　本文只简单标记下大概的步骤，具体搭建各个部分的细节，还请自行搜索。第一、二部分只是对[Jenkins](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)和Docker的简单介绍，熟悉的同学请直接跳到第三部分。

**一、关于Jenkins**

**Jenkins简介**

　　Jenkins可以帮你构建一个自动化的持续集成环境，你可以使用它来“自动化”编译、打包、分发部署你的应用，它兼容ant、maven、gradle等多种第三方构建工具，同时跟svn、git能无缝集成，也支持直接与知名源代码托管网站，比如github、bitbucket直接集成。

**Jenkins安装**

　　Jenkins的安装十分简单，你可以从官网http://mirrors.jenkins-ci.org/war/latest/ 直接下载war包，使用[java](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self) -jar jenkins.war，或者放到Tomcat类似的容器里运行都可以(JDK版本建议使用1.6以上来运行)。Jenkins默认启动端口是8080，请注意端口占用。另外，Jenkins文件其实都在~/.jenkins目录下，运行过程中你可以在这里找到你运行过的job和相应的日志文件。

　　首次安装，可以把插件都安装上，自己都试试看。如果因为网络原因，插件下载失败的，可以从失败详情中把地址拷出来，下载后，以上传hpi文件的方式安装。

**系统设置**

　　安装完成的第一步，先到Blobal Tool Configuration中配置相关的环境，如JDK、MAVEN、GIT等，相应的路径设置都是设置Jenkins这台机子本地的目录。建议把所有的配置都完善下，否则日后的运行中可能会出现一些小问题，可以试下点击？查看帮助，还是很有用的。

**Jenkins集群配置**

　　集群配置也十分简单，就是添加slave节点的过程。点击系统管理->管理节点->新建节点，各个配置后面都有详细的帮助说明的。在启动方法这里推荐使用 Launch slave agents on UNIX machines via SSH，这样master节点会主动使用SSH登陆到slave节点上，并运行java -jar slave.jar，这样一个slave节点就添加完成了。

**构建Job**

　　其实到这一步，就是jenkins的常规流程了，你可以参照创建Job选项的界面的各个配置，制定你构建计划，或者我们直接使用脚本来完成我们的构建操作（建议）选择构建Execute [shell](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)，[文章](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)末尾会附上一个完整项目的shell构建流程。

**Pipeline**

　　Pipeline字面意思就是流水线，很好用的Jenkins的一个插件，将很多步骤按顺序排列好，做完一个执行下一个。真实的工作环境有很多job，比如先编译，然后执行静态代码检查、[单元测试](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)、然后部署服务器、服务器重启、进行ui测试等。我们需要对这些job进行一些设置将它们的上下游关系配置好。

　　这时候Pipeline就派上用场了。在Jenkins主页点击新建视图，然后选择Build Pipeline View，然后填写Select Initial Job，构建后的操作可以选择其他的Job，以此类推，这样一个流水线就完成了。Pipeline提供视图界面，你可以在视图上形象地看到整个构建计划的执行流程和完成度。

**二、关于Docker**

**Docker 简介**

　　Docker现在是Github社区最火的项目之一，Docker是个容器，或许你听过lxc，你可能知道Tomcat这个[Web](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)容器，容器是什么概念，意会就好。问个问题，你想在一台机子上运行一个java6的应用，又想运行一个java8应用怎么办；或者说你开发出一个软件，mac版、[windows](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)版本，我都想运行，怎么办？装虚拟机？不不不，虚拟机太浪费资源了，Docker这时候就派上用场了，在电脑上，你可以同时装一个mac的容器和windows的容器哦。一个虚拟机可能会消耗2到3G的内存，一个容器仅仅只消耗几百M的资源哦。

