企业级应用落地实践系列

Cloud2.0时代容器技术一本通

企业级落地实践

秦小康 主编◎

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

企业级应用落地实践系列

Cloud2.0 时代

容器技术一本通

企业级落地实践

秦小康　主编◎

電子工業出版社

Publishing House of Electronics Industry

北京・BEIJING

内 容 简 介

本书将以企业落地实践为切入点，分享作为终端用户的企业在关键业务环境中落地使用 Docker 及 Kubernetes 技术的经验和心得。内容既有扎实的技术实现的方式和方法，又有各行业容器技术企业级落地实践的深度解读。

本书内容将包含以行业领域为划分的企业级容器技术落地典型案例，以场景为出发点的容器落地常用场景，并列举企业容器落地应该关注的几大技术点。本书对数十家企业一线架构师和工程技术人员对容器技术的使用实践做了整理和分析，同时也详细列举了具体的生产环境技术栈和配置参数，供相关企业技术人员进行复盘和参照。

图书在版编目（CIP）数据

Cloud2.0 时代容器技术一本通：企业级落地实践 / 秦小康主编. —北京：电子工业出版社，2019.7
（企业级应用落地实践系列）

ISBN 978-7-121-36569-0

Ⅰ. ①C… Ⅱ. ①秦… Ⅲ. ①云计算—研究 Ⅳ.①TP393.027

中国版本图书馆 CIP 数据核字（2019）第 092765 号

策划编辑：刘志红（lzhmails@phei.com.cn）
责任编辑：刘志红　　特约编辑：李　姣
印　　刷：涿州市京南印刷厂
装　　订：涿州市京南印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1 092　1/16　印张：25.5　字数：652.8 千字
版　　次：2019 年 7 月第 1 版
印　　次：2019 年 7 月第 1 次印刷
定　　价：108.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254479，lzhmails@phei.com.cn。

编 委 会

序

若你是一个容器技术初学者，你会不可避免地连续听到容器（Container）、Docker、Kubernetes、Mesos、CaaS、编排（Orchestration）、调度（Schedule）、微服务、DevOps、CI/CD（持续集成/持续交付）、Containerd、RunC 等名词。而且在被这些名词和概念不断冲击的同时，会有另外的信息从另外的角度传递进来："Docker 有 xxxx 缺点和问题！""容器技术之后的下一代技术 ServerLess 更厉害！""函数服务才是未来！"，等等，不一而足。

但事实上，容器技术本身只是一种 IT 技术，它属于工程科学的范畴。和 IT 历史上诸多的技术一样，都是从计算机技术实践中大量的工程经验积累到一定程度从而总结优化而来的产品，而不是依据某种理论和规划，先形成产品，然后被广泛使用的。

在和团队筹划这本书的时候，我一直在想，我们究竟该传递什么知识和信息、用什么方式来传递这些知识和信息，才能对本书的读者有更大的帮助。我给自己和团体提出这个问题的原因是目前技术阅读领域和几年前相比，正在发生天翻地覆的变化。说起我们当下的时代，我认为用知识爆炸这个词比较合适。同时，当前也是互联网发展到顶峰的时代。这个时代给阅读方式带来的改变是十分巨大的，一方面在搜索引擎的帮助下，需要的信息唾手可得；而另一方面，由于传播的便捷性，我们很难分辨各个渠道获取的信息是否就是正确信息。因此，读书则逐渐转变成为一种为了追寻更高境界和跟进最新技术动向的行为。

在我们所处的 IT 领域，纸质的书籍阅读被网络信息和电子读物大规模替代，平时查找行业相关资料，获得疑难问题知识点、获知行业动态等基本都是通过网络获得的。当我们需要在某个领域深入研究、查阅经典的时候，我们很容易发现这样的书籍名称《xx 权威指南》《xx 宝典》《xx 大全》，而事实上，我们极少真正得到及时的"权威指南"，也很少真正从"宝典"和"大全"里及时获得系统的知识。究其原因，主要是因为技术知识迭代的速度大大提高，而讲求精确的传统书籍出版进程则相对滞后。如果我们把目光聚焦到本书要谈及的容器（Container）技术领域，情况就更加极端。容器技术领域是 IT 领域有史以来更新迭代最为频繁，新技术和新工具颠覆频率最高，而且对原有被颠覆技术抛弃最为彻底的领域。原来以几年为单位的更新周期，在容器领域则变为不足一个月。如此高频率的技术创新和技术更迭，一方面是对工业界生产力的大幅革新和升级，另一方面，对于非相关专业和领域技术人员也带来了极大的挑战，尤其是刚刚接触容器领域的技术人员，会感到十分困扰。

