10 | Kubernetes一键部署利器：kubeadm

10 | Kubernetes一键部署利器：kubeadm

张磊 2018-09-14

17:20

讲述：张磊 大小：7.95M

你好，我是张磊。今天我和你分享的主题是：Kubernetes 一键部署利器之 kubeadm。

通过前面几篇文章的内容，我其实阐述了这样一个思想：**要真正发挥容器技术的实力，你就不能仅仅局限于对 Linux 容器本身的钻研和使用。**

这些知识更适合作为你的技术储备，以便在需要的时候可以帮你更快的定位问题，并解决问题。

而更深入的学习容器技术的关键在于，**如何使用这些技术来“容器化”你的应用。**

比如，我们的应用既可能是 Java Web 和 MySQL 这样的组合，也可能是 Cassandra 这样的分布式系统。而要使用容器把后者运行起来，你单单通过 Docker 把一个 Cassandra 镜像跑起来是没用的。

要把 Cassandra 应用容器化的关键，在于如何处理好这些 Cassandra 容器之间的编排关系。比如，哪些 Cassandra 容器是主，哪些是从？主从容器如何区分？它们之间又如何进行自动发现和通信？Cassandra 容器的持久化数据又如何保持，等等。

这也是为什么我们要反复强调 Kubernetes 项目的主要原因：这个项目体现出来的容器化“表达能力”，具有独有的先进性和完备性。这就使得它不仅能运行 Java Web 与 MySQL 这样的常规组合，还能够处理 Cassandra 容器集群等复杂编排问题。所以，对这种编排能力的剖析、解读和最佳实践，将是本专栏最重要的一部分内容。

不过，万事开头难。

作为一个典型的分布式项目，Kubernetes 的部署一直以来都是挡在初学者前面的一只“拦路虎”。尤其是在 Kubernetes 项目发布初期，它的部署完全要依靠一堆由社区维护的脚本。

其实，Kubernetes 作为一个 Golang 项目，已经免去了很多类似于 Python 项目要安装语言级别依赖的麻烦。但是，除了将各个组件编译成二进制文件外，用户还要负责为这些二进制文件编写对应的配置文件、配置自启动脚本，以及为 kube-apiserver 配置授权文件等等诸多运维工作。

目前，各大云厂商最常用的部署的方法，是使用 SaltStack、Ansible 等运维工具自动化地执行这些步骤。

但即使这样，这个部署过程依然非常繁琐。因为，SaltStack 这类专业运维工具本身的学习成本，就可能比 Kubernetes 项目还要高。

难道 Kubernetes 项目就没有简单的部署方法了吗?

这个问题，在 Kubernetes 社区里一直没有得到足够重视。直到 2017 年，在志愿者的推动下，社区才终于发起了一个独立的部署工具，名叫：kubeadm。

这个项目的目的，就是要让用户能够通过这样两条指令完成一个 Kubernetes 集群的部署：

```
# 创建一个 Master 节点
$ kubeadm init
```

```
# 将一个 Node 节点加入到当前集群中
$ kubeadm join <Master 节点的 IP 和端口 >
```

复制代码

是不是非常方便呢？

不过，你可能也会有所顾虑：**Kubernetes 的功能那么多，这样一键部署出来的集群，能用于生产环境吗？**

为了回答这个问题，在今天这篇文章，我就先和你介绍一下 kubeadm 的工作原理吧。

kubeadm 的工作原理

在上一篇文章《从容器到容器云：谈谈 Kubernetes 的本质》中，我已经详细介绍了 Kubernetes 的架构和它的组件。在部署时，它的每一个组件都是一个需要被执行的、单独的二进制文件。所以不难想象，SaltStack 这样的运维工具或者由社区维护的脚本的功能，就是要把这些二进制文件传输到指定的机器当中，然后编写控制脚本来启停这些组件。

不过，在理解了容器技术之后，你可能已经萌生出了这样一个想法，**为什么不用容器部署 Kubernetes 呢？**

这样，我只要给每个 Kubernetes 组件做一个容器镜像，然后在每台宿主机上用 docker run 指令启动这些组件容器，部署不就完成了吗？

事实上，在 Kubernetes 早期的部署脚本里，确实有一个脚本就是用 Docker 部署 Kubernetes 项目的，这个脚本相比于 SaltStack 等的部署方式，也的确简单了不少。

