

# Continuous integration для C++ разработчика

Филонов Павел  
Pavel.Filonov@kaspersky.com

C++ Russia 2017  
Slides: <http://goo.gl/raUUcD>

23 февраля 2017 г.

## Вместо вступления

Зачем мы здесь?

# Как все начиналось



C++



C++



# Наши дни



# Содержание

- Основные понятия и определения
- Часть 1. Работаем с GitHub
  - Hello, world!
  - Запускаем сборку на TravisCI и Appveyor
  - Разрешаем зависимости с помощью conan.io
  - Настраиваем сборку под множество целей
  - Собираем свои пакеты с conan.io
- Часть 2. Работаем с GitLab
  - Разворачиваем Gitlab с помощью VPS и Docker
  - Зеркалируем зависимости и пакеты conan.io
  - Настраиваем Gitlab CI
  - Пакуем ПО в контейнеры
- Заключение

# Основные понятия и определения

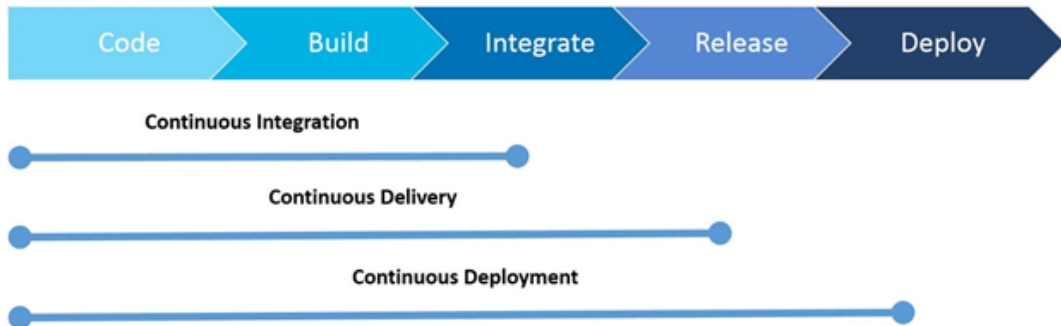
**Непрерывная интеграция** (Continuous integration, CI) — практика, в которой исходный код разработчиков постоянно интегрируется в основную ветку разработки.

**Непрерывная доставка** (Continuous delivery, CD) — практика, которая обеспечивает возможность выпуска продукта в любое время.

**Непрерывное развертывание** (Continuous deployment, CD) — практика, в которой версия продукта обновляется автоматически с добавлением в основную ветку нового функционала.



# CI/CD/CD workflows



Ref: [4]

# Что нам потребуется на laptop

- умение программировать на C++ (спасибо, Кэп!)
- laptop с любой ОС
- один из компиляторов:
  - Visual Studio  $\geq$  2015
  - g++  $\geq$  5.4
  - clang++  $\geq$  3.9
  - code  $\geq$  8.0
- make  $\geq$  3.6 (нужно понимать синтаксис CMakeLists.txt)
- git  $\geq$  2.10
- python  $\geq$  3.5
- pip3  $\geq$  9.0
- учетная запись на github.com
- учетная запись на travis-ci.org (привязанная к github)
- учетная запись на appveyor.com (привязанная к github)
- учетная запись на conan.io
- учетная запись на digitalocean.com

# Пишем Hello, world

- 1 Создаем репозиторий на github.com
- 2 Клонировем его на машину разработчика

```
git clone https://github.com/<username>/<repo>.git
```

- ### 3 Создаем структуру проекта

hello/

```
|-- CMakeLists.txt
```

| -- LICENSE

| -- README.md

```
|-- src
```

```
|-- main.cpp
```

- #### 4 Пишем `main.cpp`

```
1 #include <iostream>
```

2

```
3  int main() {
```

```
4      std::cout << "Hello, world!" << std::endl;
```

5 }

## 5 Пишем CMakeLists.txt

```
1 cmake_minimum_required(VERSION 2.8.7)
2
3 project(hello CXX)
4
5 set(${PROJECT_NAME}_SRC
6     src/main.cpp
7 )
8
9 add_executable(${PROJECT_NAME} ${${PROJECT_NAME}_SRC})
```

