迭代三部署文档

值得注意的是，本文档提供两种部署方式：直接部署和通过jenkinsfile部署。

本地部署对于环境要求较低，无需Jenkins，如果只是想复现部署结果，没有持续集成的要求，可以选择此种方法；而通过jenkinsfile部署还需要安装jenkins，以及处理各种配置和挂载，并且需要linux环境，如果需要在本地进一步进行开发和持续集成，可以选择此种方法。

此次迭代已有的Jenkins地址为http://120.77.16.147:8088/，用户名为admin，密码为keha1993

已部署好的前端项目地址是http://121.41.59.63:8082/

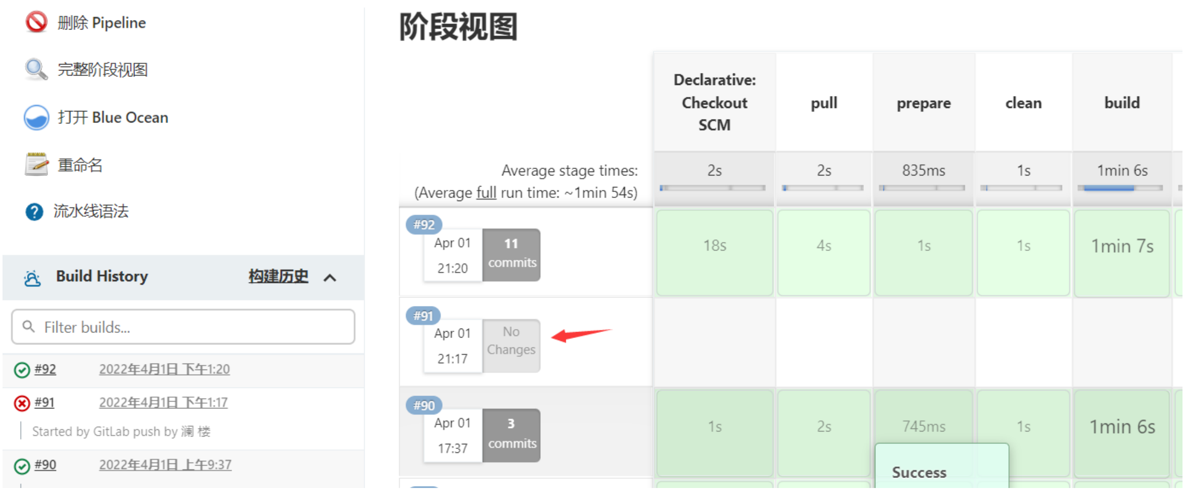
已部署好的后端项目地址是http://1.15.174.191:8080/ 可通过knife4j 接口文档查看后端实现接口

已部署好的python项目地址是http://1.15.174.191:5000/

部署注意事项

1. 不要同时构建多个项目，容易出现服务器卡死
2. 如果在新服务器上第一次部署，需要运行各种install命令，会比较慢，要十几分钟，请耐心等待
3. 有时候因为网络问题，git会超时，请手动停止此次构建，并重新构建就可以顺利git。但这种情况次数较少
4. 如果在新增数据的情况下重新部署，有可能当前测试会有部分无法通过，这是正常情况
5. python项目的`sgns.merge.word`与`sgns.merge.word.freq`获得比较麻烦，如果不大方便可以私戳我们组组长获取（这两个文件很大，所以没有放在gitlab上，直接存在服务器中）

6. **如果在五分钟内进行两次连续的push操作，由于内网gitlab和外网gitlab有五分钟之内只能同步一次的限制，所以可能不能及时同步。可以稍加等待一段时间，会自动同步；如果助教不想等待，可以等五分钟一到，在内网仓库手动同步。**
7. 如果部署过程中，遇到任何问题都可以戳`191850124 楼澜`
8. 因为网络等等问题，有时候push是明明有改动，jenkins却显示no change然后构建失败，可以看到此时流水线都没开始走。但是这种情况十分少，请助教如果遇到了麻烦重新push一下或者手动构建



直接部署过程

0. 部署前的准备

- 准备一台服务器，配置在2核4G左右，`centos7`操作系统
- 打开这台服务器`8080`,`5000`,`8082`端口的安全组
- 安装`git`，使用`yum -y install git`命令，并且从相应gitlab仓库中clone相应项目
- 安装`docker`，输入`yum install docker-ce`即可，并且输入`systemctl start docker`打开docker服务

1. 部署前端

- 安装node（**版本16.14.0**），输入以下命令

```
wget https://nodejs.org/dist/v16.14.0/node-v16.14.0-linux-x64.tar.xz
xz -d node-v16.14.0-linux-x64.tar.xz
tar xf node-v16.14.0-linux-x64.tar
ln -s /root/node-v16.14.0-linux-x64/bin/node /usr/local/bin/node
ln -s /root/node-v16.14.0-linux-x64/bin/npm /usr/local/bin/npm
```

- 创建`nginx.config`文件，路径为`/export/nginx/nginx.conf`，内容如下：

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    #include /etc/nginx/conf.d/*.conf;
    server {
        listen       80;
        listen  [::]:80;
        server_name  1.15.174.191;

        #charset koi8-r;
        #access_log  /var/log/nginx/host.access.log  main;

        location / {
            root   /usr/share/nginx/html;
            index  index.html index.htm;
        }

        location ^~/api/{
            proxy_set_header Host $http_host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header REMOTE-HOST $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://1.15.174.191:8080/api/;
        }

        #error_page  404              /404.html;

        # redirect server error pages to the static page /50x.html
        #
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
}
```

其中，注意更改server_name xx.xx.xx.xx与proxy_pass http://xx.xx.xx.xx:xxxx/api/，确保其为目前的后端地址

- 进入项目根目录，输入`make`命令，即可完成部署。

以上都完成后，便可在这台机器的`8082`端口访问到部署完成的前端

2. 部署后端

- 安装java，输入`yum -y install java-1.8.0-openjdk*`命令即可，并把java加入环境变量（方法同maven）

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0
export JRE_HOME=$JAVA_HOME/jre
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
```

- 安装maven（**版本3.6.3**），步骤如下：
  - 下载maven，使用命令`wget https://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz`
  - 解压，使用命令`tar -xf apache-maven-3.6.3-bin.tar.gz`
  - 将maven文件夹拷贝到/usr目录，使用命令`cp -r ./maven3.6 /usr`
  - 加入环境变量
    - `vi /etc/profile`，打开后在最后一行加上`export PATH=$PATH:/usr/maven3.6/bin`
    - 退出vim，输入`source /etc/profile`使配置生效
- 进入项目，更改Makefile（由于正式部署采用CICD，Jenkins是用docker起的，所以要挂载host机maven文件夹获取mvn命令，故直接部署要更改）

  将第一行改成`MVN  := mvn`即可
- 在项目根目录输入`make`命令，即可一键部署

以上都完成后，便可在这台机器的`8080`端口访问到部署完成的后端

3. 部署python

- 根据[Embedding/Chinese-Word-Vectors: 100+ Chinese Word Vectors 上百种预训练中文词向量 (github.com)](), 生成`sgns.merge.word`词向量文件
- 根据项目中的`doc similarity.ipynb`文件，生成`sgns.merge.word.freq`词频文件
- 将`sgns.merge.word`与`sgns.merge.word.freq`拷贝至`/pythonProject`目录下
- 在项目根目录输入`make`命令，即可一键部署