因此，我们确定本书的第一个目标就是，为容器技术的初学者明确学习和提高的方向，这个方向就是通过不断的实践操作，然后在实践过程中不断加深和扩展对上述提到的各种技术产品和概念的理解。

本书的第二个目标是为企业的 IT 人员提供一种新的思路，引导传统企业的研发和运维人员主动参与技术革新，从自身朴素和重复但最具有意义的开发、测试、运维工作实践中，不断提炼和推进一个非常伟大的进程——实践技术化和技术民主化进程。

一直以来，IT 技术的使用实践都是被引导的行为，很多研发、测试和运维工程师是不断被动地从网络媒介和各种推广渠道接受某种技术和某种产品，进而应用到自己的生产实践环境中，然后不断地试错、调优、解决问题、问题累积、寻找下一种产品，进而再重复这个循环过程。而事实上，因为时间和精力的限制，企业技术人员很难有精力对需要使用的技术进行全面的掌握和测评，甚至很难完成对所用产品的高质量的 PoC（Proven of Concept）。

在这种情况下，参考同行业的使用经验和方法，参考不同行业对同一个技术或产品的使用实践及分析，学习交叉行业的使用场景就成为一种有效的参考手段，而本书对容器技术的分享和引导，主要专注于早期的成熟用户、创新用户对容器在各种行业、各种生产系统、各种细分领域的使用实践的分享。

事实上，在每个企业生产环节，因为政策、环境、资源、技术栈等各方面的不同，很小的差异都有可能会带来十分巨大的差别，所以，完全照搬技术应用实践的方法也不可行。我们想展现给读者的是从众多 Rancher 用户多样的容器技术实践中，基于一线技术人员的实践经验，从中提炼出利用容器技术解决生产环节实际问题的方法论，然后根据自己企业实际情况，形成理性和最优的方案。

从内容的选择上，本书对数十家企业一线架构师和工程技术人员对容器技术的使用实践做了整理和分析，同时也详细列举了具体的生产环境技术栈和配置参数，可以供相关企业技术人员进行复盘和参照。

和 Rancher 一直秉承的一线技术人员技术民主化和技术创新理念一样，本书也邀请这些一线技术人员作为编辑，对相关的技术实践部分做了严格的实验验证和校对，也希望广大的读者以本书为起点，从众多一线技术人员的智慧结晶中提取对自己有用和宝贵的方法，把容器技术优化地应用到自己的生产和实验环境中，为企业生产力的提高提供原动力。

随着更多技术人员的参与，本书也是作为丛书的第一本面市。正如前文提到的，容器技术本身的发展日新月异，我们也会紧跟技术发展的步伐，持续编辑和总结，不断把最新的技术、最佳的方法、最优的实践推广和分享给广大技术读者。

在此特别感谢美丽的 Shirley Huang（黄绪）女士为本书面市所做的大量整理和编辑工作。这其中包括和 40 多位技术人员的反复对接和确认、对内容上下衔接的编排、对实践环境的确认，以及对内容合规性的各种检查等。Shirley 的辛勤工作是本书如期面市的最大保障，希望广大读者朋友和我一样，为优秀而努力的 Shirley 点赞！

2019 年 3 月 于深圳

秦小康

目录

第一章

容器技术的典型行业落地案例

☑ 保险、证券行业 | ☑ 银行、金融行业

1.1 通联数据如何使用 Docker+Rancher 构建自动发布管道

蔡云飞

本文分享了通联数据使用 Docker 和 Rancher 构建自动化发布管道的经验。文章介绍了通联数据自动化发布的流程及方案设计，从踩过的一些坑中总结出来的经验，并分析了自动化发布管道的系统运行现状。

