加餐 | 怎么保障发布的效率与质量?

2019-10-22 许式伟

讲述：姚迪迈
时长 12:32 大小 11.49M

你好，我是七牛云许式伟。

为什么要有发布流程?

在 “49 | 发布、升级与版本管理” 一讲中我们提到过：

> 变更是故障之源。

这种由于业务需要而主动发起的软硬件升级与各类配置变更，我们可以统一称之为发布。例如：

更换交换机的类型，或升级版本。

更换所依赖的基础软件，或升级版本。基础软件包括操作系统、负载均衡、数据库等等。

升级业务软件本身。

调整软硬件环境的配置项。

特殊地，如果集群的服务对扩容缩容有很好的自动化支持，可以非常便捷地增加或减少服务器，那么这种情况虽然发生了集群的变化，我们可以不把它看作变更，不走发布相关的流程。尤其在硬件已经完全池化时，增加、减少服务器可能是个非常标准化且低成本的操作。

我们通常说的 “版本发布”，往往侧重点是在升级业务软件的版本，这是发布中最常发生的情况，当然也是我们最为关注的。

传统的软件公司的发布周期往往很长，有几个月甚至有的是按年来计算。而互联网公司的发布周期则非常不同。之所以快速发布、快速迭代变得简单的原因是，它们仅仅需要在服务器端发布，而不需要发布到每个使用者的电脑上。

一个每三年发布一次新产品的公司不需要详细的发布流程。因为发布的频率太低了，发布流程的优化能够带来的收益太小。

但是如果我们每天都在发布，甚至每天发布很多次，那么如此高频的发布速度，就要求我们创建和维护一个效率与质量都能够兼顾的精简的发布流程。

一个精简的发布流程，通常需要有发布平台这样的基础设施，把发布过程中反复遇到的问题对应的解决方案固化到系统中。

但是系统并不能解决所有的发布问题。变更终究是存在未知的新东西，需要人工进行检查判断。为此，SRE 部门往往还建立了一个专门的团队负责发布，即发布协调小组。团队成员称为 “发布协调工程师（Launch Coordination Engineering，LCE）”。

发布协调小组会针对每个业务，维护一个该业务的 “发布检查列表”，包括针对每次发布需要检查的常见问题，以及避免常见问题发生的手段。只有在发布检查表中的检查点都得到了确认，才会给版本发布放行。

这个列表在实践中被证实，它是保障发布可靠性的重要工具。

建立在系统之上的灰度发布

除了 “发布检查列表”，我们还有一个至关重要的保障发布质量的做法：灰度发布。

不管你如何小心，发布检查做得多全面，仍然只是在尽可能减少发布的风险，而不是消除。任何改动都具有一定的危险性，而任何危险性都应该被最小化，这样才能保障系统的可靠性。

在小型的测试环境上测试成功的变更，不见得在生产环境就没有问题，更何况从 SRE 的角度，测试的覆盖率也是不能假设的。

任何发布都应该灰度进行，并且在整个过程中还需要穿插必要的校验步骤。刚开始，新的服务可能会在某个数据中心的一台或几台机器上安装，并且被严密监控一段时间。如果没有发现异常，新版本会在更多台机器上安装并再次监控，直至最后完成整个发布过程。

发布的第一阶段通常被称为 “金丝雀”。这和煤矿工人带金丝雀下矿井检测有毒气体类似，通过使用这些“金丝雀” 服务线上流量，我们可以观察任何异常现象的发生。

“金丝雀” 测试适用于正常的软件版本发布，也适用于配置项的变更。负责配置变更的工具通常都会对新启动的程序监控一段时间，保证服务没有崩溃或者返回异常。如果在校验期间出现问题，系统会自动回退。

灰度式发布的理念甚至并不局限于软件和服务的发布。例如，我们商业上的高成本的运营活动，往往会先选择一到两个地区先做实验，然后再把成功经验复制到全国各地。

所以灰度发布思想的一个自然延伸是做功能开关，也就是大家熟悉的 AB 测试。很多东西在测试环境中无法模拟时，或者在真实环境中仍然存在不可预知的情况时，灰度机制就非常有用了。