以上都完成后，便可在这台机器的`5000`端口访问到部署完成的后端

采用CI/CD部署过程

下面提供的部署策略主要针对前端、后端、python、jenkins在一台机器上的情况。

但是由于内存等等限制，实际上现实中我们采用后端和jenkins在一台服务器，python在另一台服务器，前端又在第三台服务器的部署策略。

所以，我们会在此部分的最后附上如何为jenkins设置`slave`节点的过程，以便于用户更好地选择适合自己的部署策略。

1. 前期准备

- 安装git、docker、java、maven、node，流程同上面所述
- 生成`nginx.conf`文件，并拷贝至`/export/nginx/nginx.conf`，流程同上面所述
- 生成`sgns.merge.word`与`sgns.merge.word.freq`文件，并拷贝到`/pythonProject`目录中，流程同上面所述
- 如果不配置slave节点，即前端与jenkins跑在一台机器上，需要把前端项目中`Makefile`第一行替换成

```
NPM := /usr/local/node/bin/npm
```

  反之，则不用换

- 如果配置slave节点，即后端与jenkins不跑在一台机器上，需要把后端项目中`Makefile`第一行替换成

```
MVN  := mvn
```

  反之，则不用换

- 如果不配置slave节点，即项目与jenkins跑在一台机器上，需要将此项目的`Jenkinsfile`中第二行改成

```
agent any
```

- 安装docker版的jenkins：此次Jenkins用的docker版本为**jenkinsci/blueocean:latest**，直接`docker pull jenkinsci/blueocean`即可。

2. 启动jenkins

在命令行输入以下命令启动Jenkins。值得注意的是，`/export/jenkins` **为jenkins数据持久化的挂载目录**，可以更换；`/export/nginx/nginx.conf`为nginx的配置文件路径；`/usr/maven3.6`为mvn命令挂载路径、`/usr/local/node`为node命令挂载路径、`/usr/bin/make`为make命令挂载路径，**请根据实际情况自行调整**。例如，如果前端在slave节点上运行，则不用挂载node命令；如果后端在slave节点上运行，则不用挂载mvn命令。值得注意的是，如果这些目录服务器中没有，会报错，请进行相应的文件夹创建/安装处理。

```
docker run \
  -u root \
  --rm \
  -d \
  -p 8088:8080 \
  -p 50000:50000 \
  --privileged \
  -v /export/jenkins:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /export/nginx/nginx.conf:/etc/nginx/nginx.conf \
  -v /usr/bin/make:/usr/bin/make \
  -v /usr/maven3.6:/usr/local/maven \
  -v /usr/local/node:/usr/local/node \
  jenkinsci/blueocean
```

3. 配置jenkins

（此次迭代已有的Jenkins地址为http://120.77.16.147:8088/，用户名为admin，密码为keha1993）

3.1 进入Jenkins

通过浏览器输入本机地址加**8088**端口进入Jenkins，配置完用户名密码后，选择下载默认插件。等待完成后，进入主界面。

3.2 下载插件

在系统管理->插件管理中，下载**GitLab Plugin**、**JaCoCo plugin**、**SSH Agent Plugin**、**Generic Webhook Trigger Plugin** *（这个可省）*插件。

3.3 凭据配置

- ssh_key

新建一个ssh凭证，类型为SSH Username with private key，ID设为**ssh_key**，名称随意。私钥设置为：

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAq/zT51gy9ufj5gaxWRVgBa+95xcuycoRtLGxYsL5wfSQ7+wd
JoZrHmH6jUM07ER+ulw/2doabQW++DdWDefzGca+OahSUQC/7YalLc7WN3k8FqwG
di7x9uF6UJqWzNQmec7JOKLM7d1RfdQcu7JS370r3TE7UfLMhcCbK28TMiejEC7f
mmxnoiimMmB8tkkXhbVq/xXJ9ZbnLPK5qtpLCyPkYnpdfCPCqxjltXHCsY3MjGHr
ycm6QMC1J+u9ia32V/DNb6+Ua5u0lA9/xCbkgJVgC7mc5UPdj3FayYbzB2KbCd04
+XjQkalWIM7l66H3qKOo26KLoRNykHKniRUiMQIDAQABAoIBAQCREgHr8gmLGq6x
KJQPgGguY77bXkKpEzk8IWZu1+e0Iobe2Vr9VASm7B5baYUKhfwfVhDFwaLosyAS
XxrHQA9efrWWDNTiXq+Fuz05Fm66/BgjR5druM2AYVC7DKqGu6x4smQo0anq0fSz
HmfeKtycvQRy30QaK6M4bu50BI08j7CCoros2n07PDM6cFGJCEU3AsglAOR4RmsE
2Z7OUjpeC4LvCtqI+6i4c/ghWSsR4vHJIF/5WqlDMndo3VsfZF9OIa8zbvn8p1KB
DtbHkTj9vPF7RXgUazRolVDYi0t4RExolb6aLeuI1+47eEs/kNOUGWK7xJ0qRvEP
4+oUfajBAoGBANVb8Q+tSOvSPAI3DceDgvsbtQeKvQaIeK160D3pv98mgzj2Qbqh
nHbxKCnz5upRMwO1jLZe6E7vDuzgIs9D4+Ltv19aX/JDi+WW+TzDkc1UdglBimSD
QyzY8Egqu1NXqL3FbLFnngABfjMEdXt+xizl/tGmp/3QjWfSdeXAXGFDAoGBAM5c
M3QfuWgAj7tde7OGYd4tWrb/h8Upfhb3imwoUD+oI3GUUTEGcD6ErldU/IY6NGLT
7uCuaAnkuEDt9YSNqhKWklv68QaAhpZqOTLR3Xqv/CWzVLJySdsT97rmr5jaZ/eV
7OYR1XQPhcwbWxa8KLBJETXzhLNi191wKfuwFg17AoGBAMjtGulekSu0Ats7KXjY
esMr5qHBoB3E65DD9dQ4i38E79L25hNyGr1Qgjhv/uhvq5E0qd0dJ70eGHou4dk6
4CNXLkAIBg9KWTbPpMv6iRZLEhXJaSEbFGnpqu7rfxoPlVOR1riDEiKDRWuaKWEx
lEO4HO8m+VFn06MQagMB0n2tAoGAdYSYYMc4RPwc3mzsZ15eGbLmeFSpMyTgA6BR
GisTGE1ece4vFqY+Abx5tI5XiPFYp/ddkGKCKTAxpfhd23D5q8BH9U3BORy0iLBR
hplxcc8K30VzNHRVjweeCrgYxAmNL7gZHWRGlOPKJGRnyVi6KzpRLNJTff0KRbb8
kbLDvEUCgYA5qnP8UCK3GVlUu+idUFyN/WxYTMHe9eCAK2Ajw3TYbtUibHRzxKuh
Iv65V5UD/1CyAy6rHAyyqSMpEWrXLWRv1YC/2X5oDGYCJK9GYCn/edbx0xOydJe3
fCaw4O1kuUnqUw8NGuDL2CTSz3/56EF7V38KQnjH0SZI7r9ozHZuzw==
-----END RSA PRIVATE KEY-----
```

- GitLab API token

新建一个GitLab API token凭证，输入token：`glpat-RAbwzR34kq-fwDh9-1jv`，以便`3.4 系统配置`配置git时使用

3.4 其他配置

- 点击`系统管理->系统配置`，基本配置同已有jenkins的配置，直接进入网站照着配置就好
- 点击`系统管理->全局工具配置`，基本配置同已有jenkins的配置，直接进入网站照着配置就好

4. 创建流水线

- 对于前端、后端、python，分别创建三个流水线任务
- `构建触发器`中，选择`Build when a change is pushed to GitLab. GitLab webhook URL: http://120.77.16.147:8088/project/xxxx Enabled GitLab triggers`，并且勾选希望触发构建的events （对于gitlab外网仓库，要根据这里提供的URL配置webhook，此处略）
- `流水线`中，设为设为`Pipeline script from SCM`，并且下面选择git，凭证选择刚刚创建的**ssh**凭证，分支选择***/develop**。前端仓库地址为`git@git.nju.edu.cn:monian/frontend-collect.git`，后端仓库地址为`git@git.nju.edu.cn:monian/backend-collect.git`，python仓库地址为`git@git.nju.edu.cn:monian/python-collect.git`。点击保存即可。

  （**pipeline脚本全都保存在各个项目的`Jenkinsfile`里**）

5. 构建项目

如果想要手动触发构建，对于三个流水线任务，直接点击`立即构建`即可。

附：jenkins新增slave节点

通过给jenkins配置slave节点，可以让jenkins将任务在其他服务器上跑，分担jenkins所属服务器内存压力。

- 在要当slave节点的服务器上下载相应软件，并做相应准备工作，具体见`1. 前期准备`
- 值得注意的是，无论这台服务器将要构建前端、后端、python，都需要装**java**
- 准备所需文件，并按需修改对应Makefile，具体见`1. 前期准备`
- 接下来根据持续集成工具jenkins slave节点配置和Pipeline任务构建(三)*运维的技术博客*51CTO博客一步步配置即可，直到`三.创建流水任务Pipeline`之前
- 修改对应项目下的的`Jenkinsfile`，第二行改为

