

# 学成在线-第20天-讲义-DevOps

## 1 DevOps介绍

### 1.1 什么是DevOps

DevOps是Development和Operations两个词的缩写，引用百度百科的定义：

✚ | ★ 收藏 | 👍 107 | 🔗 149

devops [编辑](#)

本词条由“科普中国”百科科学词条编写与应用工作项目 审核。

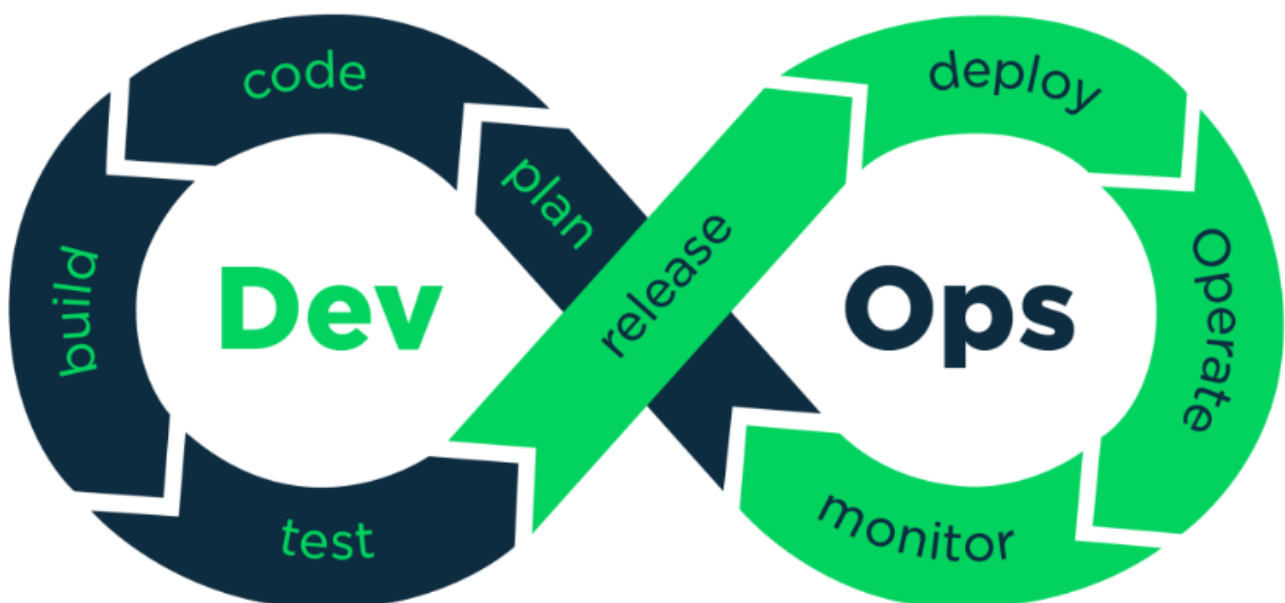
DevOps（Development和Operations的组合词）是一组过程、方法与系统的统称，用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。

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。<sup>[1]</sup>

中文名	过程、方法与系统的统称	全 称	Development和Operations的组合
外文名	DevOps	优 点	尽可能地自动化
		用 途	协作与整合等

DevOps是一种方法或理念，它涵盖开发、测试、运维的整个过程。DevOps是提高软件开发、测试、运维、运营等各部门的沟通与协作质量的方法和过程，DevOps强调软件开发人员与软件测试、软件运维、质量保障（QA）部门之间有效的沟通与协作，强调通过自动化的方法去管理软件变更、软件集成，使软件从构建到测试、发布更加快捷、可靠，最终按时交付软件。



## 1.2 DevOps工具链

DevOps兴起于2009年，近年来由于云计算、互联网的发展，促进了DevOps的基础设施及工具链的发展，涌现了一大批优秀的工具，这些工具包括开发、测试、运维的各各领域，例如：GitHub、Git/SVN、Docker、Jenkins、Hudson、Ant/Maven/Gradle、Selenium、JUnit、JMeter等。下图是DevOps相关的工具集：



## 2 Git/GitLab

### 2.1 Git与GitLab介绍

引用百度百科中对Git和GitLab的描述：

★ 收藏 | 552 | 182

#### GIT (分布式版本控制系统) [编辑](#)

Git(读音为/git/)是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。<sup>[1]</sup> Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Torvalds 开始着手开发 Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。例如 很多 Freedesktop 的项目迁移到了 Git 上。<sup>[1]</sup>

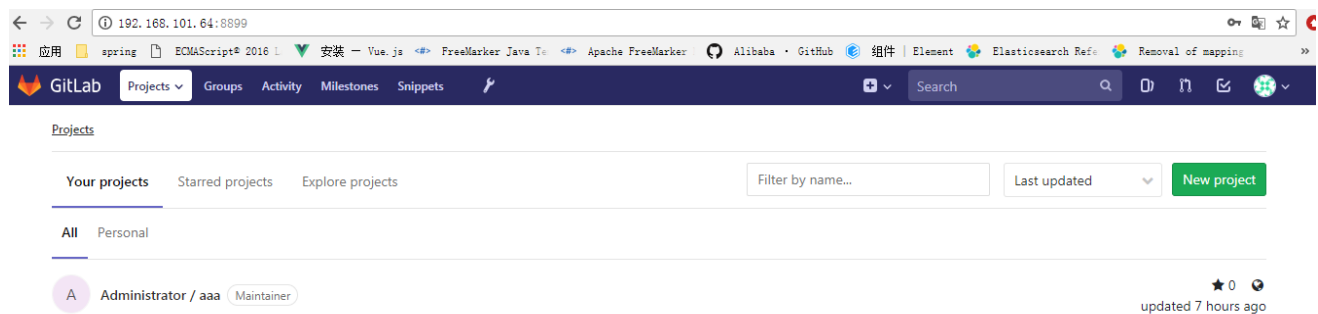
## gitlab [编辑](#)

本词条由“科普中国”百科科学词条编写与应用工作项目 审核。

GitLab 是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务。安装方法是参考GitLab在GitHub上的Wiki页面。

Git是一个版本控制系统，GitLab是一个远程的Git仓库，GitLab与GitHub及国内的码云类似，用户在GitLab上注册账号，上传项目到GitLab，通过GitLab对项目进行版本控制。通常企业使用GitLab在局域网搭建自己的Git代码管理仓库，不过随着云计算的发展，很多企业也开始使用提供Git仓库管理公共服务的平台，比如：码云、GitHub等。

下图是GitLab的界面图：



## 2.2 安装 Git及GitLab

个人电脑上安装Git，过程略。

注意：Git安装完成需配置user.name和user.email

```
查询user.name和user.email
git config user.name
git config user.email
设置：
git config --global user.name "itcast"
git config --global user.email "itcast@itcast.cn"
```

在Linux服务器上安装GitLab，安装方法见“GitLab安装文档”。

注意：个人测试Git版本控制推荐建议直接使用GitHub或码云，两者提供免费创建项目的功能。

## 2.3 在GitLab创建项目

进入GitLab首页：<http://192.168.101.64:8889>，点击“New Project”创建新项目

输入项目信息，提交。

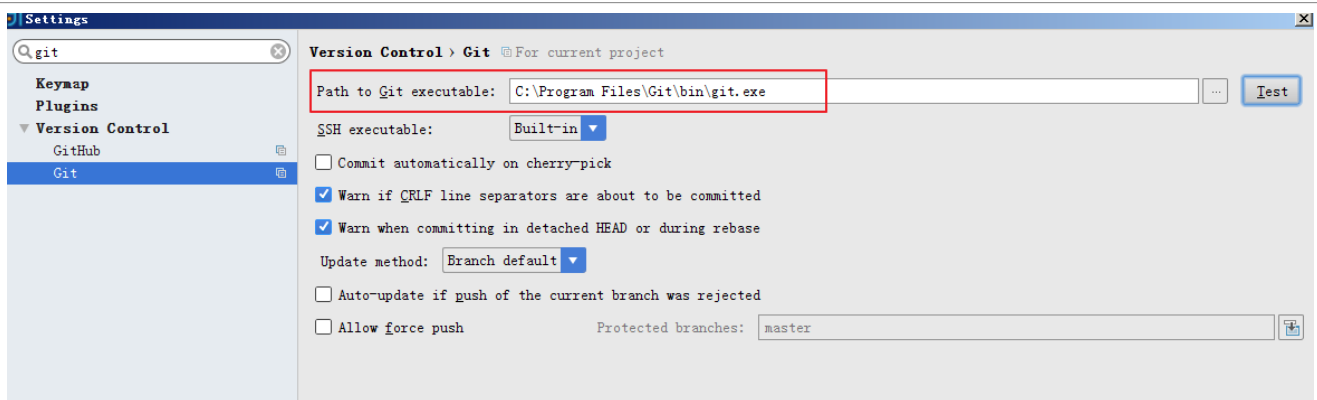
项目新建成功，进入项目主页，通过项目主页即可访问项目内容，并得到项目的Git项目仓库地址：

