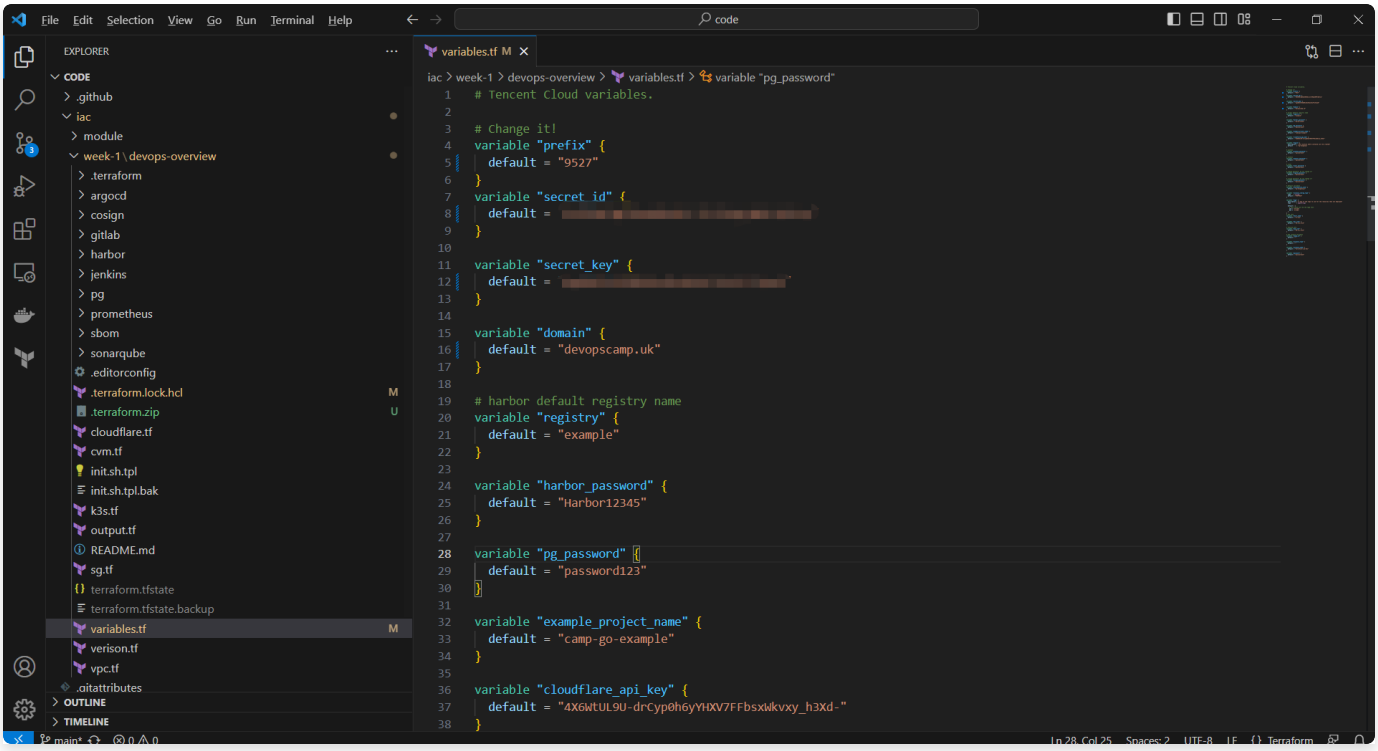
模块一：作业

一、请按照 Demo 做一遍实验，并提出一些改进建议，如果没办法完成所有 Demo，可以根据你体验到的 De...
1.1 基础设施即代码（Iac）
1.2 分支代码变更，自动触发构建
1.3 SnoarQube演示
1.4 快速拉起一套新的环境
1.5 镜像签名
1.6 镜像扫描
1.7 推送飞书
1.8 问题总结
二、请你思考敏捷开发中的“迭代”、“用户故事”和“任务”有哪些区别?
三、DevOps 工作流中如何避免“配置漂移”的问题（基础设施和应用配置）。
3.1 什么是“配置漂移”
3.2“配置漂移”会造成哪些问题
3.3 如何避免“配置漂移”

一、请按照 Demo 做一遍实验，并提出一些改进建议，如果没办法完成所有 Demo，可以根据你体验到的 Demo 提出一些改进建议。

1.1 基础设施即代码（Iac）



执行如下命令：

Shell

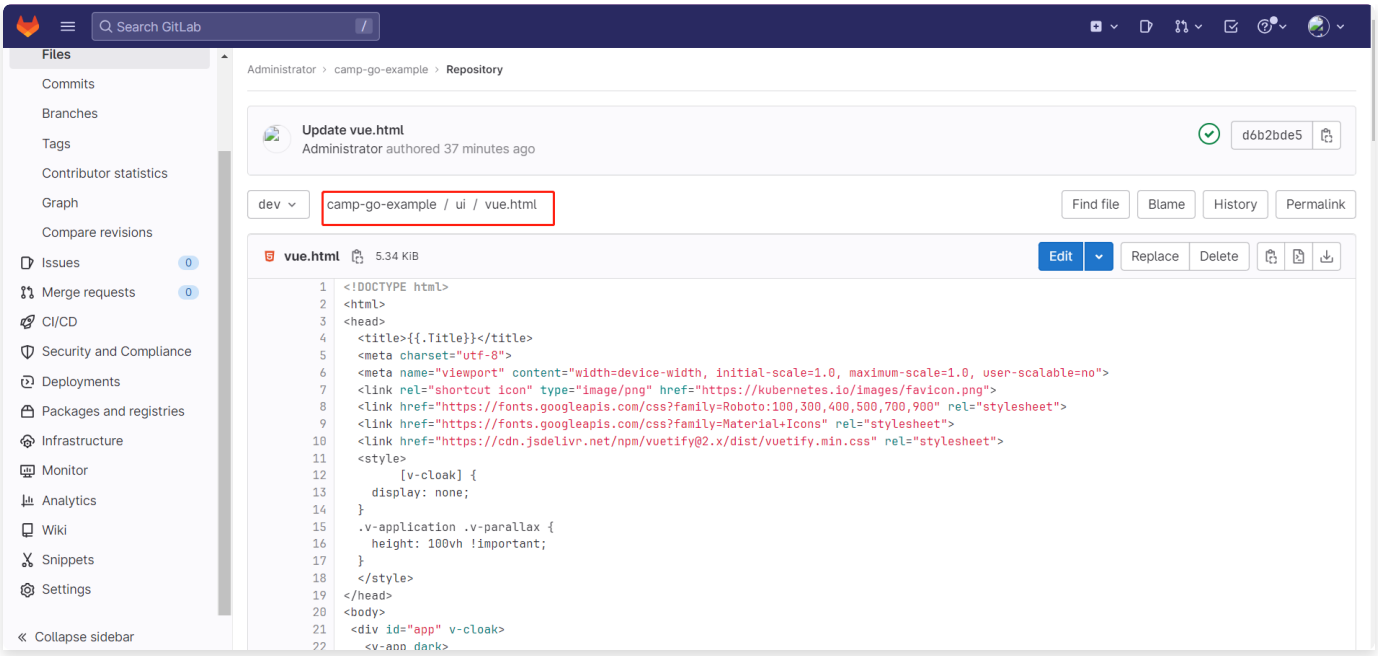
```
// 在 devops-overview 目录下执行
terraform init
export TF_VAR_secret_id=腾讯云SecretId
export TF_VAR_secret_key=腾讯云SecretKey
terraform apply -auto-approve
```

执行成功，会输出下面的信息 Shell

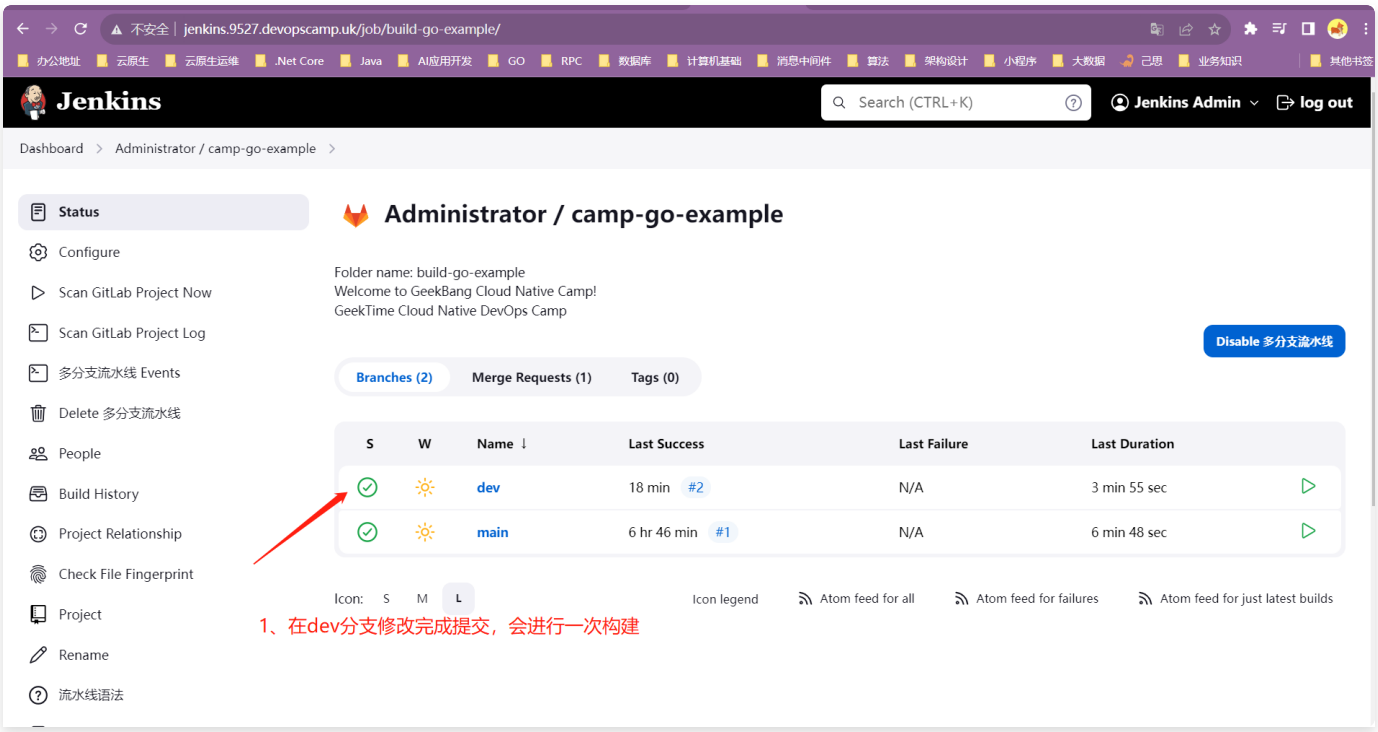
```
null_resource.connect_ubuntu (remote-exec): ##############################
####### Access Infrastruct ########################################

null_resource.connect_ubuntu (remote-exec): Access Grafana: http://grafan
a.9527.devopscamp.uk, admin, password123

null_resource.connect_ubuntu (remote-exec): Access Prometheus: http://prom
etheus.9527.devopscamp.uk

null_resource.connect_ubuntu (remote-exec): Access GitLab: https://gitlab.
9527.devopscamp.uk, root, j1sYvD3dyJRXRs8ZxuMyGYXzZ2ux5OPFwdsr2rfUMlGI55k3
LInoM7TzcHGAx5Ne, Access Token: glpat-A8zWQj-JT7vrJdvUKFgn

null_resource.connect_ubuntu (remote-exec): Access Jenkins: http://jenkin
s.9527.devopscamp.uk, admin, password123

null_resource.connect_ubuntu (remote-exec): Access Argo CD: http://argocd.
9527.devopscamp.uk, admin, password123

null_resource.connect_ubuntu (remote-exec): Access Harbor: https://harbor.
9527.devopscamp.uk, admin, Harbor12345

null_resource.connect_ubuntu (remote-exec): Access SonarQube: http://sona
r.9527.devopscamp.uk, admin, password123

null_resource.connect_ubuntu (remote-exec): Example Application Dev Enviro
nment: dev.podinfo.local, Add hosts: 43.129.249.118 dev.podinfo.local

null_resource.connect_ubuntu (remote-exec): Example Application Prod Envir
onment: prod.podinfo.local, Add hosts: 43.129.249.118 main.podinfo.local

null_resource.connect_ubuntu: Creation complete after 14m23s [id=438888081
1948447167]

Apply complete! Resources: 29 added, 0 changed, 0 destroyed.
```

