

# 极狐 GitLab DevOps 解决方案

DevOps（开发运维一体化）是指通过编写代码来实现自动化运维，将「运维左移」到开发阶段，提高部署效率和频率，实现持续部署。

有了自动化部署的技术准备，还需要「正确的代码」和「顺畅的工作流」，才能有信心和流程进行持续部署，所以广义的 DevOps 还包括：

- • 测试左移：开发人员对代码的正确性负责，编写代码实现自动化测试，和业务代码一起提交，触发「持续集成」流水线，在代码合并之前进行自动强制检查，比传统的「Dev + Test」反馈更及时，降低了缺陷率。
- • 代码质量：在代码中引入书写规范和检查工具，提交代码时在本地和「持续集成」中自动强制检查，把不良代码拦截在合并之前。
- • Git workflow（工作流）：对「分支命名与合并方向」、「代码关联需求」、「tag 版本发布」等 Git 流程进行规范化，让开发流程更加顺畅。

在黑客攻击愈演愈烈的形势下，DevSecOps 应运而生，提出了「安全左移」理念——把安全扫描工具左移到开发阶段的「持续集成」流水线中，在代码合并之前进行自动检查，比传统的「DevOps + Sec」反馈更及时，实现了安全内建。

极狐 GitLab 作为开源的一体化软件研发平台，提供多种版本：

- • 免费版：提供基本的 DevOps 功能，社区互助支持；
- • 专业版：提供完整的 DevOps 功能，官方售后支持；
- • 旗舰版：提供完整的 DevSecOps 功能，官方售后支持；

每个版本均支持两种部署方式：

- • 私有化
- • SaaS

极狐 GitLab 专业版 可提供 DevOps 解决方案，能够帮助研发团队实现这些目标：提高部署效率、提高代码质量、规范 Git 流程、降低 bug 率。

## 提高部署效率

通过编写程序实现自动化部署，可提高部署效率，降低运维成本。在容器化、K8s 等技术的加持下，部署进一步标准化，集群管理更加便捷。

GitLab 支持两种自动化部署方式：

1. CI/CD 流水线
2. GitOps

「CI/CD 流水线」由业务代码驱动部署，当业务代码变化时，流水线生成制品，并且将制品或制品库链接（如 Docker）传递给 CD 流水线，执行自定义的命令程序（如 kubectl patch、scp），可部署到 K8s、对象存储、Linux/Windows/macOS 服务器、Serverless 等各种架构。缺点是：无法修改服务器配置等基础设施。

GitOps 由基础设施代码驱动部署，遵循「基础设施即代码」的理念，将基础设施配置文件（如 Terraform 和 K8s yaml）放在 Git 中，变化时触发流水线推送（如执行

kubectl apply) 或等待 GitLab agent (客户端) 持续拉取, 实现部署。

## GitOps

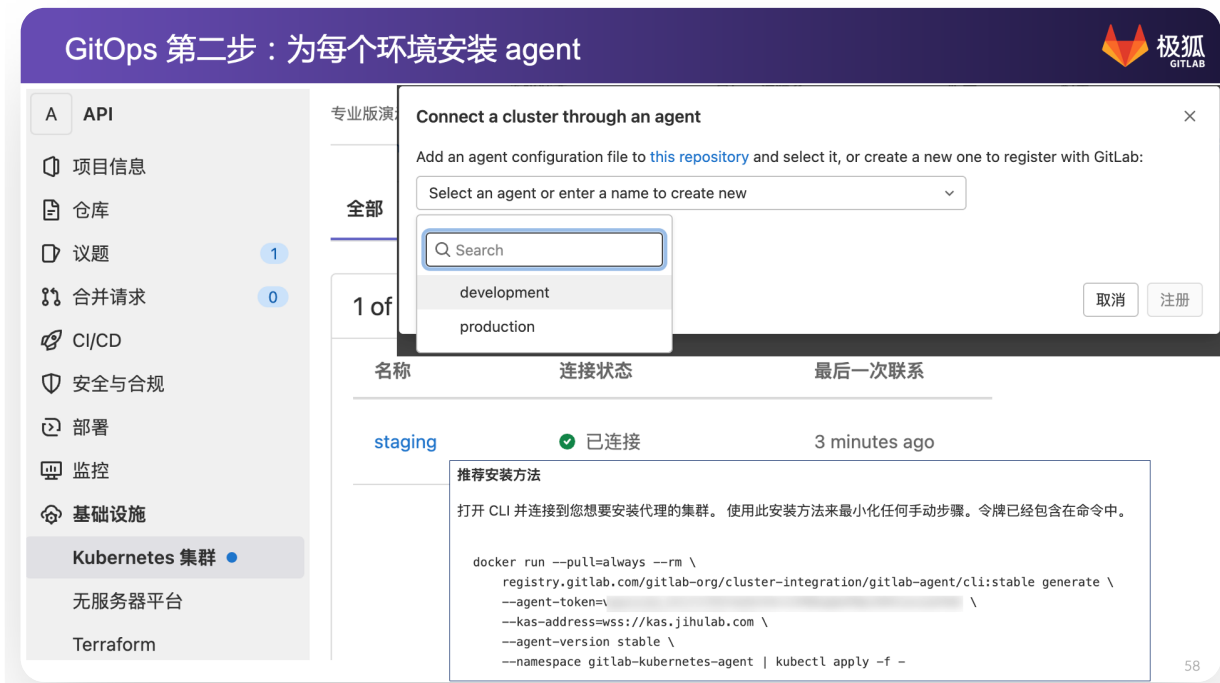
GitOps 特点:

- 基础设施即代码 (IaC): Git 作为单一可信源, 保存完整的基础架构信息, 可实现快速部署新环境 (灾难恢复、销售私有化产品)。
- 合并请求 (MR): 借助 Git 的合并请求评审机制, 让部署可评审、记录可追溯。
- 更安全: 由于 Git 中保存了完整的基础设施配置, 可通过 Agent 拉取生效, 而不像 CD 流水线推送需要暴露服务器网络权限。

GitLab 专业版提供 GitOps K8s 工作流:

1. 声明环境和 manifest 仓库路径
2. 为每个环境安装 agent
3. 提交 K8s manifest 到 Git

参考链接: <https://docs.gitlab.com/ee/user/clusters/agent/gitops.html>



## 提高代码质量

代码质量包括: 书写规范、注释、重复率、复杂度、设计模式、英文术语设计等多方面。各个语言/框架普遍具有业界知名的代码规范和开源扫描工具, 可以扫描出部分问题, 其余的问题交给人工代码评审 (Code Review)。

代码评审是提高代码质量、团队培养新人的重要方式, 比如:

- 初级工程师的代码需要高级工程师的评审, 并给出更好的代码建议, 可直接接受并使用;

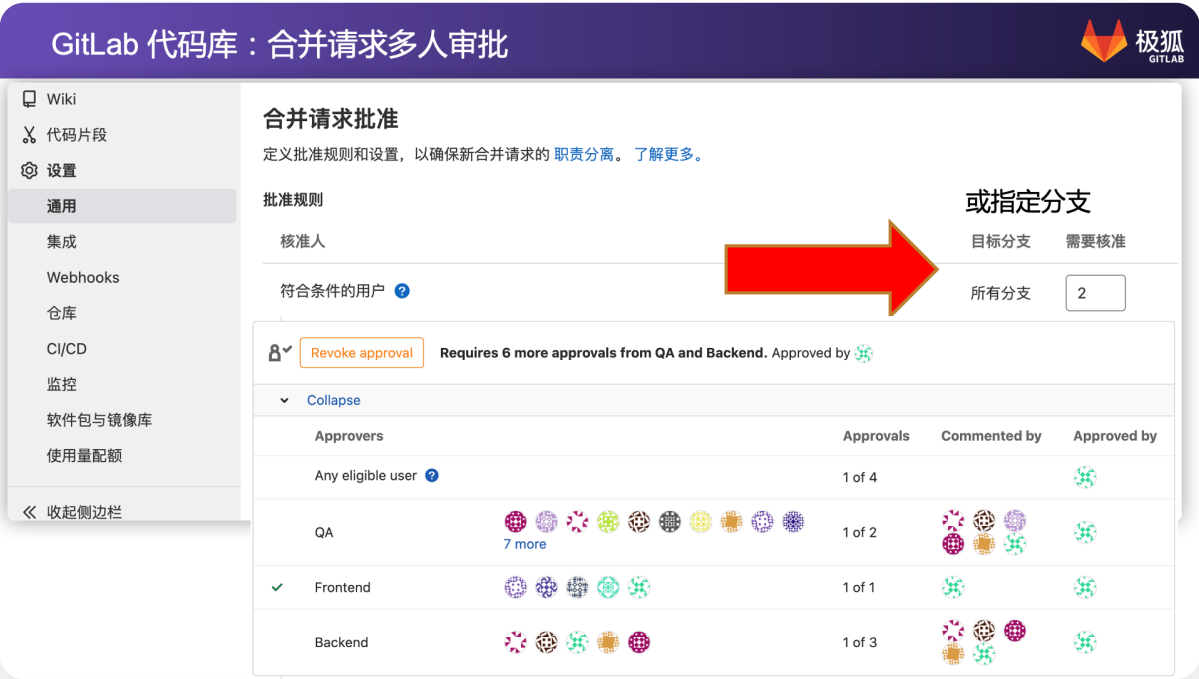
- 核心功能需要多人甚至多个小组（比如 安全小组）的评审；

GitLab 免费版支持 Code Review 和在持续集成流水线中运行代码扫描工具，而 GitLab 专业版的功能更加强大。

## 多人代码评审

GitLab 专业版的「合并请求批准」功能可为每个仓库的每个分支、每个目录配置不同的评审规则，可以指定多人或多个小组评审。

参考链接：[https://docs.gitlab.cn/jh/user/project/merge\\_requests/approvals/rules.html](https://docs.gitlab.cn/jh/user/project/merge_requests/approvals/rules.html)



## 代码质量报告

GitLab 专业版的「代码质量报告」功能可在流水线页面的「代码质量」标签页中显示完整报告。

参考链接：[https://docs.gitlab.com/ee/user/project/merge\\_requests/code\\_quality.html#code-quality-reports](https://docs.gitlab.com/ee/user/project/merge_requests/code_quality.html#code-quality-reports)

Pipeline Needs Jobs 142 Failed Jobs 1 Tests 160956 Security Licenses 23 **Code Quality**