## 6 Собираем локально

```
1 mkdir build && cd build
2 cmake ..
3 cmake --build .
4 ./hello
```

## Подключаем TravisCI

- 1 Регистрируемся на [travis-ci.org](https://travis-ci.org)
- 2 Подключаем к репозиторию сборку на travis
- 3 Добавляем [.travis.yml](#)

```
1 dist: trusty
2 language: cpp
3
4 script:
5     - mkdir build && cd build
6     - cmake ..
7     - cmake --build .
8     - ./hello
```

- Добавляем ссылку на бедж в README.md
- добавляем изменения в репо  
`git push origin master`
- Смотрим результаты сборки на [travis-ci.org](https://travis-ci.org)

```

1  dist: trusty
2  language: cpp
3  script:
4      - mkdir build && cd build
5      - cmake .. -DCMAKE_CXX_COMPILER=g++-6
6      - cmake --build .
7      - ./hello
8
9  env:
10     - COMPILER=g++-6
11
12  addons:
13     apt:
14         sources:
15             - ubuntu-toolchain-r-test
16         packages:
17             - g++-6

```

`.travis.yml`

14







# Добавляем сборки под MacOS

## .travis.yml

```
1 matrix:
2   include:
3     - os: osx
4       osx_image: xcode6.4
5       env: BUILD_TYPE=Debug
6     - os: osx
7       osx_image: xcode7.3
8       env: BUILD_TYPE=Debug
9     - os: osx
10      osx_image: xcode8
11      env: BUILD_TYPE=Debug
12    - os: osx
13      osx_image: xcode8.1
14      env: BUILD_TYPE=Debug
15    - os: osx
16      osx_image: xcode8.2
17      env: BUILD_TYPE=Debug
```

```
1  - OS: osx
2    osx_image: xcode6.4
3    env: BUILD_TYPE=Release
4  - OS: osx
5    osx_image: xcode7.3
6    env: BUILD_TYPE=Release
7  - OS: osx
8    osx_image: xcode8
9    env: BUILD_TYPE=Release
10 - OS: osx
11   osx_image: xcode8.1
12   env: BUILD_TYPE=Release
13 - OS: osx
14   osx_image: xcode8.2
15   env: BUILD_TYPE=Release
```

✓ # 65.9	🍏 </> Xcode: xcode6.4 C++	📦 BUILD_TYPE=Debug	🕒 5 min 43 sec	🔒
✓ # 65.10	🍏 </> Xcode: xcode7.3 C++	📦 BUILD_TYPE=Debug	🕒 4 min 40 sec	🔒
✓ # 65.11	🍏 </> Xcode: xcode8 C++	📦 BUILD_TYPE=Debug	🕒 4 min 21 sec	🔒
✓ # 65.12	🍏 </> Xcode: xcode8.2 C++	📦 BUILD_TYPE=Debug	🕒 4 min 57 sec	🔒
✓ # 65.13	🍏 </> Xcode: xcode6.4 C++	📦 BUILD_TYPE=Release	🕒 4 min 25 sec	🔒
✓ # 65.14	🍏 </> Xcode: xcode7.3 C++	📦 BUILD_TYPE=Release	🕒 4 min 32 sec	🔒
✓ # 65.15	🍏 </> Xcode: xcode8 C++	📦 BUILD_TYPE=Release	🕒 4 min 19 sec	🔒
✓ # 65.16	🍏 </> Xcode: xcode8.2 C++	📦 BUILD_TYPE=Release	🕒 4 min 57 sec	🔒

# Добавляем сборку под Windows

## ❶ Добавляем [appveyor.yml](#)

```
1  build:
2
3  build_script:
4    - mkdir build && cd build
5    - cmake -G "Visual Studio %TOOLCHAIN_VERSION% Win64" ..
6    - cmake --build . --config %BUILD_TYPE%
7
8  test_script:
9    - cmd: "%BUILD_TYPE%\\hello"
10
11 environment:
12   matrix:
13     - TOOLCHAIN_VERSION: 14
14       BUILD_TYPE: Release
15     - TOOLCHAIN_VERSION: 14
16       BUILD_TYPE: Debug
```