但是，**这样做会带来一个很麻烦的问题，即：如何容器化 kubelet。**

我在上一篇文章中，已经提到 kubelet 是 Kubernetes 项目用来操作 Docker 等容器运行时的核心组件。可是，除了跟容器运行时打交道外，kubelet 在配置容器网络、管理容器数据卷时，都需要直接操作宿主机。

而如果现在 kubelet 本身就运行在一个容器里，那么直接操作宿主机就会变得很麻烦。对于网络配置来说还好，kubelet 容器可以通过不开启 Network Namespace（即 Docker 的 host network 模式）的方式，直接共享宿主机的网络栈。可是，要让 kubelet 隔着容器的 Mount Namespace 和文件系统，操作宿主机的文件系统，就有点儿困难了。

比如，如果用户想要使用 NFS 做容器的持久化数据卷，那么 kubelet 就需要在容器进行绑定挂载前，在宿主机的指定目录上，先挂载 NFS 的远程目录。

可是，这时候问题来了。由于现在 kubelet 是运行在容器里的，这就意味着它要做的这个“mount -F nfs”命令，被隔离在了一个单独的 Mount Namespace 中。即，kubelet 做的挂载操作，不能被“传播”到宿主机上。

对于这个问题，有人说，可以使用 setns() 系统调用，在宿主机的 Mount Namespace 中执行这些挂载操作；也有人说，应该让 Docker 支持一个–mnt=host 的参数。

但是，到目前为止，在容器里运行 kubelet，依然没有很好的解决办法，我也不推荐你用容器去部署 Kubernetes 项目。

正因为如此，kubeadm 选择了一种妥协方案：

> *把 kubelet 直接运行在宿主机上，然后使用容器部署其他的 Kubernetes 组件。*

所以，你使用 kubeadm 的第一步，是在机器上手动安装 kubeadm、kubelet 和 kubectl 这三个二进制文件。当然，kubeadm 的作者已经为各个发行版的 Linux 准备好了安装包，所以你只需要执行：

```
$ apt-get install kubeadm
```

复制代码
就可以了。

接下来，你就可以使用“kubeadm init”部署 Master 节点了。

kubeadm init 的工作流程

当你执行 kubeadm init 指令后，**kubeadm 首先要做的，是一系列的检查工作，以确定这台机器可以用来部署 Kubernetes**。这一步检查，我们称为“Preflight Checks”，它可以为你省掉很多后续的麻烦。

其实，Preflight Checks 包括了很多方面，比如：

Linux 内核的版本必须是否是 3.10 以上?
Linux Cgroups 模块是否可用?
机器的 hostname 是否标准？在 Kubernetes 项目里，机器的名字以及一切存储在 Etcd 中的 API 对象，都必须使用标准的 DNS 命名（RFC 1123）。
用户安装的 kubeadm 和 kubelet 的版本是否匹配?
机器上是不是已经安装了 Kubernetes 的二进制文件?

```
Kubernetes 的工作端口 10250/10251/10252 端口是不是已经被占用?
ip、mount 等 Linux 指令是否存在?
Docker 是否已经安装?
……
```

在通过了 Preflight Checks 之后，kubeadm 要为你做的，是生成 Kubernetes 对外提供服务所需的各种证书和对应的目录。

Kubernetes 对外提供服务时，除非专门开启“不安全模式”，否则都要通过 HTTPS 才能访问 kube-apiserver。这就需要为 Kubernetes 集群配置好证书文件。

kubeadm 为 Kubernetes 项目生成的证书文件都放在 Master 节点的 /etc/kubernetes/pki 目录下。在这个目录下，最主要的证书文件是 ca.crt 和对应的私钥 ca.key。

此外，用户使用 kubectl 获取容器日志等 streaming 操作时，需要通过 kube-apiserver 向 kubelet 发起请求，这个连接也必须是安全的。kubeadm 为这一步生成的是 apiserver-kubelet-client.crt 文件，对应的私钥是 apiserver-kubelet-client.key。

