



GitLab 白名單控制



gitlab.yml，我需要能設置白名單控制job能不能執行，例如白名單: white-1/white-2/apple，代表只有project_namespace=white-1或white-2或apple才可以執行，"white"沒有完全匹配就不能執行，我的白名單很多，所以給我通用的設置白名單方式



在 GitLab CI/CD 中，可以通过使用 `rules` 和环境变量的组合来实现白名单控制特定 Job 是否执行。以下是一个通用的 `.gitlab-ci.yml` 配置示例，可以根据项目的 `CI_PROJECT_NAMESPACE` 来控制 Job 是否执行：

通用配置 `.gitlab-ci.yml`

```
stages:
  - test

# 定义一个白名单变量
variables:
  WHITELIST: "white-1,white-2,apple"

job_with_whitelist:
  stage: test
  script:
    - echo "This job runs only for whitelisted namespaces."
  rules:
    # 使用脚本检查是否属于白名单
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```

关键部分说明

1. `WHITELIST` 变量：

- 定义一个包含多个白名单值的逗号分隔字符串，例如 `white-1,white-2,apple`。
- 可以根据需要扩展或修改白名单内容。

2. `rules` 关键字：

- 使用 `if` 条件检查 `CI_PROJECT_NAMESPACE` 是否在 `WHITELIST` 中。
- 正则表达式 `(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)` 用于精确匹配单独的名称，避免部分匹配问题。

3. 条件规则：

- 如果 `CI_PROJECT_NAMESPACE` 属于白名单，Job 会正常执行。
- 如果不在白名单中，Job 被设置为 `when: never`，不会执行。



白名单扩展

如果白名单非常长，可以通过环境变量管理，例如使用 CI/CD Variables 在 GitLab Web 界面中配置全局的 `WHITELIST`，避免直接在 `.gitlab-ci.yml` 中硬编码。

- 前往 GitLab 项目的 `Settings > CI/CD > Variables`。
- 添加 `WHITELIST` 环境变量，值为 `white-1,white-2,apple,...`。

测试方法

1. 提交代码到测试项目。
2. 确保 `CI_PROJECT_NAMESPACE` 设置正确。
3. 检查是否正确触发符合白名单规则的 Job。

这样可以灵活地通过白名单控制 Job 的执行，且易于维护和扩展。



請詳細說明检查是否属于白名单



检查是否属于白名单的核心逻辑是使用条件语句和正则表达式来验证当前的 `CI_PROJECT_NAMESPACE` 是否出现在白名单中。这部分主要在 `rules` 的 `if` 条件中完成。

逻辑细节

1. 白名单存储:

- 将白名单值存储为一个逗号分隔的字符串，例如：`WHITELIST="white-1,white-2,apple"`。
- 使用 GitLab 的环境变量 (`variables`) 管理白名单，便于后续扩展。

2. 验证 `CI_PROJECT_NAMESPACE`:

- 使用 GitLab 提供的环境变量 `\$CI_PROJECT_NAMESPACE`，它表示当前项目的命名空间（通常是项目的 Group）。
- 将 `\$CI_PROJECT_NAMESPACE` 与 `WHITELIST` 中的值进行逐一比对，确保精确匹配（避免部分匹配）。

3. 正则表达式匹配:

- 正则表达式 `(?:^|,)\$CI_PROJECT_NAMESPACE(?:,\$)` 用于精确匹配，解析如下：
 - `(?:^|,)`：匹配开头或逗号，确保命名空间是从开头或前一个值的末尾开始。
 - `\$CI_PROJECT_NAMESPACE`：当前项目命名空间。
 - `(?:,\$)`：匹配逗号或字符串结尾，确保命名空间是最后一个值或后面有分隔符。
- 这种方式避免了部分匹配问题，例如：





- 如果白名单中有 `white-1`，而项目命名空间是 `white-10`，正则表达式会阻止错误匹配。

4. 条件规则:

- 通过 `rules` 的 `if` 条件检查：

```
- if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
```

- `WHITELIST =~ ...`: 使用正则表达式匹配 `CI_PROJECT_NAMESPACE`。
- `when: on_success`: 如果匹配成功，Job 执行。
- `when: never`: 如果不匹配，Job 不执行。

示例代码解读

```
job_with_whitelist:
  stage: test
  script:
    - echo "This job runs only for whitelisted namespaces."
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'\
      when: on_success
    - when: never
```

运行逻辑：

1. 检查变量是否存在:

- `CI_PROJECT_NAMESPACE`: 确保项目命名空间变量存在。
- `WHITELIST`: 确保白名单变量存在。

2. 正则表达式匹配:

- 示例：
 - `WHITELIST="white-1,white-2,apple"`.
 - `CI_PROJECT_NAMESPACE="white-1"`:
 - 正则匹配：`(?:^|,)white-1(?:,|\$)`。
 - 匹配成功，Job 执行。
 - `CI_PROJECT_NAMESPACE="white"`:
 - 正则匹配失败，Job 不执行。

3. 控制 Job 执行:

- 如果匹配成功，`when: on_success` 执行 Job。
- 如果匹配失败，`when: never` 跳过 Job。

测试匹配规则



为了确认正则表达式是否正常工作，可以使用简单的 Shell 脚本测试：

测试脚本

