目录

1. 前言
2. 参考
3. 方法
4. 架构
5. 成果

识别运维平台的边界在哪儿，才能更好地构建平台，从而协助运维的日常工作。

在之前的文章中，谈到过“运维的本质——可视化”，在可视化的篇幅中，着重介绍自动化的可视化和数据的可视化；在后续的篇章中又介绍了“互联网运维的价值体系”，里面分解了几个维度：质量、成本、效率、安全等。以上都是为了清楚地梳理运维的内容边界，基于这个边界，我们再考虑如何进行平台支撑。可以说前两篇文章都是为今天这篇文章作为铺垫，用理念先行，然后再考虑平台落地，最后再细化其中每个内容。我更习惯用如下的方式来整体表达运维的工作方法和思路。

首先，价值导向。找到一个价值方向来牵引整个团队很难，但又必须找到，因这个牵引力就决定了团队的气质及后续的工作方法；之前的文章“运维价值体系”有详述，在此不细谈。

其次要有一个分而治之的系统，最后面向业务自底向上的集成，此时便能帮忙实现更好、更快、更省的交付价值。平台的建设需遵循一些的方法（自底向上、先后顺序等），先建设各个运维专业子系统，通过API的方式对上暴露服务，最后不同的业务平台去调用这些服务接口即可。缺少平台的支持，运维的质量、成本、效率都会直接受到影响。

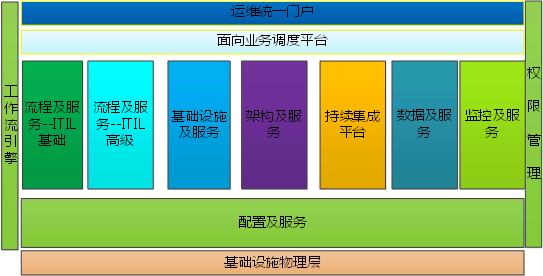
如果要做好服务器精细化成本控制，此时需要一个平台来处理从服务器资源上采集的资源使用状态数据，并生成可视化数据报表，共享到所有团队中，在一致理解下，去驱动成本优化，越海量的业务对这个平台的要求就越高，从采集、处理、模型算法等都有很高的要求。

不要忘了这个平台还包含面向业务技术栈构建的平台。这地方有一个非常好的例子，在2012年左右，我了解到Google有一个非常强大的资源管理平台Borg（后面叫Omega），它的设计目标是“把数据中心看成一个芯片”。Google研发人员将开发的服务交给Borg，后续的服务生命周期（扩容、缩容、调度）都由Borg统一接管，服务被Borg部署到哪个IDC、哪个服务器，研发人员不用关心。后来Twitter根据Borg的思想，也开源实现了一个平台——Mesos，不过Mesos对LongTime的服务调度（如Nginx）支持不是太好，更适合MapReduce的事务调度。这两个资源管理平台背后的思想都值得深究，建议看看。

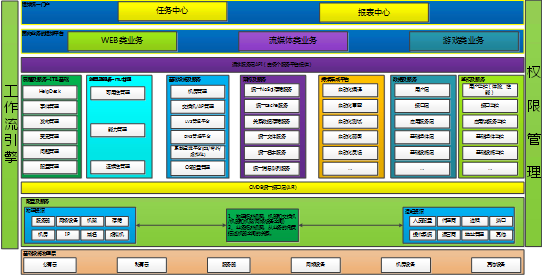
第三，基于平台，提供透明服务，确保服务提供者和服务交互者之间的交互越少越好。有了整合性的平台，透明提供服务也成为可能。平台整合就是避免服务被碎片化，从而让使用的用户看到的不是一个一个工具或者孤立系统，而是面向业务的整合服务。此时成本便可降低、变更的质量也会变成一个稳定态。不同的人、不同的时间执行相同的事务流程都能取得一致的执行结果。

最后，数据驱动。因所有线上业务服务和线下运维服务都有状态，需数据平台提供服务状态数据的采集、处理、分析处理能力，最后还能让运维人员自定义分析报表。技术运营数据和产品数据的一个很大的区别是，前者在数据挖掘方面的能力要求很少。这个地方有个建议，把线上服务的数据驱动作为重点（80%），把运维内部服务的数据驱动为辅（20%）。因为线上服务的状态会反作用于运维内部事务的优化。比如说从数据中发现现网的服务有一个故障，需要紧急发布版本，此时就会直接检验运维的变更部署流程、平台的完备性。

在平台体系部分，我采用逐级构建的方法，不断去细化其中的内容，因此会有一级视图和二级视图，在这个地方，我不敢到三级的模块级别，基本上不可看，下图是参照的是eTOM模型构建方法。



继续往下，可以分解出二级视图。



有了整体的平台体系视图，接下来看看每一部分到底是干什么的。

工作流引擎、权限管理。这两者都是基本的功能，因为其中会涉及流程，所以需要统一的流程引擎平台。另外需要部门、角色、用户的权限管理统一管理，不同业务配置不同系统的使用策略即可，这一块可以统一实现在单点登陆系统中。

1. 基础设施物理层。这个视角和传统模式有些不同，主要是公有云的存在。因此在基础设施物理层这块，已经把云端资源当作一个底层基础设施来看待，后续的资源获取完全不同，其他的资源对象依然没有变化，依然是机房、机柜、网络、服务器，等等。
2. 配置及服务，把配置当作服务来看待。在ITIL中叫CMDB，Configuration Management Database， CMDB也可以理解成统一的元数据库，比如说机房信息、服务器信息、人员信息、服务信息、业务信息以及他们之间的物理和业务拓扑关系等，上层的所有系统都应该关联到CMDB，变更后的信息必须实时反馈到CMDB中，确保其他系统能同步这份变化。因此大家都把CMDB系统当作运维的核心系统来对待，便于后续各个系统之间的互通。

