

Docker+Jenkins+GitLab+Maven+SpringBoot&SpringCloud

自动化构建与部署指南文档说明书&工程源代码案例

目录

1、前言.....	4
1.1 目的与初衷.....	4
1.2 什么是 DevOps.....	5
1.3 软件环境搭建内容.....	5
1.4 操作系统目录知识.....	5
1.5 准备工作与事项.....	6
2、Docker 基础知识.....	7
2.1、Docker 理念与出现原因.....	7
2.1.1 Docker 理念.....	7
2.1.2 Docker 为什么出现.....	7
2.2、为什么说“一次构建，到处运行”.....	8
2.2.1 更快的应用交付和部署.....	8
2.2.2 更便捷的升级和扩缩容.....	8
2.2.3 更简单的系统运维.....	8
2.2.4 更高效的技术资源利用.....	8
2.3、Docker 底层原理.....	9
2.3.1 Docker 是怎么工作的.....	9
2.3.2 为什么 Docker 比虚拟机 VM 快.....	9
2.4、Docker 有哪些优势.....	10
2.4.1 Docker 安装软件更简单.....	10
2.4.2 Docker 五大优势.....	10
2.5、Docker 系统架构.....	10
2.5.1 Docker 系统架构图.....	10
2.5.2 Docker 架构几个概念.....	11
2.6、Docker 安装步骤.....	12
2.6.1 Docker CE 与 Docker EE 的区别.....	12
2.6.2 移除旧的版本：.....	12
2.6.3 更新 yum 缓存.....	12
2.6.4 启动 Docker 后台服务.....	12
2.6.5 开机自动启动 Docker.....	12
2.6.6 测试 Docker.....	13
2.6.7 有哪些加速器服务进行选择呢？.....	13
2.7、Docker 与 NetWork.....	14

2.7.1 为什么要使用 NetWork.....	14
2.7.2 常用命令.....	14
2.8、Docker 与 Registry.....	14
2.8.1 为什么用 registry.....	14
2.8.2 拉取 registry 镜像.....	15
2.8.3 运行 registry.....	15
2.8.4 修改 daemon.json 文件.....	15
2.8.5 重新加载 daemon 文件&重启 docker.....	15
2.8.6 浏览器验证是否成功.....	15
3、Docker 与 Gitlab 详解.....	15
3.1 Linux 版本.....	16
3.2 Docker 版本.....	16
3.4 获取 gitlab 镜像包.....	16
3.5 在本机准备 gitlab 工作目录.....	17
3.6 运行脚本启动 GitLab.....	17
3.7 修改 gitlab.rb 配置文件.....	17
3.8 进去 gitlab 容器重启服务.....	18
3.9 重启 gitlab 容器命令.....	19
3.10 检查启动信息.....	19
3.11 查看本机端口状态.....	19
3.12 GitLab 命令.....	19
3.13 浏览器检查是否安装成功.....	20
3.14 GitLab 主界面.....	20
4、CentOS 安装 Git.....	21
4.1 git 简介.....	21
4.2 git 的作用.....	21
4.3 git 安装命令.....	21
4.4 git 常见命令.....	21
5、CentOS7 与 JDK 详解.....	23
5.1 什么是 JDK.....	23
5.2 JDK 安装准备工作.....	23
5.3 创建指定文件目录.....	24
5.4 解压指定文件.....	24
5.5 拷贝指定目录.....	24
5.6 配置环境变量.....	25
6、CentOS7 与 Maven 详解.....	25
6.1 什么是 Maven.....	26
6.2 Maven 安装准备工作.....	26
6.3 指定文件目录安装.....	27
6.4 解压指定文件.....	27
6.5 拷贝指定目录.....	27
6.6 配置环境变量.....	27
7、Docker 与 Jenkins 安装与事项详解.....	28
7.1 什么是 Jenkins.....	28

7.2 Jenkins 安装准备工作.....	28
7.3 Jenkins 安装图解说明.....	29
7.4 设置全局工具.....	30
7.5 系统设置.....	32
7.6 添加全局用户名凭证.....	34
7.7 凭证类型方式.....	34
7.8 插件安装.....	35
8、私钥与公钥详解.....	37
8.1 为什么要公钥和私钥.....	37
8.2 公钥与公钥的区别.....	37
8.3 公钥与公钥通信图解.....	38
8.4 公钥与公钥通信原理.....	38
8.5 公钥与公钥生成&免密登录操作.....	39
8.5.1 公钥和私钥生成.....	39
8.5.2 免密登录操作.....	39
8.6 配置 Gitlab 公钥.....	40
9、Docker、Jenkins 等编译镜像与部署详解.....	41
9.1、需要准备的工作有哪些.....	41
9.2、SpringBoot 配置和代码详解.....	42
9.2.1 SpringBoot 简要.....	42
9.2.2 工程的 pom.xml 配置.....	42
9.2.3 no main manifest attribute 错误解决.....	44
9.2.4 env 环境变量文件.....	45
9.2.5 Dockerfile 打包工程镜像细讲.....	45
9.2.6 工程文件结构.....	46
9.2.7 build.sh 文件 shell 脚本详解.....	46
9.2.8 不同环境的配置文件.....	52
9.2.9 Controller 测试代码.....	52
9.3、非多台机器免密远程登录&Jenkins 部署流程详解.....	52
9.3.1 特别说明.....	52
9.3.2 新建 maven 工程.....	53
9.3.3 参数化构建过程说明.....	53
9.3.4 源码管理.....	55
9.3.5 Build 编译设置.....	55
9.3.6 SSH Publishers 设置.....	55
9.3.7 构建与编译部署项目.....	56
9.3.8 运行&部署结果.....	57
9.4、多台机器免密远程登录&Jenkins 部署流程详解.....	59
9.4.1 特别说明.....	59
9.4.2 新建 maven 工程.....	59
9.4.3 参数化构建过程说明.....	59
9.4.4 源码管理.....	61
9.4.5 Build 编译设置.....	61
9.4.6 SSH Publishers 设置.....	61

9.4.7 构建与编译部署项目.....	63
9.4.8 运行&部署结果.....	64
10、 总结&建议&学习.....	65
10.1 总结与建议.....	66
10.2 工程源代码&推荐学习&参考文章.....	66

姓名	日期	版本	说明
梁继龙	2019-07-11	V.0.0.1	初稿创建
梁继龙	2019-07-12	V.0.0.1	添加 Docker 基础知识
梁继龙	2019-07-13	V.0.0.1	添加 Gitlab 详解
梁继龙	2019-07-15	V.0.0.1	添加 JDk、Maven
梁继龙	2019-07-16	V.0.0.1	添加私钥和公钥
梁继龙	2019-07-17	V.0.0.1	添加 Jenkins 部署过程
梁继龙	2019-07-18	V.0.0.1	添加操作系统目录&修改文档
梁继龙	2019-07-19	V.0.0.1	优化文档&修改文档知识点
梁继龙	2019-08-13	V.0.0.1	优化文档&添加自动启动 Docker

1、前言

1.1 目的与初衷

- 本文档主要结合工作过程中部署不同环境服务器的项目案例场景为初心进行实际细讲
- 文档会涉及 Docker 常见的知识点的结合一起使用的场景实操。
- 本文档会讲述 Docker、Jenkins、GitLab、SpringBoot、Maven 等技术结合实现自动化运维部署（OpsDev）应用工程，[适合 SpringCloud 部署](#)。

1.2 什么是 DevOps

➤ **DevOps** (Development 和 Operations 的组合词) 是一组过程、方法与系统的统称, 用于促进开发 (应用程序/软件工程)、技术运营和质量保障 (QA) 部门之间的沟通、协作与整合, 它是一种重视“软件开发人员 (Dev)”和“IT 运维技术人员 (Ops)”之间沟通合作的文化、运动或惯例。透过自动化“软件交付”和“架构变更”的流程, 来使得构建、测试、发布软件能够更加地快捷、频繁和可靠。

➤ 它的出现是由于软件行业日益清晰地认识到: 为了按时交付软件产品和服务, 开发和运营工作必须紧密合作。

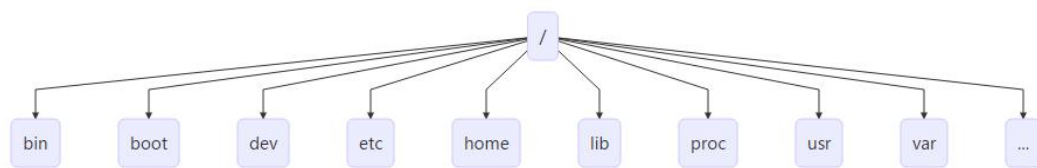
1.3 软件环境搭建内容

- 基于 **Linux** 内核的 [CentOS-7-x86_64-Minimal-1810.iso](#) 操作系统上完成
- 在文档会涉及 **JDK** (**JDK1.8**), **Maven**(**Maven3.6.x**)的安装过程讲解
- 如何在 **Docker** 上面安装 **GitLab** 详解和实际应用
- 如何在 **Docker** 创建 **NetWork** 网络与在工程中使用该网络
- 如何在 **Docker** 安装 **Registry** 私服与在工程中如何推送和拉取镜像
- 如何在 **Docker** (**Version:18.09.6**) 上面如何安装 **Jenkins** 详解和实际应用
- 如何使用 **Maven** 结合 **Docker** 把 **SpringBoot** 应用编译成可用的镜像进行部署。
- 其中 **JDK** 和 **Maven** 是传统方式进行安装, 由于有些软件在 **Docker** 安装过程并没传统方式安装简单, 比如: **Jenkins**。

1.4 操作系统目录知识

- 在整个文档里面都会提到**软件安装/usr/local**这个目录,但是有些包含**/home**是存放源码的路径,当初学习 [Docker-compose](#) 的时候, 不知道这个软件是一个二进制文件, 然后不按照系统的要求来安装, 一直以为是下载过程文件损坏了或者哪里不正确, 当初权限也授权了还是没执行成功。故去谷歌找了几篇文章才知道其中问题所在, 科普一下 **Linux** 操作系统目录知识点。

- 目录结构图



➤ 目录说明

目录	目录说明
bin	存放二进制可执行文件(ls,cat,mkdir 等)
boot	存放用于系统引导时使用的各种文件
dev	用于存放设备文件
etc	存放系统配置文件
home	存放所有用户文件的根目录
lib	存放跟文件系统中的程序运行所需要的共享库及内核模块
mnt	系统管理员安装临时文件系统的安装点
opt	额外安装的可选应用程序包所放置的位置
proc	虚拟文件系统，存放当前内存的映射
root	超级用户目录
sbin	存放二进制可执行文件，只有 root 才能访问
tmp	用于存放各种临时文件
usr	用于存放系统应用程序，比较重要的目录/usr/local 本地管理员软件安装目录
var	用于存放运行时需要改变数据的文件

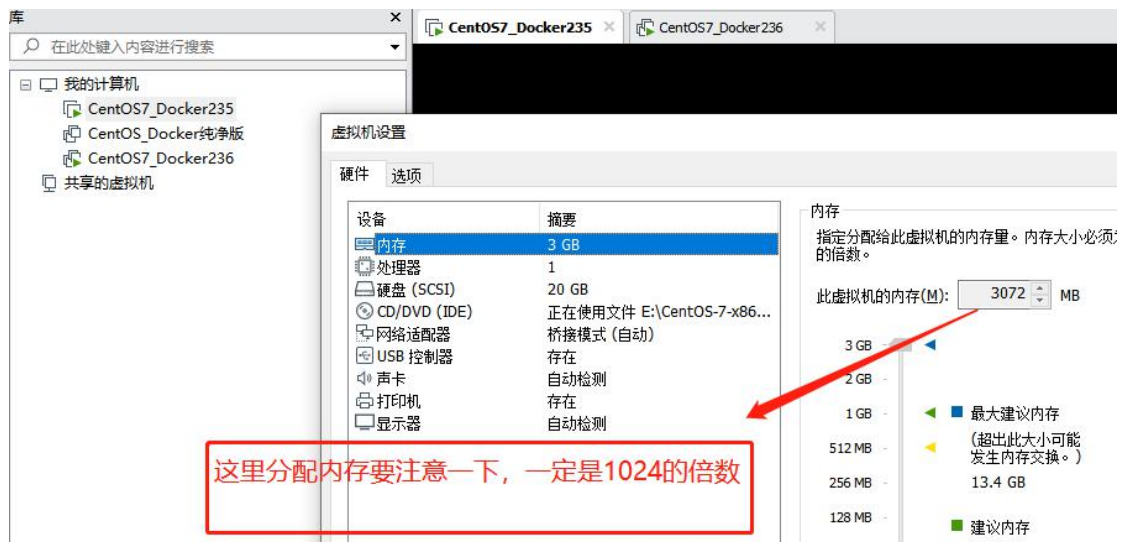
1.5 准备工作与事项

➤ 至少两台机器，这里描述两台机器的配置环境，是在虚拟机 VMware Workstation Pro 上面安装两个 [CentOS-7-x86_64-Minimal-1810.iso](#) 操作系统。

➤ 注意事项：其中 Gitlab、Registry、Jenkins 都安装在 node1 机器上面，也就是 node1 作为主机(master),node2 作为 slave（从机或副机），机器名起有意义或能区分即可，推荐起 master 和 slave，这里就不作过多的阐述，为了避免看文档有疑问，请看清单列表。

机器名称	机器分配 IP	机器分配内存	处理器	硬盘
node1(master)	192.168.1.235	3G	1 核	20G
node2(slave)	192.168.1.236	3G	1 核	20G

➤ 虚拟机设置事项



2、Docker 基础知识

2.1、Docker 理念与出现原因

2.1.1 Docker 理念

- Docker 的主要目标是“**Build, Ship and Run Any App,Anywhere**”，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管理，使用户的 APP（可以是一个 WEB 应用或数据库应用等等）及其运行环境能够做到“**一次封装，到处运行**”。
- Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核的 cgroup, namespace, 以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于 操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。最初实现是基于 LXC，从 0.7 版本以后开始去除 LXC，转而使用自行开发的 libcontainer，从 1.11 开始，则进一步演进为使用 runC 和 containerd。
- 为了解决运行环境和配置问题的软件容器，方便做持续集成并有助于整体发布的容器虚拟化技术

2.1.2 Docker 为什么出现

- 一款产品从开发到上线，从操作系统，到运行环境，再到应用配置。作为开发+运维之间的协作我们需要关心很多东西，这也是很多互联网公司都不得不面对的问题，特别是各种版本的迭代之后，不同版本环境的兼容，对运维人员都是考验

- Docker 之所以发展如此迅速，也是因为它对此给出了一个标准化的解决方案。
- 环境配置如此麻烦，换一台机器，就要重来一次，费力费时。很多人想到，能不能从根本上解决问题，软件可以带环境安装？也就是说，安装的时候，把原始环境一模一样地复制过来。开发人员利用 Docker 可以消除协作编码时“在我的机器上可正常工作”的问题。
- 传统上认为，软件编码开发/测试结束后，所产出的成果即是程序或是能够编译执行的二进制字节码等(java 为例)。而为了让这些程序可以顺利执行，开发团队也得准备完整的部署文件，让运维团队得以部署应用程式，开发需要清楚的告诉运维部署团队，用的全部配置文件、所有软件环境。不过，即便如此，仍然常常发生部署失败的状况。Docker 镜像的设计，使得 Docker 得以打破过去「程序即应用」的观念。透过镜像(images)将作业系统核心除外，运作应用程式所需要的系统环境，由下而上打包，达到应用程式跨平台间的无缝接轨运作。

