容器云平台的备份与恢复技术

容器云平台本身是一个非常复杂的底层基础设施平台，它主要用于实现业务的持续集成与发布、自动部署、滚动升级、自动扩缩容、租户管理、安全认证等功能模块。这些都是为了实现业务的易用、易发布、强安全等。如何能保证自身的高可用性，这将是衡量用户体验的关键指标，比如在其所依赖的存储、网络等物理设备出现故障时，如何保证其上运行的业务的高可用，无中断或者用户无感知，就需要通过有效的备份与容灾等技术手段来实现。

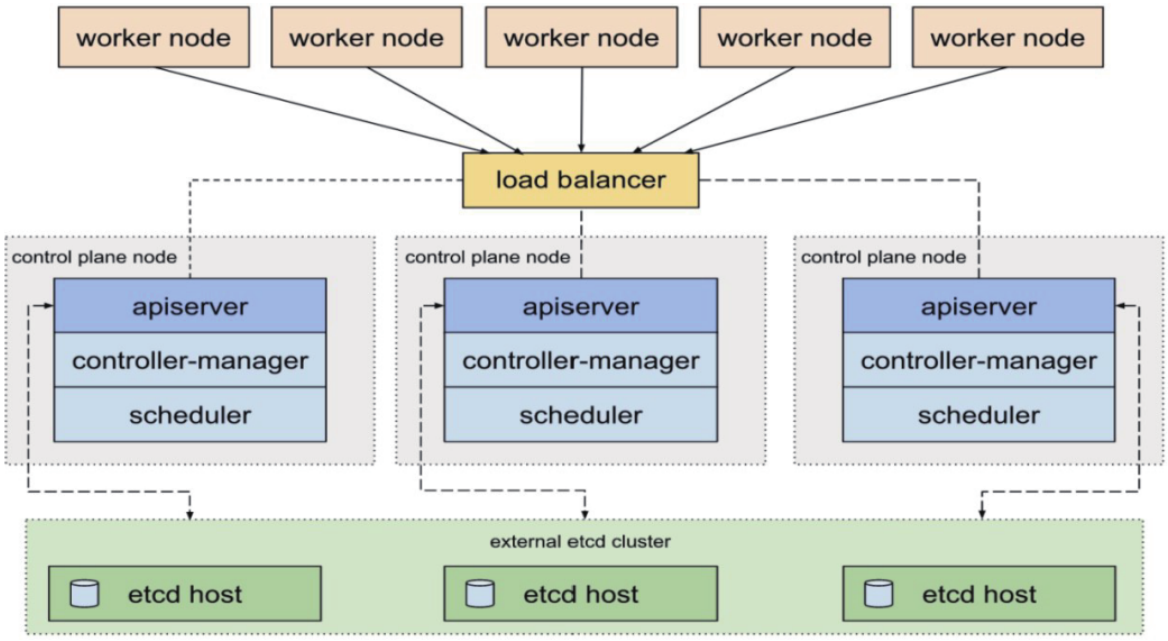
大家都知道，容器云平台包括容器底层的容器编排系统及自身的平台软件系统。底层可以使用多种容器编排系统，开源的主要有 Mesos、Swarm 及 Kubernetes。这里主要讲解 Kubernetes(K8S)容器编排集群的备份与恢复，软件平台本身则通过 K8S 来实现。