- ② Регистрируемся на [appveyor.com](https://appveyor.com) через GitHub
- ③ Подключаем сборку для проекта
- ④ Запускаем сборку вручную

# cpp-russia-hello

[LATEST BUILD](#)[HISTORY](#)[DEPLOYMENTS](#)[SETTINGS](#)[NEW BUILD](#)[RE-BUILD COMMIT](#)

## fix travis build script

a day ago by Pavel Filonov



feature/conan



7e605f9d

**1.0.38**

a day ago in 4 min 16 sec

[JOBS](#)

JOB NAME

TESTS

DURATION

Environment: TOOLCHAIN\_VERSION=14, BUILD\_TYPE=Release

29 sec

Environment: TOOLCHAIN\_VERSION=12, BUILD\_TYPE=Release

37 sec

Environment: TOOLCHAIN\_VERSION=10, BUILD\_TYPE=Release

34 sec

Environment: TOOLCHAIN\_VERSION=14, BUILD\_TYPE=Debug

37 sec

Environment: TOOLCHAIN\_VERSION=12, BUILD\_TYPE=Debug

49 sec

Environment: TOOLCHAIN\_VERSION=10, BUILD\_TYPE=Debug

36 sec

# Добавляем зависимости с conan

## ❶ Пишем `conanfile.txt`

```
1  [requires]
2  gtest/1.7.0@lasote/stable
3
4  [generators]
5  cmake
6
7  [options]
8  gtest:shared=False
```

## ❷ Подключаем к `CMakeLists.txt`

```
1  include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
2  conan_basic_setup()
3
4  target_link_libraries(${PROJECT_NAME} ${CONAN_LIBS})
```



## ① Меняем `.travis.yml`

```
1  install:
2    - ./travis/install.sh
3
4  script:
5    - ./travis/run.sh
```

## ② `install.sh`

## ③ `run.sh`

## ④ Меняем `appveyor.yml`

```
1  install:
2    - set PATH=%PATH%;%PYTHON%/Scripts/
3    - pip.exe install conan
4    - mkdir build && cd build
5    - conan install .. --build=missing -s build_type=%BUILD_TYPE% -s
      ↪ compiler="Visual Studio" -s compiler.version=%TOOLCHAIN_VERSION%
6
7  environment:
8    PYTHON: "C:\\Python27"
9    PYTHON_VERSION: "2.7.8"
10   PYTHON_ARCH: "32"
```

# Собираем пакет для conan

## 1 Новая структура проекта

```
hello/
|-- CMakeLists.txt
|-- LICENSE
|-- README.md
|-- conanfile.py      # вместо conanfile.txt
|-- .travis.yml
|-- appveyor.yml
|--include
|   |--hello
|       |-- hello.h
|-- src
|   |-- hello.cpp
|-- test_package      # имя test_package обязательно
|   |-- CMakeLists.txt
|   |-- conanfile.py
|   |-- test_hello.cpp
```

## ② conanfile.py

```
1  from conans import ConanFile, CMake
2
3  class HelloConan(ConanFile):
4      name = "hello"
5      version = "0.1"
6      exports = "include/*", "src/*", "CMakeLists.txt"
7      settings = "os", "compiler", "arch", "build_type"
8      generators = "cmake"
9
10     def build(self):
11         cmake = CMake(self.settings)
12         self.run('cmake "{} {}".format(self.conanfile_directory,
13                                         cmake.command_line))
14         self.run('cmake --build . {}'.format(cmake.build_config))
15
16     def package(self):
17         self.copy("*.h", dst="include", src="include")
18         self.copy("*.lib", dst="lib", src="lib")
19         self.copy("*.a", dst="lib", src="lib")
20
21     def package_info(self):
22         self.cpp_info.libs = ["hello"]
```

