СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компания Evrone является международным поставщиком IT-услуг, специализирующимся на разработке программного обеспечения, дизайне и техническом консалтинге. Основанная в 2008 году, компания успешно реализует сложные проекты для клиентов в таких отраслях, как финансы, медицина, ритейл и стартапы. Специализируется на реализации проектов с использованием таких технологий, как Ruby on Rails. Evrone предлагает решения в области веб-разработки, мобильных приложений, DevOps и блокчейн-технологий, демонстрируя высокий уровень экспертизы и инновационного подхода. Компания активно поддерживает open-source-сообщество, проводит конференции и способствует развитию профессиональных сообществ разработчиков.

В период с 1 по 28 июля я проходил практику в компании «ООО Эвроне.ру». Во время практики я изучал методологию DevOps, создавал инфраструктуру для проекта аналитики данных профилей ВК.

Поставленные цели и задачи:

1) Изучить методологию DevOps, а также используемые инструменты;

2) Контейнеризировать приложение;

3) Создать CI пайплайн для приложения с помощью Github Actions;

4) Настроить выкладку приложения с использованием фреймворка Kamal;

5) Научиться искать и использовать информацию для разработки ПО.

Содержание и оформление отчета осуществлялось с учетом требований ГОСТ 7.32.2017.

1 Изучение методологии DevOps и используемых инструментов

Методология DevOps представляет собой набор практик и инструментов, направленных на объединение процессов разработки программного обеспечения (Development) и его эксплуатации (Operations). Основной целью DevOps является повышение эффективности работы и улучшение качества выпускаемых продуктов за счет автоматизации, непрерывной интеграции (CI) и непрерывной доставки (CD).

Ключевые элементы DevOps включают: автоматизацию рабочих процессов, использование контейнеризации для управления средами разработки и развертывания. Важной частью методологии является построение культуры сотрудничества между разработчиками и операционными командами для быстрого и качественного внедрения изменений в продукт.

Одним из используемых инструментов является Docker. Docker — это открытая платформа для разработки, доставки и эксплуатации приложений с использованием контейнеризации. Она позволяет изолировать приложение и все его зависимости в отдельной среде, что гарантирует его стабильную работу в любой инфраструктуре. Основу Docker составляет Docker Engine — движок, который управляет созданием и запуском контейнеров. Образы Docker представляют собой неизменяемые шаблоны, содержащие операционную систему, программные библиотеки, зависимости и само приложение. Эти образы создаются с использованием файлов Dockerfile, где прописаны инструкции по сборке контейнера.

Контейнеры Docker — это изолированные среды, созданные на основе образов, которые запускают приложения с минимальными затратами на ресурсы за счет использования общей операционной системы хоста. Управление контейнерами и их взаимодействие осуществляется через интерфейс командной строки Docker CLI.

Для хранения данных контейнеров Docker поддерживает систему томов и монтирование директорий с хоста, данное решение позволяет сохранять данные между перезапусками контейнеров.

Помимо Docker используется также его подпрограмма: Docker Compose — это инструмент для управления многоконтейнерными приложениями, позволяющий описывать архитектуру приложения, состоящего из нескольких контейнеров, в декларативном формате через файл docker-compose.yml. В подобных файлах описывается конфигурация сервисов: используемые образы, порты, переменные окружения, тома и сети.

Таким образом Docker и Docker Compose в совокупности обеспечивают удобную и стандартизированную платформу для разработки, тестирования и развертывания приложений, значительно упрощая процессы их управления.

GitHub Actions — это платформа для автоматизации рабочих процессов, которая интегрирована в GitHub и предоставляет возможность выполнять CI/CD-пайплайны прямо в репозиториях. Она позволяет автоматизировать тестирование, сборку и развертывание приложений при помощи гибкой системы триггеров и правил, запускаемых на основе событий в репозитории, таких как коммиты, pull requests или релизы. GitHub Actions поддерживает широкий спектр языков программирования и операционных систем, а также интеграцию с внешними сервисами и облачными платформами.

