H2 一、Docker介绍

H3 1.1 Docker解决痛点

> 1. 我本地运行没问题啊【环境不一致】。
> 2. 哪个哥们又写死循环了，怎么这么卡【在多用户的操作系统下，会相互影响】
> 3. 淘宝在双11的时候，用户量暴增【大量增加实体机，运维成本过高问题】
> 4. 学习一门技术，学习安装成本过高【关于安装软件成本过高】

H3 1.2 Docker的思想

> 1. 集装箱：
>
>   会将所有需要的内容放到不同的集装箱中，谁需要这些环境就直接拿到这个集装箱就可以了
> 2. 标准化：
>   - 运输的标准化：Docker有一个码头，所有上传的集装箱都放在了这个码头上，当谁需要某一个环境，就直接指派大海豚搬运这个集装箱就可以了
>   - 命令的标准化：Docker提供了一系列的命令，帮助我们去获取集装箱等等操作。
>   - 提供了REST的API：衍生出了很多的图形化界面，如Rancher
> 3. 隔离型：
>
>   Docker在运行集装箱内的内容时，会在Linux的内核中，单独的开辟一片空间，这片空间不会影响到其他程序。

> 1. 注册中心：超级码头，上面放的是集装箱
> 2. 镜像：集装箱
> 3. 容器：运行起来的镜像

H2 二、Docker的基本操作

H3 2.1 Ma'c 中安装Docker

```
# macOS 我们可以使用 Homebrew 来安装 Docker
brew cask install docker
# 在载入 Docker app 后，点击 Next，可能会询问你的 macOS 登陆密码，你输入即可。之后会弹出一个 Docker 运行的提示窗口，状态栏上也有有个小鲸鱼的图标
```

H3 2.2 Docker的中央仓库【注册中心】

> 1. Docker官方的中央仓库：这个仓库是镜像最全的，但是下载速度较慢
>
>   https://hub.docker.com/
> 2. 国内的网站：网易蜂巢，daoCloud...
>
>   https://c.163yun.com/hub#/home
>
>   http://hub.daocloud.io/ 【推荐使用】
> 3. 在公司内部会采用私服的方式拉去镜像【需要添加配置】

```
# 需要在/etc/docker/daemon.json
{
  "registry-mirrors":["https://registry.docker-cn.com"]
  "insecure-registries":["ip:port"]
}
# 重启两个服务
systemctl daemon-reload
systemctl restart docker
```

H3 2.3 镜像的操作

```
# 1、拉取镜像到本地
docker pull 镜像名称[:tag]       # 如果不写tag版本号会拉去默认版本
# example
docker pull daocloud.io/library/mysql:5.7.4
```

```
# 2、查看本地所有镜像
docker images
```

```
# 3、删除本地镜像
docker rmi 镜像的唯一标识【IMAGE ID】
```

```
# 4、镜像的导入导出（不规范）
# 将本地的镜像导出
docker save -o 导出的路径 镜像id
# 加载本地的镜像文件
docker load -i 镜像文件
# 修改镜像名称
docker tag 镜像id 新镜像名称:版本
```

H3 2.4 容器的操作

```
# 1、运行容器
# 简单操作，采用“镜像名称[:tag]”时，如果有则运行，如果没有则先下载在运行
docker run 镜像标识|镜像名称[:tag]
# 常用的参数
docker run -d -p 宿主机端口:容器端口 --name 容器名称 镜像标识|镜像名称[:tag]
# -d：代表后台运行容器
# -p：宿主机端口:容器端口：为了映射当前Linux的端口和容器的端口
# --name 容器名称：指定容器的名称
```

```
# 2、查看正在运行的容器
docker ps [-qa]
# -a：查看全部的容器，包括没有运行
# -q：只查看容器的标识不查看其他信息
```

```
# 3、查看容器的日志
docker logs -f 容器id
# -f：可以滚动查看日志的最后几行
```

```
# 4、进入到容器内部
docker exec -it 容器id bash
```

```
# 5、删除容器（删除容器前，需要先停止容器）
# 停止制动的容器
docker stop 容器id
# 停止全部的容器
docker stop $(docker ps -qa)
# 删除指定容器
docker rm 容器id
# 删除全部容器
docker rm $(docker ps -qa)
```

```
# 6、启动容器
docker start 容器id
```

```
# 7、容器与宿主机内容互拷
# 从容器里面拷文件到宿主机
docker cp 容器名：要拷贝的文件在容器里面的路径 要拷贝到宿主机的相应路径
# 从宿主机拷文件到容器里面
docker cp 要拷贝的文件路径 容器名：要拷贝到容器里面对应的路径
```

```
# 8、查看Docker的底层信心
# docker inspect 会返回一个 JSON 文件记录着 Docker 容器的配置和状态信息
# ID/NAMES：容器ID/容器名称
# 查看容器所有状态信息
docker inspect NAMES
# 查看 容器ip 地址
docker inspect --format='{{.NetworkSettings.IPAddress}}' ID/NAMES
# 容器运行状态
docker inspect --format '{{.Name}} {{.State.Running}}' NAMES
# 查看进程信息
docker top NAMES
# 查看端口；(使用容器ID 或者 容器名称)
docker port ID/NAMES
# 查看IP地址 也可以直接通过用 远程执行命令也可以（Centos7）
docker exec -it ID/NAMES ip addr
```

```
# 9、查看宿主机与镜像映射端口
docker port 容器id
```

