

电商自动化部署实现与优化及日志收集体系简介

-- 楼兰

大家拿到代码后，要如何运行呢？导入IDEA，然后启动？开发过程肯定没有问题，那生产环境呢？在互联网大环境下，越来越要求开发运维一体化。如果对于企业级的项目管理方式不了解，那么开发工作将举步维艰。这一节课主要带大家快速理解一下电商项目的运维部署方式。电商项目经过综合考虑，即迎合互联网大厂的DevOps一体化运维趋势，又尽量降低运维的门槛，实现了一套简化版的自动化部署体系。其目的，是为了让大部分同学都能够理解互联网大厂的DevOps是怎么回事。当然，有基础的同学也可以根据电商的部署方案拓展出虚拟化，云原生等更贴近现代化的部署方式。并且可以基于自己的开发经验，开始思考如何落地DevOps，来提高项目部署的效率。

今天课程的关注点主要是以下几个问题：

- 熟悉SpringBoot常见的Maven打包方式
- 基于Jenkins和GitLab部署基础自动化运维体系
- CI\CD的优化方案
- 基于FileBeat+Logstash+ES实现的典型分布式日志收集体系

一、大厂都在做的DevOps，CI\CD都是什么？

谈到DevOps，只要你有实际项目的开发经验，那么对这个词一定不会陌生。各大互联网厂商都在不断推出自己的DevOps实践落地的理论、规则、产品。我们这个电商项目也不例外。代码最终需要经过DevOps的一系列操作才能部署到服务器上真实运行。那到底什么是DevOps呢？

1、开发与运维割裂的问题

当开发人员基于本地开发环境完成了代码开发后，最终是需要部署到生产的服务器上执行的。在传统的运维体系下，开发和运维通常都是割裂的。很多大型项目中，开发人员不允许接触生产环境的服务器，服务器只能由运维团队进行操作。这样可以极大的提高服务器的安全性。尤其对于像我们的电商这样面向互联网的项目，不经保护的开放服务，就是给黑客提供攻击的靶子。

因此，在现代化的大型软件项目中，对于开发人员的要求也更为全面。虽然开发人员不要求像专业的运维人员一样，掌握服务器的各种安全策略、参数调优等。但是对于基础的运行环境运维操作也必须要了解，这样才能指导运维人员进行业务环境部署，也就是开发运维一体化。虽然现在有很多工具能够帮助开发人员减少一些复杂的操作，但是开发人员还是需要更多的接触运维的工作。

但是这种运维方式，同时也给项目开发过程中带来了很多困难。

一方面，开发人员只能向运维人员描述具体的部署方式。但是由于开发人员无法接触到生产服务器，所以文字描述的方式往往很难保证操作的准确性。经常会出现开发人员在开发环境运行得很好的迭代包，升级到生产环境上之后无法保证升级的效果。对于现在流行的基于敏捷开发的大型项目来说，很多需求需要以代码分支的方式进行并行开发，然后再合并部署，这其中更是非常容易出现错误，造成生产环境不稳定。

另一方面，当项目在线上运行出现故障时，开发人员也很难第一时间接触到错误日志。如果线上出现问题，开发人员往往需要找运维部门协同，才能获取到生产环境的服务日志。这会极大的延缓错误排查的及时性。

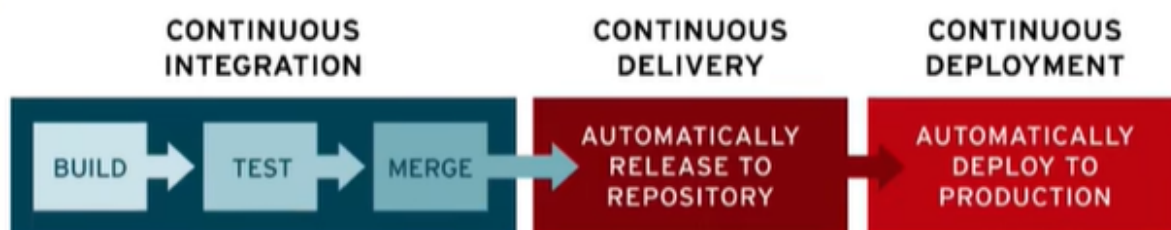
2、DevOps与CI\CD

以上那些问题的核心，其核心就是在传统运维体系下，开发和运维之间是有天然的壁垒和鸿沟的。而DevOps则是试图打破这些壁垒鸿沟的一种方法论。DevOps是Development(开发)和Operation(运维)两个单词的组合，它是一种重视软件开发人员和运维技术人员之间沟通合作的文化、运动或者惯例。通过自动化软件交互和架构变更的流程，使得构建、测试、发布软件时能够更加快捷、频繁和可靠。

总而言之，DevOps是一个标准，一种方法论或者说是一个目标，并不指一个特定的规则或者一系列特殊的工具。那要如何落地DevOps呢？通常，这就需要CI\CD出马了。

CI\CD中的CI指的是持续集成Continuous Integration，他是开发人员的自动化过程。成功的CI意味着人员同的新代码变更会定期构建、测试并合并到共享存储库(比如Git或者SVN)。而CD指的是持续交付Continuous Delivery和持续部署Continuous Deployment。成功的CD意味着运维人员可以从共享存储库中持续获取到最新的产品副本，并确保最新的产品副本可以正确更新到服务器上。

CI/CD



关于如何达成CI\CD，各大互联网厂商提供了大量的方法论以及工具。下面就以电商项目为例，基于最为典型的GitLab和Jenkins搭建一套简单的自动部署环境。

二、使用GitLab+Jenkins搭建CI\CD执行环境

关于GitLab和Jenkins的安装过程，这里只简单介绍一下关键步骤。如果对安装有问题的同学，可以自行到网上搜索一下安装教程，网上的资料非常多。

1、GitLab安装

GitLab是企业中最为常用的私有代码仓库解决方案。你可以把他理解为一个企业自己搭建的GitHub或者Gitee(程序员交友平台，不会没用过把？)。企业通常会通过GitLab搭建自己的代码仓库，开发人员的应用代码都通过GitLab进行协同开发。

