计算机系《软件工程》小作业讲解

CI/CD

计算机系《软件工程》助教团队

清华大学计算机科学与技术系

2024 年 9 月 21 日

小作业部署说明

起因

- 往年大作业前期往往难以实际将项目部署上线，因此我们希望在小作业中模拟大作业部署全流程，让同学们对 CI/CD 概念有基本了解，能够快速上手大作业
- (以前学期情况) SECoder 分配容器 IP 受到网段限制，不足以为每位同学都开放前端和后端两个容器

小作业部署说明

新方案

- 仅为各位同学开放前端容器供大家完成 CI/CD 小作业的前端部署部分
- 部署一个标准后端供大家对接、测试、使用
- 后端部署部分则由各位同学自行完成并测试，不部署到 SECoder 平台

小作业部署说明

作业文档

https://thuse-course.github.io/course-index/handout/ci-cd/

- 后端部署请务必在本地测试，否则可能造成不必要的失分
- 前端部署请务必仔细阅读文档，按照要求的方式 (standalone) 部署。采用其他方式也可成功部署 (例如直接全部 `COPY` 后 `yarn dev`)，但由于不完全符合作业要求，我们会酌情扣分。

小作业部署说明

提醒

- 小作业设计中对安全并没有过多考虑，例如直接明文传输、存储了密码，这可能带来一定安全隐患
  - 大作业时请务必做好用户身份识别和数据库安全
- 数据量可能会达到上百条，但前端小作业列表页面并未设置分页器，可能会导致加载速度慢等问题
  - 大作业时涉及到的数据可能会更大，请务必设计分页器等限制措施
- 同学间可以更方便地共享有趣的游戏记录 (?)
- 切勿攻击共享后端

小作业部署说明

提醒

- 切勿赶 DDL. 在 DDL 前 Runner 压力可能非常高，这将可能导致排队时间过长而错过 DDL
  - 大作业也同理，尽量不要赶组会前集中迭代
- 仓库权限要开成 *Private*
  - 而不是 Internal，否则所有人都会看到你的作业

Docker 的作用

您配吗?

English README

你是否经常有这样的困扰——

接手了一个新项目，但是仓库里的依赖文件已经年久失修，比如老的依赖装不上了，或者是代码和依赖的版本不一致，甚至根本没有依赖文件！

你硬着头皮配了半天环境，还是遇到了各种各样的问题，根本跑不起来。

于是你只好去问之前的owner，结果他就演示给你看说他那里运行得好好的啊。

——这个时候你要怎么办呢?

> A. 忍气吞声，自己去配环境。
>
> B. 使用莉沫酱最新发明的工具「您配吗」快速解决问题！

使用方法

你只要把「您配吗」偷偷发给同事启动，他电脑上装好的依赖就会被删掉，然后他就得自己配环境了！

这个时候你只要在旁边盯着他，就可以得到详细的安装步骤啦！

(当然也有一种情况是他自己也装不上了)

