**Лабораторная работа №20**

**Изучение методов разработки и развертывания программного обеспечения с использованием GitHub Actions**

Цель: изучить принципы Continuous Integration (CI) и Continuous Deployment (CD) и их внедрение с использованием инструмента GitHub Actions.

**Теоретические сведения**

Continuous Integration (CI) и Continuous Deployment (CD) – это практики разработки программного обеспечения, которые направлены на автоматизацию процессов сборки, тестирования и развертывания приложений. CI означает постоянное интегрирование изменений кода в общий репозиторий, после чего выполняются автоматические сборка и тестирование. CD включает в себя автоматическое развертывание успешно протестированных изменений в целевую среду.

Можно выделить следующие этапы CI/CD.

1. **Написание кода и интеграция**. Разработчики создают и тестируют код модулей, затем интегрируют их в основную ветку.
2. **Автоматизированная сборка и тестирование**. Весь продукт тестируется в виртуальной среде — CI-сервере.
3. **Тестирование в ручном режиме**. Выполняется проверка продукта на ошибки.
4. **Релиз**. После отладки код передается на релиз (становится доступным клиентам).
5. **Развертывание**. Актуальный программный релиз размещается на серверах.
6. **Поддержка и отслеживание**. Разработавшая продукт компания получает обратную связь, дает рекомендации и производит мониторинг реакции пользователей.
7. **Планирование разработок новых версий и патчей**. Разработчики готовят план доработок на основе User Experience (опыта взаимодействия пользователей с программой), вносит необходимые изменения в программу, и описанный выше процесс запускается заново.

Каждый из этих этапов выполняется последовательно и тщательно контролируется: это помогает минимизировать число ошибок и уязвимостей в продукте.

На рисунке 1 отражена последовательность и цикличность жизненных этапов CI и CD в процессе разработки и развертывания программного обеспечения.

Plan (Планирование): на этом этапе определяются требования к проекту, разрабатывается план действий и составляется список задач. В процессе планирования определяются приоритеты и сроки выполнения задач.

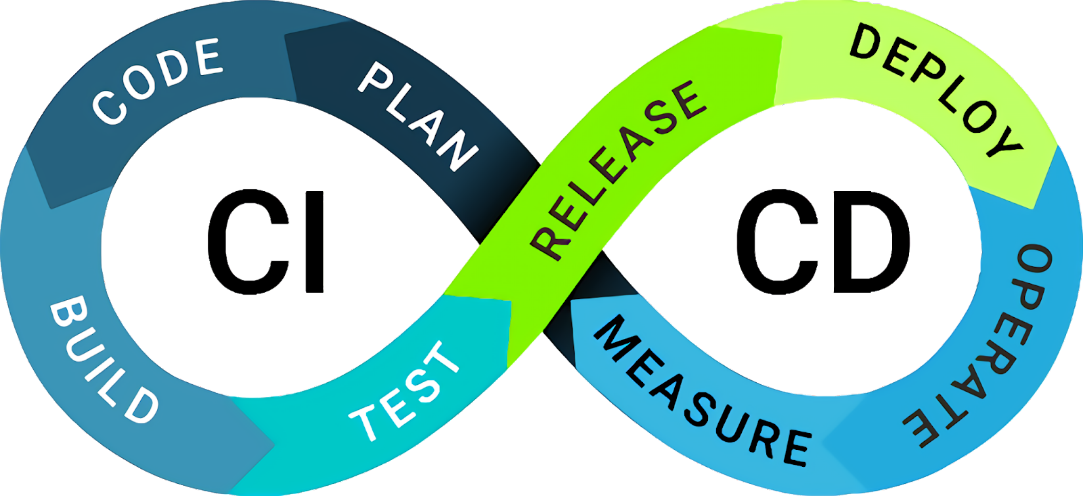


Рисунок 1 – Этапы CI/CD

Code (Написание кода): разработчики пишут исходный код, реализуя требуемую функциональность на основе утвержденного плана. На этом этапе может использоваться система контроля версий для управления изменениями и совместной работы над кодом.