一、通联数据的需求及选择 Rancher 的原因

通联数据是一家近几年新崛起的金融科技公司，主要目标是致力于将大数据、云计算、人工智能等技术和专业投资理念相结合，打造国际一流的、具有革命性意义的金融服务平台。这里面涉及几个关键词：大数据、云计算、人工智能，这些热词已经有很多企业提及过，因此我也不赘述了。

通联数据作为一家创业公司，也会遇到很多创业公司都会遇到的问题，比如产品太多，而且每年很多产品都会被推翻重来，会遇到各种各样的问题。比如，应用在开发时，涉及需要多少 CPU、需要用什么语言去编程等问题，开发人员不会与后台人员事先沟通，出来一个产品，直接让运维人员上线。

基于这样一个背景，我们迫切需要打通上线这条管道，因为在可预见的将来，通联数据每年将会有大量的新应用出现，如何打通上线管道问题亟待解决。

我们做的第一件事，就是**持续集成**。由于我们公司每年会有上百个新的项目产生，每个项目的语言、框架、部署方法都不尽相同，发布流程也是比较冗长和低效的，部署应用占用了运维人员大量时间。

我们决定用容器解决这个问题，评估了多家供应商后选择了 Rancher，主要原因在于：首先，**Rancher 的操作界面非常简单**，相信用过 Rancher 的人都会有这种感受，它不需要太多专业的知识，很容易上手；其次，Rancher 在部署时也非常简单，可以**一键部署**，Rancher 还提供了**良好的 API 支持**，方便集成。

二、自动化发布的流程

随后，我们开始搭建自己的 CI/CD。当时我们在流程方面遇到了很多困境，图 1-1 所示已经是简化制作后的流程，实际上还有更多的分叉和分支。

图 1-1 流程图

就我们原来的流程来说，流程最开始的部分是研发，随后进入 QA 环境部署，这时候就需要人去部署，通常是运维人员，但是运维一般不愿意做这件事情。部署完成后，进入 QA 环境测试阶段，通知开发和测试人员进行测试，过程中可能会出现延迟，因为测试人员可能正忙于其他事情，不能马上进行测试。QA 环境测试通过后，进入 STG 环境部署，随后再进入 STG 环境测试阶段，这几个过程可能会循环很多次。之后进入安全测试阶段，测试通过后，还有一个正式包准备过程，最后才是生产部署。这个简化后的流程是非常复杂的，而且其中涉及很多线下的沟通，效率不可能高得起来。

在使用 Rancher 之后，原本的流程大大简化了，改进后的简化流程如图 1-2 所示。

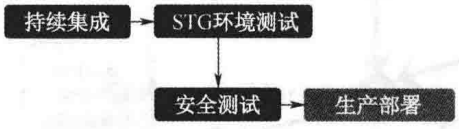

图 1-2 使用 Rancher 的流程图

流程的第一步即持续集成，意味着开发人员写好代码后可以直接通过 CI，CI 触发自动编译，随后自动部署脚本，测试环境已经就绪。简单来说，每次开发人员提交代码之后，测试环境就始终处于一个就绪的状态，这时候就可以直接进入测试阶段了，整个过程都处于线下，不需要走任何流程，全部实现自动化了。

测试人员完成测试后，再进行 STG 环境测试，因为后台已经跟 Rancher 完成对接并实现自动化了，这赋予了 QA 环境测试从未有过的强大的自动化能力，意味着 QA 环境测试可以自动对接到 STG 环境测试。测试通过后，进入安全测试阶段，这个阶段是公司要求的，无法避免，安全测试通过后就进入到生产部署。以前绕不开的线下沟通那些步骤和一些部署就可以省去了，整个流程优化并且简洁，提升了效率。

CI/CD 说起来很简单，比如推送代码、QA 环境自动整备，但是实际操作起来并非如此，仍然存在很多需要解决的问题。例如开发的分支模型涉及开发代码的时候，是要求 PUSH 的什么分支才能部署，还是 PUSH 所有分支都能部署？

三、开发分支模型

我们曾设想，最好就是 PUSH 的任意分支都能部署，这样就非常方便了。但随后就发现这种方法行不通，会造成混乱，而且难以管理。此前我们 Git 上保管的一个 The Successful Branch Modeling 分支模型就类似于此，如图 1-3 所示，此模型规定了一个 develop 分支、一个 feature 分支、一个 release 分支、一个 hotfixes 分支和一个 master 分支。

