

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

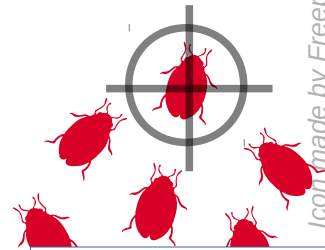
– what do I need to do?

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:
 - **R**ough **A**uditing **T**ool for **S**ecurity
 - ***Lint** (PyLint, JSLint, CppLint, ...)
 - **PMD** (Project Mess Detector?)
 - ...



Apple's goto fail

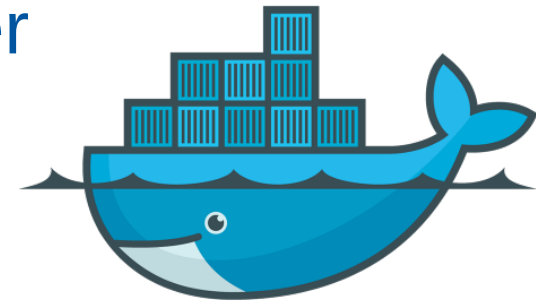
```
...
if ((err = SSLFreeBuffer(&ctx)) != 0)
    goto fail;
if ((err = Ready(&SSL, &ctx)) != 0)
    goto fail;
if ((err = SSL.update(&ctx, &crnd)) != 0)
    goto fail;
if ((err = SSL.update(&ctx, &srnd)) != 0)
    goto fail;
if ((err = SSL.update(&ctx, &prms)) != 0)
    goto fail;
    goto fail;
if ((err = SSL.final(&ctx, &out)) != 0)
    goto fail;
...
```

unreachable code

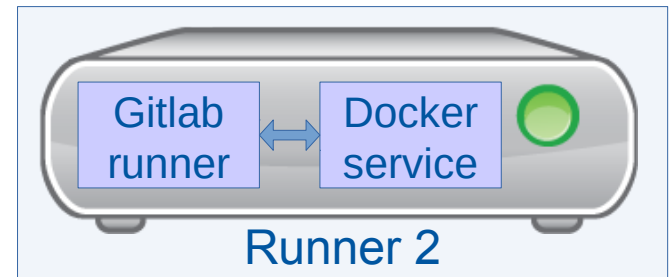
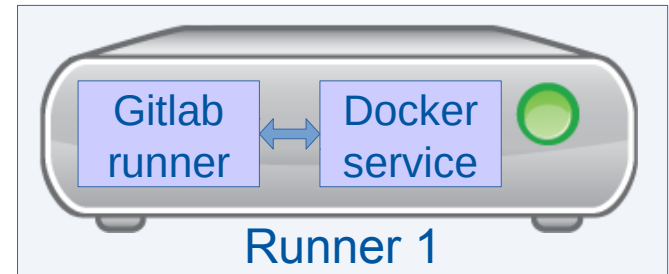
2) Overview – Gitlab-CI



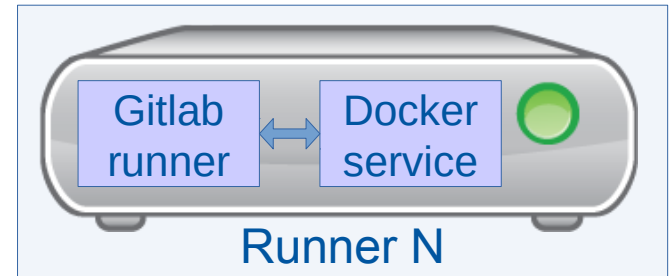
- **part** of Gitlab (since version 8.0)
- allows one to introduce **Continuous Integration** easy way – automate tests, build and deploy
- **jobs** configured per repository – simply in a file
- multiple **executors** available – run your jobs:
 - directly on a local or remote server
 - inside a container
 - inside a virtual machine



