# DevOps流程说明书

为了规范开发和运维的流程，使得开发的应用快速测试部署并上线使用，采用相关工具和技术搭建DevOps开发运维平台，目前其主体流程如下图1所示。

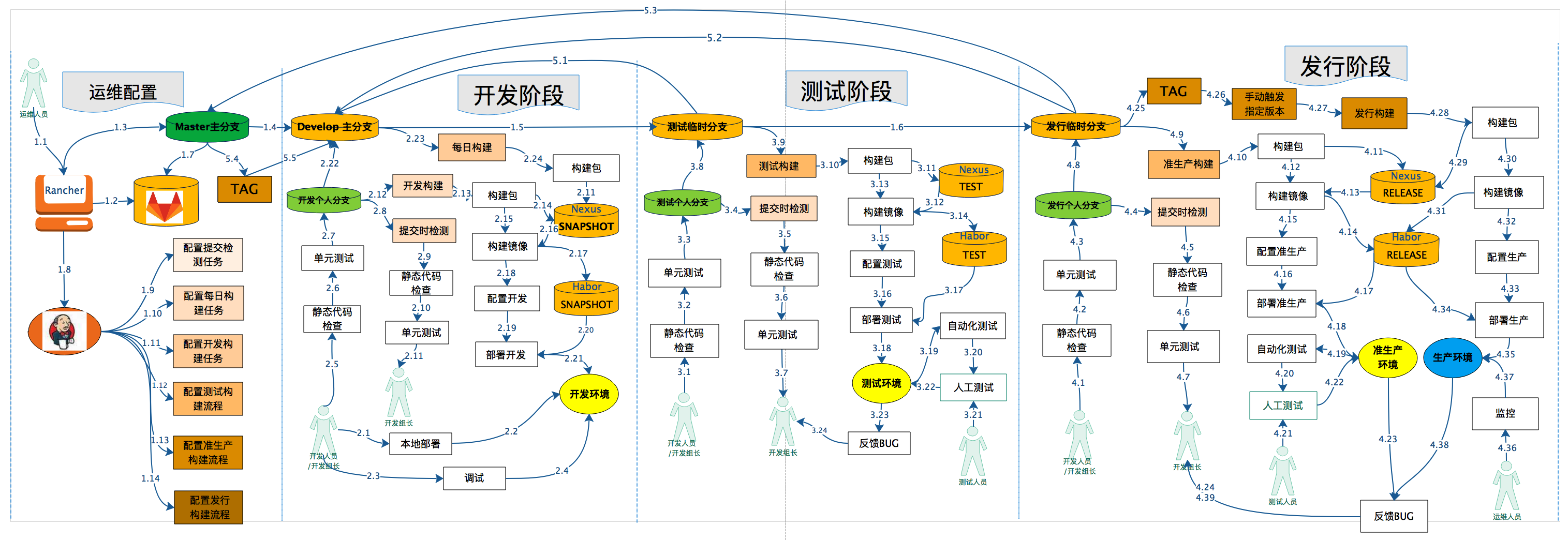


图 1 DevOps的流程

目前该流程把DevOps划分成四个阶段。分别为运维配置、开发阶段、测试阶段和发行阶段。

1. 运维配置：新开一个项目之后，运维人员需要通过相关工具创建Git库，然后在Git库中创建Master分支及其它分支,然后运维人员针对这些分支分别在Jenkins中制作相关作业（JOB），让这些作业自动地或手动地进行持续集成和持续交付。
2. 开发阶段：开发阶段主要工作是开发相关业务代码，对于DevOps而言则是对提交到Git的代码进行静态代码检测和单元测试，降低编译环节错误的几率。另外对开发主分支合并的代码进行每日构建，形成对应编译包，以便其它项目能引用或集成最新开发功能。
3. 测试阶段：某个版本的开发完成之后，提交到测试分支，测试分支会根据运维人员制定的作业自动构建包、镜像及自动地配置并部署到测试环境，先经过自动化测试再到人工测试，然后把相关Bug返回开发主管及开发人员，让他们在测试分支上进行修改，修改完成之后又自动发布到测试环境测试。
4. 发行阶段：通过了测试阶段的测试，测试分支会提交到发行分支。根据运维人员制定的作业自动构建包、镜像及自动地配置并部署到准生产环境。测试通过之后再由人工触发，根据相关作业自动构建、配置和部署到生产环境中，然后监控生产环境，出现问题反馈Bug并在发行分支进行修改，并在准生产环境中进行测试，然后部署到生产环境中。