```
agent {label 'slave的名字'}
```

- 按照之前流程创建流水线，点击立即构建即可

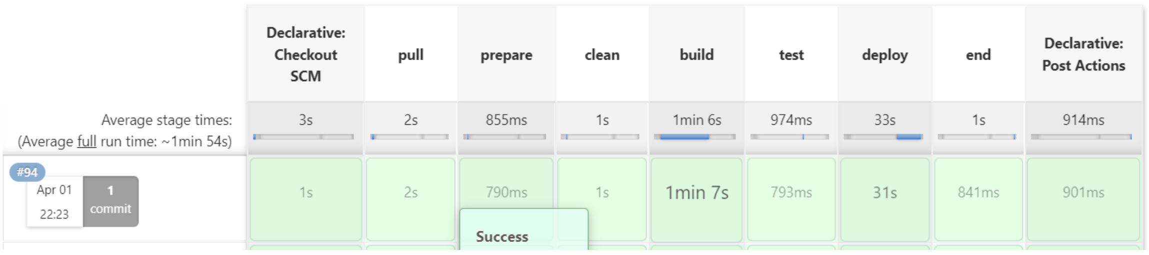
使用Jenkins流水线进行CI/CD集成

- 为了更好地一键部署，我们采用了先写好`Makefile`文件，然后在pipeline脚本调用各种命令来实现分阶段构建（这样最终交付时对方只需输入`make`就好）。**流水线执行的详细指令可参考同一项目的`Makefile`**。
- 按照目前的配置，前后端项目只有`develop`分支上的`push`与`merge`操作才会触发流水线，python项目只有`master`分支上的`push`与`merge`操作才会触发流水线。并且jenkins会从**相应分支**git clone代码。
- 当`push`或`merge`时，大多数情况（有时网络不顺畅可能要手动处理，但次数极少）jenkins会触发自动构建，具体流程如下：

前端流水线流程

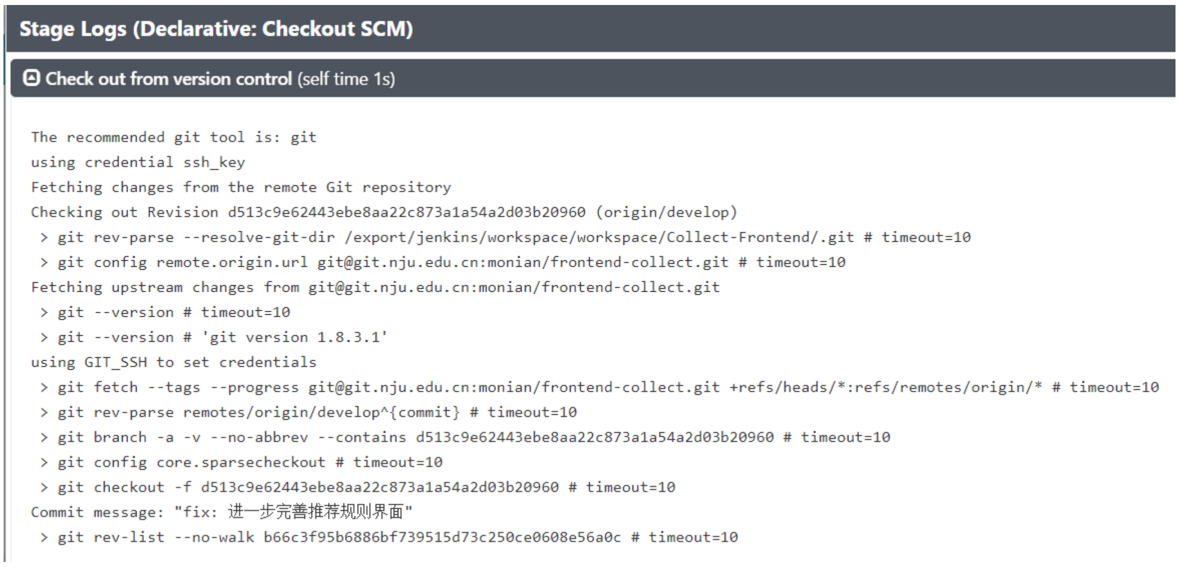
前端项目名称为`Collect-Frontend`，流水线阶段如下：

阶段视图



	Declarative: Checkout SCM	pull	prepare	clean	build	test	deploy	end	Declarative: Post Actions
Average stage times: (Average full run time: ~1min 54s)	3s	2s	855ms	1s	1min 6s	974ms	33s	1s	914ms
#94 Apr 01 22:23 1 commit	1s	2s	790ms	1s	1min 7s	793ms	31s	841ms	901ms

- **SCM阶段**：从相应仓库拉取`Jenkinsfile`，里面存放着流水线脚本

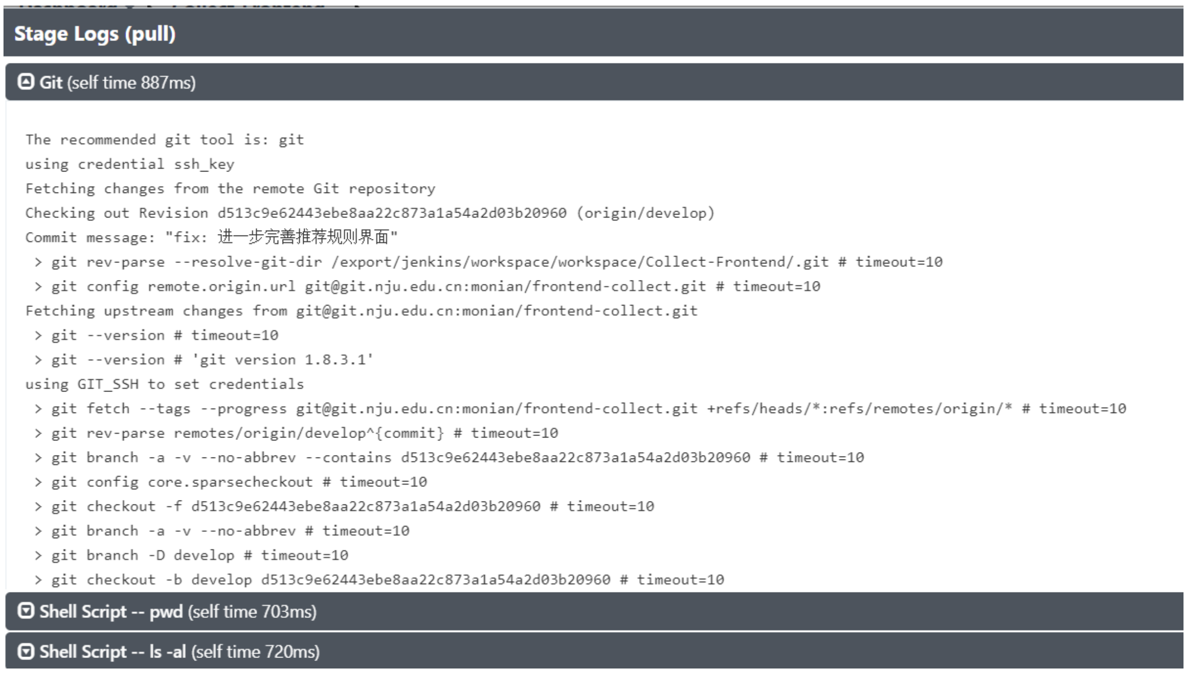


Stage Logs (Declarative: Checkout SCM)

Check out from version control (self time 1s)