2.2、为什么说“一次构建，到处运行”

2.2.1 更快的应用交付和部署

- 传统的应用开发完成后，需要提供一堆安装程序和配置说明文档，安装部署后需根据配置文档进行繁杂的配置才能正常运行。Docker 化之后只需要交付少量容器镜像文件，在正式生产环境加载镜像并运行即可，应用安装配置在镜像里已经内置好，大大节省部署配置和测试验证时间。

2.2.2 更便捷的升级和扩缩容

- 随着微服务架构和 Docker 的发展，大量的应用会通过微服务方式架构，应用的开发构建将变成搭乐高积木一样，每个 Docker 容器将变成一块“积木”，应用的升级将变得非常容易。当现有的容器不足以支撑业务处理时，可通过镜像运行新的容器进行快速扩容，使应用系统的扩容从原先的天级变成分钟级甚至秒级。

2.2.3 更简单的系统运维

- 应用容器化运行后，生产环境运行的应用可与开发、测试环境的应用高度一致，容器会将应用程序相关的环境和状态完全封装起来，不会因为底层基础架构和操作系统的不一致性给应用带来影响，产生新的 BUG。当出现程序异常时，也可以通过测试环境的相同容器进行快速定位和修复。

2.2.4 更高效的技术资源利用

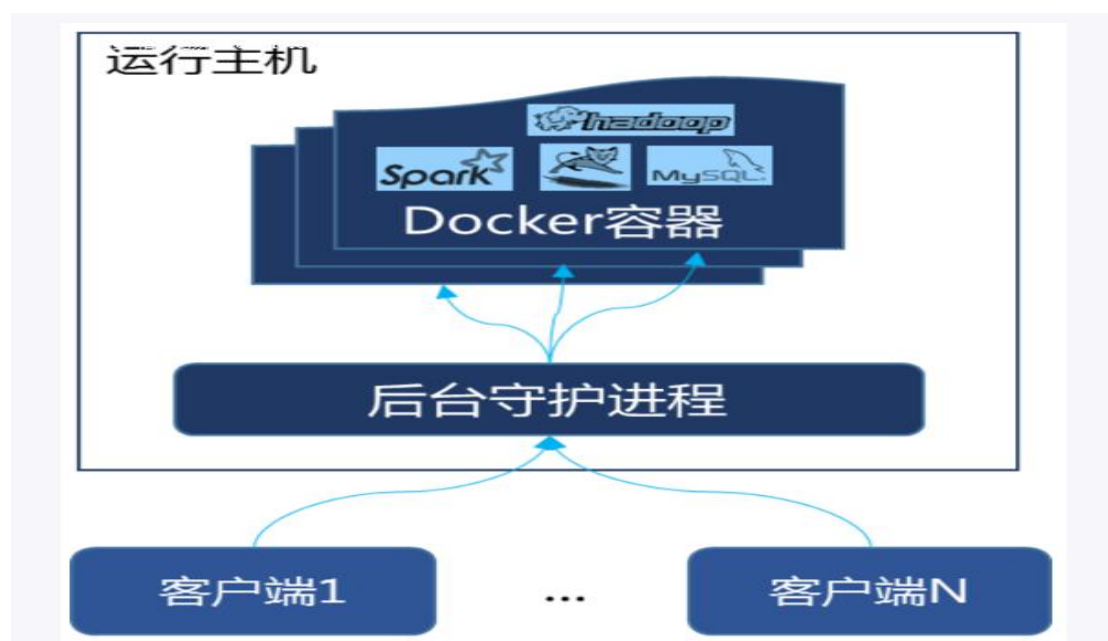
- Docker 是内核级虚拟化，其不像传统的虚拟化技术一样需要额外的 Hypervisor 支持，所以在一台物理机上可以运行很多个容器实例，可大大提升物理服务器的 CPU 和内存的

利用率。

2.3、Docker 底层原理

2.3.1 Docker 是怎么工作的

- Docker 是一个 Client-Server 结构的系统，Docker 守护进程运行在主机上，然后通过 Socket 连接从客户端访问，守护进程从客户端接受命令并管理运行在主机上的容器。容器，是一个运行时环境，就是我们前面说到的集装箱。



2.3.2 为什么 Docker 比虚拟机 VM 快

- (1)docker 有着比虚拟机更少的抽象层。由于 docker 不需要 Hypervisor 实现硬件资源虚拟化,运行在 docker 容器上的程序直接使用的都是实际物理机的硬件资源。因此在 CPU、内存利用率上 docker 将会在效率上有明显优势。
- (2)docker 利用的是宿主机的内核,而不需要 Guest OS。因此,当新建一个容器时,docker 不需要和虚拟机一样重新加载一个操作系统内核。仍而避免引导、加载操作系统内核返个比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载 Guest OS,返个新建过程是分钟级别的。而 docker 由于直接利用宿主机的操作系统,则省略了返个过程,因此新建一个 docker 容器只需要几秒钟

- 特性对比

特性	容器	虚拟机
启动&部署	秒级别	分钟级别
硬盘使用	一般为 MB	一般为 GB

性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个
移植性	轻便，灵活	笨重，与 VM 技术耦合度高

2.4、Docker 有哪些优势

2.4.1 Docker 安装软件更简单

- 使用 docker 安装软件过程稍微比较容易和简单，由于网上很多活雷锋把你所需的软件都一起打包好成镜像(Images)上传到 Docker Hub 官方私服，我们只需要了解些 docker 的命令或在安装过程中修改些配置即可。

2.4.2 Docker 五大优势

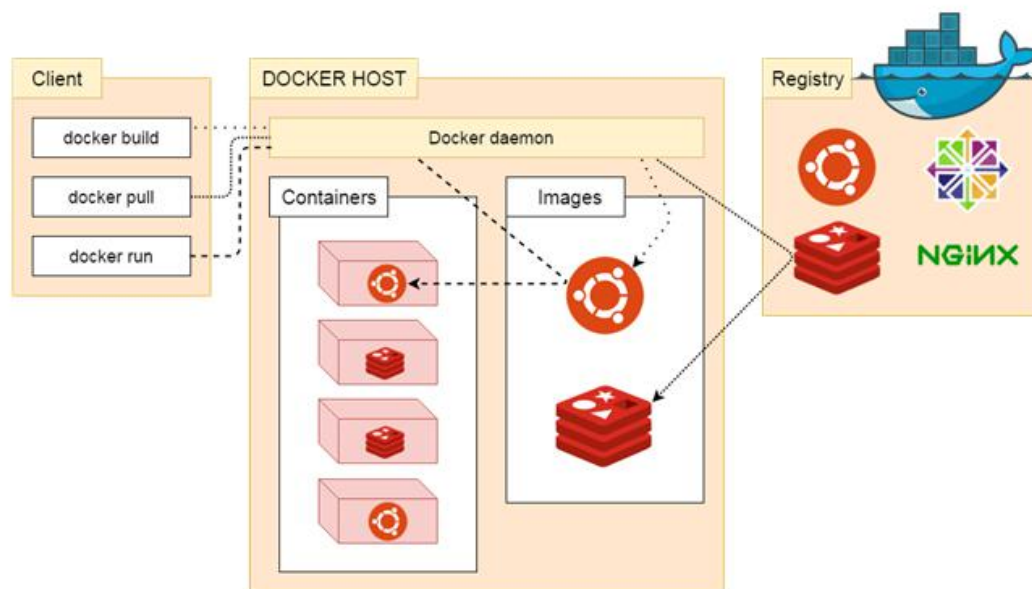
- 持续集成、版本控制、可移植性、隔离性、安全性

2.5、Docker 系统架构

2.5.1 Docker 系统架构图

- Docker 使用客户端-服务器 (C/S) 架构模式，使用远程 API 来管理和创建 Docker 容器
- Docker 容器通过 Docker 镜像来创建。
- 容器与镜像的关系类似于面向对象编程中的对象与类。
- 鲸鱼背上有集装箱,蓝色的大海里面（宿主机系统 Window/CentOS/Mac）、鲸鱼(Docker) 集装箱(容器实例) from 来自我们的进行模板，鲸鱼上的每个集装箱都是隔离的。

Docker	面向对象
容器	对象
镜像	类



2.5.2 Docker 架构几个概念

标题	说明
镜像(Images)	Docker 镜像是用于创建 Docker 容器的模板。
容器(Container)	容器是独立运行的一个或一组应用。
客户端(Client)	Docker 客户端通过命令行或者其他工具使用 Docker API 与 Docker 的守护进程通信。
主机(Host)	一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。
仓库(Registry)	Docker 仓库 用来保存镜像，可以理解为代码控制中的代码仓库。 Docker Hub 提供了庞大的镜像集合供使用。
Docker Machine	Docker Machine 是一个简化 Docker 安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装 Docker，比如 VirtualBox、Digital Ocean、Microsoft Azure。

2.6、Docker 安装步骤

2.6.1 Docker CE 与 Docker EE 的区别

- 2017 年 3 月开始 docker 在原来的基础上分为两个分支版本: Docker CE 和 Docker EE
- Docker CE 即社区免费版, Docker EE 即企业版, 强调安全, 但需付费使用。

2.6.2 移除旧的版本:

- 移除旧版本

```
sudo yum remove docker \
docker-client \
docker-client-latest \
docker-common \
docker-latest \
docker-latest-logrotate \
docker-logrotate \
docker-selinux \
docker-engine-selinux \
docker-engine
```

- 安装一些必要的系统工具:

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

2.6.3 更新 yum 缓存

```
sudo yum makecache fast
```

或

```
sudo yum -y install docker-ce
```

2.6.4 启动 Docker 后台服务

```
sudo systemctl start docker
```

2.6.5 开机自动启动 Docker

- 此操作非常重要,Docker 官方[&国内网站](#)安装步骤并没提到此操作.

```
sudo systemctl enable docker
```

2.6.6 测试 Docker

```
docker run hello-world
```

2.6.7 有哪些加速器服务进行选择呢？

- 1、鉴于国内网络问题,拉取 Docker Hub 镜像十分缓慢,我们可以需要配置加速器来解决.
- 建议使用[网易镜像](#)或[阿里镜像网站](#)
- 2、目前有阿里、腾讯、网易云这几个巨头的容器镜像服务做得比较好,服务比较稳定,这里推荐阿里的容器镜像服务,而且阿里在国内的技术毋庸置疑,而且经过学习过程中安装过程中会比较顺利,如果使用其他的话不敢保证,举例一个亲身经历的案例,在 docker 中文社区拉下来的镜像安装 gitlab 的时候中遇到各种报错。
- 3、容器镜像服务,容器镜像服务注意事项、如果没有账号的童鞋们可以通过注册一个或者淘宝号可以登录。



➤ 4 加速器服务配置步骤

- (1) 由于 centos7 安装 Docker 之后是无 daemon.json 文件, 需要自己手动创建一个, 创建一个目录如下命令:

```
sudo mkdir -p /etc/docker
```

- (2) 追加加速器地址到 daemon.json 文件里面命令:

➤ EOF 是一个 shell 的一个标识符, 作用是标识 shell 脚本的开始<<-'EOF'和结束 EOF

```
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://fxde.mirror.aliyuncs.com"]
}
```

```
}  
EOF
```

(3) 重新加载 daemon 文件命令:

```
sudo systemctl daemon-reload
```

(4) 重启 docker 服务命令:

```
sudo systemctl restart docker
```

(5) 开机自动启动 docker 服务命令

```
sudo systemctl enable docker
```

2.7、Docker 与 NetWork

2.7.1 为什么要使用 NetWork

➤ 由于在 Dockerfile 的文件配置需要设置网络参数。

2.7.2 常用命令

➤ 不指定网络驱动时默认创建的 bridge 网络

```
docker network create default_network
```

➤ 查看网络内部信息

```
docker network inspect default_network
```

➤ 列所有列表的网络

```
docker network ls
```

➤ 移除指定的网络

```
docker network rm default_network
```

2.8、Docker 与 Registry

2.8.1 为什么用 registry

➤ 外网访问官方的 registry 速度很慢，而国内的 registry 服务大多需要花钱。私有 registry 免费，搭建之后，能使内网的主机加快访问速度，对 CI,CD 的效率有很大提升。

➤ 能有效保护内部代码，防止放到公网泄漏出去。

2.8.2 拉取 registry 镜像

```
docker pull registry:2
```

2.8.3 运行 registry

- -d 表示后台(detach)运行
- -p 表示 port 端口的意思

```
docker run -d \
--restart=always -p 5000:5000 \
--name registry -v /usr/local/docker/data/registry:/var/lib/registry registry:2
```

2.8.4 修改 daemon.json 文件

- 编辑文件 vim /etc/docker/daemon.json
- registry-mirrors 加速器地址,建议使用阿里云或网易云
- insecure-registries 表示私服的路径

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"],
  "insecure-registries":["192.168.1.235:5000"]
}
```

2.8.5 重新加载 daemon 文件&重启 docker

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

2.8.6 浏览器验证是否成功

```
http://192.168.1.235:5000/v2/ 浏览器输出{}表示成功 或
http://192.168.1.235:5000/v2/_catalog
```

3、Docker 与 Gitlab 详解

3.1 Linux 版本

- 1、Linux 的版本是以 **CentOs7** 为主.命令如下:

```
cat /proc/version
```

- 2、命令输出的结果信息

```
[root@localhost ~]# cat /proc/version
Linux version 3.10.0-957.12.2.el7.x86_64
[root@localhost ~]#
```

3.2 Docker 版本

- 1、查看 docker 的版本命令

现在最新 **Docker** 版本是 **19.x**, 在 1 个月前安装版本 **18.x**

```
docker version
```

- 2、命令输出的结果信息

```
Client:
Version:      18.09.6
API version:  1.39
Go version:   go1.10.8
Git commit:   481bc77156
Built:        Sat May 4 02:34:58 2019
OS/Arch:      linux/amd64
Experimental: false

Server: Docker Engine - Community
Engine:
Version:      18.09.6
API version:  1.39 (minimum version 1.12)
Go version:   go1.10.8
Git commit:   481bc77
Built:        Sat May 4 02:02:43 2019
OS/Arch:      linux/amd64
Experimental: false

[root@node1 ~]#
```

Annotations in the image:

- Red arrow from **18.09.6** (Client Version) to **docker 版本号**
- Red arrow from **1.39** (Client API version) to **golang版本号**
- Red arrow from the **Client** section to **Docker客户端**
- Red arrow from the **Server** section to **Docker引擎**

3.4 获取 gitlab 镜像包

- 这里从阿里镜像服务里面拉取 **gitlab** 镜像有点大, 需要耐心等待, 下载完镜像之后通过 **docker** 命令可以看到它的大小, 可以看到它的文件有 **1.85G**, 为什么有那么大呢? 因为 **gitlab** 集成了很多依赖软件

