

这篇文章是我的云计算技术的大作业实验报告，搭建了一个基于Jenkins的Devops自动化CI/CD流程。主要参考资料是这个视频，[【DevOps教程】DevOps最新教程2022版 目前最好的DevOps课程 从入门到进阶 DevOps实践 DevOps运维 零基础入学 手把手教会哔哩哔哩bilibili](#)，只要跟着来是可以顺利搭建好这个框架的。

但这篇博文除了视频内容的具体实践之外，也展现了将项目部署到华为云服务器上的具体操作。同时还有一个很重要的内容是解决了Jenkins使用 docker.sock 映射宿主机Docker时Docker报错：**docker: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' not found (required by docker)**，这个错误网络上没有找到相应的解决办法，也花了很久时间才解决。

这个项目的博客源码以及Jenkinsfile都已上传到我的Github仓库[No-drink/blog_\(github.com\)](#)中，希望可以给到一些帮助。

Devops简介

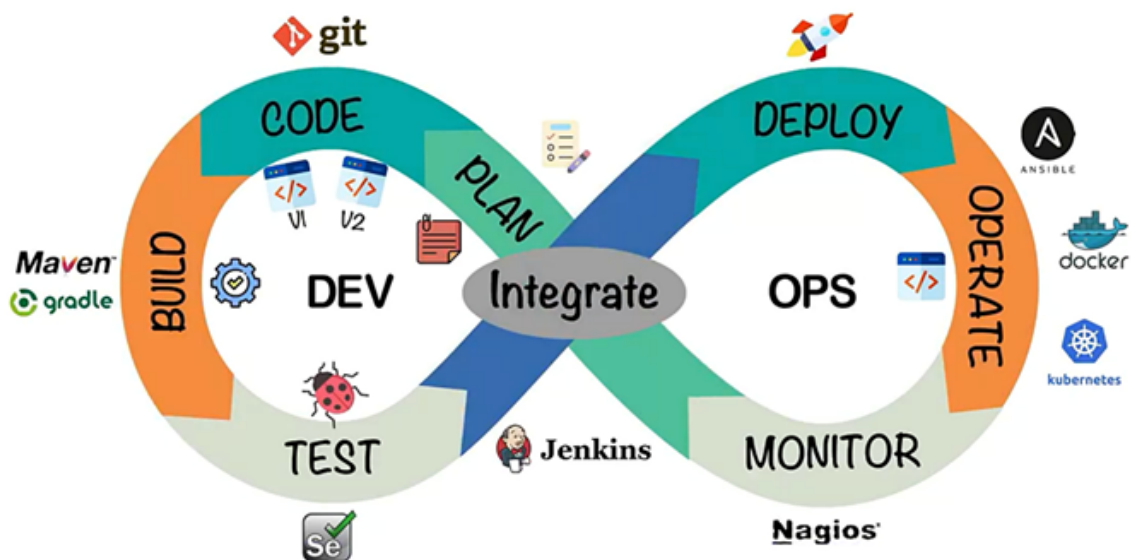
基于现在的互联网现状，更推崇敏捷式开发，这样就导致项目的迭代速度更快，但是由于开发团队与运维团队的沟通问题，会导致新版本上线的时间成本很高。这又违背的敏捷式开发的最初的目的。通过DevOps可以有效解决这个问题。

DevOps，字面意思是Development & Operations的缩写，也就是开发&运维，是一个不断提高效率并且持续不断工作的过程。通过这种方式可以让公司能够更快地应对更新和市场发展变化，开发可以快速交付，部署也更加稳定。其核心就在于简化Dev和Ops团队之间的流程，使整体软件开发过程更快速。

整体的软件开发流程包括：

- PLAN：开发团队根据客户的目标制定开发计划
- CODE：根据PLAN开始编码过程，需要将不同版本的代码存储在一个库中。
- BUILD：编码完成后，需要将代码构建并且运行。
- TEST：成功构建项目后，需要测试代码是否存在BUG或错误。
- DEPLOY：代码经过手动测试和自动化测试后，认定代码已经准备好部署并且交给运维团队。
- OPERATE：运维团队将代码部署到生产环境中。
- MONITOR：项目部署上线后，需要持续的监控产品。
- INTEGRATE：然后将监控阶段收到的反馈发送回PLAN阶段，整体反复的流程就是DevOps的核心，即持续集成、持续部署。

