小白对jenkins运维的使用有点简单的想法，这里开个记录贴记录下。

2019-09-26

由于未找到jenkins构建失败后执行其他脚本的插件，也暂时没有使用其他运维工具。所以想自己写一个shell脚本，一是方便其他人使用，二是可以失败后回滚。

本文使用gitlab webhook触发jenkins构建，jenkins执行shell命令---》mvn或者npm打包---》打包成docker镜像---》发布---》发布成功后用jenkins的git publisher插件push一个gitlab tag作为备份。

如果构建失败，则checkout最新的tag代码---》重新打包---》打包成docker镜像---》发布运行。

最后发现，结果是可以正常构建，构建失败也可以回滚。但是jenkins的发布状态一直是success，无法知道是正常构建还是回滚构建成功。

也想过，回滚的时候发布邮件或者短信提醒。或者修改jenkins源码，增加构建状态roolback。

也想用这个shell脚本做个简单的jenkins------easy-ci

通过java调用linux命令，写几个前端页面。页面上填写shell脚本的参数，点击发布，查看日志，查看容器，停止容器，重启容器等操作。

但是gitlab的webhook如何触发java自动构建是个问题。

后来发现有个gitlab runner可以触发命令，后期可以研究下。

此处先贴出shell脚本：清除旧的构建、构建、回滚。

```
#镜像名称
IMAGE_NAME=$1
#容器名称
CONTAINER_NAME=$2
#端口映射(9500:9000)
PORT=$3
#语言类型(java or vue)
LANG=$4
#日志路径夹
logs_path="logs/"
#日志名称
log_name=$5
#日志路径
log_path=$logs_path$log_name

function deploy(){

    echo "=====================删除旧容器====================="
    t=`sudo docker ps -a | grep $1|awk '{print  $1}'|sed 's/%//g'`;
    if [ $t ];
    then
    sudo docker stop  $t
    echo "停止容器成功"
    sudo docker rm $t
    echo "删除容器成功"
    fi

    echo "=====================删除无用镜像====================="
    docker images|grep none|awk '{print $3}'|xargs docker rmi

    echo "================删除多余tag，保留最新三个==============="
    tagnum=`git tag | wc -l`;
    tag=(`git tag`)
    a=0;
    echo "当前tag数目为：" $tagnum "个"
    until [ $tagnum -lt 3 ]
    do
       echo $a 准备删除tag：${tag[$a]}
       git tag -d ${tag[$a]}
       echo "=====================本地删除成功====================="
       git push $2 :refs/tags/${tag[$a]}
       echo "=====================远程删除成功====================="
       a=`expr $a + 1`
       tagnum=`expr $tagnum - 1`
    done

}

function build(){
```

```
    if [ $1 = java ];then

        echo "=====================jar包打包开始====================="
        mvn package  -DskipTests
        echo "=====================jar包打包完成====================="

        echo "===================docker镜像打包开始"==================="
        mvn dockerfile:build
        echo "===================docker镜像打包完成"==================="

    else

        echo "=====================vue打包开始====================="
        sudo /usr/local/bin/npm install
        sudo /usr/local/bin/npm run build
        echo "=====================vue打包完成====================="

        echo "===================docker镜像打包开始"==================="
        sudo docker build -t $2 .
        echo "===================docker镜像打包完成"==================="

    fi

    echo '================开始推送镜像到docker私服=================='
    sudo docker push $2
    echo '======================推送镜像完成======================'

    echo '======================删除旧的构建======================'
#    deploy $2 $4 >> $5 2>&1
    echo '====================旧的构建删除成功==================='

    echo '======================容器运行开始======================'
    sudo docker run -d --name $4 -p:$3 $2
    sudo docker ps | grep $2
    echo '======================容器运行完成======================'
    #port=($3//:/ )
    #echo 访问路径为：`ifconfig ens192 | grep "inet" | awk '{ print $2}' | awk -F: '{print $1}' | grep 192`:${port[0]}

}

function rollback(){

    echo "=================================开始执行回滚================================="

    echo '======================删除旧的构建======================'
        deploy $2 $4 >> $5 2>&1
        echo '====================旧的构建删除成功==================='

    tag=(`git tag`)
    echo "检测到的git tag版本为：" ${tag[*]}

    maxtag=(${tag[0]//_/ })
    maxnum=${maxtag[1]}

    for i in ${!tag[*]};do
        nexttag=(${tag[$i]//_/ })
        nextnum=${nexttag[1]}

        if [[ $maxnum -lt $nextnum ]];then
            maxnum=$nextnum
        fi
    done
    echo "检测到最新的git tag版本为：" rc_$maxnum
    echo "拉取上一次构建成功的tag代码"

    if [ `git checkout rc_$maxnum` ];then
        echo "拉取代码失败，请手动选择代码构建"
    else
        echo "拉取代码成功"
    fi
    build $1 $2 $3 $4
}



mkdir logs
echo '创建日志文件成功' >> $log_path 2>&1
echo `date` >> $log_path 2>&1
echo "开始构建" >> $log_path 2>&1
build $LANG $IMAGE_NAME $PORT $CONTAINER_NAME $log_path >> $log_path 2>&1
resule=`sudo docker ps | grep pig-ui|awk '{print  $1}'|sed 's/%//g'`
if [ $resule != '' ];then
    echo "构建成功"
else
    echo "回滚"
    rollback $LANG $IMAGE_NAME $PORT $CONTAINER_NAME $log_path >> $log_path 2>&1
fi
```

