

Сборка, анализ кода, unit-тестирование и публикация приложения .NET Framework с использованием GitLab CI, SonarQube и OpenCover

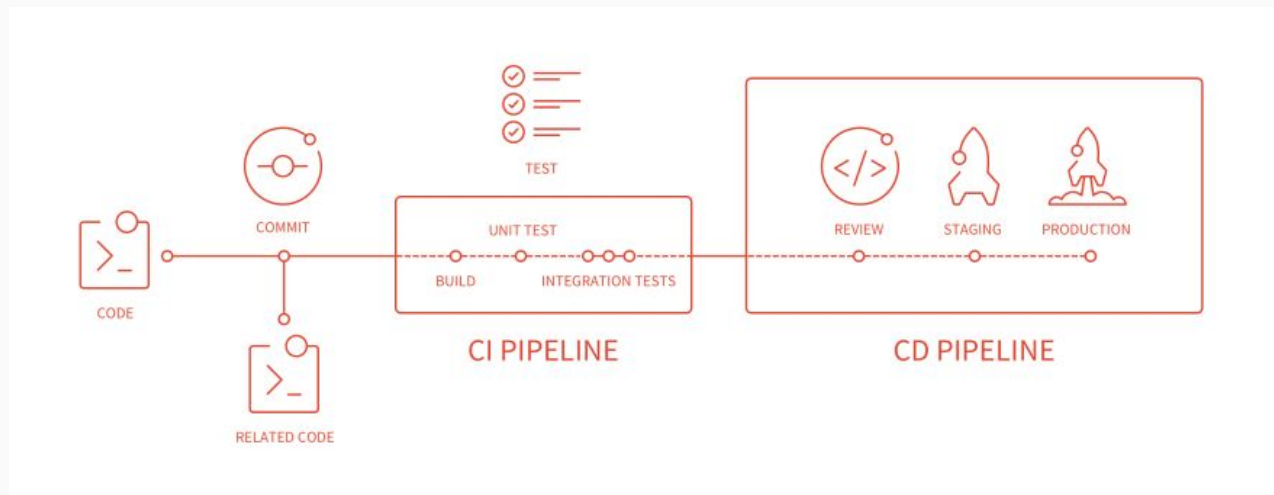
Хрулев Павел
.NET Разработчик в Ростелеком ИТ

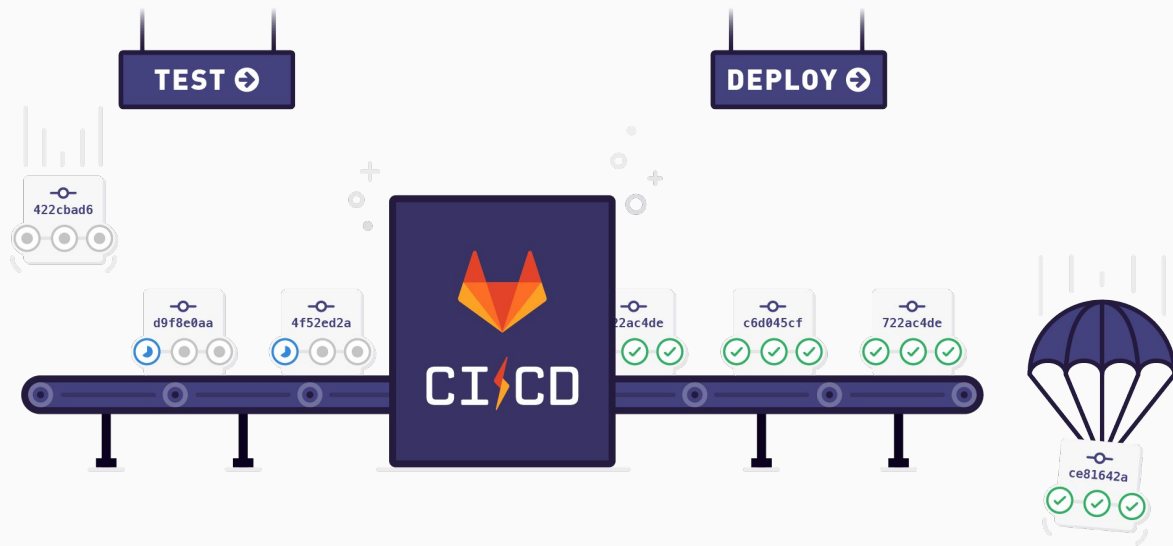
1. Что такое CI и что такое CD. GitLab CI.
2. Описание `.gitlab-ci.yml` для сборки проекта.
3. Активация режима Developer PowerShell.
4. Установка версии библиотек и исполняемого файла.
5. Статический анализ кода с помощью SonarQube.
6. Подсчет тестового покрытия с помощью OpenCover.
7. Деплой в Telegram.