```
WHITELIST="white-1,white-2,apple"
CI_PROJECT_NAMESPACE="white-1"

if [[ $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$) ]]; then
    echo "Namespace is in the whitelist."
else
    echo "Namespace is NOT in the whitelist."
fi
```

测试结果

1. ``CI_PROJECT_NAMESPACE="white-1"``:
 - 输出：``Namespace is in the whitelist.``
2. ``CI_PROJECT_NAMESPACE="white"``:
 - 输出：``Namespace is NOT in the whitelist.``

总结

这种方法使用环境变量存储白名单，通过正则表达式精确匹配 ``CI_PROJECT_NAMESPACE``，实现灵活、可维护的 Job 白名单控制。



`(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$) ??`



``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)`` 是一个正则表达式，用于精确匹配白名单中的项目命名空间 (``CI_PROJECT_NAMESPACE``)，确保匹配时不会误匹配到部分字符串。下面详细解释这个正则表达式的每一部分及其作用。

正则表达式结构拆解

```
(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)
```

1. ``(?:^|,)``

- 作用：匹配字符串的开头 ``^`` 或逗号 ``,``。
- 解析：
 - ``^``：匹配字符串的开始位置。
 - ``,``：匹配白名单中逗号分隔的值。
 - ``(?:...)``：非捕获组，表示将这一部分作为逻辑分组但不捕获到匹配结果中。
- 意义：确保 ``CI_PROJECT_NAMESPACE`` 是白名单的第一个值或紧跟在一个逗号之后。



2. ``$CI_PROJECT_NAMESPACE``

- 作用：动态插入项目的命名空间（例如 ``white-1``、``apple``）。
- 意义：这个值将从 GitLab 的环境变量中获取，表示当前运行的项目命名空间。

3. ``(?:,|$)``

- 作用：匹配字符串的结尾 ``$`` 或逗号 ``,``。
 - 解析：
 - ``,``：匹配白名单值之后的逗号。
 - ``$``：匹配字符串的结尾。
 - ``(?:,|$)``：非捕获组，和上面一样，用于逻辑分组而不捕获。
 - 意义：确保 ``$CI_PROJECT_NAMESPACE`` 是白名单的最后一个值或其后紧跟逗号分隔的值。
-

正则表达式整体匹配逻辑

- 匹配的字符串模式为：
 - ``$CI_PROJECT_NAMESPACE`` 是白名单中的完整值，且：
 - 是白名单的第一个值。
 - 或者是白名单中以逗号分隔的值。
 - 或者是白名单的最后一个值。
-

举例分析

假设 ``WHITELIST="white-1,white-2,apple"``

情况 1：``CI_PROJECT_NAMESPACE="white-1"``

- 字符串：``"white-1,white-2,apple"``
 - 匹配流程：
 - ``(?:^|,)``：匹配开头（``^``）。
 - ``white-1``：匹配当前项目命名空间。
 - ``(?:,|$)``：匹配后续的逗号（``,``）。
 - 结果：匹配成功。
-

情况 2：``CI_PROJECT_NAMESPACE="white"``

- 字符串：``"white-1,white-2,apple"``
- 匹配流程：



- ``(?:^|,)``: 匹配开头 (``^``)。
- ``white``: 不完整匹配, 失败。
- 结果: 匹配失败。

情况 3: ``CI_PROJECT_NAMESPACE="apple"``

- 字符串: ``"white-1,white-2,apple"``
- 匹配流程:
 - ``(?:^|,)``: 匹配逗号 (```,``)。
 - ``apple``: 匹配当前项目命名空间。
 - ``(?:,|$)``: 匹配字符串结尾 (``$``)。
- 结果: 匹配成功。

部分匹配问题的解决

使用这个正则表达式的好处是, 它避免了误匹配, 例如:

错误匹配示例 (如果不使用 ``(?:^|,)`` 和 ``(?:,|$)``):

- 白名单: ``white-1,white-2,apple``
- 项目命名空间: ``white``
- 如果仅使用 ``WHITELIST =~ $CI_PROJECT_NAMESPACE``, 正则可能会错误匹配到 ``white-1`` 的前半部分 ``white``。

通过 ``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)``, 确保只有完整值匹配时才会触发 Job。

总结

正则表达式 ``(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)`` 的主要特点是:

1. 确保精确匹配白名单中的项目命名空间。
2. 避免部分匹配的问题。
3. 支持白名单的多值逗号分隔格式。
4. 灵活适用于 GitLab CI/CD 的动态环境变量。

这样就能有效地控制哪些命名空间的项目可以触发 Job。



dockercompose的部屬pipeline可用上述方法嗎



在使用 Docker Compose 的部署 Pipeline 中，也可以用上述方法结合 GitLab CI/CD 的 `rules` 和环境变量来控制特定命名空间是否可以运行 Pipeline。以下是详细实现方式和注意事项：

目标

控制 Pipeline 的执行，使得只有在特定 `CI_PROJECT_NAMESPACE`（白名单）下，Docker Compose 的部署 Job 才能执行。

示例 `.gitlab-ci.yml` 配置

