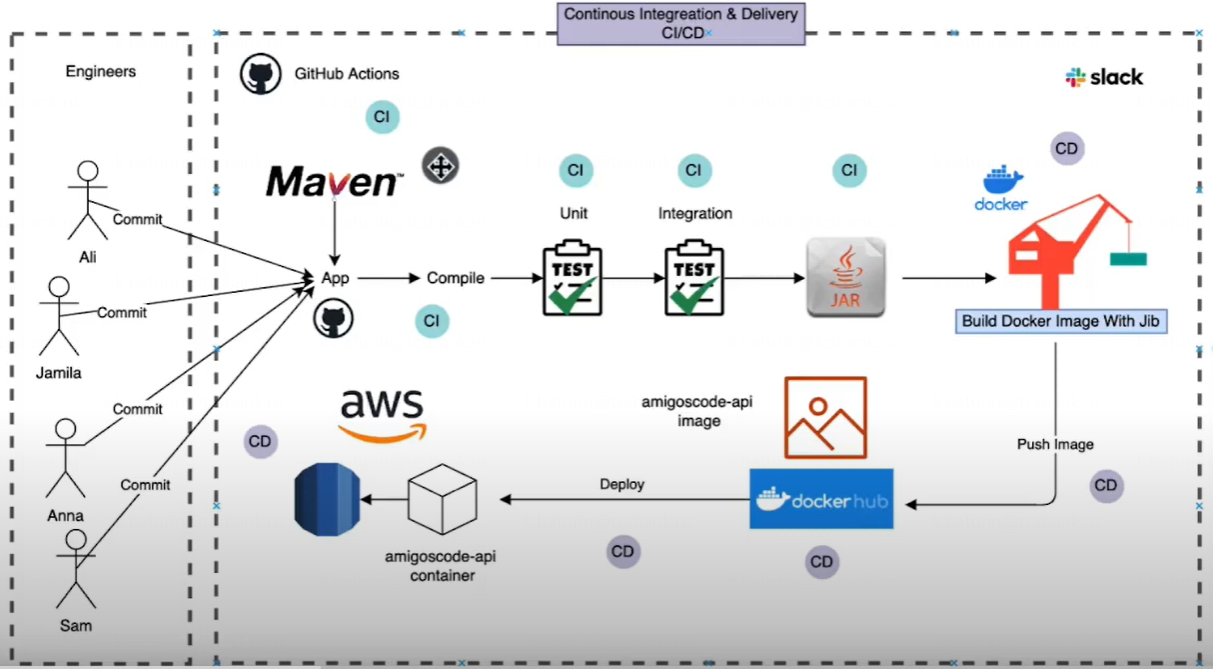
***Continuous Integration & Continuous Delivery*** – это способ интеграции изменений кода и доставки его на прод. Он позволяет автоматизировать процесс обновления, тестирования кода и деплоя новой версии приложения.

Рассмотрим построение *CI/CD* для ***java*** приложения.



Вручную запаковать *jar*-файл можно командой ***mvn package***

Вручную запустить *jar*-файл можно командой ***java -jar .\target\spring-boot-example-0.0.1-SNAPSHOT.jar***

Остановить работу *jar*-файла можно комбинацией ***Ctrl+C***

Для того чтобы автоматически скомпилировать, протестировать и собрать проект в *jar*-файл, в *pom*-файл проекта нужно добавить ***spring-boot-maven-plugin***. Все эти команды можно запустить через готовые скрипты *Maven* в *Idea*.

*<build>*

*<plugins>*

*<plugin>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-maven-plugin</artifactId>*

*</plugin>*

*</plugins>*

*</build>*

В качестве инструмента для сборки и загрузки во вне докер-образа можно использовать либо *[dokerfile](https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/" \l "build-image.build-image-goal)* [+ скрипт](https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/" \l "build-image.build-image-goal) *[maven](https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/" \l "build-image.build-image-goal)* на сборку образа, либо плагин [jib](https://github.com/GoogleContainerTools/jib) от google.

Пример *jib*-плагина в *pom*-файле:

*<build>*

*<plugins>*

*<plugin>*

*<groupId>com.google.cloud.tools</groupId>*

*<artifactId>jib-maven-plugin</artifactId>*

*<version>3.3.1</version>*

*<configuration>*

*<from>*

*<image>eclipse-temurin:17</image> // версия java sdk*

*<platforms>*

*<platform> // тип ядра и ОС*

*<architecture>arm64</architecture>*

*<os>linux</os>*

*</platform>*

*<platform>*

*<architecture>amd64</architecture>*

*<os>linux</os>*

*</platform>*

*</platforms>*

*</from>*

*<to> // загрузка в dockerHub (название докер-образа и теги)*

*<image>docker.io/${docker.username}/${project.artifactId}:${project.version}</image>*

*<tags>*

*<tag>latest</tag>*

*</tags>*

*</to>*

*</configuration>*

*</plugin>*

*</plugins>*

*</build>*

Для сборки проекта и загрузки образа в *dockerhub* нужно ввести команду:

*mvn clean package jib:build*

Для построения ***CI/CD*** можно также использовать ***GitActions***. Большинство повторяющихся действий уже описано другими пользователями. Их можно найти на <https://github.com/marketplace> .