### ③ Пишем test\_package/conanfile.py

```
1  from conans import ConanFile, CMake
2  import os
3
4  class HelloReuseConan(ConanFile):
5      settings = "os", "compiler", "build_type", "arch"
6      requires = ("hello/0.1@testing/demo", "gtest/1.8.0@lasote/stable")
7      generators = "cmake"
8      default_options = "gtest:shared=False"
9
10     def build(self):
11         # For following issue https://github.com/conan-io/conan/issues/475
12         if (self.settings.compiler == "Visual Studio" and
13             self.settings.build_type == "Debug" and
14             not self.settings.compiler.runtime.value.endswith("d")):
15             self.settings.compiler.runtime.value += "d"
16
17         cmake = CMake(self.settings)
18         self.run('cmake "%s" %s' % (self.conanfile_directory,
19                                     cmake.command_line))
20         self.run("cmake --build . %s" % cmake.build_config)
21
22     def test(self):
23         self.run(os.sep.join([".", "bin", "test_hello"]))
```

# Проверяем

- ④ Запускаем локально

```
conan test_package
```

- ⑤ Меняем [.travis/run.sh](#)

```
conan test_package -s build_type=$BUILD_TYPE --build=missing
```

- ⑥ Меняем [appveyor.yml](#)

```
build_script:
```

```
- conan test_package --build=missing -s build_type=%BUILD_TYPE%
```

```
↪ -s compiler="Visual Studio" -s
```

```
↪ compiler.version=%TOOLCHAIN_VERSION%
```

# Загружаем пакеты на conan.io

## ① Добавляем новый шаг в [.travis.yml](#)

```
after_script:
```

```
- ./travis/upload.sh
```

## ② Добавляем новый скрипт [.travis/upload.sh](#)

```
1 conan user filonovpv -p $CONAN_PASSWORD -r conan.io
2 conan copy hello/0.1@demo/testing filonovpv/testing --all
3 conan upload hello/0.1@filonovpv/testing --all -r conan.io
```

## ③ Добавляем upload в [appveyor.yml](#)

```
1 on_success:
2   - conan user filonovpv -p %CONAN_PASSWORD% -r conan.io
3   - conan copy hello/0.1@demo/testing filonovpv/testing --all
4   - conan upload hello/0.1@filonovpv/testing -r conan.io --all
```

# Пакеты для Windows

Windows

Linux

Macos

▼ Windows/x86\_64/Visual Studio 10/Debug (c00d919bc2e3477c4d28c3e288cff1de91e65516) 0 downloads

▼ Windows/x86\_64/Visual Studio 12/Debug (2c0843cc59ff2d07e33c808b3398bc624e6b54e4) 0 downloads

▼ Windows/x86\_64/Visual Studio 12/Release (e0a02d496bbb652b6295152dfce0d3937acc0b56) 0 downloads

▼ Windows/x86\_64/Visual Studio 14/Release (63da998e3642b50bee33f4449826b2d623661505) 0 downloads

▼ Windows/x86\_64/Visual Studio 10/Release (85f780d0530411a64b0be4407b381706014b445d) 0 downloads

▼ Windows/x86\_64/Visual Studio 14/Debug (56fd5f54f07a0483c2943eea801af00e17403f93) 0 downloads

# Пакеты для Linux

Windows

Linux

Macos

▼ Linux/x86\_64/clang 3.9/Release (76b058609dd46eef2b897e5b18e0846cfe4ac5ed) 0 downloads

▼ Linux/x86\_64/gcc 4.9/Debug (87fb0b451a964f931487412a0e21bdef04c50390) 0 downloads

▼ Linux/x86\_64/gcc 5.4/Release (81607951755e61cf17149d40bed6aba5b3a079a9) 0 downloads

▼ Linux/x86\_64/clang 3.9/Debug (5c7e33ece678b972241f6943785d2ad617527725) 0 downloads

▼ Linux/x86\_64/gcc 5.4/Debug (1278e51f42508fd2430ca34e3b819b879ffb2e4b) 0 downloads

▼ Linux/x86\_64/gcc 6.2/Debug (305851e45746f570f6b62040650aefc28b90ece3) 0 downloads

▼ Linux/x86\_64/gcc 6.2/Release (5f1f71e5f81572eb63e3ce82c8c876a9e189d77b) 0 downloads

▼ Linux/x86\_64/gcc 4.9/Release (75afd4a5a43dc297f65bd5338e9887b268cb97ba) 0 downloads



# Пакеты для MacOS

[Windows](#)[Linux](#)[Macos](#)