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1, namespace-2, namespace-3"

build_job:
  stage: build
  script:
    - echo "Building Docker images..."
    - docker-compose build

deploy_job:
  stage: deploy
  script:
    - echo "Deploying using Docker Compose..."
    - docker-compose up -d
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```

配置说明

1. `variables` 部分

- 定义 `WHITELIST` 变量，存储允许的命名空间列表（逗号分隔）。
- 通过环境变量灵活配置白名单内容，便于后续维护。

2. `build_job`

- 不受白名单限制，负责构建 Docker 镜像。
- 使用 `docker-compose build` 构建所需服务的镜像。



3. `deploy_job`

- 部署时，检查 `CI_PROJECT_NAMESPACE` 是否在 `WHITELIST` 中：
 - `\$WHITELIST =~ (?:^|,)\$CI_PROJECT_NAMESPACE(?:,|\$)`：精确匹配。
 - 如果匹配成功，部署 Job 执行。
 - 如果匹配失败，跳过部署 Job。

4. `rules`

- 控制 Job 的执行条件：
 - 如果条件满足（命名空间在白名单中），Job 设置为 `when: on_success`，正常执行。
 - 如果条件不满足，Job 设置为 `when: never`，跳过执行。

测试示例

情况 1：命名空间在白名单中

- 配置：
 - `CI_PROJECT_NAMESPACE="namespace-1"`
 - `WHITELIST="namespace-1,namespace-2,namespace-3"`
- 执行流程：
 - `build_job` 正常执行。
 - `deploy_job` 匹配白名单，执行成功。

情况 2：命名空间不在白名单中

- 配置：
 - `CI_PROJECT_NAMESPACE="other-namespace"`
 - `WHITELIST="namespace-1,namespace-2,namespace-3"`
- 执行流程：
 - `build_job` 正常执行。
 - `deploy_job` 不匹配白名单，跳过执行。

情况 3：白名单为空

- 配置：
 - `CI_PROJECT_NAMESPACE="namespace-1"`
 - `WHITELIST=""`（未配置或为空）。
- 执行流程：
 - `build_job` 正常执行。
 - `deploy_job` 规则不匹配，跳过执行。



在 Docker Compose 部署中的优势

1. 白名单管理：

- 可以灵活地调整白名单，避免在不允许的命名空间中执行部署操作。

2. 避免误操作：

- 防止误将部署 Pipeline 执行到不应该的环境中（例如测试环境或非生产命名空间）。

3. 动态扩展：

- 如果白名单需要动态更新，可以通过 GitLab CI/CD 的 `Settings > CI/CD > Variables` 页面配置 `WHITELIST`，无须更改 `.gitlab-ci.yml` 文件。

注意事项

1. GitLab Runner 权限：

- 确保 GitLab Runner 有足够的权限运行 `docker-compose` 命令。
- 如果使用 Docker-in-Docker (DinD)，需要正确配置 `services`。

2. 环境隔离：

- 在白名单中执行 `docker-compose` 时，确保正确配置 `.env` 文件或其他环境变量，以便将服务部署到正确的环境。

3. 错误处理：

- 在 `deploy_job` 中，可以添加更多脚本以处理部署失败的情况，例如日志收集或通知。

完整 `.gitlab-ci.yml` 示例（含 Docker-in-Docker）

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1, namespace-2, namespace-3"
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_TLS_CERTDIR: ""

services:
  - docker:dind

build_job:
  stage: build
  script:
```



```
- docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
- echo "Building Docker images..."
- docker-compose build

deploy_job:
  stage: deploy
  script:
    - echo "Deploying using Docker Compose..."
    - docker-compose up -d
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ (?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)'
      when: on_success
    - when: never
```

以上配置结合 Docker Compose 和白名单控制，可以安全地实现命名空间限定的部署流水线。



出現yaml語法錯誤



如果 ``.gitlab-ci.yml`` 出现语法错误，可能是以下原因之一：

1. **缩进问题**：YAML 文件对缩进非常敏感，错误的缩进会导致解析失败。
2. **特殊字符转义**：某些特殊字符（如 ``$`` 和 ``&``）需要正确处理。
3. **规则格式错误**：``rules`` 的条件表达式可能写得不够规范。
4. **Docker Compose 配置问题**：在 CI 中使用 ``docker-compose`` 时，环境变量设置不正确。

