Continuous integration для C++ разработчика

Филонов Павел
Pavel.Filonov@kaspersky.com

C++ Russia 2018
Slides: http://goo.gl/raUUcD

19 апреля 2018 г.

Вместо вступления

Зачем мы здесь?

Как все начиналось


C++


C++

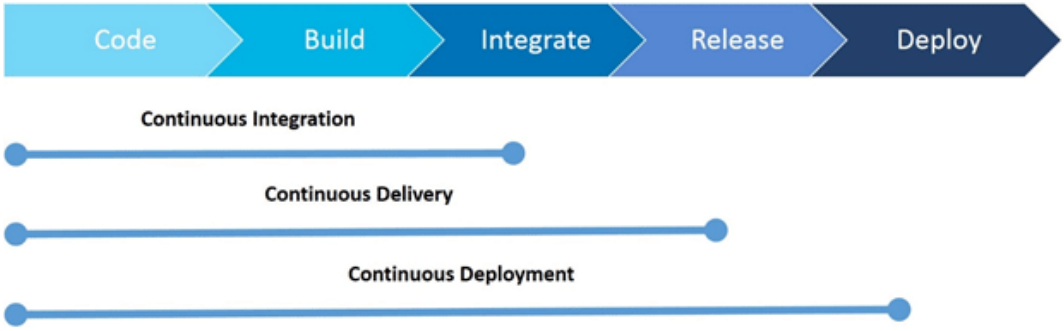
Наши дни

Содержание

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 >= 2015
  - g++ >= 5.4
  - clang++ >= 3.9
  - code >= 8.0
- make >= 3.6 (нужно понимать синтаксис CMakeLists.txt)
- git >= 2.10
- python >= 3.5
- pip3 >= 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. Клонируем 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

```cpp
#include <iostream>

int main() {
        std::cout << "Hello, world!" << std::endl;
}
```

Пишем Hello, world

5. Пишем CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.7)

project(hello CXX)

set(${PROJECT_NAME}_SRC
        src/main.cpp
)

add_executable(${PROJECT_NAME} ${${PROJECT_NAME}_SRC})
```

6. Собираем локально

```
mkdir build && cd build
cmake ..
cmake --build .
./hello
```

Подключаем TravisCI

1. Регистрируемся на travis-ci.org
2. Подключаем к репозиторию сборку на travis
3. Добавляем .travis.yml

```
dist: trusty
language: cpp

script:
  - mkdir build && cd build
  - cmake ..
  - cmake --build .
  - ./hello
```

4. Добавляем ссылку на бедж в README.md
5. добавляем изменения в репо

   `git push origin master`
6. Смотрим результаты сборки на travis-ci.org

Выбираем версию компилятора

```
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
```

Создаем матрицу сборки

.travis.yml

```
script:
  - mkdir build && cd build
  - cmake .. -DCMAKE_BUILD_TYPE=$BUILD_TYPE
  - cmake --build .
  - ./hello

env:
  - BUILD_TYPE=Release COMPILER=g++-4.9
  - BUILD_TYPE=Release COMPILER=g++-5
  - BUILD_TYPE=Release COMPILER=g++-6
  - BUILD_TYPE=Release COMPILER=clang++-3.9
  - BUILD_TYPE=Release COMPILER=clang++-4.0
  - BUILD_TYPE=Debug COMPILER=g++-4.9
  - BUILD_TYPE=Debug COMPILER=g++-5
  - BUILD_TYPE=Debug COMPILER=g++-6
  - BUILD_TYPE=Debug COMPILER=clang++-3.9
  - BUILD_TYPE=Debug COMPILER=clang++-4.0
```

Создаем матрицу сборки