Kamal — это современный инструмент для развертывания веб-приложений, обеспечивающий скользящее развертывание с нулевым простоем и упрощенное управление развернутыми сервисами. Инструмент был разработан компанией 37signals как простая и эффективная альтернатива Kubernetes и другим сложным облачным решениям. Kamal использует контейнеры Docker, предлагая независимое от поставщиков решение, которое можно развертывать как в облачной инфраструктуре, так и на физических серверах. Это особенно привлекательно для компаний, которые хотят сохранить контроль над своей инфраструктурой, пользуясь преимуществами контейнеризации.

2 Контейнеризация приложения

Для контейнеризации приложения были написаны Dockerfile[1], Dockerfile.dev (контейнер для разработки) и docker-compose.yml. Также с целью повышения удобства был сделан Makefile, описывающий возможные операции над проектом, покрывающие весь цикл разработки новых компонент сервиса: установка зависимостей, сборка контейнеров, запуск необходимых контейнеров, чистка образов, линтинг, работа с базой данных.

В Dockerfile.dev сборка приложения разделена на несколько этапов: установка системных зависимостей (base), установка зависимостей для фронтенда (node) и финальная сборка проекта (run). С помощью созданного файла Dockerfile.dev разработчик может в автоматическом режиме собирать образ изолированного приложения со всеми зависимостями и необходимыми настройками, что упрощает процесс разработки.

В процессе прохождения практики были проделаны работы по уменьшению размера образа, времени его сборки. Таким образом для увеличения скорости сборки стадии base и node работают параллельно и независимо. Также для уменьшения времени сборки у команд-установщиков зависимостей выставлены специальные флаги, уменьшающие объем используемой памяти и количество необходимого для отработки трафика, выключена прекомпиляция asset'ов для фронтенда (компиляция происходит во время исполнения по запросу тех или иных asset'ов). настроено кеширование этапов сборки для уменьшения повторяющихся вычислений. Для уменьшения общего размера образа в качестве базового слоя используется урезанный дистрибутив linux -- alpine. За счет перечисленных оптимизаций была достигнута максимальная скорость сборки и минимальный размер образа. В листинге 1 представлено описание сборки контейнера.

Листинг 1 – содержание файла Dockerfile.dev

```
ARG RUBY_VERSION=3.2.4
FROM  registry.docker.com/library/ruby:$RUBY_VERSION-alpine  AS
base

WORKDIR /rails
```

Продолжение листинга 1

```
ENV RAILS_ENV="development" \
    BUNDLE_DEPLOYMENT="1" \
    BUNDLE_PATH="/usr/local/bundle" \
    NODE_PATH="/node_modules"

FROM base AS node

# Install node modules
RUN apk add --virtual .build-deps yarn
COPY package.json yarn.lock ./
RUN yarn install --frozen-lockfile --modules-folder $NODE_PATH

FROM base AS run

# Install system dependencies
RUN apk add --no-cache --virtual .build-deps build-base git
postgresql-dev vips-dev tzdata yarn pkgconfig
RUN apk add --no-cache curl vips-dev postgresql-client tzdata
RUN rm -rf /var/lib/apt/lists /var/cache/apt/archives

# Install application gems
COPY Gemfile Gemfile.lock ./
RUN bundle install
RUN    rm    -rf    ~/.bundle/    "${BUNDLE_PATH}"/ruby/*/cache
"${BUNDLE_PATH}"/ruby/*/bundler/gems/*/.git
RUN bundle exec bootsnap precompile --gemfile

# Copy application code
COPY . .

COPY --from=node /node_modules /node_modules
RUN yarn global add nodemon sass postcss-cli --prefix /usr/local

# Precompile bootsnap code for faster boot times
RUN bundle exec bootsnap precompile app/ lib/

# Precompiling assets for production without requiring secret
RAILS_MASTER_KEY
RUN SECRET_KEY_BASE_DUMMY=1 ./bin/rails assets:precompile

RUN adduser -D rails --shell /bin/bash
RUN mkdir -p /usr/local/bundle/ruby/3.2.0/cache
RUN chown -R rails:rails \
    db log storage tmp \
    /usr/local/bundle/ruby/3.2.0/cache
RUN chmod -R 777 /usr/local/bundle/ruby/3.2.0/cache

USER rails:rails
EXPOSE 3000
CMD [ "./bin/dev" ]
```

Поскольку приложение предполагает запуск нескольких контейнеров, то был использован инструмент Docker Compose[2], для его работы был описан docker-compose.yml. Сервис состоит из контейнера с приложением и контейнера с базой данных. В файле настроены зависимости между контейнерами, тома для сохранения данных, переменные окружения с необходимыми для работы секретами, healthcheck'и, проверяющие успешность поднятия приложения. Описание сервисов docker-compose.yml представлено в листинге 2.

