自动化构建python项目镜像

概述

目的

1. 规范代码交付标准——统一以Docker镜像作为交付物
2. 构建镜像自动化

需要步骤

1. 准备生成项目镜像需要的dockerfile模板
2. 配合gitlab代码仓库，为项目注册runner
3. 项目镜像会随着项目代码的不断修改变化而进行更新和升级，设计能够实现自动化构建项目镜像，配合gitlab-runner使用的pipeline模板

工具使用

类别	工具
代码仓库	gitlab
CI/CD	gitlab-runner
运行环境	Docker

项目镜像设计

使用Dockerfile文件定制镜像

> Dockerfile 是一个文本文件，其内包含了一条条的 指令，每一条指令构建一层所添加的配置，运行的命令等，因此每一条指令的内容，就是描述该层应当如何构建。

项目镜像——Dockerfile设计模板

```
# FROM 用来指定基础镜像，FROM 是必备的指令，并且必须是第一条指令
# 基础镜像可以选择在Docker Hub上已有的官方镜像（https://hub.docker.com/search），也可以根据我们的实际情况自定义基础镜像
FROM ZKR_IMAGE_NAME:v1

# 进入容器内/usr/src/app目录下，作为我们的工作目录，WORKDIR可以理解为我们常用的cd命令
WORKDIR /usr/src/app

# 将宿主机上项目代码复制到容器内当前地址
COPY . .

# 下载我们需要的安装包，可以指定地址下载
RUN pip install  -i https://pypi.tuna.tsinghua.edu.cn/simple --no-cache-dir -r requirements.txt

# 如果使用其他途径引进的包，可以参考目录附加中基础镜像dockerfile设计详情
# COPY PACKAGES /usr/local/lib/python3.7/dist-packages/

# 设定运行镜像时的默认操作
CMD [ "python", "./your-daemon-or-script.py" ]
```

