Практическая работа №7. CI/CD

Цель работы

Основной целью данной работы является приобретение первичных навыков работы с настройкой CI/CD конвейеров, а также serverless-стеком.

Теоретическое введение

Разработка любой системы заканчивается её доставкой до конечного пользователя. В случае с монолитными web-приложениями это достаточно простая задача. Нужна виртуальная машина с публичным ip-адресом. На ней настраивается окружение, а затем запускается монолитное приложение. В некоторых ситуациях эту операцию достаточно провести один раз, а затем приложение будет работать без изменений несколько месяцев. При необходимости внести изменения в работу приложения старое приложение выключают, загружают новое и запускают. Сделать это не так сложно, поэтому процесс вполне себе выполним даже вручную.

Когда речь заходит про микросервисную архитектуру, количество работы автоматически возрастает в разы. Во-первых, микросервисная архитектура – это набор сервисов. Как правило этот набор достаточно большой, поэтому ручное развёртывание каждого сервиса – это не вариант. В дополнение система микросервисов должна достаточно просто масштабироваться, что невозможно при ручном развёртывании.

Для упрощения процесса было создано множество различных DevOps практик, автоматизирующих развёртывание сервисов, а также сильно уменьшающих время между добавлением кода в репозиторий и запуском новой версии в production-среде. Ключевыми здесь являются конвейеры сборки и доставки.

CI/CD

В основе CI/CD лежит три концепции: непрерывная интеграция, непрерывная доставка и непрерывное развёртывание.

К непрерывной интеграции (CI, continuous integration) относятся все действия, которые необходимо совершить с исходным кодом сервиса, чтобы получить собранное и готовое к разворачиванию решение. Как правило здесь производят установку необходимых модулей, компиляцию решения, его упаковку в контейнер, выполнение pre-deployment тестов (например, интеграционные, unit). В идеале после выполнения CI на выходе должен получиться docker-образ с решением, которое было полностью протестировано изолированно от других сервисов.

CD может обозначать два разных понятия. Более простым является непрерывная доставка (continuous delivery). В рамках данного процесса происходит доставка собранного на этапе CI решения в специальные хранилища или на машины, где это решение будет развёрнуто. Однако, доставленное решение не будет запущено до тех пор, пока лицо, принимающее решения, не отдаст команду запустить именно эту версию. Такой подход позволяет произвести ревью кода до того, как будет запущена новая версия кода, а успешное прохождение CI будет проводиться ради проверки кода.

Второй вариант чтения CD – непрерывное развёртывание (continuous deployment). Оно включает в себя все автоматизации, которые производятся в процессе непрерывной доставки, а также автоматически запускает собранное и доставленное решение на необходимых машинах.

GitHub Actions

Для реализации CI/CD конвейеров существует много разных инструментов. GitHub Actions – это относительно молодая система. Она имеет низкий порог вхождения, хорошую документацию и большое комьюнити пользователей, разрабатывающих для неё свои расширения.

В GitHub Actions используется императивный подход для описания конвейеров. Пользователю необходимо указать все шаги и команды, которые необходимо выполнить в конвейере, в workflow-файле с расширением `.yml`.

Такие файлы могут иметь любое название, но обязательно должны находиться в папке `.github/workflows`.

Файл имеет несколько глобальных секций. В начале, в обязательной секции `on`, как правило указываются условия, при выполнении которых запускается выполнение конвейера. Например, конвейер, приведённый ниже будет выполняться при любой отправке изменений в ветку `master` или создании пулл реквеста в эту ветку.

```
# /.github/workflows/ci-cd.yml

name: CI/CD pipline

on:
  push:
    branches: ["master"]
  pull_request:
    branches: ["master"]

...
```

Следующая секция, необходимая для настройки CI/CD – это `jobs`. Здесь указывается, что необходимо для выполнения конвейера. Работ может быть несколько, каждая работа представляет собой объединённый по смыслу набор шагов, выполняемых в конвейере. Например, первая работа – сборка решения, вторая – тестирование, третья – доставка, четвёртая – развёртывание. Как разделить шаги на работы решает только разработчик конвейера.

```
# /.github/workflows/ci-cd.yml

...

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
      ...

  deploy:
    runs-on: ubuntu-latest
    steps:
```

```
...
```

В каждой работе необходимо указать машину в поле `runs-on`, на которой работа будет выполняться. Это могут быть как машины, которые предоставляет сам GitHub, так и собственные. Важно помнить, что для каждой работы создаётся своё собственное окружение, а также, что они могут запускаться на разных машинах.