1.2 分支代码变更，自动触发构建

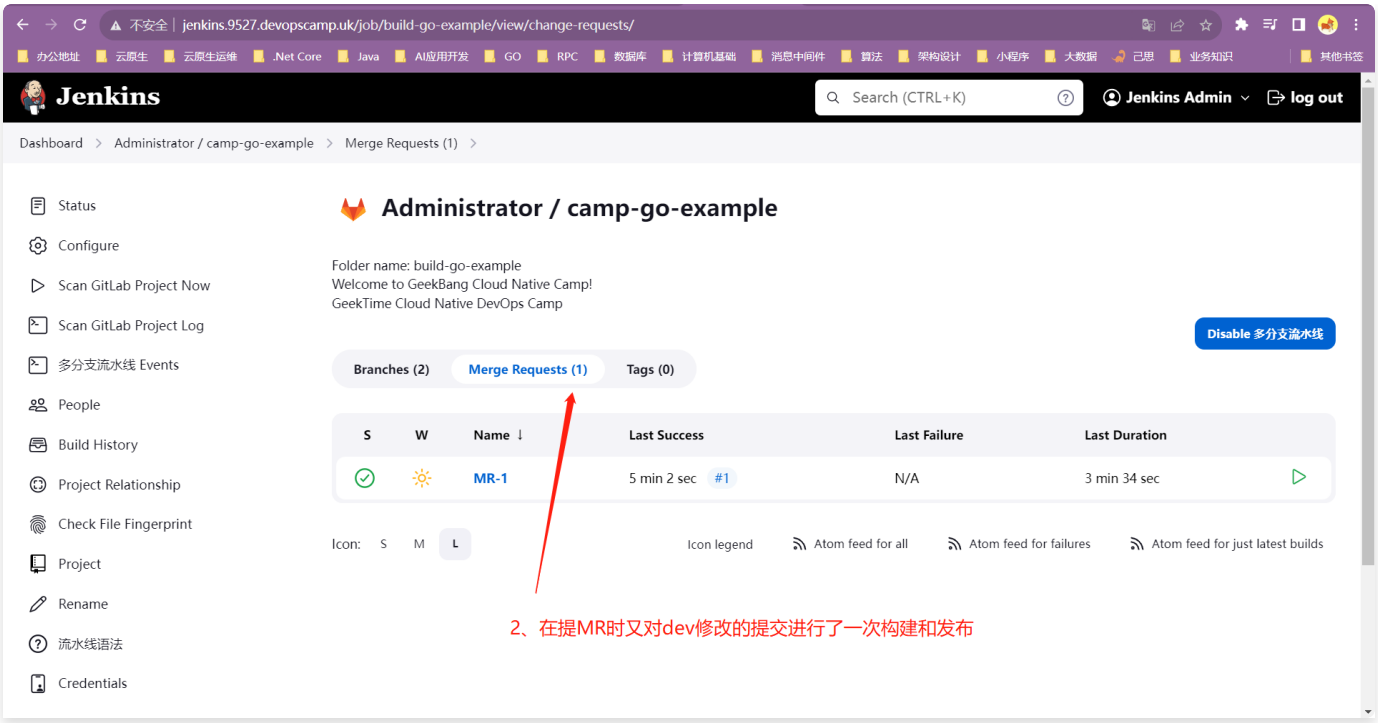
（1）开发人员修改dev分支中的代码，如下图所示：

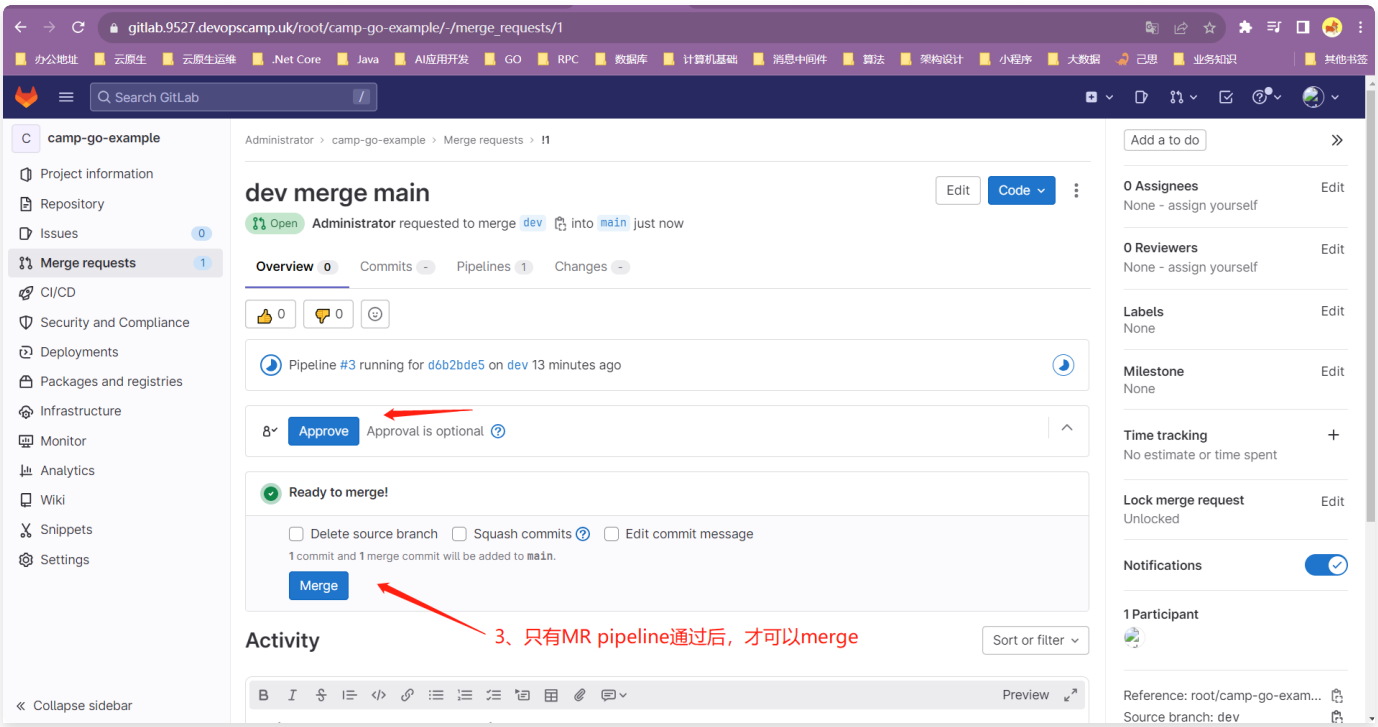


（2）会自动触发dev的代码构建，如下图所示：

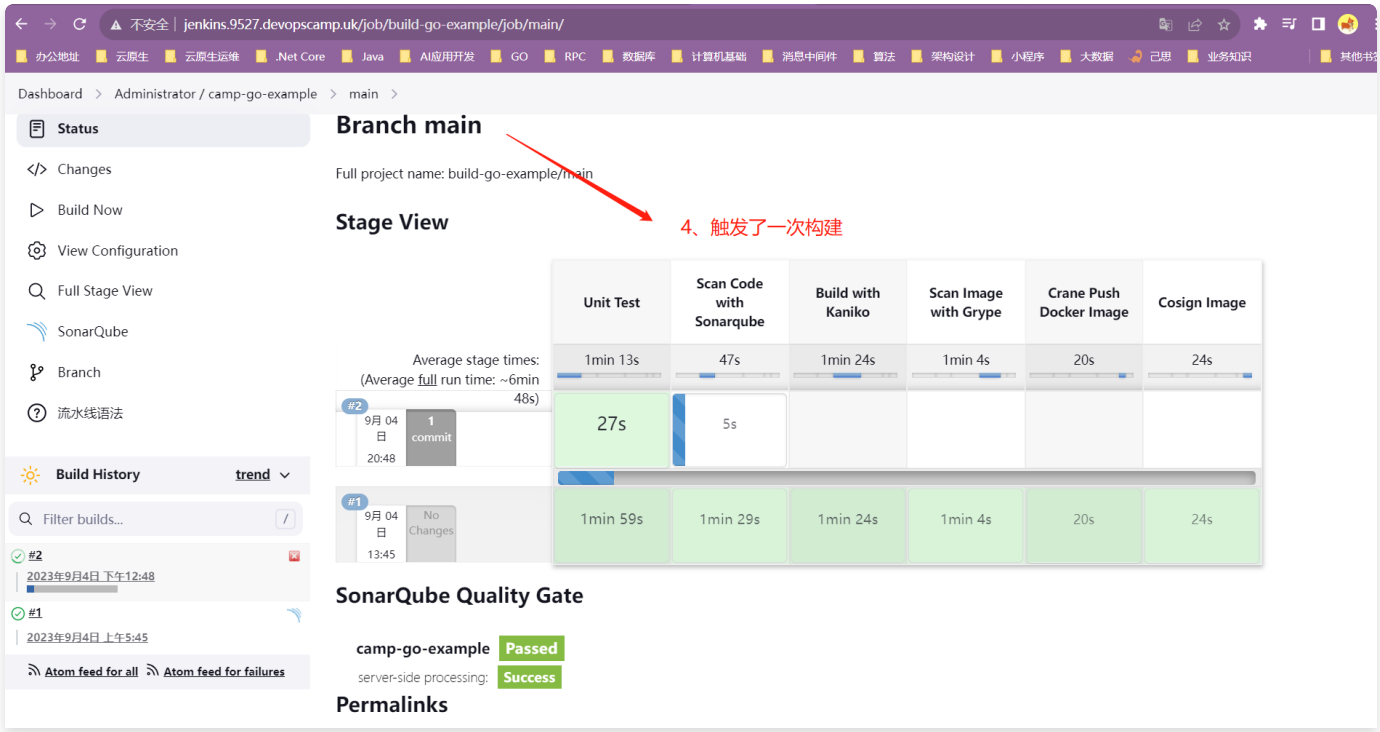


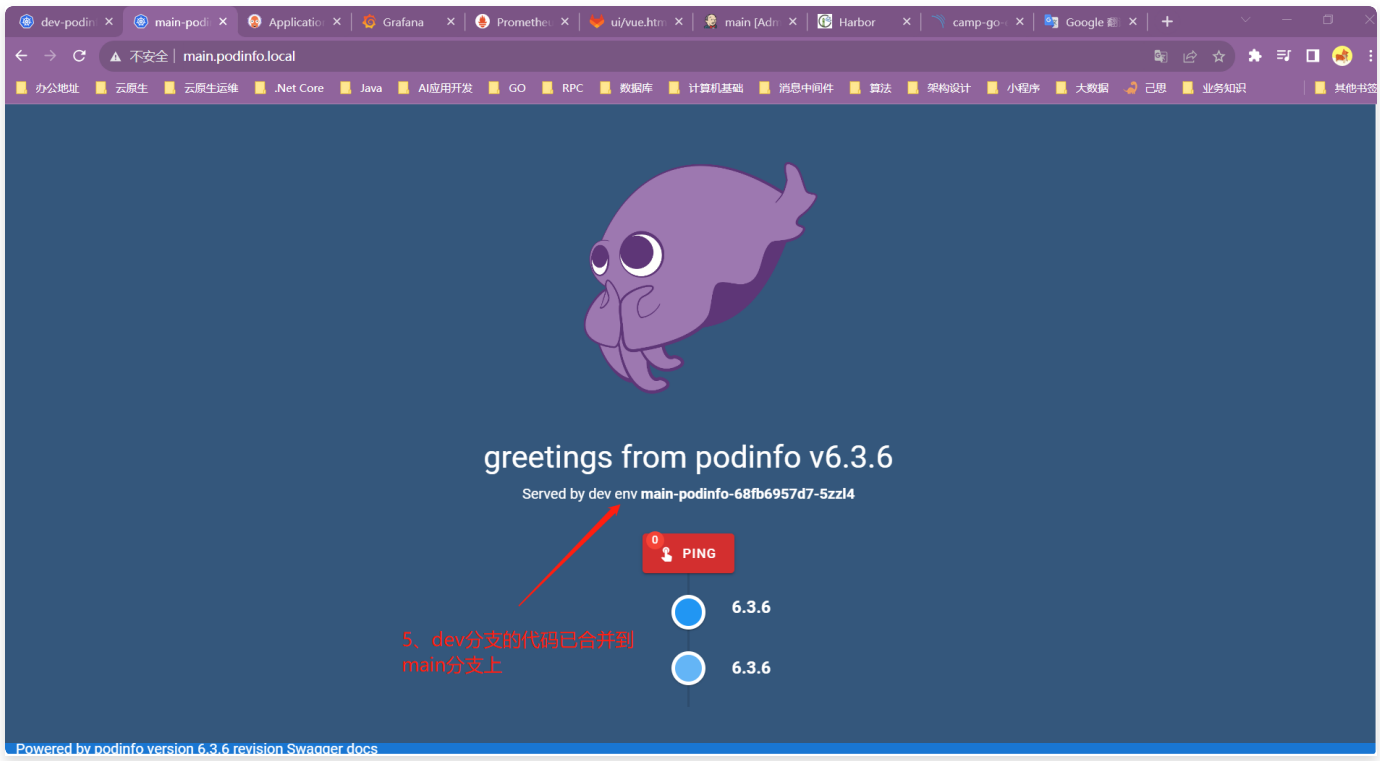
（3）上面的构建成功后，开发人员发起合并请求MR，如下图所示：