```
docker pull gitlab/gitlab-ce
```


3.5 在本机准备 gitlab 工作目录

```
mkdir -p /usr/local/gitlab/config  创建 config 目录
mkdir -p /usr/local/gitlab/logs    创建 logs 目录
mkdir -p /usr/local/gitlab/data    创建 dat 目录
```

3.6 运行脚本启动 GitLab

```
docker run --detach \
  --privileged=true \
  --hostname 192.168.1.235 \
  --publish 7001:443 --publish 7002:80 --publish 7003:22 \
  --name gitlab --restart always \
  --volume /usr/local/gitlab/config:/etc/gitlab \
  --volume /usr/local/gitlab/logs:/var/log/gitlab \
  --volume /usr/local/gitlab/data:/var/opt/gitlab 8e28c88b6a21
```

➤ 参数说明:

参数名称	参数说明
detach	指定容器运行于前台还是后台
hostname	指定主机地址，如果有域名可以指向域名
publish	指定容器暴露的端口,左边的端口代表宿主机的端口，右边的是代表容器的端口
name	给容器起一个名字，
restart always	重启，只要 docker 自动重启，容器就会自动重启.减少人工重启工作。
volume	数据卷，在 docker 中是最重要的一个知识点.
--privileged=true	解决 Docker 挂载主机目录 Docker 访问出现 cannot open directory .: Permission denied

备注: 8e28c88b6a21 或容器名称代表阿里云拉下的镜像 id 这里只列举上面脚本的, [详情请看官方文档](#).

3.7 修改 gitlab.rb 配置文件

- 注意事项: external_url 和 gitlab_rails 这两个 ip 参数, **建议固定操作系统的静态不变的 IP 或说是域名进行配置**, 假设 IP 变得的话在 GitLab 新建项目的时候, 生成的 IP 还是原来的 IP, 此时就无法推送代码在 Gitlab 里面。
- 按上面的方式, gitlab 容器运行没问题, 但在 gitlab 上创建项目的时候, 生成项目的 URL 访问地址是按容器的 hostname 来生成的, 也就是容器的 id。作为 gitlab 服务器, 我们需要一个固定的 URL 访问地址, 于是需要配置

gitlab.rb（宿主机路径：/usr/local/gitlab/config/gitlab.rb）配置有三个参数如：

```
external_url 'http://192.168.1.235'
gitlab_rails['gitlab_ssh_host'] = '192.168.1.235'
gitlab_rails['gitlab_shell_ssh_port'] = 703
```

3.8 进去 gitlab 容器重启服务

- 由于我们运行是使用数据卷参数进行运行的，宿主机的 gitlab.rb 文件修改了，gitlab 容器里面的文件会跟着改，但是容器的文件不会跟着生效，必须要进去容器里面进行命令执行，重置配置文件比较耗费时间，需要耐心等待，如果时间比较短说明成功率不高，而且进去容器之后就退出啦。

```
docker exec -it gitlab /bin/bash 进去 gitlab 容器的命令
gitlab-ctl reconfigure 重置 gitlab 客户端的命令
```

- 进入 gitlab 容器截图

```
[root@localhost /]# docker exec -it gitlab /bin/bash
root@192:/# gitlab-ctl reconfigure
Starting Chef Client, version 13.6.4
resolving cookbooks for run list: ["gitlab"]
Synchronizing Cookbooks:
- gitlab (0.0.1)
- package (0.1.0)
- postgresql (0.1.0)
- redis (0.1.0)
- registry (0.1.0)
- mattermost (0.1.0)
- consul (0.1.0)
- gitaly (0.1.0)
- letsencrypt (0.1.0)
- nginx (0.1.0)
- runit (4.3.0)
- acme (3.1.0)
```

- 备注：下面信息如果出现了，恭喜你在 docker 安装 Gitlab 成功通过了。

```
* ruby_block[disable grafana] action run (skipped due to only_if)
(up to date)
Recipe: gitlab::deprecate-skip-auto-migrations
* file[/etc/gitlab/skip-auto-reconfigure] action create (skipped due to only_if)
* ruby_block[skip-auto-migrations deprecation] action run (skipped due to only_if)

Running handlers:
Running handlers complete
Chef Client finished, 4/598 resources updated in 01 minutes 04 seconds
gitlab Reconfigured!
```

3.9 重启 gitlab 容器命令

docker restart gitlab 命令

docker restart gitlab 这里重启容器也需要耐心等待.

3.10 检查启动信息

➤ docker ps 命令

docker ps

➤ 容器参数说明

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
容器 Id	镜像	命令	创建	运行状态	端口

备注：如果 status 显示 up 代表启动状态，如果 status 显示 Exited 代表无启动容器

3.11 查看本机端口状态

➤ netstat -tnl 命令

netstat -tnl

➤ netstat -tnl 截图

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22               0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25               0.0.0.0:*               LISTEN
tcp6       0      0 :::6379                  :::*                     LISTEN
tcp6       0      0 :::22                    :::*                     LISTEN
tcp6       0      0 :::7001                   :::*                     LISTEN
tcp6       0      0 :::7002                   :::*                     LISTEN
tcp6       0      0 :::7003                   :::*                     LISTEN
```

3.12 GitLab 命令

命令	命令说明
gitlab-ctl reconfigure	重新应用 gitlab 的配置
gitlab-ctl restart	重启 gitlab 服务
gitlab-ctl status	查看 gitlab 运行状态
gitlab-ctl stop	停止 gitlab 服务
gitlab-ctl tail	查看 gitlab 运行日志

3.13 浏览器检查是否安装成功

- 浏览器输入 <http://机器 IP:端口>，出现了此界面表示成功，由于 gitlab 安装之后需要重置密码，



Please create a password for your new account.

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Change your password

New password


Confirm new password

Change your password

Didn't receive a confirmation email? [Request a new one](#)

Already have login and password? [Sign in](#)

3.14 GitLab 主界面



Projects ▾ Groups ▾ More ▾


+ ▾

Search or jump to... 🔍

📄 🔗 📧 ? ▾ 🌐 ▾


Welcome to GitLab

Code, test, and deploy together




Create a project

Projects are where you store your code, access issues, wiki and other features of GitLab.




Create a group

Groups are a great way to organize projects and people.



Add people

Add your team members and others to GitLab.



Configure GitLab

Make adjustments to how your GitLab instance is set up.

4、CentOS 安装 Git

4.1 git 简介

- [Git](#) 是一个分布式版本控制软件，最初由林纳斯·托瓦兹创作，于 2005 年以 GPL 发布。最初目的是为更好地管理 Linux 内核开发而设计。应注意的是，这与 GNU Interactive Tools（一个类似 Norton Commander 界面的文件管理器）有所不同。
- git 最初的开发动力来自于 BitKeeper 和 Monotone。git 最初只是作为一个可以被其他前端（比如 Cogito 或 Stgit）包装的后端而开发的，但后来 git 内核已经成熟到可以独立地用作版本控制。很多著名的软件都使用 git 进行版本控制，其中包括 Linux 内核、X.Org 服务器和 OLPC 内核等项目的开发流程。

4.2 git 的作用

- 1、它的作用是 CentOS 进行拉取 GitLab 托管的代码，故 GIT 非 GitLab,它只是一个客户端协助软件工具
- 2、此工具可以使用 ssh-keygen 命令进行生成公钥和使用提供 gitlab 或 GitHub,Jenkins 等软件进行通信。

4.3 git 安装命令

- 以下命令来自 GitLab 新建项目生成的命令，在 GitHub 等平台同理也会有如下命令：
- CentOS 安装 git

```
yum install git -y 或者  
yum install -y git
```

- Ubuntu/Debian 安装 git

```
apt-get install git -y
```

4.4 git 常见命令

- Git 全局设置

```
git config --global user.name "admin"
```

```
git config --global user.email "admin@example.com"
```

➤ 推送到现有的 Git 存储库

命令	说明
user.name	表示用户名
user.email	表示邮箱

➤ 创建一个仓库

```
git clone ssh://git@192.168.1.235:7003/root/springboot.git
cd springbootx
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

➤ 命令参数说明

命令	说明
git clone	克隆命令
touch	命令用于修改文件或者目录的时间属性，包括存取时间和更改时间。若文件不存在，系统会建立一个新的文件
git add	添加文件到 git 的暂存区
git commit -m	添加并添加备注说明
git push -u origin	master 推送到远程分支
git clone	克隆命令

➤ 推送现有文件夹

```
cd existing_folder
git init
git remote add origin ssh://git@192.168.1.235:7003/root/springboot.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

➤ 命令参数说明

命令	说明
git init	初始化一个仓库
git remote add origin	远程添加

➤ 推送到现有的 Git 存储库

```
cd existing_repo
git remote rename origin old-origin
git remote add origin ssh://git@192.168.1.235:7003/root/springbootx.git
git push -u origin --all
git push -u origin --tags
```

➤ 命令参数说明

命令	说明
git remote rename	重命名分支
git push -u origin --all	推送所有
git push -u origin --tags	推送到标签分支

5、CentOS7 与 JDK 详解

5.1 什么是 JDK

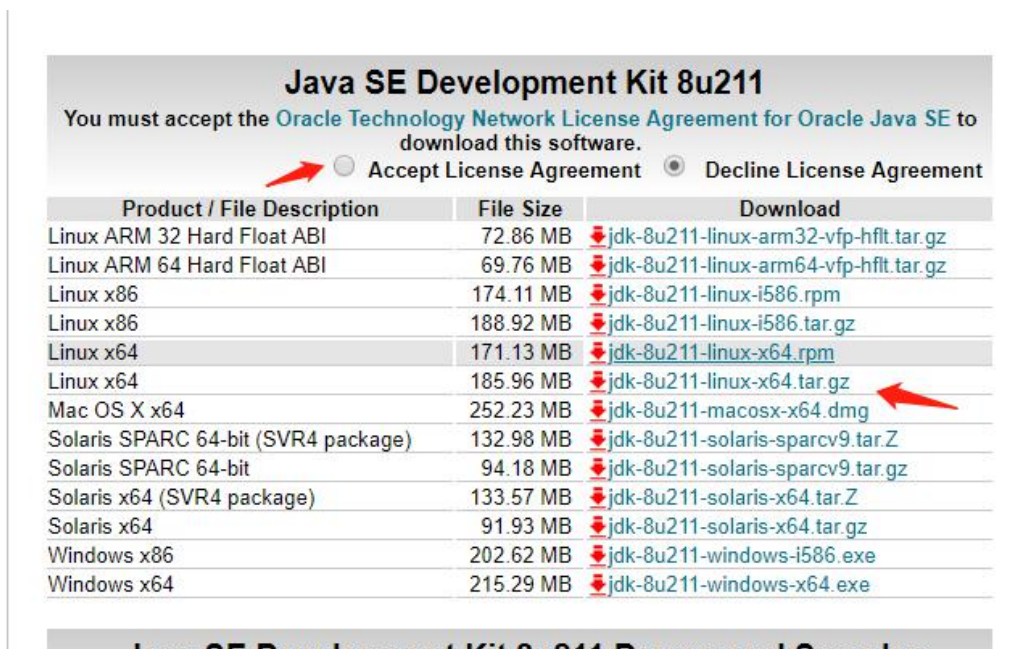
➤ JDK(Java Development Kit) 是 Java 语言的软件开发工具包(SDK)

5.2 JDK 安装准备工作

➤ [Oracle 中文官方](#)

Java SE Development Kit 8u181		
您必须接受 针对 Java SE 的 Oracle 二进制代码许可协议才能下载该软件。		
<input checked="" type="radio"/> 接受许可协议 <input type="radio"/> 不接受许可协议		
产品 / 文件说明	文件大小	下载
Linux ARM 32 硬浮点 ABI	72.95 MB	jdk-8u181-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 硬浮点 ABI	69.89 MB	jdk-8u181-linux-arm64-vfp-hflt.tar.gz
Linux x86	165.06 MB	jdk-8u181-linux-i586.rpm
Linux x86	179.87 MB	jdk-8u181-linux-i586.tar.gz
Linux x64	162.15 MB	jdk-8u181-linux-x64.rpm
Linux x64	177.05 MB	jdk-8u181-linux-x64.tar.gz
Mac OS X x64	242.83 MB	jdk-8u181-macosx-x64.dmg
Solaris SPARC 64 位 (SVR4 软件包)	133.17 MB	jdk-8u181-solaris-sparcv9.tar.Z
Solaris SPARC 64 位	94.34 MB	jdk-8u181-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 软件包)	133.83 MB	jdk-8u181-solaris-x64.tar.Z
Solaris x64	92.11 MB	jdk-8u181-solaris-x64.tar.gz
Windows x86	194.41 MB	jdk-8u181-windows-i586.exe
Windows x64	202.73 MB	jdk-8u181-windows-x64.exe

➤ [Oracle 英文官方](#)



- 本文档以 Oracle 英文官方的 [jdk-8u211-linux-x64.tar.gz](#) 版本进行详解

5.3 创建指定文件目录

- 进入/usr/local/目录进行创建一个 jdk1.8 文件夹
- 或者直接指定参数进行创建,-p 表示父目录下面的子目录也进行创建

```
mkdir jdk1.8 或者  
mkdir -p /usr/local/jdk1.8
```

5.4 解压指定文件

- 使用 tar 命令进行解压 tar.gz 文件

```
tar -zxvf jdk-8u211-linux-x64.tar.gz
```

5.5 拷贝指定目录

- 把解压的文件拷贝到指定目录
- -r :递归持续复制,用于目录的复制行为;
- * 星号代表所有内容都进行拷贝

```
cp -r * /usr/local/jdk1.8
```


5.6 配置环境变量

- 编辑 profile 文件进行配置环境变量

```
vim /etc/profile
```

- 方法一：配置内容如下

```
# set jdk
export JAVA_HOME=/usr/local/jdk1.8
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

- 方法 2：配置内容如下
- 1、两个>代表追加到指定的文件，且是追加到此文件的末尾行，而两个<表示 shell 的一个标识符的规范约束。
- 2、EOF 是一个 shell 的一个标识符，作用是标识 shell 脚本的开始<<EOF 和结束 EOF,可以任意字符，但是必须要有开始和结束，通常习惯用 EOF，而且必须无有特殊字符：比如空格。

```
echo >> /etc/profile <<-EOF
export JAVA_HOME=/usr/local/jdk1.8
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
EOF
```

- 重启加载 profile 让文件系统文件生效

