Домашняя работа №3. Настройка CI/CD (GitHub Actions)

1. Введение

В проекте `ChatThing` настроен конвейер непрерывной интеграции и доставки (CI/CD) с использованием **GitHub Actions**. Пайплайн автоматически собирает Docker-образы микросервисов и деплоит их в Kubernetes кластер при обновлении ветки `main`.

Файл конфигурации: `.github/workflows/deploy.yaml`

2. Структура Workflow

Workflow состоит из трех основных джобов:

1. `build-chat-service`: Сборка бэкенда.
2. `build-webapp`: Сборка фронтенда.
3. `deploy`: Деплой в Kubernetes.

Триггер запуска:

```
on:
  push:
    branches: [main]
  workflow_dispatch: # Возможность ручного запуска
```

3. Этап сборки (Build & Push)

Для каждого сервиса (`chat-service` и `webapp`) выполняются идентичные шаги. Рассмотрим на примере `chat-service`.

Используемые экшены:

- `docker/setup-buildx-action`: Настройка сборщика Buildx.
- `docker/login-action`: Авторизация в Container Registry (используется сторонний регистр, данные в secrets).
- `docker/metadata-action`: Генерация тегов для образов (sha-commit, latest).
- `docker/build-push-action`: Непосредственно сборка и пуш.

Особенность сборки `chat-service`: собираются два образа из одного контекста (target `final` для сервиса и `migrate` для мигратора).

```
  build-chat-service:
    steps:
      - name: Build and push Docker image
        uses: docker/build-push-action@v5
```

```
          with:
            context: ./chat-service
            target: final
            push: true
            tags: ${{ steps.meta.outputs.tags }}
            cache-from: type=gha # Кэширование слоев в GitHub Actions
            cache-to: type=gha,mode=max
```

4. Этап деплоя (Deploy to Kubernetes)

Джоб `deploy` зависит от успешного завершения билдов (`needs: [build-chat-service, build-webapp]`).

Основные шаги:

1. **Настройка окружения**: Установка `kubectl` и конфигурация доступа к кластеру через `secrets.KUBECONFIG`.
2. **Подготовка секретов**: Создание docker-registry secret для пулла образов и generic secret с переменными окружения (DB_PASSWORD, API keys, etc.) из GitHub Secrets.
3. **Обновление версий**: Использование `sed` для подстановки хеша коммита в манифесты деплоймента. Это гарантирует, что кластер обновится на свежесобранную версию, а не просто перетянет `latest`.

   ```
   SHORT_SHA=$(echo ${{ github.sha }} | cut -c1-7)
   sed -i "s|image:.*chat-service:.*|image: ${{ env.REGISTRY }}/${{
   env.CHAT_SERVICE_IMAGE }}:${SHORT_SHA}|g" chat-
   service/k8s/deployment.yaml
   ```

4. **Применение конфигурации**:

   ```
   kubectl apply -k chat-service/k8s/
   kubectl apply -k webapp/k8s/
   ```

   Используется `kustomize` (флаг `-k`) или просто директория с манифестами.
5. **Верификация**: Ожидание успешного роллаута (`kubectl rollout status`) и проверка статуса подов.

5. Вывод

Реализован полный цикл CI/CD:

1. Код пушится в репозиторий.
2. GitHub Actions собирает оптимизированные Docker-образы с кэшированием.
3. Образы пушатся в приватный регистр.
4. K8s манифесты патчатся новой версией образа.
5. `kubectl apply` обновляет состояние кластера без даунтайма (Rolling Update).

6. Секреты безопасно доставляются в кластер через GitHub Secrets.