GitLab

Unbox GitLab CI/CD

Agenda

- What is GitLab CI/CD
- Why GitLab CI/CD
- How to benefit from GitLab CI/CD
  - Quick start
  - Advanced workflows
    - Faster pipeline
    - Templating(include)
    - Dynamic child pipeline
    - Manual approval flow
    - K8s deployment
    - Security tests

What is GitLab CI/CD

GitLab CI/CD is a capability built into GitLab for software development through the continuous methodologies:

Continuous Integration (CI)

Automated testing and artifact creation

Continuous Delivery (CD)

Automated deployment to test and staging environments
Manual deployment to Production

Continuous Deployment (CD)

Automated deployment to Production

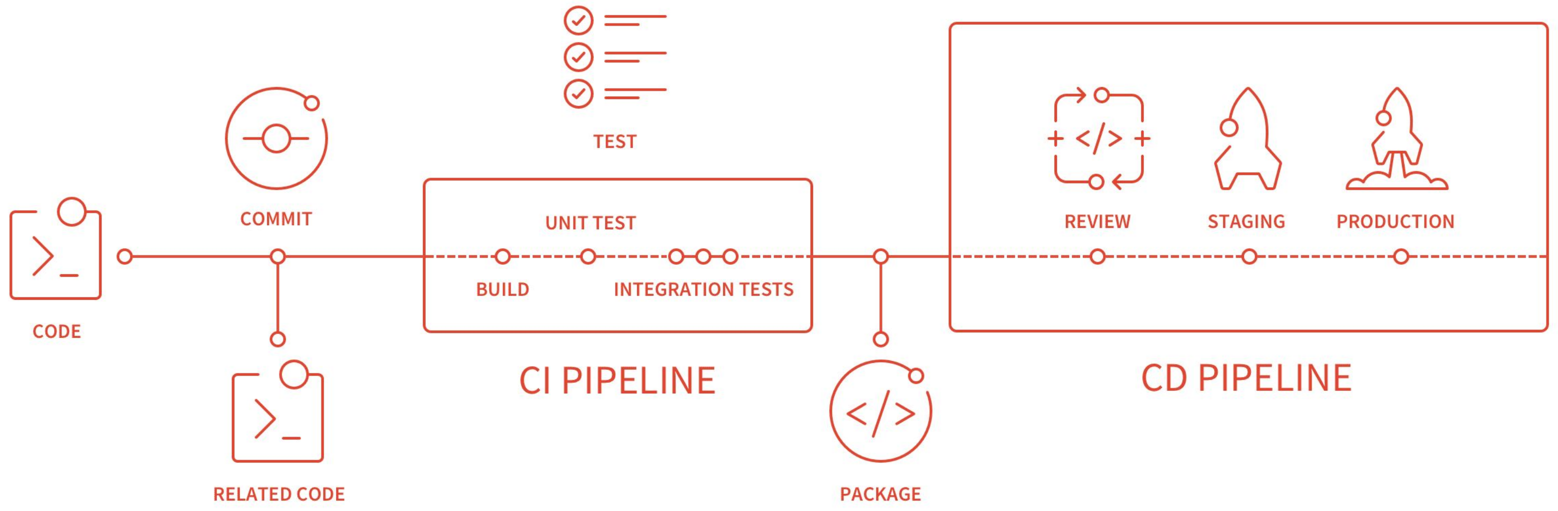
Why consider GitLab CI/CD

- **Versioned build & tests**: a .gitlab-ci.yml file contains your tests and build scripts, ensuring every branch gets build & tests it needs.
- **Build artifacts & test results**: binaries, other build artifacts and test results can be stored and explored in GitLab.
- **Native Docker support**: custom Docker images, spin up services as part of testing, build new Docker images, even run on Kubernetes.
- **Multi-language**: build scripts are command line driven and work with any language.
- **Real time logging**: a link in the merge request takes you to the current log.
- **One application**: no integrations to maintain, no extra license costs, no switching back and forth between applications

Get Started with GitLab CI/CD

Run your first GitLab CI/CD pipeline

Step 1: Define what to run
Step 2: Define where to run
Step 3: Give it a go!



Basic pipeline definition syntax

Yaml format for pipeline definition (.gitlab-ci.yml by default)

```yaml
image: "ruby:2.5"

before_script:
  - apt-get update -qq && apt-get install -y -qq sqlite3 libsqlite3-dev nodejs
  - ruby -v
  - which ruby
  - gem install bundler --no-document
  - bundle install --jobs $(nproc)  "${FLAGS[@]}"

rspec:
  script:
    - bundle exec rspec

rubocop:
  script:
    - bundle exec rubocop
```

GitLab runner/executors brief

GitLab CI Runner is where the task is executed.

