

Экосистема open source проектов

Полезные инструменты и сервисы

Юрий Литвинов
yurii.litvinov@gmail.com

22.03.2019г

Continuous Integration

Непрерывная интеграция — практика слияния всех изменений по несколько раз в день, сборки их в известном окружении и запуска юнит-тестов.

- ▶ Автоматический билд
 - ▶ Всё, что нужно для сборки, есть в репозитории, может быть получено на чистую (ну, практически) машину и собрано одной консольной командой
- ▶ Большое количество юнит-тестов, запускаемых автоматически
- ▶ Выделенная машина, слушающая репозиторий и выполняющая билд
 - ▶ Чаще всего каждый билд запускается на заранее настроенной виртуалке

Continuous Integration


- ▶ Извещение всех разработчиков о статусе
 - ▶ Если билд не прошёл, разработка приостанавливается до его починки
- ▶ Автоматическое выкладывание
- ▶ Пока билд не прошёл, задача не считается сделанной
 - ▶ Короткие билды (<10 мин.)
 - ▶ deployment pipeline
 - ▶ Отдельная машина для сборки, для коротких тестов, для длинных тестов, для выкладывания

AppVeyor



- ▶ <https://www.appveyor.com/> — пример бесплатной для open source-проектов облачной CI-системы
- ▶ Виртуальная машина с ОС Windows и настроенными инструментами сборки .NET-приложений
 - ▶ Windows Server 2016 + VS 2017 или Windows Server 2012 R2 + VS 2015
- ▶ Интегрируется с GitHub-ом, Slack-ом, умеет деплоить
- ▶ Собирает по умолчанию системой сборки MSBuild
 - ▶ Можно переубедить и собирать хоть C++-приложения
- ▶ Окружение настраивается конфигурационным файлом или “вручную” из скрипта сборки

AppVeyor, настройка сборки



- ▶ Зайти на <https://www.appveyor.com/> по GitHub-аккаунту
- ▶ Добавить проект (разрешив AppVeyor просматривать список репозиторий на гитхабе)
- ▶ Положить в корень проекта файл `appveyor.yml` с конфигурацией сборки
 - ▶ Пустой тоже ок, это конфигурация по умолчанию, VS 2015 + MSBuild, ищет `.sln` в корне репозитория и пытается его собрать
- ▶ Закоммитить и запустить, это инициирует процесс сборки
- ▶ Результаты будут видны прямо на гитхабе, у каждого коммита и в пуллреквесте:

 All checks have passed
2 successful checks

[Hide all checks](#)

  continuous-integration/appveyor/pr — AppVeyor build succeeded

[Details](#)

  continuous-integration/travis-ci/pr — The Travis CI build passed

[Details](#)

AppVeyor, пример файла конфигурации

image: Visual Studio 2017

init:

- git config --global core.autocrlf input

build_script:

- cmd: build.cmd All

test: off

version: 0.0.1.{build}

artifacts:

- path: bin
name: bin

AppVeyor, жизненный цикл сборки

- ▶ Запуск скриптов из раздела `init`
- ▶ Клонирование репозитория, переход в его корень
- ▶ Запуск скриптов из раздела `install`
- ▶ Запуск скриптов из раздела `before_build`
- ▶ Запуск `msbuild` (или скрипта из раздела `build_script`)
- ▶ Запуск скриптов из раздела `after_build`
- ▶ Поиск и запуск тестов (перед — `before_test`, после — `after_test`)
- ▶ Упаковка и выкладывание собранного
- ▶ Финализация (`on_success/on_failure` и `on_finish`)

Билд ограничен 60 минутами

AppVeyor, Build matrix

- ▶ Предназначена для сборки проекта в разном окружении
 - ▶ Операционная система
 - ▶ Платформа
 - ▶ Конфигурация
 - ▶ Переменные окружения

Пример:

configuration:

- Debug
- Release

environment:

matrix:

- MY_VAR: A
- MY_VAR: B

Способ эксплуатировать матрицу для сборки домашек

environment:

matrix:

- solution_name: <путь от корня репозитория>/solution1.sln
- solution_name: <путь от корня репозитория>/solution2.sln

build_script:

- msbuild %solution_name%

Небольшое отступление про сборку из консоли

В Windows, остальные и так умеют

- ▶ Developer Command Prompt
- ▶ Основные консольные команды: `cd`, `dir`
- ▶ Переменные окружения, `PATH`
- ▶ `msbuild`
- ▶ NuGet Command Line
- ▶ Как сделать жизнь более удобной
 - ▶ FAR (<https://www.farmanager.com/>)
 - ▶ Chocolatey (<https://chocolatey.org/>)

Travis

- ▶ <https://travis-ci.org/> — ещё одна бесплатная для Open Source CI-система
- ▶ Linux и OS X
- ▶ Умеет всё, что и AppVeyor
- ▶ Собирать проект двумя разными CI-системами вполне ок
 - ▶ AppVeyor под винду, Travis под линукс

Travis, настройка сборки

- ▶ Установить commit hook на гитхабе
 - ▶ Travis умеет это делать сам, надо залогиниться под своим GitHub-аккаунтом на Travis и выбрать нужный репозиторий в профиле
- ▶ Добавить .travis.yml в корень репозитория
- ▶ Закоммитить и запустить, это иницирует процесс сборки
 - ▶ Коммит, где в комментарии есть подстрока “[ci skip]”, игнорируется Travis-ом, остальные он собирает