**Docker 安装**

　　具体安装细节，可以参照https://docs.docker.com，文档说得很清楚了。 Docker原本是开发在[Linux](http://www.51testing.com/html/47/javascript:;" \t "http://www.51testing.com/html/47/_self)上的，Windows下是采用Docker machine的方式，起一个虚拟机在虚拟机里面跑Docker。本来呢Mac也是采用虚拟机的方式，后来有了Docker for mac，采用HyperKit，一种更轻量级的虚拟化方式。当然，你还是可以用Docker machine的方式运行。

**Docker 基础**

　　搞清楚两个概念，容器与镜像。打个比方吧。装机，容器就是装完成后的系统，镜像就是ISO安装盘，系统镜像。

　　作为一个合格的程序员，你一定用过Git的吧，pull，commit，push……写代码过程就是这样，先拉取代码pull，然后是在一个基础的文本上不断地增加commit，加积木一样，一层层叠加，累积上去，最后再push上去。没错Docker就是采用这种形式。

**简单介绍下Docker命令：**

　　Docker version / Docker info 查看基本信息，遇到使用问题或者BUG，可以到社区里报告，记得带着这些信息哈。

　　Docker pull 拉取镜像。

　　Docker run 从镜像创建一个容器并运行指定的命令常用参数-i -d，建议用—-name命名这个容器，命名后可以使用容器名访问这个容器。

　　Docker attach（不推荐使用）。

　　Docker exec -ti CONTAINER /bin/bash 连接到容器上运行bash

　　Docker logs CONTAINER 查看日志，如run命令后的运行结果，Docker logs -f 查看实时的日志。

　　Docker kill 杀死Docker容器进程，你可以使用Docker kill $(Docker ps -aq)杀死所有的Docker进程，后者打印出所有的容器的容器id（包括正在运行的，和没有运行的）。

　　Docker rm CONTAINER 删除一个容器，记得要先停止正在运行的容器，再去删除它。

　　Docker exec -it <container\_id> bash -c 'cat > /path/to/container/file' < /path/to/host/file/容器外向容器内复制文件（也可以用挂载的形式哦）。

　　Docker commit -a “mike” -m “镜像的一些改动” CONTAINER 当你在容器内做了某种操作后，如增加了一个文件，你可以用这个命令把修改提交，重新打包为镜像。

　　Docker push 推送镜像。。到这里是不是觉得跟Git的模式已经有点像了呢。

　　Docker history IMAGES 查看镜像的修改历史。

　　Docker ps -a | grep "Exited" | awk '{print $1 }'| xargs Docker rm 删除exited的容器。

　　Docker rmi $(Docker images | awk '/^<none>/ {print $3}') 删除tag为NONE的容器。

**Dockerfile基础**

　　Dokcerfile，是的，你还是要稍微掌握下Dockerfile的写法。

　　From 每个Dockerfile镜像的构建都会基于一个基础镜像，这是你一来的基础镜像name:tag，git。

　　MAINTAINER （不用记，作者签名）。

　　ENV 配置环境变量。

　　COPY 复制本地。

　　EXPOSE 暴露容器的端口。

　　WORKDIR 后续命令的执行路径。

　　RUN important！，执行相应的命令，这一步是在容器构建这一步中执行的命令，一般用作安装软件，操作的结果是持久化在容器里面保存下来的。

　　Tips：每次执行RUN的时候都是再默认路径执行的，如果要到固定路径下执行命令请在之前加WORKDIR，或者使用RUN (cd workpath && echo "mike")这样把cd命令跟相应的执行命令用括号括起来。

　　ENTRYPOINT 容器启动后执行的命令。

**三、Jenkins+Docker自动化集成环境搭建**

　　这一步，我们把Jenkins和Docker结合起来。Jenkins本身版本比较老，虽然插件多，但许多插件没有人在维护更新，环境本身还是比较旧的，且各式各样的工程如果都用插件来安装，这样的插件不一定满足要求，所以建议配合Docker来搭建测试环境。