Runner installations:

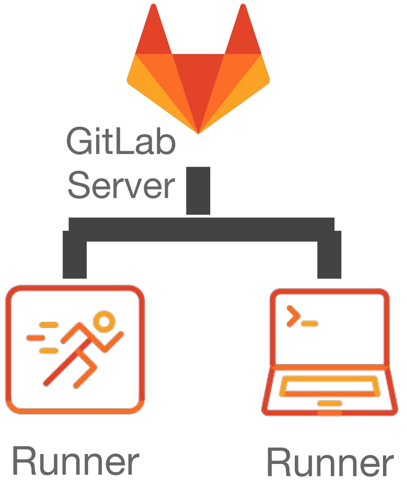
- Linux
- Windows
- MacOS
- Container/K8S

Common executor types:

- Shell (not ssh)
- Docker (most common)
- Kubernetes

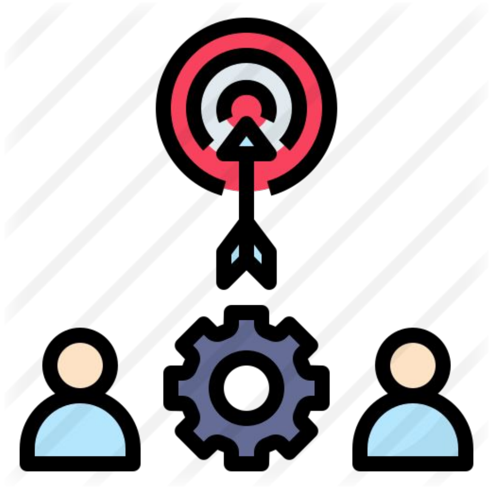
Runner types:

- Shared
- Group
- Project specific



Ways to trigger GitLab pipeline

- Push your code to GitLab repository*
- Run it manually from the UI
- Schedule it to run at later time
- “Trigger”ed by upstream pipeline
- Use API to launch a pipeline with “trigger”

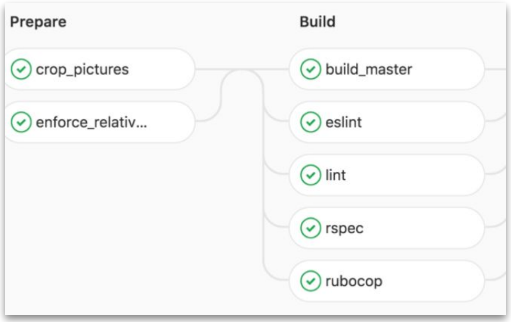


Advanced GitLab CI/CD Workflows

How to get my pipeline run faster?

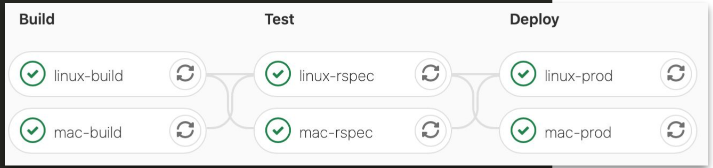
- Parallel

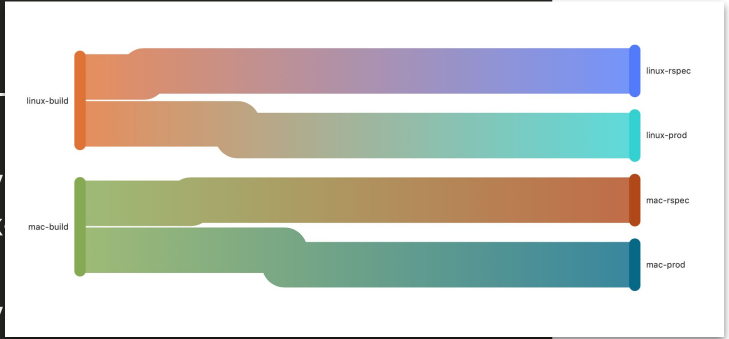
```
crop_pictures
  stage: Prepare
  script: crop_pics.sh

enforce_relative_links:
  stage: Prepare
  script: src/other/code/links.sh
```



- Directed Acyclic Graph

```
linux-build:
  stage: build
mac-build:
  stage: build
linux-rspec:
  stage: test
  needs: [“linux-build”]
mac-rspec:
  stage: test
  needs: [“mac-
linux-prod:
  stage: deploy
  needs: [“linux
mac-prod:
  stage: deploy
  needs: [“mac-build”]
```