## 2.4 使用Git管理项目

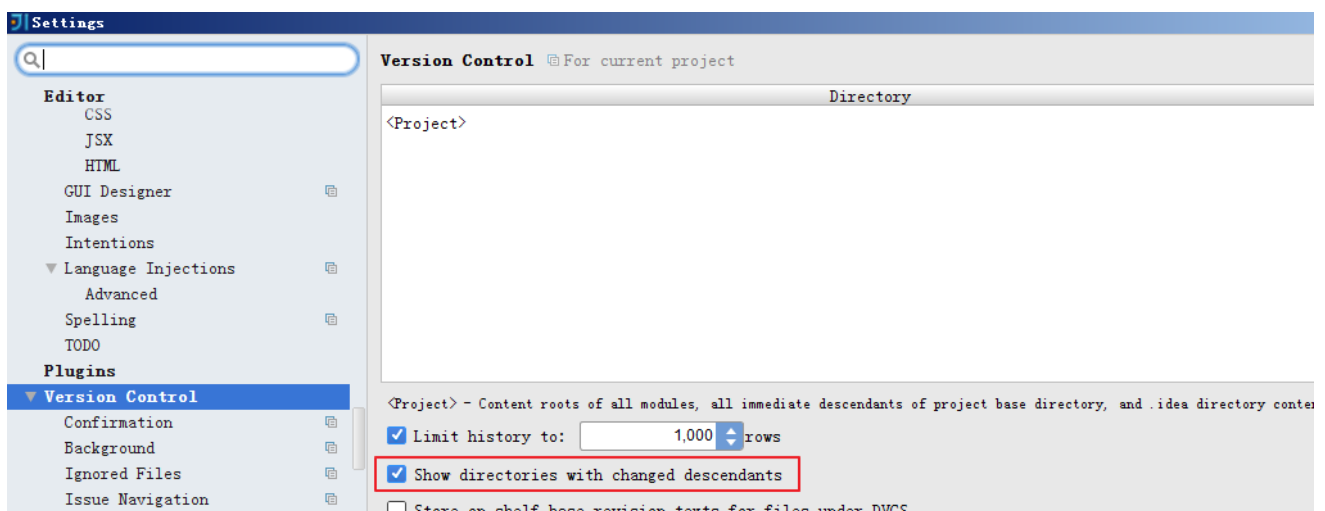
在Idea中集成Git，并用Git管理学成在线项目。

### 2.4.1 设置Git

在开发电脑安装Git，并在Idea中设置Git

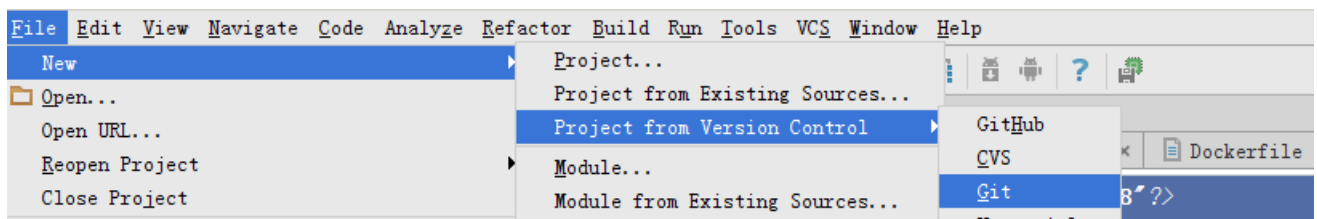


设置 修改代码后父级目录颜色变化：



## 2.4.2 从GitLab检出项目

打开idea，按下图菜单指示从GitLab克隆项目：



输入项目的Git仓库地址进行克隆：



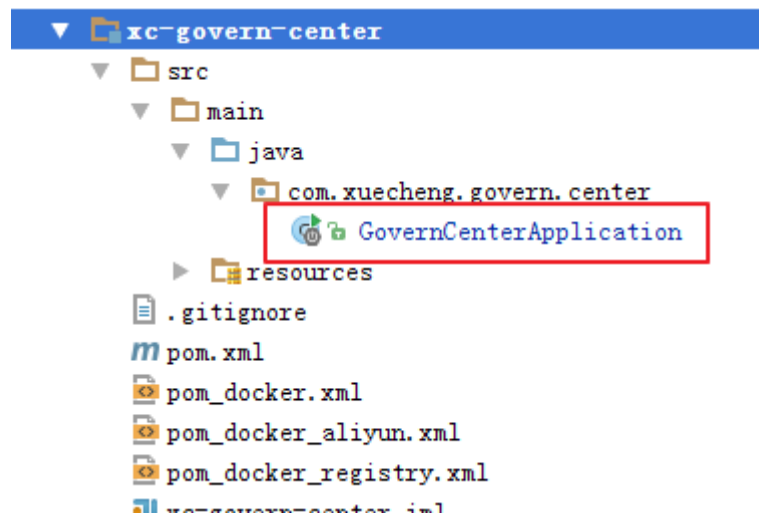
## 2.4.3 提交代码到GitLab

1、在工程根目录创建 .gitignore

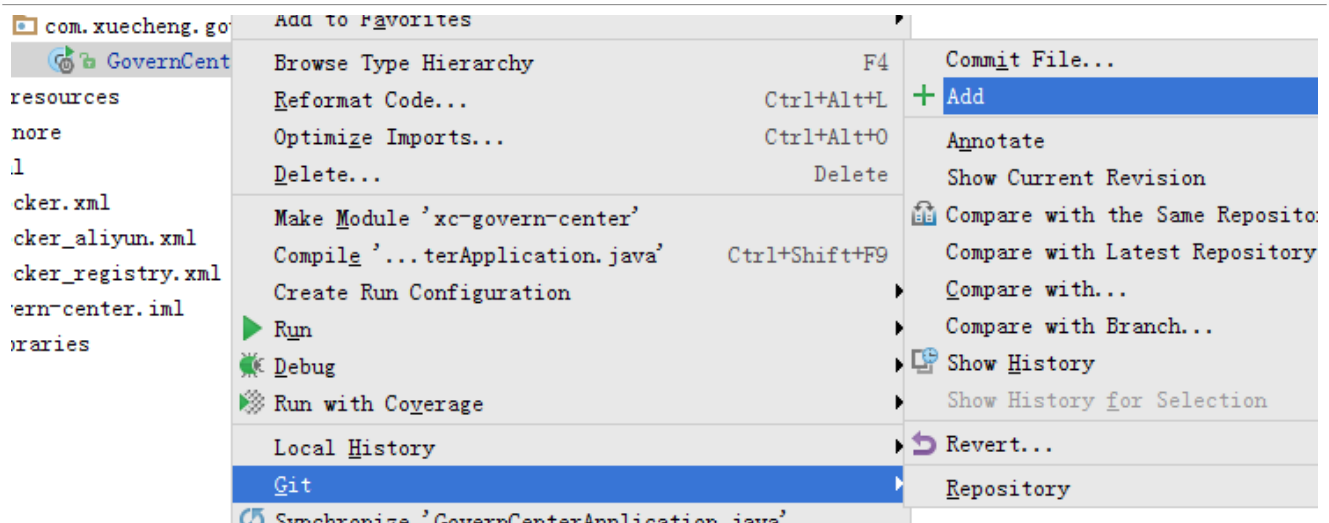
此文件中记录了在提交代码时哪些文件或目录被忽略

```
.idea/  
target/  
*.iml
```

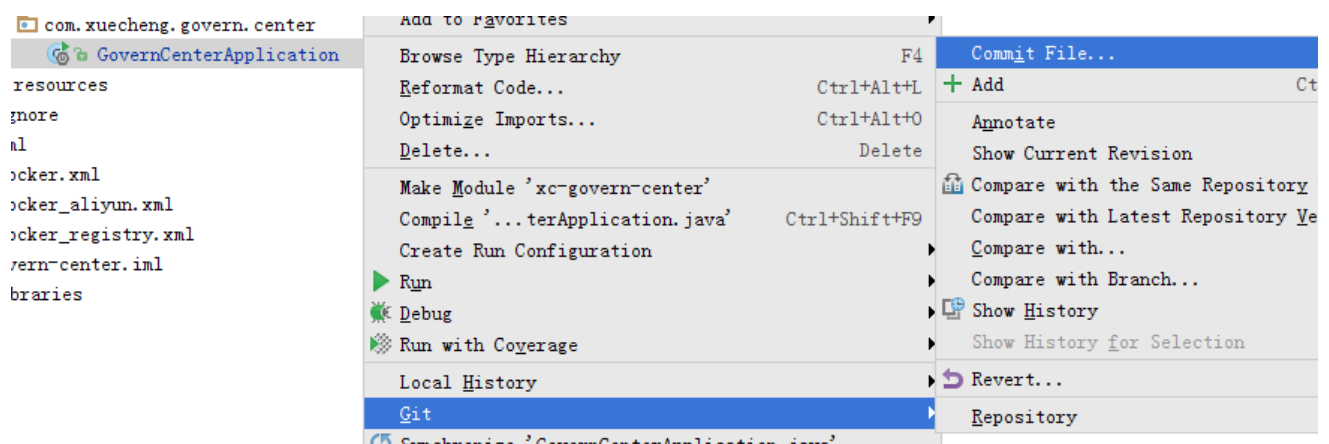
2、代码修改后文件的颜色会出现变化：



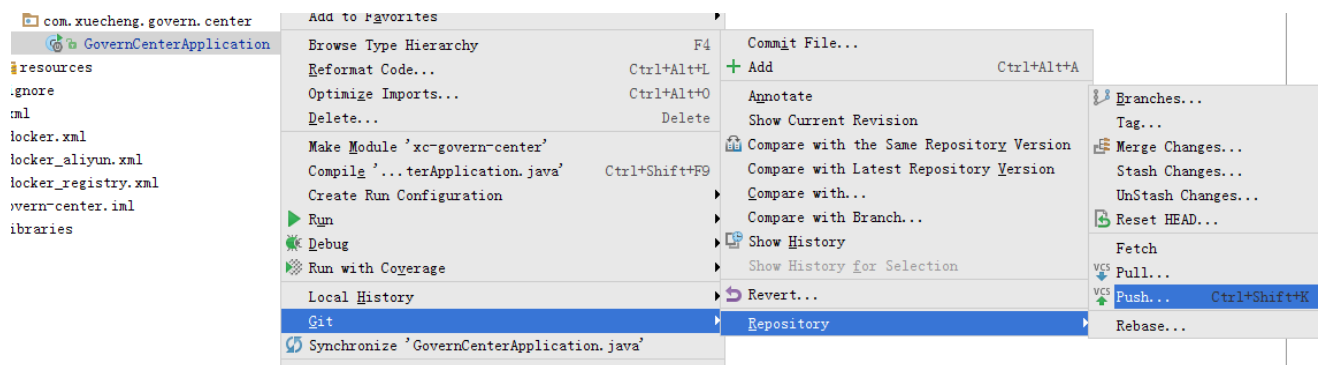
首先执行添加文件到暂存区：



再执行commit 提交文件到本地仓库



最终代码确认无误可以提交到远程仓库



## 3 Docker

### 3.1 虚拟化技术

#### 3.1.1 问题描述



互联网的发展使软件业发生了巨大的变化，其中一个显著的变化是软件的规模越来越大，基于微服务架构的软件在生产部署时遇到了这样的挑战：

- 1、微服务的开发语言、操作系统、依赖库等环境配置不同，如何快速安装、迁移、配置软件？
- 2、一个软件由若干微服务组成，如何快速批量部署微服务？
- 3、如何有效的利于计算机资源？

针对前两个问题的思考：

传统的软件部署流程是：安装操作系统-->安装依赖软件/库-->安装软件（微服务）-->配置软件-->最终软件上线运行，面对大量的微服务及微服务集群，使用此方案不仅效率低下，而且还可能会出现环境兼容问题，显然此方案不适合用在微服务部署。

设想：如果有一项技术可以快速的将软件及所需要的各种环境配置打包、批量复制将会解决以上问题。

针对第三个问题的思考：

在一台计算机只安装一个微服务对计算机资源极大的浪费，如果安装多个微服务就可以有效的利于计算机资源，但是对于批量软件的安装部署还是会面临1、2问题。

设想：在一台计算机安装多个微服务，使用一种技术将微服务打包、复制部署，并且微服务之间隔离互不影响。

### 3.1.2 虚拟化技术

引用百度百科（<https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%8C%96/547949>）

★ 收藏 | 496 | 49

## 虚拟化

 本词条由“科普中国”百科科学词条编写与应用工作项目 审核。

虚拟化，是指通过虚拟化技术将一台计算机虚拟为多台逻辑计算机。在一台计算机上同时运行多个逻辑计算机，每个逻辑计算机可运行不同的操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。

虚拟化使用软件的方法重新定义划分IT资源，可以实现IT资源的动态分配、灵活调度、跨域共享，提高IT资源利用率，使IT资源能够真正成为社会基础设施，服务于各行各业中灵活多变的应用需求。

总结：虚拟化技术是对软件基础设施、操作系统、软件等IT资源进行有效的管理，使用户不再受物理资源的限制，提高计算机资源的利用率。虚拟化技术是云计算的基础，例如阿里云的云主机、腾讯云等都应用了虚拟化技术。

虚拟化技术整体上包括两个方面：硬件虚拟化和软件虚拟化，具体分为：网络虚拟化、存储虚拟化、桌面虚拟化、服务器虚拟化等，我们平常说的最多的是服务器虚拟化。

服务器虚拟化就是在同一个物理服务器上运行多个虚拟机，让服务器的cpu、内存、磁盘、I/O等硬件设施为每个虚拟机服务，在每个虚拟机中运行不同的软件，虚拟机之间是隔离状态。