以下是调整后的 ``.gitlab-ci.yml`` 文件，确保语法正确：

修正后的 ``.gitlab-ci.yml``

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1, namespace-2, namespace-3"
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_TLS_CERTDIR: ""

services:
```



```
- docker:dind

build_job:
  stage: build
  script:
    - echo "Logging into Docker registry..."
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" "$CI_REGISTRY"
    - echo "Building Docker images..."
    - docker-compose build

deploy_job:
  stage: deploy
  script:
    - echo "Deploying using Docker Compose..."
    - docker-compose up -d
  rules:
    - if: '$CI_PROJECT_NAMESPACE && $WHITELIST =~ "(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)"'
      when: on_success
    - when: never
```

语法检查与改进

1. 环境变量引用：

- ``$CI_PROJECT_NAMESPACE`` 和 ``$WHITELIST`` 在 ``if`` 条件中使用时，引用的格式必须正确。
- 使用双引号 ``"`` 包裹正则表达式，以防止 YAML 解析错误。

2. 缩进规范：

- YAML 文件对缩进非常敏感，必须使用两个空格的缩进。
- 确保 ``stages``、``variables``、``services``、``build_job``、``deploy_job`` 等都在正确的层级。

3. 正则表达式处理：

- 使用 ``"...`` 包裹正则表达式，避免 ``$`` 被误认为是 YAML 的锚点。
- 正则表达式部分 ``"(?:^|,)$CI_PROJECT_NAMESPACE(?:,|$)"`` 在 ``rules`` 中引用时需要正确转义。

4. Docker 服务配置：

- ``services`` 部分确保配置了 ``docker:dind``，并设置 ``DOCKER_HOST`` 和 ``DOCKER_TLS_CERTDIR`` 变量。

验证 Pipeline

为了确保修改后的文件没有语法错误，可以：



1. 使用 GitLab 提供的 **CI Lint 工具**：

- 在项目中打开 ``CI/CD > Pipelines > Editor``，将配置粘贴进去，点击 **Validate** 检查语法。

2. 使用本地 YAML 验证工具：

- 例如，安装 ``yamllint``：

```
pip install yamllint
yamllint .gitlab-ci.yml
```

3. 提交到 GitLab：

- 提交修改后的 ``gitlab-ci.yml`` 文件到仓库，观察 Pipeline 是否正确触发。

常见问题排查

- **`rules` 语法错误**：如果规则中引用的环境变量不正确，会报错。确保 ``$CI_PROJECT_NAMESPACE`` 和 ``$WHITELIST`` 在运行时都已定义。
- **`docker-compose` 错误**：如果 ``docker-compose`` 命令失败，检查：
 - Docker 服务是否运行。
 - ``docker-compose.yml`` 文件是否正确。
- **语法错误仍存在**：可以将问题所在的 ``gitlab-ci.yml`` 文件粘贴在聊天中，我可以帮助你进一步修正。



我的deploy_job是被另一个repo(user) trigger的，我要以那个repo的project name space来查看是否符合我的白名单，而不是用deploy_job的project name space



在 GitLab CI/CD 中，如果你的 ``deploy_job`` 是被另一个仓库 (``user`` repo) 触发的，那么你需要使用触发器相关的变量来获取触发仓库的命名空间，而不是当前 Job 的 ``CI_PROJECT_NAMESPACE``。GitLab 提供了以下变量来处理这种情况：

1. ``CI_PIPELINE_SOURCE``：

- 表示当前 Pipeline 的触发来源。
- 值可以是 ``pipeline``，``web``，``trigger`` 等。

2. ``CI_JOB_TRIGGERED_BY_PROJECT_PATH``：



- 当 Pipeline 被触发器启动时，这个变量包含触发 Pipeline 的项目路径（`namespace/project` 格式）。

我们可以通过解析 `CI_JOB_TRIGGERED_BY_PROJECT_PATH` 来提取触发项目的命名空间，然后和白名单进行匹配。

修正后的 `.gitlab-ci.yml`

以下是根据触发器的命名空间检查白名单的配置：

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1, namespace-2, namespace-3"
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_TLS_CERTDIR: ""

services:
  - docker:dind

build_job:
  stage: build
  script:
    - echo "Building Docker images..."
    - docker-compose build

deploy_job:
  stage: deploy
  script:
    - |
      echo "Triggered by project: $CI_JOB_TRIGGERED_BY_PROJECT_PATH"
      echo "Deploying using Docker Compose..."
      docker-compose up -d
  rules:
    - if: '$CI_PIPELINE_SOURCE == "trigger" && $CI_JOB_TRIGGERED_BY_PROJECT_PATH =~ "^($WHITELIST)/"'
      when: on_success
    - when: never
```