2019-09-27

昨天初步写好了以下几个脚本

jenkins.sh jenkins调用传参

auto_build.sh 自动构建

auto_destroy.sh 清理旧的构建

auto_rollback.sh 自动回滚

sendmail.sh 构建成功或者回滚成功通过sendmail发送邮件附加构建日志

现在依旧用的jenkins的git拉取代码、git publisher推送tag、git webhook自动触发构建。

在脚本中加入git clone git@url可以解决git拉取代码问题。 git tag **可以解决推送tag问题。

但是如何调用gitlab的webhook，使提交代码到gitlab，不依靠jenkins自动触发构建是个问题？

可以考虑gitlab的runner？不过这个使用也要写一点代码。

不满足我傻瓜式ci/cd的初衷，用户只需要输入几个参数就ok。

这里先贴上分离后的脚本。

jenkins.sh

```
#镜像名称
IMAGE_NAME=$1
#容器名称
CONTAINER_NAME=$2
#端口映射(9500:9000)
PORT_MAPPING=$3
#语言类型(java or vue)
PROJECT_LANG=$4
#日志名称
LOG_NAME=$5
#日志路径
LOG_PATH=logs/$LOG_NAME
#收件邮箱
TO_LIST=$6
#是否回滚成功
IS_ROLLBACK=0

mkdir logs/
echo '创建日志文件成功' >> $LOG_PATH
bash /home/jenkins/auto_build.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME | tee ./$LOG_PATH
```

1：PROJECT_LANG 判断项目的语言，这里只做了java和vue的判断，分别执行各自语言的打包和bulid docker镜像

2：IMAGE_NAME 镜像名称，用于docker打包时指定生成image的名称，因为这里我使用了docker私服，所有加入了docker push操作。 镜像名称也用于后续的docker ps | grep **查找容器、删除容器等等操作

3：PORT_MAPPING 端口映射，用户指定docker run 时的-p 端口映射。也用于构建成功时通过（//：/ ）等操作提取宿主机端口，生成服务的访问url

4：CONTAINER_NAME 容器名称，docker run --name指定的容器名称，也用于后续的git push CONTAINER_NAME ：refs/tags/rc_**远程删除tag，这里一般取值为gitlab中项目的名称。