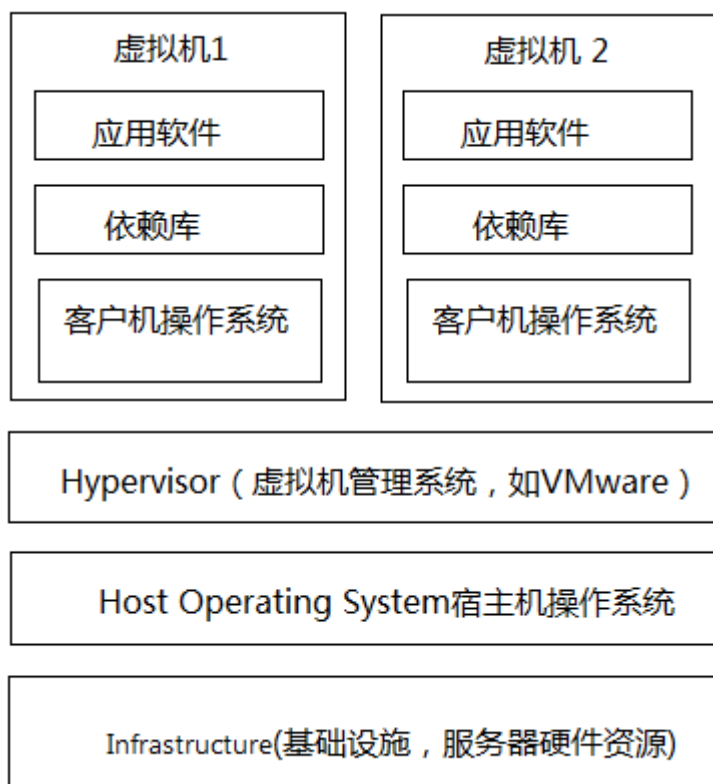
服务器虚拟化主要有两种技术：

- 1、Hypervisor也叫VMM（virtual machine monitor）即虚拟机监视器

Hypervisor是一种将操作系统与硬件抽象分离的方法，实现在宿主机（host machine）上能同时运行多个客户机（guest machine），每个客户机就是一个虚拟机，这些虚拟机高效地分享宿主机的硬件资源。

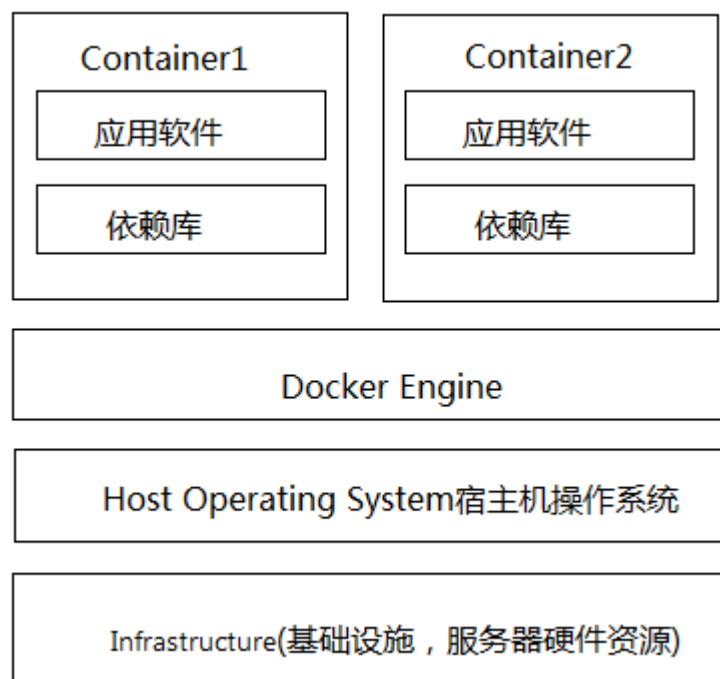
如下图：





在服务器（宿主机）上安装操作系统，并安装hypervisor虚拟机管理软件，如VMware、VirtualBox等，由hypervisor管理多个虚拟机，每个虚拟机上需要安装客户操作系统、依赖库、应用软件。

## 2、Containers容器化技术



容器技术中docker引擎取代了hypervisor，docker引擎是运行在宿主操作系统上的一个进程，该进程管理了多个docker容器，每个docker容器集成了应用软件、依赖库，容器之间相互隔离。

### 3、技术对比：

资源占用：

虚拟机由于是独立的操作系统，占用资源比docker多。

启动速度：

虚拟机包括操作系统，启动虚拟机相当于启动一个操作系统，容器则不一样，**容器中只包括操作系统的内核**，启动一个容器实例相当于启动一个进程，容器的启动速度比虚拟机快。

体积：

容器包括操作系统内核、软件及依赖库，虚拟机不仅包括软件和依赖库还将完整的操作系统打包进去，虚拟机的体积比容器大的多。

## 3.1.3 Docker介绍

引用百度百科 ( <https://baike.baidu.com/item/Docker> )

★ 收藏 | 367 | 403

### Docker

Docker 是一个**开源**的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 **Linux** 机器上，也可以实现**虚拟化**。容器是完全使用**沙箱**机制，相互之间不会有任何接口。 <sup>[1]</sup>

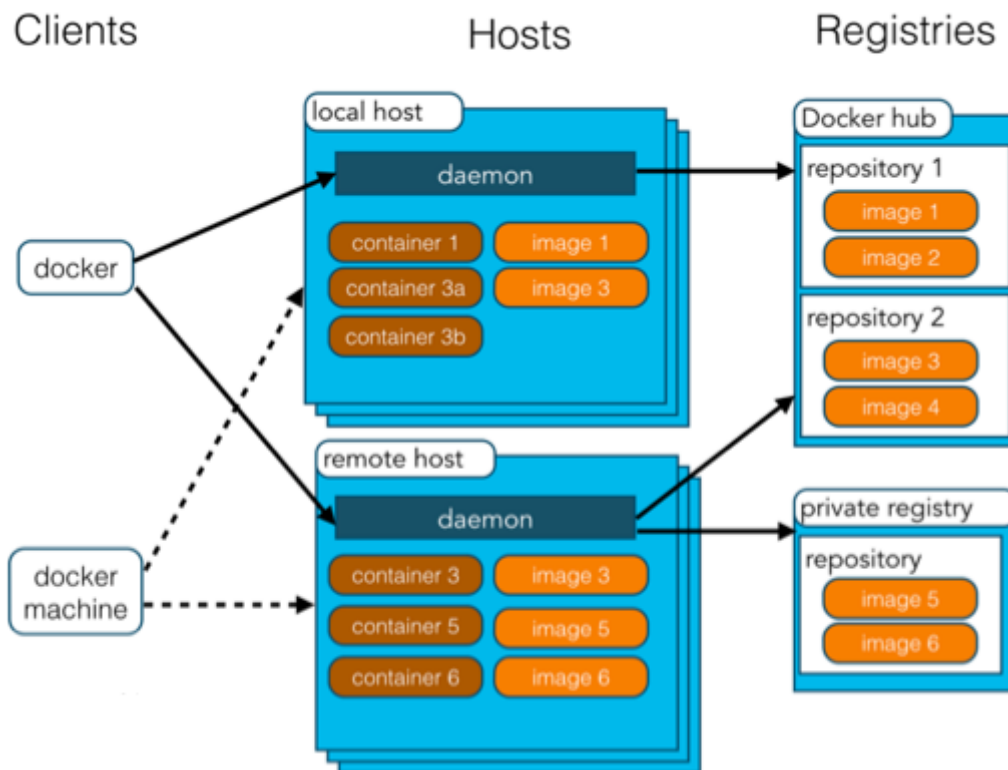
中文名	应用容器引擎	提供商	Docker, Inc.
外文名	Docker	发行日期	2013年
类 别	操作系统层虚拟化	许可协议	Apache License 2.0
		编程语言	Go

Docker 源代码托管在 [Github](#) 上, 基于**go语言**并遵从Apache2.0协议开源。

官网：<https://www.docker.com/>

Docker包括以下部分：

引用百度百科<https://baike.baidu.com/item/Docker>



- 1、Docker daemon ( Docker守护进程 )：Docker守护进程是部署在操作系统上，负责支撑Docker Container的运行以及本地Image的管理。
- 2、Docker client：用户不直接操作Docker daemon，用户通过Docker client访问Docker，Docker client提供pull、run等操作命令。
- 3、Docker Image：Docker 镜像就是一个只读的模板。例如：一个镜像可以包含一个完整的 ubuntu 操作系统环境，里面仅安装了 Tomcat或用户需要的其它应用程序。 镜像可以用来创建 Docker 容器。Docker 提供了一个很简单的机制来创建镜像或者更新现有的镜像，用户甚至可以直接从其他人那里下载一个已经做好的镜像来直接使用。
- 4、Docker Container：Docker 利用容器来运行应用。容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。打个比方，镜像相当于类，容器相当于对象。
- 5、Docker Registry：Docker 仓库分为公开仓库 ( Public ) 和私有仓库 ( Private ) 两种形式 最大的公开仓库是 Docker Hub，存放了数量庞大的镜像供用户下载。用户也可以在本地网络内创建一个私有仓库。当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

## 3.2 部署微服务到Docker

### 3.2.1 安装Docker

Docker可以运行MAC、Windows、Centos、DEBIAN、UBUNTU等操作系统上，提供社区版和企业版，本教程基于Centos安装Docker。Centos6对docker支持的不好，使用docker建议升级到centos7。

- 1、在Centos7上安装Docker

直接通过yum安装即可：

```
yum install -y docker
```

启动docker：service docker start

查询docker版本:docker version

## 2、在Centos6上安装Docker

```
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86\_64/epel-release-6-8.noarch.rpm
```

```
yum install -y docker-io
```

```
service docker start
```

## 3.2.2 部署流程

本项目微服务采用SpringBoot开发，将每个微服务工程打成jar包，最终在Docker容器中运行jar，部署流程如下：

1、SpringBoot工程最终打成 Jar包

2、创建Docker镜像

3、创建容器

4、启动容器

## 3.2.3 打包

1、使用maven的打包插件：

将下边的插件依赖拷贝到微服务工程中，本例子将学成在线的Eureka工程打包：

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

完整的Eureka工程pom.xml文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>xc-framework-parent</artifactId>
    <groupId>com.xuecheng</groupId>
```

```
<version>1.0-SNAPSHOT</version>
<relativePath>../xc-framework-parent/pom.xml</relativePath>
</parent>
<modelVersion>4.0.0</modelVersion>