Build (Сборка): исходный код компилируется в исполняемые файлы или другие форматы, готовые для развертывания. Включает в себя проверку синтаксиса, создание исполняемых файлов и упаковку приложения.

Test (Тестирование): созданные сборки проходят через автоматизированные тесты для обеспечения качества и надежности приложения. Включает в себя модульные, интеграционные, функциональные и другие виды тестирования.

Release (Выпуск): на этом этапе подготавливается к выпуску версия приложения, прошедшая успешное тестирование. Включает в себя упаковку приложения, обновление версии и подготовку к развертыванию.

Deploy (Развертывание): выпускаемая версия приложения автоматически разворачивается в целевой среде, будь то тестовая, пред продукционная или продукционная. Включает в себя копирование сборок на сервер, настройку окружения и обновление конфигурации.

Operate (Эксплуатация): на этапе приложение активно используется пользователями. Включает в себя мониторинг производительности, отслеживание журналов событий, управление доступом и т.д.

Measure (Измерение): в течение всего процесса CI/CD собираются метрики производительности и качества приложения. На этапе измерения анализируются собранные данные для выявления узких мест и улучшения процессов разработки.

Эти этапы образуют цикл, который постоянно повторяется в процессе разработки и обслуживания программного обеспечения, обеспечивая его постоянное улучшение и совершенствование.

GitHub Actions — это инструмент автоматизации, предоставляемый GitHub, который позволяет создавать и настраивать различные рабочие процессы (workflows) для автоматизации различных задач в процессе разработки программного обеспечения. Для настройки CI используются файлы с расширением yml.

**Практическое задание**

1. Создайте новый репозиторий на GitHub.
2. Инициализируйте репозиторий с простым приложением, который содержит unit-тесты.
3. Загрузите код приложения в репозиторий.
4. В репозитории перейдите в раздел "Actions" (рисунок 2).

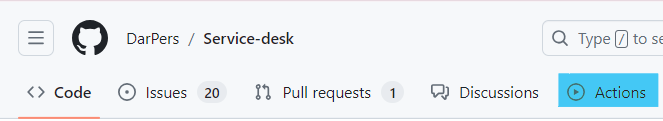


Рисунок 2 – Раздел "Actions" в репозитории

1. Нажмите "New workflows" (рис. 3) для создания нового workflow файла в верхнем левом углу экрана.

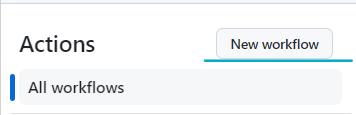


Рисунок 3 – Создание нового workflow

1. Далее вам будет представлено множество шаблонов (рис. 4), поскольку проект, находящийся в репозитории написан с помощью .NET, то и предложенный шаблон также заточен под .NET. Если предложенный шаблон не подходит вам, можно найти нужный с помощью поиска (рис. 5). И подтвердить выбор кнопкой «Configure».