Docker 的作用

Docker

Docker 将应用及其所需要的环境打包并交付给其他人使用，使得应用能够在一致的环境下一键运行，而不需要手动配置环境

特点

- 更高效的利用系统资源
- 更快速的启动时间
- 更方便持续交付和部署

Docker 中的重要概念：镜像、容器与 Registry

镜像（Image）

Docker 镜像是一个特殊的文件系统

- 提供容器运行时所需的程序、库、资源、配置等文件
- 包含一些为运行时准备的一些配置参数
- 不包含任何动态数据，其内容在构建之后也不会被改变

Docker 中的重要概念：镜像、容器与 Registry

镜像（Image）

Docker 镜像的构建：Dockerfile 示例

```
FROM pytorch/pytorch:1.8.1-cuda10.2-cudnn7-devel

ENV TORCH_CUDA_ARCH_LIST="6.0 6.1 7.0+PTX"
ENV TORCH_NVCC_FLAGS="-Xfatbin -compress-all"
ENV CMAKE_PREFIX_PATH="$(dirname $(which conda))/../"

RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys A4B469963BF863CC && \
  apt-get update && \
  apt-get install -y ffmpeg libsm6 libxext6 git ninja-build libglib2.0-0 libsm6 libxrender-dev libxext6

# Install MMCV, MMDetection and MMSegmentation
RUN pip install mmcv-full==1.3.8 -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/index.html
RUN pip install mmdet==2.14.0
RUN pip install mmsegmentation==0.14.1

# Install MMDetection3D
RUN conda clean --all
RUN git clone https://github.com/samsunglabs/fcaf3d.git /mmdetection3d
WORKDIR /mmdetection3d
ENV FORCE_CUDA="1"
RUN pip install -r requirements/build.txt
RUN pip install --no-cache-dir -e .
```

- 语法规范详见：https://yeasy.gitbook.io/docker_practice/image/dockerfile
- 之后使用 docker build 命令构建镜像

Docker 中的重要概念：镜像、容器与 Registry

容器（Container）

- 镜像的实例：镜像是静态的定义，容器是镜像运行时的实体
- 具有易失性：任何保存于容器存储层的信息都会随容器删除而丢失（大作业需要将用户数据保存在持久化存储中）

容器的启动：

- （本地）docker run -itd -p 10001:8000 –name <Container Name> <Image>
- （其他部署工具）查阅对应文档...

Docker 中的重要概念：镜像、容器与 Registry

Registry

概念辨析：仓库（Repository）、注册服务器（Registry）

- 镜像构建完成后，可以很容易的在当前宿主机上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。
- 一个 Docker Registry 中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。