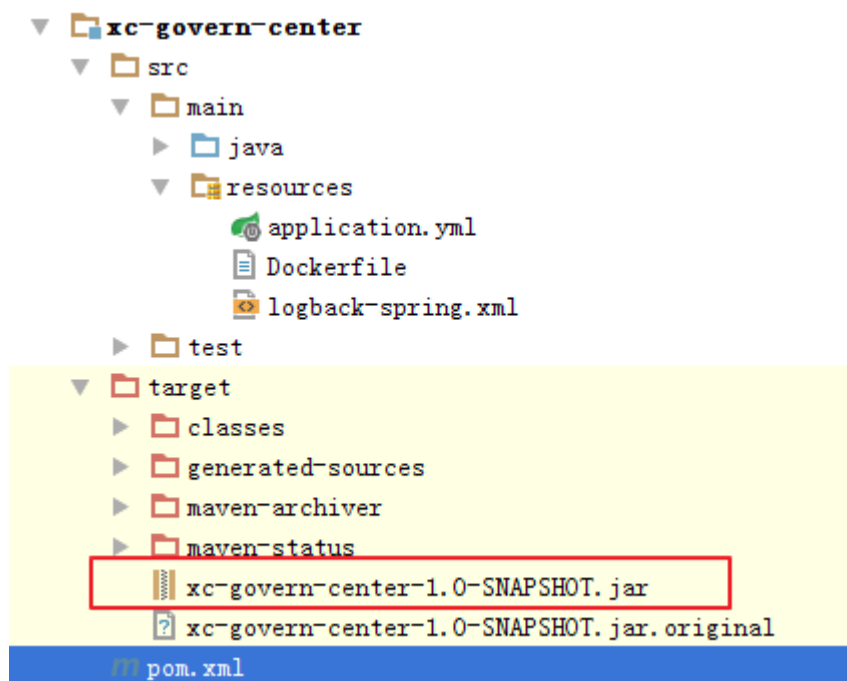
<artifactId>xc-govern-center</artifactId>
<dependencies>
  <!-- 导入Eureka服务的依赖 -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
</dependencies>
<build>
  <finalName>${project.artifactId}-${project.version}</finalName>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>
```

## 2、maven打包

在工程目录运行：mvn clear package

或通过IDEA执行clear package打包命令。

打包成功，如下图：



## 3.2.4 创建镜像

将上一步的jar包拷贝到Linux服务器，准备创建镜像。

测试jar包是否可以运行，执行：java -jar xc-govern-center-1.0-SNAPSHOT.jar

在xc-govern-center-1.0-SNAPSHOT.jar 位置编写Dockerfile文件

```
FROM java:8
ENV ARTIFACTID xc-govern-center
ENV ARTIFACTVERSION 1.0-SNAPSHOT
ENV HOME_PATH /home
WORKDIR $HOME_PATH
ADD /$ARTIFACTID-$ARTIFACTVERSION.jar $HOME_PATH/$ARTIFACTID.jar
ENTRYPOINT ["java", "-jar", "xc-govern-center.jar"] 启动容器直接运行这个命令
```

在Dockerfile文件所在目录执行：docker build -t xc-govern-center:1.0-SNAPSHOT .

镜像创建成功，查询镜像：

```
[root@localhost xc-govern-center]# docker images
```

REPOSITORY	TAG	IMAGE ID
xc-govern-center	1.0-SNAPSHOT	
ad14dcce6666	35 seconds ago	684 MB

## 3.2.5 创建容器

基于xc-govern-center:1.0-SNAPSHOT镜像创建容器，容器名称为xc-govern-center-test

```
docker create --name xc-govern-center-test -t -p 50101:50101 -e PORT=50101 -e
EUREKA_SERVER=http://192.168.72.139:50101/eureka/,http://192.168.72.139:50102/eureka/
xc-govern-center:1.0-SNAPSHOT
```

容器创建成功，可通过docker ps -a命令查看

## 3.2.6 启动容器

docker start xc-govern-center-test

容器启动完成可以通过docker ps 查询正在运行中的容器。

```
[root@localhost xc-govern-center]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
688cabee187	xc-govern-center:1.0-SNAPSHOT	"java -jar xc-gove..."	2 minutes ago
Up 2 minutes	0.0.0.0:50101->50101/tcp		xc-govern-center-
test			

## 3.2.7 停止与删除

要删除的一个镜像重新创建，需要通过如下步骤：



### 1、停止正在运行的容器

docker stop 容器名

例如：docker stop xc-govern-center-test

### 2、删除容器

docker rm 容器名

例如：docker rm xc-govern-center-test

### 3、删除镜像

docker rmi 镜像名或镜像Id

例如：docker rmi xc-govern-center:1.0-SNAPSHOT

## 3.2.8 maven构建镜像 [windows](#)

上边构建的过程是通过手工一步一步完成，maven提供docker-maven-plugin插件可完成从打包到构建镜像、构建容器等过程。

### 1、编写pom\_docker.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>xc-framework-parent</artifactId>
        <groupId>com.xuecheng</groupId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../xc-framework-parent/pom.xml</relativePath>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>xc-govern-center</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- 导入Eureka服务的依赖 -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
    </dependencies>
    <build>
        <finalName>${project.artifactId}-${project.version}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>1.0.0</version>
  <!--docker镜像相关的配置信息-->
  <configuration>
    <!--镜像名，这里用工程名-->
    <imageName>${project.artifactId}-${project.version}</imageName>
    <!--Dockerfile文件所在目录-->
    <dockerDirectory>${project.basedir}/src/main/resources</dockerDirectory>
    <!--TAG,这里用工程版本号-->
    <imageTags>
      <imageTag>${project.version}</imageTag>
    </imageTags>
    <imageName>${project.artifactId}:${project.version}</imageName>
    <!--构建镜像的配置信息-->
    <resources>
      <resource>
        <targetPath>/</targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.artifactId}-${project.version}.jar</include>
      </resource>
    </resources>
  </configuration>
</plugin>
</plugins>
</build>
</project>
```

2、将Dockerfile文件拷贝到src/main/resource下

3、删除之前创建的xc-govern-center镜像

先把工程拷贝到centos7，在centos中使用mvn命令

4、进入工程根目录（pom\_docker.xml所在目录）执行

```
mvn -f pom_docker.xml clean package -DskipTests docker:build
```

指定pom文件，不适用默认

创建镜像成功，结果如下：

