Automate your life with Gitlab-CI

Szymon Tomasz Datko
Summer Student
at
Computer Security Team

9th of August, 2016
Presentation in a nutshell

1) Motivation  – why it may be useful?
2) Overview  – how it works in general?
3) Usage  – what do I need to do?
4) Summary  – where to look for more?
Presentation in a nutshell

1) Motivation  – why it may be useful?
2) Overview   – how it works in general?
3) Usage      – what do I need to do?
4) Summary    – where to look for more?
1) Motivation

Stages of programming:

- building → compilation
- testing → unit tests, code analyzers
- deploying → packaging
1) Motivation

Stages of programming:
- building
  - compilation
- testing
  - unit tests
  - code analyzers
- deploying
  - packaging
1) Motivation

Stages of programming:

- building
  - compilation
- testing
  - unit tests
  - code analyzers
- deploying
  - packaging

Continuous Integration is about automating this and performing regularly.
Presentation in a nutshell

1) Motivation – why it may be useful?
2) Overview – how it works in general?
3) Usage – what do I need to do?
4) Summary – where to look for more?
2) Overview – Code Analyzers

- **static** code analysis  (without executing a program)
- sometimes integrated into advanced code editors
- simply **scans** source code for unsafe calls
- complex **structure** analysis of syntax tree
- a lot of tools available:
  - **Rough Auditing Tool for Security**
  - ***Lint** (Pylint, JSLint, CppLint, ...)
  - **PMD** (Project Mess Detector?)
  - ...
2) Overview – Gitlab-CI

- **part** of Gitlab (since version 8.0)

- allows one to introduce **Continuous Integration**
  easy way – automate tests, build and deploy

- **jobs** configured per repository – simply in a file

- multiple **executors** available – run your jobs:
  – directly on a local or remote server
  – inside a container
  – inside a virtual machine
2) Overview - architecture
2) Overview - architecture

[Diagram showing the architecture of Gitlab runner and Docker service]

Runner 1

Gitlab runner  
Docker service

Runner 2

Gitlab runner  
Docker service

Runner N

gitlab.cern.ch
2) Overview - architecture

Runner 1

Runner 2

Runner N

somedirectory/
somefile.cpp
somefile.hpp
somefile.txt

Gitlab runner
Docker service

Gitlab runner
Docker service

Gitlab runner
Docker service

gitlab.cern.ch
2) Overview - architecture

- Gitlab runner
- Docker service

Runner 1

Runner 2

Runner N

somedirectory/
somefile.cpp
somefile.hpp
somefile.txt
.gitlab-ci.yml

gitlab.cern.ch

Szymon Datko

Automate your life with Gitlab-CI
2) Overview - architecture

$ git commit
$ git push
2) Overview - architecture

2) Overview - architecture

Runner 1

Runner 2

Runner N

$ git commit
$ git push

/* job request */
2) Overview - architecture

$ git commit
$ git push

/* job request */
2) Overview - architecture

$ git commit
$ git push

/* job request */

/* job result */
Presentation in a nutshell

1) Motivation – why it may be useful?
2) Overview – how it works in general?
3) Usage – what do I need to do?
4) Summary – where to look for more?
# Example CI workflow definition for Gitlab-CI

**image:** dockercern.ch/security-services/code-checking:latest

**job1:**
- **type:** test
- **script:**
  - rats ./* > result.txt
  - check.sh 'result.txt'
- **artifacts:** untracked: true

**job2:**
- **type:** test
- **script:**
  - pylint ./*
3) Usage - jobs results

- **badges**  - out of box, always there

  \[\text{http://}<\text{gitlab-server}>/<\text{user}>/<\text{project}>/\text{badges}/<\text{branch}>/\text{build}.svg\]
  
  Example: \[\text{http://gitlab.cern.ch/sdatko/Vulnerable-Codes/badges/master/build.svg}\]

  Get it through user interface: Gitlab > Project > Project settings > Badges

- **artifacts**  - some configuration required

  „... specify a list of files and directories which should be attached to the build after success.”

