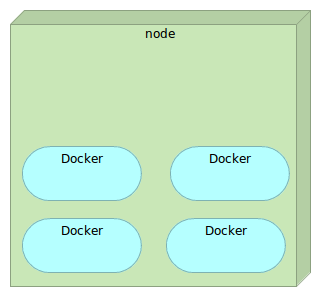
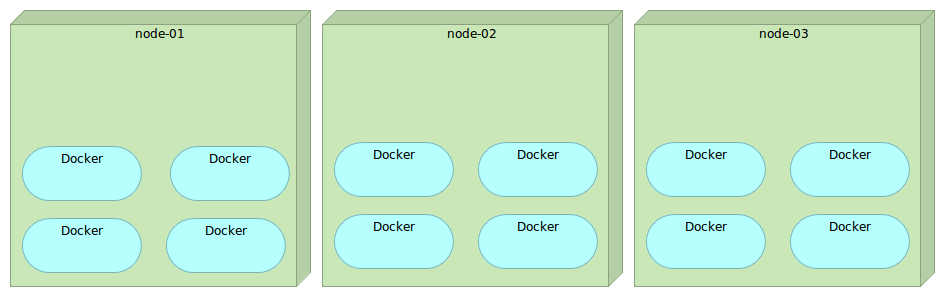
服务发现：Zookeeper vs etcd vs Consul

【编者的话】本文对比了Zookeeper、etcd和Consul三种服务发现工具，探讨了最佳的服务发现解决方案，仅供参考。  
  
如果使用预定义的端口，服务越多，发生冲突的可能性越大，毕竟，不可能有两个服务监听同一个端口。管理一个拥挤的比方说被几百个服务所使用的所有端口的列表，本身就是一个挑战，添加到该列表后，这些服务需要的数据库和数量会日益增多。因此我们应该部署无需指定端口的服务，并且让[Docker](https://www.docker.com/)为我们分配一个随机的端口。唯一的问题是我们需要发现端口号，并且让别人知道。

[](http://dockerone.com/uploads/article/20150913/e31fd491a376d3398fd4ca2dfcc98a9c.png)

当我们开始在一个分布式系统上部署服务到其中一台服务器上时，事情会变得更加复杂，我们可以选择预先定义哪台服务器运行哪个服务的方式，但这会导致很多问题。我们应该尽我们所能尽量利用服务器资源，但是如果预先定义每个服务的部署位置，那么要实现尽量利用服务器资源是几乎不可能的。另一个问题是服务的自动伸缩将会非常困难，更不用说自动恢复了，比方说服务器故障。另一方面，如果我们将服务部署到某台只有最少数量的容器在运行的服务器上，我们需要添加IP地址到数据列表中，这些数据需要可以被发现并存储在某处。

[](http://dockerone.com/uploads/article/20150913/db251f52fbdf63869f789ec3b6eee067.png)

当我们需要存储和发现一些与正在工作的服务相关的信息时，还有很多其他的例子。  
  
为了能够定位服务，我们需要至少接下来的两个有用的步骤。

* **服务注册**——该步骤存储的信息至少包括正在运行的服务的主机和端口信息
* **服务发现**——该步骤允许其他用户可以发现在服务注册阶段存储的信息。

除了上述的步骤，我们还需要考虑其他方面。如果一个服务停止工作并部署/注册了一个新的服务实例，那么该服务是否应该注销呢？当有相同服务的多个副本时咋办？我们该如何做负载均衡呢？如果一个服务器宕机了咋办？所有这些问题都与注册和发现阶段紧密关联。现在，我们限定只在服务发现的范围里（常见的名字，围绕上述步骤）以及用于服务发现任务的工具，它们中的大多数采用了高可用的分布式键/值存储。

服务发现工具

服务发现工具的主要目标是用来服务查找和相互对话，为此该工具需要知道每个服务，这不是一个新概念，在Docker之前就已经存在很多类似的工具了，然而，容器带给了这些工具一个全新水平的需求。  
  
