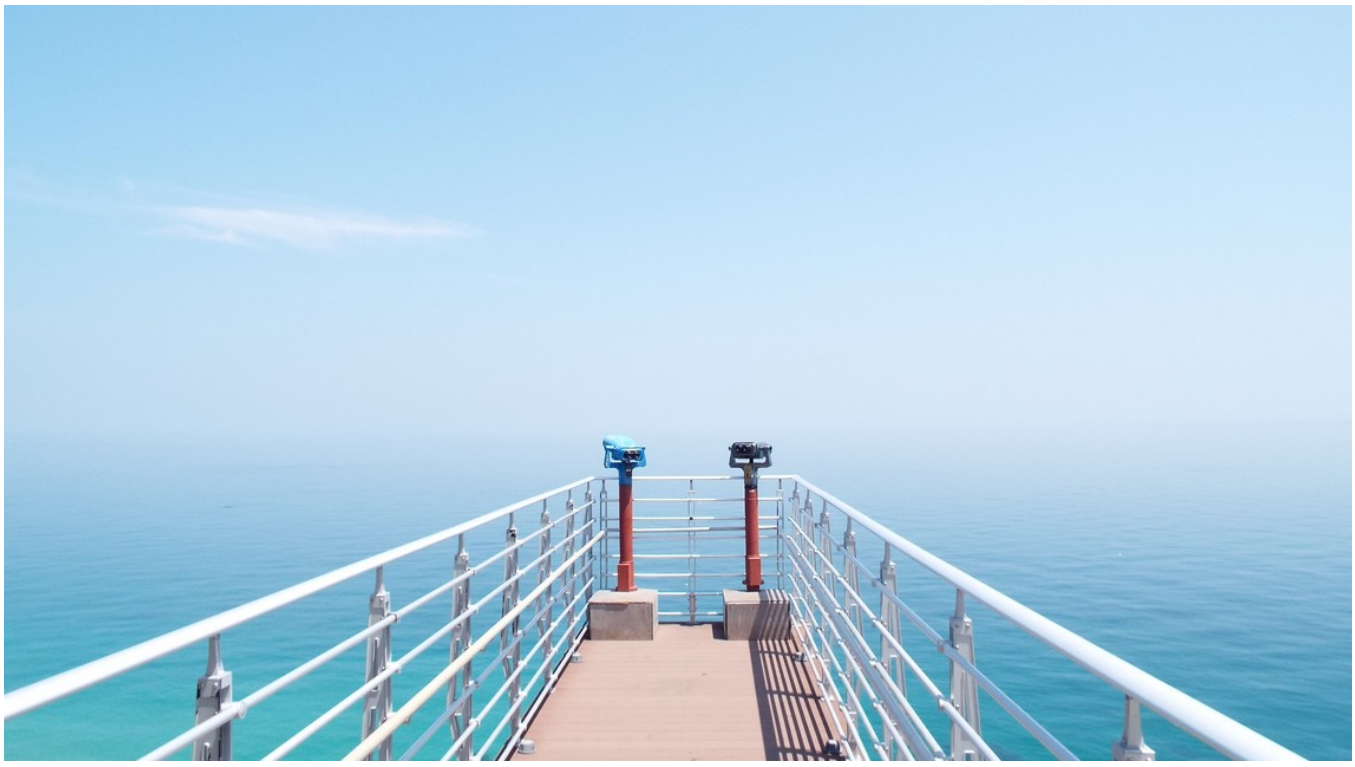
19 | 发布是持续交付的最后一公里

2018-08-16 王潇俊



19 | 发布是持续交付的最后一公里
朗读人：王潇俊　12′58″ | 5.95M

你好，我是王潇俊。我今天分享的主题是：发布是持续交付的最后一公里。

在开始我今天的分享之前，我们先来搞清楚一个问题：部署和发布是不是一回事儿？

有一些观点认为，部署和发布是有区别的，前者是一个技术范畴，而后者则是一种业务决策。这样的理解应该说是正确的。应用被部署，并不代表就是发布了，比如旁路运行（dark launch）方式，对于客户端产品更是如此的。

但对互联网端的产品来说，这个概念就比较模糊了，所以从英文上来看，我们通常既不用 deploy 这个词，也不用 release 这个词，而是使用 rollout 这个词。所以，从用词的选择上，我们就可以知道，发布是一个慢慢滚动向前、逐步生效的过程。

因此，我在《发布及监控》系列文章中提到的“发布”，均泛指 rollout 这样的过程。

发布，头疼的最后一步

无论是为新需求添加的代码，还是静态配置的变更，线上应用的任何变动都要经过发布这道工序才能最终落地，完成交付。通常，发布意味着应用重启、服务中断，这显然不符合如今系统高可用的需求。