Кроме машины необходимо в секции `steps` указать шаги, которые будут выполняться в конвейере. Шаги могут быть двух видов. Это могут быть какие-то команды, которые будут выполняться в командной строке машины, используемой для сборки. В таком случае необходимо указать нужную команду в поле `run`. Кроме прописывания команд вручную можно воспользоваться готовыми компонентами, написанными ранее сообществом разработчиков. В таком случае вместо поля `run` необходимо указать в `uses` конкретный модуль, который надо использовать.

```yaml
# /.github/workflows/ci-cd.yml

...

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@3

      - name: Build docker image
        run: |
          docker build . --file Dockerfile --tag delivery-service:latest

...
```

Все выполняемые в рамках конвейеров команды и их логи выводятся в консоль. Если репозиторий является публичным, то любой человек может зайти и посмотреть, что вывелось в логи. Важно помнить про это и не допускать использования различных секретов в конвейерах в открытом виде. Все секреты можно указать в настройках репозитория (<repository_url> /settings/secrets/actions), а затем использовать в конвейерах с помощью

специальных конструкций. В примере ниже используются переменные `DOCKER_USERNAME` и `DOCKER_PASSWORD` для загрузки собранного образа в реестр образов DockerHub.

```
# /.github/workflows/ci-cd.yml

...
      - name: Log in to DockerHub
        uses: docker/login-action@v3.0.0
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Push tag to DockerHub
        run: |
          docker push ${{ secrets.DOCKER_USERNAME }}/delivery-service:latest
...
```

В данный момент в конвейере описаны шаги для сборки и отправки решения в реестр контейнеров DockerHub. То есть уже есть реализованные непрерывная интеграция и непрерывная доставка.

Непрерывное развёртывание

В идеале конвейер должен заканчиваться автоматическим развёртыванием на какой-то инфраструктуре. В данной практической работе будет рассматриваться развёртывание в виде serverless-контейнера в Яндекс.Облаке.

Концепция бессерверных контейнеров позволяет разработчикам перестать заботиться об инфраструктуре, на которой запускается приложение, и масштабировании приложений. Этот подход сокращает запуск приложения до нескольких пунктов: упаковать приложение в контейнер, указать требования к процессору и памяти, запустить приложение. Вся остальная работа будет происходить автоматически на стороне провайдера облачных услуг.

Ещё одним достоинством данного подхода является тарификация приложений: платить надо лишь за те ресурсы, которые реально используются, исключая время простоя приложения.

Чтобы приступить к созданию бессерверного контейнера в Яндекс.Облаке, необходимо настроить работу с сервисом Container Registry. Для этого надо найти в списке данный сервис и перейти в него (Рисунок 6.2). Если в системе будут отсутствовать реестры, надо будет создать один, где будут храниться все образы учебной группы. Назвать реестр необходимо по шаблону `ikbo<номер_группы>`.

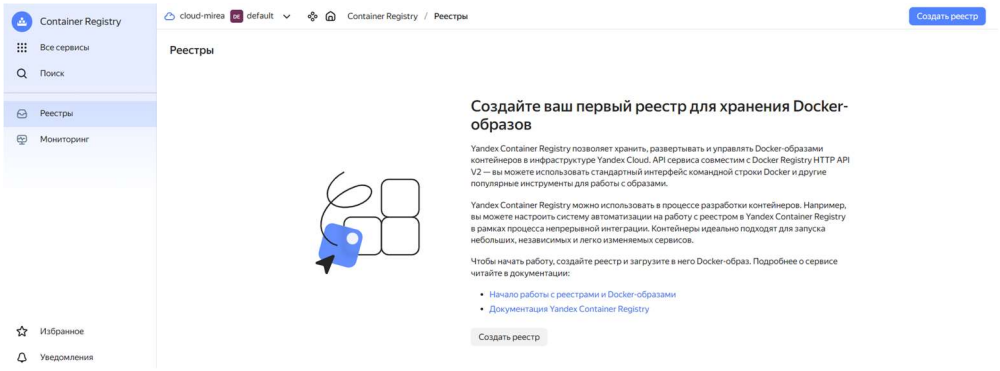

Рисунок 6.2 – Страница работы с реестрами контейнеров

Убедившись, что реестр существует, можно переходить к следующему шагу. Для загрузки docker-образа в реестр понадобится авторизация в реестре. Для авторизации приложений в Яндекс.Облаке используются сервисные аккаунты. Перейти к сервисным аккаунтам можно, открыв вкладку «Сервисные аккаунты» на странице каталога (Рисунок 6.3). Для выполнения практической работы уже создан сервисный аккаунт с нужными ролями (`sa-cicd`).