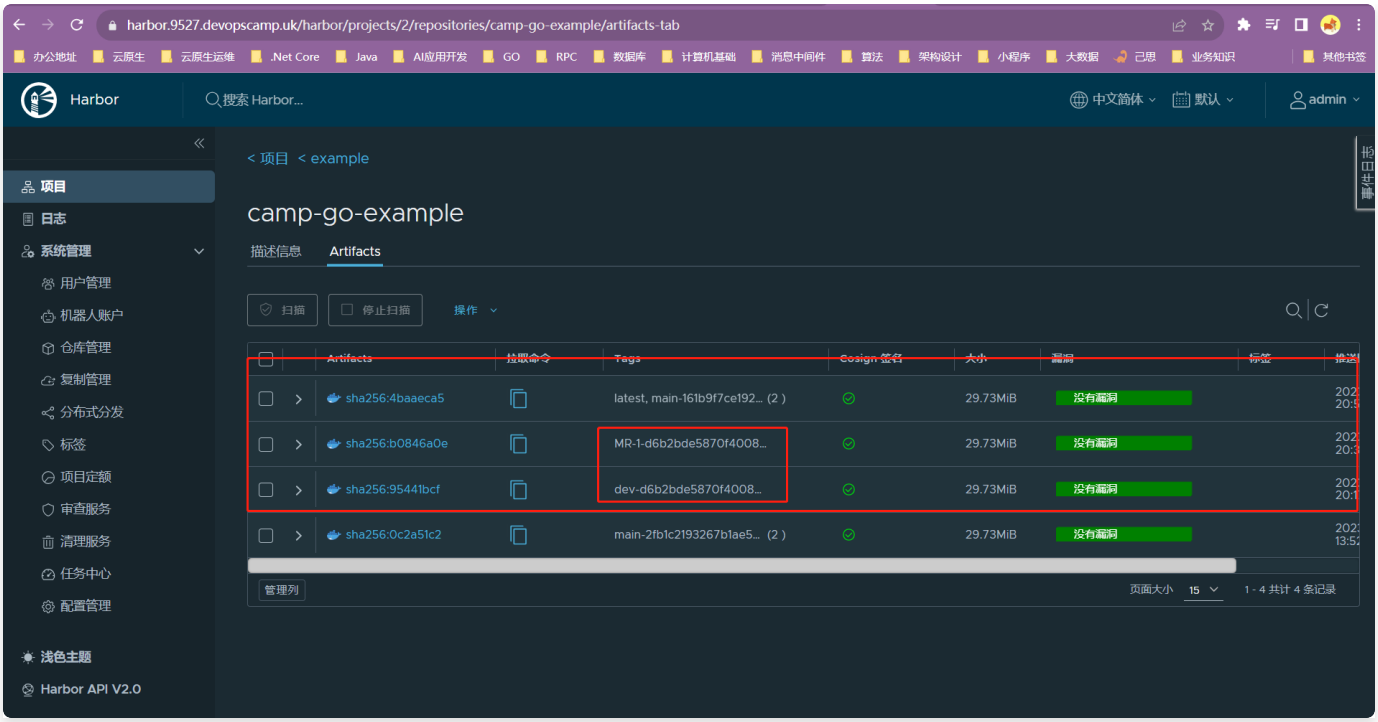
（4）这个时候dev修改的代码还没有合并到main分支上，点击“Merge”按钮后，就会触发main分支的构建。如下图所示：

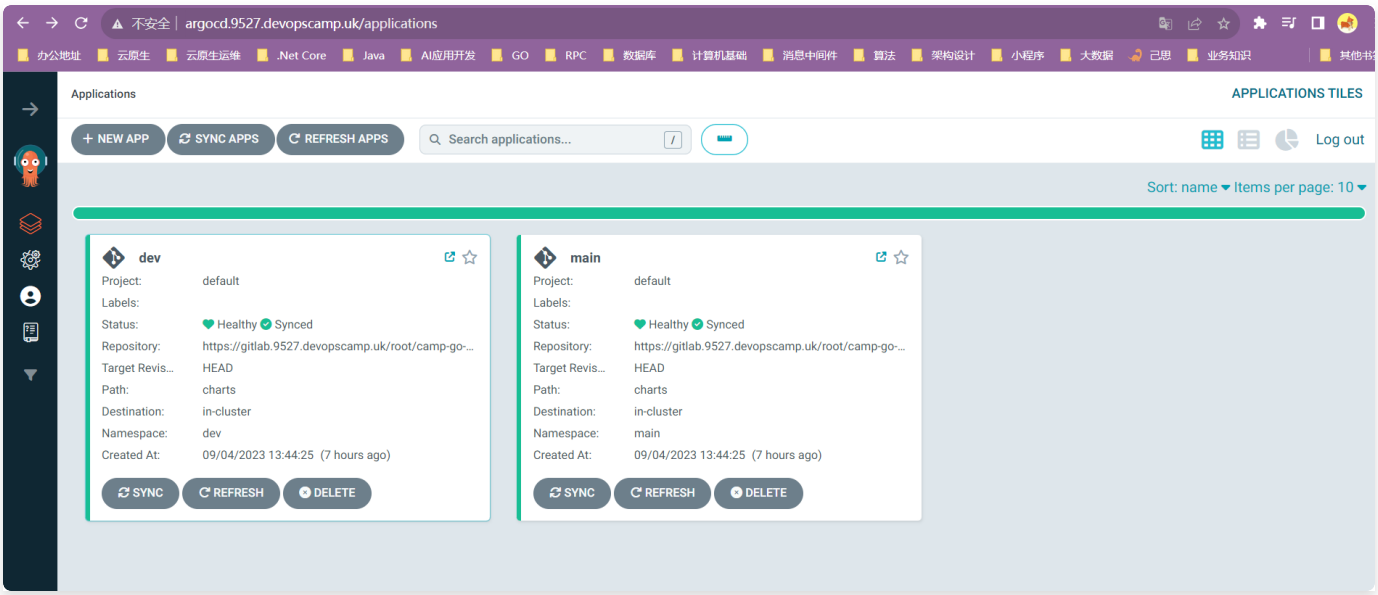




体验此过程有以下疑惑：

a. dev分支的提交会触发构建、更新dev环境；dev分支合并到main分支时的MR，也会触发构建，上传镜像，但是ArgoCD没有进行部署，MR的构建流程可以进行简化，感觉目前的MR流程太长，不知道企业里面都是怎么实现的，老师可以指点下。