配置解析

1. `CI_JOB_TRIGGERED_BY_PROJECT_PATH`

- 这个变量是触发 Job 的项目路径，例如 `namespace-1/project-A`。
- 如果这个变量为空，则表示当前 Pipeline 不是通过触发器启动的。

2. 提取触发项目的命名空间

- 使用正则表达式 `^(\$WHITELIST)/` 检查项目路径中的命名空间部分。



- ``$WHITELIST`` 是白名单，包含命名空间列表（例如 ``namespace-1,namespace-2,namespace-3``）。
- ``^($WHITELIST)/`` 确保只匹配路径的开头部分，且完全匹配命名空间。

3. ``rules`` 中的条件

- 条件 1：``$CI_PIPELINE_SOURCE == "trigger"``
 - 确保当前 Pipeline 是由触发器启动的。
- 条件 2：``$CI_JOB_TRIGGERED_BY_PROJECT_PATH =~ "^($WHITELIST)/"``
 - 确保触发的项目命名空间在白名单中。

4. ``when: never``

- 如果不满足上述条件，Job 将被跳过。

测试流程

情况 1：触发项目在白名单中

- ``CI_JOB_TRIGGERED_BY_PROJECT_PATH="namespace-1/project-A"``
- ``WHITELIST="namespace-1,namespace-2,namespace-3"``
- 匹配成功，``deploy_job`` 执行。

情况 2：触发项目不在白名单中

- ``CI_JOB_TRIGGERED_BY_PROJECT_PATH="other-namespace/project-B"``
- ``WHITELIST="namespace-1,namespace-2,namespace-3"``
- 匹配失败，``deploy_job`` 跳过。

情况 3：未通过触发器启动

- ``CI_PIPELINE_SOURCE="push"``
- ``CI_JOB_TRIGGERED_BY_PROJECT_PATH`` 未定义。
- 匹配失败，``deploy_job`` 跳过。

注意事项

1. 触发器变量的可用性：

- 确保你的触发器配置能正确传递 ``CI_JOB_TRIGGERED_BY_PROJECT_PATH``。

2. 环境变量调试：

- 在调试时，可以在 Job 的 ``script`` 中输出关键变量值：

```
script:
  - echo "Trigger source: $CI_PIPELINE_SOURCE"
```





```
- echo "Triggered by: $CI_JOB_TRIGGERED_BY_PROJECT_PATH"
```

3. 白名单的维护：

- 如果白名单较长，可以通过 GitLab 的 CI/CD Variables 管理 `WHITELIST`，避免在文件中硬编码。

扩展

如果需要从触发器的完整路径中提取更多信息，可以使用 Shell 脚本配合 `sed` 或 `awk` 提取命名空间部分，例如：

```
triggered_namespace=$(echo "$CI_JOB_TRIGGERED_BY_PROJECT_PATH" | awk -F '/' '{print $1}')
if [[ $WHITELIST =~ (^|,)$triggered_namespace(,|$) ]]; then
    echo "Namespace is whitelisted"
else
    echo "Namespace is NOT whitelisted"
    exit 1
fi
```



確定有CI_JOB_TRIGGERED_BY_PROJECT_PATH可以用嗎，gitlab官網沒有這個變數



确实，GitLab 官方文档中并未提到 `CI_JOB_TRIGGERED_BY_PROJECT_PATH` 这样的内置变量。我之前的建议存在误导性，对此深表歉意！

如果你需要获取触发 Pipeline 的命名空间或项目信息，实际可以通过触发器的 **变量传递** 或 **API 获取** 的方式实现。这是更可靠的方式。

解决方案：通过触发器传递变量

1. 设置触发器

在触发 Pipeline 的仓库中，配置一个触发器，并手动传递需要的命名空间信息。

触发代码示例（`user` repo）

```
trigger:
  stage: trigger
  script:
    - curl -X POST -F token=$TRIGGER_TOKEN -F ref=main \
      -F variables[TRIGGER_NAMESPACE]=$CI_PROJECT_NAMESPACE \
      https://gitlab.com/api/v4/projects/<PROJECT_ID>/trigger/pipeline
```

- `variables[TRIGGER_NAMESPACE]`：传递当前触发项目的命名空间作为变量。





- `<PROJECT_ID>`: 替换为目标仓库的 GitLab 项目 ID。

2. 接收触发变量

在目标 Pipeline 中，接收并使用传递的变量。

目标仓库的 `.gitlab-ci.yml`

```
stages:
  - build
  - deploy

variables:
  WHITELIST: "namespace-1,namespace-2,namespace-3"

deploy_job:
  stage: deploy
  script:
    - echo "Triggered by namespace: $TRIGGER_NAMESPACE"
    - if [[ $WHITELIST =~ (^|,)$TRIGGER_NAMESPACE(,|$) ]]; then
        echo "Namespace is whitelisted. Proceeding with deployment.";
        docker-compose up -d;
      else
        echo "Namespace is not whitelisted. Skipping deployment.";
        exit 1;
      fi
  rules:
    - if: '$TRIGGER_NAMESPACE'
      when: on_success
    - when: never
```