三、Docker的应用

3.1 准备SSM工程

> MySQL数据库的连接用户名和密码改变了，修改db.properties
>
> ssm工程中的jdbc连接url中ip写为宿主机ip

3.2 准备MySQL容器

```
# 运行MySQL容器
docker run -d -p 3306:3306 --name mysql -e MYSQL_ROOT_PASSWORD=root
daocloud.io/library/mysql:5.7.4
```

3.3 准备Tomcat容器

```
# 运行Tomcat容器，前面已经搞定，只需要将SSM项目的war包部署到Tomcat内部即可
# 可以通过命令将宿主机的内容复制到容器内部
docker cp 文件名 容器id:容器内部路径
# example
docker cp ssm.war fe:/user/local/tomcat/webapps
```

H3 3.4 数据卷

> 为了部署SSM的工程，需要使用到cp的命令将宿主机内的ssm.war文件复制到容器内部【由于新下载的容器没有ssm.war】但是宿主机的ssm.war配置文件改后还需要重新上传容器，或者是在容器中修改，太麻烦
>
> 可以用数据卷解决这个问题
>
> 数据卷：将宿主机的一个目录映射到容器的一个目录中
>
> 可以在宿主机中操作目录中的内容，那么容器内部映射的文件，也会跟着一起改变

```
# 1. 创建数据卷
docker volume create 数据卷名称
# 创建数据卷之后，默认会存放在一个目录下 /var/lib/docker/volumes/数据卷名称/_data
```

```
# 2. 查看数据卷的详细信息
docker volume inspect 数据卷名称
```

```
# 3. 查看全部数据卷
docker volume ls
```

```
# 4. 删除数据卷
docker volume rm 数据卷名称
```

```
# 5. 应用数据卷
# 当你映射数据卷时，如果数据卷不存在，Docker会帮你自动创建,会将容器内部自带的文件,存储在默认的存放路径中
docker run -v 数据卷名称:容器内部的路径 镜像id
# 直接指定一个路径作为数据卷的存放位置【推荐】
docker run -v 路径:容器内部的路径 镜像id
```

四、Docker自定义镜像

> 中央仓库上的镜像，也是Docker的用户自己上传过去的。

4.1 Dockerfile文件说明

> Dockerfile文件中常用的内容
>
> - from: 用于指定其后构建新镜像所使用的基础镜像。FROM 指令必是 Dockerfile 文件中的首条命令，启动构建流程后，Docker 将会基于该镜像构建新镜像，FROM 后的命令也会基于这个基础镜像。
> - copy: 将相对路径下的内容复制到自定义镜像中
> - workdir: 用于在容器内设置一个工作目录。
> - add: 更高级的复制命令，将原路径文件赋值到目标路径中
> - run: 在容器中执行Shell命令，requirments.txt中反正索要下载以来的名称
> - expose: 为构建的镜像设置监听端口，使容器在运行时监听
> - env: 设置环境变量而已，无论是后面的其它指令，如 RUN，还是运行时的应用，都可以直接使用这里
> - 定义的环境变量。
> - cmd: 用于指定在容器启动时所要执行的命令（在workdir下执行的。cmd可以写多个，但是只以最后一个为准）