我们接下来，会从 K8S 平台架构开始，通过介绍平台架构，让我们了解平台在故障时，哪些组件才是影响服务的高可用性及恢复性的关键，并分享相关的备份恢复与容灾技术。

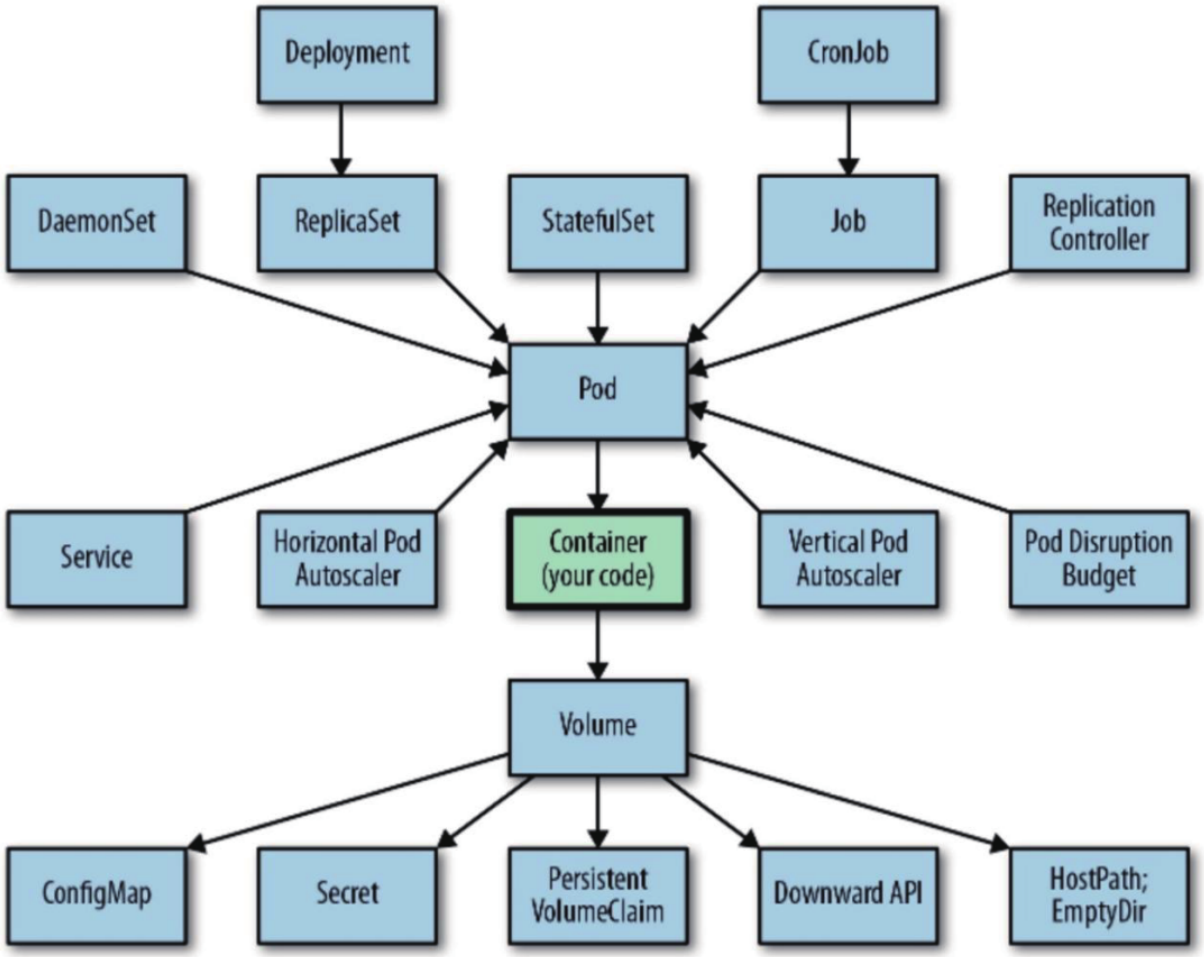
1 K8S 平台架构

Kubernetes(K8S)是用来进行对业务容器实现有效的高可用性的容器编排系统，并可以用来管理集群节点，来按需分配 CPU、内存等资源给相关的业务容器。既然是管理集群，那么就存在被管理节点，针对每个 Kubernetes 集群都由 3~5 个 Master 做为高可用来负责管理和控制集群节点。我们通过 Master 对每个节点 Node 发送命令来实现业务容器的运行。简单来说，Master 就是管理者，Node 就是被管理者。Node 可以是一台物理机器或者一台虚拟机。在 Node 上面可以运行多个 Pod，Pod 是 Kubernetes 管理的最小单位，同时每个 Pod 可以包含多个容器。