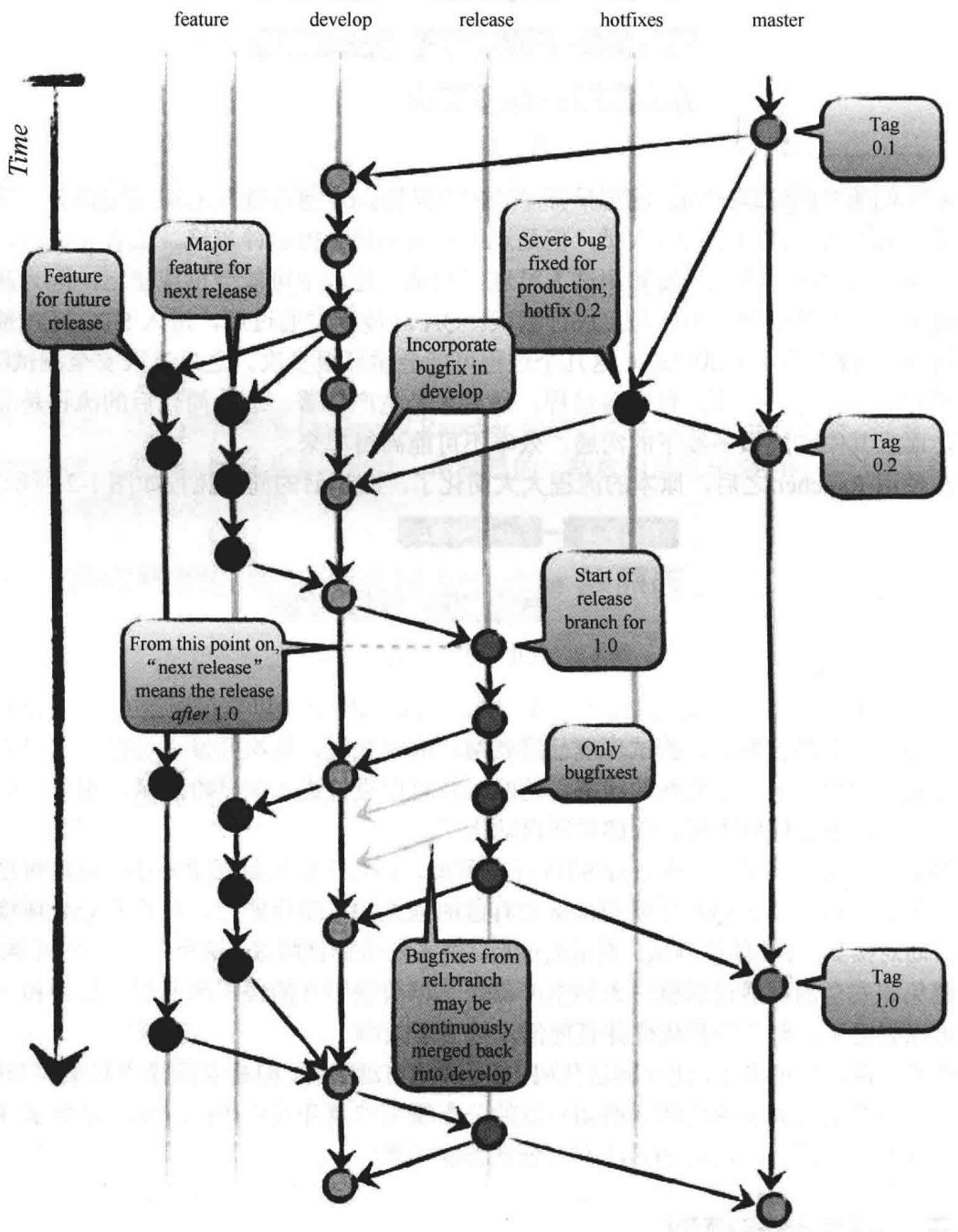

图 1-3 分支模型

在平时开发时，开发人员常常会在 develop 分支上切出一个 feature 的分支，比如，开发一个包含很多功能的 story，那么所有人切一个 story，这边 story 就有自己的 ID，然后再去开发，把 story 开发完之后，再把它合并过来。最终，我们只选择了一条线做 CI，当代码推送或者合并到 develop 分支时，我们帮你去做这项工作，而这个 feature 分支则另作处理。

feature 分支意味着当用户提交一个 feature 分支之后，我们会另外帮你部署一套环境，每一个 feature 分支部署一套环境，相当于每一个 story 都可以分开独立去测试，最后把它合并到 develop 分支。那么测试的时候，测试人员可以根据自己的需求来决定它到底在哪