Что такое CI и что такое CD

CI - автосборка и тестирование приложения при любых изменениях.

CD - доставка приложения до тестовых/боевых стендов или пользователей.





Ключевые возможности:

- интеграция с GitLab;
- быстрое развертывание;
- масштабируемость;
- гибкая настройка пайплайна.

1. [Устанавливаем Git for Windows.](#)
2. [Устанавливаем Build Tools.](#)
3. [Устанавливаем xUnitRunner:](#)
 - a. скачать [NugetPackageExprorer](#);
 - b. открыть с его помощью nuget-пакет xUnitRunner-а и распаковать содержимое по пути /tools/net52 в папку C:\Tools\xUnitRunner.

Build Tools для Visual Studio 2019

Инструменты Build Tools позволяют выполнять сборку проектов Visual Studio в интерфейсе командной строки. Поддерживаются следующие проекты: ASP.NET, Azure, классические приложения C++, ClickOnce, контейнеры, .NET Core, классические приложения .NET, Node.js, Office и SharePoint, Python, TypeScript, модульные тесты, UWP, WCF и Xamarin.

Downloading Git



You are downloading the latest (**2.28.0**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **3 months ago**, on 2020-07-28.

[Click here to download manually](#)

Other Git for Windows downloads

Git for Windows Setup

[32-bit Git for Windows Setup.](#)



[64-bit Git for Windows Setup.](#)

Загрузка ↓

GitLab CI. Устанавливаем и регистрируем GitLab Runner

1. Создаем на диске C папку GitLab-Runner.
2. [Скачиваем Runner](#) и кладем его в созданную папку.
3. Запускаем команду установки в PowerShell, запущенном с правами администратора:
`.\gitlab-runner.exe install`
4. Регистрируем Runner, указывая URL и токен из GitLab, а также имя, тэги и используемый shell:
`.\gitlab-runner.exe register`
5. Проверяем, что регистрация завершена успешно.

Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:
GitLab URL 
3. Use the following registration token during setup:
Token 
4. Start the Runner!

Runners activated for this project

 S78xE_dn...  

win10_eval_dotnet

#665

dotnetcictest

Добавляем шаблон для сборки .gitlab-ci.yml

```
variables:
  # Максимальное количество параллельно собираемых проектов при сборке решения; зависит от количества ядер ПК, выбранного для сборки
  MSBUILD_CONCURRENCY: 4
  # Тут куча путей до утилит, которые просто обязаны лежать там, где ожидается
  NUGET_PATH: 'C:\Tools\Nuget\nuget.exe'
  MSBUILD_PATH: 'C:\Program Files (x86)\Microsoft Visual Studio\2017\BuildTools\MSBuild\15.0\Bin\msbuild.exe'
  XUNIT_PATH: 'C:\Tools\xunit.runner.console.2.3.1\xunit.console.exe'
  TESTS_OUTPUT_FOLDER_PATH: '.\tests\CiCdExample.Tests\bin\Release\'

# Тут указываются стадии сборки. Указывайте любые названия которые вам нравятся, но по умолчанию используют три стадии: build, test и deploy.
# Стадии выполняются именно в такой последовательности.
stages:
  - build
  - test

# Далее описываются задачи (job-ы)
build_job:
  stage: build # указание, что задача принадлежит этапу build
  # tags: windows # если тут указать тэг, задача будет выполняться только на Runner-е с указанным тэгом
  only: # для каких сущностей требуется выполнять задачу
    - branches
  script: # код шага
    - '& "$env:NUGET_PATH" restore'
# сборка; ключ clp:ErrorsOnly оставляет только вывод ошибок; ключ nr:false завершает инстансы msbuild
    - '& "$env:MSBUILD_PATH" /p:Configuration=Release /m:$env:MSBUILD_CONCURRENCY /nr:false /clp:ErrorsOnly'
  artifacts: # где по завершении задачи будут результаты, которые надо сохранить в gitlab (т.н. артефакты) и которые можно будет передать другим задачам по цепочке
    expire_in: 2 days # сколько хранить артефакты
    paths: # список путей, по которым находятся файлы для сохранения
      - '$env:TESTS_OUTPUT_FOLDER_PATH'

test_job:
  stage: test
  only:
    - branches
  script:
    - '& "$env:XUNIT_PATH" "$env:TESTS_OUTPUT_FOLDER_PATH\CiCdExample.Tests.dll"'
  dependencies: # указание, что для запуска этой задачи требуется успешно завершенная задача build_job
    - build_job
```