! Found 10488 code quality issues  
This report contains all Code Quality issues in the source branch.

- ✖ Critical - Consider simplifying this complex logical expression.  
in [scripts/frontend/stylelint/stylelint-utils.js:15](#)
- ✖ Critical - Consider simplifying this complex logical expression.  
in [app/assets/javascripts/projects/settings/access\\_dropdown.js:248](#)
- ✖ Critical - CSRF vulnerability in OmniAuth's request phase  
in [Gemfile.lock:810](#)
- ✖ Major - Function `buildConfig` has 7 return statements (exceeds 4 allowed).  
in [workhorse/main.go:67](#)
- ✖ Major - Function `run` has 8 return statements (exceeds 4 allowed).  
in [workhorse/main.go:152](#)
- ✖ Major - Method `Resizer.Inject` has 5 return statements (exceeds 4 allowed).  
in [workhorse/internal/imageresizer/image\\_resizer.go:164](#)

## 在合并请求中显示代码质量问题

按照业界最佳实践，应在合并之前拦截代码质量问题，禁止合并，而事后扫描往往于事无补。

GitLab 可在合并请求页面显示代码质量问题，开发人员可获得及时反馈，自助修复，提高研发效率和代码质量。

参考链接：[https://docs.gitlab.com/ee/user/project/merge\\_requests/code\\_quality.html#code-quality-widget](https://docs.gitlab.com/ee/user/project/merge_requests/code_quality.html#code-quality-widget)

! Code quality degraded on 7746 points ?