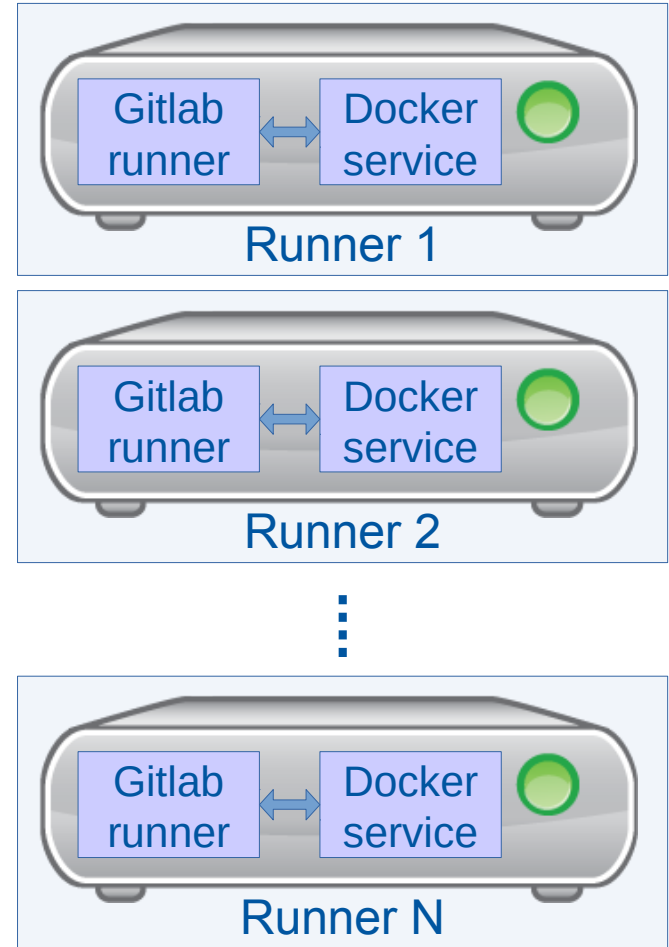
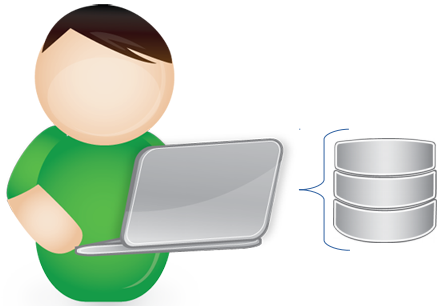
2) Overview - architecture



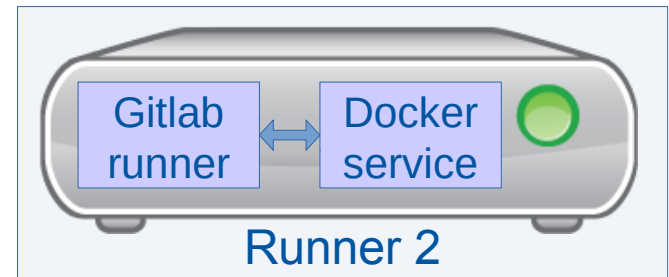
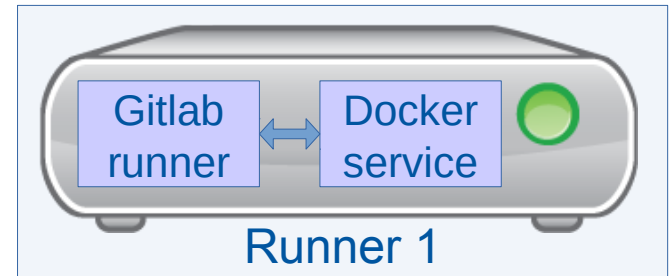
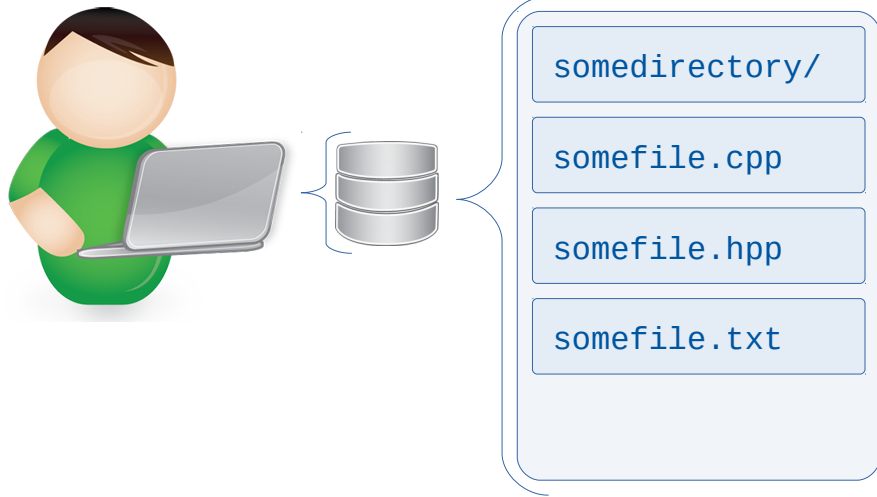
⋮