通过下面的 Kubernetes 架构简图可以看到 Master 和 Node 之间的关系及底层 etcd 节点的关系。



从 K8S 架构来看，底层是以 etcd 为键值数据库，它是 K8S 的大脑，集群中所有的数据都存储在其中。上层是各系统组件，本身是一个分布式系统构件，各自通过代码实现了高可用，可以暂时不用考虑数据的备份。所有 K8S 数据都在 etcd 中，所以 etcd 是一个我们需要重点关注的备份点。



上图是 K8S 主要的工作负载(workloads），都是以 Pod 为基础进行封装与扩展的资源对象，比如：用于在每台主机上最多只部署一个副本的 DaemonSet，可以多副本控制的 ReplicaSet，及其上的可以用于控制滚动升降级的 Deployment 用于部署有状态应用的 StatefulSet，用于完成一次性作业就退出的 Job 及基于其扩展的控制定时作业的 CronJob，还有用于控制副本数量的 ReplicationController；用于暴露内部服务，负责内部流量负载均衡的 Service，用于水平和横向扩展 Pod 副本及 CPU/Mem 资源的控制器 HPA/VPA，用于限制在同一时间自愿中断的复制应用程序中宕机的 Pod 的数量的 Pod DisruptionBudget(Pod 中断预算）；用于控制有状态服务或提供数据挂卷服务的 Volume，及基于 Volume 扩展的用于提供业务应用配置信息的 ConfigMap，用于传递及保存密钥的 Secret，用于提供持久卷申请的 PVC，用于将 Host 或者 Pod 信息引用给容器的 DownwardAPI，用于 Host 中本地临时目录做数据卷的 HostPath/EmptyDir 等等。这些 workloads 也就是一些提供给用户使用的资源类型或对像实体，它是业务（比如 Tomcat 服务）的承载体，用户通过 kubectl 或者 client-go 等 RestfulAPI 接口给 Master 中的 APIServer 下发部署命令。这些 workloads 通常以“.yaml”结尾的配置文件，主要包含副本的类型、副本个数、名称、端口、模版等信息。APIServer 接受到请求以后，会分别进行以下操作：

◎ 权限验证（包括特殊控制），取出需要创建的资源，保存副本信息到 etcd。

◎ APIServer 和 ControllerManager，Scheduler 以及 kubelete 之间通过 List-Watch 方式通信（事件发送与监听）。

◎ ControllerManager 通过 etcd 获取需要创建资源的副本数，交由 Scheduler 进行调度策略分析。

◎ kubelet 负责最终的 Pod 创建和容器加载。部署好容器（比如：Deployment，StatefulSet）以后，通过 Service 进行访问，通过 cAdvisor 监控资源使用情况。

◎ 最后以 L4 或 L7 负载均衡的方式暴露于集群外供用户访问。

所以，当业务以不同的 workloads 运行在 K8S 上时，这些业务的稳定性及数据安全性，就是依赖于 K8S 来维护其高可用的，当这些业务被误删时，我们需要能完整的进行还原。一般通过备份这些业务的 yaml 部署文件，有状态业务的数据，之后进行重建与数据迁移、恢复等方式来保证业务的还原，所以对于这些业务的配置 yaml 文件及相关的依赖关系，也是我们本节要讲的另一个重要的备份与恢复点。