Листинг 2 – содержание файла docker-compose.yml

```
services:
  app:
    platform: linux/amd64
    build:
      context: .
      dockerfile: Dockerfile.dev
    env_file:
      - .env.dev
    stdin_open: true
    tty: true
    volumes:
      - .:/rails
    ports:
      - 80:3000
      - 12345:12345
    depends_on:
      - postgres
    command: ash -c "rm -f tmp/pids/server.pid && bin/dev"
    healthcheck:
      test: curl --fail http://localhost:3000/up || exit 1
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 10s

  postgres:
    platform: linux/amd64
    image: postgres:16-alpine
    restart: always
    env_file:
      - .env.dev
    ports:
      - 5432:5432
    volumes:
      - postgres-data:/var/lib/postgresql/data

  redis:
```

Продолжение листинга 2

```
    platform: linux/amd64
    image: redis:7.2-alpine
    ports:
      - 6379:6379
    command: redis-server
    volumes:
      - redis:/data

volumes:
  postgres-data:

  redis:
```

3 Создание CI-пайплайна

В рамках данной платформы GitLab CI[3] необходимо было создать пайплайн проверки приложения при создании запросов на слияние в главную ветку репозитория. Для выполнения поставленного задания был написан файл ci.yml. В данном файле описаны две задачи, выполняющиеся поочередно: build и codestyle. Build осуществляет сборку образа по описанию из docker-compose.yml, после чего выгружает его как артефакт сборки. Вторая задача выгружает образ из артефактов и запускает на нем проверку стиля кода. Если обе задачи успешно выполнились, то запрос на слияние разрешен, иначе слияние запрещено, так как поломает код в целевой ветке. Данная автоматика является дополнительным фактором защиты и позволяет гарантировать работоспособность кода в целевой ветке после влития изменений при условии, что код в ветке также проходит успешно пайплайн. В листинге 3 представлено содержимое файла ci.yml

Листинг 3 – содержание файла ci.yml

```
name: "CI"
on:
  push:
    branches: [ "dev", "main" ]
  pull_request:
    branches: [ "dev", "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Prepare
        run: cp ./.env.example ./.env.dev

      - name: Install ruby
        uses:                                                ruby/setup-
ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
        with:
          ruby-version: '3.2.0'

      - name: Build image
        run: make ci-build
```

Продолжение листинга 3

```
      - name: Save image
        run: docker save --output /tmp/bmstu_2024-app.tar
bmstu_2024-app

      - name: Upload image
        uses: actions/upload-artifact@v4
        with:
          name: bmstu_2024-app
          path: /tmp/bmstu_2024-app.tar

      - name: Clear
        run: make ci-clear

  codestyle:
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Prepare
        run: cp ./.env.example ./.env.dev

      - name: Download image
        uses: actions/download-artifact@v4
        with:
          name: bmstu_2024-app
          path: /tmp

      - name: Load container
        run: docker load --input /tmp/bmstu_2024-app.tar

      - name: Check codestyle
        run: make ci-rubocop
      - name: Clear
        run: make ci-clear
```

4 Настройка инструмента развертывания Kamal

Для настройки развертывания через Kamal[4] необходимо создать файл config/delpoy.yml. Проще всего это можно сделать с помощью команды Kamal setup, при выполнении которой выполняются все необходимые инициализации. После создания файла с стандартным содержимым необходимо подредактировать файл под свой проект.

Одна из проблем, с которая встретилась во время настройки, была связана с особенностями работы используемого регистра образов: для проекта был настроен регистр в Yandex cloud, для аутентификации в данном регистре необходимо получать временный JWT-токен. Алгоритм получения токена был написан отдельно в файле bin/load-token, который вызывался при каждой аутентификации.