More ways to make your faster

- Caching

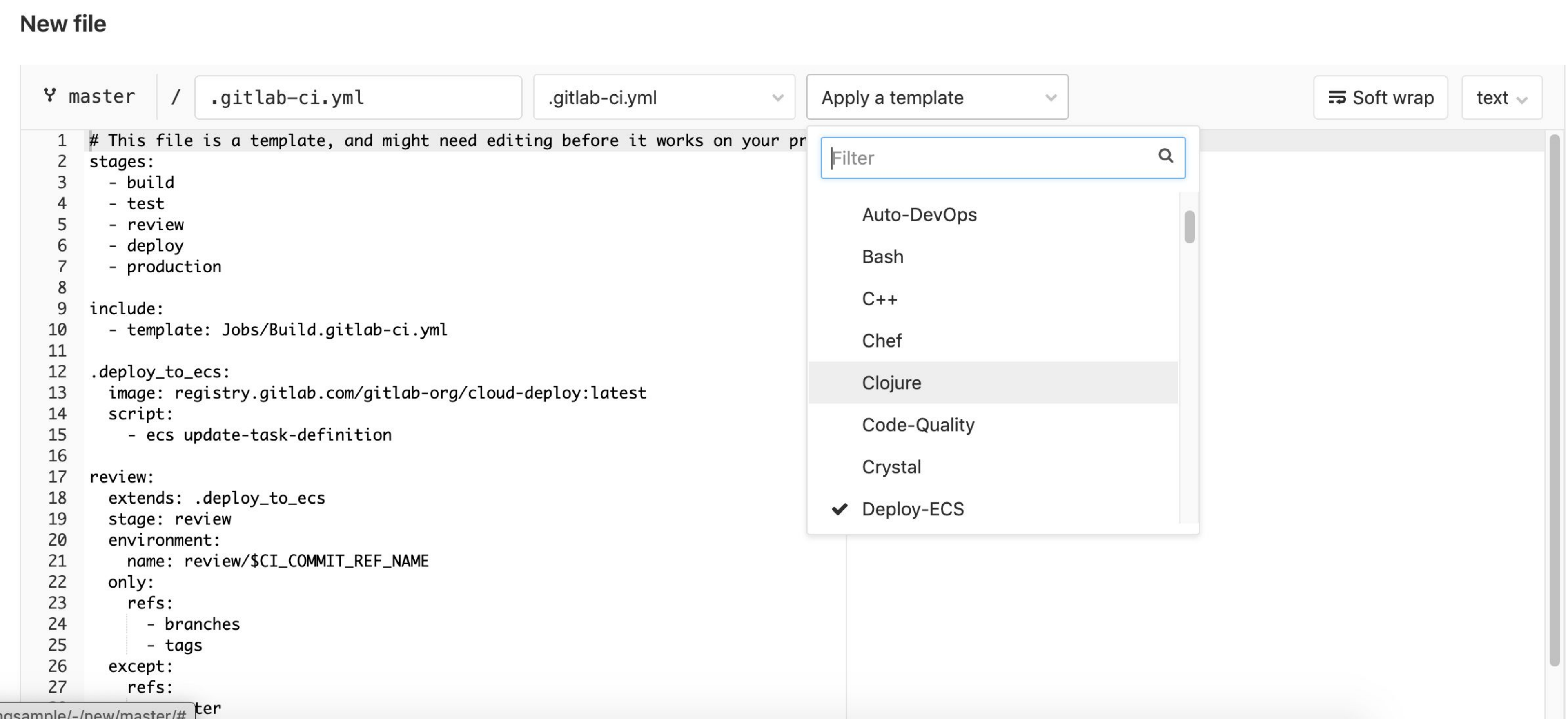
```
cache:
  paths:
     - binary/
     - .config
```

- Rules/condition

```
pseudo-deploy:
  stage: deploy
  only:
  - branches
  except:
  - master
```

```
job:
  script: "echo Hello, Rules!"
  rules:
    - if:
'$CI_MERGE_REQUEST_TARGET_BRANCH_NAME
== "master"'
      when: always
    - if: '$VAR =~ /pattern/'
      when: manual
    - when: on_success
```

Bored with writing every code block?



New file

master / .gitlab-ci.yml

.gitlab-ci.yml

Apply a template

Soft wrap

text

```
# This file is a template, and might need editing before it works on your pr
stages:
  - build
  - test
  - review
  - deploy
  - production

include:
  - template: Jobs/Build.gitlab-ci.yml

.deploy_to_ecs:
  image: registry.gitlab.com/gitlab-org/cloud-deploy:latest
  script:
    - ecs update-task-definition

review:
  extends: .deploy_to_ecs
  stage: review
  environment:
    name: review/$CI_COMMIT_REF_NAME
  only:
    refs:
      - branches
      - tags
  except:
    refs:
```

Use of ‘include’

- Reuse code from the same project with `include:local`

```
include: '/templates/.after-script-template.yml'
```

- Reuse code from the another project with `include:file`

```
include:
  - project: 'my-group/my-project'
    ref: master
    file: '/templates/.gitlab-ci-template.yml'
```

- Reuse code from arbitrary http(s) location with `include:remote`

```
include:
  - remote: 'https://gitlab.com/awesome-project/raw/master/.gitlab-ci-template.yml'
```

- Reuse code from template with `include:template`

```
include:
  - template: Auto-DevOps.gitlab-ci.yml
```

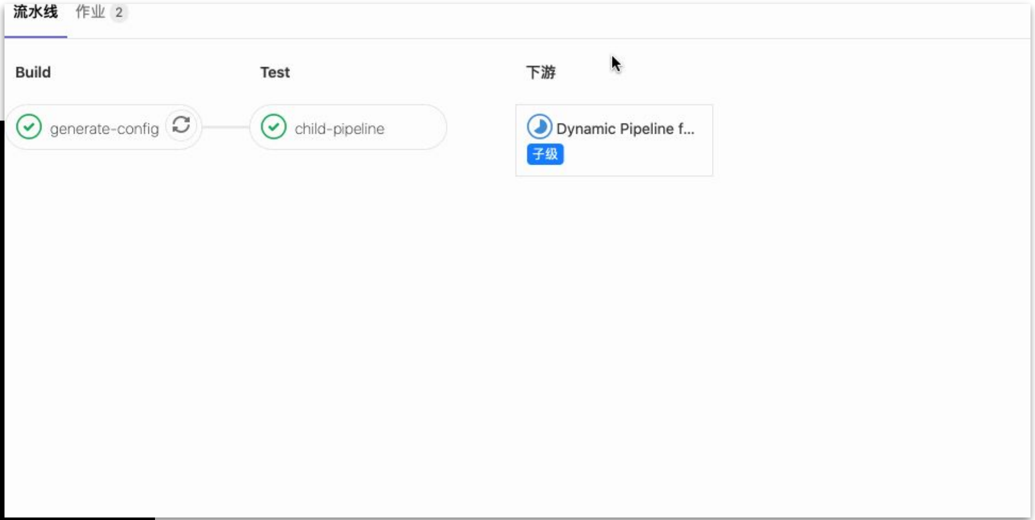
Need to “compute” a pipeline?