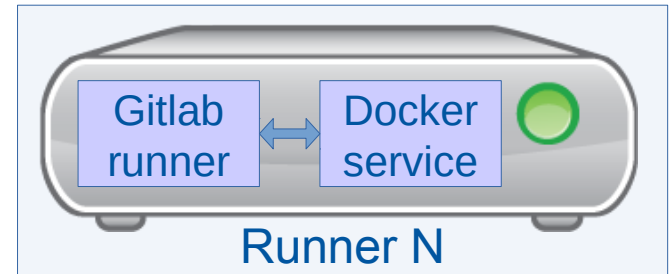
2) Overview - architecture



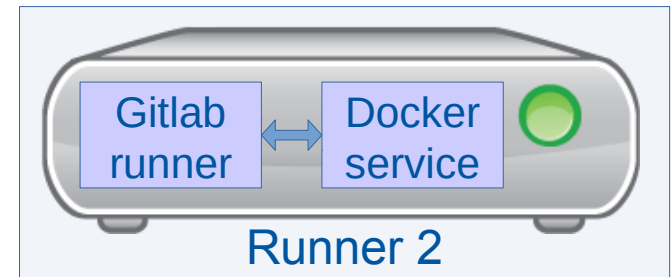
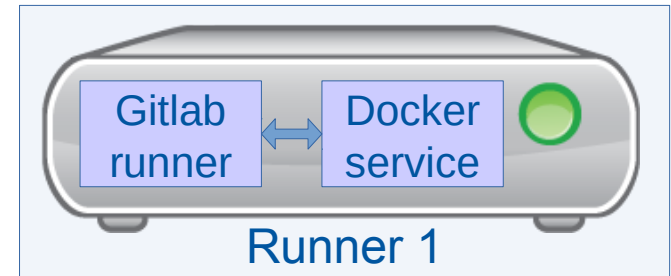
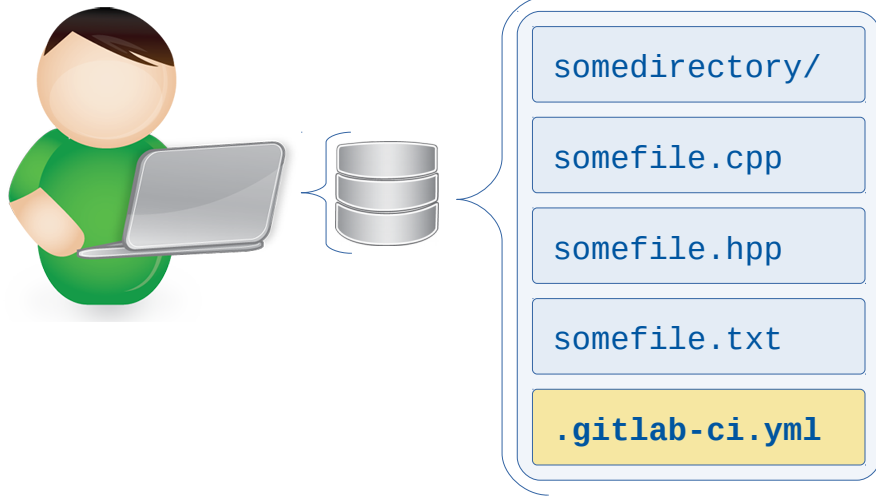
2) Overview - architecture