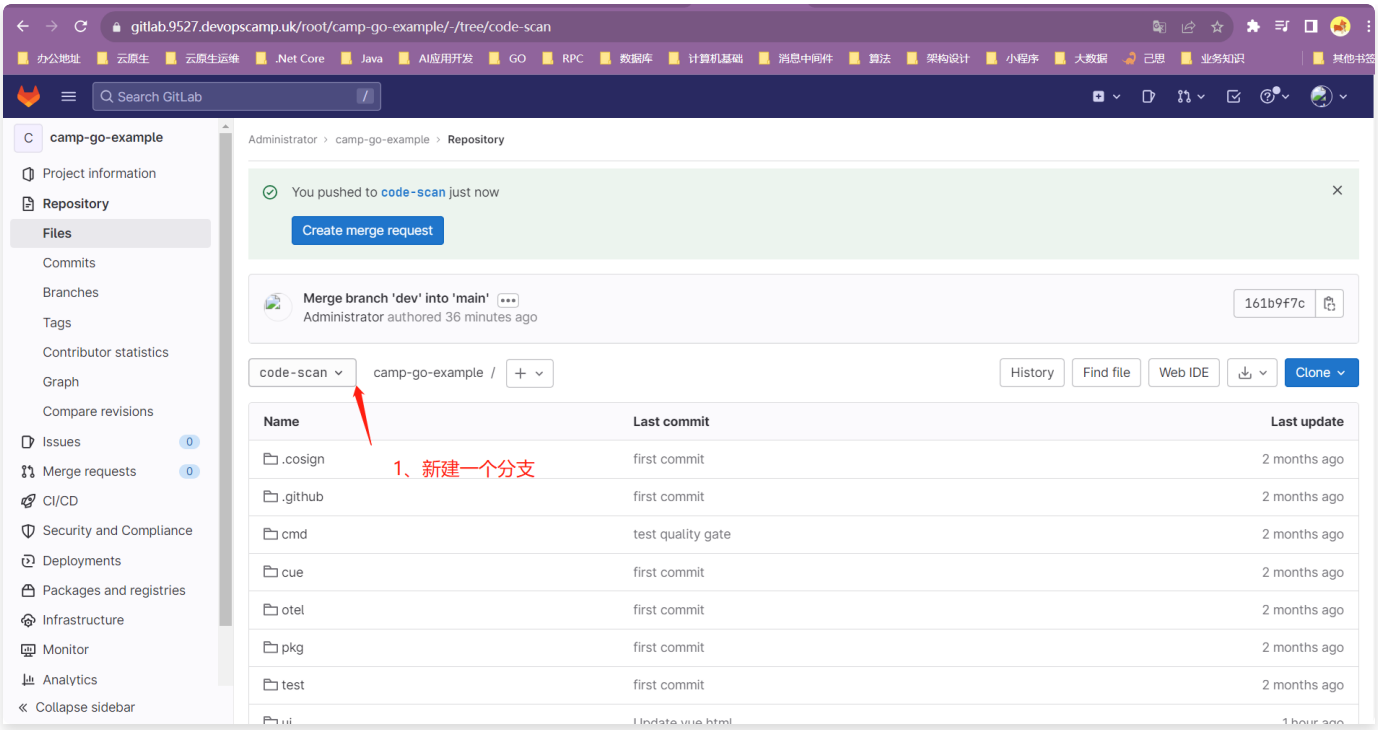




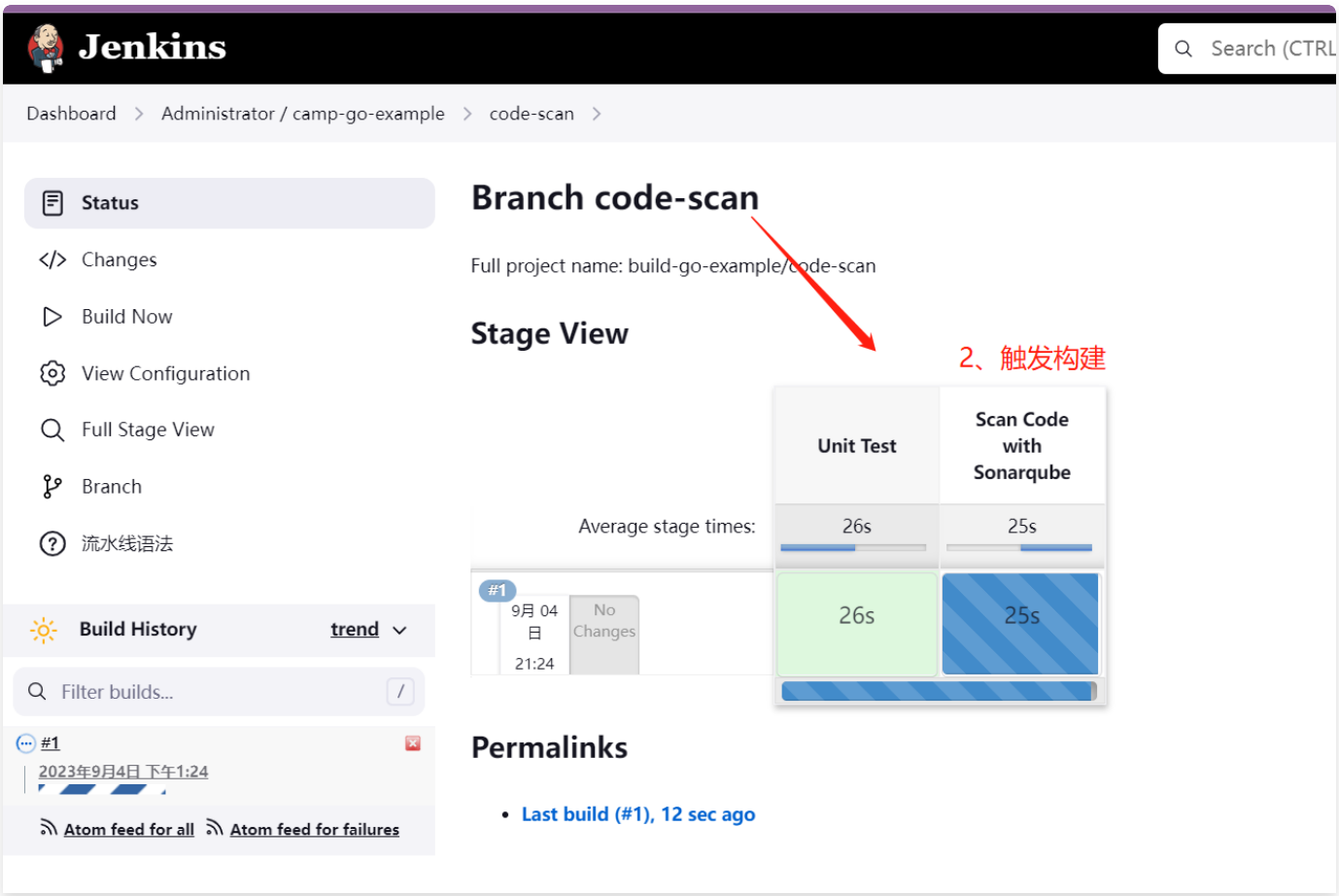
b. Jenkins的构建过程中，sonar生产的代码检查报告，还要登录sonar进行查看，能不能不用登录进去，直接可以查看，一般企业中的最佳实践是啥。