为了保证整体流程可以高效的完成，各个阶段都有比较常见的工具，如下图：



最终可以给DevOps下一个定义：DevOps 强调的是高效组织团队之间如何通过自动化的工具协作和沟通来完成软件的生命周期管理，从而更快、更频繁地交付更稳定的软件。通过自动化的工具协作和沟通来完成软件的生命周期管理。

系统设计与实现

系统架构设计

使用三台主机。一台Windows11系统进行代码编写及发布。其余两台为Ubuntu系统虚拟机。IP地址为192.168.182.129的虚拟机1用作gitlab仓库，Harbor仓库并安装Jenkins进行项目部署。IP地址为192.168.182.128的虚拟机2则作为服务器用来进行项目的部署和运行。另外还有一台华为云服务器提供MySQL服务（起初希望项目部署到此但由于Harbor仓库没有公网IP故只好作罢）。

系统技术选型与相关工具

本实验所用工具如下表：

	工具	用途	安装位置
Code	Git	用于提交业务代码或克隆业务代码仓库	192.168.182.129/Windows11
	Gitlab	用于存储业务代务	192.168.182.129: 8929
Build	Maven	用于编译业务代务	192.168.182.129/Windows11
Operate	Harbor	用于存储业务代码构建的容器镜像存储	192.168.182.129: 80
	Docker	用于构建容器镜像，部署项目	192.168.182.129/192.168.182.128
Integrate	Jenkins	用于利用插件完成业务代码编译、构建、推送至Harbor容器镜像仓库及项目部署	192.168.182.129: 8080

方案实施过程与实现

相关工具安装：

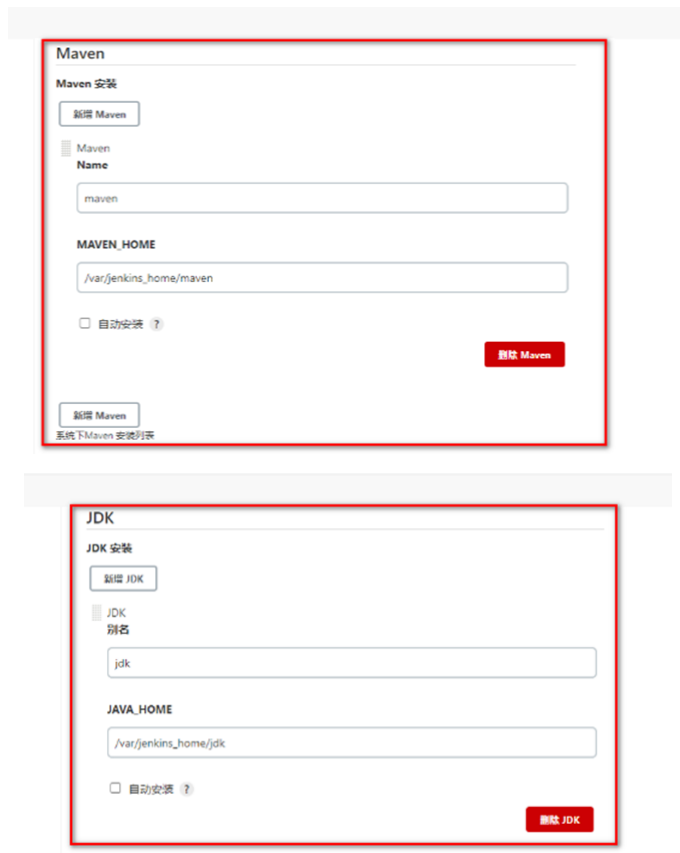
使用VMware安装两台Ubuntu主机，网络连接方式使用NAT模式，查看IP地址留以备用。