```
Successfully built 64682c9ba311
[INFO] Built xc-govern-center:1.0-SNAPSHOT
[INFO] Tagging xc-govern-center:1.0-SNAPSHOT with 1.0-SNAPSHOT
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.409 s
[INFO] Finished at: 2018-07-19T14:21:16+08:00
[INFO] Final Memory: 52M/414M
[INFO] -----
```

## 4 持续集成

### 4.1 持续集成介绍

#### 4.1.1 问题描述

传统的软件开发流程如下：

- 1、项目经理分配模块给开发人员
- 2、每个模块的开发人员并行开发，并进行单元测试
- 3、开发完毕，将代码集成部署到测试服务器，测试人员进行测试。
- 4、测试人员发现bug，提交bug、开发人员修改bug
- 5、bug修改完毕再次集成、测试。

问题描述：

- 1、模块之间依赖关系复杂，在集成时发现大量bug
- 2、测试人员等待测试时间过长
- 3、软件交付无法保障

解决上述问题的思考：

- 1、能否把集成测试时间提前？
- 2、能否使用自动化工具代替人工集成部署的过程？

#### 4.1.2 什么是持续集成

 |  收藏 |  147 |  18

### 持续集成

 编辑

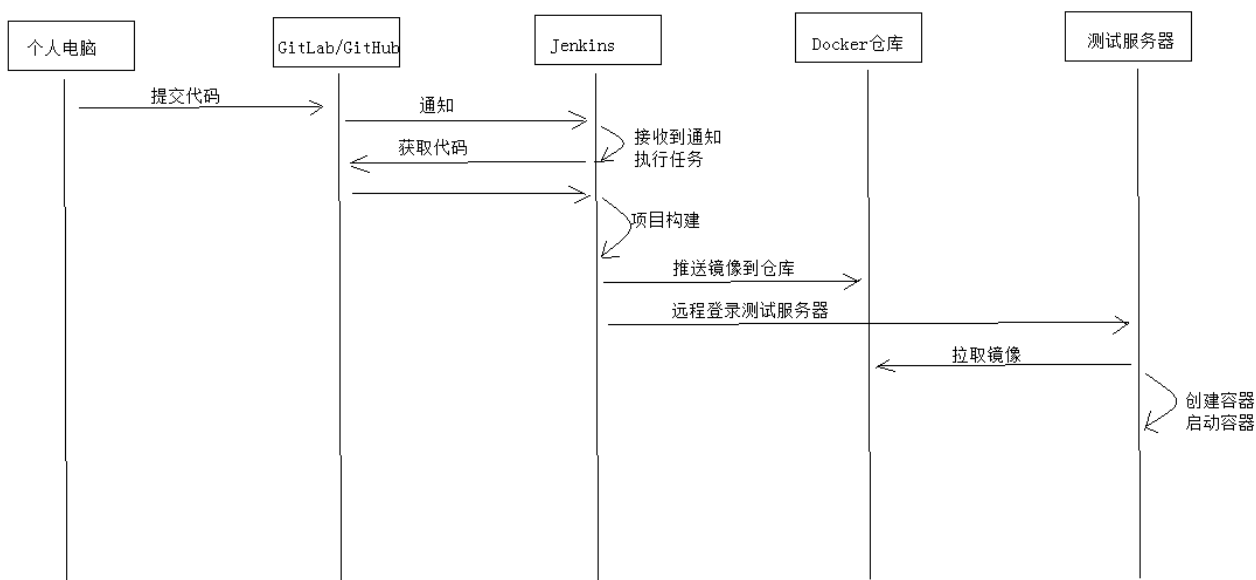
持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通过每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。

持续集成（Continuous integration）简称CI，持续集成的思想是每天要多次将代码合并到主干，并进行集成、测试，这样就可以提早发现错误，进行修正。持久集成也属于DevOps

持续集成的好处：

- 1、自动化集成部署，提高了集成效率。
- 2、更快的修复问题。
- 3、更快的进行交付。
- 4、提高了产品质量。

### 4.1.3 本项目持续集成流程

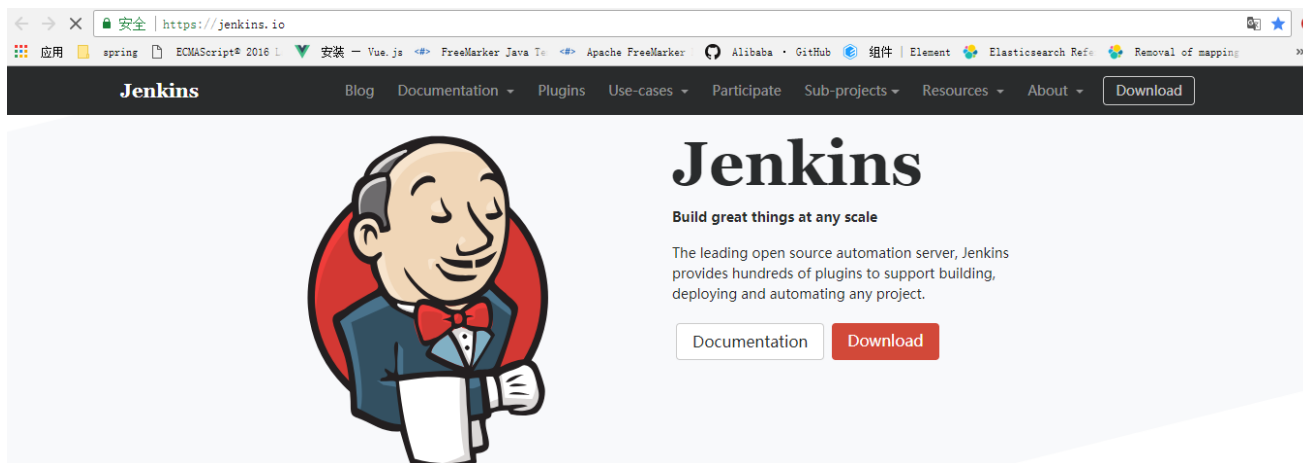


## 4.2 搭建环境

### 4.2.1 安装 Jenkins

Jenkins是一个领先的开源自动化服务器，可用于自动化构建，测试，部署软件等相关任务。

官网地址：<https://jenkins.io>





#### 持续集成和持续交付

作为可扩展的自动化服务器，Jenkins可以用作简单的CI服务器，也可以用作任何项目的持续交付中心。



#### 简易安装

Jenkins是一个独立的基于Java的程序，随时可以运行，包含Windows，Mac OS X和其他类Unix操作系统的软件包。



#### 配置简单

Jenkins可以通过其Web界面轻松设置和配置，其中包括即时错误检查和内置帮助。



#### 插件

Jenkins在更新中心拥有数百个插件，几乎集成了持续集成和持续交付工具链中的所有工具。



#### 扩展

Jenkins可以通过其插件架构进行扩展，为Jenkins提供了几乎无限的可能性。



#### 分散式

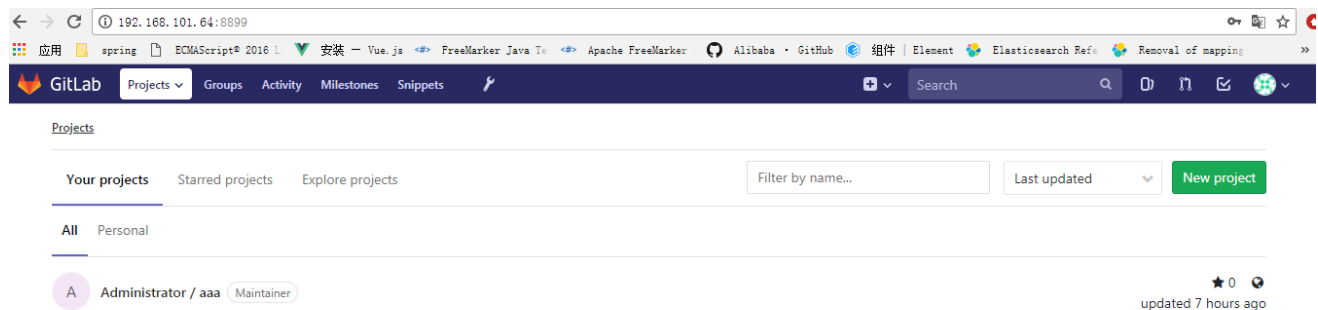
Jenkins可以轻松地跨多台计算机分配工作，帮助更快地跨多个平台推动构建，测试和部署。

安装方法见“Jenkins安装文档”。

## 4.2.2 安装GitLab

GitLab 是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务。

GitLab与GitHub的功能相似，通常企业使用GitLab在局域网搭建自己的Git代码管理仓库。



安装方法见“GitLab安装文档”

## 4.2.3 编写Pom.xml

本例子将xc-govern-center工程使用jenkins进行构建。

在xc-govern-center工程根目录编写pom\_docker\_registry.xml

此文件相比工程原有pom.xml增加了docker-maven-plugin插件，其作用是构建docker镜像并将镜像推送到Docker私有仓库（192.168.101.64:5000）。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>xc-framework-parent</artifactId>
    <groupId>com.xuecheng</groupId>
    <version>1.0-SNAPSHOT</version>
    <relativePath>../xc-framework-parent/pom.xml</relativePath>
```



```
</parent>
<modelVersion>4.0.0</modelVersion>

<artifactId>xc-govern-center</artifactId>
<version>1.0-SNAPSHOT</version>
<dependencies>
    <!-- 导入Eureka服务的依赖 -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
<build>
    <finalName>${project.artifactId}-${project.version}</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.0.0</version>
            <!-- docker镜像相关的配置信息 -->
            <configuration>
                <!-- 镜像名，这里用工程名 -->
                <imageName>${project.artifactId}-${project.version}</imageName>
                <!-- Dockerfile文件所在目录 -->
                <dockerDirectory>${project.basedir}/src/main/resources</dockerDirectory>
                <!-- TAG,这里用工程版本号 -->
                <imageTags>
                    <imageTag>${project.version}</imageTag>
                </imageTags>
                <registryUrl>192.168.101.64:5000</registryUrl>
                <pushImage>true</pushImage>
                <imageName>192.168.101.64:5000/${project.artifactId}:${project.version}
</imageName>

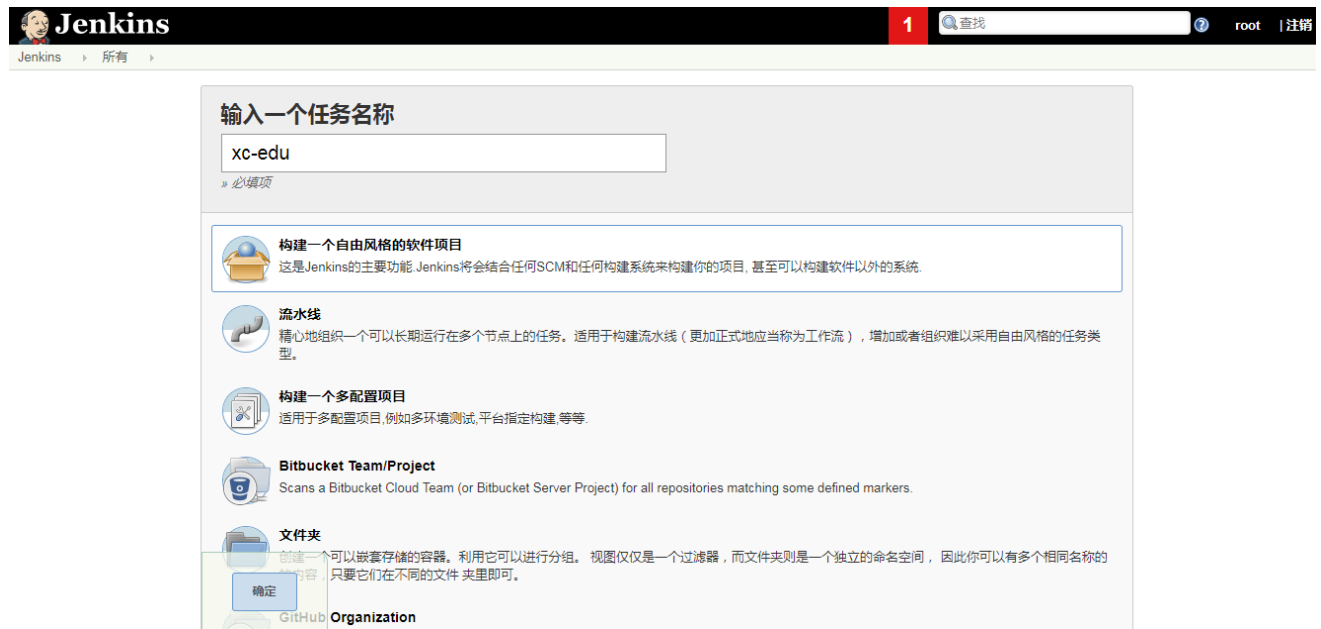
                <!-- 构建镜像的配置信息 -->
                <resources>
                    <resource>
                        <targetPath>/</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.artifactId}-${project.version}.jar</include>
                    </resource>
                </resources>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```



## 4.4 创建持续集成任务

### 4.4.1 创建构建任务

创建学成在线的构建任务：

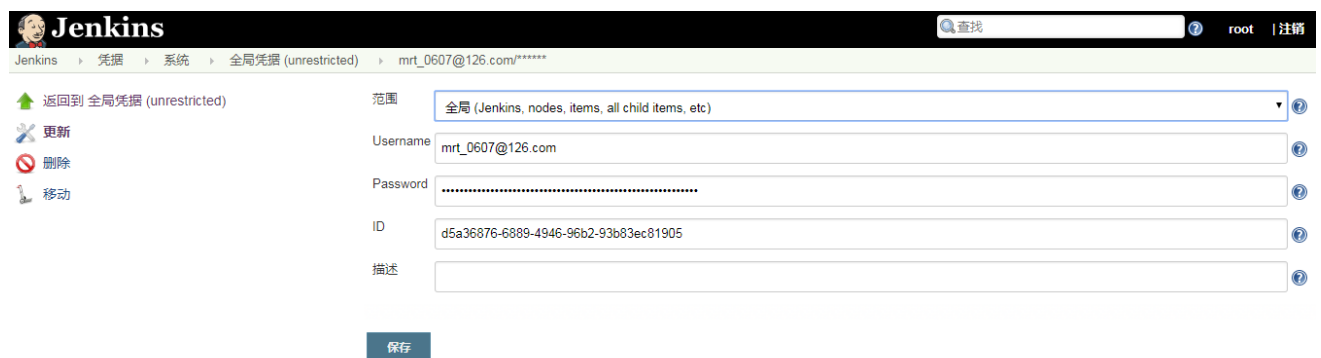


### 4.4.2 配置git仓库

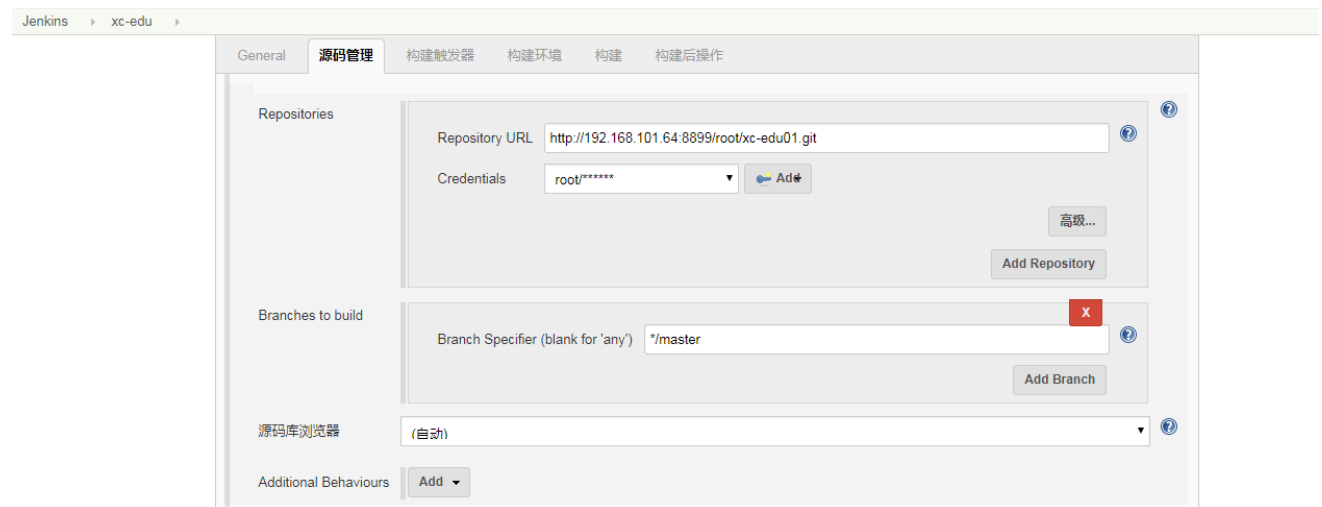
#### 1、配置git凭证

此凭证用于远程从git仓库克隆工程源代码

输入git仓库的账号和密码，这里如果使用码云，下边需要配置码云的账号和密码。



#### 2、配置git仓库地址，此地址即xc-edu项目的地址



### 4.4.3 maven构建配置

目标：

使用jenkins重复构建不要产生重复镜像

使用jenkins停止容器、删除容器、删除镜像之间进行判断

构建过程分为三步：

本例子以构建xc-govern-center工程为例，其它工程构建方式类似。

1、使用shell脚本停止容器、删除容器、删除镜像

远程登录192.168.101.64(测试服务器)

停止xc-govern-center容器

删除xc-govern-center容器

删除192.168.101.64:5000/xc-govern-center:1.0-SNAPSHOT镜像

shell脚本如下：