除此之外，Kubernetes 集群中还有 Aggregate APIServer 等特性，也需要用到专门的证书，这里我就不再一一列举了。需要指出的是，你可以选择不让 kubeadm 为你生成这些证书，而是拷贝现有的证书到如下证书的目录里：

```
/etc/kubernetes/pki/ca.{crt,key}
```

复制代码

这时，kubeadm 就会跳过证书生成的步骤，把它完全交给用户处理。

证书生成后，kubeadm 接下来会为其他组件生成访问 kube-apiserver 所需的配置文件。这些文件的路径是：/etc/kubernetes/xxx.conf：

```
ls /etc/kubernetes/
admin.conf controller-manager.conf kubelet.conf scheduler.conf
```

复制代码

这些文件里面记录的是，当前这个 Master 节点的服务器地址、监听端口、证书目录等信息。这样，对应的客户端（比如 scheduler，kubelet 等），可以直接加载相应的文件，使用里面的信息与 kube-apiserver 建立安全连接。

接下来，kubeadm 会为 Master 组件生成 Pod 配置文件。我已经在上一篇文章中和你介绍过 Kubernetes 有三个 Master 组件 kube-apiserver、kube-

controller-manager、kube-scheduler，而它们都会被使用 Pod 的方式部署起来。

你可能会有些疑问：这时，Kubernetes 集群尚不存在，难道 kubeadm 会直接执行 docker run 来启动这些容器吗?

当然不是。

在 Kubernetes 中，有一种特殊的容器启动方法叫做“Static Pod”。它允许你把要部署的 Pod 的 YAML 文件放在一个指定的目录里。这样，当这台机器上的 kubelet 启动时，它会自动检查这个目录，加载所有的 Pod YAML 文件，然后在这台机器上启动它们。

从这一点也可以看出，kubelet 在 Kubernetes 项目中的地位非常高，在设计上它就是一个完全独立的组件，而其他 Master 组件，则更像是辅助性的系统容器。

在 kubeadm 中，Master 组件的 YAML 文件会被生成在 /etc/kubernetes/manifests 路径下。比如，kube-apiserver.yaml：

```
apiVersion: v1
kind: Pod
metadata:
annotations:
scheduler.alpha.kubernetes.io/critical-pod: ""
creationTimestamp: null
labels:
component: kube-apiserver
tier: control-plane
name: kube-apiserver
namespace: kube-system
spec:
containers:
- command:
- kube-apiserver
- --authorization-mode=Node,RBAC
- --runtime-config=api/all=true
- --advertise-address=10.168.0.2
```

```
...
- --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
- --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
image: k8s.gcr.io/kube-apiserver-amd64:v1.11.1
imagePullPolicy: IfNotPresent
livenessProbe:
...
name: kube-apiserver
resources:
requests:
cpu: 250m
volumeMounts:
- mountPath: /usr/share/ca-certificates
name: usr-share-ca-certificates
readOnly: true
...
hostNetwork: true
priorityClassName: system-cluster-critical
volumes:
- hostPath:
path: /etc/ca-certificates
type: DirectoryOrCreate
name: etc-ca-certificates
...
```

□复制代码

关于一个 Pod 的 YAML 文件怎么写、里面的字段如何解读，我会在后续专门的文章中为你详细分析。在这里，你只需要关注这样几个信息：

1. 这个 Pod 里只定义了一个容器，它使用的镜像是：k8s.gcr.io/kube-apiserver-amd64:v1.11.1 。这个镜像是 Kubernetes 官方维护的一个组件镜像。
2. 这个容器的启动命令（commands）是 kube-apiserver --authorization-mode=Node,RBAC …，这样一句非常长的命令。其实，它就是容器里 kube-apiserver 这个二进制文件再加上指定的配置参数而已。
3. 如果你要修改一个已有集群的 kube-apiserver 的配置，需要修改这个 YAML 文件。
4. 这些组件的参数也可以在部署时指定，我很快就会讲解到。

在这一步完成后，kubeadm 还会再生成一个 Etcd 的 Pod YAML 文件，用来通过同样的 Static Pod 的方式启动 Etcd。所以，最后 Master 组件的 Pod YAML 文件如下所示：

```
$ ls /etc/kubernetes/manifests/
etcd.yaml kube-apiserver.yaml kube-controller-manager.yaml kube-scheduler.yaml
```