An example use case:

A project has 1000+ test cases with in-house test harness and want to run all of them in parallel during CI process. Authoring and maintaining the pipeline might be tedious.

Example code:

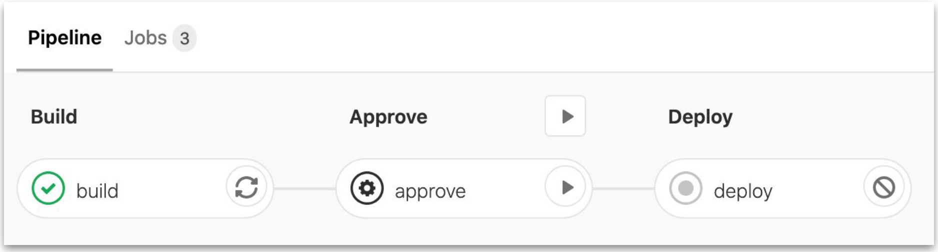
```
generate-config:
  stage: build
  script: generate-ci-config > generated-config.yml
  artifacts:
    paths:
      - generated-config.yml

child-pipeline:
  stage: test
  trigger:
    include:
      - artifact: generated-config.yml
        job: generate-config
```



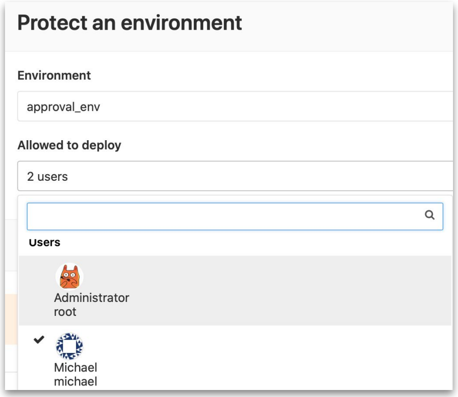
Want to get the pipeline wait for approval?

- Create a manual step with some `environment`.

```
approve:
  stage: Approve
  script:
    - echo Approved!
  environment:
    name: approval_env
  when: manual
  allow_failure: false
  only:
      - master
```



- Protect the `approval_env` environment in the protected environments settings by adding only needed user to "Allowed to Deploy" list.


Protect an environment

Environment: approval_env

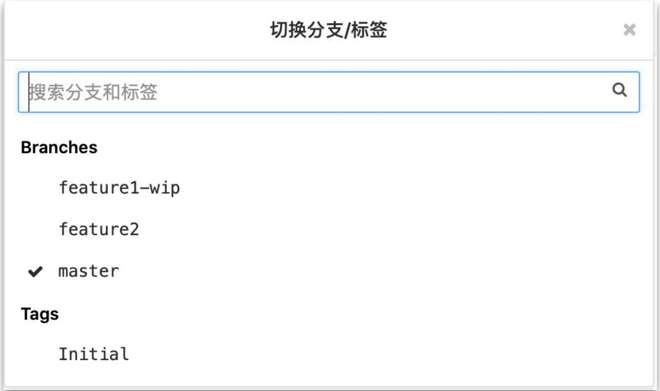
Allowed to deploy: 2 users

Users
Administrator
root
✓ Michael
michael

How can I run the whole pipeline conditionally?

- `Workflow:rules` controls to the entirety of a pipeline

```
workflow:
  rules:
    - if: $CI_COMMIT_REF_NAME =~ /-wip$/
      when: never
    - if: $CI_COMMIT_TAG
      when: never
    - when: always
```