4.2 EXAMPLE-1（tomcat镜像+ssm工程）

> 自定义一个tomcat镜像，并且将ssm.war部署到tomcat中

```
# Dockerfile文件内容如下

from daocloud.io/library/tomcat:8.5.15-jre8
copy ssm.war /usr/local/tomcat/webapps
```

```
# 将准备好的Dockerfile和相应的文件拖拽到Linux操作系统中，通过Docker的命令制作镜像，最后点表示从当地按目录获取dockerfile等相关文件
docker build -t 镜像名称:[tag] .
# example
docker build -t ssm-tomcat:1.0.0 .

# 查看刚才制作的镜像
docker images
# 启动制作的镜像
docker run -d -p 8081:8080 --name custom-ssm-tomcat ssm-tomcat:1.0.0
```

EXAMPLE-2（python编写的web应用）

> 制作一个docker，其中部署一个用Python编写的Web应用

```
'''web应用代码如下app.py'''

from flask import Flask
import socket
import os

app = Flask(__name__)

@app.route('/')
def hello():
 html = "<h3>Hello {name}!</h3>" \
        "<b>Hostname:</b> {hostname}<br/>"
 return html.format(name=os.getenv("NAME", "world"),
hostname=socket.gethostname())

if __name__ == "__main__":
 app.run(host='0.0.0.0', port=80)
```

```
'''Dockerfile文件内容如下'''

# 使用官方提供的 Python 作为我们镜像的基础环境
FROM daocloud.io/library/python:3.5-slim

# 将工作目录切换为
/Users/ssh/Documents/private/project/github/programmer-learning-
notes/docker/code/example-2/test
WORKDIR /Users/ssh/Documents/private/project/github/programmer-
learning-notes/docker/code/example-2/test

# 将当前目录下的所有内容复制到
/Users/ssh/Documents/private/project/github/programmer-learning-
notes/docker/code/example-2/test 下
ADD . /Users/ssh/Documents/private/project/github/programmer-
learning-notes/docker/code/example-2/test

# 使用 pip 命令安装这个应用所需要的依赖
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# 允许外界访问容器的 80 端口
EXPOSE 80

# 设置环境变量
ENV NAME World

# 设置容器进程为：python app.py，即：这个 Python 应用的启动命令
CMD ["python", "app.py"]
```

```
## requirements.txt 内容
flask==1.1.1
```

```
# 在存放Dockerfile目录下执行一下命令
docker build -t python_web:1.0.0 .
# 启动镜像，如果不指定绑定4000端口的话，那么就会在宿主机上随机分配一个端口与镜像进行连接
docker run -d -p 4000:80 --name python_web python_web:1.0.0
```

H2 五、Docker-Compose

> - 之前运行一个镜像，需要添加大量的参数，可以通过Docker-Compose编写这些参数
> - Docker-Compose可以帮助我们批量的管理容器，只需要通过一个docker-compose.yml文件维护即可

H3 5.1 安装docker-compose

> mac 版本通过brew进行安装的docker 默认docker-compose已经安装

H4 5.1.1 手动安装

```
## 以linux安装为例
# 1、去github官网搜索并下载docker-compose
https://github.com/docker/compose/releases/tag/1.26.0
# 2、将下载好的文件放到宿主机
# 3、对DockerCompose重命名，并授予可执行权限
mv docker-compose-Linux-x86_64 docker-compose
chmod 777 docker-compose
# 4、为了方便后期操作，配置环境变量
# 将docker-compose文件移动到/usr/local/bin，修改/etc/profile文件，给/usr/local/bin配置到PATH中
mv docker-compose /usr/local/bin
vi /etc/profile
  export PATH=$PATH;/usr/local/bin
source /etc/profile
# 5、测试
# 在任意目录下数据docker-compose,返回如下图片说明安装成功
```

```
(base) ~ docker-compose
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
```