```
addons:
 apt:
   sources:
     - ubuntu-toolchain-r-test
     - llvm-toolchain-trusty-3.9
     - llvm-toolchain-trusty-4.0
   packages:
     - g++-4.9
     - g++-5
     - g++-6
     - clang-3.9
     - clang-4.0
```

Build Jobs

✓	# 23.1	C++	BUILD_TYPE=Release COMPILER=g++-4.9	3 min 18 sec	
✓	# 23.2	C++	BUILD_TYPE=Release COMPILER=g++-5	3 min 5 sec	
✓	# 23.3	C++	BUILD_TYPE=Release COMPILER=g++-6	5 min 15 sec	
✓	# 23.4	C++	BUILD_TYPE=Release COMPILER=clang++-3.9	4 min 54 sec	
✓	# 23.5	C++	BUILD_TYPE=Release COMPILER=clang++-4.0	3 min 15 sec	
✓	# 23.6	C++	BUILD_TYPE=Debug COMPILER=g++-4.9	4 min 30 sec	
✓	# 23.7	C++	BUILD_TYPE=Debug COMPILER=g++-5	5 min 42 sec	
✓	# 23.8	C++	BUILD_TYPE=Debug COMPILER=g++-6	5 min 15 sec	
✓	# 23.9	C++	BUILD_TYPE=Debug COMPILER=clang++-3.9	4 min 21 sec	
✓	# 23.10	C++	BUILD_TYPE=Debug COMPILER=clang++-4.0	4 min 31 sec	

Добавляем сборки под MacOS

.travis.yml

```
matrix:
  include:
    - os: osx
      osx_image: xcode6.4
      env: BUILD_TYPE=Debug
    - os: osx
      osx_image: xcode7.3
      env: BUILD_TYPE=Debug
    - os: osx
      osx_image: xcode8
      env: BUILD_TYPE=Debug
    - os: osx
      osx_image: xcode8.1
      env: BUILD_TYPE=Debug
    - os: osx
      osx_image: xcode8.2
      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

1. Добавляем appveyor.yml

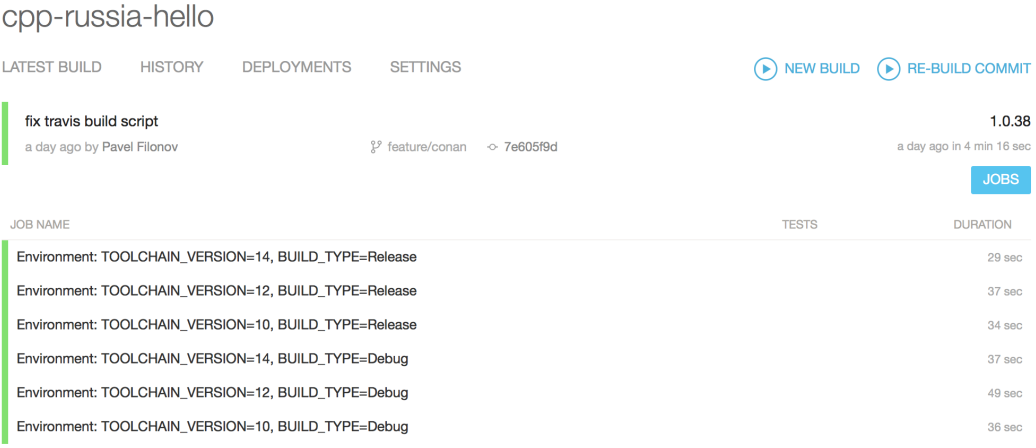
```
build:

build_script:
  - mkdir build && cd build
  - cmake -G "Visual Studio %TOOLCHAIN_VERSION% Win64" ..
  - cmake --build . --config %BUILD_TYPE%

test_script:
  - cmd: "%BUILD_TYPE%\\hello"

environment:
  matrix:
    - TOOLCHAIN_VERSION: 14
      BUILD_TYPE: Release
    - TOOLCHAIN_VERSION: 14
      BUILD_TYPE: Debug
```

2. Регистрируемся на appveyor.com через GitHub
3. Подключаем сборку для проекта
4. Запукаем 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

1. Пишем conanfile.txt

```
[requires]
gtest/1.7.0@lasote/stable

[generators]
cmake

[options]
gtest:shared=False
```

2. Подключаем к CMakeLists.txt

```
include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup()

target_link_libraries(${PROJECT_NAME} ${CONAN_LIBS})
```