服务发现背后的基本思想是对于服务的每一个新实例（或应用程序），能够识别当前环境和存储相关信息。存储的注册表信息本身通常采用键/值对的格式，由于服务发现经常用于分布式系统，所以要求这些信息可伸缩、支持容错和分布式集群中的所有节点。这种存储的主要用途是给所有感兴趣的各方提供最起码诸如服务IP地址和端口这样的信息，用于它们之间的相互通讯，这些数据还经常扩展到其它类型的信息服务发现工具倾向于提供某种形式的API，用于服务自身的注册以及服务信息的查找。  
  
比方说我们有两个服务，一个是提供方，另一个是第一个服务的消费者，一旦部署了服务提供方，就需要在服务发现注册表中存储其信息。接着，当消费者试图访问服务提供者时，它首先查询服务注册表，使用获取到的IP地址和端口来调用服务提供者。为了与注册表中的服务提供方的具体实现解耦，我们常常采用某种代理服务。这样消费者总是向固定IP地址的代理请求信息，代理再依次使用服务发现来查找服务提供方信息并重定向请求，在本文中我们稍后通过反向代理来实现。现在重要的是要理解基于三种角色（服务消费者、提供者和代理）的服务发现流程。  
  
服务发现工具要查找的是数据，至少我们应该能够找出服务在哪里？服务是否健康和可用？配置是什么样的？既然我们正在多台服务器上构建一个分布式系统，那么该工具需要足够健壮，保证其中一个节点的宕机不会危及数据，同时，每个节点应该有完全相同的数据副本，进一步地，我们希望能够以任何顺序启动服务、杀死服务或者替换服务的新版本，我们还应该能够重新配置服务并且查看到数据相应的变化。  
  
让我们看一下一些常用的选项来完成我们上面设定的目标。

手动配置

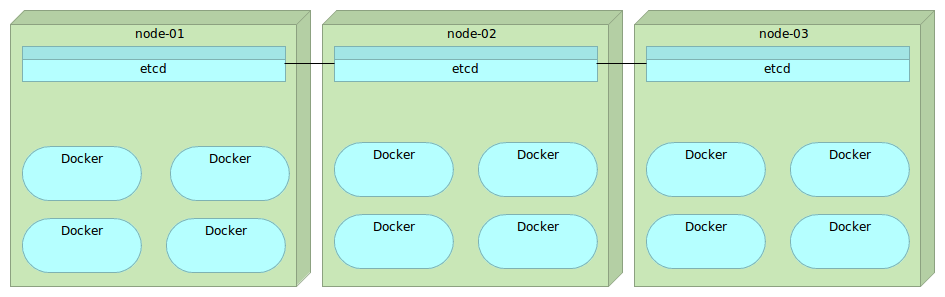
大多数服务仍然是需要手动管理的，我们预先决定在何处部署服务、如何配置和希望不管什么原因，服务都将继续正常工作，直到天荒地老。这样的目标不是可以轻易达到的。部署第二个服务实例意味着我们需要启动全程的手动处理，我们需要引入一台新的服务器，或者找出哪一台服务器资源利用率较低，然后创建一个新的配置集并启动服务。情况或许会变得越来越复杂，比方说，硬件故障导致的手动管理下的反应时间变得很慢。可见性是另外一个痛点，我们知道什么是静态配置，毕竟是我们预先准备好的，然而，大多数的服务有很多动态生成的信息，这些信息不是轻易可见的，也没有一个单独的地方供我们在需要时参考这些数据。  
  
反应时间会不可避免的变慢，鉴于存在许多需要手动处理的移动组件，故障恢复和监控也会变得非常难以管理。  
  
尽管在过去或者当服务/服务器数量很少的时候有借口不做这项工作，随着服务发现工具的出现，这个借口已经不存在了。

Zookeeper

[Zookeeper](http://zookeeper.apache.org/)是这种类型的项目中历史最悠久的之一，它起源于Hadoop，帮助在Hadoop集群中维护各种组件。它非常成熟、可靠，被许多大公司（YouTube、eBay、雅虎等）使用。其数据存储的格式类似于文件系统，如果运行在一个服务器集群中，Zookeper将跨所有节点共享配置状态，每个集群选举一个领袖，客户端可以连接到任何一台服务器获取数据。  
  