H3 5.2 docker-compose管理MySQL和Tomcat容器

> - yml文件以key: value方式来指定配置文件信息
> - 多个配置信息以换行+索引的方式来区分
> - 在docker-compose.yml文件中，不要使用制表符，使用缩进采用两个空格

```
version: '3.1'
services:
  mysql:                                        # 服务的名称
    restart: always                             # 代表只要docker启动
    image: daocloud.io/library/mysql:5.7.4      # 指定镜像路径
    container_name: mysql                       # 指定容器名称
    ports:                                      # 指定端口号的映射，可以指
定多个
      - 3306:3306
    environment:                                # 指定一些环境
      MYSQL_ROOT_PASSWORD: root                 # 指定MySQL的ROOT用户登录
密码
      TZ: Asia/Shanghai                         # 指定时区
    volumes:
      - /opt/docker_mysql_tomcat/mysql_data:/var/lib/mysql    # 映射数据
卷，将容器目录：/var/lib/mysql映射到宿主机目
录：/opt/docker_mysql_tomcat/mysql_data
  tomcat:
    restart: always
    image: daocloud.io/library/tomcat:8.5.15-jre8
    container_name: tomcat
    ports:
      - 8080:8080
    environment:
      TZ: Asia/Shanghai
    volumes:
      -
/opt/docker_mysql_tomcat/tomcat_webapps:/var/local/tomcat/webapps
      - /opt/docker_mysql_tomcat/tomcat_logs:/var/local/tomcat/logs
```

H3 5.3 使用docker-compose命令管理容器

> 在使用docker-compose的命令时，默认会在当前目录下找docker-compose.yml文件

```
# 1、基于docker-compose.yml启动管理的容器
docker-compose up -d
```

```
# 2、关闭并删除容器
docker-compose down
```

```
# 3、开启或关闭已经存在的有docker-compose维护的容器
docker-compose start|stop|restart
```

```
# 4、查看有docker-compose管理的容器
docker-compose ps
```

```
# 5、查看日志
docker-compose logs -f
```

H3 5.4 docker-compose配置Dockerfile使用

> 使用docker-compose.yml文件以及Dockerfile文件在生成自定义镜像的同时启动当前镜像，
>
> 并且有docker-compose去管理容器

docker-compose.yml

```
# yaml文件
version: '3.1'
services:
  ssm:                                          # 服务的名称
  build:                                        # 构建自定义镜像
    context: ../                                # 指定dockerfile文件的所在
路径
    dockerfile: Dockerfile                      # 指定Dockerfile文件名称
  image: ssm:1.0.1
  container_name: ssm
  ports:
    - 8081:8080
  environment:
    TZ: Azsia/Shanghai
```

Dockerfile文件

```
from daocloud.io/library/tomcat:8.5.15-jre8
copy ssm.war /usr/local/tomcat/webapps
```

```
# 1、可以直接启动基于docker-compose.yml以及Dockerfile文件构建的自定义镜像
docker-compose up -d
# 如果自定义镜像不存在，会帮助我们创建出自定义镜像，如果自定义镜像已经存在，会直接运行这个自定义镜像
# 重新构建的话
# 重新构建自定义镜像
docker-compose build
# 运行前，重新构建
docker-compose up -d --build
```

H2 六、Docker CI、CD

H3 6.1 引言

> 项目部署
>
> 1. 将项目通过m编译打包
> 2. 将文件上传到指定的服务器中
> 3. 将war包放到tomcat的目录中
> 4. 通过Dockerfile将Tomcat和war包转成一个镜像，由DockerCompose去运行容器
>
> 项目更新了
>
> 将上述I流程再次的从头到尾的执行一次

H3 6.2 CI介绍

> CI（continuous intergration）持续集成
>
> 持续集成：编写代码时，完成了一个功能后，立即提交代码到Git仓库中，将项目重新的构建并且测试
>
> - 快速发现错误
> - 防止代码偏离主分支

H3 6.3 实现持续集成

H4 6.3.1 搭建Gitlab服务器

> 1、创建一个全新的虚拟机，并且至少指定4G的运行内存

> 2、安装docker以及docker-compose

> 3、将ssh的默认22端口，修改为60022端口

```
vi /etc/ssh/sshd_config
  PORT 22 -> 60022
systemctl restart sshd
```