一条分支线上去测试。

比如，A 测试中如果用户只关心 story A，那就在 story A 这个分支测试环境去测，这些 story 全部合并进来之后，再进行一次集中测试，测试通过后，在发布时把这个分支切到 release 的一个分支上来，然后，在 release 上发布正式包，让 QA 在 STG 环境继续进行测试，就如前面看的那个流程图。分支模型非常混乱，为了做 CI，我们会跟开发人员定义好每一个分支，同时定义好每个分支对应的不同行为，这在混乱的分支模型下非常有用。

四、版本号规则

为了做 CI，**版本号的规则必须一致**，如果每个团队版本号命名不一样的话，匹配规则就会非常麻烦。后来，我们选择了 Semantic Versioning 的一个版本号规则，就是几点几点 X，这是一种常见的版本号命名方法，版本号包含一个标准文档，文档描述了此版本号的具体定义，第一个叫 MAJOR，第二个叫 MINOR，第三个叫 PATCH，后面还可以加各种自己的版本号，如图 1-4 所示。

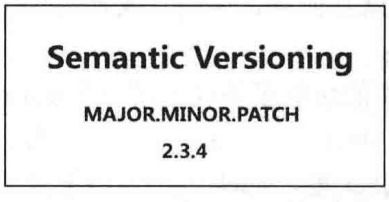

图 1-4 版本号命名规则示意图

五、CI 触发路径

下面将介绍我们的 CI 触发路径——Git push，push 到 develop 分支，Git push 就会 push 到 GitLab 的 server 上，随后通过 webhook 去调用 Jenkins，Jenkins 会把这个包编译出来。原本我们想通过 Jenkins 调用 Rancher 的 API，后来发现直接调用 Rancher 有困难，过程进行得也不那么顺利，为了解决 Jenkins 和 Rancher 之间的间隙（gap）这一问题，我们在它们之间装了一个 Ponyes 软件。CI 触发路径流程如图 1-5 所示。

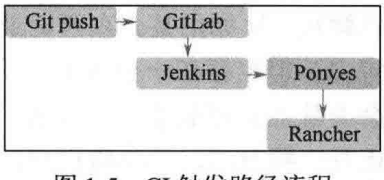

图 1-5 CI 触发路径流程

为什么需要 Ponyes 这一中间层？它到底提供了哪些作用？

1. 动态修改版本号。

首先，它可以解决动态修改版本号的问题，大家用 Rancher 的时候，发现 Rancher 的商店非常好用，我们可以在商店里把一些东西定义好，接着 QA 只要填几个参数，就可以把一个应用部署起来了，在没有 Ponyes 这一中间层之前，这部分必须找运维人员去做，过程也比较复杂。

为了使用 Rancher 应用商店，我们还定义了一个应用商店的模板，这样我们就可以在

每次推送代码的时候，用这个模板生成一个真正可以部署的应用。但是，版本号还是存在一些问题的，我们每次推送代码的时候，到 Jenkins，我们会根据阅览数给 Jenkins 升级一次版本号，比如说 1.0.1.0-1，第一次阅览是-1，第二次阅览即-2，Jenkins 的版本号根据阅览次数相应变化。这时候应用商店也要随之而变。

因此，我们做了这样一个模板，通过这样一种方式，在 Ponyes 中，每个 Jenkins 可以获得对应的版本号，然后把这个版本号通过变量注入进去：

```
sample:
  image:    {{ REGISTRY }}/automation/auto-sample:{{ m['auto-sample']}}
```

这一过程还涉及注册，因为 QA 环境和 STG 环境是完全分开的，在进行模板渲染时，我们需要知道到底是发到 QA 环境还是 STG 环境，从而对注册的地址做出相应改变，这样做的话，上面说的修改版本号的问题就解决了。