```
source /etc/profile
```

- 查看是否生效

```
java -version
```

//输出的内容

Java(TM) SE Runtime Environment (build 1.8.0_121-b13)

Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

6、CentOS7 与 Maven 详解

6.1 什么是 Maven

- Apache Maven 是一个软件项目管理和理解工具。Maven 基于项目对象模型(POM)的概念，可以从一个中心信息段管理项目的构建、报告和文档。

6.2 Maven 安装准备工作

- [Apache Maven 官方网](#)

Link	
Binary tar.gz archive	apache-maven-3.6.1-bin.tar.gz
Binary zip archive	apache-maven-3.6.1-bin.zip
Source tar.gz archive	apache-maven-3.6.1-src.tar.gz
Source zip archive	apache-maven-3.6.1-src.zip

Windows版本 → [apache-maven-3.6.1-bin.zip](#) linux版本 → [apache-maven-3.6.1-bin.tar.gz](#)

源码安装包 → [apache-maven-3.6.1-src.tar.gz](#)

- [Release Notes](#)
- [Reference Documentation](#)

- 点击下载即可或者 Linux 的 wget 命令下载，不同 Linux 有不同的命令，通过官方网的网站的，然后通过浏览器鼠标右击查看元素可以[获取连接](#)

Link		Checksums
Binary tar.gz archive	apache-maven-3.6.1-bin.tar.gz	apache-maver
Binary zip archive	apache-maven-3.6.1-bin.zip	apache-maver
Source tar.gz archive	apache-maven-3.6.1-src.tar.gz	apache-maver
Source zip archive	apache-maven-3.6.1-src.zip	apache-maver

Release Notes

Sources Network Performance Memory Application Security Audits

ad>

'b">

y tar.gz archive</td>

F="http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz">apache-maven-3.6.1-bin.tar.gz

- wget 命令下载,但是必须要安装 wget 命令
- 安装命令: yum install -y wget

```
wget
http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz
```

- 下载结果

```

va HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
root@localhost jdk1.8]# cd /home/devsoft/
root@localhost devsoft]# wget http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz
2019-07-10 22:42:28-- http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.1/binaries/apache-maven-3.6.1-bin.tar.gz
solving mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)... 101.6.8.193, 2402:f000:1:408:8100::1
nnecting to mirrors.tuna.tsinghua.edu.cn (mirrors.tuna.tsinghua.edu.cn)|101.6.8.193|:80... connected.
TP request sent, awaiting response... 200 OK
length: 9136463 (8.7M) [application/x-gzip]
ving to: 'apache-maven-3.6.1-bin.tar.gz'

0%[-----]
19-07-10 22:42:29 (8.80 MB/s) - 'apache-maven-3.6.1-bin.tar.gz' saved [9136463/9136463]

root@localhost devsoft]# █

```

6.3 指定文件目录安装

- 进入/usr/local/目录进行创建一个 maven3 文件夹
- 或者直接指定参数进行创建,-p，如果父目录下面的子目录无也进行创建

```

mkdir maven3 或

mkdir -p /usr/local/maven3

```

6.4 解压指定文件

- 使用 tar 命令进行解压 tar.gz 文件

```

tar -zxvf apache-maven-3.6.1-bin.tar.gz

```

6.5 拷贝指定目录

- 把解压的文件拷贝到指定目录
- -r 表示递归持续复制,用于目录的复制行为;
- * 星号代表所有内容都进行拷贝

```

cp -r * /usr/local/maven3

```

6.6 配置环境变量

- 编辑 profile 文件进行配置环境变量

```

vim /etc/profile

```

- 方法 1: 配置内容如下

```

# set mvn
export M2_HOME=/usr/local/maven3
export PATH=$PATH:$JAVA_HOME/bin:$M2_HOME/bin

```

➤ 方法 2: 配置内容如下

1、两个>代表追加到指定的文件，且是追加到此文件的末尾行，而两个<表示 shell 的一个标识符的规范约束。

2、EOF 是一个 shell 的一个标识符，作用是标识 shell 脚本的开始<<EOF 和结束 EOF,可以任意字符，但是必须要有开始和结束，通常习惯用 EOF，而且必须无有特殊字符：比如空格。

```
echo >> /etc/profile <<-EOF
export M2_HOME=/usr/local/maven3
export PATH=$PATH:$JAVA_HOME/bin:$M2_HOME/bin
EOF
```

➤ 重启加载 profile 让文件系统文件生效

```
source /etc/profile
```

➤ 查看是否生效，这里出现，Java version: 1.8.0_121, vendor: Oracle，因为 Maven 是依赖 Jdk 环境，故要安装 jdk 才能运行。

```
[root@localhost devsoft]# mvn -version
Apache      Maven      3.6.1      (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555;
2019-04-05T03:00:29+08:00)
Maven home: /usr/local/maven3
Java version: 1.8.0_121, vendor: Oracle Corporation, runtime: /usr/local/jdk1.8/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-957.12.2.el7.x86_64", arch: "amd64", family:
"unix"
```

7、Docker 与 Jenkins 安装与事项详解

7.1 什么是 Jenkins

➤ [Jenkins](#) 的前身是 [Hudson](#)

➤ Jenkins 是开源 CI&CD 软件领导者，提供超过 1000 个插件来支持构建、部署、自动化，满足任何项目的需要。

7.2 Jenkins 安装准备工作

➤ docker 脚本安装，指定 Jenkins 默认路径/root/.jenkins/workspace 拉取代码的路径同步到宿主机路径/usr/local/jenkins/workspace。

➤ 由于 jdk 和 maven,git 都在宿主机，所以要 v（数据卷参数）指定 jenkins 容器的路径把宿主机的软件同步到容器

```
docker run -d --restart=always -p 9001:8080 \
-v /usr/local/jenkins/workspace:/root/.jenkins/workspace \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/bin/git:/usr/bin/git \
-v /usr/local/jdk1.8:/usr/local/jdk1.8 \
-v /usr/local/maven3:/usr/local/maven3 --name jenkins jenkins:latest
```

➤ 参数说明

参数	参数说明
docker.sock	守护进程文件
--restart=always	设置自动重启命令
--name jenkins	给这个容器起一个名称, jenkins:latest 指定版本进行安装
/usr/bin/git	git 安装的目录
/usr/local/jdk1.8	JDK 安装的目录
/usr/local/maven3	maven3 安装的目录
\	表示 shell 脚本换行转义符

7.3 Jenkins 安装图解说明

- 1、获取解锁密钥，此密钥是在 Jenkins 容器里面，此时必须要进入容器里面获取

解锁 Jenkins

为了确保管理员安全地安装 Jenkins，密码已写入到日志中（不知道在哪里？）该文件在服务器上：

`/root/.jenkins/secrets/initialAdminPassword`

请从本地复制密码并粘贴到下面。

管理员密码

如果docker安装的话就在Jenkins的容器里面，非docker就在宿主机里面

- 2、进入 Jenkins 容器里面命令

```
docker exec -it jenkins /bin/bash
```

- 3、cat 指定文件获取

```
[root@localhost ~]# docker exec -it jenkins /bin/bash
root@27d6f591d0e0:/usr/local/tomcat# cat /root/.jenkins/secrets/initialAdminPassword
6710e2a1d84046658852bbb9b1708549
root@27d6f591d0e0:/usr/local/tomcat#
```

- 4、进入安装插件，选择自定义安装
然后下一步下一步.等待安装.

自定义Jenkins

插件通过附加特性来扩展Jenkins以满足不同的需求。

安装推荐的插件

安装Jenkins社区推荐的插件。

选择插件来安装

选择并安装最适合的插件。

- 5、创建管理用户和密码

创建第一个管理员用户

用户名:	<input type="text" value="admin"/>
密码:	<input type="password" value="....."/>
确认密码:	<input type="password"/>
全名:	<input type="text"/>
电子邮件地址:	<input type="text"/>

7.4 设置全局工具

- 进入 Global Tool Configuration 菜单



Configure System

Configure global settings and paths.



Configure Global Security

Secure Jenkins; define who is allowed to access/use the s



凭据配置

配置凭据的提供者和类型



Global Tool Configuration

Configure tools, their locations and automatic installers.



Reload Configuration from Disk

Discard all the loaded data in memory and reload everythin



Manage Plugins

Add, remove, disable or enable plugins that can extend the



System Information

Displays various environmental information to assist troub

- 配置 Maven 的 setting.xml 文件



Global Tool Configuration

Maven 配置

默认 settings 提供

文件系统中的 settings 文件

文件路径

/usr/local/maven3/conf/settings.xml

默认全局 settings 提供

文件系统中的全局 settings 文件

文件路径

/usr/local/maven3/conf/settings.xml

- 配置 Maven 环境变量

Maven

Maven 安装

新增 Maven

Maven

Name

maven3

MAVEN_HOME

/usr/local/maven3

☐ 自动安装

- 注意：MAVEN_HOME，如果是 Docker 安装 Jenkins 的话，必须是 Jenkins 里面指定路径的路径，否则找不到文件路径报错，不管是 JDK，还是 Git 都是同理。

Maven

Name

maven3

MAVEN_HOME

/usr/local/maven3x

 /usr/local/maven3x is not a directory on the Jenkins master (but perhaps it exists on some agents)

☐ 自动安装

- 配置 Git 环境变量

Git

Git installations

Git

Name

Default

Path to Git executable

/usr/bin/git

☐ 自动安装

Add Git ▼

7.5 系统设置

- SSH remote hosts 目标 SSH 机器连接设置

Hostname: 192.168.1.235 → 目标机器ip
 Port: 22 → 默认22端口
 Credentials: root (node1 root系统用户) → 目标机器的用户名的凭证 添加
 Pty: ☐
 serverAliveInterval: 0
 timeout: 0

➤ Publish over SSH 目标 SSH 机器连接设置

参数名称	参数说明
Passphrase	操作用户 SSH Key 的密码，如果未设置，可以不填
Path to key	jenkins 用户 SSH 私钥 key 的路径
Key	jenkins 用户的 ssh 私钥的内容
name	ssh 连接的名称，可以随意取，有意义就 OK
Host Name	远程服务器的域名/IP，建议是 IP
Username	远程系统的用户名
Remote Directoey	远程目录

➤ Publish over SSH 机器图解

机器235

SSH Server Name: docker_server1 → 这个名字任意起，有意义即可 不同机器，起的名字不一样

Hostname: 192.168.1.235 → 机器IP

Username: root → 用户名

Remote Directory: /home/jenkins/workspace/ → 远程目标机器文件目录

Success → 测试是否成功

机器236

SSH Server Name: docker_server2

Hostname: 192.168.1.236

Username: root

Remote Directory: /home/jenkins/workspace/

高级... Test Configuration 删除

7.6 添加全局用户名凭证

Jenkins > 凭据 > 系统 > 全局凭据 (unrestricted)

返回到凭据域列表

添加凭据

类型

Username with password

范围

全局 (Jenkins, nodes, items, all child items, etc)

用户名

admin

密码

.....

ID

描述

确定

7.7 凭证类型方式

unrestricted)

类型

Username with password

Username with password

Docker Host Certificate Authentication

SSH Username with private key

Secret file

Secret text

Certificate

描述

用户名和密码方式进行连接

docker组件配置授权方式连接

以系统SSH的私钥方式进行连接

以密钥文件方式连接

以密钥文本方式连接

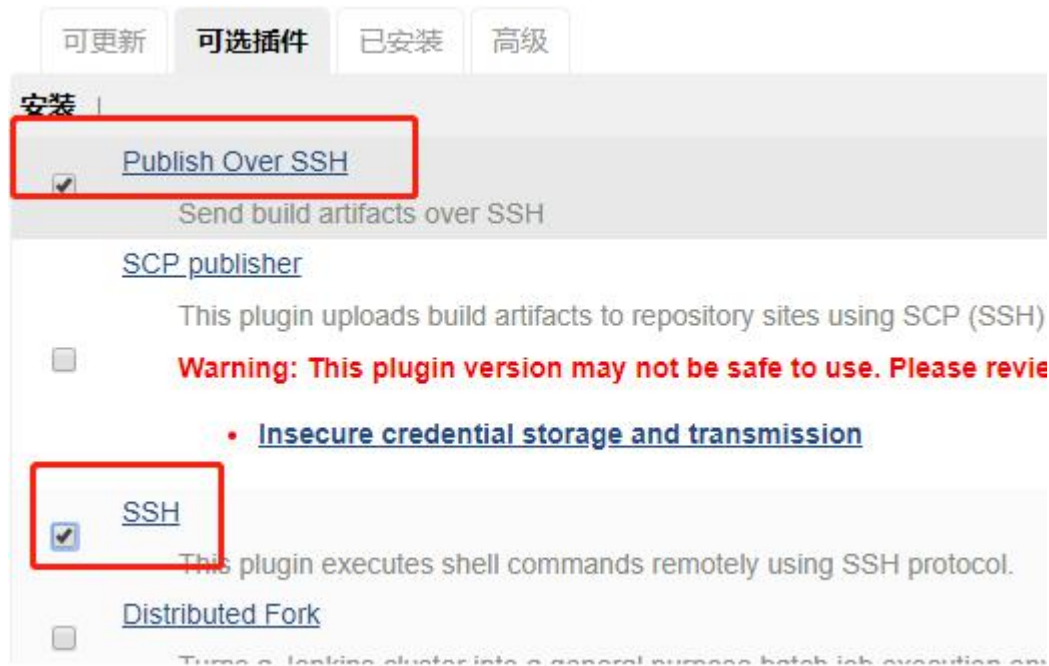
以凭证方式连接

确定

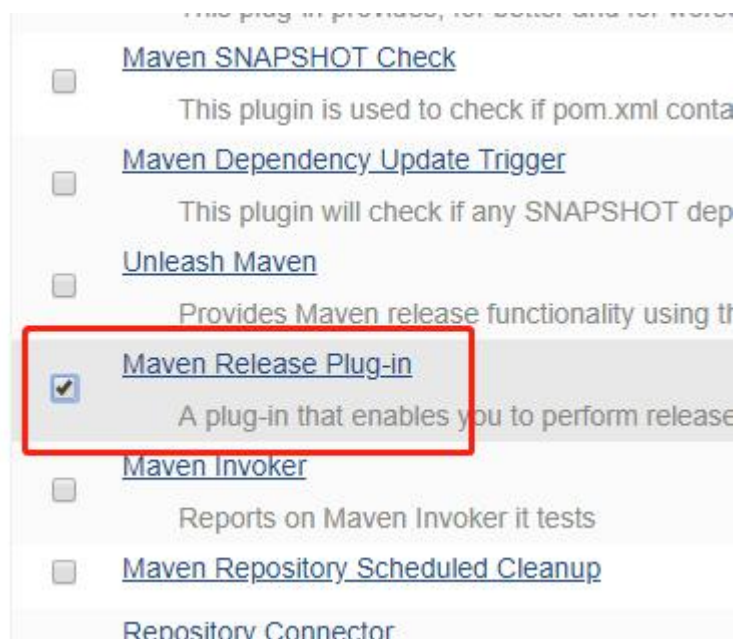
7.8 插件安装

【注意】安装插件这边非常重要,整个部署会依赖这些插件才能整合 devops 部署

- Publish Over SSH 用于 SSH 发布
- SSH 用于 SSH 连接服务器



- Maven Release Plug-in 用于 Maven 项目编译



- Git Parameter Plug-In 用于 Git 的动态参数获取

Git Parameter

Name: git_tag

Description: git标签参数

Parameter Type: Tag (selected)

Default Value: Tag

添加参数 ▼

is required. Example origin/master

设置git参数类型，在部署可以根据标签、分支，版本等进行部署

- 根据\$git_tag 动态获取 git 的分支或者标签

Git

Repositories

Repository URL: ssh://git@192.168.1.235:7003/root/springboot.git

Credentials: root (node2 git) 添加 ▼

Branches to build

Branch Specifier (blank for 'any'): \$git_tag

源码库浏览器: (自动)

Additional Behaviours: 新增 ▼


动态获取git参数的值

- 根据构建的 git_tag 参数的选择

Maven project springboot_test

需要如下参数用于构建项目:

appName	springboot
	项目名称
server	192.168.1.236
	服务地址
version	0.0.1
	版本号
userName	root
	用户名
serverPath	/home/jenkins/workspace/springboot_test/
	目标服务器文件路径
targetServerPath	/home/jenkins/workspace/
	拷贝到远程的目录
port	7011
	端口
env	dev
	发布的环境: dev 联调环境, test 测试环境, pre 预发布环境, fix 修复环境, prod 生产环境
git_tag	dev test



8、私钥与公钥详解

8.1 为什么要公钥和私钥

1. 由于 git 拉取或推送代码的时候到私钥和公钥进行 SSH 通信
2. 在一套 Jenkins 部署多台目标机器应用, 此时免密远程拷贝代码的时候私钥和公钥进行 SSH 通信体现尤为重要

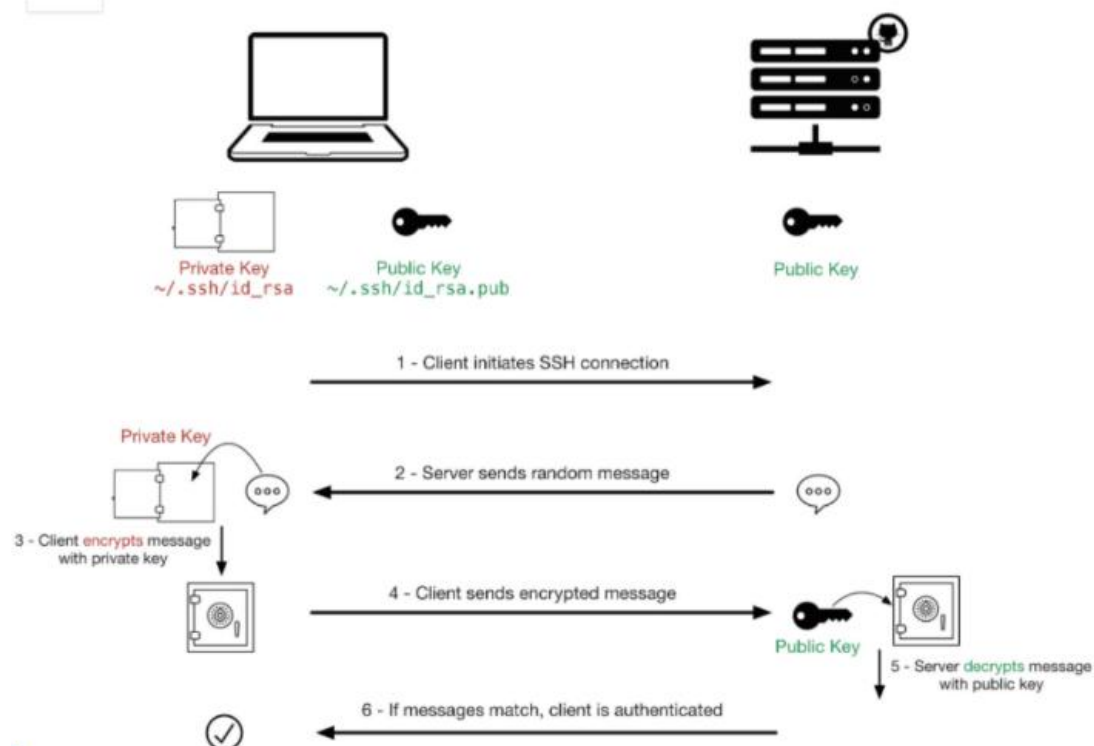
8.2 公钥与公钥的区别

- 公钥 (Public Key) 与私钥 (Private Key) 是通过一种算法得到的一个密钥对 (即一个公钥和一个私钥), 公钥是密钥对中公开的部分, 私钥则是非公开的部分。公钥通常用于加密会话密钥、验证数字签名, 或加密可以用相应的私钥解密的数据。
- 通过这种算法得到的密钥对能保证在世界范围内是独一的。使用这个密钥对的时候, 如

果用其中一个密钥加密一段数据，必须用另一个密钥解密。比如用公钥加密数据就必须用私钥解密，如果用私钥加密也必须用公钥解密，否则解密将不会成功。

8.3 公钥与公钥通信图解

- 多台机器免密登录的图解



8.4 公钥与公钥通信原理

- 首先在客户端生成一对密钥(ssh-keygen)
- 并将客户端的公钥 ssh-copy-id 拷贝到服务端
- 当客户端再次发送一个连接请求，包括 ip 用户名
- 服务端将使用客户端的请求后，会到 authorized_keys 中查找, 如果有响应的 IP 和用户，就会随机生成一个字符串。如：ABC
- 服务端将用户客户端拷贝过来的公钥镜像加密，然后发送给客户端
- 得到服务端发来的消息后，客户端会使用私钥进行解密，然后将解密后的字符串发送给服务端
- 服务端接受到客户端发来的字符串后，跟之前的字符串进行对比，如果是一致，就允许免密登录

8.5 公钥与公钥生成&免密登录操作

8.5.1 公钥和私钥生成

- 指定一个名称生成公钥和私钥