```
The recommended git tool is: git
using credential ssh_key
Fetching changes from the remote Git repository
Checking out Revision d513c9e62443ebe8aa22c873a1a54a2d03b20960 (origin/develop)
 > git rev-parse --resolve-git-dir /export/jenkins/workspace/workspace/Collect-Frontend/.git # timeout=10
 > git config remote.origin.url git@git.nju.edu.cn:monian/frontend-collect.git # timeout=10
Fetching upstream changes from git@git.nju.edu.cn:monian/frontend-collect.git
 > git --version # timeout=10
 > git --version # 'git version 1.8.3.1'
using GIT_SSH to set credentials
 > git fetch --tags --progress git@git.nju.edu.cn:monian/frontend-collect.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse remotes/origin/develop^{commit} # timeout=10
 > git branch -a -v --no-abbrev --contains d513c9e62443ebe8aa22c873a1a54a2d03b20960 # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f d513c9e62443ebe8aa22c873a1a54a2d03b20960 # timeout=10
Commit message: "fix: 进一步完善推荐规则界面"
 > git rev-list --no-walk b66c3f95b6886bf739515d73c250ce0608e56a0c # timeout=10
```

- **pull阶段**：从相应仓库中拉取所有代码



Stage Logs (pull)

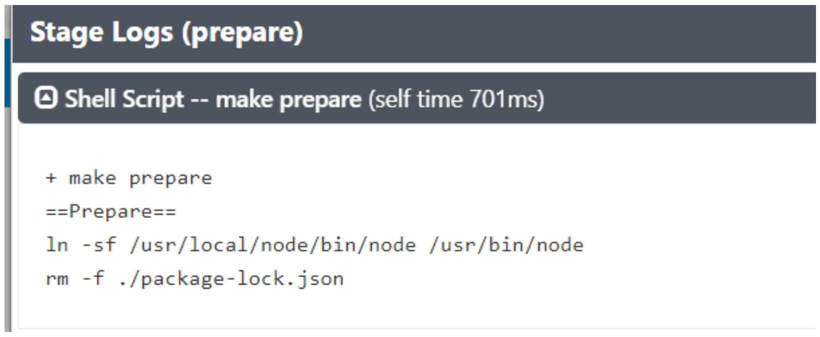
Git (self time 887ms)

```
The recommended git tool is: git
using credential ssh_key
Fetching changes from the remote Git repository
Checking out Revision d513c9e62443ebe8aa22c873a1a54a2d03b20960 (origin/develop)
Commit message: "fix: 进一步完善推荐规则界面"
 > git rev-parse --resolve-git-dir /export/jenkins/workspace/workspace/Collect-Frontend/.git # timeout=10
 > git config remote.origin.url git@git.nju.edu.cn:monian/frontend-collect.git # timeout=10
Fetching upstream changes from git@git.nju.edu.cn:monian/frontend-collect.git
 > git --version # timeout=10
 > git --version # 'git version 1.8.3.1'
using GIT_SSH to set credentials
 > git fetch --tags --progress git@git.nju.edu.cn:monian/frontend-collect.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse remotes/origin/develop^{commit} # timeout=10
 > git branch -a -v --no-abbrev --contains d513c9e62443ebe8aa22c873a1a54a2d03b20960 # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f d513c9e62443ebe8aa22c873a1a54a2d03b20960 # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D develop # timeout=10
 > git checkout -b develop d513c9e62443ebe8aa22c873a1a54a2d03b20960 # timeout=10
```

Shell Script -- pwd (self time 703ms)

Shell Script -- ls -al (self time 720ms)

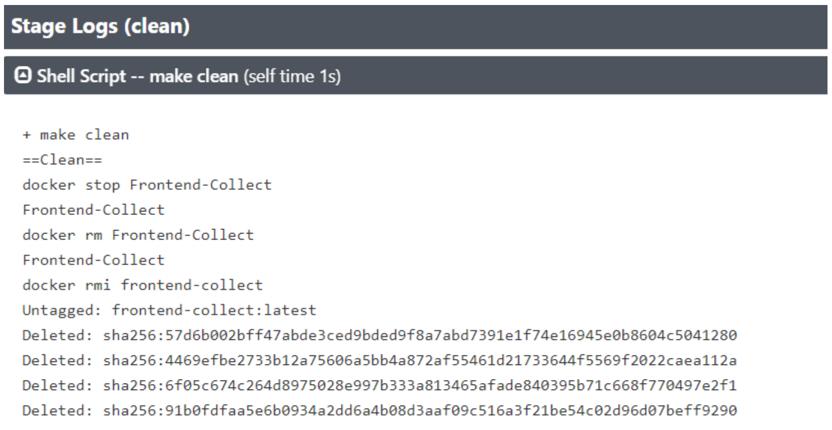
- **prepare阶段**:
    - ln是不配置slave节点时做的一个软连接，使npm命令能顺利使用
    - 顺便删除目录下`package-lock.json`文件，因为有了它会让`npm install`很慢


Stage Logs (prepare)

Shell Script -- make prepare (self time 701ms)