Zookeeper的主要优势是其成熟、健壮以及丰富的特性，然而，它也有自己的缺点，其中采用Java开发以及复杂性是罪魁祸首。尽管Java在许多方面非常伟大，然后对于这种类型的工作还是太沉重了，Zookeeper使用Java以及相当数量的依赖使其对于资源竞争非常饥渴。因为上述的这些问题，Zookeeper变得非常复杂，维护它需要比我们期望从这种类型的应用程序中获得的收益更多的知识。这部分地是由于丰富的特性反而将其从优势转变为累赘。应用程序的特性功能越多，就会有越大的可能性不需要这些特性，因此，我们最终将会为这些不需要的特性付出复杂度方面的代价。  
  
Zookeeper为其他项目相当大的改进铺平了道路，“大数据玩家“在使用它，因为没有更好的选择。今天，Zookeeper已经老态龙钟了，我们有了更好的选择。

etcd

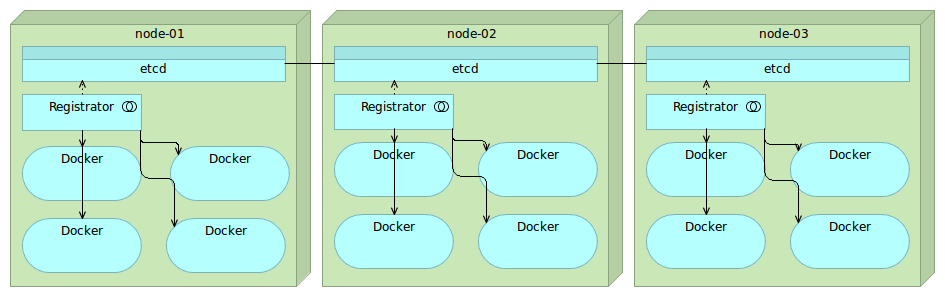
[etcd](https://github.com/coreos/etcd)是一个采用HTTP协议的健/值对存储系统，它是一个分布式和功能层次配置系统，可用于构建服务发现系统。其很容易部署、安装和使用，提供了可靠的数据持久化特性。它是安全的并且文档也十分齐全。  
  
etcd比Zookeeper是比更好的选择，因为它很简单，然而，它需要搭配一些第三方工具才可以提供服务发现功能。

[](http://dockerone.com/uploads/article/20150913/0234b2c789e6432d8c292f209dd96ff7.png)

现在，我们有一个地方来存储服务相关信息，我们还需要一个工具可以自动发送信息给etcd。但在这之后，为什么我们还需要手动把数据发送给etcd呢？即使我们希望手动将信息发送给etcd，我们通常情况下也不会知道是什么信息。记住这一点，服务可能会被部署到一台运行最少数量容器的服务器上，并且随机分配一个端口。理想情况下，这个工具应该监视所有节点上的Docker容器，并且每当有新容器运行或者现有的一个容器停止的时候更新etcd，其中的一个可以帮助我们达成目标的工具就是Registrator。

Registrator

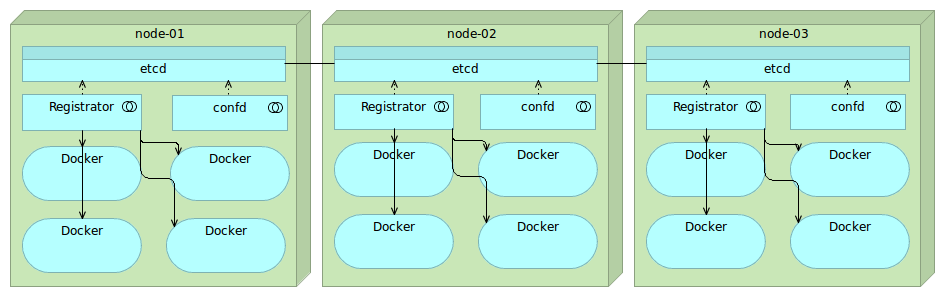
[Registrator](https://github.com/gliderlabs/registrator)通过检查容器在线或者停止运行状态自动注册和去注册服务，它目前支持etcd、Consul和SkyDNS 2。  
  
Registrator与etcd是一个简单但是功能强大的组合，可以运行很多先进的技术。每当我们打开一个容器，所有数据将被存储在etcd并传播到集群中的所有节点。我们将决定什么信息是我们的。

[](http://dockerone.com/uploads/article/20150913/9055759a2146bf31ed881ac9b2a0dcd1.png)