```
ssh-keygen -t rsa -C "root"
```

- 不指定名称生成公钥和私钥

```
ssh-keygen -t rsa
```

- 下面的操作一直回车即可.此操作跟 Windows、Mac 等系统同等，只不过存储的文件路径不一样.

```
Administrator@DESKTOP-36L4HH7 MINGW64 ~/.ssh
$ ssh-keygen -t rsa -C 'github'
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): id_rsa_github
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa_github.
Your public key has been saved in id_rsa_github.pub.
The key fingerprint is:
SHA256:X2CJbXtWG2yB5yHti+etzsv7S5LJwvAU8CCQeW5RHjk github
The key's randomart image is:
+---[RSA 2048]---+
|      .+..=.  o.  |
|    o ooE=.o.+  |
|   o o.B0 ==.  |
|    o o o.ooo  |
|   . S...+...  |
|    .+=+o =  |
|    .+ B o  |
|    .o+ .  |
|    .B*o  |
+-----[SHA256]-----+
```

8.5.2 免密登录操作

- ✧ 拷贝实现三种方式

- cat 命令追加公钥文件内容指定文件

```
cat id_rsa.pub >> authorized_keys
```

- 方法一、将 master 生成的公钥拷贝到 slave(从机或副机)指定主机名称进行远程拷贝

```
scp authorized_keys slave:/root/.ssh/
```

- 方法 2、根据机器 IP 进行拷贝

```
scp authorized_keys root@192.168.1.236:/root/.ssh/
```

```
scp -p ~/.ssh/id_rsa.pub root@192.168.1.236:/root/.ssh/authorized_keys
```


- 方法 3、通过 `ssh-copy-id` 命令进行拷贝

```
ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.1.236
```

- 开启必要的参数,进入 `vim /etc/ssh/sshd_config` 配置, 此操作非常重要, 由于不同操作系统不一样, 网上很多文章资料都没提到此操作.



The image shows a terminal window displaying the contents of the `/etc/ssh/sshd_config` file. The configuration includes the following lines:

```
#LoginGraceTime 2m
#PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
RSAAuthentication yes
PubkeyAuthentication yes
StrictModes no
# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile .ssh/authorized_keys
```

Annotations with red boxes and arrows:

- A red box highlights `RSAAuthentication yes` and `PubkeyAuthentication yes`. An arrow points from this box to a text box containing:
RSAAuthentication 开启RSA授权
StrictModes 禁止严格模型, 这个非常关键
PubkeyAuthentication 开启公钥授权
- A red box highlights `AuthorizedKeysFile .ssh/authorized_keys`. An arrow points from this box to a text box containing:
授权秘钥文件

- 启动 ssh 服务

```
/bin/systemctl start sshd.service
```

- 启动系统

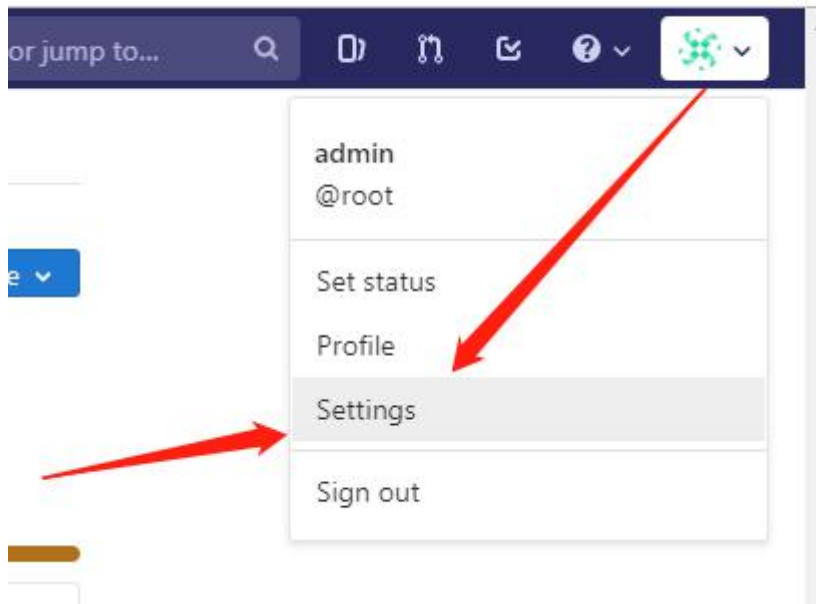
```
reboot
```

- ssh 远程登录, 第一次登录需要密码, 后面就不需要密码了

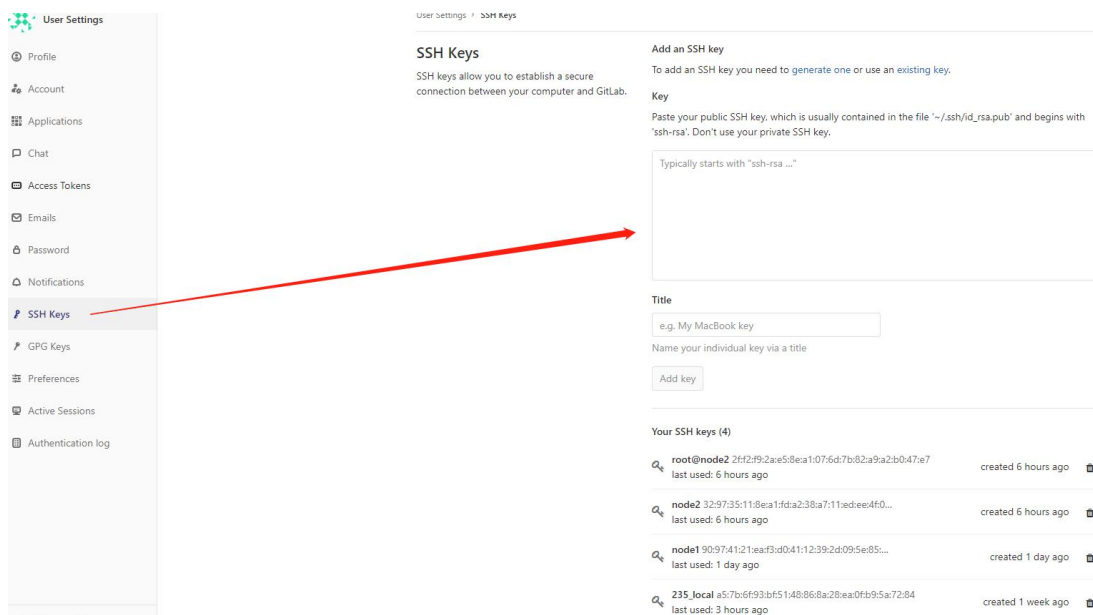
```
ssh slave 或 ip
ssh 192.168.1.236
```

8.6 配置 Gitlab 公钥

- 登录 Gitlab 平台,找到用户下面 Settings



➤ 找到 SSH Key 把生成的公钥拷贝与黏贴保存.



9、Docker、Jenkins 等编译镜像与部署详解

9.1、需要准备的工作有哪些

1、打开 IDEA 或 Eclipse 新建一个 SpringBoot 的应用.

2、以上菜单所有工作准备好就可以进入 Jenkins 操作了（gitlab、Maven、jdk、Jenkins 等）

9.2、SpringBoot 配置和代码详解

9.2.1 SpringBoot 简要

- 随着动态语言的流行 (Ruby、Groovy、Scala、Node.js), Java 的开发显得格外的笨重：繁多的配置、低下的开发效率、复杂的部署流程以及第三方技术集成难度大。
- 在上述环境下, Spring Boot 应运而生。它使用“习惯优于配置”（项目中存在大量的配置, 此外还内置了一个习惯性的配置, 让你无需手动进行配置）的理念让你的项目快速的运行起来。使用 Spring Boot 很容易创建一个独立运行（运行 Jar, 内嵌 Servlet 容器）准生产级别的基于 Spring 框架的项目, 使用 Spring Boot 你可以不用或者只需很少的 Spring 配置。

9.2.2 工程的 pom.xml 配置

```
<dependencies>
    <!-- Springboot 依赖的 Jar 包 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Springboot 热部署 jar-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>

    <!--yml 配置文件提示插件-->
    <dependency>
```

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>

<!-- spring-boot 测试 jar -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

</dependencies>
<build>
    <finalName>springboot</finalName>
    <!-- 一定要声明如下配置 打包 xml 到 Jar 包 -->
    <!-- <resources>
        <resource>
            <directory>src/main/java</directory>
            是否替换资源中的属性
            <filtering>false</filtering>
        </resource>
    </resources>
    <sourceDirectory>${project.basedir}/src/main/java</sourceDirectory>
    -->
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <!-- 默认支持 jdk1.8 编译 -->
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

```

        </executions>
    </plugin>
    <!--docke rmaven 编译插件-->
    <plugin>
        <groupId>com.spotify</groupId>
        <artifactId>docker-maven-plugin</artifactId>
        <version>0.4.12</version>
        <configuration>
            <dockerDirectory>${project.basedir}</dockerDirectory>
            <resources>
                <resource>
                    <targetPath></targetPath>
                    <directory>${project.build.directory}</directory>
                    <include>${project.build.finalName}.jar</include>
                </resource>
            </resources>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
            <archive>
                <manifest>
                    <mainClass>com.flong.SpringbootApplication</mainClass>
                </manifest>
            </archive>
        </configuration>
    </plugin>
</plugins>
</build>

```

9.2.3 no main manifest attribute 错误解决

➤ 配置工程主入口

```

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <mainClass>com.flong.SpringbootApplication</mainClass>
            </manifest>
        </archive>
    </configuration>
</plugin>

```