1.3 SnoarQube演示

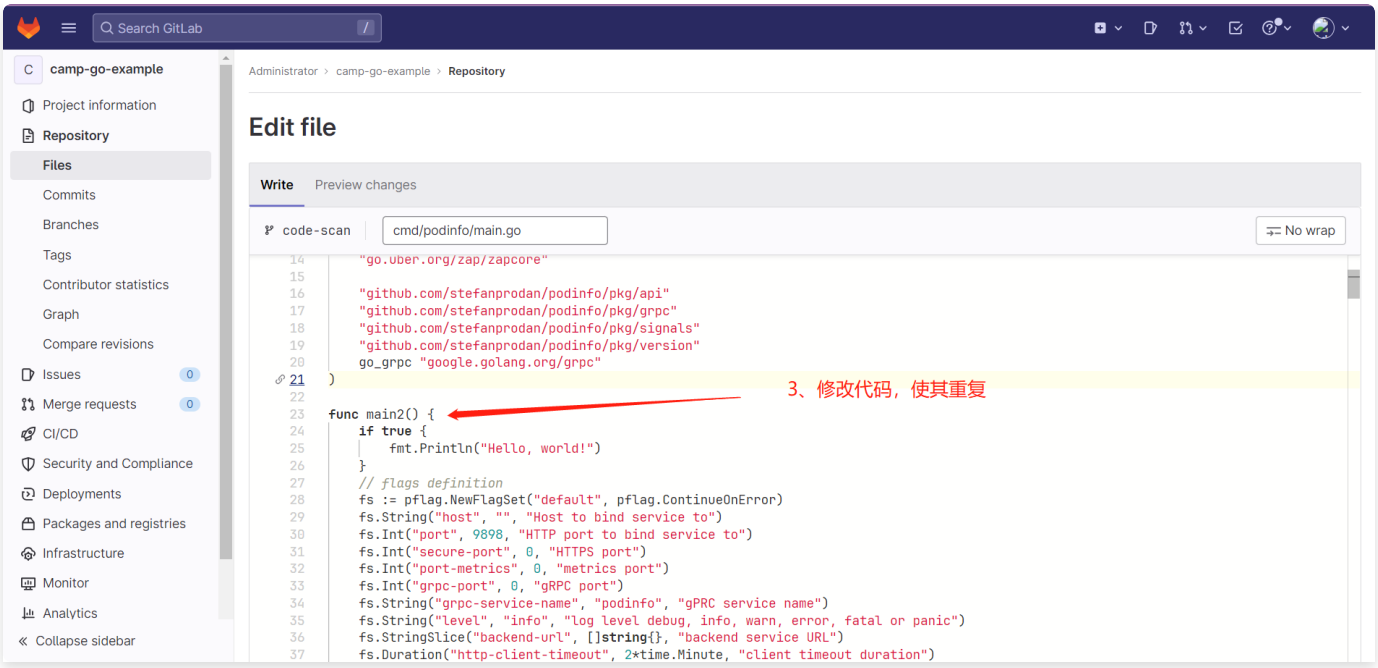
创建一个新分支，如下图所示：



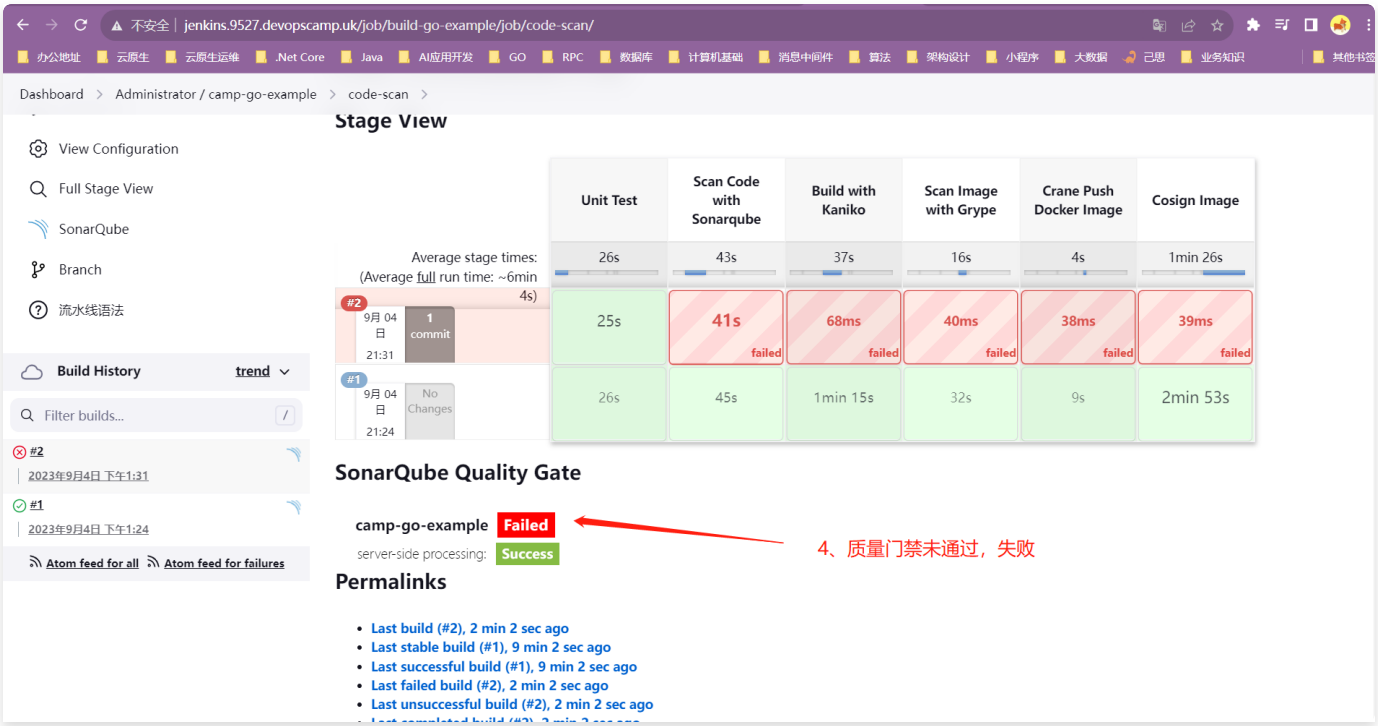
触发自动构建，如下图所示：

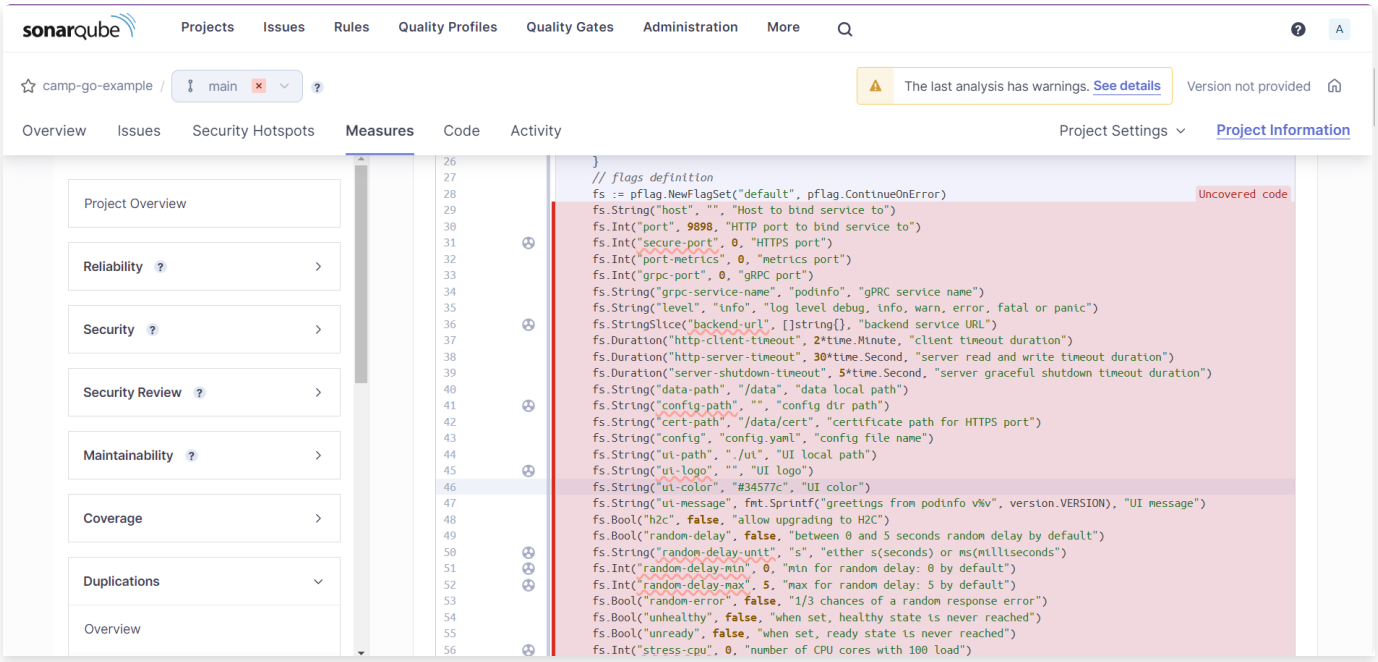


修改代码，使其重复，最后提交代码，如下图所示：

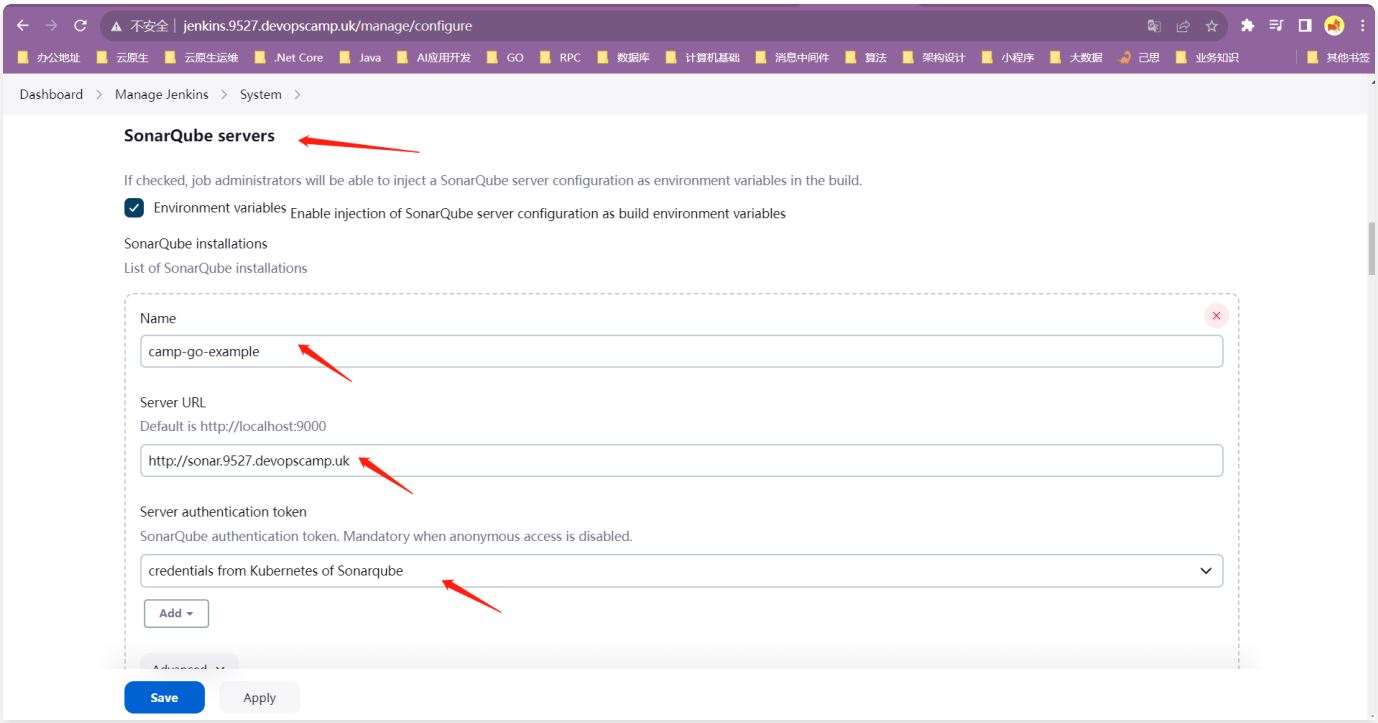


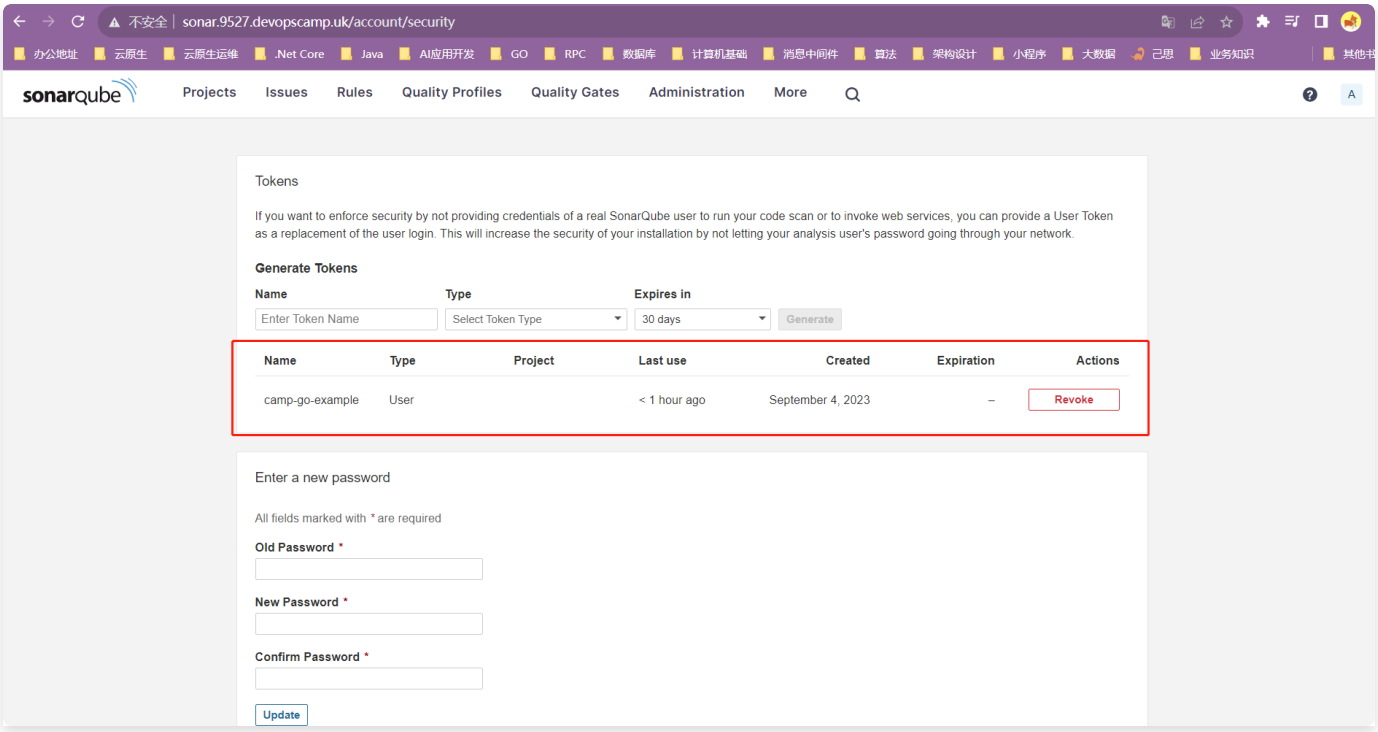
触发构建，如下图所示，代码质量门禁不通过，流水线失败：





Jenkins中 SonarQube servers 配置，如下图所示：



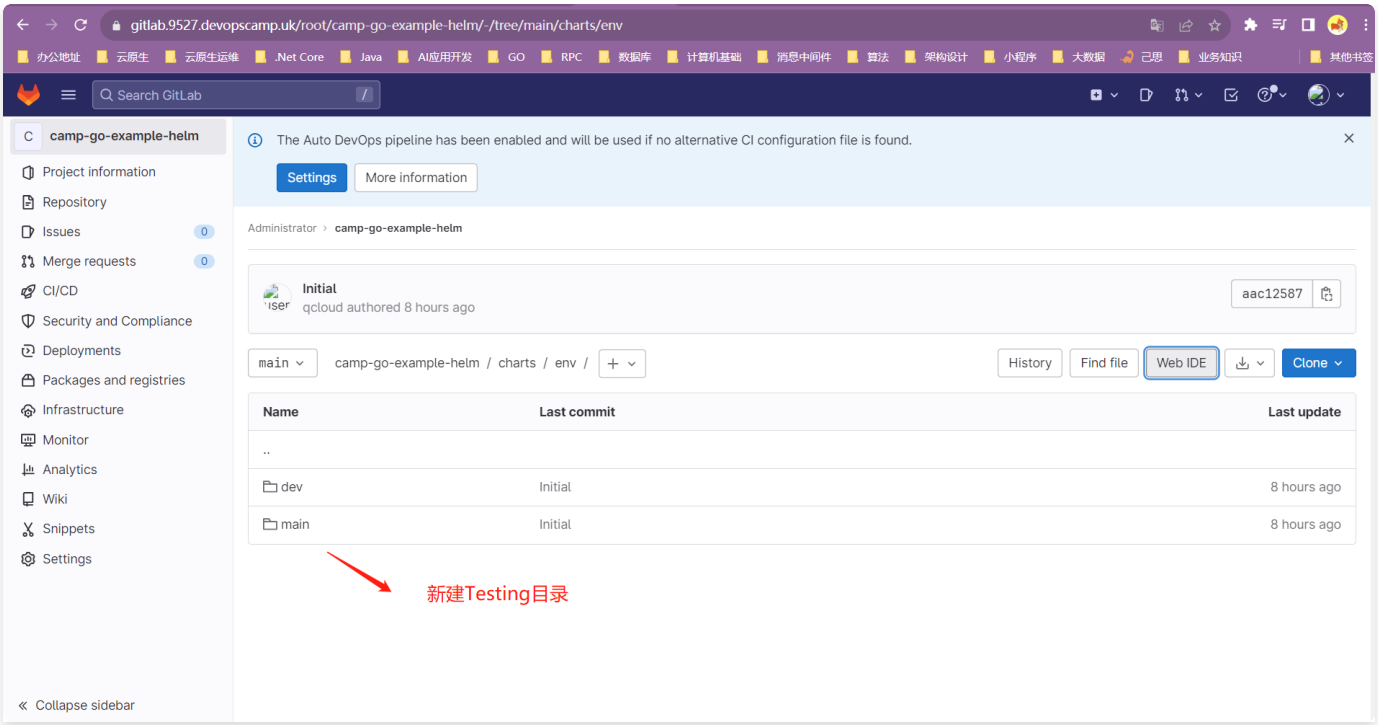


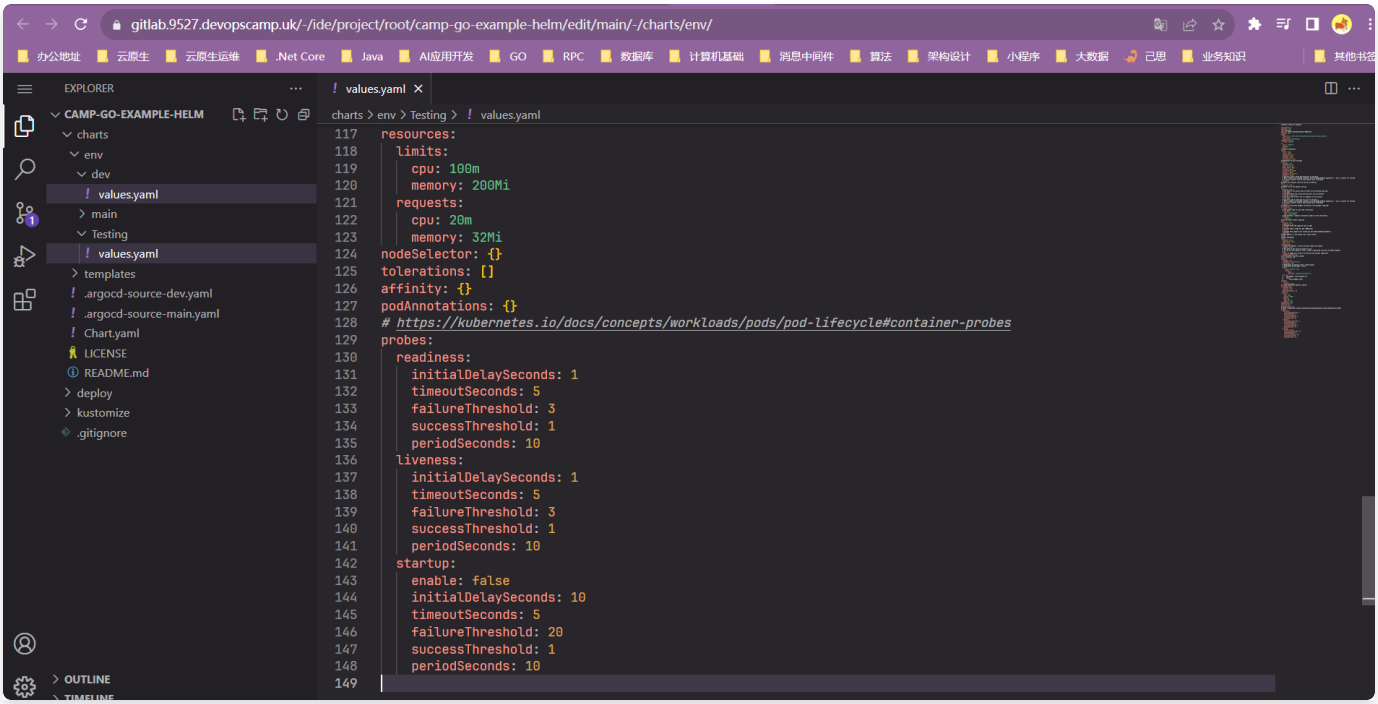
> 备注：Community Edition Version 10.1 (build 73491) 属性“sonar.login”已弃用，并将在将来删除。 传递令牌时请使用“sonar.token”属性。

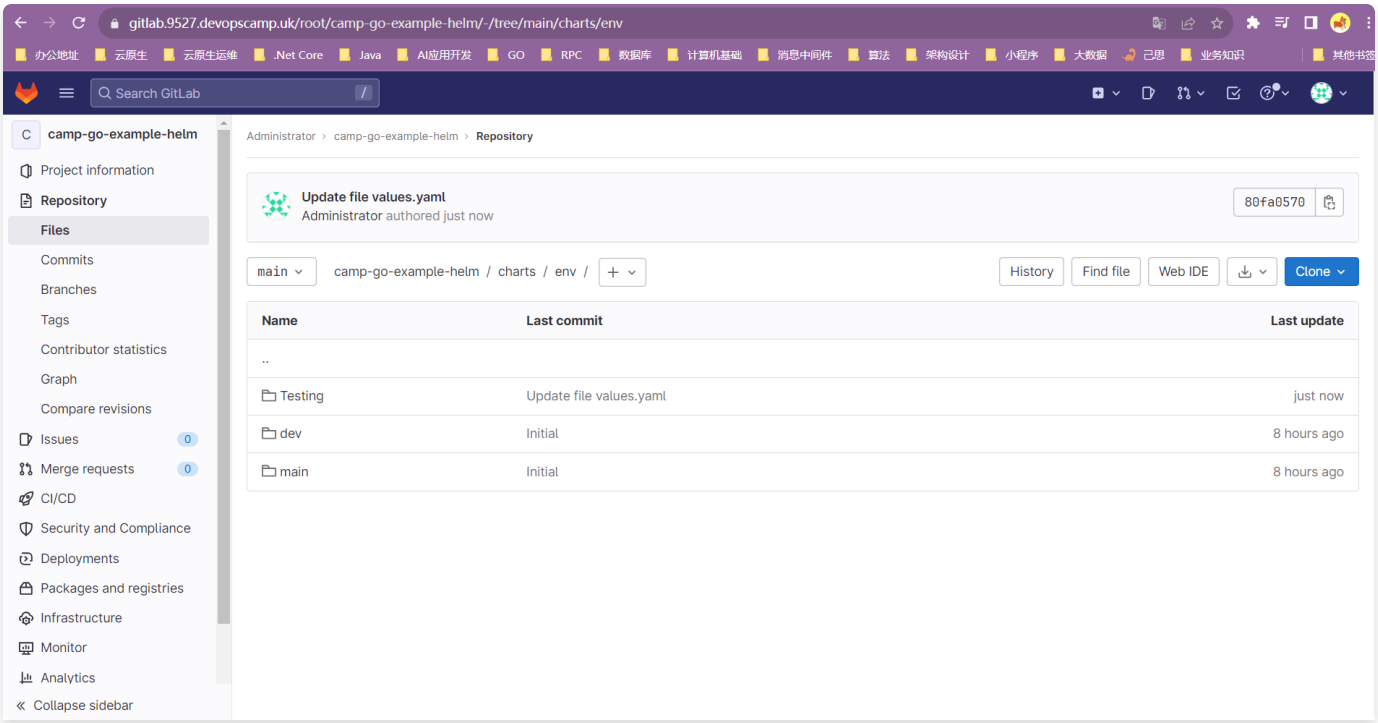
1.4 快速拉起一套新的环境

最佳实践：所有的应用定义只保持一个分支，不要多分支，多分支不好管理，可以单分支，多目录的方式进行管理。