Collapse

- ✖ Critical - Consider simplifying this complex logical expression.  
in [scripts/frontend/stylelint/stylelint-utils.js:15](#)
- ✖ Critical - Consider simplifying this complex logical expression.  
in [app/assets/javascripts/projects/settings/access\\_dropdown.js:248](#)
- ✖ Critical - CSRF vulnerability in OmniAuth's request phase  
in [Gemfile.lock:810](#)
- ▼ Major - Function `buildConfig` has 7 return statements (exceeds 4 allowed).  
in [workhorse/main.go:67](#)
- ▼ Major - Function `run` has 8 return statements (exceeds 4 allowed).  
in [workhorse/main.go:152](#)
- ▼ Major - Method `Resizer.Inject` has 5 return statements (exceeds 4 allowed).  
in [workhorse/internal/imageresizer/image\\_resizer.go:164](#)
- ▼ Major - Method `Resizer.tryResizeImage` has 7 return statements (exceeds 4 allowed).  
in [workhorse/internal/imageresizer/image\\_resizer.go:268](#)
- ▼ Major - Function `unpackFileFromZip` has 7 return statements (exceeds 4 allowed).  
in [workhorse/internal/artifacts/entry.go:64](#)

## 禁止评审自己的代码

开发人员在发起代码评审后，可能自行评审合并，而未经同事评审，导致有问题的代码直接进入主干。

GitLab 专业版的「合并请求批准」功能可以开启「禁止自行评审」。

参考链接：[https://docs.gitlab.cn/jh/user/project/merge\\_requests/approvals/settings.html](https://docs.gitlab.cn/jh/user/project/merge_requests/approvals/settings.html)

## 修改代码必须重新评审

代码评审通过后，开发人员可能再次提交，如果直接合并，会产生风险。

GitLab 专业版的「合并请求批准」功能可以开启「修改代码必须重新评审」。

参考链接：[https://docs.gitlab.cn/jh/user/project/merge\\_requests/approvals/settings.html](https://docs.gitlab.cn/jh/user/project/merge_requests/approvals/settings.html)



## 规范 Git 流程

多人开发时需要统一的 Git 规范，比如代码关联需求、分支命名规范等，这就是 Git workflow（工作流）。业界常用的有 Git Flow、Simplified Git Flow、GitLab Flow 等。

如果口头传达规范，监督成本高，难以落地。

极狐 GitLab 专业版可配置 Git workflow，自动强制执行。

### git commit 规范（如：代码关联需求）

开发人员可能乱写 git commit message，比如「做了一个需求」，没有意义，难以维护。

按照业界规范，应使用统一前缀（比如：feat、fix、docs 等），并且关联需求/bug 的编号（来自 GitLab issue 或 Jira 等项目管理工具），比如 feat: #123 login。

GitLab 专业版的「推送规则」功能可以配置此规范，如果开发人员推送不规范的信息会被自动拒绝，并提示如何修改。

参考配置:

```
(feat|fix|docs|style|refactor|perf|test|build|ci|chore|revert): #\d+ .+
```

参考链接: [https://docs.gitlab.cn/jh/user/project/repository/push\\_rules.html](https://docs.gitlab.cn/jh/user/project/repository/push_rules.html)

GitLab (专业版+) : 配置 git commit message 规范
极狐  
GITLAB

设置
通用
集成
Webhooks
仓库

### 推送规则

Require expression in commit messages

```
(feat|fix|docs|style|refactor|perf|test|build|ci|chore|revert): #\d+ .+
```

```
(feat|fix|docs|style|refactor|perf|test|build|ci|chore|revert): JIRA\-\d+ .+
```

```
$ git commit -m "hello world"
$ git push

Writing objects: 100% (25/25), 2.01 KiB | 2.01 MiB/s, done.
remote: GitLab: Commit message does not follow the pattern
' (feat|fix|docs|style|refactor|perf|test|build|ci|chore|revert): #\d+ .+'
! [remote rejected] 3-lint -> 3-lint (pre-receive hook declined)
error: failed to push some refs to 'jihulab.com:foo/bar/api.git'
```

## 防止提交垃圾文件

多人开发时, 难免有人提交 apk、jar、node\_modules 等文件, 导致 Git 仓库变慢, 影响大家的工作效率, 甚至产生安全漏洞 (maven/npm 官方网络仓库的包频繁更新修复漏洞, 而下载的包往往无人升级)。

GitLab 专业版的「推送规则」功能可以禁止提交某些文件名, 以及禁止提交大文件 (比如超过 1MB 一般不是代码)。

参考配置:

```
(\.apk|\.jar|(^node_modules\/.+))$
```

参考链接: [https://docs.gitlab.com/ee/user/project/repository/push\\_rules.html#prohibited-file-names](https://docs.gitlab.com/ee/user/project/repository/push_rules.html#prohibited-file-names)