```
</archive>
</configuration>
</plugin>
```

9.2.4 env 环境变量文件

- 用于设置环境动态参数,文件是以.env 为格式

```
JAVA_OPTS_DEFAULT=-Xmx512m
```

9.2.5 Dockerfile 打包工程镜像细讲

- 以开发环境的 Dockerfile 为例,如果是测试环境则,把所有路径包含 springboot_dev 改成 springboot_test

```
FROM frovlad/alpine-oraclejdk8:slim
MAINTAINER jilongliang@sina.com
RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
RUN mkdir -p /home/devsoft/springboot_dev
WORKDIR /home/devsoft/springboot_dev
EXPOSE 7011
ADD ./target/springboot.jar ./
CMD java ${JAVA_OPTS_DEFAULT} -Djava.security.egd=file:/dev/./urandom -jar springboot.jar
```

- 参数说明

参数	参数说明
FROM	基于什么环境构建, 一个 dockerfile 必须要有一个 FROM 关键字
MAINTAINER	构建作者
RUN ln -sf	设置时区
mkdir	创建指定目录进行运行, -p 表示创建父级下面的所有文件
WORKDIR	工作目录, 通过 docker exec -it 交互式终端模式进入容器之后进入的目录.
EXPOSE	暴露端口, 可以支持多个端口, 用户空格隔开
ADD	添加

- WORKDIR 工作目录说明

进入容器此时会有一个.jar 是在 Dockerfile 的 ADD 添加进去

```
docker exec -it 容器名称或容器 id /bin/sh
或使用 sh 和 bash 要看 COMMAND, -it
docker exec -it 容器名称或容器 id/bin/bash
```

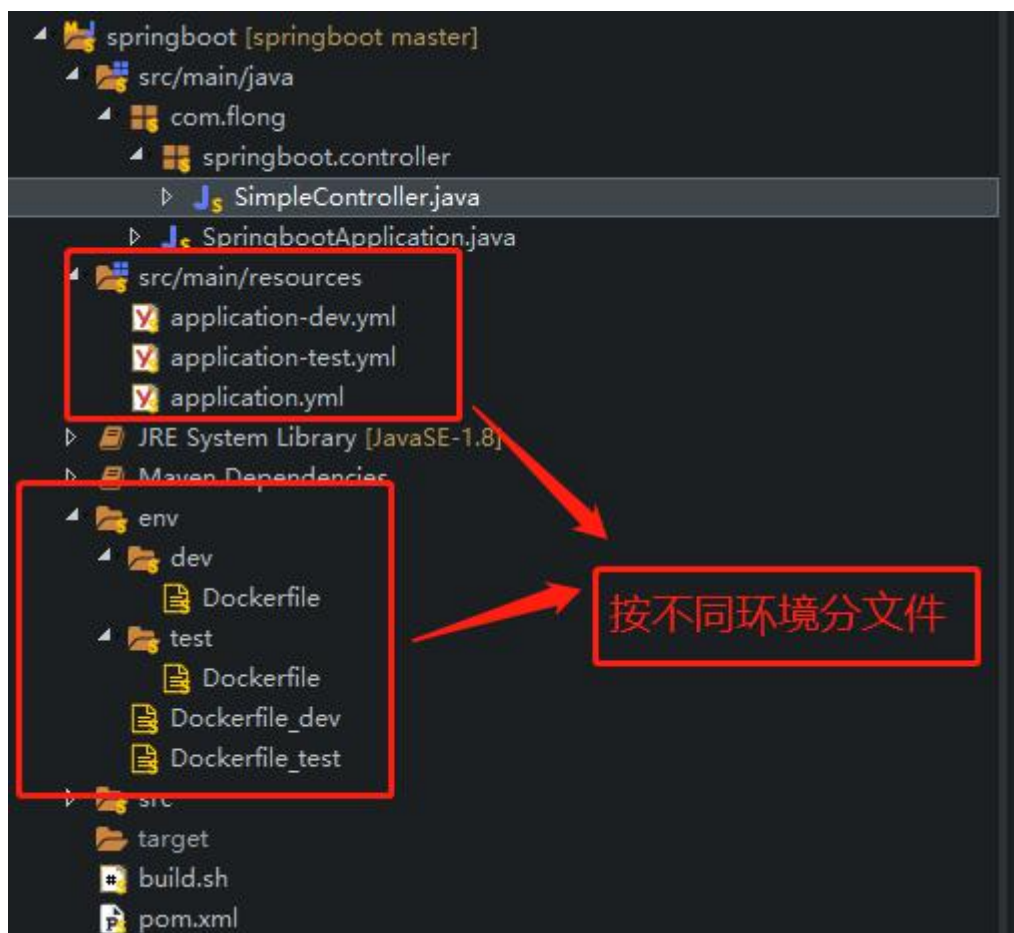
```
[root@node2 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
d994a9fa9b61   springboot:0.0.1  "/bin/sh -c 'java ${_}"  2 days ago    Up 2 days    0.0.0.0:7011->7011/tcp    springboot
27d6f591d0e0   jenkins:latest   "catalina.sh run"        12 days ago   Up 5 days    0.0.0.0:9001->8080/tcp    jenkins

[root@node2 ~]# docker exec -it springboot /bin/sh
/home/jenkins/workspace/springboot_dev # ls
springboot.jar
/home/jenkins/workspace/springboot_dev # cd /
/ # ls
bin    dev    etc    home  lib    lib64  media  mnt    proc  root  run    sbin   srv    sys    tmp    usr    var
```

通过docker exec -it 交互式的命令进入运行的镜像容器，此时会切换到Dockerfile定义的WORKDIR工作目录。此时证明，这个可以根据自定义要进入容器的目录。

9.2.6 工程文件结构

- env 表示环境变量文件
- dev 表示开发环境
- test 表示测试环境



9.2.7 build.sh 文件 shell 脚本详解

- 注意点：经过测试动态变量的【等号】不能有空格和 tab 键置位,否则获取不了值，而且在 shell 脚本代码里面不支持空格格式化，支持 tab 置位格式化。
- 在终端(ssh 软件端)或 Jenkins 客户端 shell 命令,『位置变量』的参数以空格隔开。如：

```
sh build.sh 192.168.1.235 springboot 0.0.1 7011 /home/jenkins/workspace/springboot_dev
```

- # \$IMG_NAME:\$IMG_VERSION 这个 IMG_VERSION 版本(tag)参数不指定默认 latest
- 注意点: 通常情况下 Docker 是默认执行 Dockerfile, 但是可以自定义后缀文件进行编译, 前提必须要-f (force) 强制指定文件进行运行

```
#!/usr/bin/env bash
# 动态变量的【等号】不能有空格和 tab 键置位, 否则获取不了值, 而且在 shell 脚本
# 代码里面不支持空格格式化, 支持 tab 置位格式化。
# 在终端(ssh 软件端)或 Jenkins 客户端 shell 命令, 参数以空格隔开。
# 如: sh build.sh 192.168.1.235 springboot 0.0.1 7011
# /home/jenkins/workspace/springboot_dev
IMG_SERVER="$1"
IMG_NAME="$2"
IMG_VERSION="$3"
IMG_PORT="$4"
RUN_ENV="$5"
IMG_PATH="$6"

echo "服务地址: $IMG_SERVER"
echo "工程镜像名称: $IMG_NAME"
echo "工程版本号: $IMG_VERSION"
echo "工程端口: $IMG_PORT"
echo "服务环境: $RUN_ENV"

echo "create $IMG_PATH"
mkdir -p $IMG_PATH

# 私服访问 url 路径和编译之后镜像文件存放到指定路径固定, 不动态参数进行处理传
# 值。
REGISTRY_URL="192.168.1.235:5000"
IMG_TAR_GZ_PATH="/home/img_tar_gz_path/"

# 判断动态参数不为空字符串的时候才执行下面操作
if [ "$IMG_SERVER" != "" ] && [ "$IMG_NAME" != "" ] && [ "$IMG_VERSION" != "" ] && [ "$IMG_PORT" != "" ]; then

    echo " .....进入删除 Container & Images 操作 ....."
    docker rm -f $(docker ps -a | grep $IMG_NAME | awk '{ print $1 }')
    docker rmi $(docker images | grep $IMG_NAME | awk '{ print $3 }')

    # $IMG_NAME:$IMG_VERSION 这个 IMG_VERSION 版本(tag)参数不指定默认
    # latest, 通过不同参数执行不同环境文件
    # -f 表示强制指定 Dockerfile 文件进行编译
```

```

echo " .....进入 Building & Images 操作 ..... "

#方法 1、指定不同文件存放默认的 Dockerfile，使用-f 进行强制编译
#docker build -t $IMG_NAME:$IMG_VERSION -f
$IMG_PATH"env/"$RUN_EVN/Dockerfile $IMG_PATH

#方法 2、跟据不同 Dockerfile 文件的后缀进行编译不同环境的文件
docker build -t $IMG_NAME:$IMG_VERSION -f
$IMG_PATH"env/"Dockerfile_$RUN_EVN $IMG_PATH

# 将镜像打一下标签，然后按照标签进行推送到私服里面，标签名就以服务名即可
docker tag $IMG_NAME:$IMG_VERSION
$REGISTRY_URL/$IMG_NAME:$IMG_VERSION

# 推镜像到私服里面
docker push $REGISTRY_URL/$IMG_NAME:$IMG_VERSION

# 创建目录
mkdir -p $IMG_TAR_GZ_PATH
# 保存编译之后镜像文件存放到指定路径
docker save $IMG_NAME -o $IMG_TAR_GZ_PATH/$IMG_NAME.tar.gz

echo " .....进入 Runing 操作 ....."
docker run -d --network simple-network --restart=always
--env-file=./.env -e spring.profiles.active=$RUN_EVN
--expose=$IMG_PORT --name=$IMG_NAME -p $IMG_PORT:$IMG_PORT
$IMG_NAME:$IMG_VERSION

echo " .....Build & Run Finish Success~...."
else
echo " .....Illegal Command Operation ....."
fi

```

- docker save 命令是保存编译的 **tar.gz** 或 **tar** 压缩文件，此命令也可以备份运行的镜像，所以是一个非常重要的命令。语法如：

```

docker save 镜像名 -o 路径/镜像名.tar.gz
或
docker save 镜像名 -o 路径/镜像名.tar

```

- 查看文件


```
springboot.tar.gz
[root@node1 img_tar_gz_path]# pwd
/home/img_tar_gz_path
[root@node1 img_tar_gz_path]# ls
springboot.tar.gz
[root@node1 img_tar_gz_path]#
```

查看文件保存的路径

- docker save 指定镜像备份成压缩文件从镜像里面备份镜像与备份镜像源码.tar 或.gz 后缀文件,备份之后就可以将容器和镜像删除,假设要恢复此版本,就可以使用 docker load -i 进行重新生成可执行的镜像运行即可。

- [docker save](#) 命令语法

```
sudo docker save -o /home/img_tar_gz_path/工程实例名 20190813.tar 镜像 id 或名称
```

- docker save 参数说明

参数	参数说明
-o	-o 代表--output 文件输出到哪里

- docker load 命令是用于导入使用 [docker save](#) 命令导出的镜像,此命令非常重要,由于有些客户要求项目工程要求部署在内网,此时这个命令在无网络的内网情况下部署项目的时候就体现它重要的地位了.语法 docker load [OPTIONS], 在加载的过程有点慢,因为文件有点大,其中显示 Loady Layer [=====]输出信息,证实镜像是分层关系。

```
docker load -i /home/img_tar_gz_path/springboot.tar.gz
```

- 截图说明

```
[root@node1 img_tar_gz_path]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
192.168.1.235:5000/springboot  0.0.1              a0e693baec79       17 hours ago       187MB
springboot           0.0.1              a0e693baec79       17 hours ago       187MB
<none>               <none>             8cac5025692b       18 hours ago       187MB
redis                latest             bb0ab8a99fe6       13 days ago        95MB
tomcat               latest             5377fd8533c3       4 weeks ago        506MB
jenkins              latest             ac7a43918252       5 weeks ago        583MB
minio/minio          latest             64d028295177       6 weeks ago        60.7MB
gitlab/gitlab-ce     latest             8e28c88b6a21       7 weeks ago        1.85GB
openjdk              8-jdk-alpine      a3562aa0b991       2 months ago       105MB
mysql                5.6               73829d7b6139       2 months ago       256MB
registry             2                 f32a97de94e1       4 months ago       25.8MB
registry             latest            f32a97de94e1       4 months ago       25.8MB
frolvlad/alpine-oraclejdk8  slim              7372598bbfc2       6 months ago       165MB
redis                3.2               87856cc39862       9 months ago       76MB
```

[root@node1 img_tar_gz_path]# docker rmi -f a0e693baec79 8cac5025692b

Untagged: 192.168.1.235:5000/springboot:0.0.1

Untagged: 192.168.1.235:5000/springboot@sha256:3f92b822065068d6241a3d7d79c6e398502de553997f844555e29e81c5c924c2

Untagged: springboot:0.0.1

Deleted: sha256:a0e693baec795f685b3e492e4af28c6bd542ddb57b0257d3c444e5f13ff763ba

Deleted: sha256:4fec3a6a520d5dde749eacdcc946ec70dd8d83a33ac2379a55a4f0258433c8b5

Deleted: sha256:3c42d9022e5cdb6325118dd2886bc524b9801106178730fb447968f516bd95c8

Deleted: sha256:8cac5025692b6eb65085d5409c25744b814b23ad3e4fd158f72c05275df4d87

Deleted: sha256:73c2284ad9027dceffa628fb84f000d334c2a8945465724c442f33ebe1f6a0ac

Deleted: sha256:439dc751072d8677247e33de16f6c22203d48617adc17ac408184d8b0ece6d1d

Deleted: sha256:cc90a8c16c6b0687d0f1d1e1b9104177f759dabebf2fd9b401e05b8e2556ded6

Deleted: sha256:c358f66a2273d6b15547d22c902eac62c1d62d0898b1bbb2d53f967710029261

Deleted: sha256:7e04e4574a11f15bddd50ab6e16868f5e5d7a5e19289161babd776dd0a9a3e1c

Deleted: sha256:dd27fc00c3c766a4b365d9c6112a4683ed36b93106f3c996fba27305807658ab

Deleted: sha256:28b07890f0b7119837a6c4f00665bd7f123fbc1a062eeb3b235cc1d96b791fe

Deleted: sha256:045e28d91331e2eb1dabe6380f47cceff77c89f3790c78a86da209e7920c9f45d

Deleted: sha256:029655c7962d46bfcdeec45231116caf1a500d110e20205b1ff266e96cd0a31

[root@node1 img_tar_gz_path]# docker load -i springboot.tar.gz

c006d402fef2: Loading layer [=====]>] 2.048kB/2.048kB

399fb9e3dcde: Loading layer [=====]>] 3.584kB/3.584kB

4f946d617393: Loading layer [=====]>] 21.03MB/21.03MB

Loaded image: springboot:0.0.1