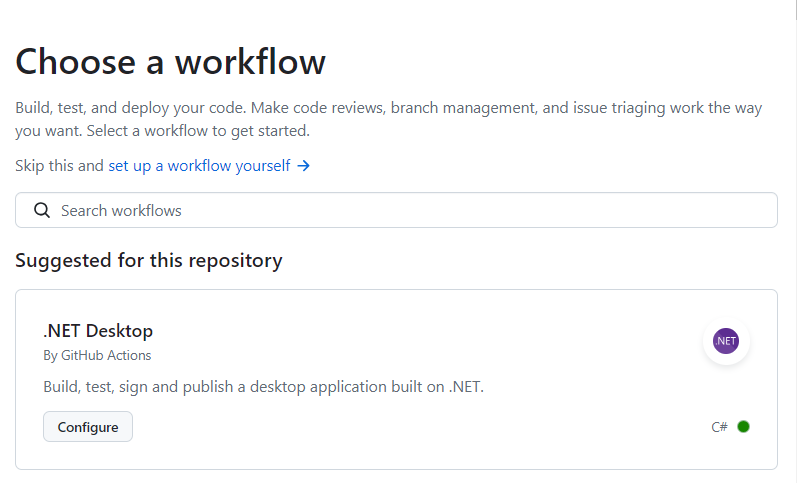


Рисунок 4 – Шаблоны workflows

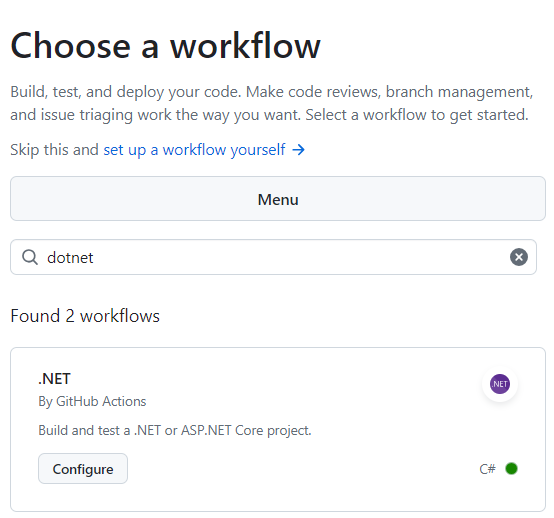


Рисунок 5 – Поиск шаблона

1. Далее вы попадаете на страницу, где можно задать имя .yml файла, а также настроить CI (рис. 6).

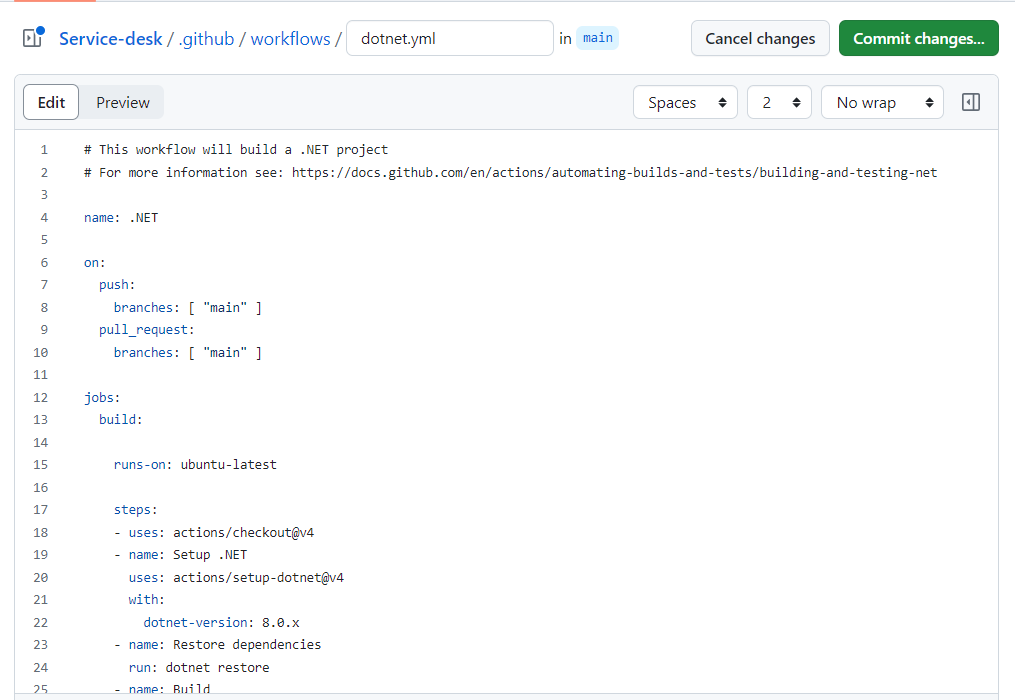


Рисунок 6 – Страница настройки CI

1. Настройка файла. Обычно данный файл состоит из нескольких блоков. Ниже, в таблице 1, будет приведен пример блоков кода шаблона проекта, и объяснения к нему.