Travis, примеры

- ▶ Веб-приложение из нескольких сервисов на Java:
 - ▶ <https://github.com/qreal/wmp/blob/master/.travis.yml>
- ▶ Относительно большое десктопное приложение на C++:
 - ▶ <https://github.com/qreal/qreal/blob/master/.travis.yml>
- ▶ Билд в Docker-окружении (все зависимости носим с собой):
 - ▶ <https://github.com/trikset/trikRuntime/blob/master/.travis.yml>

Анализ тестового покрытия, CodeCov

- ▶ <https://codecov.io/>
- ▶ Визуализатор для функциональности компиляторов или специальных инструментов по слежению за исполнявшимися строками
- ▶ Чем больше операторов было исполнено во время тестового прогона, тем меньше вероятность пропустить баг
 - ▶ 100% покрытие не гарантирует работоспособность программы
- ▶ Интегрируется с гитхабом (комментит пуллреквесты информацией о тестовом покрытии)
- ▶ Пример конфигурации для .NET, AppVeyor и Travis:
 - ▶ <https://github.com/codecov/example-csharp>

Статический анализ, Codacy

- ▶ <https://www.codacy.com/>
- ▶ Ищет типичные ошибки: потенциальные баги, стайлгайд, мёртвый код, производительность и т.д.
- ▶ Поддерживает много языков (в том числе C#, C++, Java, Kotlin, Python, Scala)
- ▶ Не требует дополнительных манипуляций с репозиторием
- ▶ Очень настраиваема

Инструменты планирования, Trello

- ▶ <https://trello.com/>
- ▶ Интерактивная доска с карточками, организованными в списки
- ▶ Карточки легко редактируются и перетаскиваются между списками
 - ▶ Типичные списки: TODO, In Progress, Done (возможны варианты)
- ▶ Поддерживает дедлайны, чеклисты, вложения, комментарии, голосования, метки
- ▶ Легковесный инструмент планирования, подходящий, тем не менее, и для больших проектов

Инструменты планирования, Pivotal Tracker

- ▶ <https://www.pivotaltracker.com>
- ▶ Более “тяжеловесный” инструмент, ориентированный на Scrum
- ▶ Всего три списка
 - ▶ Icebox — что было бы неплохо сделать
 - ▶ Backlog — запланированные задачи
 - ▶ Current — задачи на текущую итерацию
- ▶ Задачи можно оценивать, задачи имеют тип и статус
 - ▶ По оценкам задач и статистике работы команды считается team velocity, позволяющая предсказать линейные сроки
- ▶ Есть релизы с дедлайнами, метки, еріс-и, чеклисты, вложения, комментарии
- ▶ Умеет считать статистику, рисовать графики (burndown charts)

Средства коммуникации, Slack и Gitter

- ▶ Instant messenger-ы, ориентированные на команды и интегрированные со средствами разработки
 - ▶ Информация о коммитах и пуллреквестах
 - ▶ Статус CI
 - ▶ Другие тулы
- ▶ Синтаксическая подсветка (markdown), вложения, отображение картинок, ...
- ▶ Gitter интегрирован с гитхабом и “более открыт” (предназначается прежде всего для общения сообщества)
- ▶ Slack интегрирован с чем угодно, предназначается прежде всего для общения внутри команды

GitHub: Issues, Projects, Wiki, Pages

- ▶ GitHub сам многое умеет
- ▶ Issues — довольно удобный багтрекер
 - ▶ Майлстоуны, дедлайны, метки на багах, возможность закрывать баги автоматически (если в сообщении коммита есть “close” или “fix” и #<номер бага>)
 - ▶ Пуллреквест тоже считается Issue
- ▶ Projects — представляет Issues в виде набора списков, между которыми их можно перетаскивать в духе Trello
- ▶ Wiki — викистраницы, куда можно выкладывать полезную информацию о проекте
 - ▶ Тоже git-репозиторий
- ▶ Pages — хостинг для статических сайтов <имя проекта>.github.io

Авторское право

- ▶ Open source-кодом можно пользоваться, только если автор явно это разрешил, так что просто код на GitHub — не совсем open source
- ▶ Бывают исключительные и личные неимущественные права
 - ▶ Личные неимущественные права неотчуждаемы
 - ▶ Исключительные права можно передать
 - ▶ Права появляются в момент создания произведения и принадлежат автору
 - ▶ Если произведение создано по служебному заданию — работодателю
 - ▶ Знак копирайта служит только для информирования, регистрация прав не требуется
 - ▶ Соавторы владеют произведением в равной степени
- ▶ Идея не охраняется, охраняется её физическое выражение

Open source-лицензии

- ▶ Лицензия — способ передачи части прав на произведение
- ▶ Пример — “Do what the **** you want to public license”
 - ▶ “Want to” может включать в себя патентование произведения и подачу в суд на автора за нарушение патента, поэтому обычно лицензии более длинны и унылы
 - ▶ В России и Европе программы не патентуют, в США — да
- ▶ Каждый нормальный open source-проект должен иметь лицензию

Open source-лицензии

- ▶ Часто используемые open source-лицензии:
 - ▶ GPL, LGPL (GPL вирусная, поэтому использовать её, внезапно, плохая практика)
 - ▶ MIT License
 - ▶ Apache License 2.0 (может применяться пофайлово)
 - ▶ BSD License (в разных вариантах)
 - ▶ The Unlicense — явная передача произведения в Public Domain
 - ▶ Семейство лицензий Creative Commons — не для софта, но хорошо подходит для ресурсов (картинок, текстов и т.д.)