解决方案：使用 GitLab API 获取触发者信息

如果不能通过变量传递，可以在目标仓库的 Pipeline 中通过 GitLab API 查询触发 Pipeline 的来源。

示例脚本

```
deploy_job:
  stage: deploy
  script:
    - echo "Fetching trigger information from GitLab API..."
    - trigger_project=$(curl --header "PRIVATE-TOKEN: $GITLAB_API_TOKEN" \
      "https://gitlab.com/api/v4/projects/$CI_PROJECT_ID/pipelines/$CI_PIPELINE_ID" | jq -r '.source_project_id')
    - trigger_namespace=$(curl --header "PRIVATE-TOKEN: $GITLAB_API_TOKEN" \
      "https://gitlab.com/api/v4/projects/$trigger_project" | jq -r '.namespace.full_path')
    - echo "Triggered by namespace: $trigger_namespace"
    - if [[ $WHITELIST =~ (^|,)$trigger_namespace(,|$) ]]; then
        echo "Namespace is whitelisted. Proceeding with deployment.";
        docker-compose up -d;
      else
        echo "Namespace is not whitelisted. Skipping deployment.";
        exit 1;
      fi
```




```
    echo "Namespace is not whitelisted. Skipping deployment.";
    exit 1;
  fi
rules:
  - when: always
```

注意事项

1. GitLab API 权限：

- 确保 `GITLAB_API_TOKEN` 拥有访问触发项目和 Pipeline 的权限。

2. 安全性：

- 触发器传递的变量或 API 请求可能包含敏感信息，建议使用 GitLab 的 CI/CD 变量加密存储 `TRIGGER_TOKEN` 和 `GITLAB_API_TOKEN`。

3. 白名单验证：

- 始终验证命名空间是否在白名单中，以确保部署的安全性。

如果使用第一种（变量传递）方案更简单直接，建议优先采用。如果 API 获取方式更适合你的需求，也可以参考示例。需要进一步调整的话，可以随时沟通！



如果trigger job沒有CI_PROJECT_NAMESPACE可用，但有提供CI_PROJECT_ID，那可以對trigger job的project name space作白名單控制嗎



如果 `trigger job` 没有提供 `CI_PROJECT_NAMESPACE`，但提供了 `CI_PROJECT_ID`，我们仍然可以通过 **GitLab API** 查询该项目的详细信息，从而获取其命名空间（`namespace`）。然后，结合白名单验证逻辑实现控制。

解决方案：通过 `CI_PROJECT_ID` 使用 GitLab API 获取命名空间

GitLab 提供了一个 API，可以通过项目 ID 获取项目的详细信息，包括命名空间：

API Endpoint:

```
GET https://gitlab.com/api/v4/projects/:id
```

关键字段



- ``namespace.full_path``：命名空间的完整路径。
- ``id``：项目 ID（对应 ``CI_PROJECT_ID``）。

示例配置

以下是一个完整的 `.gitlab-ci.yml`，使用 ``CI_PROJECT_ID`` 获取触发项目的命名空间并验证白名单。

```
stages:
  - deploy

variables:
  WHITELIST: "namespace-1, namespace-2, namespace-3"

deploy_job:
  stage: deploy
  script:
    - echo "Fetching project information for CI_PROJECT_ID=$CI_PROJECT_ID..."
    - trigger_namespace=$(curl --silent --header "PRIVATE-TOKEN: $GITLAB_API_TOKEN" \
      "https://gitlab.com/api/v4/projects/$CI_PROJECT_ID" | jq -r '.namespace.full_path')
    - echo "Triggered by namespace: $trigger_namespace"
    - |
      if [[ $WHITELIST =~ (^|,)$trigger_namespace(,|$) ]]; then
        echo "Namespace is whitelisted. Proceeding with deployment.";
        docker-compose up -d;
      else
        echo "Namespace is not whitelisted. Skipping deployment.";
        exit 1;
      fi
  rules:
    - when: always
```

配置说明

1. 获取命名空间

- 使用 GitLab API，通过 ``CI_PROJECT_ID`` 获取触发项目的 ``namespace.full_path``。
- ``PRIVATE-TOKEN`` 是 GitLab 的 API Token，需要事先在 GitLab 的 **CI/CD Variables** 中配置，例如 ``GITLAB_API_TOKEN``。

2. 验证白名单

- ``WHITELIST`` 定义了允许的命名空间列表，逗号分隔。
- 使用正则表达式验证 ``trigger_namespace`` 是否在白名单中。

3. 条件控制

- 如果命名空间匹配白名单，执行 ``docker-compose up -d`` 部署。
- 如果不匹配，直接 ``exit 1`` 跳过部署。



测试流程

示例变量

- ``CI_PROJECT_ID=123``
- ``WHITELIST="namespace-1,namespace-2,namespace-3"``

情况 1：项目在白名单中

- API 返回：

```
{
  "id": 123,
  "namespace": {
    "full_path": "namespace-1"
  }
}
```