> 4、docker-compose.yml文件去安装gitlab（下载和运行的时间比较常的）

```
version: '3.1'
services:
  gitlab:
    image: 'gitlab/gitlab-ce:latest'
    restart: always
    hostname: 'gitlab'
    privileged: true
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url '192.168.2.106:8929' # http协议所使用的访问地址,不加端口默认80
        gitlab_rails['gitlab_shell_ssh_port'] = 2224 # 此端口是run时22端口映射的2224端口
        gitlab_rails['smtp_enable'] = true
        gitlab_rails['time_zone'] = 'Asia/Shanghai'
    ports:
      - '8929:8929'
      - '2224:22'
    volumes:
      - '/opt/docker_gitlab/config:/etc/gitlab'
      - '/opt/docker_gitlab/logs:/var/log/gitlab'
      - '/opt/docker_gitlab/data:/var/opt/gitlab'
```

```
# 启动容器
docker-compose up -d
```

```
# 出现问题
# 在启动过程中会出现错误
================================================================================
==========
    Error executing action `create` on resource
'storage_directory[/var/opt/gitlab/git-data/repositories]'
================================================================================
==========

================================================================================
==========
```

```
      Error executing action `run` on resource 'ruby_block[directory
resource: /var/opt/gitlab/git-data/repositories]'
================================================================================

# 错误原因是文件夹权限不够，需要在宿主机上进行授权
# 原文中显示路径为/var/opt/gitlab/git-data/repositories 但是做了数据
卷：/opt/docker_gitlab/data:/var/opt/gitlab
# 所以需要下宿主机上进行相应授权
chmod -R 2770 /opt/docker_gitlab/data/git-data/repositories
```

H4 6.3.2 搭建GitLab-Runner

H5 6.3.2.1 安装环境准备

```
# 目录结构
gitlab-runner
    ├── docker-compose.yml
    └── environment
        ├── daemon.json
        ├── Dockerfile
        └── jdk-8u261-linux-x64.tar.gz
```

- 创建构建目录 `gitlab-runner/environment`
- 下载 jdk-8u261-linux-x64.tar.gz 并复制到 `environment`
- 在 `environment` 目录下创建`daemon.json`

```
# daemon.json内容如下
{
  "registry-mirrors": [
    "http://aad0405c.m.daocloud.io"
  ],
  "insecure-registries": [
    "192.168.75.131:5000"
  ]
}
```

`registry-mirrors`: docker镜像地址

`insecure-registries`: Registry镜像私服地址

在`environment`目录下创建`Dockerfile`

```
FROM gitlab/gitlab-runner:v11.1.0
MAINTAINER Gbx <454368813@163.com>
```

```
# 修改软件源
RUN echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted
universe multiverse' > /etc/apt/sources.list && \
    echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-security main
restricted universe multiverse' >> /etc/apt/sources.list && \
    echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main
restricted universe multiverse' >> /etc/apt/sources.list && \
    echo 'deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main
restricted universe multiverse' >> /etc/apt/sources.list && \
    apt-get update -y && \
    apt-get clean

# 安装 Docker
RUN apt-get -y install apt-transport-https ca-certificates curl
software-properties-common && \
    curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg |
apt-key add - && \
    add-apt-repository "deb [arch=amd64]
http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs)
stable" && \
    apt-get update -y && \
    apt-get install -y docker-ce
COPY daemon.json /etc/docker/daemon.json

# 安装 Docker Compose
WORKDIR /usr/local/bin
RUN curl -L
https://get.daocloud.io/docker/compose/releases/download/1.21.2/docker
-compose-`uname -s`-`uname -m` > ./docker-compose
RUN chmod +x docker-compose

# 安装 Java
RUN mkdir -p /usr/local/java
WORKDIR /usr/local/java
COPY jdk-8u261-linux-x64.tar.gz /usr/local/java
RUN tar -zxvf jdk-8u261-linux-x64.tar.gz && \
    rm -fr jdk-8u261-linux-x64.tar.gz

# 安装 Maven
RUN mkdir -p /usr/local/maven
WORKDIR /usr/local/maven
RUN wget http://mirrors.hust.edu.cn/apache/maven/maven-
3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
# COPY apache-maven-3.6.3-bin.tar.gz /usr/local/maven
RUN tar -zxvf apache-maven-3.6.3-bin.tar.gz && \
    rm -fr apache-maven-3.6.3-bin.tar.gz
# COPY settings.xml /usr/local/maven/apache-maven-
3.6.3/conf/settings.xml
```

```
# 配置环境变量
ENV JAVA_HOME /usr/local/java/jdk1.8.0_261
ENV MAVEN_HOME /usr/local/maven/apache-maven-3.6.3
ENV PATH $PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin

WORKDIR /
```