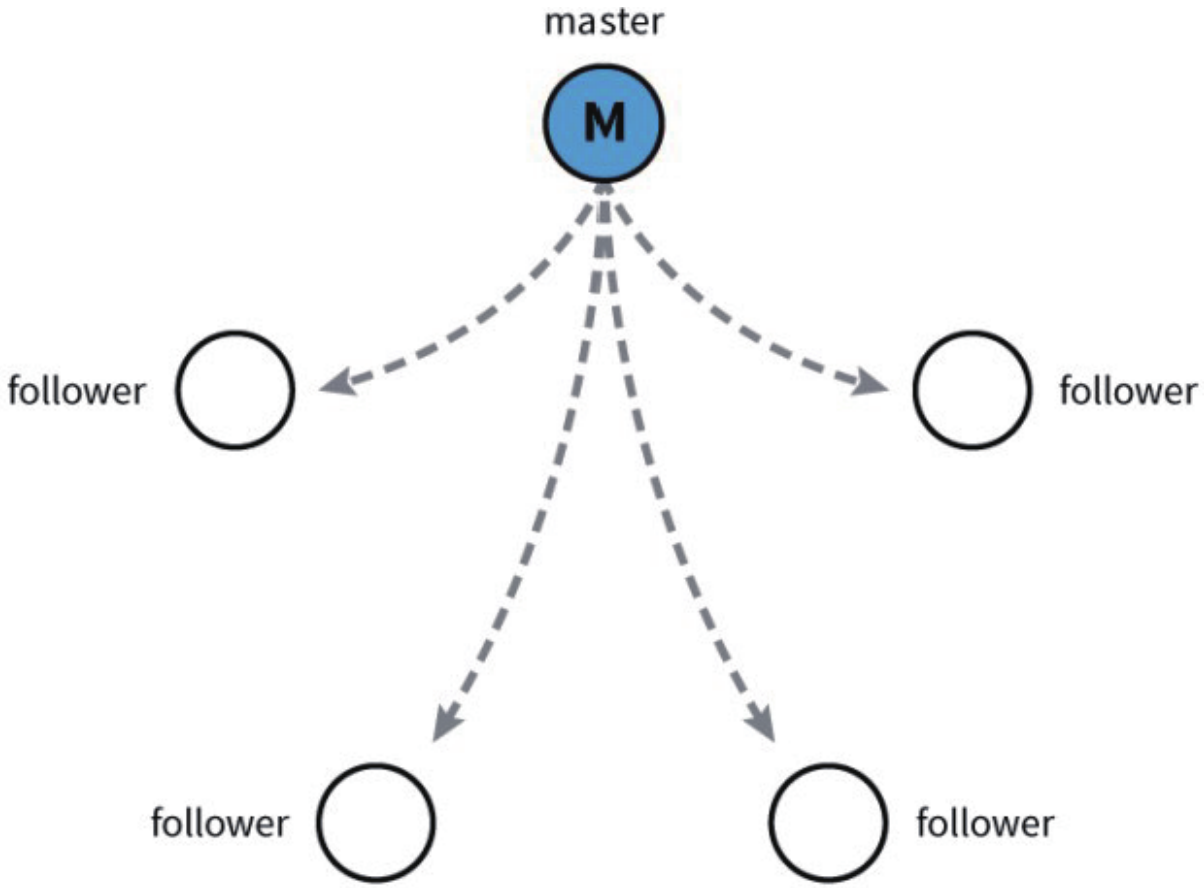
2 Etcd 备份与恢复

Etcd 是一个为分布式系统提供关键的高可靠的键值存储服务，聚集于简单、安全、快速、

可靠等应用场景。它是 Kubernetes 集群的大脑，存储了集群所有的数据信息。在

Kubernetes 中 etcd 可以通过 Raft 分布式共识算法来交换信息，并且可以为分布式系统安全存储关键数据。所以如果发生灾难或者 etcd 的数据丢失时，都会影响集群数据的恢复。

etcd 集群与 zoomkeeper 集群相似，一般采用奇数设备来搭建集群，最多只允许有(N-1)/2 的故障点，即假设集群数量 N 是 3 台设备，最多只能允许 1 台设备出现故障，而还剩余 2 台，这样将不影响集群的可用性。



