## 几种原生kubernetes集群搭建方式的优缺点及区别

1. **二进制包部署/yum源安装**

使用二进制文件安装，优点是可以安装任意版本的Kubernetes，但是配置比较复杂，很多软件包需要各种依赖包，且需要对二进制程序做系统级进程维护。

可从官方下载发行版的二进制包，手动部署每个组件，组成Kubernetes集群。

官方地址：

<https://github.com/kubernetes/kubernetes/releases>

yum安装与二进制部署相比最大的好处就是可以解决自动解决依赖问题，但使用 yum 源安装的 Kubernetes 的版本也会受到限制，通常会低于官方很多版本。这里不列出yum源码安装方式的官方地址，依据操作系统而定，目前CentOS官方已经把Kubernetes源放入到自己的默认 extras 仓库里面。

1. **Kubeadm部署**

kubeadm是 Kubernetes 官方提供的用于快速安装Kubernetes集群的工具，伴随Kubernetes每个版本的发布都会同步更新，kubeadm会对集群配置方面的一些实践做调整。

kubeadm 可以运行在物理机、虚拟机以及云主机，并且可以与chef或puppet配合可以完成自动化部署安装。kubeadm 是 kubernetes 在1.4版推出安装 kubernetes 集群。目前还处于alpha，不建议在生产环境使用。在安装过程中可选的配置参数太少，所以基本需要在使用过程中自行修改。

官方地址：

<https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/>

1. **minikubeMinikube部署**

minikubeMinikube是一个工具，可以在本地快速运行一个单点的Kubernetes，仅用于尝试Kubernetes或日常开发的用户使用。

官方地址：

<https://kubernetes.io/docs/setup/minikube/>

## 高可用下的kubernetes整体集群的配置建议

**1、高可用架构的挑战：**

-应用程序的异常退出

-操作系统的突然宕机

-服务器的意外断电

-运维人员人为操作失误

-地震等不可抵抗因素

-业务量的突然增大

…

**2、高可用性的指标及定义**

高可用指标

|  |  |  |  |
| --- | --- | --- | --- |
| 指标 | 说明 | 含义 | 备注 |
| MTBF | Mean Time Between Failure | 平均故障间隔时间 | 越长越好 |
| MTTR | Mean Time To Repair | 平均修复时间 | 越短越好 |
| MTTF | Mean Time To Failure | 平均故障时间 | 越长越好 |

MTBF = MTTF + MTTR

MTBF =~ MTTF

一般来说达到N个九的高可用性，具体含义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 级别 | 系统可用性比率 | MAX最大可能服务不可用时间 | 备注说明 |
| 2个九 | 99% | 87.6小时 | 高可用的入门阶段，属于基本可用 |
| 3个九 | 99.9% | 8.76小时 | 较高的可用性 |
| 4个九 | 99.99% | 52.56分 | 具有自动恢复能力的高可用性 |
| 5个九 | 99.999% | 5.256分 | 极高可用性 |
| 6个九 | 99.9999% | 31.536秒 | 超高可用性 |
| 级别 | 系统可用性比率 | MAX最大可能服务不可用时间 | 备注说明 |

系统可用性比率 = MTTF/MTBF

|  |  |  |
| --- | --- | --- |
| 指标 | 详细 | 备注 |
| RTO | Recovery Time Objective | 业务恢复指标，理想值为0 |
| RPO | Recovery Point Obejective | 数据恢复指标，理想值为0 |

容灾标准

根据 GB20988-2007-T 信息安全技术信息系统灾难恢复规范 ， 根据其定义，RTO/RPO与灾难恢复能力等级的关系具体如下所示：

|  |  |  |
| --- | --- | --- |
| 灾难恢复能力等级 | RTO | RPO |
| 1 | 2天以上 | 1天至7天 |
| 2 | 24小时以上 | 1天至7天 |
| 3 | 12小时以上 | 数小时至1天 |
| 4 | 数小时至2天 | 数小时至1天 |
| 5 | 数分钟至2天 | 0至30分钟 |
| 6 | 数分钟 | 0 |