Рассмотрим пример создания простого ***CI*** процесса тестирования нового ***java*** кода. Для этого в корне проекта создадим папку *.github\workflows* и новый файл-*workflow* с именем ***backend-ci.yml*** :

*name: CI - Build Backend*

*on:*

*pull\_request: // workflow срабатывает при pull\_request*

*branches:*

*- main // в ветку main*

*paths:*

*- backend/\*\* // с обновлением любых файлов по этому пути*

*jobs:*

*build-job: // имя job может быть любым*

*runs-on: ubuntu-latest // работа будет выполняться на ВМ с ubuntu*

*services:*

*postgres: // создаем и настраиваем сервис с тестовой БД*

*image: postgres:14.1*

*env:*

*POSTGRES\_USER: straigt*

*POSTGRES\_PASSWORD: password*

*POSTGRES\_DB: customer*

*ports:*

*- 5332:5432*

*options: >- // проверка работоспособности тестовой БД*

*--health-cmd pg\_isready*

*--health-interval 10s*

*--health-timeout 5s*

*--health-retries 5*

*defaults: // указываем папку с проектом*

*run:*

*working-directory: ./backend*

*steps: // список шагов*

*- uses: actions/checkout@v3 // проверка и клонирование репозитория в Linux runner*

*- uses: actions/setup-java@v3 // установка JDK temurin 17 с кешированием зависимостей*

*with:*

*distribution: 'temurin'*

*java-version: '17'*

*cache: 'maven'*

*- name: Build and run Unit/Integration Tests with Maven*

*run: mvn -ntp -B verify // запуск скрипта maven на проверку (без логов зависимостей)*

Рассмотрим пример создания простого ***CD*** процесса деплоя нового ***java*** кода. Для этого в корне проекта создадим папку *.github\workflows* и новый файл-*workflow* с именем ***backend-cd.yml*** :

*name: CD - Deploy Backend*

*on:*

*workflow\_dispatch: // ручной запуск workflow*

*push: // workflow срабатывает при push*

*branches:*

*- main // в ветку main*

*paths:*

*- backend/\*\* // с обновлением любых файлов по этому пути*

*jobs:*

*deploy-job: // имя job может быть любым*

*runs-on: ubuntu-latest // работа будет выполняться на ВМ с ubuntu*

*services:*

*postgres: // создаем и настраиваем сервис с тестовой БД*

*image: postgres:14.1*

*env:*

*POSTGRES\_USER: straigt*

*POSTGRES\_PASSWORD: password*

*POSTGRES\_DB: customer*

*ports:*

*- 5332:5432*

*options: >- // проверка работоспособности тестовой БД*

*--health-cmd pg\_isready*

*--health-interval 10s*

*--health-timeout 5s*

*--health-retries 5*

*defaults: // указываем папку с проектом*

*run:*

*working-directory: ./backend*

*steps: // список шагов*

*- uses: actions/checkout@v3 // проверка и клонирование репозитория в Linux runner*

*- uses: actions/setup-java@v3 // установка JDK temurin 17 с кешированием зависимостей*

*with:*

*distribution: 'temurin'*

*java-version: '17'*

*cache: 'maven'*

*- name: Login to Docker Hub // логинимся на DockerHub*

*uses: docker/login-action@v2*

*with: // используем секреты из github для текущего проекта*

*username: ${{ secrets.DOCKERHUB\_USERNAME }}*

*password: ${{ secrets.DOCKERHUB\_ACCESS\_TOKEN }}*

*// создаем переменную BUILD\_NUMBER и экспортируем её в GITHUB\_OUTPUT*

*- name: Set build number*

*id: build-number // id шага для вызова переменной*

*run: echo "BUILD\_NUMBER=$(date '+%d.%m.%Y.%H.%M.%S')" >> $GITHUB\_OUTPUT*

*// собираем и запаковываем приложение в docker образ и пушим на DockerHub*

*- name: Build Package Push with Maven*

*run: mvn -ntp -B verify -Ddocker.image.tag=${{steps.build-number.outputs.BUILD\_NUMBER}} jib:build*