上述的拼图游戏还缺少一块，我们需要一种方法来创建配置文件，与数据都存储在etcd，通过运行一些命令来创建这些配置文件。

Confd

[Confd](https://github.com/kelseyhightower/confd)是一个轻量级的配置管理工具，常见的用法是通过使用存储在etcd、consul和其他一些数据登记处的数据保持配置文件的最新状态，它也可以用来在配置文件改变时重新加载应用程序。换句话说，我们可以用存储在etcd（或者其他注册中心）的信息来重新配置所有服务。

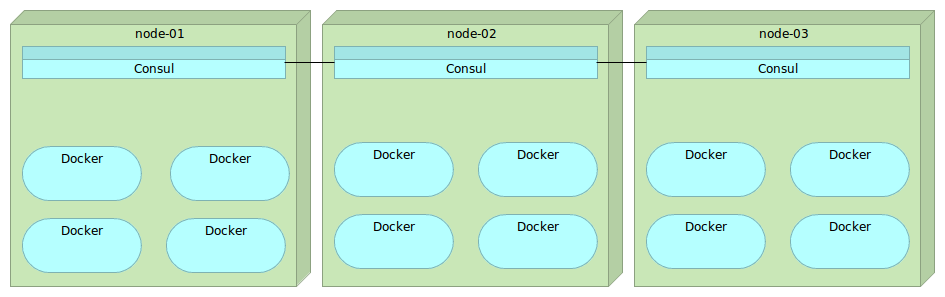
[](http://dockerone.com/uploads/article/20150913/fad23ce3906daaa0378a7b324be360d8.png)

对于etcd、Registrator和Confd组合的最后的思考

当etcd、Registrator和Confd结合时，可以获得一个简单而强大的方法来自动化操作我们所有的服务发现和需要的配置。这个组合还展示了“小”工具正确组合的有效性，这三个小东西可以如我们所愿正好完成我们需要达到的目标，若范围稍微小一些，我们将无法完成我们面前的目标，而另一方面如果他们设计时考虑到更大的范围，我们将引入不必要的复杂性和服务器资源开销。  
  
在我们做出最后的判决之前，让我们看看另一个有相同目标的工具组合，毕竟，我们不应该满足于一些没有可替代方案的选择。

Consul

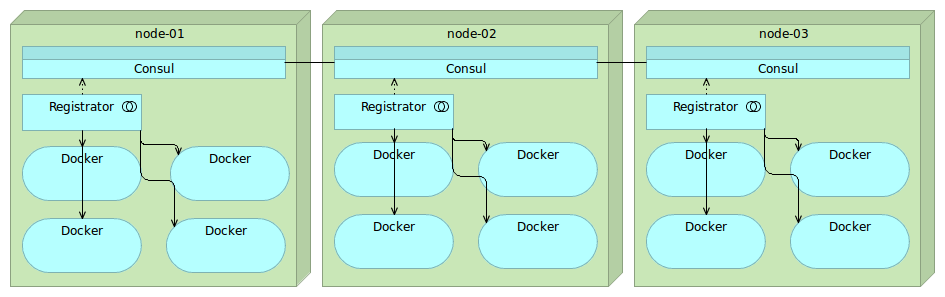
[Consul](https://www.consul.io/)是强一致性的数据存储，使用gossip形成动态集群。它提供分级键/值存储方式，不仅可以存储数据，而且可以用于注册器件事各种任务，从发送数据改变通知到运行健康检查和自定义命令，具体如何取决于它们的输出。  
  
与Zookeeper和etcd不一样，Consul内嵌实现了服务发现系统，所以这样就不需要构建自己的系统或使用第三方系统。这一发现系统除了上述提到的特性之外，还包括节点健康检查和运行在其上的服务。  
  
Zookeeper和etcd只提供原始的键/值队存储，要求应用程序开发人员构建他们自己的系统提供服务发现功能。而Consul提供了一个内置的服务发现的框架。客户只需要注册服务并通过DNS或HTTP接口执行服务发现。其他两个工具需要一个亲手制作的解决方案或借助于第三方工具。  
  