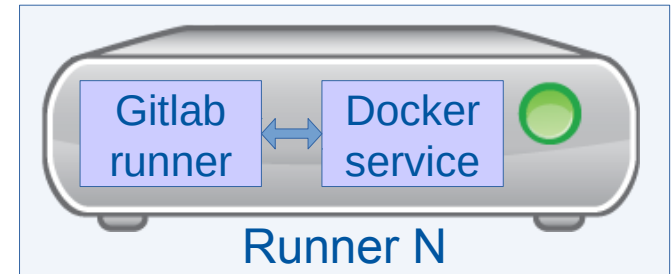
⋮



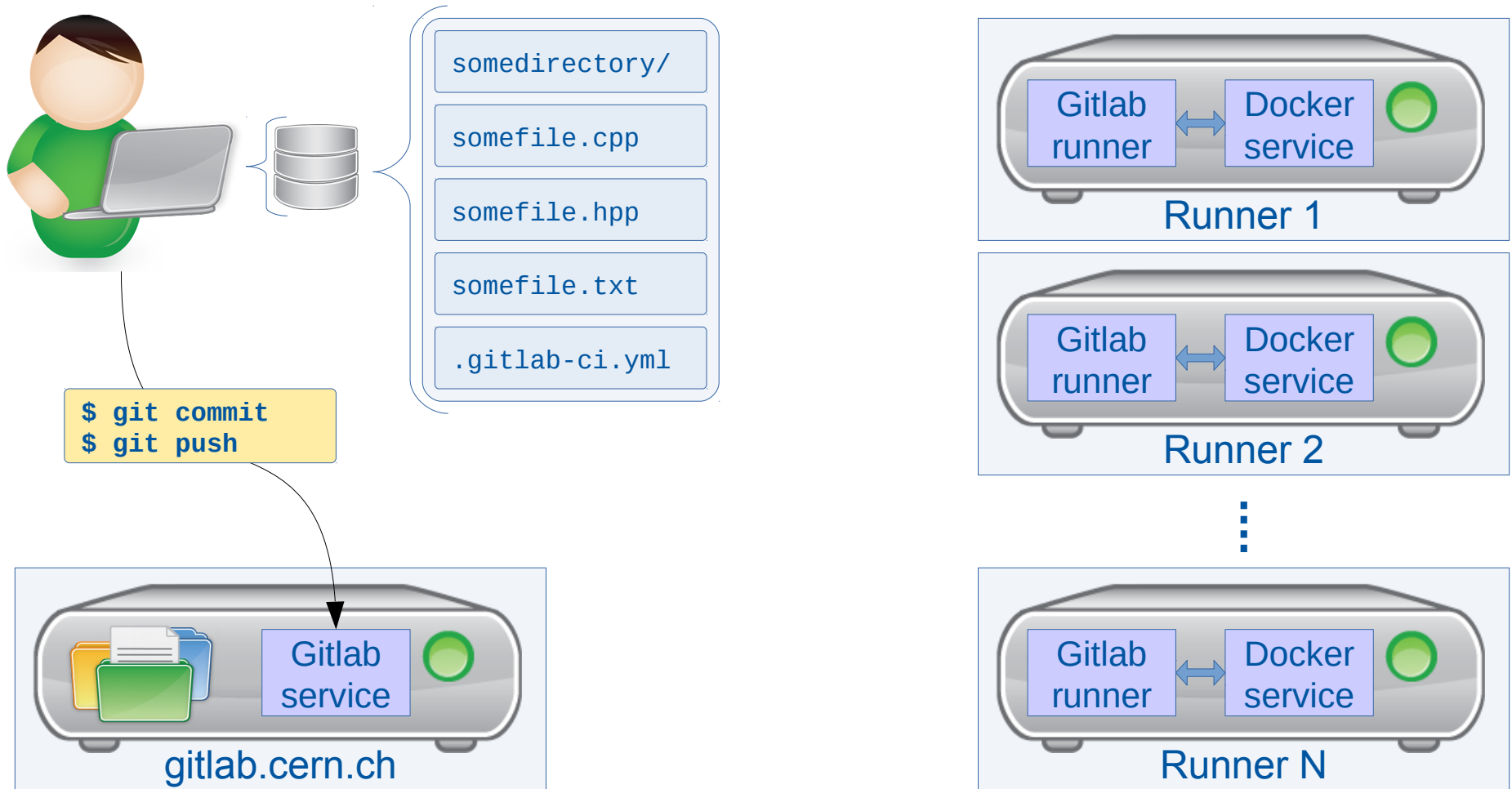
2) Overview - architecture



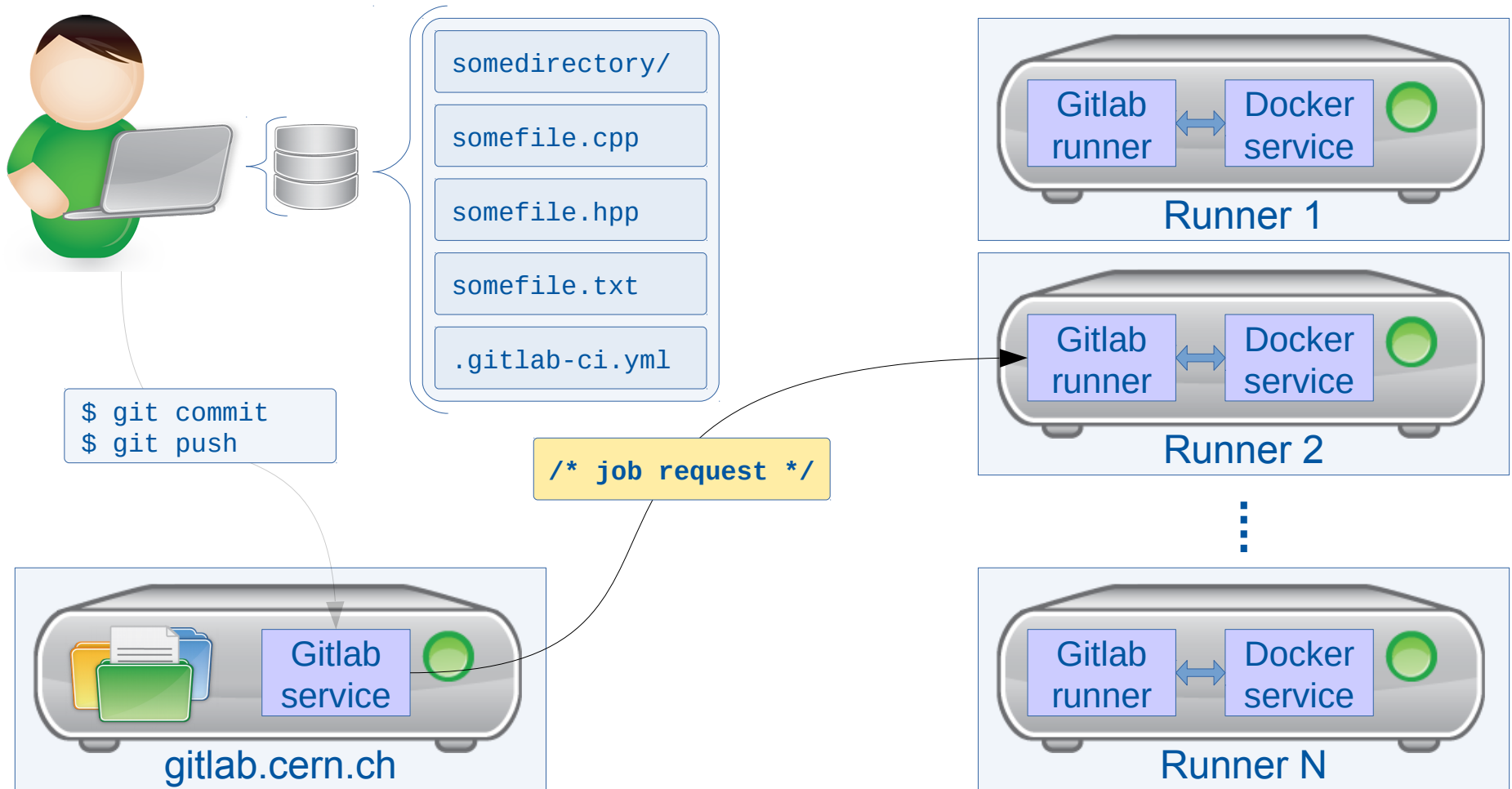
⋮



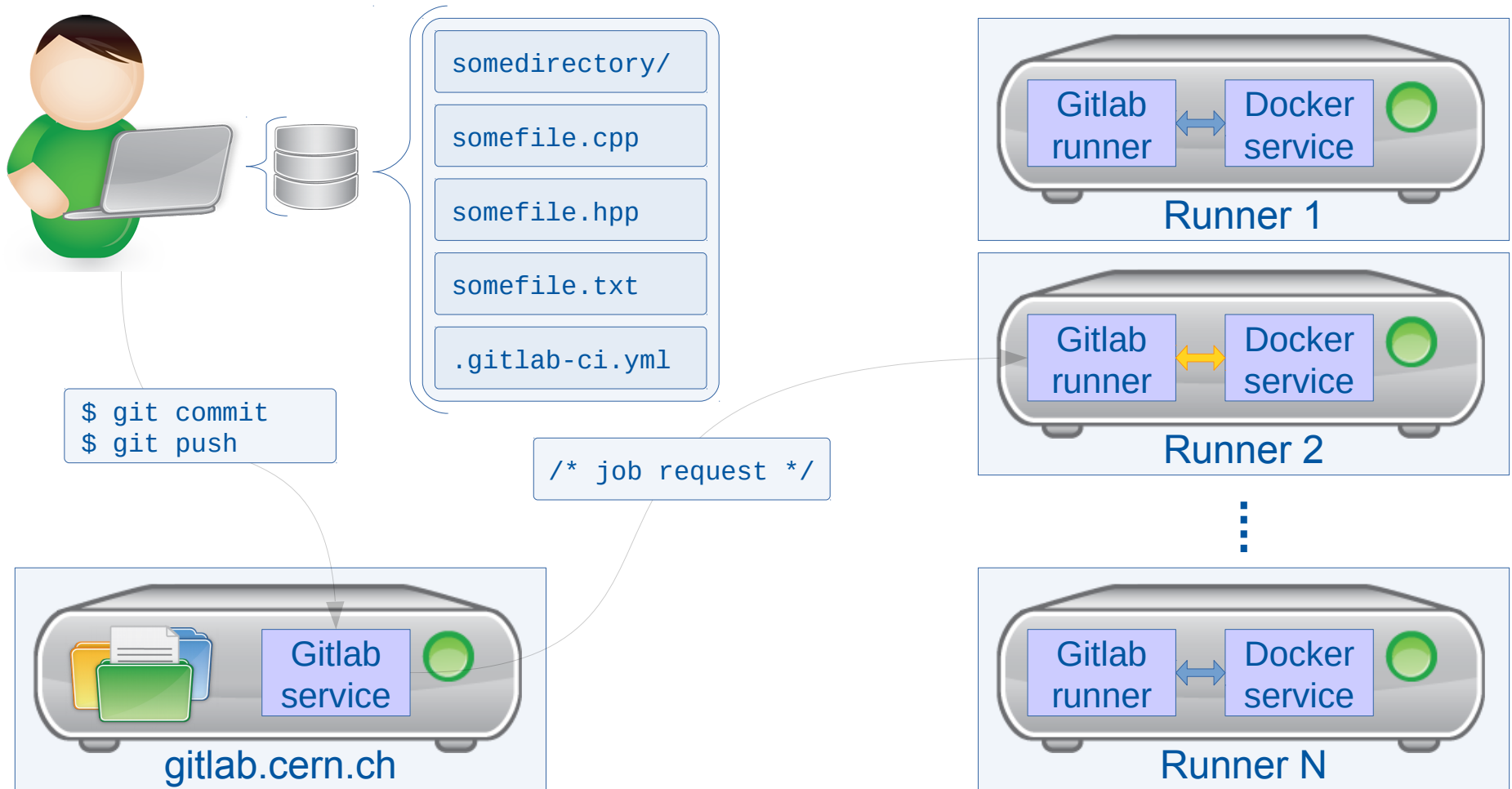
2) Overview - architecture



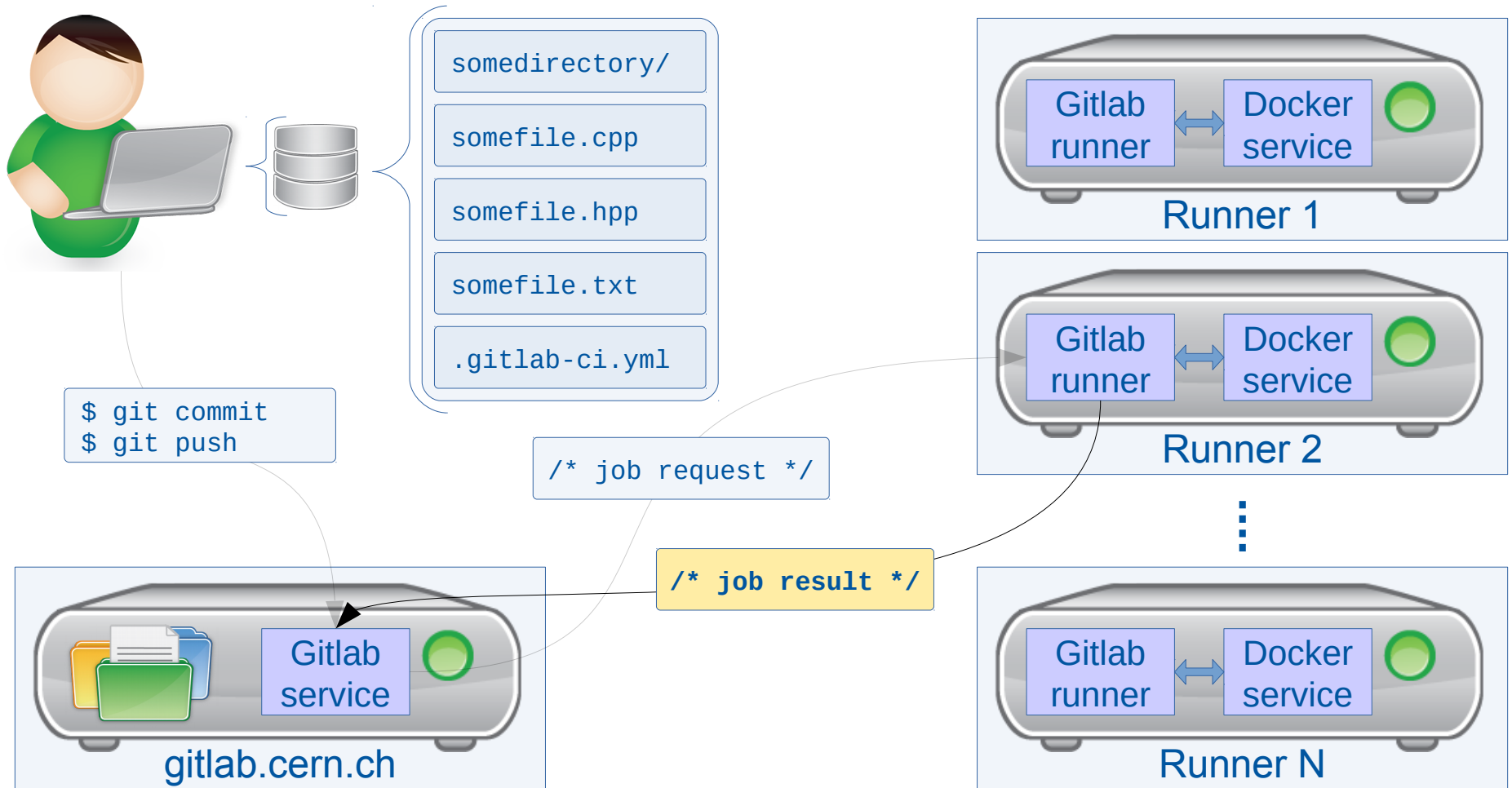
