

管理设计篇之"部署升级策略"

2018-05-08 陈皓



管理设计篇之"部署升级策略"

朗读人：柴巍 09'10" | 4.20M

在分布式系统的世界里，一个服务有多个实例，所以部署或是升级一个服务也会变得比较麻烦一些。这里，我们讨论一些服务部署的模式。一般来说，我们有下面的一些服务部署模式。

- 停机部署（Big Bang / Recreate）：把现有版本的服务停机，然后部署新的版本。
- 蓝绿部署（Blue/Green /Stage）：部署好新版本后，把流量从老服务那边切过来。
- 滚动部署（Rolling Update / Ramped）：一点一点地升级现有的服务。
- 灰度部署（Canary）：把一部分用户切到新版上来，然后看一下有没有问题。如果没有问题就继续扩大升级，直到全部升级完成。
- AB 测试（A/B Testing）：同时上线两个版本，然后做相关的比较。

下面，我们来看一下每种方式的使用场景和优缺点。

停机部署

停机部署其实是最简单粗暴的方式，就是简单地把现有版本的服务停机，然后部署新的版本。在一些时候，我们必需使用这样的方式来部署或升级多个服务。比如，新版本中的服务使用到了和老版本完全不兼容的数据表的设计。这个时候，我们对生产有两个变更，一个是数据库，另一个是服务，而且新老版本互不兼容，所以只能使用停机部署的方式。

这种方式的优势是，在部署过程中不会出现新老版本同时在线的情况，所有状态完全一致。停机部署主要是为了新版本的一致性问题。

这种方式最不好的问题就是会停机，对用户的影响会很大。所以，一般来说，这种部署方式需要事前挂公告，选择一个用户访问少的时间段来做。

蓝绿部署

蓝绿部署与停机部署最大的不同是，其在生产线上部署相同数量的新的服务，然后当新的服务测试确认 OK 后，把流量切到新的服务这边来。蓝绿部署比停机部署好的地方是，它无需停机。

我们可以看到这种部署方式，就是我们说的预发环境。在我以前的金融公司里，也经常用这种方式，生产线上有两套相同的集群，一套是 Prod 是真实服务的，另一套是 Stage 是预发环境，发布发 Stage，然后把流量切到 Stage 这边，于是 Stage 就成了 Prod，而之前的 Prod 则成了 Stage。有点像换页似的。

这种方式的优点是没有停机，实时发布和升级，也避免有新旧版本同时在线的问题。但这种部署的问题就是有点浪费，因为需要使用双倍的资源（不过，这只是在物理机时代，在云计算时代没事，因为虚拟机部署完就可以释放了）。

另外，如果我们的服务中有状态，比如一些缓存什么的，停机部署和蓝绿部署都会有问题。

滚动部署

滚动部署策略是指通过逐个替换应用的所有实例，来缓慢发布应用的一个新版本。通常过程如下：在负载调度后有个版本 A 的应用实例池，一个版本 B 的实例部署成功，可以响应请求时，该实例被加入到池中。然后，版本 A 的一个实例从池中删除并下线。

这种部署方式直接对现有的服务进行升级，虽然便于操作，而且在缓慢地更新的过程中，对于有状态的服务也是比较友好的，状态可以在更新中慢慢重建起来。但是，这种部署的问题也是比较多的。

- 在发布过程中，会出现新老两个版本同时在线的情况，同一用户的请求可能在新老版中切换而导致问题。
- 我们的新版程序没有在生产线上经过验证就上线了。
- 在整个过程中，生产环境处于一个新老更替的中间状态，如果有问题要回滚就有点麻烦了。

- 如果在升级过程中，需要做别的一些运维工作，我们还要判断哪些结点是老版本的，哪些结点是新版本的。这太痛苦了。
- 因为新老版本的代码同时在线，所以其依赖的服务需要同时处理两个版本的请求，这可能会带来兼容性问题。
- 而且，我们无法让流量在新老版本中切换。

灰度部署（金丝雀）

灰度部署又叫金丝雀部署。其得名来源于矿井中的金丝雀 --17 世纪，英国矿井工人发现，金丝雀对瓦斯这种气体十分敏感。空气中哪怕有极其微量的瓦斯，金丝雀也会停止歌唱。而当瓦斯含量超过一定限度时，虽然鲁钝的人类毫无察觉，金丝雀却早已毒发身亡。当时在采矿设备相对简陋的条件下，工人们每次下井都会带上一只金丝雀作为 " 瓦斯检测指标 "，以便在危险状况下紧急撤离。

灰度部署是指逐渐将生产环境流量从老版本切换到新版本。通常流量是按比例分配的。例如 90% 的请求流向老版本，10% 的流向新版本。然后没有发现问题，就逐步扩大新版本上的流量，减少老版本上的流量。

除了切流量外，对于多租户的平台，例如云计算平台，灰度部署也可以将一些新的版本先部署到一些用户上，如果没有问题，扩大部署，直到全部用户。一般的策略是，从内部用户开始，然后是一般用户，最后是大客户。