## 分支命名规范

多人开发时需要拉取临时分支，经过审批，合并到公共分支。

开发人员可能使用了错误的分支命名，比如用人名「dev-zhangsan」。

按照业界规范，应该「每个任务一个分支」，使用 `issue-123` 或者区分类型：

- • feature/1-login
- • bugfix/2-sms
- • hotfix/1.2.1
- • release/1.2.0
- • support/1.x

由于前端/后端/客户端的工作流不同，所以分支规范应在每个代码仓库上单独配置，不过整个公司的各个后端小组可以相同。

GitLab 专业版的「推送规则」功能可以为每个代码库配置不同的分支命名规范，可由各个代码库的管理员（开发组长）或部门统一配置，如果开发人员推送不规范命名的分支会被自动拒绝，并提示如何修改。

参考配置：

```
(main|develop|((feature|bugfix|hotfix)\/.+))
```

参考链接：[https://docs.gitlab.cn/jh/user/project/repository/push\\_rules.html](https://docs.gitlab.cn/jh/user/project/repository/push_rules.html)





## 降低 bug 率

基于 Google 等知名软件公司的「人工测试转型自动化测试」的实践，降低 bug 率的核心方法是：单元测试。GitLab 免费版即支持在持续集成流水线中运行单元测试，并且把测试报告显示在「合并请求」网页中。而 GitLab 专业版的功能更加强大。

## 覆盖率下降评审


各个项目的单元测试覆盖率进展不同，有的已经达到 80%，有的 20%，难以统一要求，但可以要求：新增代码必须具有单元测试，不得导致覆盖率下降，增加 bug 隐患。

GitLab 专业版的「合并请求批准」功能可以配置「覆盖率下降评审」，如果开发人员新写的代码导致覆盖率下降，必须经过同事审核，可配置多人、多个小组评审。

参考链接：[https://docs.gitlab.cn/jh/user/project/merge\\_requests/approvals/settings.html](https://docs.gitlab.cn/jh/user/project/merge_requests/approvals/settings.html)



GitLab（专业版+）设置合并请求：覆盖率下降需批准



基础设施

软件包与镜像库

分析

Wiki

代码片段

设置

通用

集成

Webhooks

仓库

收起侧边栏

合并请求批准

定义批准规则和设置，以确保新合并请求的

批准规则

核准人

需要核准


符合条件的用户 0

Vulnerability-Check 需要对漏洞进行审批。 启用

License-Check Requires approval for Denied licenses. 启用

Coverage-Check 测试覆盖率降低，需批准。 启用

GitLab（专业版+）合并请求：覆盖率降低需批准



开放中 Created 17分钟前 by 杨周 Owner

编辑 标记为草稿

Resolve "欢迎光临"

概览 0 提交 2 流水线 1 变更 2

流水线 #70190 已通过 使用提交 78e72a0c 于 5-welcome 16 minutes ago

测试覆盖率 54.84% (-2.85%) 来自1个作业

需要 1 个来自 Coverage-Check 的批准。

查看具备相关资格的核准人

测试总结报告 包含 未发生变化的测试结果 out of 3 total tests

查看完整报告 展开

合并 合并被阻止：必须批准此合并请求。

群组覆盖率报表

GitLab 专业版的「群组覆盖率报表」可查看旗下所有项目的覆盖率，比如：公司级、部门级、产品级、项目级，并且可以下载。便于全局管理。

参考链接：[https://docs.gitlab.com/ee/user/group/repositories\\_analytics/#current-group-code-coverage](https://docs.gitlab.com/ee/user/group/repositories_analytics/#current-group-code-coverage)

https://archergu.gitlab.io/why-gitlab-web/report

9/10

GitLab ( 专业版+ ) : 群组级查看所有项目测试覆盖率



Test Code Coverage

Latest test coverage results

3 projects selected

Project	Coverage	Coverage Jobs	Last Update
GitLab FOSS	40.79%	1	7 hours ago
GitLab Development Kit	87.69%	1	7 hours ago
gitlab-runner	76.00%	1	1 day ago

Code coverage statistics for master Mar 12 - Jun 10

Download raw data (.csv)