2) Overview - architecture



2) Overview - architecture



2) Overview - architecture



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?

3) Usage - jobs configuration



```
# Example CI workflow definition for Gitlab-CI
image: docker.cern.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 ./*
```

Comment

Docker image to be used

Job definition

Job type

Shell commands to execute

Definition of job artifacts

3) Usage - jobs results

- badges – out of box, always there

`http:// <gitlab-server> / <user> / <project> / badges / <branch> / build.svg`

Example: `http://gitlab.cern.ch/sdatko/Vulnerable-Codes/badges/master/build.svg`



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

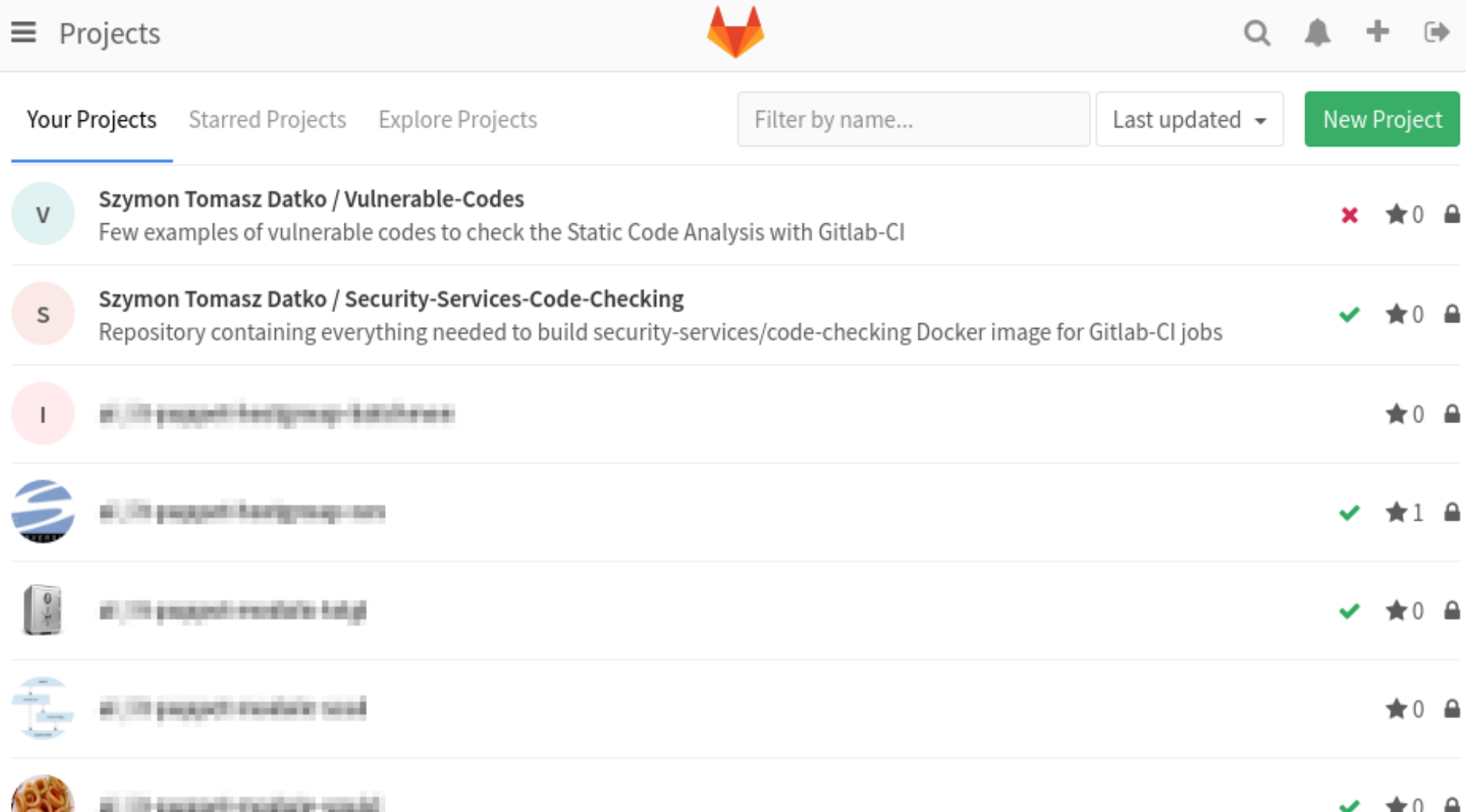
- artifacts – some configuration required