Таблица 1 – Описание основных блоков кода workflow

|  |  |
| --- | --- |
| **Описание** | **Блок кода** |
| 1 | 2 |
| Назначение имени |  |
| Означает, что данный файл будет срабатывать при каждом pull request и push в ветку main. |  |
| В этом блоке описывается основная работа для всех шагов (о шагах будет говориться ниже). Данный блок кода обозначает операционную систему, в которой будет тестироваться наше приложение. В данном случае следует учитывать, что различные операционные системы устанавливаются быстрее других. Например, ubuntu устанавливается быстрее чем windows. |  |

Продолжение таблицы 1

|  |  |
| --- | --- |
| 1 | 2 |
| Блок **steps** содержит основные этапы CI. Каждый этап содержит **name** – имя, а также **run** – команда, которая должны выполнится. Команда прописывается так, как бы она вводилась в консоль для настройки вашего приложения на новом устройстве, так как запускаются в терминале.  Ключевое слово uses, указывает использовать внешний workflow, для корректной работы нашего. Обычно они автоматически прописываются при использовании шаблонного workflow из GitHub Actions. |  |
| Step с именем «Setup .NET» устанавливает .NET 8 версии. |  |
| Step с именем «Restore dependencies» восстанавливает зависимости, необходимые для работы приложения, используя запуск команды dotnet restore в терминале. |  |
| Step с именем «Build» собирает проект, используя запуск команды dotnet build --no-restore --configuration Release в терминале. |  |
| Step с именем «Test» запускает тесты, которые есть в нашем проекте, используя запуск команды dotnet test --no-build --verbosity normal --configuration Release в терминале. |  |

**Пример выполнения практического задания.**

Для данного примера будет рассмотрен код по ссылке: <https://github.com/Gh0stF1ghter/Library-API-Trainee-Task>

Все .yml файлы, которые содержат в себе настойки для ci, обычно хранятся по следующему адресу .github/workflows. Следовательно, файлы можно найти по такому адресу.

Рассмотрим каждую строку файла dotnet.yml (рис.7):

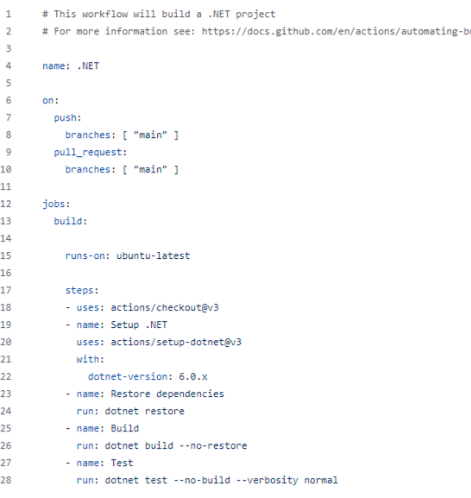


Рисунок 7 – Содержимое файла dotnet.yml

На строке 4 можно увидеть название данного Action (Далее будет показано, где их посмотреть).

В блоке на строках 6-10 показывается, на какое действие запускать данный код. В данном случае будет запускаться на push в ветку main, а также на pull\_request в ветку main.

Рассмотрим все шаги, которые написаны ниже:

Строка 15 обозначает, что данный проект будет запускаться на операционной системе ubuntu последней версии.

Сам проект был написан с использованием Windows, однако стоит учесть, что .net core кроссплатформенный, а значит мы можем выбирать то, что производительнее. Ubuntu является довольно быстрым поэтому для тестирования и тд обычно выбирают именно его.

Строка 18 позволяет рабочему процессу склонировать код из GitHub репозитория на виртуальную машину, где выполняется задание. `@v3` указывает на версию действия.

Строка 19 указывает название блока следующих шагов, а именно строк 20-22.

Строка 20 подготавливает среду для работы с dotnet, а именно для работы с 6 версией (строка 22).