  1) Declare them for each job in .gitlab-ci.yml
  2) Download or browse directly in web interface after job execution

  In the future: nice summary dashboard with table/preview of tests result? (~JUnit XML parser)
3) Usage - demonstration

Projects

Your Projects   Starred Projects   Explore Projects

Filter by name...   Last updated   New Project

Szymon Tomasz Datko / Vulnerable-Codes
Few examples of vulnerable codes to check the Static Code Analysis with Gitlab-CI

Szymon Tomasz Datko / Security-Services-Code-Checking
Repository containing everything needed to build security-services/code-checking Docker image for Gitlab-CI jobs

Szymon Datko
Automate your life with Gitlab-CI
3) Usage - demonstration

Vulnerable-Codes
Few examples of vulnerable codes to check the Static Code Analysis with Gitlab-CI

Star 0  Fork 0  KRB5  https://@gitlab.cern.ch:8443/sdatko/Vulnerable

Files (170 KB)  Commits (4)  Branch (1)  Tags (0)  Add Changelog  Add License  Add Contribution guide

e2830911  Added verification of RATS output · 26 minutes ago by Szymon Tomasz Datko
3) Usage - demonstration

- **Added verification of RATS output**
  - Szymon Tomasz Datko authored 13 minutes ago
  - b1cde1d2

- **Added .gitlab-ci.yml file**
  - Szymon Tomasz Datko authored 39 minutes ago
  - 2836f00c

- **Added example vulnerable C codes**
  - Szymon Tomasz Datko authored an hour ago
  - 0b4ae4a3

- **add README**
  - Szymon Tomasz Datko authored about 3 hours ago
  - 7f39d10c
### 3) Usage - demonstration

**Image Description:**
- The image shows a GitLab user interface with a pipeline for a repository named `master·b1cde1d2`. The pipeline has a status of failed, indicated by an 'x' symbol.
- The pipeline added verification of RATS output.
- The duration of the pipeline is 15 seconds.

**Table:**
<table>
<thead>
<tr>
<th>ID</th>
<th>Commit</th>
<th>Tests</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>#23077</td>
<td>master·b1cde1d2</td>
<td>×</td>
<td>15 seconds</td>
</tr>
</tbody>
</table>
3) Usage - demonstration

Pipeline #23077 with 1 build for master in 15 seconds

Authored by Szymon Tomasz Datko 26 minutes ago

Commit b1cde1d27a608fd4e15db9c33bbd8ea2d04d3e31

Added verification of RATS output

<table>
<thead>
<tr>
<th>Status</th>
<th>Build ID</th>
<th>Name</th>
<th>Tags</th>
<th>Duration</th>
<th>Finished at</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>#46820</td>
<td>run_rats</td>
<td></td>
<td>15 seconds</td>
<td>13 minutes ago</td>
</tr>
</tbody>
</table>
3) Usage - demonstration

Build #46820 for commit b1cde1d2 from master by @sdatko 14 minutes ago

```bash
$ gitlab-ci-multi-runner 1.3.2 (0323456)
Using Docker executor with image docker.cern.ch/security-services/code-checking:latest
Pulling docker image docker.cern.ch/security-services/code-checking:latest ...
Running on runner-96f70e8a-project-10294-concurrent-0 via gitlabci03.cern.ch...:
Cloning repository...
Cloning into '/builds/sdatko/Vulnerable-Codes'...
Checking out b1cde1d2 as master...
$ echo 'New job started'
New job started
$ rats -l 'c' ./ */ >> rats.txt
$ test $(grep -c 'High' rats.txt) -eq 0
Running after script...
$ echo 'Job finished'
Job finished
Uploading artifacts...
untracked: found 1 files
Uploading artifacts to coordinator... ok
id=46820 responseStatus=201 Created

ERROR: Build failed: exit code 1
```
Presentation in a nutshell

1) Motivation – why it may be useful?
2) Overview – how it works in general?
3) Usage – what do I need to do?
4) Summary – where to look for more?
4) Summary

- Gitlab-CI can check your code automatically
- container image with necessary tools is provided
- only one very simple configuration file required...
- ...yet it is powerful!

For details:
gitlab.cern.ch/gitlabci-examples/Static_Code_Analysis
Automate your life with Gitlab-CI

Szymon Tomasz Datko
Summer Student at Computer Security Team
9th of August, 2016