在虚拟机1上安装Git, Docker, Docker-compose, 虚拟机2安装Docker即可。**需要注意的是虚拟机1的Docker版本必须为20.10.12及以下！**原因见总结部分的遇到问题及解决。

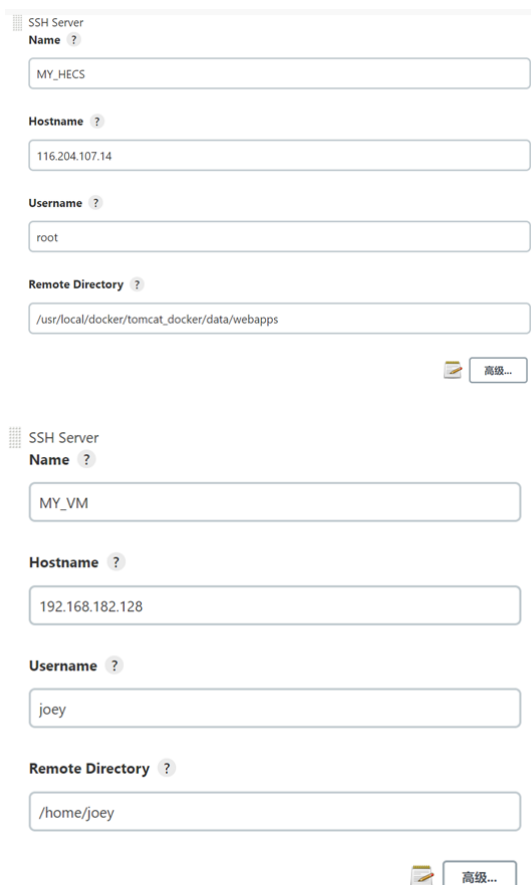
在虚拟机1使用Docker拉取Gitlab, Jenkins镜像。为了方便管理容器内部的映射关系，使用docker-compose的方式运行容器，映射Jenkins的 `var/jenkins_home/` 的文件夹到虚拟机1的 `/usr/local/docker/Jenkins_docker/data` 目录，方便在后续操作过程中查看Jenkins的workspace文件夹，并安装初始化推荐安装插件。在Gitlab中新建公开仓库blog作为项目的代码仓库。

Jenkins的基本配置

(1) 在Jenkins中配置JDK和Maven，以使Jenkins可以使用Maven构建代码。
将maven和JDK安装包通过数据卷映射到jenkins的jenkins_home文件夹中并解压，在Jenkins的操作界面设置添加maven和JDK设置。



(2) 配置Publish Over SSH插件



(3) Jenkins容器使用宿主机Docker

构建镜像和发布镜像到harbor都需要使用到docker命令。而在Jenkins容器内部安装Docker官方推荐直接采用宿主机带的Docker即可。

设置 docker.sock 权限:

```
sudo chown root:root /var/run/docker.sock
sudo chmod o+rw /var/run/docker.sock
```

在docker-compose.yml中添加数据卷:

```
-/usr/bin/docker:/usr/bin/docker
- /var/run/docker.sock:/var/run/docker.sock
- /etc/docker/daemon.json:/etc/docker/daemon.json
```

docker-compose up -d即可使用宿主机Docker

Harbor安装及配置

将Harbor安装包解压, 编辑harbor.yml文件, 修改hostname并注释Https:

```
# The IP address or hostname to access admin UI and registry service.
# DO NOT use localhost or 127.0.0.1, because Harbor needs to be accessed by external clients
hostname: 192.168.182.129

# http related config
http:
  # port for http, default is 80. If https enabled, this port will redirect to https port
  port: 80

# https related config
#https:
  # https port for harbor, default is 443
  # port: 443
  # The path of cert and key files for nginx
  #certificate: /your/certificate/path
  #private_key: /your/private/key/path

# # Uncomment following will enable tls communication between all harbor components
# internal_tls:
#   # set enabled to true means internal tls is enabled
#   enabled: true
#   # put your cert and key files on dir
#   dir: /etc/harbor/tls/internal

# Uncomment external_url if you want to enable external proxy
# And when it enabled the hostname will no longer used
# external_url: https://reg.mydomain.com:8433
```