GitLab是一个开源项目，分为免费的ce社区版和收费的ee企业版。这里介绍ce社区版的安装过程。

首先要检查服务器配置。GitLab需要部署非常多的后台服务，通常建议单机内存不要低于4G。如果配置太低的话，会出现很多奇怪的问题。Linux服务器需要提前安装几个服务 `yum install -y curl policycoreutils-python openssh-server`。如果已经安装了，这一步可以省略。

然后获取GitLab安装包。社区版的GitLab下载地址<https://packages.gitlab.com/gitlab/gitlab-ce>。电商项目中采用的Linux服务器，就可以选择下载gitlab-ce-15.1.0-ce.0.el7.x86_64.rpm。

接下来就可以安装GitLab了。执行`rpm -Uvh gitlab-ce-15.1.0-ce.0.el7.x86_64.rpm` 开始安装。

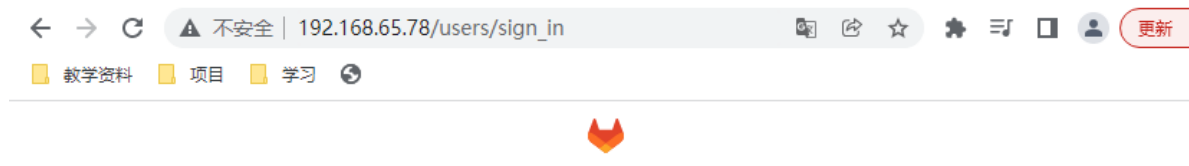
安装完成后，第一次运行GitLab前，需要执行一次配置初始化操作。 `gitlab-ctl reconfig`。这个过程耗时比较长。

接下来就可以使用gitlab-ctl指令来操作gitlab服务了。

- `gitlab-ctl reconfigure` 重新配置gitlab。
- `gitlab-ctl start` 启动gitlab
- `gitlab-ctl stop` 停止gitlab

- gitlab-ctl restart 重启gitlab
- gitlab-ctl status 查看gitlab服务状态
- gitlab-ctl tail 查看gitlab服务日志。

服务启动完成后，就可以访问gitlab服务了。默认的服务端口就是80端口。默认的用户名和密码是root/123456(通常建议登录后立即修改默认密码)。



GitLab

A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

Username or email

Password

☐ Remember me [Forgot your password?](#)





Sign in

Don't have an account yet? [Register now](#)

如果需要修改访问的地址和端口，可以修改/etc/gitlab/gitlab.rb配置文件，修改其中的external_url属性即可。

```
## GitLab URL
##! URL on which GitLab will be reachable.
##! For more details on configuring external_url see:
##! https://docs.gitlab.com/omnibus/settings/configuration.html#configuring-
the-external-url-for-gitlab
##!
##! Note: During installation/upgrades, the value of the environment
variable
##! EXTERNAL_URL will be used to populate/replace this value.
##! On AWS EC2 instances, we also attempt to fetch the public hostname/IP
##! address from AWS. For more details, see:
##! https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-
retrieval.html
external_url 'http://192.168.65.78'
```

接下来就可以登录进入GitLab，维护基础权限信息，并将项目代码上传到Git仓库当中。这些基础的操作跟GitHub或者Gitee基本上是一样的。

M	tlmall / mall-admin-web  Owner	★ 0 🗨 0 📄 0 📁 0	更新于 1个月前
M	GitLab Instance / Monitoring  Owner This project is automatically generated and helps monitor this GitLab instance. Learn more.	★ 0 🗨 0 📄 0 📁 0	更新于 2个月前
T	tlmall / tmall-front  Owner	★ 0 🗨 0 📄 0 📁 0	更新于 1个月前
T	tlmall / tmallIV5  Owner	★ 0 🗨 0 📄 0 📁 0	更新于 4天前

2、Jenkins安装

Jenkins是企业最常用的一个自动化部署软件。下载地址为<https://www.jenkins.io/download/>。建议下载LTS(长期支持)版本的war包部署。下载获取jenkins.war文件。

首先，需要安装JDK。Jenkins运行需要JDK环境支持，目前Jenkins建议使用JDK11版本。

然后，就可以直接启动Jenkins。启动指令 `java -jar Jenkins.war`。当然，你也可以使用后台执行的方式。 `nohup java -jar jenkins.war &`。这种方式不会占用当前命令行窗口，日志输出到nohup.out下。

在启动时，也可以指定端口。 `java -jar Jenkins.war --httpPort=8080`。端口默认就是8080端口。

在第一次启动的过程中，Jenkins会在日志文件中打印默认的admin用户密码。这个需要留意一下。

```
*****

Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:

e3c2de8a2084429ea733ef30512a0523

This may also be found at: /root/.jenkins/secrets/initialAdminPassword

*****
```

启动完成后，就可以访问Jenkins的前台管理页面<http://192.168.65.200:8080/>。第一次访问时，前端页面会引导进行一些初始化工作。例如，需要输入admin用户的默认密码，这个密码就在启动日志当中。