切换分支/标签

搜索分支和标签

Branches

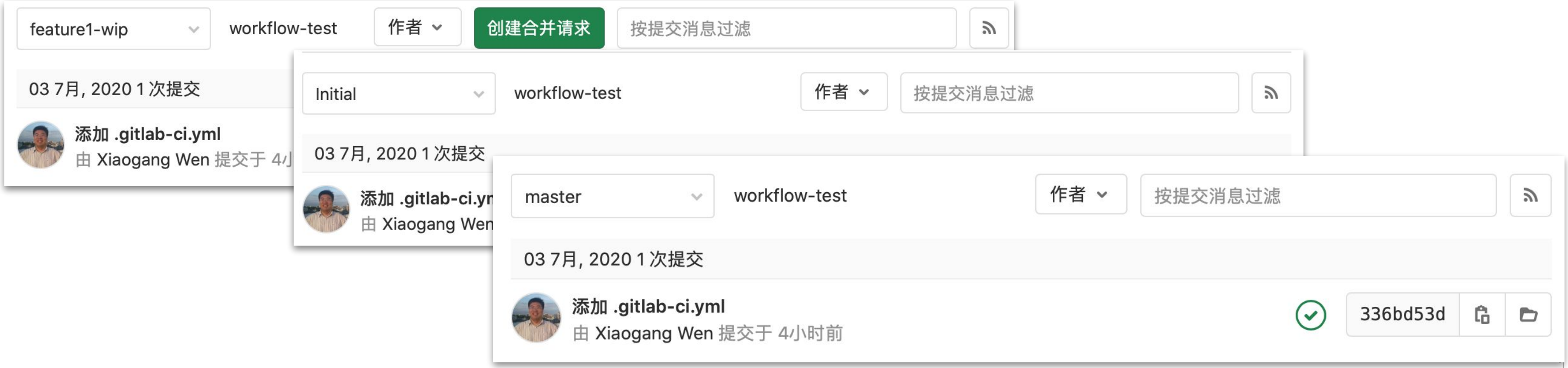
feature1-wip

feature2

✓ master

Tags

Initial



feature1-wip　workflow-test　作者　创建合并请求　按提交消息过滤

03 7月, 2020 1 次提交

添加 .gitlab-ci.yml
由 Xiaogang Wen 提交于 4小

Initial　workflow-test　作者　按提交消息过滤

03 7月, 2020 1 次提交

添加 .gitlab-ci.yn
由 Xiaogang Wen

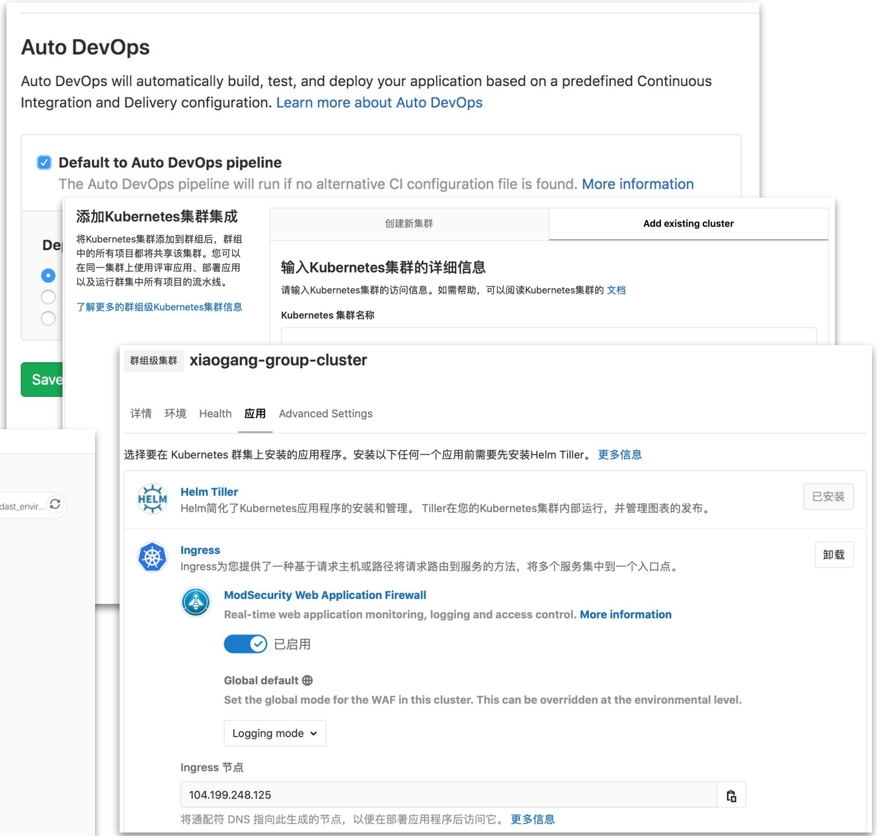
master　workflow-test　作者　按提交消息过滤

03 7月, 2020 1 次提交

添加 .gitlab-ci.yml
由 Xiaogang Wen 提交于 4小时前　✓　336bd53d

Minimal code to build and deploy to k8s?

- Prepare your Dockerfile in the repo
- Enabled Auto DevOps
- Connect to your K8S cluster
- Install Helm, Ingress and Prometheus (optional)
- Deploy your application!



Auto DevOps

Auto DevOps will automatically build, test, and deploy your application based on a predefined Continuous Integration and Delivery configuration. Learn more about Auto DevOps

[✓] Default to Auto DevOps pipeline
The Auto DevOps pipeline will run if no alternative CI configuration file is found. More information

添加Kubernetes集群集成

将Kubernetes集群添加到群组后，群组中的所有项目都将共享该集群。您可以在同一集群上使用评审应用、部署应用以及运行群集中所有项目的流水线。

了解更多的群组级Kubernetes集群信息

输入Kubernetes集群的详细信息

请输入Kubernetes集群的访问信息。如需帮助，可以阅读Kubernetes集群的 文档

Kubernetes 集群名称

群组级集群 xiaogang-group-cluster

选择要在 Kubernetes 群集上安装的应用程序。安装以下任何一个应用前需要先安装Helm Tiller。更多信息

Helm Tiller
Helm简化了Kubernetes应用程序的安装和管理。 Tiller在您的Kubernetes集群内部运行，并管理图表的发布。 已安装

Ingress
Ingress为您提供了一种基于请求主机或路径将请求路由到服务的方法，将多个服务集中到一个入口点。 卸载

ModSecurity Web Application Firewall
Real-time web application monitoring, logging and access control. More information

已启用

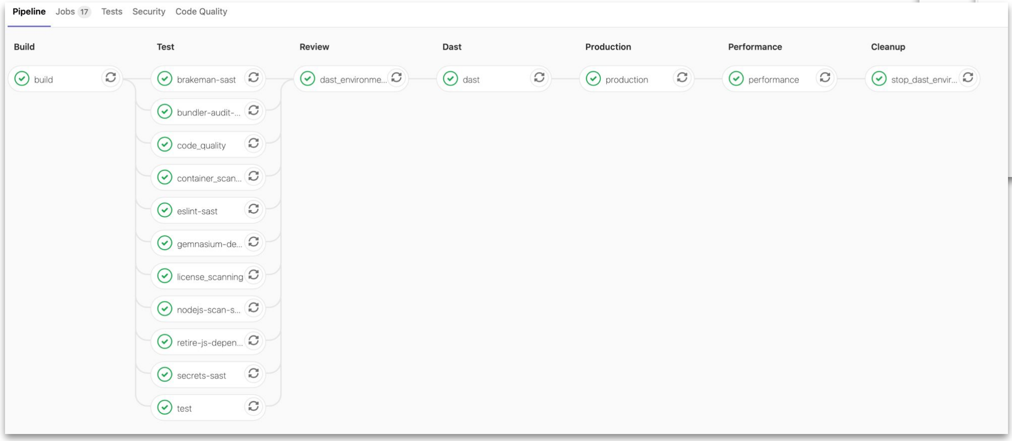
Global default
Set the global mode for the WAF in this cluster. This can be overridden at the environmental level.

