12 | 极限挑战，如何做到分钟级搭建环境?

2018-07-31 王潇俊

12 | 极限挑战，如何做到分钟级搭建环境?
朗读人：王潇俊　12′18″ | 5.64M

在上两篇文章中，我介绍了环境管理中最关键的几个概念，环境的标准化，让环境自己说话以及环境配置的几种方法。

今天，我分享的主题就是，如何从零出发, 实现一套完整的环境创建。并且尝试挑战一下，如何做到分钟级交付。毕竟，天下武功，无坚不摧，唯快不破。

环境构建流水线

当开发人员向你申请一套新环境时，作为测试环境的维护者，你首先需要明确打造环境构建流水线需要关注的三大内容：

1. 虚拟机环境准备，根据环境的应用数、每个应用需要的硬件配置，准备好环境的硬件资源。
2. 应用部署流水线，在标准化的虚拟机上进行应用部署，当出现问题时如何容错。
3. 环境变更，在 SOA 或微服务的架构体系下，常常会因为测试的需求，将几套环境合并或拆分，创建环境时，你需要考虑如何高效地完成这些操作。

ol>

接下来，我会针对这三大内容进行展开，带你快速搭建一套环境。

虚拟机环境准备

在部署应用之前，我们首先需要创建应用部署的虚拟机环境。目前在携程，我们使用 OpenStack 做物理机和虚拟机的初始化的工作。

1. 当物理机接到机架上以后，打开交换机端口，等待机器被发现后，调用 Nova 进行物理机基本的硬件配置。

2. 物理机环境准备完毕后，从 OpenStack 获取虚机所需的镜像、网络等信息，调用 OpenStack API 进行虚拟机部署。虚拟机配置的一个关键点是，如何对网络进行配置。
   携程的测试环境使用的是大二层的网络架构，配置简单。但如果你对测试环境的网络规划是，需要做每个测试环境的独立的网段切分的话，配置会更复杂。

3. 虚拟机初始化后，需要在虚拟机上进行一些基础软件比如 JDK，Tomcat 的安装和配置。业界一般采用的方式是，通过自动化的配置管理工具来进行操作。
   目前，市场上主流的开源配置管理工具有 Puppet、Chef、Ansible、SaltStack 等。这几款工具都能帮助你很好地处理配置问题，当然它们也有自己独特的设计思想，实现语言也不同，你可以根据自己的技术背景和要管理的环境情况挑选适合自己的工具。

讲到这里，你肯定会有疑问。虚拟机的初始化流程已经这么复杂了，这个过程已经远远不是分钟级了，那我在文章开始部分说的分钟级是如何实现的呢？

我的建议是，采用资源池的方案。你可以根据用户平时使用虚拟机的情况，统计每天虚拟机申请和销毁的具体数量，预先初始化一定量的虚拟机。 这样用户从上层的 PaaS 平台创建环境时，就不用等待初始化了，可以直接从资源池中获取虚拟机，这部分的时间就被节省下来了。

但是，采用资源池的方式也有一定的复杂性，比如机型多、资源使用率难以预先估计等问题，当然这些问题对云计算来说，可以被轻松搞定。

应用部署流水线

由于不同公司的中间件和运维标准不同，部署流水线的差异也会很大，所以这里我只会从单应用部署标准化、应用部署的并行度，以及流水线的容错机制，这三个关键的角度，分享如何提速环境的搭建。

1. 单应用部署标准化，这是整个环境部署的基础。对一套测试环境而言，每个应用就像是环境上的一个零件，如果单个应用无法自动发布或者发布失败率很高，那么整个环境就

更难以构建起来。而如何实现一个好的发布系统，提升单应用部署速度，我会在后面的文章中详细介绍。

2. 应用部署的并行度，为了提高环境的部署速度，需要尽可能得最大化应用部署的并行度。理想的情况下，环境中的所有应用都可以一次性地并行部署。
   然而，做到一次性并行部署并不容易，需要保证：应用都是无状态的，并且可以不依赖别的应用进行启动，或者仅仅依赖于基础环境中的应用就可以启动，且可以随时通过中间件进行调用链的切换。
   在携程，我们力求做到所有应用都可以一次性并行部署，但这条运维标准并不通用。
   当我们需要更复杂的应用部署调度规则时，一个原则是将应用部署的次序、并行方式的描述交给开发人员去实现，并基于 DevOps 的理念，即调度策略和规则可以通过工具代码化，保证同一套环境反复创建的流水线是一致的。

3. 流水线的容错机制。对于环境构建工具，通常的做法是力求做到全面的标准化、代码化。但是因为环境的创建本身是一个非常复杂的工作流，在创建过程总会有一些异常中断整个流程。比如，某个应用启动失败了。
   而对于这些工作流中的异常，我们应该如何处理呢？