1. Меняем .travis.yml

```
install:
  - ./.travis/install.sh

script:
  - ./.travis/run.sh
```

2. install.sh
3. run.sh
4. Меняем appveyor.yml

```
install:
  - set PATH=%PATH%;%PYTHON%/Scripts/
  - pip.exe install conan
  - mkdir build && cd build
  - conan install .. --build=missing -s build_type=%BUILD_TYPE%  -s
    compiler="Visual Studio" -s compiler.version=%TOOLCHAIN_VERSION%

environment:
  PYTHON: "C:\\Python27"
  PYTHON_VERSION: "2.7.8"
  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
```

❷ 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)
12         self.run('cmake "{}" {}'.format(self.source_folder,
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

```
from conans import ConanFile, CMake
import os

class HelloReuseConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    requires = ("hello/0.1@testing/demo", "gtest/1.8.0@lasote/stable")
    generators = "cmake"
    default_options = "gtest:shared=False"

    def build(self):
        # For following issue https://github.com/conan-io/conan/issues/475
        if (self.settings.compiler == "Visual Studio" and
            self.settings.build_type == "Debug" and
                not self.settings.compiler.runtime.value.endswith("d")):
            self.settings.compiler.runtime.value += "d"

        cmake = CMake(self)
        self.run('cmake "%s" %s' % (self.source_folder,
                                    cmake.command_line))
        self.run("cmake --build . %s" % cmake.build_config)

    def test(self):
        self.run(os.sep.join([".","bin", "test_hello"]))
```

Проверяем

4. Запускаем локально

```
conan test_package
```

5. Меняем .travis/run.sh

```
conan create . demo/testing -s build_type=$BUILD_TYPE --build=missing
```

6. Меняем appveyor.yml

```
build_script:
  - conan create . demo/testing  --build=missing -s
↪   build_type=%BUILD_TYPE% -s compiler="Visual Studio" -s
↪   compiler.version=%TOOLCHAIN_VERSION%
```

Загружаем пакеты на bintray.com

1. Добавляем новый шаг в .travis.yml

```
after_script:
  - ./.travis/upload.sh
```

2. Добавляем новый скрипт .travis/upload.sh

```
conan remote add bintray <YOUR_BINTRAY_REPO_URL>
conan user -p <APIKEY> -r bintray <USERNAME>
conan upload hello/0.1@demo/testing --all -r bintray
```

3. Добавляем upload в appveyor.yml

```
on_success:
  - conan remote add bintray <YOUR_BINTRAY_REPO_URL>
  - conan user -p %APIKEY% -r bintray %USERNAME%
  - conan upload hello/0.1@demo/testing -r bintray --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

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

Conan package tools

1. Добавляем файл build.py

```
from conan.packager import ConanMultiPackager

if __name__ == "__main__":
    builder = ConanMultiPackager(username="filonovpv")
    builder.add_common_builds()
    builder.run()
```

2. Меняем .travis/install.sh

```
pip install conan_package_tools
```

3. Меняем .travis/run.sh

```
python build.py
```

1. Меняем .travis.yml

```
services:
   - docker
sudo: required
language: python
env:
  global:
    - CONAN_UPLOAD=0
    - CONAN_REFERENCE="hello/0.1"
    - CONAN_USERNAME="filonovpv"
    - CONAN_CHANNEL="testing"
    - CONAN_TOTAL_PAGES=1
    - CONAN_CURRENT_PAGE=1

  matrix:
    - CONAN_GCC_VERSIONS=4.6 CONAN_USE_DOCKER=1
    - CONAN_GCC_VERSIONS=4.8 CONAN_USE_DOCKER=1
    - CONAN_GCC_VERSIONS=4.9 CONAN_USE_DOCKER=1
    - CONAN_GCC_VERSIONS=5.2 CONAN_USE_DOCKER=1
    - CONAN_GCC_VERSIONS=5.3 CONAN_USE_DOCKER=1
```

1. Меняем appveyor.yml

```
build: false
environment:
    PYTHON: "C:\\Python27-x64"
    PYTHON_VERSION: "2.7.11"
    PYTHON_ARCH: "64"