在我的经验中，CMDB建设还是有非常多的坑。如果你把iTop或者oneCMDB的产品当着标杆（都是开源，没见过商业的），那你的CMDB建设就完了。之前在一家传统企业，他们把文档都放到CMDB中管理，不建议这么做，文档就是SCM的事情。CMDB建设的核心准则：CMDB管理的数据一定要为了业务管理，业务管理上不需要的东西，就果断舍弃，比如说文档，和业务没有任何关系，就可以不考虑纳入，后续会有专门的文章介绍。

1. ITIL服务——基础、ITIL服务——高级。在早期的文章中把DevOps和ITIL做了对比，ITIL是面向流程的，这个可以在运维平台建设中不做重点，不要主动去构建流程，会影响运维的敏捷性。基础部分实现一个事件和HelpDesk即可，事件管理在告警转换成事件之后，可以完整地记录，便于我们事后的原因分析，能挖掘一些问题，比如说是否某个业务、某个人、某类机器经常性故障，那就需要重点关注下。高级服务的部分，大家需关注一下，它是可以带来价值的，比如说可用性管理、能力管理和连续性管理。可用性直接的导向就是业务的质量；能力管理直接的导向就是成本管理；连续性管理也是和质量戚戚相关，如业务的容灾、备份管理等。但这些管理都不要在流程层面上去看，需要在一个平台中进行全面的可视化管理。后续的篇章也会有相应的介绍。
2. 基础设施及服务。把底层运维资源的管理封装成一个一个的服务，供业务自动化平台使用。我把DNS、LVS（或者F5）甚至OS上的配置管理都看着基础设施部分，适当地向上延伸了一下。简单的划分原则是，在业务架构之外的，都可当着基础架构部分了。很多运维团队的建设重点都在这块。
3. 架构及服务。把业务架构中的共性需求都剥离出来，抽象成一个一个的服务，最终让研发只需要关注自己的业务代码即可，比如说统一文件存储、统一Nosql存储、统一RDS存储、统一队列等。这块对运维的质量、效率、能力等影响最大，在之前的文章“如何化解研发和产品之间的矛盾”中重点阐述过服务公共化是唯一的解决之道。现实中如果有研发开发了一个公共组件交给运维，而不提供完整的Webadmin或者API的话，你也就可以认为他是在耍流氓，运维必须有严格的完整性交付要求。
4. 数据及服务。只要有线上服务在运行，服务数据流经过的一切节点产生的数据，你都要采集、存储和分析起来，供不同的运维场景使用。比如说自动化调度，可以根据业务涉及的基础节点资源使用情况，制定对应的自动化调度策略；可以在数据中直接进行故障定位；可以在数据中做安全分析。之前的文章“数据驱动运维”中介绍过我做的一个数据分层体系。
5. 监控及服务，有数据的地方才有监控。脱离这个原则，你做的都是告警，并且告警的成本会越来越大，不成体系。个人观点：所有的监控视图都是来源于我们对数据的采集以及我们到底有多少经验来看待数据。
6. 持续集成。这条线是把一个个的程序包交付到各个环境，在【持续部署】之上的部分可以通过和持续集成工具Jenkins或者Go作对接即可。持续反馈非常重要，一个程序部署到生产环境之后，需要实时的运行报告反馈回来，确认变更的效果。如果持续部署平台化之后，真正的执行部署工作会不断前移，甚至可能直接交付给研发。此时的状态报告，更是有必要，不需要人去登录主机tail日志看是否正常。这个地方和“数据及服务”的能力关联很大，没有前面强大的数据服务能力。
7. 面向业务的运维平台。不同的业务会有不同的调度策略和服务使用策略，需要在更上层完成面向业务的统一调度，这个是全应用的视角，和持续集成是有一些区别的。在没有这个平台之前，一个完整的业务上线，需要做很多操作，比如说DNS变更、LVS变更、OS初始化、自动化测试、持续部署、持续反馈、监控、业务调用关系配置，等等。面向业务的调度平台，就需要有一种调度能力，指挥底层各个平台为它服务，它本身不实现任何服务接口，是一个服务的集成者。
8. 运维统一门户。每个运维系统都有任务或者信息与自己相关，如果运维人员每天要去面对那么多的运维系统，会非常痛苦。在统一门户里面分成两个部分，一部分是任务中心，把底层所有的事务状态都同步到任务中心中，表示我要做什么；信息中心，就是让运维人平时关注的业务状态Dashboard直接推送到信息中心中，表示我要关注什么。

自动化运维平台的目标就是自动化和数据化一切，并且最终可视化，从而确保质量、效率和成本几者之间的平衡。

但对于这么一个庞大的复杂体系来说，不可能一蹴而就，可以借鉴一下经验。

1. 自底向上。一定要把握这个原则，这就相当于我们造车一样，把各个零件造好了，最后就是组装。
2. 加强跨团队之间的合作与沟通。很多事情一旦研发、测试和运维彼此合作，事半功倍。在合作的过程中，把彼此的需求都统一到平台中，这样有利于后续的推广和使用。
3. 平台建设先后有序，优先级顺序如下：

l P1（最高）：CMDB、基础架构及服务、数据及服务、监控及服务、持续集成；

l P2（次高）：面向业务的运维平台；

l P3（低）：ITIL相关、运维统一门户。