Активируем режим Developer PowerShell for VS

Проблема: неудобно прописывать абсолютные пути к разным установкам Visual Studio.

Решение: утилита vswhere для поиска всех инсталляций Visual Studio на компьютере + трансформация PowerShell в Developer PowerShell.

```
variables:  
  VSWHERE_PATH: '%ProgramFiles(x86)%\Microsoft Visual Studio\Installer\vswhere.exe'
```

```
.before_msbuild: &enter_vsdevshell  
  before_script:  
    - chcp 65001  
    - '$vsWherePath = [System.Environment]::ExpandEnvironmentVariables($env:VSWHERE_PATH)'  
    - '& $vsWherePath -latest -format value -property installationPath -products Microsoft.VisualStudio.Product.Community Microsoft.VisualStudio.Product.BuildTools | '  
    - 'Tee-Object -Variable visualStudioPath'  
    - 'Join-Path "$visualStudioPath" "\\Common7\Tools\Microsoft.VisualStudio.DevShell.dll" | Import-Module'  
    - 'Enter-VsDevShell -VsInstallPath:"$visualStudioPath" -SkipAutomaticLocation'
```


Проставляем версию библиотек и приложения

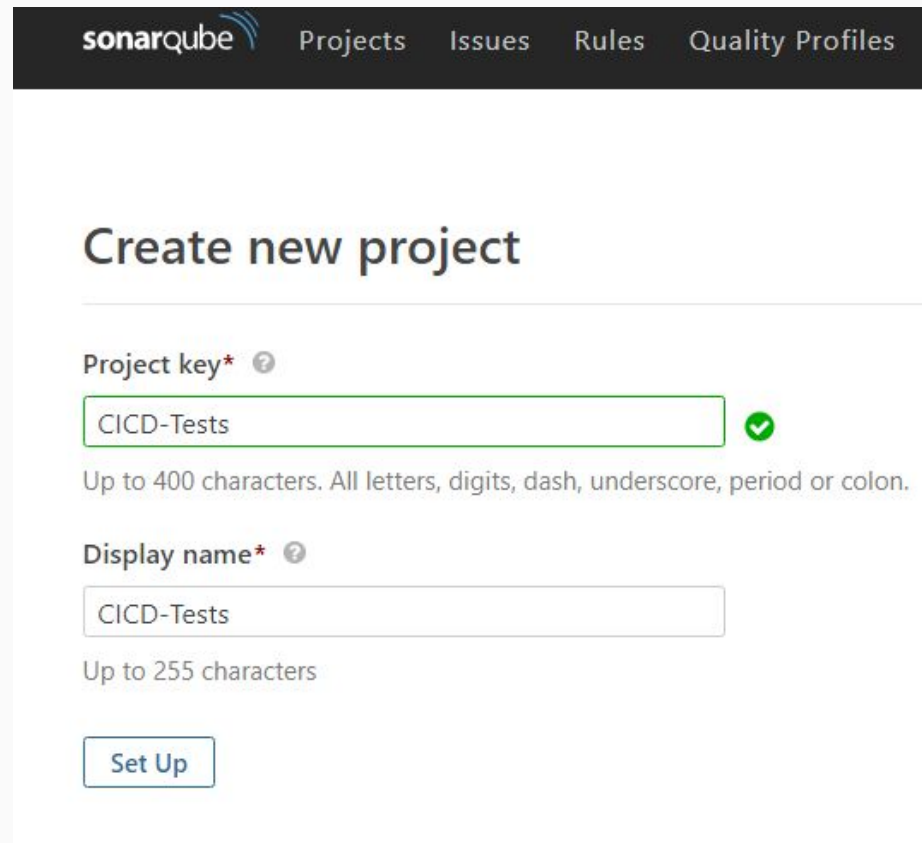
1. Устанавливаем во все проекты приложения NuGet-пакет: [MSBuild.AssemblyVersion](#).
2. Помечаем коммит именованным тэгом.
3. `git describe --long`: отдает версию вида 1.2.3.20-g342d6c.
4. Устанавливаем параметр версии `AssemblyVersionNumber` в `msbuild` для простановки версии во все библиотеки и исполняемый файл приложения.
5. `.gitlab-ci.yml` будет иметь следующий вид:

```
- '$versionGroup = git describe --long | Select-String -Pattern "(?<major>[0-9]+)\.(?<minor>[0-9]*)\.(?<patch>[0-9]*)\-(?<commit>[0-9]+)\-g[0-9a-f]+" | Select-Object -First 1'
- '[int]$major, [int]$minor, [int]$patch, [int]$commit = $versionGroup.Matches[0].Groups["major", "minor", "patch", "commit"].Value'
- '[string]$version = "$major.$minor.$patch.$commit"'
- 'Write-Host "Building Release | AnyCpu application v$version..."'
- 'msbuild /p:Configuration="Release" /p:AssemblyVersionNumber=$version /p:Platform="Any CPU" /m:$env:MSBUILD_CONCURRENCY /nr:false /clp:ErrorsOnly'
```