```
[root@node1 img_tar_gz_path]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
springboot           0.0.1              a0e693baec79       17 hours ago       187MB
redis                latest             bb0ab8a99fe6       13 days ago        95MB
tomcat               latest             5377fd8533c3       4 weeks ago        506MB
```

删除镜像, 其中2个是相同的镜像

重新加载镜像源码文件

查看镜像列表

➤ 参数说明

参数	参数说明
-i	input 输入参数指定导入的文件
-q	quiet 精简输出信息（抑制负载输出）

- docker tag 和 docker push 命令是一起结合使用，先 tag 后 push，每个镜像名和版本是以冒号区分，而 docker pull 根据情况使用。

```
# 将镜像打一下标签，然后按照标签进行推送到私服里面，标签名就以服务名即可
docker tag 镜像名:版本号 私服路径/镜像名:版本号
# 推镜像到私服里面
docker push 私服路径/镜像名:版本号
```

➤ 查看镜像

```
[root@node1 data]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
springboot          0.0.1              8cac5025692b       2 minutes ago      187MB
192.168.1.235:5000/springboot 0.0.1              8cac5025692b       2 minutes ago      187MB
<none>              <none>             5fb8b1577b44       11 minutes ago     187MB
redis               latest             bb0ab8a99fe6       12 days ago        95MB
tomcat              latest             5377fd8533c3       4 weeks ago        506MB
jenkins             latest             ac7a43918252       5 weeks ago        583MB
minio/minio         latest             64d028295177       6 weeks ago        60.7MB
gitlab/gitlab-ce    latest             8e28c88b6a21       7 weeks ago        1.85GB
openjdk             8-jdk-alpine      a3562aa0b991       2 months ago       105MB
mysql               5.6                73829d7b6139       2 months ago       256MB
registry            2                  f32a97de94e1       4 months ago       25.8MB
frolvlad/alpine-oraclejdk8 slim               7372599bbfc2       6 months ago       165MB
redis               3.2                87856cc39862       9 months ago       76MB
[root@node1 data]#
```

- 浏览器验证 docker push 推送上私服的镜像

http://192.168.1.235:5000/v2/_catalog

- 火狐浏览器访问显示信息

192.168.1.235:5000/v2/_catalog

JSON 原始数据 头

复制

响应头

Content-Length 32
 Content-Type application/json; charset=utf-8
 Date Thu, 08 Aug 2019 15:46:39 GMT
 Docker-Distribution-API-Version registry/2.0
 X-Content-Type-Options nosniff

请求头

Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
 Accept-Encoding gzip, deflate
 Accept-Language zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
 Cache-Control max-age=0
 Connection keep-alive
 Host 192.168.1.235:5000
 Upgrade-Insecure-Requests 1
 User-Agent Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0

192.168.1.235:5000/v2/_catalog

JSON 原始数据 头

保存 复制 全部折叠 全部展开

repositories:

0: "springboot"

- 虚悬镜像：在 docker 编译不成功会或者是新版本覆盖旧版本归类为虚悬镜像,生成这个个镜像既没有仓库名，也没有标签，均为 <none>。一般来说，虚悬镜像已经失去了存在的价值，是可以随意删除的，可以用下面的命令删除。

```
[root@node1 data]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
192.168.1.235:5000/springboot 0.0.1             8cac5025692b       27 seconds ago     187MB
springboot          0.0.1             8cac5025692b       27 seconds ago     187MB
<none>              <none>            5fb8b1577b44       9 minutes ago      187MB
redis               latest            bb0ab8a99fe6       12 days ago        95MB
tomcat              latest            5377fd8533c3       4 weeks ago        506MB
jenkins             latest            ac7a43918252       5 weeks ago        583MB
minio/minio         latest            64d028295177       6 weeks ago        60.7MB
gitlab/gitlab-ce    latest            8e28c88b6a21       7 weeks ago        1.85GB
openjdk             8-jdk-alpine     a3562aa0b991       2 months ago       105MB
mysql               5.6              73829d7b6139       2 months ago       256MB
registry            2                f32a97de94e1       4 months ago       25.8MB
frolvlad/alpine-oraclejdk8 slim              7372599bbfc2       6 months ago       165MB
redis               3.2              87856cc39862       9 months ago       76MB
[root@node1 data]# docker
```

9.2.8 不同环境的配置文件

➤ 配置文件说明

配置文件	配置文件
application.yml	默认配置文件
application-dev.yml	开发配置文件
application-test.yml	测试配置文件

➤ 开发配置文件内容

```
server:
  port: 7011
  runEvn: '开发环境'
```

➤ 测试配置文件内容

```
server:
  port: 7011
  runEvn: '测试环境'
```

9.2.9 Controller 测试代码

```
@RestController
public class SimpleController {
    //读取配置动态参数
    @Value("${runEvn}")
    private String runEvn;

    @GetMapping("/test")
    public String test() {
        return "this spring boot " + runEvn + " date long " +
            System.currentTimeMillis();
    }
}
```

9.3、非多台机器免密远程登录&Jenkins 部署流程详解

9.3.1 特别说明

- 以开发环境为例子进行说明
- 开发环境部署目标机器是与 Jenkins 机器同一台机器，一般情况，Jenkins 是单独一台机

器，这里为了节省自身电脑内存，故放在同一台机器进行演示与学习。

9.3.2 新建 maven 工程

- 点击 Jenkins 的新建任务菜单

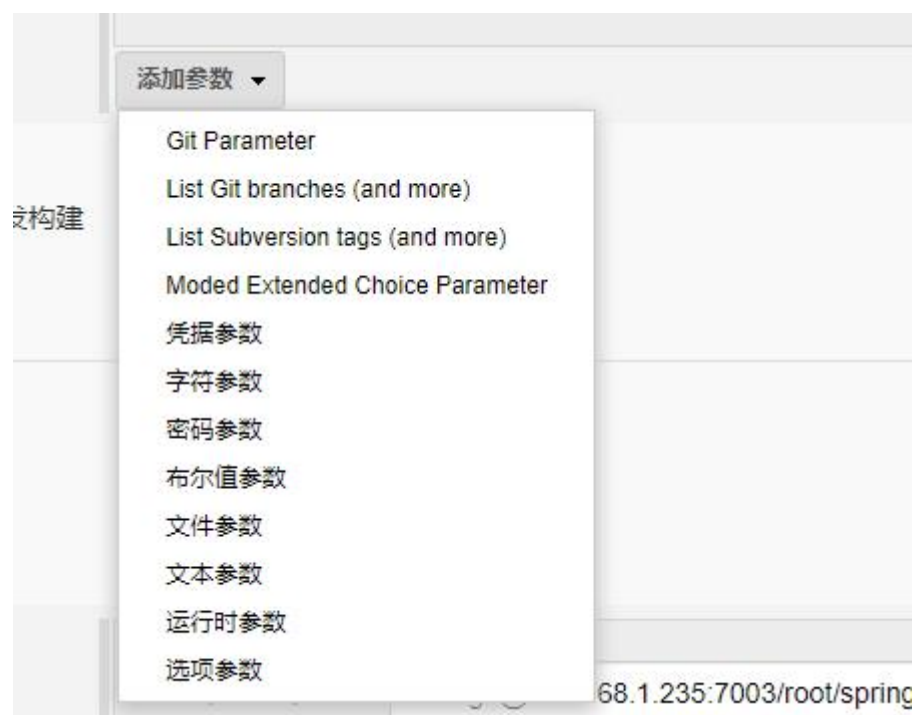


The image shows the 'New Item' dialog box in Jenkins. At the top, there is a text input field labeled '输入一个任务名称' (Enter a task name) with a red border and a '» 必填项' (» Required) label below it. Below this, there are three selectable options, each with an icon and a description:

- 构建一个自由风格的软件项目** (Build a free-style software project): This is Jenkins's main function. Jenkins will combine any SCM and any build system to...
- 构建一个maven项目** (Build a maven project): This option is highlighted with a red border. The description says: '构建一个maven项目. Jenkins利用你的POM文件, 这样可以大大减轻构...' (Build a maven project. Jenkins uses your POM file, which can greatly reduce the burden of building...)
- 流水线** (Pipeline): Carefully organize a task that can run on multiple nodes. Suitable for building pipelines...

9.3.3 参数化构建过程说明

- 添加参数



➤ 参数说明 以开发环境为案例

参数名称	参数值	参数描述
appName	springboot	项目名称
server	机器 ip 地址	服务地址
version	0.0.1	版本号，可以根据情况定义
serverPath	/home/jenkins/workspace/springboot_dev/	远程存放代码的路径或者说是远程的文件目录
port	7011	端口
env	dev	发布的环境;dev 联调环境，test 测试环境，pre 预发布环境，fix 修复环境，prod 生产环境

9.3.4 源码管理

源码管理

☐ 无
☒ Git

Repositories

Repository URL:

Credentials:

Branches to build

Branch Specifier (blank for 'any'):

源码库浏览器:

目标机器生成的密钥，在添加按钮进行设置

以什么方式进行部署，比如：标签，分支

9.3.5 Build 编译设置

Build

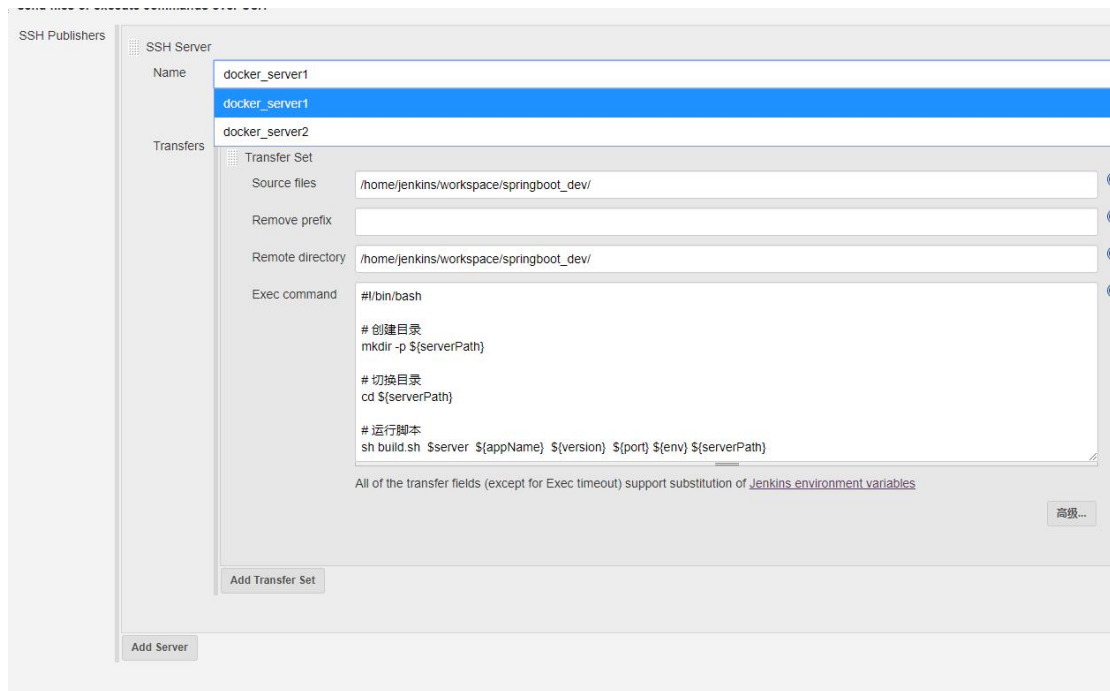
Root POM:

Goals and options:

maven编译工程的命令，忽略测试类进行编译

9.3.6 SSH Publishers 设置

- 其中 SSH Server Name 就是在 <http://jenkins 地址:端口/jenkins/configure> 设置好进行选择
- Transfer Set Source file 传输文件的路径,可以使用参数构建的占位符`${serverPath}`获取
- Remote directory 远程文件目录，同理也参数构建的占位符`${serverPath}`获取



- SSH Publishers shell 脚本 `#!/bin/bash` 表示告诉终端使用 **bash 解析器** 进行执行，而且只有第一行 `bash` 才有效。

```
#!/bin/bash

# 创建目录
mkdir -p ${serverPath}

# 切换目录
cd ${serverPath}

# 运行脚本
sh build.sh $server ${appName} ${version} ${port} ${env} ${serverPath}
```

9.3.7 构建与编译部署项目

- 截图的构建参数都是在参数化构建过程配置的参数

jenkins

springboot_dev

返回面板

状态

修改记录

工作空间

Build with Parameters

删除 Maven project

配置

模块

重命名

Build History

构建历史

find

X

#41

2019-7-16 下午12:00

#40

2019-7-16 上午11:58

#39

2019-7-16 上午11:52

#38

2019-7-16 上午11:48

#37

2019-7-16 上午11:46

#36

2019-7-16 上午11:41

#35

2019-7-16 上午11:36

Maven project springboot_dev

需要如下参数用于构建项目:

appName

springboot

项目名称

server

192.168.1.235

服务地址

version

0.0.1

版本号

serverPath

/home/jenkins/workspace/springboot_dev/

port

7011

工程端口

env

dev

dev

test

pre

fix

prod

境;dev 联调环境, test 测试环境, pre 预发布环境, fix 修复环境, prod 生产环境

开始构建

9.3.8 运行&部署结果

- 编译结果

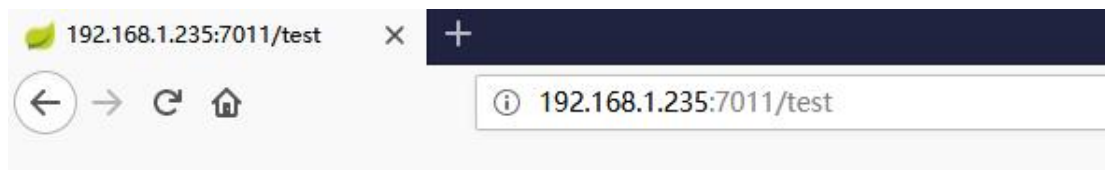
```

Remove one or more images
.....进入Building & Images 操作 .....
Sending build context to Docker daemon 21.28MB