Logging mode

Ingress 节点

104.199.248.125

将通配符 DNS 指向此生成的节点，以便在部署应用程序后访问它。 更多信息



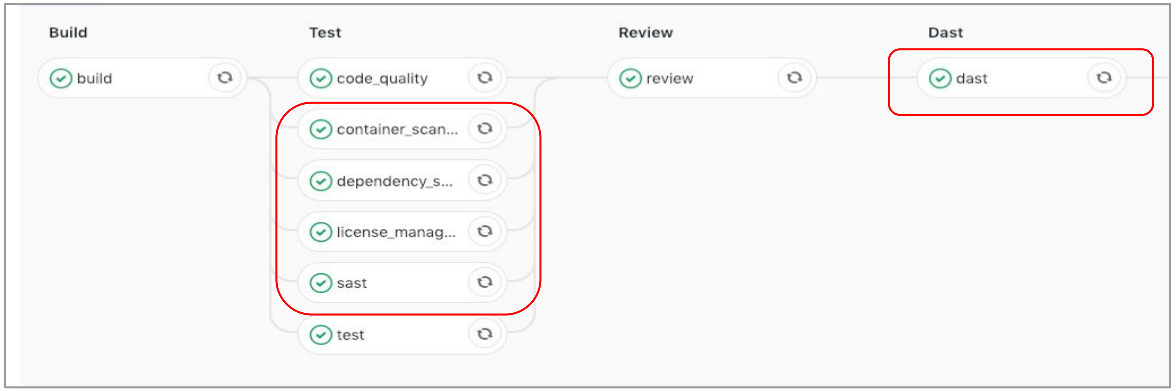
Build	Test	Review	Dast	Production	Performance	Cleanup
build	brakeman-sast	dast_environme...	dast	production	performance	stop_dast_envir...
	bundler-audit-...					
	code_quality					
	container_scan...					
	eslint-sast					
	gemnasium-de...					
	license_scanning					
	nodejs-scan-s...					
	retire-js-depen...					
	secrets-sast					
	test					

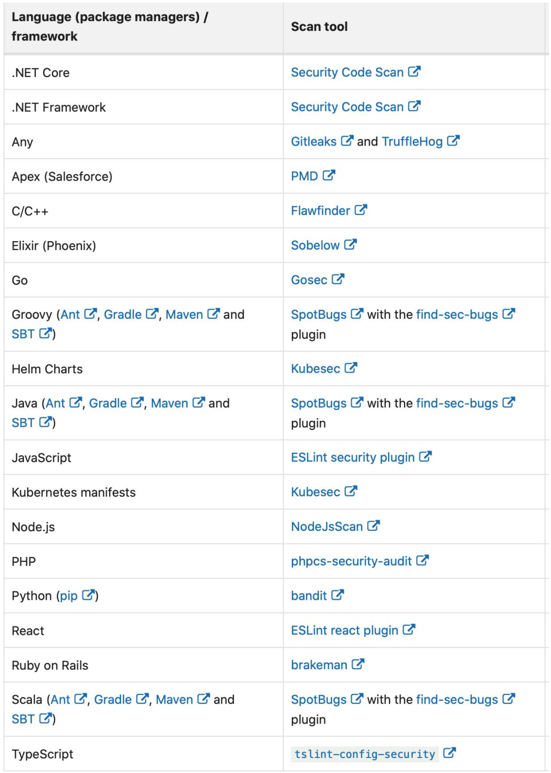
Wish to run security tests more frequently?

- Include your security test template in your `.gitlab-ci.yml` file

```
include:
  - template: SAST.gitlab-ci.yml
```