输入命令./install.sh启动Harbor即可。在Harbor界面添加项目, 项目名为repo:

The screenshot shows a dashboard with the following components:

- Summary Cards:**
 - 项目 (Projects):** 私有: 0, 公开: 2, 总计: 2
 - 镜像仓库 (Mirror Repositories):** 私有: 0, 公开: 2, 总计: 2
 - 已使用的存储空间 (Used Storage Space):** 1.15 GiB
- Actions:** + 新建项目, × 删除
- Search/Filter:** 所有项目, search icon, refresh icon
- Table:**

<input type="checkbox"/>	项目名称	访问级别	角色	类型	镜像仓库数	创建时间
<input type="checkbox"/>	library	公开	项目管理员	项目	0	2022/11/30 上午10:20
<input type="checkbox"/>	repo	公开	项目管理员	项目	2	2022/11/30 上午10:42
- Footer:** 页面大小 15, 1 - 2 共计 2 条记录

修改两台虚拟机的/etc/docker/daemon.json文件，设置私有仓库并重启docker：

This screenshot is identical to the one above, showing the same dashboard layout with project statistics, actions, search, table, and footer information.

至此，所有需要安装配置的工具全部介绍完成。项目的部署过程见应用案例的应用部署部分。

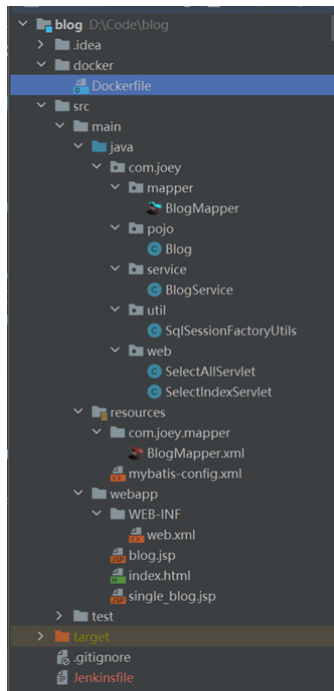
应用案例

应用描述

编写了一个简单的博客页面，可以显示全部博客以及查看具体某篇博文。HTTP协议部分由tomcat处理。没错这就是我那个极其失败的动态博客项目

应用开发

本项目的文件结构如下：



(1) 环境准备:

- 创建模块blog, 引入坐标
- 创建三层架构的包结构
- 数据库表 tb_blog
- 实体类 Blog
- MyBatis 基础环境
- Mybatis-config.xml
- BlogMapper.xml
- BlogMapper接口

(2) 编写 BlogMapper, 用注解的方式定义 selectAll(), selectIndex() 方法, 实现在数据库中查询所有文章和指定文章。

```
public interface BlogMapper {  
  
    1 个用法  👤 No-drink  
    @Select("select * from tb_blog")  
    List<Blog> selectAll();  
  
    1 个用法  👤 No-drink  
    ⚠ @Select("SELECT * FROM `myblogs`.`tb_blog` WHERE `index` = #{index}")  
    Blog selectIndex(int index);  
}
```

(3) 创建 utils 包, 在该包下创建名为 SqlSessionFactoryUtils 工具类用以获取MySQL的连接。

(4) 在 service 包下创建 BlogService 类, 实现 BlogMapper 接口中的两个方法。

```

1 个用法  No-drink
public List<Blog> selectAll(){

    SqlSession sqlSession = factory.openSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    List<Blog> blogs = mapper.selectAll();
    sqlSession.close();
    return blogs;
}

1 个用法  No-drink
public Blog selectIndex(int index){
    SqlSession sqlSession = factory.openSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    Blog blog = mapper.selectIndex(index);
    sqlSession.close();
    return blog;
}

```

(5) 在 web 包下创建名为 SelectAllServlet 和 SelectIndexServlet 的 servlet，该 servlet 的逻辑如下：

- 调用 BlogService 的 selectAll() 和 selectIndex() 方法进行业务逻辑处理，并接收返回的结果
- 将上一步返回的结果存储到 request 域对象中
- 跳转到 blog.jsp 和 single_blog.jsp 页面进行数据的展示