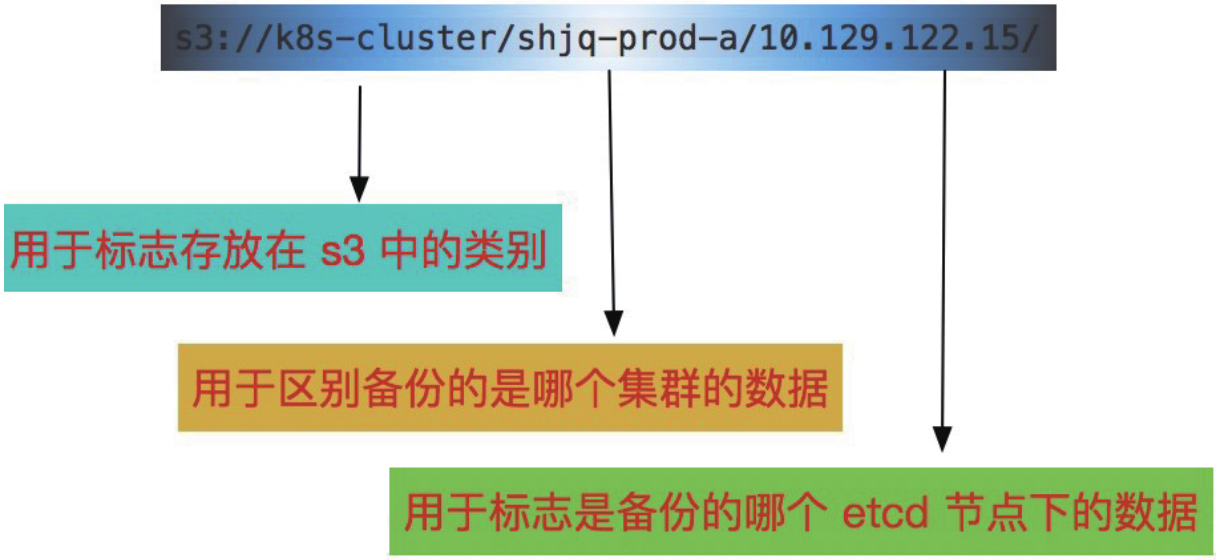
本小节将重点讲解如何备份和恢复 etcd 集群。数据备份方式是借助 etcd 的 snapshot 功能做备份的，需要时可以把 etcd 集群回滚到某个备份的时间点。目前我们一般采用的是 etcd v3 版本，因为它的速度更快，但备份时，需要注意不能使用非一致性版本。如果我们使用 API V2 来备份数据，那么使用 etcd API v3 版本是无法进行恢复的。**2.1 etcd 备份**

etcd 存放的数据如此重要，那我们备份就需要做到万无一失。我们采用本地备份、远程备份。具体方案如下：

◎ 一重备份：本地会每隔 6 小时备份一次，并保留最近 28 个备份，也就是 1 周的数据。

◎ 二重备份：远程备份一个月或更长时间段的数据，并定期清理。

备份在 S3 中的存储格式如下：



本地使用某个 etcd-backups 文件目录存放一周数据，远程使用 S3 存储。在 S3 中，存放的目录的组织方式有些讲究，我们可以将某个 S3bucket 规划为 K8S 集群所有，每个集群都对应其中的一个以集群名命名的子目录，每个子目录下备份的又是当前 etcd 节点的 db 文件。比如 s3://k8s-cluster/shjq-prod-a/10.129.122.15/这个表示当前 etcd 节点是 10.129.122.15，它的数据会被存放到 S3 集群中的以 k8s-cluster 命名的 bucket 下面，该 etcd 属于 shjq-prod-a 环境，所以将其作为子目录，最后以自身节点来命名文件夹，这样数据就相应的备份好了。这种命名规范很重要，当有多个 K8S 集群的数据需要备份时，我们可以在恢复时，很容易找到备份文件进行还原，这也特别适合自动化脚本的编写。

```
#!/bin/bash

# crontab定期执行备份脚本，每6小时备份一次，本地、异地都备份（暂定：本地备份保留最近28个备份， 异地保留一个月的备份）

ENDPOINTS=http://10.129.122.15:2379,http://10.129.122.16:2379,http://10.129.122.17:2379
BACKUP_DIR='/data/etcd-backups'
DATE=`date +%Y%m%d-%H%M%S`
[ ! -d $BACKUP_DIR ] &&mkdir -p $BACKUP_DIR
export ETCDCTL_API=3;etcdctl --endpoints=$ENDPOINTS snapshot save
$BACKUP_DIR/snapshot-$DATE\.db
sudos3cmdput$BACKUP_DIR/snapshot-$DATE\.dbs3://k8s-cluster/shjq-prod- a/10.129.122.15/
cd $BACKUP_DIR;ls -lt $BACKUP_DIR|awk '{if(NR>28){print "rm -rf"$9}}'|sh
```

对于 OpenShift4，不要在第一个证书轮转完成前（安装后的 24 小时内）进行 etcd 备份，否则备份将包含过期的证书。另外，建议您在非使用高峰时段对 etcd 进行备份，因为备份可能会影响到系统性能。