> 在`gitlab-runner`目录下创建`docker-compose.yml`

```
version: '3.1'
services:
  gitlab-runner:
    build: environment
    restart: always
    container_name: gitlab-runner
    privileged: true
    volumes:
      - /opt/docker_runner/config:/etc/gitlab-runner
      - /var/run/docker.sock:/var/run/docker.sock
```

> 在宿主机启动docker程序后先执行`sudo chown root:root /var/run/docker.sock`（如果重启过docker，重新执行）

H5 6.3.2.2 运行并注册Runner

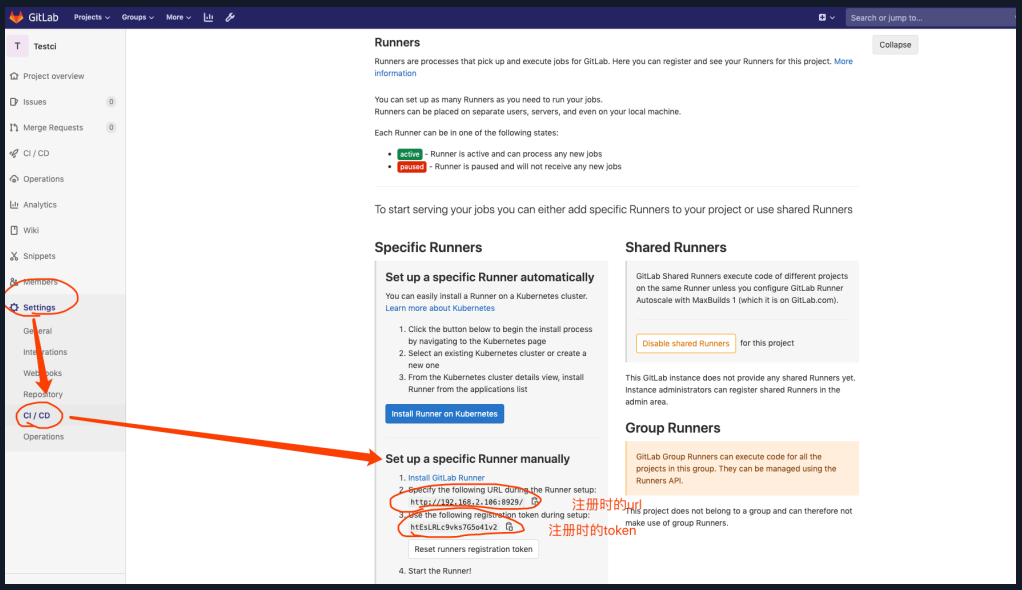
> 创建工程`testci`

> 在`gitlab-runner`目录下运行

```
docker-compose up -d
```

> 添加容器权限，保证容器可以使用宿主机的`docker exec -it gitlab-runner usermod -aC root gitlab-runner`

> 打开Gitlab要持续集成的仓库,打开`设置` -> `CI/CD` -> `Runner`
>
> 在`Setup a specific Runner manually`下获取仓库地址和注册令牌备用

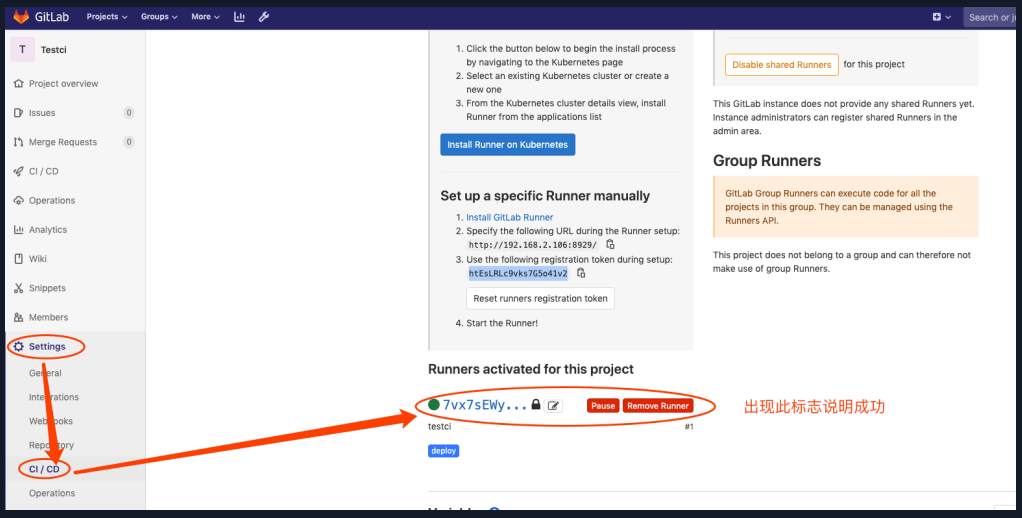



```
# 输入注册命令
docker exec -it gitlab-runner gitlab-runner register
```