(6) blog.jsp 和 single_blog.jsp 两个 jsp 页面通过 EL 表达式显示查询的内容。

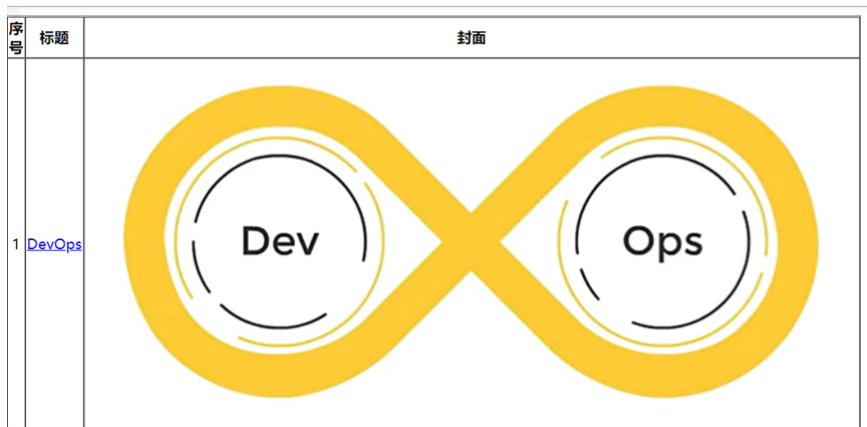
```

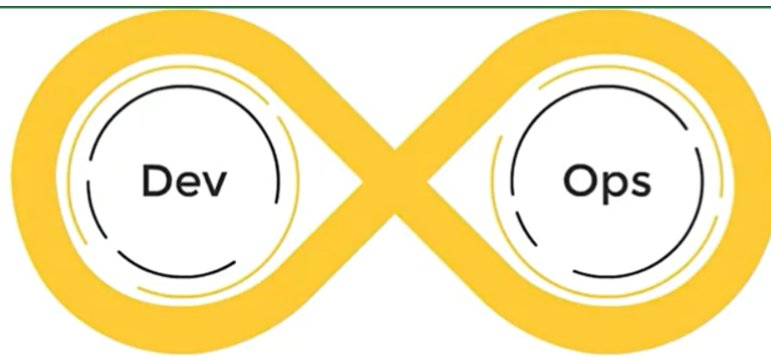
<c:forEach items="${blogs}" var="blog" varStatus="status">
  <tr align="center">
    <td>--<td>${blog.index}</td>-->
    <td>${status.count}</td>
    <td><a href="/blog/selectIndexServlet?index=${blog.index}">${blog.title}</a></td>
    <td>${blog.image}</td>
    <td>${blog.text}</td>
  </tr>
</c:forEach>

```

3.3 应用测试

使用 IntelliJ 的 tomcat 插件进行构建，运行，运行界面正常，达到预期效果：





DevOps

一、DevOps介绍

软件开发最开始是由两个团队组成:

- 开发计划由**开发团队**从头开始设计和整体系统的构建。需要系统不停的迭代更新。
- **运维团队**将开发团队的Code进行测试后部署上线。希望系统稳定安全运行。

这看似两个目标不同的团队需要协同完成一个软件的开发。

在开发团队指定好计划并完成coding后, 需要提供到运维团队。

运维团队向开发团队反馈需要修复的BUG以及需要返工的任务。

这时开发团队需要经常等待运维团队的反馈。这无疑延长了事件并推迟了整个软件开发的周期。

应用部署

本实验在进行中共采用了三种部署方式: 1.直接将源代码打包为war包后使用ssh发送至华为云服务器tomcat的webapps文件夹下, tomcat自动解压运行; 2.将源代码利用Docker打包成镜像发布至Harbor, 再通知虚拟机2拉去镜像运行容器部署; 3.使用pipeline风格进行第二种方式的部署。下面将分别介绍:

直接发送war包到华为云服务器

华为云服务器运行有tomcat容器, 其docker-compose.yml配置如图:

```
version: "3.1"
services:
  tomcat:
    restart: always
    image: tomcat:8.5.38
    container_name: tomcat
    ports:
      - 80:8080
    volumes:
      - ./data/webapps:/usr/local/tomcat/webapps/
      - ./data/conf:/usr/local/tomcat/conf
      - ./data/logs:/usr/local/tomcat/logs
      - ./data/bin:/usr/local/tomcat/bin
      - /etc/localtime:/etc/localtime
    environment:
      TZ: Asia/Shanghai
```

将war包发送至webapps文件夹即可。Jenkins新建自由风格工程, 命名为myblog_HECS, 关键配置如下:

General 源码管理 构建触发器 构建环境 构建 构建后操作

参数化构建过程 ?

Git 参数 ?

名称 ?

tag

描述 ?

[纯文本] 预览

参数类型 ?

标签

General 源码管理 构建触发器 构建环境 构建 构建后操作

源码管理

无

Git ?

Repositories ?

Repository URL ?

http://192.168.182.129:8929/root/blog

Credentials ?

- 无 - 添加

高级...

构建

执行 shell ?

命令

git checkout \$tag

查看 可用的环境变量列表

高级...

调用顶层 Maven 目标 ?

Maven 版本

maven

目标

clean
package -DskipTests

General 源码管理 构建触发器 构建环境 构建 构建后操作

Send build artifacts over SSH ?

SSH Publishers

SSH Server

Name ?

MY HECS

高级...

Transfers

Transfer Set

Source files ?

target/*.war

Remove prefix ?

target/

Remote directory ?

Exec command ?

保存 应用

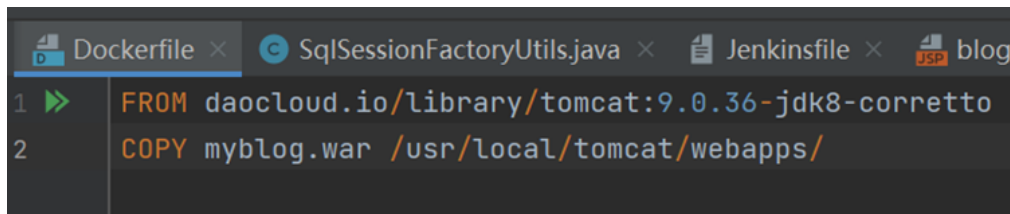
保存配置后点击参数化构建，选择相应版本，构建成功：

```
SSH: Connecting from host [06e49792fc57]
SSH: Connecting with configuration [MY_HECS] ...
SSH: Disconnecting configuration [MY_HECS] ...
SSH: Transferred 1 file(s)
Finished: SUCCESS
```