Consul为多种数据中心提供了开箱即用的原生支持，其中的gossip系统不仅可以工作在同一集群内部的各个节点，而且还可以跨数据中心工作。

[](http://dockerone.com/uploads/article/20150913/3ce39e0e0c2220366173c81e29c37278.png)

Consul还有另一个不错的区别于其他工具的功能，它不仅可以用来发现已部署的服务以及其驻留的节点信息，还通过HTTP请求、TTLs（time-to-live）和自定义命令提供了易于扩展的健康检查特性。

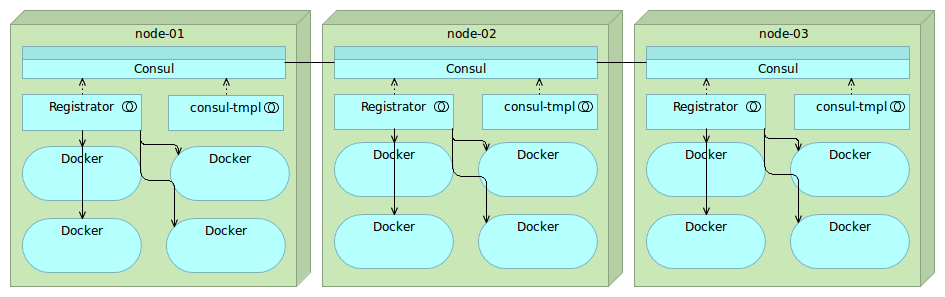
Registrator

[Registrator](https://github.com/gliderlabs/registrator)有两个Consul协议，其中consulkv协议产生类似于etcd协议的结果。  
  
除了通常的IP和端口存储在etcd或consulkv协议中之外，Registrator consul协议存储了更多的信息，我们可以得到服务运行节点的信息，以及服务ID和名称。我们也可以借助于一些额外的环境变量按照一定的标记存储额外的信息。

[](http://dockerone.com/uploads/article/20150913/2f5c16f99b619ab52a6abbb8488e9c68.png)

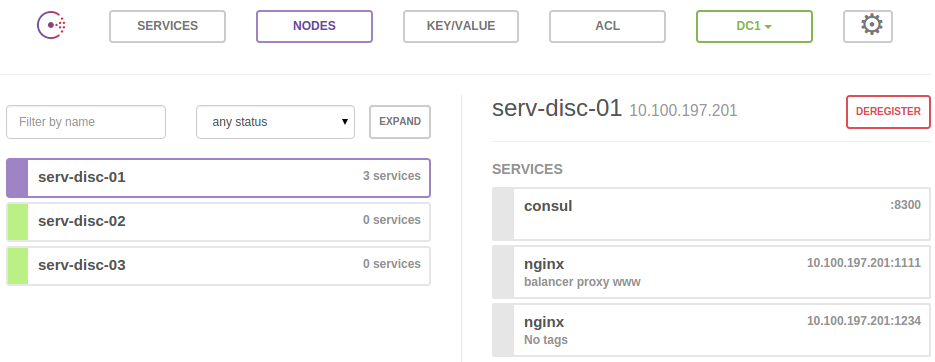
Consul-template

confd可以像和etce搭配一样用于Consul，不过Consul有自己的模板服务，其更适配Consul。  
  
通过从Consul获得的信息，[Consul-template](https://github.com/hashicorp/consul-template)是一个非常方便的创建文件的途径，还有一个额外的好处就是在文件更新后可以运行任意命令，正如confd，Consul-template也可以使用[Go模板](http://golang.org/pkg/text/template/)格式。

[](http://dockerone.com/uploads/article/20150913/8540c6a4a6cb7d32dcb47a6b8ac2c3ac.png)

Consul健康检查、Web界面和数据中心

监控集群节点和服务的健康状态与测试和部署它们一样的重要。虽然我们应该向着拥有从来没有故障的稳定的环境努力，但我们也应该承认，随时会有意想不到的故障发生，时刻准备着采取相应的措施。例如我们可以监控内存使用情况，如果达到一定的阈值，那么迁移一些服务到集群中的另外一个节点，这将是在发生“灾难”前执行的一个预防措施。另一方面，并不是所有潜在的故障都可以被及时检测到并采取措施。单个服务可能会齿白，一个完整的节点也可能由于硬件故障而停止工作。在这种情况下我们应该准备尽快行动，例如一个节点替换为一个新的并迁移失败的服务。Consul有一个简单的、优雅的但功能强大的方式进行健康检查，当健康阀值达到一定数目时，帮助用户定义应该执行的操作。  
  