```
+ make prepare
==Prepare==
ln -sf /usr/local/node/bin/node /usr/bin/node
rm -f ./package-lock.json
```

- **clean阶段**：
  - 检测有无名为`Frontend-Collect`的容器，如果有，则停止并删除它；检查有无名为`frontend-collect`的镜像，如果有，则删除它


Stage Logs (clean)

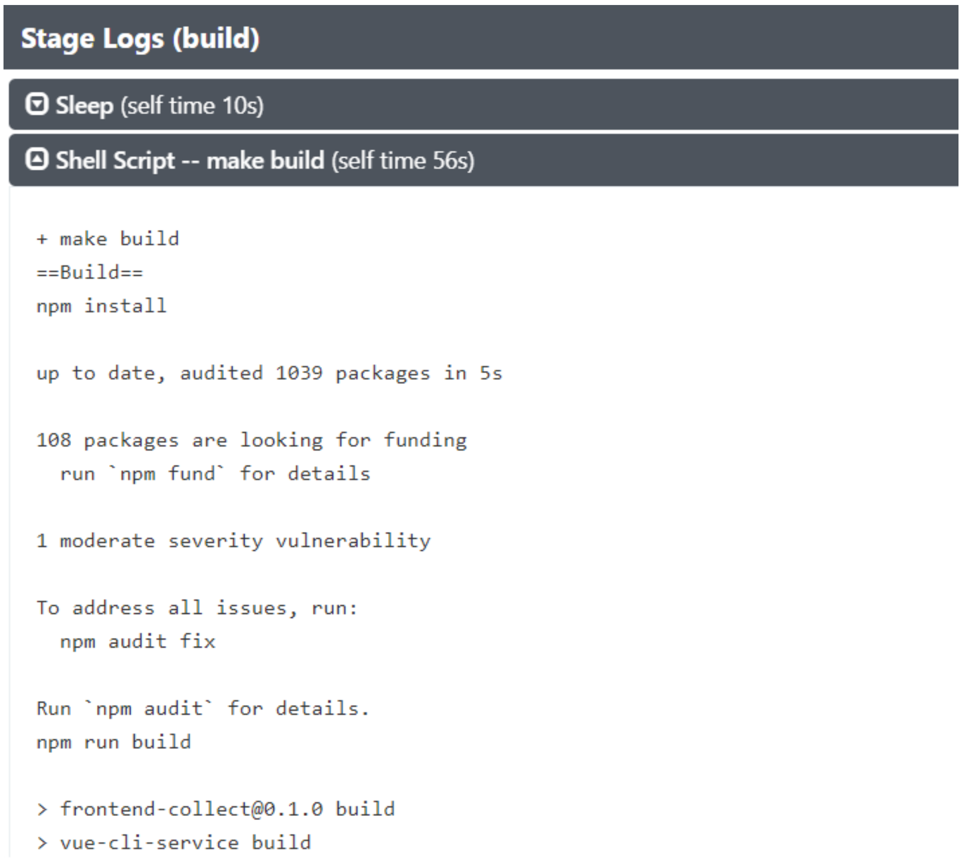
Shell Script -- make clean (self time 1s)

```
+ make clean
==Clean==
docker stop Frontend-Collect
Frontend-Collect
docker rm Frontend-Collect
Frontend-Collect
docker rmi frontend-collect
Untagged: frontend-collect:latest
Deleted: sha256:57d6b002bff47abde3ced9bded9f8a7abd7391e1f74e16945e0b8604c5041280
Deleted: sha256:4469efbe2733b12a75606a5bb4a872af55461d21733644f5569f2022caea112a
Deleted: sha256:6f05c674c264d8975028e997b333a813465afade840395b71c668f770497e2f1
Deleted: sha256:91b0fdfaa5e6b0934a2dd6a4b08d3aaf09c516a3f21be54c02d96d07beff9290
```

（主要实现方式见`Makefile`，大致代码如下）

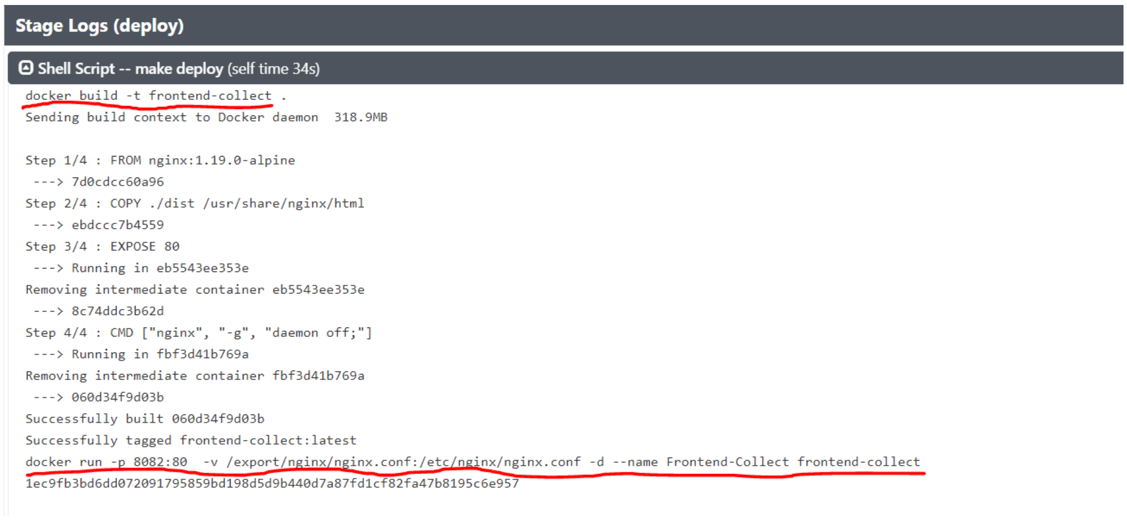
```
ifneq ($(shell docker ps | grep $(CON_NAME)),)
	docker stop $(CON_NAME)
endif
```

- **build阶段**：执行`npm install`和`npm run build`两个命令



```
+ make build
==Build==
npm install

up to date, audited 1039 packages in 5s

108 packages are looking for funding
  run `npm fund` for details

1 moderate severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
npm run build

> frontend-collect@0.1.0 build
> vue-cli-service build
```

- **test阶段**：暂时前端通过使用桩手动测试，所以此阶段不做任何事。但仍然分出一个阶段，以备后续之需。
- **deploy阶段**：根据Dockerfile生成镜像，并且根据镜像启动容器，具体执行以下两个命令：



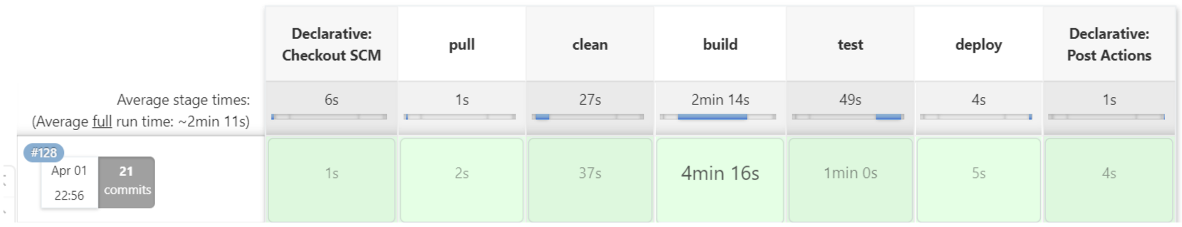
```
docker build -t frontend-collect .
Sending build context to Docker daemon  318.9MB