在上面的四个阶段中主要分成如下几条流程：

1. 创建Git及分支：运维人员首先登录Rancher（1.1），然后在Rancher中创建Git库（1.2），接下来在Git库中仅包括初始化ReadME.md的Master主分支（1.3），根据该分支创建Develop主分支（1.4）、测试临时分支（1.5）、发行临时分支（1.6），然后为该项目主管创建Master用户，让其进行该项目的源代码及版本控制的主要管理。
2. 配置Jenkins及作业：运维人员配置或创建Jenkins软件（1.8），然后根据Git库的分支分别配置相关作业，包括提交检测任务作业（1.9）、每日构建任务作业（1.10）、开发环境构建任务作业（1.11）、测试构建任务作业（1.12）、准生产构建流程作业（1.13）、发行构建流程作业（1.14）。
3. 初始化Develop主分支：首先开发组长从中拉下运维人创建的Develop主分支。然后在此基础上搭建基础的代码结构，创建并上传到其个人开发分支，上传个人开发分支之前首先需要在本地进行静态代码检查（2.5）和单元测试（2.6）。当提交到个人分支时会触发已经配置好提交检测作业，在服务器上进行静态代码检查（2.9）和单元测试（2.10），通过之后开发组长合并自己分支代码到Develop主分支。
4. 开发人员提交代码：首先开发人员从中拉下开发组长上传到Develop主分支的代码。然后在此基础上进行开发，在本地部署到开发环境（2.1）或直接借助开发环境进行调试（2.3），进行静态代码检查（2.2）和单元测试（2.6）后上传到个人开发分支上。当提交到个人分支时会触发已经配置好提交检测作业，在服务器上进行静态代码检查（2.10）和单元测试（2.11）。
5. 开发组长合并代码：首先开发组长在IDE中拉下所有开发人员的代码和Develop主分支代码，在本机进行合并后，提交到个人分支进行检测，通过之后再次提交到Develop主分支中进行合并处理 (2.8)。
6. 每日构建：根据配置每日构建的脚本，系统每天定时自动从Develop主分支中拉下代码进行相关检测并构建包，然后以name-[date]-SNAPSHOT.jar的命令类似形式保存到Nexus的SNAPSHOT的库中（2.11），最后把其构建的信息发给开发组长（2.12）。
7. 开发构建：根据配置开发构建的脚本，系统会自动从Develop主分支中拉下代码进行相关检测并构建包(2.12)，然后以name-[version]-SNAPSHOT.jar的命名类似形式保存到Nexus的SNAPSHOT的库中（2.14），接下来构建Docker镜像（如果需要）（2.15）并把其镜像保存到私有Docker镜像库（2.17）。 接下来根据构建作业来自动配置测试环境参数（2.18）,然后从Docker镜像拉下镜像（2.20）,接下来把Docker镜像部署到开发环境（2.21）中。
8. 初始化每个版本测试分支：开发组长在本机上复制Develop主分支的代码，并进行代码检测（3.1）和单元测试（3.2），提交到其个人测试分支上，经过系统的静态代码检测和单元测试之后，在本地把该份代码提交到测试临时分支（3.8），如果测试临时分支存在上一版本的代码，先清除再进行全量合并。
9. 测试构建：根据配置测试构建的脚本，向测试临时分支中提交代码时，系统会自动从测试临时分支中拉下代码进行相关检测并构建包(3.10)，然后以name-[version]-test.jar的命名类似形式保存到Nexus的Test的库中（3.11），接下来构建Docker镜像（如果需要）（3.13）并把其镜像保存到私有Docker镜像库（3.14）。 接下来根据构建测试作业来自动配置测试环境参数（3.15）,然后从Docker镜像拉下镜像（3.16）,接下来把Docker镜像部署到测试环境（3.18）中。
10. 自动化测试和手工测试：当Docker镜像部署到测试环境之后，测试构建作业会调用自动化测试工具进行自动化测试（3.19），睡在动化测试完成人工测试介入（3.20，3.21），无论是自动化测试还是人工测试，出现了Bug都需要反馈到Bug系统中（3.23）并通知给开发组长进行相关修复（3.24）。