> 根据提示输入刚才获得的地址和令牌,进行注册

```
# 输入 GitLab 地址
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
http://192.168.2.106:8929/

# 输入 GitLab Token
Please enter the gitlab-ci token for this runner:
htEsLRLc9vks7G5o41v2

# 输入 Runner 的说明
Please enter the gitlab-ci description for this runner:
可以为空

# 设置 Tag，可以用于指定在构建规定的 tag 时触发 ci
Please enter the gitlab-ci tags for this runner (comma separated):
deploy(也可以为空)

# 选择 runner 执行器，这里我们选择的是 shell
Please enter the executor: virtualbox, docker+machine, parallels,
shell, ssh, docker-ssh+machine, kubernetes, docker, docker-ssh:
shell
```

> 绑定成功标识如下图

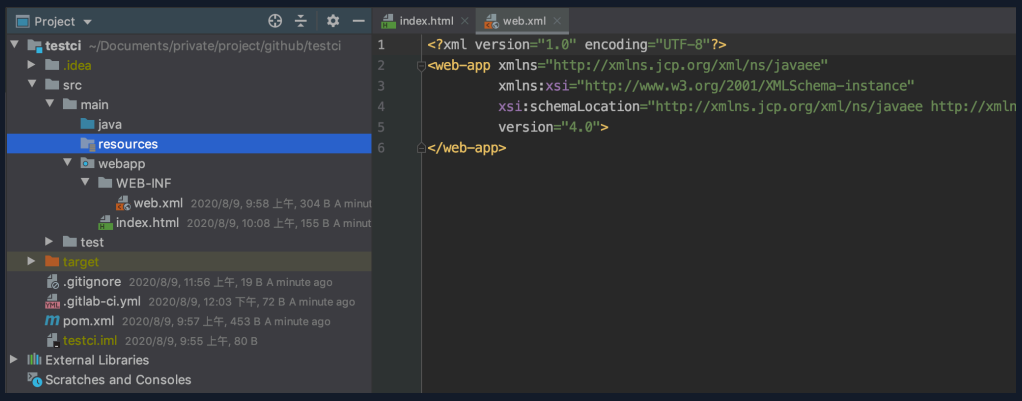


H5 6.3.2.3 代码的持续集成

- 在代码中编写 `.gitlab-ci.yml` 以及需要用到的 `docker-compose.yml` 和 `Dockerfile`
- 推送代码的时候就会在Gitlab的页面中看到持续集成的作业进度

H4 6.3.3 整合项目入门测试

1、创建maven工程，添加we'b.xml文件，编写html页面

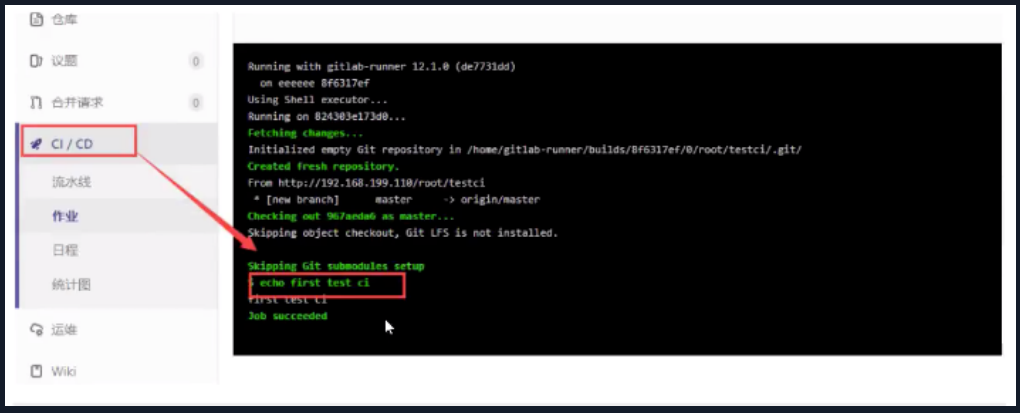


2、编写gitlab-ci.yml文件

```yaml
stages:
  - test

test:
  stage: test
  script:
    - echo first test ci  # 输入的命令，可以输入多个
```