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

 initialAdminPassword

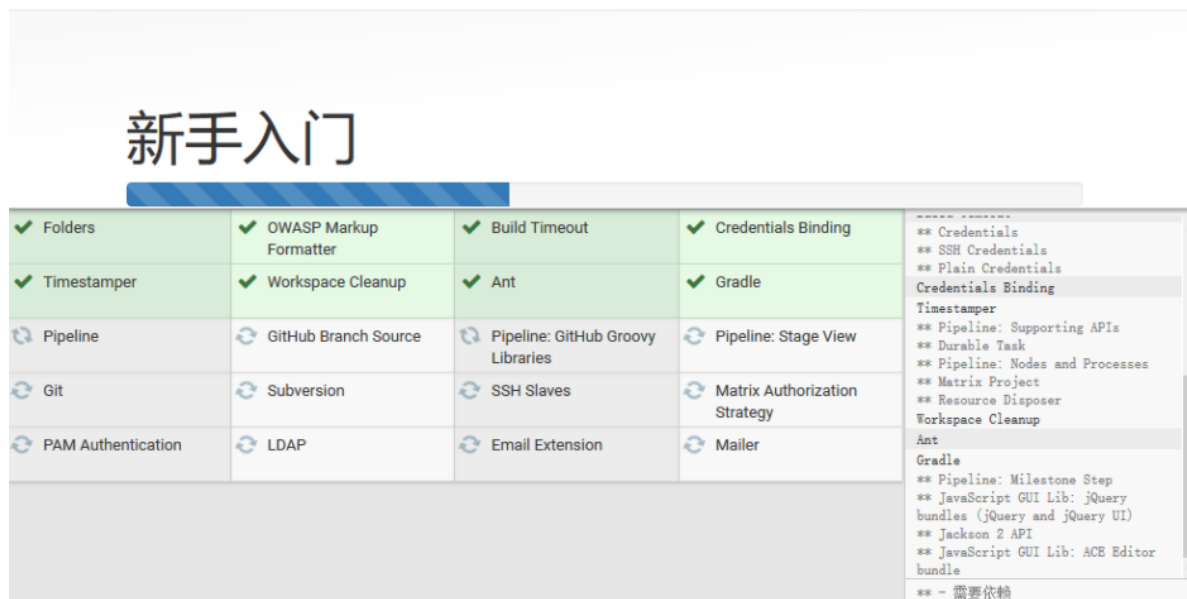
Please copy the password from either location and paste it below.

Administrator password

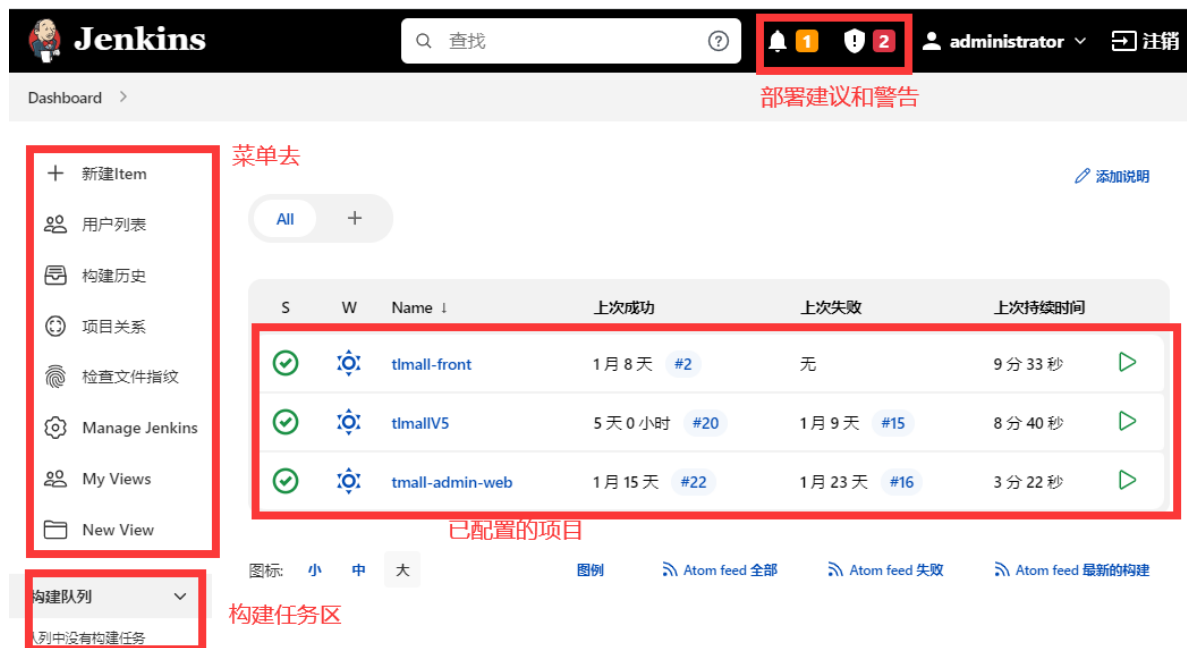
后续还会引导设定admin用户的用户名。

然后会引导安装一些插件。这一步比较自由。你可以按照默认方式安装，也可以选择一些你认识的常用插件安装。关键插件漏了没有关系，后续也可以再安装插件。

新手入门



引导步骤安装完成后，就可以进入Jenkins的首页了。



接下来需要安装几个核心的插件。选择 Manage Jenkins-> Manage Plugins，进入插件管理页面。

Plugin Manager

可更新

可选插件

已安装

高级

搜索插件

Q 过滤

安装

名称 ↓

选择插件

Apache HttpComponents Client 4.x API 4.5.13-138.v4e7d9a_7b_a_e61

Library plugins (for use by other plugins)



Bundles Apache HttpComponents Client 4.x and allows it to be used by Jenkins

This plugin is up for adoption! We are looking for new maintainers. Visit

Bootstrap 5 API 5.2.0-1

Library plugins (for use by other plugins)

Provides Bootstrap 5 for Jenkins Plugins. Bootstrap is (according to their self) the best way to build on the web.