Docker 中的重要概念：镜像、容器与 Registry

Registry

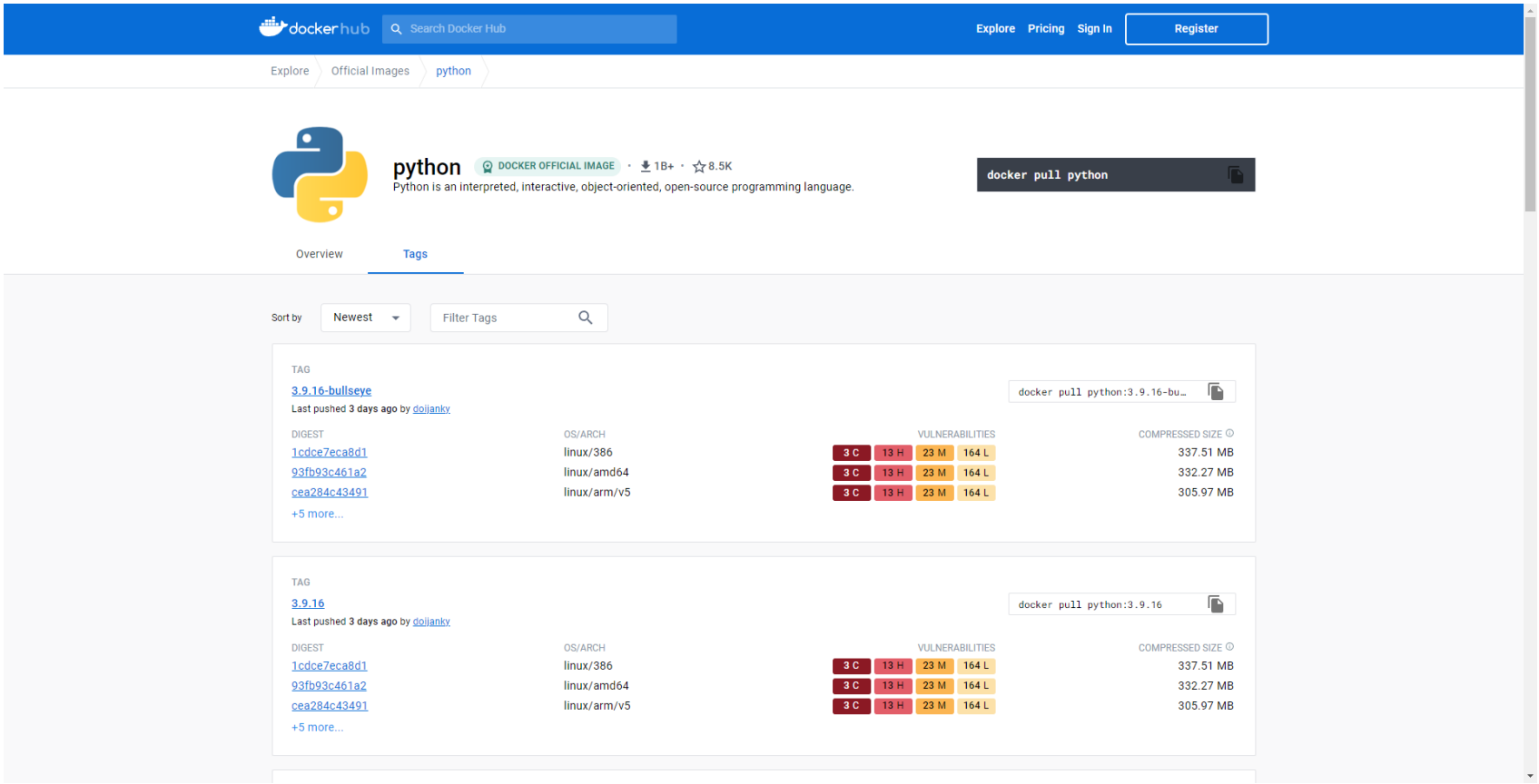


图 1: 概念辨析：Docker Hub; 图中的 python; python:3.9.16

Dockerfile 编写入门

参考 课程文档 Docker 部分 与 2023 酒井科协暑培 Docker 课程

持续集成与持续部署

持续集成 (Continuous Integration/CI)

- 在代码构建过程中持续地进行代码的集成、构建、以及自动化测试等
- 可以在代码提交的过程中通过单元测试等尽早地发现引入的错误

持续部署 (Continuous Deployment/CD)

- 在代码构建完毕后迅速将新版本部署上线
- 有利于快速迭代并交付产品

持续集成与持续部署

为什么需要 CI/CD

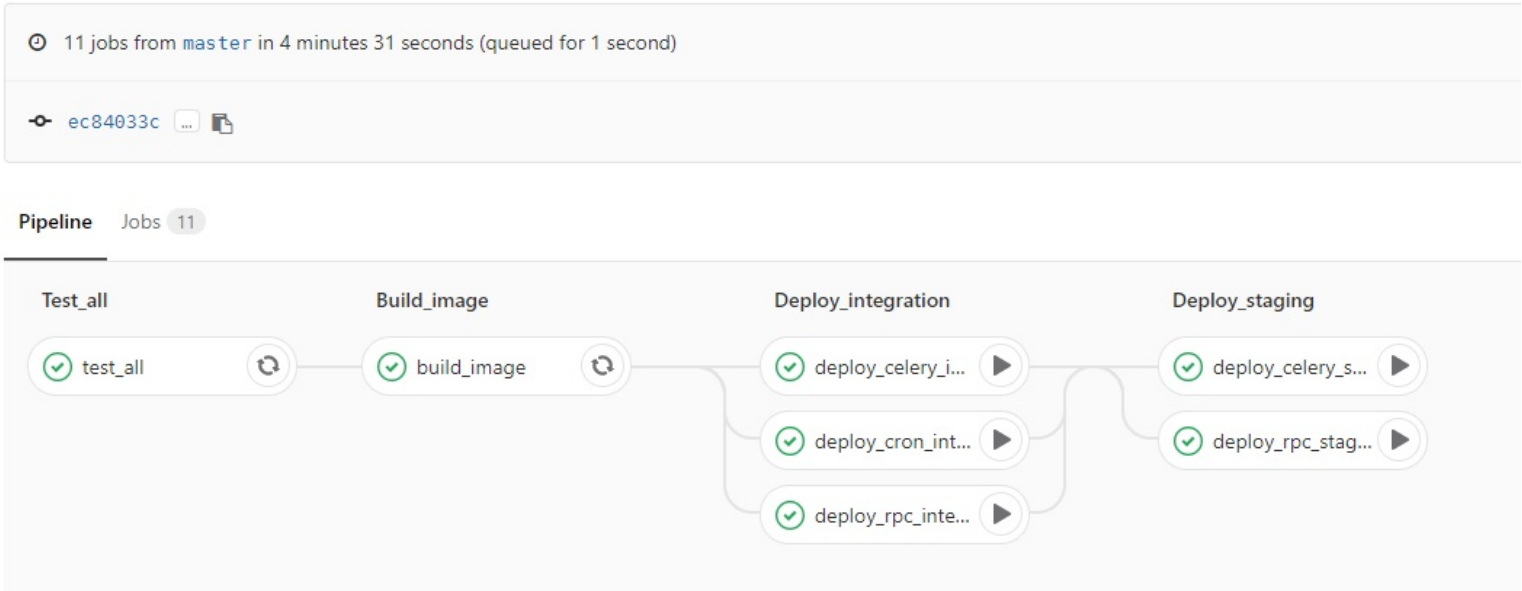
- 自动化：不用手动执行测试、部署等
- 标准化：定义一套固定的流程，避免人为部署带来的错误
- 尽早发现错误：避免长时间后难以定位问题根源