11. 修复Bug: 首先开发人员从测试临时分支中拉下代码。然后在此基础上进行bug修复，在本地进行静态代码检查（3.1）和单元测试（3.2）后上传到个人开发分支上。当提交到个人分支时会触发已经配置好提交检测作业，在服务器上进行静态代码检查（3.5）和单元测试（3.6）。
12. 初始化每个版本的发行分支：开发组长在本机上复制测试临时分支的代码，并进行代码检测（4.1）和单元测试（4.2），提交到其个人发行分支上，经过系统的静态代码检测（4.5）和单元测试（4.6）之后，在本地把该份代码提交到发行临时分支（4.8），如果发行临时分支存在上一版本的代码，先清除再进行全量合并。
13. 准生产环境alpha版本构建：根据配置准生产构建的脚本，向发行临时分支中提交代码时，系统会自动从发行临时分支中拉下代码进行相关检测并构建包(4.10)，然后以name-[version]-alpha[1…n].jar的命名类似形式保存到Nexus的RELEASE的库中（4.11），接下来构建Docker镜像（如果需要）（4.13）并把其镜像保存到私有Docker镜像库（4.14）。 接下来根据准生产测试作业来自动配置测试环境参数（4.15）,然后从Docker镜像拉下镜像（4.16）,接下来把Docker镜像部署到准生产环境（4.18）中，如果存在bug，参见第16步的流程修改bug。
14. 生产环境beta版本构建：根据配置生产构建的脚本，手动触发并指定Beta版本，系统会自动从发行临时分支中拉下代码进行相关检测并构建包(4.10)，然后以name-[version]-beta[1…n].jar的命名类似形式保存到Nexus的RELEASE的库中（4.11），接下来构建Docker镜像（如果需要）（4.13）并把其镜像保存到私有Docker镜像库（4.14）。 接下来根据生产测试作业来自动配置生产环境参数（4.15）,然后从Docker镜像拉下镜像（4.16）,接下来把Docker镜像部署到生产环境（4.18）中，如果存在bug，参见第17步的流程修改bug。
15. 生产环境GA版本构建：根据配置生产构建的脚本，手动触发并指定GA版本，系统会自动从发行临时分支中拉下代码进行相关检测并构建包(4.10)，然后以name-[version]-ga[1…n].jar的命名类似形式保存到Nexus的RELEASE的库中（4.11），接下来构建Docker镜像（如果需要）（4.13）并把其镜像保存到私有Docker镜像库（4.14）。接下来根据生产测试作业来自动配置生产环境参数（4.15）,然后从Docker镜像拉下镜像（4.16）,接下来把Docker镜像部署到生产环境（4.18）中，如果存在bug，参见第17步的流程修改bug。
16. 生产环境发行版本构建：根据配置生产构建的脚本，手动触发并指定正式版本，系统会自动从发行临时分支中拉下代码进行相关检测并构建包(4.28)，然后以name-[version].jar的命名类似形式保存到Nexus的RELEASE的库中（4.29），接下来构建Docker镜像（如果需要）（4.30）并把其镜像保存到私有Docker镜像库（4.31）。 接下来根据生产测试作业来自动配置生产环境参数（4.32）,然后从Docker镜像拉下镜像（4.34）,接下来把Docker镜像部署到生产环境（4.35）中。
17. 修复发行Bug: 首先开发人员从发行临时分支中拉下代码。然后在此基础上进行bug修复，在本地进行静态代码检查（4.1）和单元测试（4.2）后上传到个人发行分支上。当提交到个人分支时会触发已经配置好提交检测作业，在服务器上进行静态代码检查（4.5）和单元测试（4.6）。
18. 同步Develop、测试、发行代码。测试分支的修复的代码需要合并到Develop主分支中。开发组长根据其代码修复的频率选择合适时机合并，降低合并冲突（5.1）。发行分支的修改代码需要合并到Develop主分支中，开发组长根据其代码修复的频率选择合适时机合并，降低合并冲突（5.2）。当正式发行版发布之后，其发行分支代码，合并到Master主分支，采用全量覆盖的方式进行合并（5.3），合并完成之后打上标签（Tag），最后该Master合并到Develop主发行分支中。