Step 1/4 : FROM nginx:1.19.0-alpine
 ---> 7d0cdcc60a96
Step 2/4 : COPY ./dist /usr/share/nginx/html
 ---> ebdccc7b4559
Step 3/4 : EXPOSE 80
 ---> Running in eb5543ee353e
Removing intermediate container eb5543ee353e
 ---> 8c74ddc3b62d
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in fbf3d41b769a
Removing intermediate container fbf3d41b769a
 ---> 060d34f9d03b
Successfully built 060d34f9d03b
Successfully tagged frontend-collect:latest
docker run -p 8082:80  -v /export/nginx/nginx.conf:/etc/nginx/nginx.conf -d --name Frontend-Collect frontend-collect
1ec9fb3bd6dd072091795859bd198d5d9b440d7a87fd1cf82fa47b8195c6e957
```

- **end阶段**：现在不做他事。本来留这个阶段的目的是和后端在一台服务器部署时，会出现服务器内存不够，导致build进程被killed，所以要先停掉后端容器（原本在clean阶段中），然后最后在启动后端容器。但是现在前端项目在另外机器上构建，所以此阶段废弃，不做事情，但暂时保留。
- **post Action阶段**：将构建结果推回至gitlab仓库。

后端流水线流程

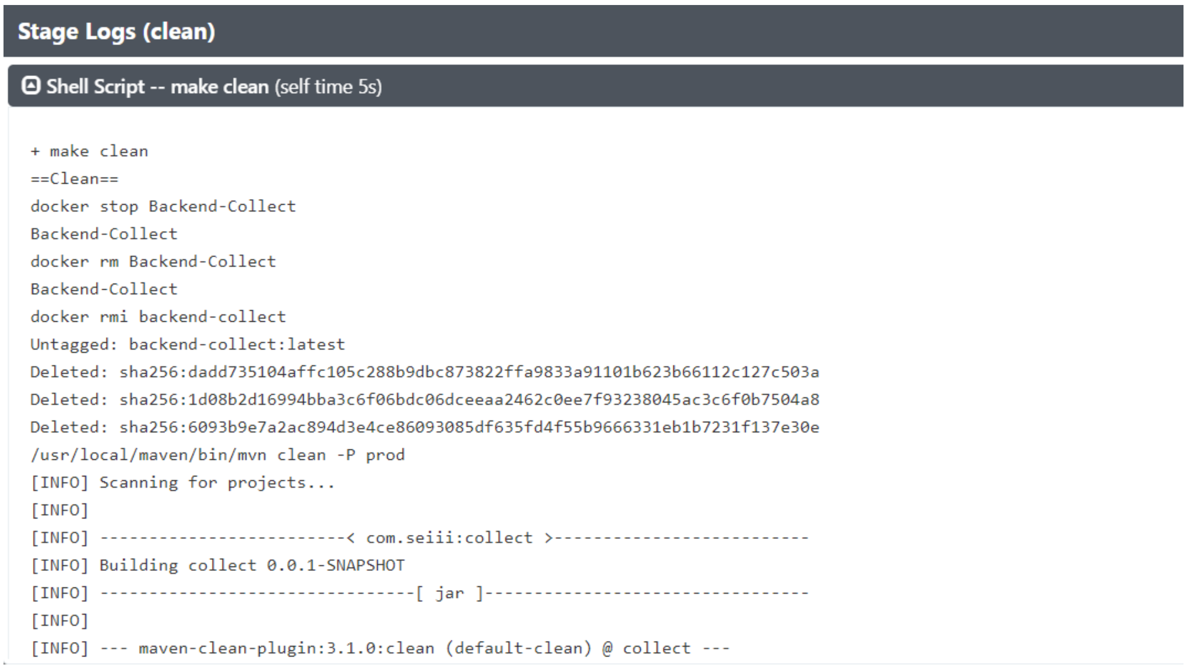
后端项目名称为`Backend-Collect`，流水线阶段如下：

阶段视图



	Declarative: Checkout SCM	pull	clean	build	test	deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 11s)	6s	1s	27s	2min 14s	49s	4s	1s
#128 Apr 01 22:56 21 commits	1s	2s	37s	4min 16s	1min 0s	5s	4s

- **SCM阶段**：同上
- **pull阶段**：同上
- **clean阶段**：基本同上，不过多了一个`mvn clean`

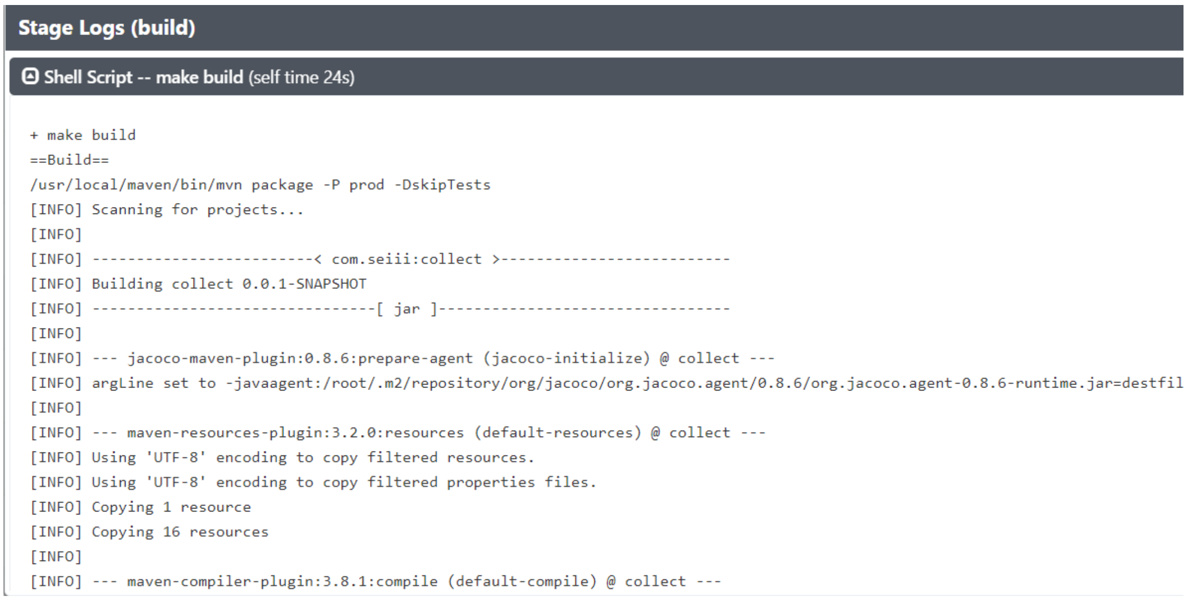


Stage Logs (clean)

Shell Script -- make clean (self time 5s)