　　如果说好处的话，还有比较激进的一段话（建议略过哈）：

　　我们把一个工程代码开发后的测试流程简单分为编译、部署（事实上可能更多，如项目代码有多个工程，工程间有相互依赖）。如果我们想在一个slave节点上既运行Java7应用,又运行Java8应用，哦，Jenkins有相应的插件解决？那如果是NodeJs4应用和NodeJs5应用呢？JAVA升级到9了，插件没更新？像一些复杂的工程，如JDK自身的编译、如Docker项目的编译，我需要安装各种依赖，就可能会污染全局空间，这时候就建议把编译过程放到容器里，或者说我要在Linux上编译部署一个Windows应用，在一个Windows容器里部署就可以了。

　　再比如，如果我们用omad部署web应用可能要考虑端口冲突，部在Docker容器里，工程代码的端口，都不用变。还有，容器作测试环境的话，如果要保证线上，线下环境一致，甚至可以复制一个线下容器，直接在线上跑容器的。还有......

　　大致的想法是，我们用Jenkins来做节点控制、版本管理、流程设置、触发Job，用Docker来搭建编译部署环境。把这一切连接起来的就是流程脚本和Dockerfile的编写。

　　实现细节可以参照下面的示例。

**触发条件：**由Jenkins控制，如设置定时执行，或者github中的每一个commit，或者每一个PR执行一次。

**构建流程：**各个环节，Job，之间用Pipeline来实现整个构建流程的控制。

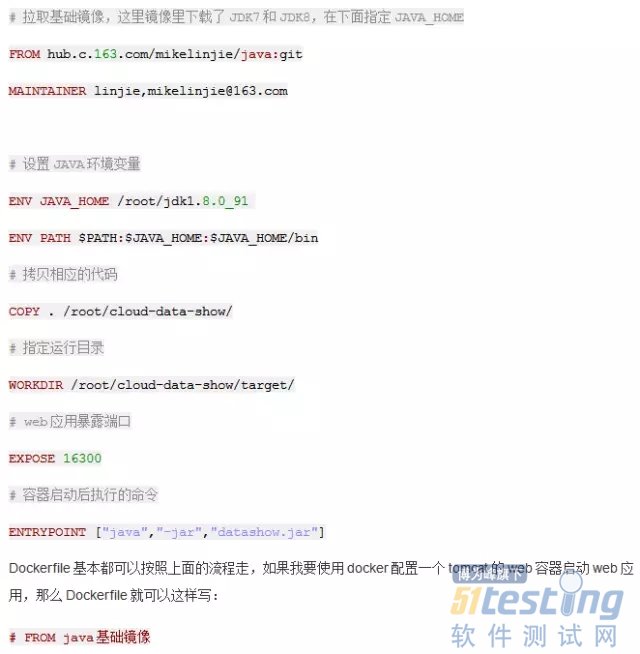
**构建Job：**这里，我觉得可以分为两种情况，一种是把拉取代码的过程和构建编译环境的过程都放到Dockfile里面，这适用于比较复杂的编译环境，如Docker本身的编译需要安装许多的依赖，对依赖的版本都有不同的要求，还是建议把编译环境放在容器的里面。另一种情况，为求省事方便，如java工程，编译一般用maven等构建工具来完成的话，大可以放在外面，只是把运行环境搬到Docker容器里。

**Execute shell：**

[](http://www.51testing.com/batch.download.php?aid=79459)

[](http://www.51testing.com/batch.download.php?aid=79460)

**Dockerfile：**

[](http://www.51testing.com/batch.download.php?aid=79461)

[](http://www.51testing.com/batch.download.php?aid=79462)

　　配置基础运行环境。

　　如果需要编译，配置编译环境，可以在容器内拉取代码，执行编译。

　　web应用暴露端口。

　　配置应用启动命令。

　　另外，容器里你都可以配置成8080端口，暴露8080端口，只要再Docker run -p \*:8080，运行时改下相应的端口映射就可以了。