- Run your pipeline

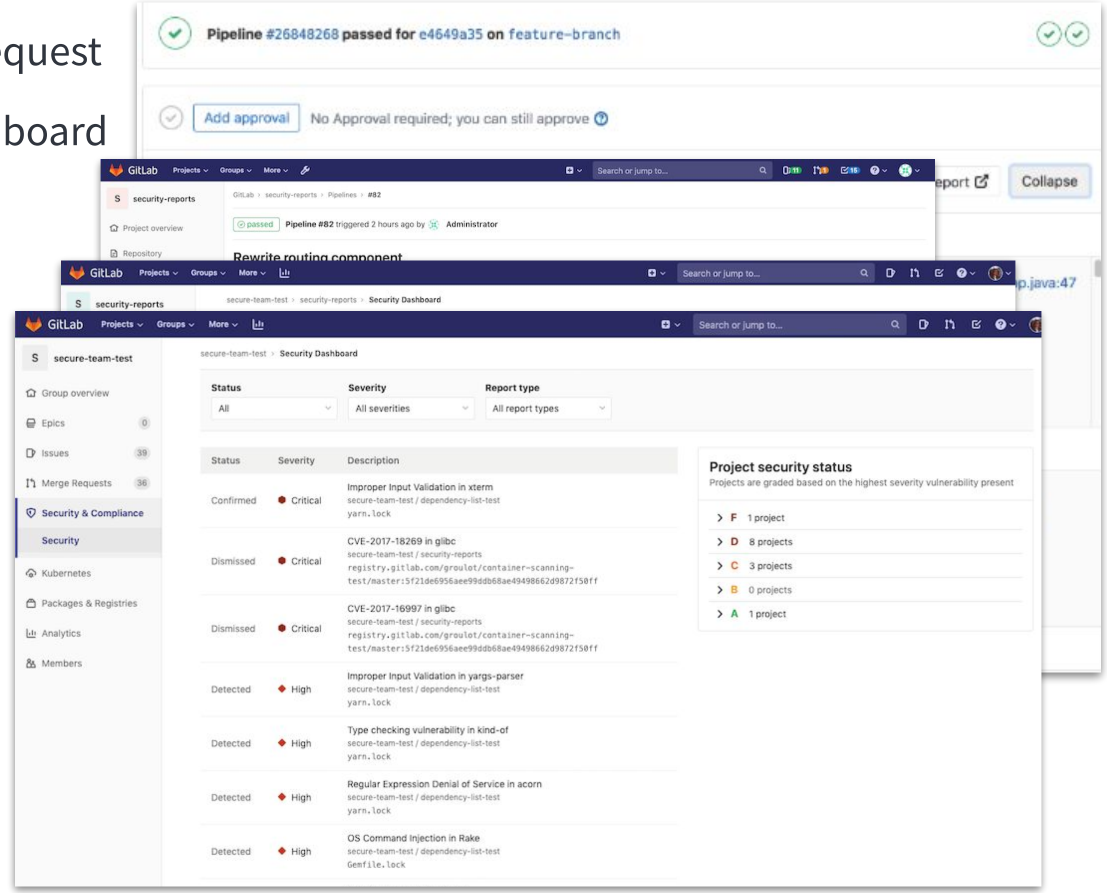




Language (package managers) / framework	Scan tool
.NET Core	Security Code Scan
.NET Framework	Security Code Scan
Any	Gitleaks and TruffleHog
Apex (Salesforce)	PMD
C/C++	Flawfinder
Elixir (Phoenix)	Sobelow
Go	Gosec
Groovy (Ant, Gradle, Maven and SBT)	SpotBugs with the find-sec-bugs plugin
Helm Charts	Kubesec
Java (Ant, Gradle, Maven and SBT)	SpotBugs with the find-sec-bugs plugin
JavaScript	ESLint security plugin
Kubernetes manifests	Kubesec
Node.js	NodeJsScan
PHP	phpcs-security-audit
Python (pip)	bandit
React	ESLint react plugin
Ruby on Rails	brakeman
Scala (Ant, Gradle, Maven and SBT)	SpotBugs with the find-sec-bugs plugin
TypeScript	`tslint-config-security`

Review your security test result

- Check the result in your merge request
- View the report via Security Dashboard
    - Pipeline level
    - Project level
    - Group level



Pipeline #26848268 passed for e4649a35 on feature-branch

Add approval No Approval required; you can still approve

GitLab › security-reports › Pipelines › #82

passed Pipeline #82 triggered 2 hours ago by Administrator

Rewrite routing component

secure-team-test › security-reports › Security Dashboard

secure-team-test › Security Dashboard

Status: All
Severity: All severities
Report type: All report types

Status	Severity	Description
Confirmed	Critical	Improper Input Validation in xterm secure-team-test / dependency-list-test `yarn.lock`
Dismissed	Critical	CVE-2017-18269 in glibc secure-team-test / security-reports `registry.gitlab.com/groulot/container-scanning-test/master:5f21de6956aee99ddb68ae49498662d9872f50ff`
Dismissed	Critical	CVE-2017-16997 in glibc secure-team-test / security-reports `registry.gitlab.com/groulot/container-scanning-test/master:5f21de6956aee99ddb68ae49498662d9872f50ff`
Detected	High	Improper Input Validation in yargs-parser secure-team-test / dependency-list-test `yarn.lock`
Detected	High	Type checking vulnerability in kind-of secure-team-test / dependency-list-test `yarn.lock`
Detected	High	Regular Expression Denial of Service in acorn secure-team-test / dependency-list-test `yarn.lock`
Detected	High	OS Command Injection in Rake secure-team-test / dependency-list-test `Gemfile.lock`

Project security status

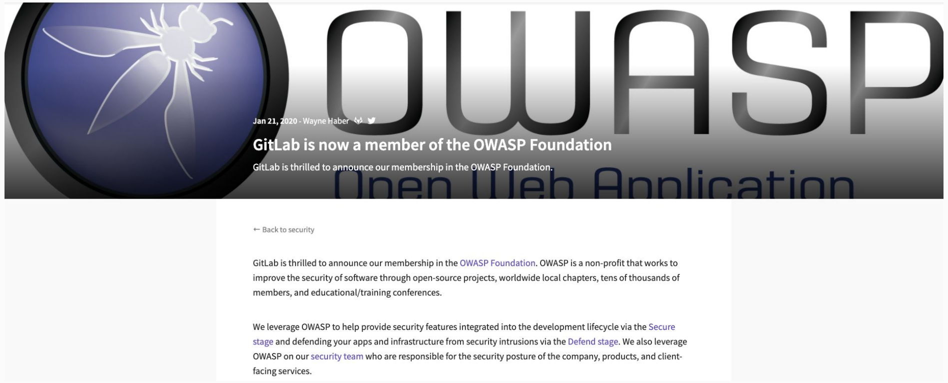
Projects are graded based on the highest severity vulnerability present

F	1 project
D	8 projects
C	3 projects
B	0 projects
A	1 project

The Top 10 OWASP vulnerabilities

The Top 10 OWASP vulnerabilities in 2020 are:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfigurations
7. Cross Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with known vulnerabilities
10. Insufficient logging and monitoring



Jan 21, 2020 · Wayne Haber

GitLab is now a member of the OWASP Foundation

GitLab is thrilled to announce our membership in the OWASP Foundation.