```
+ make clean
==Clean==
docker stop Backend-Collect
Backend-Collect
docker rm Backend-Collect
Backend-Collect
docker rmi backend-collect
Untagged: backend-collect:latest
Deleted: sha256:dadd735104affc105c288b9dbc873822ffa9833a91101b623b66112c127c503a
Deleted: sha256:1d08b2d16994bba3c6f06bdc06dceeaa2462c0ee7f93238045ac3c6f0b7504a8
Deleted: sha256:6093b9e7a2ac894d3e4ce86093085df635fd4f55b9666331eb1b7231f137e30e
/usr/local/maven/bin/mvn clean -P prod
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.seiii:collect >------------------------
[INFO] Building collect 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ collect ---
```

- **build阶段**：执行`mvn package -P prod -DskipTests`，这里先跳过测试，后面有单独的`test阶段`。值得一提的是，这里因为jenkins在docker容器中跑，所以得挂载host机的mvn命令，所以此处没有直接用`mvn`

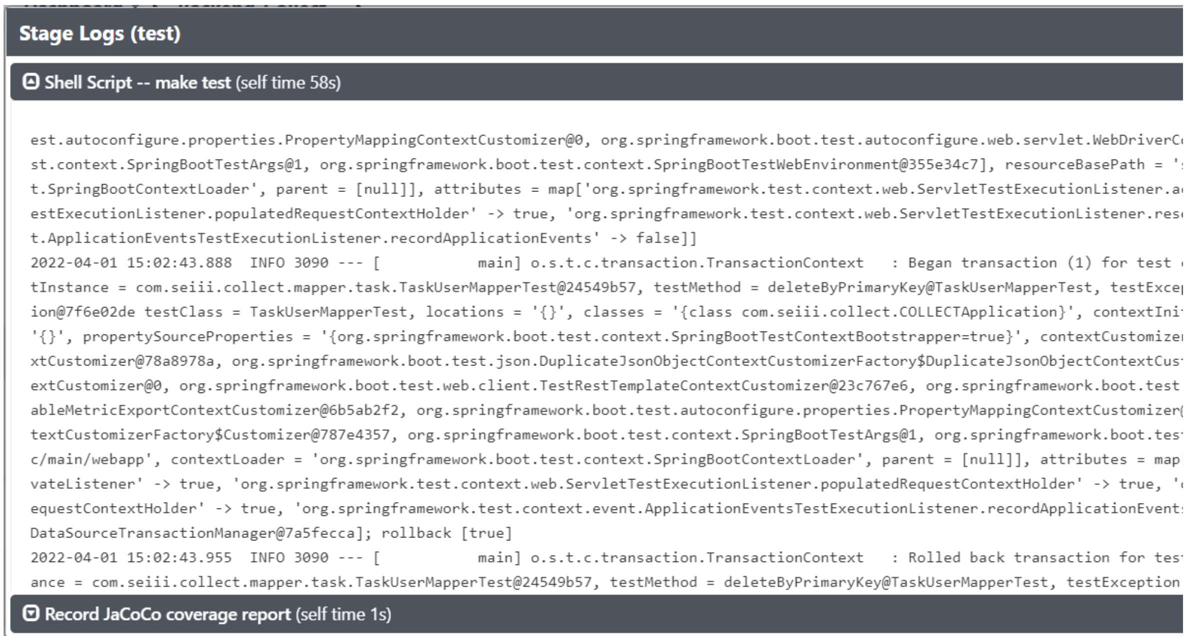


Stage Logs (build)

Shell Script -- make build (self time 24s)

```
+ make build
==Build==
/usr/local/maven/bin/mvn package -P prod -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.seiii:collect >------------------------
[INFO] Building collect 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:prepare-agent (jacoco-initialize) @ collect ---
[INFO] argLine set to -javaagent:/root/.m2/repository/org/jacoco/org.jacoco.agent/0.8.6/org.jacoco.agent-0.8.6-runtime.jar=destfil
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ collect ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 16 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ collect ---
```

- **test阶段**：执行`mvn test`，并用jacoco记录测试覆盖率。

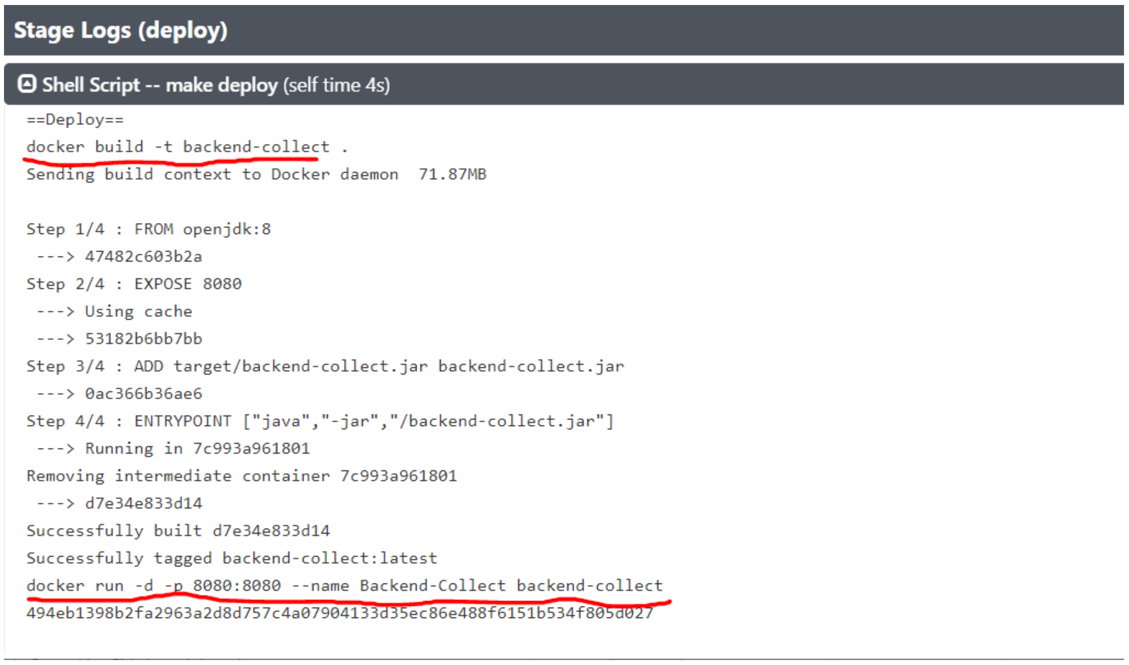

Stage Logs (test)
Shell Script -- make test (self time 58s)
Record JaCoCo coverage report (self time 1s)

```
est.autoconfigure.properties.PropertyMappingContextCustomizer@0, org.springframework.boot.test.autoconfigure.web.servlet.WebDriverC
st.context.SpringBootTestArgs@1, org.springframework.boot.test.context.SpringBootTestWebEnvironment@355e34c7], resourceBasePath = '
t.SpringBootContextLoader', parent = [null]], attributes = map['org.springframework.test.context.web.ServletTestExecutionListener.a
estExecutionListener.populatedRequestContextHolder' -> true, 'org.springframework.test.context.web.ServletTestExecutionListener.res
t.ApplicationEventsTestExecutionListener.recordApplicationEvents' -> false]]
2022-04-01 15:02:43.888  INFO 3090 --- [           main] o.s.t.c.transaction.TransactionContext   : Began transaction (1) for test
tInstance = com.seiii.collect.mapper.task.TaskUserMapperTest@24549b57, testMethod = deleteByPrimaryKey@TaskUserMapperTest, testExce
ion@7f6e02de testClass = TaskUserMapperTest, locations = '{}', classes = '{class com.seiii.collect.COLLECTApplication}', contextIni
'{}', propertySourceProperties = '{org.springframework.boot.test.context.SpringBootTestContextBootstrapper=true}', contextCustomize
xtCustomizer@78a8978a, org.springframework.boot.test.json.DuplicateJsonObjectContextCustomizerFactory$DuplicateJsonObjectContextCus
extCustomizer@0, org.springframework.boot.test.web.client.TestRestTemplateContextCustomizer@23c767e6, org.springframework.boot.test
ableMetricExportContextCustomizer@6b5ab2f2, org.springframework.boot.test.autoconfigure.properties.PropertyMappingContextCustomizer
textCustomizerFactory$Customizer@787e4357, org.springframework.boot.test.context.SpringBootTestArgs@1, org.springframework.boot.tes
c/main/webapp', contextLoader = 'org.springframework.boot.test.context.SpringBootContextLoader', parent = [null]], attributes = map
vateListener' -> true, 'org.springframework.test.context.web.ServletTestExecutionListener.populatedRequestContextHolder' -> true, '
equestContextHolder' -> true, 'org.springframework.test.context.event.ApplicationEventsTestExecutionListener.recordApplicationEvent
DataSourceTransactionManager@7a5fecca]; rollback [true]
2022-04-01 15:02:43.955  INFO 3090 --- [           main] o.s.t.c.transaction.TransactionContext   : Rolled back transaction for tes
ance = com.seiii.collect.mapper.task.TaskUserMapperTest@24549b57, testMethod = deleteByPrimaryKey@TaskUserMapperTest, testException
```

*此处做了一个处理，如果测试失败，虽然会返回`FAIL`的结果，但**仍然会进行deploy阶段**，原因是有时候因为数据库问题，测试部分没有及时更新，但是此时并不代表程序有问题，应该仍然部署并且通知用户，测试失败。*