*// заменяем номер сборки в файле-настройке aws*

*- name: Update Dockerrun.aws.json api image tag with new build number*

*run: |*

*echo "Dockerrun.aws.json before updating tag"*

*cat ../Dockerrun.aws.json*

*sed -i -E 's\_(straigt/straigt-api:)([^"]\*)\_\1'${{steps.build-number.outputs.BUILD\_NUMBER}}'\_' ../Dockerrun.aws.json*

*echo "Dockerrun.aws.json after updating tag"*

*cat ../Dockerrun.aws.json*

*- name: Deploy to Elastic Beanstalk // деплоим новую версию в aws*

*uses: einaregilsson/beanstalk-deploy@v21*

*with:*

*aws\_access\_key: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}*

*aws\_secret\_key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}*

*application\_name: ${{ secrets.EB\_APPLICATION\_NAME }}*

*environment\_name: ${{ secrets.EB\_ENVIRONMENT\_NAME }}*

*version\_label: ${{ steps.build-number.outputs.BUILD\_NUMBER }}*

*version\_description: ${{ github.SHA }}*

*region: ${{ secrets.EB\_REGION }}*

*deployment\_package: Dockerrun.aws.json*

*// загружаем обновленный файл-настройку aws на github*

*- name: Commit and push Dockerrun.aws.json*

*run: |*

*git config user.name github-actions*

*git config user.email* [*github-actions@github.com*](mailto:github-actions@github.com)

*git add ../Dockerrun.aws.json*

*git commit -m "Update Dockerrun.aws.json docker image with new tag ${{ steps.build-number.outputs.BUILD\_NUMBER }}"*

*git push*

*// отправляем сообщение в slack*

*- name: Slack commit message and sha*

*run: >*

*curl -X POST -H 'Content-type: application/json'*

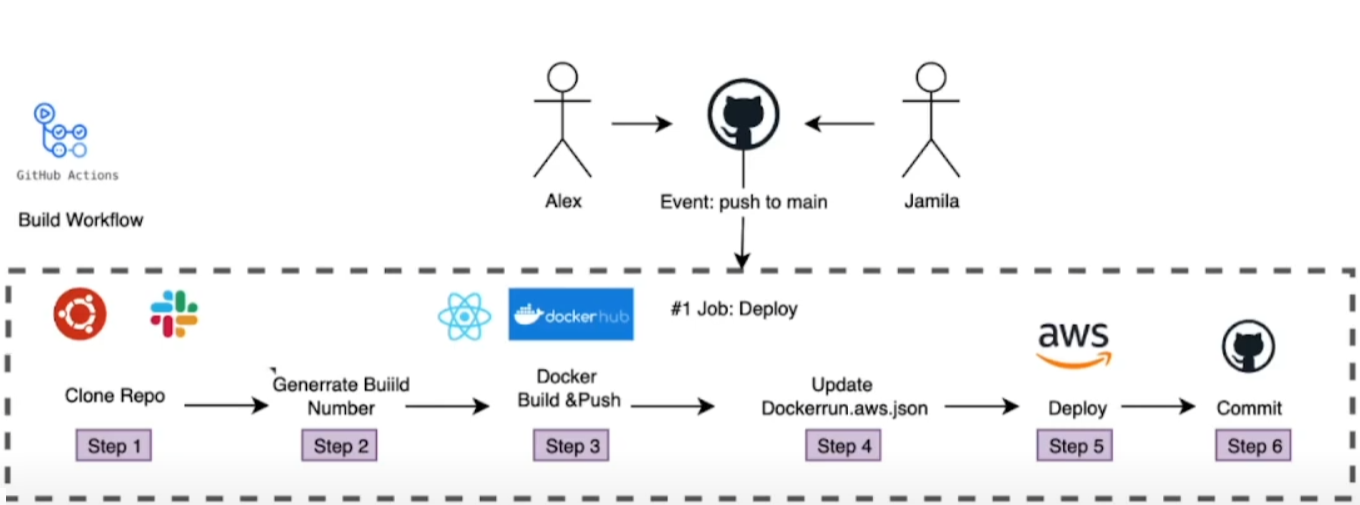
*--data '{"text":":github: https://github.com/straigt/spring-boot-full-stack/commit/${{ github.sha }} - ${{ github.event.head\_commit.message }}"}'*

*${{ secrets.SLACK\_WEBHOOK\_URL }}*

Символ **>** позволяет выполнить все тело команды *run* в одну строку.

Символ **|** позволяет выполнить каждую строку команды *run* отдельно. Для переноса одной строки и её продления используется символ **\** .

Рассмотрим построение *CI/CD* для ***react*** приложения.