2. 多 service 管理。

还有一个比较棘手的问题，即一个 stack 中有多个服务怎么办？比如，一个比较小的团队可能总共就几个人，每个人负责好几个项目，与微服务的关系有些相似，那么一个 stack 可能有好几个服务，最典型的就是前端、后端，或者是其他的一些中间件，每一项是一个服务。

开始部署时，Rancher 里面的这些服务也要用一个 stack 来管理，有多个服务就会面临怎样管理的问题。比如，一个 stack 有 A、B、C 三个服务，服务 C 更新时，整个 stack 也应该更新版本号，因为在 Rancher 里，stack 被部署出来之后，有个 update available 的黄色按钮，如果有新版本，点这个按钮，就可以升级这个 stack。注意：是必须升级整个 stack，而不是只升级某个服务，这时候就需要管理多个服务。Ponyes 记录了服务和 stack 之间的关系，任意一个服务更新，都会触发 stack 的更新，更新的方法就是每个服务每次更新，stack 的版本号+1。

3. 多版本并行发布。

接下来还有一个更严重的问题，如果多版本并行发布，我们应该怎么去处理？比如，现有 A、B、C 三个模块，A 发布过 1.0 和 2.0，B 发布过 3.0 和 4.0，C 发布过 5.0 和 6.0。如果发布一个 C 应用的 5.0.2，那么对应的 A 和 B 模块的版本号应该选什么，这个问题困扰了我们很久。

这个问题有好几种方法可以解决，比如用某个东西记录 C5.0 和 A、B 的一个版本号之间的关系，这表示用户可以自定义。但也存在一个问题，用户需要自己去管理它们之间的版本号关系，时间长了，可能会弄乱或弄错版本号，或者弄错版本号之间的关系，随后上线到生产环境，后果更严重。还有一种方法，就是将用户的 C 版本号最后一次发布到 5.0 时 A、B 是什么版本静态地记录下来。但这样的话系统会变得相当复杂，很容易出错。

最后，我们选择去除多版本并行发布的能力，只支持单个版本的发布，这意味着如果要发布，必须是最新版本，历史版本可以用另外的方法进行人工处理。这样的话，系统会更简单，而且不容易出错，也不需要用户去维护版本号之间的关系。以上这些功能都是借助我们自己写的 Ponyes 解决的。

Rancher 2.0 也提到了 CI/CD 这样一个功能，在实际过程中，会遇到一个非常现实的问题，从代码研发到整个生产环境部署有许多环节，具体情况也非常复杂，而 CI 的作用可能仅止于 QA，后面的环节还会有新的难题，这时就需要一个体系把整个生命周期贯穿起来

管理，Rancher 承担的作用就在于此。

六、总结的三条经验

1. 部署几套 Rancher 环境。

这问题看似很小，但在我们内部也引发过不少讨论。最初我们只部署了一套 Rancher，在 Rancher 里面用环境去区分 QA 或 production。部署一套 Rancher 的方式最简单，但是存在一个严重的问题：遇到 Rancher 升级的情况怎么办？这套 Rancher 既管了 QA，又管了 production，升级的时候需要把生产环境也升级，此刻如果出现 bug，问题将非常严重。

后来我们把它**拆成四套**，Rancher 平台本身也需要有一个升级的环境。建议大家在前期部署 Rancher 的时候要部署多套环境(至少两套)，我们实际上是 dev、QA、staging、production 每个环境都有一套，总共四套。

2. 配置项爆炸。

我们最开始是使用 Rancher compose 的，它非常简单且功能强大，我们在 Rancher Catalog 里点一下“部署”，配置选项会被全部弹出来，我们只需要点选一些东西，就可以部署一个很复杂的应用。后来我们却发现，应用的配置项越来越多（甚至多达上百个），这导致它们难以在一个页面里展现。同时，上百项的配置让我们无从填写，运维也无法成功部署，从而面临了配置项爆炸的问题。

我们的解决经验是，在容器平台里面，**配置项一定要集中管理**。我们把配置项全部采用 consul 管理，每次容器启动时到 consul 里把配置拉到对应的容器里来。如此一来，容器就可以在任意平台漂移。另外，配置项本身在原 server 存有副本，我们复制原配置项加以修改，就可以部署另一个实例了。所以说，配置项一定要全部集中起来管理。