**高可用行设计的策略**

保证整体架构的高可用性，有很多策略和手段，比如：

冗余

服务多重化

节点多重化

两地三中心

…

**3、高可用的Kubernetes**

高可用的Kubernetes平台应该能保障如下三种层次的高可用性：

应用层级

Kubernetes自身

业务需求激增下的高可用性

**层级的高可用行**

容器化的微服务在kubernetes上运行，依靠着k8s的RC/deployment/DaemonSet等机制，保证服务的高可用性。

依靠这种机制，Kubernetes平台本身对运行在其上的服务来说，会监控运行在其上的应用的replication的数量，多了删，少了补。

**Kubernetes自身的高可用性**

依靠冗余策略来消除单点以保证ETCD和Master无论何时都始终可用，从而保证了平台自身的高可用。

ETCD是coreos的开源项目用于提供可靠的key/value的数据存储。而kubernetes用来保存数据。使用ETC集群提供稳定的服务保证Kubernetes的API Server能够正常访问到ETCD服务。

同样，Kubernetes的Master大家都很熟悉，通过APISERVER与ETCD进行交互，提供统一的API入口，使用Scheduler进行资源调度，Controller-Manager进行资源管理。

一旦Master不可用，则会造成较大的影响，所以可以采用多个备用状态的Master，一旦出错便可随时切换的机制则能降低或近似消除MASTER的单点故障的可能性，从而使得Kubernetes基础平台自身更加可靠。

为了实现没有单点故障的目标，需要为以下几个组件建立高可用方案：etcd

kube-apiserver

kube-controller-manager与kube-scheduler

kube-dns

下面详细介绍各个组件的高可用策略。

**etcd高可用**

etcd是Kubernetes当中唯一带状态的服务，也是高可用的难点。Kubernetes选用etcd作为它的后端数据存储仓库正是看重了其使用分布式架构，没有单点故障的特性。

虽然单节点的etcd也可以正常运行。但是推荐的部署方案均是采用3个或者5个节点组成etcd集群，供Kubernetes使用。

如果使用的kubeadm工具默认是在一个单节点上启动etcd以及所有的Master组件。虽然使用起来非常方便，但是要用到生产环境还是要注意这个节点宕机的风险。

etcd的高可用基本有三种思路：

一是使用独立的etcd集群，使用3台或者5台服务器只运行etcd，独立维护和升级。甚至可以使用CoreOS的update-engine和locksmith，让服务器完全自主的完成升级。这个etcd集群将作为基石用于构建整个集群。 采用这项策略的主要动机是etcd集群的节点增减都需要显式的通知集群，保证etcd集群节点稳定可以更方便的用程序完成集群滚动升级，减轻维护负担。

二是在Kubernetes Master上用static pod的形式来运行etcd，并将多台Kubernetes Master上的etcd组成集群。 在这一模式下，各个服务器的etcd实例被注册进了Kubernetes当中，虽然无法直接使用kubectl来管理这部分实例，但是监控以及日志搜集组件均可正常工作。在这一模式运行下的etcd可管理性更强。

三是使用CoreOS提出的self-hosted etcd方案，将本应在底层为Kubernetes提供服务的etcd运行在Kubernetes之上。 实现Kubernetes对自身依赖组件的管理。在这一模式下的etcd集群可以直接使用etcd-operator来自动化运维，最符合Kubernetes的使用习惯。

这三种思路均可以实现etcd高可用的目标，但是在选择过程中却要根据实际情况做出一些判断。简单来讲预算充足但保守的项目选方案一， 想一步到位并愿意承担一定风险的项目选方案三。折中一点选方案二。

**kube-apiserver高可用**

apiserver本身是一个无状态服务，要实现其高可用相对要容易一些，难点在于如何将运行在多台服务器上的apiserver用一个统一的外部入口暴露给所有Node节点。

说是难点，其实对于这种无状态服务的高可用，我们在设计业务系统的高可用方案时已经有了相当多的经验积累。需要注意的是apiserver所使用的SSL证书要包含外部入口的地址，不然Node节点无法正常访问apiserver。

apiserver的高可用也有三种基本思路：

一是使用外部负载均衡器，不管是使用公有云提供的负载均衡器服务或是在私有云中使用LVS或者HaProxy自建负载均衡器都可以归到这一类。 负载均衡器是非常成熟的方案，在这里略过不做过多介绍。如何保证负载均衡器的高可用，则是选择这一方案需要考虑的新问题。

二是在网络层做负载均衡。比如在Master节点上用BGP做ECMP，或者在Node节点上用iptables做NAT都可以实现。采用这一方案不需要额外的外部服务，但是对网络配置有一定的要求。

三是在Node节点上使用反向代理对多个Master做负载均衡。这一方案同样不需要依赖外部的组件，但是当Master节点有增减时，如何动态配置Node节点上的负载均衡器成为了另外一个需要解决的问题。

