# 生产环境使用K8s一年后，我们总结了这些经验教训

**从容器和容器编排工具开始**

我们相信容器会是未来最主流的部署格式，这项技术让应用封装变得简单了许多。类似于Docker之类的工具提供了实际的容器，但我们还需要复制、故障排除等工具，以及可实现自动部署到多台机器的API，好让容器技术发挥出最大的作用。

在2015年初，Kubernetes、Docker Swarm等集成工具还不成熟，仅提供alpha版本。不过我们还是决定试一试，并选择了Docker Swarm。

我们用Docker Swarm来处理网络，利用Ambassador模式和一组脚本来实现自动化部署。这种方式难度之大，超乎想象，我们也因此收获了第一个教训：**容器集成、网络、部署自动化是非常棘手的问题**。

好在我们很快意识到了这一点，并决定将筹码压在Kubernetes身上——一款由Google、Red Hat、Core OS及一些对大规模部署颇有见地的组织提供支持的集成工具。

**通过Kubernetes实现负载均衡**

使用Kubernetes，需要对pods、services、replication controller等概念了然于心。好消息是，包括Kubernetes官方文档在内，网络上有海量资源，可以帮助你快速上手。

成功建立并运行Kubernetes集群后，即可通过kubectl（Kubernetes CLI）部署应用了。然而当我们想要自动化部署时，却发现光有kubectl是不够的。不过，在此之前我们需要解决另一个问题：**如何从Internet访问部署的应用**？

部署前的服务有一个IP地址，但这个地址仅在Kubernetes集群中可用。这意味着无法通过网络访问该服务！在Google Cloud Engine上运行时，Kubernetes会自动配置一个负载均衡用以访问应用；如果不在Google Cloud Engine上运行（比如我们），那就需要做一些额外的工作来获得负载均衡了。

直接在主机端口上开放服务是一个可行的解决方案（很多人一开始的确是这么做的），但我们发现，这样的做法等于放弃了Kubernetes所提供的许多好处。如果我们依赖主机上的端口，部署多个应用时会遇到端口冲突。另外这样的做法会加大扩展集群和更换主机的难度。

**二级负载均衡器配置**

我们发现，解决以上问题的更好办法，是在Kubernetes集群前配置负载均衡器，例如HAProxy或者NGINX。于是我们开始在AWS上的VPN中运行Kubernetes集群，并使用AWS ELB将外部web流量路由到内部HAProxy集群。HAProxy为每个Kubernetes服务配置了“后端”，以便将流量交换到各个pods。

这种“二级负载均衡器配置”主要也是为了适应AWS ELB相当有限的配置选项。其中一个限制是，它不能处理多个vhosts。这也是我们同时使用HAProxy的原因。只使用HAProxy（不用ELB）也能工作，只不过需要你在DNS级别解决动态AWS IP地址问题。

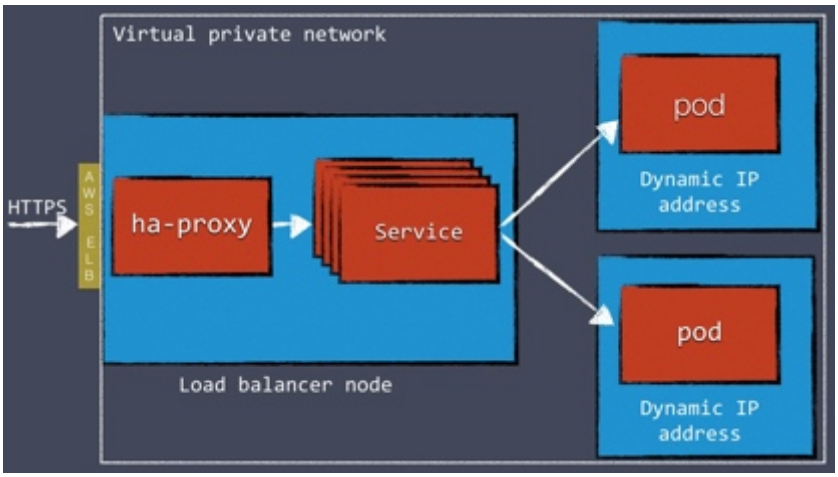

图1:我们的“二级负载均衡器配置流程“

在任何情况下，创建新的Kubernetes服务，我们都需要一种机制动态重新配置负载均衡器（在我们的例子中是HAProxy）。

Kubernetes社区目前正在开发一个名为**ingress**的功能，用来直接从Kubernetes配置外部负载均衡器。可惜的是，目前开发工作还未完成。过去一年，我们采用的是API配合一个小的开源工具来配置负载均衡。