```
#!/bin/bash
result=$(docker ps | grep "192.168.72.139:5000/xc-govern-center")
if [[ "$result" != "" ]]
then
echo "stop xc-govern-center"
docker stop xc-govern-center
fi
result1=$(docker ps -a | grep "192.168.72.139:5000/xc-govern-center")
if [[ "$result1" != "" ]]
then
echo "rm xc-govern-center"
docker rm xc-govern-center
fi
result2=$(docker images | grep "192.168.72.139:5000/xc-govern-center")
if [[ "$result2" != "" ]]
then
echo "192.168.72.139:5000/xc-govern-center:1.0-SNAPSHOT"
docker rmi 192.168.72.139:5000/xc-govern-center:1.0-SNAPSHOT
fi
```



## 2、执行maven构建：



执行如下maven指令：

```
clean package -f xc-govern-center/pom_docker_registry.xml -DskipTests docker:build
```

这里没有push命令也能push,在pom命令中写

## 3、拉取镜像，创建容器，启动容器

从docker私有仓库拉取镜像并创建容器，启动容器

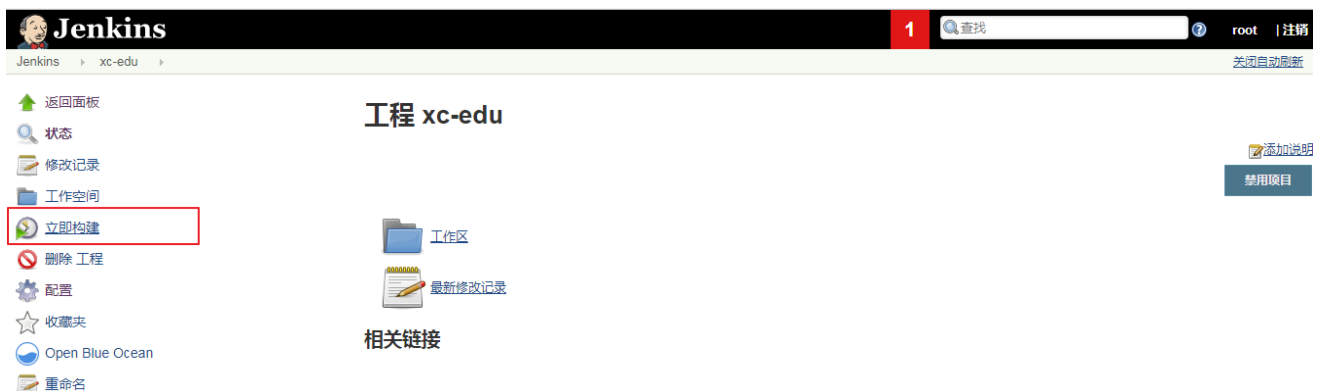
显示容器运行日志



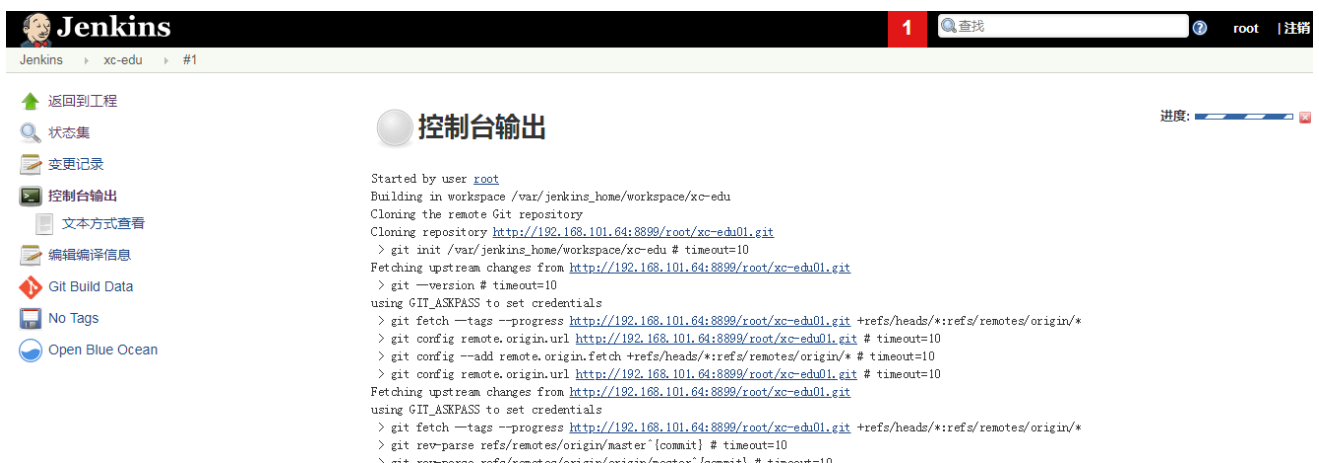
```
docker run --name xc-govern-center -p 50101:50101 -idt 192.168.72.139:5000/xc-govern-center:1.0-SNAPSHOT docker logs -f xc-govern-center
```

## 4.4.4 执行任务

1、进入任务页面，点击“立即构建”



2、开始构建，查看日志



## 4.4.5 自动构建

使用Gitlab提供的webhook功能可以实现开发人员将代码push到Git仓库自动进行构建。

#### 1、设置webhook

参考“GitLab安装文档”配置webhook

#### 2、在Idea中修改项目代码，push到GitLab。

#### 3、Jenkins任务收到GitLab通知自动执行构建。

jenkins会等待队列，所以手动停止进程

# GitLab安装文档

GitLab 是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务。

GitLab与GitHub的功能相似，通常企业使用GitLab在局域网搭建自己的Git代码管理仓库。

## 1 Docker下安装Gitlab

拉取gitlab、redis、postgresql，gitlab依赖redis和postgresql。

```
docker pull sameersbn/redis
docker pull sameersbn/postgresql
docker pull gitlab/gitlab-ce:latest
```

创建postgresql、redis容器：

```
docker run -d --name postgresql --privileged=true -e 'DB_NAME=gitlabhq_production' -e
'DB_USER=gitlab' -e 'DB_PASS=password' -e 'DB_EXTENSION=pg_trgm' -v
/home/root/opt/postgresql/data:/var/lib/postgresql sameersbn/postgresql

docker run --name redis -d --privileged=true -v /home/root/opt/redis/data:/var/lib/redis
sameersbn/redis
```

这里端口做了映射，不是22端口  
git客户端连接ssh端口不是22的gitlab

创建gitlab容器：

```
docker run --name gitlab -d --link postgresql:postgresql --link redis:redisio --hostname
192.168.101.64 -p 10022:22 -p 8899:80 -e 'GITLAB_PORT=8899' -e 'GITLAB_SSH_PORT=10022' -e
'GITLAB_SECRETS_DB_KEY_BASE=long-and-random-alpha-numeric-string' -e
'GITLAB_SECRETS_SECRET_KEY_BASE=long-and-random-alpha-numeric-string' -e
'GITLAB_SECRETS_OTP_KEY_BASE=long-and-random-alpha-numeric-string' -e
'GITLAB_HOST=192.168.101.64' -e 'SMTP_AUTHENTICATION=login' -v
/home/root/opt/gitlab/data:/home/git/data docker.io/gitlab/gitlab-ce
```

启动gitlab需要同时启动三个容器，并且gitlab启动时间很长，内存消耗巨大

浏览器访问：<http://192.168.72.139:8899>

初次访问需要等待一段时间。

GitLab镜像克隆修改项目地址，先进入容器

```
#/var/opt/gitlab/gitlab-rails/etc/gitlab.yml

## GitLab settings
gitlab:
  ## Web server settings (note: host is the FQDN, do not include http://)
  host: 这个地方填gitlab所在服务器的ip即可
  port: 80
  https: false
```

## 2 添加公钥 账号密码也可以

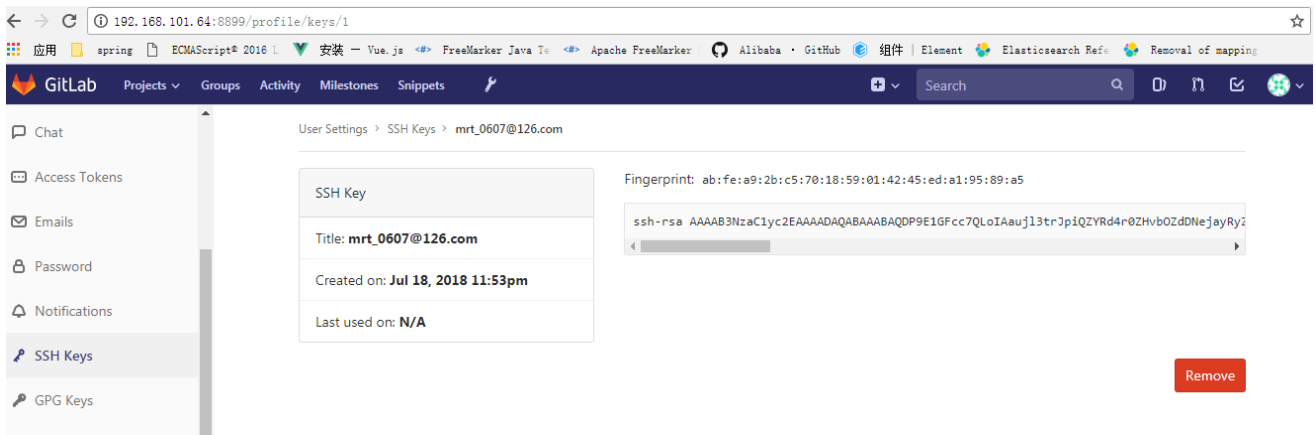
Jenkins远程访问GitLab获取源代码，这里需要在GitLab配置公钥。



## 1、进入jenkins容器，生成公钥：