„... 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



The screenshot shows the GitLab interface for the 'Projects' section. At the top, there is a navigation bar with the GitLab logo, search, notification, and user icons. Below this, there are tabs for 'Your Projects', 'Starred Projects', and 'Explore Projects'. A search filter 'Filter by name...' and a dropdown menu 'Last updated' are visible. A green 'New Project' button is on the right. The main content area displays a list of projects:

- Szymon Tomasz Datko / Vulnerable-Codes**: Few examples of vulnerable codes to check the Static Code Analysis with Gitlab-CI. Status: ✖ ★ 0 🔒
- Szymon Tomasz Datko / Security-Services-Code-Checking**: Repository containing everything needed to build security-services/code-checking Docker image for Gitlab-CI jobs. Status: ✔ ★ 0 🔒
- [Blurred]**: Status: ★ 0 🔒
- [Blurred]**: Status: ✔ ★ 1 🔒
- [Blurred]**: Status: ✔ ★ 0 🔒
- [Blurred]**: Status: ★ 0 🔒
- [Blurred]**: Status: ✔ ★ 0 🔒



3) Usage - demonstration

The screenshot shows a GitLab repository page for a user named Szymon Tomasz Datko. The repository is named 'Vulnerable-Codes' and is described as 'Few examples of vulnerable codes to check the Static Code Analysis with Gitlab-CI'. The page includes navigation tabs for Project, Activity, Repository, Pipelines, Graphs, Issues, and Merge Requests. It also features a star and fork count (both 0), a URL, and a commit history section. A commit with ID e2830911 is shown as failed, with the message 'Added verification of RATS output · 26 minutes ago by Szymon Tomasz Datko'.

Szymon Tomasz Datko...

Project Activity Repository Pipelines Graphs Issues Merge Requests 0

V

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/Vulneral>

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

failed e2830911 Added verification of RATS output · 26 minutes ago by Szymon Tomasz Datko

3) Usage - demonstration









☰ Szymon Tomasz Datk...  🔍 🔔 + ➡

Project Activity **Repository** Pipelines Graphs Issues Merge Requests 0


Files **Commits** Network Compare Branches Tags

master ▼ Vulnerable-Codes

06 Aug, 2016 4 commits

	Added verification of RATS output Szymon Tomasz Datko authored 13 minutes ago	✖ 	b1cde1d2	Browse Files
	Added .gitlab-ci.yml file Szymon Tomasz Datko authored 39 minutes ago		2836f00c	Browse Files
	Added example vulnerable C codes Szymon Tomasz Datko authored about an hour ago		0b4ae4a3	Browse Files
	add README Szymon Tomasz Datko authored about 3 hours ago		7f39d10c	Browse Files

3) Usage - demonstration

☰ Szymon Tomasz Datk...  🔍 🔔 + ↗


Project Activity Repository **Pipelines** Graphs Issues Merge Requests 0

Pipelines Builds Environments

All 1 Running 0 Branches Tags New pipeline CI Lint

ID	Commit	Tests	Duration	
✖ #23077	master · b1cde1d2 latest Added verification of RATS output	✖	15 seconds	⬇ ↻

3) Usage - demonstration

☰ Szymon Tomasz Datk...  🔍 🔔 + ➔

Project Activity Repository **Pipelines** Graphs Issues Merge Requests 0

Pipeline #23077 with 1 build for `master` in 15 seconds ✖ failed

Authored by **Szymon Tomasz Datko** 26 minutes ago


Commit `b1cde1d27a608fd4e15db9c33bbd8ea2d04dde31` 📄

Added verification of RATS output


Retry failed

Status	Build ID	Name	Tags	Duration	Finished at	
✖ Tests						
✖ failed	#46820	run_rats		15 seconds	13 minutes ago	↓ ↺

3) Usage - demonstration

☰ Szymon Tomasz Datk...  🔍 🔔 + 🏠

Project Activity Repository **Pipelines** Graphs Issues Merge Requests 0

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

Build #46820 »

```
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
```

Build artifacts

Download	Browse
----------	--------

Build details Retry

Duration: 15 seconds
Finished: 13 minutes ago
Runner: #235

Raw	Erase
-----	-------

Commit message
Added verification of RATS output

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