A newer version than being offered for installation exists (version 5.2.0-3). The current requirements, e.g. a recent version of Jenkins, are not satisfied. If you are not ready to use it yet. See the [plugin documentation](#) for information about its requirements.

Build Timeout 1.24



Build Wrappers

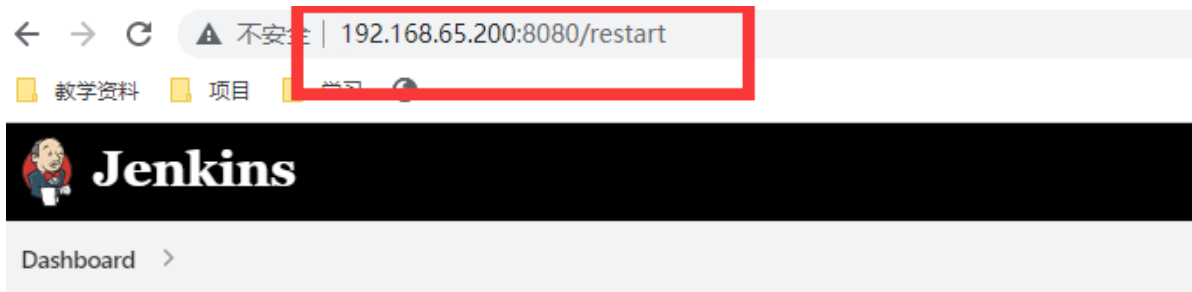
安装插件

下载待重启后安装

5 小时 34 分 之前获取了更新信息

立即获取

在这里需要安装几个核心的插件。包括Git、Git client、NodeJS Plugin、Maven integration plugin。如果你希望Jenkins能够更多的显示中文，还可以安装 Localization:Chinese(Simplified)插件。下载完成后有些插件需要重启才能生效。Jenkins重启的方式是直接浏览器上访问restart接口。



+ 新建Item

你确定要重启 Jenkins 吗?

是

田白列丰

重启完成后，还需要配置几个基础的组件。进入Manage Jenkins->Global Tool Configuration页面。在这里需要对Maven、Git和NodeJS组件进行配置。你可以选择按照页面提示，自动下载安装对应的组件。当前电商环境中是直接另外去安装对应的组件。

例如对于git，只要服务器能够支持git指令接口。如果没有安装的话，可以使用yum -install git 安装git客户端。

```
[root@192-168-65-200 ~]# git --version
git version 1.8.3.1
```

而Maven和NodeJS插件，可以去官网下载对应的压缩包，解压后，将bin子目录配置到环境变量当中。跟安装JDK的方式差不多。同样只要服务器能够直接支持mvn指令和node指令即可。

```
[root@192-168-65-200 ~]# mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /app/maven/apache-maven-3.6.3
Java version: 11.0.10, vendor: Oracle Corporation, runtime: /app/jdk/jdk-11.0.10
Default locale: zh_CN, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-1160.el7.x86_64", arch: "amd64", family:
"unix"
[root@192-168-65-200 bin]# node -v
v14.15.0
```

到这里基础环境就算搭建完成了。

3、基于GitLab+Jenkins快速实现CI\CD

接下来需要在Jenkins中配置一个构建任务。下面就以电商项目的后端工程为例，演示配置过程。另外电商项目的两个前端工程也可以以类似的方式进行配置。

1、创建一个maven项目

在Jenkins首页选择新建Item，然后选择构建一个maven项目

输入一个任务名称

tulingmallV5

» 必填项

 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be used to build any project.

 **构建一个maven项目**
构建一个maven项目.Jenkins利用你的POM文件,这样可以大大减轻构建配置.

2、配置项目构建及部署过程

接下来在Jenkins中配置项目的配置项还是挺多的，这里只列出几个关键的配置。其他部分可以自行调整。

首先需要在**源码管理**部分配置对应的git仓库地址。

源码管理

☐ 无

☒ Git ?

Repositories ?

Repository URL ? git仓库地址

http://192.168.65.78/tlmall/tlmallv5.git

Credentials ? git用户名密码，没有就添加

roykingw@163.com/****** (tlmall项目git仓库用户名)

+ 添加

高级...

Add Repository

Branches to build ?

指定分支 (为空时代表any) ?

*/RunnableVer 要构建的分支

构建触发器部分，可以选择配置Poll SCM选项。这个选项可以定时扫描Git代码仓库。当发现Git仓库代码有变化，即有代码提交时，就会触发一次构建任务。当前电商项目采用手动控制发布的方式，就没有选择配置了。

构建触发器

☐ Build whenever a SNAPSHOT dependency is built ?

☐ 触发远程构建 (例如,使用脚本) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

Configure Jenkins to poll changes in SCM.--
Note that this is going to be an expensive operation for CVS, as every polling requires Jenkins to scan the entire workspace and verify it with the server. Consider setting up a "push" trigger to avoid this overhead, as described in [this document](#)

这里面Poll SCM是一个比较常用的配置。通过这个配置，可以让jenkins定期去检查代码库。如果发现代码库有更新，则自动触发当前任务，完成项目的构建以及部署。当前电商项目编译任务太重，提交也不太频繁，所以就选择不配置该属性。

构建环境部分建议选择一下JDK版本，因为电商项目采用的是JDK8版本进行开发。

构建环境

☐ Delete workspace before build starts

☒ Use secret text(s) or file(s) ?

☐ Provide Configuration files ?

☐ Send files or execute commands over SSH before the build starts ?

☐ Send files or execute commands over SSH after the build runs ?

☐ Add timestamps to the Console Output

☐ Inspect build log for published Gradle build scans

☐ Provide Node & npm bin/ folder to PATH

☐ Terminate a build if it's stuck

☒ With Ant ?