在浏览器访问<http://116.204.107.14/myblog/> (具有公网IP, 所有人均可访问), 任务部署成功。

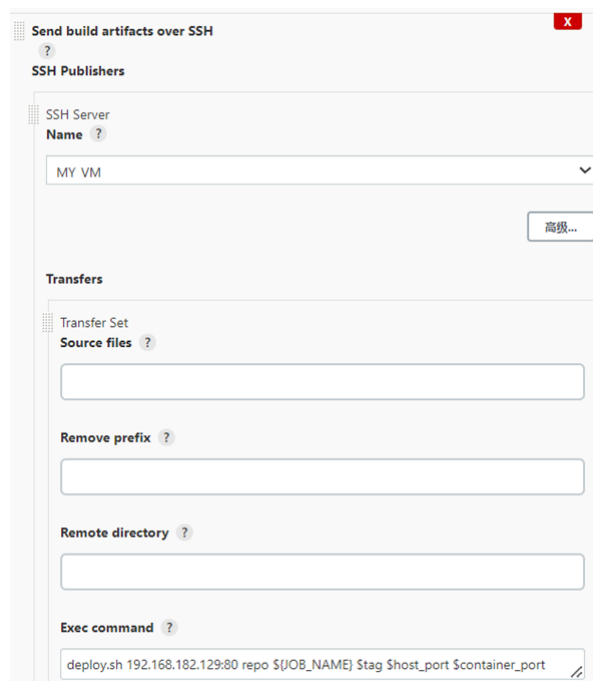
以镜像方式发布

新建工程myblog, 与上一工程配置不同之处为增加了两个字符参数 `host_port` 和 `container_port` 用以指定容器运行的端口映射。构建操作中maven打包完成后需要通过Dockerfile进行镜像构建和推送, 再通知目标服务器拉取镜像并运行。Dockerfile如图:



```
1 FROM daocloud.io/library/tomcat:9.0.36-jdk8-corretto
2 COPY myblog.war /usr/local/tomcat/webapps/
```

Jenkins构建镜像和通知目标服务器配置为:



deploy.sh为自定义的脚本命令，添加运行权限并放置在/usr/bin目录下使其可以在全局使用：

```
harbor_url=$1
harbor_project_name=$2
project_name=$3
tag=$4
host_port=$5
container_port=$6

imageName=$harbor_url/$harbor_project_name/$project_name:$tag

containerId=$(docker ps -a | grep ${project_name} | awk '{print $1}')
if [ "$containerId" != "" ]; then
    docker stop $containerId
    docker rm $containerId
    echo "Delete Container Success"
fi

imageId=$(docker images | grep ${project_name} | awk '{print $3}')

if [ "$imageId" != "" ]; then
    docker rmi -f $imageId
    echo "Delete Image Success"
fi

docker login -u admin -p Harbor12345 $harbor_url

docker pull $imageName

docker run -d -p $host_port:$container_port --name $project_name $imageName

echo "Start Container Success"
echo $project_name
```

配置完成后点击参数化构建，选择构建版本即可。输出如下：

```
-
9cfade67d536: Layer already exists
479704e33790: Layer already exists
1ca8585e06ef: Layer already exists
626b7a23b7a8: Layer already exists
57d09bee3af4: Layer already exists
62ae9fa2eaea: Pushed
v2.0.0: digest: sha256:f052daf1e335b0e04d23696d8ae3aa13c53dcd96b294dff29a6a2bd4ae094fae size: 1580
SSH: Connecting from host [06e49792fc57]
SSH: Connecting with configuration [MY_VM] ...
SSH: EXEC: completed after 7,820 ms
SSH: Disconnecting configuration [MY_VM] ...
SSH: Transferred 0 file(s)
Finished: SUCCESS
```

查看虚拟机，容器成功运行。访问<http://192.168.182.128:8081/myblog/>，页面显示没有问题：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAME
a4b8acrf00b08	192.168.182.129:80/repo/myblog_pipeline:v1.0.0	"catalina.sh run"	12 hours ago	Up About an hour	0.0.0.0:8080->8080/tcp, :::8081->8080/tcp	myblog_pipeline

使用pipeline方式部署项目

Pipeline可以使我们在部署过程中清楚看到每个阶段的运行情况输出，整个流程更加清晰。新建流水线风格的工程myblog_pipeline，Git仓库和参数设置与上一项目相同。脚本路径选择jenkinsfile即可：

脚本路径 ?

Jenkinsfile

轻量级检出 ?

编辑jenkinsfile，分为四个阶段“拉取Git代码”“构建代码”“制作自定义镜像并发布Harbor”“目标服务器拉取镜像并运行”。

```

stage('拉取Git代码') {
    steps {
        checkout([$class: 'GitSCM', branches: [[name: "${TAG}"]], extensions: [], userRemoteConfigs: [[
    ]
    ]
}

stage('构建代码') {
    steps {
        sh '/var/jenkins_home/maven/bin/mvn clean package -DskipTests'
    }
}

stage('制作自定义镜像并发布Harbor') {
    steps {
        sh '''mv ./target/*.war ./docker
docker build -t ${JOB_NAME}:${TAG} docker/'''

        sh '''docker login -u ${harborUser} -p ${harborPasswd} ${harborHost}
docker tag ${JOB_NAME}:${TAG} ${harborHost}/${harborRepo}/${JOB_NAME}:${TAG}
docker push ${harborHost}/${harborRepo}/${JOB_NAME}:${TAG}'''
    }
}
}