5：LOG_PATH 日志的路径。用于sendmail时附带日志

6：IS_ROLLBACK 是否回滚：jenkins.sh调用时为0表示正常构建，rollback调用时为1表示回滚构建。以此判断发送邮件的标题。

7：TO_LIST 收件人邮箱，多个以 “，” 分割

8：LOG_NAME 日志名称，由于这里的日志名称我是在jenkins中传递的 rc_$BUILD_NUMBER ，$BUILD_NUMBER为jenkins构建的当前次数。把次数分离开加入到邮件标题， ***项目第***次构建

9：| tee ./$LOG_PATH 通过管道和tee 实现控制台输出日志的同时将日志保存到日志文件中。（这里的控制台输出日志指得是jenkins构建日志，如果直接用>>重定向的话，jenkins构建日志将不会显示脚本的日志输出）。

auto_build.sh

```
#项目语言
PROJECT_LANG=$1
#镜像名称
IMAGE_NAME=$2
#端口映射
PORT_MAPPING=$3
```

```
#容器名称
CONTAINER_NAME=$4
#日志路径
LOG_PATH=$5
#是否回滚成功
IS_ROLLBACK=$6
#收件邮箱
TO_LIST=$7
#日志名称
LOG_NAME=$8
#构建次数
NUMBER=(${LOG_NAME//_/ })

echo "`date`第${NUMBER[1]}次构建"
echo '==========删除旧的构建=========='
bash /home/jenkins/auto_destroy.sh $IMAGE_NAME $CONTAINER_NAME
echo '==========删除成功=========='

if [[ $PROJECT_LANG = java ]];then

    echo '==========jar包打包开始=========='
    mvn package  -DskipTests
    if [[ `echo $?` -eq 0 ]];then
        echo '==========jar包打包完成=========='
    else
        echo '==========jar包打包失败=========='
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi

    echo '==========docker打包开始=========='
    mvn dockerfile:build
    if [[ `echo $?` -eq 0 ]];then
        echo '==========docker打包完成=========='
    else
        echo '==========docker打包失败=========='
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi

else

    echo '==========vue环境安装开始=========='
    sudo /usr/local/bin/npm install
    if [[ `echo $?` -eq 0 ]];then
        echo '==========vue环境安装完成=========='
    else
        echo '==========vue环境安装失败=========='
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi

    echo '==========vue打包开始=========='
    sudo /usr/local/bin/npm run build
    if [[ `echo $?` -eq 0 ]];then
        echo '==========vue打包完成=========='
    else
        echo '==========vue打包失败=========='
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi

    echo '==========docker打包开始=========='
    sudo docker build -t $2 .
    if [[ `echo $?` -eq 0 ]];then
        echo '==========docker打包完成=========='
    else
        echo '==========docker打包失败=========='
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi

fi

echo '==========容器运行开始=========='
sudo docker run -d --name $CONTAINER_NAME -p:$PORT_MAPPING $IMAGE_NAME
if [[ `echo $?` -eq 0 ]];then

    resule=`sudo docker ps | grep $IMAGE_NAME|awk '{print  $1}'|sed 's/%//g'`
    if [[ $resule != '' ]];then
        sudo docker ps | grep $IMAGE_NAME
        echo '==========容器运行完成=========='
        echo '==========开始推送镜像到docker私服=========='
        sudo docker push $IMAGE_NAME
        echo '==========推送镜像完成=========='
        port=(${PORT_MAPPING//:/ })
        echo 访问路径为：http://`ifconfig ens192 | grep "inet" | awk '{ print $2}' | awk -F: '{print $1}' | grep 192`:${port[0]}
        if [[ $IS_ROLLBACK -eq 0 ]];then
            echo "构建成功"
            bash /home/jenkins/sendmail.sh $TO_LIST 成功:$CONTAINER_NAME第${NUMBER[1]}次构建 $LOG_PATH
            exit 0
        else
            echo "回滚成功"
            bash /home/jenkins/sendmail.sh $TO_LIST 回滚:$CONTAINER_NAME第${NUMBER[1]}次构建 $LOG_PATH
```

```
            exit 0
        fi
        exit 0
    else
        echo "执行回滚"
        bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
        exit 0
    fi
    exit 0
else
    echo "执行回滚"
    bash /home/jenkins/auto_rollback.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
    exit 0

fi
exit 0
```