同时，软件工程和经验也告诉我们，世界上不存在没有 Bug 的代码，即便经过详尽细致地测试，线下也很难百分之一百地复制线上的环境、依赖、流量，更难穷举千变万化的用户行为组合。

于是，发布变更，在许多时候是一件被标记为“高风险系数”的工作，工程师和测试人员经常在深夜搞得筋疲力尽，甚至焦头烂额。

进入持续交付的时代后，这个痛点只会更加突显，因为持续交付意味着持续发布。例如，在测试环境小时级的持续集成场景中，如果没有办法将发布过程流程化、自动化，显然会频繁打断最终的交付过程，大幅降低开发测试效率。

好在上帝创造了一个问题，一定会留下一套解决方案，更多的时候是许多套，我们的目标就是找到它，然后实现最佳实践。

发布的需求

你不妨先问自己一个问题，作为开发人员，或者其他研发角色，你理想中的发布是什么样的呢？

答案当然是：够快够傻瓜。最好点一下鼠标，就立刻能看到线上的变更，整个体验跟本地开发环境调试毫无区别。

更加贪心的同学甚至希望连点击鼠标都不用，而是每小时、每天、甚至每 commit 自动发布，希望系统神奇地将自己从 SSH 和乱七八糟的线上环境中解放出来。

另一方面，从运维的角度来讲，线上系统的稳定性和可用性则是第一考量。运维上线变更前，首先会思考如果这中间出了什么岔子该如何应对，找不到问题时能否快速回滚到之前的状态，整个过程如何最小限度地减少服务的宕机时间。对他们而言，完美的方案就像是能够稳如泰山地给飞行中的飞机更换引擎。

因此，我们想要的应该是：一个易用、快速、稳定、容错力强，必要时有能力迅速回滚的发布系统。

什么是好的发布流程？

好的系统依赖好的设计，而好的工作流方案可以显著减少需要考虑的问题集，有助于创造出高健壮性的系统。对于发布系统，单机部署方案和集群工作流同样重要。

第一，把大象放进冰箱分几步？

单机部署这件事说复杂很复杂，说简单也很简单。

复杂在于，不同技术栈的部署方式千差万别，脚本语言 PHP 和需要编译的 Golang 的上线步骤差很多；同样是 Java，使用 Tomcat 和 Netty 的命令也完全不一样。

简单在于，发布过程高度抽象后其实就三个步骤：

1. 在目标机器上执行命令停掉运行中的服务；
2. 把提前准备好的变更产物传上机器覆盖原来的目录；
3. 运行命令把服务再跑起来。

但只是按照这三步走，你很容易就能设想到一些反例场景：服务虽然停止，但新的请求还在进入，这些请求全部返回 503 错误；或者，假如有 Bug 或者预料之外的问题，服务根本起不来，停服时间就不可预知了。

更糟糕的是，假如此时情况紧急，我们想回滚到之前的状态，回滚时就会发现，由于之前的目录被覆盖了，基本回不去了。

第二，靠谱的单机部署

那么，比较完善的发布变更流程应该是怎样的呢？在我看来，可以抽象成五步：

1. 下载新的版本，不执行覆盖；
2. 通知上游调用方，自己现在为暂停服务状态；
3. 运行命令 load 变更重启服务；
4. 验证服务的健康状况；
5. 通知上游调用方，自己服务恢复正常。

假设我们实现了一个程序，简单地顺序执行上面的算法，让我们一起来检验一下这套程序是否能满足发布的需求：快速、易用、稳定、容错、回滚顺滑。

- 易用：执行脚本就好，填入参数，一键执行。
- 快速：自动化肯定比手工快，并且有提升空间。比如，因为有版本的概念，我们可以跳过相同版本的部署，或是某些步骤。
- 稳定：因为这个程序逻辑比较简单，而且执行步骤并不多，没有交叉和并行，所以稳定性也没什么大的挑战。
- 容错性强：表现一般，脚本碰到异常状况只能停下来，但因为版本间是隔离的，不至于弄坏老的服务，通过人工介入仍能恢复。

- 回滚顺滑：因为每个版本都是完整的可执行产物，所以回滚可以视作使用旧版本重新做一次发布。甚至我们可以在目标机器上缓存旧版本产物，实现超快速回滚。

通过这个程序的简单执行过程，我们可以看到这套流程的简单实现，基本满足了我们对发布的需求。而且，可以通过添加更复杂的控制流，获得更大的提升空间。

我在这里提到的三个重要概念：版本、通知调用方、验证健康（又被称之为点火），可以说是实现目标的基石。我会在后续章节，详细介绍版本、通知调用方、验证健康这三方面的实现方式和取舍。

第三，扩展到集群

如今应用架构基本告别了单点世界，面向集群的发布带来了更高维度的问题。当发布的目标是一组机器而不是一台机器时，主要问题就变成了如何协调整个过程。

比如，追踪、同步一组机器目前发布进行到了哪一步，编排集群的发布命令就成为了更核心功能。好消息是，集群提供了新的、更易行的方法提高系统的发布时稳定性，其中最有用的一项被称为灰度发布。