JDK

jdk1.8

接下来的**Build**部分，就可以选择需要执行的编译脚本。

Build

Root POM ?

pom.xml

Goals and options ?

package -Dmaven.test.skip=true

高级...

Post Steps

☒ Run only if build succeeds

☐ Run only if build succeeds or is unstable

☐ Run regardless of build result

Should the post-build steps run only for successful builds, etc.

这一步相当于指定使用 `mvn package -Dmaven.test.skip=true` 指令对后端项目进行重新打包构建。

如果是前端项目，就需要用Nodejs的`npm run build`指令来构建。

构建完成之后，就会在后端项目的各个模块的`target`目录下生成可执行的包。这时，可以选择用Jenkins将这些Jar包分发到远程服务器上，并直接运行。

Send files or execute commands over SSH ?

SSH Publishers

SSH Server

Name ?

timallapp1 目标服务器

高级...

Transfers

Transfer Set

Source files ? 执行的jar包地址。从项目根路径开始查找

tulingmall-admin/target/tulingmall-admin-0.0.5-exec.jar

Remove prefix ? 不要前缀。表示只传对应的jar包。

tulingmall-admin/target

Remote directory ? 远程服务器上的目标地址

/tulingmall

Exec command ? 在远端服务器上执行的启动脚本

ps -ef | grep "tulingmall-admin" | grep -v grep | awk '{print \$2}' | xargs kill -9
java -jar /app/tulingmall/tulingmall-admin-0.0.5-exec.jar 1>/app/tulingmall/tulingmall-admin.log 2>&1 &

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

高级...

前端项目也可以用同样的方式传递到远端服务器上。远端指令只需要执行 `nginx -s reload` 更新一下即可。

这样就完成了一个基础项目的配置过程。接下来，保存之后，就可以选择Jenkins首页对应项目右侧的三角指令发起一次构建了。构建过程中如果有问题，可以查看构建日志，进行排查。

这里完成的是一个最简单的Maven项目构建。实际上，在企业中，还会构建更复杂的任务。

例如集成Docker，进行虚拟化部署。

添加Blue Ocean插件做一些代码质量检测，邮件通知等复杂的步骤。甚至使用Pipeline构建更为复杂的任务流水。

或者搭建SonarQube服务，并通过jenkins集成。这样就可以使用SonarQube服务来进行代码质量检测。

三、电商后端项目打包及部署方式

从刚才的部署过程中可以看到，基于Jenkins的CI/CD配置方式就是执行Maven对项目进行编译，然后将Jar包传到远端服务器上执行。这个过程跟我们手动进行任务部署是差不多的，只不过Jenkins将这些过程自动完成了。

但是这里其实有一个小问题。当前电商项目在编译时，是在pom.xml中采用SpringBoot的Maven插件将整个项目打成了一个可执行的Jar包。这样打出来的Jar包可以直接使用java -jar指令执行。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>2.3.2.RELEASE</version>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
      <!-- 解决运行包不能被其他包依赖的问题 -->
      <configuration>
        <classifier>exec</classifier>
      </configuration>
    </plugin>
  </plugins>
</build>
```

但是，这种方式也有一个很大的问题，就是Jar包太大了。这么大的Jar包，编译会很耗时，并且在网络中传输是非常麻烦的。所以，对于一些大型项目，通常不会采用这种一体化的fat Jar的方式。而会选择将依赖单独打成小的Jar包。这样后续每次更新只需要更新调整过的少量Jar包即可。那有哪些Maven打包方式可以帮助我们打出小的jar包呢？

其实Maven提供了很多的打包插件。例如将上面的plugin部分替换为maven-dependency-plugin，就可以将项目打成小包。

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <!-- <version>2.10</version> -->
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>export</outputDirectory> <!-- 将依赖包放入export文
文件夹 -->
        <excludeTransitive>false</excludeTransitive>
        <stripVersion>true</stripVersion>
      </configuration>
    </execution>
```

```
</executions>
</plugin>
```

通过这种方式，就可以将所有依赖的jar包都放到export文件夹中，target目录下的jar包只包含当前项目的源码，文件大小就会小很多。

将export目录下的所有jar包和target下的当前项目jar包上传到服务器的同一个目录当中，就可以直接运行了。这种方式就跟很多开源框架的运行方式相似了。至于要如何运行呢？当然就不能用java -jar指令简单执行了，需要通过java -cp指令指定依赖包和主启动类执行。至于如何使用这个指令，你之前学过的很多开源组件都可以提供帮助了。比如shardingproxy、RocketMQ等，都有这样的脚本。

这里也给出一个简单的Linux执行脚本示例，供你参考。

```
more runapp.sh

#!/bin/sh
#执行jar包
RUN_LIBS=""
#依赖jar包 自行制定目录
SUPPORT_LIBS=""
RUN_LIB_PATH="/app/lib"
SUPPORT_LIB_PATH="/app/support"
#加载程序包
for i in ${RUN_LIB_PATH}/* ; do
    RUN_LIBS=${RUN_LIBS}:${i}
done
#加载依赖包
for i in ${SUPPORT_LIB_PATH}/* ; do
    SUPPORT_LIBS=${SUPPORT_LIBS}:${i}
done
#整合classpath
CLASSPATH=${RUN_LIBS}:${SUPPORT_LIBS}
export CLASSPATH
#调用java指令执行。-D输入参数 java中可以用 System.getProperties读取。同时指定执行入口类
SpringBootApplication 这是一个典型的Springboot的执行方式。
java -Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,address=27899,suspend=n -cp $CLASSPATH -
Dspring.profiles.active=prod com.tuling.TulingmallCartApplication -D
user.timezone=GMT+08 1>tulingmall-admin.out 2>tulingmall-admin.err &
echo Start App Success!
```