注册gitlab-runner

环境准备

1. 搭建gitlab服务
2. 搭建gitlab-runner服务

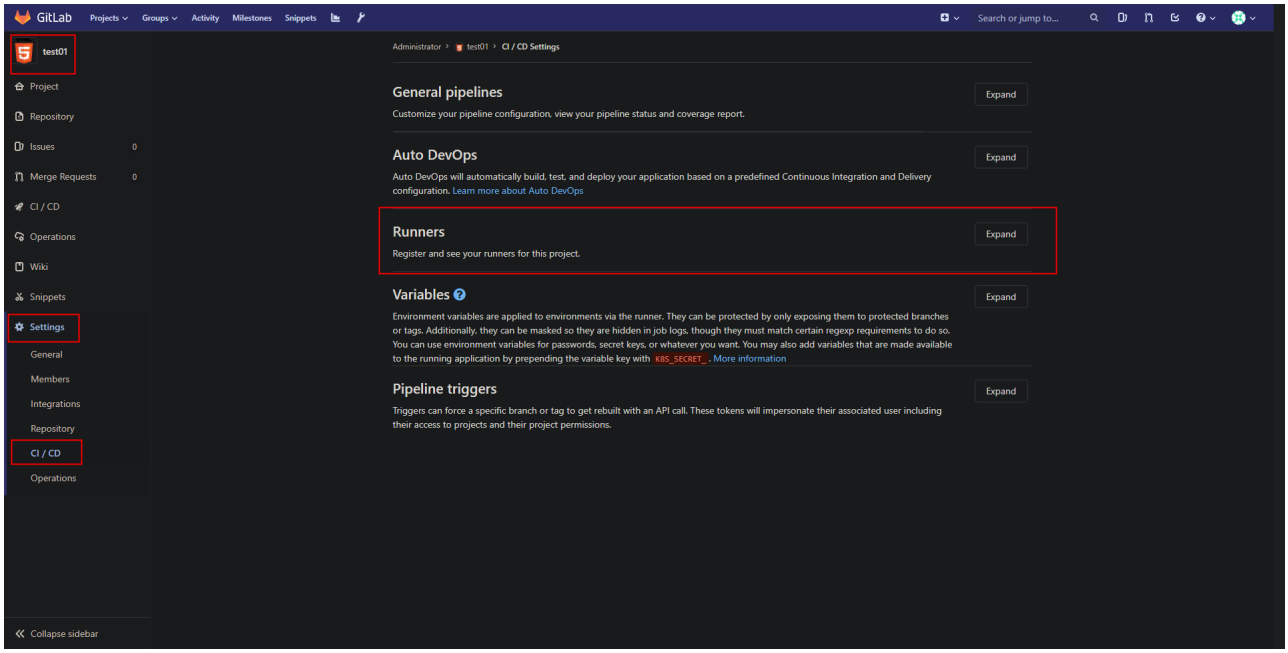
注册runner具体步骤

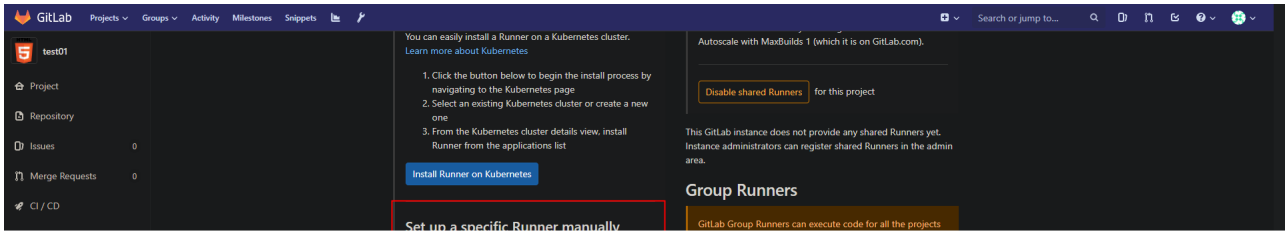
获取**url**和**token**

- gitlab页面操作

1. 在gitlab服务上创建项目

2. 在项目settings找到CICD的runners扩展，获取url和token





注册**runner**

- gitlab-runner shell操作

1. 为项目注册runner，在gitlab-runner所在的服务器上操作

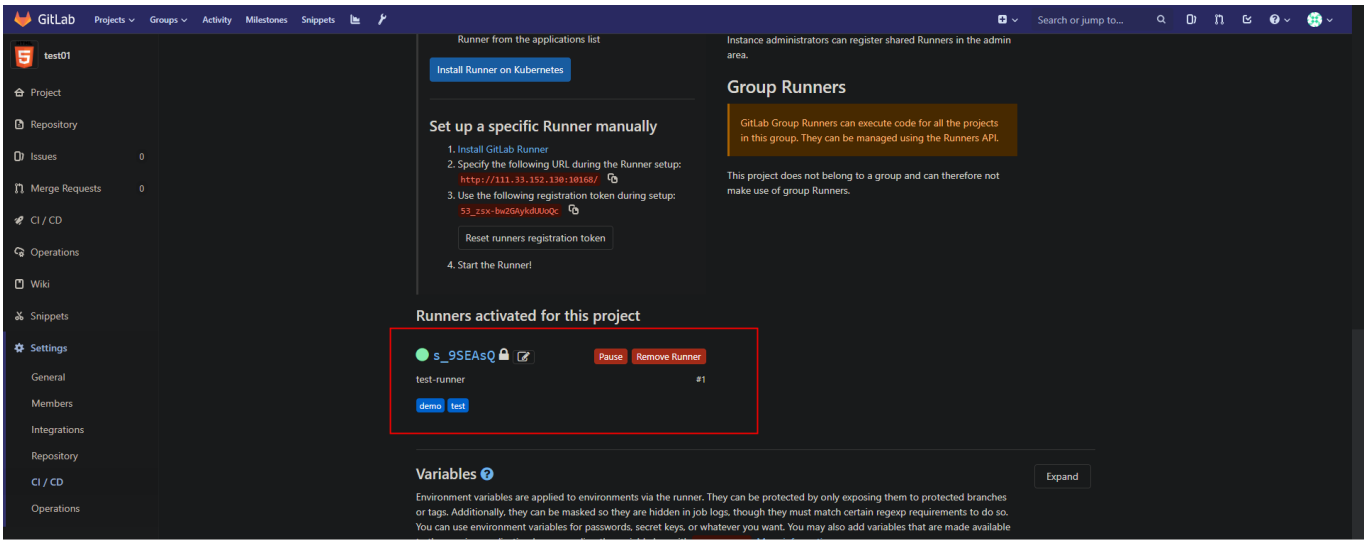
```
root@9a7d03e2fefd:/# gitlab-ci-multi-runner register # 注册命令
Runtime platform                                    arch=amd64 os=linux pid=34
revision=de7731dd version=12.1.0
Running in system-mode.

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
http://111.33.152.130:10168/ # 填写在gitlab获取的url
Please enter the gitlab-ci token for this runner:
53_zsx-bw2GAykdUUoQc # 写在gitlab获取的token
Please enter the gitlab-ci description for this runner:
```

```
[9a7d03e2fefd]: test test-project runner  # 描述这个runner
Please enter the gitlab-ci tags for this runner (comma separated):
demo,test  # 可以触发构建的tag
Registering runner... succeeded                     runner=n5AvLiFm
Please enter the executor: shell, docker-ssh+machine, kubernetes, docker+machine,
custom, docker, docker-ssh, parallels, ssh, virtualbox:
docker # 好像是执行器之类啥的
Please enter the default Docker image (e.g. ruby:2.6):