    CONAN_UPLOAD: 1
    CONAN_REFERENCE: "hello/0.1"
    CONAN_USERNAME: "filonovpv"
    CONAN_CHANNEL: "testing"
    CONAN_TOTAL_PAGES: 4

    matrix:
        - CONAN_CURRENT_PAGE: 1
        - CONAN_CURRENT_PAGE: 2
        - CONAN_CURRENT_PAGE: 3
        - CONAN_CURRENT_PAGE: 4
install:
  - set PATH=%PATH%;%PYTHON%/Scripts/
  - C:\Python27-x64\Scripts\pip.exe install conan_package_tools
  - conan user # It creates the conan data directory
test_script:
  - 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

Bootstrap

$69 PER MONTH

IDEAL FOR HOBBY PROJECTS

- **1** Concurrent job
- ✓ Unlimited build minutes
- ✓ Unlimited repositories
- ✓ Unlimited collaborators

Startup

$129 PER MONTH

BEST FOR SMALL TEAMS

- **2** Concurrent jobs
- ✓ Unlimited build minutes
- ✓ Unlimited repositories
- ✓ Unlimited collaborators

Start Trial

Small Business

$249 PER MONTH

GREAT FOR GROWING TEAMS

- **5** Concurrent jobs
- ✓ Unlimited build minutes
- ✓ Unlimited repositories
- ✓ Unlimited collaborators

Premium

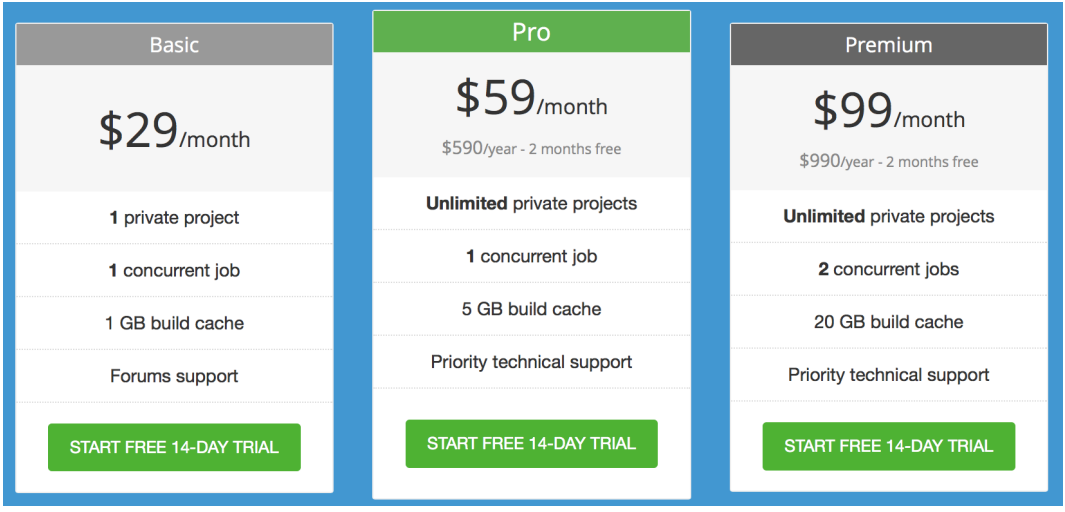
$489 PER MONTH

PERFECT FOR LARGER TEAMS

- **10** Concurrent jobs
- ✓ Unlimited build minutes
- ✓ Unlimited repositories
- ✓ Unlimited collaborators

ALL PRICES SHOWN IN USD

Appveyor pricing



Gitlab

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

Virtual Private Server

Создаем VPS с docker

1. Создаем учетную запись на sprintbox.ru с использованием промокода
2. Если у Вас Windows, то ставим PuTTY
3. Генерируем SSH ключ `https://habrahabr.ru/post/127521/` и добавляем его на sprintbox.ru
4. Изменяем тариф VPS на максимальный
5. Переустанавливаем из базового образа с аутентификацией по ключу
6. Заходим по ssh (или VNC) на созданный сервер
7. устанавливаем docker