GitLab CI/CD

GitLab CI/CD

- 一套基于 GitLab 的 CI/CD 系统
- 可以让开发人员通过.gitlab-ci.yml 在项目中配置 CI/CD 流程
- 在每次 push 到仓库后，系统都可以自动（或手动）地执行 CI/CD 所定义的操作

基本概念


11 jobs from master in 4 minutes 31 seconds (queued for 1 second)
ec84033c
Pipeline
Jobs 11
Test_all
Build_image
Deploy_integration
Deploy_staging
test_all
build_image
deploy_celery_i...
deploy_cron_int...
deploy_rpc_inte...
deploy_celery_s...
deploy_rpc_stag...

基本概念

作业 (Job)

- CI/CD 流程的最小执行单元
- 包含了一系列需要执行的命令
- 需要指定一个 Docker 镜像，在执行 CI/CD 时将会基于此镜像运行一个容器，在其中执行命令
- 成功状态将取决于其最后一条命令的返回值

基本概念

阶段 (Stage)

- CI/CD 流程划分为多个阶段分别进行，每个阶段可以包含一个或多个作业
- 同一个阶段的多个作业可以并行执行
- 一个阶段的所有作业均成功执行后，才会执行下一个阶段的作业时将会基于此镜像运行一个容器，在其中执行命令

基本概念

流水线 (Pipeline)