实际上这样定制脚本对于提高运行效率是非常重要的，你学了很多次的VM调优就体现在脚本定制的过程中。在脚本最后的java指令中，可以添加哪些优化参数？

另外，这种部署方式，你可以自己尝试用jenkins配置自动部署吗？

实际上，Maven还提供了非常多插件。比如对于很多复杂的项目，可能需要将不同的模块输出到多个不同的jar包当中，而不是所有代码全都输出一个jar包。下面就是使用maven-jar-plugin插件的一个可行的示例，将不同模块的代码分别整合到myapp.jar和myapp2.jar中。有兴趣你可以自己尝试下。

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-jar-plugin</artifactId>
<version>3.0.2</version>
```

```

<configuration> <!-- manifest配置信息 主要是可以配置主执行类。有主执行类，可以用java-
jar直接执行。没有的话就需要指定执行类 -->
  <archive>
    <manifest>
      <addClasspath>true</addClasspath>
      <classpathPrefix>support/</classpathPrefix>
      <mainClass>com.myapp.MyAppApplication</mainClass>
    <!-- 可以按上面的方式自己配置，也可以指定MF文件打包。 -->
    <manifestFile>${project.build.outputDirectory}/META-
INF/MANIFEST.MF</manifestFile>
  </manifest>
</archive>
</configuration>
<executions>
  <execution>
    <id>myapp1-jar</id>
    <phase>package</phase>
    <goals>
      <goal>jar</goal>
    </goals>
    <configuration>
      <classifier>myapp</classifier>
      <includes>
        <include>com/myapp/**</include>
        <include>mybatis/**</include>
        <include>templates/**</include>
        <include>*.properties</include>
        <include>dubbo.xml</include>
      </includes>
    </configuration>
  </execution>
  <execution>
    <id>myapp2-jar</id>
    <phase>package</phase>
    <goals>
      <goal>jar</goal>
    </goals>
    <configuration>
      <classifier>myapp2</classifier>
      <includes>
        <include>com/myapp2/crawler/*</include>
        <include>com/myapp2/crawler/*</include>
        <include>com/myapp2/utis/**</include>
        <include>log4j.properties</include>
      </includes>
    </configuration>
  </execution>
</executions>
</plugin>

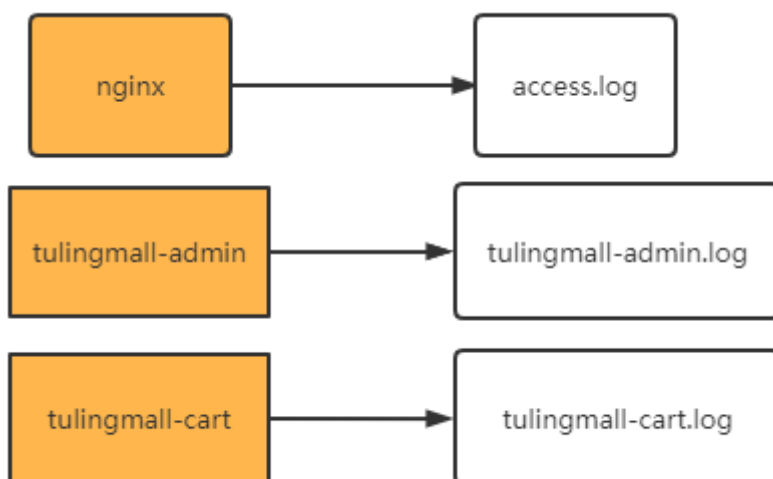
```

如果你对Maven感兴趣，可以去Maven官网上看看Maven目前官方提供的插件。 <https://maven.apache.org/plugins/index.html>。这上面能找到非常多有趣的插件，并且都有详细的说明。比如changelog插件，可以打印出Maven仓库中最近的提交记录。checkstyle和pmd插件可以对代码进行静态检查。javadoc插件可以打印出项目文档，你还可以用pdf插件，打印出pdf版本的项目文档。或者使用antrun

插件去执行一些ant脚本(老程序员应该对ant很熟悉)。很多精彩等你发现。

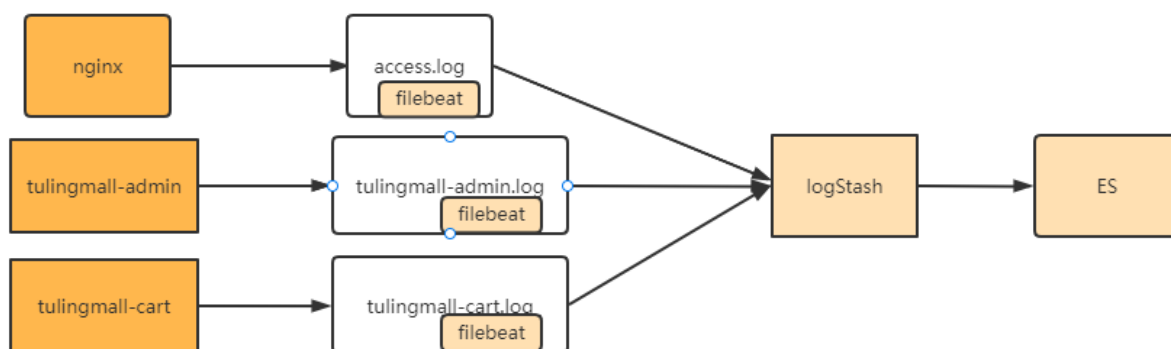
四、使用FileBeat+Logstash+ES实现分布式日志收集。

这一章节是一个补充的内容。在大型项目中，往往服务都是分布在非常多不同的机器上，每个机器都会打印自己的log日志。



但是，这样分散的日志，本来就无法进行整体分析。再加上微服务的负载均衡体系，甚至连请求打到了哪个服务器上都无法确定。给问题排查带来了很多的困难。因此就需要将分散的日志收集到一起，这样才能整体进行分析。

在Java应用中，后续我们会介绍使用skywalking，基于微服务架构进行整体链路追踪。但是这种方式会显得比较重。如果只是分析nginx这样的中间件，skywalking显然就无能为力了。因此，还需要一个比较简单快捷，对应用无侵入的方式统一收集日志。通常，业界常用的还是通过ELK中间件来收集日志。整体的流程是这样的。



filebeat,logstash和es都是ELK组件中的标准处理组件。其中，ES是一个高度可扩展的全文搜索和分析引擎，能够对大容量的数据进行接近实时的存储、搜索和分析操作，通常会跟Kibana部署在一起，由Kibana提供图形化的操作功能。LogStash是一个数据收集引擎，他可以动态的从各种数据源搜集数据，并对数据进行过滤、分析和统一格式等简单操作，并将输出结果存储到指定位置上。但是LogStash服务过重，如果在每个应用上都部署一个logStash，会给应用服务器增加很大的负担。因此，通常会在应用服务器上，部署轻量级的filebeat组件。filebeat可以持续稳定的收集简单数据，比如Log日志，统一发给logstash进行收集后，再经过处理存到ES。

这一套流程是企业中最为基础的分布式日志收集方案。这一节课就带大家实际搭建一个filebeat和logstash服务，用来收集前端项目的nginx日志，然后将nginx日志经过logstash保存到es中。

关于ES以及配套的Kibana，有VIP课程带大家搭建使用，这里就不介绍如何搭建了。只是介绍一下filebeat和logstash的搭建过程。

首先，搭建LogStash

去官网下载与ES配套的LogStash 7.17.3版本发布包logstash-7.17.3-linux-x86_64.tar.gz。下载地址：<https://www.elastic.co/cn/downloads/past-releases#logstash>。使用tar -zxvf logstash-7.17.3-linux-x86_64.tar.gz 将压缩包解压到es用户根目录。

解压完成后需要配置Logstash需要的JDK。这个JDK不需要额外下载，在logstash的安装目录下有一个jdk目录，里面有内置的配套JDK。这时，需要配置一个环境变量LS_JAVA_HOME指向这个内置的JDK即可。

接下来可以简单启动一下logstash进行测试。进入logstash的安装目录，启动一个简单的logstash任务。

