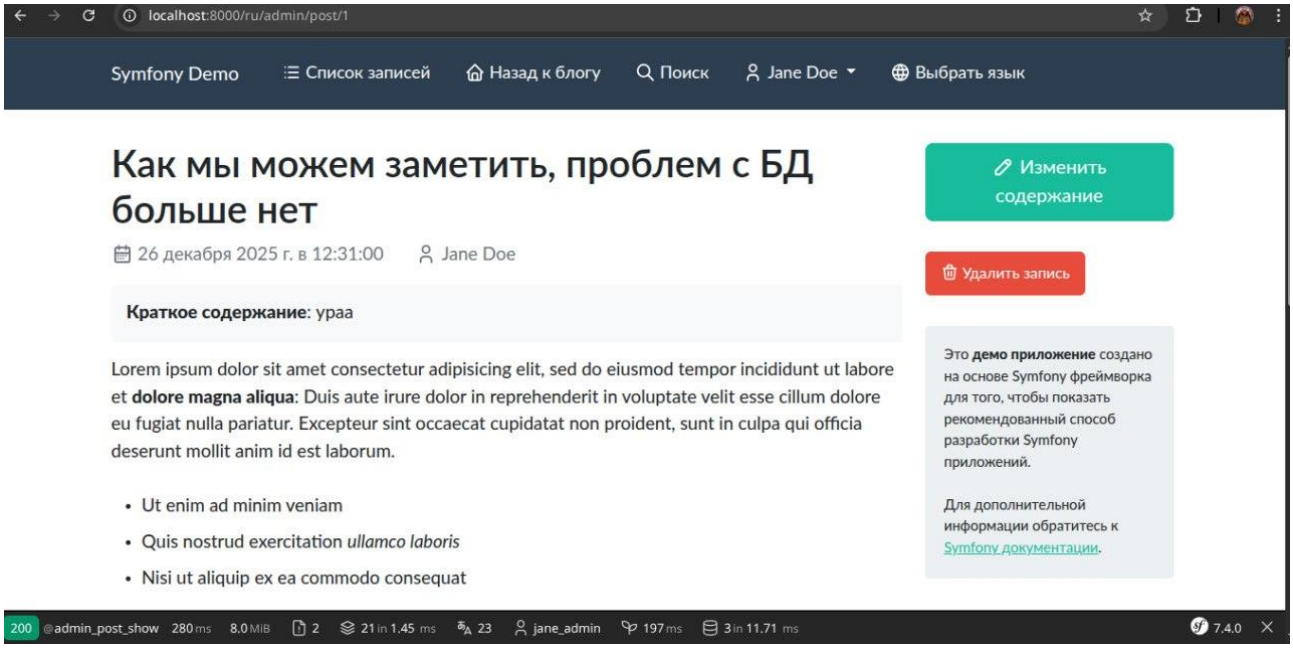

Рис. 7 – Итоговое приложение

На рисунке 7 мы авторизовались под администратором и изменили запись, что демонстрирует полную работоспособность приложения.

Вывод

Мной была изучена базовая процедура развёртывания Symfony-проекта, я ознакомился с современными методами разработки ПО в динамических и распределенных средах на примере контейнеров Docker.

Контрольные вопросы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией.

A, C

2. Назовите основные компоненты Docker.

B, D.

3. Какие технологии используются для работы с контейнерами?

A, C.

4. Найдите соответствие между компонентом и его описанием:

- Образы — доступные только для чтения шаблоны приложений.
- Контейнеры — изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения.
- Реестры (репозитории) — сетевые хранилища образов.

5. В чем отличие контейнеров от виртуализации?

Виртуальные машины запускают полноценную гостевую ОС поверх гипервизора (больше ресурсов, сильнее изолированы, медленнее старт), контейнеры разделяют ядро хоста, легче и быстрее, но изоляция слабее.

6. Перечислите основные команды утилиты Docker с их кратким описанием

docker build — сборка Docker-образа из Dockerfile.
docker pull — загрузка образа из репозитория (Docker Hub или приватного).
docker push — отправка локального образа в репозиторий.
docker run — создание и запуск нового контейнера.
docker ps — список запущенных контейнеров (-a — все).
docker start — запуск уже созданного контейнера.
docker stop — корректная остановка контейнера.
docker restart — перезапуск контейнера.
docker exec — выполнение команды внутри работающего контейнера.
docker logs — просмотр логов контейнера.
docker rm — удаление контейнера.
docker rmi — удаление Docker-образа.
docker images — список локальных образов.
docker network — управление сетями Docker.
docker volume — управление томами (volume).
docker-compose up — запуск нескольких контейнеров по docker-compose.yml.
docker-compose down — остановка и удаление контейнеров.

7. Поиск образов контейнеров

Через registry (Docker Hub и приватные registry) — команда docker search <name> или веб-интерфейс; затем docker pull <image:tag> для загрузки.

8. Запуск контейнера

Через docker run с опциями (-d, -p, --name, -v, -e) либо docker start для ранее созданного контейнера; для мультиконтейнерных стеков (приложение в этой работе) — docker-compose up.

9. Что значит управлять состоянием контейнеров?

Это комплекс действий по их жизненному
циклу: создание, запуск (start), остановка (stop/kill), перезапуск, удаление (rm), а также мониторинг, масштабирование, обновление и обеспечение безопасности контейнеров и их образов, чтобы поддерживать приложение в нужном рабочем состоянии

10. Как изолировать контейнер?

Использовать namespaces, cgroups, user namespaces, seccomp, AppArmor/SELinux, drop capabilities, read-only файловые системы, отдельные сетевые пространства и rootless режимы.

11. Последовательность создания образов и назначение Dockerfile

Dockerfile — рецепт сборки (FROM, RUN, COPY, ENV, EXPOSE, CMD/ENTRYPOINT и т.д.). Сборка: написать Dockerfile → docker build -t name:tag . → тестирование → docker push в registry.

12. Можно ли работать с контейнерами без Docker Engine?

Да: альтернативы — containerd, CRI-O, runc, podman (daemonless). Docker Engine не обязателен.

13. Назначение Kubernetes и основные объекты

Kubernetes автоматизирует развёртывание, масштабирование и управление контейнерными приложениями. Основные объекты: Pod, Deployment, ReplicaSet, Service, StatefulSet, DaemonSet, Job/CronJob, ConfigMap, Secret, PersistentVolume/PersistentVolumeClaim, Namespace, Ingress.