```

```

stage('目标服务器拉取镜像并运行') {
    steps {
        sshPublisher(publishers: [sshPublisherDesc(configName: 'MY_VM', transfers: [sshTransfer(cleanRe
        //sshPublisher(publishers: [sshPublisherDesc(configName: 'MY_VM', transfers: [sshTransfer(cleanRe
    ]
}
}
}

```

点击参数化构建，选择版本与指定端口号后构建，结果如图：

阶段视图



查看目标服务器和<http://192.168.182.128:8081/myblog/>，构建成功！

总结

问题及解决措施

- 起初打算Jenkins等工具也全部在华为云服务器上进行，但考虑到服务器负载，以及DevOps的具体工作场景，决定在虚拟机上完成相关工具的安装。好在使用docker-compose的运行方式，数据可以很方便地进行迁移，只要将jenkins_home映射的data文件夹复制到虚拟机重新docker-compose up -d。即可将之前的所有配置移动到虚拟机1；
- Jenkins使用宿主机Docker时，采用官方推荐的方式将docker.sock文件映射到jenkins容器内部，但在容器内部使用Docker时报错：**docker: /lib/x86_64-linux-gnu/libc.so.6: version 'GLIBC_2.32' not found (required by docker)**。原因是Jenkins容器内部GLIBC的版本较低，推测为Docker版本过高所以要求的版本也较高。尝试升级容器内部GLIBC，但因为容易出错且容器内

部缺少很多命令，故尝试拉取Jenkins最近镜像，但发现最新的镜像内部GLIBC版本仍不满足。最后只好卸载Docker，使用 `apt-cache madison docker-ce` 命令发现可以自动安装的最老版本为 `20.10.13`，安装后仍报相同错误，最后采用手动安装 `20.10.12` 版本Docker，问题解决。需要注意之前也有提到使用docker-compose可以很方便地进行数据的转移，这次也以为使用了数据卷的映射，使这次Docker重新安装没有必要重新拉取镜像并运行容器，所以无论是否使用docker-compose，**通过数据映射的数据永久化措施还是非常必要的。**

3. 内网穿透问题。原先打算将目标服务器设置为华为云服务器，但无法连接到Harbor仓库，最后发现是因为Harbor地址为内网地址，服务器无法访问，故重新安装一台虚拟机以解决问题。
4. Jenkinsfile语法问题，一些参数变量无法被写入命令，通过查阅文档来解决问题，但最后阶段通知目标服务器却一直无法正确成功发送命令，通过echo命令查看发送的命令，有两个以变量表示的参数无法写入，最后只能手动写入，放弃以变量方式表示。

项目存在问题与不足

在具体部署的应用上，和起初预想的会差很多。自己动手搭建博客确实难度会比较大，最后也只做了一个很简单的页面，后期可能会借助一些工具来搭建。结果上也没有达到最后可以在服务器上部署镜像的设想，但原因是因为Harbor仓库没有公网IP，所以也可以接受。

但在pipeline构建上有一个问题是使用ssh命令通知服务器拉取镜像的时候，命令 `deploy.sh $harborHost $harborRepo $JOB_NAME $TAG $host_port $container_port` 中 `$harborHost $harborRepo` 无法顺利传入，可能是由于environment变量的问题，最后只好采用了写死的方式解决，这是一个比较大的遗憾。

项目展望

对于部署的应用还是太过简陋，后续可能会增加一些功能和CSS配置。另外也考虑更换数据库，一方面不喜欢MySQL数据库，另一方面其作为关系型数据库也不适合存放博客数据，可能可以更换为MongoDB。也可以采用Hugo，WordPress等博客工具。

对于整个DevOps流程而言，可以使Jenkins自动检测代码是否更新，自动拉取新代码。此外可以集成Sonar Qube在构建代码后进行代码质量检测，利用Jenkins插件在部署完成后通过钉钉等通知程序员。还可以集成Kubernetes进行多集群管理，这些都可以让DevOps的流程更加适应实际工程应用。