```
[root@localhost conf]# docker exec -it jenkins /bin/bash
Error response from daemon: Container
10f316cedd5be97b499db231054a8505ed3b68c36d72cf0e26bbe253cb8a9615 is not running
[root@localhost conf]# docker start jenkins
jenkins
[root@localhost conf]# docker exec -it jenkins /bin/bash
bash-4.4# ssh-keygen -t rsa -C "mrt_0607@126.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:6c5k0DkGUhikicIZzxrOU5+ZvZeKwnvsUkBEuxAW69c mrt_0607@126.com
The key's randomart image is:
+---[RSA 2048]-----+
| oo++o          |
| oo*oo..        |
| =+.*o.         |
| + =.o.B   .    |
| =...B.ES       |
| .... =. .      |
|   .O.+o        |
|   .ooXo        |
|   ..O+oo       |
+----[SHA256]-----+
bash-4.4# cat ~/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDP9E1GFcc7QLoIAauj13trJpiQZYRd4r0ZHvb0ZdNeJayRyZodDwu8W1emHaIdn3P
jqzNjkatkIayPL5/ccWmJbZWLNSGusJLaJW+vRxUBSNsG5QBrw7rogJqxhI878DCch7JpFFBe4pVNCQPlW1Ym+jquTn3vVcO
AJZw3UOdRnijYQ0d+AotL5Gmj/7lhHi6L1Dssr8LbyuJFLmZPXzCMpFL68JjcppqJbunWEp06s03GEyYndk1NDuqX0v8XJ0
AXLTfbz3mT0u4irDKyI91mc6loDdYFpAYUSsstwXzVIRcjkZyRQNT0oAXMgROSox+kR0KswoU5aJKFD0ipbx
mrt_0607@126.com
bash-4.4#
```

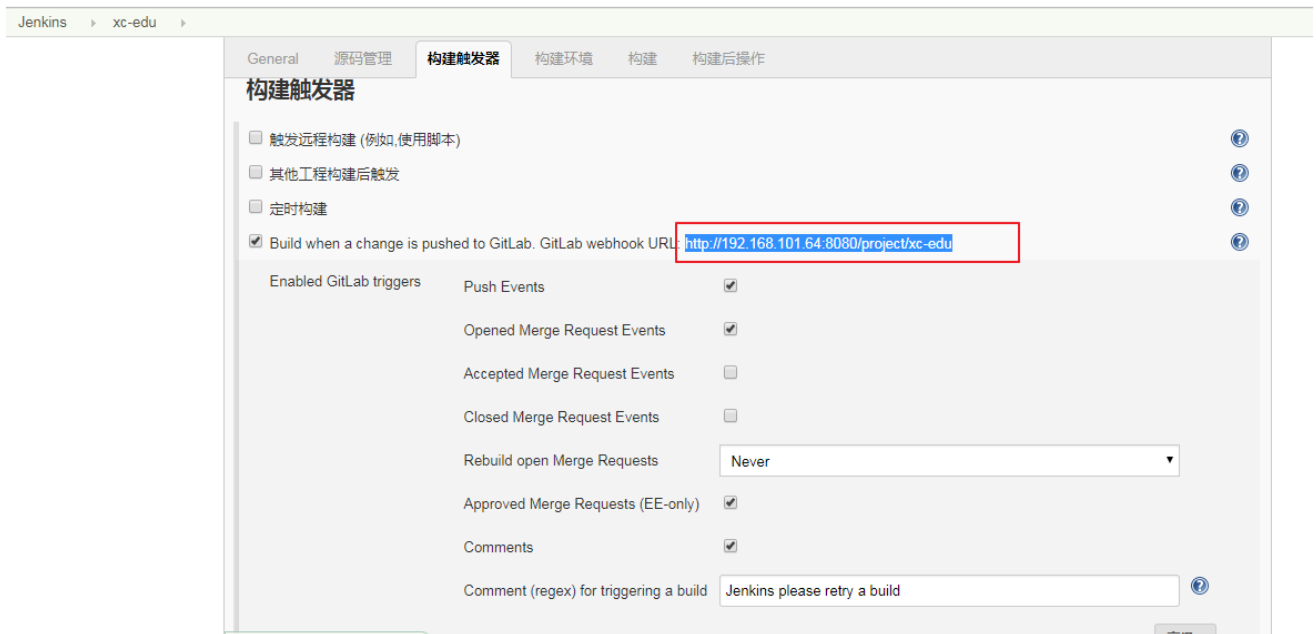
## 2、进入GitLab配置公钥：



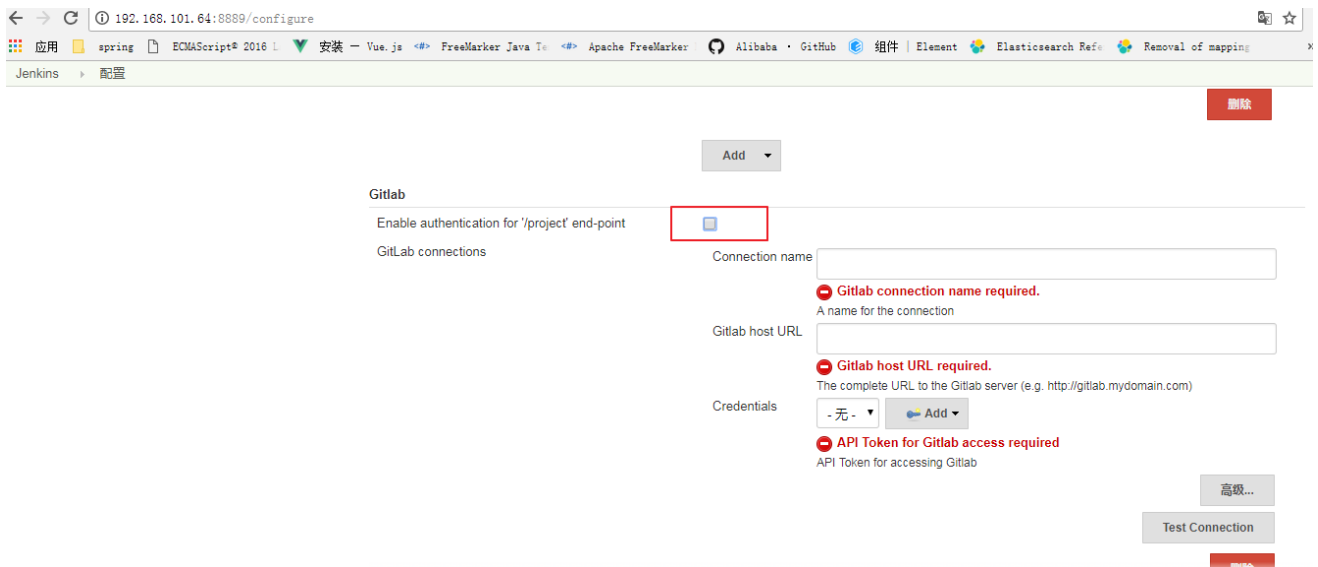
## 3 配置webhook

GitLab中使用webhook向jenkins通知，当有代码push后将通知jenkins进行构建。

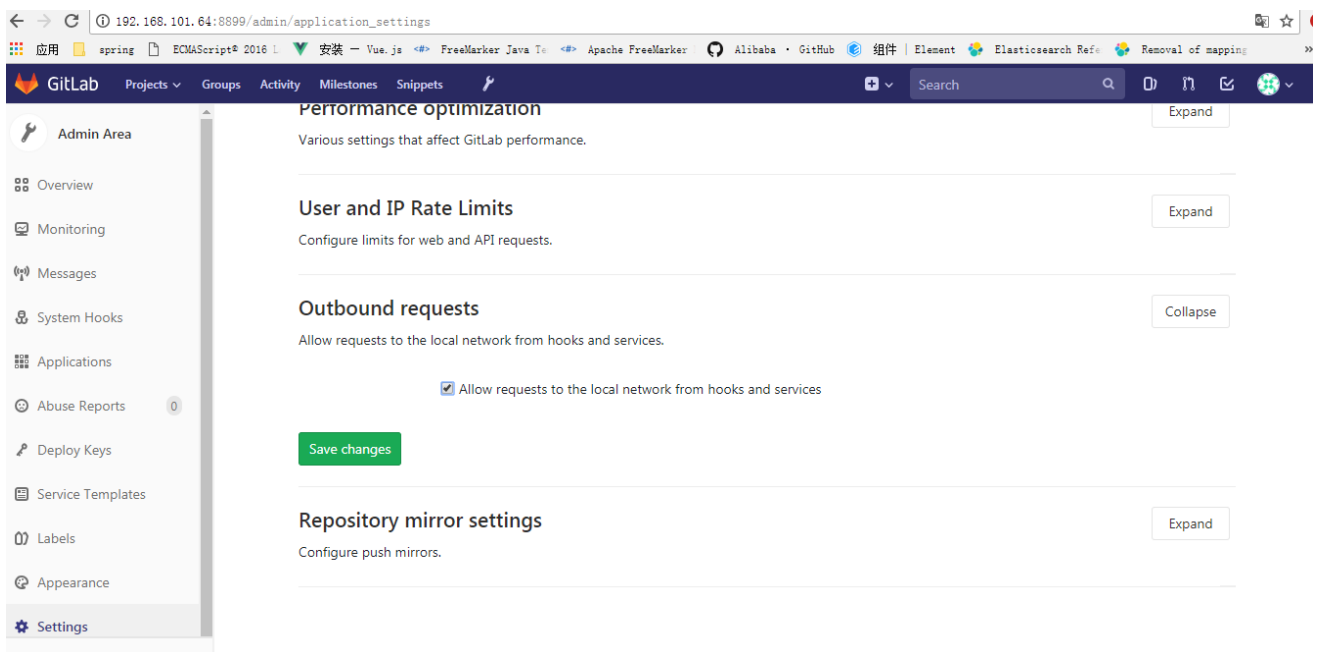
### 1、在Jenkins中找到通知地址



### 2、进入Jenkins设置允许匿名访问jenkins，这样GitLab才要成功通知jenkins



3、使用管理员帐号[admin@example.com](mailto:admin@example.com)登录Gitlab，密码就是你gitlab搭建好之后第一次输入的密码,设置允许请求本地网络服务



4 设置钩子地址，即jenkins中的项目地址

xc-edu01

Project

Repository

Issues0

Merge Requests0

CI / CD

Operations

Wiki

Snippets

Settings

<< Collapse sidebar

Administrator > xc-edu01 > Integrations Settings

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

http://192.168.101.64:8080/project/xc-edu

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ Push events

This URL will be triggered by a push to the repository

☐ Tag push events

This URL will be triggered when a new tag is pushed to the repository

☐ Comments

This URL will be triggered when someone adds a comment

☐ Confidential Comments

This URL will be triggered when someone adds a comment on a confidential issue

钩子添加成功

xc-edu01

Project

Repository

Issues0

Merge Requests0

CI / CD

Operations

Wiki

Snippets

Settings

<< Collapse sidebar

Add webhook

Webhooks (1)

http://192.168.101.64:8080/project/xc-edu

Push Events

SSL Verification: enabled

Edit

Test

Service	Description	Last edit
---------	-------------	-----------

# Jenkins安装文档

## 1 Docker下安装Jenkins

本教程在docker下安装Jenkins：

拉取镜像：

`docker pull jenkinsci/blueocean`

创建容器：

```
docker create --name jenkins -u root -p 8889:8080 --privileged=true
-v jenkins-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock
-v /home/jenkins:/home docker.io/jenkinsci/blueocean
```

启动容器：

`docker start jenkins`

输入<http://192.168.72.139:8889>

初次运行Jenkins会经过一个安装过程，一般情况使用默认配置，下一步安装即可，其中一步需要输入密码，如下图：



出现上边的画面需要进入Jenkins容器查看密码，并将密码拷贝到上图输入框中：