这个技术大多数用于缺少足够测试，或者缺少可靠测试，或者对新版本的稳定性缺乏信心的情况下。

把一部分用户切到新版上来，然后看一下有没有问题。如果没有问题就继续扩大升级，直到全部升级完成。

AB 测试

AB 测试和蓝绿部署或是金丝雀灰度部署完全是不一样的。

AB 测试是同时上线两个版本，然后做相关的比较。是用来测试应用功能表现的方法，例如可用性、受欢迎程度、可见性等。

蓝绿部署是为了不停机，灰度部署是对新版本的质量没信心。而 AB 测试是对新版的功能没信心。注意，一个是质量，一个是功能。

比如，网站 UI 大改版，推荐算法的更新，流程的改变，我们不知道新的版本否会得到用户青睐或是能得到更好的用户体验，我们需要收集一定的用户数据才能知道。

于是我们需要在生产线上发布两个版本，拉一部分用户过来当小白鼠，然后通过科学的观测得出来相关的结论。AB 测试旨在通过科学的实验设计、采样样本代表性、流量分割与小流量测试等方式来获得具有代表性的实验结论，并确信该结论在推广到全部流量时可信。

我们可以看到 AB 测试，其包含了灰度发布的功能。也就是说，我们的观测如果只是观测有没有 bug，那就是灰度发布了。当然，如果我们复杂一点，要观测用户的一些数据指标，这完全也可能做成自动化的，如果新版本数据好，就自动化地切一点流量过来，如果不行，就换一批用户（样本）再试试。

对于灰度发布或是 AB 测试可以使用下面的技术来选择用户。

- 浏览器 cookie。
- 查询参数。
- 地理位置。
- 技术支持，如浏览器版本、屏幕尺寸、操作系统等。
- 客户端语言。

小结

部署应用有很多种方法，实际采用哪种方式取决于需求和预算。当发布到开发或者模拟环境时，停机或者滚动部署是一个好选择，因为干净和快速。当发布到生产环境时，滚动部署或者蓝绿部署通常是一个好选择，但新平台的主流程测试是必须的。

蓝绿部署也不错，但需要额外的资源。如果应用缺乏测试或者对软件的功能和稳定性影响缺乏信心，那么可以使用金丝雀部署或者 AB 测试发布。如果业务需要根据地理位置、语言、操作系统或者浏览器特征等参数来给一些特定的用户测试，那么可以采用 AB 测试技术。

策略	不停机	网关流量	用户采样	成本	回滚时长	复杂度
停机	✗	✗	✗	低	长	低
蓝绿	✓	✗	✗	中	短	中
滚动	✓	✗	✗	低	长	低
灰度	✓	✓	✗	中	一般	中
A/B	✓	✓	✓	高	一般	高

好了，我们来总结一下今天分享的主要内容。首先，常见的部署升级策略有停机、蓝绿、滚动、灰度和 AB 测试这几种。然后，我讲述了每一种部署策略的含义和优缺点。最后，我将它们放在

一起做了一个比较。下篇文章是《分布式系统设计模式》第三部分——性能设计的第一篇 " 缓存 "。希望对你有帮助。

也欢迎你分享一下你接触到的部署方式有哪些？在什么场景下使用哪一种部署方式？

文末给出了《分布式系统设计模式》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

- 弹力设计篇
 - [认识故障和弹力设计](#)
 - [隔离设计 Bulkheads](#)
 - [异步通讯设计 Asynchronous](#)
 - [幂等性设计 Idempotency](#)
 - [服务的状态 State](#)
 - [补偿事务 Compensating Transaction](#)
 - [重试设计 Retry](#)
 - [熔断设计 Circuit Breaker](#)
 - [限流设计 Throttle](#)
 - [降级设计 degradation](#)
 - [弹力设计总结](#)
- 管理设计篇
 - [分布式锁 Distributed Lock](#)
 - [配置中心 Configuration Management](#)
 - [边车模式 Sidecar](#)
 - [服务网格 Service Mesh](#)
 - [网关模式 Gateway](#)
 - [部署升级策略](#)
- 性能设计篇
 - [缓存 Cache](#)
 - [异步处理 Asynchronous](#)
 - [数据库扩展](#)
 - [秒杀 Flash Sales](#)
 - [边缘计算 Edge Computing](#)



版权归极客邦科技所有，未经许可不得转载

精选留言



曹铮

3

我觉得现在很多方案混合了蓝绿和灰度，这二者不会细分了。首先微服务架构下，单个服务的部署带来的冗余成本很低。同时也通过网关做流量的逐步迁移。发现问题回滚很快。尤其是基有了容器编排之后就更方便了

2018-05-08



画圈圈

2

太笼统了，关键是实现细节。

2018-05-12



shufang

0

滚动部署：逐个上；灰度部署：全上，逐步切换；蓝绿部署：全上，预发生产切换；AB测试：全上，同时存在。不知道理解的对不对？

2018-05-25



秋天

0

有些东西能不能具体说说实现方式呢？谢谢！

2018-05-08