从目前各个集群管理工具的选择来看，这三种模式都有被使用，目前还没有明确的推荐方案产生。建议在公有云上的集群多考虑第一种模式，在私有云环境中由于维护额外的负载均衡器也是一项负担，建议考虑第二种或是第三种方案。

**使用HAproxy+Keepalive进行改进**

keepalived 提供 kube-apiserver 对外服务的 VIP。

haproxy 监听 VIP，后端连接所有 kube-apiserver 实例，提供健康检查和负载均衡的功能。

运行 keepalived 和 haproxy 的节点称为 LB 节点。由于 keepalived 是一主多备运行模式，故至少两个 LB 节点。haproxy 监听的端口(8443) 需要与 kube-apiserver的端口 6443 不同，避免冲突。

keepalived 在运行过程中周期检查本机的 haproxy 进程状态，如果检测到 haproxy 进程异常，则触发重新选主的过程，VIP 将飘移到新选出来的主节点，从而实现 VIP 的高可用。

所有组件（如 kubelet、apiserver、controller-manager、scheduler 等）都通过 VIP 和haproxy 监听的 8443 端口访问 kube-apiserver 服务。

**kube-controller-manager与kube-scheduler高可用**

这两项服务是Master节点的一部分，他们的高可用相对容易，仅需要运行多份实例即可。这些实例会通过向apiserver中的Endpoint加锁的方式来进行leader election， 当目前拿到leader的实例无法正常工作时，别的实例会拿到锁，变为新的leader。

目前在多个Master节点上采用static pod模式部署这两项服务的方案比较常见，激进一点也可以采用self-hosted的模式，在Kubernetes之上用DaemonSet或者Deployment来部署。

**Kube-dns高可用**

严格来说kube-dns并不算是Master组件的一部分，因为它是可以跑在Node节点上，并用Service向集群内部提供服务的。但在实际环境中， 由于默认配置只运行了一份kube-dns实例，在其升级或是所在节点当机时，会出现集群内部dns服务不可用的情况，严重时会影响到线上服务的正常运行。

为了避免故障，请将kube-dns的replicas值设为2或者更多，并用anti-affinity将他们部署在不同的Node节点上。这项操作比较容易被疏忽，直到出现故障时才发现原来是kube-dns只运行了一份实例导致的故障。

**高可用组件方案总结**

上面介绍了Kubernetes Master各个组件高可用可以采用的策略。其中etcd和kube-apiserver的高可用是整个方案的重点。由于存在多种高可用方案，集群管理员应当根据集群所处环境以及其他限制条件选择适合的方案。

这种没有绝对的通用方案，需要集群建设者根据不同的现状在多个方案中做选择的情况在Kubernetes集群建设过程中频频出现， 也是整个建设过程中最有挑战的一部分。

在实际建设过程中，在完成了上述四个组件的高可用之后，最好采取实际关机检验的方式来验证高可用方案的可靠性，并根据检验的结果不断调整和优化整个方案。

**业务需求激增下的高可用性**

严格来说，横向扩展并不一定是一个高可用性架构的必须，但是考虑到动态变化对资源需求的变化以及资源的有效利用，不用说ROI，访问量的突然增大，而资源没有及时调整也会使得原本正常访问的网站也变得缓慢无比，而这个时候则需要横向扩容。

容器化的方式之下，横向扩展变得非常容易。而且kubernetes能够在整体的基础上进行资源的协调和分配从而达到横向扩展的目的。而达到按需扩容则需要结合监控。

实时可靠的监控对高可用系统非常重要，利用很多商用的软件或者很多开源的工具进行整合甚至自行开发可以对整体的业务状况以及系统状况进行把握。在监控中确认采集到指标是否达到了动态调整的阈值，从而进行横向扩展，当然这一切，则需要监控的功能是准确的基础之上的。

**4、Kubernetes使用集群联邦实现多集群管理**

Kubernetes在1.3版本之后，增加了“集群联邦”Federation的功能。这个功能使企业能够快速有效的、低成本的跨区跨域、甚至在不同的云平台上运行集群。这个功能可以按照地理位置创建一个复制机制，将多个kubernetes集群进行复制，即使遇到某个区域连接中断或某个数据中心故障，也会保持最关键的服务运行。在1.7版以后支持本地多个集群联邦管理，不需要使用依赖云平台。