Step 1/8 : FROM frolvlad/alpine-oraclejdk8:slim
--> 7372599bbfc2
Step 2/8 : MAINTAINER jilongliang@sina.com
--> Running in ea7e2acf3552
Removing intermediate container ea7e2acf3552
--> e8a707ee7699
Step 3/8 : RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
--> Running in 9d0f34e57a61
Removing intermediate container 9d0f34e57a61
--> 35221ad212b6
Step 4/8 : RUN mkdir -p /home/jenkins/workspace/springboot_dev
--> Running in 2ec15644b299
Removing intermediate container 2ec15644b299
--> 146562ae8ef4
Step 5/8 : WORKDIR /home/jenkins/workspace/springboot_dev
--> Running in e5bcd92c0f7f
Removing intermediate container e5bcd92c0f7f
--> 60b554eb039e
Step 6/8 : EXPOSE 7011
--> Running in 7c9f935ec812
Removing intermediate container 7c9f935ec812
--> a46cf48ac213
Step 7/8 : ADD ./target/springboot.jar ./
--> 997ca653cfee
Step 8/8 : CMD java ${JAVA_OPTS_DEFAULT} -Djava.security.egd=file:/dev/./urandom -jar springboot.jar
--> Running in 3a0681d9aa58
Removing intermediate container 3a0681d9aa58
--> 7b33b3fcc8fe
Successfully built 7b33b3fcc8fe
Successfully tagged springboot:0.0.1
.....进入Runing操作 .....
4e2dc5417b591d012826259637a089bc4e207a2e0937d54a566d436e41c9f8f1
.....Build & Run Finish Success~....
SSH: EXEC: completed after 6,048 ms
SSH: Disconnecting configuration [docker_server1] ...
SSH: Transferred 0 file(s)
Finished: SUCCESS

```

➤ 部署&运行结果



this is spring boot 开发环境,date long 1565282475860



9.4、多台机器免密远程登录&Jenkins 部署流程详解

9.4.1 特别说明

- 以测试环境为例子进行说明，步骤流程几乎一样，唯一是在 SSH Publishers 和源码存放路径不一样
- 测试环境部署目标机器是与 Jenkins 机器不同一台机器

9.4.2 新建 maven 工程

- 点击 Jenkins 的新建任务菜单

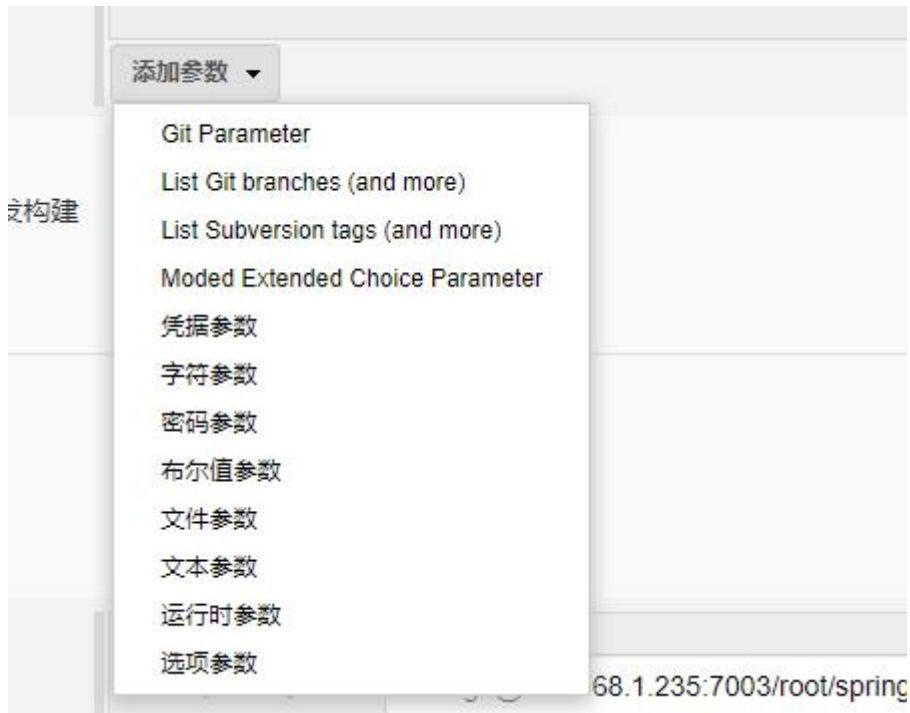


The image shows the 'New Item' dialog box in Jenkins. At the top, there is a text input field labeled '输入一个任务名称' (Enter a task name) with a red border and a '» 必填项' (» Required) label below it. Below this, there are three options, each with an icon and a description:

- 构建一个自由风格的软件项目** (Build a free-style software project): This is Jenkins's main function. Jenkins will combine any SCM and any build system to...
- 构建一个maven项目** (Build a maven project): Build a maven project. Jenkins uses your POM file, which can greatly reduce the...
- 流水线** (Pipeline): Carefully organize a task that can run for a long time on multiple nodes. Suitable for building...

9.4.3 参数化构建过程说明

- 添加参数



➤ 参数说明以测试环境为案例

参数名称	参数值	参数描述
appName	springboot	项目名称
server	机器 ip 地址	服务地址
version	0.0.1	版本号，可以根据情况定义
serverPath	/home/jenkins/workspace/springboot_test/	远程存放代码的路径或者说是远程的文件目录
port	7011	端口
env	test	发布的环境;dev 联调环境，test 测试环境，pre 预发布环境，fix 修复环境，prod 生产环境
targetServerPath	/home/jenkins/workspace/	目标服务器文件路径

9.4.4 源码管理

源码管理

☐ 无
☒ Git

Repositories

Repository URL

Credentials

Branches to build

Branch Specifier (blank for 'any')

源码库浏览器

目标机器生成的密钥，在添加按钮进行设置

以什么方式进行部署，比如：标签，分支

9.4.5 Build 编译设置

Build

Root POM

Goals and options

maven编译工程的命令，忽略测试类进行编译

9.4.6 SSH Publishers 设置

- 其中 SSH Server Name 就是在 <http://jenkins 地址:端口/jenkins/configure> 设置好进行选择
- Transfer Set Source file 传输文件的路径,可以使用参数构建的占位符`${serverPath}`获取
- Remote directory 远程文件目录，同理也参数构建的占位符`${serverPath}`获取
- `docker_server1` 表示与 Jenkins 部署同一个宿主机，使用脚本有远程机器 shell 脚本操作免登陆操作。

SSH Publishers

SSH Server

Name:

Transfers

Transfer Set

Source files:

Remove prefix:

Remote directory:

Exec command:

```
#!/bin/bash

# 打印信息
echo "用户名${userName}"
echo "服务器${server}"
echo "服务器目录${serverPath}"

# 远程创建存放远程上传的代码目录路径
ssh $server mkdir -p ${targetServerPath}

# 远程拷贝代码到目标机器指定路径
scp -r ${serverPath}/ ${userName}@${server}:${targetServerPath}
```

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

高级

- **docker_server1** Shell 脚本 **#!/bin/bash** 表示告诉终端使用 **bash** 解析器进行执行，而且只有第一行 **bash** 才有效。

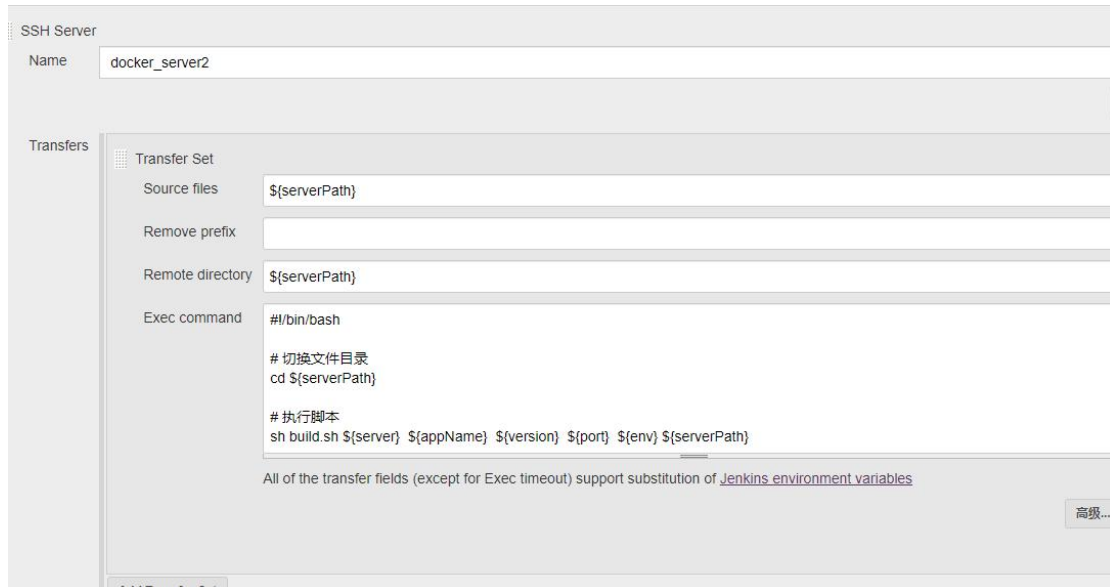
```
#!/bin/bash

# 打印信息
echo "用户名${userName}"
echo "服务器${server}"
echo "服务器目录${serverPath}"

# 远程创建存放远程上传的代码目录路径
ssh $server mkdir -p ${targetServerPath}

# 远程拷贝代码到目标机器指定路径
scp -r ${serverPath}/ ${userName}@${server}:${targetServerPath}
```

- **docker_server2** 表示要部署那台目标机器,所以它的脚本跟 **docker_server1** 不一样.



➤ **docker_server2** Shell 脚本

```
#!/bin/bash
# 切换文件目录
cd ${serverPath}
# 执行脚本
sh build.sh ${server} ${appName} ${version} ${port} ${env} ${serverPath}
```

9.4.7 构建与编译部署项目

➤ 截图的构建参数都是在参数化构建过程配置的参数

测试环境 > springboot_test >

反

良

司

th Parameters

iven project

History 构建历史

2019-7-16 下午12:12

2019-7-15 下午4:08

2019-7-15 下午4:07

2019-7-15 下午3:54

2019-7-15 下午3:49

2019-7-15 下午1:23

2019-7-15 下午1:07

2019-7-15 下午1:05

2019-7-15 下午12:55

2019-7-15 下午12:53

2019-7-15 下午12:50

2019-7-15 下午12:48

Maven project springboot_test

需要如下参数用于构建项目:

appName springboot 项目名称

server 192.168.1.236 服务地址

version 0.0.1 版本号

userName root 用户名

serverPath /home/jenkins/workspace/springboot_test/ 目标服务器文件路径

targetServerPath /home/jenkins/workspace/ 拷贝到远程的目录

port 7011 端口

env dev dev,dev 联调环境, test 测试环境, pre 预发布环境, fix 修复环境, prod 生产环境

开始构建

9.4.8 运行&部署结果

➤ 编译结果

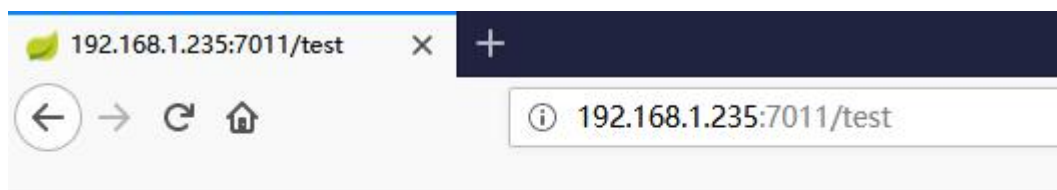

```

Deleted: sha256:ead/4traiff/Uc3blaf130/cadeede/0tra9520a4e812b1b3f1/3b3a32013932
Deleted: sha256:1260e0e1cfb7538a0119a9dce26c9eee4b71da7789065f12e2274c601bbf3340
Deleted: sha256:520e1c09c52b7900089e834f9e2b32d8653807c3918762c1033ab50898cf27bf
Deleted: sha256:4a75ebc4b5e41911971b6880bc6ca459491997f1aa8e341281f9e9ef79fbc3c6
.....进入Building & Images 操作 .....
Sending build context to Docker daemon 21.27MB

Step 1/8 : FROM frovlad/alpine-oraclejdk8:slim
--> 7372599bbfc2
Step 2/8 : MAINTAINER jilongliang@sina.com
--> Running in b00e32776ade
Removing intermediate container b00e32776ade
--> fdc769341a4f
Step 3/8 : RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
--> Running in 01908c47a665
Removing intermediate container 01908c47a665
--> bd19183768e0
Step 4/8 : RUN mkdir -p /home/jenkins/workspace/springboot_test
--> Running in 4d80664a42e8
Removing intermediate container 4d80664a42e8
--> 859e0d12277b
Step 5/8 : WORKDIR /home/jenkins/workspace/springboot_test
--> Running in 5cfd8e4263f2
Removing intermediate container 5cfd8e4263f2
--> df3d6dd62elf
Step 6/8 : EXPOSE 7011
--> Running in cee516507c0e
Removing intermediate container cee516507c0e
--> 78d2500c6195
Step 7/8 : ADD ./target/springboot.jar springboot.jar
--> 08e300f6bf70
Step 8/8 : CMD java ${JAVA_OPTS_DEFAULT} -Djava.security.egd=file:/dev/./urandom -jar springboot.jar
--> Running in 92cf48e64402
Removing intermediate container 92cf48e64402
--> 16b918261f35
Successfully built 16b918261f35
Successfully tagged springboot:0.0.1
.....进入Runing操作 .....
3f20949f1908aab23b1c66b583c6adff2e8613b926613de6469df974e29253ea
.....Build & Run Finish Success~....
SSH: EXEC: completed after 4,611 ms
SSH: Disconnecting configuration [docker_server2] ...
SSH: Transferred 0 file(s)
Finished: SUCCESS

```

➤ 部署&运行结果



this is spring boot 测试环境,date long 1565282560559



10、总结&建议&学习

10.1 总结与建议

- 此文档仅供提供参考学习指引，如需要系统得学习可以根据自身找资料去学习。
- 以上问题都是根据个人实际学习过程中遇到的问题进行一个一个问题进行梳理与总结整理，除了技术问题查很多网上资料通过进行学习之后整理与分享。
- 在学习过程中也遇到很多困难和疑点，如有问题或误点，望各位老司机多多指出或者提出建议。本人会采纳各种好建议和正确方式不断完善现况，人在成长过程中的需要优质的养料。
- 当遇到问题的时候建议多问『谷歌、必应、stackoverflow、度娘』这些大神。
- 计算机是一门『做中学』的学科，不是会了再去做，而是做了才会，多练，常言道熟能生巧。
- 建议学技术『先 Know how，再 Know Why』，意思就说先入门，搞一个 HelloWorld，再深究的意思。
- 建议看官方手册更权威，由于随着技术的发展与迭代,通常官方的文档更新较快，国内的网站资料更新较慢。
- 希望此文档能帮助各位老铁们更好去了解 Docker+Jenkins+GitLab+SpringBoot&SpringCloud+Maven 自动构建镜像与部署服务应用，整个学习流程与搭建会有点小曲折，并不会那么顺利，也希望你看此文档或者通过找资料进行手动安装效果会更好。

10.2 工程源代码&推荐学习&参考文章

- [Docker&SpringBoot 工程源代码](#)
- [VMware Workstation 12 Pro 安装 CentOS7](#)
- [docker-maven-plugin & GitHub](#)
- [Docker-Maven-Plugin&官方网](#)
- [Docker 官方命令手册](#)
- [Docker 从入门到实践](#)
- [Docker 国内学习网站](#)
- [SSH 配置 Linux 下实现免密码登录](#)
- [CentOS7 配置免密码登陆](#)
- [CentOS7 下安装 JDK 详细过程](#)
- [Docker NetWork 基础](#)
- [Dockerfile 指令详解](#)
- [Git 中文官方权威手册在线文档](#)
- [Jenkins+gitlab+maven+docker 自动化部署 pringboot](#)