- 多个阶段顺序连接组成一个流水线
- 将代码推送到远程仓库或是发起合并请求时，GitLab 会基于该版本的代码执行流水线

.gitlab-ci.yml

镜像指定

```
image:  python:3.9
```

阶段定义

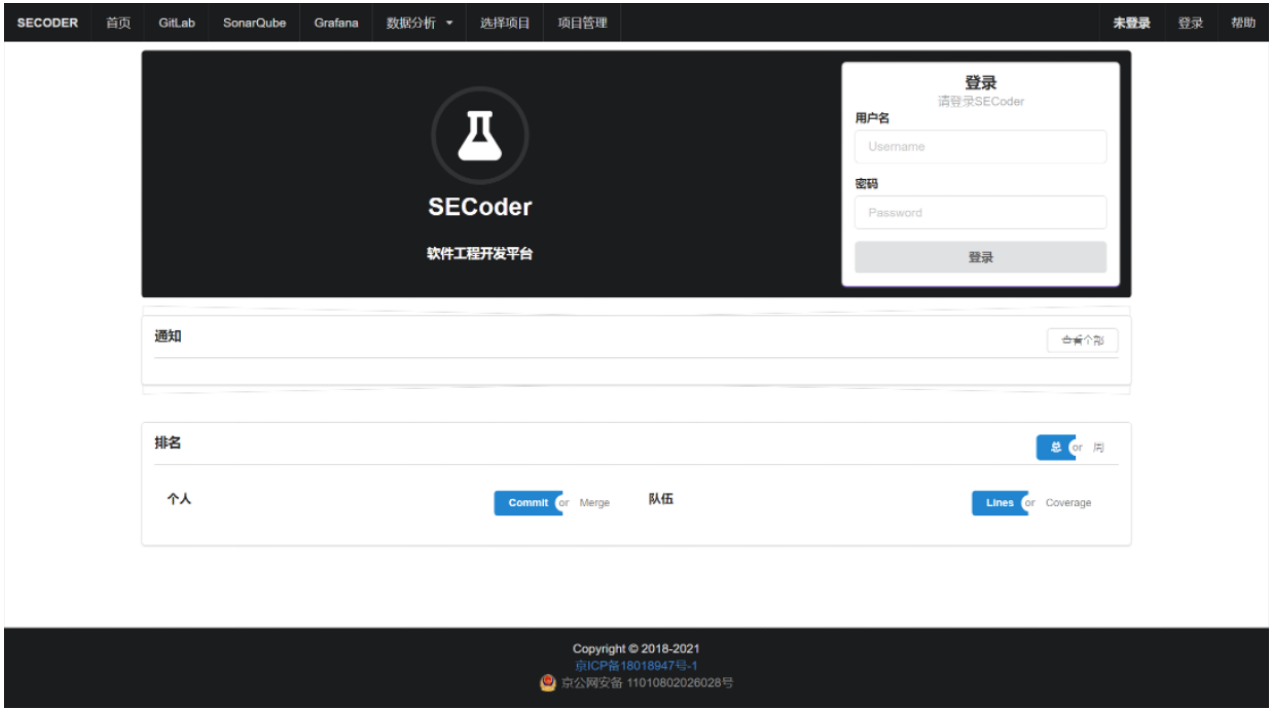
```
stages:
  - build
  - test
  - deploy
```

.gitlab-ci.yml

作业定义

```
1 style-test:
2   stage:  test
3
4   before_script:
5     - pip install pylint
6   script:
7     - pylint **/*.py
```

SECoder



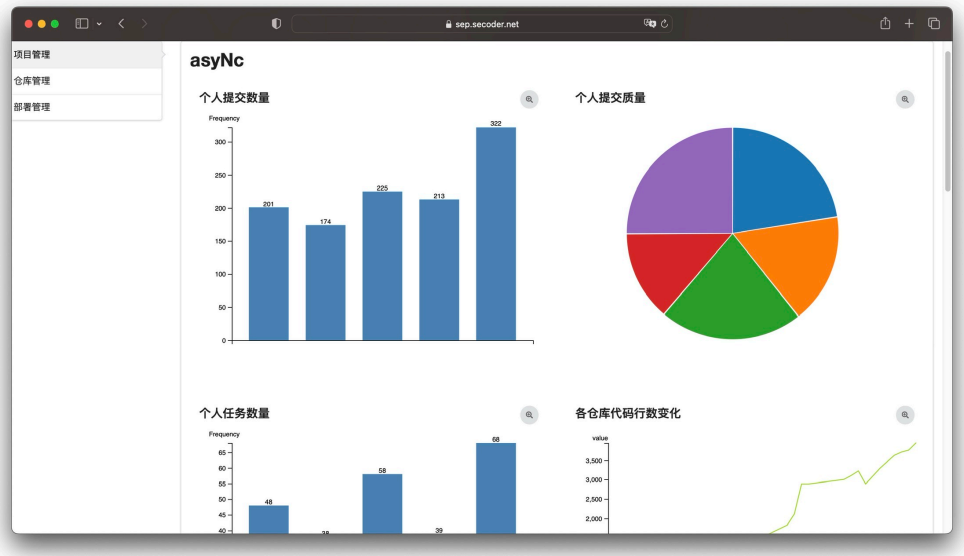
软件工程开发平台 (SECoder)