▼ MacOS/x86\_64/apple-clang 7.3/Release (f38e981ab93119e082fa013de58daa826eaa9dcb) 0 downloads

▼ MacOS/x86\_64/apple-clang 8.0/Debug (bf9425c3e086842167338a990fcb3b9e99b21fd2) 0 downloads

▼ MacOS/x86\_64/apple-clang 6.1/Debug (a98564e4a8b41020ab4cddaf104c88910d7fc52a) 0 downloads

▼ MacOS/x86\_64/apple-clang 7.3/Debug (2b31d17272e0538065ac1b8fc29ebbeb1394ce71) 0 downloads

▼ MacOS/x86\_64/apple-clang 6.1/Release (3729370f26b53fb8ee3358b43accb0a53fd697bb) 0 downloads

▼ MacOS/x86\_64/apple-clang 8.0/Release (a47fd1f3db1f83c7cb0da8cea82a3e1683ae91ea) 0 downloads

# Multitarget builds and packages

$$\left\{ \begin{array}{c} \text{Windows} \\ \text{Linux} \\ \text{Macos} \\ \text{Andriod} \\ \text{iOS} \end{array} \right\} \times \left\{ \begin{array}{c} \text{GCC 4.4, \dots, 6.1} \\ \text{Clang 3.3, \dots, 3.8} \\ \text{VS 8, \dots, 14} \\ \text{Apple-clang 5.0, \dots, 7.3} \end{array} \right\} \times \left\{ \begin{array}{c} \text{x86} \\ \text{x86\_64} \\ \text{armv6} \\ \dots \end{array} \right\} \times \left\{ \begin{array}{c} \text{Release} \\ \text{Debug} \end{array} \right\}$$

# Conan package tools

## ① Добавляем файл `build.py`

```
1 from conan.packager import ConanMultiPackager
2
3 if __name__ == "__main__":
4     builder = ConanMultiPackager(username="filonovpv")
5     builder.add_common_builds()
6     builder.run()
```

## ② Меняем `.travis/install.sh`

```
pip install conan_package_tools
```

## ③ Меняем `.travis/run.sh`

```
python build.py
```

## ① Меняем `.travis.yml`

```
1  services:
2    - docker
3  sudo: required
4  language: python
5  env:
6    global:
7      - CONAN_UPLOAD=0
8      - CONAN_REFERENCE="hello/0.1"
9      - CONAN_USERNAME="filonovpv"
10     - CONAN_CHANNEL="testing"
11     - CONAN_TOTAL_PAGES=1
12     - CONAN_CURRENT_PAGE=1
13
14  matrix:
15     - CONAN_GCC_VERSIONS=4.6 CONAN_USE_DOCKER=1
16     - CONAN_GCC_VERSIONS=4.8 CONAN_USE_DOCKER=1
17     - CONAN_GCC_VERSIONS=4.9 CONAN_USE_DOCKER=1
18     - CONAN_GCC_VERSIONS=5.2 CONAN_USE_DOCKER=1
19     - CONAN_GCC_VERSIONS=5.3 CONAN_USE_DOCKER=1
```

## ① Меняем `appveyor.yml`

```
1  build: false
2  environment:
3      PYTHON: "C:\\Python27-x64"
4      PYTHON_VERSION: "2.7.11"
5      PYTHON_ARCH: "64"
6
7      CONAN_UPLOAD: 1
8      CONAN_REFERENCE: "hello/0.1"
9      CONAN_USERNAME: "filonovpv"
10     CONAN_CHANNEL: "testing"
11     CONAN_TOTAL_PAGES: 4
12
13     matrix:
14         - CONAN_CURRENT_PAGE: 1
15         - CONAN_CURRENT_PAGE: 2
16         - CONAN_CURRENT_PAGE: 3
17         - CONAN_CURRENT_PAGE: 4
18
19 install:
20     - set PATH=%PATH%;%PYTHON%/Scripts/
21     - C:\\Python27-x64\\Scripts\\pip.exe install conan_package_tools
22     - conan user # It creates the conan data directory
23
24 test_script:
25     - C:\\Python27-x64\\python build.py
```

# Github pricing

## Personal

**\$7** / month

Build your own projects on GitHub.com and invite collaborators to join you in unlimited private repositories.