复制代码

而一旦这些 YAML 文件出现在被 kubelet 监视的 /etc/kubernetes/manifests 目录下，kubelet 就会自动创建这些 YAML 文件中定义的 Pod，即 Master 组件的容器。

Master 容器启动后，kubeadm 会通过检查 localhost:6443/healthz 这个 Master 组件的健康检查 URL，等待 Master 组件完全运行起来。

然后，kubeadm 就会为集群生成一个 bootstrap token。在后面，只要持有这个 token，任何一个安装了 kubelet 和 kubadm 的节点，都可以通过 kubeadm join 加入到这个集群当中。

这个 token 的值和使用方法会，会在 kubeadm init 结束后被打印出来。

在 token 生成之后，kubeadm 会将 ca.crt 等 Master 节点的重要信息，通过 ConfigMap 的方式保存在 Etcd 当中，供后续部署 Node 节点使用。这个 ConfigMap 的名字是 cluster-info。

kubeadm init 的最后一步，就是安装默认插件。Kubernetes 默认 kube-proxy 和 DNS 这两个插件是必须安装的。它们分别用来提供整个集群的服务发现和 DNS 功能。其实，这两个插件也只是两个容器镜像而已，所以 kubeadm 只要用 Kubernetes 客户端创建两个 Pod 就可以了。

kubeadm join 的工作流程

这个流程其实非常简单，kubeadm init 生成 bootstrap token 之后，你就可以在任意一台安装了 kubelet 和 kubeadm 的机器上执行 kubeadm join 了。

可是，为什么执行 kubeadm join 需要这样一个 token 呢?

因为，任何一台机器想要成为 Kubernetes 集群中的一个节点，就必须在集群的 kube-apiserver 上注册。可是，要想跟 apiserver 打交道，这台机器就必须要获取到相应的证书文件（CA 文件）。可是，为了能够一键安装，我们就不能让用户去 Master 节点上手动拷贝这些文件。

所以，kubeadm 至少需要发起一次“不安全模式”的访问到 kube-apiserver，从而拿到保存在 ConfigMap 中的 cluster-info（它保存了 APIServer 的授权信息）。而 bootstrap token，扮演的就是这个过程中的安全验证的角色。

只要有了 cluster-info 里的 kube-apiserver 的地址、端口、证书，kubelet 就可以以“安全模式”连接到 apiserver 上，这样一个新的节点就部署完成了。

接下来，你只要在其他节点上重复这个指令就可以了。

配置 kubeadm 的部署参数

我在前面讲解了 kubeadm 部署 Kubernetes 集群最关键的两个步骤，kubeadm init 和 kubeadm join。相信你一定会有这样的疑问：kubeadm 确实简单易用，可是我又该如何定制我的集群组件参数呢？

比如，我要指定 kube-apiserver 的启动参数，该怎么办？

在这里，我强烈推荐你在使用 kubeadm init 部署 Master 节点时，使用下面这条指令：

```
$ kubeadm init --config kubeadm.yaml
```

复制代码

这时，你就可以给 kubeadm 提供一个 YAML 文件（比如，kubeadm.yaml），它的内容如下所示（我仅列举了主要部分）：

```
apiVersion: kubeadm.k8s.io/v1alpha2
kind: MasterConfiguration
kubernetesVersion: v1.11.0
api:
advertiseAddress: 192.168.0.102
bindPort: 6443
...
etcd:
local:
dataDir: /var/lib/etcd
image: ""
imageRepository: k8s.gcr.io
kubeProxy:
```

```
config:
bindAddress: 0.0.0.0
...
kubeletConfiguration:
baseConfig:
address: 0.0.0.0
...
networking:
dnsDomain: cluster.local
podSubnet: ""
serviceSubnet: 10.96.0.0/12
nodeRegistration:
criSocket: /var/run/dockershim.sock
...
```