**管理多個kubernetes集群**

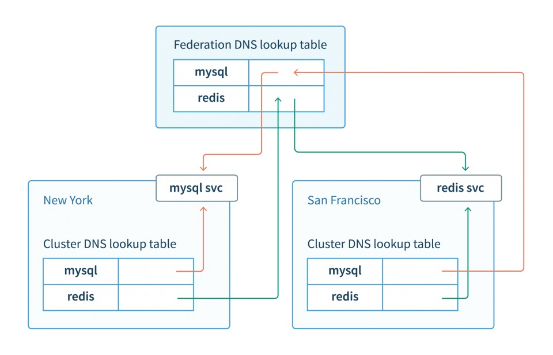
“集群联邦”在架构上同kubernetes集群很相似。有一个“集群联邦”的API server提供一个标准的Kubernetes API，并且通过etcd来存储状态。不同的是，一个通常的Kubernetes只是管理节点计算，而“集群联邦”管理所有的kubernetes集群。

**跨集群服务发现**

Kubernetes有一个标准的插件：kube-dns，这个插件可以在集群内部提供DNS服务，通过DNS解析service名字来访问kubernetes服务。

Kubernetes服务是由一组kubernetesPOD组成的，这些POD是一些已经容器化了的应用，这些POD前面使用到了负载均衡器。假如我们有一个kubernetes集群，这个集群裡面有一个服务叫做mysql，这个服务是由一组mysql POD组成的。在这个kubernetes集群中，其他应用可以通过DNS来访问这个mysql服务。

集群联邦federation/v1beta1 API扩展基于DNS服务发现的功能。利用DNS，让POD可以跨集群、透明的解析服务。



集群发现

**跨集群调度**

为了追求高可用性和更高的性能，集群联邦能够把不同POD指定给不同的Kubernetes集群中。集群联邦调度器将决定如何在不同kubernetes集群中分配工作负载。

通过跨集群调度，我们可以：

-跨kubernetes集群均匀的调度任务负载

-将各个kubernetes集群的工作负载进行最大化，如果当前kubernetes集群超出了承受能力，那麽将额外的工作负载路由到另一个比较空闲的kubernetes集群中

-根据应用地理区域需求，调度工作负载到不同的kubernetes集群中，对于不同的终端用户，提供更高的带宽和更低的延迟。

**集群高可用，故障自动迁移**

集群联邦可以跨集群冗馀部署，当某个集群所在区域出现故障时，并不影响整个服务。集群联邦还可以检测集群是否为不可用状态，如果发现某个集群为不可用状态时，可以将失败的任务重新分配给集群联邦中其他可用状态的集群上。

**5、目前各公有云对kubernetes集群规模主机的配置的建议**

谷歌云GCE：

|  |  |
| --- | --- |
| 1-5节点 | n1-standard-1（ 1 个 vCPU 和 3.75 GB 内存） |
| 6-10节点 | n1-standard-2（2 个 vCPU 和 7.5 GB 内存） |
| 11-100节点 | n1-standard-4（4 个 vCPU 和 15 GB 内存） |
| 101-250节点 | n1-standard-8（8 个 vCPU 和 30 GB 内存） |
| 251-500节点 | n1-standard-16（16 个 vCPU 和 60 GB 内存） |
| 超过500节点 | n1-standard-32（32 个 vCPU 和 120 GB 内存） |

详见文档地址：

<https://cloud.google.com/compute/docs/machine-types>

亚马逊EC2：

|  |  |
| --- | --- |
| 1-5 节点 | m3.medium（ 1 个 vCPU 和 3.75 GB 内存） |
| 6-10 节点 | m3.large（2 个 vCPU 和 7.5 GB 内存） |
| 11-100 节点 | m3.xlarge（4 个 vCPU 和 15 GB 内存） |
| 101-250 节点 | m3.2xlarge（8 个 vCPU 和 30 GB 内存） |
| 251-500 节点 | c4.4xlarge（16 个 vCPU 和 30 GB 内存） |
| 超过 500 节点 | c4.8xlarge（36个 vCPU 和 60 GB 内存） |

详见文档地址：

<https://aws.amazon.com/cn/ec2/pricing/on-demand/>

阿里云ECS：

|  |  |
| --- | --- |
| 节点规模 | Master规格 |
| 1-5个节点 | 4C8G(不建议2C4G) |
| 6-20个节点 | 4C16G |
| 21-100个节点 | 8C32G |
| 100-200个节点 | 16C64G |

详见文档地址：

<https://help.aliyun.com/document_detail/98886.html?spm=a2c4g.11186623.6.817.56412d5662rxOY>