- 匹配成功，执行部署。

情况 2：项目不在白名单中

- API 返回：

```
{
  "id": 123,
  "namespace": {
    "full_path": "other-namespace"
  }
}
```

- 匹配失败，跳过部署。

情况 3：API 调用失败

- 无法获取命名空间信息：
 - 检查 ``GITLAB_API_TOKEN`` 是否配置正确。
 - 确保 GitLab API 的访问权限。

注意事项

1. GitLab API 权限：

- 确保 ``GITLAB_API_TOKEN`` 拥有读取项目信息的权限（``read_api`` 权限即可）。

2. 命名空间验证的准确性：

- 使用正则表达式确保白名单匹配是严格的完整匹配，避免误匹配。

3. 脚本调试：



- 在开发和调试过程中，可以在 ``script`` 中添加调试信息，输出 ``trigger_namespace`` 和 API 返回值。

通过这种方法，可以灵活地基于 ``CI_PROJECT_ID`` 实现触发项目的命名空间白名单控制，确保部署操作的安全性和可控性。



如果trigger job沒有CI_PROJECT_NAMESPACE可用，但有提供CI_PROJECT_ID，那可以直接用trigger job的CI_PROJECT_ID作白名單控制嗎



如果 ``trigger job`` 提供了 ``CI_PROJECT_ID``，可以直接通过 ``CI_PROJECT_ID`` 实现白名单控制，而无需额外获取命名空间。这种方法简单高效，只需将允许的 ``CI_PROJECT_ID`` 列入白名单即可。

配置示例：基于 ``CI_PROJECT_ID`` 的白名单控制

以下是一个 ``.gitlab-ci.yml`` 配置，使用 ``CI_PROJECT_ID`` 直接控制是否允许执行 ``deploy_job``：

```
stages:
  - deploy

variables:
  WHITELIST: "123,456,789" # 允许的 CI_PROJECT_ID 列表

deploy_job:
  stage: deploy
  script:
    - echo "Triggered by project ID: $CI_PROJECT_ID"
    - |
      if [[ $WHITELIST =~ (^|,)$CI_PROJECT_ID(,|$) ]]; then
        echo "Project ID is whitelisted. Proceeding with deployment.";
        docker-compose up -d;
      else
        echo "Project ID is not whitelisted. Skipping deployment.";
        exit 1;
      fi
  rules:
    - when: always
```



配置解析

1. `WHITELIST` 变量

- 定义一个逗号分隔的 `CI_PROJECT_ID` 列表，例如 `123,456,789`。
- 这些 ID 对应允许触发的项目。

2. 脚本中验证白名单

- 使用正则表达式 `(^[|,|,\$]|,\$)` ：
 - 确保 `CI_PROJECT_ID` 是完整匹配白名单中的一个值。
 - 避免部分匹配，例如 `123` 不应匹配 `1234`。

3. 规则控制

- `rules` 设置为 `when: always`，表示始终运行 `deploy_job`，但在脚本中验证白名单控制是否真正执行。

4. 跳过或执行

- 如果匹配白名单，执行 `docker-compose up -d` 部署。
- 如果不匹配白名单，打印提示信息并退出 `exit 1`。

测试流程

情况 1：`CI_PROJECT_ID` 在白名单中

- `CI_PROJECT_ID=123`
- `WHITELIST="123,456,789"`
- 输出：

```
Triggered by project ID: 123
Project ID is whitelisted. Proceeding with deployment.
```

情况 2：`CI_PROJECT_ID` 不在白名单中

- `CI_PROJECT_ID=999`
- `WHITELIST="123,456,789"`
- 输出：

```
Triggered by project ID: 999
Project ID is not whitelisted. Skipping deployment.
```

情况 3：白名单为空

- `CI_PROJECT_ID=123`
- `WHITELIST=""`
- 输出：



```
Triggered by project ID: 123
Project ID is not whitelisted. Skipping deployment.
```

注意事项

1. 项目 ID 的来源

- ``CI_PROJECT_ID`` 是触发 Pipeline 项目的唯一标识，确保触发器配置正确。

2. 维护白名单

- 如果白名单较长，建议在 GitLab **CI/CD Variables** 中配置 ``WHITELIST``，避免直接硬编码在 ``.gitlab-ci.yml`` 中。