**配置负载均衡**

首先，我们需要一个地方存储负载均衡器配置。负载均衡器配置可以存储在任何地方，不过因为我们已经有etcd可用，就把这些配置放在了etcd里。我们使用一个名为“confd”的工具观察etcd中的配置变动，并用模板生成了一个新的HAProxy配置文件。当一个新的服务添加到Kubernetes，我们向etcd中添加一个新的配置，一个新的HAProxy配置文件也就此产生。

**不断完善的Kubernetes**

Kubernetes中仍然存在众多待解决的问题，很多开发者在社区上、设计文档中讨论解决这些问题的新功能，但开发适用于每一个人的解决方案是需要时间的。不过从长远来看，这也是一件好事，用shortcut的方式设计新功能很多时候会适得其反。

当然，问题的存在并不意味着我们现在所使用的Kubernetes功能有限。配合API，Kubernetes几乎可以完成你想要的一切。等Kubernetes增加新功能后，我们再用标准方案替代自己的解决方案不迟。

话说回来，在我们开发了用于负载均衡的解决方案后，另一项挑战接踵而至：**蓝绿部署**（Blue-green deployments）。

**Kubernetes上的蓝绿部署**

蓝绿部署是一种不中断服务的部署。与滚动更新不同，蓝绿部署在旧版本仍然正常工作的情况下，通过启用一个运行着新版本的副本集群来实现更新。当新版本的副本集群完全启动并运行时，负载均衡器配置才会更改并将负载切换到新版本上。这种方式的好处是，永远保持至少一个版本的应用正常运行，减少了处理多个并发版本带来的复杂性。蓝绿部署在副本数量很少时也能很好的工作。