Также было необходимо настроить Traefik – инструмент балансировки нагрузки, поставляемый вместе с Kamal. Для корректной работы необходимо настроить автоматическое получение сертификатов на домен, под которым работает сервис (wheremylikes.com): выставить верные данные о домене, а также пробросить доступ из контейнера к файлам конфигурации Letsencrypt – сервиса подписи доменов для поддержки TLS.

В конфигурации Kamal описывается окружение для работы непосредственно сервиса, а также необходимых вспомогательных сервисов (в рамках данного проекта – СУБД PostgreSQL и Redis), используемые секреты, настройки доступа к целевой машине развертывания, а также настройки сети для группы контейнеров.

Итоговое содержимое файла deploy.yml представлено в листинге 4.

Листинг 4 – содержание файла deploy.yml

```
service: wheremylikes

image: wheremylikes

servers:
  web:
    hosts:
      - 104.248.132.229
```

Продолжение листинга 4

```
    labels:
      traefik.http.routers.taska.rule: Host(`wheremylikes.com`)
      traefik.http.routers.taska_secure.entrypoints: websecure
      traefik.http.routers.taska_secure.rule:
Host(`wheremylikes.com`)
      traefik.http.routers.taska_secure.tls: true
      traefik.http.routers.taska_secure.tls.certResolver:
letsencrypt
    options:
      network: "private"

registry:
  server: cr.yandex/crplr3c4ndmeef487upv
  username: iam
  password: <%= %x(bin/load-token) %>

env:
  secret:
    - RAILS_MASTER_KEY
    - HOST
    - POSTGRES_DB
    - POSTGRES_USER
    - POSTGRES_PASSWORD
    - DATABASE_URL
    - REDIS_URL
    - VK_AUTH_REDIRECT_URL
    - VK_APP_ID

ssh:
  user: deploy

builder:
  multiarch: false
  context: .

accessories:
  db:
    image: postgres:16-alpine
    host: 104.248.132.229
    env:
      secret:
        - POSTGRES_DB
        - POSTGRES_USER
        - POSTGRES_PASSWORD
        - DATABASE_URL
    port: 5432
    directories:
      - postgres-data:/var/lib/postgresql/data
    options:
      network: "private"
  redis:
```

Продолжение листинга 4

```
    image: redis:7.2-alpine
    host: 104.248.132.229
    port: 6379
    cmd: "redis-server --appendonly yes --replica-read-only no"
    directories:
      - redis-data:/data
    options:
      network: "private"

traefik:
  options:
    network: "private"
    publish:
      - "443:443"
    volume:
      - "/letsencrypt/acme.json:/letsencrypt/acme.json"
  args:
    entryPoints.web.address: ":80"
    entryPoints.websecure.address: ":443"
    entryPoints.web.http.redirections.entryPoint.to:  websecure
entryPoints.web.http.redirections.entryPoint.scheme: https
    entryPoints.web.http.redirections.entrypoint.permanent:
true
    certificatesResolvers.letsencrypt.acme.email:
"oleg@evrone.com"
    certificatesResolvers.letsencrypt.acme.storage:
"/letsencrypt/acme.json" # Must match the path in `volume`
    certificatesResolvers.letsencrypt.acme.httpchallenge: true

certificatesResolvers.letsencrypt.acme.httpchallenge.entrypoint
: web

asset_path: /rails/public/assets
```

ЗАКЛЮЧЕНИЕ

В ходе практики в компании «Эвроне.ру» были приобретены практические навыки применения методологии DevOps при сопровождении рабочего проекта. Освоены различные инструменты для выполнения задач, такие как Docker, Docker Compose, Github Actions, Kamal.

В рамках практики выполнены следующие задачи:

1) Изучена методология DevOps и сопутствующие инструменты;

2) Выполнена контейнеризация приложения;

3) Настроен CI-пайплайн на платформе Github Actions;

4) Настроен процесс развертывания приложения с использованием инструмента Kamal.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Документация Docker [Электронный ресурс]. URL: https://docs.docker.com/ (08.07.2024).

2 Документация Docker Compose [Электронный ресурс]. URL: https://docs.docker.com/compose/ (08.07.2024).

3 Документация Github Actions [Электронный ресурс]. URL: https://docs.github.com/en/actions (дата обращения 10.07.2024).

4 Документация Kamal [Электронный ресурс]. URL: https://kamal-deploy.org/docs/ (дата обращения 20.07.2024).