   ```
   apt update && apt install docker.io
   ```

Запускаем GitLab

1. Запускаем GitLab run-gitlab.sh

```
docker run \
  --detach \
  --hostname <insert ip here> \
  --env GITLAB_OMNIBUS_CONFIG="external_url 'http://<insert ip here>/'; gitlab_rails['gitlab_shell_ssh_port'] = 2222;" \
  --publish 443:443 --publish 80:80 --publish 2222:22 --publish 5005:5005 \
  --name gitlab \
  --restart always \
  --volume /srv/gitlab/config:/etc/gitlab \
  --volume /srv/gitlab/logs:/var/log/gitlab \
  --volume /srv/gitlab/data:/var/opt/gitlab  \
  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

```
docker run \
 --detach \
 --name gitlab-runner \
 --restart always \
 --volume /var/run/docker.sock:/var/run/docker.sock \
 --volume /srv/gitlab-runner/config:/etc/gitlab-runner \
 --privileged \
 gitlab/gitlab-runner
```

2. Регистрируем gitlab-runner

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

Gitlab CI

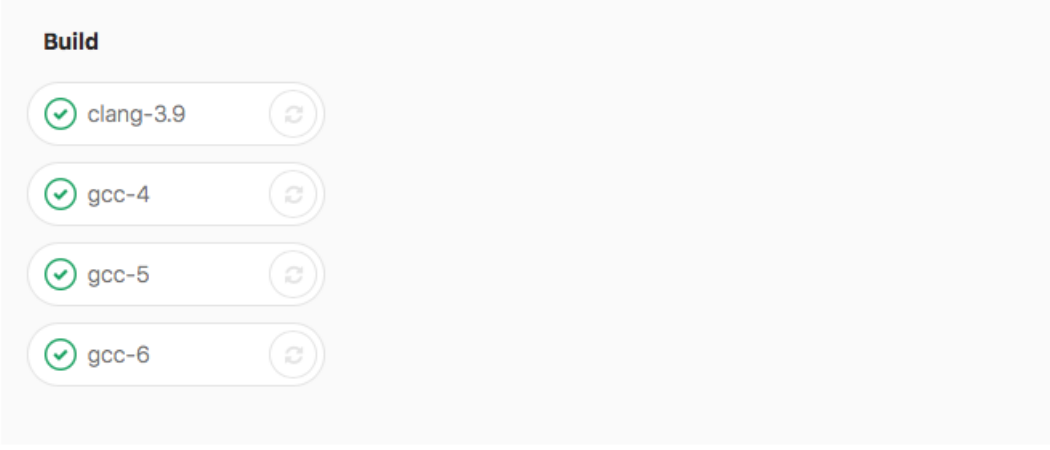
1. создаем в корне файл .gitlab-ci.yml

```
before_script:
  - apt update && apt install -y cmake python-pip python-dev
  - pip install conan
  - conan user

stages:
  - build

gcc-4:
  stage: build
  image: gcc:4
  script:
    - conan test_package --build=missing
  tags:
    - docker
```

Результат



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. Правим hostname в /src/conan/server.conf
3. Добавляем upload в .gitlab-ci.yml

Enable gitlab docker registry

1. Генерируем сертификаты с помощью letsencrypt

```
apt install letsencrypt
letsencrypt certonly -a standalone -d <domain name> --http-01-port 8080
cd /etc/letsencrypt/live/<address>
mkdir -p /srv/gitlab/config/ssl/
cat cert.pem fullchain.pem > /srv/gitlab/config/ssl/<address>.crt
cat privkey.pem > /srv/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"]
```

1. Пишем .gitlab-ci.yml

```
deploy:
  stage: deploy
  image: docker:git
  services:
    - docker:dind
  tags:
    - docker
  before_script:
    - docker info
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN <address>:5005
  script:
    - docker build -t <address>:5005/cpp-russia/wello .
    - docker push <address>:5005/cpp-russia/wello
  dependencies:
    - 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 собсвенный 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 собсвенный 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 собсвенный 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 собсвенный 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