Строка 23 указывает название для следующего блока.

Строка 24 запускает команду dotnet restore, которая восстанавливает зависимости. Под зависимостями подразумеваются различные пакет, которые были установлены через NuGet.

Строка 25 указывает название для блока.

Строка 26 запускает команду для сборки. dotnet build автоматически подтягивает зависимости, т. к. ранее мы уже это сделали, то необходимо указать —no-restore. Данный параметр указывает, что проект соберется без повторного восстановление зависимостей.

Строка 27 — это название для следующего блока.

Строка 28 запускает наши тесты. dotnet test - основная часть команды. —no-build использоваться, чтобы не делать сборку ещё раз. —verbosity normal используется что бы указать насколько подробно нужно выводить информации о прохождении тестов.

В данном проекте представлены unit тесты для одного из сервисом, которые можно найти в папке Tests. На рисунке 8 представлен пример одного из тестов.

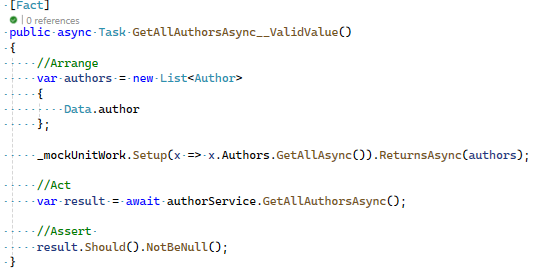


Рисунок 8 – Пример unit теста

Чтобы данный код заработал необходимо либо сделать push либо pull request в main. После этого во вкладке Actions начнет выполняться данный файл.

На рисунке 9 показан результат выполнения данного action.

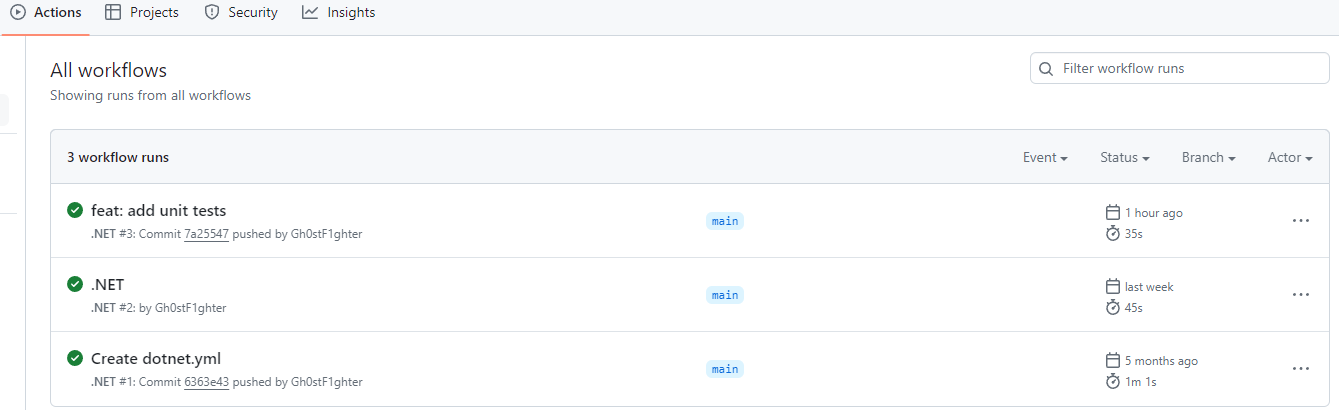


Рисунок 9 – Страница Actions с выполненными workflows.

Их 3, это связано с тем, что было 3 push, ведь код срабатывает на каждый push и pull request.

Если зайти в самый последний workflow, а затем в левом меню выбрать build, то можно информацию, представленную на рисунке 10.