   - 第一种方法是，错误中断法。 创建环境过程中，各种资源申请、应用部署出现问题时，我们将工作流快照下来，然后收集所有的异常信息，返回给用户。由用户判断当前的情况，等用户确认问题已经得到解决后，可以触发一次快照重试，继续被中断的流程。
   - 第二种方法是，优先完成法。 创建环境过程中发生错误时，先进行几次重试。如果重试依然发生错误的话，就忽略当前错误，先走完剩余的流程，等所有的流程都走完了，再一次性将错误返回给用户。
     从整体速度上来看，第二种优先完成的处理方式是更优的，而且也会更少地打断用户。只是方式二需要保证的关键原则是：所有的部署脚本的操作都是幂等的，即两次操作达成的效果是一致的，并不会带来更多的问题。

环境变更

实现了应用部署流水线后，创建环境的主流程，即虚机准备和应用部署已经完成，环境已经可以工作了。但还是不能忽略了后续环境变更的需求和工作。一般情况下，研发人员变更环境主要有以下 4 种场景。

1. 已经有一套新环境，当有新项目时，开发人员会挑选部分应用，组成一个独立的子环境。这里的重点是，要保证子环境和完整环境的调用是互相隔离的。

2. 当存在多个子环境时，可能在某个时间点需要做多个项目的集成，这时开发人员需要合并多个环境。

3. 和合并的情况相反，有些情况下，开发人员需要将一个子环境中的应用切分开来，分为两个或者多个环境分别进行隔离测试。

4. 已经存在一个子环境，当多个并行项目时，开发人员会克隆一套完整的子环境做测试。

对于这 4 个场景，我们需要关注的是在多并行环境的情况下应用拓扑图，包括用户访问应用的入口、应用之间调用链的管理，以及应用对数据库之类的基础设施的访问。

1. 用户访问应用的入口管理。 以最常用的访问入口（域名）为例，我推荐的做法是根据约定大于配置的原则，当环境管理平台识别到这是一个 Web 应用时，通过应用在生产环境中的域名、路由，环境名等参数，自动生产一个域名并在域名服务上注册。
   这里需要注意的是，域名的维护尽量是在 SLB（负载均衡，Server Load Balancer）类似的软负载中间件上实现，而不要在 DNS 上实现。因为域名变更时，通过泛域名的指向，SLB 二次解析可以做到域名访问的实时切换。而如果配置在 DNS 上，域名的变更就无法做到瞬时生效了。

2. 应用之间调用链的管理。 对于 service 的调用关系，我在《“配置”是把双刃剑，带你了解各种配置方法》这篇文章中，提到了携程开源的配置中心 Apollo 的实现策略，所有的服务调用的路由都是通过环境描述文件 server.spec 自发现的，你只要保证文件的环境号、IDC 等属性是正确的，整个调用链就不会被混淆。
   同时，服务调用中间件需要可以做到自动判断，被隔离的环境内是否有需要被调用的服务，并在当前环境以及基础环境中间进行自动选择，以保证服务被正确调用到。

3. 对数据库的访问。 一是，数据库连接串的维护问题，与 SOA 调用链（即服务之间的调用关系）的维护类似，完全可以借鉴；二是，数据库的快速创建策略。
   对于数据库中的表结构和数据，我们采取的方式是根据生产中实际的数据库结构，产生一个基准库，由用户自己来维护这个基准库的数据，保证数据的有效性。并在环境创建时，提供数据库脚本变更的接口，根据之前的基准库创建一个新的实例，由此保证环境中的数据符合预期。

对于环境的创建和拆分，最主要的问题就是如何复制和重新配置环境中的各个零件。环境创建，就是不断提高虚拟机准备和应用部署两个流水线的速度和稳定性；环境拆分，则需要关注以上所说的三个最重要的配置内容。

而环境的合并需要注意的问题是，合并后的环境冲突。 比如，两套环境中都存在同一个服务应用，而两者的版本是不一致的；又或者，两个环境各自配置了一套数据库。此时该如何处理呢。

因为环境的描述已经被代码化了，所以我们解决这些问题的方式类似于解决代码合并的冲突问题。在环境合并前，先进行一次环境的冲突检测，如果环境中存在不可自动解决的冲突，

就将这些冲突罗列出来，由用户选择合适的服务版本。

如何高效、自动化地实现环境变更的关键点还是在于，我在前面几篇文章中提到的如何管理和实现应用配置和环境配置，以及如何配合环境管理在速度上的需求。

总结

对于如何快速搭建一套环境，我从虚拟机环境准备、应用部署流水线和环境变更，这三个方面给你总结了一些常见问题和原则：

1. 可以使用虚拟机资源池，提升获取机器资源的速度；
2. 合理打造并行的应用部署流水线，是进一步提升环境创建速度的方法；
3. 利用配置等方式快速达到环境变更需求，可以再次有效地提升整个环境部署的效率。

思考题

你所在的公司，新环境应用部署的流水线是怎样的？如果要进一步提速的话，还有哪些优化空间呢？

欢迎你给我留言。

通过留言可与作者互动