← Back to security

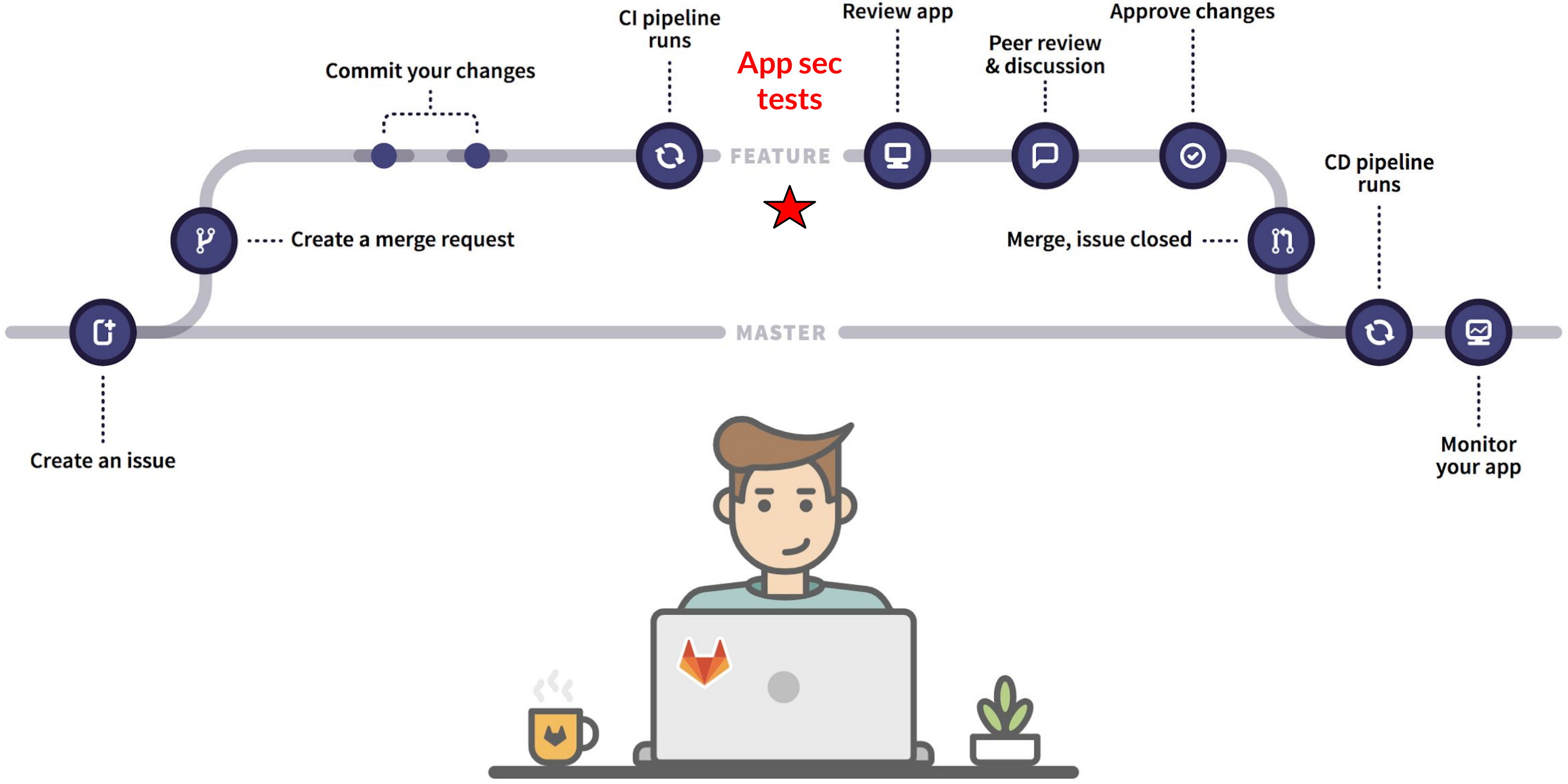
GitLab is thrilled to announce our membership in the OWASP Foundation. OWASP is a non-profit that works to improve the security of software through open-source projects, worldwide local chapters, tens of thousands of members, and educational/training conferences.

We leverage OWASP to help provide security features integrated into the development lifecycle via the Secure stage and defending your apps and infrastructure from security intrusions via the Defend stage. We also leverage OWASP on our security team who are responsible for the security posture of the company, products, and client-facing services.

https://about.gitlab.com/blog/2020/01/21/gitlab-is-now-a-member-of-the-owasp-foundation/

https://owasp.org/www-project-top-ten/
https://owasp.org/www-community/Source_Code_Analysis_Tools
https://about.gitlab.com/solutions/pci-compliance/

Seamlessly test for vulnerabilities within the developer workflow



GitLab

Q & A