 Free for students as part of the [Student Developer Pack](#).

[Upgrade your account](#)

## Organization

**\$9** per user / month

Work with your team on GitHub.com in unlimited private repositories. Manage team and user level permissions.


 Starting at **\$25** / month which includes your first 5 users.

[Create an organization](#)

## Enterprise

**\$21** per user / month

Host your team's code on your own servers or in a private cloud with your existing security controls.

 Sold in packs of **10** users and billed annually.

[Start a free Enterprise trial](#)

# TravisCI pricing

<p><b>\$69</b></p> <p>PER MONTH</p> <p><b>Bootstrap</b></p> <p>IDEAL FOR HOBBY PROJECTS</p> <p><b>1</b> Concurrent job</p> <ul style="list-style-type: none"><li>✓ Unlimited build minutes</li><li>✓ Unlimited repositories</li><li>✓ Unlimited collaborators</li></ul>	<p><b>\$129</b></p> <p>PER MONTH</p> <p><b>Startup</b></p> <p>BEST FOR SMALL TEAMS</p> <p><b>2</b> Concurrent jobs</p> <ul style="list-style-type: none"><li>✓ Unlimited build minutes</li><li>✓ Unlimited repositories</li><li>✓ Unlimited collaborators</li></ul> <p><a href="#">Start Trial</a></p>	<p><b>\$249</b></p> <p>PER MONTH</p> <p><b>Small Business</b></p> <p>GREAT FOR GROWING TEAMS</p> <p><b>5</b> Concurrent jobs</p> <ul style="list-style-type: none"><li>✓ Unlimited build minutes</li><li>✓ Unlimited repositories</li><li>✓ Unlimited collaborators</li></ul>	<p><b>\$489</b></p> <p>PER MONTH</p> <p><b>Premium</b></p> <p>PERFECT FOR LARGER TEAMS</p> <p><b>10</b> Concurrent jobs</p> <ul style="list-style-type: none"><li>✓ Unlimited build minutes</li><li>✓ Unlimited repositories</li><li>✓ Unlimited collaborators</li></ul>
---	--	---	--

ALL PRICES SHOWN IN USD

# Appveyor pricing

Basic	Pro	Premium
<b>\$29/month</b>	<b>\$59/month</b> \$590/year - 2 months free	<b>\$99/month</b> \$990/year - 2 months free
1 private project	<b>Unlimited</b> private projects	<b>Unlimited</b> private projects
1 concurrent job	1 concurrent job	2 concurrent jobs
1 GB build cache	5 GB build cache	20 GB build cache
Forums support	Priority technical support	Priority technical support
START FREE 14-DAY TRIAL	START FREE 14-DAY TRIAL	START FREE 14-DAY TRIAL



# Gitlab

- self-hosted
- continuous integration
- issues
- code review
- wiki



# Virtual Private Server



Google Compute Engine



# Запускаем GitLab

## 1 Запускаем GitLab `run-gitlab.sh`

```
1 docker run \  
2   --detach \  
3   --hostname <insert ip here> \  
4   --env GITLAB_OMNIBUS_CONFIG="external_url 'http://<insert ip  
↪ here>/' ; gitlab_rails['gitlab_shell_ssh_port'] = 2222;" \  
5   --publish 443:443 --publish 80:80 --publish 2222:22 --publish  
↪ 5005:5005 \  
6   --name gitlab \  
7   --restart always \  
8   --volume /srv/gitlab/config:/etc/gitlab \  
9   --volume /srv/gitlab/logs:/var/log/gitlab \  
10  --volume /srv/gitlab/data:/var/opt/gitlab \  
11  gitlab/gitlab-ce:latest
```

## 2 Создаем новый проект

## 3 Подключаем репозиторий

```
git remote add gitlab http://<server ip>/<path>
```

# Запускаем Gitlab CI Runner

## ① Запускаем контейнер `run-gitlab-runner.sh`

```
1 docker run \  
2   --detach \  
3   --name gitlab-runner \  
4   --restart always \  
5   --volume /var/run/docker.sock:/var/run/docker.sock \  
6   --volume /srv/gitlab-runner/config:/etc/gitlab-runner \  
7   --privileged \  
8   gitlab/gitlab-runner
```