SonarQube — это платформа, предназначенная для непрерывного анализа и измерения качества кода.

Ключевые возможности:

- генерация отчетов о дублировании кода, соблюдении стандартов кодирования, покрытии кода тестами, возможных ошибках в коде, техническом долге и т.д.;
- сохранение истории метрик и построение графиков их изменения;
- интеграция с внешними инструментами.



The screenshot shows the 'Create new project' interface in SonarQube. At the top, there is a navigation bar with the SonarQube logo and links for 'Projects', 'Issues', 'Rules', and 'Quality Profiles'. The main heading is 'Create new project'. Below this, there are two input fields. The first is 'Project key*' with a help icon, containing the text 'CICD-Tests' and a green checkmark icon to its right. Below the input field is a note: 'Up to 400 characters. All letters, digits, dash, underscore, period or colon.' The second field is 'Display name*' with a help icon, also containing 'CICD-Tests'. Below it is a note: 'Up to 255 characters'. At the bottom of the form is a 'Set Up' button.

sonarqube Projects Issues Rules Quality Profiles

Create new project

Project key* ?

CICD-Tests ✓

Up to 400 characters. All letters, digits, dash, underscore, period or colon.

Display name* ?

CICD-Tests

Up to 255 characters

Set Up

1. Создаем на диске C папку Tools\SonarScanner.
2. [Скачиваем](#) SonarScanner и распаковываем его в созданную папку.
3. [Скачиваем](#) и устанавливаем Java Runtime.

SonarScanner for MSBuild

By [SonarSource](#) | [GNU LGPL 3](#) | [Issue Tracker](#)

4.10

[Show more versions](#)

2020-06-29

Support FIPS compliant cryptographic algorithm

[.NET Framework 4.6+](#) [.NET Core 2.0+](#) [.NET Core Global Tool](#) [Release notes](#)



[Download](#) [Help](#)

64-bit Java for Windows

Recommended Version 8 Update 261 (filesize: 79.19 MB)