用于执行构建过程，清理旧的构建，以及构建失败时调用回滚、发送构建邮件

auto_destroy.sh

```
#镜像名称
IMAGE_NAME=$1
#容器名称
CONTAINER_NAME=$2

echo "=====================删除旧容器====================="
t=`sudo docker ps -a | grep $IMAGE_NAME|awk '{print  $1}'|sed 's/%//g'`;
if [[ $t ]];
then
sudo docker stop  $t
echo "停止容器成功"
sudo docker rm $t
echo "删除容器成功"
fi

echo "=====================删除无用镜像====================="
docker images|grep none|awk '{print $3}'|xargs docker rmi

echo "================删除多余tag，保留最新三个==============="
tagnum=`git tag | wc -l`;
tag=(`git tag`)
a=0
echo "当前tag数目为：" $tagnum "个"
until [[ $tagnum -lt 3 ]]
do
   echo $a 准备删除tag：${tag[$a]}
   git tag -d ${tag[$a]}
   echo "=====================本地删除成功====================="
   git push $CONTAINER_NAME :refs/tags/${tag[$a]}
   echo "=====================远程删除成功====================="
   a=`expr $a + 1`
   tagnum=`expr $tagnum - 1`
done
exit 0
```

用于清除没有tag的镜像、删除旧的容器、删除多余的tag

auto_rollback.sh

```
#项目语言
PROJECT_LANG=$1
#镜像名称
IMAGE_NAME=$2
#端口映射
PORT_MAPPING=$3
#容器名称
CONTAINER_NAME=$4
#日志路径
LOG_PATH=$5
#是否回滚成功
IS_ROLLBACK=1
#收件邮箱
TO_LIST=$7
#日志名称
LOG_NAME=$8
#构建次数
NUMBER=(${LOG_NAME//_/ })


echo "=====================开始执行回滚====================="

echo '=====================删除旧的构建====================='
bash /home/jenkins/auto_destroy.sh $IMAGE_NAME $CONTAINER_NAME
echo '=====================旧的构建删除成功=================='

tag=(`git tag`)
echo "检测到的git tag版本为：" ${tag[*]}

maxtag=(${tag[0]//_/ })
maxnum=${maxtag[1]}
```

```
for i in ${!tag[*]};do
    nexttag=(${tag[$i]//_/ })
    nextnum=${nexttag[1]}

    if [[ $maxnum -lt $nextnum ]];then
        maxnum=$nextnum
    fi
done
echo "检测到最新的git tag版本为：" rc_$maxnum
echo "拉取上一次构建成功的tag代码"

if [[ `git checkout rc_$maxnum` ]];then
    echo "拉取代码失败，请手动选择代码构建"
else
    echo "拉取代码成功"
    bash /home/jenkins/auto_build.sh $PROJECT_LANG $IMAGE_NAME $PORT_MAPPING $CONTAINER_NAME $LOG_PATH $IS_ROLLBACK $TO_LIST $LOG_NAME
    exit 0
fi
exit 0
```