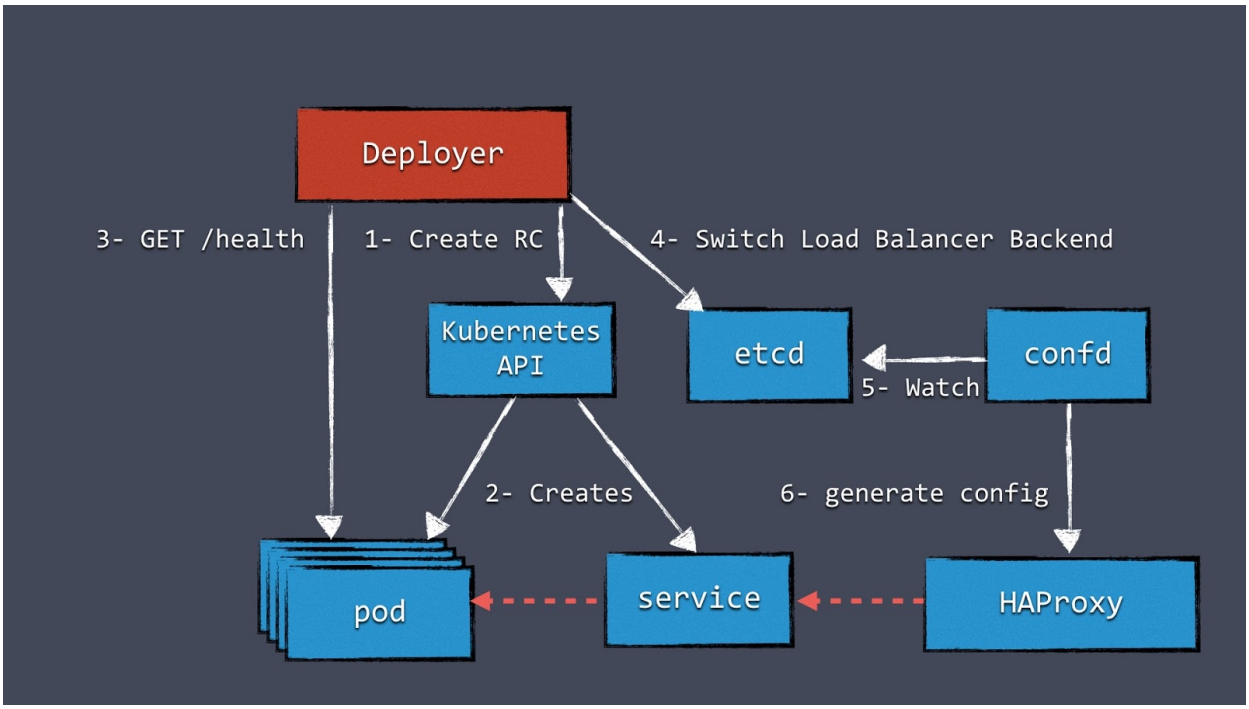


图2:Kubernetes下的蓝绿部署

图2展示了“Deployer“如何编排部署。基于Apache License，并作为Amdatu umbrella project的一部分，我们开源了这个组件的实现，供大家参考使用。另外，这个组件带web UI，可以用来配置部署。这种机制的一个要点是在重新配置负载均衡器之前，执行在pods上的运行状态检查。我们希望每部署的每一个组件都能提供状态检查。目前的做法通常是为每个组件添加一个通过HTTP访问的状态检查。

**实现部署自动化**

有了Deployer，我们就可以把部署集成到构建流程中了。我们的构建服务器可以在构建成功之后，将新的镜像推送到registry（如Git Hub），而后构建服务器可以调用新版本应用并自动部署至测试环境中。同一个镜像也可以通过触发生产环境中的Deployer被推送上生产。

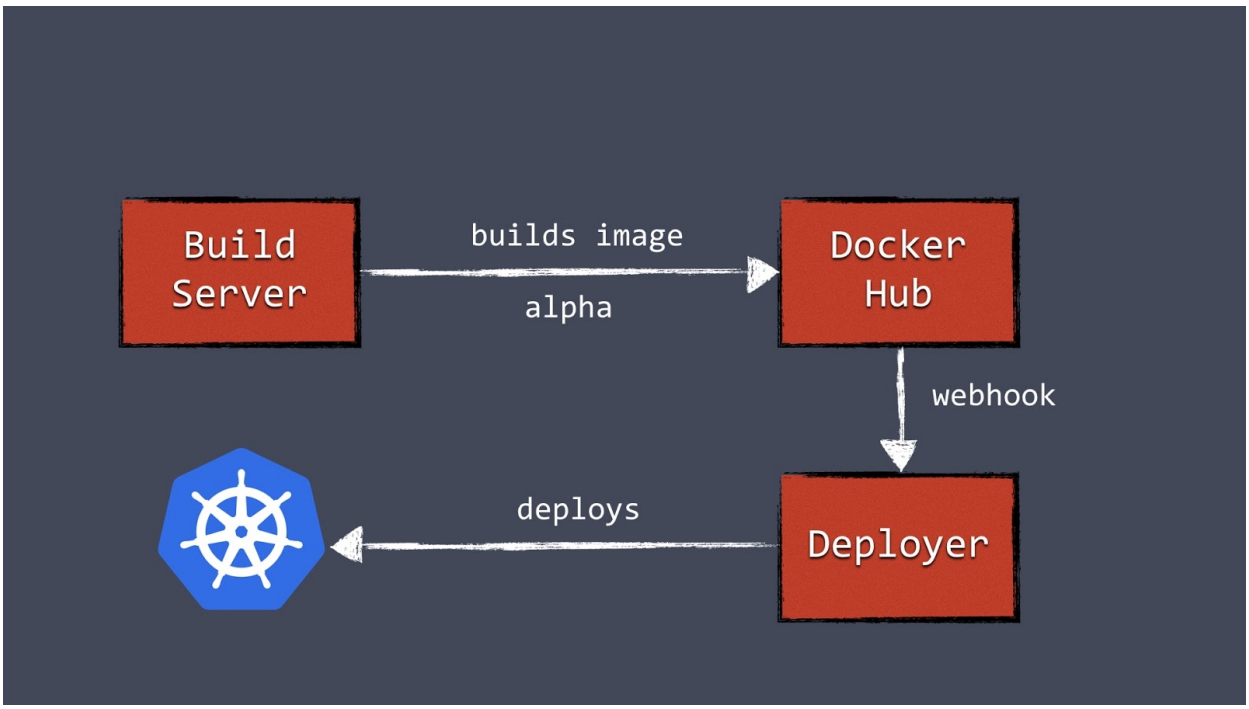


图3：容器化自动部署流程

**资源限制**

使用Kubernetes时，搞清楚资源限制很重要。你可以在每个pod上配置资源请求和CPU / 内存限制，也可以控制资源保证和bursting limits。

这些设置对于高效运行多个容器极为重要，防止容器因分配内存不足而意外停止。

建议尽早设置和测试资源限制。没有限制时，看起来运行良好，不代表把重要负载放到容器中不会出现问题。

**Kubernetes监控**

当我们快要搭建好Kubernetes时，我们意识到监控和日志在这个新的动态环境中非常重要。当我们面对大规模服务和节点时，进入服务器查看日志文件是不现实的，建一个中心化的日志和监控系统需要尽快提上议程。