## ② Регистрируем gitlab-runner

```
docker exec -it gitlab-runner gitlab-runner register
```

# Gitlab CI

① создаем в корне файл `.gitlab-ci.yml`

```
1  before_script:
2    - apt update && apt install -y cmake python-pip python-dev
3    - pip install conan
4    - conan user
5
6  stages:
7    - build
8
9  gcc-4:
10    stage: build
11    image: gcc:4
12    script:
13      - conan test_package --build=missing
14    tags:
15      - docker
```

# Результат

## Build



clang-3.9



gcc-4



gcc-5



gcc-6



# Self-hosted conan server

- 1 Запускаем conan server [run-conan-server.sh](#)

```
docker run \  
  --detach \  
  --restart always \  
  --volume /srv/conan:/root/.conan_server \  
  --publish 9300:9300 \  
  --name conan-server \  
  sdukshis/conan_server
```

- 2 Добавляем upload в [.gitlab-ci.yml](#)

# Enable gitlab docker registry

- 1 Генерируем сертификаты с помощью letsencrypt

```
apt install letsencrypt
letsencrypt certonly --manual
cd /etc/letsencrypt/live/<address>
cat cert.pem fillchain.pem > /src/gitlab/config/ssl/<address>.cert
cat privkey.pem > /src/gitlab/config/ssl/<address>.key
```

- 2 Включаем docker registry в файле /srv/gitlab/config/gitlab.rb

```
registry_external_url 'https://<address>:5005'
```



# Создаем docker контейнеры

① Создаем проект для wello на gitlab

② Создаем [Dockerfile](#)

```
1 FROM frolvlad/alpine-glibc
2
3 RUN apk add --update libstdc++ && rm -rf /var/cache/apk/*
4
5 COPY build/bin/wello /opt/cpp-russia/bin/wello
6
7 ENTRYPOINT ["/opt/cpp-russia/bin/wello"]
```

## ❶ Пишем .gitlab-ci.yml

```
1  deploy:
2    stage: deploy
3    image: docker:git
4    services:
5      - docker:dind
6    tags:
7      - docker
8    before_script:
9      - docker info
10     - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN <address>:5005
11    script:
12      - docker build -t <address>:5005/cpp-russia/wello .
13      - docker push <address>:5005/cpp-russia/wello
14    dependencies:
15      - build:gcc-4
```

# Что мы научились делать

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan
- Как настроить CD с помощью conan

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan
- Как настроить CD с помощью conan
- Как развернуть собственный GitLab

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan
- Как настроить CD с помощью conan
- Как развернуть собственный GitLab
- Как настроить Gitlab CI



# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan
- Как настроить CD с помощью conan
- Как развернуть собственный GitLab
- Как настроить Gitlab CI
- Как настроить собственный conan server

# Что мы научились делать

- Как настроить CI с помощью GitHub, TravisCI и Appveyor
- Как управлять зависимостями с помощью conan
- Как настроить CD с помощью conan
- Как развернуть собственный GitLab
- Как настроить Gitlab CI
- Как настроить собственный conan server
- Как настроить CD с помощью Docker

# Источники

- [1] Grady Booch. Object Oriented Design: with applications — Book
- [2] Martin Fowler. Continuous Integration (original version) — Blogpost
- [3] Martin Fowler. Continuous Integration — Blogpost
- [4] Continuous Integration Vs Continuous Delivery Vs Continuous Deployment
- [5] Kent Beck. Extreme Programming Explained
- [6] docs.conan.io
- [7] Nick Sarten — Travis CI and Modern C++
- [8] Travis CI - The OS X Build Environment
- [9] Вы еще не авторизуетесь по ключам? Тогда мы идем к вам
- [10] Get Docker for Ubuntu

# Источники

- [1] [GitLab Docker images](#)
- [2] [Run gitlab-runner in a container](#)
- [3] [GitLab Runner](#)
- [4] [Running your conan server](#)
- [5] [Certbot manual](#)
- [6] [GitLab Container Registry](#)