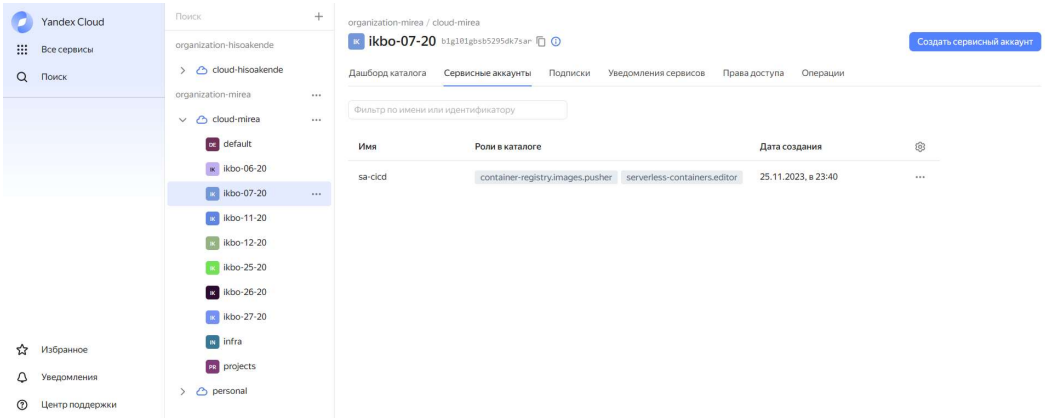

Рисунок 6.3 – Страница работы с реестрами контейнеров

Перейдя внутрь аккаунта, необходимо создать новый авторизованный ключ. Для этого необходимо нажать на одноимённую кнопку (Рисунок 6.4).

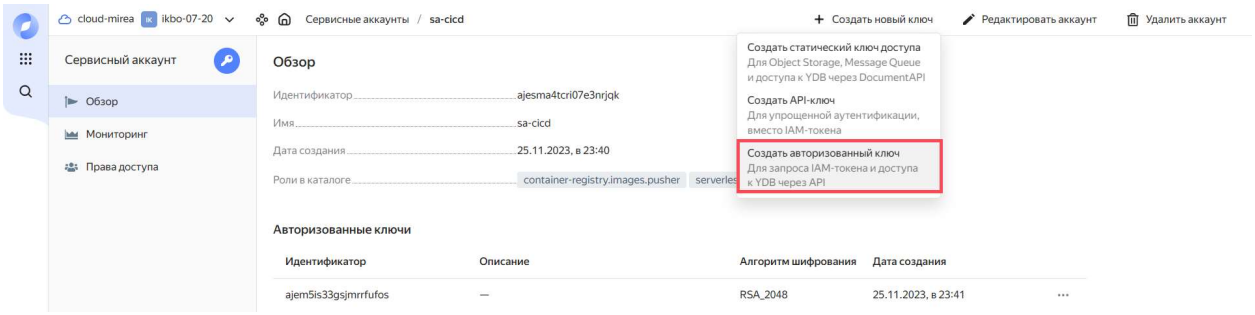

Рисунок 6.4 – Кнопка «Создать авторизованный ключ»

В появившемся меню в описании необходимо указать, чей это ключ, чтоб их потом можно было идентифицировать. После нажатия на кнопку «Создать» откроется окошко, в котором будут записаны сгенерированные ключи (Рисунок 6.5). На этом этапе необходимо нажать «Скачать файл с ключами». Содержимое скачанного файла необходимо поместить в переменную `YC_KEYS` в секреты репозитория на GitHub.