Рассмотрим пример создания простого ***CI*** процесса тестирования нового ***react*** кода. Для этого в корне проекта создадим папку *.ci* и новый файл-скрипт с именем ***build-publish.sh*** :

*// проверка корректности переданных переменных*

*: "${USERNAME:?USERNAME not set or empty}"*

*: "${REPO:?REPO not set or empty}"*

*: "${TAG:?TAG not set or empty}"*

*// сборка и отправка на DockerHub*

*docker buildx create --use*

*docker buildx build \*

*--platform=linux/amd64,linux/arm64 \*

*-t "${USERNAME}/${REPO}:${TAG}" \*

*-t "${USERNAME}/${REPO}:latest" \*

*"${@:2}" \ // использование второго аргумента (api\_base\_url)*

*--push \*

*"$1" // использование первого аргумента (точка)*

Рассмотрим пример создания простого ***CD*** процесса деплоя нового ***react*** кода. Для этого в корне проекта создадим папку *.github\workflows* и новый файл-*workflow* с именем ***frontend-react-cd.yml*** :

*name: CD - Deploy Backend*

*on:*

*workflow\_dispatch: // ручной запуск workflow*

*push: // workflow срабатывает при push*

*branches:*

*- main // в ветку main*

*paths:*

*- frontend/react/\*\* // с обновлением любых файлов по этому пути*

*jobs:*

*deploy-job: // имя job может быть любым*

*if: true // можно добавить условие запуска*

*runs-on: ubuntu-latest // работа будет выполняться на ВМ с ubuntu*

*defaults: // указываем папку с проектом*

*run:*

*working-directory: ./fronend/react*

*steps: // список шагов*

*- uses: actions/checkout@v3 // проверка и клонирование репозитория в Linux runner*

*- name: Login to Docker Hub // логинимся на DockerHub*

*uses: docker/login-action@v2*

*with: // используем секреты из github для текущего проекта*

*username: ${{ secrets.DOCKERHUB\_USERNAME }}*

*password: ${{ secrets.DOCKERHUB\_ACCESS\_TOKEN }}*

*// создаем переменную BUILD\_NUMBER и экспортируем её в GITHUB\_OUTPUT*

*- name: Set build number*

*id: build-number // id шага для вызова переменной*

*run: echo "BUILD\_NUMBER=$(date '+%d.%m.%Y.%H.%M.%S')" >> $GITHUB\_OUTPUT*

*// собираем и запаковываем приложение в docker образ и пушим на DockerHub*

*// можно найти действие в маркетплейсе*

*- name: Docker Build and push*

*run: |*

*chmod +x ../../.ci/build-publish.sh // запускаем ci скрипт*

*USERNAME=straigt \ // пробрасываем переменные в скрипт*

*REPO=straigt-react \*

*TAG=${{ steps.build-number.outputs.BUILD\_NUMBER }} \*

*../../.ci/build-publish.sh . \*

*--build-arg api\_base\_url=http://straigtapi-env.eu-west-1.elasticbeanstalk.com:8080*

*// заменяем номер сборки в файле-настройке aws*

*- name: Update Dockerrun.aws.json api image tag with new build number*

*run: |*

*echo "Dockerrun.aws.json before updating tag"*

*cat ../../Dockerrun.aws.json*

*sed -i -E 's\_(straigt/straigt-react:)([^"]\*)\_\1'${{steps.build-number.outputs.BUILD\_NUMBER}}'\_' ../../Dockerrun.aws.json*

*echo "Dockerrun.aws.json after updating tag"*

*cat ../../Dockerrun.aws.json*

*- name: Deploy to Elastic Beanstalk // деплоим новую версию в aws*

*uses: einaregilsson/beanstalk-deploy@v21*

*with:*

*aws\_access\_key: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}*

*aws\_secret\_key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}*

*application\_name: ${{ secrets.EB\_APPLICATION\_NAME }}*

*environment\_name: ${{ secrets.EB\_ENVIRONMENT\_NAME }}*

*version\_label: ${{ steps.build-number.outputs.BUILD\_NUMBER }}*

*version\_description: ${{ github.SHA }}*

*region: ${{ secrets.EB\_REGION }}*

*deployment\_package: Dockerrun.aws.json*

*wait\_for\_environment\_recovery: 60*

*// загружаем обновленный файл-настройку aws на github*

*- name: Commit and push Dockerrun.aws.json*

*run: |*

*git config user.name github-actions*

*git config user.email* [*github-actions@github.com*](mailto:github-actions@github.com)

*git add ../../Dockerrun.aws.json*

*git commit -m "Update Dockerrun.aws.json docker image with new tag ${{ steps.build-number.outputs.BUILD\_NUMBER }}"*

*git push*

*// отправляем сообщение в slack*

*- name: Slack commit message and sha*

*run: >*

*curl -X POST -H 'Content-type: application/json'*

*--data '{"text":":github: https://github.com/straigt/spring-boot-full-stack/commit/${{ github.sha }} - ${{ github.event.head\_commit.message }}"}'*

*${{ secrets.SLACK\_WEBHOOK\_URL }}*