3. 安全性

- 此方法直接依赖于项目 ID 的白名单控制，简单高效，但需要确保白名单准确无误。

示例扩展：白名单存储在 GitLab CI/CD Variables

可以通过 GitLab 的变量管理来存储白名单，避免硬编码在 ``.gitlab-ci.yml`` 文件中。

配置变量

1. 在 GitLab 项目中，导航到 ``Settings > CI/CD > Variables``。
2. 添加一个变量：

- Key: ``WHITELIST``
- Value: ``123,456,789``

修改 ``.gitlab-ci.yml``

```
variables:
  WHITELIST: $WHITELIST

deploy_job:
  stage: deploy
  script:
    - echo "Triggered by project ID: $CI_PROJECT_ID"
    - |
      if [[ $WHITELIST =~ (^|,)$CI_PROJECT_ID(,|$) ]]; then
        echo "Project ID is whitelisted. Proceeding with deployment.";
        docker-compose up -d;
      else
        echo "Project ID is not whitelisted. Skipping deployment.";
        exit 1;
      fi
  rules:
    - when: always
```



这样，你可以直接基于 `CI_PROJECT_ID` 实现高效的白名单控制，无需额外调用 GitLab API 或查询命名空间。



deploy_job用rules以trigger job的CI_PROJECT_ID作白名单控制



如果需要通过 `rules` 使用 `trigger job` 的 `CI_PROJECT_ID` 进行白名单控制，可以直接在 `rules` 中编写条件表达式。以下是完整的 `.gitlab-ci.yml` 配置示例：

示例 `.gitlab-ci.yml`

```
stages:
  - deploy

variables:
  WHITELIST: "123,456,789" # 允许的 CI_PROJECT_ID 列表

deploy_job:
  stage: deploy
  script:
    - echo "Deploying triggered by project ID: $CI_PROJECT_ID"
    - docker-compose up -d
  rules:
    - if: '$WHITELIST =~ (^|,)$CI_PROJECT_ID(,|$)'
      when: on_success
    - when: never
```

配置解析

1. 变量 `WHITELIST`

- 定义一个逗号分隔的项目 ID 列表，列出允许触发的 `CI_PROJECT_ID`，例如：

```
WHITELIST: "123,456,789"
```

- 可通过 GitLab 的 CI/CD 变量界面配置，以动态管理。

2. `rules` 的条件控制

- 条件：

```
if: '$WHITELIST =~ (^|,)$CI_PROJECT_ID(,|$)'
```



- ``WHITELIST``：白名单变量。
- ``CI_PROJECT_ID``：触发项目的 ID。
- 正则表达式：
 - ``(^|,)``：匹配逗号或字符串开头，确保匹配从列表开始。
 - ``$CI_PROJECT_ID``：动态插入当前触发的项目 ID。
 - ``(,|$)``：匹配逗号或字符串结尾，确保匹配列表中的完整 ID。

3. 规则结果

- 如果 ``CI_PROJECT_ID`` 在白名单中：
 - 执行 ``deploy_job``。
- 如果不在白名单中：
 - 跳过 ``deploy_job``。

4. ``script`` 部分

- ``script`` 中可以输出调试信息，确认 ``CI_PROJECT_ID``：

```
script:
- echo "Triggered by CI_PROJECT_ID=$CI_PROJECT_ID"
- docker-compose up -d
```

测试流程

示例变量

- ``CI_PROJECT_ID=123``
- ``WHITELIST="123,456,789"``

情况 1：``CI_PROJECT_ID`` 在白名单中

- 条件匹配：

```
$WHITELIST =~ (^|,)123(,|$)
```

- ``deploy_job`` 执行。

情况 2：``CI_PROJECT_ID`` 不在白名单中

- 示例：

```
CI_PROJECT_ID=999
WHITELIST="123,456,789"
```

- 条件不匹配：

```
$WHITELIST =~ (^|,)999(,|$) # 不匹配
```





- ``deploy_job`` 跳过。

扩展：通过 GitLab CI/CD Variables 动态管理白名单

在 GitLab 中配置

1. 前往 ``Settings > CI/CD > Variables``。
2. 添加变量：

- Key: ``WHITELIST``
- Value: ``123,456,789``

修改 ``.gitlab-ci.yml``

```
variables:
  WHITELIST: $WHITELIST

deploy_job:
  stage: deploy
  script:
    - echo "Deploying triggered by project ID: $CI_PROJECT_ID"
    - docker-compose up -d
  rules:
    - if: '$WHITELIST =~ (^|,)$CI_PROJECT_ID(,|$)'
```

注意事项

1. 正则表达式的完整匹配：
 - 确保正则表达式只匹配完整的 ``CI_PROJECT_ID``，避免部分匹配导致误执行。
2. 白名单的维护：
 - 如果项目 ID 发生变化，及时更新 ``WHITELIST``。
3. 规则的执行优先级：
 - ``rules`` 的匹配顺序至关重要，第一个匹配成功的规则会决定 Job 的状态。

通过这种方式，基于 ``trigger job`` 的 ``CI_PROJECT_ID`` 实现白名单控制可以高效、安全地限制部署权限。