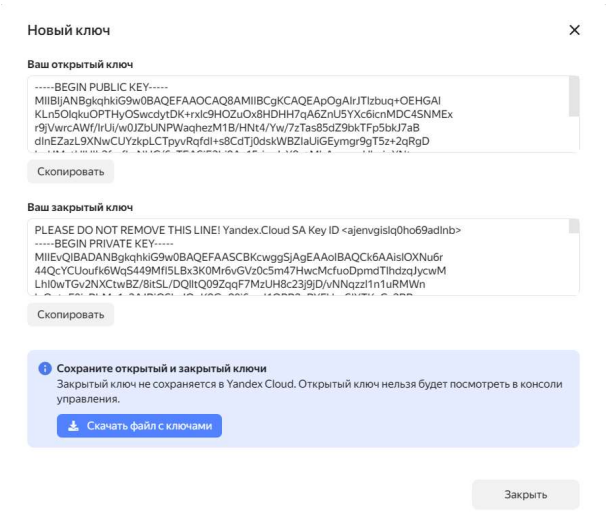

Рисунок 6.5 – Страница с созданным ключом

Теперь можно модифицировать CI/CD конвейер так, чтобы он отправлял docker-образ в реестр Яндекс.Облаке. Для этого понадобится узнать id реестра. Этот номер надо будет поместить в секрет `YC_REGISTRY_ID`.

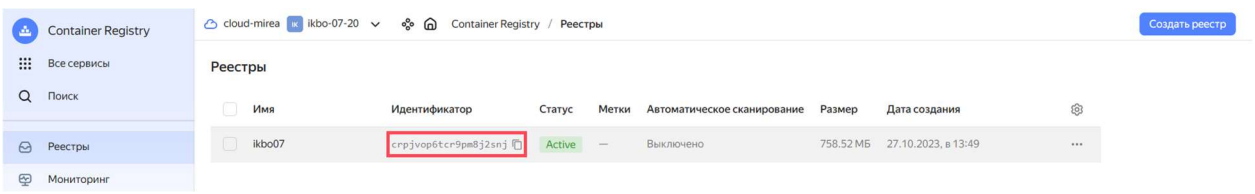

Рисунок 6.6 – Поле с уникальным номером реестра

Изменить надо будет все команды, где идёт работа с docker-образом. После изменения логин и пуш образа будет происходить не в DockerHub, а в хранилище образов Яндекс.Облака.

```
# /.github/workflows/ci-cd.yml

...
      - name: Build docker image
        run: |
          docker build . --file Dockerfile --tag cr.yandex/${{
secrets.YC_REGISTRY_ID }}/delivery-service:latest
...
      - name: Login to YC Registry
        uses: docker/login-action@v3.0.0
        with:
          registry: cr.yandex
          username: json_key
          password: ${{ secrets.YC_KEYS }}

      - name: Push tag to YC Registry
        run: |
          docker push cr.yandex/${{ secrets.YC_REGISTRY_ID }}/delivery-
service:latest
...
```

После выполнения конвейера внутри реестра образов группы появится новый репозиторий с образом внутри (Рисунок 6.7). Перед переходом к следующему шагу необходимо выполнить загрузку образа в реестр хотя бы один раз.

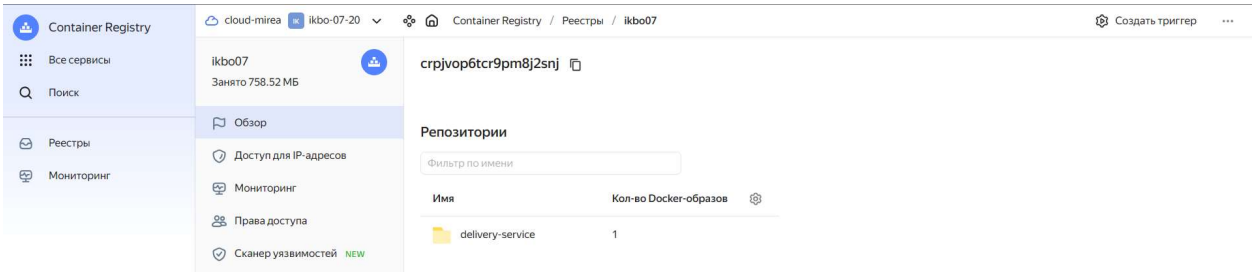

Рисунок 6.7 – Список репозиториев в реестре образов

Для создания бессерверного контейнера необходимо перейти к сервису Serverless Containers и нажать «Создать новый». Название контейнера в начале должно содержать фамилию студента, выполняющего практическое задание. После создания контейнера понадобится занести в секреты репозитория на GitHub некоторые данные: идентификатор сервисного