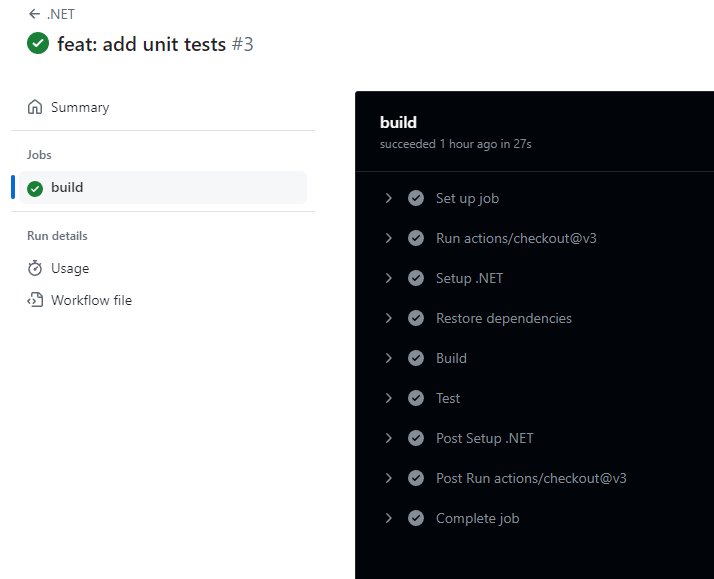


Рисунок 10 – Информация о текущем workflow.

Тут можно найти name для каждого нашего блока. Выбирая необходимый блок, можно просмотреть логи (Рис. 11).

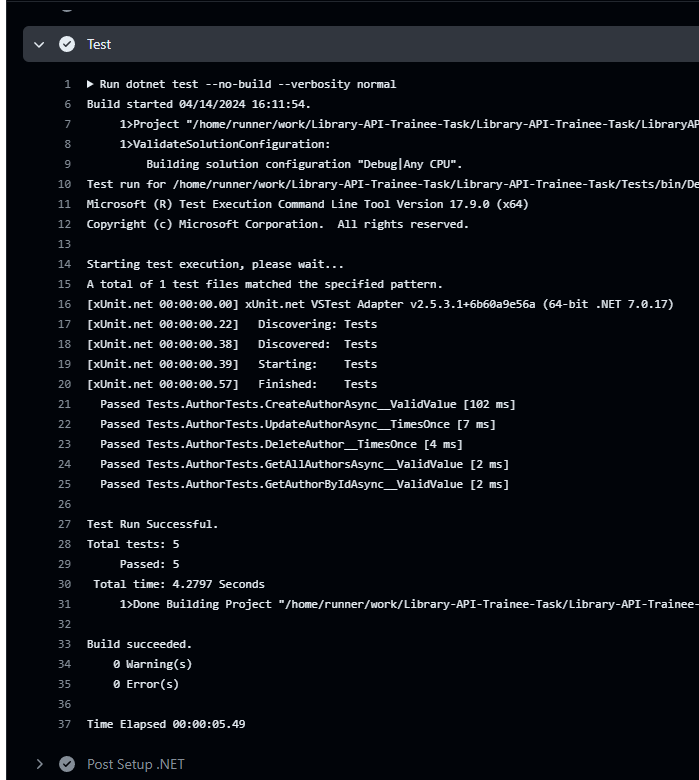


Рисунок 11 – Логи для Test

## Практическое задание

Взять свой готовый проект на GitHub с тестами (или создать новый, написать к нему тесты (тип тестов)), настроить ci/cd с помощью GitHub Actions. Сделать отчет о проделанной работе с пояснениями блоков кода вашего workflow.

**Контрольные вопросы**

1. Что такое непрерывная интеграция (CI)?
2. Как связаны непрерывная интеграция (Continuous Integration — CI) и управление версиями (Version Control — VC)?
3. Как связаны понятия «непрерывная интеграция» и «непрерывное развертывание»?
4. Какая пользу приносит технология CI/CD?
5. Какое отношение имеет тестирование к CI-процессам?
6. Какие тесты чаще других применяют в CI-процессах?
7. Сколько тестов может быть в проекте?
8. Что такое Gitflow?