docker:18.09.0 # 我就随便写了一下，，，，
Runner registered successfully. Feel free to start it, but if it's running already
the config should be automatically reloaded!
root@9a7d03e2fefd:/# gitlab-ci-multi-runner list # 查看构建成功的runner列表
Runtime platform                                    arch=amd64 os=linux pid=45
revision=de7731dd version=12.1.0
Listing configured runners                          ConfigFile=/etc/gitlab-
runner/config.toml
test test-project runner                            Executor=docker Token=53_zsx-
bw2GAykdUUoQc URL=http://111.33.152.130:10168/
root@9a7d03e2fefd:/# gitlab-ci-multi-runner list
Runtime platform                                    arch=amd64 os=linux pid=53
revision=de7731dd version=12.1.0
Listing configured runners                          ConfigFile=/etc/gitlab-
runner/config.toml
test test-project runner                            Executor=docker
Token=w_AjCrJ8xbU7YG_9yWLK URL=http://111.33.152.130:10168/
```

查看已注册的**runner**

- gitlab页面操作

在gitlab项目settings中CICD的runners扩展页面，可以看到，增加了刚刚我们激活的runner



Pipeline模板设计

使用Pipeline triggers实现自动化构建镜像

我们使用gitlab-runner作为我们的CI/CD，gitlab-runner提供了一个很方便的配置工具，就是gitlab-ci.yml，编辑这个文件来设计流水线pipeline的具体步骤，将该文件放入到project的根目录下即可。gitlab-ci.yml会告诉gitlab-runner项目构建的整个流程。