您可以在任何运行 etcd 实例的 master 主机上执行 etcd 数据备份（不需要为每个 master 主机进行备份），也可以通过自动化来定期进行 etcd 的备份。

1. 为 master 节点启动一个 debug 会话：

$ oc debug node/<node_name>

2. 将您的根目录改为主机：

sh-4.2# chroot /host

3.（optional）如果启用了集群范围的代理，请确定已导出了 NO_PROXY、HTTP_PROXY 和 HTTPS_PROXY 环境变量。

4. 运行 cluster-backup.sh 脚本，输入保存备份的位置。

sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup

在这个示例中，在 master 主机上的/home/core/assets/backup/目录中创建了两个文件：

◎ snapshot_<datetimestamp>.db：这个文件是 etcd 快照

◎ static_kuberesources_<datetimestamp>.tar.gz：此文件包含静态 Pod 的资源。如果启用了 etcd 加密，它也包含 etcd 快照的加密密钥。

如果启用了 etcd 加密，建议出于安全考虑，将第二个文件与 etcd 快照分开保存。但是，需要这个文件才能从 etcd 快照中进行恢复。

etcd 仅对值进行加密，而不对键进行加密。这意味着资源类型、命名空间和对象名称是不加密的。

◎ 替换机器没有运行或者节点未就绪的不健康的 etcd 成员

使用 cluster-admin，选择一个不在受影响节点上的 Pod

```
oc get pods –n openshift–etcd | grep etcd
etcd–ip–10–0–131–183.ec2.internal          3/3    Running    0        123m
etcd–ip–10–0–164–97.ec2.internal           3/3    Running    0        123m
etcd–ip–10–0–154–204.ec2.internal          3/3    Running    0        124m
```

连接到正在运行的 etcd 容器

oc rsh -n openshift-etcd etcd-ip-10-0-154-204.ec2.internal

查看成员列表：

sh-4.2# etcdctl member list -w table

通过向 etcdctl member remove 命令提供 ID 来删除不健康的 etcd 成员

sh-4.2# etcdctl member remove 6fc1e7c9db35841d

确认删除

sh-4.2# etcdctl member list -w table

◎ 替换 etcd pod 处于 crashlooping 状态的不健康 etcd 成员

◎ 恢复到以前的集群状态

2.2 Etcd 恢复

当 etcd 集群的 leader 故障时，etcd 集群会自动选择一个新 leader。在 leader 选举期间，集群不能处理任何写入操作。在选举期间发送的写入请求需要排队等待处理，直到选出新的 leader。已经发送给 oldleader 但尚未提交的数据可能会丢失。新 leader 有权重写 oldleader 的任何未提交的条目。从用户的角度来看，一些写入请求可能会超时，但是没有提交的写入会丢失。

为了从灾难性故障中恢复，etcdv3 提供了快照和恢复功能，以便在没有 v3 密钥数据丢失的情况下重新创建群集。要恢复集群，只需要一个快照“db”文件。集群恢复将创建新的 etcd 数据目录; 所有成员应该使用相同的快照进行恢复。恢复会覆盖某些快照元数据（特别是成员 ID 和群集 ID），该成员失去了其以前的身份，该元数据覆盖可防止新成员无意中加入现有群集。因此，为了从快照启动集群，还原必须重启 etcd 集群。

在还原时可以选择验证快照完整性。如果使用快照 etcdctlsnapshotsave，它将由通过完整性散列检查的 etcdctl snapshot restore 来恢复。如果快照是从数据目录复制的(配置文件中的 data-dir)，则不存在完整性哈希，并且只能使用--skip-hash-check 来恢复。