3、将maven工程推送到gitlab中

4、可以在gitlab中查看到gitlab-ci.yml编写的内容，如下图



H4 6.3.4 编写.gitlab-ci.yml文件

> 1、编写.gitlab-ci.yml测试命令使用

```
stages:
  - test

test:
  stage: test
  script:
    - echo first test ci
    - /usr/local/maven/apache-maven-3.6.3/bin/mvn package
```

> 2、编写关于dockerfile以及docker-compose.yml文件的具体内容

```
# 1. 准备Dockerfile
FROM daocloud.io/library/tomcat:8.5.15-jre8
COPY testci.war /user/local/tomcat/webapps
```

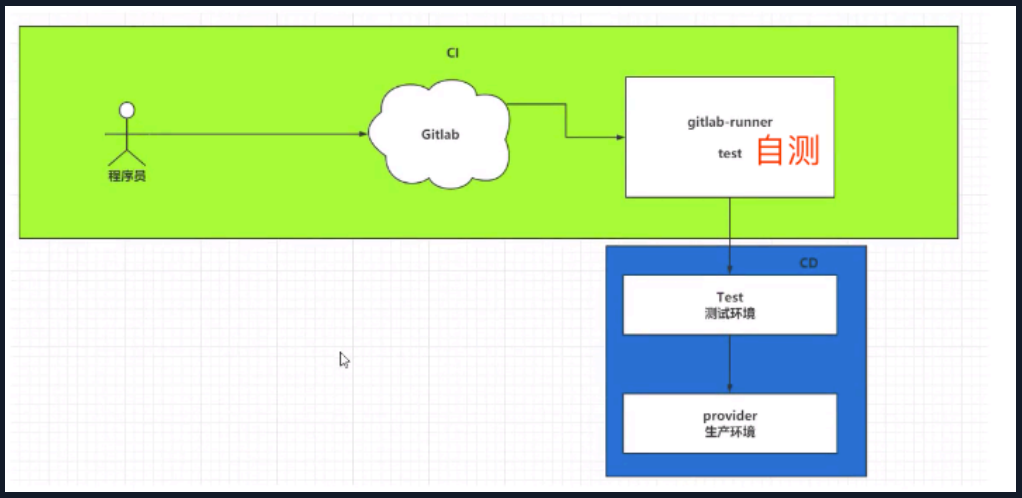
```
# 2. docker-coompose.yml
version: "3.1"
services:
  testci:
    build: docker
    restart: always
    container_name: testci
    ports:
      - 8080:8080
```

```
# 3. .gitlab-ci.yml
stages:
  - test

test:
  stage: test
  script:
    - echo first test ci
    - /usr/local/maven/apache-maven-3.6.3/bin/mvn package
    - cp target/testci-1.0-SNAPSHOT.war docker/testci.war
    - docker-compose down
    - docker-compose up -d --build
    - docker rmi $(docker images -qf dangling=true)   # 删除build后老的
images 变成None的情况
```

> 3、测试效果：每次推送后都可以在浏览器中看到变化

H3 6.4 CD介绍

> CD（持续交付，持续部署）
>
> 持续交付：将代码交付给专业的测试团队去测试（gitlab-runner）
>
> 持续部署：将测试通过的代码，发布到生产环境（jenkins）



> 参考文献
>
> 自定义Docker容器镜像并将其上传到DockerHub中

> 2020 Docker最新超详细版教程通俗易懂
>
> 手把手教你入门SSM框架开发
>
> Docker 查看容器 IP 地址
>
> 基于docker-compose搭建gitlab
>
> 使用Docker 搭建Gitlab-Runner持续集成平台
>
> 解决：Connecting to raw.githubusercontent.com failed 以及 Connection refused
>
> 使用Gitlab-Runner持续集成代码