在项目根目录添加.gitlab-ci.yml文件触发pipeline

- .gitlab-ci.yml模板

```
image: docker:stable # 使用docker环境进行自动化构建

services:
  - docker:dind

variables: # 可以在pipeline文件里指定环境变量，也可以在gitlabCI对应项目页面进行手动添加设置
  DOCKER_HOST: tcp://192.168.129.13:2376 # 设置docker运行环境
  DOCKER_DRIVER: overlay2
  CI_REGISTRY_USER: admin           # 私有仓库用户
  CI_REGISTRY_PASSWORD: asd1234     # 私有仓库密码
  CI_REGISTRY: 60.28.140.210:10167  # 私有仓库地址

# stages表示整个自动化构建的步骤，自定义设置
stages:
  - build       # 构建镜像
  - test        # 测试镜像
  - release     # 发布/上传镜像

build:
  stage: build
  script:
    - cp domain.crt /etc/ssl/certs/ca-certificates.crt                          # 上传私有仓库证书
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY  # 登录私有仓库
    - docker build --pull -t $CONTAINER_TEST_IMAGE . # 创建测试镜像，$CONTAINER_TEST_IMAGE作为环境变量可以提前在项目的环境里设置。
    - docker push $CONTAINER_TEST_IMAGE # 上传测试镜像

test1:
  stage: test
  script:
    - docker pull $CONTAINER_TEST_IMAGE
    - docker run $CONTAINER_TEST_IMAGE /script/to/run/tests
# 任务可以并行执行，这里的test可以测试多个例子，测试我们新建立的镜像是否可用
test2:
  stage: test
  script:
    - docker pull $CONTAINER_TEST_IMAGE
```

```
    - docker run $CONTAINER_TEST_IMAGE /script/to/run/another/test

release-image:
  stage: release  # 如果上面的测试镜像可用，则分支可以提交merge请求，告知master，
master会触发构建，发布镜像
  script:
    - docker pull $CONTAINER_TEST_IMAGE
    - docker tag $CONTAINER_TEST_IMAGE $CONTAINER_RELEASE_IMAGE
    - docker push $CONTAINER_RELEASE_IMAGE
  only:
    - master # only 规定了只有master提交时，才会触发此步骤
```

附加

基础镜像构建

- dockerfile

```
# FROM 用来指定基础镜像，FROM 是必备的指令，并且必须是第一条指令
# 此处我们指定的基础镜像是DockerHub官方仓库里的python镜像，如果本地没有获取，在构建镜像
过程中会自动下载到本地并使用
FROM python:3.7

# 进入容器（指构建镜像过程中，由python:3.7生成的虚拟容器）的/usr/src/app目录，作为我们
的工作目录，WORKDIR可以理解为我们常用的cd命令
WORKDIR /usr/src/app

# 将宿主机上项目代码复制到容器内
COPY . .

# 升级容器内的pip，RUN表示执行命令
RUN pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pip -U

# case 1 pip直接安装 -i
RUN pip install --no-cache-dir -r  -i https://pypi.tuna.tsinghua.edu.cn/simple
requirements.txt

# case 2 离线pip安装，需要提前将打包好的包whl放入项目目录内
RUN pip install --no-index --find-index=packages -r requirements.txt

# case 3 直接导入宿主机上现有package
COPY package /usr/local/lib/python3.7/dist-packages/

# 如果有shell命令要执行，如下
# 尽量写成一行，多行可以用\隔开
RUN ping www.baidu.com && \
    ifconfig && \
    ls
```

- 构建

构建基础镜像生成命令，在Dockerfile文件所在目录执行。

```
# build表示构建 （必选）
# -t IMAGE_NAME:v1 指定镜像的名字以及版本号（-t必选，名称自定义）
# . 表示使用根目录下的Dockerfile文件（必选）
[root@node project] docker build -t ZKR_IMAGE_NAME:1 .

Sending build context to Docker daemon  334.6MB
Step 1/5 : FROM python:3.7
 ---> d2300e0a0f3f
Step 2/5 : WORKDIR /usr/src/app
 ---> Using cache
 ---> 3dddec273adb
Step 3/5 : COPY . .
 ---> Using cache
 ---> f8ce0769a2d8
Step 4/5 : RUN pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pip -U
 ---> Using cache
 ---> 99da7fa03b98
Step 5/5 : RUN pip install --no-cache-dir -r  -i
https://pypi.tuna.tsinghua.edu.cn/simple requirements.txt
 ---> Using cache
 ---> fed0877df075
Successfully built 2ee9ecdd642e
Successfully tagged ZKR_IAMGE_NAME:V1
```

- 查看

查看构建完成的镜像

```
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
ZKR_IAMGE_NAME                       v1                  107a00fca49b        13
hours ago        126MB
```