```
bin/logstash -e 'input { stdin { } } output { stdout { } }'
```

这个任务启动需要一定的时间。

启动完成后，就可以从logstash的控制台输入信息，然后又重新输出到控制台中。使用ctrl+D退出控制台。

```
#控制台输入hello
hello
# 控制台输出logstash处理结果
{
  "message" => "hello",
  "@version" => "1",
  "host" => "es-node3",
  "@timestamp" => 2022-09-14T02:14:05.709Z
}
```

这样一个简单的logstash就安装完成了。

接下来需要对logstash的输入和输出目录进行配置。进入config目录，在目录下直接修改logstash-sample.conf文件即可。

配置文件名字可以随便取，后续启动时需要指定配置文件。

```
# Sample Logstash configuration for creating a simple
# Beats -> Logstash -> Elasticsearch pipeline.

input {
  beats {
    port => 5044
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
}
```



```

output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    #index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
    index => nginxlog
    user => "elastic"
    password => "123456"
  }
}

```

这个配置中：

input表示输入，这里表示从filebeat输入消息，接收的端口是5044。

output表示数据的输出，这里表示将结果输出到本机的elasticsearch中，索引是nginxlog。

filer表示对输入的内容进行格式化处理。这里指定的grok是logstash内置提供的一个处理非结构化数据的过滤器。他可以以一种类似于正则表达式的方式来解析文本。简单的配置规则比如：%{NUMBER:duration} %{IP:client} 就是从文本中按空格，解析出一个数字型内容，转化成duration字段。然后解析出一个IP格式的文本，转换成client字段。而示例中使用的COMBINEDAPACHELOG则是针对APACHE服务器提供的一种通用的解析格式，对于解析Nginx日志同样适用。

一条nginx的日志大概是这样：

```

83.149.9.216 - - [04/Jan/2015:05:13:42 +0000] "GET /presentations/logstash-monitorama-2013/images/kibana-search.png
HTTP/1.1" 200 203023 "http://semicomplete.com/presentations/logstash-monitorama-2013/" "Mozilla/5.0 (Macintosh; Intel
Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77
Safari/537.36"

```

解析出来的是一个json格式的数据，包含以下字段

Information	Field Name
IP Address	clientip
User ID	ident
User Authentication	auth
timestamp	timestamp
HTTP Verb	verb
Request body	request
HTTP Version	httpversion
HTTP Status Code	response
Bytes served	bytes
Referrer URL	referrer

Information	Field Name
User agent	agent

配置好这个文件后，就可以直接启动了。

```
nohup bin/logstash -f config/logstash-sample.conf --config.reload.automatic &
```

config.reload.automatic表示配置自动更新，也就是说以后只要改动了配置文件，就会及时生效，不需要重启logstash

nohup指令只是表示不要占据当前控制台，将控制台日志打印到nohup.out文件中。

logstash更详细的配置说明参见官方文档：<https://www.elastic.co/guide/en/logstash/7.17>

然后，搭建filebeat

之前已经启动了logstash服务，通过5044端口监听filebeat服务。接下来就需要在各个应用服务器上部署filebeat，往logstash发送日志消息即可。

filebeat的下载地址：<https://www.elastic.co/cn/downloads/past-releases#filebeat>。同样选择配套的7.17.3版本filebeat-7.17.3-linux-x86_64.tar.gz。并使用tar -zxvf filebeat-7.17.3-linux-x86_64.tar.gz指令解压。

在解压目录下已经提供了一个模版配置文件filebeat.yml，我们只需要修改这个文件即可。这个模板文件里面的示例非常清楚，从文件读取日志，输出到logstash的配置，文件当中都有。这里只列出修改的部分。

先修改文件输入的部分配置