```
docker exec -it jenkins /bin/bash
cat /var/jenkins_home/secrets/initialadminPassword
```

密码: 36e057695b614f2a9b0affc6071cf7eb

## 2 Jenkins全局工具配置 dcoker容器的方式需要把全局工具也安装在docker中 不适用docker比较好

本项目使用Jenkins需要配置Jdk1.8、Git、maven。

### 1、Maven安装：



可以采用“自动安装”，在上图中勾选自动安装，选择maven版本即可。

本教程将maven程序目录拷贝到宿主机/home下，在上图配置即可。  
宿主机找得到吗

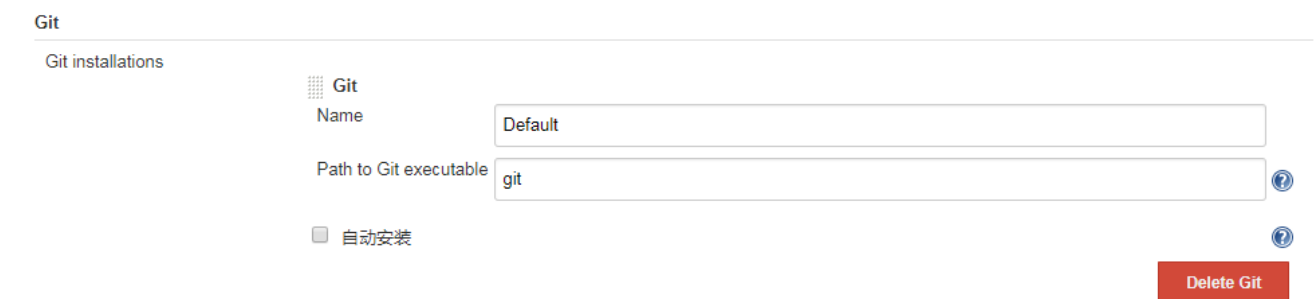
### 2、Jdk安装配置



与maven安装相同，也可以采用自动安装或手动安装的方法。

### 3、Git安装方法同上

git安装完成配置如下图：





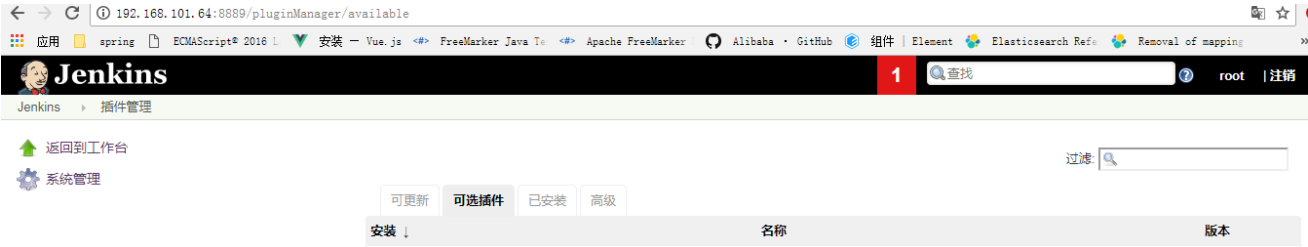
# 3 SSH Remote Hosts

现在使用的是容器，因此要远程操作Linux

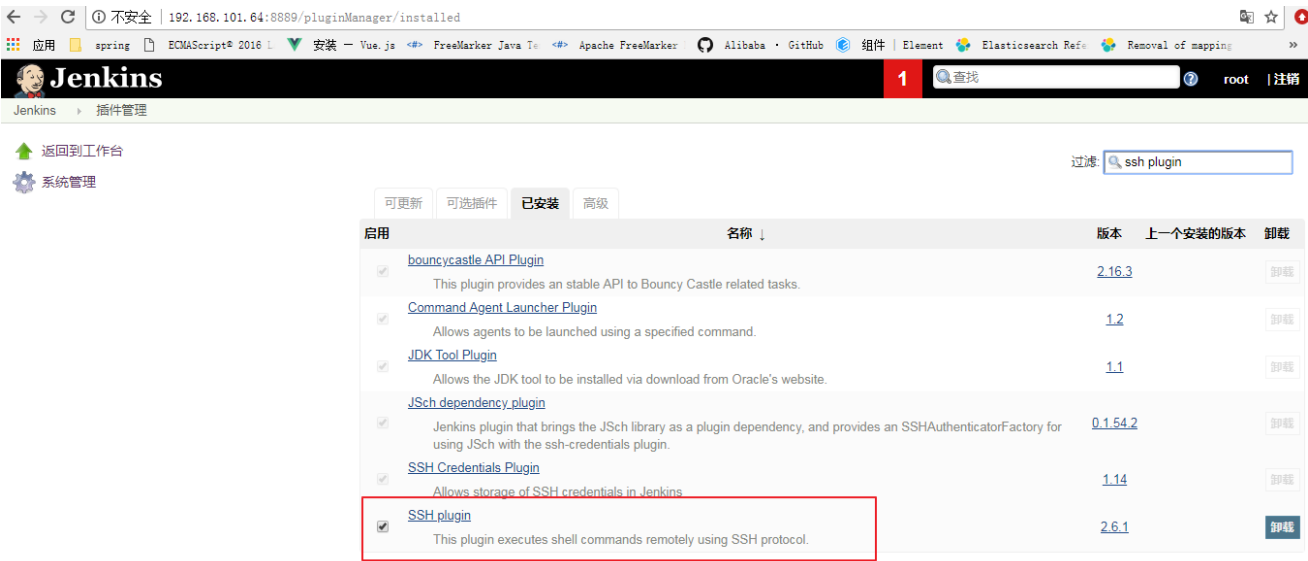
SSH remote hosts是Jenkins提供的一种远程访问ssh服务器的方法，通过如下步骤测试此功能：

## 1、安装插件SSH plugin

从可选插件中选择SSH Plugin进行安装



安装成功可在已安装插件中查询：



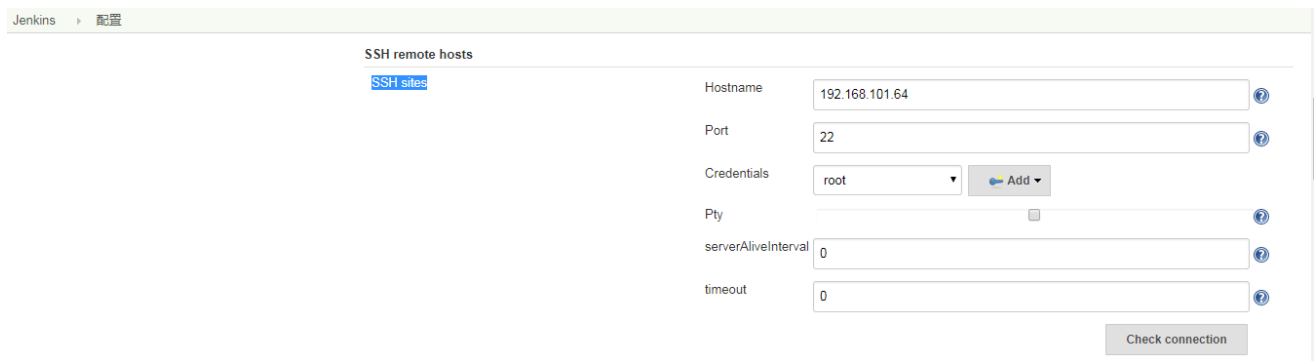
## 2、配置凭证

配置远程SSH服务的账号和密码：Linux的地址



## 3、配置SSH sites

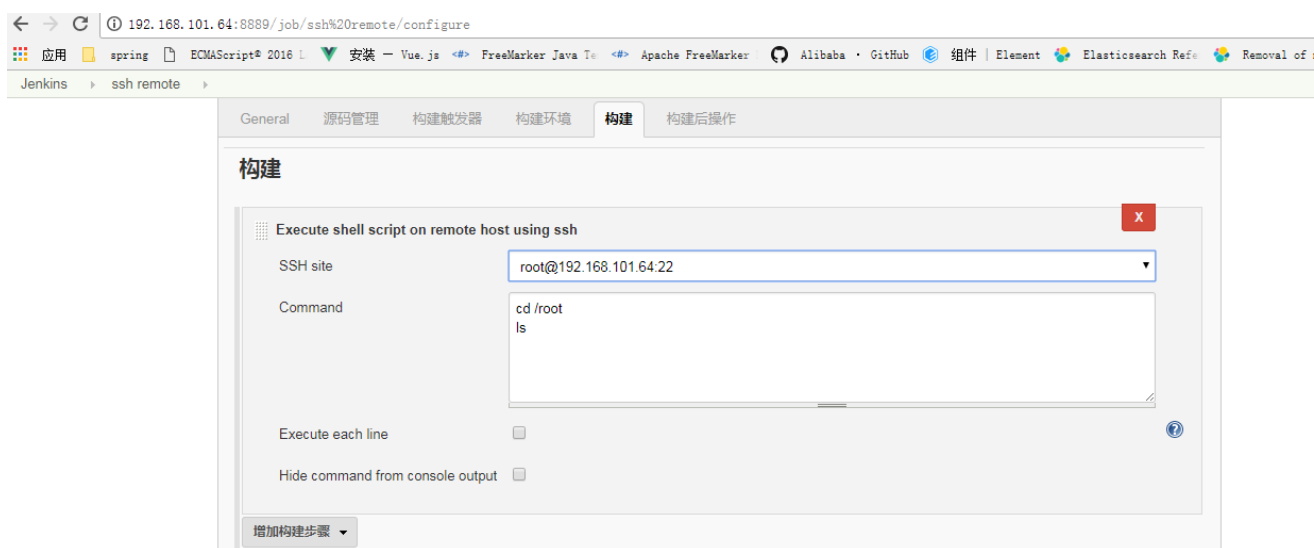
配置远程ssh服务器的ip和端口，并选择第2步设置凭证



The image shows the 'SSH remote hosts' configuration page in Jenkins. It includes fields for Hostname (192.168.101.64), Port (22), Credentials (root), Pty (unchecked), serverAliveInterval (0), and timeout (0). A 'Check connection' button is at the bottom right.

#### 4、在任务构建中编写脚本

选择第3步定义的SSH site，并在command输入脚本。



The image shows the 'Build' configuration page in Jenkins. It includes a dropdown for 'SSH site' (root@192.168.101.64:22) and a text area for 'Command' (cd /root, ls). There are checkboxes for 'Execute each line' and 'Hide command from console output'. A '增加构建步骤' button is at the bottom left.

配置完成可以测试是否可以远程执行SSH指令。

## 4 搭建Docker私有仓库

微服务的镜像会上传到Docker仓库保存，常用的公网Docker仓库有阿里云，网易云等，在企业局域网也可以搭建自己的Docker私有仓库，本教程使用Docker提供的私有仓库 registry。

进入Docker私有仓库所在服务器执行：

- 1、安装Docker
- 2、创建私有仓库容器

```
docker run --name docker-registry -d -p 5000:5000 registry
```

执行结果如下：

```
[root@localhost xc-govern-center]# docker run --name docker-registry -d -p 5000:5000 registry
Unable to find image 'registry:latest' locally
Trying to pull repository docker.io/library/registry ...
latest: Pulling from docker.io/library/registry
4064ffdc82fe: Pull complete
c12c92d1c5a2: Pull complete
4fbc9b6835cc: Pull complete
765973b0f65f: Pull complete
3968771a7c3a: Pull complete
Digest: sha256:51bb55f23ef7e25ac9b8313b139a8dd45baa832943c8ad8f7da2ddad6355b3c8
Status: Downloaded newer image for docker.io/registry:latest
aaf774e6dcc76173dceb4ff16831a129238e7dbcdd770d48c25e214b972c9553
[root@localhost xc-govern-center]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
aaf774e6dcc7	registry	"/entrypoint.sh /e..."	20 seconds ago
Up 20 seconds	0.0.0.0:5000->5000/tcp	docker-registry	

上边显示 registry 镜像已经创建，并且 docker-registry 容器已经启动成功。

访问：[http://192.168.101.64:5000/v2/\\_catalog](http://192.168.101.64:5000/v2/_catalog)

响应结果如下：

```
{"repositories":[]}
```

上边的响应结果说明在 docker 私有仓库中还没有镜像。

默认 docker-registry 只允许 https 提交镜像，如下配置使 docker-registry 支持 http

在 /etc/docker 下，创建 daemon.json 文件，写入：

```
{ "insecure-registries":["192.168.101.64:5000"]}
```

重启 docker：

```
systemctl restart docker.service
```

启动 docker-registry

```
docker start docker-registry
```