- 帮助同学们顺利完成软件工程课程大作业的在线网站，提供了 GitLab、SonarQube 等用于更好管理开发流程的工具
- `https://sep.secoder.net`

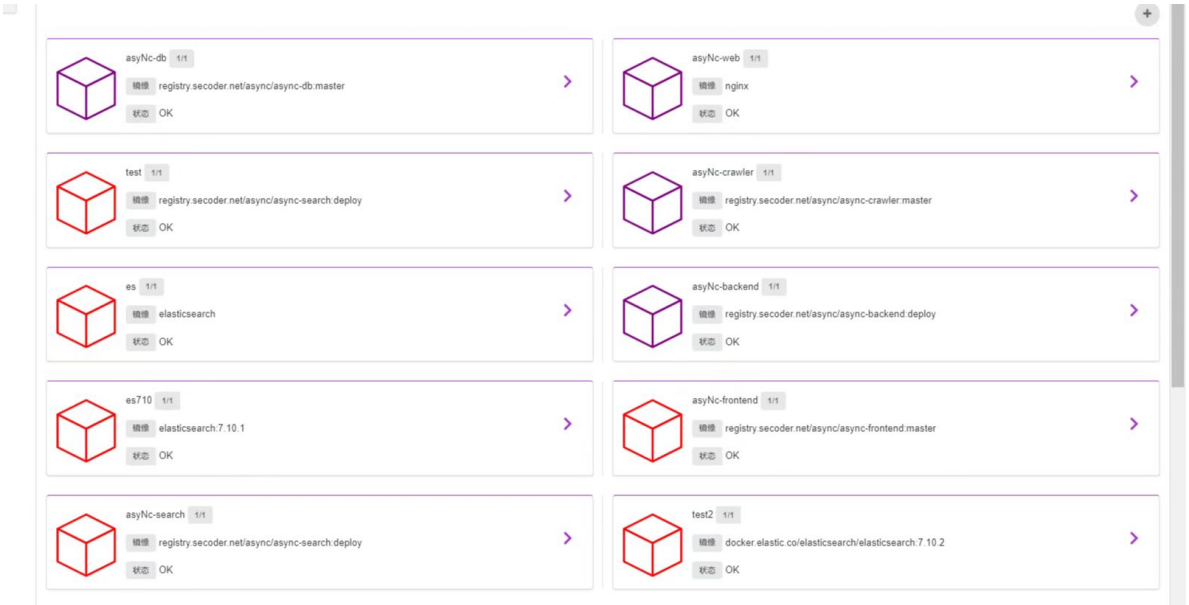
SECoder

项目管理

- 提交数量和质量
- 任务数量
- 各仓库代码行数
- 注释比例
- 构建成功率
- 测试覆盖率



SECoder



部署管理

- 对团队的容器进行各种管理操作
- 建立仓库时选择「启用部署」即可创建一个关联的容器

配置项

- 在开发时和正式部署时使用不同的配置是一种常见做法
- 然而，由于容器的易失性，一个容器在删除后其中所做的修改都将丢失
  - 这意味着每次都手动修改配置会很麻烦
- 可以通过配置项来简化这个流程
  - 配置项是只读的挂载项，可以用于存放各种配置文件
  - 配置项可以作为一个目录被挂载到容器中
  - 这样，在部署时将会自动使用挂载的配置
- 可参考文档中的 配置挂载演示

持久存储

- 数据库容器等需要在容器内保存数据
- 但同样由于容器的易失性，在容器重启时其中的数据将会丢失，因此我们需要一种能够持久保存数据的方法
- 持久存储与配置项类似，都可以挂载到容器的某一目录
  - 不同点在于持久存储是可写的，并且你不需要为其提供初始内容
  - 在不同的容器实例间保持数据
- 操作步骤与配置项类似
- 可参考 SECoder 帮助文档中的 数据库教程

Thanks!

Questions?