如果用户Google搜索“etcd ui”或者“etec dashboard”时，用户可能看到只有几个可用的解决方案，可能会问为什么我们还没有介绍给用户，这个原因很简单，etcd只是键/值对存储，仅此而已。通过一个UI呈现数据没有太多的用处，因为我们可以很容易地通过etcdctl获得这些数据。这并不意味着etcd UI是无用的，但鉴于其有限的使用范围，它不会产生多大影响。  
  
Consu不仅仅是一个简单的键/值对存储，正如我们已经看到的，除了存储简单的键/值对，它还有一个服务的概念以及所属的数据。它还可以执行健康检查，因此成为一个好的候选dashboard，在上面可以看到我们的节点的状态和运行的服务。最后，它支持了多数据中心的概念。所有这些特性的结合让我们从不同的角度看到引入dashboard的必要性。  
  
通过Consul Web界面，用户可以查看所有的服务和节点、监控健康检查状态以及通过切换数据中心读取设置键/值对数据。

[](http://dockerone.com/uploads/article/20150913/730398a0827ea6fac6b0d43397d78aea.png)

对于Consul、Registrator、Template、健康检查和Web UI的最终思考

Consul以及上述我们一起探讨的工具在很多情况下提供了比etcd更好的解决方案。这是从内心深处为了服务架构和发现而设计的方案，简单而强大。它提供了一个完整的同时不失简洁的解决方案，在许多情况下，这是最佳的服务发现以及满足健康检查需求的工具。

结论

所有这些工具都是基于相似的原则和架构，它们在节点上运行，需要仲裁来运行，并且都是强一致性的，都提供某种形式的键/值对存储。  
  
**Zookeeper**是其中最老态龙钟的一个，使用年限显示出了其复杂性、资源利用和尽力达成的目标，它是为了与我们评估的其他工具所处的不同时代而设计的（即使它不是老得太多）。  
  
**etcd**、**Registrator**和**Confd**是一个非常简单但非常强大的组合，可以解决大部分问题，如果不是全部满足服务发现需要的话。它还展示了我们可以通过组合非常简单和特定的工具来获得强大的服务发现能力，它们中的每一个都执行一个非常具体的任务，通过精心设计的API进行通讯，具备相对自治工作的能力，从架构和功能途径方面都是**微服务**方式。  
  
**Consul**的不同之处在于无需第三方工具就可以原生支持多数据中心和健康检查，这并不意味着使用第三方工具不好。实际上，在这篇博客里我们通过选择那些表现更佳同时不会引入不必要的功能的的工具，尽力组合不同的工具。使用正确的工具可以获得最好的结果。如果工具引入了工作不需要的特性，那么工作效率反而会下降，另一方面，如果工具没有提供工作所需要的特性也是没有用的。Consul很好地权衡了权重，用尽量少的东西很好的达成了目标。  
  
Consul使用gossip来传播集群信息的方式，使其比etcd更易于搭建，特别是对于大的数据中心。将存储数据作为服务的能力使其比etcd仅仅只有健/值对存储的特性更加完整、更有用（即使Consul也有该选项）。虽然我们可以在etcd中通过插入多个键来达成相同的目标，Consul的服务实现了一个更紧凑的结果，通常只需要一次查询就可以获得与服务相关的所有数据。除此之外，Registrator很好地实现了Consul的两个协议，使其合二为一，特别是添加Consul-template到了拼图中。Consul的Web UI更是锦上添花般地提供了服务和健康检查的可视化途径。  
  
我不能说Consul是一个明确的赢家，而是与etcd相比其有一个轻微的优势。服务发现作为一个概念，以及作为工具都很新，我们可以期待在这一领域会有许多的变化。秉承开放的心态，大家可以对本文的建议持保留态度，尝试不同的工具然后做出自己的结论。  
  
**原文链接：**[**Service Discovery: Zookeeper vs etcd vs Consul**](http://technologyconversations.com/2015/09/08/service-discovery-zookeeper-vs-etcd-vs-consul/)**（翻译：胡震）**