不是所有的改动都可以一样对待。有时我们仅仅是想检查某个界面上的改动是否能提升用户感受。这样的小改动不需要几千行的程序或者非常重量级的发布流程。我们可能希望同时测试很多这方面的改动。

有时候我们只是想要知道是否有足够多的用户会喜欢使用某个新功能，就通过发布一个简单的原型给他们测试。这样我们就不用花费数个月的时间来优化一个没人想要使用的功能。

通常来说，这类 AB 测试框架需要满足以下几个要求：

可以同时发布多个变更，每个变更仅针对一部分服务器或用户起作用。

变更可以灰度发布给一定数量的服务器或用户，比如 1%。

在严重 Bug 发生，或者有其他负面影响时，可以迅速单独屏蔽某个变更。

用数据来度量每个变更对用户体验的提升。

LCE 的职责

LCE 团队负责管理发布流程，以确保整个发布过程做到又快又好。LCE 有如下这些职责：

审核新产品及相关的内部服务，确保它们的可靠性标准达到要求。如果不达预期，提供一些具体的建议来提升可靠性。

在发布过程中作为多个团队之间的联系纽带。

负责跟进发布系统相关的所有技术问题。

作为整个发布过程中的一个守门人，决定某次发布是否是 “安全的” 。

整体来说，LCE 的要求其实是相当高的。LCE 的技术要求与其他的 SRE 成员一样，但这个岗位打交道的外部团队很多，需要有很强的沟通和领导能力。他需要将分散的团队聚合在一起达成一个共同目标，同时还需要偶尔处理冲突问题，还要能够为软件开发工程师提供建议和指导。

发布检查列表

我们前面已经提过，发布检查列表可以用来保障发布质量，它是可靠发布产品与服务的重要组成部分。一个完备的检查列表通常包含以下这些方面的内容。

其一，架构与依赖相关。针对系统架构的评审可以确定该服务是否正确使用了某类基础设施，并且确保这些基础设施的负责人加入到发布流程中来。为什么要引入基础设施的负责人，是因为需要确认相关依赖的服务都有足够的容量。

一些典型的问题有：

从用户到前端再到后端，请求流的顺序是什么样的？

是否已经将非用户请求与用户请求进行隔离？

预计的请求数量是多少？单个页面请求可能会造成后端多个请求。

其二，集成和公司最佳实践相关。很多公司的对外服务都要运行在一个内部生态系统中，这些系统为如何建立新服务器、配置新服务、设置监控、与负载均衡集成，以及设置 DNS 配置等提供了指导。

其三，容量规划相关。新功能通常会在发布之初带来临时的用量增长，在几天后会趋于平稳。这种尖峰式的负载或流量分布可能与稳定状态下有显著区别，之前内部的压力测试可能失效。

公众的兴趣是很难预测的，有时甚至需要为预计容量提供 15 倍以上的发布容量。这种情况下灰度发布会有助于建立大规模发布时的数据依据与信心。

一些典型的问题有：

本次发布是否与新闻发布会、广告、博客文章或者其他类型的推广活动有关？

发布过程中以及发布之后预计的流量和增速是多少？

是否已经获取到该服务需要的全部计算资源？

其四，故障模式相关。针对服务进行系统性的故障模式分析可以确保发布时服务的可靠性。

在检查列表的这一部分中，我们可以检查每个组件以及每个组件的依赖组件来确定当它们发生故障时的影响范围。

一些典型的问题有：

该服务是否能够承受单独物理机故障？单数据中心故障？网络故障？

如何应对无效或者恶意输入，是否有针对拒绝服务攻击（DoS）的保护？

是否已经支持过载保护？

如果某个依赖组件发生故障，该服务是否能够在降级模式下继续工作？

该服务在启动时能否应对某个依赖组件不可用的情况？在运行时能否处理依赖不可用和自动恢复情况？

其五，客户端行为相关。最常见的客户端滥发请求的行为，是配置更新间隔的设置问题。比如，一个每 60s 同步一次的新客户端，会比 600s 同步一次的旧客户端造成 10 倍的负载。