3. 泛域名+Rancher LB。

我们每一个业务都有一个 Web 服务，要申请自己的域名。每年我们有上百个项目上线，这意味着有上百个域名要申请。加上这些域名都分别需要在开发、测试、生产环境中使用，所以我们每年要申请将近 500 个域名。这是一件“恐怖”的事情。后来我们就用了泛域名的方法。比如用*.sub.example.com 的域名，直接 CNM 到 Rancher 环境中的其中一台主机上。然后在 Rancher 上设置它的 LB，LB 就可以分布在所有的主机上，每个主机上都会有同样的 LB。这个域名任意指向一台主机，它都可以工作。

比如说，在 QA 环境，LB 上服务了如下域名，若主机坏掉了，容器本身会启动，入口问题则可以很简单地通过修改*量来搞定。如果是生产环境则可以在上面再加层 nginx，配三个 upstream，任意一台出现问题，还可以通过另外两个入口进来。

使用泛域名的方法，即在配置好泛域名之后，在 Rancher LB 上再加任意域名都不用再去申请新的域名，而是可以直接写 123.sub.example.com，然后直接在 LB 上配置，配完之后域名即可用，无须再走申请的流程了。

☑ 互联网企业

1.2 移动医疗公司顺能网络基于 Spring Cloud 的微服务实践

赵安家

一、相关趋势图

首先给大家看一张百度指数上关于微服务、Spring Boot、Spring Cloud、Dubbo 的趋势图，如图 1-6 所示。

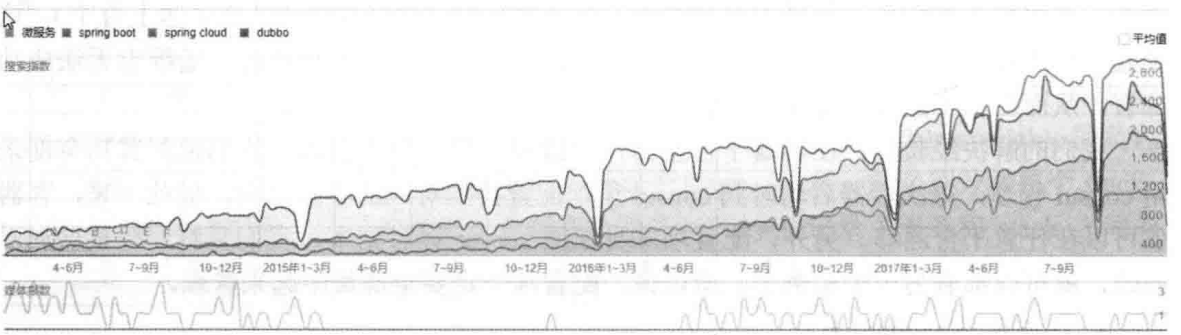

图 1-6 微服务、Spring Boot、Spring Cloud、Dubbo 趋势图

从图 1-6 可见，Dubbo 的搜索量增势放缓，Spring Boot 从 2016 年中下旬开始发力，一路高涨。学习了 Spring Boot 再学习 Spring Cloud 几乎顺理成章。

Spring Boot 旨在解决 Spring 越来越臃肿的全家桶方案的配置问题(具有讽刺意义的是，Spring 刚出道时高举轻量化解决方案大旗一路冲杀，现在自己也开始慢慢“胖”起来了)，提供了很多简单易用的 Starter。特点是预定大于配置。

Dubbo 放缓是由于阿里巴巴中间断更近三年（dubbo-2.4.11 2014-10-30，dubbo-2.5.4 2017-09-07），很多框架和技术都较为陈旧，也不接纳社区的更新（当然，最近开始恢复更新，后面会有说到），导致当当另起炉灶，更新了：https://github.com/ dangdangdotcom/ dubbox（现在已断更）。而且 Dubbo 仅相当于 Spring Cloud 的一个子集，可参考文章《微服务架构的基础框架选择：Spring Cloud 还是 Dubbo？》（文章可参考链接：http://blog.csdn.net/ kobejayandy/ article/details/52078275）。

另外，我们可以看看 K8s（Kubernetes 的简称）、Kubernetes、Docker 的搜索趋势，如图 1-7 所示。