灰度发布是指，渐进式地更新每台机器运行的版本，一段时期内集群内运行着多个不同的版本，同一个 API 在不同机器上返回的结果很可能不同。 虽然灰度发布涉及到复杂的异步控制流，但这种模式相比简单粗暴的“一波流”显然要安全得多。

不仅如此，当对灰度发布的进度有很高的控制能力时，事实上这种方式可以提供 A/B 测试可能性。 比如，你可以说，将 100 台机器分成 4 批，每天 25 台发布至新的版本，并逐步观察新版本的效果。

其实，集群层面的设计，某种程度上是对单机部署理念的重复，只不过是在更高的维度上又实现了一遍。 例如，单机部署里重启服务线程堆逐批停止实现，与集群层面的分批发布理念，有异曲同工之妙。

几种常见的灰度方式

灰度发布中最头疼的是如何保持服务的向后兼容性，发现苗头不对后如何快速切回老的服务。这在微服务场景中，大量服务相互依赖，A 回滚需要 B 也回滚，或是 A 的新 API 测试需要 B 的新 API 时十分头疼。为了解决这些问题，业界基于不同的服务治理状况，提出了不同的灰度理念。

接下来，我将分别介绍蓝绿发布、滚动发布和金丝雀发布，以及携程在发布系统上的实践。

1. 蓝绿发布，是先增加一套新的集群，发布新版本到这批新机器，并进行验证，新版本服务器并不接入外部流量。此时旧版本集群保持原有状态，发布和验证过程中老版本所在的服务器

仍照常服务。验证通过后，流控处理把流量引入新服务器，待全部流量切换完成，等待一段时间没有异常的话，老版本服务器下线。

- 这种发布方法需要额外的服务器集群支持，对于负载高的核心应用机器需求可观，实现难度巨大且成本较高。
- 蓝绿发布的好处是所有服务都使用这种方式时，实际上创造了蓝绿两套环境，隔离性最好、最可控，回滚切换几乎没有成本。

2. 滚动发布，是不添加新机器，从同样的集群服务器中挑选一批，停止上面的服务，并更新为新版本，进行验证，验证完毕后接入流量。重复此步骤，一批一批地更新集群内的所有机器，直到遍历完所有机器。
   这种滚动更新的方法比蓝绿发布节省资源，但发布过程中同时会有两个版本对外提供服务，无论是对自身或是调用者都有较高的兼容性要求，需要团队间的合作妥协。但这类问题相对容易解决，实际中往往会通过功能开关等方式来解决。

3. 金丝雀发布，从集群中挑选特定服务器或一小批符合要求的特征用户，对其进行版本更新及验证，随后逐步更新剩余服务器。这种方式，比较符合携程对灰度发布的预期，但可能需要精细的流控和数据的支持，同样有版本兼容的需求。

结合实际情况，携程最终选择的方式是：综合使用滚动发布和金丝雀发布。 首先允许对一个较大的应用集群，特别是跨 IDC 的应用集群，按自定义规则进行切分，形成较固定的发布单元。基于这种设计，我们开发了携程开源灰度发布系统，并命名为 Tars 。其开源地址为：https://github.com/ctripcorp/tars

关于携程灰度发布的设计和实施，以及如何把灰度发布的理念贯穿到你的持续交付体系中，我会在后面的第 22 篇文章《发布系统架构功能设计实例》中详细介绍。

其他考量

处于持续交付最后一环的发布，实际上是非常个性化的，与实际实现相关，甚至是 case by case 的。因为每个上游系统的少许变更和设计瑕疵，层层下压最终都会影响到发布系统。这不但要求发布系统了解链条上绝大多数环节，知道发生了什么以便 debug，甚至时常还需要为其“兜底”。

除此以外，软件工程中没有“银弹”，适用于所有场景的系统设计是不存在的。上面的设计有许多值得探讨的地方，比如发布时到底是使用增量、还是全量，单机切断流量使用哪种手段，集群发布的控制流设计，都是值得探讨的主题。这些内容，我将会在后面的文章中详细展开。

总结

作为《发布与监控》系列文章的开篇，我介绍了发布在持续交付中的位置和需求，并提出了一个可靠的单机部署流程的概念，即我们想要的应该是：一个易用、快速、稳定、容错力强，必要时

有能力迅速回滚的发布系统。

明确了发布的需求后，我推演了集群发布中灰度发布的概念和常用方式，包括蓝绿发布、滚动发布和金丝雀发布，并分析了这三种发布方式，给出了携程选择的方案，希望可以帮你选择适合自己团队的发布策略。

思考题

你能详细地整理和描述出你的应用的单机部署过程吗？

欢迎你给我留言。

通过留言可与作者互动