正常构建失败后，拉取最新的tag分支代码。重新执行构建。

sendmail.sh

```
#收件邮箱列表
TO_LIST=$1
#邮件标题
MAIL_TITLE=$2
#附件地址
LOG_PATH=$3

fromAdd="=?UTF-8?B?`echo $MAIL_TITLE | base64`?="
tolist="$TO_LIST"
cclist=""
subject="=?UTF-8?B?`echo $MAIL_TITLE | base64`?="
attach="$LOG_PATH"
data=`cat $attach`
(
echo "From: $fromAdd"
echo "To: $tolist"
echo "Cc: $cclist"
echo "Subject: $subject"
echo "MIME-Version: 1.0"
echo 'Content-Type: multipart/mixed; boundary="GvXjxJ+pjyke8COw"'
#echo "Content-Disposition: inline"
echo
echo "--GvXjxJ+pjyke8COw"
echo "Content-Type: text/html; charset=US-ASCII"
echo "Content-Disposition: inline"
echo
echo "<h1>Please check the attachment log.</h1>"
echo
echo "--GvXjxJ+pjyke8COw"
echo "Content-Type: text/plain; charset=US-ASCII;"
echo "Content-Disposition: attachment;filename="build.log""
echo
echo "$data"
echo
echo "--GvXjxJ+pjyke8COw"
) | /usr/lib/sendmail -t
```

用于发送邮件

jenkins执行shell脚本：bash /home/jenkins/jenkins.sh IMAGE_NAME CONTAINER_NAME PORT_MAPPING PROJECT_LANG LOG_NAME TO_LIST

这里可以不用jenkins的Editable Email Notification插件

2019-10-21

见https://www.cnblogs.com/jxd283465/p/11703557.html

EasyCi系统开发的目的是免去远程发布的免密登陆、拉取gitlab代码的认证、手动添加gitlab hook、查看gitlab中该项目的git地址等等多余的操作。这些操作均有后台自动完成，系统提供运行环境一键安装脚本、自动化安装部署本系统、开箱即用，只需要几个参数即可实现项目的远程构建，暂时只支持vue和java项目的构建。

Easyci系统采用B/S架构，后端采用springboot框架、前端采用Vue的element ui、数据库采用mysql、运行工具为shell脚本、采用websocket进行实时日志传输。

由于系统由本人独立开发，对前端开发不是很擅长，页面比较简单，只为实现基本功能，后续会对功能和页面进行优化。

工具链

EasyCI：easyci系统后台，调用shell脚本、通过接口将结果返回前端

EasyCI-UI：easyci系统前台，vue。调用后台接口展示数据

Gitlab：代码托管工具，通过gitlab hook触发easyci持续集成交付

shell脚本：linux脚本

Docker私服：私有docker镜像仓库，用于远程构建

MySQL：easyci系统数据库

流程

1.根据安装文档安装系统，启动系统

2.访问系统首先需要验证gitlab：gitlab的url、用户名和密码。用于选择项目构建、服务器拉取代码验证、自动添加hook，免去手动操作。

3.添加远程发布服务器：服务器ip、端口、用户名、密码。后台自动完成服务器间的免密登录，用于项目远程发布、查看服务器容器以及容器的各种操作。

4.部署：选择项目url、输入docker容器端口映射关系、选择项目类型、输入收件人邮箱、选择部署服务器，只需要输入两项内容即可完成部署。

5.点击部署，会弹出窗口显示系统部署日志。

6.部署会自动添加easyci的hook接口url到该部署项目的hook中，用于自动触发构建。

7.部署完成自动发送邮件给收件人邮箱，显示构建结果、构建时间、构建项目、构建日志。

8.后续开发人员提交代码到该项目的gitlab，会触发gitlab的hook调用easyci接口，查询之前部署的信息进行自动部署，实现持续集成。

9.页面展示easyci本机正在运行的容器和添加的远程服务器正在运行的容器，支持数据自动刷新与关闭

10.可以选择添加的远程服务器，查看该服务器的容器列表

11.容器操作：可点击启动 、停止、重启、销毁在页面对服务器中运行的容器进行操作，即docker start|stop|restart|rm 容器名称

12..容器实时日志：点击日志，可以查看该容器的实时日志，即docker logs -f 容器名称

安装教程

https://www.cnblogs.com/jxd283465/p/11703557.html