重试逻辑也有一些常见问题会影响到用户触发的行为，或者客户端自动触发的行为。假设我们有一个处于过载状态的服务，该服务由于过载，某些请求会处理失败。如果客户端重试这些失败请求，会对已经过载的服务造成更大负载，于是会造成更多的重试，更多的负载。客户端这时应该降低重试的频率，一般需要增加指数型增长的重试延迟，同时仔细考虑哪些错误值得重试。例如，网络错误通常值得重试，但是 4xx 错误（这一般意味着客户端侧请求有问题）一般不应该重试。

自动请求的同步性往往还会造成惊群效应。例如，某个手机 APP 开发者可能认为夜里 2 点是下载更新的好时候，因为用户这时可能在睡觉，不会被下载影响。然而，这样的设计会造成夜里 2 点时有大量请求发往下载服务器，每天晚上都是如此，而其他时间没有任何请求。这种情况下，每个客户端应该引入一定随机性。

其他的一些周期性过程中也需要引入随机性。回到之前说的那个重试场景下：某个客户端发送了一个请求，当遇到故障时，1s 之后重试，接下来是 2s、4s 等。没有随机性的话，短暂的请求峰值可能会造成错误比例升高，这个周期会一直循环。为了避免这种同步性，每个延迟都需要一定的抖动，也就是加入一定的随机性。

一些典型的问题有：

客户端在请求失败之后，是否按指数型增加重试延时？

是否在自动请求中实现随机延时抖动？

其六，流程与自动化相关。虽然我们鼓励自动化，但是对于发布这件事情来说，完全自动化是灾难性的。为了保障可靠性，我们应该尽量减少发布流程中的单点故障源，包括人在内。

这些流程应该在发布之前文档化，确保在工程师还记得各种细节的时候就完全转移到文档中，这样才能在紧急情况下派上用场。流程文档应该做到能使任何一个团队成员都可以在紧急事故中处理问题。

一些典型的问题有：

- 是否已将所有需要手动执行的流程文档化？
- 是否已将构建和发布新版本的流程自动化？

其七，外部依赖相关。有时候某个发布过程依赖于某个不受公司控制的因素。尽早确认这些因素的存在可以使我们为它们的不确定性做好准备。

例如，服务依赖于第三方维护的一个类库，或者另外一个公司提供的服务或者数据。当第三方提供商出现故障、Bug、系统性的错误、安全问题，或者未预料到的扩展性问题时，尽早计划可以使我们有办法避免影响到直接用户。

一些典型的问题有：

- 这次发布依赖哪些第三方代码、数据、服务，或者事件？
- 是否有任何合作伙伴依赖于你的服务？发布时是否需要通知他们？
- 当我们或者第三方提供商无法在指定截止日期前完成工作时，会发生什么？

结语

今天我们探讨 “发布与升级” 的实践，如何既保证质量，又能够兼顾效率。正确的做法当然不是为了快而去忽略流程，而是在不断的发布经历中总结经验教训，把每个环节干得更快更有效率。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们聊聊 “故障域与故障预案”。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇　50 | 日志、监控与报警

下一篇　51 | 故障域与故障预案

精选留言 (5)

写留言

Aaron Cheung
2019-10-22

流程文档化 推动有难度 每个人都习惯别人提供文档 让自己写文档就懒得写😂

作者回复: 是的

2

Charles
2019-10-22

涨了很多平常小项目不怎么能接触到和用到的知识！但是架构师能力和汽车动力一样，可以不用，但是一定要有😄，谢谢老师加餐

展开


1

Aaron Cheung

2019-10-22

SRE 部门往往还建立了一个专门的团队负责发布，即发布协调小组。团队成员称为 “发布协调工程师（Launch Coordination Engineering，LCE）”。 这个是第一次听到 学习了


1

郝志强

2019-10-25

“发布检查列表可以用来保障发布质量，它是可靠发布产品与服务的重要组成部分“，这种列清单的方法及意义与《清单革命》一书所倡导的一致。检查工作流程化而不是靠经验。

作者回复: 能够自动化的事情自动化，不能自动化的事情用检查列表。

觉

2019-10-23

许老师，后面会不会说一下应该如何做好系统架构、业务架构、技术架构相关的知识点呢？还有架构重构相关的知识，如果不知道什么是好的架构，又怎么能找到重构的突破点呢？

作者回复: 下一章会谈