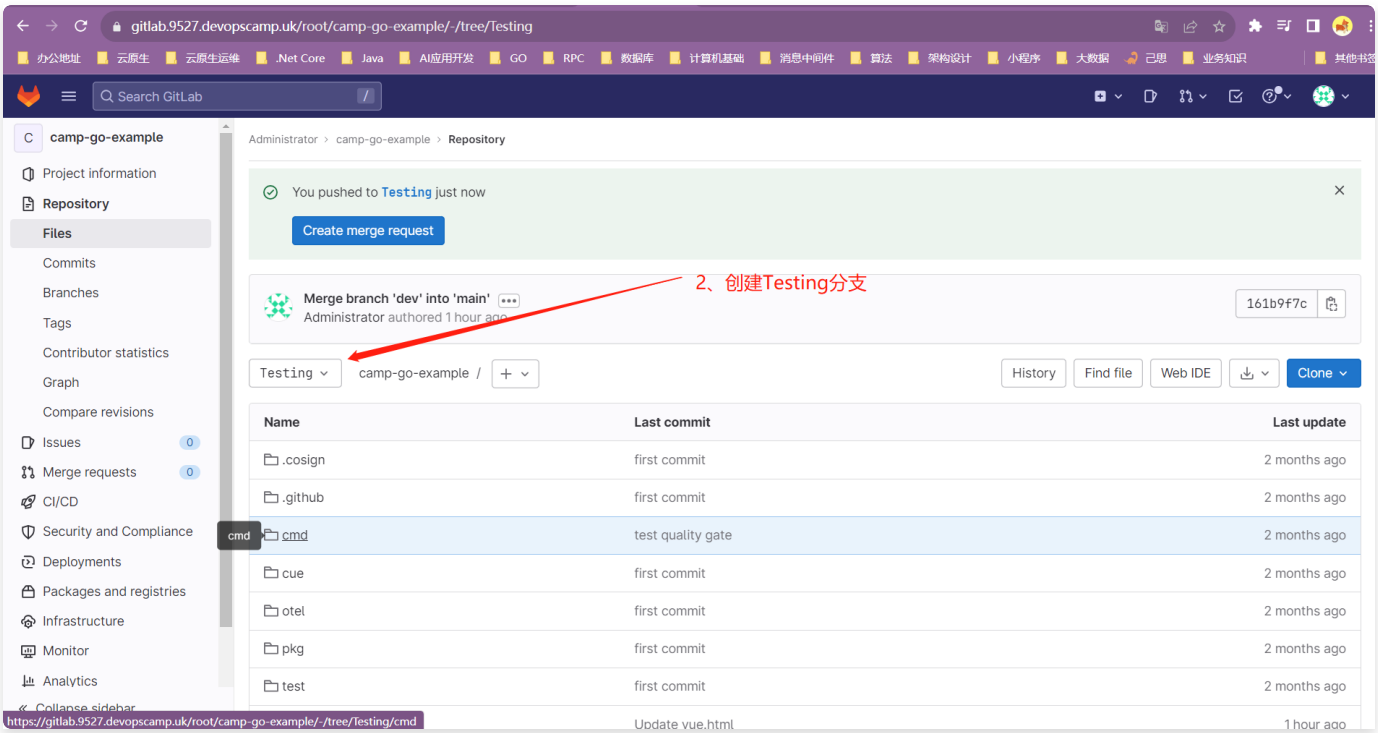
（1）定义Testing 环境


gitlab.9527.devopscamp.uk/root/camp-go-example-helm/-/tree/main/charts/env
Search GitLab
camp-go-example-helm
Project information
Repository
Issues
Merge requests
CI/CD
Security and Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets
Settings
Collapse sidebar
The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found.
Settings
More information
Administrator › camp-go-example-helm
Initial
qcloud authored 8 hours ago
aac12587
main
camp-go-example-helm / charts / env /
History
Find file
Web IDE
Clone
Name
Last commit
Last update
..
dev
Initial
8 hours ago
main
Initial
8 hours ago
新建Testing目录