аккаунта (`YC_SA_ID`), имя созданного бессерверного аккаунта (`YC_CONTAINER_NAME`), идентификатор каталога (`YC_FOLDER_ID`). Вслед за этим можно добавить ещё одну работу в уже имеющийся CI/CD конвейер. По умолчанию необходимо задать пять обязательных полей для создания новой ревизии контейнера. Тем не менее этих параметров может оказаться недостаточно, так как для запуска приложения может понадобиться больше 1 vCPU и 128 Мб ОЗУ. Чтобы ознакомиться с полным списком параметров можно обратиться к документации. Последним параметром шага указаны переменные окружения, которыми конфигурируется сервис.

```
# /.github/workflows/ci-cd.yml

...
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    needs: [build-and-push-to-yc]
    steps:
    - name: Deploy serverless container
      uses: yc-actions/yc-sls-container-deploy@v1
      with:
        yc-sa-json-credentials: ${{ secrets.YC_KEYS }}
        container-name: ${{ secrets.YC_CONTAINER_NAME }}
        folder-id: ${{ secrets.YC_FOLDER_ID }}
        revision-image-url: cr.yandex/${{ secrets.YC_REGISTRY_ID }}/delivery-
service:latest
        revision-service-account-id: ${{ secrets.YC_SA_ID }}
        revision-env: |
          AMQP_URL=${{ secrets.ENV_AMQP_URL }}
          POSTGRES_URL=${{ secrets.ENV_POSTGRES_URL }}
```

Важно помнить, что бессерверные контейнеры при отсутствии активности автоматически отключаются, поэтому такой подход не годится, если надо хранить в приложении какие-то данные.

Сам по себе бэкенд данные не хранит, для этого он использует базу данных. Чтобы приложение корректно работало, можно указать ссылки на СУБД, развёрнутые в сети. Например, для выполнения данной практической работы была развёрнута СУБД PostgreSQL (`postgresql://secUREusER:StrongEnoughPassword)@51.250.26.59:5432/`).

Чтобы к сервису можно было выполнять запросы, необходимо в настройках включить публичный доступ (Рисунок 6.8).

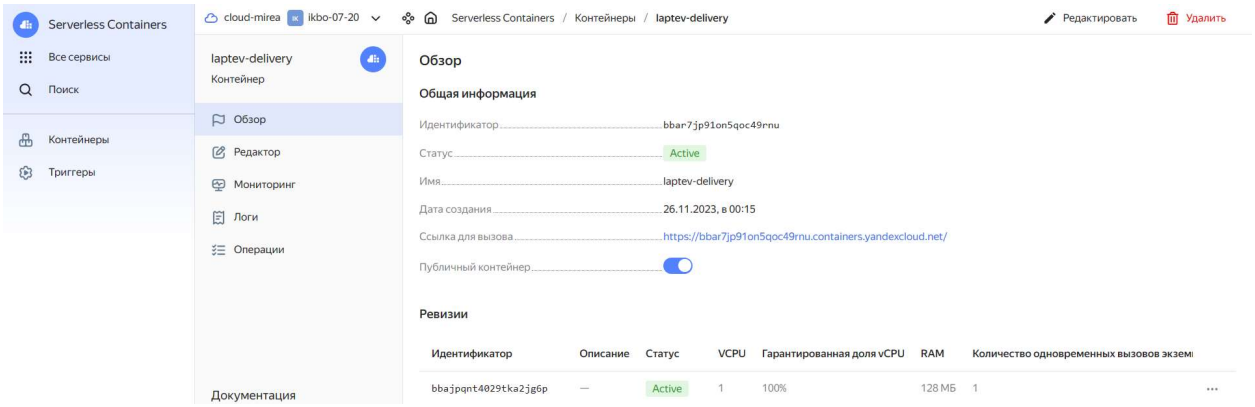

Рисунок 6.8 – Настройки бессерверного контейнера

Задание на самостоятельную работу

1. Настроить для одного из сервисов из Практической работы 6 непрерывную интеграцию и доставку в реестр контейнеров DockerHub.
2. Настроить для второго сервиса непрерывную интеграцию и развёртывание в виде бессерверного контейнера в Яндекс.Облаке.