```
stage('test') {
    steps {
                    script{
            try{
                sh 'make test'
                                jacoco()
            }catch(err){
                sh 'make deploy'
                throw err
            }
        }
    }
}
```

- **deploy阶段**：基本同上

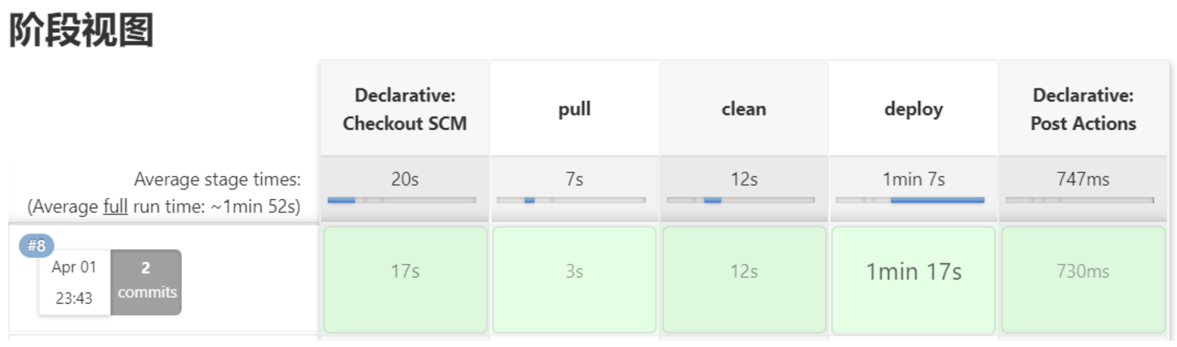


```
==Deploy==
docker build -t backend-collect .
Sending build context to Docker daemon  71.87MB

Step 1/4 : FROM openjdk:8
 ---> 47482c603b2a
Step 2/4 : EXPOSE 8080
 ---> Using cache
 ---> 53182b6bb7bb
Step 3/4 : ADD target/backend-collect.jar backend-collect.jar
 ---> 0ac366b36ae6
Step 4/4 : ENTRYPOINT ["java","-jar","/backend-collect.jar"]
 ---> Running in 7c993a961801
Removing intermediate container 7c993a961801
 ---> d7e34e833d14
Successfully built d7e34e833d14
Successfully tagged backend-collect:latest
docker run -d -p 8080:8080 --name Backend-Collect backend-collect
494eb1398b2fa2963a2d8d757c4a07904133d35ec86e488f6151b534f805d027
```

- **post Action阶段**：将构建结果推回至gitlab仓库。

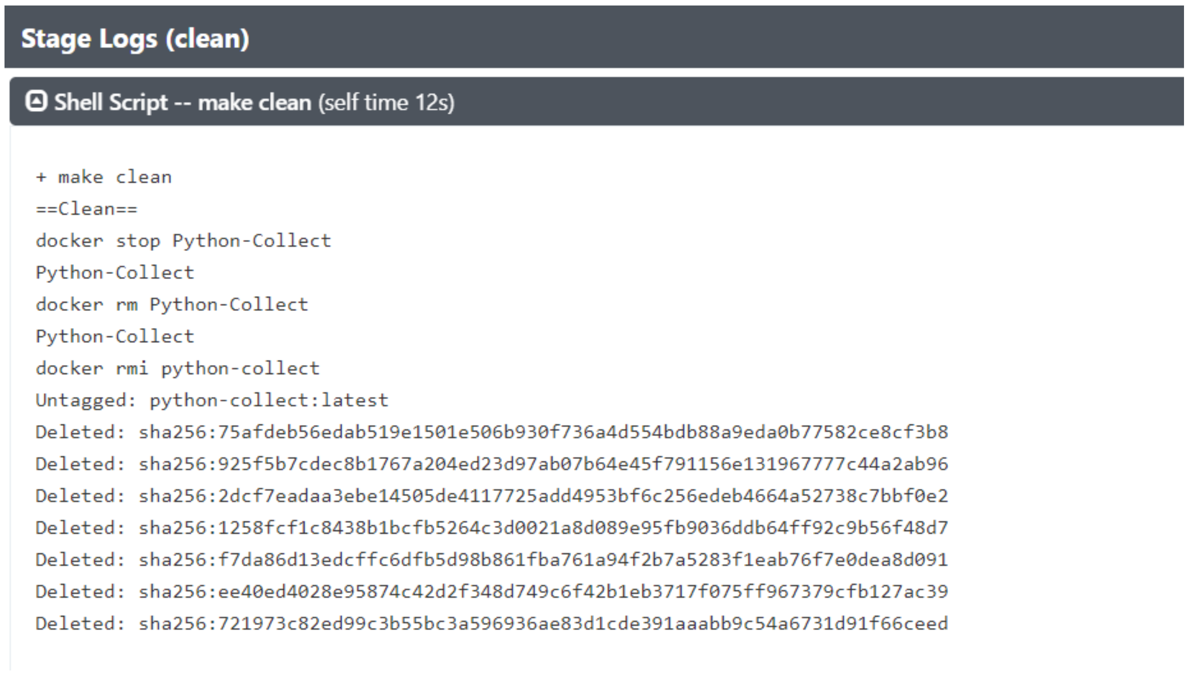
python流水线流程

后端项目名称为`Python-Collect`，流水线阶段如下：


阶段视图

	Declarative: Checkout SCM	pull	clean	deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~1min 52s)	20s	7s	12s	1min 7s	747ms
#8 Apr 01 23:43 2 commits	17s	3s	12s	1min 17s	730ms

- **SCM阶段**：同上
- **pull阶段**：同上
- **clean阶段**：同上


Stage Logs (clean)

Shell Script -- make clean (self time 12s)

```
+ make clean
==Clean==
docker stop Python-Collect
Python-Collect
docker rm Python-Collect
Python-Collect
docker rmi python-collect
Untagged: python-collect:latest
Deleted: sha256:75afdeb56edab519e1501e506b930f736a4d554bdb88a9eda0b77582ce8cf3b8
Deleted: sha256:925f5b7cdec8b1767a204ed23d97ab07b64e45f791156e131967777c44a2ab96
Deleted: sha256:2dcf7eadaa3ebe14505de4117725add4953bf6c256edeb4664a52738c7bbf0e2
Deleted: sha256:1258fcf1c8438b1bcfb5264c3d0021a8d089e95fb9036ddb64ff92c9b56f48d7
Deleted: sha256:f7da86d13edcffc6dfb5d98b861fba761a94f2b7a5283f1eab76f7e0dea8d091
Deleted: sha256:ee40ed4028e95874c42d2f348d749c6f42b1eb3717f075ff967379cfb127ac39
Deleted: sha256:721973c82ed99c3b55bc3a596936ae83d1cde391aaabb9c54a6731d91f66ceed
```

- **deploy阶段**：同上

```
deploy:
	@echo "==Deploy=="
	docker build -t $(IMG_NAME) .
	docker run -p 5000:5000 -v /pythonProject:/pythonProject -d --name $(CON_NAME) $(IMG_NAME)
```

- **post Action阶段**：同上

补充说明

详见各个项目中`Jenkinsfile`和`Makefile`。通过将流水线分成一个个阶段，可以更好地掌握出错地点。