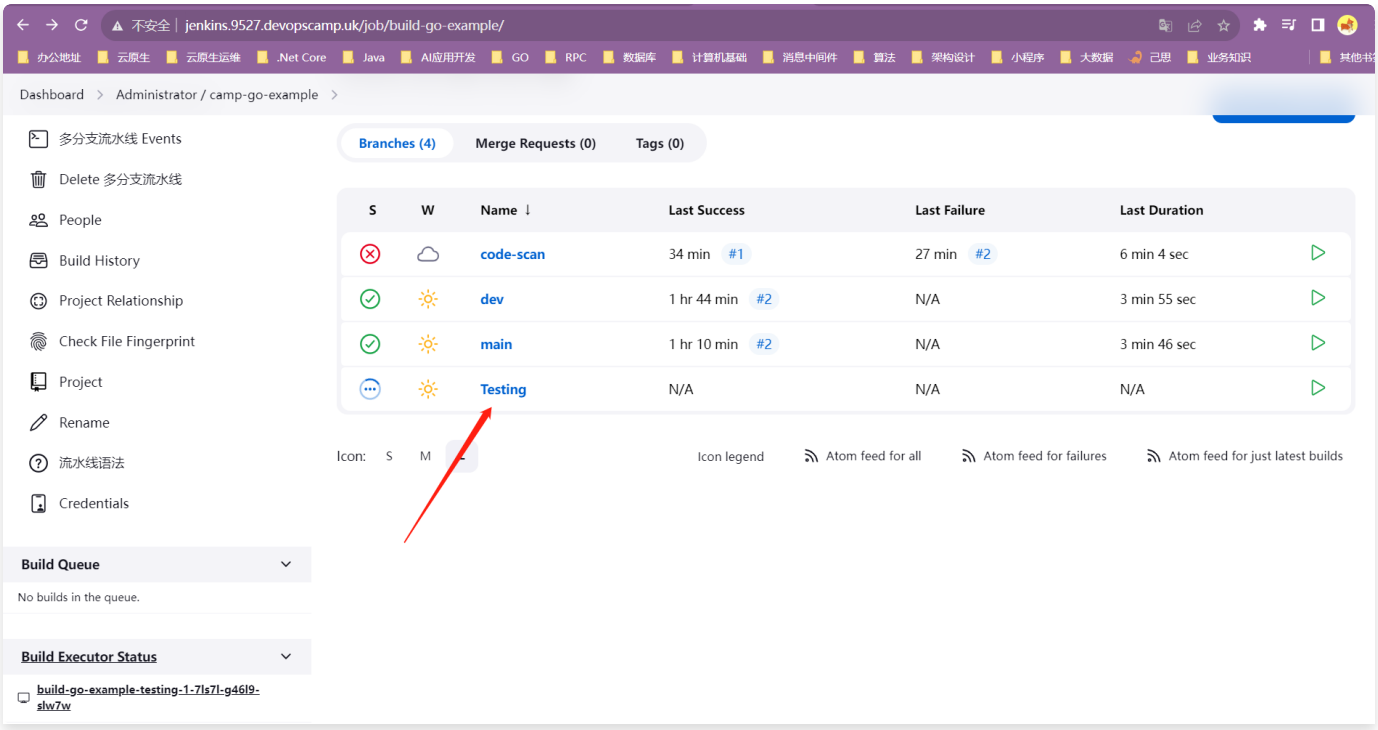

gitlab.9527.devopscamp.uk/-/ide/project/root/camp-go-example-helm/edit/main/-/charts/env/
EXPLORER
CAMP-GO-EXAMPLE-HELM
charts
env
dev
values.yaml
main
Testing
values.yaml
templates
.argocd-source-dev.yaml
.argocd-source-main.yaml
Chart.yaml
LICENSE
README.md
deploy
kustomize
.gitignore
OUTLINE
values.yaml
charts > env > Testing > values.yaml
resources:
limits:
cpu: 100m
memory: 200Mi
requests:
cpu: 20m
memory: 32Mi
nodeSelector: {}
tolerations: []
affinity: {}
podAnnotations: {}
https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes
probes:
readiness:
initialDelaySeconds: 1
timeoutSeconds: 5
failureThreshold: 3
successThreshold: 1
periodSeconds: 10
liveness:
initialDelaySeconds: 1
timeoutSeconds: 5
failureThreshold: 3
successThreshold: 1
periodSeconds: 10
startup:
enable: false
initialDelaySeconds: 10
timeoutSeconds: 5
failureThreshold: 20
successThreshold: 1
periodSeconds: 10



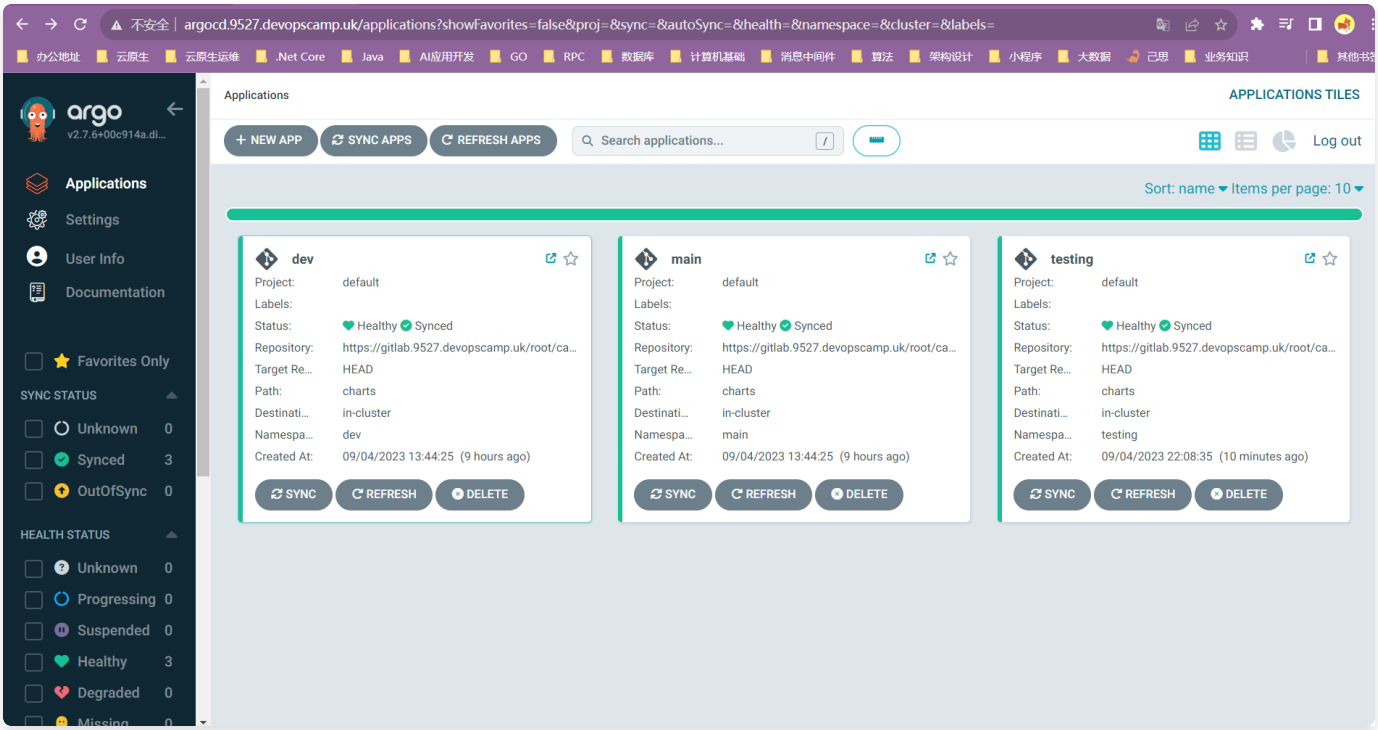
（2）创建Testing代码分支

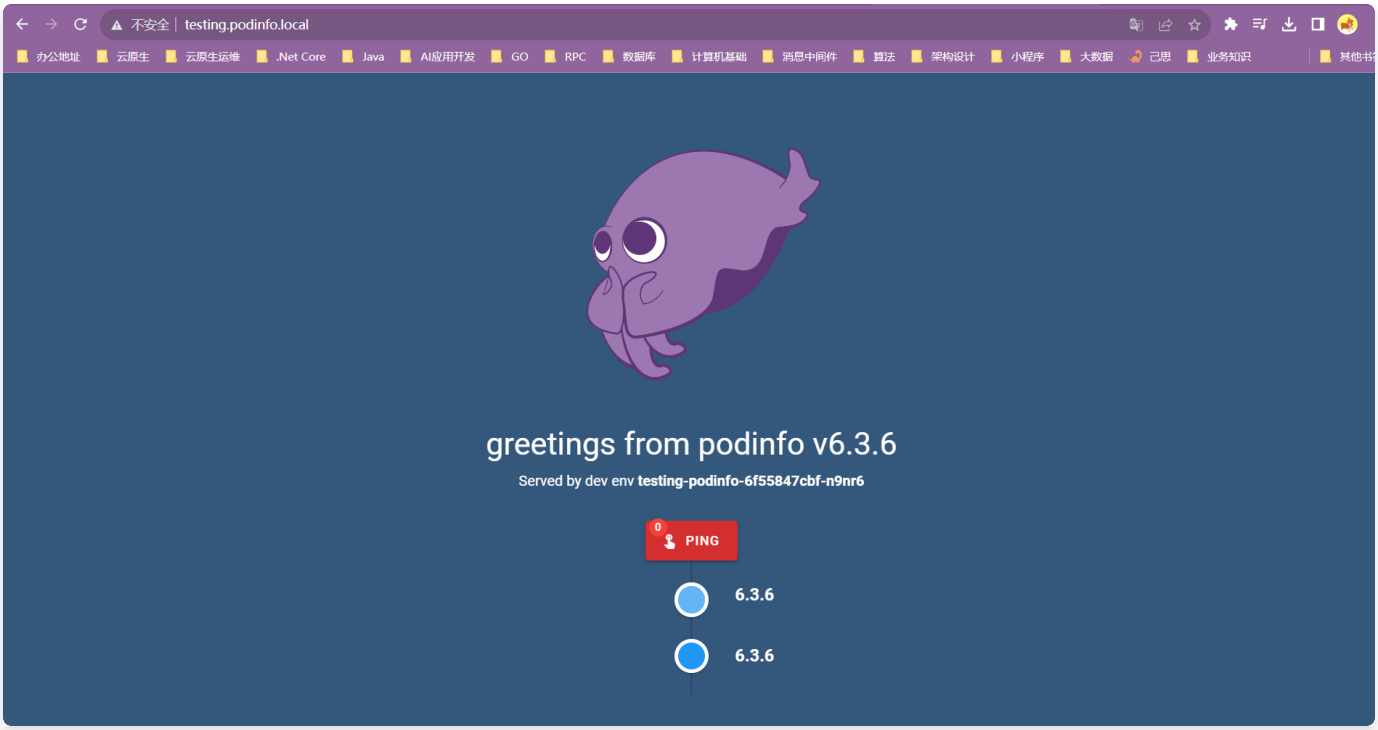


（3）自动触发构建



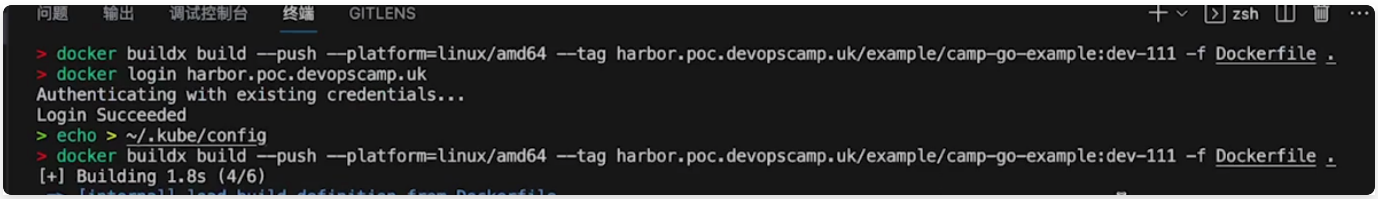
（4）在ArgoCD 中没有出现对应的Testing环境，后来尝试统一把Testing 修改为 testing，Argo CD 中就出现了testing环境，如下图所示：





