CI改进之需求分析

|  |  |
| --- | --- |
| 修订记录 | |
| 2015-02-22 Ting YU | Init初稿 |
| 2015-09-26 Ting YU | 第二次修正 |

## 持续集成介绍

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，通常每天发生多次集成。每次集成都通过自动化的构建（包括编译、自动化测试、发布）来验证，从而尽快地发现错误。持续集成的目的，是让产品可以快速迭代，从而快速发现错误纠正错误，保持高质量的产品。

## CI待解决的问题

由于CI功能日益繁杂，CI逐渐臃肿不够灵活、缺少部分功能、易用性较差等问题，提出重新修整CI。从稳定性、功能性、易用性和可维护性这几个方面，来分析当前CI遇到的问题以及如何解决。

### 稳定性

稳定性是指CI能否稳定地输出高质量的测试报告。如果不能保证稳定性，例如：一些随机问题导致CI工程挂掉，或者用例弱一致性问题导致用例失败，像这些情况总是需要投入人力去解决。CI作为自动化构建平台，每天发生多次集成，频繁地自动化编译和测试，如果不能保证稳定性，将需要投入不必要的人力去解决问题。

|  |  |  |
| --- | --- | --- |
| 稳定性的问题 | 解决办法 | 优先级 |
| 分支机器卡导致用例失败 | 进一步分析机器卡的原因 |  |
| svn更新时停止构建导致下次svn更新失败 | 升级jenkins版本 |  |
| 网络异常导致构建失败 | 监控机器的内存、网络、CPU |  |
|  |  |  |

* 分支测试机器卡导致分支用例经常报-15、-104、-252等错误，进程卡顿导致coord节点连接不上或者主节点降备等问题。尤其在suse机器上出现的最为频繁。每天需要测试人员去定位是不是机器卡导致的问题。需要进一步分析机器/进程卡顿的原因。
* svn更新时停止构建会导致下次svn更新失败，需要CI负责人手动干预才能解决。这个在jenkins的论坛上确认是jenkins的bug，需要升级jenkins的版本。当前jenkins是1.502版本，有些插件只能在更高的版本上使用。
* 网络异常导致CI工程构建失败，从staf日志中可以推测很可能是断网，仍需要监控工具来确认是否是断网。

### 易用性

CI构建是为了及时发现产品中的问题，开发人员定位问题，最后修复问题。当CI用例失败后，不会保留环境，会继续执行下一个用例。所以CI应该提供详细的环境信息，以供开发人员定位问题。将这个点也纳入易用性方便。

另外，当前CI的每日构建还有很多使用不方便的地方，例如：有些测试报告还需要测试人员去控制台查看详细的打印信息。

|  |  |  |  |
| --- | --- | --- | --- |
| 易用性分类 | 问题 | 解决办法 | 优先级 |
| 便于定位问题 | 集群相关信息不足 |  | 1 |
| 部分用例打印信息呈现不足 |  | 1 |
| 工具日志没有归档 | 指定固定的执行路径，再到路径上收集日志 | 1 |
| 用例卡住时没有保留进程，难以定位 | 用例卡住后保留环境，不再往下执行测试 | 1 |
| 感知不到节点异常重启 |  | 1 |
| 机器之间时间不同步 |  | 1 |
| 轻便性 | 编译速度45分钟 | 全量编译改为增量编译 | 提交构建 |
| 全量测试速度5-8个小时 | 不分发行版  分摊到不同的机器上执行  驱动用例并发执行 | 1 |
| 不能支持只编译引擎等 | 支持分开编译 | 提交构建 |
| 优化每日构建 | 日志和编译产物的归档路径不够清晰 |  | 1 |
| java测试用例报告层次太多导致报告展示不清晰 | 修改报告 | 1 |
| 部分报告界面上不能下载日志 |  | 1 |
| rpm包打包还未写入调度脚本 |  | 1 |
| 跑测试用例前还必须重新安装部署 |  | 1 |
| 新增js用例需要配置文件 |  | 1 |
| 所有机器并发执行用例 |  | 1 |
| Rpm、run打包拷贝文件 |  | 1 |
| Core文件名加入时间信息 |  | 1 |
| 指定执行某个测试用例、执行次数 |  |  |
| …… |  |  |
|  |  |  |
|  |  |  |

### 功能性

需要新增以下功能：

|  |  |  |
| --- | --- | --- |
| 新增功能 | 描述 | 优先级 |
| 提交构建 | 提交代码后执行冒烟测试，发送邮件给相关人员 |  |
| 代码覆盖率 | 提供代码覆盖率指导测试人员补充测试 |  |
| 数据量大的测试 | 大数据量的测试用例执行时间长，目前没有上CI |  |
| c#驱动编译 | C#驱动编译目前都是人工编译，再上传到svn。需新增windows编译机 |  |
| 测试修改后pg+fdw | 目前只测试了原生的pg+fdw |  |
| 可靠性场景测试 | 增删节点/组等操作需要一套单独环境测试 |  |
| 稳定性场景测试 |  |  |
| 节点配置相关的测试 | 需配置节点的测试，需要一套单独环境测试 |  |
| windows上的编译 | 目前都是人工在windows上编译 |  |
| 指定在任意机器上跑测试 | 方便测试人员调试新增的测试用例 |  |
| Pg打run包 |  |  |

### 可维护性

指调度脚本的可维护性。

* ant脚本之间的调度过于复杂，目前一个ant脚本往往需要一层一层调其他ant脚本，达到5、6个脚本之间的相互调度。有些脚本需要传入8、9个参数。 不便于调试、功能修改和模块增加。
* 使用多套调度，目前有测试的调度脚本、编译分支的脚本、编译主干企业版的脚本、编译主干社区版的脚本、编译pg、测试pg的脚本，并且分别在不同的分支上，还有大量的重复代码，增加维护的难度，应该将他们合成一套调度。

|  |  |  |
| --- | --- | --- |
| 可维护性的问题 | 解决 | 优先级 |
| 调度脚本难以调试和新增模块 | 减少输入参数  减少嵌套层数 | 1 |
| 调度有重复代码 | 统一分支和主干  统一pg编译测试 | 1 |
|  |  |  |

## 计划

以上任务主要归纳为：对现有功能（每日构建）的优化，以及新增功能，包括提交构建、代码覆盖率等。由于现有的ant调度脚本写得比较乱，需要先重新调整结构，以便于后期灵活地新增功能。先优化每日构建，在每日构建优化中，提升易用性和可维护性。

|  |  |  |
| --- | --- | --- |
| 任务列表 | 计划花费时间 | 备注 |
| 优化每日构建 | 1个月 |  |
| 数据量大的测试 |  | 合并入每日构建 |
| 可靠性场景测试 | 1个礼拜 |  |
| 节点配置相关的测试 | 1个礼拜 |  |
| 代码覆盖率 | 2个礼拜 |  |
| 新增提交构建 | 2个礼拜 |  |
| 测试修改后的pg+fdw | 1个礼拜 |  |
| 稳定性测试 |  | 暂时没有用例，先不上CI |
| C#驱动编译 |  |  |
| windows上编译 |  |  |
| Pg打run包 |  |  |