Release date July 14, 2020

SonarQube. Настраиваем проект для запуска статического анализатора

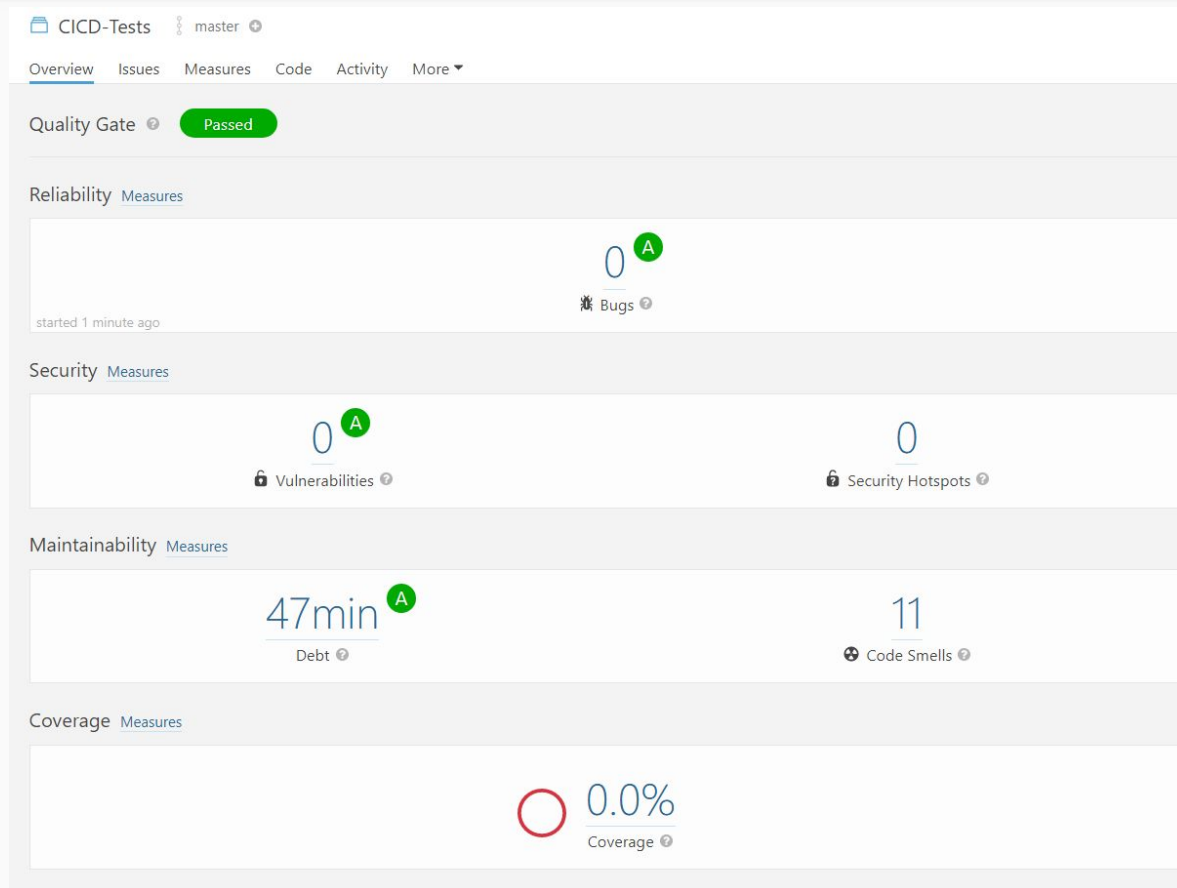
1. Прописываем ключ проекта, URL SonarQube и токен авторизации в параметрах проекта в GitLab.
2. Обновляем .gitlab-ci.yml.
3. Теперь при каждой сборке ветки master в SonarQube будет отправляться подробный анализ кода.

Variables ?

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they can be masked so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. [More information](#)

Type	Key	Value	Protected	Masked	Environments	
Variable	SONARQUBE_AUTH_TOKEN	*****	×	✓	All (default)	
Variable	SONARQUBE_HOST_URL	*****	×	×	All (default)	
Variable	SONARQUBE_PROJECT_KEY	*****	×	×	All (default)	
						<div>Reveal values Add Variable</div>

SonarQube. Смотрим результаты статического анализа



OpenCover. Добавляем запуск автотестов и генерацию отчета о покрытии

OpenCover — это утилита, позволяющая проводить подробный анализ покрытия кода тестами.