1.5 镜像签名

简单暂时可以略过



1.6 镜像扫描

简单暂时可以略过

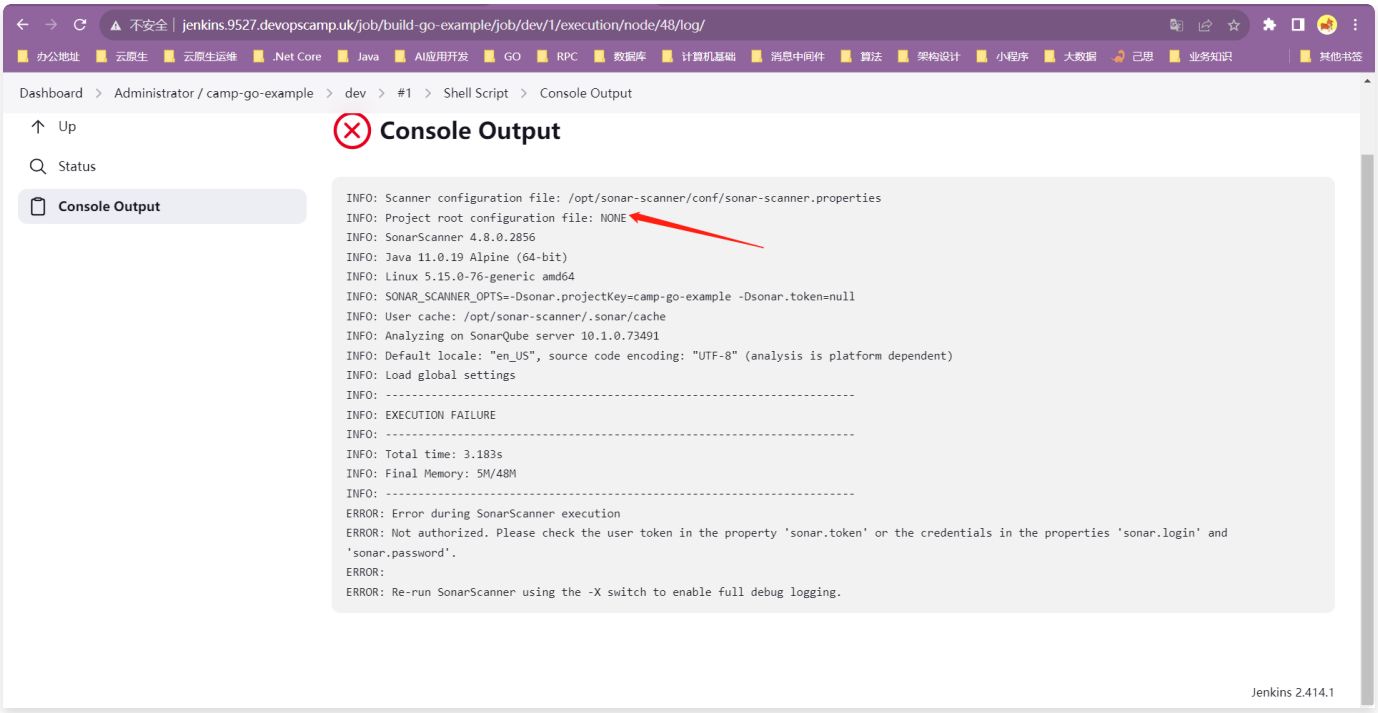
1.7 推送飞书

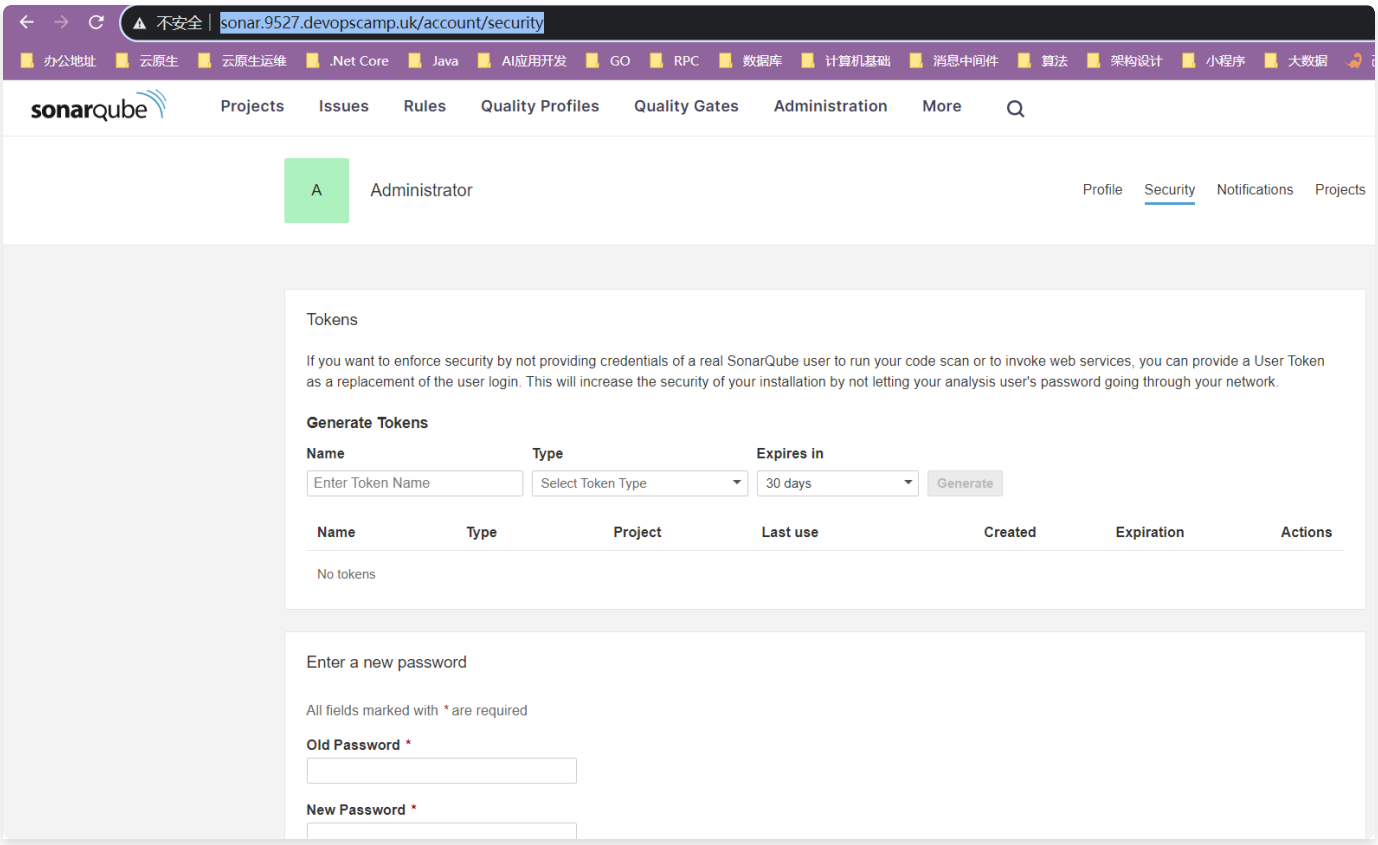
简单暂时可以略过

1.8 问题总结

（1）1.2 和 1.4 已列出

（2）第一次实验时遇到的问题





解决办法：可以手动创建一下，然后在jenkins系统设置里配一下。

二、请你思考敏捷开发中的“迭代”、“用户故事”和“任务”有哪些区别?

在敏捷开发中，"迭代"、"用户故事"和"任务"是三个相关但具有不同含义的概念。

1. 迭代（Iteration）：

迭代是敏捷开发中时间上的划分单位。它代表了一个固定的时间段，通常是几周到一个月的时间，用于进行软件开发工作。每个迭代都是一个完整的开发周期，包括需求分析、设计、编码、测试和交付。迭代的目的是迭代交付可用的软件功能，以便在每个迭代结束时，团队可以获得一个可工作的软件版本。

2. 用户故事（User Story）：
   用户故事是敏捷开发中用于描述软件功能需求的一种工具。它以用户的角度描述软件的功能需求，并强调用户期望达到的目标。用户故事通常采用以下格式：作为一个[角色]，我希望[功能]，以便[目标]。用户故事是较为简短的描述，重点在于传达用户需求的核心要点和价值，而不是详细说明实现细节。
3. 任务（Task）：
   任务是实现用户故事所需的具体工作单元。当用户故事确定后，团队将分解用户故事为更小的任务，以便更好地安排和管理工作。任务是具体的、可执行的工作项，通常由开发人员执行。任务可以包括编码、设计、测试、文档编写等活动，每个任务都有一个预估的工作量和截止日期。

总结：
迭代是一个时间上的划分单位，代表一个完整的开发周期；用户故事是描述软件功能需求的工具，以用户的角度描述功能和目标；任务是实现用户故事所需的具体工作项，用于安排和管理开发工作。在敏捷开发中，团队通过迭代周期性地交付可用的软件功能，用户故事用于描述需求，而任务用于实现需求。

三、DevOps 工作流中如何避免“配置漂移”的问题（基础设施和应用配置）。

3.1 什么是“配置漂移”

（1）发生在GitOps系统之外对基础设施和应用配置进行了修改，而不是向Git仓库提交所需的修改，我们称之为“配置漂移”。在用GitOps 管理系统的时候，这通常是一个很大的“禁区”，应该避免在GitOps 之外直接修改系统。

（2）可以使用 kubectl diff 和 kubediff 工具来检查应用程序的同步状态和配置漂移情况。

3.2“配置漂移”会造成哪些问题

（1）当发生时，GitOps Operator 需要“观测”当前的状态，发现与期望状态的差异，并向用户指出应用程序的不同步，有些系统会认为配置漂移是一种错误状态，可能会重新部署之前最近一次的部署配置，从而覆盖了手动更改的；有些系统可能会检测到这种漂移，并允许将手动修改的整合保

存在git 中的声明状态（例如双向同步），双向同步是不可取的，因为它允许并鼓励对集群进行手动更改，并绕过了GitOps 作为其核心优势之一提供的安全和审查过程。

（2）不一致的配置可能导致应用程序在不同环境中的行为不一致，从而导致错误和故障。

（3）由于配置的不一致性，当出现问题时，排查问题变得更加困难，因为我们不能确定应用程序的实际配置是什么。

（4）配置漂移可能导致系统的安全性降低，例如，如果数据库连接字符串被错误地更改为公共可访问的值，可能会导致数据泄露的风险。

3.3 如何避免“配置漂移”

（1）**自动化配置管理**：采用自动化工具和流程来管理配置是减少配置漂移的关键。使用配置管理工具（如Ansible、Puppet、Chef）来自动化配置的部署和管理，确保配置的一致性和正确性。自动化配置管理可以确保在不同环境中使用相同的配置，并减少人为错误。

（2）**基础设施即代码（Iac）**：将基础设施配置和部署过程纳入版本控制系统，并采用基础设施即代码的方法来管理和部署基础设施。通过使用工具如Terraform或CloudFormation，可以将基础设施定义为可重复、可管理的代码，并确保基础设施的配置与预期一致。

（3）**环境隔离**：将不同环境（如开发、测试和生产）的配置隔离开来，每个环境都有其独立的配置。这可以通过使用不同的配置文件、环境变量或配置管理工具的分支来实现。确保在每个环境中使用适当的配置，避免将测试配置或开发配置误用于生产环境。

（4）**配置版本控制**：将应用程序和基础设施的配置文件纳入版本控制系统，以便能够跟踪配置的变化并进行回滚或还原。确保所有配置更改都通过版本控制进行管理，并记录每个更改的目的、时间和责任人。这样可以追溯配置漂移问题的根源，快速恢复到稳定的配置状态。

（5）**持续集成和持续部署（CI/CD）**：使用CI/CD流水线来自动化应用程序的构建、测试和部署过程。将配置管理集成到CI/CD流水线中，确保每个部署都使用正确的配置。通过自动化部署过程，减少手动配置的机会，从而降低配置漂移的风险。

（6）**监控和告警**：设置监控系统来监视应用程序和基础设施的配置状态。通过监测配置的变化并及时发出告警，可以快速检测到配置漂移问题，并采取相应的纠正措施。

（7）**定期审查和验证**：定期审查配置文件和环境，确保配置的一致性和合规性。进行配置验证和合规性检查，确保配置符合最佳实践和安全标准。

综上所述，通过自动化配置管理、基础设施即代码、环境隔离、配置版本控制、CI/CD流水线、监控和告警以及定期审查，可以帮助避免配置漂移问题，并确保应用程序和基础设施的配置始终与预期一致。

参考资料：

（1）chatgpt

（2）GitOps and Kubernetes 使用GitOps 实现kubernetes 的持续部署模式、流程及工具