复制代码

通过制定这样一个部署参数配置文件，你就可以很方便地在这个文件里填写各种自定义的部署参数了。比如，我现在要指定 kube-apiserver 的参数，那么我只要在这个文件里加上这样一段信息：

```
...
apiServerExtraArgs:
advertise-address: 192.168.0.103
anonymous-auth: false
enable-admission-plugins: AlwaysPullImages,DefaultStorageClass
audit-log-path: /home/johndoe/audit.log
```

复制代码

然后，kubeadm 就会使用上面这些信息替换 /etc/kubernetes/manifests/kube-apiserver.yaml 里的 command 字段里的参数了。

而这个 YAML 文件提供的可配置项远不止这些。比如，你还可以修改 kubelet 和 kube-proxy 的配置，修改 Kubernetes 使用的基础镜像的 URL（默认的 k8s.gcr.io/xxx镜像 URL 在国内访问是有困难的），指定自己的证书文件，指定特殊的容器运行时等等。这些配置项，就留给你在后续实践中探索了。

总结

在今天的这次分享中，我重点介绍了 kubeadm 这个部署工具的工作原理和使用方法。紧接着，我会在下一篇文章中，使用它一步步地部署一个完整的 Kubernetes 集群。

从今天的分享中，你可以看到，kubeadm 的设计非常简洁。并且，它在实现每一步部署功能时，都在最大程度地重用 Kubernetes 已有的功能，这也就使得我们在使用 kubeadm 部署 Kubernetes 项目时，非常有“原生”的感觉，一点都不会感到突兀。

而 kubeadm 的源代码，直接就在 kubernetes/cmd/kubeadm 目录下，是 Kubernetes 项目的一部分。其中，app/phases 文件夹下的代码，对应的就是我在这篇文章中详细介绍的每一个具体步骤。

看到这里，你可能会猜想，kubeadm 的作者一定是 Google 公司的某个“大神”吧。

实际上，kubeadm 几乎完全是一位高中生的作品。他叫 Lucas Käldström，芬兰人，今年只有 18 岁。kubeadm，是他 17 岁时用业余时间完成的一个社区项目。

所以说，开源社区的魅力也在于此：一个成功的开源项目，总能够吸引到全世界最厉害的贡献者参与其中。尽管参与者的总体水平参差不齐，而且频繁的开源活动又显得杂乱无章难以管控，但一个有足够热度的社区最终的收敛方向，却一定是代码越来越完善、Bug 越来越少、功能越来越强大。

最后，我再来回答一下我在今天这次分享开始提到的问题：kubeadm 能够用于生产环境吗？

到目前为止（2018 年 9 月），这个问题的答案是：不能。

因为 kubeadm 目前最欠缺的是，一键部署一个高可用的 Kubernetes 集群，即：Etcd、Master 组件都应该是多节点集群，而不是现在这样的单点。这，当然也正是 kubeadm 接下来发展的主要方向。

另一方面，Lucas 也正在积极地把 kubeadm phases 开放给用户，即：用户可以更加自由地定制 kubeadm 的每一个部署步骤。这些举措，都可以让这个项目更加完善，我对它的发展走向也充满了信心。

当然，如果你有部署规模化生产环境的需求，我推荐使用kops或者 SaltStack 这样更复杂的部署工具。但，在本专栏接下来的讲解中，我都会以 kubeadm 为依据进行讲述。

> 一方面，作为 Kubernetes 项目的原生部署工具，kubeadm 对 Kubernetes 项目特性的使用和集成，确实要比其他项目“技高一筹”，非常值得我们学习

和借鉴；
另一方面，kubeadm 的部署方法，不会涉及到太多的运维工作，也不需要我们额外学习复杂的部署工具。而它部署的 Kubernetes 集群，跟一个完全使用二进制文件搭建起来的集群几乎没有任何区别。

因此，使用 kubeadm 去部署一个 Kubernetes 集群，对于你理解 Kubernetes 组件的工作方式和架构，最好不过了。

思考题

1. 在 Linux 上为一个类似 kube-apiserver 的 Web Server 制作证书，你知道可以用哪些工具实现吗？
2. 回忆一下我在前面文章中分享的 Kubernetes 架构，你能够说出 Kubernetes 各个功能组件之间（包含 Etcd），都有哪些建立连接或者调用的方式吗？（比如：HTTP/HTTPS，远程调用等等）