以我们下面的生产环境脚本为例，etcd 以 docker 方式运行，并需要在恢复前，提前清理当前 etcd 数据目录下的数据，并借助 snapshot 功能，使用备份的 db 文件进行 restore，最后重启 etcd 服务，以使当前数据恢复到我们给定的某个时刻。当然这恢复操作，需要在不同的节点上依次进行，来保证恢复后的 etcd 集群正常运行。

```
[root@SCSP01208etcd]#    cat/usr/bin/etcd-restore.sh
#!/bin/bash

# restore
[ -d 10.129.122.15.etcd ] &t&trm -rf 10.129.122.15.etcd


#TODO:therestoresnapshotshouldbedownloadfroms3manullyatfirst. # then substitue blow db, and run thescript.
ETCDCTL_API=3etcdctlsnapshotrestoreplease-use-a-same-snapshot-which-would- be-downloaded-from-s3.db\
    --name10.129.122.15   \
    --initial-cluster 10.129.122.15=http://10.129.122.15:2380,10.129.122.16=http://10.129.122.16:2380,
10.129.122.17=http://10.129.122.17:2380 \
    --initial-advertise-peer-urls http://10.129.122.15:2380


# mv etcd data
mv/data/etcd/member/data/etcd/member-$(date+%Y%m%d%H%M%S)&t&t\ mv
/data/etcd/10.129.122.15.etcd/member/data/etcd/

# restart etcd docker restart etcd

# check healthy sleep 5
etcdctl cluster-health


#
rm -rf 10.129.122.15.etcd
```

Kubernetes 集群备份主要是备份 etcd 集群，在恢复时，根据组件之间的依赖关系，整个恢复流程的顺序如下：

◎ 停止 kube-apiserver 组件，关闭集群数据写入

◎ 停止 etcd 节点，准备替换底层 db 数据

◎ 恢复数据 snapshot restore，用需要回滚的 db 开始替换现存的数据

◎ 启动 etcd，重新加载新的数据库信息

◎ 启动 kube-apiserver 组件，开启集群数据写入

注意：备份 etcd 集群时，只需要备份一个 etcd 就行，恢复时，拿同一份备份数据恢复。

对于 OpenShift4，您可以使用已保存的 etcd 备份恢复到先前的集群状态。您可以使用 etcd 备份来恢复单个 control plane 主机。然后，etcd cluster Operator 会处理剩余的 master 主机的扩展。

流程如下：

1. 选择一个要用作恢复主机的 control plane 主机。这是您要在其中运行恢复操作的主机。

2. 建立到每个 control plane 节点（包括恢复主机）的 SSH 连接。恢复过程启动后，Kubernetes API 服务器将无法访问，因此您无法访问 control plane 节点。因此，建议在一个单独的终端中建立到每个 control plane 主机的 SSH 连接。

3. 将 etcd 备份目录复制复制到恢复 control plane 主机上。此流程假设您将 backup 目录（其中包含 etcd 快照和静态 Pod 资源）复制到恢复 control plane 主机的/ home/core/ 目录中

4. 在所有其他 control plane 节点上停止静态 Pod

◎ 将现有 etcd Pod 文件从 Kubelet 清单目录中移出：

◎ 验证 etcd Pod 是否已停止。

◎ 将现有 Kubernetes API 服务器 Pod 文件移出 kubelet 清单目录：

◎ 验证 Kubernetes API 服务器 Pod 是否已停止。

◎ 将 etcd 数据目录移到不同的位置：

◎ 在其他不是恢复主机的 master 主机上重复这个步骤。

5. 访问恢复 control plane 主机。

6. (optional）如果启用了集群范围的代理，请确定已导出了 NO_PROXY、HTTP_PROXY 和 HTTPS_PROXY 环境变量。

7. 在恢复 control plane 主机上运行恢复脚本，提供到 etcd 备份目录的路径：

[core@ip-10-0-143-125 ~]$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/backup

8. 在所有 master 主机上重启 kubelet 服务。

[core@ip-10-0-143-125 ~]$ sudo systemctl restart kubelet.service

9. 确认单个成员 control plane 已被成功启动。

◎ 从恢复主机上，验证 etcd 容器是否正在运行。

[core@ip-10-0-143-125 ~]$ sudo crictl ps | grep etcd

◎ 从恢复主机上，验证 etcd Pod 是否正在运行。

[core@ip-10-0-143-125 ~]$ oc get pods -n openshift-etcd | grep etcd

如果状态是 Pending，或者输出中列出了多个正在运行的 etcd Pod，请等待几分钟，然后再次检查。

10. 强制 etcd 重新部署。