```
# ===== Filebeat inputs =====
filebeat.inputs:
- type: filestream
  # Change to true to enable this input configuration.
  enabled: true
  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /www/wwwlogs/access.log
    #- c:\programdata\elasticsearch\logs\*
```

然后修改输出到logstash的部分配置

```
# ----- Logstash Output -----
output.logstash:
  # The Logstash hosts
  hosts: ["192.168.65.114:5044"]
```

默认打开的是output.elasticsearch，输入到es，这部分配置要注释掉。

这样就完成了最简单的filebeat配置。接下来启动filebeat即可

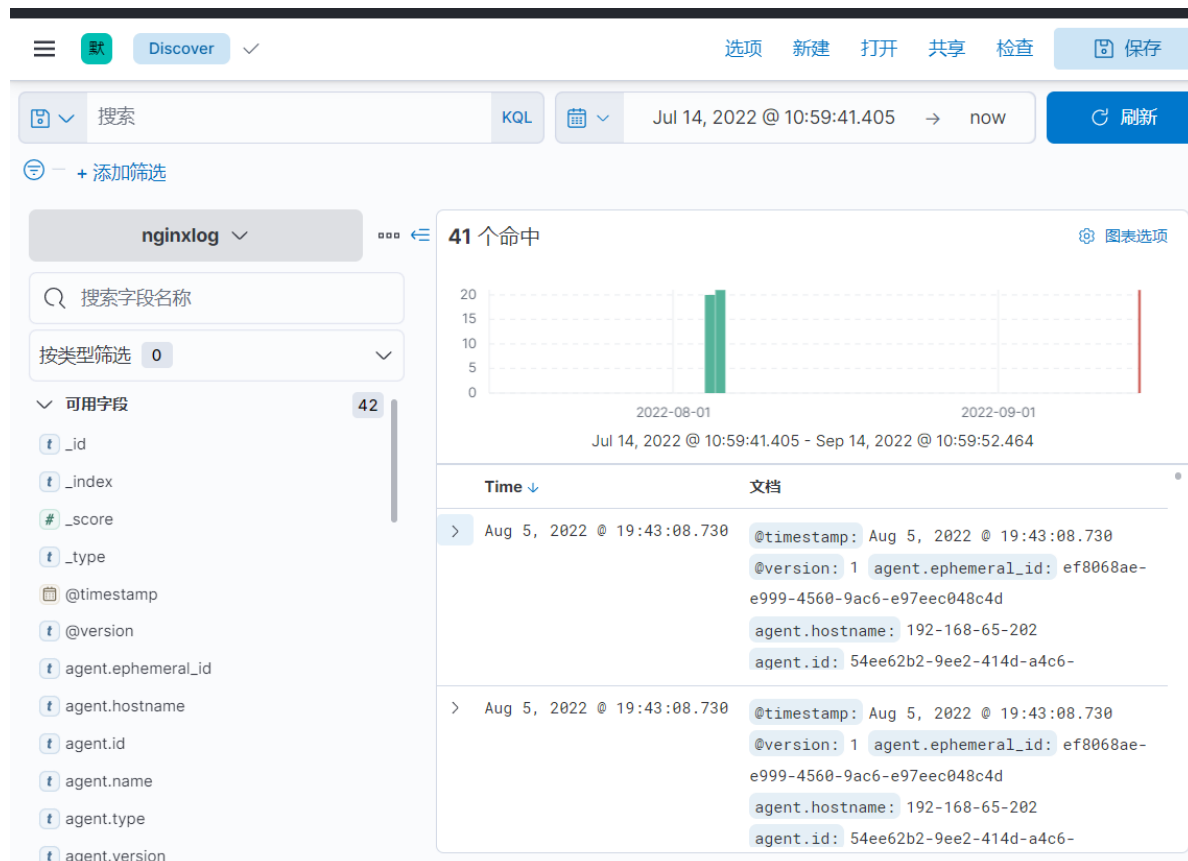
```
nohup ./filebeat -e -c filebeat.yml -d "publish" &
```

filebeat任务启动后，就会读取nginx的日志，一旦有新的日志记录，就会将日志转发到logstash，然后经由logstash再转发到ES中。并且filebeat对于读取过的文件，都是有记录的，即便文件改了名字也不会影响读取的进度。比如对log日志，当前记录的log文件，即便经过日志轮换改成了其他的名字，读取进度也不会有变化。而新生成的log日志也可以继续从头读取内容。如果需要清空filebeat的文件记录，只需要删除安装目录下的data/registry目录即可。

更详细的配置参见官方文档：<https://www.elastic.co/guide/en/beats/filebeat/7.17/logstash-outputs.html>

接下来，进入ES查看数据是否生效

进入Kibana的前端页面，即可查询到nginxlog索引下的日志记录



后续就可以针对这些nginx的日志信息，进行分析。nginx的日志基本上是所有大型项目进行日志收集必不可少的一个重要数据来源，从nginx的日志中可以分析出大量有用的结果。比如最常见的PV，UV，还有热点功能等。

课上就只带大家搭建最简单的一组服务了。在搭建过程中可以看到，filebeat和logstash对于常见的输入输出源都已经提供了实现，大部分情况下，只需要简单配置即可。在实际项目中，往往会以此为基础构建更复杂的分布式日志处理方案。比如在logstash后增加一个Kafka，将LogStash收集的日志消息存入到kafka，再经过基于Kafka的流式计算，将PV，UV这类的统计结果存入ES。

有道云笔记链接地址：<https://note.youdao.com/s/5otHMx0x>