**日志**

有很多用于日志记录的开源工具，我们使用的是Graylog和Apache Kafka（从容器收集摘录日志的消息传递系统）。容器将日志发送给Kafka，Kafka交给Graylog进行索引。我们让应用组件将日志打给Kafka，方便将日志流式化，成为易于索引的格式。也有一些工具可以从容器外收集日志，然后发送给日志系统。

**监控**

Kubernetes具备超强的**故障恢复机制**，Kubernetes会重启意外停止的pod。我们曾遇到过因内存泄露，导致容器在一天内宕机多次的情况，然而令人惊讶的是，甚至我们自己都没有察觉到。

Kubernetes在这一点上做得非常好，不过一旦问题出现，即使被及时处理了，我们还是想要知道。因此我们使用了一个自定义的运行状况检查仪表盘来监控Kubernetes节点和pod，以及包括数据存储在内的一些服务。可以说在监控方面，Kubernetes API再次证明了其价值所在。

检测负载、吞吐量、应用错误等状态也是同样重要的，有很多开源软件可以满足这一需求。我们的应用组件将指标发布到InfluxDB，并用Heapster去收集Kubernetes的指标。我们还通过Grafana（一款开源仪表盘工具）使存储在InfluxDB中的指标可视化。有很多InfluxDB / Grafana堆栈的替代方案，无论你才用哪一种，对于运行情况的跟踪和监控都是非常有价值的。

**数据存储和Kubernetes**

很多Kubernetes新用户都有一个问题：**我该如何使用Kubernetes处理数据**?

运行数据存储时（如MangoDB或MySQL），我们很可能会有持久化数据储存的需求。不过容器一但重启，所有数据都会丢失，这对于无状态组件没什么影响，但对持久化数据储存显然行不通。好在，Kubernetes具有Volume机制。

Volume可以通过多种方式备份，包括主机文件系统、EBS（AWS的Elastic Block Store）和nfs等。当我们研究持久数据问题是，这是一个很好的方案，但不是我们运行数据存储的答案。

**副本问题**

在大多数部署中，数据存储也是有副本的。Mongo通常在副本集中运行，而MySQL可以在主 / 副模式下运行。我们都知道数据储存集群中的每个节点应该备份在不同的volume中，写入相同volume会导致

数据损坏。另外，大多数数据存储需要明确配置才能使集群启动并运行，自动发现和配置节点并不常见。

同时，运行着数据存储的机器通常会针对某项工作负载进行调整，例如更高的IOPS。扩展（添加/删除节点）对于数据存储来说，也是一个昂贵的操作，这些都和Kubernetes部署的动态本质不相符。

**决定不在生产环境数据存储上使用Kubernetes**

以我们的情况，在Kubernetes内运行数据存储没有想象中那么完美，设置起来也比其他Kubernetes部署复杂得多。

于是我们决定不在生产环境数据存储上使用Kubernetes，而是选择在不同的机器上手动启动这些集群，我们在Kubernetes内部运行的应用正常连接到数据存储集群。

所以说，**没必要运行Kubernetes的一切**，按自身情况与其他工具配合使用，会有意想不到的效果，比如我们的数据存储和HAProxy服务器。

**未来**

看看我们现在的部署，可以负责任的说，Kubernetes的表现绝对是梦幻级的，而Kubernetes API更是自动化部署流程的利器。由于不需要处理VM，我们现在的部署相比之前更快、更可靠。更简单的容器测试和交付，也让我们在构建和部署可靠性上得到了巨大提升。

这种新的部署方式迅速高效，让我们得以跟上其他团队的节奏，这绝对是必要的。

**成本计算**

任何事情都有两面性。运行Kubernetes，需要一个etcd集群以及一个Master节点，对于较小的部署来说，这一开销还是比较大的，适合通过一些云服务达成。

对于大规模部署，Kubernetes可以帮助节省大量服务器成本，etcd集群和Master节点这点开销就显得微不足道了。Kubernetes让很多容器在一个主机上运行变得非常容易，最大程度上利用了现有资源，减少了服务器数量，成本自然下降了。不过这样的集群也给运维工作提出了更高的要求，必须要的时候，我们可以选择一些云计算平台提供的云服务来轻松达成。

Author