Установка:

1. Создаем на диске C папку Tools\OpenCover.
2. [Скачиваем](#) OpenCover и распаковываем его в созданную папку.
3. Дописываем переменные к секции variables в .gitlab-ci.yml.

```
OPENCover_PATH: 'C:\Tools\OpenCover\OpenCover.Console.exe'
OBJECTS_TO_TEST_REGEX: '^(FriendStorage)[^\\n]*\\.(dll|exe)$'
OPENCover_FILTER: '+[FriendStorage.*]* -[*UITests]* -[*AssemblyInfo]*'
OPENCover_REPORT_FILE_PATH: '.\cover.xml'
```

4. Модифицируем задачу тестирования, чтобы она включала команды вызова OpenCover.

OpenCover. Смотрим отчет о покрытии тестами в SonarQube

CICD-Tests

View as

List



to select files



to navigate

17 files

Coverage 41.6%



New code: since previous version

	Coverage	Uncovered Lines	Uncovered Conditions
 FriendStorage.UI/App.xaml.cs	0.0%	6	–
 FriendStorage.UI/Startup/Bootstrapper.cs	0.0%	10	–
 FriendStorage.UI/Command/DelegateCommand.cs	0.0%	11	6
 FriendStorage.DataAccess/FileDataService.cs	0.0%	33	10
 FriendStorage.UI/DataProvider/FriendDataProvider.cs	0.0%	14	–
 FriendStorage.UI/View/FriendEditView.xaml.cs	0.0%	3	–
 FriendStorage.UI/View/MainWindow.xaml.cs	0.0%	8	–

Deploy. Отправляем собранное приложение в Telegram

1. [Регистрируем бота](#) и создаем канал для публикации.
2. Добавляем бота в канал.
3. Узнаем идентификатор канала с использованием бота @getmyid_bot.
4. Устанавливаем на билд-машине PowerShell 7.
5. Устанавливаем модуль PoshGram для PowerShell 7, запущенном с правами администратора: `Install-Module -Name "PoshGram" -Scope AllUsers`
6. Создаем на диске C скрипт в папке Tools\Scripts:



```
param(  
    [string] $botToken,  
    [Int64] $chatID,  
    [string] $filePath,  
    [string] $caption  
)  
  
if ((Get-Command "Send-TelegramLocalDocument" -errorAction SilentlyContinue)) {  
    Send-TelegramLocalDocument -BotToken $botToken -ChatID $chatID -File $filePath -Caption $caption  
}
```


Deploy. Отправляем собранное приложение в Telegram

7. Добавляем stage и job для деплоя в .gitlab-ci.yml:

```
deploy_telegram:  
  when: manual  
  stage: deploy  
  tags: [dotnetcictest]  
  dependencies:  
    - build_job  
  only:  
    - /^master$/  
  script:  
    - 'if (Get-Command "pwsh.exe") {'  
    - ' pwsh.exe -NoLogo -NonInteractive -NoProfile -File "$env:DEPLOY_TELEGRAM_SCRIPT" -BotToken "$env:TELEGRAM_BOT_TOKEN" -ChatID "$env:TELEGRAM_CHAT_ID" -File "$e'  
    - '}'
```

8. Добавляем переменные в настройках проекта:

Variable	TELEGRAM_BOT_TOKEN	*****	×	✓	All (default)	
Variable	TELEGRAM_CHAT_ID	*****	×	✓	All (default)	

9. Запускаем deploy job после успешной сборки.
10. Получаем сообщение с прикрепленными файлами в канале в телеграме.

1. Настроена сборка приложения и запуск тестов.
2. Реализована простановка версии из тэга.
3. Можно посмотреть анализ кода в SonarQube.
4. Можно увидеть анализ покрытия кода тестами.
5. Организована доставка приложения пользователям.

Спасибо за внимание!

Telegram для связи:

Хрулев Павел
@pashakhrulev

Ссылки на презентацию и тестовый проект:

1. <https://docs.google.com/presentation/d/1bfQ4kilLGPSu6G4hocgjYBoeXNb2jHh9wVYsrD1UOPk/>
2. <https://github.com/Rostelecom-IT/Gitlab-CI-Example-for-dotNET>
3. <https://gitlab.com/Rostelecom-IT/Gitlab-CI-Example-for-dotNET>